ОГЛАВЛЕНИЕ

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, предстваленных 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 программирвоания 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.

ОБ АВТОРЕ

Александр Макаров – опытный инженер из Воронежа, успевший попробовать себя в роли PM. 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 rmcreative.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 рукводстве: 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 покрытии 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, гридам 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 расказывает, 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
{
    // скрытие $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 класс: `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 инициирует 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.