# 附录

## 外文翻译

### Original (Excerpt from *Pro PHP: Patterns, Frameworks, Testing and More*)

**MVC Architecture**

**M**odel-View-Controller (MVC) is a design pattern that simplifies application development and maintenance. It achieves this by separating the application into three logical components:

*Model*: The model layer is responsible for the business logic of an application. It will encapsulate access to data stores and will provide a reusable class library. Typically, within the model, you will find facilities for database abstraction, e-mail delivery, validation, and authentication.

*View*: The view layer is typically what would be considered web design, or templating. It controls the look and feel of data and provides facilities to collect data from the user. Technologies exclusively found in the view are HTML, CSS, and JavaScript.

*Controller*: The controller layer glues everything together and merges the styling of the view with the functionality of the model. It is responsible for collecting input data from the view and deciding program execution. The controller will call model facilities and interpret the returning data so that it can be rendered by the view. It is also responsible for all application exception and flow control.

This chapter introduces the MVC pattern and its advantages. In the final section, you will create a minimal MVC framework to see how such a framework actually works.

**Why Use MVC?**

You might be wondering why MVC is so popular within web application development. While there are many compelling reasons to use an MVC framework—from cleanliness of code to ease of upgrading—the number one reason companies choose MVC development is the development workflow that it encourages.

In teams, MVC allows for efficient collaboration by splitting up responsibilities into three main roles:

*Development*: The developers are programmers working on the model. They will typically have skills related to PHP, database administration, algorithms, architecture, and data validation. This role will typically be responsible for the programming details (the “how”) of an application, and will provide APIs and enforce policy for interacting with data.

*Design*: The designers handle the view and are responsible for how an application looks and feels. They will be highly skilled in technologies like HTML, CSS, JavaScript, and graphic design. Typically, this role will be responsible for interacting with both internal and external communication sources to determine realistic business rules for which an application will be developed or improved. Design usually leads to prototype development, creating mock-up designs showing ideal functionality.

*Integration*: The integration exists in the controller layer, which glues the work of designers and developers together. Integrators will typically have less experience than developers, and will be responsible for cutting up static templates and making the required regions dynamic. They are also responsible for brokering data from request sources. They will take request information from forms, pass the information to the model, interpret the results, and pass the information along to the view.

By providing division of responsibilities, the details of *how* are separated from the details of *why*, which allows the creative and technical disciplines to interact more easily.

In a typical workflow, first a designer will put together a static, stand-alone, prototype design based on the existing business requirements and how the application should behave. The designers will then pass the design to the developers for review.

Developers are responsible for ensuring that all of the requested functionality is feasible and fits with the company’s security and privacy policies. If everything checks out, a development plan, including a mock-up API, will be created and passed to the integration role. If there are any issues with the prototype, the project will go back to the designer, and the process will continue again from the start.

Once the design prototype and the mock-up API are in place, the integration work begins. The integrators will dissect the prototype design and convert it into a template language like plain PHP, Smarty, and so on. This will enable the design to handle dynamic data, and will integrate the site URL structures for which the integrator role is solely responsible. Then a controller component is written to broker requests (forms, URL parameters, cookies, and so on) between the web server and the model information they are meant to manipulate. When a result is produced, the view will be retrieved and rendered, with the controller providing the values for any dynamic content.

Finally, when the mock-up API is fully implemented, the application can be handed off for testing. If the integrator wrote the controller to spec, and the model developers conformed to their API, then everything should work as expected.

As well as distributing responsibility between team members, the MVC architecture also offers several other key benefits. One is that it allows the presentation layer to be modified and updated at any time, as long as it does not break the contract with the controller. This means that copy editing or theme changes can be readily accomplished without changing the way the application works or sorting through a lot of PHP code.

The MVC architecture also provides a useful subdivision of the file structure. Each MVC framework has a different structure and layout, but with few exceptions, they all provide separate files for models, views, and controllers. This feature will reduce code conflicts and will save time that would otherwise be spent merging changes in your revision control system.

**MVC Application Layout**

The layout for MVC applications differs by framework; however, some concepts are common to all MVC frameworks. In the next chapter, you will learn about the specifics of the Zend Framework layout. The explanations in this section refer to the typical implementation of the MVC pattern in PHP.

**From the Web Server**

Unlike other PHP applications, MVC sites typically implement a centralized PHP script that handles all requests on a web site. So, instead of the browser going to /*path*/*to*/*somefile*.php, the browser goes to /*controller*/*action*. However, you won’t find a folder named controller or a file named action. Instead, a URL rewriter like mod\_rewrite is used to redirect *all* HTTP requests to a centralized script that is called the *bootstrap file*.

The bootstrap file is responsible for initializing the framework: loading files, reading configuration data, parsing the URL into actionable information, and populating the objects that encapsulate the request. Finally, it is responsible for initializing the controllers.

**Actions and Controllers**

After bootstrapping, a class called the front controller (FC) is instantiated and takes over. The front controller is typically a built-in class that is responsible for interpreting request variables and routing code execution to a user-defined class, which is called an action controller (AC). Usually, an action controller will implement a standard interface or descend from an abstract class, so that the front controller and action controller may interact using a common API.

