



Yii. Сборник рецептов

Более 80 рецептов, которые помогут вам
эффективно использовать PHP-фреймворк Yii

Александр Макаров,
один из разработчиков фреймворка Yii

Вступление от Qiang Xue, ведущего разработчика Yii



Yii. Application Development Cookbook

Over 80 recipes to help you master using the
Yii PHP framework

Alexander Makarov

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

Yii. Сборник рецептов

Издание рекомендовано в качестве учебного пособия
для студентов технических вузов

Более 80 рецептов, которые помогут вам
использовать PHP-фреймворк Yii

Александр Макаров



Москва, 2013

УДК 004.738.5:004.45Yii
ББК 32.973.202-018.2
M15

M15 Макаров А.С.

Yii. Сборник рецептов. – М.: ДМК Пресс, 2013. – 372 с.: ил.

ISBN 978-5-94074-786-4

Данная книга познакомит вас с самыми важными особенностями и внутренними механизмами PHP-фреймворка Yii, что позволит вам использовать его наиболее эффективно.

Сборник поможет вам изучить часто упускаемые из вида, но очень полезные особенности фреймворка и повысить свой уровень как разработчика приложений.

Наиболее интересные темы касаются разработки приложений и расширений, обработки ошибок, отладки, вопросов безопасности и улучшения производительности.

Издание предназначено для веб-разработчиков как уже знакомых с Yii, так и начинающих пользователей фреймворка.

УДК 004.738.5:004.45Yii
ББК 32.973.202-018.2

Original English language edition published by Published by Packt Publishing Ltd., Livery Place, 35 Livery Street, Birmingham B3 2PB, UK. Copyright © 2011 Packt Publishing. Russian-language edition copyright (c) 2012 by DMK Press. All rights reserved.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-1-849515-48-1 (англ.)
ISBN 978-5-94074-786-4 (рус.)

Copyright © 2011 Packt Publishing.
© Оформление, перевод на русский язык
ДМК Пресс, 2013



ОГЛАВЛЕНИЕ

Предисловие	10
Рецензенты английской версии	11
Благодарности.....	12
Об авторе	13
Введение	14
Что вы найдёте в данной книге	14
Что понадобится для чтения этой книги	16
На кого рассчитана эта книга.....	17
Соглашения	17
Обратная связь.....	17
Глава 1. Под капотом.....	18
Вступление.....	18
Использование getters и setters	18
Использование событий Yii	21
Использование импорта и автозагрузки	29
Использование исключений	32
Настройка компонентов	35
Настройка виджетов по умолчанию	38
Использование коллекций ядра Yii	40
Работа с запросами.....	44
Глава 2. Маршрутизация, контроллеры и представления	48
Введение.....	48
Правила маршрутизации	49
Автоматическая генерация URL-адресов	52
Регулярные выражения в правилах маршрутизации	56
Правила маршрутизации для статических страниц	60
Добавление правил маршрутизации в рабочее приложение....	62
Базовый контроллер.....	66

Подключение внешних действий	68
Отображение статических страниц при помощи CViewAction	71
Использование flash-сообщений	73
Контекст контроллера в представлении	74
Повторное использование вложенных представлений	76
Клипы	78
Декораторы	80
Несколько макетов в приложении	81
Постраничная разбивка и сортировка данных	84

Глава 3. AJAX и jQuery

Введение	86
Загрузка блока через AJAX	86
Управление ресурсами	91
Подключение ресурсов	96
Работа с JSON	99
Передача параметров из PHP в JavaScript	102
Обработка переменного числа полей в форме	104

Глава 4. Работа с формами

Введение	111
Пишем свой валидатор	111
Загрузка файлов	114
Добавление CAPTCHA	118
Настройка CAPTCHA	122
Создаем виджет для ввода при помощи CInputWidget	125

Глава 5. Тестирование приложений

Введение	129
Настройка тестового окружения	129
Написание и запуск юнит-тестов	133
Фикстуры	139
Функциональное тестирование	144
Генерация отчетов о покрытии кода	149

Глава 6. База данных, Active record и трюки

с моделями

Введение	153
Получение данных из базы данных	154
Создание и использование нескольких подключений к базам данных	160
Использование именованных групп условий для создания многоязычных моделей	163

Обработка полей модели с помощью методов-событий	
Active Record	167
Применение markdown и HTML	169
Подсветка кода с помощью Yii	172
Автоматический timestamp	178
Автоматическое указание автора	180
Реализация наследования с одной таблицей	182
Использование CDbCriteria	186

Глава 7. Использование компонентов Zii 188

Введение	188
Использование источников данных	188
Использование гридов	195
Использование списков	202
Создание своих столбцов грида	206

Глава 8. Расширение Yii 213

Введение	213
Создание поведений модели	213
Создание компонентов	220
Создание действий контроллера, пригодных для повторного использования	224
Создание контроллеров, пригодных для повторного использования	227
Создание виджета	231
Создание консольных команд	234
Создание фильтров	237
Создание модулей	239
Создание своего обработчика шаблонов	246
Подготовка расширений к публикации	251

Глава 9. Обработка ошибок, отладка и журналирование 255

Введение	255
Использование различных маршрутов для журналов	255
Анализ трассировки стека при ошибках	262
Журналирование и использование контекстной информации	265
Реализация собственного умного обработчика кода 404	270

Глава 10. Безопасность 275

Введение	275
Использование фильтров контроллера	275

Использование CHtml и CHtmlPurifier для предотвращения XSS.....	280
Предотвращение SQL-инъекций.....	285
Предотвращение CSRF.....	290
Использование RBAC	294

Глава 11. Настройка производительности..... 302

Введение.....	302
Использование передового опыта.....	302
Ускорение управления сессиями.....	308
Использование зависимостей кеша и цепочек	312
Профилирование приложений с помощью Yii	318

Глава 12. Использование постороннего кода... 329

Введение.....	329
Использование Zend Framework из Yii	329
Настройка автозагрузчика Yii	334
Использование Kohana внутри Yii.....	339
Использование PEAR внутри Yii	346

Глава 13. Развертывание 349

Введение.....	349
Изменение структуры директорий Yii	349
Перемещение приложения из корневой директории сервера	352
Совместное использование директории фреймворка	355
Перемещение части настроек в отдельные файлы.....	356
Использование нескольких конфигураций для упрощения развертывания	362
Реализация и исполнение заданий cron.....	366
Режим обслуживания	368



ПРЕДИСЛОВИЕ

Когда Александр рассказал мне, что собирается писать сборник рецептов по Yii, я задумался, насколько уникальными они будут, ведь в то время уже была создана официальная wiki, пополняющаяся силами сообщества. Мои опасения были напрасны.

Книга получилась полной информации об эффективном использовании фреймворка. Информация была подана настолько методично, что вполне могла использоваться как необходимое дополнение к полному руководству по Yii.

В процессе написания Александр просил членов команды Yii высказать своё мнение о черновиках и в процессе сумел заинтересовать всех нас. Как автор и ведущий разработчик Yii я считаю, что эту книгу должен прочитать каждый, кто работает с фреймворком.

В книге нет формального описания правил разработки. Вместо этого она показывает, как программировать на Yii с практической точки зрения. Материал особенно пригодится тем, кто работает со сжатыми сроками, так как в нём представлено множество решений проблем, с которыми разработчики могут столкнуться в своих проектах. Тем, кто уже знаком с фреймворком, книга также будет интересна. Большинство решений, представленных в данной книге, можно считать рекомендуемыми официально, так как они прошли тщательное рецензирование каждым членом команды Yii.

Александр показал себя данной книгой и его активным участием в проекте Yii как отличный программист и писатель.

*Qiang Xue,
ведущий разработчик фреймворка Yii*



РЕЦЕНЗЕНТЫ АНГЛИЙСКОЙ ВЕРСИИ

Anatoliy Dimitrov – обладатель сертификата O'Reilly по программированию PHP/MySQL и активный член сообщества Yii. Кроме того, он опытен в вопросах безопасности веб-приложений.

В прошлом занимал ведущие технические должности в нескольких крупнейших хостинговых компаниях и платёжных системах. Имеет за плечами множество успешных фриланс-проектов.

Antonio Ramirez Cobos (tonydspaniard) – программист-самоучка. Нырнул в мир программирования в процессе изучения «железа» в TAFE в Мельбурне.

До встречи с PHP и чудесами OpenSource успел накопить более тринадцати лет опыта с Javascript, C++, Java, ASP.net (C#), Visual Basic (COM, COM+) и Dynamic DLL. После знакомства с PHP работал исключительно с ним и специализировался на веб-приложениях.

Любитель Yii, ведёт блог по адресу www.ramirezcobos.com. Активный участник форума Yii.

SAKURAI, atsushi – эксперт по микропроцессорам и PHP программист уже более десяти лет. В качестве руководителя команды разработчиков микропроцессоров занимался в том числе и сайтом для их поддержки. Благодаря Yii эффективность разработки в этом направлении заметно повысилась. Его главный вклад в Yii – перевод документации на японский язык.



БЛАГОДАРНОСТИ

Хочу поблагодарить команду разработчиков Yii, в особенности Qiang Xue, Maurizio Domba и Sebastian Thierer, которые рецензировали черновики данной книги и поддерживают Yii в отличной форме, постоянно его улучшая.

Спасибо Александру Кочетову, Antonio Ramirez Cobos и всем, кто присылал замечания как по предварительной версии, так и по первой редакции. Ваши замечания очень помогли улучшить книгу.

Спасибо Qiang Xue и Wei Zhuo за Yii.

Спасибо Packt Publishing за предложение написать книгу и за то, что помогли её закончить.

Огромное спасибо тем, кто помог с русскоязычной версией: Александру Бордун, Ивану Левчунец, Сергею Сыцевич, Максиму Фуртуна и Дмитрию Мовчан.

Без вас я бы не справился!



ОБ АВТОРЕ

Александр Макаров – опытный инженер из Воронежа, успевший попробовать себя в роли РМ. Активный участник OpenSource проектов и один из разработчиков PHP-фреймворка Yii.

С 2008 по 2010 годы способствовал росту русскоязычного сообщества CodeIgniter. Примерно в то же время начал активно участвовать в OpenSource.

В 2009 году заинтересовался Yii, создал yiiframework.ru и закончил перевод официальной документации на русский язык.

С мая 2010 года присоединился к команде разработчиков фреймворка. Ведёт блог gmcreative.ru.

Выступает на различных конференциях. Работает в Stay.com, где занимается созданием крутых штук на Yii и не только.

Увлекается английским и русским языками, дизайном и UX. Любит смотреть хорошее кино, путешествовать и фотографировать.



ВВЕДЕНИЕ

Yii – очень гибкий и высокопроизводительный PHP-фреймворк, предназначенный для разработки веб-приложений. От небольших страничек до масштабируемых приложений уровня предприятия. Название фреймворка расшифровывается как Yes It Is, что является точным ответом на большинство вопросов, пока ещё не знакомых с Yii, разработчиков: «А он быстр?», «Безопасен?», «Подходит ли для профессиональной разработки?», «Подойдёт ли для моего следующего проекта?». Ответ на все эти вопросы один: «Yes, it is!»

Данный сборник рецептов состоит из 13 независимых глав, каждая из которых полна решений, которые помогут вам эффективно использовать фреймворк. Вы узнаете о скрытых возможностях, работе ядра, создании своих компонент, разработке через тестирование и ещё множестве интересных тем.

Что вы найдёте в данной книге

Глава 1. Под капотом

Рассказывает о наиболее интересных возможностях Yii, про которые мало говорится в официальном руководстве: события, импорт и автозагрузка классов, исключения, компоненты, настройка виджетов и другое.

Глава 2. Маршрутизация, контроллеры и представления

О полезных приёмах, относящихся к обработке и постоению URL, контроллерам и представлениям. В данной главе описаны правила URL, внешние действия контроллера и сами контроллеры, декораторы представлений и многое другое.

Глава 3. AJAX и jQuery

Поведает о клиентской части Yii, в которой используется jQuery – самая широко применяемая JavaScript-библиотека в мире. Главным образом рассматриваются особенности Yii, а не самой jQuery.

Глава 4. Работа с формами

Yii сильно облегчает работу с формами. Несмотря на то, что документация на эту тему довольно полная, есть некоторые особенности, требующие разъяснений и примеров.

В этой главе среди прочего описывается создание своих валидаторов и виджетов форм, загрузка файлов, использование и настройка CAPTCHA.

Глава 5. Тестирование приложений

Описывает, помимо модульного тестирования, функциональное тестирование и генерацию отчёта по покрытию кода тестами. В рецептах этой главы применяется разработка через тестирование: сначала пишутся тесты, после разрабатывается само приложение.

Глава 6. База данных, Active record и трюки с моделями

Показывает, как эффективно работать с базой данных как при использовании моделей, так и напрямую, через DAO. Рассказывает про работу с несколькими базами данных одновременно, про построение и использование критерия запроса и предварительную обработку полей модели.

Глава 7. Использование компонентов Zii

Посвящена провайдерам данных, градам и спискам. Учит настраивать сортировку и поиск, использовать грид с несколькими связанными моделями, создавать свои типы столбцов и многому другому.

Глава 8. Расширение Yii

Не только учит реализации своих расширений Yii, но и рассказывает, как сделать их действительно гибкими и полезными для сообщества. В дополнение мы рассмотрим множество деталей, позволяющих сделать ваше расширение лучше.

Глава 9. Обработка ошибок, отладка и журналирование

Журналирование, анализ стека исключения, реализация своего обработчика ошибок.

Глава 10. Безопасность

Необходимая информация о том, как сделать приложение безопасным в соответствии с главным принципом «фильтруй входные данные, экранируй выходные». Рассматриваются такие темы, как создание своих фильтров контроллера, предотвращение атак типа XSS, CSRF и SQL-инъекций. Экранирование выходных данных и использование контроля доступа на основе ролей.

Глава 11. Настройка производительности

Показывает как настроить Yii для получения повышенной производительности. В этой главе вы познакомитесь с приёмами разработки, которые позволят вашему приложению работать достаточно быстро для того, чтобы дорасти до высоких нагрузок.

Глава 12. Использование стороннего кода

Использование стороннего кода и библиотек в приложениях на Yii. Рассматриваются Zend Framework, Kohana и PEAR, но после того, как поймёте, как это работает, вы сможете использовать любые другие библиотеки.

Глава 13. Развёртывание

Данная глава познакомит вас с различными приёмами, которые особенно полезны при развёртывании приложения, работе в команде и позволяют сделать жизнь разработчика проще и удобней.

Что понадобится для чтения этой книги

Для того, чтобы запускать примеры из этой книги, потребуется следующее:

- Apache 2.x. Другие вебсерверы тоже будут работать, но инструкции приведены именно для Apache.
- MySQL 5 с поддержкой InnoDB.
- PHP любой версии, начиная с 5.2.
- Последняя версия Yii 1.1.x.

Для некоторых рецептов дополнительно потребуется следующее:

- PHPUnit;
- XDebug;
- Selenium RC;
- PEAR;
- Smarty;
- memcached.

На кого рассчитана эта книга

Если вы разработчик с хорошим знанием PHP5, знаете основы Yii, хотя бы бегло прочли полное руководство и уже попробовали разрабатывать свои приложения, можете смело начинать читать эту книгу. Знание ООП и MVC пригодится, потому как в Yii они используются повсеместно.

Соглашения

В этой книге вы встретите некоторое количество стилей текста, которые помогают различать различные блоки информации.

Код, названия файлов и пути выделены
моноширинным шрифтом.

Консольный ввод и вывод также оформлены
моноширинным жирным шрифтом.

Те части, на которые стоит обратить внимание, выделены
жирным шрифтом.

Предупреждения и важные заметки выделены в отдельные блоки.

Обратная связь

Ваши вопросы, замечания и пожелания всегда кстати. Связаться с автором можно через специальную форму по адресу <http://ru.yiicookbook.org/feedback>. Через эту же форму можно отправить найденные в книге ошибки. Они будут размещены по адресу <http://ru.yiicookbook.org/errata>.

Код к книге можно найти по адресу <http://ru.yiicookbook.org/code>.



ГЛАВА 1.

Под капотом

В этой главе мы рассмотрим:

- Использование getters и setters.
- Использование событий Yii.
- Использование импорта и автозагрузки.
- Использование исключений.
- Настройка компонентов.
- Настройка виджетов по умолчанию.
- Использование коллекций.
- Работа с запросами.

Вступление

В этой главе мы рассмотрим самые интересные особенности Yii, которые скрыты «под колпаком». Главным образом они описаны в API фреймворка, но поскольку они не упомянуты в официальном руководстве (<http://www.yiiframework.com/doc/guide/>) или же упомянуты очень поверхностно, обычно их используют только опытные разработчики. Тем не менее, особенности, описанные здесь, являются относительно простыми, и их использование делает разработку с Yii намного более увлекательной и продуктивной.

Использование getters и setters

В Yii присутствует много особенностей, которые пришли из других языков, таких как Java или C#. Одна из них – это определение свойств с помощью геттеров (getters) и сеттеров (setters) для любого класса, который наследует класс CComponent (то есть практически любого класса Yii).

Из этого рецепта вы научитесь определять свои собственные свойства, используя getters и setters делать свойства доступными только для

чтения и скрывать собственную обработку сохраняя обычный синтаксис присваивания значения PHP.

Как это делается

1. Так как в PHP свойства отсутствуют на уровне языка, мы можем использовать только методы получения (getters) и назначения (setters) следующим образом:

```
class MyClass
{
    // скрываем $property
    private $property;

    // получение
    public function getProperty()
    {
        return $this->property;
    }

    // назначение
    public function setProperty($value)
    {
        $this->property = $value;
    }
}

$object = new MyClass();

// установка значения
$object->setProperty('value');

// получение значения
echo $object->getProperty();
```

2. Этот синтаксис очень распространен в мире Java, но требует немного больше времени для использования в PHP. Тем не менее, мы хотим использовать ту же функциональность, которую нам дают свойства C#: вызов методов получения и назначения как членов класса. С Yii мы можем сделать это следующим образом:

```
// необходимо наследовать CComponent
class MyClass extends CComponent
{
    private $property;
    public function getProperty()
    {
```



```
        return $this->property;
    }
    public function setProperty($value)
    {
        $this->property = $value;
    }
}

$object = new MyClass();
$object->property = 'value'; // равнозначно $object
->setProperty('value');
echo $object->property; // равнозначно $object
->getProperty();
```

3. Используя эту функцию, вы можете сделать свойства только для чтения или только для записи, сохраняя при этом простой синтаксис PHP следующим образом:

```
class MyClass extends CComponent
{
    private $read = 'read only property';
    private $write = 'write only property';

    public function getRead()
    {
        return $this->read;
    }

    public function setWrite($value)
    {
        $this->write = $value;
    }
}

$object = new MyClass();

// дает ошибку, так как мы пытаемся изменить
// свойство, доступное только для чтения
$object->read = 'value';

// выводит 'read only property'
echo $object->read;

// дает ошибку, так как мы пытаемся получить
// свойство, доступное только для записи
echo $object->write;

// запишет 'value' в приватное поле $write
$object->write = 'value';
```

4. Yii широко использует эту технику, так как почти всё – это компоненты. Например, когда вы вызываете `Yii::app()->user->id`, чтобы получить ID текущего авторизованного пользователя, то на самом деле вызывается `Yii::app()->getUser()->getId()`.

Как это работает

Для использования геттеров и сеттеров как свойств `CComponent` использует магические методы PHP: `__get`, `__set`, `__isset` и `__unset` (<http://php.net/manual/en/language.oop5.magic.php>). Следующий пример показывает, как в Yii выглядит `CComponent::__get`:

```
public function __get($name)
{
    $getter='get'.$name;
    if(method_exists($this,$getter))
        return $this->$getter();
    ...
}
```

Магический метод PHP перехватывает все вызовы к недостающим реальным свойствам, поэтому, когда мы вызываем `$myClass->property`, он получает свойство как параметр `$name`. Если же существует метод с именем `getProperty`, тогда PHP использует его возвращаемое значение как значение свойства.

И ещё

Для получения дополнительной информации обратитесь по следующей ссылке: <http://www.php.net/manual/en/language.oop5.overloading.php#language.oop5.overloading.members>.

Смотрите также

- Рецепт «Использование событий Yii» в этой главе.
- Рецепт «Настройка компонентов» в этой главе.

Использование событий Yii

Большинство классов фреймворка наследуют `CComponent`, что позволяет достичь большой гибкости приложений с помощью событий. Событие представляет собой сообщение о том, что приложение что-то выполнило. Мы можем зарегистрировать несколько обработчиков, которые будут реагировать на определенные типы событий. Обра-

ботчик может получить параметры от события, с которым работает, и среагировать соответствующим образом.

В этом рецепте вы узнаете, как объявлять и использовать как предопределенные, так и пользовательские события.

Как это делается

Чтобы объявить событие в дочернем классе `CComponent`, вы должны добавить метод с именем, начинающимся с `on`. Например, если вы добавите метод `onRegister`, то получите соответствующее объявленное событие.

Метод, который объявляет событие, является обработчиком события по умолчанию.

Обычно события используются следующим образом:

- объявите событие, добавив соответствующий метод;
- подключите один или несколько обработчиков событий;
- компонент вызывает событие с помощью метода `CComponent::raiseEvent`;
- все подписанные обработчики автоматически вызываются.

Давайте посмотрим, как мы можем присоединить обработчик к событию. Чтобы сделать это, мы можем использовать метод `CComponent::attachEventHandler`. Он принимает два параметра:

- `$name`: имя события;
- `$handler`: обработчик события; стандартный обратный вызов PHP.

В PHP у нас есть несколько способов определения обратного вызова:

- использовать глобальную функцию и просто передать ее в виде строки, как, например, `'my_function'`;
- использовать статический метода класса: `array('ClassName', 'staticMethodName')`;
- использовать метод объекта: `array($object, 'objectMethod')`;
- создать и передать анонимную функцию с помощью `create_function`, например:

```
$component->attachEventHandler('onClick',  
    create_function('$event', 'echo "Click!";'));
```

- начиная с PHP 5.3, вы можете использовать анонимные функции без `create_function`:

```
$component->attachEventHandler('onClick',  
    function($event){ echo "Click!"; })
```

При использовании `CComponent::attachEventHandler` обработчик события добавляется в конец списка обработчиков.

- Чтобы сократить объём кода, вы можете использовать свойства компонента для управления обработчиками событий следующим образом:

```
$component->onClick=$handler;  
// или:  
$component->onClick->add($handler);
```

- Для более точного управления обработчиками событий вы можете получить список обработчиков (`CList`) с помощью `CComponent::getEventHandlers` и работать с ним. Например, вы можете присоединить обработчик события так же, как с `attachEventHandler`, используя следующий код:

```
$component->getEventHandlers('onClick')->add($handler);
```

- Чтобы добавить обработчик события в начало списка обработчиков, используйте:

```
$component->getEventHandlers('onClick')->insertAt  
(0, $handler);
```

- Чтобы удалить определенный обработчик, можно использовать `CComponent::detachEventHandler` следующим образом:

```
$component->detachEventHandler('onClick', $handler);
```

- Кроме этого, можете получить список обработчиков, как было показано ранее, удалять обработчики из него.

`CComponent::hasEvent` проверяет, определено ли указанное событие в компоненте. `CComponent::hasEventHandler` проверяет, есть ли присоединенные к указанному событию.

Поскольку мы теперь знаем, как определить и использовать обработчики, давайте рассмотрим некоторые примеры из реальной жизни:

- для ускорения загрузки страницы принято сжимать данные при помощи `gzip` непосредственно перед отдачей. Если у вас есть доступ к тонкой настройке сервера, то вы можете этим воспользоваться, однако в некоторых средах, таких как виртуальный хостинг, вам это не удастся;

- к счастью, PHP может сжать вывод приложения при помощи буфера вывода и `ob_gzhandler`. Чтобы это сделать, мы должны начать буферизацию вывода при запуске приложения и освободить сжатый вывод, когда приложение завершится;
- у приложения Yii есть два события, которые пригодятся в данном случае: `CApplication::onBeginRequest` и `CApplication::onEndRequest`. Воспользуемся ими. Вставьте следующий код в файл `index.php` после настройки приложения, но перед его запуском:

```
...
require_once($yii);
$app = Yii::createWebApplication($config);
// присоединение обработчика к запуску приложения
Yii::app()->onBeginRequest = function($event)
{
    // запуск буферизации вывода с обработчиком gzip
    return ob_start("ob_gzhandler");
};
// присоединение обработчика к завершению приложения
Yii::app()->onEndRequest = function($event)
{
    // освобождение буфера вывода
    return ob_end_flush();
};
$app->run();
```

Есть множество удобных событий, определенных внутри классов ядра Yii. Вы можете получить их при помощи поиска текста «function on» в папке фреймворка, используя вашу любимую IDE.

Теперь давайте посмотрим на другой пример. В Yii вы можете перевести строки на различные языки с помощью `Yii::t`. В идеале все переводы должны быть актуальными. Если это не так, мы хотели бы получать сообщение по этому поводу на почту.

События снова приходят к нам на помощь. В частности, событие `CMessageSource::onMissingTranslation`, которое вызывается в случае, когда перевод для строки, передаваемой в `Yii::t`, отсутствует.

На этот раз мы будем использовать файл конфигурации приложения `protected/config/main.php`, чтобы присоединить обработчик события:

```
...
'components' => array(
```

```
...
// CPhpMessageSource класс компонента messages
по умолчанию
'messages' => array(
    // использование статического метода класса как
    обработчика события
    'onMissingTranslation' =>
array('MyEventHandler', 'handleMissingTranslation'),
),
...
}
```

Теперь мы должны реализовать наш обработчик. Создадим `protected/components/MyEventHandler.php`:

```
class MyEventHandler
{
    static function handleMissingTranslation($event)
    {
        // CMissingTranslationEvent - класс этого события,
        // поэтому мы можем получить дополнительную
        информацию о сообщении
        $text = implode("\n", array(
            'Language: '.$event->language,
            'Category: '.$event->category,
            'Message: '.$event->message
        ));
        // отправляем email
        mail('admin@example.com', 'Отсутствует перевод',
            $text);
    }
}
```

Рассмотрим ещё один пример. У нас есть блог, и мы должны отправить email, когда появляется новый комментарий (`Comment`) к записи блога (`Post`).

Комментарий (`Comment`) представляет собой стандартную модель AR, которая генерируется с помощью Gii. Запись (`Post`) – такая же модель, генерируемая Gii, за исключением некоторых изменённых методов. Нам понадобится пользовательское событие `NewCommentEvent` для хранения как модели записи (`Post`), так и комментария (`Comment`) и обработчик класса `Notifier`, который будет делать основную работу.

1. Давайте начнём с `protected/components/NewCommentEvent.php`:

```
class NewCommentEvent extends CModelEvent {
    public $comment;
    public $post;
}
```

Это довольно просто. Мы всего-лишь добавили два свойства.

2. Теперь давайте перейдем к `protected/models/Post.php`. Все стандартные методы AR опущены, чтобы сделать акцент на то, что было добавлено:

```
class Post extends CActiveRecord {
    // пользовательский метод для добавления
    // комментария к текущей записи
    function addComment(Comment $comment){
        $comment->post_id = $this->id;
        // создание экземпляра класса события
        $event = new NewCommentEvent($this);
        $event->post = $this;
        $event->comment = $comment;
        // запуск события
        $this->onNewComment($event);
        return $event->isValid;
    }
    // определение события onNewComment
    public function onNewComment($event) {
        // Событие на самом деле срабатывает здесь.
        // Таким образом, мы можем использовать
        // onNewComment вместо метода raiseEvent.
        $this->raiseEvent('onNewComment', $event);
    }
}
```

3. Теперь пришло время реализовать уведомления. Создаем `protected/components/Notifier.php` следующим образом:

```
class Notifier {
    function comment($event){
        $text = "Новый комментарий от
        {$event->comment->author} к записи
        {$event->post->title}";
        mail('admin@example.com', 'Новый
        комментарий', $text);
    }
}
```

4. Теперь пришло время собрать их вместе в `protected/controllers/PostController.php`:

```
class PostController extends CController
```

```
{
    function actionAddComment()
    {
        $post = Post::model()->findByPk(10);
        $notifier = new Notifier();
        // присоединение обработчика события
        $post->onNewComment = array($notifier, 'comment');
        // в настоящем приложении данные должны
        //   приходить из $_POST
        $comment = new Comment();
        $comment->author = 'Sam Dark';
        $comment->text = 'Yii events are amazing!';
        // добавление комментария
        $post->addComment($comment);
    }
}
```

5. После того как комментарий был добавлен, администратор получит уведомление об этом на email.

И ещё

Не всегда есть необходимость присоединять обработчик событий. Давайте посмотрим, как мы можем обработать событие, которое уже объявлено, внутри существующего компонента путём переопределения методов базового класса. Например, у нас есть форма модели `UserForm`, которая используется для сбора некой информации о нашем пользователе приложения, и мы должны получить имя и фамилию, введённые им.

К счастью, в `CModel`, который является базовым классом для всех моделей Yii, в том числе форм, определен метод `CModel::afterValidate`. Этот метод вызывается после успешной проверки формы. Используем его в нашей модели `protected/models/UserForm.php`:

```
class UserForm extends CFormModel
{
    public $firstName;
    public $lastName;
    public $fullName;
    public function rules()
    {
        return array(
            // Имя и фамилия обязательны к заполнению
            array('firstName, lastName', 'required'),
        );
    }
}
```



```

    }

    // аргумент $event - это экземпляр CEvent,
    // который был создан при вызове метода
    // события. На этот раз это произошло внутри
    // CModel::afterValidate().
    function afterValidate()
    {
        // Если этот метод был вызван, значит,
        // модель уже заполнена данными и данные
        // корректны
        $this->fullName = $this->firstName.'
            '.$this->lastName;

        // Важно вызвать метод родительского класса,
        // чтобы вызвались все остальные
        // обработчики события
        return parent::afterValidate();
    }
}

```

Мы должны вызвать родительский метод внутри `afterValidate`, потому что родительская реализация вызывает `onAfterValidate`, которая на самом деле иницирует события.

```

protected function afterValidate()
{
    $this->onAfterValidate(new CEvent($this));
}

```

Название метода события должно всегда быть определено как `function eventHandler ($event) {...}`, где `$event` - это экземпляр `CEvent`. Класс `CEvent` содержит только два свойства, которые называются `sender` и `handled`. Первое свойство содержит объект, который вызвал текущее событие, а второй может быть использован для предотвращения вызова всех остальных обработчиков, которые ещё не были выполнены, если установить его в `false`.

Описанный выше подход может быть использован для настройки моделей `Active Record` и реализации своей собственной модели поведения.

Дополнительно

Для получения дополнительной информации обратитесь к следующим ссылкам:

- <http://www.yiiframework.com/doc/api/CComponent/#raiseEventdetail>;

- <http://www.yiiframework.com/doc/api/CComponent/#attachEventHandler-detail>;
- <http://www.yiiframework.com/doc/api/CComponent/#getEventHandlers-detail>;
- <http://www.yiiframework.com/doc/api/CComponent/#detachEventHandler-detail>.

Смотрите также

- Рецепт «Использование getters и setters» в этой главе.
- Рецепт «Настройка компонентов».

Использование импорта и автозагрузки

При программировании на PHP одна из самых раздражающих вещей – это загрузка дополнительного кода при помощи `include` и `require`. К счастью, вы можете это автоматизировать с помощью загрузчика классов SPL (<http://php.net/manual/en/function.spl-autoload.php>).

Автозагрузка является одной из особенностей, на которых основывается Yii. Тем не менее по ней довольно много вопросов. Давайте выясним, как мы можем её использовать.

Когда мы используем класс, например `CDbCriteria`, мы не подключаем его явно, поэтому PHP изначально не может найти его и пытается полагаться на функцию автозагрузки; на автозагрузчик SPL, если быть точным. В большинстве случаев по умолчанию будет использоваться автозагрузчик Yii (`YiiBase::autoload`).

Ради скорости и простоты почти все основные классы фреймворка загружаются по мере необходимости без включения или импорта их в явном виде. Они перечислены в `YiiBase::$coreClasses`, так что загрузка основных классов происходит очень быстро. Классы Zii, такие как `CMenu`, классы расширений или свои собственные классы не загружаются автоматически, поэтому сначала мы должны их импортировать.

Для импорта классов мы будем использовать `Yii::import`:

- по умолчанию импорт не загружает класс немедленно;
- класс не загружается, если он не используется;
- класс не загружается два раза, поэтому безопасно импортировать один класс несколько раз.

Как это делается

1. Давайте предположим, что у нас есть специальный класс `LyricsFinder`, который находит слова для заданной песни. Мы поместили его в `protected/apis/lyrics/`, и в нашем `protected/controllers/TestController.php` мы постараемся использовать его следующим образом:

```
class TestController extends CController
{
    public function actionIndex($song)
    {
        $lyric = 'Nothing was found.';
        $finder = new LyricsFinder();
        if(!empty($song))
            $lyric = $finder->getText($song);
        echo $lyric;
    }
}
```

2. При выполнении этого кода мы получим следующее сообщение об ошибке PHP:

```
include(LyricsFinder.php) [<a href='function.
include'>function.include</a>]: failed to open
stream: No such file or directory.
```

3. На экране отображается ошибка автозагрузчика о том, что он не знает, где искать наш класс. Поэтому давайте изменим наш код:

```
class TestController extends CController
{
    public function actionIndex($song)
    {
        $lyric = 'Nothing was found.';

        // импортируем класс
        Yii::import('application.apis.lyrics.
                    LyricsFinder');

        $finder = new LyricsFinder();

        if(!empty($song))
            $lyric = $finder->getText($song);

        echo $lyric;
    }
}
```

Теперь наш код работает.

Встроенный загрузчик классов Yii требует, чтобы каждый класс был помещён в отдельный файл с таким же именем, как и сам класс.

Как это работает

Разберём, что такое `application.apis.lyrics.LyricsFinder`:

`application` – это стандартный псевдоним, который указывает на защищённую директорию вашего приложения и переведен в путь файловой системы. В следующей таблице приведены несколько стандартных псевдонимов:

Псевдоним	Путь
<code>application</code>	<code>path_to_webroot/protected</code>
<code>system</code>	<code>path_to_webroot/framework</code>
<code>zii</code>	<code>path_to_webroot/framework/zii</code>
<code>webroot</code>	<code>path_to_webroot</code>
<code>ext</code>	<code>path_to_webroot/protected/extensions</code>

Вы можете определить свои собственные псевдонимы, используя метод `Yii::setPathOfAlias`. Обычно это делается в первых строках `protected/config/main.php`, чтобы все остальные части конфигурации могли использовать эти новые псевдонимы.

`apis.lyrics` соответствует `apis/lyrics` и добавляются к пути, который получен из псевдонима `application`. `LyricsFinder` – имя класса, который мы хотим импортировать.

Если `LyricsFinder` требует некоторых дополнительных классов, расположенных в этом же каталоге, то мы можем использовать `Yii::import('application.apis.lyrics.*')`, чтобы импортировать всю папку. Обратите внимание, что `*` не включает вложенных папок, поэтому, если вам нужны `lyrics/includes`, вы должны добавить ещё один оператор импорта `Yii::import('application.apis.lyrics.includes.*')`.

Для повышения производительности лучше использовать явные пути с именем класса, а не `*` при импорте одного класса.

И ещё

Если вы хотите, чтобы ваши классы были импортированы автоматически как и классы ядра Yii, то вы можете настроить глобальный импорт в файле конфигурации `main.php`:

```
return array(  
    // ...  
    // глобальный импорт  
    'import'=>array(  
        'application.models.*',  
        'application.components.*',  
        'application.apis.lyrics.*',  
        'application.apis.lyrics.includes.*',  
        'application.apis.albums.AlbumFinder',  
    ),  
);
```

Заметим, что использование * с огромным количеством глобального импорта может снизить скорость вашего приложения из-за слишком большого количества каталогов для проверки.

Использование исключений

Исключения – это функция самого PHP, но они используются достаточно редко. Yii делает исключения очень полезными.

Существуют две основные области, где исключения Yii пригодятся:

1. Исключения позволяют упростить процесс обнаружения и исправления ошибок приложения в особых ситуациях, таких как сбои соединения базы данных или сбои работы API.
2. Исключения позволяют отправлять различные HTTP-ответы очень удобным способом.

Как правило, исключения должны быть выброшены тогда, когда компонент не может справиться с особой ситуацией и нужно обработать её на более высоком уровне.

Как это делается

1. Давайте предположим, что у нас есть класс `application/apis/lyrics/LyricsFinder.php`, который выполняет HTTP-запрос к API с помощью CURL и возвращает текст песни на основе её имени. Пример того, как мы можем использовать исключения внутри приложения:

```
// создадим несколько пользовательских исключений,  
// чтобы иметь возможность обработать их при  
// необходимости  
  
// общее исключение поисковика текста
```

```
class LyricsFinderException extends CException {}

// используется при проблеме с HTTP-соединением
class LyricsFinderHTTPException extends
    LyricsFinderException{}

class LyricsFinder
{
    private $apiUrl = 'http://example.com/
                        lyricsapi&songtitle=%s';

    function getText($songTitle)
    {
        $url = $this->getUrl($songTitle);
        $curl = curl_init();
        curl_setopt($curl, CURLOPT_URL, $url);
        curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
        $result = curl_exec($curl);

        // если возникла ошибка HTTP, тогда
        // выбрасываем исключение
        if($result===false)
        {
            $errorText = curl_error($curl);
            curl_close($url);
            throw new LyricsFinderHTTPException($errorText);
        }

        curl_close($curl);
        return $result;
    }

    private function getRequestUrl($songTitle)
    {
        return sprintf($this->apiUrl,
                        urlencode($songTitle));
    }
}
```

2. Поскольку мы не знаем, каким образом конкретному приложению необходимо обрабатывать свое соединение с API, мы оставим это на само приложение, выбрасывая пользовательское исключение `LyricsFinderHTTPException`. Обработаем его в нашем контроллере `protected/controllers/TestController.php`:

```
class TestController extends CController
{
    public function actionIndex($song)
    {
```

```
$lyric = 'Nothing was found.';

// импортируем класс api
Yii::import('application.apis.lyrics.
            LyricsFinder');

$finder = new LyricsFinder();

if(!empty($song))
{
    // Мы не хотим показывать пользователю
    // сообщение об ошибке.
    // Вместо этого мы извиним и пригласим
    // его повторить попытку позже.
    try {
        $lyric = $finder->getText($song);
    }
    // мы ищем здесь конкретное исключение
    catch (LyricsFinderHttpException $e)
    {
        echo 'Извините, сейчас мы не можем
            обработать ваш запрос.
            Повторите позже.';
    }
}

echo $lyric;
}
```

3. Другим использованием исключений Yii является создание различных HTTP-ответов путем выбрасывания `CHttpException`. Например, действие, которое отображает запись блога, представленную моделью `Post`, загруженной по ID, будет выглядеть следующим образом:

```
class PostController extends CController
{
    function actionView()
    {
        if(!isset($_GET['id']))
            // Если не указан идентификатор поста,
            // значит, запрос неправильный. Согласно
            // спецификации HTTP, код ответа 400.
            throw new CHttpException(400);

        // Поиск поста по его ID
        $post = Post::model()->findByPk($_GET['id']);

        if(!$post)
```

```
// Если нет поста с указанным ID, мы
// генерируем HTTP ответ с кодом
// 404 Not Found.
throw new CHttpException(404);

// Если все хорошо, отобразить пост
$this->render('post', array('model' => $post));
    }
}
```

Как это работает

Yii автоматически преобразовывает все *нефатальные* ошибки приложения в исключения `CException`.

Кроме того, обработчик исключения по умолчанию вызывает событие `onError` или `onException`. Обработчик событий по умолчанию записывает в журнал сообщение об ошибке уровня `error`. Кроме того, если значение константы `YII_DEBUG` приложения установлено в `true`, необработанное исключение или ошибка будет аккуратно отображена на экране. Он будет содержать стек вызовов, код области, где произошло исключение, а также файл и строки, в которых вы можете исправить это.

И ещё

Для получения дополнительной информации обратитесь к следующим ссылкам:

- <http://php.net/manual/en/language.exceptions.php>;
- <http://www.yiiframework.com/doc/api/CException/>;
- <http://www.yiiframework.com/doc/api/CHttpException/>.

Настройка компонентов

Yii – довольно гибкий фреймворк. Как и в любом гибком коде, должен быть удобный способ для настройки разных частей приложения. В Yii это осуществляется через конфигурационный файл `main.php`, расположенный в `protected/config/`.

Как это делается

Если раньше вы работали с Yii, то вы, скорее всего, уже настроили соединение с базой данных:


```
return array(
    ...
    'components'=>array(
        'db'=>array(
            'class'=>'system.db.CDbConnection',
            'connectionString'=>'mysql:host=localhost;
                               dbname=database_name',
            'username'=>'root',
            'password'=>'',
            'charset'=>'utf8',
        ),
        ...
    ),
    ...
);
```

Такой способ настройки компонента используется, когда вы хотите применять компонент во всех частях приложения. С данной конфигурацией вы можете получить доступ к компоненту по его имени, например `Yii::app()->db`.

Как это работает

При первом обращении к `Yii::app()->db` непосредственно или через модель Active Record, Yii создает компонент и инициализирует его публичные свойства соответствующими значениями, установленным в массиве `db` раздела `components` файла конфигурации приложения `main.php`. В предыдущем коде значение `'connectionString'` будет присвоено `CDbConnection::connectionString`, `'username'` будет назначен `CDbConnection::username` и т. д.

Если вы хотите узнать, что означает `'charset'`, и, что ещё можно настроить в компоненте `db`, то вы должны знать его класс. В случае с `db` его класс — `CDbConnection` и вы можете обратиться к API: <http://www.yiiframework.com/doc/api/CDbConnection/>.

В приведенном коде свойство `'class'` особенное, потому что оно используется для указания имени класса компонента. Оно не существует в классе `CDbConnection` и может быть использовано для переопределения класса следующим образом:

```
return array(
    ...
    'components'=>array(
        'db'=>array(
            'class'=>'application.components.
                    MyDbConnection',
```

```

        ),
        ...
    ),
    ...
);

```

Таким образом, вы можете настроить каждый компонент приложения, что очень полезно, когда стандартные настройки компонента не подходят для вашего приложения.

И ещё

Рассмотрим, какие стандартные компоненты приложения Yii можно настроить. Существуют два типа приложений в поставке с Yii:

- 1) веб-приложение (CWebApplication);
- 2) консольное приложение (CConsoleApplication).

Оба они расширяют класс CApplication, поэтому и консольное, и веб-приложение используют его компоненты.

Вы можете получить имена компонентов со страниц API (<http://www.yiiframework.com/doc/api/>) или из исходного кода метода registerCoreComponents, но давайте перечислим их здесь, чтобы использовать этот список в качестве справки.

И консольное, и веб-приложение упомянуты в следующей таблице:

Имя компонента	Класс по умолчанию	Описание
coreMessages	CPhpMessageSource	Является источником для перевода сообщений фреймворка
db	CDbConnection	Обеспечивает соединение с базой данных
messages	CPhpMessageSource	Является источником для перевода сообщений приложения
errorHandler	CErrorHandler	Обработывает ошибки PHP и необработанные исключения
securityManager	CSecurityManager	Предоставляет услуги по обеспечению безопасности, такие как хэширование, шифрование и т. д.
statePersister	CStatePersister	Предоставляет глобальные методы сохранения состояния
format	CFormatter	Предоставляет набор наиболее часто используемых методов форматирования данных
cache	CFileCache	Обеспечивает кэширование

Дополнительные компоненты, доступные только для веб-приложения, перечислены ниже:

Имя компонента	Класс по умолчанию	Описание
session	CSession	Обеспечивает работу с сессией
request	CHttpRequest	Инкапсулирует \$_SERVER и устраняет несоответствия между различными веб-серверами. Также управляет пользовательскими файлами cookie
urlManager	CUrlManager	URL-маршрутизатор, используется как для создания, так и для разбора ссылок приложения
assetManager	CAssetManager	Управляет публикацией ресурсов проекта
user	CWebUser	Представляет информацию о сессии пользователя
themeManager	CThemeManager	Управления темами приложения
authManager	CPhpAuthManager	Управляет доступом на основе ролей (RBAC)
clientScript	CClientScript	Управляет клиентскими скриптами (JavaScript и CSS)
widgetFactory	CWidgetFactory	Создает виджеты и поддерживает темы виджетов

Вы можете добавлять свои собственные компоненты приложения (классы должны наследовать CApplicationComponent), просто добавляя новые элементы конфигурации, при этом указывая класс компонента в свойстве 'class'.

Смотрите также

- Рецепт «Настройка виджетов по умолчанию» в этом разделе.

Настройка виджетов по умолчанию

При разработке на Yii, части кода, которые часто используются в представлениях, помещают в виджеты. Например, виджет может отобразить облако тегов или предоставить собственный тип формы

ввода. Основные виджеты легко конфигурируются и используются следующим образом:

```
<?php $this->widget('CLinkPager', array(
    'pages' => $pages,
    'pageSize' => 15,
))?>
```

В приведённом выше коде мы используем `$this->widget`, который вызывает виджет `CLinkPager` с набором параметров для отображения постраничной разбивки. `pages` и `PageSize` назначены соответствующим публичным свойствам `CLinkPager` до его отображения.

Обратите внимание, что в нашем примере мы увеличили количество элементов на странице до 15. Если мы хотим, чтобы наша постраничная навигация отображала 15 элементов на страницу на всех страницах нашего приложения, то мы будем должны назначить параметру `pageSize` значение 15 для всех вызовов виджета `CLinkPager`. Существует ли способ лучше? Определенно да.

Веб-приложение Yii предоставляет множество компонентов. Одним из них является фабрика виджетов (*widget factory*), которая с версии Yii 1.1.3 может быть использована для задания значений по умолчанию для виджетов.

- Давайте используем это, чтобы установить значение `pageSize` для всего приложения. Нам необходимо отредактировать файл конфигурации приложения `main.php` следующим образом:

```
return array(
    ...
    'components'=>array(
        'widgetFactory'=>array(
            'widgets'=>array(
                'CLinkPager'=>array(
                    'pageSize'=>15,
                ),
                ...
            ),
        ),
        ...
    ),
    ...
);
```

- Теперь значение по умолчанию для `pageSize` класса `CLinkPager` равно 15, поэтому, если мы опустим этот параметр для всех вызовов `CLinkPager` во всем приложении, он будет равен 15.

- Кроме того, мы все ещё можем изменить значение `pageSize` для конкретного виджета:

```
<?php $this->widget('CLinkPager', array(  
    'pages' => $pages,  
    'pageSize' => 5,  
))?>
```

Это работает так же, как правила CSS. Вы устанавливаете общий стиль во внешнем файле по умолчанию, но всё ещё можете изменить его с помощью встроенных стилей для отдельных виджетов.

Смотрите также

- Рецепт «Настройка компонентов» в этом разделе.

Использование коллекций ядра Yii

В Yii есть набор классов коллекций, которые используются в основном для внутренних целей и которые не описаны в полном руководстве, но всё же полезны для приложений:

- Списки: `CList`, `CTypedList`;
- Ассоциативные коллекции: `CMap`, `CAttributeCollection`;
- Очередь: `CQueue`;
- Стек: `CStack`.

Как это делается

Все структуры данных реализуют интерфейсы `SPL IteratorAggregate`, `Traversable` и `Countable`. Списки и коллекции также реализуют `SPL ArrayAccess`. Это позволяет использовать их как стандартные конструкции PHP. Ниже приводится отрывок из API `CList`:

- Следующий отрывок из API `CList`:

```
// добавить в конец  
$list[]=$item;  
  
// $index должен находиться между 0 и $list->count  
$list[$index]=$item;  
  
// удалить элемент с индексом $index  
unset($list[$index]);
```

```
// если в списке есть элемент с индексом $index
if(isset($list[$index]))

// пройтись по каждому элементу в списке
foreach($list as $index=>$item)

// возвращает количество элементов в списке
$n=count($list);
```

- CList – это коллекция с целочисленными индексами. По сравнению с родным массивом PHP, она добавляет строгие проверки, которые могут быть использованы в стиле OO, и позволяет реализовать доступ только для чтения:

```
$list = new CList();
$list->add('python');
$list->add('php');
$list->add('java')

if($list->contains('php'))
    $list->remove('java');

$anotherList = new CList(array('python', 'ruby'));
$list->mergeWith($anotherList);

$list->setReadOnly(true);
print_r($list->toArray());
```

- Существует ещё один список под названием CTypedList, который содержит только элементы определенного типа:

```
$typedList = new CTypedList('Post');
$typedList->add(new Post());
$typedList->add(new Comment());
```

Если мы пытаемся добавить комментарий в список постов, предыдущий код даст вам следующее исключение:

CTypedList<Post> может содержать только объекты класса Post.

- CMap позволяет использовать любое значение, число и не только в качестве ключа. Как и CList, он также может быть использован в родном стиле PHP, имеет почти тот же набор OO-методов, а также позволяет делать коллекцию доступной только для чтения:

```
$map = new CMap();
$map->add('php', array('facebook', 'wikipedia',
    'wordpress', 'drupal'));
```

```
$map->add('ruby', array('basecamp', 'twitter'));  
print_r($map->getKeys());
```

- Существует ещё один удобный статический метод с именем `CMap::mergeArray`, который может быть использован для рекурсивного слияния двух ассоциативных массивов с заменой скалярных значений:

```
$apps1 = array(  
    'apps' => array(  
        'task tracking',  
        'bug tracking',  
    ),  
    'is_new' => false  
);  
  
$apps2 = array(  
    'apps' => array(  
        'blog',  
        'task tracking',  
    ),  
    'todo' => array(  
        'buy milk',  
    ),  
    'is_new' => true  
);  
  
$apps = CMap::mergeArray($apps1, $apps2);  
CVarDumper::dump($apps, 10, true);
```

В результате мы получим:

```
array  
(  
    'apps' => array  
        (  
            '0' => 'task tracking'  
            '1' => 'bug tracking'  
            '2' => 'blog'  
            '3' => 'task tracking'  
        )  
    'is_new' => true  
    'todo' => array  
        (  
            '0' => 'buy milk'  
        )  
)
```

- `CAttributeCollection` включает в себя всю функциональность `CMap` и может работать с данными как со свойствами:

```
$col = new CAttributeCollection();

// $col->add('name','Alexander');
$col->name='Alexander';

// echo $col->itemAt('name');
echo $col->name;
```

- CQueue и CStack реализуют очередь и стек соответственно. Стек работает по принципу LIFO: последний пришел, первым ушел, а очередь – как FIFO: первый пришел, первым ушел. Так же как список и карта, коллекции они могут быть использованы в родном стиле PHP и иметь OO-методы:

```
$queue = new CQueue();

// добавить некоторые задания
$queue->enqueue(new Task('buy milk'));
$queue->enqueue(new Task('feed a cat'));
$queue->enqueue(new Task('write yii cookbook'));

// выполнить задание (удалить из очереди и вернуть его)
echo 'Done with ' . $queue->dequeue();
echo count($queue).' items left.';

// вернуть следующий элемент без удаления
echo 'Next one is ' . $queue->peek();

foreach($queue as $task)
    print_r($task);

$garage = new CStack();

// получение нескольких авто в гараж
$garage->push(new Car('Ferrari'));
$garage->push(new Car('Porsche'));
$garage->push(new Car('Kamaz'));

// Ferrari и Porsche не могут выехать, так как есть...
echo $garage->peek(); // Kamaz!

// в первую очередь мы должны получить КамАЗ
$garage->pop();

$porsche = $garage->pop();
$porsche->drive();
```


Работа с запросами

Вы можете работать непосредственно с данными запроса с помощью суперглобальных переменных PHP, таких как `$_SERVER`, `$_GET` или `$_POST`, но лучше всего использовать мощный класс `Yii::$app->getRequest()`. Давайте рассмотрим наиболее полезные методы, которые возвращают различные части текущего URL. В приведенной ниже таблице возвращаемые части выделены жирным шрифтом.

Как это делается

Вы можете получить компонент запроса в веб-приложении с помощью метода `Yii::$app()->getRequest()`. Давайте рассмотрим наиболее полезные методы, которые возвращают различные части текущего URL. В приведенной ниже таблице возвращаемые части выделены жирным шрифтом.

<code>getUrl</code>	<code>http://cookbook.local/test/index?var=val</code>
<code>getHostInfo</code>	<code>http://cookbook.local/test/index?var=val</code>
<code>getPathInfo</code>	<code>http://cookbook.local/test/index?var=val</code>
<code>getRequestUri</code>	<code>http://cookbook.local/test/index?var=val</code>
<code>getQueryString</code>	<code>http://cookbook.local/test/index?var=val</code>

Также полезна ещё одна группа методов. Методы, которые позволяют нам определить тип запроса `getIsPostRequest`, `getIsAjaxRequest` и `getRequestType`.

- Например, мы можем использовать `getIsAjaxRequest` для отдачи особенных данных:

```
class TestController extends CController
{
    public function actionIndex()
    {
        if(Yii::$app()->request->isAjaxRequest)s
            $this->renderPartial('test');
        else
            $this->render('test');
    }
}
```

В приведённом выше коде мы отображаем представление без макета, если запрос сделан через AJAX.

- В то время как PHP предоставляет суперглобальные перемен-

ные для POST и GET, способ Yii позволяет опустить некоторые дополнительные проверки:

```
class TestController extends CController
{
    public function actionIndex()
    {
        $request = Yii::app()->request;

        $param = $request->getParam('id', 1);
        // равнозначно
        $param = isset($_REQUEST['id']) ?
            $_REQUEST['id'] : 1;

        $param = $request->getQuery('id');
        // равнозначно
        $param = isset($_GET['id']) ? $_GET['id'] : null;

        $param = $request->getPost('id', 1);
        // равнозначно
        $param = isset($_POST['id']) ? $_POST['id'] : 1;
    }
}
```

- `getPreferredLanguage` пытается определить язык пользователя на основе отданных его браузером данных, что хоть и не может быть абсолютно точным, но хорошо подходит в качестве запасного варианта в случае, когда пользователь не указал язык вручную.

```
class TestController extends CController
{
    public function actionIndex()
    {
        $request = Yii::app()->request;
        $lang = $request->preferredLanguage;

        // пытается получить настройки языка из БД
        $criteria = new CDbCriteria();
        $criteria->compare('user_id', $request->
            getQuery('userid'));
        $criteria->compare('key', 'language');
        $setting = Settings::model()->
            find($criteria);
        if($setting)
            $lang = $setting->value;

        Yii::app()->setLanguage($lang);
    }
}
```

```
        echo Yii::t('app', 'Language is: ').$lang;
    }
}
```

- `sendFile` позволяет начать загрузку файла:

```
class TestController extends CController
{
    public function actionIndex()
    {
        $request = Yii::app()->getRequest();
        $request->sendFile('test.txt', 'File
            content goes here.');
```

Это действие вызовет загрузку файла и отправит все необходимые заголовки, в том числе тип содержимого (`mimetype`) и его длину. `MimeType`, если его не установить вручную третьим параметром, будет угадан на основе расширения имени файла.

- Последнее, что мы собираемся показать в этой главе, – это метод `getCookies`. Он возвращает экземпляр класса `CCookieCollection`, который позволяет нам работать с `cookie`. Так как `CCookieCollection` расширяет `CMap`, мы можем использовать некоторые родные методы PHP следующим образом:

```
class TestController extends CController
{
    public function actionIndex()
    {
        $request = Yii::app()->request;
        // получение cookie
        $cookie = $request->cookies['test'];
        if($cookie)
            // вывод значения cookie
            echo $cookie->value;
        else {
            // создание новой cookie
            $cookie=new CHttpCookie('test','I am a cookie!');
            $request->cookies['test'] = $cookie;
        }
    }
}
```

И ещё

Если вы работаете с большим количеством значений cookie и хотите укоротить код, то можете использовать следующий класс:

```
class Cookie
{
    public static function get($name)
    {
        $cookie=Yii::app()->request->cookies[$name];
        if(!$cookie)
            return null;

        return $cookie->value;
    }

    public static function set($name, $value,
        $expiration=0)
    {
        $cookie=new CHttpCookie($name,$value);
        $cookie->expire = $expiration;
        Yii::app()->request->cookies[$name]=$cookie;
    }
}
```

После помещения этого кода `protected/components/Cookie.php` вы сможете выполнить следующие действия:

```
class TestController extends CController
{
    public function actionIndex()
    {
        $cookie = Cookie::get('test');
        if($cookie)
            echo $cookie;
        else
            Cookie::set('test','I am a cookie!!');
    }
}
```



ГЛАВА 2.

Маршрутизация, контроллеры и представления

В этой главе мы рассмотрим:

- Правила маршрутизации.
- Автоматическая генерация URL-адресов.
- Регулярные выражения в правилах маршрутизации.
- Правила маршрутизации для статических страниц.
- Добавление правил маршрутизации в рабочее приложение.
- Базовый контроллер.
- Подключение внешних действий.
- Отображение статических страниц с использованием `CViewAction`.
- Использование flash-сообщений.
- Контекст контроллера в представлении.
- Повторное использование вложенных представлений.
- Клипы.
- Декораторы.
- Несколько макетов в приложении.
- Постраничная разбивка и сортировка данных.

Введение

Эта глава познакомит вас с некоторыми полезными возможностями маршрутизации, контроллеров и представлений в Yii. После ее изучения вы легко сможете сделать контроллеры и представления более гибкими.

Правила маршрутизации

В состав Yii входит действительно функциональный маршрутизатор, который решает две основные задачи: сопоставляет URL-адресам внутренние маршруты и генерирует ссылки для этих маршрутов. Описания правил маршрутизации разбросаны по официальному руководству и документации API. Попробуем разобраться на примере, как настроить правила для приложения.

Подготовка

1. Создайте новое приложение при помощи `yiic webapp`, как описано в официальной документации (<http://www.yiiframework.com/doc/guide/1.1/ru>), и найдите файл `protected/config/main.php`. В нем должны содержаться следующие строки:

```
// application components
'components'=>array(
    // раскомментируйте следующие строки, чтобы
    // включить ссылки в path-формате
    /*
    'urlManager'=>array(
        'urlFormat'=>'path',
        'rules'=>array(
            '<controller:\w+>/<id:\w+>'=>'<controller>/view',
            '<controller:\w+>/<action:\w+>/<id:\w+>'=>'<controller>/<action>',
            '<controller:\w+>/<action:\w+>'=>'<controller>/<action>',
        ),
    ),
```

2. Удалите всё из `rules`, мы собираемся начать с нуля.
3. В каталоге `protected/controllers` создайте `WebsiteController.php` со следующим содержимым:

```
class WebsiteController extends CController
{
    public function actionIndex()
    {
        echo "Главная страница";
    }
    public function actionPage($alias)
    {

```

```
        echo "Страница $alias.";
    }
}
```

Это контроллер нашего приложения, доступ к которому мы будем получать с использованием наших собственных URL-адресов.

4. Настройте веб-сервер для использования человекопонятных URL-адресов. Если используется Apache с модулем `mod_rewrite` и включенной директивой `AllowOverride`, достаточно добавить следующие строки в файл `.htaccess`, расположенный в корневом каталоге сервера:

```
Options +FollowSymLinks
IndexIgnore */*
RewriteEngine on
# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# otherwise forward it to index.php
RewriteRule . index.php
```

Как это делается

Наш сайт должен отображать домашнюю страницу по адресу `/home`, остальные страницы – по адресу `/page/<alias_страницы>`. Кроме того, адрес `/about` должен указывать на страницу с псевдонимом `about`.

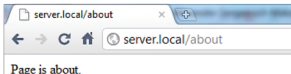
1. Добавим следующие строки в раздел `rules` файла `protected/config/main.php`:

```
'home' => 'website/index',
'<alias:about>' => 'website/page',
'page/<alias>' => 'website/page',
```

2. После сохранения изменений в браузере должны быть доступны следующие адреса:

- `/home`;
- `/about`;
- `/page/about`;
- `/page/test`.

На следующем скриншоте показана часть страницы, которая доступна по адресу `/about`:



Как это работает

Теперь рассмотрим, что было сделано и почему это работает. Начнем с правой части первого правила:

```
'home' => 'website/index',
```

Что же такое `website/index`?

В приложении Yii каждому контроллеру и его действию соответствует внутренний маршрут вида `moduleID/controllerID/actionID`. К примеру, метод `actionPage` контроллера `WebsiteController` соответствует маршруту `website/page`. То есть, чтобы получить идентификатор контроллера, следует отбросить от его имени постфикс `Controller` и заменить первую букву на строчную. Аналогично, чтобы получить идентификатор действия, следует отбросить префикс `action` у соответствующего метода и опять-таки начать со строчной буквы.

Итак, что же такое `home`?

Чтобы лучше понять, нам нужно знать хотя бы поверхностно, что же происходит, когда мы обращаемся к приложению, используя различные URL.

При обращении по адресу `/home` роутер перебирает правила маршрутизации, начиная с первого, и проверяет, соответствует ли введенный адрес этому правилу. Если совпадение найдено, роутер получает контроллер и его действие, которое соответствует внутреннему маршруту, и передает управление ему. То есть `/home` – это шаблон, определяющий, какие адреса будут обработаны этим правилом.

Чем меньше правил в приложении, тем меньше используется проверок для адресов, не требующих маршрутизации, а значит, и выше производительность.

И ещё

Вы можете создавать параметризованные правила, используя специальный синтаксис. Рассмотрим третье правило:


```
'page/<alias>' => 'website/page',
```

Здесь мы определяем параметр `alias`, который должен указываться в URL после `/page/`. Он может быть практически любым и будет передан в переменную `$alias` в `WebsiteController::actionPage($alias)`.

Вы можете определить шаблон для такого параметра – так мы сделали во втором правиле:

```
'<alias:about>' => 'website/page',
```

В этом случае псевдоним должен соответствовать шаблону `about`, иначе правило будет проигнорировано.

Полезные материалы

Для получения дополнительной информации обратитесь по следующим ссылкам:

- <http://www.yiiframework.com/doc/guide/ru/basics.controller>;
- <http://www.yiiframework.com/doc/guide/ru/topics.url>;
- <http://www.yiiframework.com/doc/api/1.1/CUrlManager>.

Смотрите также

- Автоматическая генерация URL-адресов.
- Регулярные выражения в правилах маршрутизации.
- Правила маршрутизации для статических страниц.
- Добавление правил маршрутизации в рабочее приложение.

Автоматическая генерация URL-адресов

Yii позволяет не только назначать адреса различным действиям контроллера, но и автоматически генерировать URL-адреса, указывая реальный внутренний маршрут и его параметры. Это действительно полезно, поскольку позволяет во время разработки сосредоточиться на внутренних маршрутах и выбрать реальные адреса непосредственно перед запуском проекта.

