Урок 5

MVC и ООП.

В этом уроке

- Вы узнаете, что такое перегрузка свойств и методов класса;
- научитесь делать автоматическую загрузку классов;
- изучите архитектуру системы, реализованной с помощью MVC и ООП;
- познакомитесь с технологией создания одной точки входа на сайте.

Наконец-то мы применим ООП для чего-то полезного. По окончании данного урока Вы получите систему, реализованную согласно концепции MVC и ООП. Но для начала изучим небольшую вспомогательную тему про перегрузку свойств и методов класса.

1. Перегрузка методов

Понятие перегрузки в PHP кардинально отличается от других языков. Перегрузка в PHP означает возможность динамически создавать свойства и методы с помощью объявления специальных методов внутри класса. Всего существует три метода: __get, __set и __call.

С помощью объявления метода __get внутри класса мы можем определить поведение объекта в ситуации, когда у него попросят несуществующее свойство, например:

```
<?php

class Some
{
    public $name;

    public function __construct($name){
        $this->name = $name;
    }

    public function __get($name){
        return "Параметр $name отсутствует";
    }
}

$some = new Some('aaa');
echo $some->age;          // Выведет «Параметр age отсутствует»
```

Данный метод принимает один параметр – имя свойства, к которому было совершено обращение. Обратите внимание на то, что он вызывается автоматически при обращении к несуществующему полю класса.

Аналогичным образом работают методы __set и __call, только первый из них отвечают за обработку установки значения в свойство, которого нет, а второй - за обращение к несуществующему методу класса. Например:

```php
<?php

class Some
{
    public $name;

    public function __construct($name){
        $this->name = $name;
    }

    public function view(){
        echo "<h1>$this->name</h1>";
    }

    public function __get($name){
        return null;
    }

    public function __set($name, $value){
        $this->$name = $value;
    }

    public function __call($name, $params){
        die("У этого класса нет метода $name, с параметрами  $params");
    }
}

$some = new Some('aaa');
$some->name = 'bbb';
$some->view();

echo $some->a; // обращение к свойству, которого нет
$some->b = 'ccc'; // установка свойства, которого нет

$some->what(); // обращение к методу, которого нет
```

Метод __set принимает два параметра: имя свойства и его значение. В данном примере мы создаём такое поле классу.

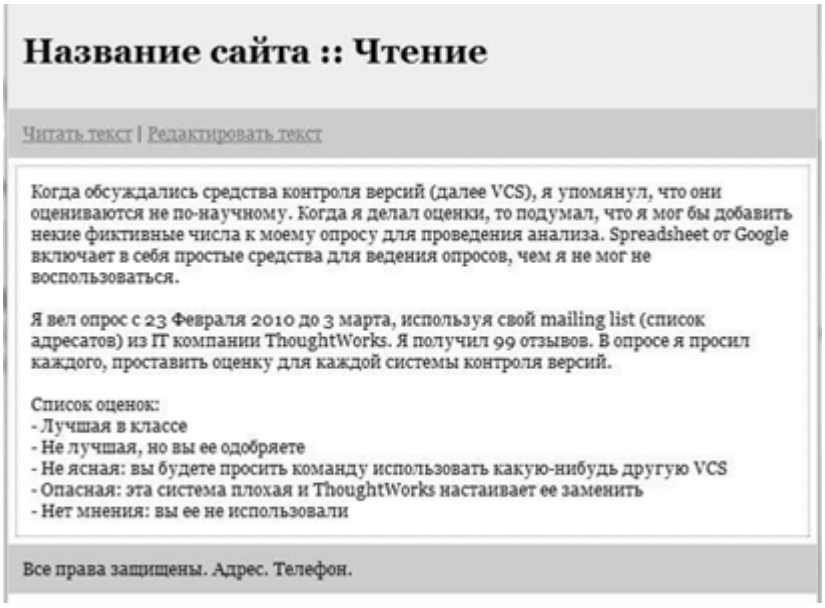
Метод __call также принимает два параметра: имя вызванного метода и параметры, переданные в него. В данном примере мы завершаем PHP-скрипт и выводим сообщение об ошибке.