The front controller will then invoke a method on the action controller specified by the URL. This method is called an *action*, and its name, like the action controller’s name, is determined from the URL. The action is responsible for all the dirty work. In this method, you will instantiate model classes, parse view templates, and output the results.

**Models**

The models are typically the easy part. They usually don’t follow any particular structure other than to exist in a common location, so that they may be automatically loaded by the framework. Models are simply utility classes that provide the required data manipulation and parsing functionality.

**Views**

The views are templates and can be written in any template language. The primary goal here is to not include any sort of processing logic or data manipulation in the view, and output only data that has been provided specifically by the controller. Typically, this means limiting yourself to displaying arrays, not data objects, when operating in the view.

The choice of templating language is your own, depending on whether it can be integrated with your framework. Personally, I find that plain PHP makes for the best templating language when working in MVC environments. FastTemplate and Smarty are viable second choices.

**Criteria for Choosing an MVC Framework**

Every MVC framework has certain advantages and certain downsides. You should consider five key factors when choosing a framework to use: architecture, documentation, community, support, and flexibility. I suggest that you sit down and score all the frameworks you are considering using these criteria, and then pick the most appropriate one for you.

**Architecture of the MVC Framework**

Every developer is different. What makes sense to one developer might be totally illogical to another. You should choose a framework that you understand. I have met far too many distressed developers using a complicated MVC framework because they have heard about all the benefits it may provide. I assure you that without an architectural understanding of your framework, it will hinder, not help, your development. Choose one that you are comfortable using.

You should also consider whether you want a framework that follows a linear function and callback style or an object-oriented approach. Personally, I prefer the latter because the concept of encapsulation really works wonders with model classes.

You will need to understand the framework’s technical requirements and file structure, so that you can be sure it will work effectively with your deployment procedures. For example, some frameworks have very backward-compatible APIs, whereas others do not even attempt to maintain backward compatibility. There are valid arguments for both approaches. Choose the one that works for your organization.

**MVC Framework Documentation**

All frameworks come with documentation, and that’s where the similarities end. Documentation can be as simple as an install file, or as complex as formal training courses.

When I look at a framework’s documentation, I ask three key questions:

• Is it current? Documentation is useless if it’s talking about a version that you are not using. Ensure that the version you download is the version that is documented.

• Is it clear? Is the documentation well written and easy to understand? If you can’t understand what it is trying to say, move on.

• Is it complete? Is the documentation comprehensive? Does it cover everything in the framework or just the basics? All too often, you will get a combination of great introductory documentation and machine-generated advanced docs.

Good docs are essential to meeting deadlines and keeping costs down.

**MVC Framework Community**

The existence of community around an MVC framework is critical. You should look at what amount of participation is welcomed, and how long it takes members of the community to respond to a bug and produce a patch.

Community is also the number one factor managers will need to be concerned with, as it will heavily affect their ability to hire new staff members that are knowledgeable with the framework.

You should be able to communicate directly with the developers of your framework to report bugs, suggest features, and obtain fixes. However, the framework developers should *not* be considered technical support, which is an entirely separate, though related, topic.

The time between releases will give you an idea of how active the project is. This usually gives a good indication of the frequency of future updates. You will need to evaluate how fast your business can reasonably adapt to a changing framework. There are definite trade-offs between API stability and the introduction of new features, so your mileage may vary.

**MVC Framework Support**

As I noted in the previous section, the developers of your framework are not support. If you send them so much as a single e-mail with an “ignorance” question, you will be highly unlikely to receive future assistance that only they can provide. Consider this “crying wolf” to the development team.

So with that in mind, who can you ask when you have a support question? Some MVC frameworks are supported through a support-revenue model, where the framework is free, but support will cost you. Others offer no support at all; in those cases, you will need to find a knowledgeable third party.

Mailing lists and support chat channels will provide free support for a given framework. These sources can be highly useful, but may also prove highly unreliable compared with paid support. To get the best results from these sources, a share-and-share-alike attitude must be adopted—those with the most to offer should also get the most help.

Before beginning development with any particular framework, be sure you fully understand your support position. Getting stuck on problems for which you cannot quickly find the answer can be very costly.

**MVC Framework Flexibility**

How flexible is the MVC framework’s implementation? Can you integrate your existing applications within the framework, and make sure it supports your databases?

Some frameworks are incredibly rigid and will place many restrictions on how you write and organize your code. They may dictate templating languages, and they may have very specific syntax that will incur a significant training cost.

Look for a flexible framework that imposes as few hard limits as possible.

**Roll Your Own MVC Framework**

It is generally recommended to choose from one of the publicly available frameworks, rather than creating your own. No matter how good a programmer you are, you will always flunk in the community department. However, to truly understand MVC, it helps to know how to write your own basic framework, so that’s what you’ll do now.

In this example, you will follow a layout similar to the Zend Framework, utilizing techniques you have learned in prior chapters and implementing a typical front controller/action routing system.

For this example, you will need a web server running Apache 2, PHP 5, mod\_rewrite, and a database like PostgreSQL. The instructions in this chapter are for a Debian Linux installation, but they will work on all PHP-supported platforms.

■**Note** If you don’t have a server with the required configuration, all the necessary applications can be downloaded to your desktop. Their installation is not complicated. Dive right in, and you’ll more than likely be able to figure it out.