Никогда не указывайте URL-адреса непосредственно, для генерации ссылок используйте инструментарий Yii – это позволит изменять адреса без переписывания кода.

Подготовка

1. Создайте новое приложение, используя `yii webapp`, как описано в официальной документации, и найдите файл `protected/config/main.php`. Замените массив `rules` следующими строками:

```
// компоненты приложения
'components'=>array(
    ...
    // раскомментируйте следующий блок для включения ЧПУ
    'urlManager'=>array(
        'urlFormat'=>'path',
        'rules'=>array(
            '<alias:about>' => 'website/page',
            'page/about/<alias:authors>' => 'website/page',
            'page/<alias>' => 'website/page',
        ),
    ),
),
```

2. В каталоге `protected/controllers` создайте `WebsiteController` с содержимым:

```
class WebsiteController extends CController
{
    public function actionIndex()
    {
        echo "Главная страница";
    }
    public function actionPage($alias)
    {
        echo "Страница $alias.";
    }
}
```

Это контроллер приложения, для которого мы будем генерировать URL-адреса.

3. Настройте веб-сервер для работы с ЧПУ. Если используется Apache с модулем `mod_rewrite` и включенной директивой `AllowOverride`, достаточно добавить следующие строки в файл `.htaccess`, расположенный в корневом каталоге сервера:

```
Options +FollowSymLinks
IndexIgnore /*
RewriteEngine on
# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# otherwise forward it to index.php
RewriteRule . index.php
```

Как это делается

Нам нужно создать адреса, указывающие на действия `index` и `page` контроллера `WebsiteController`. В зависимости от того, в каком месте нам требуются ссылки, возможны различные способы их генерации, но основы при этом остаются неизменными. Перечислим некоторые методы, используемые для генерации URL.

`CHtml::link()` и ряд других методов `CHtml`, например `form`, `refresh` и `ajaxLink`, принимают URL и обычно применяются в представлениях. В них используется функция `CHtml::normalizeUrl` для определения внутреннего маршрута. Следует передавать данные в одном из следующих форматов:

- строка адреса URL – в этом случае будет использован переданный URL;
- массив `array` (внутренний маршрут, параметр => значение, параметр => значение, ...) – в этом случае нужный адрес будет создан автоматически.

Что такое внутренний маршрут? Каждому контроллеру и его действию соответствует маршрут в формате `moduleID/controllerID/actionID`. Например, метод `actionPage` контроллера `WebsiteController` соответствует маршруту `website/page`. Чтобы получить идентификатор контроллера, следует от его имени отбросить постфикс `Controller` и заменить первую букву на строчную. По аналогии, чтобы получить идентификатор действия, следует от имени метода отбросить префикс `action` и опять же заменить первую букву на строчную.

Параметры – это `$_GET`-переменные, передаваемые в действие, которое определяется указанным внутренним маршрутом. К примеру, если мы хотим создать ссылку на `WebsiteController::actionIndex`, которая передает `$_GET['name']`, это можно сделать так:

```
echo CHtml::link('Нажми меня!', array(
    'website/index', 'name' => 'Qiang'));
```

Использование внутренней адресации полезно и в контроллерах. В контроллере можно использовать функции `createUrl` и `createAbsoluteUrl` для генерации относительных и абсолютных адресов.

```
class WebsiteController extends CController
{
    public function actionTest()
```

```
{
    echo $this->createUrl('website/page', array(
        'alias' => 'about'));
    echo $this->createAbsoluteUrl('website/page',
        array('alias' => 'test'));
}
```

Поскольку у нас определены правила в конфигурации роутера, мы получим следующие адреса:

- /about
- <http://example.com/about>

Относительные адреса можно использовать внутри сайта, а абсолютные ссылки – в каналах RSS, электронной почте и т. д.

В случаях, когда отсутствует доступ к контроллеру, можно использовать методы приложения:

```
echo Yii::app()->createUrl('website/page', array(
    'alias' => 'about'));
echo Yii::app()->createAbsoluteUrl('website/page', array(
    'alias' => 'test'));
```

Различие заключается в том, что в методах внутри контроллера можно не указывать название модуля и контроллера – в этом случае будет использован текущий модуль и текущий контроллер:

```
class WebsiteController extends CController
{
    public function actionIndex()
    {
        // Поскольку мы находимся внутри контроллера,
        // createUrl посчитает, что адрес относится
        // к текущему контроллеру
        echo $this->createUrl('index');
    }
}
```

Как это работает

Все инструменты генерации URL-адресов, которые мы рассмотрели, используют метод `CWebApplication::createUrl`, который вызывает `CUrlManager::createUrl`. Он пытается применить правила маршрутизации по очереди, начиная с первого. Если ни одного соответствия не найдено, будет создан адрес по умолчанию.

Дополнительно

- <http://www.yiiframework.com/doc/guide/ru/basics.controller>;
- <http://www.yiiframework.com/doc/guide/ru/topics.url>;
- <http://www.yiiframework.com/doc/api/1.1/CUrlManager>;
- <http://www.yiiframework.com/doc/api/CHtml/#normalizeUrl-detail>;
- <http://www.yiiframework.com/doc/api/CHtml/#link-detail>;
- <http://www.yiiframework.com/doc/api/CController/#createUrl-detail>;
- <http://www.yiiframework.com/doc/api/CWebApplication/#createUrl-detail>.

Смотрите также

- Правила маршрутизации.
- Регулярные выражения в правилах маршрутизации.
- Правила маршрутизации для статических страниц.
- Добавление правил маршрутизации в рабочее приложение.

Регулярные выражения в правилах маршрутизации

Одной из скрытых особенностей роутера в Yii является возможность использования регулярных выражений – очень мощного инструмента для работы со строками.

Подготовка

1. Создайте новое приложение при помощи `yii create webapp`, как описано в официальной документации, и найдите файл `protected/config/main.php`. В нем должны содержаться следующие строки:

```
// компоненты приложения
'components'=>array(
    ...
    // раскомментируйте следующий блок для включения ЧПУ
    'urlManager'=>array(
        'urlFormat'=>'path',
        'rules'=>array(
            '<controller:\w+>/<id:\d+>'=>'<controller>/view',
            '<controller:\w+>/<action:\w+>/<id:\d+>'=>'<controller>/view',
        )
    )
);
```

```
d+>'=>'<controller>/<action>',  
        '<controller:\w+>/<action:\w+>'  
w+>'=>'<controller>/<action>',  
        ),  
    ),  
),
```

2. Удалите всё из rules, мы начинаем все заново.
3. В каталоге protected/controllers создайте PostController со следующим содержимым:

```
class PostController extends CController  
{  
    public function actionView($alias)  
    {  
        echo "Заметка с псевдонимом $alias.";  
    }  
    public function actionIndex($order = 'DESC')  
    {  
        echo "Заметки отсортированы в порядке $order.";  
    }  
    public function actionHello($name)  
    {  
        echo "Привет, $name!";  
    }  
}
```

Это контроллер нашего приложения, доступ к которому мы будем получать с использованием настраиваемых URL-адресов.

4. Сконфигурируйте сервер для использования чистых URL-адресов. Если используется Apache с установленным mod_rewrite и включенной директивой AllowOverride, следует добавить следующие строки в файл .htaccess, расположенный в корневой директории сервера:

```
Options +FollowSymLinks  
IndexIgnore */*  
RewriteEngine on  
# if a directory or a file exists, use it directly  
RewriteCond %{REQUEST_FILENAME} !-f  
RewriteCond %{REQUEST_FILENAME} !-d  
# otherwise forward it to index.php  
RewriteRule . index.php
```

Как это делается

Мы хотим, чтобы действия контроллера PostController принимали параметры в соответствии с определенными правилами и возвращали

HTTP-заголовок 404 для остальных параметров. Кроме того, индексная страница должна открываться по адресу `archive`.

Для реализации используем регулярные выражения:

```
'post/<alias:[-a-z]+>' => 'post/view',  
'(posts|archive)' => 'post/index',  
'(posts|archive)/<order:(DESC|ASC)>' => 'post/index',  
'sayhello/<name>' => 'post/hello',
```

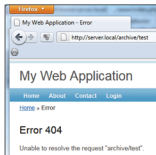
Теперь попробуем обратиться по следующим адресам:

```
// успешно  
http://example.com/post/test-post  
// ошибка  
http://example.com/post/another_post  
// успешно  
http://example.com/posts  
// успешно  
http://example.com/archive  
// ошибка  
http://example.com/archive/test  
// успешно  
http://example.com/posts/ASC
```

Следующий скриншот показывает, что адрес `http://example.com/post/test-post` выполняется успешно:



Нижеприведенный скриншот показывает, что адрес `http://example.com/archive/test` выполняется с ошибкой:



Как это работает

Можно использовать регулярные выражения не только в определении параметров, но и в любой части правила. Разберем правила по отдельности.

```
'post/<alias:[-a-z]+>' => 'post/view',
```

Параметр `alias` может состоять из одного и более символов английского алфавита или тире. Никакие другие символы не разрешены.

```
'(posts|archive)' => 'post/index',
```

И `posts`, и `archive` передадут управление на `post/index`.

```
'(posts|archive)/<order:(DESC|ASC)>' => 'post/index',
```

И `posts`, и `archive` передают управление на `post/index`. При этом параметр `order` может принимать только два значения – `DESC` или `ASC`.

```
'sayhello/<name>' => 'post/hello',
```

Следует указать параметр `name`, на который не налагается никаких ограничений.

Обратите внимание, что, несмотря на используемое правило, разработчик никогда не должен полагаться на то, что переданные данные безопасны.

Дополнительно

Чтобы узнать больше о регулярных выражениях, вы можете обратиться к следующим источникам:

- <http://www.php.net/manual/ru/reference.pcre.pattern.syntax.php>;
- *Mastering Regular Expressions*, автор Jeffrey Friedl. <http://regex.info/>.

Смотрите также

- Правила маршрутизации.
- Правила маршрутизации для статических страниц.

Правила маршрутизации для статических страниц

На большинстве сайтов присутствуют статические страницы – обычно это страницы «о компании», «контакты», «соглашения» и аналогичные. И, как правило, обработка этих страниц происходит в одном действии контроллера. Создадим правила маршрутизации для этого типа страниц.

Подготовка

1. Создайте новое приложение при помощи `yii create webapp`, как описано в официальной документации, и найдите файл `protected/config/main.php`. В нем должны содержаться следующие строки:

```
// компоненты приложения
'components'=>array(
    ...
    // раскомментируйте следующий блок для включения ЧПУ
    'urlManager'=>array(
        'urlFormat'=>'path',
        'rules'=>array(
            '<controller:\w+>/<id:\d+>'=>'<controller>/view',
            '<controller:\w+>/<action:\w+>/<id:\d+>'=>'<controller>/<action>',
            '<controller:\w+>/<action:\w+>'=>'<controller>/<action>',
        ),
    ),
),
```

2. Удалите всё из `rules`, мы начинаем все заново.
3. В каталоге `protected/controllers` создайте `WebsiteController` со следующим содержимым:

```
class WebsiteController extends CController
{
    public function actionPage($alias)
    {
        echo "Текущая страница: $alias. ";
    }
}
```

4. Настройте сервер для использования ЧПУ. Если используется Apache с установленным `mod_rewrite` и включенной директивой `AllowOverride`, следует добавить следующие строки в файл `.htaccess`, расположенный в корневой директории сервера:

```
Options +FollowSymLinks
IndexIgnore */*
RewriteEngine on
# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# otherwise forward it to index.php
RewriteRule . index.php
```

Как это делается

Наиболее простой способ – задать правила для каждой страницы:

```
'<alias:about>' => 'website/page',
'<alias:contact>' => 'website/page',
'<alias:tos>' => 'website/page',
```

Используя регулярные выражения, мы можем поместить всё в одно правило:

```
'<alias:(about|contact|tos)>' => 'website/page',
```

Как быть, если мы хотим, чтобы при обращении к адресу /tos параметр `alias` принимал значение `terms_of_service`?

Никаких проблем – в этом случае мы можем задать начальные значения:

```
'tos' => array('website/page', 'defaultParams' =>
array('alias' => 'terms_of_service')),
```

Отлично. Но как быть, если у нас множество страниц и требуется возможность динамически создавать страницы без добавления правил или изменения уже имеющихся?

Это можно реализовать, используя следующее правило:

```
'<alias>' => 'website/page'
```

Так как это правило соответствует любому адресу, его следует размещать последним, чтобы не перекрывать остальные правила. Кроме того, правила с одним слэшем (например, имя контроллера) перестанут работать. Чтобы исправить это, следует вставить перед последним правилом строки обработки правил по умолчанию, которые мы удалили на этапе подготовки.

Как это работает

Разберём правила, которые мы использовали.

```
'<alias:about>' => 'website/page',
```

Если указан URL-адрес /about, передадим его в качестве параметра alias в website/page.

```
'<alias:(about|contact|tos)>' => 'website/page',
```

Если URL-адрес принимает одно из значений /about, /contact или /tos, передадим его в качестве параметра alias в website/page.

```
'tos' => array('website/page', 'defaultParams' =>  
array('alias' => 'terms_of_service')),
```

Если указан URL-адрес /tos, передадим в параметре alias значение terms_of_service.

Последнее правило специфично, поскольку задает начальные параметры. Если необходимо определить дополнительные опции для правила, можно использовать следующий синтаксис:

```
'pattern' => array('internal/route', 'option' =>  
'value', 'option' => 'value', ...),
```

Список допустимых опций можно уточнить на странице документации: <http://www.yiiframework.com/doc/api/1.1/CUrlRule>.

Смотрите также

- Правила маршрутизации.
- Регулярные выражения в правилах маршрутизации.

Добавление правил маршрутизации в рабочее приложение

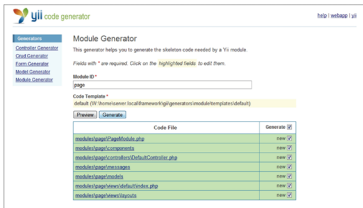
При разработке приложений с модульной архитектурой может появиться необходимость встроить правила маршрутизации для модуля в уже существующее приложение.

Подготовка

1. Создайте новое приложение при помощи `yiic webapp`.
2. В корне разместите файл .htaccess, как описано в официальной документации в разделе «Управление URL».
3. Добавьте параметр `'showScriptName' => false` в конфигурацию URL-менеджера.

4. Создайте модуль `page`, используя Gii.
5. Не забудьте добавить новый модуль в перечень модулей в конфигурации приложения.

Генератор кода Yii показан на скриншоте ниже:



Как это делается

1. Создадим файл `ModuleUrlManager.php` в каталоге `protected/components` со следующим содержанием:

```
<?php
class ModuleUrlManager
{
    static function collectRules()
    {
        if(!empty(Yii::app()->modules))
        {
            foreach(Yii::app()->modules as $moduleName =>
                $config)
            {
                $module = Yii::app()->getModule(
                    $moduleName);
                if(!empty($module->urlRules))
                {
                    Yii::app()->getUrlManager()->addRules(
                        $module->urlRules);
                }
            }
        }
    }
}
```

```
        return true;
    }
}
```

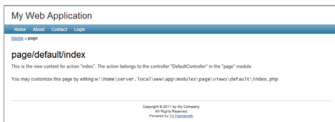
2. В конфигурацию приложения добавим следующую строку:

```
'onBeginRequest' => array('ModuleUrlManager',
    'collectRules'),
```

3. Теперь в модуле `page` можно добавлять собственные правила маршрутизации. Чтобы сделать это, добавим в файл `PageModule.php` строки:

```
public $urlRules = array(
    'test' => 'page/default/index',
);
```

4. Для проверки откроем в браузере страницу `http://example.com/test` – она содержит следующее:



5. Вы по-прежнему можете переопределить URL-правила, описанные в главном конфигурационном файле приложения. Все правила, указанные в `urlRules`, будут использованы только в том случае, если основные правила приложения не применялись.

Как это работает

Рассмотрим работу метода `ModuleUrlManager::collectRules`.

Если в нашем приложении определены модули, мы проверяем, существует ли `public`-свойство `urlRules`. Если да – значит в модуле определены правила, которые добавляются к основным с помощью `CUrlManager::addRules`.

В описании метода `CUrlManager::addRules` указано, что, для того чтобы добавленные правила работали, этот метод должен быть вызван ДО `CWebApplication::processRequest`.


```
        if ($cache)
            $cache->set('module.urls.' .
                $moduleName, $urlRules);
    }
    if (!empty($urlRules))
        Yii::app()->getUrlManager()->addRules
            ($urlRules);
    }
}
return true;
}
```

Такая реализация кеширует правила адресации для каждого модуля. Поэтому при добавлении нового модуля проблем не возникнет, однако в случае изменения существующих правил потребуется очистить кеш вручную.

Смотрите также

- Правила маршрутизации.

Базовый контроллер

Во многих фреймворках понятие базового расширяемого контроллера описано непосредственно в руководстве. В руководстве по Yii на этом не делается акцента, потому как вы можете достичь гибкости и другими способами. Однако использование базового контроллера возможно и иногда довольно полезно.

Подготовка

Создайте новое приложение при помощи `yiiic webapp`.

Допустим, мы хотим добавить контроллеры, которые будут доступны только для авторизованных пользователей. Конечно, можно задать ограничение отдельно для каждого контроллера, но мы сделаем лучше.

Как это делается

1. Для начала нам потребуется базовый контроллер, который будет использоваться в контроллерах, требующих авторизации. Создадим файл `SecureController.php` в `protected/components` со следующим содержанием:

```
<?php
class SecureController extends Controller
{
    public function filters()
    {
        return array(
            'accessControl',
        );
    }

    public function accessRules()
    {
        return array(
            array('allow',
                'users' => array('@'),
            ),
            array('deny',
                'users' => array('*'),
            ),
        );
    }
}
```

2. Теперь в Gii на странице генератора контроллера в поле Base Class вводим SecureController. В результате мы получим что-то вроде:

```
class TestController extends SecureController
{
    public function actionIndex()
    {
        $this->render('index');
    }
    ...
}
```

3. Теперь индексная страница TestController доступна только для авторизованных пользователей, однако в самом контроллере это явно не указано.

Как это работает

Весь трюк – не более чем обычное наследование классов: если фильтры или accessRules не найдены в TestController, они будут вызваны из SecureController.

Подключение внешних действий

В Yii можно определить действие контроллера как отдельный класс, а затем подключать его к контроллеру. Таким образом, можно многократно использовать схожую функциональность.

Например, можно вынести серверную часть для автоподсказки полей в отдельное действие и сэкономить время на повторном написании кода.

Мы рассмотрим ещё один простой пример – удаление модели.

Подготовка

1. Создайте новое приложение при помощи `yii webapp`.
2. Создайте схему базы данных запросом:

```
CREATE TABLE 'post' (  
    'id' int(10) unsigned NOT NULL auto_increment,  
    'created_on' int(11) unsigned NOT NULL,  
    'title' varchar(255) NOT NULL,  
    'content' text NOT NULL,  
    PRIMARY KEY ('id')  
);  
CREATE TABLE 'user' (  
    'id' int(10) unsigned NOT NULL auto_increment,  
    'username' varchar(200) NOT NULL,  
    'password' char(40) NOT NULL,  
    PRIMARY KEY ('id')  
);
```

3. Создайте модели `Post` и `User` при помощи `Gii`.

Как это делается

1. Сначала напомним обычное удаление для модели `Post`:

```
class PostController extends CController  
{  
    function actionIndex()  
    {  
        $posts = Post::model()->findAll();  
        $this->render('index', array('posts' => $posts));  
    }  
  
    function actionDelete($id)  
    {  
        $post = Post::model()->findPk($id);  
        if (!$post)  
            throw new CHttpException(404);  
  
        if ($post->delete())
```

```
        $this->redirect('post/index');  
        throw new CHttpException(500);  
    }  
}
```

Мы определили два действия. Первое отображает список всех заметок (`post`), второе – удаляет определенную заметку, если она существует, и перенаправляет обратно к действию `index`.

2. Теперь сделаем то же самое, но в отдельном классе. Создадим `DeleteAction.php` в каталоге `protected/components`:

```
class DeleteAction extends CAction  
{  
    function run()  
    {  
        if (empty($_GET['id']))  
            throw new CHttpException(404);  
        $post = Post::model()->findByPrimaryKey($_GET['id']);  
        if (!$post)  
            throw new CHttpException(404);  
        if ($post->delete())  
            $this->controller->redirect('post/index');  
        throw new CHttpException(500);  
    }  
}
```

3. Теперь используем его внутри нашего контроллера. Удалим действие `actionDelete` – оно больше не понадобится. После чего добавим метод `actions`:

```
class PostController extends CController  
{  
    function actions()  
    {  
        return array(  
            'delete' => 'DeleteAction',  
        );  
    }  
}
```

4. Отлично. Мы используем внешнее действие `delete` в `PostController`. Но, чтобы использовать `DeleteAction` в `UserController`, следует сначала его настроить.

```
class DeleteAction extends CAction  
{  
    public $pk = 'id';  
    public $redirectTo = 'index';  
}
```

```
public $modelClass;

function run()
{
    if (empty($_GET[$this->pk]))
        throw new CHttpException(404);

    $model = CActiveRecord::model($this->
        modelClass)->findByPk($_GET[$this->pk]);

    if (!$model)
        throw new CHttpException(404);

    if ($model->delete())
        $this->controller->redirect($this->redirectTo);

    throw new CHttpException(500);
}
}
```

5. Теперь мы можем использовать одно действие как в `PostController`, так и в `UserController`. Для `PostController` запишем так:

```
class PostController extends CController
{
    function actions()
    {
        return array(
            'delete' => array(
                'class' => 'DeleteAction',
                'modelClass' => 'Post',
            );
        );
    }
}
```

6. То же самое и для `UserController`:

```
class UserController extends CController
{
    function actions()
    {
        return array(
            'delete' => array(
                'class' => 'DeleteAction',
                'modelClass' => 'User',
            );
        );
    }
}
```

```
}  
}
```

7. Таким способом можно сэкономить кучу времени при реализации большого количества однотипных действий.

Как это работает

Любой контроллер можно составить из внешних действий, словно пазл из деталей. Особенность – в том, что подключаемое действие можно сделать очень гибким и использовать в нескольких местах. В конечной версии `DeleteAction` мы определили некоторые доступные свойства. Поскольку `DeleteAction` является компонентом, его свойства можно задавать в конфигурации. В примере мы передали конфигурацию в методе контроллера `actions`, который используется для добавления действий в модуль.

Дополнительно

- <http://www.yiiframework.com/doc/api/CAction/>;
- <http://www.yiiframework.com/doc/api/CController#actions-detail>.

Отображение статических страниц при помощи CViewAction

Если есть несколько статических страниц, которые не требуется часто изменять, не обязательно хранить их в базе данных и создавать для редактирования административный раздел.

Подготовка

Создайте новое приложение при помощи `yiic webapp`.

Как это делается

1. Достаточно подключить `CViewAction` к нашему контроллеру:

```
class SiteController extends CController  
{  
    function actions()  
    {  
        return array(  
            'page' => array(  
                'class' => 'CViewAction',
```

```

    }
    }
    );
}

```

2. Теперь разместим наши страницы в каталоге `protected/views/site/pages`. Назовем их `about.php` и `contact.php`.
3. Уже можно открыть страницы, набрав в браузере:
<http://example.com/index.php?r=site/page&view=contact> или
<http://example.com/site/page/view/about>, если в настройках приложения используются чистые адреса в формате `path`.

Как это работает

Мы используем внешнее действие `CViewAction`, которое просто пытается найти представление по названию, переданному в `$_GET`-параметре. Если файл найден – отображает его, если же нет – мы получим 404 ошибку.

Дополнительно

Некоторые полезные параметры `CViewAction` приведены в таблице.

Параметр	Описание
basePath	Псевдоним, который добавляется к имени представления. По умолчанию <code>pages</code> . Это означает, что страница с названием <code>faq.company</code> будет преобразована в <code>protected/views/pages/faq/company.php</code>
defaultView	Имя страницы, которая отображается, если требуемый <code>\$_GET</code> параметр не задан. По умолчанию – <code>index</code>
layout	Макет, используемый для отображения страницы. По умолчанию используется макет контроллера. Если установить равным <code>null</code> , страница будет отображена без макета
renderAsText	Если имеет значение <code>true</code> , страница будет отображена как есть. В противном случае имеющийся в ней PHP-код будет выполнен
viewParam	Имя <code>\$_GET</code> переменной, используемое в качестве параметра <code>CViewAction</code> для определения страницы. По умолчанию <code>view</code>

Полезные материалы

Для дополнительной информации обратитесь к следующему адресу: <http://www.yiiframework.com/doc/api/CViewAction>.

Смотрите также

- Подключение внешних действий.

Использование flash-сообщений

При редактировании модели с использованием форм, при удалении модели, да и вообще при любой операции неплохо бы сообщить пользователю, все ли в порядке или же имеются ошибки. Обычно после действий такого типа, как редактирование формы, происходит перенаправление и требуется отобразить сообщение на конечной странице. Как передать сообщение с текущей страницы на следующую, а затем – очистить? В этом нам помогут flash-сообщения.

Подготовка

Создайте новое приложение при помощи `yiiic webapp`.

Как это делается

1. Создадим контроллер `controllers/WebsiteController.php`:

```
class WebsiteController extends CController
{
    function actionOk()
    {
        Yii::app()->user->setFlash('success', 'Всё в порядке!');
        $this->redirect('index');
    }

    function actionBad()
    {
        Yii::app()->user->setFlash('error', 'Всё плохо!');
        $this->redirect('index');
    }

    function actionIndex()
    {
        $this->render('index');
    }
}
```

2. И добавим представление `protected/views/website/index.php`:

```
<?php if(Yii::app()->user->hasFlash('success')):?>
```

```
<div class="flash-notice">
    <?php echo Yii::app()->user->getFlash('success')?>
</div>
<?php endif?>
<?php if(Yii::app()->user->hasFlash('error')):>
<div class="flash-error">
    <?php echo Yii::app()->user->getFlash('error')?>
</div>
<?php endif?>
```

3. Теперь, если мы обратимся к странице <http://example.com/website/ok>, произойдет перенаправление на <http://example.com/website/index> с отображением успешного сообщения. Если же мы откроем страницу <http://example.com/website/bad>, то увидим ту же самую страницу, но с сообщением об ошибке. После обновления страницы сообщение пропадет.

Как это работает

Мы устанавливаем flash-сообщение, например `Yii::app()->user->setFlash('success', 'Всё в порядке!')`, обращаясь к методу `CWebUser::setFlash`. Он сохраняет сообщение в состоянии пользователя, так что на низшем уровне сообщение хранится в `$_SESSION` до тех пор, пока не будет обращения к методу `Yii::app()->user->getFlash('success')` и не будет удален соответствующий элемент массива `$_SESSION`.

Дополнительно

- <http://www.yiiframework.com/doc/api/1.1/CWebUser>.

Контекст контроллера в представлении

Представления в Yii довольно функциональны и имеют множество возможностей. Например, можно использовать контекст контроллера внутри представления. Давайте попробуем.

Подготовка

Создайте новое приложение при помощи `yiic webapp`.

Как это делается

1. Создадим контроллер:

```
class WebsiteController extends CController
{
    function actionIndex()
    {
        $this->pageTitle = 'Тест контекста контроллера';
        $this->render('index');
    }

    function hello()
    {
        if (!empty($_GET['name']))
            echo 'Привет, ' . $_GET['name'] . '!';
    }
}
```

2. Теперь создадим представление, демонстрирующее работу с контроллером:

```
<h1><?php echo $this->pageTitle?></h1>
<p>Приветствие. <?php $this->hello()?></p>
<?php $this->widget('zii.widgets.CMenu', array(
    'items' => array(
        array('label' => 'Home', 'url' =>
            array('index')),
        array('label' => 'Yiiframework home',
            'url' => 'http://yiiframework.ru/',
        ),
    ))?>
```

Как это работает

Используя `$this` в представлении, мы обращаемся к текущему контроллеру. То есть мы можем вызывать его методы и обращаться к его свойствам. Наиболее полезное свойство – `pageTitle`, содержащее заголовок текущей страницы. Кроме того, существует множество методов, которые незаменимы в представлениях, например `renderPartial` или `widget`.

Дополнительно

- <http://www.yiiframework.com/doc/api/CController>.

Повторное использование вложенных представлений

Yii поддерживает вложенные представления. Если есть блок без особой логики, который нужно использовать повторно, или требуется реализовать шаблоны для электронных писем – рекомендуется обратить внимание на данную возможность.

Подготовка

1. Создайте новое приложение при помощи `yii webapp`.
2. Создайте `WebsiteController`:

```
class WebsiteController extends CController
{
    function actionIndex()
    {
        $this->render('index');
    }
}
```

Как это делается

Начнём с создания самого блока. Например, нам нужно вставлять видео с сайта YouTube на различных страницах нашего сайта. Реализуем шаблон для повторного использования.

1. Создадим файл представления `protected/views/common/youtube.php` и вставим туда код для встраивания видео с сайта YouTube. Должно получиться что-то похожее:

```
<object width="480" height="385"><param name="movie"
value="http://www.youtube.com/v/S6u7ylr0zIg?fs=1"></
param><param name="allowFullScreen" value="true"></
param><param name="allowscriptaccess" value="always"></
param><embed src="http://www.youtube.com/v/S6u7ylr0zIg?fs=1"
type="application/x-shockwaveflash
"allowscriptaccess="always"
allowfullscreen="true" width="480" height="385"></
embed></object>
```

2. Теперь нужно сделать его пригодным для повторного использования. Добавим возможность задавать идентификатор видео и опциональное задание высоты и ширины:

```
<object width="<?php echo!empty($width) ? $width : 480?>"
height="<?php echo!empty($height) ? $height: 385?>">
```

```
<param name="movie" value="http://www.youtube.com/v/<?php echo $id?>?fs=1"></param><param name="allowFullScreen" value="true"></param><param name="allowscriptaccess" value="always"></param><embed src="http://www.youtube.com/v/<?php echo $id?>?fs=1" type="application/xshockwave-flash" allowscriptaccess="always" allowfullscreen="true" width="<?php echo !empty($width) ? $width : 480?>" height="<?php echo !empty($height) ? $height : 385?>"></embed></object>
```

3. Теперь его можно использовать в файле представления `protected/views/website/index.php` примерно так:

```
<?php $this->renderPartial('//common/youtube', array(
    'id' => '8Rp-CaIKvQs', // можно получить из
                           URL-адреса видео
    'width' => 320,
    'height' => 256,
))?>
```

Выглядит лучше, не правда ли? Обратите внимание, что мы использовали `//` при указании представления. Это означает, что Yii будет искать представление, начиная с `protected/views`, без участия имени контроллера.

4. Теперь отправим несколько писем. Поскольку мы физически не сможем написать уникальные письма тысячам пользователей, воспользуемся шаблоном, но сделаем его настраиваемым. Добавим в наш контроллер новый метод:

```
class WebsiteController extends CController
{
    function actionSendmails()
    {
        $users = User::model()->findAll();
        foreach ($users as $user)
        {
            $this->sendEmail('welcome', $user->email,
                'Добро пожаловать на сайт!', array(
                    'user' => $user));
        }
        echo 'Письма были отправлены.';
    }

    function sendEmail($template, $to, $subject, $data)
    {
        mail($to, $subject, $this->renderPartial(
            '//email/' . $template, $data, true));
    }
}
```

```
    }  
}
```

5. И наш шаблон:

```
Привет, <?php echo $user->name?>,  
Добро пожаловать на сайт!  
Вы можете просмотреть раздел наших новых видео.  
Там весёлые еноты.  
Ваша  
команда сайта.
```

Как это работает

`CController::renderPartial` обрабатывает шаблон так же, как и `CController::render`, с одним отличием – в последнем не используется макет. Поскольку в представлении мы можем обратиться к текущему контроллеру через переменную `$this`, мы также можем вызвать его метод `renderPartial`, чтобы использовать одно представление внутри другого. `renderPartial` также полезен при работе с AJAX, когда вам не нужно использовать макет.

Дополнительно

- <http://www.yiiframework.com/doc/api/CController/#renderPartialdetail>.

Смотрите также

- Контекст контроллера в представлении.

Клипы

Ещё одна особенность представлений в Yii – использование **клипов**. Основная идея состоит в том, что можно записать часть вывода и использовать ее в дальнейшем. Хорошим примером будет определение области содержимого в макете, а заполнение – в другом месте.

Подготовка

Создайте новое приложение при помощи `yiic webapp`.

Как это делается

1. Допустим, требуется определить две области в макете: `beforeContent` и `footer`. В представлении `protected/views/layouts/main.php` вставим следующее:

```
<?php if(!empty($this->clips['beforeContent']))  
echo $this->clips['beforeContent']?>
```

непосредственно перед выводом содержимого (`<?php echo $content; ?>`) и

```
<?php if(!empty($this->clips['footer']))  
echo $this->clips['footer']?>
```

внутри `<div="footer">`.

2. Теперь нам нужно как-то заполнить эти области. Мы будем использовать действие контроллера для области `beforeContent`. Откроем файл `protected/controllers/SiteController.php` и добавим следующий код в `actionIndex`:

```
$this->beginClip('beforeContent');  
echo 'Ваш IP-адрес: ' . Yii::app()->request->  
userHostAddress;  
$this->endClip();
```

3. Содержимое блока `footer` зададим из представления. Откроем `protected/views/site/index.php` и добавим следующее:

```
<?php $this->beginClip('footer')?>  
Сайт работает на Yii.  
<?php $this->endClip()?>
```

4. Теперь при открытии главной страницы сайта перед содержимым мы должны увидеть свой IP-адрес, а в подвале страницы – «работает на».

Как это работает

Мы добавили участки кода, которые проверяют существование указанного клипа и, если клип существует, выводят его. Затем мы записываем содержимое этих клипов, используя специальные методы контроллера `beginClip` и `endClip`.

Смотрите также

- Контекст контроллера в представлении

Декораторы

В Yii мы можем заключать содержимое в декоратор. Распространенным примером использования декораторов является макет. Ведь когда мы отображаем представление методом контроллера `render`, Yii автоматически помещает его в основной макет. Создадим простой декоратор, который будет правильно оформлять цитаты.

Подготовка

Создайте новое приложение при помощи `yii webapp`.

Как это делается

1. Сначала создадим файл декоратора `protected/views/decorators/quote.php`:

```
<div class="quote">
&ldquo;<?php echo $content?>&rdquo;; <?php echo $author?>
</div>
```

2. Теперь обратимся к нашему декоратору из `protected/views/site/index.php`:

```
<?php $this->beginContent('//decorators/quote',
array('author' => 'Edward A. Murphy'))?>
Если есть вероятность того, что какая-нибудь
неприятность может случиться, то она обязательно
произойдет
<?php $this->endContent()?>
```

3. Теперь на главной странице должен присутствовать такой код:

```
<div class="quote">
&ldquo; Если есть вероятность того, что какая-
нибудь неприятность может случиться, то она
обязательно произойдет &rdquo;; Edward A. Murphy
</div>
```

Как это работает

Принцип работы декораторов довольно прост. Всё, что заключено между `beginContent` и `endContent`, обрабатывается, помещается в переменную `$content` и передается в шаблон декоратора. Затем шаблон декоратора обрабатывается и вставляется в место вызова

endContent. Вторым параметром функции beginContent в декоратор можно передавать дополнительные переменные, что мы и сделали с параметром author.

Стоит отметить, что мы прописали путь к декоратору в таком относительном формате, который позволяет использовать различные темы оформления: `//decorators/quote` означает, что путь к файлу указан относительно текущего каталога представлений темы или же относительно каталога представлений всего приложения.

Дополнительно

- <http://www.yiiframework.com/doc/api/1.1/CContentDecorator/>.

Смотрите также

- Несколько макетов в приложении.
- Контекст контроллера в представлении.

Несколько макетов в приложении

Большинство приложений используют один макет для всех представлений. Однако возможны ситуации, когда требуются несколько макетов. Например, приложение может использовать различные макеты для различных страниц: две дополнительные колонки для блога, одна дополнительная колонка для раздела статей и раздел портфолио без дополнительных колонок.

Подготовка

Создайте новое приложение при помощи `yiic webapp`.

Как это делается

1. Создадим два макета в каталоге `/views/layouts`: `blog` и `articles`. В файл `blog` поместим следующий код:

```
<?php $this->beginContent('//layouts/main') ?>
<div>
    <?php echo $content?>
</div>
<div class="sidebar tags">
    <ul>
        <li><a href="#php">PHP</a></li>
        <li><a href="#yii">Yii</a></li>
```

```
        </ul>
    </div>
    <div class="sidebar links">
        <ul>
            <li><a href="http://yiiframework.com/"
                >Yiiframework</a></li>
            <li><a href="http://php.net/">PHP</a></li>
        </ul>
    </div>
    <?php $this->endContent()?>
```

2. Файл articles:

```
<?php $this->beginContent('//layouts/main') ?>
<div>
    <?php echo $content?>
</div>
<div class="sidebar toc">
    <ul>
        <li><a href="#intro">1. Introduction</a></li>
        <li><a href="#quick-start">2. Quick start</a></li>
    </ul>
</div>
<?php $this->endContent()?>
```

3. Создадим три контроллера BlogController, ArticleController и PortfolioController, для каждого зададим actionIndex:

```
class BlogController extends Controller
{
    function actionIndex()
    {
        $this->layout = 'blog';
        $this->render('//site/index');
    }
}

class ArticleController extends Controller
{
    function actionIndex()
    {
        $this->layout = 'articles';
        $this->render('//site/index');
    }
}

class PortfolioController extends Controller
{

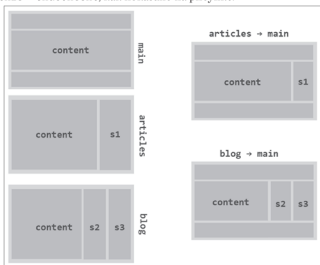
```

```
function actionIndex()
{
    $this->render('//site/index');
}
```

4. Теперь откроем <http://example.com/blog>, <http://example.com/article> и <http://example.com/portfolio>.

Как это работает

Мы задали два дополнительных макета для блога и статей. Поскольку нежелательно дублировать общие части из основного макета, применим дополнительные декораторы, используя `$this->beginContent` и `$this->endContent`, как показано на рисунке:



Так мы используем представление, полученное в результате отображения макета `articles`, в переменной `$content` макета `main`.

Смотрите также

- Контекст контроллера в представлении.
- Декораторы.

Постраничная разбивка и сортировка данных

В последних версиях Yii акцент был смещён от непосредственного использования Active Record к использованию гридов, списков и провайдеров данных. Тем не менее иногда лучше использовать Active Record напрямую. Давайте рассмотрим, как вывести список AR-записей с постраничной разбивкой и возможностью сортировки.

Подготовка

1. Создайте новое приложение при помощи `yiiic webapp`.
2. Создайте в базе данных таблицу `post` с полями `id` и `title` и добавьте в нее 10–20 записей.
3. Создайте модель `Post` при помощи `Gii`.

Как это делается

1. Для начала нужно создать `PostController`:

```
class PostController extends Controller
{
    function actionIndex()
    {
        $criteria = new CDbCriteria();
        $count = Post::model()->count($criteria);
        $pages = new CPagination($count);

        // элементов на страницу
        $pages->pageSize = 5;
        $pages->applyLimit($criteria);

        // сортировка
        $sort = new CSort('Post');
        $sort->attributes = array(
            'id',
            'title',
        );
        $sort->applyOrder($criteria);
        $models = Post::model()->findAll($criteria);
        $this->render('index', array(
            'models' => $models,
            'pages' => $pages,
            'sort' => $sort,
        ));
    }
}
```

```
    }  
}
```

2. Теперь займемся `protected/views/post/index.php`:

```
<p><?php echo $sort->link('id')?></p>  
<p><?php echo $sort->link('title')?></p>  
<ol>  
    <?php foreach ($models as $model): ?>  
        <li>  
            <h2><?php echo $model->id?> - <?php echo  
                $model->title?></h2>  
        </li>  
    <?php endforeach?>  
</ol>  
<?php $this->widget('CLinkPager', array(  
    'pages' => $pages,  
))?>
```

3. При открытии страницы `http://example.com/post` должна появиться постраничная разбивка и ссылки для сортировки по полям ID или title.

Как это работает

Сначала мы получаем общее число моделей и используем его при инициализации экземпляра компонента постраничной разбивки (`CPagination`). Затем, используя метод `applyLimit`, мы задаем смещение и количество записей для условия, которое мы использовали при подсчете. После этого создаем экземпляр класса `CSort` для модели, задаем атрибуты модели, по которым будет производиться сортировка, и добавляем условие сортировки к `$criteria`, вызывая `applyOrder`. Далее передаем измененную переменную `$criteria` в метод `findAll`. На этом этапе у нас есть список моделей, данные о страницах, используемые для постраничной навигации, и сортировщик, который мы используем для создания ссылок.

В представлении мы используем полученные данные. Сначала создаем ссылки методом `CSort::link`. Затем отображаем список моделей. И в завершение, используя виджет `CLinkPager`, выводим ссылки на страницы.

Дополнительно

- <http://www.yiiframework.com/doc/api/CPagination/>.
- <http://www.yiiframework.com/doc/api/CSort/>.



ГЛАВА 3. AJAX и jQuery

В этой главе мы рассмотрим следующие темы:

- Загрузка блока через AJAX.
- Управление ресурсами.
- Подключение ресурсов.
- Работа с JSON.
- Передача параметров из PHP в JavaScript.
- Обработка переменного числа полей в форме.

Введение

Клиентская часть Yii построена с помощью jQuery – наиболее популярного JavaScript фреймворка, который сочетает в себе мощность, удобство и легкость в изучении. В этой главе мы сосредоточимся в большей степени на специфичных для Yii приемах, нежели на особенностях jQuery. Если вы хотите больше узнать о jQuery – обратитесь к документации (<http://docs.jquery.com/>).

Загрузка блока через AJAX

Загрузка части страницы асинхронно является широко распространенной практикой на современных сайтах. Давайте реализуем блок, который отображает случайную цитату на странице сайта, а также имеет ссылку «Следующая цитата».

Подготовка

- Создайте новое приложение, используя `yiiic webapp`, как описано в официальном руководстве.

- Включите опцию 'urlFormat'=>'path' в компоненте urlManager для использования человекопонятных URL.

Как это делается

Выполните следующие шаги:

1. Создайте новый контроллер `protected/controllers/QuoteController.php`:

```
<?php
class QuoteController extends Controller {
    private $quotes = array(
        array('Ходить по воде и разрабатывать
программы, следуя спецификации, очень просто...
если они заморожены.', 'Эдвард В. Верард'),
        array('Любое дело всегда длится дольше, чем
ожидается, даже если учесть закон Хофштадтера.',
'закон Хофштадтера'),
        array('Всегда пишите код так, будто
сопровождать его будет склонный к насилию психопат,
который знает, где вы живете.', 'Рик Осборн'),
        array('Я всегда мечтал о том, чтобы моим
компьютером можно было пользоваться так же легко,
как телефоном; моя мечта сбылась: я уже не могу
разобраться, как пользоваться моим телефоном.',
'Бьерн Страуструп'),
        array('Измерять продуктивность программирования
подсчетом строк кода – это так же, как оценивать
постройку самолета по его весу.', 'Билл Гейтс'),
    );

    private function getRandomQuote() {
        return $this->quotes[array_rand($this->quotes, 1)];
    }

    function actionIndex() {
        $this->render('index', array(
            'quote' => $this->getRandomQuote()));
    }

    function actionGetQuote() {
        $this->renderPartial('_quote', array(
            'quote' => $this->getRandomQuote()),);
    }
}
```

2. Выше мы подключаем два представления, первое – `protected/views/quote/index.php`:

```

<h2>Цитата дня</h2>
<div id="quote-of-the-day">
  <?php $this->renderPartial('_quote', array(
    'quote' => $quote))?>
</div>
<?php echo CHtml::ajaxLink('Следующая цитата',
  array('getQuote'), array('update' =>
    '#quote-of-the-day'))?>

```

Следующее – `protected/views/quote/_quote.php`:

```

&ldquo;<?php echo $quote[0]?>&rdquo;;
<?php echo $quote[1]?>

```

3. Вот и всё. Теперь попробуйте открыть новый контроллер и кликнуть на ссылку **Следующая цитата**.



Как это работает

Сначала мы определяем список цитат в `private`-свойстве `$quotes` и создаем метод для получения случайной цитаты. В реальном приложении вы, скорее всего, получали бы цитату из базы, используя DAO или Active Record.

Затем мы определяем представление для действия `index` и частичное представление `_quote` для действия `getQuote`, причем последнее используется без общего шаблона с помощью метода `renderPartial`. В представлении `index` мы создаем ссылку при помощи `CHtml::ajaxLink`. По нажатии на неё отправляется запрос к `getQuote` и обновляется HTML-блок с ID `quote-of-the-day`. Обновление содержимого HTML-блока данными, пришедшими в ответ на AJAX-запрос, сделано с помощью следующего JavaScript-кода (форматирование изменено):

```

<script type="text/javascript">
/*<![CDATA[*]
jQuery(function($) {
  jQuery('body').delegate('#yt0','click',function() {
    jQuery.ajax({

```

```
'url': '/quote/getQuote',
'cache': false,
'success': function(html){ jQuery
("#quote-of-the-day").html(html) }
});
return false;
});
}); /*]]>*/
</script>
```

Данный код подключается Yii на страницу автоматически, причем только один раз, вне зависимости от того, как много раз мы его используем.

Для ссылки Yii автоматически генерирует ID `#yt0`, нам не приходится заботиться о значении атрибута ID и задавать его вручную. Тем не менее, если вы загружаете через AJAX представление, которое содержит виджеты с JavaScript или AJAX хелперы CHtml, вам необходимо задать все ID вручную, для того чтобы избежать возможных конфликтов с уже имеющимися на странице элементами.

Дополнительно

Вы можете изменить стандартный обработчик события success, задав третий параметр метода CHtml::ajaxLink следующим образом:

```
<?php echo CHtml::ajaxLink('Следующая цитата', array('getQuote'),
array('success' => new CJavaScriptExpression(function(data){
    alert (data);
}))) ?>
```

Обратите внимание, что мы использовали класс CJavaScriptExpression, который необходим, если вы хотите использовать JavaScript-код (как в примере выше), а не обычную строку.

В некоторых случаях нежелательно оставлять возможность обращаться к действию getQuote посредством не AJAX-запроса. Есть два способа ограничить доступ к действию. Первый – использование встроенного фильтра ajaxOnly:

```
<?php
class QuoteController extends Controller {
    function filters() {
        return array('ajaxOnly + getQuote',);
    }
}
```

После добавления данного фильтра, при прямом обращении к `getQuote`, будет автоматически отправляться HTTP-ответ с кодом 400: 400 Bad Request.

Второй способ принимать исключительно AJAX-запросы – определить тип запроса с помощью специального метода компонента `request`. Например, если мы хотим показать стандартную страницу 404 «Not found», то можем сделать это следующим образом:

```
function actionGetQuote() {  
    if (!Yii::app()->request->isAjaxRequest)  
        throw new CHttpException(404);  
  
    $this->renderPartial('_quote', array(  
        'quote' => $this->getRandomQuote()));  
}
```

При помощи этого же подхода мы можем использовать действие `getQuote` для обслуживания как AJAX, так и обычных запросов:

```
function actionGetQuote() {  
    $quote = $this->getRandomQuote();  
    if (Yii::app()->request->isAjaxRequest) {  
        $this->renderPartial('_quote', array(  
            'quote' => $quote));  
    } else {  
        $this->render('index', array(  
            'quote' => $quote));  
    }  
}
```

Предотвращение подключения встроенного jQuery

Иногда возникает необходимость предотвратить подключение jQuery, поставляемого вместе с Yii (например, когда ваш сайт требует какой-то конкретной версии jQuery). Это можно сделать, сконфигурировав компонент `clientScript` надлежащим образом в файле конфигурации `protected/config/main.php`:

```
array(  
    // ...  
    'components'=>array( // компоненты приложения  
        // ...  
        'clientScript' => array(  
            'scriptMap' => array(  
                'jquery.js' => false,  
                'jquery.min.js' => false)  
            )  
        ),  
    // ...  
)
```

```
);
```

Полезные материалы

За дополнительной информацией обратитесь к следующим ресурсам:

- <http://api.jquery.com/>;
- <http://docs.jquery.com/Ajax/jQuery.ajax#options>;
- <http://www.yiiframework.com/doc/api/CHtml#ajax-detail>;
- <http://www.yiiframework.com/doc/api/CHtml#ajaxButton-detail>;
- <http://www.yiiframework.com/doc/api/CHtml#ajaxSubmitButton-detail>;
- <http://www.yiiframework.com/doc/api/CHtml#ajaxLink-detail>.

Смотрите также

- Рецепт «Работа с JSON».
- Рецепт «Передача параметров из PHP в JavaScript».

Управление ресурсами

Возможность управления ресурсами (*assets*) – одна из важнейших возможностей Yii. Она особенно полезна в следующих случаях:

- когда вы хотите создать расширение, которое использует собственные файлы JavaScript, CSS и изображения, при этом хранит их в своей же директории, недоступной из веб;
- когда вам необходимо обрабатывать ресурсы перед тем, как отдать их клиенту: объединять, сжимать, обфусцировать скрипты и стили;
- когда вы используете на странице один ресурс многократно и не хотите его дублирования.

В то время как первые два пункта могут рассматриваться как приятные особенности, третий решает проблемы многократного использования виджетов.

Давайте создадим простой виджет для событий Facebook, который будет использовать собственные JavaScript, CSS и изображения.

Подготовка

- Создайте новое приложение, используя `yiic webapp`, как описано в официальном руководстве.

- Убедитесь, что папка `assets` расположена в корневой директории вашего приложения (где расположен `index.php`) и у пользователя, из-под которого запущен PHP, есть права на запись. Именно сюда будут сохраняться публикуемые ресурсы.
- Сгенерируйте и загрузите анимированное изображение с сайта <http://ajaxload.info/>.

Как это делается

Для начала спланируем предстоящую работу. В Yii вы можете поместить свои виджеты в любую директорию, обычно это `protected/components`. В ней вполне приемлемо иметь один или два класса, но когда их число увеличивается, возникает ненужный хаос. Чтобы его избежать, поместим наш виджет в отдельную директорию `protected/extensions/facebook_events`. Создайте папку `assets` внутри виджета и поместите туда файл `ajax-loader.gif`, который вы недавно загрузили. Также создайте файл `facebook_events.css` и `facebook_events.js` в той же папке.

1. Начнем непосредственно с класса самого виджета `protected/extensions/facebook_events/EFacebookEvents.php`:

```
<?php
class EFacebookEvents extends CWidget {
    public $keyword;
    private $loadingImageUrl;
    protected $url = "https://graph.facebook.com/
        search?q=%s&type=event&callback=?";

    protected function getUrl() {
        return sprintf($this->url, urlencode(
            $this->keyword));
    }

    public function init() {
        $assetsDir = dirname(__FILE__) . '/assets';
        $cs = Yii::app()->getClientScript();
        $cs->registerCoreScript("jquery");

        // Публикация и регистрация JavaScript-файла
        $cs->registerScriptFile(
            Yii::app()->assetManager->publish(
                $assetsDir . '/facebook_events.js'),
            CClientScript::POS_END);

        // Публикация и регистрация CSS-файла
```

```

$cs->registerCssFile(
    Yii::app()->assetManager->publish(
        $assetsDir.'/facebook_events.css'));

// Публикуем картинку. Метод publish возвращает
// URI картинки, по которому она будет доступна
$this->loadingImageUrl = Yii::app()->
    assetManager->publish(
        $assetsDir.'/ajax-loader.gif' );
}

public function run() {
    $this->render("body", array(
        'url' => $this->getUrl(),
        'loadingImageUrl' => $this->loadingImageUrl,
        'keyword' => $this->keyword
    ));
}
}

```

2. Теперь зададим отображение, которое будет использоваться в методе run – protected/extensions/facebook_events/views/body.php:

```

<div class="facebook-events" data-url="php echo $url?&gt;"&gt;
    &lt;h2&gt;<?php echo $keyword?&gt; events&lt;/h2&gt;
    &lt;div class="data"&gt; &lt;?php echo CHtml::image(
        $loadingImageUrl)?&gt; &lt;/div&gt;
&lt;/div&gt;
</pre

```

3. Также нам понадобится поместить следующий код в файл facebook_events.js:

```

jQuery(function($) {
    $(".facebook-events").each(function(){
        var url = $(this).data("url");
        var container = $(".data", this);
        $.getJSON(url, function(json){
            var html = "<ul>";
            $.each(json.data, function() {
                html += "<li>" + "<p><strong>" +
                    this.name + "</strong></p><p>" +
                    this.location + "</p></li>";
            });
            html += "</ul>";
            container.html(html);
        });
    });
});

```

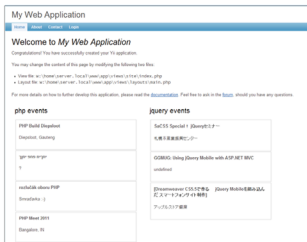
4. И немного стилей для виджета в `facebook_events.css`:

```
.facebook-events { padding: 10px; width: 400px;
                    float: left; }
.facebook-events ul { padding: 0; }
.facebook-events li { list-style: none;
                    border: 1px solid #ccc;
                    padding: 10px;
                    margin: 2px; }
```

5. Вот и все, виджет готов. Давайте приступим к его использованию. Откройте файл `protected/views/site/index.php` и подключите виджет следующим образом:

```
<?php $this->widget("ext.facebook events.
EFacebookEvents", array( 'keyword' => 'php', ))?>
<?php $this->widget("ext.facebook events.
EFacebookEvents", array( 'keyword' => 'jquery', ))?>
```

6. Теперь на главной странице приложения должны появиться два блока событий Facebook с заголовками **php events** и **jquery events**, как показано на скриншоте:



Как это работает

При вызове `$this->widget` в шаблоне `site/index` запускаются два метода класса `EFacebookEvents`: `init`, который публикует ресурсы

и подключает их на страницу, и `run`, который отображает HTML-содержимое виджета. Первым делом мы вызываем `CAssetManager::publish`, чтобы опубликовать ресурсы виджета в директории `assets`, доступной из веб. Метод возвращает URL, по которому будет доступен ресурс. В случае JavaScript и CSS мы используем методы `CClientScript`, которые добавляют необходимые тэги `<script>` и `<style>` и предотвращают дублирование. Что касается изображений, то мы передаем их URL в представление `body`, где генерируем необходимый HTML с помощью `CHtml::image`. Когда JavaScript загружается, он выполняет запрос к Facebook API (см. описание <http://developers.facebook.com/docs/api>) и помещает полученные данные в нужный блок.

И ещё

Есть ещё некоторые аспекты использования ресурсов в Yii.

Что находится внутри директории `assets`?

Давайте взглянем в директорию `assets`. Содержимое должно выглядеть подобно следующему:

```
assets/  
  1a6630a0/  
    main.css  
  2bb97318/  
    pager.css  
  4ab2ffe/  
    jquery.js
```

Директории вида `1a6630a0` используются для предотвращения коллизий файлов с одинаковыми именами из разных папок. Имя папки внутри `assets` – хеш полного пути к публикуемой папке. Поэтому ресурсы из одной директории располагаются в одной папке внутри `assets`. Это значит, что при публикации CSS-стилей и изображений вы можете использовать относительные пути к изображениям внутри CSS-файлов.

Публикация содержимого директории

С помощью `CAssetManager::publish` вы можете рекурсивно опубликовать содержимое директории. Отличие состоит в том, что при публикации одного файла он обновляется при изменении, в то время как содержимое директории – нет.

Полезные материалы

За дополнительной информацией обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/CAssetManager>;
- <http://www.yiiframework.com/doc/api/CClientScript>;
- <http://www.yiiframework.com/doc/api/CHtml#asset>.

Смотрите также

- Рецепт «Создание виджета» в главе 8.

Подключение ресурсов

В Yii имеется специальный класс `CClientScript`, позволяющий подключать необходимые скрипты, стили и прочие ресурсы.

Как это делается

Начнем с подключения скрипта. Есть три типа скриптов: внешние скрипты, скрипты ядра фреймворка и встроенные скрипты.

1. Внешний скрипт располагается в файле, доступном по определенному URL. Например, чтобы подключить скрипт с URL <http://example.com/js/main.js>, вы можете использовать следующий код:

```
Yii::app()->clientScript->registerScriptFile  
("http://example.com/js/main.js");
```

2. При публикации скрипта вы можете задать место, в котором он должен быть подключен, используя одну из следующих констант в качестве второго параметра:

<code>CClientScript::POS_HEAD</code>	Внутри тега <code><head></code> прямо перед тегом <code><title></code>
<code>CClientScript::POS_BEGIN</code>	Сразу после открывающего тега <code><body></code>
<code>CClientScript::POS_END</code>	Перед закрывающим тегом <code></body></code>

3. Скрипты ядра фреймворка – те скрипты, что поставляются вместе с Yii, например jQuery. Подключить скрипт ядра можно следующим образом:

```
Yii::app()->clientScript->registerCoreScript('jquery');
```

4. Все доступные пакеты можно увидеть в `framework/web/js/packages.php`:

```
return array(
    'jquery'=>array( 'js'=>array(YII_DEBUG ?
        'jquery.js' : 'jquery.min.js'), ),
    'yii'=>array( 'js'=>array('jquery.yii.js'),
        'depends'=>array('jquery'), ),
    'yiiatab'=>array( 'js'=>array('jquery.yiitab.js'),
        'depends'=>array('jquery'), ),
    'yiiactiveform'=>array( 'js'=>array(
        'jquery.yiiactiveform.js'),
        'depends'=>array(
            'jquery'), ),
    'jquery.ui'=>array( 'js'=>array(
        'jquery/js/jquery-ui.min.js'),
        'depends'=>array('jquery'), ),
    // ...
```

5. Ключи массива в списке выше, такие как `jquery`, `yii`, `yiitab`, `yiiactiveform`, являются именами, которые можно использовать в `registerClientScript`, а соответствующие им массивы – именами загружаемых скриптов относительно папки `framework/web/js/source`.
6. Встроенные скрипты – это те скрипты, код которых помещается прямо в тело страницы. Обычно это скрипты, код которых изменяется с каждым запросом. Подключить этот тип скриптов можно следующим образом:

```
Yii::app()->clientScript->registerScript('myscript',
    'echo "Hello, world!";', CClientScript::POS_READY);
```

Первый параметр является уникальным ID скрипта и выбирается на ваше усмотрение, второй – кодом скрипта, третий указывает на позицию внутри страницы, в которой должен находиться скрипт. Есть ещё две позиции, в которых может быть подключен скрипт методом `registerScript`:

<code>CClientScript::POS_LOAD</code>	Внутри функции <code>window.onload()</code>
<code>CClientScript::POS_READY</code>	Внутри функции <code>ready</code> из jQuery

Теперь давайте переключимся на CSS. Есть два типа CSS-стилей: встроенные и внешние (на данный момент нет стилей, включенных в ядро Yii).

1. Чтобы подключить внешний CSS, можно использовать следующий код:

```
Yii::app()->clientScript->registerCssFile(
    ('http://example.com/css/main.css');
```

Данный метод может принимать второй параметр, указывающий тип носителя (*media type*), для которого вы подключаете стиль (например, `screen` или `print`). Обратите внимание, что параметр, определяющий позицию CSS-кода, отсутствует, так как тег `<head>` (с учетом соблюдения стандартов) является единственным местом на странице, которое может содержать CSS. Следует избегать использования встроенного CSS, но если вы вынуждены его использовать, то сделать это можно так:

```
Yii::app()->clientScript->registerCss('myCSS',  
    'body {margin: 0; padding: 0}', 'all');
```

2. Первый параметр `registerCss` – уникальный ID, далее идут код CSS и тип носителя.

Как это работает

Методы класса `CClientScript`, рассмотренные выше, не подключают JavaScript и CSS немедленно, а всего лишь сохраняют ресурсы до тех пор, пока не будет вызван метод контроллера `render`. При отображении страницы скрипты и стили будут подключены в необходимых местах, причем случаи многократного подключения одного и того же ресурса не будут приводить к его дублированию на странице. Данная проблема решается автоматически, что позволяет эффективно и просто подключать ресурсы в виджетах, вложенных отображениях и в других случаях повторного использования кода.

Дополнительно

Мы рассмотрели самые распространенные типы ресурсов – JavaScript и CSS, но это ещё не все.

Использование собственных пакетов скриптов

В Yii вы можете управлять зависимостями между скриптами таким же образом, как это реализовано внутри ядра фреймворка. Данная возможность хорошо описана на странице документации

<http://www.yiiframework.com/doc/api/CClientScript#packages-detail>

Регистрация тегов link

`CClientScript` позволяет зарегистрировать собственный тег `<link ... />`. Например, данный способ окажется полезным, если вы захотите добавить RSS-ссылку на конкретный контроллер:

```
Yii::app()->clientScript->registerLinkTag(
    'alternate',
    'application/rss+xml',
    $this->createUrl('rss/articles')
);
```

Регистрация мета-тегов

В дополнение ко всему CClientScript позволяет регистрировать мета-теги, такие как `description` или `keywords`, используя метод `registerMetaTag`. Например, задать кодировку документа можно следующим образом:

```
Yii::app()->clientScript->registerMetaTag(
    'text/html; charset=utf-8', null, 'Content-Type');
```

Полезные материалы

За дополнительной информацией обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/CClientScript>;
- <http://www.yiiframework.com/doc/guide/1.1/ru/topics/performance>.

Смотрите также

- Рецепт «Управление ресурсами».
- Рецепт «Загрузка блока через AJAX».

Работа с JSON

Формат JSON очень прост и компактен, поэтому он широко распространен при обмене данными посредством AJAX. Yii предоставляет набор функций для удобной работы с JSON. Давайте создадим простое приложение со списком новостей, обновляющимся каждые две секунды.

Подготовка

1. Создайте новое приложение, используя `yii webapp`, как описано в официальном руководстве.
2. Создайте новую базу данных и настройте подключение к ней из вашего приложения.
3. Создайте таблицу `news`, содержащую поля `id`, `created_on` и `title`:


```
CREATE TABLE 'news' (  
    'id' int(11) unsigned NOT NULL AUTO_INCREMENT,  
    'created_on' int(11) unsigned NOT NULL,  
    'title' varchar(255) NOT NULL,  
    PRIMARY KEY ('id')  
)
```

4. Сгенерируйте модель News, используя Gii.

