

梦之都 Codeigniter 教程

*****梦之都*****

- 1.梦之都致力于优秀绿色的教程供应，希望能够给您前进的道路上一点帮助。
- 2.官网地址:www.mengzhidu.com，即"梦之都"的全拼。

*****与君共勉*****

- 1.学无止境，追求技术的道路上，我们一同前进。
- 2.当学习成为一种习惯，进步就是一种必然。
- 3.传播知识，传递温情。我心永恒，始终如一。

*****提示信息*****

- 1.本教程发布作者:辛星。
- 2.本教程发布时间:2016 年 12 月。
- 3.更新版本可以去官网查找。

*****更多交流*****

- 1.PHP 技术交流群:[459233896](https://t.me/459233896)。
- 2.官方网站:www.mengzhidu.com。
- 3.我是辛星，我在这里等你。

***** 目录 *****

第零节:写在前面.....	3
第一节:ci 简介.....	4
第二节:控制器.....	6
第三节:参数传递.....	10
第四节:视图初步.....	12
第五节:视图进阶.....	14
第六节:视图缓存.....	18
第七节:数据库初步.....	20
第八节:数据库进阶.....	25
第九节:增删改查.....	27
第十节:模型.....	31
第十一节:MVC.....	34
附录:致敬读者.....	35

第零节:写在前面

*****浅谈框架*****

说到 PHP 的框架,可能对于刚接触 PHP 的朋友们来说是有有点恐怖的事情,觉得有许多东西需要学,觉得它好难好难。同时也会碰到一大堆的新名词,比如"MVC"、"缓存"、"拦截器"、"日志"等等很多概念。

其实对任何人来说,学习第一个框架的成本通常都很高,所以,如果朋友们是第一次学习 PHP 的框架的话,别怕麻烦,多敲敲代码。

对于有工作经验的朋友们来说,或者说用过一些框架的朋友们,学一个新框架的代价就小很多了,整体流程基本都会很熟悉,所需要了解的只是在当前框架下的实现机制。

而对于 ci 这种小巧的框架来说,掌握它的源代码也是一件非常容易的事情,因为它实在是太过于小巧了,也正式这样,导致很多人特别喜欢它。

*****版本说明*****

截至今天,ci 的最新版本已经发布了其 4.x 系列的,不过由于还是处于开发测试阶段,因此很多正式的项目中还没有用它,还是使用更加老一些的 3.x 版本。对于 3.x 版本来说,本站有对其主要代码的解读,它的核心实现也比较简单,读者朋友们在读完该专题教程后完全可以自己去研究一下它的源码实现。

*****资源列表*****

CI 官方网站:[点此打开](#)

CI 中国社区:[点此打开](#)

CI 的 github 地址:[点此打开](#)

第一节:ci 简介

*****历史简介*****

这个框架的全名叫做"codeigniter", code 翻译成汉语即"代码", "igniter"翻译成汉语即"点火器", 所以ci的logo是一个小火苗。

说到ci的历史,它源自于2006年,值得纪念的是2006年2月28日,此时诞生了ci的1.0测试版。截止到2009年9月11日,此时的1.7.2是最后一个1.x系列的版本,在此之后ci开始进入了2.0时代。

在2014年6月5日发布了2.2.0版,而在2014年辛星codeigniter教程就是以改版本为蓝本进行介绍的。

在2015年3月30日发布了3.0.0版本,也就是ci进入了3.x时代,但是它相对于2.x的兼容性还是很好的,而且官方正在开发4.x版本,这次重写的革新的地方很多,在之后本站也会提供4.x系列的教程。

本教程仍然是以3.0为蓝本进行介绍的,它和2.x系列的差别没有那么大,因此对于那些还在使用2.x系列的朋友们来说也非常友好。

*****下载安装*****

我们可以在ci中国的下载站去下载它,可以点击[这里](#)进入下载页面,下面是下载页面的截图:



我们对下载得到的文件进行解压缩,应该会看到如下的界面:

Windows8_OS (C:) > wamp > www > ci				
<input type="checkbox"/> 名称	修改日期	类型	大小	
application	2016/3/22 0:26	文件夹		
system	2016/3/22 0:26	文件夹		
user_guide	2016/3/22 0:26	文件夹		
.gitignore	2016/3/22 0:26	文本文档	1 KB	
composer.json	2016/3/22 0:26	JSON 文件	1 KB	
contributing.md	2016/3/22 0:26	Markdown 文件	7 KB	
index.php	2016/3/22 0:26	PHP 文件	11 KB	
license.txt	2016/3/22 0:26	TXT 文件	2 KB	
readme.rst	2016/3/22 0:26	RST 文件	3 KB	