**Setting Up a Virtual Host**

First, you need to create a virtual host and a new site. Follow these steps:

**1.** Execute the following commands to create a virtual host:

> mkdir -p /usr/local/www/*yourdomain*.com/*document\_root*

> cd /etc/apache2/sites-available

**2.** Edit a file called *yourdomain*.com.

> pico –w *yourdomain*.com

**3.** Place the virtual host definition in the *yourdomain*.com file.

<VirtualHost \*>

*DocumentRoot* /usr/local/www/*yourdomain*.com/*document\_root*

*ServerName yourdomain*.com

</VirtualHost>

**4.** Execute the following commands to activate your site:

> a2ensite *yourdomain*.com

> /etc/init.d/apache2 reload

**5.** Make your browser access your site. You can do this by pointing the Domain Name System (DNS) for the domain at the server or by editing your hosts file. To edit your hosts file on Windows, locate the file at C:\WINDOWS\system32\drivers\etc\hosts. Open it in a text editor and add a new line. Type the IP address of your server, press Tab, and then type the name of the file (*yourdomain*.com) you just created. Save the file.

**6.** Open a new browser instance (if you have a browser open, it may need to be closed first to pick up the hosts file change) and browse to your new site. You should see an empty directory index.

You’re now ready to start developing your very own MVC framework.

**Creating an MVC Framework**

At this point, you will need to start creating directories. I recommend the following layout:

> cd /usr/local/www/yourdomain.com

> mkdir document\_root/images document\_root/styles

> mkdir –p application/models application/views \

> application/controllers

> find

.

./document\_root

./document\_root/images

./document\_root/styles

./application

./application/models

./application/views

./application/controllers

Next, create a placeholder for your bootstrap file, as shown in Listing 14-1.

**Listing 14-1.** *Creating the Bootstrap index.php*

> pico document\_root/index.php

Hello, World!

At this point, reloading your browser should result in “Hello, World!” being shown. Navigate to http://yourdomain.com/nonexistent. You should get a 404 File Not Found error.

For this MVC framework to work, you need every single request to hit this index.php. To do this, you enlist mod\_rewrite through an .htaccess file, as shown in Listing 14-2.

**Listing 14-2.** *Using mod\_rewrite in an .htaccess File*

> pico document\_root/.htaccess

RewriteEngine on

RewriteRule !\.(js|gif|jpg|png|css)$ index.php

Reload your browser, and you should see “Hello, World!” If you don’t, you will need to ensure that your server has mod\_rewrite available and does not restrict .htaccess overrides with a nonstandard allowOverrides directive.

**Bootstrapping**

The next stage in this process is to create a bootstrapping script in your index.php file. This file will be responsible for including files and initializing the front controller. Most MVC frameworks will use SPL autoloading mechanisms to load files, but for simplicity, you will just require\_once for now. Create the file as shown in Listing 14-3.

**Listing 14-3.** *The Bootstrap Loader*

> pico –w document\_root/index.php

<?php

//Require Components

require\_once('../application/models/front.php');

require\_once('../application/models/icontroller.php');

require\_once('../application/models/view.php');

//Require Controllers

require\_once('../application/controllers/index.php');

//Initialize the FrontController

$front = FrontController::getInstance();

$front->route();

echo $front->getBody();

■**Note** You do not need a closing ?> in Listing 14-3, as the end of file will close the tag automatically. It is not recommended to include closing tags, as newlines at the end of the file can result in premature output and may cause issues sending HTTP headers.

**The Front Controller**

The front controller’s job is to parse the URL and to instantiate the controller and invoke the action method. This is a singleton class (see Listing 3-1 in Chapter 3), which is responsible for the application’s program flow.

The theory of operation is that URLs follow the format /controller/action/key1/value1/. . . . Once the URL has been parsed, you will use the reflection API (covered in Chapter 7) to invoke the action method on the IController implementing class.

To get started, first create the FrontController class, as shown in Listing 14-4.

**Listing 14-4.** *The FrontController Class*

> pico –w application/models/front.php

<?php

class FrontController {

protected $\_controller, $\_action, $\_params, $\_body;

static $\_instance;

public static function getInstance() {

if( ! (self::$\_instance instanceof self) ) {

self::$\_instance = new self();

}

return self::$\_instance;

}

private function \_\_construct() {

$request = $\_SERVER['REQUEST\_URI'];

$splits = explode('/', trim($request,'/'));

$this->\_controller = !empty($splits[0])?$splits[0]:'index';

$this->\_action = !empty($splits[1])?$splits[1]:'index';

if(!empty($splits[2])) {

$keys = $values = array();

for($idx=2, $cnt = count($splits); $idx<$cnt; $idx++) {

if($idx % 2 == 0) {

//Is even, is key

$keys[] = $splits[$idx];

} else {

//Is odd, is value;

$values[] = $splits[$idx];

}

}

$this->\_params = array\_combine($keys, $values);

}

}

public function route() {

if(class\_exists($this->getController())) {

$rc = new ReflectionClass($this->getController());

if($rc->implementsInterface('IController')) {

if($rc->hasMethod($this->getAction())) {

$controller = $rc->newInstance();

$method = $rc->getMethod($this->getAction());

$method->invoke($controller);

} else {

throw new Exception("Action");

}

} else {

throw new Exception("Interface");

}

} else {

throw new Exception("Controller");

}

}

public function getParams() {

return $this->\_params;

}

public function getController() {

return $this->\_controller;

}

public function getAction() {

return $this->\_action;

}

public function getBody() {

return $this->\_body;

}

public function setBody($body) {

$this->\_body = $body;

}

}

**The Controller**

The controller is where you create your application code. It is just a normal class, except that it implements the IController “marker” interface that does not contain any methods. Create the IController interface as shown in Listing 14-5.

