Vitaly Swipe @vitalyswipe read-only
Пользователь

27 августа 2012 в 07:03

Реализация MVC паттерна на примере создания сайта-визитки на PH

Разработка веб-сайтов*, PHP*



Как вы уже догадались из названия статьи, сегодня речь пойдет о самом популярном, разве что после Singleton, шаблоне проектирования MVC, хотя такое сравнение не совсем уместно. Понимание концепции MVC может помочь вам в рефакторинге и разрешении неприятных ситуаций в которые, возможно попа проект. Дабы восполнить пробел, мы реализуем шаблон MVC на примере простого сайта-визитки.

Оглавление

Введение

Многие начинают писать проект для работы с единственной задачей, не подразумевая, что это может вырасти в многопользовательскую систему управлен допустим, контентом или упаси бог, производством. И всё вроде здорово и классно, всё работает, пока не начинаешь понимать, что тот код, который написа состоит целиком и полностью из костылей и хардкода. Код перемешанный с верстской, запросами и костылями, неподдающийся иногда даже прочтению. Возникает насущная проблема: при добавлении новых фич, приходится с этим кодом очень долго и долго возиться, вспоминая «а что же там такое написан было?» и проклинать себя в прошлом.

Вы можеть быть даже слышали о шаблонах проектирования и даже листали эти прекрасные книги:

- Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидесс «Приемы объектно ориентированного проектирования. Паттерны проектирования»;
- М. Фаулер «Архитектура корпоративных программных приложений».

А многие, не испугавшись огромных руководств и документаций, пытались изучить какой-либо из современных фреймворков и столкнувшись со сложностью понимания (в силу наличия множества архитектруных концепций хитро увязанных между собой) отложили изучение и применение современных интсрумен «долгий ящик».

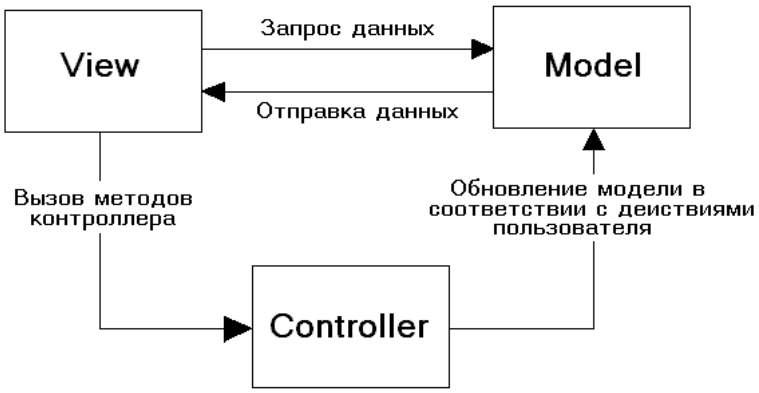
Представленная статья будет полезна в первую очередь новичкам. Во всяком случае, я надеюсь что за пару часов вы сможете получить представление о реализации MVC паттерна, который лежит в основе всех современных веб-фреймворков, а также получить «пищу» для дальнейших размышлений над тем стоит делать». В конце статьи приводится подборка полезных ссылок, которые также помогут разобраться из чего состоят веб-фреймворки (помимо MVC) они работают.

Прожженные PHP-программисты вряд ли найдут в данной статье что-то новое для себя, но их замечания и комментарии к основному тексту были бы очень Т.к. без теории практика невозможна, а без практики теория бесполезна, то сначала будет чуть-чуть теории, а потом перейдем к практике. Если вы уже зна концепцией MVC, можете пропустить раздел с теорией и сразу перейти к практике.

1. Теория

Шаблон MVC описывает простой способ построения структуры приложения, целью которого является отделение бизнес-логики от пользовательского интер
В результате, приложение легче масштабируется, тестируется, сопровождается и конечно же реализуется.

Рассмотрим концептуальную схему шаблона MVC (на мой взгляд — это наиболее удачная схема из тех, что я видел):



В архитектуре MVC модель предоставляет данные и правила бизнес-логики, представление отвечает за пользовательский интерфейс, а контроллер обеспе
взаимодействие между моделью и представлением.

Типичную последовательность работы MVC-приложения можно описать следующим образом:

1. При заходе пользователя на веб-ресурс, скрипт инициализации создает экземпляр приложения и запускает его на выполнение.
   При этом отображается вид, скажем главной страницы сайта.
2. Приложение получает запрос от пользователя и определяет запрошенные контроллер и действие. В случае главной страницы, выполняется действие
   умолчанию (*index*).
3. Приложение создает экземпляр контроллера и запускает метод действия,
   в котором, к примеру, содержаться вызовы модели, считывающие информацию из базы данных.
4. После этого, действие формирует представление с данными, полученными из модели и выводит результат пользователю.

Модель — содержит бизнес-логику приложения и включает методы выборки (это могут быть методы ORM), обработки (например, правила валидации) и
предоставления конкретных данных, что зачастую делает ее очень толстой, что вполне нормально.
Модель не должна напрямую взаимодействовать с пользователем. Все переменные, относящиеся к запросу пользователя должны обрабатываться в контро
Модель не должна генерировать HTML или другой код отображения, который может изменяться в зависимости от нужд пользователя. Такой код должен
обрабатываться в видах.
Одна и та же модель, например: модель аутентификации пользователей может использоваться как в пользовательской, так и в административной части
приложения. В таком случае можно вынести общий код в отдельный класс и наследоваться от него, определяя в наследниках специфичные для подприлож
методы.

Вид — используется для задания внешнего отображения данных, полученных из контроллера и модели.
Виды содержат HTML-разметку и небольшие вставки PHP-кода для обхода, форматирования и отображения данных.
Не должны напрямую обращаться к базе данных. Этим должны заниматься модели.
Не должны работать с данными, полученными из запроса пользователя. Эту задачу должен выполнять контроллер.
Может напрямую обращаться к свойствам и методам контроллера или моделей, для получения готовых к выводу данных.
Виды обычно разделяют на общий шаблон, содержащий разметку, общую для всех страниц (например, шапку и подвал) и части шаблона, которые использу
отображения данных выводимых из модели или отображения форм ввода данных.

Контроллер — связующее звено, соединяющее модели, виды и другие компоненты в рабочее приложение. Контроллер отвечает за обработку запросов
пользователя. Контроллер не должен содержать SQL-запросов. Их лучше держать в моделях. Контроллер не должен содержать HTML и другой разметки. В
выносить в виды.
В хорошо спроектированном MVC-приложении контроллеры обычно очень тонкие и содержат только несколько десятков строк кода. Чего, не скажешь о St
Controllers (SFC) в CMS Joomla. Логика контроллера довольно типична и большая ее часть выносится в базовые классы.
Модели, наоборот, очень толстые и содержат большую часть кода, связанную с обработкой данных, т.к. структура данных и бизнес-логика, содержащаяся в
обычно довольно специфична для конкретного приложения.

1.1. Front Controller и Page Controller

В большинстве случае, взаимодействие пользователя с web-приложением проходит посредством переходов по ссылкам. Посмотрите сейчас на адресную ст
браузера — по этой ссылке вы получили данный текст. По другим ссылкам, например, находящимся справа на этой странице, вы получите другое содержим
Таким образом, ссылка представляет конкретную команду web-приложению.

Надеюсь, вы уже успели заметить, что у разных сайтов могут быть совершенные разные форматы построения адресной строки. Каждый формат может
отображать архитектуру web-приложения. Хотя это и не всегда так, но в большинстве случаев это явный факт.

Рассмотрим два варианта адресной строки, по которым показывается какой-то текст и профиль пользователя.

Первый вариант:

1. www.example.com/article.php?id=3
2. www.example.com/user.php?id=4

Здесь каждый сценарий отвечает за выполнение определённой команды.

Второй вариант:

1. www.example.com/index.php?article=3
2. www.example.com/index.php?user=4

А здесь все обращения происходят в одном сценарии **index.php**.

Подход с множеством точек взаимодействия вы можете наблюдать на форумах с движком phpBB. Просмотр форума происходит через сценарий **viewforum** просмотр топика через **viewtopic.php** и т.д. Второй подход, с доступом через один физический файл сценария, можно наблюдать в моей любимой CMS MOI все обращения проходят через **index.php**.

Эти два подхода совершенно различны. Первый — характерен для шаблона контроллер страниц (Page Controller), а второй подход реализуется паттерном контроллер запросов (Front Controller). Контроллер страниц хорошо применять для сайтов с достаточно простой логикой. В свою очередь, контроллер запро объединяет все действия по обработке запросов в одном месте, что даёт ему дополнительные возможности, благодаря которым можно реализовать более трудные задачи, чем обычно решаются контроллером страниц. Я не буду вдаваться в подробности реализации контроллера страниц, а скажу лишь, что в практической части будет разработан именно контроллер запросов (некоторое подобие).

1.2. Маршрутизация URL

Маршрутизация URL позволяет настроить приложение на прием запросов с URL, которые не соответствуют реальным файлам приложения, а также исполь ЧПУ, которые семантически значимы для пользователей и предпочтительны для поисковой оптимизации.

К примеру, для обычной страницы, отображающей форму обратной связи, URL мог бы выглядеть так:
http://www.example.com/contacts.php?action=feedback

Приблизительный код обработки в таком случае:

```
switch($_GET['action'])
{
        case "about" :
                require_once("about.php"); // страница "О Нас"
                break;
        case "contacts" :
                require_once("contacts.php"); // страница "Контакты"
                break;
        case "feedback" :
                require_once("feedback.php"); // страница "Обратная связь"
                break;
        default :
                require_once("page404.php"); // страница "404"
        break;
}
```

Думаю, почти все так раньше делали.

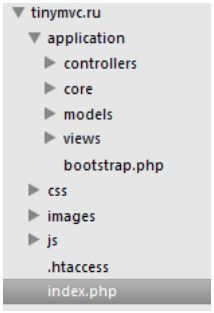
С использованием движка маршрутизации URL вы сможете для отображения той же информации настроить приложение на прием таких запросов:
http://www.example.com/contacts/feedback

Здесь contacts представляет собой контроллер, а feedback — это метод контроллера contacts, отображающий форму обратной связи и т.д. Мы еще вернемс этому вопросу в практической части.

Также стоит знать, что маршрутизаторы многих веб-фреймворков позволяют создавать произвольные маршруты URL (указать, что означает каждая часть правила их обработки.
Теперь мы обладаем достаточными теоретическими знаниями, чтобы перейти к практике.