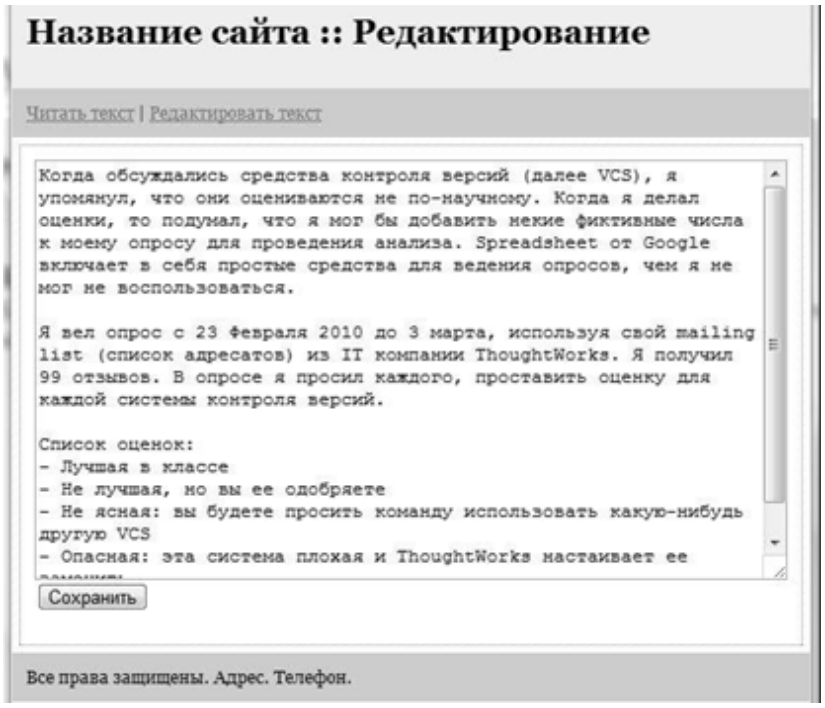
Данная тема может показаться достаточно странной, однако именно на ней будет основана одна из частей рассматриваемой в уроке системы.

2. Описание примера

Откройте пример «02_without_opp» из архива с исходниками. В нём лежит начальная заготовка, которую мы будем приводить к системе MVC + ООП.

Данный пример кода является очень простым - по сути это одностраничный сайт. У нас есть два контроллера: редактирование страницы и её просмотр.





Работа с базой данных также отсутствует, содержимое страницы хранится в обычном текстовом файле.

Сейчас мы начнём переделывать данный пример. Не удивляйтесь, когда Вы увидите, что кода станет в 2-3 раза больше. Дело в том, что это временное явление. Для любого чуть более сложного сайта система, которую мы создадим, будет являться гораздо более удобной.

Здесь также очень важно понимать, что MVC+ООП и, вообще, любые системы со сложной архитектурой хороши для больших проектов. А вот, например, делать сайт визитку с помощью какого-нибудь современного фреймворка, это всё равно, что стрелять из пулемёта по комарам.

3. Архитектура системы

В архиве с исходниками находятся все этапы эволюции данной системы, которые мы рассматривали на занятии. Здесь мы разберём её последнюю и самую лучшую конфигурацию.

Сначала расскажем про общую идею системы. В ней существует одна точка входа index.php, которая анализирует GET-параметры и передаёт управление выбранному контроллеру. Каждый контроллер – это отдельный класс, который отвечает за конкретную сущность, а его методы – за действия, которые может совершить пользователь. Например, в домашнем задании Вам нужно будет сделать контроллер C_Articles с пятью методами, стандартными для нашего блога.

При этом существуют ещё два контроллера, от которых наследуются все остальные, C_Base и C_Controller. Они отвечают за более общие вещи: C_Controller - это ядро системы, которое не нужно менять ни для какого сайта, а C_Base – базовый контроллер для текущего сайта.

Схематично это можно изобразить следующим образом:

Здесь уровни показывают наследование классов. Главная идея, почему это должно быть удобно, заключается в следующем: в C_Base выносится всё общее, что должно быть на каждой странице сайта. Это может быть не только формирование базового шаблона, но и, например, проверка авторизации, установка соединения с базой данных, формирование боковых колонок сайта и т.п. Затем, мы не прописываем этот код каждый раз заново, а просто наследуем контроллер для текущей сущности от C_Base.

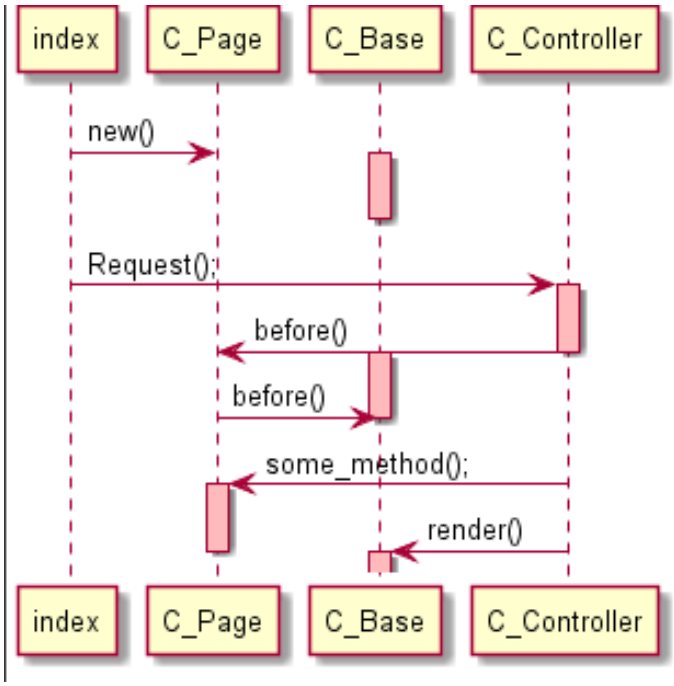
У каждого сущностного контроллера основные методы начинаются с префикса action_. Каждый из них отвечает за события на определённой странице. Также у этих контроллеров есть метод before, который наследуется от родительского класса. Данный метод предназначен для выполнения перед основным. Например, в нём C_Base может установить для наследника некоторые настройки.