**Listing 14-5.** *The IController Interface*

> pico –w application/models/icontroller.php

<?php

interface IController {}

Next, you will create a class to encapsulate your program logic. This class will be responsible for creating an instance of the view (which you’ll define shortly), binding a variable to it, and calling the render method, which will return the parsed result.

Once the view is rendered, the controller sets the body of the FrontController and returns. Create this class as shown in Listing 14-6.

**Listing 14-6.** *A Default Controller*

> pico –w application/controllers/index.php

<?php

class index implements IController {

public function index() {

$view = new View();

$view->name = "Kevin";

$result = $view->render('../views/index.php');

$fc = FrontController::getInstance();

$fc->setBody($result);

}

}

**The View**

The view model is one of the more complex classes in an MVC framework. It is designed to create a scope for variables (such as $view->name in Listing 14-6) and include the file returning the parsed output.

This is achieved by including from within a method, and thus limiting the scope to that method, and by using the output buffering feature of PHP. The actual properties support is provided by the ArrayObject SPL class, using a blank starting array and the special flag ARRAY\_AS\_PROPS, which allows for easy creation of a property overloaded object.

Create the view model as shown in Listing 14-7.

■**Note** This class is called View and is a model. It provides view rendering functionality. The “view” is actually the template defined later on, and will be found in the views folder.

**Listing 14-7.** *The View Model Class*

> pico –w application/models/view.php

<?php

class View extends ArrayObject {

public function \_\_construct() {

parent::\_\_construct(array(), ArrayObject::ARRAY\_AS\_PROPS);

}

public function render($file) {

ob\_start();

include(dirname(\_\_FILE\_\_) . '/' . $file);

return ob\_get\_clean();

}

}

Next, make a default view template, as shown in Listing 14-8.

**Listing 14-8.** *The Default View*

> pico application/views/index.php

Hello, <?php echo $this->name; ?>!

That’s it. Reload your browser, and you should see “Hello, Kevin!”

**URL Parameters**

You’re probably not named Kevin though, so let’s integrate your name with this script. Because of encapsulation, to do this, you need to modify only the controller.

Replace your application/controllers/index.php file with the file shown in Listing 14-9.

**Listing 14-9.** *Using URL Parameters*

> pico –w application/controllers/index.php

<?php

class index implements IController {

public function index() {

$fc = FrontController::getInstance();

$params = $fc->getParams();

$view = new View();

$view->name = $params['name'];

$result = $view->render('../views/index.php');

$fc->setBody($result);

}

}

Now change the URL to http://yourdomain.com/index/index/name/yourname =, and you should see “Hello, *yourname*!”

As you can see, creating an MVC framework is actually pretty basic. Well, this sample framework is so basic that it provides almost no functionality.

The frameworks you will find on the market have significant built-in features, such as database abstraction, data filtering and validation, and so on. In the next two chapters, you will learn about the built-in facilities a typical MVC framework provides.

**Just the Facts**

The MVC pattern is a design pattern that separates an application into three parts: the model, the view, and the controller. MVC is primarily designed to help with web application development workflow and allow teams to work together more efficiently by defining specific roles. These roles are developer, designer, and integrator. Developers are responsible for the model. Designers are responsible for the look and feel of the site. Integrators are in charge of putting it all together.

Most MVC frameworks follow similar concepts, providing classes for controllers and methods for actions. Views are templates, and can be created in plain PHP or a template language like Smarty.

There are a lot of considerations to bear in mind when choosing which framework is right for your project. The five key points are architecture, documentation, community, support, and flexibility.

The introduction to MVC in this chapter provides a basis for the next chapters, which discuss the specific implementation of the Zend Framework.

### 译文（节选自《PHP高级程序设计 模式、框架、测试》）

**MVC架构**

MVC（Model-View-Controller，模型-视图-控制器）是一种可以简化应用程序开发和维护的设计模式。为了实现这一目的，它将应用程序分隔到三个逻辑组件中。

模型（Model）。模型层负责应用程序的商业逻辑。它将封装访问数据库的方法并提供一个可重用的类库。通常，在模型中，你将发现用于数据库抽象、电子邮件发送、校验和验证等机制。

视图（View）。视图层通常包括Web设计或者模板设计等工作。它控制了数据的外观并且提供了从用户手机数据的机制。在视图中经常会找到视图特有的技术，包括HTML、CSS和JavaScript。

控制器（Controller）。控制器层将所有事项连接在一起，并且将视图设计和模型功能合并在一起。它负责从视图中收集输入数据，并决定程序的执行逻辑。控制器会调用模型的机制并解析返回的数据，以便使这些数据能够被视图所展示。它还负责所有应用程序的异常控制和流控制。