2. Практика

Для начала создадим следующую структуру файлов и папок:



Забегая вперед, скажу, что в папке core будут храниться базовые классы Model, View и Controller.
Их потомки будут храниться в директориях controllers, models и views. Файл **index.php** это точка в хода в приложение. Файл **bootstrap.php** инициирует загру
приложения, подключая все необходимые модули и пр.

Будем идти последовательно; откроем файл index.php и наполним его следующим кодом:

```
ini_set('display_errors', 1);
require_once 'application/bootstrap.php';
```

Тут вопросов возникнуть не должно.

Следом, сразу же перейдем к фалу **bootstrap.php**:

```
require_once 'core/model.php';
require_once 'core/view.php';
require_once 'core/controller.php';
require_once 'core/route.php';
Route::start(); // запускаем маршрутизатор
```

Первые три строки будут подключать пока что несуществующие файлы ядра. Последние строки подключают файл с классом маршрутизатора и запускают
выполнение вызовом статического метода start.

2.1. Реализация маршрутизатора URL

Пока что отклонимся от реализации паттерна MVC и займемся мрашрутизацией. Первый шаг, который нам нужно сделать, записать следующий код в **.htac**

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule .* index.php [L]
```

Этот код перенаправит обработку всех страниц на **index.php**, что нам и нужно. Помните в первой части мы говорили о Front Controller?!

Маршрутизацию мы поместим в отдельный файл **route.php** в директорию core. В этом файле опишем класс Route, который будет запускать методы контрол
которые в свою очередь будут генерировать вид страниц.

Содержимое файла route.php

```
class Route
{
	static function start()
	{
		// контроллер и действие по умолчанию
		$controller_name = 'Main';
		$action_name = 'index';

		$routes = explode('/', $_SERVER['REQUEST_URI']);

		// получаем имя контроллера
		if ( !empty($routes[1]) )
		{
			$controller_name = $routes[1];
		}

		// получаем имя экшена
		if ( !empty($routes[2]) )
		{
			$action_name = $routes[2];
		}
```

```
                // добавляем префиксы
                $model_name = 'Model_'.$controller_name;
                $controller_name = 'Controller_'.$controller_name;
                $action_name = 'action_'.$action_name;

                // подцепляем файл с классом модели (файла модели может и не быть)

                $model_file = strtolower($model_name).'.php';
                $model_path = "application/models/".$model_file;
                if(file_exists($model_path))
                {
                        include "application/models/".$model_file;
                }

                // подцепляем файл с классом контроллера
                $controller_file = strtolower($controller_name).'.php';
                $controller_path = "application/controllers/".$controller_file;
                if(file_exists($controller_path))
                {
                        include "application/controllers/".$controller_file;
                }
                else
                {
                        /*
                        правильно было бы кинуть здесь исключение,
                        но для упрощения сразу сделаем редирект на страницу 404
                        */
                        Route::ErrorPage404();
                }

                // создаем контроллер
                $controller = new $controller_name;
                $action = $action_name;

                if(method_exists($controller, $action))
                {
                        // вызываем действие контроллера
                        $controller->$action();
                }
                else
                {
                        // здесь также разумнее было бы кинуть исключение
                        Route::ErrorPage404();
                }

        }

        function ErrorPage404()
        {
        $host = 'http://'.$_SERVER['HTTP_HOST'].'/';
        header('HTTP/1.1 404 Not Found');
                header("Status: 404 Not Found");
                header('Location:'.$host.'404');
    }
}
```

Замечу, что в классе реализована очень упрощенная логика (несмотря на объемный код) и возможно даже имеет проблемы безопасности. Это было сделан намерено, т.к. написание полноценного класса маршрутизации заслуживает как минимум отдельной статьи. Рассмотрим основные моменты…

В элементе глобального массива $_SERVER['REQUEST_URI'] содержится полный адрес по которому обратился пользователь.
Например: example.ru/contacts/feedback

С помощью функции *explode* производится разделение адреса на составлющие. В результате мы получаем имя контроллера, для приведенного примера, эт контроллер *contacts* и имя действия, в нашем случае — *feedback*.

Далее подключается файл модели (модель может отсутствовать) и файл контроллера, если таковые имеются и наконец, создается экземпляр контроллера вызывается действие, опять же, если оно было описано в классе контроллера.

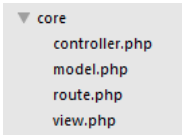
Таким образом, при переходе, к примеру, по адресу:
example.com/portfolio
или
example.com/portfolio/index
роутер выполнит следующие действия:

1. подключит файл model_portfolio.php из папки models, содержащий класс Model_Portfolio;
2. подключит файл controller_portfolio.php из папки controllers, содержащий класс Controller_Portfolio;
3. создаст экземпляр класса Controller_Portfolio и вызовет действие по умолчанию — action_index, описанное в нем.

Если пользователь попытается обратиться по адресу несуществующего контроллера, к примеру:
example.com/ufo
то его перебросит на страницу «404»:
example.com/404
То же самое произойдет если пользователь обратится к действию, которое не описано в контроллере.

2.2. Возвращаемся к реализации MVC

Перейдем в папку core и добавим к файлу route.php еще три файла: **model.php, view.php и controller.php**



Напомню, что они будут содержать базовые классы, к написанию которых мы сейчас и приступим.

Содержимое файла **model.php**

```
class Model
{
        public function get_data()
        {
        }
}
```

Класс модели содержит единственный пустой метод выборки данных, который будет перекрываться в классах потомках. Когда мы будем создавать классы потомки все станет понятней.

Содержимое файла **view.php**

```
class View
{
        //public $template_view; // здесь можно указать общий вид по умолчанию.

        function generate($content_view, $template_view, $data = null)
        {
                /*
                if(is_array($data)) {
                        // преобразуем элементы массива в переменные
                        extract($data);
                }
                */

                include 'application/views/'.$template_view;
        }
}
```

Не трудно догадаться, что метод *generate* предназначен для формирования вида. В него передаются следующие параметры:

1. $content_file — виды отображающие контент страниц;
2. $template_file — общий для всех страниц шаблон;
3. $data — массив, содержащий элементы контента страницы. Обычно заполняется в модели.

Функцией include динамически подключается общий шаблон (вид), внутри которого будет встраиваться вид
для отображения контента конкретной страницы.

В нашем случае общий шаблон будет содержать header, menu, sidebar и footer, а контент страниц будет содержаться в отдельном виде. Опять же это сделан упрощения.

Содержимое файла **controller.php**

```php
class Controller {

	public $model;
	public $view;

	function __construct()
	{
		$this->view = new View();
	}

	function action_index()
	{
	}
}
```

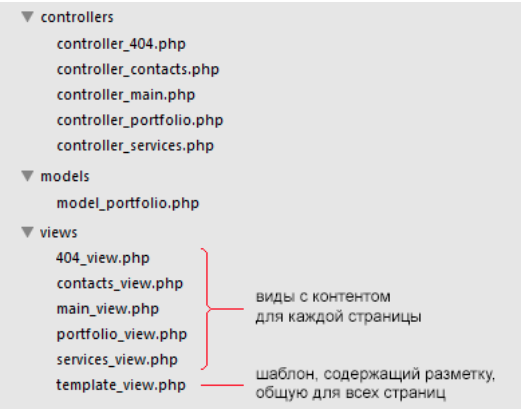
Метод *action_index* — это действие, вызываемое по умолчанию, его мы перекроем при реализации классов потомков.

2.3. Реализация классов потомков Model и Controller, создание View's

Теперь начинается самое интересное! Наш сайт-визитка будет состоять из следущих страниц:

1. Главная
2. Услуги
3. Портфолио
4. Контакты
5. А также — страница «404»

Для каждой из страниц имеется свой контроллер из папки controllers и вид из папки views. Некоторые страницы могут использовать модель или модели из п
models.



На предыдущем рисунке отдельно выделен файл **template_view.php** — это шаблон, содержащий общую для всех страниц разметку. В простейшем случае о
бы выглядеть так:

```php
<!DOCTYPE html>
<html lang="ru">
<head>
	<meta charset="utf-8">
	<title>Главная</title>
</head>
<body>
	<?php include 'application/views/'.$content_view; ?>
</body>
</html>
```

Для придания сайту презентабельного вида сверстаем CSS шаблон и интегрируем его в наш сайт путем изменения структуры HTML-разметки и подключ
CSS и JavaScript файлов:

```php
<link rel="stylesheet" type="text/css" href="/css/style.css" />
<script src="/js/jquery-1.6.2.js" type="text/javascript"></script>
```

В конце статьи, в разделе «Результат», приводится ссылка на GitHub-репозиторий с проектом, в котором проделаны действия по интеграции простенького
шаблона.

2.3.1. Создадаем главную страницу

Начнем с контроллера **controller_main.php**, вот его код:

```
class Controller_Main extends Controller
{
	function action_index()
	{
		$this->view->generate('main_view.php', 'template_view.php');
	}
}
```

В метод *generate* экземпляра класса View передаются имена файлов общего шаблона и вида с контентом страницы.
Помимо индексного действия в контроллере конечно же могут содержаться и другие действия.

Файл с общим видом мы рассмотрели ранее. Рассмотрим файл контента **main_view.php**:

```
<h1>Добро пожаловать!</h1>
<p>
<img src="/images/office-small.jpg" align="left" >
<a href="/">ОЛОЛОША TEAM</a> – команда первоклассных специалистов в области разработки веб-сайтов с многолетним опытом коллекционирования мекси
их масок, бронзовых и каменных статуй из Индии и Цейлона, барельефов и изваяний, созданных мастерами Экваториальной Африки пять-шесть веков наз
</p>
```

Здесь содержиться простая разметка без каких либо PHP-вызовов.
Для отображения главной странички можно воспользоваться одним из следующих адресов:

- example.com
- example.com/main
- example.com/main/index

Пример с использованием вида, отображающего данные полученные из модели мы рассмотрим далее.

2.3.2. Создадаем страницу «Портфолио»

В нашем случае, страница «Портфолио» — это единственная страница использующая модель.
Модель обычно включает методы выборки данных, например:

1. методы нативных библиотек pgsql или mysql;
2. методы библиотек, реализующих абстракицю данных. Например, методы библиотеки PEAR MDB2;
3. методы ORM;
4. методы для работы с NoSQL;
5. и др.

Для простоты, здесь мы не будем использовать SQL-запросы или ORM-операторы. Вместо этого мы сэмулируем реальные данные и сразу возвратим массив результатов.
Файл модели **model_portfolio.php** поместим в папку models. Вот его содержимое:

```
class Model_Portfolio extends Model
{
	public function get_data()
	{
		return array(

			array(
				'Year' => '2012',
				'Site' => 'http://DunkelBeer.ru',
				'Description' => 'Промо-сайт темного пива Dunkel от немецкого производителя Löwenbraü выпускаемого в России пив
нной компанией "САН ИнБев".'
			),
			array(
				'Year' => '2012',
				'Site' => 'http://ZopoMobile.ru',
				'Description' => 'Русскоязычный каталог китайских телефонов компании Zopo на базе Android OS и аксессуаров к ни
			),
			// todo
		);
	}
}
```

Класс контроллера модели содержится в файле **controller_portfolio.php**, вот его код:

```
class Controller_Portfolio extends Controller
{

	function __construct()
	{
		$this->model = new Model_Portfolio();
		$this->view = new View();
	}

	function action_index()
	{
		$data = $this->model->get_data();
		$this->view->generate('portfolio_view.php', 'template_view.php', $data);
	}
}
```

В переменную *data* записывается массив, возвращаемый методом *get_data*, который мы рассматривали ранее.
Далее эта переменная передается в качестве параметра метода *generate*, в который также передаются: имя файла с общим шаблон и имя файла, содержа
вид с контентом страницы.

Вид содержащий контент страницы находится в файле **portfolio_view.php**.