对于上面图片的目录，我们这里进行一下介绍：

(1) **application** 是我们应用的目录，也是我们要编写的应用的目录

(2) **system** 是 Codeigniter 框架代码的目录

(3) **user_guide** 是用户手册

(4) **.gitignore** 是 git 的忽略文件

(5) **composer.json** 是 composer 依赖管理的相关文件

(6) **contributing.md** 是 markdown 格式的说明文件

(7) **index.php** 是入口文件

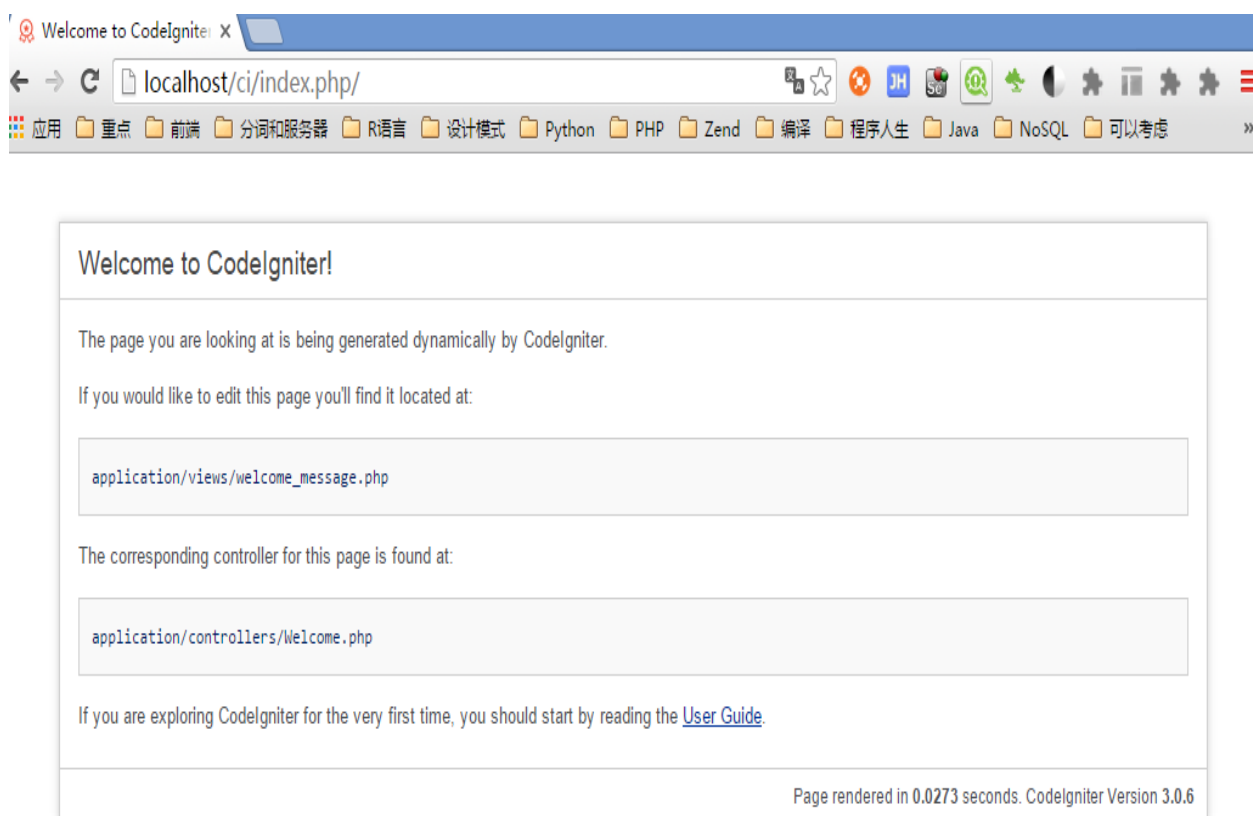
(8) **license.txt** 是版权声明文件

(9) **readme.rst** 是 rst 格式的说明文件

上面的 system 目录中的就是 Codeigniter 官方写好的目录，它定义了整个框架的执行流程和一些常用的类库，这个目录我们一般不需要去改动，而 application 目录则是我们去编写我们程序的目录。

*****开始运行*****

相信朋友们已经搭建了可以让 PHP 运行的环境了，我们把该目录放到相应的位置，然后我们访问这个入口文件，即 index.php。如果我们环境配置正确，我们将会看到如下的界面：



应该说，这个界面还是蛮清爽的。到目前为止，我们的环境就已经搭建完成了，让我们开始接下来的学习吧。

第二节:控制器

*****理论部分*****

对于一个web应用来说,大多数我们的操作都可以通过改变url来实现。因此,基本上所有的框架都对url有着很强的控制能力,主要包括对url的解析、对url的构造、对url的过滤。