本文介绍了MVC模式及其特点。在最后一节，创建了一个小型的MVC框架，从而了解这样的一个框架是如何运作的。

1. **为什么使用MVC**

为什么MVC在Web应用程序开发领域中如此流行呢？虽然有很多理由值得你去使用MVC框架，其中包括代码的简介以及升级的便利等，但是业界公司选择MVC开发应用程序的首要原因是它提倡的开发流。

在开发团队中，MVC提供了有效的合作模式，它将各人的职责分成为三个主要的角色。

* 开发。开发人员是指处理模型的程序员。他们通常拥有与PHP、数据库管理、算法、架构和数据校验等方面相关的技能。这一角色通常会负责实现应用程序的编程细节（操作），它们会提供API，同时实施处理数据的策略。
* 设计。设计人员处理视图并负责实现应用程序的外观。他们具有如HTML、CSS、JavaScript和图形设计等方面的技能。通常，这一角色负责与内外部的通信源交互以决定要开发或者增强的应用程序的现实的业务规则。设计通常会导致原型的开发，也就是创建能够显示理想功能的模型设计。
* 集成。集成工作存在于控制器层中，它将设计师和开发人员的工作连接在一起。集成人员的经验通常比开发人员要少，他们负责切割静态模板并且制作应用程序所需的动态区域。他们还负责代理来自请求源的数据。他们将从表格中获取请求信息，将这些信息传递给模型，解释结果，并且将结果传递给视图。

通过职责分工，开发方法的细节与开发需求的细节分开了，这使得创意学科和技术学科能够更加容易地交互。

在一个典型的工作流中，首先设计师会根据现存的业务需求以及应用程序所要实现的功能，制作一个静态的、单独的原型设计。然后设计师将设计内容交给开发人员评审。

开发人员负责确保所有要求实现的功能是可行的，并且符合公司的安全和隐私策略。如果开发人员没有发现任何问题，他们就会创建一个开发计划，其中包括一个API模型，然后再交给集成人员。如果原型出现问题，项目会回到设计人员手中，这一过程就再次从头开始。

准备好设计原型和API模型之后，集成工作就会开始。集成人员将研究原型设计，并将它转换成模板语言，如普通PHP、Smarty等。这使得设计内容可以处理动态数据，并且将设计集成到完全有集成人员负责的站点URL结构中。然后编写一个控制器组件，这一组件会将请求（来自于表格、URL参数、cookie等）在Web服务器和他们需要操作的模型信息之间代理分发。结果产生之后，获取视图并输出，并且控制器会提供所有动态内容的值。

最后，当API模型完全实现之后，就可以将应用程序交给测试人员测试了。如果集成人员根据规范来编写控制器，并且模型开发人员遵循API编写代码，那么一切都会顺利进行。

MVC架构除了可以在团队成员之间分配职责之外，它还提供了几个非常关键的功能。其中一个是只要表现层的设计不破坏与控制器的约定，它就允许在任何时间修改和更新表现层的设计。这意味着可以很容易地编辑副本或者变化主题，而不用改变应用程序的操作方法或者整理大量的PHP代码。

MVC架构还提供了一个有用的文件结构的分类方法。每个MVC框架都有一个不同于其他框架的结构和布局，但是他们全部为模型、视图和控制器提供了单独的文件，并且很少例外的情况。这一特性会减少代码冲突，并且会节省原本用于合并版本控制系统中的修改的时间。

1. **MVC应用程序布局**

每个框架的MVC应用程序的布局各不相同。不过，一些概念对于所有的MVC框架来说是通用的。本节介绍的是PHP中MVC模式的典型实现。

1. **从Web服务器开始**

与其他PHP应用程序不同的是，MVC站点通常会实现一个用来处理Web站点上所有请求的核心PHP脚本。所以，浏览器浏览的不是/path/to/somefile.php路径，而是/controller/action路径。不过，你无法找到一个名为controller的目录或者一个名为action的文件。相反，像mod\_rewrite这样的URL重写程序可以用来将所有的HTTP请求重定向到一个核心的脚本文件上，这个文件被称为引导文件。

引导文件负责初始化框架，其中包括加载文件，读取配置数据，将URL解析成可以被动作读取的信息，以及填充封装了请求信息的对象等操作。最后，它还负责初始化控制器程序。

1. **动作和控制器**

引导完成之后，一个名为前端控制器（FC）的类将会被实例化，并且接管程序的运行。前端控制器通常是一个负责解释请求变量的内置类，它会把代码的执行路由到一个用户自定义的类上，这个类被称为动作控制器（AC）。通常，动作控制器将实现一个标准的接口或者从一个抽象类继承，所以前端控制器和动作控制器可以使用一个公共的API来交互。

然后，前端控制器将会调用由URL指定的动作控制器上的一个方法。这个方法被称为动作，它的名称和动作控制器的名称一样，都是由URL决定的。动作负责实现所有的功能。在这个方法中，你将实例化模型类，解析视图模板，并输出结果。

1. **模型**

模型通常是最容易的部分。它们一般不遵循任何特定的结构，而只是存在于一个公共的位置中，所以它们可能会被框架自动加载。模型是提供了所需的数据操作和解析功能的简单的工具类。