```
<h1>Портфолио</h1>
<p>
<table>
Все проекты в следующей таблице являются вымышленными, поэтому даже не пытайтесь перейти по приведенным ссылкам.
<tr><td>Год</td><td>Проект</td><td>Описание</td></tr>
<?php

	foreach($data as $row)
	{
		echo '<tr><td>'.$row['Year'].'</td><td>'.$row['Site'].'</td><td>'.$row['Description'].'</td></tr>';
	}

?>
</table>
</p>
```

Здесь все просто, вид отображает данные полученные из модели.

2.3.3. Создаем остальные страницы

Остальные страницы создаются аналогично. Их код досутпен в репозитории на GitHub, ссылка на который приводится в конце статьи, в разделе «Результа

3. Результат

А вот что получилось в итоге:

Скриншот получившегося сайта-визитки

Ссылка на GitHub: https://github.com/vitalyswipe/tinymvc/zipball/v0.1

А вот в этой версии я набросал следующие классы (и соответствующие им виды):

- Controller_Login в котором генерируется вид с формой для ввода логина и пароля, после заполнения которой производится процедура аутентификаци случае успеха пользователь перенаправляется в админку.
- Contorller_Admin с индексным действием, в котором проверяется был ли пользователь ранее авторизован на сайте как администратор (если был, то отображается вид админки) и действием logout для разлогинивания.

Аутентификация и авторизация — это другая тема, поэтому здесь она не рассматривается, а лишь приводится ссылка указанная выше, чтобы было от чего оттолкнуться.

4. Заключение

Шаблон MVC используется в качестве архитектурной основы во многих фреймворках и CMS, которые создавались для того, чтобы иметь возможность разрабатывать качественно более сложные решения за более короткий срок. Это стало возможным благодаря повышению уровня абстракции, поскольку е предел сложности конструкций, которыми может оперировать человеческий мозг.

Но, использование веб-фреймворков, типа Yii или Kohana, состоящих из нескольких сотен файлов, при разработке простых веб-приложений (например, сай визиткок) не всегда целесообразно. Теперь мы умеем создавать красивую MVC модель, чтобы не перемешивать Php, Html, CSS и JavaScript код в одном фа

Данная статья является скорее отправной точкой для изучения CMF, чем примером чего-то истинно правильного, что можно взять за основу своего веб-приложения. Возможно она даже вдохновила Вас и вы уже подумываете написать свой микрофреймворк или CMS, основанные на MVC. Но, прежде чем изобретать очередной велосипед с «блекджеком и шлюхами», еще раз подумайте, может ваши усилия разумнее направить на развитие и в помощь сообще уже существующего проекта?!

P.S.: Статья была переписана с учетом некоторых замечаний, оставленных в комментариях. Критика оказалась очень полезной. Судя по отклику: коммента обращениям в личку и количеству юзеров добавивших пост в избранное затея написать этот пост оказалось не такой уж плохой. К сожалению, не возможн учесть все пожелания и написать больше и подробнее по причине нехватки времени… но возможно это сделают те таинственные личности, кто минусовал первоначальный вариант. Удачи в проектах!

5. Подборка полезных ссылок по сабжу

В статье очень часто затрагивается тема веб-фреймворков — это очень обширная тема, потому что даже микрофреймворки состоят из многих компонентов

увязанных между собой и потребовалась бы не одна статья, чтобы рассказать об этих компонентах. Тем не менее, я решил привести здесь небольшую подб ссылок (по которым я ходил при написаниие этой статьи), которые так или иначе касаются темы фреймворков.

Ссылки

php, framework, cmf, mvc, site

+12 437k 1429

Vitaly Swipe @vitalyswipe карма 0,0 рейтинг 0,0

ПОХОЖИЕ ПУБЛИКАЦИИ

30 ноября 2012 в 14:24
Нативный MVC для Silex PHP Framework
+23 23,5k 112 15

4 июня 2012 в 23:04
Laravel — PHP Framework для ремесленников
+20 81,2k 335 67

2 сентября 2008 в 15:33
Почему нужно использовать php-framework'и, на примере codeigniter
+11 18,5k 21 31

САМОЕ ЧИТАЕМОЕ Разраб

Сутки Неделя Месяц

А был ли взлом «Госуслуг»? Гипотеза Яндекса
+48 22,2k 28 55

Реверс-инжиниринг одной строчки JavaScript
+116 21,3k 133 17

Security Week 28: a Petya сложно открывался, в Android закрыли баг чипсета Broadcomm, Copycat заразил 14 млн девайсов
+14 11,3k 19 6

Дорога к C++20
+27 6,3k 22 13

IBM и ВВС США разрабатывают нейроморфный суперкомпьютер нового поколения
+14 8,3k 24 16

Комментарии (175)

masterrr 27 августа 2012 в 07:18 #

Вместо require может лучшим будет require_once использовать?

darkneo 27 августа 2012 в 07:52 #

Лучше использовать автоматическую загрузку классов, иначе все require будет отслеживать уже нереально.

g00d 28 августа 2012 в 12:59

А еще лучше начать использовать namespace'ы с автомотической загрузкой классов и тогда в этом плане наступит полная эйфория :). Хотя я сам и пользуюсь этим способом, код от этого лучше выглядеть не начинает :(вместо конструкции require*/include* появляется обратный слеш :(

VolCh 28 августа 2012 в 13:23

Ну, код

```
use Vendor\App\Models;

$blog = new Blog();
$blog->posts[] = new Post();
```

выглядит лучше чем

```
require_once 'lib/vendor/app/models/blog.php';
require_once 'lib/vendor/app/models/post.php';