为了加强对url的控制,我们通常需要一个统一的入口,也就是我们的index.php文件,不管我们最后要实现什么功能,我们都是从访问这个文件开始的。















为了实现统一入口,我们可以在改文件中定义一个常量,ci的做法是定义了一个BASEPATH常量。如果我们不通过这个入口去访问其他文件的话,其他的文件会首先检测是否定义了BASEPATH这个常量,如果没有定义,则直接退出,如果有定义的话,则说明是从同一入口进入的。

因此我们会在很多其他文件中看到这么一段代码:

```
defined('BASEPATH') OR exit('No direct script access allowed');
```

这样就能够保证我们的代码是从同一入口进入的。

我们进入application目录,然后我们可以看到如下截图:

我的电脑 > Windows8_OS (C:) > wamp > www > ci > application >				
<input type="checkbox"/> 名称	修改日期	类型	大小	
 cache	2016/3/22 0:26	文件夹		
 config	2016/3/22 0:26	文件夹		
 controllers	2016/3/22 0:26	文件夹		
 core	2016/3/22 0:26	文件夹		
 helpers	2016/3/22 0:26	文件夹		
 hooks	2016/3/22 0:26	文件夹		
 language	2016/3/22 0:26	文件夹		
 libraries	2016/3/22 0:26	文件夹		
 logs	2016/7/28 15:09	文件夹		
 models	2016/3/22 0:26	文件夹		
 third_party	2016/3/22 0:26	文件夹		
 views	2016/3/22 0:26	文件夹		
 .htaccess	2016/3/22 0:26	HTACCESS 文件	1 KB	
 index.html	2016/3/22 0:26	HTML 文件	1 KB	

这里介绍一下上述目录,如下:

- (1)cache 目录用来存放缓存内容
- (2)config 目录用来存放配置文件
- (3)controllers 用来存放控制器
- (4)core 用来存放我们自己编写的类库
- (5)helpers 用来存放助手函数
- (6)hooks 用来存放我们自己编写的钩子

(7)language 用来存放我们自己定义的语言包

(8)libraries 用来存放我们自己的类库

(9)logs 用来保存日志文件

(10)models 用来保存模型

(11)third_party 用来存放第三方库

(12)views 用来存放视图文件



(13).htaccess 用来给 Apache 使用来进行控制

(14)index.html 是用来防止目录遍历的文件

本节我们主要介绍控制器，即 controllers 目录中的内容。

*****控制器*****

这里我们进入 controllers 目录，我们可以看到已经有一个编写好的控制器了，截图如下：

电脑 > Windows8_OS (C:) > wamp > www > ci > application > controllers			
<input type="checkbox"/> 名称	修改日期	类型	大小
 index.html	2016/3/22 0:26	HTML 文件	1 KB
 Welcome.php	2016/3/22 0:26	PHP 文件	1 KB

我们点击进入这个 Welcome.php 文件，我们看到如下代码：

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Welcome extends CI_Controller {

    /**
     * Index Page for this controller.
     *
     * Maps to the following URL
     *      http://example.com/index.php/welcome
     *  - or -
     *      http://example.com/index.php/welcome/index
     *  - or -
     * Since this controller is set as the default controller in
     * config/routes.php, it's displayed at http://example.com/
     *
     * So any other public methods not prefixed with an underscore will
     * map to /index.php/welcome/<method_name>
```

```

* @see https://codeigniter.com/user_guide/general/urls.html
*/
public function index()
{
    $this->load->view('welcome_message');
}
}

```

我们可以看到这个Welcome类有一个index()方法，这个Welcome是默认访问的控制器，而index则是默认访问的方法，而这里的\$this->load->view()则是调用了视图，对于视图，我们后面再进行介绍。

我们可以仿照Welcome这个控制器来编写我们的控制器，我们在controllers目录下新建一个Star.php，需要说明的是这个文件名通常是大写的，然后我们在里面写入如下代码：

```

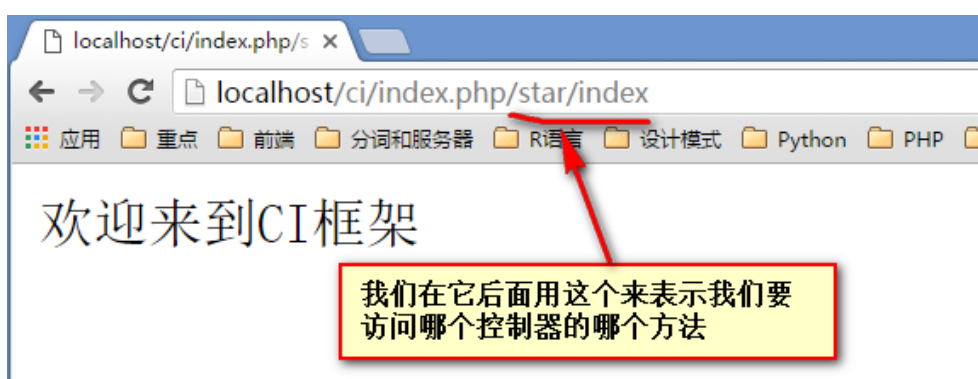
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Star extends CI_Controller {