Как это делается

1. Создайте новый контроллер `protected/controllers/NewsController.php`:

```
<?php  
class NewsController extends Controller {  
    public function filters() {  
        return array('ajaxOnly + data',);  
    }  
  
    public function actionIndex() {  
        $this->render('index');  
    }  
  
    public function actionData() {  
        $criteria = new CDbCriteria();  
        $criteria->order = 'created_on DESC';  
        $criteria->limit = 10;  
        $news = News::model()->findAll($criteria);  
        echo CJSON::encode($news);  
    }  
  
    public function actionAddRandomNews() {  
        $news = new News();  
        $news->title = "Запись #" . rand(1, 10000);  
        $news->created_on = time();  
        $news->save();  
        echo "OK";  
    }  
}
```

2. Создайте представление `protected/views/news/index.php`:

```
<div class="news-list">Записка...</div>  
<?php Yii::app()->clientScript->registerCoreScript(  
    "jquery") ?>  
<script type="text/javascript">  
    jQuery(function($) {  
        var newsList = $('<div>.news-list');  
    }  
</script>
```

```
function updateNews() {
    newsList.html("Загрузка...");
    $.ajax({
        url: "<?php echo $this->createUrl('data') ?>",
        dataType: 'json',
        cache: false,
        success: function(data) {
            var out = "<ol>";
            $(data).each(function() {
                out+="- " + this.title + "</li>";
            });
            out += "</ol>";
            newsList.html(out);
        }
    });
}
updateNews();
setInterval(function() {
    updateNews()
}, 2000);
});
</script>

```

3. Запустите действие `index` контроллера `news` и добавьте несколько записей в таблицу `news`, выполнив действие `addRandomNews`. Не перезагружайте страницу, новые новости должны добавляться раз в две секунды, как показано ниже:



Как это работает

Действие `index` отображает страницу с контейнером `div` и небольшим JavaScript-кодом. Так как мы используем jQuery, необходимо убедиться, что он подключен:

```
<?php Yii::app()->clientScript->registerPackage("jquery") ?>
```

Далее мы определяем функцию `updateNews` и вызываем ее каждые 2000 миллисекунд с помощью JavaScript-функции `setInterval`. В `updateNews` мы отправляем AJAX-запросы к действию `data` того же контроллера, обрабатываем JSON-ответ и заменяем содержимое контейнера `div` отформатированными данными.

В `actionData` мы получаем последние новости и преобразуем их в формат JSON, передавая методу `CJSON::encode`.

Дополнительно

За дополнительной информацией обратитесь к следующим ресурсам:

- <http://api.jquery.com/category/ajax/>;
- <http://www.yiiframework.com/doc/api/CJSON/>;
- <http://www.yiiframework.com/doc/api/CClientScript/#registerCoreScript-detail>.

Смотрите также

- Рецепт «Загрузка блока через AJAX».

Передача параметров из PHP в JavaScript

Параметры приложения можно хранить в конфигурационном файле `protected/config/main.php`, в этом случае они будут доступны через массив `Yii::app()->params['paramName']`. Если приложение содержит код JavaScript, то вполне оправданно хранить используемые в нем параметры в том же конфигурационном файле. Давайте покажем, как можно этого достичь простым и эффективным способом.

Подготовка

1. Создайте новое приложение, используя `yiic webapp`. При этом должен быть сгенерирован массив параметров в файле `protected/config/main.php`:

```
'params'=>array(  
    // this is used in contact page  
    'adminEmail'=>'webmaster@example.com',  
)
```

2. Добавьте дополнительные параметры:

```
'params'=>array(  
    // this is used in contact page  
    'adminEmail'=>'webmaster@example.com',  
    'alert' => array(  
        'enabled' => true, 'message' => 'Привет!', ),  
),
```

Как это делается

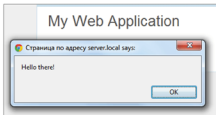
1. Создайте контроллер `protected/controllers/AlertController.php`:

```
<?php  
class AlertController extends Controller {  
    function actionIndex() {  
        $config = CJavaScript::encode(  
            Yii::app()->params->toArray());  
        Yii::app()->clientScript->registerScript(  
            'appConfig', "var config = ".$config.";",  
            CClientScript::POS_HEAD);  
        $this->render('index');  
    }  
}
```

2. Затем создайте представление `protected/views/alert/index.php`:

```
<script>  
if(config && config.alert && config.alert.enabled &&  
    config.alert.message){  
    alert(config.alert.message);  
}  
</script>
```

3. Теперь, если запустить действие `index` контроллера `alert`, мы увидим стандартное JavaScript-сообщение с надписью «Привет!», как показано на скриншоте:



Как это работает

Сначала мы преобразовываем структуры данных PHP в структуры данных JavaScript с помощью `CJavaScript::encode`. Затем мы регистрируем скрипт, который присваивает полученные данные глобальной переменной `config`. После этого в представлении `index` мы просто используем глобальную переменную `config`.

Смотрите также

- Рецепт «Управление ресурсами»;
- Рецепт «Загрузка блока через AJAX».

Обработка переменного числа полей в форме

Иногда приложение требует наличия форм с переменным числом полей ввода. Например, приложение «Менеджер задач» позволяет добавить один и более пунктов в список. Пример такого приложения показан ниже:

My Web Application

Home About Contact Login

- Title
Do some Yii stuff
Text
- Title
Buy milk
Text
Need to go to the shop and buy some milk.
- Title
Feed the cat
Text
Else he'll be very angry :)

Add task Save

По умолчанию приложение будет отображать поля ввода для создания одной задачи и две кнопки: «Добавить задачу», которая создает на странице пустые поля ввода для очередного задания, и «Сохранить», по нажатию на которую приложение сохранит все данные и снова отобразит заполненную форму.

Подготовка

- Создайте новое приложение, используя `yiiic webapp`.

Как это делается

В нашем примере мы не будем сохранять что-либо в базе данных, а просто покажем, как создать форму с переменным числом полей и как обрабатывать такие формы на сервере.

1. Начнем с модели `Task`, так как мы не собираемся использовать базу данных, то `CFormModel` будет вполне достаточно. Итак, `protected/models/Task.php`:

```
<?php
class Task extends CFormModel {
    public $title;
    public $text;

    public function rules() {
        return array(
            array('title', 'required'),
            array('text', 'safe'),
        );
    }
}
```

2. Теперь создадим контроллер `protected/controllers/TaskController.php`:

```
<?php
class TaskController extends Controller {
    public function filters() {
        return array('ajaxOnly + field');
    }

    public function actionIndex() {
        $models = array();
        if (!empty($_POST['Task'])) {
            foreach ($_POST['Task'] as $taskData) {
                $model = new Task();
                $model->setAttributes($taskData);
            }
        }
    }
}
```

```

        if ($model->validate())
            $models[] = $model;
    }
} if (!empty($models)) {
    // Мы получили несколько валидных моделей
    // Если их надо сохранять, то это стоит
    // делать здесь
} else
    $models[] = new Task();
$this->render('index', array(
    'models' => $models,));
}

public function actionField($index) {
    $model = new Task();
    $this->renderPartial('_task', array(
        'model' => $model,
        'index' => $index,));
}
}

```

3. Затем представление `protected/views/task/index.php`:

```

<div class="form">
    <?php echo CHtml::beginForm()?>
    <ul class="tasks">
        <?php for($i=0; $i<count($models); $i++)?:>
            <?php $this->renderPartial('_task', array(
                'model' => $models[$i],
                'index' => $i, ))?>
        <?php endforeach ?>
    </ul>
    <div class="row buttons">
        <?php echo CHtml::button('Добавить задачу',
            array('class' => 'tasks-add'))?>
        <?php Yii::app()->clientScript->registerCoreScript
            ("jquery")?>
        <script>
            $(".tasks-add").click(function(){
                $.ajax({
                    success: function(html){
                        $(".tasks").append(html);
                    },
                    type: 'get',
                    url: '<?php echo $this->createUrl
                        ('field')?>',
                    data: { index: $(".tasks li").size() },
                    cache: false,
                    dataType: 'html'
                })
            })
        </script>
    </div>

```

```
        ));  
    });  
</script>  
<?php echo CHtml::submitButton('Сохранить')?>  
</div>  
<?php echo CHtml::endForm()?>  
</div>
```

4. И наконец, вложенное представление `protected/views/task/_task.php`:

```
<li>  
    <div class="row">  
        <?php echo CHtml::activeLabel($model,  
            "[$index]title")?>  
        <?php echo CHtml::activeTextField($model,  
            "[$index]title")?>  
    </div>  
    <div class="row">  
        <?php echo CHtml::activeLabel($model,  
            "[$index]text")?>  
        <?php echo CHtml::activeTextArea($model,  
            "[$index]text")?>  
    </div>  
</li>
```

5. Готово. Теперь запустите действие `index` контроллера `task` и убедитесь, что форма выглядит так, как показано на рисунке.

My Web Application

Home About Contact Login

- Title
test1
Text
test1
- Title
test2
Text
test2
- Title

Text

Add task Save

Как это работает

Давайте рассмотрим, каким образом все это работает. Начнем с действия `index`. Так как в форме мы работаем более чем с одной сущностью, то и обрабатывать данные следует соответствующим образом. Как и в случае одной модели, данные будут доступны через `$_POST['model_name']`, за одним лишь исключением – переменная будет содержать массив моделей, как показано ниже:

```
<?php
if (!empty($_POST['Task'])) {
    foreach ($_POST['Task'] as $taskData) {
        $model = new Task();
        $model->setAttributes($taskData);
        if ($model->validate())
            $models[] = $model;
    }
}
```

Для каждого элемента массива мы создаем модель `Task`, задаем ее атрибуты и, если модель валидна, добавляем ее в массив `$models`:

```
if(!empty($models)){
    // Мы получили несколько валидных моделей
    // Если их надо сохранять, то это стоит делать здесь
} else
    $models[] = new Task();
```

Если массив `$models` не пуст, значит, он содержит валидные данные, переданные из формы. Если же массив не содержит данных, то мы добавляем в него экземпляр модели `Task`, чтобы отобразить пустую форму с полями для создания одной задачи.

В нашем примере данные не сохраняются. Если бы мы использовали `Active Record`, то могли бы сохранить их, используя `$model->save()`.

Затем мы просто отображаем массив моделей в представлении `index`:

```
<?php for($i=0; $i<count($models); $i++):?>
    <?php $this->renderPartial('_task', array(
        'model' => $models[$i], 'index' => $i, ))?>
<?php endfor ?>
```

Так как у нас есть набор моделей, то мы показываем каждую из них по отдельности, используя частичное представление `_task`:

```
<li>
    <div class="row">
```

```
<?php echo CHtml::activeLabel($model,
    "{$index}title")?>
<?php echo CHtml::activeTextField($model,
    "{$index}title")?>
</div>
<div class="row">
    <?php echo CHtml::activeLabel($model,
        "{$index}text")?>
    <?php echo CHtml::activeTextArea($model,
        "{$index}text")?>
</div>
</li>
```

Обратите внимание на то, как используются методы CHtml::activeLabel, CHtml::activeTextField и CHtml::activeTextArea: для каждой модели мы определяем имя в формате {номер модели}field_name. В результате будет сгенерирован следующий HTML-код:

```
<li>
    <div class="row">
        <label for="Task_0_title">Title</label>
        <input name="Task[0][title]" id="Task_0_title"
            type="text" value="" />
    </div>
    <div class="row">
        <label for="Task_0_text">Text</label>
        <textarea name="Task[0][text]" id="Task_0_text">
        </textarea>
    </div>
</li>
<li>
    <div class="row">
        <label for="Task_1_title">Title</label>
        <input name="Task[1][title]" id="Task_1_title"
            type="text" value="" />
    </div>
    <div class="row">
        <label for="Task_1_text">Text</label>
        <textarea name="Task[1][text]" id="Task_1_text">
        </textarea>
    </div>
</li>
```

Поля с именами вида Task[0][title] при отправке на сервер автоматически преобразуются в массивы PHP – это стандартная возможность PHP.

Теперь рассмотрим, как работает кнопка «Добавить задачу»:

```
<?php echo CHtml::button('Добавить задачу', array('
    class'=>'tasks-add'))?>
<?php Yii::app()->clientScript->registerCoreScript(
    "jquery")?>
<script>
    $(".tasks-add").click(function(){
        $.ajax({
            success: function(html){
                $(".tasks").append(html);
            },
            type: 'get',
            url: '<?php echo $this->createUrl('field')?>',
            data: { index: $(".tasks li").size() },
            cache: false,
            dataType: 'html'
        });
    });
</script>
```

Мы добавляем кнопку с классом `tasks-add` и jQuery-скрипт, задающий обработчик события `onClick` для этой кнопки. По нажатии на нее отправляется AJAX-запрос к действию `field`, запрос содержит параметр `index` – номер очередной сущности. В ответ мы получаем HTML-код, который добавляем к уже существующему коду формы.

И ещё

Дополнительную информацию об обработке форм с переменным числом полей можно найти по следующему адресу: <http://www.yiiframework.com/doc/guide/ru/form.table>.

Смотрите также

- Рецепт «Загрузка блока через AJAX».



ГЛАВА 4.

Работа с формами

В этой главе мы рассмотрим:

- Пишем свой валидатор.
- Загрузка файлов.
- Добавление CAPTCHA.
- Настройка CAPTCHA.
- Создаем виджет для ввода при помощи CInputWidget.

Введение

Работать с формами в Yii очень просто, да и документация в этом отношении практически полная. Тем не менее некоторые моменты не будет лишним пояснить на примерах, что мы и сделаем в этой главе.

Пишем свой валидатор

Yii предоставляет неплохой набор встроенных валидаторов форм, которые охватывают большинство типичных для разработчика задач. Но в некоторых случаях можно столкнуться с необходимостью создания собственного валидатора.

Хорошим примером может служить подтверждение прав собственности на сайт. Для некоторых сервисов Google требует загрузить на сайт файл с определенным названием и содержимыми, а затем проверяет его наличие. Попробуем реализовать то же самое.

Это можно сделать двумя способами – использовать метод класса в качестве валидатора или написать отдельный класс.

Подготовка

Создайте новое приложение при помощи `yiic webapp`, как описано в официальной документации.

Как это делается

1. Рассмотрим способ с методом класса. Сначала нужно реализовать модель формы. Создадим `protected/models/SiteConfirmation.php`:

```
class SiteConfirmation extends CFormModel
{
    public $url;

    public function rules()
    {
        return array(
            array('url', 'confirm'),
        );
    }

    public function confirm($attribute, $params)
    {
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $this->url);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        $output = curl_exec($ch);
        curl_close($ch);
        if (trim($output) != 'code here')
            $this->addError('url', 'Сначала нужно
                                загрузить файл.');
```

2. Готово. Теперь используем нашу модель в нашем тестовом контроллере. Создадим `protected/controllers/TestController.php`:

```
class TestController extends CController
{
    function actionIndex()
    {
        $confirmation = new SiteConfirmation();
        $confirmation->url = 'http://rmcreative.ru/
                                verify.html';
        if ($confirmation->validate())
            echo 'Успешно';
        else
            echo 'Пожалуйста, загрузите файл.';
    }
}
```

3. Далее попробуем запустить тестовый контроллер. Получили «Успешно», ведь файл с текстом «code here» доступен по ад-

ресу <http://rmcreative.ru/verify.html>. Если заменить адрес для подтверждения на любой другой, мы получим сообщение об ошибке «Сначала нужно загрузить файл».

Как это работает

В модели SiteConfirmation мы определили поле `$url` и добавили метод `rules`, который определяет одно правило валидации для этой формы. Так как нет встроенного валидатора с именем `confirm`, Yii считает, что мы хотим описать правила валидации в методе с названием `confirm`. В этом методе, используя стандартную библиотеку `CURL`, мы получаем содержимое файла `verify.html` с другого сервера и сравниваем его с тестовой строкой «code here». Если содержимое отличается, добавляется ошибка валидации при помощи метода `addError`.

Если потребуется, можно использовать два аргумента в методе валидации — `$attribute` и `$params`. Например, если задать правило валидации так:

```
array('url', 'confirm', 'param1' => 'value1', 'param2' => 'value3'),
```

значение `$attribute` будет `'url'`, а в `$params` будет массив:

```
array
(
    'param1' => 'value1',
    'param2' => 'value3'
)
```

И ещё

Так как мы, вероятно, захотим повторно использовать валидатор такого типа, вынесем его из метода модели в отдельный класс. Создадим файл `protected/components/RemoteFileValidator.php`:

```
<?php
class RemoteFileValidator extends CValidator
{
    public $content = '';

    protected function validateAttribute($object, $attribute)
    {
        $value = $object->$attribute;
        $ch = curl_init();
        curl_setopt($ch, CURLOPT_URL, $value);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        $output = curl_exec($ch);
```

```
curl_close($ch);  
if (trim($output) != $this->content)  
    $this->addError($object, $attribute,  
        'Please upload file first.');
```

Класс нашего валидатора должен расширять `CValidator` и реализовывать его абстрактный метод `validateAttribute`, в который передаются два аргумента: `$object` – экземпляр проверяемой модели и `$attribute` – имя проверяемого свойства. Передаваемые параметры назначаются соответствующим открытым свойствам класса валидатора.

Вот и всё. Теперь попробуем использовать его. В модели `SiteConfirmation` следует изменить правило валидации на:

```
array('url', 'RemoteFileValidator', 'content' => 'code here')
```

Здесь мы использовали имя внешнего валидатора. Если в модели нет метода с таким именем и нет одноименного встроенного валидатора, Yii попытается найти внешний файл валидатора с указанным именем или псевдонимом пути.

Остальную часть кода оставим без изменений. Теперь валидатор можно использовать в других моделях.

Полезные материалы

- <http://www.yiiframework.com/doc/api/CValidator/>;
- <http://www.yiiframework.com/doc/api/CModel#rules-detail>.

Загрузка файлов

Загрузка файлов на сервер является довольно распространённой для веб-приложений задачей. В Yii для этого есть встроенные классы. Создадим простую форму, которая позволит загружать ZIP-архивы и сохранять их в `protected/uploads`.

Подготовка

1. Создайте новое приложение, используя `yiic webapp`.
2. В каталоге `protected` создайте подкаталог `uploads`.

Как это делается

1. Начнем с модели. Создадим `protected/models/Upload.php`:

```

class Upload extends CFormModel
{
    public $file;

    public function rules()
    {
        return array(
            array('file', 'file', 'types' => 'zip'),
        );
    }
}

```

2. Теперь перейдем к контроллеру `protected/controllers/UploadController.php`:

```

class UploadController extends Controller
{
    function actionIndex()
    {
        $dir = Yii::getPathOfAlias('application.
                                uploads');

        $uploaded = false;
        $model = new Upload();
        if (isset($_POST['Upload'])) {
            $model->attributes = $_POST['Upload'];
            $file = CUploadedFile::getInstance($model,
                'file');
            if ($model->validate()) {
                $uploaded = $file->saveAs($dir .
                    '/' . $file->getName());
            }
        }
        $this->render('index', array(
            'model' => $model,
            'uploaded' =>
                $uploaded,
            'dir' => $dir,
        ));
    }
}

```

3. И наконец, представление `protected/views/upload/index.php`:

```

<?php if ($uploaded): ?>
<p>Файл успешно загружен. Проверьте <?php echo $dir?>.</p>
<?php endif ?>
<?php echo CHtml::beginForm('', 'post', array('enctype'
=>'multipart/form-data'))?>
<?php echo CHtml::error($model, 'file') ?>

```



```
<?php echo CHtml::activeFileField($model, 'file') ?>
<?php echo CHtml::submitButton('Upload') ?>
<?php echo CHtml::endForm() ?>
```

4. Это всё. Теперь запустите созданный контроллер и попробуйте загрузить ZIP и другие файлы, как показано на скриншоте:



Как это работает

Это довольно простая модель. Мы определяем только одно поле `$file` и правило валидации, использующее валидатор файлов (`CFileValidator`), которое читается как «разрешены только zip-файлы».

Контроллер немного сложнее; разберем его построчно.

```
$dir = Yii::getPathOfAlias('application.uploads');
$uploaded = false;
$model = new Upload();
if (isset($_POST['Upload'])) {
    $model->attributes = $_POST['Upload'];
```

`$dir` – это каталог, который будет хранить загруженные файлы. Мы задаем его равным `protected/uploads`, используя псевдоним пути. `$uploaded` – это маркер, определяющий, нужно ли показывать сообщение в случае успешной загрузки. Затем мы создаем экземпляр модели и, если форма отправлена, заполняем его данными из массива `$_POST`.

```
$file = CUploadedFile::getInstance($model, 'file');
if ($model->validate()) {
    $uploaded = $file->saveAs($dir . '/' . $file->getName());
}
```

Далее мы применяем `CUploadedFile::getInstance`, который возвращает экземпляр класса `CUploadedFile`. Этот класс – не что иное, как обертка для PHP-массива `$_FILE`, который заполняется при загрузке файлов.

Если мы убедились, что файл является zip-архивом, используя метод `validate` нашей модели, то сохраняем его при помощи `CUploadedFile::saveAs`.

И в завершение передадим нужную информацию в представление:

```
<?php if ($uploaded): ?>
<p>Файл успешно загружен. Проверьте <?php echo $dir?>.</p>
<?php endif ?>
```

Если маркер `$uploaded` принимает истинное значение, покажем сообщение об успешной загрузке.

Для загрузки файла форма должна удовлетворять двум обязательным требованиям:

1. Должен использоваться метод POST.
2. Атрибут `enctype` должен иметь значение `'multipart/form-data'`.

Мы можем создать нужный код, используя хелпер `CHtml` или `CActiveForm` с опцией `htmlOptions`.

```
<?php echo CHtml::beginForm('', 'post', array('enctype' =>
'multipart/form-data'))?>
```

Оставшееся – стандартная форма: мы выводим ошибку, поле для свойства `$file` нашей модели и кнопку отправки формы.

Дополнительно

Если требуется одновременно загружать несколько файлов, код следует изменить таким образом:

```
if (isset($_POST['Upload'])) {
    $model->attributes = $_POST['Upload'];
    $files = CUploadedFile::getInstance($model, 'file');
    if ($model->validate()) {
        foreach ($files as $file)
            $file->saveAs($dir . '/' . $file->getName());
    }
}
```

В представлении выводим поля для загрузки файлов:

```
<?php echo CHtml::activeFileField($model, "[0]file")?>
<?php echo CHtml::activeFileField($model, "[1]file")?>
<?php echo CHtml::activeFileField($model, "[2]file")?>
```

Валидатор файлов

Валидатор файлов, использованный в модели, позволяет не просто разрешить загрузку файлов только определенного типа, но и установить другие ограничения, как размер файла или количество файлов в случае одновременной загрузки. К примеру, следующее правило

разрешает только загрузку изображений размером не больше одного мегабайта:

```
array('file', 'file', 'types'=>'jpg, gif, png', 'maxSize'=> 1048576),
```

Полезные материалы

- <http://www.yiiframework.com/doc/api/CFileValidator>;
- <http://www.yiiframework.com/doc/api/CUploadedFile>.

Смотрите также

- Рецепт «Обработка переменного числа полей в форме» в главе 4.

Добавление CAPTCHA

В Интернете, если оставить форму без защиты от спама, вы получите кучу писем в короткий промежуток времени. В Yii присутствует компонент CAPTCHA, который делает добавление такой защиты очень легкой задачей. Единственная проблема – в отсутствии пошагового руководства по его использованию. В следующем примере мы добавим защиту CAPTCHA к простой форме.

Подготовка

1. Создайте новое приложение, используя инструмент `yiic webapp`.
2. Создайте модель формы `protected/models/EmailForm.php`:

```
class EmailForm extends CFormModel
{
    public $email;

    function rules()
    {
        return array(
            array('email', 'email'),
        );
    }
}
```

3. Создайте контроллер `protected/controllers/EmailController.php`:

```
class EmailController extends Controller
{
    public function actionIndex()
```

```
{
    $success = false;
    $model = new EmailForm();
    if (!empty($_POST['EmailForm'])) {
        $model->setAttributes($_POST['EmailForm']);
        if ($model->validate()) {
            $success = true;
            // handle form here
        }
    }
    $this->render('index', array(
        'model' => $model,
        'success' =>
            $success,
    ));
}
```

4. Создайте представление `protected/views/email/index.php`:

```
<?php if ($success): ?>
<p>Успешно!</p>
<?php endif ?>
<?php echo CHtml::beginForm() ?>
<p>
    <?php echo CHtml::activeLabel($model, 'email')?>
    <?php echo CHtml::activeTextField($model, 'email')?>
    <?php echo CHtml::error($model, 'email')?>
</p>
<p>
    <?php echo CHtml::submitButton()?>
</p>
<?php echo CHtml::endForm() ?>
```

5. Мы получили форму, которая проверяет, является ли поле email корректным адресом. Добавим защиту от спама.

My Web Application

Home About Contact Login

Email

Email is not a valid email address.

Как это делается

- Для начала настроим форму – необходимо добавить свойство `$verifyCode`, в котором будут сохраняться введенный код и правило валидации для него.

```
class EmailForm extends CFormModel
{
    public $verifyCode;
    public $email;

    function rules()
    {
        return array(
            array('email', 'email'),
            array('verifyCode', 'captcha',
                'allowEmpty' => !CCaptcha::checkRequirements()),
        );
    }
}
```

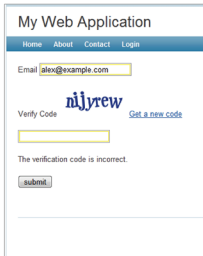
- Теперь следует подключить внешнее действие к контроллеру. Добавим в него следующий код:

```
public function actions()
{
    return array(
        'captcha' => array(
            'class' => 'CCaptchaAction',
        ),
    );
}
```

- В представлении нам нужно вывести дополнительное поле и изображение CAPTCHA. Это делает следующий код:

```
<?php if (CCaptcha::checkRequirements() && Yii::
app()->user->isGuest): ?>
<p>
    <?php echo CHtml::activeLabelEx($model,
    'verifyCode')?>
    <?php $this->widget('CCaptcha')?>
</p>
<p>
    <?php echo CHtml::activeTextField($model,
    'verifyCode')?>
    <?php echo CHtml::error($model, 'verifyCode')?>
</p>
<?php endif ?>
```

- Теперь вы можете запустить контроллер email и проверить CAPTCHA в действии, как показано на картинке ниже:



Если поле CAPTCHA не появилось в форме и не выводится никаких ошибок, скорее всего, не настроено или не установлено PHP расширение GD. Оно необходимо для работы CAPTCHA, так как используется для генерации изображений. В нескольких местах мы добавили проверку `CCaptcha::checkRequirements()`, так что приложение не будет использовать CAPTCHA, если не сможет вывести изображение, но при этом останется рабочим.

Как это работает

В представлении мы вызываем виджет `CCaptcha`, который выводит тег `<img>`, у которого атрибут `src` ссылается на действие `CCaptchaAction`, добавленное в контроллер. В этом действии генерируется изображение со случайным набором символов, который пользователь должен ввести в соответствующее поле формы. Код сохраняется в сессии пользователя, а изображение выводится на экран.

Когда пользователь вводит адрес электронной почты и код подтверждения в форме, мы присваиваем их полям модели формы, а затем проверяем. Для поля мы используем валидатор `CCaptchaVali-`

dator, который получает код из сессии и сравнивает его значение с введенным пользователем. Если они не совпадают, данные модели считаются некорректными.

Дополнительно

Если вы ограничиваете доступ к контроллеру, используя `accessRules`, не забудьте разрешить доступ к изображению для всех пользователей:

```
public function accessRules()
{
    return array(
        // ...
        array('allow',
            'actions' => array('captcha'),
            'users' => array('*'),
        ),
        array('deny',
            'users' => array('*'),
        ),
    );
}
```

Полезные материалы

- <http://www.yiiframework.com/doc/api/CCaptcha/>;
- <http://www.yiiframework.com/doc/api/CCaptchaAction/>;
- <http://www.yiiframework.com/doc/api/CCaptchaValidator/>.

Смотрите также

- Рецепт «Подключение внешних действий» в главе 2.
- Рецепт «Настройка CAPTCHA» в этой главе.

Настройка CAPTCHA

CAPTCHA в Yii хорошо подходит для защиты от спама, однако не исключены ситуации, когда потребуется ее изменить:

- вам попался спам-бот, который легко распознает текст с картинки и требуется усложнить ему задачу; есть желание упростить или разнообразить ввод текста.

В нашем примере мы изменим стандартную защиту от спама так, что вместо ввода текста с картинки от пользователя потребуется решить несложную арифметическую задачу.

Подготовка

За отправную точку возьмем результат предыдущего рецепта. Также можно использовать любую форму, использующую стандартную защиту от спама, поскольку мы не будем вносить больших изменений в уже существующий код.

Как это делается

Нам требуется изменить действие `CCaptchaAction`, которое создает код и выводит соответствующее изображение. Картинка должна содержать арифметическое выражение, значение которого равняется случайному числу.

1. Создадим `protected/components/MathCaptchaAction.php`:

```
class MathCaptchaAction extends CCaptchaAction
{
    protected function generateVerifyCode()
    {
        return mt_rand((int)$this->minLength,
            (int)$this->maxLength);
    }

    public function renderImage($code)
    {
        parent::renderImage($this->getText($code));
    }

    protected function getText($code)
    {
        $code = (int)$code;
        $rand = mt_rand(1, $code - 1);
        $op = mt_rand(0, 1);
        if ($op)
            return $code - $rand . "+" . $rand;
        else
            return $code + $rand . "-" . $rand;
    }
}
```

2. Теперь в методе `actions` контроллера заменим `CCaptchaAction` на только что созданное действие.

```
public function actions()
{
    return array(
        'captcha' => array(
```



```
'class' => 'MathCaptchaAction',  
'minLength' => 1,  
'maxLength' => 10,  
    ),  
);  
}
```

3. Далее откроем страницу с формой и опробуем новую защиту. Появится арифметическое выражение с числами от 1 до 10, результат вычисления которого нужно ввести для проверки.

My Web Application

Home About Contact Login

Email

9+1

Verify Code [Get a new code](#)

The verification code is incorrect.

Как это работает

Мы переопределили два метода в `CCaptchaAction`. В методе `generateVerifyCode` мы создаем вместо текста случайное число. Поскольку вместо отображения текста требуется вывести выражение, мы переопределили `renderImage`. Само выражение создается добавленным методом `getText`.

Полезные материалы

- <http://www.yiiframework.com/doc/api/CCaptcha/>;
- <http://www.yiiframework.com/doc/api/CCaptchaAction/>;
- <http://www.yiiframework.com/doc/api/CCaptchaValidator/>.


```
    }
}
}
```

2. Теперь проверим, как он работает. Нам потребуется модель формы `protected/models/RangeForm.php`:

```
class RangeForm extends CFormModel
{
    public $from;
    public $to;

    function rules()
    {
        return array(
            array('from, to', 'numerical', 'integerOnly'
                => true),
            array('from', 'compare', 'compareAttribute'
                => 'to', 'operator' => '<=',
                'skipOnError' => true),
        );
    }
}
```

3. Теперь создадим контроллер `protected/controllers/RangeController.php`:

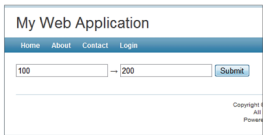
```
class RangeController extends Controller
{
    function actionIndex()
    {
        $success = false;
        $model = new RangeForm();
        if (!empty($_POST['RangeForm'])) {
            $model->setAttributes($_POST['RangeForm']);
            if ($model->validate())
                $success = true;
        }
        $this->render('index', array(
            'model' => $model,
            'success' =>
                $success,
        ));
    }
}
```

4. И представление `protected/views/range/index.php`:

```
<?php if ($success): ?>
<p>Успешно!</p>
```

```
<?php endif ?>
<?php echo CHtml::errorSummary($model) ?>
<?php echo CHtml::beginForm() ?>
<?php $this->widget('RangeInputField', array(
    'model' => $model,
    'attributeFrom' => 'from',
    'attributeTo' => 'to',
)) ?>
<?php echo CHtml::submitButton('Submit') ?>
<?php echo CHtml::endForm() ?>
```

5. Далее запустим контроллер `range`, чтобы посмотреть виджет в работе



Как это работает

Обычный виджет ввода можно использовать как совместно с моделью в качестве `active field widget`, так и без нее. `Active field widget` автоматически получает значение и результат проверки.

Поскольку в нашем виджете два поля, мы создадим три пары `public-свойств`: `attribute`, `name` и `value`. Пара `attribute` используется, если в виджет передана модель, что означает его применение в качестве активного элемента ввода. Пары `name` и `value` будут использоваться, если требуется создать элементы ввода с заданными названиями и значениями.

В нашем случае мы просто переопределили метод `run` для отображения двух полей. Фактическая обработка значений полей делегирована методам `CHtml::activeTextField` и `CHtml::textField`. Для вывода виджета мы используем метод `CController::widget`:

```
<?php $this->widget('RangeInputField', array(
    'model' => $model,
    'attributeFrom' => 'from',
```

```
'attributeTo' => 'to',  
))?>
```

Все параметры, заданные в массиве, присваиваются соответствующим public-свойствам виджета.

Дополнительно

Для более подробного изучения виджетов можно ознакомиться со следующими ресурсами:

- <http://www.yiiframework.com/doc/api/CInputWidget/>;
- <http://www.yiiframework.com/doc/api/CWidget/>.

Смотрите также

- Рецепт «Настройка компонентов» в главе 1.
- Рецепт «Настройка виджетов по умолчанию» в главе 1.



ГЛАВА 5.

Тестирование приложений

В этой главе мы рассмотрим следующие темы:

- Настройка тестового окружения.
- Написание и запуск юнит-тестов.
- Фикстуры.
- Функциональное тестирование.
- Генерация отчетов о покрытии кода.

Введение

Необходимость тестирования в малых проектах может казаться спорной, но в больших проектах всё обстоит совершенно иначе: чем больше приложение, тем сложнее модифицировать его код, ничего при этом не сломав. Процесс разработки значительно замедляется, даже если команда тестировщиков весьма профессиональна. В таких ситуациях автоматизированное тестирование частично решает проблему.

Ещё одно применение автоматизированного тестирования – **TDD (Test Driven Development** или **Разработка через тестирование**). Идея проста: если известно, как компонент приложения должен работать, то следует записать эти требования в форме автоматизированных тестов, перед тем как начать его реализацию. Таким образом, написав тесты один раз, всегда можно убедиться, работает ли компонент корректно, запустив их.

Настройка тестового окружения

Этот рецепт посвящен настройке тестового окружения для запуска функциональных и юнит-тестов. Юнит-тестирование в Yii осуществляется с помощью **PHPUnit**, а функциональное – с помощью **Selenium Server**. В дополнение ко всему нам понадобится **Xdebug** для генерации отчетов о покрытии кода тестами.

Подготовка

- Убедитесь, что PHP правильно настроен для работы из командной строки.
- Создайте новое приложение, используя `yiiic webapp`.

Как это делается

1. Начнем с RHPUnit, для которого прежде всего нужен PEAR. Если PEAR уже установлен, как это бывает в большинстве Linux-сред, просто пропустите этот пункт.
2. Определить, установлен ли PEAR, можно набрав в консоли `pear`. Вы должны увидеть следующий вывод:

```

B:\>pear
Commands:
build          Build an Extension From C Source
bundle         Unpacks a PEAR Package
channel-add    Add a Channel
channel-alias  Specify an alias to a channel name
channel-delete Remove a Channel From the List
channel-discover Initialize a Channel from its server
channel-info   Retrieve Information on a Channel
  
```

3. Если вы его увидели, значит, PEAR установлен и можно продолжать, в противном случае надо установить PEAR, выполнив следующие пункты:
 - открыть <http://pear.php.net/go-pear> и сохранить содержимое как PHP-файл `go-pear.php`. Затем запустить этот файл из консоли с помощью команды `php go-pear.php` и следовать инструкциям;
 - в ОС Windows можно добавить путь к PEAR в переменную окружения `PATH`, чтобы использовать короткую команду `pear` в консоли.
4. Теперь приступим к установке RHPUnit. Откройте консоль и наберите следующее:

```

pear channel-discover pear.phpunit.de
pear channel-discover components.ez.no
pear channel-discover pear.symfony-project.com
pear install phpunit/PHPUnit
  
```

5. Набрав в командной строке `phpunit`, вы должны увидеть следующий вывод:

```

B:\>phpunit
PHPUnit 3.5.11 by Sebastian Bergmann.

Usage: phpunit [switches] UnitTest [UnitTest.php]
       phpunit [switches] <directory>

--log-junit <file>          log test execution in JUnit XML format to file.
--log-tap <file>           log test execution in TAP format to file.
--log-stdout                log test execution to STDOUT.
--log-junit <file>         log test execution in JUnit XML format to file.
  
```

6. Готово. Теперь приступим к установке Selenium Server. Для него нет готового пакета в PEAR, поэтому необходимо перейти по адресу <http://seleniumhq.org/download/> и скачать последнюю версию в виде архива наподобие `selenium-server-standalone-2.25.0.jar`.
7. Для запуска сервера требуется JRE (Java Runtime Enviroment). Перейдите в директорию с сервером и выполните команду:

```
java -jar selenium-server-standalone-2.25.0.jar
```

Вы должны увидеть что-то наподобие этого:

```
C:\usr\local\selenium>java -jar selenium-server-standalone-2.0rc2.jar
02:17:56.758 INFO - Java: Sun Microsystems Inc. 19.1-b02
02:17:56.760 INFO - OS: Windows 7 6.1 x86
02:17:56.773 INFO - v2.0 [rc2], with Core v2.0 [rc2]
02:17:57.094 INFO - RemoteWebDriver instances should connect to: http://127.0.0.1:4444/wd/hub
02:17:57.096 INFO - Version Jetty/5.1.x
02:17:57.098 INFO - Started HttpContext[/selenium-server/driver,/selenium-server/driver]
02:17:57.100 INFO - Started HttpContext[/selenium-server,/selenium-server]
02:17:57.100 INFO - Started HttpContext[/,/]
02:17:57.180 INFO - Started org.openqa.jetty.jetty.servlet.ServletHandler@112f04
02:17:57.181 INFO - Started HttpContext[/wd,/wd]
02:17:57.193 INFO - Started SocketListener on 0.0.0.0:4444
02:17:57.204 INFO - Started org.openqa.jetty.jetty.Server@b37c2e
```

8. Сервер запущен. Теперь приступим к Xdebug. Перейдите на страницу <http://www.xdebug.org/download.php> и скачайте последние бинарные файлы или исходные коды. Для Linux расширение должно быть скомпилировано из исходных кодов, а для Windows необходимо просто скачать dll-файл и подключить его в `php.ini` таким образом:

```
[xdebug]
zend_extension=c:/path/to/your/php_xdebug_version.dll
```

Если вы устанавливаете Xdebug на Linux, то придется указать абсолютный путь к `php_xdebug_version.so`, скомпилированному в соответствии с <http://www.xdebug.org/docs/install>.

Можно воспользоваться <http://www.xdebug.org/find-binary.php>, чтобы узнать, установлен ли Xdebug.

Теперь у вас должны быть установлены и запущены все необходимые инструменты.

9. А сейчас давайте посмотрим на приложение, сгенерированное `yiiic webapp`. Все, что относится к тестированию, расположено в папке `protected/tests`:

```
fixtures
functional
```



```
report
unit
bootstrap.php
WebTestCase.php
phpunit.xml
```

Сгенерированные папки предназначены для хранения различных тестов, фикстур и отчетов о покрытии кода.

10. Файл `bootstrap.php` нужен для инициализации окружения, в котором запускаются тесты:

```
// измените пути, если потребуется
$yiit=dirname(__FILE__).'../../../framework/yiit.php';
$config=dirname(__FILE__).'../config/test.php';
require_once($yiit);
require_once(dirname(__FILE__).'../WebTestCase.php');
Yii::createWebApplication($config);
```

Мы видим, что это окружение использует отдельный файл с настройками `protected/config/test.php`, это значит, что в случае использования базы данных или кеша необходимо настроить их в этом файле.

11. `WebTestCase.php` – базовый класс для функциональных тестов. Мы должны отредактировать его и установить в `TEST_BASE_URL`-значение адреса нашего сайта. И наконец, `phpunit.xml` – обычный файл настроек для PHPUnit. Нет необходимости изменять этот файл до тех пор, пока вы не захотите добавить новые браузеры для функционального тестирования или изменить глобальные настройки PHPUnit.

И ещё

Более детальные инструкции по установке необходимого ПО можно найти по следующим ссылкам:

- <http://pear.php.net/manual/en/installation.php>;
- <http://phpunit.de/>;
- http://seleniumhq.org/docs/05_selenium_rc.html.

Смотрите также

- Рецепт «Написание и запуск юнит-тестов».
- Рецепт «Фикстуры».
- Рецепт «Функциональное тестирование».
- Рецепт «Генерация отчетов о покрытии кода».

Написание и запуск юнит-тестов

Юнит-тестирование в большинстве случаев используется в отношении к относительно самостоятельным компонентам приложения. Например, можно тестировать API различных классов или API оболочки каких-либо сервисов.

В этом рецепте мы рассмотрим структуру юнит-теста, наиболее используемые методы PHPUnit и узнаем, как запускать тесты и консоли `yiiic`.

В качестве примера мы, следуя подходу TDD, создадим класс, который генерирует HTML из тегов BBCode. Для простоты ограничимся тегами `[b]`, `[i]` и `[url]`.

Подготовка

Убедитесь, что вы установили необходимые инструменты тестирования, как описано в рецепте «Настройка тестового окружения», и у вас есть готовое к использованию приложение.

Как это делается

1. Для начала определим синтаксис и варианты использования:

Входные данные	Результат
<code>[b]test[/b]</code>	<code>test</code>
<code>[i]test[/i]</code>	<code>test</code>
<code>[url]http://yiiframework.com[/url]</code>	<code>http://yiiframework.com/</code>
<code>[url= http://yiiframework.com/]yiiframework.com[/url]</code>	<code>yiiframework.com</code>
<code>[b]test1[/b] [b]test2[/b]</code>	<code>test1 test2</code>
<code>[b] [i]test[/i] [/b]</code>	<code>test</code>

2. Назовем наш класс `EBBCode`, а его метод для конвертации BBCode в HTML – `EBBCode::process`. Теперь напишем юнит-тесты, для этого надо создать класс `protected/tests/unit/BBCodeTest.php`:

```
<?php
class BBCodeTest extends CTestCase {
    private function process($bbCode) {
        $bb = new EBBCode();
        return $bb->process($bbCode);
    }

    function testSingleTags() {
        $this->assertEquals('<strong>test</strong>',
            $this->process('[b]test[/b]'));
        $this->assertEquals('<em>test</em>',
            $this->process('[i]test[/i]'));
        $this->assertEquals(
            '<a href="http://yiiframework.com/">
                http://yiiframework.com/</a>',
            $this->process(
                ('[url]http://yiiframework.com/[url]'));
        $this->assertEquals(
            '<a href="http://yiiframework.com/">
                yiiframework.com</a>',
            $this->process(
                ('[url=http://yiiframework.com/]
                yiiframework.com[/url]')));
    }

    function testMultipleTags() {
        $this->assertEquals(
            '<strong>test1</strong><strong>test2</strong>',
            $this->process('[b]test1[/b] [b]test2[/b]'));
        $this->assertEquals(
            '<strong><em>test</em></strong>',
            $this->process('[b][i]test[/i][[/b]'));
    }
}
```

3. В тестах выше мы запускаем конвертер с различными входными данными и сверяем результат с тем, что мы ожидаем. Давайте запустим тест и убедимся, что он завершается с ошибкой. Откройте консоль и наберите следующее:

```
cd path/to/protected/tests
phpunit unit/BBCodeTest.php
```

Вы должны получить подобный вывод:

```
D:\web\home\yii\cf\www\protected\tests>phpunit unit/BBCodeTest.php
PHPUnit 3.5.11 by Sebastian Bergmann.

EE

Time: 1 second, Memory: 4.75Mb

There were 2 errors:

1) BBCodeTest::testSingleTags
include(EBBCode.php): failed to open stream: No such file or directory

D:\web\home\yii\cf\framework\YiiBase.php:396
D:\web\home\yii\cf\framework\YiiBase.php:396
D:\web\home\yii\cf\www\protected\tests\unit\BBCodeTest.php:6
D:\web\home\yii\cf\www\protected\tests\unit\BBCodeTest.php:12

2) BBCodeTest::testMultipleTags
include(EBBCode.php): failed to open stream: No such file or directory

D:\web\home\yii\cf\framework\YiiBase.php:396
D:\web\home\yii\cf\framework\YiiBase.php:396
D:\web\home\yii\cf\www\protected\tests\unit\BBCodeTest.php:6
D:\web\home\yii\cf\www\protected\tests\unit\BBCodeTest.php:28

FAILURES!
Tests: 2, Assertions: 0, Errors: 2.

D:\web\home\yii\cf\www\protected\tests>
```

4. ЕЕ означает, что произошло две ошибки, каждая из которых обозначается символом Е, эта же информация дублируется в конце отчета в виде читабельного текста. Также мы видим информацию о причине каждой из ошибок, в нашем случае причина одна и та же – отсутствующая реализация класса EBBCode.
5. Давайте устраним эту ошибку. Для этого создадим файл EBBCode.php и поместим в него класс EBBCode. Создайте protected/components/EBBCode.php:

```
<?php
class EBBCode {
}
```

И снова запустите тесты. Как мы видим, они опять выдают ошибку:

```
D:\web\home\yii\cf\www\protected\tests>phpunit unit/BBCodeTest.php
PHPUnit 3.5.11 by Sebastian Bergmann.

Fatal error: Call to undefined method EBBCode::process() in D:\web\home\yii\cf\www\protected\tests\unit\BBCodeTest.php on line 7
```

6. На этот раз причиной ошибки является вызов нереализованного метода EBBCode::process. Давайте создадим его и запустим

тим тесты снова. В этот раз мы получаем отличный от предыдущего вывод, так как прошлую ошибку мы устранили:

```
D:\web\home\yiicmf\www\protected\tests>phpunit unit/BBCodeTest.php
PHPUnit 3.5.11 by Sebastian Bergmann.

FF

Time: 1 second, Memory: 5.00Mb

There were 2 failures:

1) BBCodeTest::testSingleTags
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
- <strong>test</strong>
+

D:\web\home\yiicmf\www\protected\tests\unit\BBCodeTest.php:12

2) BBCodeTest::testMultipleTags
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
- <strong>test1</strong> <strong>test2</strong>
+

D:\web\home\yiicmf\www\protected\tests\unit\BBCodeTest.php:29

FAILURES!
Tests: 2, Assertions: 2, Failures: 2.

D:\web\home\yiicmf\www\protected\tests>
```

7. Итак, сейчас не происходит фатальных ошибок. Зато PHPUnit сообщает, что для входных данных, переданных в метод в 12-й строке файла `BBCodeTest.php`, мы получаем неверный результат – пустую строку, в то время как ожидается строка `<strong>test</strong>`. Исправим эту ошибку:

```
<?php
class EBBCode {
    function process($string) {
        $preg = array(
            '~\[b\](.*)\[\/b\]~i' => '<strong>$1</strong>',
            '~\[i\](.*)\[\/i\]~i' => '<em>$1</em>',
            '~\[url\](.*)\[\/url\]~i' =>
            '<a href="http://$1>$1</a>',
            '~\[url=([^\]]+)\](.*)\[\/url\]~i' =>
            '<a href="http://$1>$2</a>',
        );
    }
}
```

```

        return preg_replace(array_keys($preg),
            array_values($preg), $string);
    }
}

```

8. Теперь вывод теста должен быть похож на следующий:

```

D:\web\home\yiicmf\www\protected\tests>phpunit unit\BBCodeTest.php
PHPUnit 3.5.11 by Sebastian Bergmann.

.F

Time: 0 seconds, Memory: 5.00Mb

There was 1 failure:

1) BBCodeTest::testMultipleTags
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-<strong>test1</strong> <strong>test2</strong>
+<strong>test1[/b] [b]test2</strong>

D:\web\home\yiicmf\www\protected\tests\unit\BBCodeTest.php:29

FAILURES!
Tests: 2, Assertions: 5, Failures: 1.

D:\web\home\yiicmf\www\protected\tests>_

```

9. .F на скриншоте выше означает, что один тест завершен с ошибкой. Из отчета видно, что конвертер некорректно обрабатывает вложенные теги. Ошибку можно исправить, если использовать нежадные регулярные выражения, добавив к ним модификатор ?:

```

<?php
class EBBCode {
    function process($string) {
        $preg = array(
            '~\[b\](.*)\[\/b\]~i' =>
            '<strong>$1</strong>',
            '~\[i\](.*)\[\/i\]~i' => '<em>$1</em>',
            '~\[url\](.*)\[\/url\]~i' =>
            '<a href="http://$1>$1</a>',
            '~\[url=([^\]]+)\](.*)\[\/url\]~i' =>
            '<a href="http://$1>$2</a>',
        );
    }
}

```

```
        return preg_replace(array_keys($preg),  
                             array_values($preg), $string);  
    }  
}
```

10. И наконец, мы видим именно тот результат, который ожидали увидеть:

```
D:\web\home\yiiicmf\www\protected\tests>phpunit unit/BBCodeTest.php  
PHPUnit 3.5.11 by Sebastian Bergmann.  
  
..  
  
Time: 0 seconds, Memory: 4.75Mb  
  
OK (2 tests, 6 assertions)  
  
D:\web\home\yiiicmf\www\protected\tests>
```

11. Все тесты проходят успешно, и это значит, что класс BBCode реализован верно или, по крайней мере, с учетом запланированных требований.

Предшествующий процесс, который позволил нам сначала создать тест, который завершался ошибкой, а затем реализовать функционал, удовлетворяющий этому тесту, называется **Разработка через тестирование**, или **TDD (Test Driven Development)**. Наличие тестов помогает реализовать запланированный функционал и далее быть уверенным, что он продолжает работать верно после внесения каких-либо изменений в код.

Как это работает

Класс `CTestCase`, который мы использовали в качестве базового для нашего теста, является оболочкой над классом `PHPUnit_Framework_TestCase` из `PHPUnit` и не добавляет какого-либо нового функционала. Он служит для инициализации загрузчика классов `PHPUnit`. Два реализованных нами метода, названия которых начинаются со слова `test`, автоматически запускаются как тестовые методы. Внутри них мы проверяем один метод на разных наборах входных данных и сверяем результат с тем, который мы ожидаем, используя `PHPUnit_Framework_TestCase::assertEquals`. Этот метод проверяет равенство двух переданных в него переменных, и, если они не равны, генерирует ошибку и помечает тест как пройденный.

Дополнительно

Для проверки различных типов значений можно воспользоваться методами `assertTrue`, `assertFileExists` или `assertRegExp` класса `PHPUnit_Framework_TestCase`.

PHPUnit обладает более обширным функционалом, описанным в официальной документации по адресу <http://www.phpunit.de/manual/current/en/>.

Смотрите также

- Рецепт «Настройка тестового окружения».

Фикстуры

Юнит-тесты отлично справляются с тестированием логики классов. Однако когда дело доходит до классов, работающих с окружением и данными, то процесс немного усложняется. На какие данные мы можем опираться во время тестирования? Как получить одно и то же состояние окружения во время выполнения тестов?

Возможность многократного повторного выполнения тестов – основное их преимущество. Именно для обеспечения возможности многократного повторения тестов и необходимо приводить данные и окружение в одно и то же состояние перед запуском тестов. В PHPUnit это достигается с помощью **фиктур**.

В качестве примера напишем тесты для системы купонов, которая обрабатывает регистрацию кодов купонов. Купоны будут храниться в базе данных в таблице `coupon`, имеющей две колонки: `id` и `description`. Для простоты положим, что регистрация заключается в удалении купона и выводе содержимого поля `description`.

Подготовка

- Убедитесь, что вы установили необходимые инструменты тестирования, как описано в рецепте «Настройка тестового окружения», и у вас есть готовое к использованию приложение.
- Создайте модель `Active Record` для таблицы `coupon` `protected/models/Coupon.php`:

```
<?php
class Coupon extends ActiveRecord {
    public static function model($className=__CLASS__) {
```



```

        return parent::model($className);
    }

    public function tableName() {
        return 'coupon';
    }

    public function rules() {
        return array(array('description', 'required'),);
    }
}

```

- Создайте контроллер для манипуляций моделью Coupon protected/components/CouponManager.php:

```

<?php
class CouponManager {
    function registerCoupon($code) {
        $coupon = Coupon::model()->findByPk($code);
        if (!$coupon)
            return false;

        echo "Купон зарегистрирован. $coupon->description";
        return $coupon->delete();
    }
}

```

Как это делается

А сейчас напомним тесты. Внутри protected/tests/unit создайте CouponTest.php. Рассмотрим два тестовых варианта.

1. Первый вариант проверяет обработку существующего в базе купона, второй – несуществующего. Прежде всего необходимо настроить подключение к базе данных, создать таблицы и добавить в них необходимые для теста данные. В конце мы получим следующий код тестового класса:

```

<?php

class CouponTest extends CDbTestCase {
    public $fixtures = array(
        'coupon' => 'Coupon',
    );

    public static function setUpBeforeClass() {
        if (!extension_loaded('pdo') ||
            !extension_loaded('pdo_sqlite'))
            self::markTestSkipped(

```

```
'PDO and SQLite extensions are required.');
```

```
$config = array(  
    'basePath' => dirname(__FILE__),  
    'components' => array(  
        'db' => array(  
            'class' =>  
                'system.db.CDbConnection',  
                'connectionString' =>  
                'sqlite::memory:',  
        ),  
        'fixture' => array(  
            'class' =>  
                'system.test.CDbFixtureManager',  
        ),  
    ),  
);  
Yii::app()->configure($config);  
$c = Yii::app()->getDb()->createCommand();  
$c->createTable('coupon', array(  
    'id' => 'varchar(255) PRIMARY KEY NOT NULL',  
    'description' => 'text',  
));  
}  
  
public static function tearDownAfterClass() {  
    if (Yii::app()->getDb())  
        Yii::app()->getDb()->active = false;  
}  
  
protected function setUp() {  
    parent::setUp();  
    $_GET['existing_code'] = 'discount_for_me';  
    $_GET['non_existing_code'] = 'non_existing';  
}  
  
public function testCodeAcceptance() {  
    $cm = new CouponManager();  
    $this->assertTrue($cm->registerCoupon(  
        $_GET['existing_code']));  
    $this->assertFalse((boolean) Coupon::  
        model()->findByPk(  
            $_GET['existing_code']));  
}  
  
public function testCodeNotFound() {  
    $countBefore = Coupon::model()->count();  
    $cm = new CouponManager();  
    $this->assertFalse($cm->registerCoupon(  
        $_GET['non_existing_code']));
```

```
        $countAfter = Coupon::model()->count();  
        $this->assertEquals($countBefore, $countAfter);  
    }  
}
```

2. Также необходимо задать фикстуру `protected/tests/fixtures/coupon.php`:

```
<?php  
return array(  
    array('id' => 'free_book', 'description' =>  
        'Choose one book for free!'),  
    array('id' => 'merry_christmas', 'description' =>  
        '5% Christmas discount!'),  
    array('id' => 'discount_for_me', 'description' =>  
        '5% discount special for you!'),  
);
```

Ключи `id` и `description` соответствуют полям таблицы `coupon` или свойствам модели `Coupon` и описывают записи, которые будут помещены в таблицу при применении фикстуры.

3. Теперь запустите тест из консоли:

```
cd path/to/protected/tests  
phpunit unit/CouponTest.php
```

Вы должны увидеть следующий вывод:

```
D:\web\home\yiicmf\www\protected\tests>phpunit unit/CouponTest.php  
PHPUnit 3.5.11 by Sebastian Bergmann.  
  
Coupon registered. 5% discount special for you!..  
  
Time: 0 seconds, Memory: 6.50Mb  
  
OK (2 tests, 4 assertions)  
  
D:\web\home\yiicmf\www\protected\tests>
```

Как это работает

Рассмотрим порядок выполнения теста. В этот раз мы использовали специальные методы PHPUnit для работы с фикстурами: `setUpBeforeClass` и `setUp`. Первый метод выполняется сразу после того, как создается экземпляр тестового класса. Обычно в `setUpBeforeClass` помещают инициализацию общих для всех тестовых методов значений. В нашем случае метод содержит проверку наличия расширений PDO и SQLite:

```
if(!extension_loaded('pdo') || !extension_loaded('pdo_sqlite'))  
    self::markTestSkipped('PDO and SQLite extensions are required.');
```

Затем мы конфигурируем приложение для работы в тестовом режиме:

```
$config = array(  
    'basePath' => dirname(__FILE__),  
    'components' => array(  
        'db' => array(  
            'class' => 'system.db.CDbConnection',  
            'connectionString' => 'sqlite::memory:',  
        ),  
        'fixture' => array(  
            'class' => 'system.test.CDbFixtureManager',  
        ),  
    ),  
);  
Yii::app()->configure($config);
```

Для тестирования мы используем базу данных SQLite, расположенную в оперативной памяти. Делается это для того, чтобы ускорить выполнение тестов и избежать создания и удаления файлов базы данных. Также мы подключаем компонент `fixture`, который нужен для выполнения фикстур и заполнения базы необходимыми данными.

Теперь займемся созданием структуры базы данных. В нашем случае необходимо создать таблицу `coupon`:

```
$c = Yii::app()->getDb()->createCommand();  
$c->createTable('coupon', array(  
    'id' => 'varchar(255) PRIMARY KEY NOT NULL',  
    'description' => 'text',  
));
```

Готово. Экземпляр тестового класса создан, и PHPUnit выполняет тестовые методы один за другим. Перед выполнением каждого вызывается метод `setUp`. В нашем случае он выглядит так:

```
parent::setUp();  
$_GET['existing_code'] = 'discount_for_me';  
$_GET['non_existing_code'] = 'non-existing';
```

`parent::setUp` ссылается на метод `CDbTestCase::setUp`, который читает данные из фикстур и помещает их в таблицы или модели. И фикстуры, и модели определяются в свойстве `$fixtures`:

```
public $fixtures = array(  
    'coupon' => 'Coupon',  
);
```

Такое значение свойства `$fixtures` означает, что необходимо поместить данные из фикстуры `protected/tests/fixtures/coupon.php` в модель `Coupon`. Чтобы применить фикстуру к таблице в базе данных без использования модели, можно определить `$fixtures` так:

```
public $fixtures = array(  
    'coupon' => ':coupon',  
);
```

Затем мы устанавливаем переменные окружения, изменяя значения массива `$_GET`. Таким же образом можно задать куки, данные сервера, свойства классов и т. д.

После выполнения `testCodeAcceptance`, но перед выполнением `testCodeNotFound` снова вызывается `setUp`, чтобы восстановить данные в таблице `coupon`.

Когда все тестовые методы отработали, необходимо закрыть соединение с базой – это делается в методе `tearDownAfterClass`, который выполняется непосредственно перед уничтожением класса.

Дополнительно

Мы использовали фикстуры, поэтому нам не приходилось очищать данные вручную при каждом выполнении теста. В случае необходимости удалить ненужные данные можно с помощью метода PHPUnit `tearDown`, который вызывается после выполнения каждого тестового метода.

Дополнительную информацию о фикстурах можно получить здесь:

- <http://www.phpunit.de/manual/current/en/fixtures.html>;
- <http://www.yiiframework.com/doc/guide/ru/test.fixture>.

Смотрите также

- Рецепт «Настройка тестового окружения».

Функциональное тестирование

В то время как юнит-тесты используются для тестирования компонентов или групп компонентов в коде, функциональные тесты позволяют тестировать законченное приложение по принципу «черного ящика», когда неизвестно, что на самом деле происходит в приложении, а лишь проверяется вывод приложения после каких-то действий.

Под выводом приложения подразумевается то, что видно в браузере, а под действиями – различные действия пользователя, например загрузка страницы, переход по ссылке, отправка формы и т. п.

Для примера создадим простой виджет с одной кнопкой, по нажатию на которую отмечаются или снимаются отметки со всех чекбоксов на странице.

Подготовка

- Убедитесь, что вы установили необходимые инструменты тестирования, как описано в рецепте «Настройка тестового окружения», и у вас есть готовое к использованию приложение.
- Убедитесь, что сервер запущен.
- Создайте файл виджета `protected/components/ECheckAllWidget.php` со следующим кодом:

```
<?php
class ECheckAllWidget extends CWidget {
    public $checkedTitle = 'Снять выделение со всех';
    public $uncheckedTitle = 'Отметить все';

    public function run() {
        Yii::app()->clientScript->registerCoreScript(
            'jquery');
        echo CHtml::button($this->uncheckedTitle, array(
            'id' => 'button-' . $this->id,
            'class' => 'check-all-btn',
            'onclick' => '
                switch($(this).val())
                {
                    case "" . $this->checkedTitle . "":
                        $(this).val(
                            "" . $this->uncheckedTitle . "");
                        $("input[type=checkbox]").
                            attr("checked", false);
                        break;
                    case "" . $this->uncheckedTitle . "":
                        $(this).val("" .
                            $this->checkedTitle . "");
                        $("input[type=checkbox]").
                            attr("checked", true);
                        break; } ');
    }
}
```

- Создайте контроллер `protected/controllers/CheckController.php`:

```
<?php
class CheckController extends Controller {
    function actionIndex() {
        $this->render('index');
    }
}
```

- Создайте представление `protected/views/check/index.php`:

```
<?php $this->widget('ECheckAllWidget') ?>
<?php echo CHtml::checkbox('test1', true) ?>
<?php echo CHtml::checkbox('test2', false) ?>
<?php echo CHtml::checkbox('test3', true) ?>
<?php echo CHtml::checkbox('test4', false) ?>
<?php echo CHtml::checkbox('test5', true) ?>
<?php echo CHtml::checkbox('test6', true) ?>
```

Как это делается

Давайте представим, как проходило бы ручное тестирование:

- загрузка страницы;
- проверка заголовка кнопки, он должен быть «Отметить все»;
- нажатие на кнопку;
- проверка того, что все чекбоксы отмечены и заголовок кнопки изменился на «Снять выделение со всех»;
- нажатие на кнопку;
- проверка того, что выделение снято со всех чекбоксов и заголовок кнопки «Отметить все».

А теперь создадим функциональный тест, делающий абсолютно то же самое.