$blog = new Vendor_App_Models_Blog();
$blog->posts[] = new Vendor_App_Models_Post();
```

Или нет? :)

g00d 28 августа 2012 в 13:27

100% у меня он точно такой же

только вот эту мелочь стоит гда нить ранее объявить

spl_autoload_extensions(".php"); // comma-separated list
spl_autoload_register();

но вот посмотришь на код глубже и порой от такого количества обратных слешей иногда берет дрожжжж :)
но это больше проблема namespace'ов нежли способа загрузки файлов :)

g00d 28 августа 2012 в 13:42

еще кое что забыл :)

define(«APP_PATH», __DIR__. DS. '..'. DS. 'app'. DS);
set_include_path(APP_PATH); // adding new path to include_path
spl_autoload_extensions(".php"); // comma-separated list
spl_autoload_register();

вот теперь все выглядит как надо.

Ogra 28 августа 2012 в 13:33

От require можно избавиться, и останется только new Vendor_App_Models_Blog().
Это лучше, чем use Vendor\App\Models; new Blog(); по двум причинам:
1. В любом месте листинга видно, что за класс создается
2. Создание объекта класса выглядит всегда одинаково, независимо от use в начале файла.

g00d 28 августа 2012 в 14:49

думаю это на 100% зависит от выбранного вами подхода при «строительстве» вашего проекта.

т.е. все полностью упирается в вашу архитектуру, в некоторых уже готовых приложениях лучше убиться чем начинать использовать namespa

Если же вы делаете все с нуля то нет причин не использовать разумный подход, и лучше планировать свою архитектуру.

VolCh 28 августа 2012 в 15:06

Даже в таких случаях я предпочту `new \Vendor\App\Models\Blog();`, а написав `new \Vendor\App\Models\Post();`, сделаю небольшой рефакторинг и введу use ради DRY.

Ogra 28 августа 2012 в 15:12

Вводя этот use, вы теряете в понимании кода. Я не могу сказать, что хуже — нагромождение из \ и нэймспейсов, не понятно, к какому нэймспэйсу принадлежащие классы или же PSR-именованные классы. Мне все кажется плохим ((

ЗЫ: В Google C++ Style Guide конструкция use запрещена.

VolCh 28 августа 2012 в 15:23

К нэймспэйсам так или иначе приходят в сложных приложениях, вопрос в том использовать их нативные или эмуляцию через _. Что «в плохо» почти согласен, но вариант с use мне кажется оптимальным из трёх. Кроме всего прочего он явно показывает (если в в соглаше принять что он обязателен даже при единичном использование, а не только ради DRY) зависимости текущего файла.

Zelgadis 27 августа 2012 в 07:39 #

Неожидал увидеть Ололошу на хабре.

Elkan 27 августа 2012 в 07:46 #

>> Подход с множеством точек взаимодействия
это называется точка входа в приложение.
>>В общем, сегодня речь пойдет о самом популярном (разве что после Singleton) шаблоне проектирования MVC
думаю такое сравнение некорректно, т.к. MVC это шаблон другой, и он не входит в список тех 27 из книги GoF.
Картинка порадовала, правда на ней я бы руль обозвал контроллером) Кстати еще, поспешите добавить в статью дисклеймер про то что эта статья для сов новичков, а то можете рассчитывать что вас раскритикуют) я тоже сначала хотел, но потом подумал, что это будет выглядеть как-то глупо) кому надо из нс тот посмотрит, а те кто используют fw конечно так пролестают просто на предмет грубых или глупых каких-то моментов. Я смотрю вы тут уже 3 части напис оперативно однако.

vitalyswipe 27 августа 2012 в 07:48 # ↑

да вроде из контекста очевидно, что для новичков

BlessMaster 27 августа 2012 в 21:04 # ↑

Библиотека профессионала: «новичкам — от новичков»

barker 27 августа 2012 в 11:43 # ↑

> думаю такое сравнение некорректно, т.к. MVC это шаблон другой, и он не входит в список тех 27 из книги GoF.

Потому что в статье написано неправильно, MVC — это не шаблон, а парадигма. И построена может быть на разных шаблонах или без них вообще. А шаб может назвать тот, кто не очень понимает что такое шаблон. Просто помешались уже на паттернах кругом все.

Elkan 27 августа 2012 в 12:11 # ↑

Да вы абсолютно правы :)

VolCh 28 августа 2012 в 01:11 # ↑

Формально говоря, MVC является архитектурным шаблоном приложения, а тот же Singleton шаблон более низкоуровневый. Или Фаулер, по-вашему, н понимает что такое шаблоны?

barker 28 августа 2012 в 10:02 # ↑

Полагаю, что понимает. Но это уже просто вопрос терминов. Суть MVC в идее «разделение V и C (и M)», и на самом деле это всё. Сложно это «шаб назвать, имхо. То, что MVC — это может быть несколько каких-либо «классических» шаблонов Вы согласны?

Ogra 28 августа 2012 в 10:09 # ↑

Отнюдь, это именно шаблон. Он, кстати, описан в книге «Банды четырех», только не в каталоге, а в первой части, где идет рассказ о Smalltalk.

darkneo 28 августа 2012 в 10:19 # ↑

В новой редакции он описан как составной шаблон.

Ogra 28 августа 2012 в 10:21 # ↑

Не читал еще новую ;)

ivvi 13 июня 2013 в 10:12 # ↑

www.design-pattern.ru/patterns/mvc.html

barker 13 июня 2013 в 17:07 # ↑

Ну, я рад за автора сайта «Справочник паттерны проектирования», но к чему это? И MVC там стоит в одном разделе с «шаблонизатором» и «контроллером страницы», обозванным «паттерны веб-представления», что, если даже посчитать это более-менее корректным, подтверждает как мои слова выше.

Tenphi 27 августа 2012 в 09:09 #

Эх, такие бы статьи да во времена, когда я только начинал изучать PHP.
Новичкам, думаю, очень поможет.

Для меня, к счастью это уже не актуально. Сменил профу на JS.

SpiRi7 27 августа 2012 в 09:22 #

Для новичков расжовано хорошо, пару замечений:
— Почему без namespace?
— Зачем закрывающий тег ?>, потенциальны проблемы с сессиями

boramod 27 августа 2012 в 14:53

Я не знаю php. Подскажите по вашему 2-му замечанию:
— почему не должен стоять закрывающий тег ?>
— почему потенциальные проблемы с сессиями в приведеном примере.

Спасибо!

kovalevsky 27 августа 2012 в 15:03

Закрывающий тег в PHP вообще можно не ставить, если в файле только PHP код.

NoN 27 августа 2012 в 15:03

1. Всё что будет после ?> окажется контентом.
2. Некоторые редакторы могут добавлять последнюю пустую строку. Ну или случайно > может оказаться не последним байтом в файле.
3. Когда этот файл будет за'require'ен, этот хвост будет отправлен на клиент.
4. После начала отправки контента нельзя установить заголовки, в т.ч. куки, в т.ч. сессионные куки.

Вывод — особенно в подключаемых файлах, а лучше вообще всегда, нужно убирать ?>, если после него нет осмысленного контента.

boramod 27 августа 2012 в 15:08

Спасибо большое за развернутый ответ! Почерпнул для себя полезную информацию.

fritz321 27 августа 2012 в 09:40

И почему в сотне таких одинаковых статей про php и mvc — нет ни слова как убрать код php из вида и прикрутить какой либо шаблонизатор. Помоему если сделаете — Ваша статья будет качественно отличаться от других.

vitalyswipe 27 августа 2012 в 11:56

Я планировал написать про шаблонизацию, но понял, что текст статьи в таком случае будет сильно раздут и к тому же может породить очередной спор и а стоит ли использовать сторонний шаблонизатор, ведь PHP сам по себе является шаблонизатором. О реализации простого шаблонизатора вы можете почитать в следующих статьях: "Мой родной PHP шаблонизатор " и "Делим код пополам или представление по шаблону в PHP ". Кстати, эти ссылки прив в конце статьи!

Ogra 27 августа 2012 в 12:00

Может быть потому, что шаблонизаторы к ПХП прикручивать не надо?
Или потому, что в одной статье лучше разбирать одну тему, а не две?

BlessMaster 28 августа 2012 в 01:38

В данном случае у автора получилась уже не одна статья, так что это не ограничение :-)

А шаблоны к PHP прикручивать надо — как раз по причине того, что из хорошего шаблонизатора сделали посредственный язык программирования. З двумя зайцами погонишься…

VolCh 28 августа 2012 в 01:18

Достаточно отделить вид от всего остального, инкапсулировать в него шаблонизатор (или использовать шаблонизатор в качестве вида) и обеспечить бо менее слабую связанность, чтобы заменить один шаблонизатор на другой.

Stac 29 августа 2012 в 21:12

PHP сам по себе чем плох в роли шаблонизатора?

Для каких задач нужно что-то еще?

VolCh 31 августа 2012 в 16:50

Нет автоэкранирования вывода для HTML, приходит писать минимум <?= e($var) ?> вместо {{var}}.

Нетривиальна реализация наследования шаблонов.

Не предоставляет изоляции шаблона от остальной программы и глобального нэймспэйса. То есть в шаблоне вы можете сделать ровно то же самое в самой программе, например выполнить eval($_GET['expr']);

Stac 31 августа 2012 в 18:08

Не очень понял.

1. Что за автоэкранирование и зачем? Если данные не готовы для вывода, разве «модель» не должна их подготовить? Или «модель представле

2. Видимо я не сталкивался еще с такой проблемой… С другой стороны реализация разве не зависит от вас?

3. А зачем специально изолировать шаблон? Это же ваша программа, просто не используйте там eval($_GET['expr']);.

Правильно ли я понял, что надо сначала изобрести себе кучу проблем, и тогда сторонний шаблонизатор обязательно поможет их решить?

 **VolCh** 31 августа 2012 в 19:07

1. Чтобы не приходилось практически в каждом месте html-шаблона писать писать `htmlspecialchars($var, ENT_QUOTES, 'UTF-8');`. Н модель, модель не знает будут данные в html выданы или, например в JSON. И даже не модель представления, имхо, хотя тут могут быть вар

2. Зависит-то то она зависит, но решать её средствами PHP «в лоб» нетривиально, а код шаблона не очень читабельный, например

```
<?php $view->extend('::base.html.php') ?>

<?php $view['slots']->set('title', 'My cool blog posts') ?>

<?php $view['slots']->start('body') ?>
    <?php foreach ($blog_entries as $entry): ?>
        <h2><?php echo $entry->getTitle() ?></h2>
        <p><?php echo $entry->getBody() ?></p>
    <?php endforeach; ?>
<?php $view['slots']->stop() ?>
```

по сравнению с

```
{% extends '::base.html.twig' %}

{% block title %}My cool blog posts{% endblock %}

{% block body %}
    {% for entry in blog_entries %}
        <h2>{{ entry.title }}</h2>
        <p>{{ entry.body }}</p>
    {% endfor %}
{% endblock %}
```

3. Не всегда шаблон пишет разработчик движка, это может быть верстальщик, веб-мастер, а то и конечный пользователь (индивидуальное оформление своей странички, например). В общем лучше иметь возможность включить режим «песочницы», чем не иметь.

 **Stac** 31 августа 2012 в 19:57

1. Спорно. Во-первых `htmlspecialchars($var, ENT_QUOTES, 'UTF-8');` можно сделать и так:

```
foreach ($data as $k=>$v)  $data[$k] = htmlspecialchars($v, ENT_QUOTES, 'UTF-8');
```

Одна строчка. И даже если она будет не в модели (хотя @Ogra, похоже, с вами не согласен по этому поводу) по она может быть в методе, который рендерит шаблон.

Да и в модели она может быть. Модель не знает, как будут использованы данные. Но вы-то знаете :) Вызывайте метод типа getDataForHT или getDataForJSON().

2. Для меня удивительно нечитабельны оба ваших примера :).

У меня например, в шаблоне страницы может быть конструкция `<?=show("some_block_template", $page);?>`, которая значит, что на этом месте надо вставить шаблон некоего блока страницы. У каждого раздела сайта этот блок («заголовок, меню, строка поиска, whateve может иметь разный шаблон. Если у данного раздела такой шаблон не определен используется базовый, если базового нет, то используе вшитый шаблон по-умолчанию (где просто написано, что шаблон такого-то блока для страницы не определен).

Я не уверен, что это то самое наследование шаблонов про которое вы говорите. Если что вы меня поправите.

3. Тут согласен. В описанной вами ситуации разумнее использовать сторонний шаблонизатор, а не php.

 **VolCh** 31 августа 2012 в 21:36

1. Это сработает в случае если все данные нужно экранировать, а если часть надо, а часть не надо? Велик риск пропустить что-то в бе или чёрном списке.

2. Интеллектуальный include, который в зависимости от контекста возвращает разные результаты? То есть у вас каждая страница сос утрируя, из <html><?=show(«head_template», $page);?><?=show(«body_template», $page);?></html>? Я правильно понял? Если так, то вы наследуете блоки, но не наследуете основной layout.

 **Stac** 1 сентября 2012 в 10:48

1. Ваш изначальный посыл был в том, что все надо экранировать и запись `{{var}}` лучше, чем php-фаналог. Разве нет?
2. Я не делю основной layout и блоки, layout это просто блок «page», который включает в себя другие блоки (или не включает, если надо).

VolCh 1 сентября 2012 в 10:59

1. Не всё, но по умолчанию всё экранировано, если нужны raw данные, так и пишем {{varIraw}}

2. А как вы из шаблона ссылаетесь на layout?

Stac 1 сентября 2012 в 18:17

1. ок. Поэтому вопросу делаю для себя вывод: без шаблонизатора не обойтись, когда надо предоставить возможность создани изменения шаблонов человеку со стороны (не разработчику) и особенно пользователю. В других случаях выбор использовать использовать шаблонизатор (а не сам PHP) это вопрос технических требований и личного вкуса разработчика.

2. Если я правильно понял, что такое layout (`<html><head></head><body><header><div class="content"></div><foot </footer></body></html>`), то я никак не ссылаюсь на него из шаблонов других блоков. Я придерживаюсь концепции незав блоков.

VolCh 1 сентября 2012 в 19:12

1. Я бы еще добавил, если велика вероятность, что код придётся поддерживать через некоторое время, когда детали забу Или на все проекты должны быть единые соглашения.

2. Вот есть, допустим, блок user, есть post, есть admin, есть layout базовый (для user и post), куда блоки должны вставиться .content и есть расширенный для админки. Как вы задаёте, что для uri /user/ нужно использовать базовый layout, для /post/ic а для /admin — расширенный?

Stac 1 сентября 2012 в 20:44

1. Не могу согласиться. Сам долго работал на поддержке чужих проектов (к счастью простых). И тут чем меньше технол используется, тем проще поддерживать.

2. Какой layout использовать я определяю по uri, соответствующие файлы будут лежать, например, в `templates/user/page.block.user.htm, templates/post/page.block.post.htm, templates/admin/page.block.admin.htm`. Свой файл для каждого раздела/объекта. если файла по такому пути нет используется глобальный.

Т.к. layout'ы страниц весьма похожи, то общие фрагменты (header, footer) хранятся в отдельных файлах и «инклюдятся» `page.block.htm`.

Stac 1 сентября 2012 в 20:45

1. ну как долго… всего пару лет. Так что мой опыт ничтожно мал. Я могу быть не прав.

Ogra 31 августа 2012 в 20:00

{% extends '::base.html.twig' %}

Вот за это я и не люблю наследование. Не дело знать шаблону контента, где этот контент будут использовать. Один и тот же контент мо быть как показан на странице, так и отправлен клиенту по почте. Как это решать при наследовании?

VolCh 31 августа 2012 в 21:46

Ну, если допустить, что у нас есть один layout для html представления контента на странице, а другой для почты, то можно поступить т