    public function index()
    {
        echo "欢迎来到CI框架";
    }
}

```

这样，我们的一个控制器就写好了，那么我们如何来访问它呢？我们在index.php后面加上"/star/index"就可以访问到它了，我们这里来个截图吧：



我们上面定义的Star这个类，我们通常称之为控制器，而它的方法index，我们通常称之为动作，也就是说，我们可以通过"localhost/ci/index.php/star/index"来访问star控制器的index动作。，然后我们就看到了我们要输出的内容。

如果我们要访问到这个控制器，我们可以通过我们的 url 来进行访问，我们的格式一般是：**域名/index.php/控制器名/动作名**

我们还可以在 Star 这个控制器中编写其他的方法，然后尝试用 url 去访问一下这个方法。

第三节:参数传递

*****理论部分*****

在我们初学 PHP 的时候，都是在 url 中使用?、&以及=来传递参数的，由于这种传参方式不够美观，因此它在现代框架中已经很少使用了，一般我们使用另一种更加美观整齐的方式，我们一般称之为"pathinfo 模式"。

所谓 pathinfo 模式，就是可以通过一种特定的分隔符来分隔 url 中的各个段，一般我们使用"/"。它可以让 url 更加美观，而且对 SEO 更加友好，也就是所谓的"伪静态"。

比如我们的 localhost/ci/index.php/star/index 就比传统的 localhost/ci/index.php?c=star&a=index 要美观一些。

那我们要向 url 中传递参数的时候，我们通过 pathinfo 格式的写法，其格式如下：

xxx/控制器名/动作名/参数值 1/参数值 2/参数值 3/...

而我们的动作在接收这些参数的时候就可以这么写了：

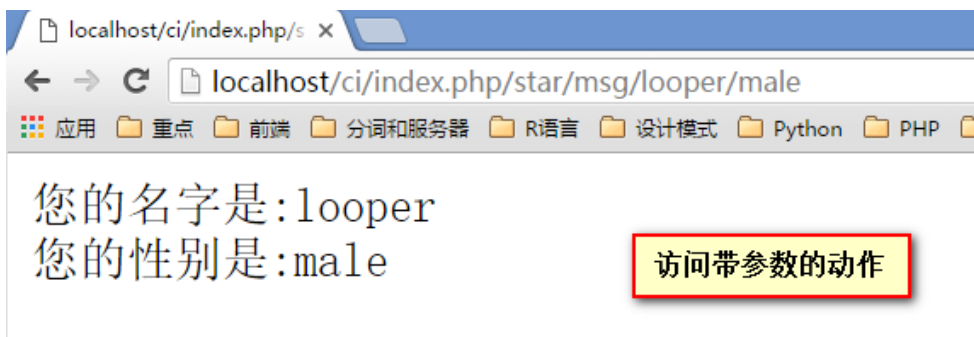
```
public function 动作名($var1,$var2....)
{
    //该动作的具体实现细节
}
```

*****代码实战*****

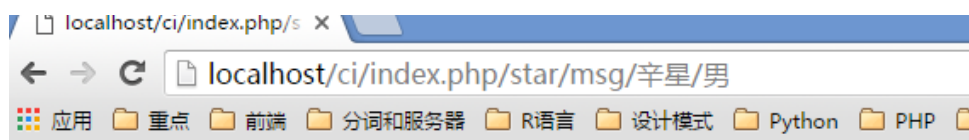
我们就来写一个真实的范例吧，我们在我们的 Star 控制器中添加如下的动作：