1. Создайте `protected/tests/functional/CheckAllWidgetTest.php`:

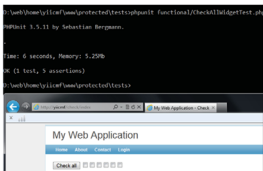
```
<?php
class CheckAllWidgetTest extends WebTestCase {
    public function testWidget() {
        $this->open('check/index');
        $this->assertEquals("Отметить все",
            $this->getAttribute(
                "class=check-all-btn@value"));
        $this->click("class=check-all-btn");
        $this->assertChecked(
            "css=input[type=checkbox]");
        $this->assertEquals("Снять выделение со всех",
            $this->getAttribute(
                "class=check-all-btn@value"));
        $this->click("class=check-all-btn");
    }
}
```

```
$this->assertNotChecked(
    "css=input[type=checkbox]");
$this->assertEquals("Отметить все",
    $this->getAttribute(
        "class=check-all-btn@value"));
    }
}
```

2. Откройте консоль и выполните:

```
cd path/to/protected/tests
phpunit functional/CheckAllWidgetTest.php
```

Вы должны увидеть следующее:



3. А теперь сломайте виджет и запустите тест снова. Если, например, закомментировать обработчик `onclick`, то произойдет следующее:

```
D:\web\home\yii\cf\www\protected\tests>phpunit functional/CheckAllWidgetTest.php
PHPUnit 3.5.11 by Sebastian Bergmann.

Time: 5 seconds, Memory: 5.25Mb

There was 1 failure:

1) CheckAllWidgetTest::testWidget
Current URL: http://yii\cf/check/index
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
Uncheck all
+Check all

D:\web\home\yii\cf\www\protected\tests\functional\CheckAllWidgetTest.php:11

FAILURES!
Tests: 1, Assertions: 3, Failures: 1.
```


Это значит, что тест завершился с ошибкой на строке 11:

```
$this->assertEquals("Снять выделение со всех",  
$this->getAttribute("class=check-all-btn@value"));
```

так как вместо ожидаемого значения «Снять выделение со всех» получено «Отметить все».

Как это работает

PHPUnit начинает один за одним выполнять методы, названия которых начинаются с `test`. У нас только один такой метод – `testWidget`, именно он и будет выполнен:

```
$this->open('check/index');
```

Так загружается страница для тестирования.

```
$this->assertEquals("Отметить все",  
$this->getAttribute ("class=check-all-btn@value"));
```

Так мы получаем текст на кнопке и сверяем его с ожидаемым значением. Метод `getAttribute` ищет в DOM элемент с классом `check-all-btn` и возвращает значение атрибута `value` этого элемента.

```
$this->click("class=check-all-btn");
```

Здесь происходит нажатие на элемент с классом `check-all-btn`, то есть на кнопку.

```
$this->assertChecked("css=input[type=checkbox]");
```

Методы `assertChecked` и `assertNotChecked` проверяют, соответственно, отмечен чекбокс или нет. В качестве аргумента мы используем CSS-селектор `input[type=checkbox]`, чтобы получить все чекбоксы на странице.

Все, что нам требуется, – это вызывать методы как `getAttribute` и `assertChecked`, всю работу с DOM и имитацию различных действий пользователя Selenium берет на себя.

И ещё

Чтобы узнать больше о функциональном тестировании, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/guide/ru/test.functional>;
- <http://www.phpunit.de/manual/current/en/selenium.html>;
- http://seleniumhq.org/docs/05_selenium_rc.html;
- http://seleniumhq.org/docs/04_selenese_commands.html.

Смотрите также

- Рецепт «Настройка тестового окружения».
- Рецепт «Написание и запуск юнит-тестов».

Генерация отчетов о покрытии кода

Важно знать, насколько полно код приложения покрыт тестами. Если вы разрабатываете в одиночку, то вы, вероятно, можете оценить это насквозь. Но в случае, когда приложение разрабатывается командой разработчиков или вы поддерживаете старый проект, такая оценка становится невозможной. К счастью, PHPUnit и Xdebug позволяют генерировать отчеты о покрытии кода тестами. Эти отчеты содержат информацию о том, насколько хорошо приложение протестировано, какие строки кода выполняются во время тестирования, а какие – нет.

Для примера давайте сгенерируем отчет для базового класса ядра Yii.

Подготовка

Тесты ядра фреймворка не включены в дистрибутив, поэтому нам придется скачать их из репозитория GIT.

Убедитесь, что тестовое окружение установлено корректно.

Как это делается

1. Перейдите по ссылке <https://github.com/yiisoft/yii> и скачайте последнюю версию фреймворка при помощи консольной команды (`git clone git@github.com:yii.git`) или других графических клиентов для работы с GIT. Также последняя версия фреймворка доступна по адресу <https://github.com/yiisoft/yii/zipball/master> в виде zip-архива.
2. Введите в консоли следующее:

```
cd path/to/checked/out/code/tests/ phpunit  
-coverage-html reports framework/base
```

где `path/to/checked/out/code/tests/` – путь к скачанному из репозитория тестам.
3. После того как отчеты сгенерированы, перейдите в папку `path/to/checked/out/code/tests/reports` и откройте файл `index.html` в браузере.

Как это работает

Отчет о покрытии кода скажет, насколько хорошо протестировано приложение, и какие части кода нуждаются в дополнительном тестировании.

Отчет генерируется только для тех тестов, которые были запущены, поэтому для более полного отчета следует запускать все тесты приложения. В примере выше для простоты и скорости была выполнена только малая часть тестов ядра Yii.

Первая страница отчета дает нам обзор файлов, которые были протестированы, а также информацию о количестве выполненных строк кода, количестве выполненных функций и классов. Так как нам важно покрыть тестами как можно больше кода, то элементы в отчете выделены зеленым или красным в зависимости от того, насколько качественно они протестированы. Из отчета ниже видно, что необходимо написать больше тестов для **CApplication** и **CStatePersister**:

Current directory: C:\projects\yii_trunk\framework\base (dashboard)									
Legend: Low: 0% to 30% Medium: 35% to 70% High: 75% to 100%									
	Lines		Coverage		Classes				
Items									
Items		64.63% 559 / 873		66.66% 90 / 132		41.67% 5 / 12			
CApplication.php		21.28% 35 / 172		21.28% 30 / 140		0.00% 0 / 1			
CApplicationComponent.php		100.00% 4 / 4		100.00% 2 / 2		100.00% 1 / 1			
CBehavior.php		100.00% 22 / 22		100.00% 6 / 6		100.00% 1 / 1			
CComponent.php		76.26% 143 / 190		76.00% 19 / 25		66.67% 2 / 3			
CEvent.php		73.12% 96 / 134		66.67% 24 / 36		0.00% 0 / 1			
CEventBehavior.php		71.43% 5 / 7		60.00% 2 / 4		0.00% 0 / 1			
CEventFilter.php		100.00% 1 / 1				100.00% 1 / 1			
CEventObject.php		87.50% 120 / 137		62.50% 17 / 27		0.00% 0 / 1			
CEventSubscriber.php		79.69% 64 / 81		61.64% 8 / 13		0.00% 0 / 1			
CErrorHandler.php		68.00% 17 / 25		66.67% 2 / 3		0.00% 0 / 1			

Если кликнуть на имя класса, к примеру, на **CComponent**, то мы увидим детальную информацию. На скриншоте ниже показан отчет для класса:

Legend: executed not executed dead code					
				Coverage	
	Classes			Functions / Methods	
Total		66.67%	2 / 3		75.00% 1
CComponent		0.00%	0 / 1		75.00% 1
get (Name)					0.00%
set (Name, \$value)					0.00%
isset (Name)					0.00%
unset (Name)					0.00%
call (Name, \$parameters)					100.00%
as (\$behavior)					100.00%
attachBehaviors (\$behaviors)					100.00%

Дополнительно

За дополнительной информацией по генерации отчетов о покрытии кода обратитесь к следующим ресурсам:

- <http://www.phpunit.de/manual/current/en/code-coverage-analysis.html>;
- <http://www.phpunit.de/manual/current/en/selenium.html>.

Смотрите также

- Рецепт «Настройка тестового окружения».



ГЛАВА 6.

База данных, Active record и трюки с моделями

В этой главе мы рассмотрим следующие темы:

- Получение данных из базы данных.
- Создание и использование нескольких подключений к базам данных.
- Использование именованных групп условий для создания многоязычных моделей.
- Обработка полей модели с помощью методов-событий Active Record.
- Применение markdown и HTML.
- Подсветка кода с помощью Yii.
- Автоматические timestamp.
- Автоматическое указание автора.
- Реализация наследования с одной таблицей.
- Использование CDbCriteria.

Введение

В Yii используются три основных метода работы с базами данных: Active Record, конструктор запросов и прямые запросы SQL посредством DAO. Эти методы различаются синтаксисом, возможностями и производительностью.

Мы изучим, как эффективно работать с базами данных, когда использовать или не использовать модели, как работать с несколькими базами данных, как автоматически выполнять предварительную обработку полей Active Record и как использовать мощные критерии баз данных.

В этой главе мы воспользуемся образцом базы данных Sakila версии 0.8, доступной на официальной веб-странице MySQL: <http://dev.mysql.com/doc/sakila/en/index.html>.

Получение данных из базы данных

Большинство современных приложений используют базы данных. Будь то веб-сайт или социальная сеть, по крайней мере некоторые части всегда работают с базой данных. Yii предлагает три способа работы с базами данных:

- Active Record;
- конструктор запросов;
- SQL через DAO.

Далее мы попробуем все эти методы для получения данных из таблиц `film`, `film_actor` и `actor` и вывода их в виде списка. Потом мы измерим время выполнения и использование оперативной памяти, чтобы определить, в каких случаях лучше применять тот или иной метод.

Подготовка

- Создайте новое приложение с помощью `yii webapp`, как это описано в официальном руководстве по адресу <http://www.yiiframework.com/doc/guide/ru/quickstart.first-app>.
- Скачайте базу данных Sakila со следующего адреса: <http://dev.mysql.com/doc/sakila/en/index.html>.

Выполните скачанные скрипты SQL: сначала создайте схему, потом добавьте данные.

- Настройте подключение к базе данных в файле `protected/config/main.php`, чтобы использовать базу данных `sakila`.
- С помощью `Gii` создайте таблицы моделей `actor` и `film`.

Как это делается

1. Создадим файл `protected/controllers/DbController.php` следующим образом:

```
<?php
class DbController extends Controller
{
    protected function afterAction($action)
```

```
{
    $time = sprintf('%0.5f', Yii::getLogger()
        ->getExecutionTime());
    $memory = round(memory_get_peak_usage()/
        (1024*1024),2)."MB";
    echo "Time: $time, memory: $memory";
    parent::afterAction($action);
}
public function actionAr()
{
    $actors = Actor::model()->findAll(array('with' => 'films',
        'order' => 't.first_name, t.last_name, films.title'));
    echo '<ol>';
    foreach($actors as $actor)
    {
        echo '<li>';
        echo $actor->first_name.' '.$actor->last_name;
        echo '<ol>';
        foreach($actor->films as $film)
        {
            echo '<li>';
            echo $film->title;
            echo '</li>';
        }
        echo '</ol>';
        echo '</li>';
    }
    echo '</ol>';
}
public function actionQueryBuilder()
{
    $rows = Yii::app()->db->createCommand()
        ->from('actor')
        ->join('film_actor', 'actor.actor_id=film_actor.actor_id')
        ->leftJoin('film', 'film.film_id=film_actor.film_id')
        ->order('actor.first_name, actor.last_name, film.title')
        ->queryAll();

    $this->renderRows($rows);
}

public function actionSql()
{
    $sql = "SELECT *
    FROM actor a
        JOIN film_actor fa ON fa.actor_id = a.actor_id
        JOIN film f ON fa.film_id = f.film_id
        ORDER BY a.first_name, a.last_name, f.title";
}
```



```

$rows = Yii::app()->db->createCommand($sql)->
queryAll();

$this->renderRows($rows);
}

public function renderRows($rows)
{
    $lastActorName = null;
    echo '<ol>';
    foreach($rows as $row)
    {
        $actorName = $row['first_name'].'
        '.$row['last_name'];
        if($actorName!=$lastActorName){
            if($lastActorName!=null){
                echo '</ol>';
                echo '</li>';
            }
            $lastActorName = $actorName;
            echo '<li>';
            echo $actorName;
            echo '<ol>';
        }
        echo '<li>';
        echo $row['title'];
        echo '</li>';
    }
    echo '</ol>';
}
}

```

Выше приведен код действий (*actions*), соответствующих трем различным методам доступа к базе данных.

- После запуска `db/ar`, `db/queryBuilder` и `db/sql` вы должны получить дерево из 200 актеров и 1000 фильмов, в которых они снимались, как показано на скриншоте на рисунке ниже (см. стр. 157).

В самом низу можно увидеть статистику по использованию памяти и времени выполнения. Когда вы будете выполнять приведенный код, у вас могут получиться другие абсолютные значения, но пропорции между методами должны быть примерно те же:

Метод	Использование памяти, мегабайт	Время выполнения, секунды
Active Record	19.74	1.14109
Конструктор запросов	17.98	0.35732
SQL (DAO)	17.74	0.35038

```
36. THEORY MERMAID
37. TITANIC BOONDOCK
38. UNFORGIVEN ZOOLANDER
39. WAGON JAWS
40. YOUTH KICK
199. JULIA FAWCETT
    1. BERETS AGENT
    2. BOILED DARES
    3. CHISUM BEHAVIOR
    4. CLOSER BANG
    5. DAY UNFAITHFUL
    6. HOPE TOOTSIE
    7. LUKE MUMMY
    8. MULAN MOON
    9. OPUS ICE
    10. POLLOCK DELIVERANCE
    11. RIDGEMONT SUBMARINE
    12. SHANGHAI TYCOON
    13. SHAWSHANK BUBBLE
    14. THEORY MERMAID
    15. WAIT CIDER
200. THORA TEMPLE
    1. AFRICAN EGG
    2. BADMAN DAWN
    3. BLANKET BEVERLY
    4. CANDIDATE PERDITION
    5. CAROL TEXAS
    6. CHRISTMAS MOONSHINE
    7. GALAXY SWEETHEARTS
    8. HOCUS FRIDA
    9. INSIDER ARIZONA
    10. INTERVIEW LIAISONS
    11. JADE BUNCH
    12. LOVER TRUMAN
    13. LOVERBOY ATTACKS
    14. MADISON TRAP
    15. RANDOM GO
    16. TELEGRAPH VOYAGE
    17. TROJAN TOMORROW
    18. VIRGINIAN PLUTO
    19. WARDROBE PHANTOM
    20. WRONG BEHAVIOR
```

Time: 0.67937, memory: 17.08MB

Как это работает

Разберем вышеприведенный код.

Метод действия `actionAr` получает экземпляры моделей с использованием `Active Record`. Мы начинаем с модели `Actor`, сгенерированной Gii, чтобы получить список всех актеров, и указываем `'with' =>`

'films', чтобы получить список соответствующих фильмов через отношение, которое нам создал генератор кода Gii на основе внешних ключей InnoDB. При этом используется всего один SQL-запрос. Это называется «жадной загрузкой». Далее мы просто обходим полученные данные. Для каждого элемента выводится его имя.

Метод `actionQueryBuilder` использует построитель запросов. Сначала мы создаем команду запроса для текущего соединения при помощи `Yii::app()->db->createCommand`. Потом добавляем поэтапно части запроса, используя методы `from`, `join` и `leftJoin`. Эти методы автоматически экранируют значения, имена таблиц и полей. Метод `queryAll()` возвращает массив необработанных строк базы данных. Каждая строка также является массивом, индексируемым по именам полей. Результат мы передаем функции `renderRows`, которая обрабатывает строки.

В функции `actionSql` мы делаем то же самое, но передаем SQL-запрос напрямую вместо добавления к нему частей по одной. Стоит отметить, что мы должны экранировать значения перед их использованием в строке запроса с помощью `Yii::app()->db->quoteValue`.

Функция `renderRows` обрабатывает строки из построителя запросов и DAO. При этом необходимо делать больше проверок. Это менее естественно, чем обработка результатов Active Record.

Можно заметить, что при одинаковом конечном результате получаются разная производительность, разный синтаксис и возможности. Сравним методы и сделаем выводы об их применимости:

Метод	Active Record	Конструктор запросов	SQL (DAO)
Синтаксис	Построит и выполнит SQL-запрос за вас. Gii создаст модели и отношения. Работает с моделями полностью в объектно-ориентированном стиле. Очень чистый API. Выдает массивы моделей с корректной вложенностью	Чистый API, пригодный для создания запросов на лету. Возвращает массивы необработанных данных	Удобен для сложных SQL-запросов. Значения и ключевые слова нужно экранировать вручную. Не очень удобно создавать запросы на лету. Возвращает массивы необработанных данных

Метод	Active Record	Конструктор запросов	SQL (DAO)
Производительность	Более требователен к объему оперативной памяти, дольше выполняется в сравнении с SQL и строителем запросов.	Нормальная	Нормальная
Дополнительные возможности	Автоматически экранирует значения и имена. Поведения. Обработчики. Валидация.	Автоматически экранирует значения и имена	Нет
Наилучшее применение	Прототипирование запросов. Изменение, удаление и создание одиночных моделей (очень удобно использовать модели с формами).	Работа с большими объемами данных, создание запросов на лету.	Сложные запросы на чистом SQL с максимальной производительностью.

И ещё

Чтобы узнать подробнее о работе с базами данных в Yii, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/guide/ru/database.dao>;
- <http://www.yiiframework.com/doc/guide/ru/database.query-builder>;
- <http://www.yiiframework.com/doc/guide/ru/database.ar>.

Смотрите также

- Рецепт «Создание и использование нескольких подключений к базам данных» в этой главе.
- Рецепт «Использование CDbCriteria» в этой главе.

Создание и использование нескольких подключений к базам данных

В создаваемых с нуля веб-приложениях нечасто используется более чем одно подключение к базе данных. Тем не менее, если вы создаете дополнение для существующей системы, это может понадобиться.

Из данного рецепта вы узнаете, как создать несколько подключений к базам данных и использовать их с моделями DAO, Active Record и построителем запросов.

Подготовка

- Создайте новое приложение с помощью `yiiic`, как описано в официальном руководстве по адресу <http://www.yiiframework.com/doc/guide/ru/quickstart.first-app>.
- Создайте две базы данных MySQL с именами `db1` и `db2`.
- Создайте таблицу с именем `post` в базе `db1` следующим образом:

```
DROP TABLE IF EXISTS 'post';
CREATE TABLE IF NOT EXISTS 'post' (
    'id' INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'title' VARCHAR(255) NOT NULL,
    'text' TEXT NOT NULL,
    PRIMARY KEY ('id')
);
```

- Создайте таблицу с именем `comment` в базе `db2` следующим образом:

```
DROP TABLE IF EXISTS 'comment';
CREATE TABLE IF NOT EXISTS 'comment' (
    'id' INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'text' TEXT NOT NULL,
    'postId' INT(10) UNSIGNED NOT NULL,
    PRIMARY KEY ('id')
);
```

Как это делается

1. Начнем с настройки подключения к базе данных. Откройте файл `protected/config/main.php` и определите основное подключение, как описано в руководстве:

```
'db'=>array(
    'connectionString' => 'mysql:host=localhost;
dbname=db1',
    'emulatePrepare' => true,
    'username' => 'root',
    'password' => '',
    'charset' => 'utf8',
),
```

2. Скопируйте строки, заменив 'db' на 'db2', и соответственно измените значение connectionString. Также нужно добавить имя класса следующим образом:

```
'db2'=>array(
    'class'=>'CDbConnection',
    'connectionString' => 'mysql:host=localhost;
dbname=db2',
    'emulatePrepare' => true,
    'username' => 'root',
    'password' => '',
    'charset' => 'utf8',
),
```

3. Вот и всё. Теперь у вас есть два подключения, которые вы можете использовать с DAO и конструктором запросов следующим образом:

```
$db1Rows = Yii::app()->db->createCommand($sql)->
queryAll();
$db2Rows = Yii::app()->db2->createCommand($sql)->
queryAll();
```

Что, если нам необходимо использовать Active Record?

1. Во-первых, надо создать модели Post и Comment с помощью Gii. Начиная с версии 1.1.11 в Gii можно выбрать нужное соединение с БД для каждой модели.
2. Теперь можно использовать модель Comment, как обычно. Создайте файл protected/controllers/DbtestController.php со следующим содержанием:

```
<?php
class DbtestController extends CController
{
    public function actionIndex()
    {
        $post = new Post();
        $post->title = "Post #".rand(1, 1000);
        $post->text = "text";
```

```
$post->save();

echo '<h1>Posts</h1>';

$posts = Post::model()->findAll();
foreach($posts as $post)
{
    echo $post->title."<br />";
}

$comment = new Comment();
$comment->postId = $post->id;
$comment->text = "comment #" . rand(1, 1000);
$comment->save();

echo '<h1>Comments</h1>';

$comments = Comment::model()->findAll();
foreach($comments as $comment)
{
    echo $comment->text."<br />";
}
}
```

3. Запустите `dbtest/index` несколько раз. Вы должны увидеть записи, добавленные в обе базы данных, как на следующем скриншоте:



Как это работает

Используя Yii, вы можете добавлять и настраивать компоненты, редактируя конфигурационный файл. Для нестандартных компонентов, таких как `db2`, вы должны указывать класс компонента. Похожим

образом можно определить компоненты db3, db4 и любые другие, например facebookApi. Имеющиеся пары ключ/значение будут назначены public свойствам соответствующих компонентов.

И ещё

В зависимости от используемой СУБД можно применять дополнительные возможности для нескольких подключений к базам данных.

Межбазовые связи

Если вы используете MySQL, то можете создавать межбазовые связи для ваших моделей. Чтобы сделать это, следует использовать имя таблицы модели Comment с именем базы данных в качестве префикса, как в следующем примере:

```
class Comment extends CActiveRecord
{
    //...
    public function tableName()
    {
        return 'db2.comment';
    }
    //...
}
```

Теперь, если у вас определено отношение comments в методе relations модели Post, то вы можете использовать следующий код:

```
$posts = Post::model()->with('comments')->findAll();
```

Полезные материалы

За подробностями обращайтесь по следующему адресу: <http://www.yiiframework.com/doc/api/CActiveRecord>.

Смотрите также

- Рецепт «Получение данных из базы данных» в этой главе.

Использование именованных групп условий для создания многоязычных моделей

Интернационализация приложения может быть непростой задачей: нужно перевести интерфейс, сообщения, использовать корректный

формат даты и т. д. Yii поможет с этим справиться, предоставив разработчику доступ к данным **CLDR** (Unicode Common Locale Data Repository, общий репозиторий данных локалей юникода) и инструментов для перевода и форматирования. Если приложение обрабатывает данные с использованием нескольких языков, то готового рецепта на такой случай нет.

Из этого рецепта вы узнаете, как можно реализовать удобную функцию модели, которая позволит работать с записями на нескольких языках.

Подготовка

- Создайте новое приложение с помощью `yii webapp`, как это описано в официальном руководстве по адресу <http://www.yiiframework.com/doc/guide/ru/quickstart.first-app>.
- Подключитесь к базе данных и создайте таблицу с именем `post` следующим образом:

```
DROP TABLE IF EXISTS 'post';
CREATE TABLE IF NOT EXISTS 'post' (
    'id' INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'lang' VARCHAR(5) NOT NULL DEFAULT 'en',
    'title' VARCHAR(255) NOT NULL,
    'text' TEXT NOT NULL,
    PRIMARY KEY ('id')
);

INSERT INTO 'post'('id','lang','title','text')
VALUES (1,'en_us','Yii news','Text in English'),
(2,'de','Yii Nachrichten','Text in Deutsch');
```

- Создайте с помощью Gii модель `Post`.

Как это делается

1. Добавьте следующие методы к файлу `protected/models/Post.php`:

```
class Post extends ActiveRecord
{
    public function defaultScope()
    {
        return array(
            'condition' => "lang=:lang",
            'params' => array(
                ':lang' => Yii::app()->language,
            )
        );
    }
}
```

```
        ),
    );
}

public function lang($lang){
    $this->getDbCriteria()->mergeWith(array(
        'condition' => "lang=:lang",
        'params' => array(
            ':lang' => $lang,
        ),
    ));
    return $this;
}
}
```

Вот и всё. Теперь можно использовать нашу модель. Создайте файл `protected/controllers/DbtestController.php` со следующим содержанием:

```
<?php
class DbtestController extends CController
{
    public function actionIndex()
    {
        // Получаем записи на языке приложения по
        умолчанию
        $posts = Post::model()->findAll();

        echo '<h1>Default language</h1>';
        foreach($posts as $post)
        {
            echo '<h2>'.$post->title.'</h2>';
            echo $post->text;
        }

        // Получаем записи на немецком
        $posts = Post::model()->lang('de')->findAll();

        echo '<h1>German</h1>';
        foreach($posts as $post)
        {
            echo '<h2>'.$post->title.'</h2>';
            echo $post->text;
        }
    }
}
```

2. Теперь запустите `dbtest/index`, и у вас должно получиться нечто, похожее на следующий скриншот:

Default language

Yii news

Text in English

German

Yii Nachrichten

Text in Deutsch

Как это работает

В вышеприведённом коде мы использовали именованные условия Active Record. Функция `defaultScope` возвращает набор условий (критерий), который будет применен ко всем методам модели `Post`, выполняющим запросы. Поскольку нам необходима возможность явно указывать язык, мы создали именованное условие с именем `lang`, которое принимает в качестве аргумента название языка. С помощью `$this->getDbCriteria()` мы получаем текущий критерий модели и объединяем его с новым. Поскольку условие такое же, как и в `defaultScope`, за исключением значения параметра, будет использоваться новое значение.

Чтобы поддерживать вызовы по цепочке, `lang` возвращает экземпляр модели.

И ещё

За подробностями обращайтесь по адресам:

- <http://www.yiiframework.com/doc/guide/ru/database.ar>;
- <http://www.yiiframework.com/doc/api/CDbCriteria/>.

Смотрите также

- Рецепт «Получение данных из базы данных» в этой главе.
- Рецепт «Использование `CDbCriteria`» в этой главе.
- Рецепт «Обработка полей модели с помощью методов-событий Active Record» в этой главе.

Обработка полей модели с помощью методов-событий Active Record

Реализация Active Record в Yii очень мощная и предоставляет множество возможностей. Одной из этих возможностей являются методы-события, которые можно использовать, например для предварительной обработки полей модели перед помещением их в базу данных или извлечения из нее, или для удаления связанных данных.

В этом рецепте мы превратим в ссылки все адреса в тексте сообщения и перечислим все существующие методы-события модели Active Record.

Подготовка

- Создайте новое приложение с помощью `yii create webapp`, как это описано в официальном руководстве по адресу <http://www.yiiframework.com/doc/guide/ru/quickstart.first-app>.
- Подключитесь к базе данных и создайте таблицу с именем `post` следующим образом:

```
DROP TABLE IF EXISTS 'post';
CREATE TABLE IF NOT EXISTS 'post' (
    'id' INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'title' VARCHAR(255) NOT NULL,
    'text' TEXT NOT NULL,
    PRIMARY KEY ('id')
);
```

- Создайте модель `Post` с помощью `Gii`.

Как это делается

1. Добавьте следующий метод в файл `protected/models/Post.php`:

```
protected function beforeSave()
{
    $this->text = preg_replace('~((?:https?|https?)://.*?)
    (|$)~iu', '<a href="\1">\1</a>\2', $this->text);
    return parent::beforeSave();
}
```

2. Вот и всё. Теперь попробуйте сохранить сообщение, содер-

жащее ссылку. Создайте файл `protected/controllers/TestController.php` со следующим содержанием:

```
<?php
class TestController extends CController
{
    function actionIndex()
    {
        $post=new Post();
        $post->title='links test';
        $post->text='test http://www.yiiframework.
com/ test';
        $post->save();
        print_r($post->text);
    }
}
```

3. Выполните `test/index`. У вас должно получиться следующее:

test <http://www.yiiframework.com/> test

Как это работает

Функция `beforeSave()` реализована в классе `CActiveRecord` и выполняется как раз перед сохранением модели. Используя регулярное выражение, мы заменяем все, что выглядит как URL, ссылкой, которая использует этот URL. Не забываем вызвать метод родителя, чтобы верно сработали настоящие события. Стоит отметить, что, для того чтобы сохранение не выполнилось, можно вернуть `false`.

И ещё

Все существующие методы-события модели AR приведены в следующей таблице:

Имя метода	Описание
<code>afterConstruct</code>	Вызывается после создания экземпляра модели оператором <code>new</code>
<code>beforeDelete/afterDelete</code>	Вызывается до/после удаления записи
<code>beforeFind/afterFind</code>	Метод вызывается до/после создания экземпляра записи в методе <code>find</code>
<code>beforeSave/afterSave</code>	Метод вызывается до/после успешного сохранения записи
<code>beforeValidate/afterValidate</code>	Метод вызывается до/после окончания валидации

Полезные материалы

Чтобы узнать подробнее о методах-событиях в Yii, можете обратиться по следующим адресам:

- <http://www.yiiframework.com/doc/api/CActiveRecord/>;
- <http://www.yiiframework.com/doc/api/CModel>.

Смотрите также

- Рецепт «Использование событий Yii» в главе 1, Под капотом.
- Рецепт «Применение markdown и HTML» в этой главе.
- Рецепт «Подсветка кода с помощью Yii» в этой главе.
- Рецепт «Автоматический timestamp» в этой главе.
- Рецепт «Автоматическое указание автора» в этой главе.

Применение markdown и HTML

При создании приложения чаще всего стоит задача реализовать инструмент для создания материалов. Разумеется, можно делать это с использованием простого текста или HTML, но текст часто слишком прост, а HTML слишком сложен и небезопасен. Именно поэтому используют специальные языки разметки, например BBCode, Textile или markdown.

В этом рецепте мы научимся создавать модели, в которых markdown автоматически конвертируется в HTML при сохранении.

Подготовка

- Создайте новое приложение с помощью `yii create webapp`, как это описано в официальном руководстве по адресу <http://www.yiiframework.com/doc/guide/ru/quickstart.first-app>.
- Подключитесь к базе данных и создайте таблицу с именем `post` следующим образом:

```
DROP TABLE IF EXISTS 'post';
CREATE TABLE IF NOT EXISTS 'post' (
    'id' INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'title' VARCHAR(255) NOT NULL,
    'text' TEXT NOT NULL,
    'html' TEXT NOT NULL,
    PRIMARY KEY ('id')
);
```

- Создайте модель `Post` при помощи Gii.

Как это делается

1. Откройте файл `protected/models/Post.php` и добавьте следующий метод:

```
protected function beforeValidate()
{
    $parser=new CMarkdownParser();
    $this->html=$parser->transform($this->text);
    return parent::beforeValidate();
}
```

2. Теперь модель `Post` можно использовать без каких-либо дополнений в коде контроллера `protected/controllers/TestController.php`:

```
<?php
class TestController extends CController
{
    function actionIndex()
    {
        $post = new Post();
        $post->title = "I promise to share my opinion
            on Yii framework";
        $post->text = "Recently I've started using [Yii
            framework](http://www.yiiframework.com/) and
            definitely will share my opinion as soon as
            I'll have some more free time.";
        $post->save();

        echo "<h1>$post->title</h1>";
        echo $post->html;
    }
}
```

3. Вот и все. Теперь запустите `test/index`. У вас должно получиться следующее:

I promise to share my opinion on Yii framework

Recently I've started using [Yii framework](#) and definitely will share my opinion as soon as I'll have some more free time.

Текст, размеченный `markdown`, который вы присвоили свойству `text`, будет автоматически сконвертирован в HTML, пригодный для вывода на экран, и сохранен в поле `html` таблицы `post`. Таким образом, `html` следует использовать при отображении материалов, а `text`, размеченный `markdown`, – при их создании и редактировании.

Как это работает

В приведенном выше коде для предварительной обработки полученных от пользователя данных мы перекрываем метод `CActiveRecord::beforeValidate`. Данный метод выполняется непосредственно перед валидацией, которая осуществляется при вызове `$post->save()`.

В состав Yii входит обертка над библиотекой «PHP Markdown Extra». `CMarkdownParser` используется при генерации документации фреймворка. Почему бы и нам ей не воспользоваться?

Преобразование текста из одного формата в другой требует значительных ресурсов: как процессорной мощности, так и памяти. Поэтому следует избегать таких преобразований, если это возможно. В нашем случае преобразование происходит единожды при сохранении записи.

При редактировании записи необходимо как-то получить исходную разметку в формате markdown. Для этого в базе данных мы храним как исходную разметку, так и получающийся HTML. Альтернативное решение – преобразовывать markdown в HTML непосредственно при просмотре записи и кешировать результат до тех пор, пока запись не обновится.

И ещё

Чтобы разобраться с markdown, вы можете обратиться по следующим адресам:

Синтаксис markdown

- <http://daringfireball.net/projects/markdown/syntax>;
- <http://michelf.com/projects/php-markdown/extra/>.

Обработка и использование markdown в Yii

- <http://www.yiiframework.com/doc/api/CMarkdownParser/>.

Смотрите также

- Рецепт «Обработка полей модели с помощью методов-событий Active Record» в этой главе
- Рецепт «Подсветка кода с помощью Yii» в этой главе.
- Рецепт «Автоматический timestamp» в этой главе.
- Рецепт «Автоматическое указание автора» в этой главе.

Подсветка кода с помощью Yii

Если вы размещаете сообщение, содержащее код, будь то во внутренней вики компании или в блоге разработчика, всегда пригодится подсветка кода.

В состав Yii входит класс `Text_highlighter` из `Peag`. Он используется для подсветки кода в примерах полного руководства по Yii и может применяться для тех же целей в наших приложениях.

В этом рецепте мы создадим простое приложение, которое позволит добавлять, редактировать и просматривать фрагменты кода.

Подготовка

- Создайте новое приложение с помощью `yiiic webapp`, как это описано в официальном руководстве по адресу <http://www.yiiframework.com/doc/guide/ru/quickstart.first-app>.
- Подключитесь к базе данных и создайте таблицу с именем `snippet` следующим образом:

```
CREATE TABLE 'snippet' (  
    'id' INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,  
    'title' VARCHAR(255) NOT NULL,  
    'code' TEXT NOT NULL,  
    'html' TEXT NOT NULL,  
    'language' VARCHAR(20) NOT NULL,  
    PRIMARY KEY ('id')  
);
```

- Создайте модель `Snippet` при помощи `Gii`.

Как это делается

1. Для начала подправим код модели `protected/models/Snippet.php`. Измените метод `rules` следующим образом:

```
public function rules()  
{  
    return array(  
        array('title, code, language', 'required'),  
        array('title', 'length', 'max'=>255),  
        array('language', 'length', 'max' => 20),  
    );  
}
```

2. Добавьте в модель `Snippet` следующие методы:
`protected function afterValidate()`

```
{
    $highlighter = new CTextHighlighter();
    $highlighter->language = $this->language;
    $this->html = $highlighter->highlight($this->code);

    return parent::afterValidate();
}

public function getSupportedLanguages()
{
    return array(
        'php' => 'PHP',
        'css' => 'CSS',
        'html' => 'HTML',
        'javascript' => 'JavaScript',
    );
}
```

3. Модель готова. Теперь нужен контроллер. Создайте файл `protected/controllers/SnippetController.php` следующего содержания:

```
<?php
class SnippetController extends CController
{
    public function actionIndex()
    {
        $criteria = new CDbCriteria();
        $criteria->order = 'id DESC';
        $models = Snippet::model()->findAll();
        $this->render('index', array(
            'models' => $models,
        ));
    }

    public function actionView($id)
    {
        $model = Snippet::model()->findByPk($id);
        if(!$model)
            throw new Cexception(404);

        $this->render('view', array(
            'model' => $model,
        ));
    }

    public function actionAdd()
    {
        $model = new Snippet();
```

```
$data = Yii::app()->request->getPost('Snippet');
if($data)
{
    $model->setAttributes($data);
    if($model->save())
        $this->redirect(array('view', 'id' =>
$model->id));
    }
    $this->render('add', array(
        'model' => $model,
    ));
}

public function actionEdit($id)
{
    $model = Snippet::model()->findByPk($id);
    if(!$model)
        throw new CHttpException(404);

    $data = Yii::app()->request->getPost('Snippet');
    if($data)
    {
        $model->setAttributes($data);
        if($model->save())
            $this->redirect(array('view', 'id' =>
$model->id));
        }
        $this->render('edit', array(
            'model' => $model,
        ));
    }
}
```

4. Теперь представления; создайте файл `protected/views/snippet/index.php`:

```
<h2>Snippets</h2>
<?php echo CHtml::link('Add snippet', array('add'))?>
<ol>
<?php foreach($models as $model)?>
    <li>
        <?php echo CHtml::link(
            CHtml::encode($model->title),
            array('view', 'id' => $model->id)
        )?>
    </li>
<?php endforeach?>
</ol>
```

5. Создайте файл `protected/views/snippet/view.php`:

```
<h2><?php echo CHtml::link('Snippets',  
array('index'))?> → <?php  
    echo CHtml::encode($model->title)?>  
</h2>  
<?php echo CHtml::link('Edit', array  
    ('edit', 'id' => $model->id))?>  
<div>  
    <?php echo $model->html?>  
</div>
```

6. Создайте `protected/views/snippet/add.php`:

```
<h2><?php echo CHtml::link('Snippets',  
array('index'))?> → Add  
    snippet  
</h2>  
<?php $this->renderPartial('_form', array('model'  
=> $model))?>
```

7. Создайте файл `protected/views/snippet/edit.php`:

```
<h2><?php echo CHtml::link('Snippets',  
array('index'))?> → Edit  
    snippet  
</h2>  
<?php $this->renderPartial('_form', array('model' => $model))?>
```

8. Создайте файл `protected/views/snippet/_form.php`:

```
<?php echo CHtml::beginForm()?>  
<ul>  
    <li>  
        <?php echo CHtml::activeLabel($model,  
            'title')?>  
        <?php echo CHtml::activeTextField($model,  
            'title')?>  
    </li>  
    <li>  
        <?php echo CHtml::activeLabel($model,  
            'code')?>  
        <?php echo CHtml::activeTextArea($model,  
            'code')?>  
    </li>  
    <li>  
        <?php echo CHtml::activeLabel($model,  
            'language')?>  
        <?php echo CHtml::activeDropDownList($model,  
            'language',  
                $model->getSupportedLanguages())?>  
    </li>  
</ul>
```

```

</li>
<li>
  <?php echo CHtml::submitButton('Save')?>
</li>
</ul>
<?php echo CHtml::endForm()?>

```

9. Теперь запустите контроллер `snippet` и попробуйте создать фрагменты кода, как показано на следующем скриншоте.

The screenshot shows a web application interface titled "My Web Application". It has a navigation bar with links: Home, About, Contact, and Login. Below the navigation bar, there is a breadcrumb trail: "Snippets → Add snippet". The form contains the following fields:

- Title: A text input field containing the word "Funny".
- Code: A text area containing the PHP code snippet `if(isset($user)) $user = $user;`.
- Language: A dropdown menu with "PHP" selected.
- Save: A blue button.

10. При просмотре фрагмент должен выглядеть примерно так:

The screenshot shows the same web application interface, but now the breadcrumb trail is "Snippets → Funny". Below the breadcrumb, there is a link "Edit" and the PHP code snippet `if(isset($user)) $user = $user;` displayed in a text area.

Как это работает

Функция модели `snippet` используется для сохранения кода и заголовка фрагмента. Мы также добавили поля `html` и `language`. Первое (`html`) используется для хранения представления в HTML подсвеченного кода, а `language` предназначается для хранения языка фраг-

мента (PHP, HTML, CSS, JavaScript и т. д.). Это значение понадобится нам при редактировании.

Мы удаляем правило `safe` из модели `Snippet` и делаем поля `title`, `code` и `language` обязательными. Правила для `html` нет, поэтому значение не может быть задано непосредственно через форму.

Метод `afterValidate` в соответствии с названием вызывается после успешного завершения валидации. В этом методе код, хранящийся в `code`, трансформируется в HTML с подсветкой и сохраняется в поле `html`. При этом используется класс `CTextHighlighter` с учетом значения `language`.

Заметьте, что вам нужно определить CSS-классы `php-hl-*`, чтобы получить подсветку. CSS по умолчанию можно найти в файле `framework/vendors/TextHighlighter/highlight.css`.

Метод `getSupportedLanguages` возвращает языки, которые мы хотим поддерживать, в виде массива значение-заголовок. Этот метод используется в форме `snippet`.

И ещё

Чтобы узнать больше о подсветке кода, воспользуйтесь следующими ресурсами:

Подсветка кода в Yii

- <http://www.yiiframework.com/doc/api/CTextHighlighter>;
- http://pear.php.net/package/Text_Highlighter/;

Другие способы подсветки кода

Если `Text_highlighter` в составе Yii вам не подходит, в сети Интернет можно найти множество альтернатив. Несколько хороших примеров:

- <http://qbnz.com/highlighter/>;
- <http://softwaremaniacs.org/soft/highlight/>.

Смотрите также

- Рецепт «Обработка полей модели с помощью методов-событий Active Record» в этой главе;
- Рецепт «Применение markdown и HTML» в этой главе;
- Рецепт «Автоматический timestamp» в этой главе;
- Рецепт «Автоматическое указание автора» в этой главе;

Автоматический timestamp

Почти в каждой модели, соответствующей некому материалу сайта, следует хранить даты создания и изменения, чтобы отображать актуальности или использовать ревизии. В Yii существуют два подходящих способа автоматизировать это:

- перекрытие метода `beforeValidate`;
- использование поведения `CTimestampBehavior` из `Zii`.

Мы рассмотрим применение этих способов к заметкам в блоге. Для хранения даты и времени будем использовать UNIX timestamp.

Подготовка

- Создайте новое приложение с помощью `yiiic webapp`, как это описано в официальном руководстве по адресу <http://www.yiiframework.com/doc/guide/ru/quickstart.first-app>.
- Подключитесь к базе данных и создайте таблицу с именем `post` следующим образом:

```
DROP TABLE IF EXISTS 'post'
CREATE TABLE IF NOT EXISTS 'post' (
    'id' INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'title' VARCHAR(255) NOT NULL,
    'code' TEXT NOT NULL,
    'created_on' INT(10) UNSIGNED NOT NULL,
    'modified_on' INT(10) UNSIGNED NOT NULL,
    PRIMARY KEY ('id')
);
```

- Создайте модель `Post` при помощи `Gii`.
- Удалите все, касающееся `created_on` и `modified_on`, из метода `rules` модели.

Как это делается

1. Начнем с перекрытия метода `beforeValidate`. Откройте файл `protected/models/Post.php` и добавьте следующий метод:

```
protected function beforeValidate()
{
    if($this->getIsNewRecord())
        $this->created_on = time();
    $this->modified_on = time();
    return parent::beforeValidate();
}
```

2. Теперь добавьте следующий код к новому контроллеру и запустите его:

```
$post = new Post();  
$post->title = "test title";  
$post->text = "test text";  
$post->save();  
echo date('r', $post->created_on);
```

3. Вы должны получить дату и время. Поскольку мы просто создаем сообщение, должны получиться примерно текущие дата и время. Другой способ состоит в использовании `CTimestampBehavior`. Удалите модель `Post` и ещё раз создайте ее с помощью `Gii`. Удалите из метода `rules` модели все, касающееся `created_on` и `modified_on`. Добавьте к модели следующий метод:

```
public function behaviors()  
{  
    return array(  
        'timestamps' => array(  
            'class' => 'zii.behaviors.CTimestampBehavior',  
            'createAttribute' => 'created_on',  
            'updateAttribute' => 'modified_on',  
            'setUpdateOnCreate' => true,  
        ),  
    );  
}
```

Как это работает

Метод `beforeValidate` выполняется как раз перед началом валидации модели. В этом методе `modified_on` заполняется всегда, а `created_on` заполняется, только если модель новая, то есть лишь при создании записи.

Используя готовое поведение из `Zii`, мы указываем для `createAttribute` и `updateAttribute` выбранные нами имена полей. `setUpdateOnCreate` срабатывает, заполняя `modified_on` при добавлении записи. Остальное делается функцией поведения.

И ещё

Чтобы узнать подробно о `CTimestampBehavior`, обратитесь к описанию API по адресу: <http://www.yiiframework.com/doc/api/CTimestampBehavior/>.

Смотрите также

- Рецепт «Обработка полей модели с помощью методов-событий Active Record» в этой главе.
- Рецепт «Применение markdown и HTML» в этой главе.
- Рецепт «Подсветка кода с помощью Yii» в этой главе.
- Рецепт «Автоматическое указание автора» в этой главе.

Автоматическое указание автора

Почти в любом приложении, использующем материалы разных авторов, должен быть способ определить, кто создал сообщение и кто является его владельцем.

Из этого рецепта вы узнаете, как автоматизировать данную задачу при помощи модели. Далее подразумевается, что приложение использует `CUserIdentity` для управления авторизацией и что `Yii::app()->user->id` возвращает целочисленный ID пользователя. Мы не будем изменять имя первоначального автора сообщения, если кто-то другой отредактировал сообщение.

Подготовка

- Создайте новое приложение с помощью `yiic webapp`, как это описано в официальном руководстве по адресу <http://www.yiiframework.com/doc/guide/ru/quickstart.first-app>.
- Подключитесь к базе данных и создайте таблицу с именем `post`:

```
DROP TABLE IF EXISTS 'post'
CREATE TABLE IF NOT EXISTS 'post' (
    'id' INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'title' VARCHAR(255) NOT NULL,
    'code' TEXT NOT NULL,
    'created_on' INT(10) UNSIGNED NOT NULL,
    'modified_on' INT(10) UNSIGNED NOT NULL,
    PRIMARY KEY ('id')
);
```

- Создайте модель `Post` при помощи `Gii`.

Как это делается

1. Добавьте метод в файл `protected/models/Post.php`:

```
protected function beforeValidate()
{
    if(empty($this->author_id))
        $this->author_id = Yii::app()->user->id;
    return parent::beforeValidate();
}
```

2. Теперь проведем тест. Создайте файл `protected/controllers/TestController.php`:

```
<?php
class TestController extends CController
{
    public function actionIndex()
    {
        $post = Post::model()->find();
        if(!$post)
            $post = new Post();
        $post->title = 'test';
        $post->text = 'test';
        $post->save();
        echo $post->author_id;
    }
}
```

3. Теперь авторизуйтесь и выполните `test/index`. Вы должны получить ID авторизованного в данный момент пользователя. Авторизуйтесь от имени другого пользователя и снова выполните код. На этот раз вы должны получить тот же самый ID, как и было задумано.

Как это работает

Метод `beforeValidate` выполняется как раз перед началом валидации модели. В этом методе мы присваиваем `author_id` значение `Yii::app()->user->id`, только если ID автора не было присвоено значение. Это может быть в момент создания сообщения, но также может быть после удаления идентификатора первоначального автора (если вы правильно задали внешний ключ в опции каскадного удаления).

Смотрите также

- Рецепт «Обработка полей модели с помощью методов-событий Active Record» в этой главе.
- Рецепт «Применение markdown и HTML» в этой главе.

- Рецепт «Подсветка кода с помощью Yii» в этой главе.
- Рецепт «Автоматический timestamp» в этой главе.

Реализация наследования с одной таблицей

Реляционные базы данных не поддерживают наследования. Если нам нужно хранить наследование в базе данных, то необходим соответствующий код. Он должен быть эффективным, то есть генерировать как можно меньше команд JOIN. Общее решение этой проблемы описал Мартин Фаулер и назвал **наследованием с одной таблицей** (single table inheritance).

При использовании этого шаблона проектирования мы сохраняем все данные о структуре класса в одной таблице и используем поле `type` для определения модели каждой из строк.

В качестве примера мы реализуем наследование с одной таблицей для класса со следующей структурой:

```
Car
|- SportCar
|- FamilyCar
```

(От класса «автомобиль» наследуют классы «спортивный автомобиль» и «семейный автомобиль».)

Подготовка

- Создайте новое приложение с помощью `yii create webapp`, как это описано в официальном руководстве.
- Создайте базу данных. Добавьте таблицу и данные:

```
CREATE TABLE 'car' (
    'id' int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
    'name' varchar(255) NOT NULL,
    'type' varchar(100) NOT NULL,
    PRIMARY KEY ('id')
);

INSERT INTO 'car' ('name', 'type')
VALUES ('Ford Focus', 'family'),
('Opel Astra', 'family'),
('Kia Ceed', 'family'),
('Porsche Boxster', 'sport'),
('Ferrari 550', 'sport');
```

Как это делается

1. Сначала создадим для класса car модель protected/models/Car.php:

```
<?php
class Car extends CActiveRecord
{
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }

    public function tableName()
    {
        return 'car';
    }

    protected function instantiate($attributes)
    {
        switch($attributes['type'])
        {
            case 'sport':
                $class='SportCar';
                break;
            case 'family':
                $class='FamilyCar';
                break;
            default:
                $class=get_class($this);
        }
        $model=new $class(null);
        return $model;
    }
}
```

2. Теперь реализуем protected/models/SportCar.php:

```
<?php
class SportCar extends Car
{
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }
    public function defaultScope()
    {
        return array(
            'condition'=>"type='sport'",
        );
    }
}
```

```

    );
}
}

```

3. Также реализуем `protected/models/FamilyCar.php`:

```

<?php
class FamilyCar extends Car
{
    public static function model($className=__CLASS__)
    {
        return parent::model($className);
    }
    public function defaultScope()
    {
        return array(
            'condition'=>"type='family'",
        );
    }
}

```

4. Теперь создадим `protected/controllers/TestController.php`:

```

<?php
class TestController extends CController
{
    public function actionIndex()
    {
        echo "<h1>All cars</h1>";
        $cars = Car::model()->findAll();
        foreach($cars as $car)
        {
            // Каждый автомобиль может быть либо класса Car,
            // либо класса SportCar, либо класса FamilyCar
            echo get_class($car).' '.$car->name."<br />";
        }
        echo "<h1>Sport cars only</h1>";
        $sportCars = SportCar::model()->findAll();
        foreach($sportCars as $car)
        {
            // Каждый автомобиль должен быть класса SportCar
            echo get_class($car).' '.$car->name."<br />";
        }
    }
}

```

5. Выполните `test/index`. Вы должны получить следующий результат:

All cars

FamilyCar Ford Focus
FamilyCar Opel Astra
FamilyCar Kia Ceed
SportCar Porsche Boxster
SportCar Ferrari 550

Sport cars only

SportCar Porsche Boxster
SportCar Ferrari 550

Как это работает

Базовая модель `Car` — это обычная модель Active Record, в которую добавлены два метода. Метод `tableName` явным образом задает имя таблицы, которая будет использоваться моделью. Для отдельно взятой модели `Car` это не имеет смысла, но для дочерних моделей метод возвращает ту же самую таблицу `car`, а именно это нам и нужно — одна таблица для всего дерева классов. Метод `instantiate` используется внутри Active Record для создания модели из необработанных данных в процессе вызова таких методов, как `Car::model()->findAll()`. Мы используем конструкцию `switch` для создания различных классов в зависимости от значения атрибута `type`. Если значение атрибута не указано или указывает несуществующий класс, используем базовый класс.

Модели `SportCar` и `FamilyCar` задают группу условий AR по умолчанию. Когда мы ищем модель с помощью метода `SportCar::model()->`, мы получим только модель `SportCar`.

И ещё

Воспользуйтесь следующими ресурсами, чтобы узнать подробнее о шаблоне наследования с одной таблицей и реализации Active Record в Yii:

- <http://martinfowler.com/eaCatalog/singleTableInheritance.html>;
- <http://www.yiiframework.com/doc/api/CActiveRecord/>.

Смотрите также

- Рецепт «Использование именованных групп условий для создания многоязычных моделей» в этой главе.

Использование CDbCriteria

При использовании методов Active Record, например `findAll` или `find`, можно передавать критерий в качестве параметра. Это может быть массив или экземпляр класса `CDbCriteria`. Этот класс хранит критерии запроса, такие как условия или порядок.

Как это делается

Обычно класс критериев используется, как в следующем примере:

```
$criteria = new CDbCriteria();
$criteria->limit = 10;
$criteria->order= 'id DESC';
$criteria->with = array('comments');
$criteria->compare('approved', 1);
$criteria->addInCondition('id', array(4, 8, 15, 16, 23, 42));
$posts = Post::model()->findAll($criteria);
```

Как это работает

Класс критериев не составляет запросы сам, а только хранит данные и позволяет нам изменять их. Вся работа выполняется внутри методов AR, где используется критерий.

Приведенный выше код можно прочитать следующим образом:

Выбрать 10 записей вместе с комментариями из числа одобренных записей, номера которых равны 4, 8, 15, 16, 23 или 42. Отсортировать по номеру сообщения в порядке убывания.

Или

```
SELECT *
FROM post p
JOIN comment c ON p.id = c.post_id
WHERE p.approved = 1
AND p.id IN (4, 8, 15, 16, 23, 42)
ORDER BY p.id DESC
LIMIT 10
```

И ещё

За подробностями обратитесь по адресам:

- <http://www.yiiframework.com/doc/api/CDbCriteria/>;
- <http://www.yiiframework.com/doc/api/CPagination/>;
- <http://www.yiiframework.com/doc/api/CSort/>.

Смотрите также

- Рецепт «Получение данных из базы данных» в этой главе.



ГЛАВА 7.

Использование компонентов Zii

В этой главе мы рассмотрим:

- Использование источников данных.
- Использование гридов.
- Использование списков.
- Создание своих столбцов грида.

Введение

В состав Yii входит полезная библиотека Zii. Она поставляется вместе с самим фреймворком и включает классы, облегчающие жизнь разработчиков. Ее наиболее интересные компоненты – гриды (*grids*) и списки, которые позволяют работать с данными как в администраторской, так и в пользовательской частях проекта очень быстро и эффективно. В этой главе вы узнаете, как использовать и настраивать эти компоненты. Также вы узнаете об источниках данных. Они являются частью ядра, а не библиотеки Zii, но поскольку они широко используются с гридами и списками, то мы рассмотрим их здесь.

В данной главе мы будем использовать базу данных Sakila версии 0.8. Скачать ее можно с официального сайта MySQL: <http://dev.mysql.com/doc/sakila/en/index.html>.

Использование источников данных

Источники данных (*data providers*) используются для инкапсулирования частых операций, производимых над моделями данных, таких как сортировка, разбивка на страницы или выполнение запросов. Ис-

точники данных нередко используются с гридами и списками. Поскольку как виджеты, так и источники стандартизованы, то можно выводить одни и те же данные с помощью разных виджетов. Также можно получать данные для одного виджета из разных источников. Переключение источников и виджетов организовано относительно прозрачно.

В настоящий момент для получения данных из моделей Active Record, массивов и SQL-запросов реализованы `CActiveDataProvider`, `CArrayDataProvider` и `CSqlDataProvider` соответственно.

Попробуем использовать их для заполнения грида данными.

Подготовка

- Создайте новое приложение с помощью `yii webapp`, как это описано в официальном руководстве.
- Загрузите базу данных Sakila с адреса <http://dev.mysql.com/doc/sakila/en/index.html> и выполните полученные скрипты SQL, сначала для схемы, затем для данных.
- Настройте подключение к базе данных в файле `protected/config/main.php`.
- С помощью Gii создайте модель для таблицы `film`.

Как это делается

1. Начнем с представления для контроллера грида. Создайте файл `protected/views/grid/index.php`:

```
<?php $this->widget('zii.widgets.grid.CGridView',
    array('dataProvider' => $dataProvider,
    ))?>
```

2. Затем создайте файл `protected/controllers/GridController.php`:

```
<?php
class GridController extends Controller
{
    public function actionAR()
    {
        $dataProvider = new CActiveDataProvider('Film', array(
            'pagination'=>array(
                'pageSize'=>10,
            ),
            'sort'=>array(
                'defaultOrder'=> array('title'=>false),
            )
        ))
    }
}
```

```
));  
  
$this->render('index', array(  
    'dataProvider' => $dataProvider,  
));  
}  
  
public function actionArray()  
{  
    $yiiDevelopers = array(  
        array(  
            'name'=>'Qiang Xue',  
            'id'=>'2',  
            'forumName'=>'qiang',  
            'memberSince'=>'Jan 2008',  
            'location'=>'Washington DC, USA',  
            'duty'=>'founder and project lead',  
            'active'=>true,  
        ),  
        array(  
            'name'=>'Wei Zhuo',  
            'id'=>'3',  
            'forumName'=>'wei',  
            'memberSince'=>'Jan 2008',  
            'location'=>'Sydney, Australia',  
            'duty'=>'project site maintenance and development',  
            'active'=>true,  
        ),  
        array(  
            'name'=>'Sebastián Thierer',  
            'id'=>'54',  
            'forumName'=>'sebas',  
            'memberSince'=>'Sep 2009',  
            'location'=>'Argentina',  
            'duty'=>'component development',  
            'active'=>true,  
        ),  
        array(  
            'name'=>'Alexander Makarov',  
            'id'=>'415',  
            'forumName'=>'samdark',  
            'memberSince'=>'Mar 2010',  
            'location'=>'Russia',  
            'duty'=>'core framework development',  
            'active'=>true,  
        ),  
        array(  
            'name'=>'Maurizio Domba',  
            'id'=>'2650',  
            'forumName'=>'mdomba',  
            'memberSince'=>'Aug 2010',  
            'location'=>'Croatia',  
            'duty'=>'core framework development',  
        ),  
    );  
}
```

```
        'active'=>true,
    ),
    array(
        'name'=>'Y!!',
        'id'=>'1644',
        'forumName'=>'Y!!',
        'memberSince'=>'Aug 2010',
        'location'=>'Germany',
        'duty'=>'core framework development',
        'active'=>true,
    ),
    array(
        'name'=>'Jeffrey Winesett',
        'id'=>'15',
        'forumName'=>'jefftulsa',
        'memberSince'=>'Sep 2010',
        'location'=>'Austin, TX, USA',
        'duty'=>'documentation and marketing',
        'active'=>true,
    ),
    array(
        'name'=>'Jonah Turnquist',
        'id'=>'127',
        'forumName'=>'jonah',
        'memberSince'=>'Sep 2009 - Aug 2010',
        'location'=>'California, US',
        'duty'=>'component development',
        'active'=>false,
    ),
    array(
        'name'=>'István Beregszászi',
        'id'=>'1286',
        'forumName'=>'pesta',
        'memberSince'=>'Sep 2009 - Mar 2010',
        'location'=>'Hungary',
        'duty'=>'core framework development',
        'active'=>false,
    ),
);
$dataProvider = new CArrayDataProvider(
    Yii::dev, array(
        'sort'=>array(
            'attributes'=>array('name', 'id', 'active'),
            'defaultOrder'=>array('active' => true,
                                'name' => false),
        ),
        'pagination'=>array(
            'pageSize'=>10,
        ),
    ),
);

$this->render('index', array(
    'dataProvider' => $dataProvider,
```

```

    ));
}

public function actionSQL()
{
    $count=Yii::app()->db->createCommand('SELECT COUNT(*)
    FROM film')->queryScalar();
    $sql='SELECT * FROM film';
    $dataProvider=new CSqlDataProvider($sql, array(
        'keyField'=>'film_id',
        'totalCount'=>$count,
        'sort'=>array(
            'attributes'=>array('title'),
            'defaultOrder'=>array('title' => false),
        ),
        'pagination'=>array(
            'pageSize'=>10,
        ),
    ));

    $this->render('index', array(
        'dataProvider' => $dataProvider,
    ));
}
}

```

3. Теперь запустите действия `grid/array` и `grid/sql` и попробуйте использовать гриды.

Advanced Search Displaying 1-10 of 599 result(s)

Customer	Store	First Name	Last Name	Email	Address	
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	 
2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	 
3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	 
4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	 
5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	 
6	2	JENNIFER	DAVIS	JENNIFER.DAVIS@sakilacustomer.org	10	 
7	1	MARIA	MILLER	MARIA.MILLER@sakilacustomer.org	11	 
8	2	SUSAN	WILSON	SUSAN.WILSON@sakilacustomer.org	12	 
9	2	MARGARET	MOORE	MARGARET.MOORE@sakilacustomer.org	13	 
10	1	DOROTHY	TAYLOR	DOROTHY.TAYLOR@sakilacustomer.org	14	 

Go to page: < Previous **1** 2 3 4 5 6 7 8 9 10 Next >

Как это работает

Представление очень простое и одинаково для любых источников данных. Мы вызываем виджет грида и передаем ему экземпляр источника данных.

Посмотрим все действия по очереди, начиная с `actionAR`:

```
$dataProvider = new CActiveDataProvider('Film', array(
    'pagination'=>array(
        'pageSize'=>10,
    ),
    'sort'=>array(
        'defaultOrder'=>array('title'=>false),
    )
));
```

`CActiveDataProvider` работает с моделями `Active Record`. Класс модели передается первым аргументом конструктора класса. Вторым аргументом является массив, определяющий `public` свойства класса. В вышеприведенном коде для постраничной разбивки мы задали размер страницы в 10 элементов и назначили начальной сортировку по наименованию.

Заметьте, что вместо строки мы используем массив, в котором ключи – имена столбцов, а значения – `true` или `false`. Значение `true` означает, что сортировка делается по убыванию (`CSort::SORT_DESC`), а значение `false` – по возрастанию (`CSort::SORT_ASC`). Определение порядка сортировки таким способом позволяет Yii отображать в заголовке столбца треугольник, показывающий направление сортировки.

В функции `actionArray` мы используем `CArrayDataProvider`, который может принимать на входе массив.

```
$dataProvider = new CArrayDataProvider($yiiDevelopers, array(
    'sort'=>array(
        'attributes'=>array('name', 'id', 'active'),
        'defaultOrder'=>array('active' => true, 'name' => false),
    ),
    'pagination'=>array(
        'pageSize'=>10,
    ),
));
```

Первый аргумент принимает ассоциативный массив, в котором ключами являются имена столбцов, а значениями – соответствующие ключам значения. Второй аргумент принимает массив с такими же опциями, как и в случае с `CActiveDataProvider`.

В функции `actionSQL` мы используем `CSqlDataProvider`, который принимает SQL-запрос и модифицирует его в соответствии с заданными настройками постраничной разбивки. Первый аргумент принимает строку с SQL-запросом, а второй – с параметрами источника данных. На этот раз нам нужно передать `calculateTotalItemCount`

количество записей вручную. Для этого мы выполняем дополнительный SQL-запрос. Также необходимо задать `keyField`, так как первичный ключ этой таблицы не `id`, а `film_id`.

Резюмируя, для каждого источника данных задаются следующие свойства:

<code>Pagination</code>	Объект <code>CPagination</code> или массив начальных значений для нового экземпляра <code>CPagination</code>
<code>Sort</code>	Объект <code>CSort</code> или массив начальных значений для нового экземпляра <code>CSort</code>
<code>totalItemCount</code>	Необходимо задавать, только если для источника, такого как <code>CSqlDataProvider</code> , не реализован метод <code>calculateTotalItemCount</code> .