```
{# /content_http.twig.html #}

{% extends '::base.html.twig' %}

{% block body %}
  {% include '::content.html.twig' %}
{% endblock %}

{# http specific #}



{# /content_mail.twig.html #}

{% extends '::mail.html.twig' %}

{% block body %}
  {% include '::content.html.twig' %}
{% endblock %}

{# mail specific #}
```

То есть content.html.twig отрисовывает себя одинаково для http и почты, не зная в каком конткесте он вызывается, а контроллер вызыв разные шаблоны для разных контекстов, в которых установлена связь между шаблоном контента и соотвествующего контенту лэйаут

Ogra 31 августа 2012 в 21:48

Т.е. те же самые инклуды контента, что и в случае без наследования, только более хитрые, да еще и с дублированием кода. Мне это не нравится.

VolCh 31 августа 2012 в 22:15

Может я слишком широко понял задачу и в конкретном случае можно выкрутиться покрасивее.

Ogra 31 августа 2012 в 18:30

1. Не думаю, что задача экранирования должна лежать на шаблоне. В конце-концов, это защита сайта от XSS, её нужно решать выше.
2. Наследование шаблонов — далеко не всегда нужно, но ладно, запишем в недостатки
3. Как обучение MVC и его принципам — изоляция полезна. Но в повседневной разработке — нет. Как только вы поняли, что такое MVC, и поче стоит обращаться из шаблона куда попало — это для вас не проблема. А если у вас в команде студенты, то доступ к репозиторию давать тольк Code Review.

VolCh 31 августа 2012 в 19:14

1. Только шаблон в идеале знает, в каком виде будут представлены данные, скажем правила экранирования немного да отличаются для htm html5, не говоря об xml.

2. Практически всегда нужно, если следовать принципу DRY. Многие сайты/приложения состоят из многих страниц с однотипной вёрсткой, различающейся только блоком контента. Есть вариант обойтись без дублирования с помощью конструкций типа `include "header.phtml"/include "footer.phtml"`, но он не очень удобен для разработки и, особенно, поддержки.

3. Не всегда разработчик контролирует код приложения.

Ogra 31 августа 2012 в 19:23

1. Шаблон не должен знать, каким данным можно доверять, а каким нет! Это не его прерогатива.
2. Вариант с layout отдельно, content отдельно, на мой взгляд лучше для таких сайтов. Наследование — для более сложных ситуаций.
3. И что? Защита от дурака, это, конечно, хорошо, но нельзя же везде поставить такие препоны — быдлокодеры всегда найдут где в код нагадить ;)

VolCh 31 августа 2012 в 19:38

1. Вопрос экранирования данных это не вопрос защиты от XSS или ещё чего, это лишь вопрос корректного вывода данных в конкретно представлении. И это область ответственности шаблона (вернее вида). Он должен знать какие данные ему выводить как есть, а какие экранировать согласно своему типу вывода, чтобы получить корректный вывод. Защита тут лишь побочный эффект. Как у экранирова помощью mysql_real_escape_string в параметрах или использование подготовленных варажений побочный эффект защита от некоторы SQL-инъекций, но защитой они не являются.

2. Это частный случай наследования, одноуровневое. Не всегда его хватает. Да и всё равно обычно не так читаемо получается.

3. Если это нам ничего не стоит, максимум строчка в конфиге, почему бы не ставить? :)

Ogra 31 августа 2012 в 19:42

1. Правильно, это область ответственности **вида**. Шаблону об этом знать не нужно.
2. По-моему это читаемее, и, что важно — более гибко. Наследование — слишком сильно связывает шаблоны друг с другом.
3. Это не строчка в конфиге, это возросшая сложность приложения. KISS не соблюдается.

VolCh 31 августа 2012 в 22:09

1. Сложно разделить ответственность между видом и шаблоном, шаблон — составная часть вида и вполне, по-моему, может знат данных столько же сколько вид. В конце-концов только разработчик шаблона знает в каком месте данные нужно экранировать, а каком нет, а вид, не говоря о контроллере и, тем более, модели должен быть способен работать с любым шаблоном. А в модели д каждого поля делать не только нативный геттер, но геттеры для каждого формата по-моему очень неразумно. Выбор, имхо, стои только в том экранировать по умолчанию или нет, основной набор данных передаваемых в шаблон. Исходя из того, что проще за заэкранировать, чем разэкранировать, я выбираю экранирование по дефолту, которое PHP без костылей реализовать не может.

2. О вкусах не спорят :)

3. Сложность инкапсулирована в лучших традициях ООП :)

Stac 1 сентября 2012 в 10:52

1. Как это шаблон и вид могут знать о данных одинаково?
«Вид» это код, который пишите вы, шаблон — разметка, которую пишет верстальщик или пользователь. Вы сами говорили об выше. Я думаю, в таком случае, разработчик знает о данных не очень много.

VolCh 1 сентября 2012 в 15:42

Вид предназначен для представления данных, шаблонизация лишь один из способов его реализации. Вид, вернее его разработчики, должны знать какие данные ему идут на вход и как их корректно вывести. В одних реализациях разработчи достаточно знать какие данные, в других — только верстальщику, в третьих — обоим необходимо знать и синхронизироват знания.

AlexChernyy 27 августа 2012 в 09:47

Я не знаю PHP. Вопрос. Из модели вернули массив. В представлении использовали обращение не по ключам массива, а «напрямую». Как это возможно?

kovalevsky 27 августа 2012 в 09:48

С массивом из модели должен работать контроллер, а не представление

AlexChernyy 27 августа 2012 в 09:53

Пусть бы и так. Только это ничего не изменило. Контроллер получил данные из модели в виде массива. Котроллер передал данные в представление массива. В представлении используется не массив.

kovalevsky 27 августа 2012 в 09:57

Правильно, потому что в классе load.php, в методе view есть строчка extract($data)
Каждый ключ массива превращается в переменную.
php.net/manual/ru/function.extract.php

AlexChernyy 27 августа 2012 в 10:25

Я был невнимателен.

Stac 29 августа 2012 в 21:16

Наоборот. Вы внимательный :)

Ведь где-то могут существовать переменные с такими же именами, как и ключи массива.
Пусть в примере их нет. Но в жизни бывает всякое. :)

zerkms 28 августа 2012 в 02:29

С чего вы так решили?

AlexChernyy 27 августа 2012 в 09:54

Я нашел ответ на вопрос. Функция extract.

Захар4еНко 27 августа 2012 в 10:20

extract не безопасная функция, попробуйте почитать про ActiveRecord или DataMapper

kovalevsky 27 августа 2012 в 10:26

Простите, причём здесь ActiveRecord или DataMapper?
Речь идёт о массиве в представлении, не о базах данных.

darkneo 27 августа 2012 в 10:33

При том, что можно отдать представлению модель и обращаться к ее свойствам из представления.

kovalevsky 27 августа 2012 в 10:38

Всё равно считаю, что в представлении должно быть представление, а работать с моделью должен контроллер.

darkneo 27 августа 2012 в 10:48

Спорить не буду, лишь отмечу, что концепции MVC это не противоречит. Можно в контроллере вбить передать значения свойств в представление, можно получать эти значения сразу в представлении.

kovalevsky 27 августа 2012 в 10:50

Просто в зависимости для кого это делать.
Если для себя, то никаких проблем, а если для кого-то, то в один прекрасный день кто-то, скорее всего, это сломает, пытаясь заменить в дизайне.

darkneo 27 августа 2012 в 10:56

Даже для приведенного примера. В чем разница между

```
<?php foreach ($someArr as $item) : ?> ... здесь какой-то вывод ... <?php endforeach; ?>
```

и