1. **视图**

视图是模板，可以用任何模板语言来编写。使用视图的主要目的是不要在视图中包含任何类型的处理逻辑或者数据操作，并且指数初有控制器特别提供的数据。通常，这意味着在视图中操作时，只能显示数组，而不是数据对象。

模板语言的选择在你自己，这取决于它是否能喝目标框架集成在一起。我个人认为，在MVC环境中编程时，普通的PHP是最佳的模板语言。FastTemplate和Smarty是值得考虑的第二选择。

1. **选择MVC框架的标准**

每种MVC框架都具有特定的优势和劣势。选择框架时，应该考虑五个关键因素，即架构、文档、社区、支持和灵活性。建议使用这些标准给考虑要用到的所有框架打分，然后选择一个对你来说最合适的框架。

1. **MVC框架的架构**

开发人员各不相同。一名开发人员认为合理的东西对另外一名开发人员来说可能完全不符合逻辑。应该选择一个自己可以理解的框架。有很多开发人员由于使用了复杂的MVC框架而遇到困难，这是由于他们曾经听说过它可能可以提供所有功能。有一点是肯定的，如果对框架的架构没有完整的了解，那么它将会妨碍程序的开发，而不是有助于开发。所以，选择自己最了解的框架即可。

还应该考虑是使用一个遵循线性函数和回调风格的框架，还是使用一个面向对象风格的框架。可能有一些开发人员比较喜欢后者，这是因为封装的概念对使用模型类来说是非常合适的。

你将需要理解框架的技术要求和文件结构，以便确保它有效地和部署流程一起运作。例如，某些框架具有很好的后向兼容的API，而另外一些则甚至不去尝试维持后向兼容性。这两种方法都有合理的一面。选择一个适合你的系统的即可。

1. **MVC框架文档**

所有的框架都带有文档，但相似点仅此而已。文档可以简单到只有一个安装文件，也可以复杂到有正式的培训课程。

当查看某个框架的文档时，我会问三个关键问题。

* 它是否是最新的？如果它的内容不是当前要使用的版本的内容，那么这个文档是没有用处的，确保下载的版本是文档中说明的版本。
* 它是否清晰？文档的编写是否清晰并且容易裂解？如果不能理解它试图说明的内容，就跳过它。
* 它是否完整？文档的内容是否全面？它是否覆盖了框架中的所有内容？还是只介绍了基本内容？你经常会获得组合文档，它是由很好的介绍文档和机器生成的高级文档组合而成的。

好的文档至少要符合这些条件，并且成本不高。

1. **MVC框架的社区**

支持MVC框架的社区的存在是非常关键的。你应该关注社区成员的参与度，以及社区成员对bug做出响应并修复的时间。

社区也是经理们需要考虑的首要因素，这是因为它会对雇用熟悉这个框架的新员工的能力产生很大影响。

你应该能够与使用框架的开发人员直接交流，以便报告问题、建议特性以及获取问题修复等。但是，不应该把框架开发人员当作技术支持，技术支持是另外一个需要单独讨论的问题，虽然它和社区也是相关联的。

发布版本的时间为你提供了项目的活跃程度的线索。这通常会很好地预示未来更新的频率。你将需要求出业务变化的速度，以便合理地适应变化中的框架。在维持API的稳定性和实现新特性之间，需要作出一些权衡，这也取决你自己的态度。

1. **MVC框架的支持**

在前一节提到过，框架的开发人员并非技术支持。如果针对一个可忽略的问题多次向他们发同一封邮件，就不太可能得到只有他们才能提供的进一步协助。在对待开发团队的问题上，应考虑到开发人员都是比较傲气的。

那么，在这种情况下，当有问题需要支持时，应该问谁呢？有些MVC框架的支持工作通过一种支持-收入模型来进行，也就是说，框架是免费的，但获得技术支持则需要付费。另外一些框架不提供任何支持。如果是这种情况，就需要寻找具有相关经验的第三方组织了。

邮件列表和用于技术支持的聊天频道将为特定框架提供免费支持。这些资源非常有用，但与付费的支持相比有很高的不确定性。为了从这些资源中获取最佳结果，必须采取共享的态度，也就是说付出最多的人也将得到最多的帮助。

当打算使用某种框架开始开发之前，确保了解可以获得支持的情况。当碰到问题时，如果不能迅速找打答案，付出的代价将是很高的。

1. **MVC框架的灵活性**

MVC框架的实现有多大的灵活性？你是否能够将现存的应用程序与这个框架集成，并确保它支持数据库？

某些框架的要求非常严格，会在编写和组织代码方面施加很多限制。这些框架可能会规定模板语言，也可能有非常独特的语法，从而导致很高的培训成本。

你应该寻找施加了尽可能少的限制的灵活框架。

1. **实现MVC框架**

一般来说，推荐你从可获得的公共框架中选择一种来使用，而不是自己去创建一个。再好的程序员也比不上社区的开发能力。不过，充分理解MVC的概念，有助于了解如何编写基本的框架，这正是我们目前要做的事情。

在这个例子中，你将遵循与Zend框架相似的布局，使用之前介绍的技术，并实现一个典型的前端控制器/操作路由系统。

在这个例子中，你将需要一个运行着Apache2、PHP5、mod\_rewrite的Web服务器，以及一个如PostgreSQL这样的数据库。这里是针对Debian Linux操作系统的，但是它们也可以在所有的支持PHP的平台上运行。