```
public function msg($name,$gender)
{
    echo "您的名字是:".$name;
    echo "<br/>";
    echo "您的性别是:".$gender;
}
```

那么当我们调用它的时候，我们就可以使用 looper 作为第一个参数，male 作为第二个参数来访问这个动作，截图如下：



但是如果我们用汉字去访问的话，则可能会出现乱码的情形，首先我们来个截图吧：



您的名字是:%E8%BE%9B%E6%98%9F
您的性别是:%E7%94%B7

那么这种情况我们该如何处理呢？这就需要我们用 `urldecode()` 来对密码进行解密了，这里我们的代码只需要这么写就可以了：

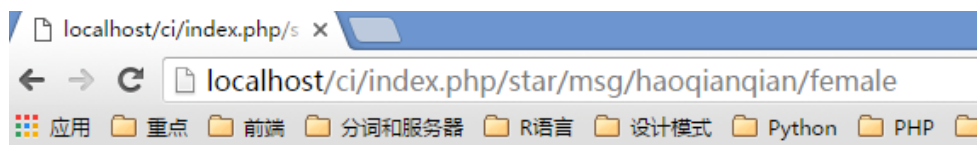
```
public function msg($name,$gender)
{
    echo "您的名字是:".urldecode($name);
    echo "<br/>";
    echo "您的性别是:".urldecode($gender);
}
```

然后我们再来看一下，这次就没问题了，截图如下：



您的名字是:辛星
您的性别是:男

当然对于英文，其实是否解码都不会乱码，这里来个截图吧：



您的名字是:haoqianqian
您的性别是:female

第四节:视图初步

*****模板*****

在很多框架中都是支持模板的，有的使用内置模板，有的则使用通用模板，比如 smarty 或者 blade 等等。但是 ci 是不支持模板的，其实理由也很简单，PHP 语法本身就不复杂，那么为什么不直接写 PHP 代码呢。

由于 ci 不支持模板，那么我们的视图文件就直接写 php 代码就可以了。我们的视图文件是放在 application 目录下的 views 目录下的，我们来个截图吧：

Windows8_OS (C:) > wamp > www > ci > application > views				
名称	修改日期	类型	大小	
errors	2016/3/22 0:26	文件夹		
index.html	2016/3/22 0:26	HTML 文件	1 KB	
welcome_message.php	2016/3/22 0:26	PHP 文件	3 KB	

这个 errors 目录中存放的都是当发生异常时的展示页面，如果是通过浏览器访问，则使用 html 子目录里的文件，如果是通过命令行访问，则使用 cli 子目录里的文件。

我们可以在控制器中通过如下格式来加载一个视图：

```
$this->load->view(视图路径);
```

比如说我们在 views 目录中创建一个 my.php 文件，我们在里面写入如下代码：

```
<?php defined('BASEPATH') OR exit('No direct script access allowed'); ?>
<p style="color:red;text-align:center">
    欢迎使用视图
</p>
```

然后我们可以在 Star 控制器中新建一个方法，然后我们可以加载视图，我们的代码范例如下：

```
public function view()
{
    $this->load->view('my');
}
```

然后我们访问它的时候，我们会看到如下的界面：



我们成功的加载了第一个视图文件。需要说明的是，我们这里使用的是相对路径，而且由于文件名的后缀默认为".php"，因此如果我们的文件后缀为".php"的话，我们就可以省略不写。

需要说明的是，我们可以同时加载多个视图文件，我们加载多个视图文件的格式可以是这样：

```
$this->load->view(视图1)->view(视图2)->..->view(视图n);
```

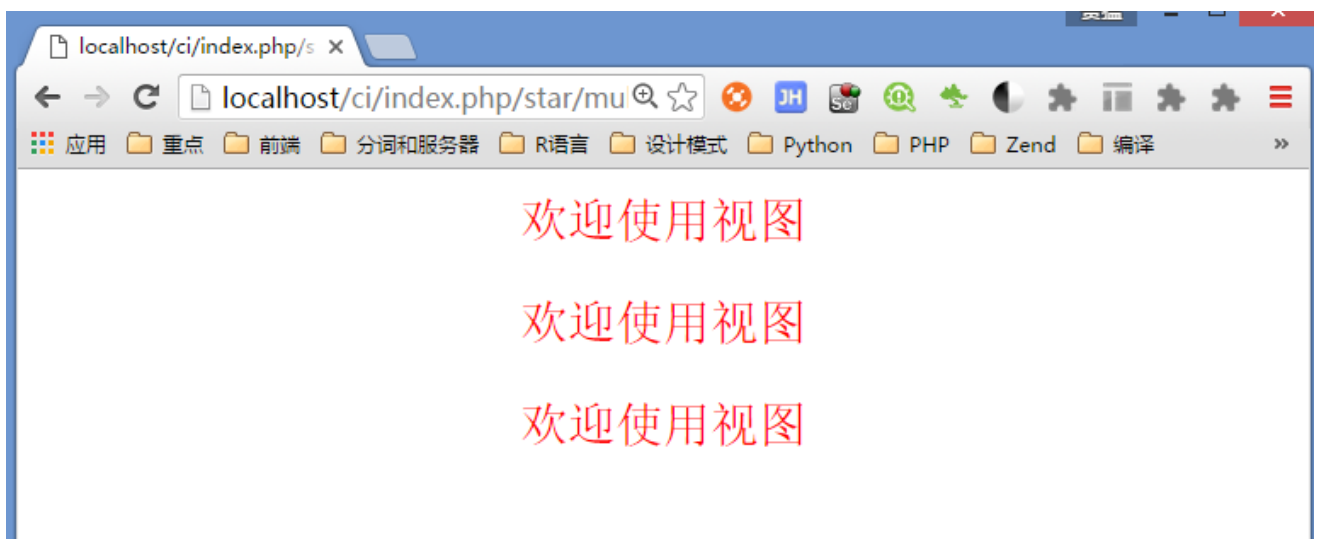
我们也可以使用这样的方式来进行加载多个视图：