```
<?php foreach ($model->getSomeData() as $item) : ?> ... здесь какой-то вывод ... <?php endforeach; ?>
```

Представление также инкапсулировано от модели геттерами как если бы через эти же геттеры нужно было получить данные в контроллере

kovalevsky 27 августа 2012 в 10:59

Если так, то никаких проблем.
Я Вас, скорее всего, не правильно понял. Простите.

darkneo 27 августа 2012 в 11:03

Причем использование геттеров, на мой взгляд, более оправдано, так как позволяет гораздо быстрее делать (кодить представления). Написал геттер, который через LazyLoad получает нужные объекты и сразу можешь к нему обратиться в представлении без необходимости вручную передавать контроллером — модель подгружает себе данные по мере обращения

BlessMaster 27 августа 2012 в 22:58

На каком этапе реализуется ACL?

darkneo 28 августа 2012 в 06:59

ACL — задача контроллера.

iDark 27 августа 2012 в 11:01

Использую шаблоны и все выглядит примерно {data}, мне так кажется удобнее.

darkneo 27 августа 2012 в 11:04

Само собой, тот же Smary вполне положительно относится к тому, что я отдаю ему объект и в представлении обращаюсь к его свойствам.

VolCh 28 августа 2012 в 01:24

В том, что можно сделать `foreach ($model->setSomeData())`

3axap4eHko 27 августа 2012 в 10:38

т.е. в представление нельзя передавать обьекты?[сарказм] Я же не говорил использовать эти паттерны, а взять идею. К примеру передав в мет объекта setFromArray() или populate() этот массив, понятно что произойдет, А представьте мы получаем массив в котором есть индекс _SESSIO extract делаем, круто правда?

iDark 27 августа 2012 в 10:46

Не знаю как остальные, а у меня представление только выводит данные, всей их обработкой занимается контроллер.

darkneo 27 августа 2012 в 10:51

Контроллер не должен заниматься из обработкой, это задача либо модели, либо сервисного слоя. Контроллер передает вызовы от одного другому, максимум — содержит простую логику, максимум, собрать все необходимые объекты чтобы отдать их модели.

Толстый контроллер не айс.

iDark 27 августа 2012 в 10:58

Непонятно я выразился, я имею ввиду, что он получает данные от модели\пользователя, в первом случае смотрит на данные и вывод и например в случае с входом пользователя в систему, только саму форму, а если от модели пришло указание, что данные неверные\сод недопустимые символы, выводит ошибку. В этом плане.

Elkan 27 августа 2012 в 10:57

ну вообще-то на такие моменты используют префиксы, хватит уже попроть чушь. если посмотреть как делаю во фреймворках, то тма как ра экстрактинг и делают с префиксом если надо. а вы тут чушь и демагогию развели.

3axap4eHko 27 августа 2012 в 10:41

вот уже и минус успели нажать, вместо того, что бы немного подумать, что я написал

kovalevsky 27 августа 2012 в 10:45

Я Вас не минусовал, Вы правы, спорить не буду.
Но, как по мне, проще не пихать в extract то, что нельзя, чем делать хардкор или работать с моделью через вид. Или же просто передавать м вид.
Опыта у меня явно меньше Вашего, так что я просто прислушаюсь. :)

3axap4eHko 27 августа 2012 в 11:08

вот примерный класс view который избавит вас от extract: simpleView
а использовать просто:
в скрипте

```
$anyArrayWithData['title']='Mega Title';
$view->assign($anyArrayWithData);
$view->render('my/mega/template.phtml');
```

в представлении:

```
<h1><?php echo $this->title ?></h1>
```

вот и все

StyleT 27 августа 2012 в 10:29

Когда работал с Opencart очень понравилась её архитектура, считаю одной из самых простых и гибких реализаций MVC.

jMas 27 августа 2012 в 11:17

OpenCart не смотрел, но считаю архитектуру Yii одной из самых приятных.

iDark 27 августа 2012 в 10:45

Сюда бы еще автозагрузчик классов, сами имена классов согласно паттерну MVC и шаблонизатор, было бы самое то.

samally 27 августа 2012 в 11:16

И давно MVC как-то определяет имена классов?))
А то я и не в курсе :)

jrip 27 августа 2012 в 11:29

шаблонизатор тут как бы есть, его роль играет сам php.

vitalyswipe 27 августа 2012 в 12:02

habrahabr.ru/post/150267/#comment_5089363

shvedovka 27 августа 2012 в 16:13

Коллеги, раз уж собрались специалисты по архитектуре PHP приложений, у меня вопрос.

Как отказаться от своего фремворка мигрирующего из приложения в приложение, если он выполняет все, что от него требуется, но имеет архитектурные проблемы, которые решаются костылями, которые в сою очередь мигрируют в новые проекты?

Поскольку задачи из проекта в проект остаются похожими, желание сделать быстро — преодолевает желание переделать все с нуля.

Ogra 27 августа 2012 в 16:55

Для начала стоит написать задачи, которые выполняет ваш фреймворк. Глядишь, посоветуют подходящую для вас замену.
Если не поможет — выложить на Github, рассказать про проблемы (и необходимые костыли), задачи, планы и т.д. Либо опять же — посоветуют замену, л
могут решить проблемы с костылями.

shvedovka 27 августа 2012 в 17:07

Задачи нехитрые, тот же Yii их выполняет прекрасно, не считая некоторых специфичных вещей. В других проектах которые поддерживаю есть и Zend
Kohana.

Проблема не в том, что я не знаю на что перейти, а в том, что это поддерживать пока проще, чем писать с нуля. Я видимо некорректно объяснил. На б
этого «фреймворка», уже написано довольно большое приложение, которое собственно и тянется видоизменяясь.

Ogra 27 августа 2012 в 17:10

Я сталкивался с похожей ситуацией, и перешел на Zend. Время, потраченное, на переход, вполне себе окупилось.

someoneisusingmyusualnick 27 августа 2012 в 17:12

Собрать волю в кулак, обуздать лень и перейти на Yii/Symfony/Zend/etc, что ваша религия примет больше.
Старые проекты поддерживать на старом, а новые писать уже на любом общепринятом fw.
Мне кажется, что если вы отчетливо понимаете, что в вашем фреймоврке есть архитектурные проблемы, то не зачем делать на нем новые проекты, неза
эти проблемы плодить.
А касаемо скорости разработки — если возьмете Yii/Code Igniter/Kohana — то это фреймворки с простым уровнем входа, в них легко разобраться и начат
писать. Вы не много потеряете времени — попробуйте, вам понравится)

kryoz 27 августа 2012 в 23:23

Вот если честно как не пытался себя заставить перейти на какой-либо популярный фреймворк — не смог. Не вижу смысла. Стрелять из пушки по вор
Применяю свой микрокаркасс для MVC.

Прошедший год занимался разработкой сайтов на Joomla, там свой фреймворк, конечно изучить его был смысл, но точно не Yii или Zend.
А сейчас работаю над проектом, который представляет собой вавилонскую башню из говнокода, накопленного наверное лет за 5, не меньше. И в ком
тому же нет единого мнения о целесообразности ООП, не то, что бы даже про фреймворк.

Так что не пойму я этой повальной моды на фреймворки.

VolCh 28 августа 2012 в 01:35

Схожая ситуация, но постепенно перевожу код на ООП и MVC с использованием тестов, ORM, шаблонизаторов, библиотек имея в виду переход полноценный фреймворк.

kryoz 28 августа 2012 в 10:01

Я тоже считаю это направление развития правильным.
Но не всегда это оказывается возможным или приемлимым с точки зрения временных затрат.

Stac 29 августа 2012 в 21:22

Проблема еще в том, что мода на фреймворки это прежде всего мода.
Которая проходит, а проекты остаются.

Stac 29 августа 2012 в 21:20

Другими словами, создаете ад? :)

BlessMaster 28 августа 2012 в 01:42

Фреймворки — как стандарт, экономят время сотрудников в крупных проектах на изучение фреймворка.

VolCh 28 августа 2012 в 01:52

В случае PHP это работает не очень :(В вакансиях чаще видишь «опыт работы с MVC-фреймворками: ..., ..., ..., и т. п.», чем «опыт работы с ...». Работодатели как бы не надеются, что достаточно соискателей знают конкретный фреймворк, их устраивает, чтобы хоть какой-то знали, ведь переход с одного на другой займёт, в общем случае, куда меньше времени чем принятие концепций фреймворков, MVC и т. п. с нуля.

darkneo 28 августа 2012 в 07:03

Чтобы разрабатывать с использованием фреймворка нужно гораздо больше знаний и опыта, чем написание простых функций, раскиданных сотне файлов. Все просто — чтобы использовать фреймворк нужно хотя бы иметь представление о принципах, которые лежат в его основе, требование опыта работы с фреймворком вполне обосновано.

VolCh 28 августа 2012 в 10:43

Я к тому что единого стандарта де-факто, как Django в Python, RoR в Ruby или ASP .NET MVC в C#, нет (я в курсе что есть и другие фрейм в них, но эти явно доминируют). И работодатели хотят чтоб хоть с каким-нибудь был опыт, чтобы сократить время на изучение, но на то, ч вообще не понадобится и новому сотруднику нужно будет изучать только проект, но не сам фреймворк, особо не рассчитывают. RoR-разработчик вакансию можно увидеть, а Yii- или ZF-разработчик как-то не встречались.

Насчёт количества знаний необходимых для разработки вопрос спорный. Фреймворки многие детали скрывают.

Stac 29 августа 2012 в 21:24

Все верно. Это нужно для того, чтобы разрабатывать с использованием фреймворка.

Но не обязательно для того, чтобы решать прикладные задачи.

kryoz 28 августа 2012 в 10:24

У меня сложилось впечатление, что это как стандарт только читая Хабр.

В вакансиях сейчас часто конечно встречается пожелание знать популярный фреймворк, но также полно еще предложений без этого требовани Я в этом убедился сам, недавно выбирая работу, ходя по собеседованиям. А мой знакомый, который даже ООП толком не понимает, но имеет с 5, устроился на ведущего программиста.

darkneo 27 августа 2012 в 19:07

Если уж есть какой-то готовый проект, который до тебя писала орда говнокодеров, то обладая какими-то навыками построения приложений есть смысл сбоку прикрутить свой фреймворк, аккуратно его подцепить, чтобы и старое работало и новое писать было удобнее. В случае, когда есть уже существую система к ней гораздо легче прицепить свой фреймворк, чем прицепить Yii/Cohana/CI/etc. Причина очень проста — каждый разработчик знает свой фрей гораздо лучше, чем любой из готовых, знает, как с минимальными изменениями адаптировать его под нужны уже имеющегося приложения.

У меня у самого была такая же ситуация. Есть проект, который просто написан стихийно на коленке, достаточно неплохо работает, но к нему нужно напи много сложных дополнений. Комментарий в коде практически никаких, зато есть собственный фреймворк, который можно урезать и как основу новых м использовать. Вроде бы и старый код продолжает работать и рефакторить его не надо, с другой стороны и зависимость нового кода от старого минимал

А переход на распространенный фреймворк обоснован, как минимум, тем, что он распространенный. Узких мест в нем найдено гораздо больше, багов и вариантов использования протестировано также гораздо больше. Да и база готовых модулей сильно сокращает время разработки.

VolCh 28 августа 2012 в 01:31

Я выбрал способ вялотекущего рефакторинга с использованием Symfony Components, Doctrine 2 и Twig, имея в виду полный переход на Symfony 2, Doctrir Twig. Вялотекущего потому что код не мой и для написания тестов фактически его реверс-инженерю, толком даже не понимая где баг, а где фича.

darkneo 28 августа 2012 в 07:04

У меня вот полного перехода не получается — к проекту пришлось написать собственный велосипед с почти полным копированием возможностей Yii.

Epsiloncool 27 августа 2012 в 17:15

Как только увидел стрелку от View к Контроллеру с пояснением «Вызов метода контроллера», то решил, что дальше читать не стОит :)

tac 28 августа 2012 в 00:20

Правильно, **от** View не должно быть стрелок!

VolCh 28 августа 2012 в 01:40

Серьёзно? Ничего что именно View отвечает за взаимодействие с пользователем? Двухстороннее взаимодействие, а не одностороннее. В случае веба стрелки от View — это ссылки, формы, ajax-запросы и т. п.

tac 28 августа 2012 в 10:01

Серьезно. Двухстороннее взаимодействие — а какого хрена мы тогда говорим о разделении визуализации и бизнес-логики? Она тогда не разделе определению, и все это тогда наука не о чем.

Вместо этой классической хрени MVC — контроллер может элементарно подписать на события View, нужные методы Модели — и устранить это излишние двухстороннее взаимодействие.

VolCh 28 августа 2012 в 10:50

Бизнес-логика — логика модели предметной области, в которой существуют термины, например, «перевести сумму S со счёта A на счёт B», но н существуют «нажать на ссылку „перевод“».

Подписка на события — один из вариантов реализации MVC, но двухсторонние взаимодействия никуда не деются.

tac 28 августа 2012 в 11:20

«Подписка на события — один из вариантов реализации MVC, но двухсторонние взаимодействия никуда не деются.»

Деются. В этом случае у View нет ссылки ни на контроллер, ни на модель.

VolCh 28 августа 2012 в 11:23

Они просто идут через посредников, диспетчера событий и модель представления.

tac 28 августа 2012 в 12:55

Но эта связь скрыта, и не влияет на повторное использование, в отличии от других где необходима непосредственная ссылка на объек Такую скрытую связь на UML диаграмма не отображают, и она не может считаться двухсторонней.

VolCh 28 августа 2012 в 15:02

Это ещё хуже, имхо. Связь есть, но она неявна, скажем модель Blog изменила своё состояние и оповещает слушателей, вызывая ме raiseEvent(new BlogState($this)). Какие классы зависят от BlogState узнать не так просто.

tac 28 августа 2012 в 12:56

Часто это зашито собственно язык программирования.

tac 28 августа 2012 в 12:57

Часто это зашито собственно **в** язык программирования.

alekciy 27 августа 2012 в 17:22 +

>Для сайта domain1.com
Поскольку статья больше ориентируется на новичков, то давай будем последовательны и не прививать плохие привычки в духе той, что я процитировал. За вы про всякие domain.com, site.ru и прочих доменных именах которые приводятся в качестве примеров. Следуйте RFC 2606 (IANA — Example domains, Доме примеров) и прочим полезным спецификациям. Уверяю вас, это не сложно ;)

kostyl 27 августа 2012 в 20:53 –

Лучше бы автор itdumka.com.ua/ на хабре написал ;) чем Вы. Вы взяли у него всю инфу и выложили сюда, в чем смысл?
Те кто знает про MVC это не нужно, тем кто не знает — материала полно. Collection топики это круто, но они должны ИМХО быть крутого качества!

tac 28 августа 2012 в 00:25 –

Классическое заблуждение о пригодности MVC продолжает жить, поразительно. А может ли мне кто-то сказать для чего нужно MVC? Разве не для раздел бизнес логики от визуализации? Нет? А Вы думаете представленная модель MVC с этим справляется? Вот оно заблуждение…

Как правильно организовать разделение визуализации и бизнес-логики я писал ранее тут.

VolCh 28 августа 2012 в 01:46

Не только, трехкомпонентная сущность MVC предполагает, как минимум, разделение логики на три части: вида (то, что вы называете визуализацией), м (бизнес-логика) и контроллера (всей остальной, логики приложения, как-то: логика хранения, логика контроля прав доступа).

VolCh 28 августа 2012 в 01:46

*доступа и т. п.

tac 28 августа 2012 в 10:06

Логика приложения — это другой слой, как правило оформляемый как ядро системы и используемый напрямую в бизнес-логике, делать ответственны это контроллер — несерьезно и незачем. Это лишь запутывает систему взаимодействий — у вас тогда никакая логика без обращения к контроллеру будет работать.

VolCh 28 августа 2012 в 11:04

Я привык к подходу где бизнес-логика не зависит от других частей приложения. Контроллер должен знать о модели, вид может знать о модели, но модель, имхо, должна быть самодостаточной, максимум содержать логику хранения (а-ля ActiveRecord), но и то, имъо, это будет лишняя связаннос запутвыающая систему.

tac 28 августа 2012 в 11:25

В том то и дело, что в MVC модель не самодостаточна. Легко проверить — часто бывает, что модель нужно использовать без визуализации — т выполнение. Тогда ни представление, ни контроллер не нужны — проверьте сможет ли заработать без этих классов модель — в общем случае будет обращаться к представлению, чтобы отправлять обновленные данные.

VolCh 28 августа 2012 в 11:31

Имхо, либо представление должно получать данные от модели непосредственно (свойства, геттеры), либо ему данные должен предоставлят контроллер (возможно подписывая представление на события модели). А модель ведёт себя либо пассивно, либо посылая сообщения об изм своих данных абстрактному подписчику на них.

tac 28 августа 2012 в 12:00

Я тут открыл Фаулера, там черному по белому написано «В основе MVC лежит разделение кода пользовательского интерфейса (представ называвшегося раньше view, а сейчас чаще presentation) и логики предметной области (модели)». Поэтому не путайте меня «логикой прил в контроллере» — это частные додумки, и далеко не эффективные.

Что касается первого варианта «представление должно получать данные от модели непосредственно (свойства, геттеры)» — это вообще вариант — это того ради чего вообще затевается MVC с его тяжеловесной конструкцией. У Фаулера опять же описан более классический вариант под названием «Дублирование видимых данных» — это тот рефакторинг который нужно осуществить если в программе есть реал вашего первого варианта.

А вот один из вариантов реализации «Дублирования видимых данных» — это через события, но в C# + WPF проще — там есть технология

VolCh 28 августа 2012 в 12:17

Всё правильно. Логика UI находится во вью, логика предметной области — в модели. Всему остальному остаётся место только в контр (декомпозицию его функций никто не отменял).

tac 28 августа 2012 в 12:19

Всего остального — НЕТ.

VolCh 28 августа 2012 в 15:08

Как нет? Логика хранения, например, контроль прав и т. п., это и не UI, и не логика предметной области.

tac 28 августа 2012 в 16:03

Как мы говорили ниже — это логика предметной области, только другой.

tac 28 августа 2012 в 11:26

И да если модель не а-ля ActiveRecord, то как же её можно считать самодостаточной, если она не сохраняется?

VolCh 28 августа 2012 в 11:50

Сохранение данных между запусками приложения (для PHP аналогично «между запросами») — задача логики приложения в общем случае, модели — моделировать предметную область, как ни странно. Скажем в бухучёте нет терминов «сохранить документ» или «загрузить докум есть «создать документ» и «уничтожить документ» по каким-то событиям предметной области (приход товара, истечение срока хранения), в которую не входит «авария в электросети, компьютеры выключились» или «PHP сбрасывает среду выполнения для каждого запроса». А так модель самодостаточна, она как бы содержится в памяти с первого запуска приложения до последнего и лишь в силу неидеальности этого м *приложению* приходится заботиться о сохранении и загрузки.

tac 28 августа 2012 в 12:07

Так то оно так, но тогда это надо решать чище. Работа с базой данных — это по сути тоже бизнес-логика, только не бухгалтерии наприме именно так и называемое «хранение данных». Тогда в модели бухучета должны быть события «ОкончаниеСозданияДокумента» и «ОкончаниеУничтоженияДокумента», тогда контроллер должен подписать методы бизнес-логики базы на эти события. Но это слишком ч для не совершенного мира программирования, и часто вместо этого у модели стоит прямо вызов метода ядра предназначенный для сохра данных (а-ля ActiveRecord).

VolCh 28 августа 2012 в 12:18

Работа с базой данных — это по сути тоже бизнес-логика

Нет, это не бизнес-логика, это техническое решение.

tac 28 августа 2012 в 12:23

Блин, (чего тупим ;)) — пусть Вы работаете в Microsofte и разрабатываете SQL Server Enterprice Manager, какая у Вас предметная об
В чем состоит бизнес-логика?

VolCh 28 августа 2012 в 13:16

Я говорил, что в общем случае. Но опять же если взять, например, phpmyadmin то он взаимодействует с БД двумя способами — и
предметной областью, моделируя в памяти структуру и данные БД, и как со средством хранения части своего состояния между
запросами.

tac 28 августа 2012 в 13:22

ИМХО, вот это и является проверкой решения на прочность, если от изменения предметной области (а работа в базой как мы
выяснили в частном случае может быть бизнес-логикой) — меняется вся архитектура приложения — значит в такой архитект
проблемы. Поэтому лучше всегда одинаково относится к бизнес-логике, и лишь разные предметные области разделять по сло
Тогда бухгалтерия — это высокоуровневая бизнес-логика, а работа с базой — это несколько вырожденная, но все равно бизн
логика, но уже низкоуровневая.

VolCh 28 августа 2012 в 15:18

Не меняется от изменения предметной области вся архитектура. Если я буду писать SQL Manager, то в модели предметной
области у меня будут классы типа Database, Table, Column, Relations и т. п. для моделирования предметной области, при эт
Database, Table и Column будут получаться из SQL-запроса `SHOW ...` аналогично другим классам, зависящим от внешних
сервисов. А Relation будет храниться архитектурно отдельно, на физическом уровне это может быть та же БД, может быть
а может вообще быть не БД.

tac 28 августа 2012 в 11:28

И да если модель не может решить, визуализировать себя или нет — как она может быть самодостаточной? А такое решение и есть бизнес-лог
вида взаимодействовать с пользователем, или работать в автономном режиме. И вот только в случае необходимости GUI возникает отделение
деталей визуализации от бизнес-логики.

VolCh 28 августа 2012 в 11:56

В предметной области нет, как правило, и термина «визуализация», есть бизнес-процессы в различном состоянии и правила изменения этих
состояний. Сохранность этих состояний между запусками приложения или их визуализация — не здача модели. Возлагая на неё эти задачи
минимум, нарушаем принцип единственной ответственности искусственно усложняя приложение.

tac 28 августа 2012 в 12:11

Вы не поняли. Я же не говорю, что в предметной области есть термин «визуализация». Но именно бизнес-логика решает взаимодействова
пользователем или нет, например, один платеж можно провести автоматически, а второй нужно показать для утверждения пользователю
в терминах предметной области и возникает необходимость в GUI. Таким образом, решение на верхнем уровне о визуализации диктуется
бизнес-логикой — и это первейшая задача модели, решить нужен ей пользователь или она может работать автономно и все это в зависим
от определенных бизнес-правил.

VolCh 28 августа 2012 в 13:11

Имхо, в терминах предметной области «нужно показать для утверждения пользователю» означает переход в состояние типа «ожидани
подтверждения» или «черновик». «Показать» — логика приложения, «как показать» — логика представления.

tac 28 августа 2012 в 13:26

Можно и так. Но это будет именно то чистое решение, о котором я говорил выше.

tac 28 августа 2012 в 13:47

По сути наличие «ожидание подтверждения» означает, что есть некий диспетчер, который выбирает такие документы из очереди,
практике многие ленятся так полноценно реализовывать.

VolCh 28 августа 2012 в 15:29

Полноценно обычно избыточно, хотя и доставляет неудобства, если правила меняются.

BlessMaster 28 августа 2012 в 01:49

А Вам не приходила идея, что «логик», так же как и представлений может быть много? Логика хранилища данных, логика бизнеспроцессов, логика
пользовательского интерфейса (да-да, интерфейс это не просто статичная страничка «отчёта» со ссылками и формочками). Всё это разные уровни един
системы и MVC прекрасно применяется на каждом.

Соотношение примерно такое же как между транзакциями базы данных и транзакциями бизнес-модели. Вроде бы и похожие вещи, но не одно и то же и
через другое не реализуется.

tac 28 августа 2012 в 10:10

«логик», так же как и представлений может быть много — естественно, только если Вы от View обращаетесь и к модели и к контроллеру — вы тем са зацементировали эту связку 3 классов. Попробуйте теперь полностью выкинуть ваш контроллер и модель и подсунуть представлению совершенно др бизнес-логику и контролер — у них не будет того, что вызывает View и все у вас захлебнется…

VolCh 28 августа 2012 в 11:20

View не подразумевает независимости от моделей и контроллеров в общем случае. Можно построить архитектуру, чтобы не зависело, если задать такой целью, но в общем случае это не подразумевается. Отделение не значит независимость.

tac 28 августа 2012 в 12:18

> Отделение не значит независимость.

Тогда грош цена такому отделению. И именно поэтому я начал с того, что «Классическое заблуждение о пригодности MVC продолжает жить, поразительно. „

VolCh 28 августа 2012 в 15:33

Представление должно знать, что оно представляет. Иначе возможны трудноуловимые ошибки, которые статически не выявишь.

tac 28 августа 2012 в 16:02

Сомнительное утверждение.

masterrr 28 августа 2012 в 08:19

Вот эту функцию можно повредить передачей $data['file_name']