**说明** 如果没有具有必须配置的服务器，可以将所有必要的应用程序下载到桌面上，这些应用程序的安装并不复杂。稍微研究一下，便能够完成它们的安装了。

1. **设置一个虚拟主机**

首先，需要创建一个虚拟主机和一个新站点。步骤如下。

1. 执行一下命令，创建一个虚拟主机。

*> mkdir -p /usr/local/www/yourdomain.com/document\_root*

*> cd /etc/apache2/sites-available*

1. 编辑一个名为youdomain.com的文件。

*> pico –w yourdomain.com*

1. 将虚拟主机的定义放在yourdomain.com文件中。

*<VirtualHost \*>*

*DocumentRoot /usr/local/www/yourdomain.com/document\_root*

*ServerName yourdomain.com*

*</VirtualHost>*

1. 执行以下命令以激活站点。

*> a2ensite yourdomain.com*

*> /etc/init.d/apache2 reload*

1. 使浏览器可以访问你的站点。通过在服务器中指定这一域名系统（DNS）或者编辑hosts文件，便可以实现这一目的。要在Windows中编辑hosts文件，需要查找C:\WINDOWS\system32\drivers\etc\hosts文件。在文本编辑器打开它并添加一个新行。输入服务器的IP地址，按Tab键，然后在输入刚刚创建的文件名称（yourdomain.com）。最后，保存这个文件即可。
2. 打开新的浏览器实例（如果已经打开了一个浏览器，可能需要先关闭它，然后才能获得hosts文件的修改信息），然后浏览新站点。应该会看到一个空的目录索引。

现在，你已经准备开始开发你自己的MVC框架了。

1. **创建一个MVC框架**

现在，需要开始创建一些目录，推荐以下的布局：

*> cd /usr/local/www/yourdomain.com*

*> mkdir document\_root/images document\_root/styles*

*> mkdir –p application/models application/views \*

*> application/controllers*

*> find*

*.*

*./document\_root*

*./document\_root/images*

*./document\_root/styles*

*./application*

*./application/models*

*./application/views*

*./application/controllers*

下一步，为引导文件创建一个占位符，如下代码清单所示。

*Listing 14-1. Creating the Bootstrap index.php*

*> pico document\_root/index.php*

*Hello, World!*

然后，重新加载浏览器，你将会看到显示出的“Hello, World!”字样。浏览<http://yourdomain.com/nonexistent>这个地址。应该会碰到一个“404文件未找到”的错误信息。

为了让MVC框架起作用，你需要一个请求来触发这个index.php文件。要做到这一点，需要通过.htaccess文件获得mod\_rewrite的帮助，如下代码清单所示。

*Listing 14-2. Using mod\_rewrite in an .htaccess File*

*> pico document\_root/.htaccess*

*RewriteEngine on*

*RewriteRule !\.(js|gif|jpg|png|css)$ index.php*

重新加载浏览器，你将会看到“Hello, World!”字样。如果没有看到，需要确保服务器上安装了mod\_rewrite，并且没有使用非标准的allowOverrides指令来限制.htaccess文件的重写。

1. 引导

这个过程的下一步是在index.php文件中创建一个引导脚本。这个文件将负责包含文件并初始化前端控制器。大多数MVC控制器将使用SPL自动加载机制来加载文件，但是为简单起见，现在我们只使用require\_once。创建的代码清单如下所示。

*Listing 14-3. The Bootstrap Loader*

*> pico –w document\_root/index.php*

*<?php*

*//Require Components*

*require\_once('../application/models/front.php');*

*require\_once('../application/models/icontroller.php');*

*require\_once('../application/models/view.php');*

*//Require Controllers*

*require\_once('../application/controllers/index.php');*

*//Initialize the FrontController*

*$front = FrontController::getInstance();*

*$front->route();*

*echo $front->getBody();*

说明 上面代码清单不需要使用结束标签?>，这是因为文件的结束字符会自动关闭标签。由于文件末尾的新行会导致过早的输出，并且可能会给发送HTTP头带来问题，所以不推荐大家使用结束标签。

1. 前端控制器

前端控制器的任务是解析URL，实例化控制器，并调用实现操作的方法。这是一个单例类，它负责控制应用程序的程序流。

实现操作的原理是URL遵循/controller/action/key1/value1/…格式。解析了URL之后，使用反射API调用IController实现类的实现操作的方法。