```
$this->load->view(视图1);  
$this->load->view(视图2);  
...  
$this->load->view(视图n);
```

下面我们就来看一个范例吧，假设我们在 Star 控制器中添加了这样的一个方法 multi:

```
public function multi()  
{  
    $this->load->view('my')->view('my')->view('my');  
}
```

这里我们加载了三次 my.php 这个视图文件，然后我们来看一下效果吧：



有些时候，我们会把一个页面切成几部分，比如分为头部、中部、尾部，而且通常头部和尾部会被多个控制器的方法加载，这个时候我们通常都是需要用到多视图的加载的。

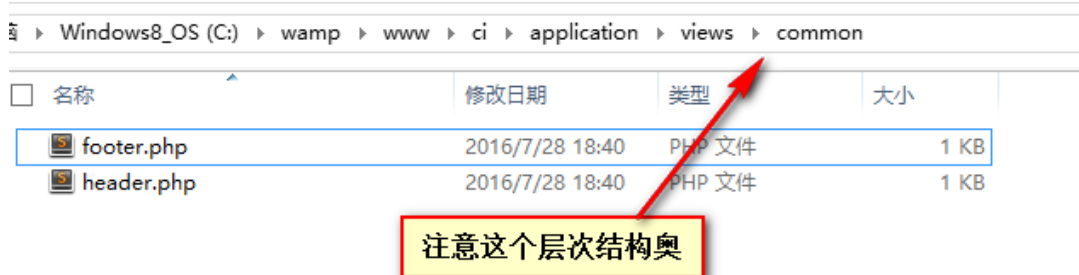
第五节:视图进阶

*****目录层级*****

这里需要说明的是, 当我们的视图文件比较多的时候, 我们就可以通过目录来进行划分, 这样可以避免同一个目录下堆放太多的文件, 而且也让层次结构更加清晰。

比如我们在 views 下的层次结构为"目录 1/目录 2/文件名", 那么我们在控制器调用的时候就可以这么用:`$this->load->view("层次 1/层次 2/文件名");`

这里我们来看一个例子吧, 这里我们的 views 下有一个 common 目录, 里面有两个文件, 一个是 footer.php, 一个是 header.php, 这里来一个截图吧:



首先是 header.php 文件的内容, 其代码如下:

```
<h2>这是头部</h2>
```

然后是 footer.php 文件的内容, 其代码如下:

```
<h2>这是尾部</h2>
```

然后我们在我们的 Star 控制器中新建一个 path() 方法, 我们写下如下代码:

```
public function path()
{
    $this->load->view('common/header')->view('my')->view('common/footer');
}
```

然后我们可以看到如下效果图, 可以看到, 我们的头部和尾部都被加载了:



一般来说, 我们很少只需要一个视图文件, 一般都是需要多个视图文件同时参与的。

*****传递数据*****

在很多时候，我们是需要向模板中传递数据的，这个时候我们就可以在 `view()` 中的第二个参数设置为携带了数据的数组，一般我们的格式如下：

`$this->load->view($path,$data);`

其中 `$data` 就可以是一个携带数据的数组。我们来看一个具体的例子吧，我们的视图为 `views` 目录下的 `data.php`，其代码如下：

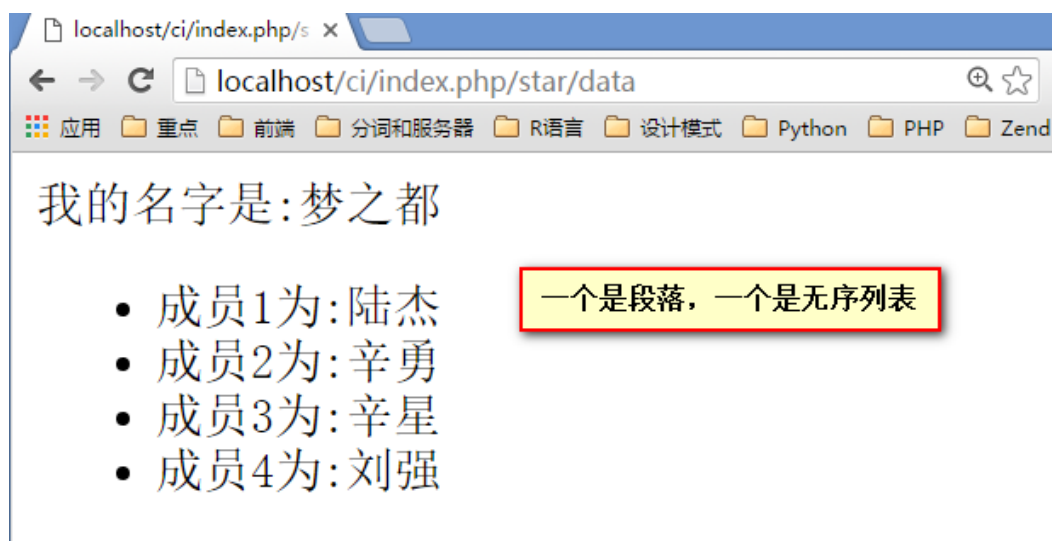
```
<p>我的名字是:<?=$name;?></p>
<ul>
<?php foreach($users as $key=>$val){ ?>
    <li>成员<?=$key+1;?>为:<?=$val;?></li>
<?php } ?>
</ul>
```

这里我们发现，我们的视图中需要两个变量，一个是 `$name`，一个是 `$users`，这里我们可以在控制器中在调用这个视图的时候把值赋给它们。这里我们在 `Star` 控制器中编写一个 `data()` 方法，其代码内容如下：