И ещё

Можно использовать источники данных без каких-либо специальных виджетов. Замените содержимое файла `protected/views/grid/index.php` следующим:

```
<?php foreach($dataProvider->data as $film):?>
    <?php echo $film->title?>
<?php endforeach?>

<?php $this->widget('CLinkPager',array(
    'pages'=>$dataProvider->pagination)) ?>
```

Дополнительные материалы

Чтобы узнать больше об источниках данных, обратитесь к описаниям API по следующим адресам:

- <http://www.yiiframework.com/doc/api/CDataProvider>;
- <http://www.yiiframework.com/doc/api/CActiveDataProvider>;
- <http://www.yiiframework.com/doc/api/CArrayDataProvider>;
- <http://www.yiiframework.com/doc/api/CSqlDataProvider>;
- <http://www.yiiframework.com/doc/api/CSort>;
- <http://www.yiiframework.com/doc/api/CPagination>.

Смотрите также

- Рецепт «Использование гридов» в этой главе.
- Рецепт «Использование списков» в этой главе.

Использование гридов

Гриды (*grids*) Yii очень полезны для быстрого создания административной части приложений и любых страниц для управления данными.

В данном рецепте мы используем Gii для генерации грида, посмотрим, как он работает и как его настраивать.

Подготовка

Выполните следующие шаги:

- создайте новое приложение с помощью `yii webapp`, как это описано в официальном руководстве;
- загрузите базу данных Sakila с адреса <http://dev.mysql.com/doc/sakila/en/index.html> и выполните полученные скрипты SQL сначала для схемы, затем для данных;
- настройте подключение к базе данных в файле `protected/config/main.php`;
- с помощью Gii создайте модель для таблиц `customer`, `address` и `city`.

Как это делается

- Откройте Gii, выберите **Crud Generator** и введите `Customer` в поле **Model Class**. Нажмите кнопку **Preview**, затем **Generate**.
- Gii создаст контроллер в файле `protected/controllers/CustomerController.php` и группу представлений в директории `protected/views/customer`.
- Запустите контроллер `customer` и перейдите по ссылке **Manage Customer**. После авторизации вы должны увидеть созданный грид:

Advanced Search

Displaying 1-10 of 599 result(s)

Customer	Store	First Name	Last Name	Email	Address	
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	 
2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	 
3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	 
4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	 
5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	 
6	2	JENNIFER	DAVIS	JENNIFER.DAVIS@sakilacustomer.org	10	 
7	1	MARIA	MILLER	MARIA.MILLER@sakilacustomer.org	11	 
8	2	SUSAN	WILSON	SUSAN.WILSON@sakilacustomer.org	12	 
9	2	MARGARET	MOORE	MARGARET.MOORE@sakilacustomer.org	13	 
10	1	DOROTHY	TAYLOR	DOROTHY.TAYLOR@sakilacustomer.org	14	 

Go to page: < Previous 1 2 3 4 5 6 7 8 9 10 Next >

Как это работает

Начнём с действия `admin` для контроллера `customer`:

```
public function actionAdmin()
{
    $model=new Customer('search');
    $model->unsetAttributes(); // clear any default values
    if(isset($_GET['Customer']))
        $model->attributes=$_GET['Customer'];

    $this->render('admin',array(
        'model'=>$model,
    ));
}
```

Модель `customer` создаётся с заданным сценарием `search`, все значения атрибутов очищаются и заполняются данными из `$_GET`. После первого запроса `$_GET` не содержит данных, но когда вы изменяете страницу или устанавливаете фильтр по первому имени атрибута в поле ввода под именем столбца, то следующие параметры `$_GET` передаются тому же действию через AJAX-запрос:

```
Customer[address_id] =
Customer[customer_id] =
Customer[email] =
Customer[first_name] = alex
Customer[last_name] =
Customer[store_id] =
Customer_page = 2
ajax = customer-grid
```

Поскольку используется сценарий `search`, то применяются соответствующие правила валидации из `Customer::rules`. Для сценария `search` Yii генерирует правило `safe`, которое позволяет массово присваивать значения всем полям:

```
array( 'customer_id', 'store_id', 'first_name', 'last_name',
    'email', 'address_id', 'active', 'create_date', 'last_update',
    'safe', 'on'=>'search' ),
```

Затем модель передается представлению `protected/views/customer/admin.php`. Оно описывает улучшенную форму поиска и передаёт модель виджету грида:

```
<?php $this->widget('zii.widgets.grid.CGridView', array(
    'id'=>'customer-grid',
    'dataProvider'=>$model->search(),
    'filter'=>$model,
```

```
'columns'=>array(
    'customer_id',
    'store_id',
    'first_name',
    'last_name',
    'email',
    'address_id',
    /*
    'active',
    'create_date',
    'last_update',
    */
    array(
        'class'=>'CButtonColumn',
    ),
),
)); ?>
```

Столбцы, используемые в гриде, задаются через `columns`. Когда указывается только имя, то используется соответствующее поле из источника данных.

Также мы можем использовать поле, представленное указанным классом. В нашем случае мы используем `CButtonColumn` для создания кнопок просмотра, обновления и удаления, которые привязаны к одноимённым действиям и передают им ID записи, с которой необходимо работать.

Свойству `filter` присваивается заполненная данными модель. Если оно задано, то грид будет отображать несколько текстовых полей-фильтров непосредственно над данными.

Свойство `dataProvider` принимает экземпляр источника данных. В нашем случае его возвращает метод `search` модели:

```
public function search()
{
    // Warning: Please modify the following code to remove
    // attributes that should not be searched.

    $criteria=new CDbCriteria;

    $criteria->compare('customer_id',$this->customer_id);
    $criteria->compare('store_id',$this->store_id);
    $criteria->compare('first_name',$this->first_name,true);
    $criteria->compare('last_name',$this->last_name,true);
    $criteria->compare('email',$this->email,true);
    $criteria->compare('address_id',$this->address_id);
    $criteria->compare('active',$this->active);
    $criteria->compare('create_date',$this->create_date,true);
```

```
$criteria->compare('last_update',$this->last_update,true);  
  
return new CActiveDataProvider(get_class($this), array(  
    'criteria'=>$criteria,  
));  
}
```

Этот метод вызывается после заполнения модели данными из массива `$_GET`, полученного от полей фильтров. Мы можем использовать значения полей, чтобы сформировать критерии для источника данных. При этом для числовых значений используется точное, а для строковых – частичное сравнение.

И ещё

Сгенерированный Gii код может пригодиться во многих простых случаях, но его часто приходится подстраивать под свои нужды.

Использование данных из связанных моделей Active Record

В нашем коде сгенерированный грид вместо названия склада и его адреса отображает ID. Исправим данный недостаток.

1. У нас есть следующие связи в модели `Customer`:

```
public function relations()  
{  
    // NOTE: you may need to adjust the relation name  
    // and the related  
    // class name for the relations automatically  
    // generated below.  
    return array(  
        'address' => array(self::BELONGS_TO, 'Address',  
            'address_id'),  
        'store' => array(self::BELONGS_TO, 'Store',  
            'store_id'),  
        'payments' => array(self::HAS_MANY, 'Payment',  
            'customer_id'),  
        'rentals' => array(self::HAS_MANY, 'Rental',  
            'customer_id'),  
    );  
}
```

2. Нам нужно вместе с моделью загрузить данные связи `address`. Это означает, что мы должны добавить их в часть `with` критерия, передаваемого источнику данных в методе `Customer::search`. Поскольку в модели адреса есть ID города, можно загрузить данные города через отношение `city` модели `Address`.

```
public function search()
{
    // Warning: Please modify the following code to
    // remove attributes that should not be searched.

    $criteria=new CDbCriteria;

    $criteria->with = array('address' => array(
        'with' => 'city'
    )); ...
}
```

Каждое отношение, использованное в части `with` критериев, может быть указано вышеописанным способом. При этом ключом является имя отношения, а значением – массив, представляющий критерий, который применяется к связанной модели.

3. Теперь изменим список столбцов, передаваемый гриду, в файле `protected/views/customer/admin.php`:

```
'columns'=>array(
    'customer_id',
    'store_id',
    'first_name',
    'last_name',
    'email',
    array(
        'name'=>'address',
        'value'=>'$data->address->address.',
            '.$data->address->city->city.',
            '.$data->address->district',
        ),
    ),
),
```

4. В вышеприведённом коде мы использовали имя отношения `name` и присвоили переменной `value` строку, состоящую из адреса, города и района.

Теперь проверьте грид. Он должен содержать адрес, город и район в поле адреса. (См. рисунок на стр. 200.)

5. У нас всё ещё нет рабочей сортировки и фильтрации по адресу.

Сначала исправим сортировку. В методе `Customer::search` нам необходимо создать экземпляр класса `CSort`, настроить его и передать источнику данных:

```
$sort = new CSort;
$sort->attributes = array(
    'address' => array(
        'asc' => 'address, city, district',
```

Displaying 1-10 of 599 results()

Customer	Store	First Name	Last Name	Email	Address	
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	1913 Manor Way, Sasebo, Nagasaki	 
2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	1121 Loja Avenue, San Bernardino, California	 
3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	692 Joliet Street, Athens, Alaska	 
4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	1555 Inverl Manor, Memprian, Mandalay	 
5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	53 Idlu Parkway, Nantou, Nanhai	 
6	2	JENNIFER	DAVIS	JENNIFER.DAVIS@sakilacustomer.org	1795 Santiago de Compostela Way, Laredo, Texas	 
7	1	MARIA	MILLER	MARIA.MILLER@sakilacustomer.org	900 Santiago de Compostela Parkway, Kragujevac, Central Serbia	 
8	2	SUSAN	WILSON	SUSAN.WILSON@sakilacustomer.org	478 Joliet Way, Hamilton, Hamilton	 
9	2	MARGARET	MOORE	MARGARET.MOORE@sakilacustomer.org	613 Kinsley Drive, Masqat, Masqat	 
10	1	DOROTHY	TAYLOR	DOROTHY.TAYLOR@sakilacustomer.org	1531 Sai Drive, Estaban, Estaban	 

Go to page: < Previous 1 2 3 4 5 6 7 8 9 10 Next >

```
'desc' => 'address DESC, city DESC, district DESC',
),
'',
);
return new CActiveDataProvider(get_class($this), array(
    'criteria'=>$criteria,
    'sort'=>$sort,
));
```

Метод `Csort::attributes` принимает список атрибутов, по которым возможна сортировка. Мы хотим, чтобы все атрибуты `Customer` можно было сортировать, поэтому добавляем к списку `''`. В дополнение мы задаем SQL-запрос для сортировки атрибута `address` по возрастанию и убыванию.

Вот и всё. Теперь сортировка должна работать.

- Исправим фильтрацию. Сначала необходимо добавить `address` к списку безопасных атрибутов в методе `rules` модели. Далее заменим сравнение в методе `Customer::search`:

```
$criteria->compare('address_id', $this->address_id);
```

должно быть заменено на:

```
$criteria->compare('address', $this->address, true);
$criteria->compare('district', $this->address, true, "OR");
$criteria->compare('city', $this->address, true, "OR");
```

7. Когда пользователь введет в поле фильтрации по адресу `california`, то приведенные выше три сравнения в результате дадут следующее:

```
WHERE address LIKE '%california%'
OR district LIKE '%california%'
OR city LIKE '%california%'
```

Displaying 1-10 of 11 result(s)

Customer	Store	First Name	Last Name	Email	Address	
					california	
2	1	PATRICK	JOHNSON	PATRICK.JOHNSON@sakilacustomer.org	1121 Loja Avenue, San Bernardino, California	  
51	1	ALICE	STEWART	ALICE.STEWART@sakilacustomer.org	1135 Isumiano Parkway, Fontana, California	  
498	2	SHANE	MILLARD	SHANE.MILLARD@sakilacustomer.org	154 Mandakurong Street, La Paz, Baja California Sur	  
426	1	JACOB	LANCE	JACOB.LANCE@sakilacustomer.org	1865 al-Gaff Avenue, El Monte, California	  
593	2	RENE	MCLUSTER	RENE.MCLUSTER@sakilacustomer.org	1895 Zhencaguan Drive, Garden Grove, California	  
214	1	KRISTIN	JOHNSTON	KRISTIN.JOHNSTON@sakilacustomer.org	226 Great Ilanor, Sunnyvale, California	  
182	1	RENEE	LANE	RENEE.LANE@sakilacustomer.org	533 al-Ayn Boulevard, Compton, California	  
14	2	BETTY	WHITE	BETTY.WHITE@sakilacustomer.org	770 Urdos122 Avenue, Citrus Heights, California	  
343	1	DOUGLAS	GRAF	DOUGLAS.GRAF@sakilacustomer.org	795 Vaduz Street, Mercah, Baja California	  
112	2	ROSA	REYNOLDS	ROSA.REYNOLDS@sakilacustomer.org	793 Cam Ranth Avenue, Lancaster, California	  

Go to page: < Previous 1 2 Next >

Дополнительные материалы

Чтобы больше узнать о гридах и их свойствах, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/CGridView>;
- <http://www.yiiframework.com/doc/api/CDbCriteria>;
- <http://www.yiiframework.com/doc/api/CSort>.

Смотрите также

- Рецепт «Использование источников данных» в этой главе.
- Рецепт «Использование списков» в этой главе.
- Рецепт «Создание своих столбцов грида» в этой главе.

Использование списков

Списки Yii – прекрасное средство для вывода данных из любого источника данных конечному пользователю, их постраничной разбивки и автоматической сортировки. CListView можно настроить для создания страницы со списком любого вида.

Подготовка

- Создайте новое приложение с помощью `yii webapp`, как это описано в официальном руководстве.
- Загрузите базу данных Sakila с адреса <http://dev.mysql.com/doc/sakila/en/index.html> и выполните полученные скрипты SQL, сначала для схемы, затем для данных.
- Настройте подключение к базе данных в файле `protected/config/main.php`.
- С помощью Gii создайте модель для таблиц `customer`, `store`, `address` и `city`.

Как это делается

1. Откройте Gii, выберите **Crud Generator** и введите **Customer** в поле **Model Class**. Нажмите кнопку **Preview**, затем **Generate**.
2. Gii создаст контроллер в файле `protected/controllers/CustomerController.php` и группу представлений в директории `protected/views/customer`.
3. Выполните действие `index` из контроллера `customer`, чтобы увидеть список клиентов.

Displaying 1-10 of 599 result(s).	
Customer: 1	
Store: 1	
First Name: MARY	
Last Name: SMITH	
Email: MARY.SMITH@sakilacustomer.org	
Address: 5	
Active: 1	
Customer: 2	
Store: 1	
First Name: PATRICIA	

Как это работает

Начнём с действия `index` контроллера `Customer`:

```
public function actionIndex()
{
    $dataProvider=new CActiveDataProvider('Customer');
    $this->render('index',array(
        'dataProvider'=>$dataProvider,
    ));
}
```

Оно очень простое: создается новый источник данных для модели Customer и передается в представление `protected/views/customer/index.php`, где он используется в виджете CListView:

```
<?php $this->widget('zii.widgets.CListView', array(
    'dataProvider'=>$dataProvider,
    'itemView'=>'_view',
)); ?>
```

`itemView` задает шаблон, который используется для отображения каждой из строк. В нашем случае это `protected/views/customer/_view.php`, в котором выводятся все имена и значения атрибутов:

```
<b><?php echo CHtml::encode($data->getAttributeLabel(
    'first_name')); ?></b>
<?php echo CHtml::encode($data->first_name); ?>
```

В вышеприведенном коде `$data` соответствует модели, представляющей строку из источника данных.

И ещё

Используя списки в своих приложениях, вы, скорее всего, захотите изменить их вид.

Добавление сортировки

Добавим возможность сортировки по фамилии и адресу электронной почты. Для этого нужно задать свойство `sortableAttributes`.

```
<?php $this->widget('zii.widgets.CListView', array(
    'dataProvider'=>$dataProvider,
    'itemView'=>'_view',
    'sortableAttributes'=>array(
        'last_name',
        'email',
    ),
)); ?>
```

Настройка шаблонов

Добавим постраничную разбивку и сортировку вверху и вни-

зу списка, удалим суммарные значения. Для этого нужно изменить только одно свойство виджета, которое называется `template`:

```
<?php $this->widget('zii.widgets.CListView', array(
    'dataProvider'=>$dataProvider,
    'itemView'=>'_view',
    'sortableAttributes'=>array(
        'last_name',
        'email',
    ),
    'template' => '{sorter} {pager} {items} {sorter} {pager}',
)); ?>
```

Go to page: < Previous 1 2 3 4 5 6 7 8 9 10 Next >	
Sort by: LastName Email	
Customer: 1	
Store: 1	
First Name: MARY	
Last Name: SMITH	
Email: MARY.SMITH@sakilacustomer.org	
Address: 5	
Active: 1	
Customer: 2	
Store: 1	
First Name: PATRICIA	
Last Name: JOHNSON	
Email: PATRICIA.JOHNSON@sakilacustomer.org	

Настройка разметки и данных для вывода

Настроим разметку и шаблоны списков, начнем с параметров виджета:

```
<?php $this->widget('zii.widgets.CListView', array(
    'dataProvider'=>$dataProvider,
    'itemView'=>'_view',
    'itemsTagName' => 'ol',
    'itemsCssClass' => 'customers',
    'sortableAttributes'=>array(
        'last_name',
        'email',
    ),
    'template' => '{sorter} {pager} {items} {sorter} {pager}',
)); ?>
```

Элемент `itemsTagName` определяет тег, который будет обрамлять список. В нашем случае список будет упорядоченным, поэтому мы используем значение `ol`. Элемент `itemsCssClass` задает HTML-класс для `ol`.

Отредактируем `protected/views/customer/_view.php`:

```
<li>
  <h2>
    <?php
      $title = CHtml::encode($data->first_name.' '.$data->last_name);
      echo CHtml::link($title, array('view', 'id'=>
        $data->customer_id));
    ?>
  </h2>

  <ul>
    <li>
      <strong><?php echo CHtml::encode($data->
        getAttributeLabel('store_id')); ?></strong>
      <?php echo CHtml::encode($data->store->
        address->address.', '.$data->store->address->city->city.',
        '.$data->store->address->district); ?>
    </li>
    <li>
      <strong><?php echo CHtml::encode($data->
        getAttributeLabel('email')); ?></strong>
      <?php echo CHtml::encode($data->email); ?>
    </li>
    <li>
      <strong><?php echo CHtml::encode($data->
        getAttributeLabel('address_id')); ?></strong>
      <?php echo CHtml::encode($data->address->address.',
        '.$data->address->city->city.',
        '.$data->address->district); ?>
    </li>
    <li>
      <strong><?php echo CHtml::encode($data->
        getAttributeLabel('active')); ?></strong>
      <?php echo $data->active ? 'Yes' : 'No'; ?>
    </li>
  </ul>
</li>
```

Добавим немного CSS. Создайте файл `protected/assets/customers.css`:

```
ol.customers {
  list-style: none;
  margin: 1em 0;
}

ol.customers>li {
  margin: 1em;
  padding: 1em;
  background: #fcfcfc;
```

```
border: 1px solid #9aafe5;
}
```

Потом добавьте следующий фрагмент к файлу `protected/views/customer/_view.php`:

```
<?php Yii::app()->clientScript->registerCssFile(
    Yii::app()->assetManager->publish(
        Yii::getPathOfAlias('application.assets').
        '/customers.css'))?>
```

После применения стилей список будет выглядеть примерно так:

Go to page: < Previous 1 2 3 4 5 6 7 8 9 10 Next >	
Sort by: LastName Email	
MARY SMITH • Store: 47 MySakila Drive, Lethbridge, Alberta • Email: MARY.SMITH@sakilacustomer.org • Address: 1913 Hanoi Way, Sasebo, Nagasaki • Active: Yes	
PATRICIA JOHNSON • Store: 47 MySakila Drive, Lethbridge, Alberta • Email: PATRICIA_JOHNSON@sakilacustomer.org	

Дополнительные материалы

Чтобы больше узнать о списках, изучите API на веб-странице по следующему адресу:

- <http://www.yiiframework.com/doc/api/CListView/>.

Смотрите также

- Рецепт «Использование источников данных» в этой главе.

Создание своих столбцов грида

В основном вам не понадобится создавать собственные типы столбцов грида, поскольку те, что входят в состав Yii, достаточно гибки и подходят для большинства применений. И все же бывают ситуации, в которых вам понадобится создать свой столбец.

Создадим класс столбца грида, который позволит переключать значение Y/N. При этом будет изменяться соответствующее значение в модели посредством AJAX-запросов.

Подготовка

- Создайте новое приложение с помощью `yii webapp`, как это описано в официальном руководстве.
- Загрузите базу данных Sakila с адреса <http://dev.mysql.com/doc/sakila/en/index.html> и выполните полученные скрипты SQL сначала для схемы, затем для данных.
- Настройте подключение к базе данных в файле `protected/config/main.php`.
- С помощью Gii создайте модель для таблицы `customer`.
- Откройте Gii, выберите «**Crud Generator**» и введите в поле «**Model Class**» значение `Customer`. Нажмите «**Preview**», затем «**Generate**».
- Gii создаст контроллер в файле `protected/controllers/CustomerController.php` и группу представлений в директории `protected/views/customer/`.
- Запустите контроллер `customer` и перейдите по ссылке «**Manage Customer**». После авторизации вы должны увидеть созданный грид.

Как это делается

В нашей таблице есть поле `active`, которое мы хотим отображать как столбец с переключающими его флагами. Столбец должен содержать значения `Y` и `N` в зависимости от значения в поле. Это значение должно изменяться при щелчке мышью по флагу в столбце. Грид должен при этом оставаться на той же странице.

1. Создадим файл `protected/components/FlagColumn.php`:

```
<?php
class FlagColumn extends CGridColumn
{
    public $name;
    public $sortable=true;
    public $callbackUrl = array('flag');
    private $_flagClass = "flag_link";

    public function init() {
        parent::init();
        $cs=Yii::app()->getClientScript();
        $gridId = $this->grid->getId();
        $script = <<<SCRIPT
            jQuery("({$this->_flagClass}").live("click",
function(e){
```

```
e.preventDefault();
var link = this;
$.ajax({
    dataType: "json",
    cache: false,
    url: link.href,
    success: function(data){
        $('#gridId').yiiGridView.update('gridId');
    }
});
});
SCRIPT;
$cs->registerScript(__CLASS__.$gridId.'#flag_link',
    $script);
}

protected function renderDataCellContent($row, $data) {
    $value=Chtml::value($data,$this->name);

    $this->callbackUrl['pk'] = $data->primaryKey;
    $this->callbackUrl['name'] = urlencode($this->name);
    $this->callbackUrl['value'] = (int)empty($value);

    $link = Chtml::normalizeUrl($this->callbackUrl);

    echo Chtml::link(!empty($value) ? 'Y' : 'N', $link, array(
        'class' => $this->_flagClass,
    ));
}

protected function renderHeaderCellContent()
{
    if($this->grid->enableSorting && $this->sortable &&
        $this->name!==null)
        echo $this->grid->dataProvider->getSort()->link(
            $this->name,$this->header);
    else if($this->name!==null && $this->header===null)
    {
        if($this->grid->dataProvider instanceof
            CActiveDataProvider)
            echo Chtml::encode($this->grid->dataProvider->
                model->getAttributeLabel($this->name));
        else
            echo Chtml::encode($this->name);
    }
    else
        parent::renderHeaderCellContent();
}
}
```

2. Теперь в `protected/controllers/CustomerController.php` реализуем метод `actionFlag`:

```
public function actionFlag($pk, $name, $value){
    $model = $this->loadModel($pk);
    $model->{$name} = $value;
    $model->save(false);

    if(!Yii::app()->request->isAjaxRequest){
        $this->redirect('admin');
    }
}
```

3. Наконец, используем реализованное в гриде, `protected/views/customer/admin.php`:

```
...
<?php $this->widget('zii.widgets.grid.CGridView', array(
    'id'=>'customer-grid',
    'dataProvider'=>$model->search(),
    'filter'=>$model,
    'columns'=>array(
        'customer_id',
        'store_id',
        'first_name',
        'last_name',
        'email',
        'address_id',
        array(
            'class' => 'FlagColumn',
            'name' => 'active',
        ),
        /*
        'create_date',
        'last_update',
        */
        array(
            'class'=>'CButtonColumn',
        ),
    ),
)); ?>
...
```

4. Теперь проверьте грид. Он должен содержать столбец `Active` со значениями `Y` и `N`:

Displaying 1-10 of 599 result(s)

Customer	Store	First Name	Last Name	Email	Address	Active	
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	☒	  
2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	☒	  
3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	☒	  
4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	☒	  
5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	☒	  
6	2	JENNIFER	DAVIS	JENNIFER.DAVIS@sakilacustomer.org	10	☒	  
7	1	MARIA	MILLER	MARIA.MILLER@sakilacustomer.org	11	☒	  
8	2	SUSAN	WILSON	SUSAN.WILSON@sakilacustomer.org	12	☒	  
9	2	MARGARET	MOORE	MARGARET.MOORE@sakilacustomer.org	13	☒	  
10	1	DOROTHY	TAYLOR	DOROTHY.TAYLOR@sakilacustomer.org	14	☒	  

Go to page: < Previous **1** 2 3 4 5 6 7 8 9 10 Next >

Как это работает

При вызове виджета грида мы можем передать массив, содержащий класс столбца и `public`-свойства класса.

```
'columns'=>array(
    'customer_id',
    'store_id',
    'first_name',
    'last_name',
    'email',
    'address_id',
    array(
        'class' => 'FlagColumn',
        'name' => 'active',
    ),
),
```

В представленном выше фрагменте кода используется класс `FlagColumn`, и его свойство `name` получает значение `active`.

```
class FlagColumn extends CGridColumn
```

Класс столбца грида должен наследоваться от `CGridColumn` и реализовывать, по крайней мере, метод `renderDataCellContent`:

```
protected function renderDataCellContent($row, $data) {
    $value=Chtml::value($data,$this->name);
    $this->callbackUrl['pk'] = $data->primaryKey;
    $this->callbackUrl['name'] = urlencode($this->name);
    $this->callbackUrl['value'] = (int)empty($value);

    $link = Chtml::normalizeUrl($this->callbackUrl);
```

```
echo CHtml::link(!empty($value) ? 'Y' : 'N', $link,
array(
    'class' => $this->_flagClass,
));
}
```

Вывод каждого оператора `echo` в этом методе отображается как содержимое ячейки. Переменная `$row` содержит номер строки, `$data` — экземпляр модели, представляющей строку. С помощью `CHtml::value` мы получаем значение поля `active` и передаем его имя и значение вместе с первичным ключом модели в `CHtml::normalizeUrl` для создания ссылки на действие `flag`.

Теперь нужно добавить немного AJAX. Мы делаем это в методе `init`:

```
public function init() {
    parent::init();
    $cs=Yii::app()->getClientScript();
    $gridId = $this->grid->getId();
    $script = <<<SCRIPT
    jQuery("({this->_flagClass}).live("click", function(e){
        e.preventDefault();
        var link = this;
        $.ajax({
            dataType: "json",
            cache: false,
            url: link.href,
            success: function(data){
                $('#$gridId').yiiGridView.update('$gridId');
            }
        });
    });
    SCRIPT;
    $cs->registerScript(__CLASS__.$gridId.'#flag_link',
    $script);
}
```

Для каждой ссылки, созданной нами в методе `renderDataCellContent`, мы добавляем обработчик события `click`. Применять `live` необходимо для корректного поведения постраничной разбивки при использовании AJAX в гриде. При щелчке мы посылаем AJAX-запрос действию `flag`. После успешного завершения грид обновляется.

Действие само по себе очень простое. Мы получаем экземпляр модели, меняем значение поля и сохраняем модель.

Для поддержки сортировки в классе `FlagColumn` реализуется ещё один метод:


```
protected function renderHeaderCellContent()
{
    if($this->grid->enableSorting && $this->sortable &&
        $this->name!==null)
        echo $this->grid->dataProvider->getSort()->link(
            $this->name,$this->header);
    else if($this->name!==null && $this->header===null)
    {
        if($this->grid->dataProvider instanceof CActiveDataProvider)
            echo CHtml::encode($this->grid->dataProvider->
                model->getAttributeLabel($this->name));
        else
            echo CHtml::encode($this->name);
    }
    else
        parent::renderHeaderCellContent();
}
```

Это точная копия одноименного метода из `CDataProvider`. Если включена сортировка и грида, и столбца, то отображается ссылка на сортировку грида.

Дополнительные материалы

За дополнительной информацией обращайтесь по адресам:

- <http://www.yiiframework.com/doc/api/CGridColumn>;
- <http://www.yiiframework.com/doc/api/CButtonColumn>;
- <http://www.yiiframework.com/doc/api/CCheckBoxColumn>;
- <http://www.yiiframework.com/doc/api/CDataColumn>;
- <http://www.yiiframework.com/doc/api/CLinkColumn>.

Смотрите также

- Рецепт «Использование гридов» в этой главе.



ГЛАВА 8.

Расширение Yii

В этой главе мы рассмотрим:

- Создание поведений модели.
- Создание компонентов.
- Создание действий контроллера, пригодных для повторного использования.
- Создание контроллеров, пригодных для повторного использования.
- Создание виджета.
- Создание консольных команд.
- Создание фильтров.
- Создание модулей.
- Создание своего обработчика шаблонов.
- Подготовка расширений к публикации.

Введение

В этой главе мы рассмотрим не только реализацию собственных расширений Yii, но и покажем, как сделать их пригодными для повторного использования и полезными для сообщества. В дополнение мы сосредоточимся на многих вещах, которые сделают ваше расширение наиболее эффективным.

Создание поведений модели

В современных веб-приложениях много похожих решений. В ведущих продуктах, таких как Google Gmail, создаются интересные решения для пользовательских интерфейсов. Один из них – «мягкое» удаление. Вместо безвозвратного удаления со множеством подтверждений Gmail позволяет быстро отметить сообщения как удаленные, а потом

с легкостью восстановить их. То же самое можно делать с любыми объектами, например записями в блогах, комментариями и т. п.

Создадим поведение (*behavior*), которое позволит отмечать модели как удаленные, восстанавливать их, выбирать удаленные, ещё не удаленные или все модели. В этом рецепте мы используем подход «разработка через тестирование» (*test driven development*) для проектирования поведения и проверки корректности реализации.

Подготовка

- Создайте базу данных и добавьте в нее таблицу `post`:

```
CREATE TABLE 'post' (  
    'id' int(11) NOT NULL auto_increment,  
    'text' text,  
    'title' varchar(255) default NULL,  
    'is_deleted' tinyint(1) NOT NULL default '0',  
    PRIMARY KEY ('id')  
)
```

- Настройте Yii для использования этой базы данных в вашем главном приложении (`protected/config/main.php`).
- Убедитесь, что у приложения `test` те же настройки (`protected/config/test.php`).
- Раскомментируйте компонент `fixture` в настройках приложения `test`.
- С помощью Gii создайте модель `Post`.

Как это делается

1. Подготовим тестовую среду, начав с определения фикстур (*fixtures*) для модели `Post` в файле `protected/tests/fixtures/post.php`:

```
<?php  
return array(  
    array(  
        'id' => 1,  
        'title' => 'post1',  
        'text' => 'post1',  
        'is_deleted' => 0,  
    ),  
    array(  
        'id' => 2,  
        'title' => 'post2',  
        'text' => 'post2',  
    )  
)
```

```
        'is_deleted' => 1,
    ),
    array(
        'id' => 3,
        'title' => 'post3',
        'text' => 'post3',
        'is_deleted' => 0,
    ),
    array(
        'id' => 4,
        'title' => 'post4',
        'text' => 'post4',
        'is_deleted' => 1,
    ),
    array(
        'id' => 5,
        'title' => 'post5',
        'text' => 'post5',
        'is_deleted' => 0,
    ),
);
```

2. Затем нам нужно создать тест `protected/tests/unit/soft_delete/SoftDeleteBehaviorTest.php`:

```
<?php
class SoftDeleteBehaviorTest extends CDbTestCase
{
    protected $fixtures = array(
        'post' => 'Post',
    );

    function testRemoved()
    {
        $postCount = Post::model()->removed()->count();
        $this->assertEquals(2, $postCount);
    }

    function testNotRemoved()
    {
        $postCount = Post::model()->notRemoved()->count();
        $this->assertEquals(3, $postCount);
    }

    function testRemove()
    {
        $post = Post::model()->findPk(1);
        $post->remove()->save();
        $this->assertNull(Post::model()->notRemoved()->
```

```

findByPk(1));
}

function testRestore()
{
    $post = Post::model()->findByPk(2);
    $post->restore()->save();

    $this->assertNotNull(Post::model()->notRemoved()->
findByPk(2));
}

function testIsDeleted()
{
    $post = Post::model()->findByPk(1);
    $this->assertFalse($post->isRemoved());

    $post = Post::model()->findByPk(2);
    $this->assertTrue($post->isRemoved());
}
}

```

3. Теперь нам нужно реализовать `behavior`, прикрепить его к модели и убедиться, что тест проходит успешно. Создайте новую директорию внутри `protected/extensions` и назовите ее `soft_delete`. В этой директории создайте файл `SoftDeleteBehavior.php`. Сначала добавим поведение в модель `Post`:

```

class Post extends CActiveRecord
{
    // ...

    public function behaviors()
    {
        return array(
            'softDelete' => array(
                'class' => 'ext.soft_delete.SoftDeleteBehavior'
            ),
        );
    }
    // ...
}

```

4. Теперь реализуем `protected/extensions/soft_delete/SoftDeleteBehavior.php`:

```

<?php
class SoftDeleteBehavior extends CActiveRecordBehavior
{

```

```
public $flagField = 'is_deleted';

public function remove()
{
    $this->getOwner()->{$this->flagField} = 1;
    return $this->getOwner();
}

public function restore()
{
    $this->getOwner()->{$this->flagField} = 0;
    return $this->getOwner();
}

public function notRemoved()
{
    $criteria = $this->getOwner()->getDbCriteria();
    $criteria->compare($this->flagField, 0);
    return $this->getOwner();
}

public function removed()
{
    $criteria = $this->getOwner()->getDbCriteria();
    $criteria->compare($this->flagField, 1);
    return $this->getOwner();
}

public function isRemoved()
{
    return (boolean)$this->getOwner()->{$this->
flagField};
}
}
```

Выполните тест, убедитесь, что он проходит успешно.

5. Вот и всё. Мы создали поведение, пригодное для повторного использования для всех будущих проектов. Достаточно будет лишь подключить его к модели.

Как это работает

Начнем с теста. Поскольку мы хотим использовать набор моделей, то нам понадобятся фикстуры. Набор фикстур применяется к базе данных каждый раз, когда выполняется тестовый метод. Для использования фикстур тестовый класс должен быть унаследован от `CDbTestCase` и иметь объявленную `protected` переменную `$fixtures`:

```
protected $fixtures = array(
    'post' => 'Post'
);
```

В приведенном коде `post` – имя файла, содержащего фикстуры, а `Post` – имя модели, к которой они будут применяться.

Начнем с тестирования именованных групп условий `removed` и `notRemoved`. Сначала следует ограничить результаты поиска только удаленными объектами, потом только неудаленными. Поскольку мы знаем, какие именно данные получим из фикстур, то можем проверить подсчет удаленных и неудаленных объектов примерно следующим образом:

```
$postCount = Post::model()->removed()->count();
$this->assertEquals(2, $postCount);
```

Теперь протестируем методы `remove` и `restore`. Ниже представлен тест для метода `remove`:

```
$post = Post::model()->findByPk(1);
$post->remove()->save();

$this->assertNull(Post::model()->notRemoved()->findByPk(1));
```

Мы находим объект по `id`, удаляем его и пытаемся найти его снова с использованием `notRemoved`. Поскольку мы удалили объект, поиск должен вернуть `null`.

Наконец, протестируем метод `isRemoved`, который просто возвращает содержимое соответствующего поля в виде `boolean`.

Теперь обратим внимание на интересные детали реализации. Поскольку мы создаем поведение для модели `Active Record`, нам нужно расширить `CActiveRecordBehavior`. В код поведения мы можем добавить свои методы. Они будут «подмешаны» к методам модели, к которой мы подключим поведение. Именно это мы используем для того, чтобы добавить методы `remove/restore/isRemoved` и группы условий `removed/notRemoved`:

```
public function remove()
{
    $this->getOwner()->{$this->flagField} = 1;
    return $this->getOwner();
}
public function removed()
{
    $criteria = $this->getOwner()->getDbCriteria();
    $criteria->compare($this->flagField, 1);
```

```
return $this->getOwner();  
}
```

Мы используем метод `getOwner`, возвращающий объект, к которому прикреплено поведение. В нашем случае это модель, поэтому мы можем работать с ее данными или изменять ее критерии поиска. Чтобы сделать возможными вызовы методов по цепочке, мы возвращаем экземпляр модели:

```
$post->remove()->save();
```

И ещё

В этом рецепте следует упомянуть ещё о некоторых вещах.

CActiveRecordBehavior и CModelBehavior

Иногда нужна большая гибкость поведения, например реагирование на события модели. Как `CActiveRecordBehavior`, так и `CModelBehavior` добавляют методы-события, которые мы можем переопределить, чтобы выполнить в определенное время полезные действия. Например, если мы хотим обрабатывать каскадное удаление в поведении, то можем переопределить метод `afterDelete`.

Другие типы поведений

Поведение можно прикрепить не только к модели, но и к любому другому компоненту. Каждое поведение наследуется от класса `CBehavior`, поэтому мы можем пользоваться его методами:

- `getOwner` для получения компонента, к которому прикреплено поведение;
- `getEnabled` и `setEnabled` для проверки, активно ли поведение, и изменения его состояния;
- `attach` и `detach` можно использовать соответственно для инициализации поведения и удаления временных данных, созданных во время его использования.

Дополнительные материалы

Чтобы больше узнать о поведениях, обратитесь к описанию API:

- <http://www.yiiframework.com/doc/api/CActiveRecordBehavior>;
- <http://www.yiiframework.com/doc/api/CModelBehavior>;
- <http://www.yiiframework.com/doc/api/CBehavior>.

Смотрите также

- Рецепт «Подготовка расширений к публикации» в этой главе.

Создание компонентов

Если у вас есть некий код, который можно использовать повторно, и при этом вы не уверены, как его оформить, в виде виджета, поведения или как-то ещё, скорее всего, это компонент. Компонент наследуется от класса `CComponent` или `CApplicationComponent` и чаще всего в том или ином виде подключается к приложению через файл конфигурации `protected/config/main.php`. В этом состоит главное преимущество перед простыми классами PHP. В дополнение мы получаем поведения, события, а также геттеры и сеттеры.

В качестве примера мы реализуем, подключим к приложению и используем простой компонент приложения `EImageManager`, который будет при помощи библиотеки GD изменять размер изображений.

Подготовка

Чтобы увидеть в действии изменение размеров изображений, вам понадобится установленное и настроенное расширение GD для PHP.

Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/components/EImageManager.php`:

```
<?php
class EImageManager extends CApplicationComponent
{
    protected $image;
    protected $width;
    protected $height;

    protected $newWidth;
    protected $newHeight;

    public function resize($width = false, $height =
false){
        if($width!==false) $this->newWidth = $width;
        if($height!==false) $this->newHeight = $height;

        return $this;
    }
}
```

```
public function load($filePath)
{
    list($this->width, $this->height, $type) =
        getimagesize($filePath);

    switch ($type)
    {
        case IMAGETYPE_GIF:
            $this->image = imagecreatefromgif($filePath);
            break;
        case IMAGETYPE_JPEG:
            $this->image = imagecreatefromjpeg($filePath);
            break;
        case IMAGETYPE_PNG:
            $this->image = imagecreatefrompng($filePath);
            break;
        default:
            throw new CException('Unsupported image type '
. $type);
    }

    return $this;
}

public function save($filePath)
{
    $ext = pathinfo($filePath, PATHINFO_EXTENSION);

    $newImage = imagecreatetruecolor($this->newWidth,
        $this->newHeight);
    imagecopyresampled($newImage, $this->image, 0, 0, 0, 0,
        $this->newWidth, $this->newHeight, $this->width,
        $this->height);

    switch($ext)
    {
        case 'jpg':
        case 'jpeg':
            imagejpeg($newImage, $filePath);
            break;
        case 'png':
            imagepng($newImage, $filePath);
            break;
        case 'gif':
            imagegif($newImage, $filePath);
            break;
        default:
```

```
        throw new CException("Unsupported image type ",
$ext);
    }

    imagedestroy($newImage);
    if(!is_file($filePath))
        throw new CException("Failed to write image.");
    }

    function __destruct()
    {
        imagedestroy($this->image);
    }
}
```

2. Теперь нужно прикрепить наш компонент к приложению. В файл `protected/config/main.php` необходимо добавить следующее:

```
...
// application components
'components'=>array(
    'image' => array(
        'class' => 'EImageManager',
    ),
...

```

3. Теперь можно использовать компонент:

```
Yii::app()->image
->load(Yii::getPathOfAlias('webroot').'/src.png')
->resize(100,100)
->save(Yii::getPathOfAlias('webroot').'/dst.png');
```

Как это работает

Чтобы компонент можно было прикрепить к приложению, он должен быть унаследован от `CApplicationComponent`. Для его использования в приложении достаточно добавить массив в секцию `components` файла конфигурации. Значение `class` задает класс компонента, а остальные значения присваиваются соответствующим им `public`-свойствам или сеттерам компонента.

Реализация сама по себе очень проста: мы оборачиваем вызовы GD в удобный API с методами `save`, `load` и `resize`. Чтобы можно было вызывать методы по цепочке, `load` и `resize` возвращают тот же экземпляр того же компонента.

Наш класс доступен по имени компонента через `Yii::app()`. В нашем случае это `Yii::app()->image`.

И ещё

Кроме создания собственных компонентов, полезно знать, как подправить существующие компоненты...

Переопределение существующих компонентов приложения

Чаще всего в создании собственных компонентов нет необходимости, так как другие типы расширений, например виджеты и поведения, подходят почти для всех видов кода, пригодного для повторного использования. Тем не менее переопределение основных компонентов фреймворка широко применяется на практике для реализации нестандартного для фреймворка поведения. При этом не требуется детального изучения исходного кода.

Например, для того чтобы получать роль пользователя из базы данных, используя `Yii::app()->user->role`, можно расширить компонент `CWebUser`:

```
<?php
class WebUser extends CWebUser {
    private $_model = null;

    function getRole() {
        if($user = $this->getModel()){
            return $user->role;
        }
        else return 'guest';
    }

    private function getModel(){
        if($this->_model === null){
            if($this->id === null) return null;
            $this->_model = User::model()->findByPk($this->id);
        }
        return $this->_model;
    }
}
```

Для замены стандартного компонента необходимо изменить настройки в `main.php`:

```
...
// application components
'components'=>array(
    'user'=>array(
        'class' => 'WebUser',
        // другие свойства
    )
)
```

```
    ),
    ...

```

В приведенном выше коде мы указали свой нестандартный класс для компонента `user`.

Дополнительные материалы

Чтобы узнать больше о компонентах, обратитесь к описанию API:

- <http://www.yiiframework.com/doc/api/CComponent/>;
- <http://www.yiiframework.com/doc/api/CApplicationComponent/>.

Создание действий контроллера, пригодных для повторного использования

Типичные действия, такие, например, как удаление модели Active Record по первичному ключу или получение данных для автодополнения AJAX, можно выделить в отдельные действия контроллера и затем прикреплять их к контроллерам, как только они понадобятся.

В этом рецепте мы создадим пригодное для повторного использования действие `delete`, удаляющее указанную модель Active Record по ее первичному ключу.

Подготовка

- Создайте новое приложение Yii с помощью `yiic webapp`.
- Создайте новую базу данных и сконфигурируйте ее.
- Выполните следующий SQL-скрипт:

```
CREATE TABLE 'post' (
    'id' int(11) NOT NULL auto_increment,
    'text' text,
    'title' varchar(255) default NULL,
    PRIMARY KEY ('id')
);

CREATE TABLE 'comment' (
    'id' int(11) NOT NULL auto_increment,
    'text' text,
    PRIMARY KEY ('id')
);
```

- Создайте модели для `post` и `comment` с помощью Gii.

Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/extensions/actions/EDeleteAction.php`:

```
<?php
class EDeleteAction extends CAction
{
    public $modelName;
    public $redirectTo = array('index');

    /**
     * Запускает действие.
     * Этот метод вызывается из контроллера, которому
     * принадлежит данное действие.
     */
    public function run($pk)
    {
        ActiveRecord::model($this->modelName)->
            deleteByPk($pk);
        if(Yii::app()->getRequest()->
            getIsAjaxRequest())
        {
            Yii::app()->end(200, true);
        }
        else
        {
            $this->getController()->redirect($this->
                redirectTo);
        }
    }
}
```

2. Теперь нужно прикрепить действие к контроллеру `protected/controllers/DeleteController.php`:

```
<?php
class DeleteController extends CController
{
    public function actions()
    {
        return array(
            'deletePost' => array(
                'class' => 'ext.actions.EDeleteAction',
                'modelName' => 'Post',
                'redirectTo' => array('indexPosts'),
            ),
            'deleteComment' => array(
```

```
        'class' => 'ext.actions.EDeleteAction',  
        'modelName' => 'Comment',  
        'redirectTo' => array('indexComments'),  
    ),  
);  
}  
  
public function actionIndexPosts()  
{  
    echo "I'm index action for Posts.";  
}  
  
public function actionIndexComments()  
{  
    echo "I'm index action for Comments.";  
}  
}
```

3. Вот и все. Теперь можно удалить сообщение, перейдя к `/delete/deletePost/pk/<pk>`, а комментарий – перейдя к `/delete/deleteComment/pk/<pk>`. После удаления вы будете перенаправлены в действие `index` соответствующего контроллера.

Как это работает

Чтобы создать внешнее действие для контроллера, нужен класс, наследуемый от `CAction`. Единственный обязательный для реализации в нем метод – `run`. В нашем случае он принимает параметр с именем `pk` из `$_GET` и пытается удалить соответствующую модель. При этом используется автоматическое связывание параметров, реализованное в Yii.

Чтобы сделать действие настраиваемым, мы создаем два `public`-свойства, изменяемых из контроллера. `modelName` содержит имя модели, с которой мы работаем. `redirectTo` указывает маршрут, по которому будет перенаправлен пользователь.

Сама настройка происходит путем реализации метода `actions` контроллера. В методе можно прикрепить действие один или несколько раз, а также изменить его `public`-свойства.

И ещё

В `CAction` реализованы два полезных метода. Первый из них – `getController`. С его помощью можно получить экземпляр контрол-

лера, к которому прикреплено действие. Это может понадобиться, например, для перенаправления на другое действие или генерации URL.

Второй метод — `getId`. Он возвращает имя действия, указанного в методе `actions` контроллера.

Дополнительные материалы

Чтобы больше узнать о внешних действиях для контроллеров, обратитесь к API:

- <http://www.yiiframework.com/doc/api/CAction/>;
- <http://www.yiiframework.com/doc/api/CController/#actions-detail>.

Смотрите также

- Рецепт «Создание контроллеров, пригодных для повторного использования» в этой главе.
- Рецепт «Подготовка расширений для публикации» в этой главе.

Создание контроллеров, пригодных для повторного использования

Если вы создаете много приложений или однотипных контроллеров, выделение общей части кода в пригодный для повторного использования контроллер сэкономит вам много времени.

В этом рецепте мы создадим простой универсальный контроллер `api`, пригодный для повторного использования и реализующий JSON CRUD API для модели. Он будет принимать входные данные из `POST` и `GET`, возвращать данные в формате JSON вместе с соответствующим кодом ответа HTTP.

Подготовка

- Создайте новое приложение Yii с помощью `yiic webapp`.
- Создайте модель с помощью `Gii`. В примере будет использоваться модель `Post`, но вы можете использовать любую понравившуюся вам модель.

Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/extensions/json_api/JsonApiController.php`:

```
<?php
class JsonApiController extends CController
{
    const RESPONSE_OK = 'OK';
    const RESPONSE_NO_DATA = 'No data';
    const RESPONSE_NOT_FOUND = 'Not found';
    const RESPONSE_VALIDATION_ERRORS = 'Validation errors';

    public $modelName;
    public function init()
    {
        parent::init();
        if(empty($this->modelName))
            throw new CException("You should set modelName
                before using JsonApiController.");
    }

    public function actionCreate()
    {
        if(empty($_POST))
            $this->respond(400, self::RESPONSE_NO_DATA);

        $model = new $this->modelName;
        $model->setAttributes($_POST);

        if($model->save())
            $this->respond(200, self::RESPONSE_OK);
        else
            $this->respond(400, self::RESPONSE_VALIDATION_
ERRORS,
                $model->getErrors());
    }

    public function actionGet($pk)
    {
        $model = CActiveRecord::model
            ($this->modelName)->findByPk($pk);
        if(!$model)
            $this->respond(404, self::RESPONSE_NOT_FOUND);

        $this->respond(200, self::RESPONSE_OK,
            $model->getAttributes());
    }
}
```

```
    }

    public function actionUpdate($pk)
    {
        if(empty($_POST))
            $this->respond(400, self::RESPONSE_NO_DATA);

        $model = CActiveRecord::model
            ($this->modelName)->findByPk($pk);
        if(!$model)
            $this->respond(404, self::RESPONSE_NOT_FOUND);

        $model->setAttributes($_POST);
        if($model->save())
            $this->respond(200, self::RESPONSE_OK);
        else
            $this->respond(400, self::RESPONSE_VALIDATION_
ERRORS,
                $model->getErrors());
    }

    public function actionDelete($pk)
    {
        if(CActiveRecord::model($this->modelName)->
deleteByPk($pk))
        {
            $this->respond(200, self::RESPONSE_OK);
        }
        else
        {
            $this->respond(404, self::RESPONSE_NOT_FOUND);
        }
    }

    protected function respond($statusCode, $status,
$data = array())
    {
        $response['status'] = $status;
        $response['data'] = $data;

        echo CJSON::encode($response);

        Yii::app()->end($statusCode, true);
    }
}
```

2. Теперь нам нужно подключить приведенный выше код к нашему приложению через `protected/config/main.php`. Это

можно сделать добавлением конфигурации контроллера к свойству `controllerMap` класса `CWebApplication`. Добавим следующее сразу после открытия массива настроек:

```
...  
'controllerMap' => array(  
    'api' => array(  
        'class' => 'ext.json_api.JsonApiController',  
        'modelName' => 'Post',  
    ),  
,  
...  
);
```

3. Вот и все. Мы подключили контроллер и указали, что он должен работать с моделью `Post`.

Вам понадобятся формы для ввода данных, однако если у вас уже есть некоторые данные, то вы можете использовать методы `get` прямо из URL: `/api/get/pk/1`. Приложения должны возвращать данные примерно следующего вида:

```
("status":"OK", "data":{"id":"1", "text":"post1",  
    "title":"post1", "is_deleted":"0"})
```

Как это работает

Когда вы запускаете приложение и передаете маршрут, такой как `api/get`, то, прежде чем выполнить `ApiController::actionGet`, Yii проверяет, задано ли свойство `controllerMap`. Поскольку мы задали в качестве контроллера `api`, Yii выполнит его, вместо того чтобы действовать, как обычно.

В самом контроллере мы создали свойство `modelName`, чтобы иметь возможность подключать его несколько раз для реализации API для нескольких моделей. Мы задаем это свойство, когда подключаем контроллер.

Контроллер сам по себе не слишком отличается от стандартного, за исключением нескольких хитростей:

- при работе с Active Record мы создаем новый класс примерно следующего вида:

```
$model = new $this->modelName;
```

- и получаем finder следующим образом:

```
$model = CActiveRecord::model($this->modelName);
```

- это позволяет нам не привязываться к одной модели, а указывать ее имя;
- мы используем CJSON для получения JSON из массивов и моделей;
- `Yii::app()->end()` используется для завершения приложения с указанным кодом ответа HTTP.

И ещё

Чтобы узнать больше о сопоставлении контроллеров (*controllers map*), обратитесь к описанию API:

- <http://www.yiiframework.com/doc/api/CWebApplication#controllerMap-detail>.

Смотрите также

- Рецепт «Создание действий контроллеров, пригодных для повторного использования» в этой главе.
- Рецепт «Подготовка расширений к публикации» в этой главе.

Создание виджета

Виджет – часть представления, которую можно использовать повторно и которая не просто отображает данные, но и делает это в соответствии с некой логикой. Виджет может запрашивать данные из моделей и использовать собственные представления. Он является чем-то вроде усеченной и пригодной для повторного использования версии модуля.

Создадим виджет, который нарисует круговую диаграмму с использованием Google API.

Подготовка

Создайте новое приложение Yii с помощью `yiiic`.

Как это делается

1. Создайте файл `protected/extensions/chart/EChartWidget.php`:

```
<?php
class EChartWidget extends CWidget
```

```

{
    public $title;
    public $data=array();
    public $labels=array();

    public function run()
    {
        echo "<img
            src=\"http://chart.apis.google.com/
chart?chtt=\".urlencode
            ($this->title).\"&cht=pc&chs=300x150&chd=\".
            $this->encodeData($this->data).\"&chl=\".implode
            ('|', $this->labels).\">\";
        }

        protected function encodeData($data)
        {
            $maxValue=max($data);

            $chars='ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

            $chartData="s:";
            for($i=0;$i<count($data);$i++)
            {
                $currentValue=$data[$i];
                if($currentValue>-1)
                    $chartData.=substr($chars,61*($currentValue/
                    $maxValue),1);
                else
                    $chartData.='_';
            }

            return $chartData."&chxt=y&chxl=0:10!\".$maxValue;
        }
    }
}

```

2. Теперь создайте контроллер `protected/controllers/Chart-Controller.php`:

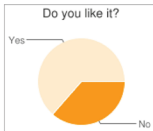
```

<?php
class ChartController extends CController
{
    public function actionIndex()
    {
        $value = rand(10, 90);
        $this->widget('ext.chart.EChartWidget', array(
            'title' => 'Do you like it?',
            'data' => array(

```

```
        $value, 100-$value
    ),
    'labels' => array(
        'No',
        'Yes',
    ),
    ),
));
}
```

3. Сейчас попробуйте выполнить действие `index` контроллера. Вы должны увидеть круговую диаграмму примерно следующего вида:



Как это работает

Как и для любого другого типа расширений, мы создаем некоторые `public` свойства, которые можно установить при вызове виджета с помощью `CController::widget`. В данном случае мы меняем заголовок, набор данных и подписи.

Основной метод виджета — `run()`. В нашем виджете мы генерируем URL, указывающий на Google API для построения графиков, и затем выводим тег `<img>`.

И ещё

Чтобы узнать больше о виджетах, обратитесь к описанию API:

- <http://www.yiiframework.com/doc/api/CWidget/>;
- <http://www.yiiframework.com/doc/api/CCaptcha/>.

Смотрите также

- Рецепт «Конфигурирование компонентов» в главе 1 «Под капотом».

- Рецепт «Конфигурирование умолчаний виджетов» в главе 1.
- Рецепт «Создание своего виджета ввода с помощью CWidget» в главе 4 «Работа с формами».
- Рецепт «Подготовка расширений к публикации» в этой главе.

Создание консольных команд

В Yii неплохо реализована поддержка консольных команд. Создать консольную команду можно гораздо быстрее, чем веб-интерфейс. Если вам необходим инструмент для приложения, который смогут использовать разработчики или администраторы, то консольные команды – хороший выбор. Чтобы показать, как создается консольная команда, мы создадим простую команду, которая очищает кеш, временные директории и т. п.

Подготовка

Создайте новое приложение Yii с помощью `yiiic webapp`.

Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/extensions/clean_command/ECleanCommand.php`:

```
<?php
class ECleanCommand extends CConsoleCommand
{
    public $webRoot;
    public function actionCache()
    {
        $cache=Yii::app()->getComponent('cache');
        if($cache!==null){
            $cache->flush();
            echo "Done.\n";
        }
        else {
            echo "Please configure cache component.\n";
        }
    }

    public function actionAssets()
    {
        if(empty($this->webRoot))
        {
```

```
        echo "Please specify a path to webRoot
            in command properties.\n";
        Yii::app()->end();
    }

    $this->cleanDir($this->webRoot.'/assets');

    echo "Done.\n";
}

public function actionRuntime()
{
    $this->cleanDir(Yii::app()->getRuntimePath());
    echo "Done.\n";
}

private function cleanDir($dir)
{
    $di = new DirectoryIterator ($dir);
    foreach($di as $d)
    {
        if(!$d->isDot())
        {
            echo "Removed ".$d->getPathname()."\n";
            $this->removeDirRecursive($d->
getPathname());
        }
    }
}

private function removeDirRecursive($dir)
{
    $files = glob($dir.'*', GLOB_MARK);
    foreach ($files as $file)
    {
        if (is_dir($file))
            $this->removeDirRecursive($file);
        else
            unlink($file);
    }
    if (is_dir($dir))
        rmdir($dir);
}
}
```

2. Теперь нужно прикрепить команду к консольному приложению. По умолчанию консольное приложение использует отдельный файл загрузки `yiic.php`:


```
<?php

// change the following paths if necessary
$yiic=dirname(__FILE__).'../../framework/yiic.php';
$config=dirname(__FILE__).'../config/console.php';

require_once($yiic);
```

3. Таким образом, конфигурация находится в файле `protected/config/console.php`. Добавим нашу консольную команду к свойству `commandMap`:

```
// Это конфигурация для консольного приложения yiic
// Показанным ниже образом могут быть заданы любые
свойства CConsoleApplication
return array(
    'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
    'name'=>'My Console Application',
    'commandMap' => array(
        'clean' => array(
            'class' => 'ext.clean_command.ECcleanCommand',
            'webRoot' => 'path/to/your/application/webroot',
        ),
    ),
);
```

4. Не забудьте заменить `webRoot` на реальный путь. Вот и всё. Теперь войдите в директорию `protected` и попробуйте выполнить команды:

```
yiic clean
yiic clean cache
yiic clean assets
yiic clean runtime
```

Как это работает

Все консольные команды должны быть унаследованы от класса `CConsoleCommand`. Поскольку все команды выполняются в `CConsoleApplication`, а не в `CWebApplication`, то у нас нет способа определить корневую директорию сервера веб-приложений. Для этой цели мы создаем изменяемое публичное свойство с именем `webRoot`.

Структура консольной команды похожа на обычный контроллер. Мы определяем несколько действий, которые можно выполнить, набрав `yiic <консольная команда> <действие>`.

Как видите, здесь не используются представления, поэтому можно сосредоточиться на реализации задачи вместо дизайна, разметки

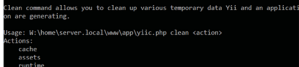
и т. п. Тем не менее пользователь должен получить информативный вывод команды, чтобы видеть, что происходит. Это делается обычными операторами `echo`.

И ещё

Если команда относительно сложная, как `message` или `migrate` из состава Yii, то разумно будет добавить дополнительное описание доступных опций и действий. Это делается переопределением метода `getHelp`:

```
public function getHelp()
{
    $out = "Clean command allows you to clean up various
        temporary data Yii and an application are
        generating.\n\n";
    return $out.parent::getHelp();
}
```

При запуске `yiiic clean` будет выведено:



```
Clean command allows you to clean up various temporary data Yii and an applicati
on are generating.

Usage: W:\home\server.local\www\app\yiiic.php clean <action>
Actions:
    cache
    assets
    runtime
```

Дополнительные материалы

- <http://www.yiiframework.com/doc/guide/ru/topics/console>;
- <http://www.yiiframework.com/doc/api/CConsoleCommand/>;
- <http://www.yiiframework.com/doc/api/CConsoleApplication/>.

Смотрите также

- Рецепт «Подготовка расширений к публикации» в этой главе.

Создание фильтров

Фильтр – это класс, который можно выполнить до или после выполнения действия. Фильтры можно использовать для изменения контекста выполнения или оформления вывода. В нашем примере мы реализуем простой фильтр вывода, который сжимает выводимый HTML, удаляя дополнительное форматирование.

Не используйте этот фильтр в реальных приложениях: GZIP гораздо лучше поможет снизить объем трафика. К тому же удаление форматирования в `pre`, `textarea` и некоторых других тегах небезопасно.

Подготовка

Создайте новое приложение Yii с помощью `yiiic`.

Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/extensions/compress_html/ECompressHtmlFilter.php`:

```
<?php
class ECompressHtmlFilter extends CFilter
{
    protected function preFilter($filterChain)
    {
        ob_start();
        return parent::preFilter($filterChain);
    }

    protected function postFilter($filterChain)
    {
        $out = ob_get_clean();
        echo preg_replace("\>(\s+|\t+|\n+)<\", *><\"",
        $out);
        parent::postFilter($filterChain);
    }
}
```

2. Вот и все. Теперь нужно подключить это к приложению. Поскольку у нас уже есть `SiteController`, то добавим фильтр к нему, переопределив метод `filters`:

```
public function filters()
{
    return array(
        array(
            'ext.compress_html.ECompressHtmlFilter'
        ),
    );
}
```

3. Теперь запустите приложение и просмотрите исходный код HTML. Это должна быть одна строка текста примерно следующего вида:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/  
xhtml1-transitional.dtd"><html xmlns="http://www.  
w3.org/1999/xhtml" xml:lang="en" lang="en"><head>  
<meta http-equiv="Content-Type" content="text/html;  
charset=utf-8" /><meta name="language" content="en"  
</><!--blueprint CSS framework -->
```

...

Как это работает

В фильтре должен быть реализован как минимум интерфейс `IFilter`, но, поскольку мы хотим добавить предварительную и последующую фильтрацию, мы можем наследовать наш фильтр от `CFilter`. План состоит в том, чтобы действие каким-то образом генерировало все содержимое, а перед выводом в браузер производилась некоторая обработка. Доступная в РНР буферизация вывода хорошо подходит для этой задачи, поэтому мы переопределяем `preFilter`, где включаем буферизацию при помощи `ob_start`, и `postFilter`, где получаем содержимое буфера с помощью `ob_get_clean`. Там же мы выполняем обработку регулярными выражениями и выводим результат.

Заметьте, что мы вызываем методы класса-родителя, когда переопределяем `preFilter` и `postFilter`. Это позволяет разработчику выстраивать цепочки фильтров при настройке контроллера.

И ещё

Чтобы больше узнать о фильтрах, обратитесь к описанию API:

- <http://www.yiiframework.com/doc/api/CFilter>;
- <http://www.yiiframework.com/doc/api/IFilter>.

Смотрите также

- Рецепт «Подготовка расширений к публикации» в этой главе.

Создание модулей

Если вы создали сложную часть приложения и хотите использовать ее с некоторыми изменениями в следующем своем проекте, то вам, скорее всего, нужно создать модуль.

В этом рецепте мы увидим, как создается модуль `wiki`. Для простоты мы не будем подробно останавливаться на разграничении прав доступа и позволим всем редактировать все:

Подготовка

Выполните следующие шаги:

1. Создайте новое приложение Yii с помощью `yii webapp`.
2. Настройте базу данных MySQL и выполните следующую SQL-команду:

```
CREATE TABLE 'wiki' (  
    'id' varchar(255) NOT NULL,  
    'text' text NOT NULL,  
    PRIMARY KEY ('id')  
)
```

3. Создайте с помощью Gii модель `wiki`.
4. Создайте с помощью Gii модуль `wiki`.
5. Переместите файл `protected/models/Wiki.php` в директорию `protected/modules/wiki/models/`.
6. Добавьте `wiki` в секцию `modules` файла `protected/config/main.php`:

```
'modules'=>array(  
    // uncomment the following to enable the Gii tool  
    'gii'=>array(  
        'class'=>'system.gii.GiiModule',  
        'password'=>false,  
    ),  
    'wiki'  
)
```

Как это делается

Начнем с планирования:

- Wiki представляет собой набор страниц, которые могут ссылаться одна на другую по имени.
- Обычно в `wiki` используется простая читаемая разметка вместо HTML.
- Когда пользователь переходит к странице, которой не существует, ему предлагают ее создать.
- Чтобы удалить страницу, нужно сохранить ее с пустым телом.

Основываясь на приведенном выше, в качестве языка разметки будем использовать `markdown`. К нему мы добавим возможность ссылаться на другую страницу по имени. Пусть это будет `[[имя страницы]]` или `[[Заголовок|имя страницы]]`, в случае когда для ссылки нужно использовать другое имя.

1. Сначала добавим wiki-ссылки в CMarkdownParser. Создайте файл `protected/modules/wiki/components/WikiMarkdownParser.php`:

```
<?php
class WikiMarkdownParser extends CMarkdownParser
{
    public function transform($text)
    {
        $text = preg_replace_callback('~\[\[([.]*)(?:\|([.]*))\]\]|~',
        array($this, 'processWikiLinks'), $text);
        return parent::transform($text);
    }

    protected function processWikiLinks($matches)
    {
        $page = $matches[1];
        $title = isset($matches[2]) ? $matches[2] : $matches[1];
        return CHtml::link(CHtml::encode($title), array(
            'view', 'id' => $page,
        ));
    }
}
```

2. Теперь используем это, добавив метод `getHtml` в модель `protected/modules/wiki/models/Wiki.php`:

```
public function getHtml()
{
    $parser = new WikiMarkdownParser();
    return $parser->transform($this->text);
}
```

Перейдем к изменению `protected/modules/wiki/controller/DefaultController.php`. Нам нужны два действия: `view` и `edit`. В дополнение мы создадим действие `index`, которое будет обращаться к действию `view` с параметром `id = index`. Поскольку `view` для действия `index` нам не пригодится, его можно спокойно удалить.

1. Теперь отредактируйте файл `protected/modules/wiki/controller/DefaultController.php`:

```
class DefaultController extends Controller
{
    public function actionIndex()
    {
        $this->actionView('index');
    }
}
```

```
public function actionView($id)
{
    $model = Wiki::model()->findByPk($id);
    if(!$model)
    {
        $this->actionEdit($id);
        Yii::app()->end();
    }
    $this->render('view', array(
        'model' => $model,
    ));
}

public function actionEdit($id)
{
    $model = Wiki::model()->findByPk($id);
    if(!$model)
    {
        $model = new Wiki();
        $model->id = $id;
    }
    if(!empty($_POST['Wiki']))
    {
        if(!empty($_POST['Wiki']['text']))
        {
            $model->text = $_POST['Wiki']['text'];
            if($model->save())
            {
                $this->redirect(array('view', 'id' => $id));
            }
        }
        else
        {
            Wiki::model()->deleteByPk($id);
        }
    }
    $this->render('edit', array(
        'model' => $model
    ));
}
```

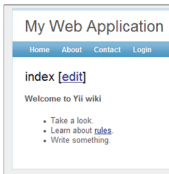
2. В дополнение нам нужны два представления. `protected/modules/wiki/views/default/view.php`:

```
<h2>
<?php echo CHtml::encode($model->id)?>
[<?php echo CHtml::link('edit', array('edit', 'id' =>
    $model->id))?>]
</h2>
<?php echo $model->html ?>
```

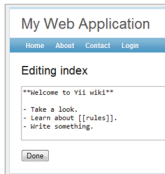
3. И ещё одно: `protected/modules/wiki/views/default/edit.php`:
- ```
<h2>Editing <?php echo CHtml::encode($model->id)?></h2>

<?php echo CHtml::beginForm()?>
 <?php echo CHtml::activeTextArea($model, 'text',
 array('cols' => 100, 'rows' => 20))?>

 <?php echo CHtml::submitButton('Done')?>
<?php echo CHtml::endForm()?>
```
4. Вот и все. Теперь запустите модуль `wiki` и опробуйте его, введя какой-нибудь текст. Не забудьте добавить внутренние ссылки типа `[[rules]]`:



5. При редактировании предыдущая страница будет выглядеть примерно так:





6. Поскольку мы вынесли модель `wiki` из приложения в модуль, модуль не имеет зависимостей от приложения, и мы можем выделить его в расширение, например в `protected/extensions/wiki/`, изменив при этом конфигурацию модуля в `protected/config/main.php`:

```
'modules'=>array(
 // uncomment the following to enable the Gii tool
 'gii'=>array(
 'class'=>'system.gii.GiiModule',
 'password'=>false,
),
 'wiki'=>array(
 'class' => 'ext.wiki.WikiModule'
),
),
```

## Как это работает

Каждый создаваемый модуль содержит главный класс, например `wikiModule/`. В нем мы можем создать конфигурируемые свойства, указать пути для импорта классов, изменить пути, подключить контроллеры и т. д. По умолчанию создаваемый с помощью Gii модуль выполняет действие `index` контроллера `default`:

```
public function actionIndex()
{
 $this->actionView('index');
}
```

В нашем модуле `wiki` действие `index` лишь вызывает `view` с передачей ему `id = index`:

```
$model = Wiki::model()->findByPrimaryKey($id);
if(!$model)
{
 $this->actionEdit($id);
 Yii::app()->end();
}

$this->render('view', array(
 'model' => $model,
));
```

Если модель с таким ID существует, мы отображаем ее с помощью `view`.

Если страницы с данным ID нет, мы вызываем действие `edit`:

```
$model = Wiki::model()->findByPk($id);
if(!$model)
{
 $model = new Wiki();
 $model->id = $id;
}

if(!empty($_POST['Wiki']))
{
 if(!empty($_POST['Wiki']['text']))
 {
 $model->text = $_POST['Wiki']['text'];
 if($model->save())
 $this->redirect(array('view', 'id' => $id));
 }
 else
 {
 Wiki::model()->deleteByPk($id);
 }
}

$this->render('edit', array(
 'model' => $model
));
```

Если модель с заданным ID не существует, мы создаем новую; если модель существует, мы ее редактируем. Данные формы для редактирования получаются из POST, и если текстовая область пуста, мы удаляем модель. Если текст есть, то мы сохраняем модель.

## И ещё

Чтобы больше узнать о модулях, обратитесь по следующим адресам:

- <http://www.yiiframework.com/doc/guide/ru/basics.module>;
- <http://www.yiiframework.com/doc/api/CWebModule/>;
- <http://www.yiiframework.com/doc/api/CModule/>;
- <http://www.yiiframework.com/doc/api/GiiModule/>.

## Смотрите также

- Рецепт «Подготовка расширений к публикации» в этой главе.

## Создание своего обработчика шаблонов

Существует множество обработчиков шаблонов для PHP. В комплект Yii включены только два типа шаблонов: чистый PHP и Prado-подобные. Если же вы хотите использовать один из существующих обработчиков или создать свой собственный, вам придется реализовать его самостоятельно. Конечно, если это уже не сделало сообщество Yii.

В этом рецепте мы реализуем поддержку шаблонов Smarty.

### Подготовка

- Создайте новое приложение Yii с помощью `yiic webapp`.
- Загрузите свежую версию Smarty 3 с адреса <http://www.smarty.net/>.
- Извлеките содержимое директории `libs` в директорию `protected/vendors/smarty/`.

### Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/extensions/smarty/ESmartyViewRenderer.php`:

```
<?php
class ESmartyViewRenderer extends CApplicationComponent
 implements IViewRenderer
{
 public $fileExtension='.tpl';
 public $filePermission=0755;

 private $smarty;

 function init()
 {
 Yii::import('application.vendors.smarty.*');
 spl_autoload_unregister(array('YiiBase','autoload'));
 require_once('Smarty.class.php');
 spl_autoload_register(array('YiiBase','autoload'));

 $this->smarty = new Smarty();

 $this->smarty->template_dir = '';
```

```

 $compileDir = Yii::app()->getRuntimePath().'/
smarty/compiled/';

 if(!file_exists($compileDir)){
 mkdir($compileDir, $this->filePermission, true);
 }

 $this->smarty->compile_dir = $compileDir;
 $this->smarty->assign('Yii', Yii::app());
 }

/**
 * Создаёт файл представления.
 * Этот метод необходим для {@link IviewRenderer}.
 * @param CBaseController контроллер или виджет,
 * создавший файл представления
 * @param string путь к файлу представления
 * @param mixed данные, передаваемые представлению
 * @param boolean должен ли возвращаться результат
 * @return mixed возвращаемый результат или null,
 * если результат не нужен
 */
 public function renderFile($context,$sourceFile,
 $data,$return)
 {
 // свойства текущего контроллера будут доступны
как {this.property}
 $data['this'] = $context;

 if(!is_file($sourceFile) || ($file=realpath(
 $sourceFile))==false)
 throw new CException(Yii::t('ext','View file
 "$sourceFile" does not exist.', array('
 {file}'=>$sourceFile)));

 $this->smarty->assign($data);
 if($return)
 return $this->smarty->fetch($sourceFile);
 else
 $this->smarty->display($sourceFile);
 }
}

```

2. Теперь нам нужно подключить обработчик шаблонов к приложению. В файле `protected/config/main.php` нужно переопределить компонент `viewRenderer`:

```

...
// application components

```

```

 'components'=>array(
 ...
 'viewRenderer'=>array(
 'class'=>'ext.smarty.ESmartyViewRenderer',
),
),
 ...
)

```

3. Теперь опробуем получившееся. Создайте `protected/controllers/SmartyController.php`:

```

<?php
class SmartyController extends Controller
{
 function actionNative()
 {
 $this->render('native', array(
 'username' => 'Alexander',
));
 }

 function actionSmarty()
 {
 $this->render('smarty', array(
 'username' => 'Alexander',
));
 }
}

```

4. Теперь нам нужны view. Файл `protected/views/smarty/native.php`:

```

Hello, <?php echo $username?>!
и protected/views/smarty/smarty.tpl:
Hello, {$username}!

```

5. Теперь опробуйте действия контроллера. В обоих случаях вы должны получить:

**Hello, Alexander!**

## Как это работает

Обработчик шаблонов наследуется от `CApplicationComponent`, который реализует интерфейс `IViewRenderer` с единственным методом `renderFile`:

```

/**
 * Создаёт файл представления.

```

```
* @param CBaseController $context контроллер или виджет,
* который отображает view
* @param string $file путь к файлу view
* @param mixed $data данные, передаваемые view
* @param boolean $return возвращать ли результат
* @return mixed возвращаемый результат, или null, если
* возвращать результат не требуется
*/
public function renderFile($context,$file,$data,$return);
```

Таким образом, мы получаем контекст, путь к шаблону, данные и флаг, который определяет, нужно ли непосредственно вывести результат обработки или вернуть его. В нашем случае собственно обработка осуществляется обработчиком шаблонов Smarty, поэтому его нужно правильно инициализировать и вызвать соответствующие методы для обработки шаблонов.

Загрузка и инициализация Smarty довольно сложны:

```
Yii::import('application.vendors.smarty.*');

spl_autoload_unregister(array('YiiBase','autoload'));
require_once('Smarty.class.php');
spl_autoload_register(array('YiiBase','autoload'));

$this->smarty = new Smarty();

$this->smarty->template_dir = '';
$compileDir = Yii::app()->getRuntimePath().'smarty/compiled/';

if(!file_exists($compileDir)){
 mkdir($compileDir, $this->filePermission, true);
}

$this->smarty->compile_dir = $compileDir;
$this->smarty->assign('Yii', Yii::app());
```

Сначала мы с помощью `Yii::import` сообщаем Yii, что хотим использовать классы из `protected/vendors/smarty/`. Поскольку автоматический загрузчик классов Yii конфликтует с тем, который включен в Smarty, и мы не можем контролировать то, что делает Smarty, то приходится отключить загрузчик Yii, подключить загрузчик Smarty и снова подключить загрузчик Yii. Так как расположение директории шаблонов Yii может быть разным, то мы задаем пустую строку в опции Smarty по умолчанию. Временные файлы при работе с Yii правильно помещать в директорию приложения runtime. Вот почему мы задаем для директории компиляции, в которой Smarty хранит свои шаблоны,

преобразованные в PHP, значение `runtime/smarty/compiled`. Чтобы не слишком тревожить разработчиков, мы создаем директорию автоматически, если ее нет. Также мы создаем специальную переменную с именем `Yii` для шаблона `Smarty`. Она указывает на `Yii::app()` и позволяет получать свойства приложения изнутри шаблона.

Сама обработка шаблона несколько проще:

```
// свойства текущего контроллера доступны через {this.property}
$data['this'] = $context;

if(!is_file($sourceFile) || ($file=realpath($sourceFile))===false)
 throw new CException(Yii::t('ext','View file
"$sourceFile" does not exist.', array('{file}'=>$sourceFile)));

$this->smarty->assign($data);

if($return)
 return $this->smarty->fetch($sourceFile);
else
 $this->smarty->display($sourceFile);
```

Мы создаем переменную `Smarty` с именем `this`, которая позволит получать свойства контроллера, например заголовок страницы.

Чтобы упростить отладку, шаблон проверяется на существование, и, если он не существует, выводится сообщение об ошибке с именем шаблона.

Все данные, задаваемые через `$this->render`, передаются шаблону `Smarty` как есть. Потом мы либо создаем шаблон, либо возвращаем его, в зависимости от параметра `$return`.

## И ещё

Вы можете взять готовый обработчик шаблонов с дополнениями и поддержкой настройки по адресу <http://www.yiiframework.com/extension/smarty-view-renderer>.

Использовать свой обработчик шаблонов в своем приложении – нормально. Однако при создании расширений для повторного использования постарайтесь использовать только шаблоны PHP.

## Полезные материалы

Чтобы больше узнать о `Smarty` и об обработке шаблонов, обращайтесь по адресам:

- <http://www.smarty.net/>;

- <http://www.yiiframework.com/doc/api/IViewRenderer/>;
- <http://www.yiiframework.com/doc/api/CViewRenderer/>;
- <http://www.yiiframework.com/doc/api/CPradoViewRenderer/>.

## **Смотрите также**

- Рецепт «Подготовка расширений к публикации» в этой главе

# **Подготовка расширений к публикации**

В этой главе вы узнали, как создавать различные типы расширений Yii. Теперь обсудим, как можно поделиться результатами с другими и почему это важно.

## **Подготовка**

Подготовим контрольный список для хорошего расширения. Хороший программный продукт должен соответствовать следующим требованиям:

- последовательный, легкий для чтения и для использования API;
- хорошая документация;
- разработчик должен как-то найти ваше расширение;
- расширение должно покрывать наиболее типичные случаи;
- расширение должно развиваться;
- протестированный код, в идеале с модульными тестами;
- поддержка.

Разумеется, это требует много усилий, но все это необходимо для создания хорошего продукта.

## **Как это делается**

Просмотрим наш список более подробно, начиная с API. API должен быть последовательным, легким для чтения и применения. Последовательность означает общий стиль. По возможности не стоит смешивать `camelCasedVariableNames` и `underscored_variable_names`. Должна быть общая логика в именовании объектов и т. д. Все должно подчиняться определенным вами правилам. Это позволяет реже обращаться к документации и больше внимания уделять собственно написанию программ.



Код без документации почти бесполезен. Исключением может быть относительно простой код, но даже если он состоит из нескольких строк, то как-то неправильно, когда не написано ни слова о том, как его устанавливать и использовать. Что должна включать в себя хорошая документация?

Назначение кода и его сильные стороны. Эти два пункта должны быть как можно более понятны, о них должно быть написано четко и ясно. Например, если расширение отправляет email, в описании должно быть что-то вроде:

*EMailer позволяет отправить email простым и удобным способом. Поддерживаются различные языки в тексте и заголовке. Кроме того, в качестве шаблона письма вы можете использовать обычные view.*

Описание расширения, как и документацию к нему, лучше всего продублировать на английском. Так вы дадите возможность воспользоваться вашим кодом программистам со всего мира.

Код бесполезен, если разработчики не знают, куда его поместить и как конфигурировать. Не рассчитывайте на то, что им известны детали реализации или даже использования фреймворка. Инструкция по установке должна быть подробной. Большинство разработчиков предпочитают пошаговое руководство. Если для выполнения кода нужны какие-либо таблицы в БД, приложите их к коду в виде SQL или миграций.

Даже если методы и свойства API названы удачно, их все равно нужно документировать с помощью phpDoc, указывая типы аргументов и возвращаемых значений, давая краткое описание каждого метода. Не забывайте о private- и protected-методах и свойствах, поскольку иногда необходимо изучить их для полного понимания работы кода. Обдумайте включение public-методов и свойств в документацию, чтобы она могла служить справочником.

Приведите примеры использования с комментариями. Постарайтесь рассмотреть все типичные случаи. В примере не пытайтесь рассматривать много проблем сразу, чтобы не запутать читателей.

Расширение должно иметь номер версии и список изменений. Это поможет сообществу узнать, какая версия самая свежая, и понять, что даст обновление.

Гибкость кода важна, так как делает его подходящим для множества разных случаев. Но, поскольку нельзя решить абсолютно все проблемы, сосредоточьтесь на самых распространенных.

Важно, чтобы пользоваться расширением было удобно. Хорошая документация – первое условие. Второе – продемонстрировать, что код работает так, как задумано, и будет работать в дальнейшем после обновлений. Лучший способ – предоставить модульные тесты.

Расширение должно поддерживаться. По крайней мере, пока не станет стабильным, не перестанут появляться запросы дополнительных функций и сообщения об ошибках. Ждите вопросов и сообщений об ошибках. Заранее выделите время для доработки кода. Если у вас недостаточно времени для полноценной поддержки расширения, но оно содержит хорошие идеи, которые ещё никем не реализованы, все же стоит показать расширение сообществу. Если расширение будет хорошо воспринято, кто-нибудь наверняка поможет с поддержкой.

Наконец, нужно сделать расширение доступным. Обычное место для расширений Yii – <http://www.yiiframework.com/extensions/>. Чтобы разместить свой код, необходимо быть зарегистрированным участником форума и иметь на нем несколько сообщений. Постарайтесь придумать хорошее «говорящее» название, написать содержательное резюме и указать подходящие категорию и теги. Не используйте форматы архивов, отличные от ZIP и GZIP, поскольку эти два формата наиболее распространены, и можно быть уверенным, что любой сможет распаковать их.

Даже если ваше расширение относительно простое и хорошо документировано, могут возникнуть вопросы, и сначала ответить на них сможет только один человек – автор. Обычно вопросы задают на официальных форумах, поэтому лучше создать тему, в которой люди смогут обсудить ваш код, и в ней же сослаться на страницу с расширением.

## **Как это работает**

Если вы хотите поделиться расширением с сообществом и уверены, что расширение будет полезным и популярным, одного лишь кода недостаточно. Чтобы подготовить расширение к публикации, нужно приложить больше усилий. Возможно, много больше, чем затрачено на собственно написание кода. Так почему вообще следует делиться расширением с сообществом, если это так сложно?

В сравнении с использованием своего кода только в своих проектах открытие исходников имеет определенные преимущества. Вы привлекаете к тестированию гораздо больше людей, чем в принципе может быть в закрытом проекте. Программисты используют ваше

расширение, тестируют его, присылают отзывы, сообщают об ошибках. Если код становится популярным, появятся разработчики-энтузиасты, которые попытаются улучшить ваш код, сделать его более гибким, более стабильным, более удобным и пригодным повторного применения. В конце концов, сделать для сообщества что-то хорошее просто приятно.

## ***И ещё***

Мы рассмотрели наиболее важные моменты. Но есть ещё множество тонкостей, на которые стоит обратить внимание. Изучите существующие расширения до того, как начнёте писать своё. Если какое-то расширение почти подходит, попробуйте связаться с автором и поделиться с ним своими идеями. Изучение чужого кода поможет найти полезные хитрости, понять, что следует и чего не следует делать. Проверяйте время от времени статьи в wiki и официальный форум. Там размещена масса полезной информации о создании расширений и вообще о разработке с использованием Yii.



## **ГЛАВА 9.**

# **Обработка ошибок, отладка и журналирование**

В этой главе мы рассмотрим:

- Использование различных маршрутов для журналов.
- Анализ трассировки стека при ошибках.
- Журналирование и использование контекстной информации.
- Реализация собственного умного обработчика кода 404.

## **Введение**

Невозможно создать приложение без ошибок, если оно относительно сложное, поэтому разработчикам приходится отлавливать ошибки и исправлять их как можно быстрее. В Yii хороший набор классов для обработки журналов и ошибок. Кроме того, в отладочном режиме Yii при ошибке выдает трассировку стека. С ее использованием ошибки можно исправлять намного быстрее.

В этой главе мы рассмотрим журналирование, анализ трассировки стека при исключениях и разработку собственного обработчика ошибок.

## **Использование различных маршрутов для журналов**

Журналирование является ключом к пониманию того, как на самом деле выполняется приложение в тех случаях, когда нет возможности его отладить. Хотите верить, хотите нет, но даже когда вы полностью уверены, что приложение делает только то, что вами задумано, при интенсивном использовании оно будет делать многие вещи, о которых вы даже не догадывались. Это нормально, потому что никто не





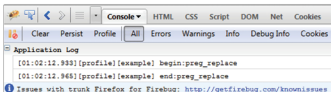
3. Выполните предыдущий шаг несколько раз. На экране вы должны увидеть веб-журнал, приблизительно как на следующем скриншоте:

done

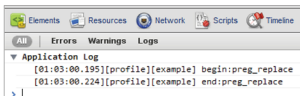
| Application Log |         |          |                                                                                                                                      |  |
|-----------------|---------|----------|--------------------------------------------------------------------------------------------------------------------------------------|--|
| Timestamp       | Level   | Category | Message                                                                                                                              |  |
| 00:34:21.960349 | trace   | example  | example trace message<br>in w:\home\yii\cmf\www\protected\controllers\LogController.php (6)<br>in w:\home\yii\cmf\www\index.php (12) |  |
| 00:34:21.960485 | info    | example  | Info<br>in w:\home\yii\cmf\www\protected\controllers\LogController.php (8)<br>in w:\home\yii\cmf\www\index.php (12)                  |  |
| 00:34:21.960575 | error   | example  | error<br>in w:\home\yii\cmf\www\protected\controllers\LogController.php (9)<br>in w:\home\yii\cmf\www\index.php (12)                 |  |
| 00:34:21.960663 | trace   | example  | trace<br>in w:\home\yii\cmf\www\protected\controllers\LogController.php (10)<br>in w:\home\yii\cmf\www\index.php (12)                |  |
| 00:34:21.960751 | warning | example  | warning<br>in w:\home\yii\cmf\www\protected\controllers\LogController.php (11)<br>in w:\home\yii\cmf\www\index.php (12)              |  |
| 00:34:21.960793 | profile | example  | begin:preg_replace                                                                                                                   |  |
| 00:34:22.006820 | profile | example  | end:preg_replace                                                                                                                     |  |

Веб-журнал содержит все сообщения, которые мы поместили в журнал, а также трассировки стека, отметки времени, уровни важности и категории.

4. Теперь откройте в Firefox консоль Firebug. Вы должны увидеть сообщения профилировщика, как на следующем скриншоте:



Yii использует API, совместимый с Firebug, поэтому вы можете увидеть эти сообщения в Chrome, как здесь:



Или в Opera:

|                                                   |  |             |   |                                                        |
|---------------------------------------------------|--|-------------|---|--------------------------------------------------------|
|                                                   |  | console.log | 6 | [00:59:00.368][profile][example]<br>begin:preg_replace |
|                                                   |  | console.log | 7 | [00:59:00.402][profile][example]<br>end:preg_replace   |
| console.log                                       |  |             |   |                                                        |
| [00:59:00.402][profile][example] end:preg_replace |  |             |   |                                                        |

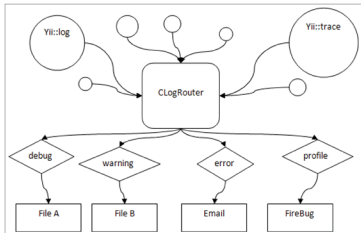
5. Поскольку мы изменили только имена файлов журналов, но не пути, то вы должны увидеть в директории `protected/runtime` файлы журналов с именами А и В. В файлах вы найдете сообщения следующего вида:

```
2011/04/17 00:58:53 [warning] [example] warning
in W:\home\yiiicmf\www\protected\controllers\
LogController.php (11)
in W:\home\yiiicmf\www\index.php (12)
2011/04/17 00:59:00 [warning] [example] warning
in W:\home\yiiicmf\www\protected\controllers\
LogController.php (11)
in W:\home\yiiicmf\www\index.php (12)
2011/04/17 00:59:56 [warning] [example] warning
in W:\home\yiiicmf\www\protected\controllers\
LogController.php (11)
in W:\home\yiiicmf\www\index.php (12)
2011/04/17 01:02:07 [warning] [example] warning
in W:\home\yiiicmf\www\protected\controllers\
LogController.php (11)
in W:\home\yiiicmf\www\index.php (12)
2011/04/17 01:02:12 [warning] [example] warning
in W:\home\yiiicmf\www\protected\controllers\
LogController.php (11)
in W:\home\yiiicmf\www\index.php (12)
2011/04/17 01:03:00 [warning] [example] warning
in W:\home\yiiicmf\www\protected\controllers\
LogController.php (11)
in W:\home\yiiicmf\www\index.php (12)
```

## Как это работает

Когда сообщение отправляется в журнал с помощью `Yii::log` или `Yii::trace`, то `Yii` передает сообщение маршрутизатору журналов. В зависимости от его настроек сообщения будут направлены по одному или нескольким маршрутам. Например, ошибки будут отосланы по электронной почте, отладочная информация записана в файл А, предупреждения записаны в файл В, результаты профилирования выведены на консоль Firebug. Это отражено на следующей диаграмме:





`CLogRouter` обычно подключен к компоненту приложения с именем `log`. Поэтому, чтобы настроить его, нужно изменить его свойства в `protected/config/main.php`, в разделе `components`. Единственное изменяемое свойство здесь – это `routes`, которое содержит массив из обработчиков логов и их настроек.

Мы определили пять маршрутов журналирования. Рассмотрим их следующим образом:

```
array(
 'class' => 'CEmailLogRoute',
 'categories' => 'example',
 'levels' => CLogger::LEVEL_ERROR,
 'emails' => array('admin@example.com'),
 'sentFrom' => 'log@example.com',
 'subject' => 'Error at example.com',
),
```

`CEmailLogRoute` отправляет сообщения об ошибках по электронной почте. Ограничимся категорией `example` и уровнем `error`. Письма будут отправляться с адреса `log@example.com` на адрес `admin@example.com` с темой «Error at example.com».

```
array(
 'class' => 'CFileLogRoute',
 'levels' => CLogger::LEVEL_WARNING,
 'logFile' => 'A',
),
```

CFileLogRoute добавляет сообщения об ошибках в указанный файл. Ограничим сообщения уровнем `warning` и используем файл `A`. Таким же образом направим сообщения уровня `info` в файл `B`.

CWebLogRoute направляет сообщения в браузер следующим образом:

```
array(
 'class' => 'CWebLogRoute',
 'categories' => 'example',
),
```

Тот же маршрут можно использовать для отправки сообщений в Firebug или другую совместимую с ним консоль.

```
array(
 'class' => 'CWebLogRoute',
 'categories' => 'example',
 'levels' => CLogger::LEVEL_PROFILE,
 'showInFireBug' => true,
 'ignoreAjaxInFireBug' => true,
),
```

В коде, приведенном выше, категория ограничена `example`, уровень – `profile`, включено журналирование в Firebug. Кроме того, мы отключили журналирование для AJAX-запросов, так как ответы в JSON могут быть испорчены журналированием.

## И ещё

Есть ещё интересные вещи, касающиеся журналирования в Yii, которые рассматриваются в следующем подразделе:

### Yii::trace и Yii::log

Yii::trace является простой оберткой вокруг Yii::log:

```
if (YII_DEBUG)
 self::log($msg, CLogger::LEVEL_TRACE, $category);
```

Yii::trace отсылает сообщения с уровнем трассировки, только когда Yii работает в режиме отладки.

### Yii::beginProfile и Yii::endProfile

Эти методы используются для измерения времени выполнения некоторой части кода приложения. В нашем LogController мы измеряем 10 000 выполнений `preg_replace` следующим образом:

```
Yii::beginProfile('preg_replace', 'regex_test');
for($i=0;$i<10000;$i++){
```

```
preg_replace('~^[a-z]+~', '', 'test it');
}
Yii::endProfile('preg_replace', 'regex_test');
```

### Немедленное журналирование сообщений

По умолчанию Yii держит сообщения в оперативной памяти, пока приложение не завершится корректным способом с помощью `Yii::app()->end()`. Это делается по причинам, связанным с производительностью, и в целом работает удовлетворительно. Тем не менее если происходит фатальная ошибка PHP или вызов `die()/exit()` в коде, то некоторые сообщения не будут записаны. Чтобы гарантировать журналирование сообщений, можно сделать это принудительно с помощью `Yii::app()->log->flush(true)`.

### Полезные материалы

Чтобы больше узнать о журналировании, обратитесь по следующим адресам:

- <http://www.yiiframework.com/doc/guide/ru/topics/logging>;
- <http://www.yiiframework.com/doc/api/CLogRouter>;
- <http://www.yiiframework.com/doc/api/CLogRoute>;
- <http://www.yiiframework.com/doc/api/CDbLogRoute>;
- <http://www.yiiframework.com/doc/api/CEmailLogRoute>;
- <http://www.yiiframework.com/doc/api/CFileLogRoute>;
- <http://www.yiiframework.com/doc/api/CWebLogRoute>;
- <http://www.yiiframework.com/doc/api/CLogger>.

### Смотрите также

- Рецепт «Журналирование и использование контекстной информации» в этой главе.

## Анализ трассировки стека при ошибках

Когда происходит ошибка, Yii может выдать вместе с сообщением об ошибке трассировку стека. Она особенно полезна, когда нужно узнать, что конкретно вызвало ошибку, а не просто факт, что ошибка произошла.

## Подготовка

- Создайте новое приложение Yii с помощью `yiiic webapp`, как это описано в официальном руководстве.
- Настройте базу данных и импортируйте следующую команду SQL:

```
CREATE TABLE 'article' (
 'alias' varchar(255) NOT NULL,
 'title' varchar(255) NOT NULL,
 'text' text NOT NULL,
 PRIMARY KEY ('alias')
);
```

- Создайте модель `Article` с помощью Gii.

## Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/controllers/ErrorController.php` следующего содержания:

```
<?php
class ErrorController extends CController
{
 public function actionIndex()
 {
 $articles = $this->getModels('php');
 foreach($articles as $article)
 {
 echo $article->title;
 echo "
";
 }
 }

 private function getModels($alias)
 {
 $criteria = new CDbCriteria();
 $criteria->addSearchCondition('alias', $alias);
 return Article::model()->findAll($criteria);
 }
}
```

2. Выполним приведенное выше действие и получим следующую ошибку:

```
CDbCommand failed to execute the SQL statement:
SQLSTATE[42S22]:
```

```
Column not found: 1054 Unknown column 'allas' in
'where clause'.
The SQL statement executed was: SELECT * FROM
'article' 't' WHERE
allas LIKE :ycp0
```

3. Кроме того, трассировка стека показывает нам следующее:

#### Stack Trace

```
#0 W:\home\yii\framework\db\CdbCommand.php(376): CdbCommand->queryInternal("FetchAll", array(2), array())
#1 W:\home\yii\framework\db\ar\ActiveRecord.php(1288): CdbCommand->queryAll()
#2 W:\home\yii\framework\db\ar\ActiveRecord.php(1412): ActiveRecord->query(CdbCriteria, true)
#3 W:\home\yii\framework\web\controllers\ErrorHandler.php(18): ActiveRecord->findAll(CdbCriteria)

13 private function getModels($alias)
14 {
15 $criteria = new CdbCriteria();
16 $criteria->addSearchCondition('allas', $alias);
17 return Article::model()->findAll($criteria);
18 }
19 }
20 }

#4 W:\home\yii\framework\web\controllers\ErrorHandler.php(6): ErrorHandler->getModels('php')

01 <?php
02 class ErrorHandler extends CController
03 {
04 public function actionIndex()
05 {
06 $articles = $this->getModels('php');
07 foreach($articles as $article)
08 {
09 echo $article->title;
10 echo "
";
11 }
12 }
13 }

#5 W:\home\yii\framework\web\actions\CInlineAction.php(50): ErrorHandler->actionIndex()
#6 W:\home\yii\framework\web\CController.php(302): CInlineAction->runWithParams(array())
#7 W:\home\yii\framework\web\CController.php(280): CController->runAction(CInlineAction)
#8 W:\home\yii\framework\web\CController.php(258): CController->runActionWithFilters(CInlineAction, array())
#9 W:\home\yii\framework\web\CWebApplication.php(329): CController->run("")
#10 W:\home\yii\framework\web\CWebApplication.php(122): CWebApplication->runController("error")
#11 W:\home\yii\framework\base\CApplication.php(155): CWebApplication->processRequest()
#12 W:\home\yii\framework\index.php(12): CApplication->run()

07 defined('YII_DEBUG') or define('YII_DEBUG',true);
```

## Как это работает

Из сообщения об ошибке мы узнаем, что где-то в коде мы использовали столбец `allas`, которого нет в базе данных. В нашем случае очень просто найти это место, просмотрев файлы проекта, но в большом проекте имя столбца может быть значением переменной. Кроме того,

у нас есть все, что нужно, чтобы исправить ошибку, не отрываясь от экрана, на который выведена трассировка стека. Надо только внимательно прочитать ее.

Трассировка стека показывает цепочку вызовов в обратном порядке, начиная с того, который вызвал ошибку. В нашем случае это `queryInternal`. В принципе, нет необходимости читать всю трассировку, чтобы понять происходящее. Код платформы хорошо протестирован, возможность ошибки в нем невелика. Поэтому пункты трассировки, относящиеся к приложению, развернуты, а пункты, относящиеся к платформе, свернуты.

Таким образом, мы берем первую развернутую секцию и ищем `allas`. После того как мы нашли ее, сразу можно сказать, что значение использовано в `ErrorController.php` в строке 17.

## И ещё

Чтобы больше узнать об обработке ошибок, обратитесь по следующему адресу:

- <http://www.yiiframework.com/doc/guide/ru/topics.error>.

## Смотрите также

- Рецепт «Журналирование и использование контекстной информации» в этой главе.

# Журналирование и использование контекстной информации

Иногда недостаточно сообщения об ошибке, чтобы исправить ее. Например, если вы следуете лучшим практикам и разрабатываете и тестируете приложение, включив все возможные уровни ошибок, вы можете получить сообщение об ошибке. При этом без контекста выполнения вы узнаете лишь о том, что ошибка произошла, а не о том, что именно ее вызвало.

В нашем примере мы используем очень простое и небрежно написанное действие, которое всего лишь пишет: «**Hello**, <имя пользователя>!», где <имя пользователя> взято непосредственно из `$_GET`.

## Подготовка

- Настройте PHP для наиболее подробной диагностики ошибок. В `php.ini` замените значение `error_reporting` на «-1». После этого изменения необходимо перезапустить сервер.
- Создайте новое приложение Yii с помощью `yiiic webapp`, как это описано в официальном руководстве.

## Как это делается

Выполните следующие шаги:

1. Нам понадобится контроллер. Создайте файл `protected/controllers/LogController.php` следующего содержания:

```
<?php
class LogController extends Ccontroller
{
 public function actionIndex()
 {
 echo "Hello, ".$_GET['username'];
 }
}
```

2. Теперь, если мы выполним действие `index`, то получим сообщение об ошибке «Undefined index: username» («Не определен индекс: username»). Настроим ведение журнала таким образом, чтобы этот тип ошибок выводился в файл `protected/config/main.php`:

```
protected/config/main.php:
array(
 ...
 'preload'=>array('log'),
 'components'=>array(
 ...
 'log'=>array(
 'class'=>'CLogRouter',
 'routes'=>array(
 array(
 'class'=>'CFileLogRoute',
 'levels'=>'error',
 'logFile' => 'errors',
),
),
),
),
),
),
)
```

3. Выполните действие index снова и проверьте файл `protected/runtime/errors`. В нем должна быть журнальная информация следующего вида:

```
2011/04/17 03:53:19 [error] [php] Undefined index:
username (W:
\home\yiicmf\www\protected\controllers\
LogController.php:30)
Stack trace:
#0 W:\home\yiicmf\framework\web\CController.php(280):
LogController->runAction()
#1 W:\home\yiicmf\framework\web\CController.php(258):
LogController->runActionWithFilters()
#2 W:\home\yiicmf\framework\web\CWebApplication.php(329):
LogController->run()
#3 W:\home\yiicmf\framework\web\CWebApplication.php(122):
CWebApplication->runController()
#4 W:\home\yiicmf\framework\base\CApplication.php(155):
CWebApplication->processRequest()
#5 W:\home\yiicmf\www\index.php(12): CWebApplication->run()
REQUEST_URI=/log/environment
in W:\home\yiicmf\www\protected\controllers\
LogController.php (30)
in W:\home\yiicmf\www\index.php (12)
```

4. Теперь мы можем отдать наше приложение команде тестировщиков и время от времени читать журналы. Мы будем знать, что ошибки происходят, но нам нужно будет иметь возможность как-то воспроизводить их. Чтобы сделать это, нам нужно воссоздать окружение. Добавим необходимую информацию в журнал следующим образом:

```
array(
 'class' => 'CFileLogRoute',
 'levels' => CLogger::LEVEL_ERROR,
 'logFile' => 'errors',
 'filter'=>'CLogFilter',
),
```

5. Теперь выполним действие снова. На этот раз информации должно быть достаточно для детального воссоздания окружения:

```
2011/04/17 04:01:16 [info] [application]
$_SERVER=array (
 'REDIRECT_STATUS' => '200',
 'HTTP_USER_AGENT' => 'Opera/9.80 (Windows NT 6.1;
U; ru)
```



```

Presto/2.8.131 Version/11.10',
'HTTP_HOST' => 'yiicmf',
'HTTP_ACCEPT' => 'text/html, application/
xml;q=0.9, application/
xhtml+xml, image/png, image/webp, image/jpeg, image/gif,
image/x-xbitmap, */*;q=0.1',
'HTTP_ACCEPT_LANGUAGE' => 'ru-
RU,ru;q=0.9,en;q=0.8',
'HTTP_ACCEPT_ENCODING' => 'gzip, deflate',
'HTTP_PRAGMA' => 'no-cache',
'HTTP_CACHE_CONTROL' => 'no-cache',
'HTTP_CONNECTION' => 'Keep-Alive',

...

'SERVER_NAME' => 'yiicmf',
'SERVER_ADDR' => '127.0.0.1',
'SERVER_PORT' => '80',
'REMOTE_ADDR' => '127.0.0.1',
'DOCUMENT_ROOT' => 'W:/home/yiicmf/www',
'SERVER_ADMIN' => 'admin@localhost',
'SCRIPT_FILENAME' => 'W:/home/yiicmf/www/index.php',
'REMOTE_PORT' => '55190',
'REDIRECT_URL' => '/log/environment',
'GATEWAY_INTERFACE' => 'CGI/1.1',
'SERVER_PROTOCOL' => 'HTTP/1.1',
'REQUEST_METHOD' => 'GET',
'QUERY_STRING' => '',
'REQUEST_URI' => '/log/environment',
'SCRIPT_NAME' => '/index.php',
'PHP_SELF' => '/index.php',
'REQUEST_TIME' => 1302998476,
'argv' =>
array (
),
'argc' => 0,
)
2011/04/17 04:01:16 [error] [php] Undefined index:
username (W:\
home\yiicmf\www\protected\controllers\
LogController.php:30)
Stack trace:
#0 W:\home\yiicmf\framework\web\CController.php(280):
LogController->runAction()
#1 W:\home\yiicmf\framework\web\CController.php(258):
LogController->runActionWithFilters()
#2 W:\home\yiicmf\framework\web\CWebApplication.php(329):
LogController->run()

```

```
#3 W:\home\yiicmf\framework\web\CWebApplication.php(122):
CWebApplication->runController()
#4 W:\home\yiicmf\framework\base\CApplication.php(155):
CWebApplication->processRequest()
#5 W:\home\yiicmf\www\index.php(12): CWebApplication->run()
REQUEST_URI=/log/environment
in W:\home\yiicmf\www\protected\controllers\
LogController.php (30)
in W:\home\yiicmf\www\index.php (12)
2011/04/17 04:01:16 [info] [application] User: Guest (ID:)
```

## Как это работает

Чтобы работать с журналированием ещё эффективнее, мы используем класс `CLogFilter`, который выполняет предварительную обработку сообщений до того, как они будут направлены по маршрутам журналирования. Это единственный фильтр журналов, который включен в ядро Yii и который можно использовать для добавления информации о контексте и окружении. Если мы журналируем сообщение вручную, то мы предположительно знаем, какая информация нам нужна, поэтому можем включить некоторые опции `CLogFilter`, чтобы записывалось лишь необходимое:

```
array(
 'class' => 'CFileLogRoute',
 'levels' => CLogger::LEVEL_ERROR,
 'logFile' => 'errors',
 'filter' => array(
 'class' => 'CLogFilter',
 'logUser' => false,
 'logVars' => array('_GET'),
),
)
```

Приведенный выше код будет журналировать ошибки в файл с именем `errors`. Кроме самого сообщения, будет записано содержимое `$_GET`, если оно не пусто.

## И ещё

Чтобы больше узнать о фильтрах журналов и контекстной информации, обратитесь по следующим адресам:

- <http://www.yiiframework.com/doc/api/CLogFilter>;
- <http://www.yiiframework.com/doc/guide/ru/topics.logging#logging-context-informations>.

## Смотрите также

- Рецепт «Использование различных маршрутов для журналов» в этой главе.

## Реализация собственного умного обработчика кода 404

В Yii обработка ошибок очень гибкая, поэтому вы можете создать собственный обработчик ошибок определенных типов. В этом рецепте мы будем обрабатывать ошибку 404 («не найдено») умным способом: мы будем отдавать специальную страницу 404 с подсказкой, основанной на том, что было введено в адресной строке.

### Подготовка

- Создайте новое приложение Yii с помощью `yiiic webapp`, как описано в официальном руководстве.
- Настройте базу данных и импортируйте следующую SQL-команду:

```
CREATE TABLE 'article' (
 'alias' varchar(255) NOT NULL,
 'title' varchar(255) NOT NULL,
 'text' text NOT NULL,
 PRIMARY KEY ('alias')
);
```

- Создайте модель Article с помощью Gii.
- Создайте CRUD Article с помощью Gii.
- Создайте следующие статьи:

| Заголовок                         | Псевдоним              | Текст                                    |
|-----------------------------------|------------------------|------------------------------------------|
| Фреймворк Yii                     | yii-framework          | Фреймворк Yii хорош.                     |
| Почему Yii                        | why-yii                | Почему мне следует использовать Yii?     |
| PHP – хороший инструмент          | php-is-a-good-tool     | PHP замечателен!                         |
| Yii как инструмент для приложений | yii-as-a-tool-for-apps | Yii – хороший инструмент для приложений. |

## Как это делается

Выполните следующие шаги:

1. Попробуйте выполнить `http://your.application/yii`. Вы должны получить стандартную страницу с ошибкой 404. Теперь нам нужно как-то изменить эту страницу, но она должна остаться прежней для других типов ошибок. Чтобы этого добиться, мы используем событие приложения `onException`. Настроим его для обработки с помощью статического метода `handle` класса `NotFoundHandler`. Мы сделаем это, используя `protected/config/main.php` следующим образом:

```
// события
'onException' => array('NotFoundHandler', 'handle'),
```

2. Теперь нам нужно реализовать сам обработчик ошибок. Создайте `protected/components/NotFoundHandler.php` со следующим содержанием:

```
<?php
class NotFoundHandler
{
 public static function handle(CExceptionEvent $event)
 {
 $exception = $event->exception;

 if($exception instanceof CHttpException &&
 $exception->statusCode===404)
 {
 $pathParts = explode('/', Yii::app()->
 getRequest()->getRequestUri());
 $pathPart = array_pop($pathParts);

 $criteria = new CDbCriteria();
 $criteria->addSearchCondition('alias', $pathPart);
 $criteria->limit = 5;

 $models = Article::model()->findAll($criteria);
 $controller = new CController(null);
 $controller->renderPartial('/error/404', array(
 'models' => $models,
));
 $event->handled = true;
 }
 }
}
```

3. Также нам нужно представление `protected/views/error/404.php`:

```
<!doctype html>
<html>
 <head>
 <meta charset="utf-8" />
 <title>404</title>
 </head>
 <body>
 <h1>404</h1>

 <?php if(!empty($models)):?>
 <p>Возможно, вы ищете:</p>

 <?php foreach($models as $model):?>
 <a href="<?php echo $this->createUrl('article/view',
 array('id' => $model->alias))?"><?php echo
 $model->title?>
 <?php endforeach?>

 <?php endif?>
 </body>
</html>
```

4. Вот и все. Теперь попробуйте следующие адреса:

- <http://your.application/yi>;
- <http://your.application/tool>.

Каждый раз вы будете получать несколько ссылок на статьи по теме.

## Как это работает

Используя конфигурационный файл, мы прикрепляем обработчик событий к событию `onException` следующим образом:

```
'onException' => array('NotFoundHandler', 'handle'),
```

Это означает, что мы будем использовать `NotFoundHandler::handle()`. Каждый обработчик событий принимает один параметр по имени `$event` с данными события внутри. Параметр обработчика исключений принимает экземпляр `CExceptionEvent`. Поскольку он содержит оригинальное исключение, мы можем проверять его тип и код ошибки следующим образом:

```
if($exception instanceof CHttpException &&
 $exception->statusCode===404)
```

Если исключение не совпадает, то Yii продолжает работать как обычно, а если совпадает, то Yii выполняет наш специальный код:

```
$pathParts = explode('/', Yii::app()->getRequest()->
getRequestUri());
$pathPart = array_pop($pathParts);
```

Мы берём последний сегмент URL, поскольку он, скорее всего, содержит псевдоним статьи, если URL в формате пути. Потом формируем критерии базы данных и получаем модели следующим образом:

```
$criteria = new CDbCriteria();
$criteria->addSearchCondition('alias', $pathPart);
$criteria->limit = 5;
$models = Article::model()->findAll($criteria);
```

Далее визуализируем представление следующим образом:

```
$controller = new CController(null);
$controller->renderPartial('//error/404', array(
 'models' => $models,
));
```

Заметьте, что мы создаём новый экземпляр CController, так как код 404 может быть получен раньше, чем приложение инициализирует контроллер.

```
$event->handled = true;
```

Наконец, мы сообщаем Yii, что событие обрабатывается и нет необходимости обрабатывать его снова.

## И ещё

Представленный выше метод – не единственный для обработки ошибок особым образом. Другие возможности:

- расширить компонент приложения CErrorHandler;
- использовать действие контроллера для обработки ошибок изменением CErrorHandler::errorAction.

## Полезные материалы

Чтобы больше узнать об обработке ошибок в Yii, обратитесь к описанию API по следующим адресам:

- <http://www.yiiframework.com/doc/api/CErrHandler/>;
- <http://www.yiiframework.com/doc/api/CApplication#onException-detail>;



- <http://www.yiiframework.com/doc/api/CApplication#onError-detail>.

### ***Смотрите также***

- Рецепт «Использование событий Yii» в главе 1 «Под капотом».



## ГЛАВА 10.

# Безопасность

В этой главе мы рассмотрим:

- Использование фильтров контроллера.
- Использование CHtml и CHtmlPurifier для предотвращения XSS.
- Предотвращение SQL-инъекций.
- Предотвращение CSRF.
- Использование RBAC.

## Введение

Из этой главы вы узнаете, как поддерживать безопасность приложения в соответствии с основным принципом безопасности веб-приложений – «фильтровать ввод, экранировать вывод» (*filter input, escape output*). Мы рассмотрим такие темы, как создание собственных фильтров контроллера, предотвращение XSS, CSRF и SQL-инъекций, экранирование вывода и использование управления доступом на основе ролей.

## Использование фильтров контроллера

Во многих случаях нам нужно фильтровать входные данные или выполнять некоторые действия, основанные на этих данных. Например, специальными фильтрами можно фильтровать посетителей по IP, заставлять использовать HTTPS, перенаправлять пользователя на страницу с настройками, перед тем как он использует приложение. В Yii есть два встроенных фильтра, которые можно использовать. Первый – это CInlineFilter, который позволяет использовать метод контроллера в качестве фильтра, а второй (на нем мы сосредоточим-



ся) – `CAccessControlFilter`, который позволяет управлять доступом к различным действиям контроллера.

В этом рецепте мы реализуем следующее:

- предоставление доступа к действию контроллера только авторизованным пользователям;
- предоставление доступа к действию контроллера только с указанных IP-адресов;
- предоставление доступа только указанным пользователям;
- предоставление доступа только пользователям указанного браузера; в этом случае мы будем также выводить специальное сообщение.

## Подготовка

- Создайте новое приложение с помощью `yiiic webapp`.
- Создайте файл `protected/controllers/AccessController.php` со следующим содержанием:

```
<?php
class AccessController extends CController
{
 public function actionAuthOnly()
 {
 echo "Похоже, что вам разрешено выполнять это действие.";
 }

 public function actionIp()
 {
 echo "Вам повезло, ваш IP находится в нашем списке!";
 }

 public function actionUser()
 {
 echo "Вы тот, кто нам нужен. Добро пожаловать!";
 }
}
```

## Как это делается

Выполните следующие шаги:

1. Применение фильтра доступа состоит из двух этапов. На первом нам нужно включить фильтр в метод `filters` контроллера. Мы сделаем это так:

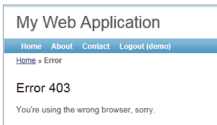
```
public function filters()
{
 return array(
```

```
 'accessControl',
);
}
```

2. Потом мы можем описать правила фильтрации в методе `accessRules`, который используется фильтром контроля доступа, следующим образом:

```
public function accessRules()
{
 return array(
 array(
 'deny',
 'expression' => 'strpos($_SERVER[\'HTTP_USER_AGENT\'],
 \'MSIE\') !== FALSE',
 'message' => "Вы используете неправильный
браузер, извините.",
),
 array(
 'allow',
 'actions' => array('authOnly'),
 'users' => array('@'),
),
 array(
 'allow',
 'actions' => array('ip'),
 'ips' => array('127.0.0.1'),
),
 array(
 'allow',
 'actions' => array('user'),
 'users' => array('admin'),
),
 array('deny'),
);
}
```

3. Теперь попробуйте выполнить действия контроллера, используя Internet Explorer и другие браузеры, от имени пользователей `admin` и `demo`.



## Как это работает

Начнем с предоставления доступа к действию контроллера только авторизованным пользователям. Добавьте следующий код в метод `accessRules`:

```
array(
 'allow',
 'actions' => array('authOnly'),
 'users' => array('@'),
),
array('deny'),
```

Каждый из массивов является правилом доступа. Вы можете использовать либо правило *allow* (разрешить), либо правило *deny* (запретить). У каждого правила несколько параметров.

По умолчанию Yii ничего не запрещает, поэтому добавляйте `array('deny')` в конце вашего списка правил, если вам нужна максимальная безопасность.

В нашем правиле мы используем два параметра. Первый – `actions`, который принимает массив действий, к которым правило применяется. Второй – `users`, который принимает массив идентификаторов пользователей (тех, которые возвращает `Yii::app()->user->id`), чтобы определить, к каким пользователям применяется правило. В нашем случае используется один из следующих специальных символов: `@` означает «все авторизованные пользователи», `*` означает «все пользователи», `?` означает «все пользователи-гости».

Правила выполняются, начиная с верхнего в списке, пока одно из них не удовлетворит условиям. Если ни одно правило не будет применено, действие считается разрешенным.

Следующая задача – предоставлять доступ только с указанных IP-адресов. В этом случае используются следующие два правила:

```
array(
 'allow',
 'actions' => array('ip'),
 'ips' => array('127.0.0.1'),
),
array('deny'),
```

Первое правило разрешает доступ к действию `ip` для IP-адресов, указанных в списке. В нашем случае используется петлевой адрес, который всегда указывает на наш компьютер. Попробуйте изменить его,

например на 127.0.0.2, чтобы увидеть, как это работает, когда адрес не совпадает. Второе правило запрещает все IP-адреса, не удовлетворяющие первой правилу.

Теперь предоставим доступ только одному указанному пользователю следующим образом:

```
array(
 'allow',
 'actions' => array('user'),
 'users' => array('admin'),
),
array('deny'),
```

Представленное выше правило разрешает пользователю с ID, равным admin, выполнять действие user. Таким образом, если вы авторизуетесь как admin, то получите доступ, но если вы авторизуетесь как demo, то, соответственно, не получите доступа. Это правило того же типа, как то, которое мы использовали, чтобы ограничить доступ множеством авторизованных пользователей. Единственное отличие – в том, что вместо подстановочного символа мы используем теперь ID. И снова второе правило также запрещает доступ, включая всех остальных пользователей.

Наконец, нам нужно запретить доступ определенному браузеру. В этом рецепте мы запрещаем все версии Internet Explorer, а также некоторые другие браузеры, использующие ту же строку агента пользователя. Правило помещено в начало списка и выполняется первым:

```
array(
 'deny',
 'expression' => 'strpos($_SERVER['HTTP_USER_AGENT'], \'MSIE\')
 !== FALSE',
 'message' => 'Вы используете неправильный браузер, извините.',
),
array('deny'),
```

Используемая здесь техника определения браузера не очень надежна, поскольку «MSIE» содержится в строках агента пользователя многих браузеров. Список возможных строк агента пользователя можно увидеть по адресу: <http://www.useragentstring.com/>.

В представленном выше коде мы используем ещё одно свойство правил фильтров, называемое выражение (*expression*). Оно принимает выражение PHP в качестве строки, в качестве анонимной функции (в PHP 5.3) или в качестве действующего обратного вызова. В нашем случае используется строка.

В PHP 5.3 анонимная функция выглядит примерно следующим образом:

```
array(
 'deny',
 'expression' => function(){
 return strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE;
 },
 'message' => "Вы используете неправильный браузер, извините.",
),
```

Приведённое выше выражение проверяет, содержит ли строка агента пользователя «MSIE». В зависимости от потребностей вы можете указать любой код PHP. Второй параметр с именем `message` используется для того, чтобы изменить сообщение, отображаемое пользователю, когда доступ запрещён.

## И ещё

Чтобы больше узнать об управлении доступом и фильтрах, обратитесь по следующим адресам:

- <http://www.yiiframework.com/doc/guide/ru/topics.auth#access-control-filter>;
- <http://www.yiiframework.com/doc/guide/ru/basics.controller#filter>;
- <http://www.yiiframework.com/doc/api/CAccessControlFilter>.

## Смотрите также

- Рецепт «Использование RBAC» в этой главе.

# Использование CHtml и CHtmlPurifier для предотвращения XSS

XSS расшифровывается как «межсайтовый скриптинг» (*cross-site scripting*) и является видом уязвимости, позволяющей внедрить скрипт со стороны клиента (обычно JavaScript) в страницу, которую видят другие пользователи. Учитывая возможности, предоставляемые скриптингом на стороне клиента, последствия могут быть очень серьезными, например обход проверок безопасности, получение чужих учетных данных, утечки информации.

В этом рецепте мы увидим, как предотвратить XSS с помощью экранирования вывода с помощью CHtml и CHtmlPurifier.

## Подготовка

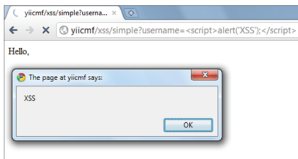
- Создайте новое приложение с помощью `yiic webapp`.
- Создайте файл `protected/controllers/XssController.php` со следующим содержанием:

```
<?php
class XssController extends CController
{
 public function actionSimple()
 {
 echo Привет, '$_GET['username'].!';
 }
}
```

Обычно используется примерно так: `/xss/simple?username=Alexander`. Тем не менее, поскольку главный принцип безопасности «фильтровать ввод, экранировать вывод» не был принят во внимание, злоумышленники смогут сделать нечто вроде:

```
/xss/simple?username=<script>alert('XSS');</script>
```

В результате при выполнении скрипта получится то, что изображено на следующем скриншоте:



Заметьте, что вместо вывода сообщения XSS можно, например, скопировать содержимое страницы или сделать нечто, специфичное для данной веб-страницы, например удалить данные пользователя.

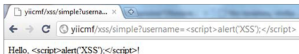
## Как это делается

Выполните следующие шаги:

1. Чтобы предотвратить вывод сообщения XSS, как показано на скриншоте выше, нужно экранировать данные до передачи в браузер. Мы сделаем это следующим образом:

```
class XssController extends CController
{
 public function actionSimple()
 {
 echo Привет, '.CHtml::encode($_GET['username']).'!';
 }
}
```

2. Теперь вместо предупреждения мы получим правильно экранированный код HTML, как на следующем скриншоте:



3. Таким образом, основное правило состоит в том, чтобы всегда экранировать динамические данные. Например, следует сделать то же самое с именем ссылки:

```
echo CHtml::link(CHtml::encode($_GET['username']), array());
```

4. Вот и все. Ваша страница свободна от XSS. А что, если мы хотим пропустить какой-то код HTML? Мы уже не можем использовать CHtml::encode, так как HTML отобразится как код, а мы хотим, чтобы он выполнялся. К счастью, в составе Yii есть инструмент, позволяющий отфильтровывать зловредный HTML. Он называется HTML Purifier (очиститель HTML) и может применяться следующим образом:

```
public function actionHtml()
{
 $this->beginWidget('CHtmlPurifier');
 echo $_GET['html'];
 $this->endWidget();
}
```

Другой вариант – использовать его так:

```
public function actionHtml()
{
```

```
$purifier=new CHtmlPurifier();
echo $purifier->purify($_GET['html']);
}
```

5. Теперь, если мы попытаемся получить доступ к действию `html`, используя URL типа `/xss/html?html=Привет, <strong>username</strong>!<script>alert('XSS')</script>`, HTML Purifier удалит зловредный фрагмент, и мы получим следующий результат:



## Как это работает

Изнутри CHtml::encode выглядит примерно так:

```
public static function encode($text)
{
 return htmlspecialchars($text, ENT_QUOTES, Yii::app()
->charset);
}
```

То есть в принципе мы используем внутреннюю функцию PHP `htmlspecialchars`, которая достаточно безопасна, если не забывать передавать правильную кодировку третьему аргументу.

CHtmlPurifier использует библиотеку HTML Purifier, которая является решением с наибольшими возможностями для предотвращения XSS внутри HTML. Мы использовали ее с настройками по умолчанию, этого достаточно для большей части данных, вводимых пользователями. Чтобы больше узнать о том, как настраивать HTML Purifier, обратитесь к адресам, приведенным в подразделе «Полезные материалы» в этом рецепте.

## И ещё

Есть вещи, которые также необходимо знать об XSS и HTML Purifier. Это тема следующего раздела:

### Типы XSS

Существуют два основных типа XSS-инъекций:

- 1) непостоянные (*non-persistent*);
- 2) постоянные (*persistent*).



Первый тип – это то, что использовалось в данном рецепте и что является наиболее распространенным типом XSS, который можно найти в самых небезопасных веб-приложениях. Данные, вводимые пользователем, нигде не сохраняются, поэтому инъецируемый скрипт выполняется только один раз и только для пользователя, который его ввел. Все же это не так безобидно, как может показаться. Злоумышленник может вставить XSS в ссылку на другую веб-страницу, и этот скрипт выполнится, когда другой пользователь перейдет по ссылке.

Второй тип гораздо серьезнее, поскольку данные, введенные злоумышленником, сохраняются в базе данных и могут быть показаны многим, или даже всем, посетителям веб-страницы. Используя XSS этого типа, можно буквально уничтожить сайт, заставив всех посетителей уничтожить все данные, к которым у них есть доступ.

## Настройка HTML Purifier

HTML Purifier можно настроить следующим образом:

```
$p = new CHtmlPurifier();
$p->options = array('URI.AllowedSchemes'=>array(
 'http' => true,
 'https' => true,
));
$text = $p->purify($text);
```

За списком всех возможных ключей, которые можно использовать в массиве `options`, обращайтесь по адресу: <http://htmlpurifier.org/live/configdoc/plain.html>.

## Производительность HTML Purifier

Поскольку HTML Purifier анализирует и обрабатывает много данных, его производительность не особенно велика. Поэтому правильно будет не обрабатывать текст каждый раз перед выводом. Вместо этого его можно сохранять в отдельном поле в базе данных, как это описано в главе 6 «База данных, Active record и трюки с моделями» в рецепте «Применение markdown и HTML».

## Полезные материалы

Чтобы больше узнать об XSS и о том, как с ним бороться, обратитесь к следующим ресурсам:

- <http://htmlpurifier.org/docs>;
- <http://hackers.org/xss.html>;
- <http://shiflett.org/blog/2007/may/character-encoding-and-xss>.

## Смотрите также

- Рецепт «Применение markdown и HTML» в главе 6.

# Предотвращение SQL-инъекций

SQL-инъекция – тип внедрения кода, который использует уязвимость на уровне базы данных и позволяет выполнить произвольные команды SQL, давая злоумышленнику возможность осуществить такие действия, как удаление данных или повышение своих привилегий.

В этом рецепте мы рассмотрим примеры уязвимого кода и исправим их.

## Подготовка

- Создайте новое приложение с помощью `yii webapp`.
- Создайте и настройте новую базу данных.
- Выполните следующую команду SQL:

```
CREATE TABLE 'user' (
 'id' int(11) unsigned NOT NULL AUTO_INCREMENT,
 'username' varchar(100) NOT NULL,
 'password' varchar(32) NOT NULL,
 PRIMARY KEY ('id')
);
INSERT INTO 'user' ('id', 'username', 'password')
VALUES ('1', 'Alex', '202cb962ac59075b964b07152d234b70');
INSERT INTO 'user' ('id', 'username', 'password')
VALUES ('2', 'Qiang', '202cb962ac59075b964b07152d234b70');
```

- Создайте модель `User` с помощью `Gii`.

## Как это делается

1. Во-первых, реализуем простое действие, которое проверяет корректность логина и пароля, полученного из URL. Создайте файл `protected/controllers/SqlController.php`:

```
<?php
class SqlController extends CController
{
 public function actionSimple()
 {
 $userName = $_GET['username'];
 $password = md5($_GET['password']);
 $sql = "SELECT * FROM user WHERE username = '$userName'";
```

```

 AND password = '$password' LIMIT 1;";
 $user = Yii::app()->db->createCommand($sql)->
 queryRow();
 if($user)
 {
 echo "Success";
 }
 else
 {
 echo "Failure";
 }
 }
}

```

2. Попробуем получить доступ к этому действию, используя URL:

```
/sql/simple?username=test&password=test
```

Поскольку мы не знаем ни имени пользователя, ни пароля, то, как и следовало ожидать, получим сообщение «Failure».

3. Попробуем теперь другой URL:

```
/sql/simple?username=%27+or+%271%27%3D%271%27%3B+--
&password=whatever
```

На этот раз мы получили доступ, хотя по-прежнему не знаем ничего о регистрационных данных. После декодирования часть значения username выглядит так:

```
' or '1'='1'; --
```

Здесь закрывается одинарная кавычка, чтобы не нарушался синтаксис.

Добавляется выражение OR '1'='1', которое всегда имеет истинное значение.

Символы ; -- экранируют оставшуюся часть строки как комментарий.

4. Поскольку экранирование вывода не было реализовано, запрос был выполнен как

```

SELECT * FROM user WHERE username = '' or
'1'='1'; --' AND password = '008c5926ca861023c1d2a36653fd88e2'
LIMIT 1;

```

Лучший способ устранить уязвимость – использовать подготовленную команду следующим образом:

```

public function actionPrepared()
{

```

```
$userName = $_GET['username'];
$password = md5($_GET['password']);
$sql = "SELECT * FROM user WHERE username = :username
 AND password = :password LIMIT 1;";
$command = Yii::app()->db->createCommand($sql);
$command->bindValue('username', $userName);
$command->bindValue('password', $password);
$user = $command->queryRow();
if($user)
{
 echo "Success";
}
else
{
 echo "Failure";
}
```

5. Проверим /sql/prepared с теми же зловредными параметрами. На этот раз все прошло благополучно, и результатом стало сообщение «Failure». Тот же принцип применяется для Active Record. Единственная разница состоит в том, что AR использует другой синтаксис:

```
public function actionAr()
{
 $userName = $_GET['username'];
 $password = md5($_GET['password']);
 $result = User::model()->exists("username = :username
 AND password = :password", array(
 'username' => $userName,
 'password' => $password,
));
 if($result)
 {
 echo "Success";
 }
 else
 {
 echo "Failure";
 }
}
```

В приведенном выше коде мы использовали параметры :username и :password и передали значения параметров во втором аргументе. Если бы мы написали этот код с использованием только первого аргумента, код был бы уязвимым:

```
public function actionWrongAr()
{
```

```
$userName = $_GET['username'];
$password = md5($_GET['password']);

$result = User::model()->exists("username = $userName
AND password = $password");
if($result)
{
 echo "Success";
}
else
{
 echo "Failure";
}
}
```

Подготовленные команды при корректном использовании могут защитить от всех типов SQL-инъекций. Но остаются несколько общих проблем:

- можно связать только одно значение с одним параметром, поэтому если вы захотите использовать в запросе `WHERE IN(1, 2, 3, 4)`, то вам придется создавать и связывать четыре параметра;
- подготовленные команды не могут использоваться для имен таблиц, имен полей и других ключевых слов.

При использовании Active Record первую проблему можно решить следующим образом:

```
public function actionIn()
{
 $criteria = new CDbCriteria();
 $criteria->addInCondition('username', array('Qiang', 'Alex'));
 $users = User::model()->findAll($criteria);
 foreach($users as $user)
 {
 echo $user->username."
";
 }
}
```

Вторую проблему можно решить разными способами. Во-первых, можно положиться на Active Record и экранирование PDO:

```
public function actionColumn()
{
 $attr = $_GET['attr'];
 $value = $_GET['value'];

 $users = User::model()->findAllByAttributes(array
```

```
 ($attr => $value));
 foreach($users as $user)
 {
 echo $user->username."
";
 }
}
```

Второй, более безопасный способ состоит в использовании принципа белых списков следующим образом:

```
public function actionWhitelist()
{
 $attr = $_GET['attr'];
 $value = $_GET['value'];

 $allowedAttr = array('username', 'id');

 if(!in_array($attr, $allowedAttr))
 throw new CException("Указанный атрибут не разрешен.");

 $users = User::model()->findAllByAttributes(array
 ($attr => $value));
 foreach($users as $user)
 {
 echo $user->username."
";
 }
}
```

## Как это работает

Основная идея при предотвращении SQL-инъекции состоит в правильной фильтрации ввода. Во всех случаях, кроме имён таблиц, мы использовали подготовленные команды – их поддерживает большинство серверов реляционных баз данных. Они позволяют создавать команды один раз и использовать их много раз, предоставляют безопасный способ связывать значения с параметрами.

В Yii вы можете использовать подготовленные команды как для Active Record, так и для DAO. При использовании DAO это делается с помощью либо `bindValue`, либо `bindParam`. Второй метод полезен, когда мы хотим выполнить несколько однотипных запросов с разными значениями параметров:

```
$command = Yii::app()->db->createCommand($sql);
$username, $password;
$command->bindParam('username', $username);
foreach($records as $record)
{
```

```
$username = $record['username'];
$command->execute();
}
```

Большинство методов Active Record принимает либо критерии, либо параметры. Для безопасности лучше использовать их, чем передавать данные напрямую.

Для экранирования имен таблиц, имен полей и других ключевых слов можно положиться на Active Record или использовать принцип белых списков.

## И ещё

Чтобы больше узнать о SQL-инъекциях и работе с базами данных через Yii, обратитесь к следующим ресурсам:

- <http://www.slideshare.net/billkarwin/sql-injection-myths-and-fallacies>;
- <http://www.yiiframework.com/doc/api/CDbConnection>;
- <http://www.yiiframework.com/doc/api/CDbCommand>.

## Смотрите также

- Рецепт «Получение данных из базы данных» в главе 6.
- Рецепт «Использование CDbCriteria» в главе 6.

## Предотвращение CSRF

CSRF или XSRF расшифровывается как «подделка межсайтовых запросов» (*Cross-Site Request Forgery*, *X-Site Request Forgery*). Злоумышленник заставляет браузер посетителя веб-страницы скрытно выполнить HTTP-запрос к сайту, на котором посетитель авторизован. Примером такой атаки может служить вставка невидимого тега изображения с атрибутом `src`, ссылающимся на `http://example.com/site/logout`. Даже если тег изображения находится на другом сайте, вы немедленно завершите сеанс на сайте `example.com`. Последствия CSRF могут быть очень серьезными: удаление данных с сайта, недопущение авторизации всех пользователей сайта, раскрытие приватной информации и т. д.

В этом рецепте мы рассмотрим, как сделать приложение устойчивым к CSRF.

## Подготовка

Создайте новое приложение с помощью `yiiic webapp`.

## Как это делается

Начнем с некоторых фактов о CSRF:

- поскольку CSRF рассчитан на выполнение в браузере жертвы, атакующий обычно не может менять заголовки посылаемых HTTP-запросов. Тем не менее находились уязвимости браузеров и расширений Flash, позволяющие подделывать заголовки, поэтому нельзя полагаться на их неизменность;
- атакующий должен передавать такие же параметры и значения, какие передавал бы пользователь.

С учётом этого хороший метод борьбы с CSRF состоит в передаче и проверке уникального токена (жетона) во время передачи форм, а также использовании метода GET в соответствии со спецификациями HTTP.

В Yii встроены генерация и проверка токенов. Кроме того, можно автоматизировать включение токенов в формы HTML.

1. Чтобы включить защиту от CSRF, следует добавить в файл `protected/config/main.php` следующее:

```
'components'=>array(
 ...
 'request'=>array(
 'enableCsrfValidation'=>true,
),
 ...
),
```

2. После настройки приложения нужно использовать `CHtml::beginForm` и `CHtml::endForm` вместо HTML тегов формы:

```
public function actionCreate()
{
 echo CHtml::beginForm();
 echo CHtml::submitButton();
 echo CHtml::endForm();
}
```

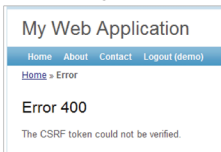
3. Yii будет автоматически добавлять скрытое поле токена следующим образом:

```
<form action="/csrf/create" method="post">
<div style="display:none"><input type="hidden">
```



```
value="e4d1021e79ac269e8d6289043a7a8bc154d7115a"
name="YII_CSRF_TOKEN" />
```

4. Если вы сохраните эту форму как HTML и попытаетесь отправить ее, то получите сообщение, как изображено на скриншоте ниже, вместо обычной обработки данных:



## Как это работает

Изнутри часть `Chtml::beginForm()` выглядит примерно так:

```
if($request->enableCsrfValidation && !strcasecmp($method,'post'))
 $shiddens[]=self::hiddenField($request->csrfTokenName,
 $request->getCsrfToken(),array('id'=>false));
if($shiddens!==array())
 $form.="\\n".self::tag('div',array('style'=>'display:none'),
 implode("\\n",$shiddens));
```

В представленном выше коде `getCsrfToken()` генерирует токен с уникальным значением и записывает его в куки. Потом, при последующих запросах, куки и значение в POST сравниваются. Если они не совпадают, то вместо обычной обработки данных выводится сообщение об ошибке.

Если вам нужно произвести запрос POST, но не создавать форму с помощью `Chtml`, вы можете передать параметр с именем из `Yii::app()->request->csrfTokenName` и значением из `Yii::app()->request->getCsrfToken()`.

## И ещё

Существуют и другие способы повысить безопасность вашего приложения, рассматриваемые в следующих подразделах:

## Дополнительные меры

Если приложению необходим очень высокий уровень безопасности, как в системе управления банковскими счетами, можно принять дополнительные меры.

Во-первых, можно отключить опцию «запомнить меня» с помощью `protected/config/main.php` следующим образом:

```
'components' => array(
 ...
 'user'=>array(
 // включить авторизацию на основе куки
 'allowAutoLogin'=>false,
),
 ...
),
```

Затем можно уменьшить тайм-аут сеанса следующим образом:

```
'components' => array(
 ...
 'session' => array(
 'timeout' => 200,
),
 ...
),
```

Разумеется, эти меры причинят неудобство пользователю, но они добавят дополнительный уровень безопасности.

## Правильное использование методов GET и POST

Стандарт HTTP возражает против использования метода GET для операций, изменяющих данные или состояние. Следует стараться соблюдать это правило. Это не предотвратит все виды CSRF, но, по крайней мере, сделает некоторые инъекции, например `<img src=`, бессмысленными.

## Полезные материалы

Чтобы больше узнать о безопасности в Yii, обратитесь по следующему адресу: <http://www.yiiframework.com/doc/guide/ru/topics.security>.

## Смотрите также

- Рецепт «Использование CHtml и CHtmlPurifier для предотвращения XSS» в этой главе.

## Использование RBAC

RBAC (*Role-Based Access Control*, управление доступом на основе ролей) является самым мощным методом управления доступом, доступным в Yii. Он описан в руководстве, но, поскольку этот метод сложный и мощный, не так-то легко понять, как он на самом деле работает, не заглядывая под капот.

В этом рецепте мы возьмем иерархию ролей из полного руководства, импортируем ее и рассмотрим, что происходит внутри.

### Подготовка

- Создайте новое приложение с помощью `yii webapp`.
- Создайте базу данных MySQL и настройте ее.
- Импортируйте SQL-запрос из `framework/web/auth/schema-mysql.sql`.
- Настройте компонент `authManager` в файле `protected/config/main.php` следующим образом:

```
return array(
 'components'=>array(
 ...
 'authManager'=>array(
 'class'=>'CDbAuthManager',
 'connectionID'=>'db',
),
),
 ...
);
```

- Добавьте дополнительные роли в файле `protected/components/UserIdentity.php`. Массив `users` должен выглядеть примерно так:

```
$users=array(
 // username => password
 'demo'=>'demo',
 'admin'=>'admin',
 'readerA'=>'123',
 'authorB'=>'123',
 'editorC'=>'123',
 'adminD'=>'123',
);
```

## Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/controllers/RbacController.php` следующего содержания:

```
<?php
class RbacController extends CController
{
 public function filters()
 {
 return array(
 'accessControl',
);
 }

 public function accessRules()
 {
 return array(
 array(
 'allow',
 'actions' => array('deletePost'),
 'roles' => array('deletePost'),
),
 array(
 'allow',
 'actions' => array('init', 'test'),
),
 array('deny'),
);
 }

 public function actionInit()
 {
 $auth=Yii::app()->authManager;

 $auth->createOperation('createPost','создать сообщение');
 $auth->createOperation('readPost','прочитать сообщение');
 $auth->createOperation('updatePost','обновить сообщение');
 $auth->createOperation('deletePost','удалить сообщение');

 $bizRule='return Yii::app()->user->id==$params
 ["post"]->authID;';

 $task=$auth->createTask('updateOwnPost','обновить
 свое сообщение ', $bizRule);
 $task->addChild('updatePost');

 $role=$auth->createRole('reader');
```

```
$role->addChild('readPost');

$role=$auth->createRole('author');
$role->addChild('reader');
$role->addChild('createPost');
$role->addChild('updateOwnPost');

$role=$auth->createRole('editor');
$role->addChild('reader');
$role->addChild('updatePost');

$role=$auth->createRole('admin');
$role->addChild('editor');
$role->addChild('author');
$role->addChild('deletePost');

$auth->assign('reader','readerA');
$auth->assign('author','authorB');
$auth->assign('editor','editorC');
$auth->assign('admin','adminD');

echo "Выполнено.";
}

public function actionDeletePost()
{
 echo "Сообщение удалено.";
}

public function actionTest()
{
 $post = new stdClass();
 $post->authID = 'authorB';
 echo "Текущие разрешения:
";
 echo "";
 echo "Создать сообщение: ".Yii::app()->user
->checkAccess
 ('createPost')."";

 echo "Прочитать сообщение: ".Yii::app()->user
->checkAccess
 ('readPost')."";
 echo "Обновить сообщение: ".Yii::app()->user
->checkAccess
 ('updatePost', array('post' => $post))."";
 echo "Удалить сообщение: ".Yii::app()->user
->checkAccess
 ('deletePost')."";
}
```

```
 echo "";
 }
}
```

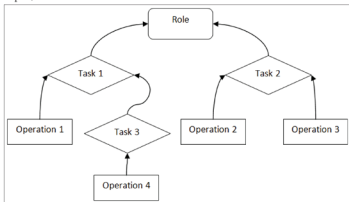
2. Теперь запустите `init` один раз для создания иерархии RBAC. Затем попробуйте авторизоваться как `readerA`, `authorB`, `editorC` и `adminD` (пароль «123») и перейти к `test` и `deletePost`.

**Current permissions:**

- Create post:
- Read post: 1
- Update post: 1
- Delete post:

## Как это работает

Иерархия RBAC является направленным ациклическим графом, то есть это множество узлов (объекты авторизации) и их направленные соединения или стороны. Узлы делятся на три типа: роли, задачи и операции:



Роль – это объект авторизации, прикрепленный к пользователю (то есть модератору или администратору). Операция определяет, может ли действие быть выполнено (то есть удаление сообщения, изменение сообщения и т. д.). Задача – это группа операций (то есть задача управления и т. д.).

Существуют два способа назначения роли пользователю:

- с помощью `Yii::app()->authManager->assign()`;
- редактирование `defaultRoles` в настройках приложения для компонента `authManager`.

Роли по умолчанию обычно используются, когда нужно назначить роль большой группе пользователей на основе некоторого выражения PHP, например `Yii::app()->user->isGuest`.

По правилам, описанным в полном руководстве, запрещается подсоединять узлы верхних уровней к узлам нижних уровней. Например, подсоединять роль к задаче. Делать наоборот можно, поэтому мы можем подсоединить задачу к роли.

Проверяя доступ, мы обычно передаем имя операции и, при необходимости, несколько параметров. Внутри `Yii` пытается найти путь от заданной операции до текущей роли пользователя, используя обратный поиск в ширину ([http://ru.wikipedia.org/wiki/Поиск\\_в\\_ширину](http://ru.wikipedia.org/wiki/Поиск_в_ширину)). Поэтому, когда мы хотим выяснить, есть ли у пользователя с ролью *Роль* доступ к выполнению операции *Операция4*, `Yii` проходит следующий путь:

Операция4 – Задача3 – Задача1 – Роль.

Каждый узел содержит бизнес-правило, или `bizRule`. Это бизнес-правило является строкой, содержащей некоторый код PHP, который возвращает истину или ложь. Возвращаемое значение определяет, можем мы пройти через узел или нет.

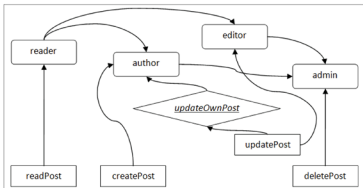
В конце концов, мы либо достигаем роли, что означает получение допуска, либо проходим все возможные пути и не достигаем роли, что означает отказ в доступе.

Есть два способа проверить, может ли пользователь выполнить указанную операцию:

1. Использовать правила `accessRules` контроллера, указав операцию, задачу или роль в параметре `roles` правила доступа.
2. Использовать `Yii::app()->user->checkAccess()`.

Используя второй способ, мы можем передать некоторые данные, которые проходят путь по иерархии авторизации и передаются каждому встреченному `bizRule`.

Теперь вернемся к нашему примеру. Код действия `init` использует компонент `authManager` для создания следующей иерархии:



Для проверки прав мы создали два действия: `test`, которое перечисляет права CRUD, и `deletePost`, которое ограничено фильтром доступа. Правило для фильтра доступа содержит следующий код:

```
array(
 'allow',
 'actions' => array('deletePost'),
 'roles' => array('deletePost'),
),
```

Это означает, что мы позволяем всем пользователям, кто имеет право `deletePost`, выполнять действие `deletePost`. Yii начинает проверку с операции `deletePost`, и единственный путь, который открыт, — это `admin`. Это означает, что только пользователи с ролью `admin` имеют возможность удалить сообщение.

Несмотря на то, что элемент правила доступа называется *roles*, вы можете указать узел в иерархии RBAC, будь то роль, задача или операция.

Когда мы проверяем разрешение `readPost` для пользователя с логином `authorB`, Yii проверяет `readPost`, `reader`, а затем `editor`. Проверка для `updatePost` сложная:

```
Yii::app()->user->checkAccess('updatePost', array('post' => $post))
```

Мы используем второй параметр, чтобы передать сообщение (в нашем случае мы симулируем его с помощью `stdClass`). Если пользователь авторизован как `authorB`, то, чтобы получить доступ, мы должны пройти от `updatePost` до `author`. В лучшем случае нам придется пройти только через `updatePost`, `updateOwnPost` и `author`.



Поскольку `updateOwnPost` имеет определенное `bizRule`, то оно будет выполнено с параметром, переданным `checkAccess`. Если результатом будет истина, то доступ будет разрешен. Поскольку Yii не знает, какой путь самый короткий, будут проверены все возможности, пока не будет успешного завершения или пока не будут пройдены все возможные пути.

## И ещё

Есть несколько полезных трюков, которые помогут вам использовать RBAC эффективно. Они рассматриваются в следующих подразделах:

### Имена узлов RBAC

Сложная иерархия становится трудной для понимания без некоторого соглашения об именах. Одно возможное соглашение, которое поможет не запутаться, состоит в следующем:

```
[group_] [own_] entity_action,
```

где `own` используется, когда правило определяет возможность изменять элемент, только если пользователь является владельцем элемента, а `group` – просто пространство имен. `Entity` – это имя сущности, с которой мы работаем, а `action` – действие, которое мы выполняем.

Например, если мы хотим создать правило, которое определяет, может ли пользователь удалить (*delete*) сообщение (*post*) в блоге (*blog*), мы назовем его `blog_post_delete`. Если правило определяет, может ли пользователь изменять (*edit*) собственный (*own*) комментарий (*comment*) в блоге (*blog*), то оно будет называться `blog_own_comment_edit`.

### Способ сохранять иерархию простой и эффективной

Следуйте этим рекомендациям, насколько это возможно, чтобы максимизировать производительность и уменьшить сложность иерархии:

- избегайте назначения нескольких ролей одному пользователю;
- не соединяйте однотипных узлов. Например, избегайте объединения задач.

## Избегать RBAC

Чтобы сохранять иерархию ещё более простой, можно избегать создания и использования дополнительных узлов, в некоторых случаях заменяя их дополнительными условиями. Хорошим примером послужит изменение сообщения. Можно создать узел `blog_own_post_edit` и следующего `bizRule`:

```
return Yii::app()->user->id==$params["post"]->author_id;
```

Или же мы можем добавить ту же логику в процедуру выбора сообщения следующим образом:

```
$post = Post::model()->findByAttributes(array(
 'id' => $id,
 'author_id' => Yii::app()->user->id,
));
if(!$post)
 throw new CHttpException(404);
```

В последнем случае мы избежим извлечения узла иерархии RBAC из хранилища и прохода по нему.

## Полезные материалы

Чтобы больше узнать об управлении доступом на основе ролей, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/guide/1.1/ru/topics.auth#sec-7>;
- [http://ru.wikipedia.org/wiki/Управление\\_доступом\\_на\\_основе\\_ролей](http://ru.wikipedia.org/wiki/Управление_доступом_на_основе_ролей);
- [http://ru.wikipedia.org/wiki/Направленный\\_ациклический\\_граф](http://ru.wikipedia.org/wiki/Направленный_ациклический_граф).

## Смотрите также

- Рецепт «Использование фильтров контроллера» в этой главе.



# **ГЛАВА 11.**

## **Настройка производительности**

В этой главе мы рассмотрим:

- Использование передового опыта.
- Ускорение управления сессиями.
- Использование зависимостей кеша и цепочек.
- Профилирование приложения с помощью Yii.

### **Введение**

Платформа Yii – одна из самых быстрых. И все же при разработке и развертывании приложений не помешает получить небольшой резерв производительности даром, а также использовать передовой опыт в самом приложении. В этой главе мы рассмотрим, как настроить Yii для получения большей производительности. В дополнение мы изучим некоторые из лучших методов разработки приложения, которое будет работать плавно, пока не получим очень высоких нагрузок.

### **Использование передового опыта**

В этом рецепте мы рассмотрим, как настроить Yii для самой высокой производительности, и увидим несколько дополнительных принципов конструирования отзывчивых приложений. Эти принципы общие и в то же время относятся к Yii. Поэтому мы сможем применить некоторые из них даже без использования Yii.

#### **Подготовка**

- Установите APC (<http://www.php.net/manual/ru/apc.installation.php>).

- Создайте новое приложение Yii с помощью `yii webapp`.

## Как это делается

Выполните следующие шаги:

1. Во-первых, нам нужно отключить отладочный режим. Это делается редактированием `index.php` следующим образом:

```
defined('YII_DEBUG') or define('YII_DEBUG',false);
```

2. Следующим шагом будет использование `yiilite.php`. Снова редактируем `index.php`, изменяем

```
$yii=dirname(__FILE__).'../framework/yii.php';
```

на следующее:

```
$yii=dirname(__FILE__).'../framework/yiilite.php';
```

3. Теперь перейдем к файлу `protected/config/main.php` и заменим его следующим:

```
<?php
// раскомментируйте следующее, чтобы определить псевдоним пути
// Yii::setPathOfAlias('local','path/to/local-folder');

// Это конфигурация главного веб-приложения. Все изменяемые
// свойства CwebApplication могут быть настроены здесь.
return array(
 'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
 'name'=>'My Web Application',

 // предварительная загрузка компонента 'log'
 'preload'=>array('log'),

 // автозагрузка классов модели и компонентов
 'import'=>array(
 'application.models.*',
 'application.components.*',
),

 'modules'=>array(
 // раскомментируйте следующее, чтобы включить Gii
 /*
 'gii'=>array(
 'class'=>'system.gii.GiiModule',
 'password'=>'Enter Your Password Here',
 // Если удалить, Gii работает только на localhost
 // Редактировать осторожно

```

```
'ipFilters'=>array('127.0.0.1','::1'),
),
*/
),

// компоненты приложения
'components'=>array(
 'user'=>array(
 // включить авторизацию по куки
 'allowAutoLogin'=>true,
),
 'urlManager'=>array(
 'urlFormat'=>'path',
 'rules'=>array(
 '<controller:\w+>/<id:\d+>'=>'<controller>/view',
 '<controller:\w+>/<action:\w+>/<id:\d+>'=>
 '<controller>/<action>',
 '<controller:\w+>/<action:\w+>'=>
 '<controller>/<action>',
),
),
 'db'=>array(
 'connectionString' => 'mysql:host=localhost;
dbname=test',
 'username' => 'root',
 'password' => '',
 'charset' => 'utf8',
 'schemaCachingDuration' => 180,
),
 'errorHandler'=>array(
 // используйте действие 'site/error' для вывода ошибок
 'errorAction'=>'site/error',
),
 'log'=>array(
 'class'=>'CLogRouter',
 'routes'=>array(
 array(
 'class'=>'CFileLogRoute',
 'levels'=>'error, warning',
),
 // раскомментируйте следующее, чтобы видеть
логи на веб-страницах
 /*
 array(
 'class'=>'CWebLogRoute',
),
 */
),
),
),
```

```
),
 'session' => array(
 'class' => 'CCacheHttpSession',
),
 'cache' => array(
 'class' => 'CApcCache',
),
),
// параметры уровня приложения, к которым можно получить
// доступ через Yii::app()->params['paramName']
'params'=>array(
 // это используется на странице с контактами
 'adminEmail'=>'webmaster@example.com',
),
);
```

4. Вот и всё. Теперь можно не беспокоиться о накладных расходах самого фреймворка и сосредоточиться на нашем приложении.

## Как это работает

Когда `YII_DEBUG` установлена в `false`, Yii отключает журналирование уровня трассировки, меньше использует код обработки ошибок, прекращает проверять код (например, Yii проверяет корректность регулярных выражений в правилах маршрутизации), использует облегченные библиотеки JavaScript.

Загрузчик `yiiite.php` содержит наиболее используемые части Yii. Используя его, мы можем избежать включения лишних скриптов и расходовать меньше оперативной памяти для кеша APC.

Заметьте, что выгода от использования `yiiite.php` может быть различной в зависимости от загрузки сервера, и иногда при использовании `yiiite.php` производительность снижается. Неплохо было бы измерить производительность и тогда уже решить, что будет работать быстрее.

Теперь рассмотрим конфигурацию дополнительных компонентов:

```
'db'=>array(
 'connectionString' => 'mysql:host=localhost;dbname=test',
 'username' => 'root',
 'password' => '',
 'charset' => 'utf8',

 'schemaCachingDuration' => 180,
),
```

Установка количества секунд в `schemaCachingDuration` позволяет кешировать схему базы данных, используемую Active Record. Очень полезно для нагруженных серверов и значительно улучшает производительность Active Record. Чтобы это работало, нужно правильно настроить компонент следующим образом:

```
'cache' => array(
 'class' => 'CApcCache',
),
```

Кеш APC – один из самых быстрых кешей, если у вас один сервер. Включение кеша также положительно влияет на другие компоненты Yii. Например, маршрутизатор Yii или `urlManager` в этом случае кеширует маршруты.

Наконец, настроим компонент `session` следующим образом:

```
'session' => array(
 'class' => 'CCacheHttpSession',
),
```

Приведенный выше код включает сохранение сеансов в APC, которое значительно быстрее, чем стандартное управление сеансами, основанное на файлах.

## И ещё

Конечно, может возникнуть ситуация, когда все эти настройки не помогут добиться достаточного уровня производительности. В большинстве случаев это означает, что «узким местом» является само приложение или аппаратная часть.

### Производительность сервера – только часть общей картины

Не только производительность сервера влияет на общую производительность. Оптимизация клиентской части: загрузки стилей, изображений и скриптов, правильное кеширование и минимизация количества HTTP-запросов могут улучшить визуальную производительность даже без оптимизации кода PHP.

### Что следует делать без помощи Yii

Некоторые вещи лучше делать без использования Yii. Например, изменение размеров изображений «на лету» лучше выполнять отдельным скриптом PHP, чтобы избежать лишних накладных расходов.

## **Active Record против построителя запросов и SQL**

Используйте построитель запросов или SQL в частях приложения, критичных по производительности. Вообще, AR наиболее полезна для добавления и изменения записей, поскольку добавляет удобный слой проверки, и менее полезна для выборки записей.

### **Всегда сначала ищите медленные запросы**

База данных может стать «узким местом» за одну секунду, если разработчик случайно забудет добавить индекс к часто используемой таблице, или наоборот, или добавит слишком много индексов к таблице, в которую часто пишут. Могут также запрашиваться данные без необходимости или выполняться лишние JOIN.

### **Кешируйте или сохраняйте результаты «тяжелых» запросов**

Если можно избежать «тяжелого» запроса при каждой загрузке страницы, это следует сделать. Например, принято сохранять или кешировать результаты разбора текста с разметкой, вычищать его (очень ресурсоемкий процесс) однажды и затем использовать готовый HTML.

### **Слишком много обработки**

Иногда данных для обработки слишком много, чтобы обработать их тотчас же. Составление сложных отчетов или простая отсылка писем (если проект сильно загружен). В этом случае лучше создать очередь заданий и выполнять их по стоп или с помощью других специальных инструментов.

### **Полезные материалы**

За дополнительной информацией обращайтесь по адресу: <http://www.yiiframework.com/doc/guide/ru/topics.performance>.

## **Смотрите также**

- Рецепт «Ускорение управления сессиями» в этой главе.
- Рецепт «Использование зависимостей кеша и цепочек» в этой главе.
- Рецепт «Профилирование приложения с помощью Yii» в этой главе.



## Ускорение управления сессиями

Родное управление сессиями в PHP удовлетворительно в большинстве случаев. По крайней мере, есть две причины, по которым вы захотите изменить обработку сессий:

- при использовании нескольких серверов вам понадобится общее хранилище сессий для них;
- по умолчанию сессии PHP используют файлы, поэтому максимальная производительность может быть ограничена дисковыми операциями.

В этом рецепте мы рассмотрим, как использовать эффективное хранилище для сеансов Yii.

### Подготовка

- Создайте новое приложение Yii с помощью `yiiic webapp`.
- Должны быть установлены расширения `php_apc` и `php_memcache`, а также собственно `memcached`.

### Как это делается

Мы проведем стресс-тест веб-сайта с помощью инструмента для тестирования Apache ab. Если вы используете Apache, то эта программа, которая поставляется с бинарными файлами Apache, скорее всего, установлена в вашей системе. Запустите следующую команду, заменив `your.website` на актуальное имя компьютера в сети:

```
ab -n 1000 -c 5 http://your.website/index.php?r=site/contact
```

Будет послано 1000 запросов, по пять за один раз, и после этого будет выведен результат примерно такого вида:

```
Z:\web\usr\local\apache\bin>ab -n 1000 -c 5 http://perf/
index.php?r=site/contact
This is ApacheBench, Version 2.0.40-dev <$Revision: 1.146 $>
apache-2.0
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.
zeustech.net/
Copyright 2006 The Apache Software Foundation, http://www.
apache.org/
```

```
Benchmarking perf (be patient)
```

```
Server Software: Apache/2.2.4
```

```

Server Hostname: perf
Server Port: 80

Document Path: /index.php?r=site/contact
Document Length: 6671 bytes

Concurrency Level: 5
Time taken for tests: 11.889185 seconds
Complete requests: 1000
Failed requests: 0
Write errors: 0
Total transferred: 7103000 bytes
HTML transferred: 6671000 bytes

Requests per second: 84.11 [#/sec] (mean)
Time per request: 59.446 [ms] (mean)
Time per request: 11.889 [ms] (mean, across all
concurrent requests)
Transfer rate: 583.39 [Kbytes/sec] received

```

#### Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	0 1.5	0	15
Processing:	28	58 58.6	47	830
Waiting:	27	57 58.5	47	827
Total:	30	58 58.7	47	830

#### Percentage of the requests served within a certain time (ms)

50%	47
66%	52
75%	57
80%	60
90%	70
95%	100
98%	205
99%	457
100%	830 (longest request)

Нас интересуют запросы в секунду. Полученное число означает, что наш веб-сайт может обслуживать 84,11 запроса в секунду, если они приходят по пять штук за один раз.

Заметьте, что отладочный режим не выключен, так как нас интересуют изменения в скорости обслуживания сеансов.

Теперь добавьте следующее в файл `protected/config/main.php`, в раздел `components`:

```
'session' => array(
 'class' =>
 'CCacheHttpSession',
 'cacheID' =>
 'sessionCache',
),
'sessionCache' => array(
 'class' => 'CApcCache',
),
```

Запустите снова `ab` с теми же опциями. На этот раз вы должны получить лучший результат. В моем случае это было 131,33 запроса в секунду. Это означает, что APC как обработчик сеансов имеет производительность на 56% выше, чем стандартный обработчик, использующий файлы.

Теперь попробуем другой бэкэнд кеша – `memcached`. Измените `CApcCache` в настройках на `CMemCache` и убедитесь, что `memcached` запущен. Теперь снова запустите `ab`. В моем случае `memcached` работал немного лучше, чем файловый кеш, обслуживая 92,04 запроса в секунду.

Не полагайтесь на точность приведенных здесь результатов. Всё зависит от версий программ, настроек и характеристик оборудования. Всегда старайтесь выполнять все тесты в той среде, где будет работать ваше приложение.

Вы можете получить существенный прирост производительности, правильно выбрав бэкэнд кеша. Yii поддерживает и другие кеширующие бэкэнды, включая `eAccelerator`, `WinCache`, `XCache`, и `Zend data cache`, который поставляется с `Zend Server`. Кроме того, можно реализовать собственный бэкэнд кеша с поддержкой быстрых хранилищ `noSQL`, как `Redis`.

## Как это работает

По умолчанию Yii использует родные сессии PHP. Это означает в большинстве случаев, что используется файловая система. Файловая система не справляется с высоким параллелизмом. Например, при изменении настройки параллелизма до 10 для теста `ab` – с использованием стандартных сеансов – `ab` не в состоянии завершить тест, только 326 запросов из 1000 проходят. Как APC, так и `memcached` справляются в этой ситуации:

```
'session' => array(
 'class' =>
 'CCacheHttpSession',
```

```
'cacheID' =>
'sessionCache',
),
'sessionCache' => array(
 'class' => 'CApcCache',
),
```

В приведенном выше разделе конфигурации мы указываем Yii использовать `CCacheHttpSession` как обработчик сессий. С этим компонентом мы можем передать управление сессиями компоненту, указанному в `cacheID`. На этот раз мы применяли `CApcCache`.

При использовании бэкэндов APC или memcached следует принимать во внимание, что при достижении максимальной ёмкости кеша сессия пользователя приложения может быть потеряна.

Заметьте, что при использовании бэкэнда кеша для сессий нельзя полагаться на сессию как временное хранилище данных, поскольку в APC и memcached не останется памяти для хранения данных. В таком случае они удалят все данные или же только их часть.

В приведенных выше тестах APC оказался самым быстрым бэкэндом, но если у вас несколько серверов, то вы не сможете использовать его, так как не сможете совместно использовать данные сессий на более чем одном сервере. В случае с memcached это делается легко, поскольку данные легко доступны со скольких серверов, со скольких вы пожелаете.

Разные бэкэнды кеша обеспечивают разные уровни стабильности. Например, известно, что APC становится нестабильным, когда заполняется или когда вы пытаетесь писать по одному ключу из многих процессов.

## И ещё

Чтобы больше узнать о кешировании и сессиях, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/CCache>;
- <http://www.yiiframework.com/doc/api/CHttpSession/>;
- <http://php.net/manual/ru/book.apc.php>;
- <http://memcached.org/>;
- <http://stackoverflow.com/questions/930877/apc-vs-eaccelerator-vs-xcache>.

## Смотрите также

- Рецепт «Использование передового опыта» в этой главе.

## Использование зависимостей кеша и цепочек

Yii поддерживает различные бэкэнды кеша, но действительно гибким Yii делает поддержка зависимостей и цепочек зависимостей. Есть ситуация, когда вы не можете просто кешировать данные в течение часа, так как кешированная информация может быть изменена в любой момент.

В этом рецепте мы рассмотрим, как кешировать целую страницу и при этом всегда получать новые данные при её обновлении. Страница типа дэшборда будет показывать пять самых последних добавленных статей и общее количество для учетной записи. Заметьте, что операцию нельзя изменить после добавления, а статью можно.

### Подготовка

- Установите APC (<http://www.php.net/manual/ru/apc.installation.php>).
- Создайте новое приложение Yii с помощью `yiic webapp`.
- Настройте кеш в разделе `components` файла `protected/config/main.php` следующим образом:

```
'cache' => array(
 'class' => 'CApcCache',
),
```

- Создайте и настройте новую базу данных.
- Выполните следующие команды SQL:

```
CREATE TABLE 'account' (
 'id' int(11) unsigned NOT NULL AUTO_INCREMENT,
 'amount' decimal(10,2) NOT NULL,
 PRIMARY KEY ('id')
);
CREATE TABLE 'article' (
 'id' int(11) unsigned NOT NULL AUTO_INCREMENT,
 'title' varchar(255) NOT NULL,
 'text' text NOT NULL,
 PRIMARY KEY ('id')
);
```

- Создайте модели для таблиц `account` и `article` с помощью Gii.
- Настройте компоненты приложения `db` и `log` в `protected/config/main.php`, чтобы можно было видеть запросы к базе данных. В итоге настройки этих компонентов должны выглядеть примерно так:

```
'db'=>array(
 'connectionString' => 'mysql:host=localhost;
dbname=test',
 'username' => 'root',
 'password' => '',
 'charset' => 'utf8',

 'schemaCachingDuration' => 180,

 'enableProfiling'=>true,
 'enableParamLogging' => true,
),
'log'=>array(
 'class'=>'CLogRouter',
 'routes'=>array(
 array(
 'class'=>'CProfileLogRoute',
),
),
),
),
```

- Создайте файл `protected/controllers/DashboardController.php` следующего содержания:

```
<?php
class DashboardController extends CController
{
 public function actionIndex()
 {
 $db = Account::model()->getDbConnection();
 $total = $db->createCommand("SELECT SUM(amount)
 FROM account")->queryScalar();

 $criteria = new CDbCriteria();
 $criteria->order = "id DESC";
 $criteria->limit = 5;
 $articles = Article::model()->findAll($criteria);

 $this->render('index', array(
 'total' => $total,
 'articles' => $articles,
));
 }

 public function actionRandomOperation()
```

```

 {
 $rec = new Account();
 $rec->amount = rand(-1000, 1000);
 $rec->save();

 echo "OK";
 }

 public function actionRandomArticle()
 {
 $n = rand(0, 1000);
 $article = new Article();
 $article->title = "Заголовок #" . $n;
 $article->text = "Текст #" . $n;
 $article->save();

 echo "OK";
 }
}

```

- Создайте `protected/views/dashboard/index.php` следующим образом:

```

<h2>Всего: <?php echo $total?></h2>
<h2>5 последних статей:</h2>
<?php foreach($articles as $article):?>
 <h3><?php echo $article->title?></h3>
 <div><?php echo $article->text?></div>
<?php endforeach ?>

```

- Запустите `dashboard/randomOperation` и `dashboard/randomArticle` несколько раз. Потом выполните `dashboard/index`, и вы должны увидеть примерно следующее:

My Web Application

[Home](#)
[About](#)
[Contact](#)
[Login](#)

Total: 1811.00

5 latest articles:

Title #849

Text #849

Title #223

Text #223

Title #279

Text #279

Title #373

Text #373

Title #886

Text #886

Copyright © 2011 by My Company

All Rights Reserved

Powered by [Yii Framework](#)

Profiling Summary Report (Time: 1.03504s, Memory: 1.77700)

Profile name	Count	Total (s)	Avg. (s)	Min. (s)	Max. (s)
system.db.CDBCommand.query(SELECT * FROM `article` "Y" ORDER BY id DESC LIMIT 5)	5	0.00307	0.00061	0.00061	0.00307
system.db.CDBCommand.query(SELECT SUM(amount) FROM account)	1	0.00116	0.00116	0.00116	0.00116

## Как это делается

Выполните следующие шаги:

1. Нам нужно изменить код контроллера следующим образом:

```
class DashboardController extends CController
{
 public function filters()
 {
 return array(
 array(
 'COutputCache +index',
 // истекает через один год
 'duration'=>24*3600*365,
 'dependency'=>array(
 'class'=>'CChainedCacheDependency',
 'dependencies'=>array(
 new CGlobalStateCacheDependency('article'),
 new CDbCacheDependency('SELECT id FROM
 account ORDER BY id DESC LIMIT 1'),
),
),
),
);
 }

 public function actionIndex()
 {
 $db = Account::model()->getDbConnection();
 $total = $db->createCommand("SELECT SUM(amount)
 FROM account")->queryScalar();

 $criteria = new CDbCriteria();
 $criteria->order = "id DESC";
 $criteria->limit = 5;
 $articles = Article::model()->findAll($criteria);

 $this->render('index', array(
 'total' => $total,
 'articles' => $articles,
));
 }

 public function actionRandomOperation()
 {
 $rec = new Account();
 $rec->amount = rand(-1000, 1000);
 $rec->save();

 echo "OK";
 }
}
```



```

 }

 public function actionRandomArticle()
 {
 $n = rand(0, 1000);

 $article = new Article();
 $article->title = "Заголовок #".$n;
 $article->text = "Текст #".$n;
 $article->save();
 Yii::app()->setGlobalState('article', $article->id);
 echo "OK";
 }
}

```

- Вот и всё. Теперь, после нескольких загрузок `dashboard/index`, вы получите только один простой запрос, как это видно на следующем скриншоте:

Также попробуйте запустить `dashboard/randomOperation` или `dashboard/randomArticle` и обновить `dashboard/index` после этого. Данные должны измениться.

My Web Application

Home Admin Contact Login

Total: -31.00

5 latest articles:

Title #488  
Text #330  
Title #304  
Text #334  
Title #649  
Text #640  
Title #223  
Text #270  
Title #279  
Text #270

Copyright © 2011 by My Company  
All Rights Reserved  
Powered by [Yii Framework](#)

Building Summary Report (Time: 1.01794s, Memory: 1.65188)

ProfileName	Event	Total (s)	Avg. T(s)	Min. T(s)	Max. T(s)
system.db.CDBCommand.query(SELECT id FROM account ORDER BY id DESC LIMIT 1)		0	0.00006	0.00006	0.00006

## Как это работает

Чтобы получить максимальную производительность при минимальном переписывании кода, мы используем кеширование целых страниц, применяя фильтр следующим образом:

```

public function filters()
{
 return array(

```

```
array(
 'COutputCache +index',
 // истекает через один год
 'duration'=>24*3600*365,
 'dependency'=>array(
 'class'=>'CChainedCacheDependency',
 'dependencies'=>array(
 new CGlobalStateCacheDependency('article'),
 new CDbCacheDependency('SELECT id FROM account
 ORDER BY id DESC LIMIT 1'),
),
),
);
```

Приведенный выше код означает, что мы применяем полностраничное кеширование к действию `index`. Страница будет кеширована в течение одного года, и кеш обновится, если что-то из зависимых данных изменится. Таким образом, в общих чертах зависимость работает следующим образом:

- первый запуск: получает свежие данные, описанные в зависимостях, сохраняет для обращения к ним впоследствии, обновляет кеш;
- получает свежие данные, как описано в зависимости, получает сохранённые данные, сравнивает с новыми;
- если нет изменений, используются кешированные данные;
- если есть различия, обновляет кеш, использует свежие данные, сохраняет новые данные зависимости для обращения к ним в будущем.

В нашем случае используются два типа зависимостей: глобальное состояние и база данных. Зависимость глобального состояния использует данные из `Yii::app()->getGlobalState()` для определения, нужно ли обновить кеш, а зависимость базы данных использует результат SQL-запроса для тех же целей.

Вы можете спросить: «почему в одном случае мы используем базу данных, а в другом – глобальное состояние?» Хороший вопрос!

Целью использования зависимости базы данных является замена сложных расчетов легкими запросами, которые выбирают как можно меньше данных. Самое лучшее в этом типе зависимости то, что не нужно добавлять логику в существующий код. В нашем случае мы можем использовать этот тип зависимости для операций с учетными записями, но не можем со статьями, так как содержание статей может

изменяться. Таким образом, для статей мы присваиваем глобальному состоянию с именем `article` значение ID добавленной статьи, что означает назначение срока обновления кеша:

```
Yii::app()->setGlobalState('article', $article->id);
```

Заметьте, что если мы изменим статью 100 раз подряд и просмотрим её только после этого, то кеш будет признан недействительным и обновлен только один раз.

## И ещё

Чтобы больше узнать о кешировании и зависимостях кеша, обратитесь по следующим адресам:

- <http://www.yiiframework.com/doc/guide/ru/caching.data#cache-dependency>;
- <http://www.yiiframework.com/doc/guide/ru/caching.page>.

## Смотрите также

- Рецепт «Создание фильтров» в главе 8 «Расширение Yii».
- Рецепт «Использование фильтров контроллера» в главе 10 «Безопасность».

# Профилирование приложений с помощью Yii

Если вы использовали все лучшие наработки в развертывании приложений, но производительность всё ещё недостаточна, то вполне возможно, что проблема в самом приложении. Главный принцип в борьбе с «узкими местами» состоит в том, чтобы ничего не принимать на веру и всегда тестировать и профилировать код прежде, чем пытаться его оптимизировать.

В этом рецепте мы попробуем найти «узкие места» в демонстрационном приложении блога из состава Yii.

## Подготовка

- Загрузите версию 1.1 Yii по следующей ссылке: <http://www.yiiframework.com/download/>.
- Распакуйте директорию `demos/blog` в корневой каталог приложения, а директорию `framework` – на один уровень выше:

```
framework
www
...
index.php
...
```

- В файле `index.php` поправьте путь к `yii.php`. Он должен быть следующим:

```
$yii=dirname(__FILE__).'../../framework/yii.php';
```

- В файле `protected/yiic.php` поправьте путь к `yiic.php`. Он должен быть:

```
$yiic=dirname(__FILE__).'../../framework/yiic.php';
```

- Измените `protected/config/console.php` следующим образом:

```
return array(
 'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
 'name'=>'My Console Application',

 'import'=>array(
 'application.models.*',
 'application.components.*',
),

 'components'=>array(
 'db'=>array(
 'connectionString' => 'mysql:host=localhost;
dbname=blog',
 'emulatePrepare' => true,
 'username' => 'root',
 'password' => '',
 'charset' => 'utf8',
 'tablePrefix' => 'tbl_',
),
),
);
```

- В файле `protected/config/main.php` закомментируйте настройки базы данных SQLite и настройте MySQL:

```
/*'db'=>array(
 'connectionString' => 'sqlite:protected/data/blog.db',
 'tablePrefix' => 'tbl_',
),*/
// раскомментируйте следующее, чтобы использовать
базу данных MySQL
```

```
'db'=>array(
 'connectionString' => 'mysql:host=localhost;
dbname=blog',
 'emulatePrepare' => true,
 'username' => 'root',
 'password' => '',
 'charset' => 'utf8',
 'tablePrefix' => 'tbl_',
),
```

- Создайте новую базу данных в MySQL и импортируйте `protected/data/schema.mysql.sql`.
- Поскольку данных не так много, нам необходимо сгенерировать ещё. Создайте файл `protected/commands/DataCommand.php` следующего содержания:

```
<?php
class DataCommand extends CConsoleCommand
{
 public function actionIndex()
 {
 $db = Yii::app()->db;

 echo "Создание тергов.\n";
 for($t=1; $t<=50; $t++)
 {
 $db->createCommand()->insert('tbl_tag', array(
 'name' => "тер $t",
 'frequency' => rand(1, 20),
));
 }
 echo "Готово.\n";

 for($i=1; $i<=1000; $i++)
 {
 $tags = array();
 for($rt=1; $rt<=10; $rt++)
 {
 $tags[] = "тер ".rand(1, 100);
 }

 $db->createCommand()->insert('tbl_post', array(
 'title' => "Сообщение #{$i}",
 'content' => "Привет! Это
 содержание #{$i}",
 'tags' => implode(" ", $tags),
 'status' => Post::STATUS_PUBLISHED,
 'create_time' => time(),
));
 }
 }
}
```

```
'update_time' => time(),
'author_id' => 1,
));

$postId = $db->getLastInsertID();

for($j=1; $j<=10; $j++)
{
 $db->createCommand()->insert('tbl_comment', array(
 'content' => "Текст комментария $j.",
 'status' => Comment::STATUS_APPROVED,
 'create_time' => time(),
 'author' => "Комментатор $j",
 'email' => "commenter$j@example.com",
 'url' => "http://example.com/",
 'post_id' => $postId,
));
}

if($i%50==0)
 echo "\nДобавлено $i сообщений.\n";
}

echo "Все выполнено.\n";
}
```

- Выполните в консоли `yiiic data`, съешьте ещё этих мягких французских булочек, да выпейте... кофе.

## Как это делается

У нас есть блог со множеством сообщений и комментариев, и он как-то работает, но не достаточно быстро. Нам нужно проверить каждую страницу и найти в них «узкие места».

1. Начнем с настройки кеша и включения профилировщика SQL. Файл `protected/config/main.php` должен выглядеть примерно так:

```
...
return array(
...
 'components'=>array(
 ...
 'db'=>array(
 'connectionString' => 'mysql:host=localhost;
dbname=blog',
 'username' => 'root',
```

```
'password' => '',
'charset' => 'utf8',
'tablePrefix' => 'tbl_',

'schemaCachingDuration' => 180,

'enableProfiling'=>true,
'enableParamLogging' => true,
),
...
'log'=>array(
 'class'=>'CLogRouter',
 'routes'=>array(
 array(
 'class' => 'CProfileLogRoute',
),
),
),
),
...
'session' => array(
 'class' => 'CCacheHttpSession',
),
'cache' => array(
 'class' => 'CApcCache',
),
),
...
);
```

2. Теперь откройте главную страницу блога несколько раз и посмотрите, что показывает профилировщик:

[illegible]

## 3. Самый медленный запрос:

```
SELECT 't'.id AS 't0_c0', 't'.content AS 't0_c1', 't'.status AS 't0_c2', 't'.create_time AS 't0_c3', 't'.author AS 't0_c4', 't'.email AS 't0_c5', 't'.url AS 't0_c6', 't'.post_id AS 't0_c7', 'post'.id AS 't1_c0', 'post'.title AS 't1_c1', 'post'.content AS 't1_c2', 'post'.tags AS 't1_c3', 'post'.status AS 't1_c4', 'post'.create_time AS 't1_c5', 'post'.update_time AS 't1_c6', 'post'.author_id AS 't1_c7' FROM 'tbl_comment' 't' LEFT OUTER JOIN 'tbl_post' 'post' ON ('t'.post_id='post'.id) WHERE (t.status=2) ORDER BY t.create_time DESC LIMIT 10
```

4. Теперь мы можем добавить EXPLAIN в начало запроса и выполнить в консоли SQL или любой другой утилите для управления SQL. Это покажет нам отсутствующие индексы при фильтрации или сортировке записей. Тогда, если мы добавим индексы для `tbl_comment.status` и `tbl_comment.create_time`, это улучшит производительность запроса SELECT, как показано на следующем скриншоте:

Profiling Summary Report (Time: 0.545806s, Memory: 3.15768)					
Procedure	Count	Total (s)	Avg (s)	Min (s)	Max (s)
system.db.CDBCommand.query(SELECT 'author'.id AS 't0_c0', 'author'.username AS 't0_c1', 'author'.password AS 't0_c2', 'author'.email AS 't0_c3', 'author'.url AS 't0_c4', 'author'.profile AS 't0_c5' FROM 'tbl_user' 'author' WHERE (t.author.id = :id) bound with (ip:192.168.1.1)	10	0.00377	0.00108	0.00098	0.00119
system.db.CDBCommand.query(SELECT COUNT(DISTINCT 't'.id) FROM 'tbl_post' 't' WHERE (status=2))	1	0.00206	0.00206	0.00206	0.00206
system.db.CDBCommand.query(SELECT 't'.id AS 't0_c0', 't'.title AS 't0_c1', 't'.content AS 't0_c2', 't'.tags AS 't0_c3', 't'.status AS 't0_c4', 't'.create_time AS 't0_c5', 't'.update_time AS 't0_c6', 't'.author_id AS 't0_c7' FROM 'tbl_post' 't' WHERE (status=2) ORDER BY update_time DESC LIMIT 10)	1	0.00183	0.00183	0.00183	0.00183
system.db.CDBCommand.query(SELECT 't'.id AS 't0_c0', 't'.content AS 't0_c1', 't'.status AS 't0_c2', 't'.create_time AS 't0_c3', 't'.author AS 't0_c4', 't'.email AS 't0_c5', 't'.url AS 't0_c6', 't'.post_id AS 't0_c7', 'post'.id AS 't1_c0', 'post'.title AS 't1_c1', 'post'.content AS 't1_c2', 'post'.tags AS 't1_c3', 'post'.status AS 't1_c4', 'post'.create_time AS 't1_c5', 'post'.update_time AS 't1_c6', 'post'.author_id AS 't1_c7' FROM 'tbl_post' 't' LEFT OUTER JOIN 'tbl_post' 'post' ON ('t'.post_id = 'post'.id) WHERE (t.status=2) ORDER BY create_time DESC LIMIT 10)	1	0.00140	0.00140	0.00140	0.00140
system.db.CDBCommand.query(SELECT * FROM 'tbl_tag' 't' ORDER BY frequency DESC LIMIT 20)	1	0.00119	0.00119	0.00119	0.00119
system.db.CDBCommand.query(SELECT 'post_id' AS 'c', COUNT(*) AS 'c' FROM 'tbl_comment' 't' WHERE (status=2) AND ('t'.post_id IN (998, 999, 1000, 976, 971, 972, 973, 974, 975, 976)) GROUP BY 'post_id')	1	0.00115	0.00115	0.00115	0.00115

5. Далее, у нас 10 запросов для получения имени автора каждого сообщения. Наиболее вероятно, что есть способ объединить их в один запрос. Выполняя `post/index`, мы проверим метод `actionIndex` контроллера `PostController`:

```
$criteria=new CDbCriteria(array(
 'condition'=>'status=:STATUS_PUBLISHED',
 'order'=>'update_time DESC',
 'with'=>'commentCount',
));
if(isset($_GET['tag']))
 $criteria->addSearchCondition('tags',$_GET['tag']);
```



```
$dataProvider=new CActiveDataProvider('Post', array(
 'pagination'=>array(
 'pageSize'=>Yii::app()->params['postsPerPage'],
),
 'criteria'=>$criteria,
));

$this->render('index',array(
 'dataProvider'=>$dataProvider,
));
```

6. Когда источник данных получает сообщения, он использует критерии, определенные ранее. Как видим, критерий позволяет получить количество комментариев с помощью наиболее эффективного запроса из всех возможных. `commentCount` является отношением, определенным в модели `Post`, и если мы проверим метод `relations`, то обнаружим, что есть ещё отношение `author`. Заменяв часть `with` критерия на

```
'with'=> array('commentCount', 'author'),
```

мы избавились от 10 дополнительных запросов. Вместо них мы получили один очень производительный запрос:

Profiling Summary Report (Time: 0.15462s, Memory: 5.177MB)					
Procedure	Count	Total (s)	Avg. (s)	Min. (s)	Max. (s)
system.db.CDbCommand.query(SELECT COUNT(DISTINCT "t"."id") FROM "tbl_post" "t" LEFT OUTER JOIN "tbl_user" "author" ON ("t"."author_id" = "author"."id") WHERE ("status"=2))	1	0.00262	0.00262	0.00262	0.00262
system.db.CDbCommand.query(SELECT "t"."id" AS "id", "t"."title" AS "id_1", "t"."content" AS "id_2", "t"."tags" AS "id_3", "t"."status" AS "id_4", "t"."create_time" AS "id_5", "t"."update_time" AS "id_6", "t"."author_id" AS "id_7", "author"."id" AS "id_8", "author"."username" AS "id_9", "author"."password" AS "id_10", "author"."email" AS "id_11", "author"."profile" AS "id_12" FROM "tbl_post" "t" LEFT OUTER JOIN "tbl_user" "author" ON ("t"."author_id" = "author"."id") WHERE ("status"=2) ORDER BY "update_time" DESC LIMIT 10)	1	0.00236	0.00236	0.00236	0.00236
system.db.CDbCommand.query(SELECT "t"."id" AS "id", "t"."content" AS "id_1", "t"."status" AS "id_2", "t"."create_time" AS "id_3", "t"."author" AS "id_4", "t"."email" AS "id_5", "t"."url" AS "id_6", "t"."post_id" AS "id_7", "post"."id" AS "id_8", "post"."title" AS "id_9", "post"."content" AS "id_10", "post"."tags" AS "id_11", "post"."status" AS "id_12", "post"."create_time" AS "id_13", "post"."update_time" AS "id_14", "post"."author_id" AS "id_15" FROM "tbl_post" "t" LEFT OUTER JOIN "tbl_post" "post" ON ("t"."post_id" = "post"."id") WHERE ("status"=2) ORDER BY "create_time" DESC LIMIT 10)	1	0.00157	0.00157	0.00157	0.00157
system.db.CDbCommand.query(SELECT "tbl_tag"."t" ORDER BY frequency DESC LIMIT 20)	1	0.00134	0.00134	0.00134	0.00134
system.db.CDbCommand.query(SELECT "post"."id" AS "t", COUNT(*) AS "c" FROM "tbl_post" "t" WHERE ("status"=2) AND ("t"."post_id" IN (916, 916, 1000, 970, 971, 972, 973, 974, 975, 976)) GROUP BY "post_id")	1	0.00115	0.00115	0.00115	0.00115

7. Теперь SQL работает лучше. Можно ещё улучшить эту часть, но вы уже поняли суть и сможете справиться с этим самостоятельно. В целом же совершенство не достигнуто. Добавим профилирующие маркеры в код контроллера следующим образом:

```
Yii::beginProfile('preparing_data');
$criteria=new CDbCriteria(array(
 'condition'=>'status='.Post::STATUS_PUBLISHED,
 'order'=>'update_time DESC',
 'with'=> array('commentCount', 'author'),
```

```

));
if(isset($_GET['tag']))
 $criteria->addSearchCondition('tags',$_GET['tag']);

$dataProvider=new CActiveDataProvider('Post', array(
 'pagination'=>array(
 'pageSize'=>Yii::app()->params['postsPerPage'],
),
 'criteria'=>$criteria,
));
Yii::endProfile('preparing_data');
Yii::beginProfile('rendering_data');
$this->render('index',array(
 'dataProvider'=>$dataProvider,
));
Yii::endProfile('rendering_data');

```

8. Теперь откройте главную страницу и посмотрите в профиле:

Profiling Summary Report (Time: 0.14253s, Memory: 3,142KB)					
Procedure	Count	Total (s)	Avg (s)	Min (s)	Max (s)
rendering_data	1	0.11444	0.11444	0.11444	0.11444
preparing_data	1	0.01111	0.01111	0.01111	0.01111
system.db.CDBCommand.query(SELECT COUNT(DISTINCT "t"."id") FROM "tbl_post" "t" LEFT OUTER JOIN "tbl_user" "u" ON ( "t"."author_id" = "u"."id" ) WHERE (status=2))	1	0.00276	0.00276	0.00276	0.00276
system.db.CDBCommand.query(SELECT "t"."id" AS "id","t"."title" AS "title","t"."content" AS "content","t"."tags" AS "tags","t"."status" AS "status","t"."create_time" AS "create_time","t"."update_time" AS "update_time","t"."author_id" AS "author_id","t"."author"."id" AS "id","t"."author"."username" AS "username","t"."author"."password" AS "password","t"."author"."email" AS "email","t"."author"."update_time" AS "update_time","t"."author"."profile" AS "profile" FROM "tbl_post" "t" LEFT OUTER JOIN "tbl_user" "u" ON ( "t"."author_id" = "u"."id" ) WHERE (status=2) ORDER BY update_time DESC LIMIT 20)	1	0.00227	0.00227	0.00227	0.00227
system.db.CDBCommand.query(SELECT "t"."id" AS "id","t"."content" AS "content","t"."status" AS "status","t"."create_time" AS "create_time","t"."author_id" AS "author_id","t"."author"."id" AS "id","t"."author"."username" AS "username","t"."author"."password" AS "password","t"."author"."email" AS "email","t"."author"."update_time" AS "update_time","t"."author"."profile" AS "profile" FROM "tbl_post" "t" LEFT OUTER JOIN "tbl_user" "u" ON ( "t"."author_id" = "u"."id" ) WHERE (status=2) ORDER BY create_time DESC LIMIT 20)	1	0.00142	0.00142	0.00142	0.00142
system.db.CDBCommand.query(SELECT * FROM "tbl_tag" "t" ORDER BY frequency DESC LIMIT 20)	1	0.00122	0.00122	0.00122	0.00122
system.db.CDBCommand.query(SELECT "post_id" AS "id","COUNT(*)" AS "c" FROM "tbl_comment" "t" WHERE (status=2) AND ( "t"."post_id" IN (995, 999, 1005, 972, 971, 972, 972, 974, 972, 976)) GROUP BY "post_id" )	1	0.00118	0.00118	0.00118	0.00118

9. Похоже, что rendering\_data занимает большую часть времени. Поскольку это связано с представлениями, проверим protected/views/post/index.php:

```

<?php if(!empty($_GET['tag'])): ?>
<h1>Сообщения с тегом <i><?php echo CHtml::encode(
$_GET['tag']);
?></i></h1>
<?php endif; ?>