```
function view($file_name, $data = null)
{
if(is_array($data)) {
// преобразуем элементы массива в переменные
extract($data);
}
```

Stac 29 августа 2012 в 20:11

О… прочел теорию. Дайте-ка ругнусь, пока не приступил к практике.

Сам я не особо силен в MVC и прочих поименованных паттернах. Иногда думаю, почему?
Да потому что в подобных статьях всегда непойми как объясняется теория.

Вот, концептуальная диаграмма с кучей связей между тремя элементами. Хотя ниже в сопроводительном тексте сказано, что контроллер нужен для для св модели и представления. Минуточку… Но на концептуальной схеме мы видим, что модель и представление могут взаимодействовать между собой.

Чему верить? Уже в этот момент у новичка начинаются трудности с пониманием.

Дальше идет описание жизненного цикла приложения с использованием кучи умных и не нужных слов. И это запутывает.

Я например, знаю, что жизненный цикл веб-приложения примерно такой:

1. — клиент посылает HTTP-запрос серверу (метод, uri).
2. — приложение запускается, определяет метод и uri
3. — выполняет код в соответствии с uri и методом.
4. — возвращает сгенерированный HTML (XML, JSON, GIF, whatever)
5. — приложение завершается.

Просто, понятно, знакомо. (IMHO. Допускаю, что для кого-то не просто, не понятно и не знакомо).

И где тут что. Например, п.2. это контроллер, п.3 — модель, п.4 — представление.

Прав я или нет? Верно я понял концепнию или не совсем?

Я пока не знаю, что мы будем делать в Практике (не дочитал топик и не знакомился еще с другими комментами), надеюсь будет интересно.

Но вот как я вижу мини-фреймворк после прочтени теории.

```php
<?php
// контролер
$method = $_SERVER["REQUEST_METHOD"];
$uri = $_SERVER["REQUEST_URI"];

// обращаемся за данными к модели
$model = getModel($uri);
$data = getModelData($model, $method, $uri);

// обращаемся за HTML-кодом к представлению.
$view = getView($method, $uri, $data["result"]); // вид зависит от того, как отработала модель
$output = getViewOutput($view, $data);
```

```
echo $output;
?>
```

Как-то так, да?

Ну все, пошел читать дальше.

Спасибо за статью, вызвала неподдельный интерес.

VolCh 31 августа 2012 в 17:07

1. — клиент посылает HTTP-запрос серверу (метод, uri).
2. — приложение запускается, определяет метод и uri
3. — выполняет код в соответствии с uri и методом.
4. — возвращает сгенерированный HTML (XML, JSON, GIF, whatever)
5. — приложение завершается.

С использованием MVC будет скорее так:
1. — клиент посылает HTTP-запрос серверу (метод, uri).
2. — приложение запускается, определяет метод и uri, вызывает контроллер
3.1 — получает данные (как правило один или несколько объектов/массивов) в соответствии с запросом из модели
3.2 — передаёт данные в вид для генерации HTML,…
4. — возвращает сгенерированный HTML (XML, JSON, GIF, whatever)
5. — приложение завершается.

Важно понимать, что пункт 2 это ещё не контроллер, обычно это называют маршрутизацией, осуществляется или веб-сервером, или фронт-контроллеро
есть в вашем примере контроллер это, как минимум,

```
$model = getModel($uri);
$data = getModelData($model, $method, $uri);

// обращаемся за HTML-кодом к представлению.
$view = getView($method, $uri, $data["result"]); // вид зависит от того, как отработала модель
```

, а то и остальные две строчки. Ну и, как правило, uri и method не передаются в модель и представление напрямую, а из них извлекаются необходимые д
данного запроса параметры и определяется какие модели вызывать и какие вью использовать. Модель и вью как бы изолируются от HTTP, не
`User::getByUri($uri)`, а `User::getById($_GET['user_id'])`

Stac 31 августа 2012 в 18:15

Тут вроде понятно… весь мой пример скрипта — контроллер.

Модель и представление, типа, определены в других файлах.

Насчет uri не совсем понятно. С учетом того, что Uri — universal resourse identifier, чем он хуже любого другого идентификатора?

Впрочем, я тоже не всегда использую uri, а извлекаю из него параметры. Но иногда (в удачный день :) получается использовать uri без дополнительно
обработки.

VolCh 31 августа 2012 в 19:28

В принципе да, весь код контроллер, но когда таких контроллеров будет два, три и т. п. вы наверняка выделите часть кода в один, скажем, index.p
котором будете инклудить зависящий от запроса конкретный контроллер.

Слишком универсальный в контексте контроллера, Допустим, для запроса `GET /users/volch` к моменту получения объекта с id volch уже ясны ти
объекта, что нужно его именно получить (а не удалить и т. п.) и достаточно вызова $users->getById('volch'), а не $users->getByIUri('/users/volch'), кото
придётся дополнительно разбирать. Хотя, конечно, это вопрос удобства и архитектуры могут быть разными, но как-то наиболее популярна при кот
при разборе метода и части uri определяется собственно обработчик (контроллер), а при разборе оставшейся части uri и post/put данных определяи
параметры, которые ему передаются, часть которых он передаёт в модель, а часть использует сам например для выбора формата представления.

Stac 31 августа 2012 в 20:09

Разобрать uri не так и сложно, если заранее о его (или ее? мне нравится «она», правильнее, особенно по-английски — «он») структуре подумать

```
list($empty, $db_name, $id) = explode("/",$uri);
```

Да и вообще сам uri может выступать идентификатором ресурсе в БД, XMLфайле или key-value хранилище.

Конечно, не всегда это может быт правильно или удобно.

VolCh 31 августа 2012 в 22:19

Разобрать, конечно, не сложно, вопрос в том, где это делать и сколько раз. Особенно если есть вероятность, что правила разбора могут меня

vitalyswipe 1 сентября 2012 в 12:35

Ругаться тут не надо. Представьте свою схему и пояснения к ней и почему она более актуальна чем приведенная в статье.

Stac 1 сентября 2012 в 18:20

Вы — учитель. Я лишь заметил странное несоответствие схемы и ее описания.

taboo1387 31 августа 2012 в 14:16

Пешил попробовать.

```
Fatal error: Call to undefined method Load::veiw() in /Applications/MAMP/htdocs/backoffice/application/controller.php on line 19
```

И дальше никак :(

taboo1387 31 августа 2012 в 14:17

*Решил
Сорри

namikiri 11 ноября 2012 в 18:35

Модель, **ВИД**, контроллер… Оригинально!

Scorpion97 11 июня 2013 в 20:08 *(комментарий был изменён)*

Спасибо за статью!

battrack 25 сентября 2013 в 14:24 *(комментарий был изменён)*

Занялся изучением Yii. Решил начать с повторения основ ООП и изучения паттерна MVC. Ваша статья ОЧЕНЬ помогла! Большое спасибо!

olololchik 30 октября 2014 в 22:58

Видео вариант статьи

mvcphp.ru

Только полноправные пользователи могут оставлять комментарии. Войдите, пожалуйста.

ИНТЕРЕСНЫЕ ПУБЛИКАЦИИ

Прокачиваем NES Classic Mini — продолжение
+37 3,5k 19 3

Дорога к C++20
+27 6,3k 22 13

Метод BFGS или один из самых эффективных методов оптимизации. Пример реализации на Python
+10 3,5k 51 3

Инопланетная болтовня: на космической вечеринке разум будет слышно лучше всего GT
+23 7,6k 21 47

Астрономы открыли самую маленькую звезду за все время наблюдений GT

+25 8,5k 13 13

Аккаунт
Войти
Регистрация

Разделы
Публикации
Хабы
Компании
Пользователи
Песочница

Информация
О сайте
Правила
Помощь
Соглашение
Конфиденциальность

Услуги
Реклама
Тарифы
Контент
Семинары

Приложения
Загрузите в App Store
ДОСТУПНО В Google

© 2006 – 2017 «ТМ»
Служба поддержки
Мобильная версия