```
public function data()
{
    $data = array();
    $data['name'] = "梦之都";
    $data['users'] = array("陆杰","辛勇","辛星","刘强");

    $this->load->view('data',$data);
}
```

然后我们来访问一下这个方法，我们就会看到如下的效果图：



我们在传递数据中的\$data，它的下标 name 对应的数据会传递给视图中的\$name，它的下标 users 对应的数据会传递给视图中的\$users。

需要说明的是，传递数据是非常重要的一个内容，因为我们通常必须把数据传递到视图，并且展示出来才有意义。

我们除了可以在加载视图的时候对视图中的变量进行赋值，我们还可以使用\$this->load->vars()中进行赋值，需要说明的是，此时的赋值是对当前方法加载的所有视图都有效的。

我们在\$this->load->vars()中有两种调用方式，第一种是传递一个关联数组，此时它会把键作为变量，把该键对应的值赋予它，第二种就是传递两个参数，第一个参数是键，第二个参数是值。

这里我们来看第一个动作吧，它的方法名为 assign1，其代码如下：

```
public function assign1()
{
    $var = array();
    $var['name'] = "鹰眼";
    $var['users'] = array("青龙","白虎","朱雀");

    //这里传递了一个关联数组
    $this->load->vars($var);

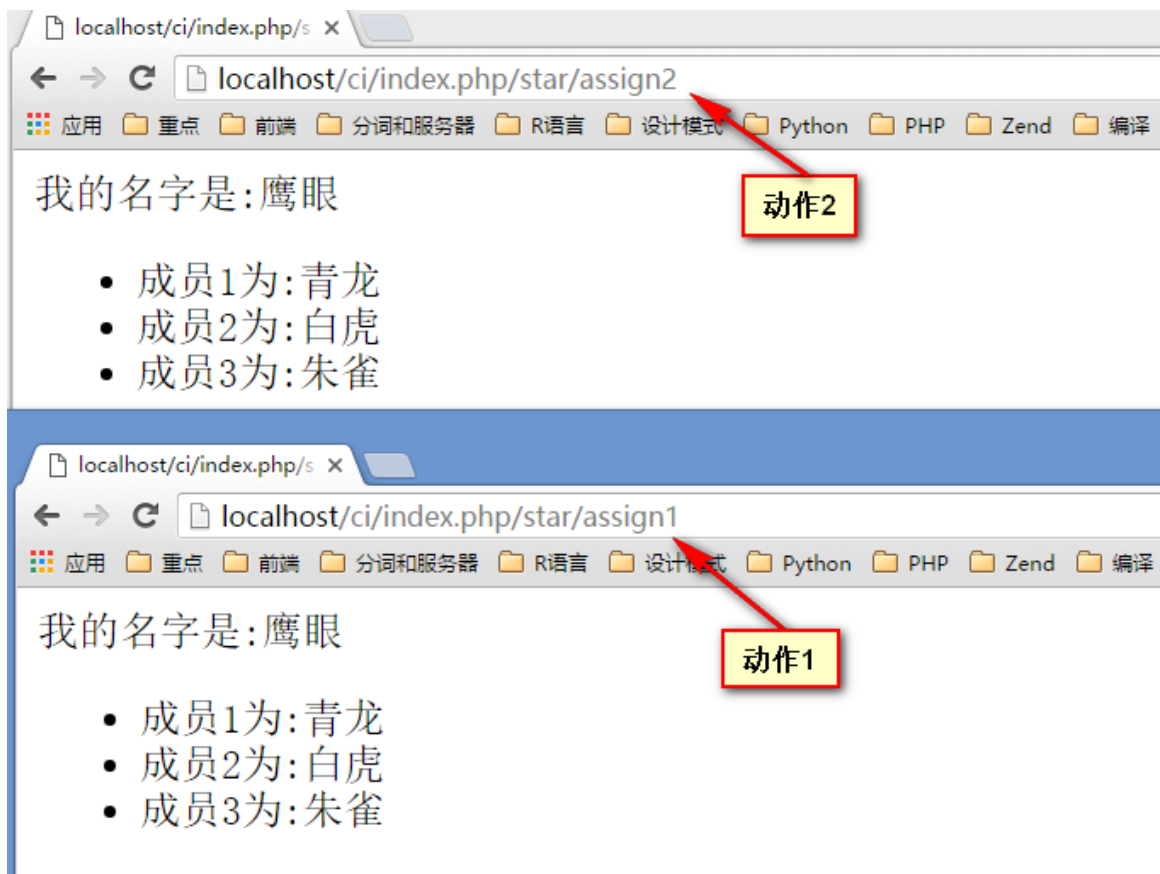
    $this->load->view('data');
}
```

第二个动作的方法名为 assign2，其代码如下：