Рассмотрим всё происходящее последовательно. Начнём с точки входа:

```
<?php

function __autoload($classname){
    include_once("c/$classname.php");
}

$action = 'action_';
$action .= (isset($_GET['act'])) ? $_GET['act'] : 'index';

switch ($_GET['c'])
{
    case 'articles':
        $controller = new C_Page();
        break;
    default:
        $controller = new C_Page();
}

$controller->Request($action);
```

Фукнция __autoload позволяет навсегда забыть о необходимости подключать контроллеры вручную. При попытке создать экземпляр класса, который не был подключён с помощью include, PHP-скрипт сам вызывает __autoload и передаёт туда имя класса. Нам остаётся лишь подключить его, используя это имя.

Скрипт всегда ориентируется на два GET-параметра: c – имя контроллера и act – имя вызываемого метода. По умолчанию будут использоваться метод «action_index» и контроллер «C_Page». Дальнейшее выполнение программы выглядит следующим образом:



Мы вызываем метод Request, который есть у самого базового контроллера – C_Controller. Он разбивает выполнение программы на три части:

1) вызов метода before();
2) вызов основного запрошенного метода;
3) вызов генерации базового шаблона и вывода содержимого на экран.

Метод before позволяет родителю и наследнику обменяться данными. Огромную силу этого приёма Вы увидите в 8-ом уроке - фактически это и есть сила ООП, а пока что в C_Base::before просто устанавливается название сайта, которое будет дополнять контроллер C_Page.

Основной запрошенный метод проводит всю работу на конкретной странице. Например, вот как коротко теперь выглядит код для страницы просмотра текста:

```
public function action_index(){
   $this->title .= '::Чтение';
   $text = text_get();
   $this->content = $this->Template('v/v_index.php', array('text' => $text));
}
```

Мы переопределяем title страницы, затем просим модель отдать текст из файла и генерируем внутренний шаблон.

И всё! Никаких повторений кода, так как вся логика уже вынесена в ядро системы, а общие действия для страничек сайта - в C_Base.

Код для страницы редактирования будет не намного больше:

```
public function action_edit(){
    $this->title .= '::Редактирование';

    if($this->isPost())
    {
        text_set($_POST['text']);
        header('location: index.php');
        exit();
    }

    $text = text_get();
    $this->content = $this->Template('v/v_edit.php',array('text' = >$text));
}
```

Фактически происходит всё то же самое, только ещё обрабатываем возможную отправку формы. Заметьте, что мы нигде не задумываемся о генерации базового шаблона, так как об этом позаботятся C_Base и C_Controller.

Чтобы понять, что это вовсе не магия, рассмотрим ключевой метод C_Controller – Request.

```
public function Request($action)
{
    $this->before();
    $this->$action();
    $this->render();
}
```

Он принимает параметр $action – имя основного вызываемого метода. Request сначала вызывает обмен данными – before(), а затем выдаёт очень хитрую запись: $this->$action(). Т.е., мы наугад обращаемся к текущему контроллеру по переданному методу. Поэтому необходимо обработать ситуацию его отсутствия. Самое время вернуться к пункту №1 урока – перегрузке методов.

```
public function __call($name, $params){
    die('Не пишите бред в url-адресе!!!');
}
```

Получается, что если будет передано название метода, который не существует, то мы завершим скрипт. Важно понимать, что это только функция для примера, а в реальности в такой ситуации нужно показывать сообщение об ошибке 404.

Подводя итог по данной системе, составим алгоритм её расширения:

1) создать новый контроллер и унаследовать его от C_Base;
2) создать метод action_index, который будет вызываться по умолчанию;
3) в index.php добавить в switch запись о соотнесении параметра «c» и данного контроллера.

Такие же механизмы, которые мы разбирали сегодня на уроке используются в большинстве современных фреймворков. Поэтому обязательно постарайтесь их освоить.

Самоконтроль

- ✓ Что такое перегрузки в PHP
- ✓ __get
- ✓ __set
- ✓ __call
- ✓ Принцип работы __autoload
- ✓ За что отвечает класс C_Controller
- ✓ За что отвечает класс C_Base
- ✓ За что отвечают все остальные контроллеры
- ✓ Как расширять данную систему

Домашнее задание

Переписать блог с использованием системы, которую мы рассмотрели на текущем занятии.