<?php $this->widget('zii.widgets.CListView', array(
 'dataProvider'=>$dataProvider,
 'itemView'=>'_view',
 'template'=>"{items}\n{pager}",
)); ?>

```



Можно добиться большего, если продолжать профилировать и исправлять. Вы можете сказать, что производительность была приемлемой с самого начала. Верно, до тех пор, пока у блога не появится много читателей и сервер не перестанет справляться с генерацией страниц. Полученное нами повышение производительности на самом деле означает, что если вначале было 10 000 читателей и производительность начала падать, то после оптимизации можно обслуживать ещё 4200 читателей, не покупая нового оборудования.

## **Как это работает**

Сначала мы настроили кеш приложения и кешировали схему базы данных, чтобы она не стала «узким местом». При рабочем режиме она наверняка будет кеширована. Потом мы включили профилирование запросов к базе данных и запустили приложение несколько раз: при первом запуске Yii кеширует схему и маршруты, второй запуск будет чистым.

Узким местом типичного веб-приложения является база данных, поэтому мы стали искать аномалии в SQL-запросах: медленные запросы и повторяющиеся однотипные запросы.

Медленные запросы обычно означают плохую архитектуру базы данных (неверная расстановка индексов, излишняя нормализация и т. д.). Мы посылаем запрос MySQL с добавлением EXPLAIN и получаем профиль запроса, из которого становится ясно, что можно исправить.

Когда однотипные запросы повторяются много раз, это может быть нечто, повторяющееся на каждой странице, например имя автора сообщения. В большинстве случаев можно объединить эти запросы в один.

Что касается не-SQL частей, мы разделили контроллер пополам, взяли более медленную половину и разделили снова. Мы повторяли это действие, пока не нашли источника проблем. Разумеется, мы строили некоторые предположения, чтобы сократить рутину, но иногда лучше этого не делать, поскольку узкое место может внешне ничем не выделяться.

## **И ещё**

Чтобы больше узнать о профилировании, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/guide/1.1/ru/topics.logging#sec-6>;

- <http://www.yiiframework.com/doc/guide/1.1/ru/topics.logging#sec-7>;
- <http://www.xdebug.org/docs/profiler>;
- <http://pecl.php.net/package/xhprof>.

### ***Смотрите также***

- Рецепт «Использование различных маршрутов для журналов» в главе 9 «Обработка ошибок, отладка и журналирование».
- Рецепт «Использование передового опыта» в этой главе.
- Рецепт «Ускорение управления сессиями» в этой главе.



## **ГЛАВА 12.**

# **Использование постороннего кода**

В этой главе мы рассмотрим:

- Использование Zend Framework из Yii.
- Настройка автозагрузчика Yii.
- Использование Kohana внутри Yii.
- Использование PEAR внутри Yii.

## **Введение**

Обычно приложению нужно больше, чем может предложить любой отдельный фреймворк. Иногда необходима полнофункциональная библиотека для отправки электронных писем, иногда нужна реализация определенного API. Ни один фреймворк не может покрыть все задачи, которые могут стоять перед разработчиком. Поэтому Yii покрывает наиболее часто встречающиеся, а остальные разработчик может реализовать сам или подключить внешнюю библиотеку.

В этой главе мы попробуем использовать в Yii код не из Yii, в том числе Zend Framework, Kohana и Pear.

## **Использование Zend Framework из Yii**

Yii предоставляет много прекрасных решений для построения приложений. Все же вам может понадобиться что-то ещё. Один из лучших вариантов – классы Zend Framework. Они качественные и решают много задач, например использование Google API или работа с электронной почтой.

В этом рецепте мы рассмотрим, как использовать пакет `Zend_Mail` для отправки электронных писем из приложения Yii. Мы используем простой подход задействования целого фреймворка, а также реализуем специальный автозагрузчик, позволяющий работать только с `Zend_Mail` и его зависимостями.

## Подготовка

- Создайте новое приложение с помощью `yiiic webapp`.
- Загрузите Zend Framework со следующего ресурса: <http://framework.zend.com/downloads/latest>.  
В этом рецепте мы используем версию 1.11.6.
- Извлеките директорию `library/Zend` из полученного архива в директорию `protected/vendors/Zend`.

## Как это делается

Выполните следующие шаги:

1. Мы создадим простой контроллер, который отправляет электронное письмо. Создайте файл `protected/controllers/MailtestController.php` следующего содержания:

```
<?php
class MailtestController extends CController
{
 public function actionIndex()
 {
 $mail = new Zend_Mail('utf-8');
 $mail->setHeaderEncoding(Zend_Mime::ENCODING_
QUOTEDPRINTABLE);
 $mail->addTo("alexander@example.com",
"Alexander Makarov");
 $mail->setFrom("robot@example.com", "Robot");
 $mail->setSubject("Test email");
 $mail->setBodyText("Hello, world!");
 $mail->setBodyHtml("Hello, world!");
 $mail->send();
 echo "OK";
 }
}
```

2. Запустите `mailtest/index`, и вы получите следующий результат:

## PHP Error

```
include(Zend_Mail.php): failed to open stream: No such file or directory
```

```
W:\home\ext\framework\YiiBase.php(398)
```

```

386 * @return boolean whether the class has been loaded successfully
387 */
388 public static function autoload($className)
389 {
390 // use include so that the error PHP file may appear
391 if(isset(self::$_coreClasses[$className]))
392 include(Yii::getPathOfClasses($className));
393 else if(isset(self::$_classMap[$className]))
394 include(self::$_classMap[$className]);
395 else
396 {
397 if(strpos($className, '\\') === false)
398 include($className.'.php');
399 else // class name with namespace in PHP 5.3
400 {
401 $namespace=str_replace('\\', '.', ltrim($className, '\\'));
402 if(($path=self::getPathOfAlias($namespace))!==false)
403 include($path.'.php');
404 else
405 return false;
406 }
407 return class_exists($className,false) || interface_exists($className,false);
408 }
409 return true;
410 }

```

## Stack Trace

```

#0 W:\home\ext\framework\YiiBase.php(398): YiiBase::autoload()
#1 unknown(0): YiiBase::autoload('Zend_Mail')
#2 W:\home\ext\www\protected\controllers\MailtestController.php(8): sol_autoload_call('Zend_Mail')

```

3. Это означает, что автозагрузчик Yii не включил класс Zend\_Mail. Этого следовало ожидать, поскольку автозагрузчик Yii не знает соглашения об именах в Zend Framework. У нас есть два решения данной проблемы:

- включать классы явно;
- создать собственный автозагрузчик.

4. Начнем с включения классов. Все классы Zend Framework имеют инструкции `require_once` для всех зависимостей. Эти инструкции полагаются на добавление дополнительного пути к библиотекам PHP и выглядят примерно так:

```
require_once 'Zend/Mail/Transport/Abstract.php';
```

5. Использование `Yii::import` для импорта директории работает как добавление директории в путь к библиотекам PHP, и мы можем решить свою проблему следующим образом:



```

class MailtestController extends CController
{
 public function actionIndex()
 {
 Yii::import('application.vendors.*');
 require "Zend/Mail.php";

 $mail = new Zend_Mail('utf-8');
 $mail->setHeaderEncoding(Zend_Mime::ENCODING_
QUOTEDPRINTABLE);
 $mail->addTo("alexander@example.com",
"Alexander Makarov");
 $mail->setFrom("robot@example.com", "Robot");
 $mail->setSubject("Test email");
 $mail->setBodyText("Hello, world!");
 $mail->setBodyHtml("Hello, world!");
 $mail->send();

 echo "OK";
 }
}

```

6. Теперь письмо будет отослано без ошибок. Использованный способ работает, если не приходится использовать много классов Zend Framework. Если вы используете их активно, вам придется много их включать, и это вызовет ненужное усложнение. Теперь используем Zend\_Loader\_Autoloader для достижения цели.
7. Самое подходящее место для добавления автозагрузчика – начальная загрузка index.php. Таким образом, вы сможете автоматически загружать классы в течение всего процесса исполнения:

```

// измените следующие пути, если понадобится
$yii=dirname(__FILE__).'/../framework/yii.php';
$config=dirname(__FILE__).'/protected/config/main.php';

// удалите следующие строки перед использованием в
// рабочем режиме
defined('YII_DEBUG') or define('YII_DEBUG',true);
// укажите, сколько уровней стека вызовов должно
// отображаться в каждом сообщении лога
defined('YII_TRACE_LEVEL') or define('YII_TRACE_LEVEL',3);
require_once($yii);
$app = Yii::createWebApplication($config);

// добавляем автозагрузчик Zend Framework

```

```
Yii::import('application.vendors.*');
require "Zend/Loader/Autoloader.php";
Yii::registerAutoloader(array('Zend_Loader_Autoloader',
 'autoload'), true);

$app->run();
```

#### 8. Теперь можно удалить

```
Yii::import('application.vendors.*');
require "Zend/Mail.php";
```

из MailtestController, и он будет выполняться без ошибок, и это означает, что автозагрузка Zend Framework работает.

## Как это работает

Рассмотрим, что происходит за кадром и как это работает, начиная с первого способа. Мы воспользовались методом `Yii::import`, который использован как `Yii::import('path.alias.*')`, действует как добавление пути к библиотекам PHP. Поскольку в Zend Framework изначально не было автозагрузчика, используются все необходимые инструкции `require_once`. Если вы применяете один компонент, например `Zend_Mail`, вам не понадобится более чем одна инструкция `require_once`.

Второй способ не требует использования инструкций `require`. Поскольку Yii позволяет применять множество автозагрузчиков, и в более поздних версиях у Zend Framework появился собственный автозагрузчик, мы можем использовать его в нашем приложении. Наиболее подходящий момент сделать это – сразу после начальной загрузки приложения, когда оно ещё не выполняется. Чтобы сделать это, мы разделяем `Yii::createWebApplication($config)->run()` на две части в `index.php` и вставляем инициализацию автозагрузчика между ними:

```
Yii::import('application.vendors.*');
require "Zend/Loader/Autoloader.php";
Yii::registerAutoloader(array('Zend_Loader_Autoloader',
 'autoload'), true);
```

Нам все же нужна команда `Yii::import('application.vendors.*')`, так как классы Zend Framework продолжают использовать `require_once`. Потом мы затребовали класс автозагрузчика и добавили в конец стека автозагрузки PHP с помощью `Yii::registerAutoloader`, где второй параметр равен `true`.

## И ещё

Чтобы больше узнать об импорте в Yii, автозагрузке и использовании Zend Framework, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/YiiBase/#import-detail>;
- <http://www.yiiframework.com/doc/api/YiiBase/#registerAutoloader-detail>;
- <http://framework.zend.com/>;
- <http://www.yiiframework.com/doc/guide/ru/extension.integration>;
- <http://framework.zend.com/manual/ru/zend.loader.autoloader.html>.

## Смотрите также

- Рецепт «Настройка автозагрузчика Yii» в этой главе.

## Настройка автозагрузчика Yii

Yii использует соглашение об именах и автозагрузчик, чтобы загружать только те классы, которые необходимы, и чтобы избежать необходимости перечислять все файлы поименно. Другие фреймворки и библиотеки могут использовать другие соглашения об именах, Yii предоставляет возможность изменения правил автозагрузки классов. В рецепте «Использование Zend Framework из Yii» в этой главе мы применяли `Zend_Loader_Autoloader`, чтобы иметь возможность использовать классы Zend Framework, не включая их явно. Если мы используем только основные классы Zend Framework, то его сложный автозагрузчик несколько чрезмерен. Кроме того, в каждом классе Zend Framework по-прежнему вызывается `require_once`, поэтому загружается множество неиспользуемых файлов. В этом рецепте мы создадим простой и быстрый автозагрузчик, который позволит нам делать то же самое, но быстрее.

## Подготовка

- Создайте новое приложение с помощью `yiiic webapp`.
- Загрузите Zend Framework со следующего ресурса: <http://framework.zend.com/downloads/latest>.  
В этом рецепте используется версия 1.11.6.
- Извлеките директорию `library/Zend` из полученного архива в директорию `protected/vendors/Zend`.

- Создайте файл `protected/controllers/MailtestController.php` следующего содержания:

```
<?php
class MailtestController extends CController
{
 public function actionIndex()
 {
 $mail = new Zend_Mail('utf-8');
 $mail->setHeaderEncoding(Zend_Mime::ENCODING_
QUOTEDPRINTABLE);
 $mail->addTo("alexander@example.com",
"Alexander Makarov");
 $mail->setFrom("robot@example.com", "Robot");
 $mail->setSubject("Test email");
 $mail->setBodyText("Hello, world!");
 $mail->setBodyHtml("Hello, world
!");
 $mail->send();

 echo "OK";
 }
}
```

## Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/components/EZendAutoloader.php` следующего содержания:

```
<?php
class EZendAutoloader
{
 /**
 * @var массив префиксов классов
 */
 static $prefixes = array(
 'Zend'
);

 /**
 * @var string путь к расположению корневой папки
 * классов Zend
 */
 static $basePath = null;

 /**
 * Загрузчик класса autoload.
```

```

*
* @static
* @param string $className
* @return boolean
*/
static function loadClass($className)
{
 foreach(self::$prefixes as $prefix)
 {
 if(strpos($className, $prefix.'_')!==false)
 {
 if(!self::$basePath) self::$basePath =
 Yii::getPathOfAlias("application.vendors").'/';
 include self::$basePath.str_replace
 ('_', '/', $className).'.php';
 return class_exists($className, false) ||
 interface_exists($className, false);
 }
 }
 return false;
}
}

```

Теперь измените index.php. Замените

```
Yii::createWebApplication($config)->run();
```

на

```

$app = Yii::createWebApplication($config);

// добавляем специальный автозагрузчик Zend Framework
Yii::import("application.vendors.*");
Yii::import("application.components.EZendAutoloader", true);
Yii::registerAutoloader(array('EZendAutoloader',
 'loadClass'), true);

$app->run();

```

Попробуйте запустить mailtest/index. Он должен послать электронное письмо и вывести «ОК», что означает успешную автозагрузку. Мы все-таки импортируем директорию vendors, чтобы удовлетворить вызовы require\_once из Zend Framework, загружая все возможные классы явным образом. Единственный способ исправить это – удалить все вхождения require\_once из Zend Framework. В Linux это можно сделать так:

```

% cd path/to/ZendFramework/library
% find . -name '*.php' -not -wholename '*/Loader/Autoloader.php' \

```

```
-not -wholename '*/Application.php' -print0 | \
xargs -0 sed --regexp-extended --in-place 's/(require_
once)/\\/\ \1/g'
```

В MacOSX это можно сделать так:

```
% cd path/to/ZendFramework/library
% find . -name '*.php' | grep -v './Loader/Autoloader.php' | \
xargs sed -E -i~ 's/(require_once)/\\/\ \1/g'
% find . -name '*.php~' | xargs rm -f
```

Или используйте другие инструменты для замены `require_once` на `//require_once`.

После этого вы можете удалить `Yii::import("application.vendors.*")` из `index.php` и попробовать загрузить `mailtest/index` снова. Он должен послать ещё одно письмо и вывести «ОК». Это означает, что классы Zend Framework теперь работают без `require_once`.

## Как это работает

Фреймворки и библиотеки, использующие автозагрузку, полагаются на SPL автозагрузку в PHP. Она запускается, когда используется класс, который ещё не включен, и PHP собирается выдать отказ. С помощью `spl_autoload_register` вы можете зарегистрировать несколько обратных вызовов автозагрузки. Тогда если первый откажет, то другой перехватит инициативу и попытается загрузить класс. Yii не является исключением. По умолчанию Yii использует собственную реализацию автозагрузчика `YiiBase::autoload`. Мы использовали `Yii::registerAutoloader` в `index.php`, чтобы добавить дополнительный автозагрузчик. Реализация метода такова:

```
public static function registerAutoloader($callback,
$append=false)
{
 if($append)
 {
 self::$enableIncludePath=false;
 spl_autoload_register($callback);
 }
 else
 {
 spl_autoload_unregister(array('YiiBase','autoload'));
 spl_autoload_register($callback);
 spl_autoload_register(array('YiiBase','autoload'));
 }
}
```

Таким образом, внутри тоже автозагрузчик SPL, и метод `registerAutoloader` просто добавляет ещё один обратный вызов и по умолчанию проверяет, зарегистрирован ли он раньше, чем собственный автозагрузчик Yii. Если мы вторым параметром передаем `true`, то специальный автозагрузчик регистрируется после внутреннего автозагрузчика Yii. Это предотвращает срабатывания специального автозагрузчика для классов Yii.

Теперь перейдем к нашему специальному автозагрузчику. Все обратные вызовы автозагрузки SPL принимают один аргумент, содержащий имя класса, который необходимо загрузить. Происходит попытка включить файл, содержащий класс с указанным именем. Для большей гибкости мы определили следующие два свойства:

1. Свойство `prefixes` определяет список префиксов, с которых должно начинаться имя класса, чтобы загружаться нашим автозагрузчиком. По умолчанию это `Zend`.
2. Свойство `basePath` определяет путь к директории, внутри которой находится директория `Zend`. По умолчанию это `protected/vendors`.

Для каждого префикса мы проверяем, начинается ли с него класс, который мы пытаемся загрузить, и нужен ли для этого автозагрузчик. Если имя подходит префиксу, то мы заменяем «`_`» на «`/`» в имени класса и используем его как полный путь к файлу, который мы включаем.

На последнем шаге мы избавляемся от `require_once`, чтобы загружать только абсолютно необходимые классы.

## И ещё

Так как загрузка классов происходит все время, мы используем внешнюю библиотеку; производительность должна быть максимальной. Это означает, что метод автозагрузки должен быть простым и эффективным. Что ещё следует отметить:

- `require_once` медленнее, чем `require`;
- использование `file_exists` или `is_file` замедлит загрузку;
- следует использовать абсолютные, а не относительные пути, чтобы APC работал эффективно с параметром `apc.stat = 0` (это позволяет не проверять, изменился ли файл, и увеличить производительность рабочего сервера).

## Полезные материалы

Чтобы больше узнать об автозагрузке в Yii и об APC, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/YiiBase/#import-detail>;
- <http://www.yiiframework.com/doc/api/YiiBase/#registerAutoloader-detail>;
- <http://php.net/manual/ru/function.spl-autoload.php>;
- <http://www.php.net/manual/ru/apc.configuration.php#ini.apc.stat>.

## Смотрите также

- Рецепт «Использование Zend Framework из Yii» в этой главе.

# Использование Kohana внутри Yii

Иногда для написания специального автозагрузчика нужно углубиться в исходный код другого фреймворка. Примером послужит фреймворк Kohana. В этом рецепте мы используем один из его классов для изменения размеров изображений.

## Подготовка

- Создайте новое приложение с помощью `yiiic webapp`.
- Загрузите архив фреймворка Kohana со следующего ресурса: <http://kohanaframework.org/download>.  
В этом рецепте используется версия 3.1.
- Извлеките директории `system` и `modules` в директорию `protected/vendors/Kohana`.

## Как это делается

Выполните следующие шаги:

1. Во-первых, нам нужен код, который выполняет изменение размеров изображения и выводит изображение на экран. Создайте файл `protected/controllers/ImageController.php` следующего содержания:

```
<?php
class ImageController extends CController
{
 public function actionIndex()
```



```

 {
 $image = new Image_GD(Yii::getPathOfAlias(
 "system")."/yii-powered.png");
 $image->resize(80, 80);
 Yii::app()->request->sendFile("image.png",
 $image->render());
 }
}

```

2. Попробуйте запустить `image/index`, и вы получите следующую ошибку:

### PHP Error

include(Image\_GD.php): failed to open stream: No such file or directory

W:\home\ext\framework\YiiBase.php(398)

```

386 * @return boolean whether the class has been loaded successfully
387 */
388 public static function autoload($className)
389 {
390 // use include so that the error PHP file may appear
391 if(isset(self::$_coreClasses[$className]))
392 include(Yii::getPathOfAlias($className));
393 else if(isset(self::$_classMap[$className]))
394 include(self::$_classMap[$className]);
395 else
396 {
397 if(strpos($className, '\\')===false)
398 include($className.".php");
399 else // class name with namespace in PHP 5.3
400 {
401 $namespace=str_replace('\\', '.', ltrim($className, '\\'));
402 if(($path=self::getPathOfAlias($namespace))!==false)
403 include($path.".php");
404 else
405 return false;
406 }
407 return class_exists($className,false) || interface_exists($className,false);
408 }
409 return true;
410 }

```

### Stack Trace

```

#0 W:\home\ext\framework\YiiBase.php(398): YiiBase::autoload()
#1 unknown(0): YiiBase::autoload("Image_GD")
#2 W:\home\ext\www\protected\controllers\ImageController.php(9): spl_autoload_call("Image_GD")

```

3. Это означает, что Yii не может найти классы Kohana. Чтобы исправить это, нам нужен специальный автозагрузчик. Создайте файл `protected/components/EKohanaAutoloader.php` следующего содержания:

```

<?php
class EKohanaAutoloader

```

```
{
 /**
 * @var Список путей, в которых происходит поиск классов
 * Добавьте сюда полные пути к модулям
 */
 static $paths = array();
 /**
 * Загрузчик класса autoload.
 *
 * @static
 * @param string $className
 * @return boolean
 */
 static function loadClass($className)
 {
 if(!defined("SYSPATH"))
 define("SYSPATH", Yii::getPathOfAlias(
 "application.vendors.Kohana.system"));

 if(empty(self::$paths))
 self::$paths = array(Yii::getPathOfAlias(
 "application.vendors.Kohana.system"));

 $path = 'classes/'.str_replace(
 ('_', '/', strtolower($className)).'.php';

 foreach (self::$paths as $dir)
 {
 if (is_file($dir."/".$path))
 require $dir."/".$path;
 }

 return false;
 }
}
```

4. Чтобы использовать его, нам нужно изменить index.php. Замените

```
Yii::createWebApplication($config)->run();
```

следующим:

```
$app = Yii::createWebApplication($config);
// добавляем специальный автозагрузчик Kohana
Yii::import("application.components.EKohanaAutoloader", true);
EKohanaAutoloader::$paths = array(Yii::getPathOfAlias(
 "application.vendors.Kohana.modules.image"));
Yii::registerAutoloader(array(
 ('EKohanaAutoloader', 'loadClass'), true);
$app->run();
```

5. Теперь выполните `image/index` снова, и вы должны увидеть примерно то, что изображено на скриншоте ниже, вместо ошибки:



Это означает, что классы Kohana успешно загружены.

Заметьте, что предлагаемый загрузчик классов Kohana не оптимизирован в аспекте производительности и не разрабатывался для интенсивного профессионального использования.

## Как это работает

Kohana 3 полагается на автозагрузку и имеет очень своеобразное соглашение об именах. В результате для вызова его классов напрямую нужно слишком много потрудиться, и создание специального автозагрузчика является единственным разумным способом без модификации классов Kohana.

Мы заглянем в автозагрузчик Kohana, который находится в следующем месте: `protected/vendors/Kohana/system/classes/kohana/core.php`

Имя метода – `auto_load`:

```
public static function auto_load($class)
{
 try
 {
 // Преобразование имени класса в путь
 $file = str_replace('_', '/', strtolower($class));

 if ($path = Kohana::find_file('classes', $file))
 {
 // Загрузка файла с классом
 require $path;
 // Класс найден
 return TRUE;
 }
 }
}
```

```
 // Класс не найден в файловой системе
 return FALSE;
}
catch (Exception $e)
{
 Kohana_Exception::handler($e);
 die;
}
}
```

По этому фрагменту можно сказать, что используется класс для формирования относительного пути, который потом используется для нахождения файла внутри директории `classes`:

```
$file = str_replace('_', '/', strtolower($class));
```

Теперь углубимся в `find_file`:

```
public static function find_file
($dir, $file, $ext = NULL, $array = FALSE)
{
 if ($ext === NULL)
 {
 // Используем расширение по умолчанию
 $ext = EXT;
 }
 elseif ($ext)
 {
 // Добавим точку перед расширением
 $ext = ".$ext";
 }
 else
 {
 // Без расширения
 $ext = '';
 }

 // Создадим частичный путь
 $path = $dir.DIRECTORY_SEPARATOR.$file.$ext;
 if (Kohana::$caching === TRUE AND isset
 (Kohana::$_files[$path].($array ? '_array' : '_path'))))
 {
 // Этот путь ке широван
 return Kohana::$_files[$path].($array ? '_array' : '_path');
 }

 if (Kohana::$profiling === TRUE AND class_exists
 ('Profiler', FALSE))
```

```
{
 // Запустим новый бенчмарк
 $benchmark = Profiler::start('Kohana', __FUNCTION__);
}

if ($array OR $dir === 'config' OR $dir === 'i18n'
 OR $dir === 'messages')
{
 // Включим пути для поиска в обратном порядке
 $paths = array_reverse(Kohana::$_paths);
 // Массив найденных файлов
 $found = array();

 foreach ($paths as $dir)
 {
 if (is_file($dir.$path))
 {
 // В этом пути файл есть, добавим его к списку
 $found[] = $dir.$path;
 }
 }
}
else
{
 // Файл ещё не найден
 $found = FALSE;

 foreach (Kohana::$_paths as $dir)
 {
 if (is_file($dir.$path))
 {
 // Путь найден
 $found = $dir.$path;

 // Прекращаем поиск
 break;
 }
 }
}

if (Kohana::$caching === TRUE)
{
 // Добавим путь в кеш
 Kohana::$_files[$path.($array ? '_array' : '_path')] =
 $found;

 // Файлы были изменены
 Kohana::$_files_changed = TRUE;
}
```

```
 }

 if (isset($benchmark))
 {
 // Остановим бенчмарк
 Profiler::stop($benchmark);
 }
 return $found;
}
```

Как нам известно, расширение файла всегда .php, директория всегда classes, и нас сейчас не интересуют кеширование и профилирование. Полезная часть:

```
$path = $dir.DIRECTORY_SEPARATOR.$file.$ext;
foreach (Kohana::$_paths as $dir)
{
 if (is_file($dir.$path))
 {
 // Путь найден
 $found = $dir.$path;
 // Прекращаем поиск
 break;
 }
}
```

Мы довольно близко. Осталась одна вещь – Kohana::\$paths:

```
/**
 * @var array Включите пути для поиска файлов
 */
protected static $_paths = array(APPPATH, SYSPATH);
```

Нас не интересует приложение, поэтому мы можем пропустить часть APPPATH. Кроме того, SYSPATH – это путь к директории system. Поскольку большая часть классов Kohana находится в ней, разумно будет сделать ее умолчанием.

Когда класс автозагрузчика готов, мы используем Yii::registerAutoloader из index.php, чтобы зарегистрировать его. Важно зарегистрировать наш автозагрузчик после стандартного встроенного автозагрузчика Yii, поэтому вторым параметром Yii::registerAutoloader мы передаем значение true. Наш класс для изображений не в ядре и находится в модуле image, поэтому мы задаем пути к модулю image следующим образом:

```
EKohanaAutoloader::$_paths = array(Yii::getPathOfAlias(
 "application.vendors.Kohana.modules.image"));
```

## И ещё

Изменение размеров изображений – повседневная задача, с точки зрения как повторного использования, так и производительности ее лучше выделить и создать отдельный скрипт PHP, который будет изменять размеры изображений. Например, это позволит использовать следующий код:

```

```

Это означает: взять исходное изображение `avatar.png`, изменить его размеры до `100×100` пикселей. Возможные действия, которые предпримет скрипт `image.php`:

- если есть уже обработанное изображение, вернуть его;
- если готового изображения нет, то прочитать исходное изображение, изменить размеры и сохранить как обработанное.

Чтобы достичь большей производительности, можно настроить веб-сервер, чтобы существующие изображения отдавались напрямую, без задействования скрипта, а отсутствующие запрашивались у скрипта.

### Полезные материалы

Чтобы больше узнать об автозагрузке Yii и о Kohana, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/YiiBase/#registerAutoloader-detail>;
- <http://kohanaframework.org/>.

## Смотрите также

- Рецепт «Настройка автозагрузчика Yii» в этой главе.

## Использование PEAR внутри Yii

Ещё одно традиционное место для поиска библиотек PHP – это PEAR. В нем особое соглашение об именах, поэтому для использования кода из PEAR мы можем либо реализовать ещё один автозагрузчик, либо включать файлы напрямую. В этом рецепте мы используем класс `Text_Password` из PEAR для генерации случайного пароля.

## Подготовка

- Создайте новое приложение с помощью `yiiic webapp`.
- Убедитесь, что PEAR установлен и корректно настроен (<http://pear.php.net/manual/en/installation.php>).

## Как это делается

Пакет PEAR, который мы хотим использовать, находится на веб-странице [http://pear.php.net/package/Text\\_Password](http://pear.php.net/package/Text_Password). (*Важно*: есть «легкая установка» и документация).

1. Сначала установим пакет. Откройте консоль и наберите то, что предложено в «Разделе легкой настройки»:

```
pear install Text_Password
```

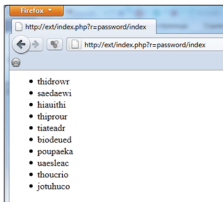
2. Результат должен быть следующим:

```
downloading Text_Password-1.1.1.tgz ...
Starting to download Text_Password-1.1.1.tgz (4,357 bytes)
.....
done: 4,357 bytes
install ok: channel://pear.php.net/Text_Password-1.1.1
```

3. Теперь можно попробовать использовать его. Мы сгенерируем 10 случайных паролей длиной 8 символов. Создайте файл `protected/controllers/PasswordController.php` следующего содержания:

```
class PasswordController extends CController
{
 public function actionIndex()
 {
 require "Text/Password.php";
 $textPassword = new Text_Password();
 $passwords = $textPassword->createMultiple(10, 8);
 echo "";
 foreach($passwords as $password)
 {
 echo "".$password."";
 }
 echo "";
 }
}
```





## ***Как это работает***

Использовать пакеты PEAR в Yii легко. Вам не придется настраивать Yii или писать дополнительный код, кроме того, что представлен в руководстве к пакету PEAR.

## ***И ещё***

Чтобы больше узнать о PEAR, обратитесь к следующим ресурсам:

- <http://pear.php.net/manual/en/installation.php>;
- [http://pear.php.net/package/Text\\_Password](http://pear.php.net/package/Text_Password);
- <http://pear.php.net/>.

## ***Смотрите также***

- Рецепт «Использование Zend Framework из Yii» в этой главе



## ГЛАВА 13.

# Развертывание

В этой главе мы рассмотрим:

- Изменение структуры директорий Yii.
- Перемещение приложения из корневой директории сервера.
- Совместное использование директории фреймворка.
- Перемещение части настроек в отдельные файлы.
- Использование нескольких конфигураций для упрощения развертывания.
- Реализация и исполнение заданий cron.
- Режим обслуживания.

## Введение

В этой главе приведены различные советы, которые особенно полезны при развертывании приложения и при разработке приложения в команде или когда вы просто хотите сделать свою среду разработки удобнее.

## Изменение структуры директорий Yii

В Yii предусмотрено соглашение о размещении директорий. Это позволяет значительно сгладить кривую обучения, но иногда для проекта лучше подходит особая структура директорий.

В этом рецепте мы переименуем несколько директорий и будем совместно использовать общие библиотеки в отдельных проектах.

План действий:

- переименовать `protected` в `app`;
- создать директорию `shared`, в которой мы сможем хранить компоненты, используемые совместно несколькими приложениями;
- переместить директорию `runtime` из `app`.

## Подготовка

- Загрузите копию фреймворка с веб-страницы Yii.
- Создайте структуру директорий:

```
/var/www/example/
 framework/
 www/
```

- Извлеките содержимое директории `framework` в `/var/www/example/framework/`.

## Как это делается

Выполните следующие шаги:

1. Перейдите в директорию `framework`.
2. Запустите `yii webapp /var/www/example/www/`.
3. Перейдите к `/var/www/example/www` и переименуйте `protected` в `app`.
4. Замените все вхождения `protected` на `app` в `index.php` и `index-test.php`.

Теперь у нас есть специальная директория `app`.

5. Создайте директорию `/var/www/shared`.
6. В конфигурации `main.php` добавьте:

```
// раскомментируйте следующее, чтобы определить
псевдоним для пути
Yii::setPathOfAlias('shared', '/var/www/shared');
...
// Это главная конфигурация веб-приложения. Все
изменяемые настройки
// CWebApplication можно изменить здесь.
return array(
 ...
 // автозагрузка классов модели и компонентов
 'import'=>array(
 ...
 'shared.*',
),
);
```

Это все. Теперь вы можете разместить свои собственные компоненты в папке `/var/www/shared`, и приложение будет о них знать. В дополнение к этому, если вы добавите такие же настройки в конфигурацию другого приложения, это приложение также сможет использовать эти компоненты.

1. Переместите директорию runtime из папки `/var/www/example/www/app` в `/var/www/example/runtime`.
2. Измените ваш файл конфигурации `main.php` следующим образом:

```
return array(
 ...
 'runtimePath' => Yii::getPathOfAlias('system').
 '/../runtime/',
 ...
);
```

3. Это все. Теперь наша папка runtime находится вне директории приложения.

## Как это работает

Имя директории приложения и путь определяются только в двух местах: `index.php` и `index-test.php`, поэтому их относительно легко изменить. Нам лишь нужно обновить два файла начальной загрузки после переименования директории приложения.

Создавая директорию для совместного использования, мы определяем специальный псевдоним пути. Это удобный способ использования дополнительных директорий, если вы часто к ним обращаетесь:

```
Yii::setPathOfAlias('shared', '/var/www/shared');
```

`setPathOfAlias` принимает два аргумента. Первый – имя, которое мы будем использовать, задавая опции путями, `Yii::getPathOfAlias` и `Yii::import`. Второй – реальный путь к директории. Поскольку мы хотим использовать компоненты прозрачно, то добавляем `shared.*` к списку импортов приложения. Это позволяет классам из директории `/var/www/shared` загружаться автоматически.

Последний путь, который мы изменим, будет путь к директории `runtime`. Для этого случая Yii определяет свойство приложения с именем `runtimePath`. Мы можем задать его, чтобы изменить используемый путь, следующим образом:

```
'runtimePath' => Yii::getPathOfAlias('system').'../runtime/',
```

Поскольку мы хотим поместить `runtime` на тот же уровень файловой структуры, где находится директория `framework`, то мы получаем путь к директории `framework` с помощью `getPathOfAlias`, а затем добавляем относительный путь к нашей директории `runtime`.

Приложение определяет ещё некоторые свойства, позволяющие вам изменять путь к расширениям, путь к переводам и путь к модулям. Другие пути, например путь к представлениям или путь к фай-

лам кеша, могут быть изменены в свойствах другого компонента. Для представления это `CController::viewPath`, а для кеша (файлового) — `CFileCache::cachePath`.

## И ещё

Чтобы больше узнать о путях к директориям Yii, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/YiiBase#getPathOfAlias-detail>;
- <http://www.yiiframework.com/doc/api/YiiBase#setPathOfAlias-detail>;
- <http://www.yiiframework.com/doc/api/YiiBase#import-detail>;
- <http://www.yiiframework.com/doc/api/CApplication#setExtensionPath-detail>;
- <http://www.yiiframework.com/doc/api/CApplication#setRuntimePath-detail>;
- <http://www.yiiframework.com/doc/api/CApplication#setLocaleDataPath-detail>;
- <http://www.yiiframework.com/doc/api/CModule#setModulePath-detail>;
- <http://www.yiiframework.com/doc/api/CController#viewPath-detail>.

## Смотрите также

- Рецепт «Перемещение приложения из корневой директории сервера» в этой главе.

## Перемещение приложения из корневой директории сервера

По умолчанию, создавая веб-приложение с помощью команды `yii webapp`, Yii помещает файл `index.php` и директорию `protected` в одно место, обычно это корневая директория сервера. Это позволяет запускать Yii в очень ограниченных средах, но в целях безопасности и для простоты разработки лучше держать код вне корневой директории сервера, если это возможно.

В этом рецепте мы рассмотрим, как переместить приложение Yii из корневой директории сервера, расположенной в `/var/www/website/www/`.

## Подготовка

- Скопируйте директорию `framework` в директорию `/var/www/website/`.
- Перейдите в `/var/www/website/framework/` и запустите `yii webapp /var/www/website/www/`.
- Вы должны получить файлы стандартного веб-приложения в `/var/www/website/www/`.

## Как это делается

Выполните следующие шаги:

1. Во-первых, нам нужно переместить `/var/www/website/www/protected/` в `/var/www/website/protected/`. Поскольку путь изменился, приложение теперь не сможет запуститься. Оба файла `index.php` и `index-test.php` требуют некоторых исправлений. Файл `index.php` содержит следующее:

```
// измените следующие пути, если понадобится
$yii=dirname(__FILE__).'../framework/yii.php';
$config=dirname(__FILE__).'../protected/config/main.php';

// удалите следующие строки перед использованием в
// рабочем режиме
defined('YII_DEBUG') or define('YII_DEBUG',true);
// укажите, сколько уровней стека вызовов должно
// показываться в каждом сообщении журнала
defined('YII_TRACE_LEVEL') or define('YII_TRACE_LEVEL',3);

require_once($yii);
Yii::createWebApplication($config)->run();
```

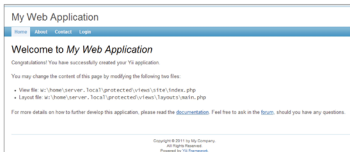
2. Определены два пути: путь к директории `framework`, который не изменился, и путь к директории `config`, который изменился. Обновим этот последний путь следующим образом:

```
$config=dirname(__FILE__).'../protected/config/main.php';
```

3. Нам нужно сделать то же самое с `index-test.php`:

```
$config=dirname(__FILE__).'../protected/config/test.php';
```

4. Вот и все. Теперь попробуйте запустить приложение, и вы должны увидеть стандартный экран приветствия, как на следующем скриншоте:



5. Теперь поправим ещё один путь в файле `protected/yiic.php`. Сделаем это следующим образом:

```
// измените следующие пути при необходимости
$yiic=dirname(__FILE__).'../framework/yiic.php';
$config=dirname(__FILE__).'../config/console.php';

require_once($yiic);
```

## Как это работает

Приложение Yii можно переместить в любое место файловой системы, которое мы пожелаем. Нужно только поправить пути в `index.php` и `index-test.php`. Этот простой ход дает несколько лучшую безопасность, поскольку код приложений не будет запускаться напрямую и не будет утечек в исходном коде через метафайлы контроля версий и т. д. Следовательно, если ваша рабочая среда позволяет переместить код приложений из корневой директории сервера, определенно следует над этим подумать.

## И ещё

Следующая статья даст вам представление, почему следует держать как можно меньше кода в корневой директории сервера:

- <http://www.smashingmagazine.com/2009/09/25/svn-strikes-back-a-serious-vulnerability-found/>.

## Смотрите также

- Рецепт «Совместное использование директории фреймворка» в этой главе.

## Совместное использование директории фреймворка

Если у вас запущено несколько проектов Yii на одном веб-сервере, то вы можете подумать о совместном использовании кода несколькими проектами. Это сэкономит некоторую часть дискового пространства и потребует меньше работы, когда вы будете обновлять свои приложения при смене версии фреймворка.

### Подготовка

Скопируйте содержимое директории `framework` в директорию `/var/www/common/yii/latest`.

### Как это делается

Выполните следующие шаги:

1. Перейдите в директорию `/var/www/common/yii/latest` и выполните команду `yiic webapp: /var/www/website1/www/`.
2. Перейдите в директорию `/var/www/common/yii/latest` и выполните команду `yiic webapp: /var/www/website2/www/`.
3. Вы должны получить файлы стандартного веб-приложения в директориях `/var/www/website1/www/` и `/var/www/website2/www/`.
4. Вот и всё. Попробуйте запускать приложения, чтобы убедиться в том, что всё работает.

### Как это работает

Выполнение `yiic webapp` из одного экземпляра фреймворка создаст приложения, обращающиеся к этому экземпляру. У нас два приложения используют один экземпляр фреймворка, поэтому при обновлении фреймворка нам достаточно обновить содержимое одной директории.

Для уже существующих приложений можно сделать то же самое, отредактировав в них файлы `index.php` и `index-test.php`, чтобы переменной `$yii` присваивались значения:

```
$yii='/var/www/common/yii/latest/yii.php';
```

### И ещё

Поскольку новые версии Yii могут в принципе нести некоторые не совместимые с более ранними версиями изменения (обычно таких



изменений не бывает при минорных релизах), то не мешает простой способ быстро откатить все изменения. Для этой цели вы должны держать несколько версий фреймворка в директории `/var/www/common/yii/` (например, 1.1.8 и 1.1.7). При обновлении приложения до версии 1.1.8 вы меняете путь к `yii.php` в `index.php` и `index-test.php` одного приложения и проверяете на регрессии. Если они обнаруживаются, вы либо исправляете их, либо откатываете все назад, изменяя путь, чтобы он указывал на 1.1.7. Если обновление прошло удачно, можно перейти к тесту следующего приложения.

## Смотрите также

- Рецепт «Перемещение приложения из корневой директории сервера» в этой главе.

## Перемещение части настроек в отдельные файлы

По умолчанию приложение Yii хранит все настройки веб-приложения в одном файле `protected/config/main.php`. То же относится и к консольному приложению. Это удобно и для ознакомления с фреймворком, и для небольших приложений, где хранение всех настроек в одном месте дает разработчику возможность быстро просмотреть все настройки приложения. Когда мы имеем дело с чем-то покрупнее, то можем столкнуться с некоторыми неудобствами, например:

- конфигурационный файл разрастается. В большом приложении также обычно используется много компонентов;
- если надо что-то настроить, то, скорее всего, придется те же изменения вносить ещё раз в конфигурацию консольного приложения.

## Подготовка

Создайте новое приложение с помощью `yii create webapp`.

## Как это делается

Выполните следующие шаги:

1. Мы рассмотрим сначала конфигурацию по умолчанию, чтобы выделить части, которые можно использовать повторно, а так-

же части, которые с наибольшей вероятностью будут слишком велики, чтобы держать их в одном файле:

```
<?php

// раскомментируйте следующее, чтобы определить
// псевдоним пути
// Yii::setPathOfAlias('local','path/to/local-
folder');
// Это главная конфигурация веб-приложения. Все изменяемые
// свойства CWebApplication можно определить здесь
return array(
 'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
 'name'=>'My Web Application',

 // предзагрузка компонента 'log'
 'preload'=>array('log'),

 // автозагрузка классов модели и компонентов
 'import'=>array(
 'application.models.*',
 'application.components.*',
),
 'modules'=>array(
 // раскомментируйте следующее, чтобы включить Gii
 /*
 'gii'=>array(
 'class'=>'system.gii.GiiModule',
 'password'=>'Enter Your Password Here',
 // если удалить, то Gii доступна только
 // с localhost
 // редактируйте осторожно
 'ipFilters'=>array('127.0.0.1',':::1'),
),
 */
),

 // компоненты приложения
 'components'=>array(
 'user'=>array(
 // включить авторизацию по куки
 'allowAutoLogin'=>true,
),
 // раскомментируйте, чтобы включить возможность
 // создания ссылок (URL) в path-формате
 /*
 'urlManager'=>array(
 'urlFormat'=>'path',
 'rules'=>array(
```

```

 '<controller:\w+>/<id:\d+>'=>'<controller>/
view',
 '<controller:\w+>/<action:\w+>/<id:\d+>'=>'
 <controller>/<action>',
 '<controller:\w+>/<action:\w+>'=>'
 <controller>/<action>',
),
),
*/
'db'=>array(
 'connectionString' =>
 'sqlite:'.dirname(__FILE__).'../data/
testdrive.db',
),
// раскомментируйте, чтобы использовать базу
// данных MySQL
/*
'db'=>array(
 'connectionString' =>
 'mysql:host=localhost;dbname=testdrive',
 'emulatePrepare' => true,
 'username' => 'root',
 'password' => '',
 'charset' => 'utf8',
),
*/
'errorHandler'=>array(
 // использовать действие 'site/error' для
 // отображения ошибок
 'errorAction'=>'site/error',
),
'log'=>array(
 'class'=>'CLogRouter',
 'routes'=>array(
 array(
 'class'=>'CFileLogRoute',
 'levels'=>'error, warning',
),
 // раскомментируйте следующее, чтобы
 // показывать сообщения из логов на
 // веб-страницах
 /*
 array(
 'class'=>'CWebLogRoute',
),
 */
),
),
),
),

```

```
),
 // параметры уровня приложения,
 // используя Yii::app()->params['paramName']
 'params'=>array(
 // это используется на странице контактов
 'adminEmail'=>'webmaster@example.com',
),
),
)
```

Список импортов и конфигурация модулей обычно не слишком велики. Это же можно сказать о конфигурации большинства компонентов. Что может расти с усложнением приложения, так это маршруты компонента `urlManager` и параметры уровня приложения. Что касается повторного использования, то, вероятно, нужны импорты, соединение с базой данных и параметры уровня приложения для веб-приложения и консольного приложения.

1. Теперь создадим следующие файлы настроек в директории `protected/config`:

- `routes.php`;
- `params.php`;
- `import.php`;
- `db.php`.

2. Далее нужно переместить соответствующие разделы из файла `main.php` в отдельные файлы. Сделаем это следующим образом:

```
// раскомментируйте следующее, чтобы определить
// псевдоним пути
// Yii::setPathOfAlias('local','path/to/local-folder');

// Это главная конфигурация веб-приложения. Все изменяемые
// свойства CWebApplication можно определить здесь
return array(
 'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
 'name'=>'My Web Application',

 // предзагрузка компонента 'log'
 'preload'=>array('log'),

 // автозагрузка классов модели и компонента
 'import'=>require(dirname(__FILE__).'/import.php'),

 'modules'=>array(
 // раскомментируйте следующее, чтобы включить Gii
```

```

/*
'gii'=>array(
 'class'=>'system.gii.GiiModule',
 'password'=>'Enter Your Password Here',
 // если удалить, то Gii доступна только с
 // localhost
 // редактируйте осторожно
 'ipFilters'=>array('127.0.0.1','::1'),
),
*/
),

// компоненты модели
'components'=>array(
 'user'=>array(
 // включить авторизацию по куки
 'allowAutoLogin'=>true,
),
 // раскомментируйте, чтобы включить возможность
 // создания ссылок (URL) в path-формате
 'urlManager'=>array(
 'urlFormat'=>'path',
 'rules'=>require(dirname(__FILE__).'/routes.php'),
),
 'db'=>require(dirname(__FILE__).'/db.php'),
 'errorHandler'=>array(
 // использовать действие 'site/error' для
 // отображения ошибок
 'errorAction'=>'site/error',
),
 'log'=>array(
 'class'=>'CLogRouter',
 'routes'=>array(
 array(
 'class'=>'CFileLogRoute',
 'levels'=>'error, warning',
),
 // раскомментируйте следующее, чтобы
 // показывать сообщения из логов на
 // веб-страницах
 /*
 array(
 'class'=>'CWebLogRoute',
),
 */
),
),
),

// параметры уровня приложения, которые можно

```

```
// получить через Yii::app()->params['paramName']
'params'=>require(dirname(__FILE__).'./params.php'),
);
```

3. Каждый новый конфигурационный файл будет содержать те же значения, что были в главном файле. Например, `protected/config/params.php` будет содержать следующее:

```
<?php
return array(
 // это используется на странице с контактами
 'adminEmail'=>'webmaster@example.com',
);
```

4. Теперь нам нужно изменить конфигурацию консольного приложения `protected/config/console.php`. Сделаем это таким образом:

```
// Это конфигурация консольного приложения yiiс. Все
// изменяемые свойства CConsoleApplication могут
// быть определены здесь
return array(
 'basePath'=>dirname(__FILE__).DIRECTORY_SEPARATOR.'..',
 'name'=>'My Console Application',
 // автозагрузка классов модели и компонента
 'import'=>require(dirname(__FILE__).'./import.php'),

 // компоненты приложения
 'components'=>array(
 'db'=>require(dirname(__FILE__).'./db.php'),
),
 // параметры уровня приложения, которые можно
 // получить через Yii::app()->params['paramName']
 'params'=>require(dirname(__FILE__).'./params.php'),
);
```

5. Вот и всё. Теперь у нас отдельные конфигурационные файлы для импорта, базы данных, маршрутов приложения и параметров приложения.

## Как это работает

Описанная выше техника основывается на том, что файлы настроек Yii являются родными файлами PHP с массивами:

```
<?php
return array(...);
```

Когда мы используем конструкцию `require`:

```
'db'=>require(dirname(__FILE__).' /db.php'), -
```

она читает указанный файл и, если в файле есть команда `return`, возвращает значение. Таким образом, перемещение части настроек из главного конфигурационного файла в отдельные файлы требует создания отдельного файла, помещения в него части конфигурации сразу после команды `return` и использования в главном файле команды `require`.

Если отдельные приложения (в нашем примере это веб-приложения и консольные приложения) требуют некоторых общих настроек, то мы можем использовать команды `require` для выделения этих настроек в отдельный файл.

## **И ещё**

Чтобы больше узнать о командах PHP `require` и `include`, обратитесь к следующим ресурсам:

- <http://php.net/manual/ru/function.require.php>;
- <http://php.net/manual/ru/function.include.php>.

## **Смотрите также**

- Рецепт «Использование нескольких конфигураций для упрощения развертывания» в этой главе.

# **Использование нескольких конфигураций для упрощения развертывания**

В некоторых случаях удобно использовать разные конфигурационные файлы для разных случаев. Например, можно использовать разные конфигурации для разработки приложения и для его эксплуатации.

В этом рецепте мы рассмотрим, как выбирать конфигурационный файл автоматически и как реализовывать наследование конфигурации.

## **Подготовка**

Создайте новое приложение с помощью `yiiic webapp`.

## Как это делается

Выполните следующие шаги:

1. Допустим, мы используем `http://example.com/` в эксплуатации, а `http://example.local` в разработке. Тогда мы можем выбрать соответствующую конфигурацию следующим образом:

```
// измените следующие части при необходимости
$yii=dirname(__FILE__).'../framework/yii.php';

if($_SERVER['HTTP_HOST']=='example.com')
{
 $config=dirname(__FILE__).'../protected/config/
production.php';
}
else
{
 // удалите следующие строки перед запуском в
 // эксплуатацию
 defined('YII_DEBUG') or define('YII_DEBUG',true);
 // укажите, сколько уровней стека вызовов должно
 // показываться в каждом сообщении логов
 defined('YII_TRACE_LEVEL') or define('YII_TRACE_LEVEL',3);
 $config=dirname(__FILE__).'../protected/config/
development.php';
}

require_once($yii);
Yii::createWebApplication($config)->run();
```

2. План состоит в том, чтобы оставить все общие настройки в `main.php` и переопределять настройки в зависимости от среды в файлах `development.php` и `production.php`. Таким образом, `main.php` не изменяется. Поместим в файл `protected/config/development.php` следующее:

```
<?php
return CMap::mergeArray(
 require(dirname(__FILE__).'../main.php'),
 array(
 'modules'=>array(
 'yii'=>array(
 'class'=>'system.gii.GiiModule',
 'password'=>false,
),
),
 'components'=>array(
```



```

 'db'=>array(
 'class'=>'system.db.CDbConnection',
 'connectionString'=>'mysql:host=localhost;
 dbname=example',
 'username'=>'root',
 'password'=>'',

 'charset'=>'utf8',
 'enableProfiling'=>true,
 'enableParamLogging'=>true,
),
 'log'=>array(
 'class'=>'CLogRouter',
 'routes'=>array(
 array(
 'class'=>'CProfileLogRoute',
),
),
),
),
);

```

3. Поместим в файл `protected/config/production.php` следующее:

```

<?php
return CMap::mergeArray(
 require(dirname(__FILE__) . '/main.php'),
 array(
 'components'=>array(
 'db'=>array(
 'class'=>'system.db.CDbConnection',
 'connectionString'=>'mysql:host=localhost;
 dbname=example',
 'username'=>'example',
 'password'=>'2WXyVNb4dBSEK3HW',
 'charset'=>'utf8',
 'schemaCachingDuration'=>60*60,
),
 'cache'=>array(
 'class'=>'CFileCache',
),
),
),
);

```

4. Вот и всё. Теперь мы можем просто загрузить файлы на рабочий сервер по адресу `http://example.com`, и приложение бу-

дет использовать конфигурационный файл `production.php`, который наследует все настройки из `main.php`, переопределяет некоторые из них и добавляет ещё некоторые настройки.

## Как это работает

Когда мы создаем экземпляр веб-приложения в `index.php`, можно передать один аргумент в `Yii::createWebApplication`. Этот аргумент – путь к конфигурационному файлу приложения. Таким образом, мы можем передавать один или другой путь к конфигурации в зависимости от неких критериев. В нашем случае критерием является имя узла, на котором выполняется наше приложение:

```
if($_SERVER['HTTP_HOST']=='example.com')
{
 $config=dirname(__FILE__).'../protected/config/production.php';
}
else
{
 // удалите следующие строки перед запуском в
 // эксплуатацию
 defined('YII_DEBUG') or define('YII_DEBUG',true);
 // укажите, сколько уровней стека вызовов должно
 // показываться в каждом сообщении логов
 defined('YII_TRACE_LEVEL') or define('YII_TRACE_LEVEL',3);
 $config=dirname(__FILE__).'../protected/config/
 development.php';
}
```

Если имя узла равно `example.com`, то используется эксплуатационный файл конфигурации. Иначе используется файл конфигурации для разработки и включается отладка.

Вы можете использовать что угодно в качестве критерия выбора конфигурационного файла. Например, можно выполнять приложение в режиме отладки на рабочем сервере, если получена куки с определенным именем и значением.

Поскольку большинство настроек, например параметры приложения и маршруты, будут одинаковыми в разных конфигурациях, то они останутся в `main.php`. Кроме того, поскольку конфигурационные файлы Yii являются массивами PHP, мы можем использовать `CMap::mergeArray` для реализации наследования следующим образом:

```
return CMap::mergeArray(
 require(dirname(__FILE__).' /main.php'),
 array (...)
```

```
};
```

## И ещё

Чтобы больше узнать о том, как именно склеиваются конфигурационные файлы, обратитесь к следующему ресурсу:

- <http://www.yiiframework.com/doc/api/CMap#mergeArray>.

## Смотрите также

- Рецепт «Перемещение части настроек в отдельные файлы» в этой главе.

# Реализация и исполнение заданий cron

Иногда приложение требует некоторого фонового сопровождения, например обновления карты сайта или сбора статистики. Общепринято реализовывать такие задачи, используя планировщик (cron). В Yii есть два способа делать это:

- 1) эмулировать браузер для вызова контроллера веб-приложения;
- 2) использовать консоль для передачи задания планировщику.

В этом рецепте мы рассмотрим, как реализовать оба способа. Для этого рецепта мы реализуем запись текущей временной отметки в файл `timestamp.txt` в директории `protected`.

## Подготовка

Создайте новое приложение с помощью `yiiic webapp`.

## Как это делается

Выполните следующие шаги:

1. Создайте файл `protected/controllers/CronController.php` следующего содержания:

```
<?php
class CronController extends CController
{
 public function actionIndex()
 {
```

```
$filename = Yii::getPathOfAlias
 ("application")."/timestamp.txt";
file_put_contents($filename, time());
}
)
```

2. Теперь нам нужен способ вызвать этот контроллер. Поскольку это контроллер веб-приложения, то нам нужно как-то эмулировать браузер. В Linux можно использовать одну из следующих команд в задании планировщика (crontab):

```
GET http://example.com/index.php?r=cron
wget -O - http://example.com/index.php?r=cron
lynx --dump http://example.com/index.php?r=cron >/dev/null
```

Когда мы используем контроллер таким способом, нам нужно убедиться, что он используется только как задание планировщика. Например, можно проверить значение специфической переменной `$_GET`.

3. Создайте `protected/commands/CronCommand.php` следующего содержания:

```
<?php
class CronCommand extends CConsoleCommand
{
 public function run($args)
 {
 $filename = Yii::getPathOfAlias
 ("application")."/timestamp.txt";
 file_put_contents($filename, time());
 }
}
```

4. Можно использовать следующее в задании cron, чтобы выполнить ее:

```
/path/to/./yiic cron
```

Заметьте, что в задании cron должен быть указан полный путь.

## Как это работает

GET, wget и lynx запрашивают страницу по HTTP, поэтому мы можем использовать их для запуска нормального выполнения веб-приложения. Этот метод имеет свои достоинства и недостатки. Главное достоинство состоит в том, что мы выполняем веб-приложение, и контекст такой же самый, как в других частях приложения. Главный недоста-

ток – недостаточная безопасность. Даже если мы используем `$_GET` в качестве пароля, нет гарантии, что он не будет раскрыт.

Второй метод гораздо более безопасен, потому что консольная команда может быть выполнена, только если есть доступ через SSH на сервер. Недостаток тот, что контекст отличается от контекста веб-приложения.

## **И ещё**

Другим способом решения проблемы является использование очереди сообщений. Большинство реализаций возьмут на себя проблемы с параллелизмом и позволят вам выполнять обработку почти мгновенно, если запросов не слишком много. Реализации, которые стоит изучить:

- <http://www.rabbitmq.com/>;
- <http://kr.github.com/beanstalkd/>;
- <http://activemq.apache.org/>;
- <http://gearman.org/>;
- <https://github.com/s0enke/dropr/>;
- <http://www.zeromq.org/>.

## **Полезные материалы**

Чтобы больше узнать о консольных приложениях Yii, обратитесь к следующему ресурсу:

- <http://www.yiiframework.com/doc/guide/ru/topics/console>.

## **Смотрите также**

- Рецепт «Создание консольных команд» в главе 8 «Расширение Yii».

## **Режим обслуживания**

Иногда нужно провести тонкую настройку приложения или восстановить базу данных из резервной копии. Работая над подобными задачами, нежелательно позволять кому-либо использовать приложение, так как это может привести к потере последних сообщений или раскрытию деталей реализации приложения.

В этом рецепте мы рассмотрим, как показать всем, кроме разработчика, сообщение о проводимых работах.

## Подготовка

Создайте новое приложение с помощью `yiiic webapp`.

## Как это делается

Выполните следующие шаги:

1. Во-первых, нужно создать `protected/controllers/MaintenanceController.php` следующего содержания:

```
<?php
class MaintenanceController extends CController
{
 public function actionIndex()
 {
 $this->renderPartial("index");
 }
}
```

2. Потом создайте представление `protected/views/maintenance/index.php` следующим образом:

```
<!doctype html>
<head>
 <meta charset="utf-8" />
 <title><?php echo CHtml::encode(Yii::app()->name)?>
 на обслуживании</title>
</head>
<body>
 <h1><?php echo CHtml::encode(Yii::app()->name)?>
 на обслуживании</h1>
 <p>Мы скоро вернемся. Если же нас слишком долго нету,
 пожалуйста, сбросьте письмо на
 <?php echo Yii::app()->params
 ['adminEmail']?>.</p>
 <p>Тем временем у вас есть время выпить кофе,
 почитать книгу или проверить почту.</p>
</body>
```

3. Теперь нам нужно добавить одну строку кода к `protected/config/main.php` следующим образом:

```
return array(
 'catchAllRequest'=>file_exists(dirname(__FILE__).'/.
 maintenance')
 && !(isset($_COOKIE['secret']) &&
 $_COOKIE['secret']=="password") ?
 array('maintenance/index') : null,
 ...
);
```

4. Вот и все. Теперь, чтобы войти в режим обслуживания, нужно создать файл `.maintenance` в директории `protected/config/`.

### My Web Application is under maintenance

We'll be back soon. If we aren't back for too long, please drop a message to [webmaster@example.com](mailto:webmaster@example.com).

Meanwhile, it's a good time to get a cup of coffee, to read a book or to check email.

Для того чтобы вернуть приложение в нормальный режим, просто удалите файл `.maintenance`. Чтобы увидеть веб-сайт в режиме обслуживания, вы можете создать куки с именем `secret` и значением, равным паролю.

## Как это работает

Приложение Yii предлагает способ перехватывать все возможные запросы и перенаправлять их одному действию контроллера. Вы можете сделать это присвоением `CWebApplication::catchAllRequests` массива, содержащего маршрут приложения, следующим образом:

```
'catchAllRequest'=>array('maintenance/index'),
```

Контроллер обслуживания сам по себе ничем не выдающийся; он просто отображает представление с текстом.

Нам нужен легкий способ включать и выключать режим обслуживания. Поскольку конфигурационный файл приложения является обычным файлом PHP, мы можем сделать это с помощью простой проверки существования файла:

```
file_exists(dirname(__FILE__).'/.maintenance')
```

Вдобавок мы проверяем содержание куки, чтобы обойти режим обслуживания. Мы делаем это так:

```
!(isset($_COOKIE['secret']) && $_COOKIE['secret']=="password")
```

## И ещё

Чтобы больше узнать о том, как перехватывать все запросы в приложении Yii, и о пригодном для производства решении режима обслуживания, обратитесь к следующим ресурсам:

- <http://www.yiiframework.com/doc/api/CWebApplication/#catchAllRequest-detail>;
- <https://github.com/karagodin/MaintenanceMode>.

***Смотрите также***

- Рецепт «Перемещение части настроек в отдельные файлы» в этой главе.
- Рецепт «Использование нескольких конфигураций для упрощения развертывания» в этой главе.



Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЪЯНС БУКС» наложенным платежом, выслав открытку или письмо по почтовому адресу: **123242, Москва, а/я 20** или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. **(499) 725-50-27, 725-54-09**.

Электронный адрес **books@alians-kniga.ru**.

Макаров Александр Сергеевич

## **Yii. Сборник рецептов**

Более 80 рецептов, которые помогут вам  
использовать PHP-фреймворк Yii

Главный редактор *Мовчан Д. А.*  
dm@dmk-press.ru

Корректор *Синяева Г. И.*

Верстка *Паранская Н. В.*

Дизайн обложки *Мовчан А. Г.*

Подписано в печать 18.12.2012. Формат 60×90 <sup>1</sup>/<sub>16</sub>.

Гарнитура «Петербург». Печать офсетная.

Усл. печ. л. 23,25. Тираж 300 экз.

Веб-сайт издательства: **www.dmk-press.ru**

## Yii. Сборник рецептов

Данная книга познакомит вас с самыми важными особенностями и внутренними механизмами РНР-фреймворка Yii, что позволит вам использовать его наиболее эффективно.

Сборник поможет вам изучить часто упускаемые из вида, но очень полезные особенности фреймворка и повысить свой уровень как разработчика приложений.

В книге показаны не только многие упрощающие разработку трюки, но и перечислены нежелательные приемы. Наиболее интересные темы касаются разработки приложений и расширений, обработки ошибок, отладки, вопросов безопасности и улучшения производительности.

### Эта книга расскажет вам...

- Как использовать внутренние возможности Yii, такие как события или коллекции;
- Как выжать максимум из MVC и сделать его компоненты пригодными для повторного использования;
- Как работать с jQuery, JavaScript и AJAX в стиле Yii;
- Как использовать скрытые возможности виджетов Yii и реализовать свои виджеты;
- Как избежать ошибок, используя разработку через тестирование;
- Как использовать компоненты Yii, такие как гриды и провайдеры данных;
- Как автоматизировать отслеживание ошибок и научиться разбирать ошибки Yii;
- Как получить максимальную производительность.



#### Внутри книги вы найдете:

- четкие пошаговые инструкции;
- хорошо подобранный набор рецептов, охватывающий наиболее важные темы;
- оптимальные решения проблем;
- полную аргументацию принятия решений;
- варианты применения рецепта к другим ситуациям.



Интернет-магазин: [www.dmk-press.ru](http://www.dmk-press.ru)  
Книга-почтой: [orders@aliens-kniga.ru](mailto:orders@aliens-kniga.ru)  
Оптовая продажа: «Альянс-книга»  
(499)725-5409, [books@aliens-kniga.ru](mailto:books@aliens-kniga.ru)

ISBN 978-5-94074-786-4



9 785940 747864 >