首先，需要创建如下代码清单所示的FrontController类。

*Listing 14-4. The FrontController Class*

*> pico –w application/models/front.php*

*<?php*

*class FrontController {*

*protected $\_controller, $\_action, $\_params, $\_body;*

*static $\_instance;*

*public static function getInstance() {*

*if( ! (self::$\_instance instanceof self) ) {*

*self::$\_instance = new self();*

*}*

*return self::$\_instance;*

*}*

*private function \_\_construct() {*

*$request = $\_SERVER['REQUEST\_URI'];*

*$splits = explode('/', trim($request,'/'));*

*$this->\_controller = !empty($splits[0])?$splits[0]:'index';*

*$this->\_action = !empty($splits[1])?$splits[1]:'index';*

*if(!empty($splits[2])) {*

*$keys = $values = array();*

*for($idx=2, $cnt = count($splits); $idx<$cnt; $idx++) {*

*if($idx % 2 == 0) {*

*//Is even, is key*

*$keys[] = $splits[$idx];*

*} else {*

*//Is odd, is value;*

*$values[] = $splits[$idx];*

*}*

*}*

*$this->\_params = array\_combine($keys, $values);*

*}*

*}*

*public function route() {*

*if(class\_exists($this->getController())) {*

*$rc = new ReflectionClass($this->getController());*

*if($rc->implementsInterface('IController')) {*

*if($rc->hasMethod($this->getAction())) {*

*$controller = $rc->newInstance();*

*$method = $rc->getMethod($this->getAction());*

*$method->invoke($controller);*

*} else {*

*throw new Exception("Action");*

*}*

*} else {*

*throw new Exception("Interface");*

*}*

*} else {*

*throw new Exception("Controller");*

*}*

*}*

*public function getParams() {*

*return $this->\_params;*

*}*

*public function getController() {*

*return $this->\_controller;*

*}*

*public function getAction() {*

*return $this->\_action;*

*}*

*public function getBody() {*

*return $this->\_body;*

*}*

*public function setBody($body) {*

*$this->\_body = $body;*

*}*

*}*

1. 控制器

控制器是创建应用程序代码的地方。除了需要实现不包含任何方法的IController“标记”接口之外，它只是一个普通的类。创建如下代码清单所示的IController接口。

*Listing 14-5. The IController Interface*

*> pico –w application/models/icontroller.php*

*<?php*

*interface IController {}*

下一步，需要创建用于封装程序逻辑的类。这个类将负责创建视图的实例，将变量绑定到视图上，然后调用提交方法，最后，这个方法将返回解析后的结果。

完成视图输出之后，控制器将设置FrontController的内容，并返回。创建如下代码清单所示的控制器类。

*Listing 14-6. A Default Controller*

*> pico –w application/controllers/index.php*

*<?php*

*class index implements IController {*

*public function index() {*

*$view = new View();*

*$view->name = "Kevin";*

*$result = $view->render('../views/index.php');*

*$fc = FrontController::getInstance();*

*$fc->setBody($result);*

*}*

*}*

1. 视图

视图模型是MVC框架中比较复杂的类之一。它的作用是为变量创建作用域，并且包含了返回解析结果的文件。

通过在方法内部包含输出结果，从而限制那个方法的作用域，并使用PHP的输出缓存特性，就可以实现这个功能。实际的属性支持是有SPL类ArrayObject提供的，它使用的是起始为空的数组和ARRAY\_AS\_PROPS特殊标记，这样便允许轻松地创建属性重载对象了。

创建如下代码清单所示的视图模型。

说明 这个类被称为View，它是一个模型。它提供了视图输出的功能。“视图”实际上是之后将要定义的模板，模板文件将会放在views目录中。

*Listing 14-7. The View Model Class*

*> pico –w application/models/view.php*

*<?php*

*class View extends ArrayObject {*

*public function \_\_construct() {*

*parent::\_\_construct(array(), ArrayObject::ARRAY\_AS\_PROPS);*

*}*

*public function render($file) {*

*ob\_start();*

*include(dirname(\_\_FILE\_\_) . '/' . $file);*

*return ob\_get\_clean();*

*}*

*}*

下一步，需要创建一个默认的视图模板，如下代码清单所示。

*Listing 14-8. The Default View*

*> pico application/views/index.php*

*Hello, <?php echo $this->name; ?>!*

重新加载浏览器，你将看到“Hello, Kevin!”字样。

1. URL参数

你可能不叫Kevin，所以让我们将你的名字与这个脚本集成在一起。由于类封装的关系，要实现这一点，需要修改控制器类。

将application/controller/index.php文件的内容替换为如下代码清单内容。

*Listing 14-9. Using URL Parameters*

*> pico –w application/controllers/index.php*

*<?php*

*class index implements IController {*

*public function index() {*

*$fc = FrontController::getInstance();*

*$params = $fc->getParams();*

*$view = new View();*

*$view->name = $params['name'];*

*$result = $view->render('../views/index.php');*

*$fc->setBody($result);*

*}*

*}*

现在，将URL修改为*http://youdomain.com/index/index/name/youname=*，将会看到“Hello, yourname!”字样。

你可以看到，创建一个MVC框架实际上非常简单。不过，这个示例的框架太简单，以至于几乎没有提供任何功能。

市场上的框架具有大量的内置特性，例如数据库抽象、数据过滤和验证等。

1. **小结**

MVC模式是一种设计模式，它将应用程序分隔为三个部分：模型、视图和控制器。MVC的作用主要在于为Web应用程序开发工作流提供帮助，通过定义特定的角色，使得团队成员更加高效地一起工作。这些角色是开发人员、设计师和集成人员。开发人员负责实现模型，设计师负责设计站点的外观，集成人员负责将它们集成在一起。

大多数MVC框架遵循相似的概念，它们都提供了控制器类和用于实现操作的方法。视图是模板，可以使用普通PHP代码来创建，也可以使用如Smarty这样的模板语言来创建。

选择适合项目的框架时，需要考虑许多因素，其中五个关键的因素是：架构、文档、社区、支持和灵活性。