```
public function assign2()
{
    //这里第一个参数是键，第二个是值
    $this->load->vars("name","鹰眼");
    //这里第一个参数是键，第二个是值
    $this->load->vars("users",array("青龙","白虎","朱雀"));

    $this->load->view('data');
}
```

其实上面两种方式的效果都是一样的，它们传递的数据也都相同，因此这里就一次性进行截图展示了，如下：



对于如何向视图中传递数据，我们就介绍到这里啦。

第六节:视图缓存

*****缓存简介*****

要提升网站的响应效率,缓存是必须要过的一关,它对于大数据量的访问上能够更好的应对,缓存的实现形式有很多,我们这里介绍的只是对页面的缓存。

我们可以对每个独立的页面进行缓存,并且我们可以设置每个页面的缓存时间。当页面首次被加载的时候,缓存将会被写入到 application/cache 目录下的文件中。

当我们之后再次去请求这个页面的时候,就可以直接从缓存文件中读取内容,因为是静态的 html 文件,因此加载速度会快很多。

而且缓存的默认的存放目录位于 application 目录下的 cache 目录中,我们来对该目录来个截图吧:

Windows8_OS (C:) > wamp > www > ci > application > cache				
名称	修改日期	类型	大小	
.htaccess	2016/3/22 0:26	HTACCESS 文件	1 KB	
index.html	2016/3/22 0:26	HTML 文件	1 KB	

如果我们要在当前动作启用缓存的话,我们只需要在动作中使用如下代码即可:

```
$this->output->cache(分钟数);
```

比如我们要在某个动作上进行缓存 50 分钟,这里我们给一个范例:

```
public function cache()
{
    $this->output->cache(50);
    $this->load->view('common/header')->view('my')->view('common/footer');
}
```

那么我们的 cache 目录下就会多一个缓存文件,这个名称是跟我们当前的 url 有关,我们来看一下截图吧:

Windows8_OS (C:) > wamp > www > ci > application > cache				
名称	修改日期	类型	大小	
.htaccess	2016/3/22 0:26	HTACCESS 文件	1 KB	
0618ef220a90e1eae831611a59309567	2016/7/29 10:49	文件	1 KB	
index.html	2016/3/22 0:26	HTML 文件	1 KB	

多了一个缓存文件

然后我们可以看一下缓存生成的是什么:

```
a:2:{s:6:"expire";i:1469763580;s:7:"headers";a:0:{}}ENDCI---<h2>这是头部</h2><p style="color:red;text-align:center">
```

欢迎使用视图

</p>

<h2>这是尾部</h2>

我们知道前面的部分是一个序列化的数据，其中 expire 中表示的是缓存到期的时间戳，比如我们现在的机制是缓存 50 分钟，那么这个数字就是当前的时间戳加上 3000 秒之后得到的时间。而它后面，则是具体的整个输出的内容，需要说明的是，这里它是缓存的整个的输出，而不是每个视图做一个缓存。

需要说明的是，我们这里使用 `$this->output` 得到的是一个 `CI_Output` 的对象，它的代码在 `system` 目录下的 `core` 目录下的 `Output.php` 文件中，我们可以调用它的很多方法，比如我们可以使用 `$this->output->delete_cache()` 来删除缓存文件等等。

需要说明的是，合理的使用缓存对于响应速度的提升是很明显的，但是缓存也会带来一些问题，这个就需要我们结合具体的开发场景来进行相应的处理。

第七节:数据库初步

*****数据库配置*****

我想数据库的重要性也无需我多言了, 对于 web 开发而言, 基本都是围绕着数据进行的, 由于目前大多数项目可能使用的依然是 MySQL, 因此我们这里也是以 MySQL 为例进行说明。

首先我们需要进行一下配置, 这里需要修改的文件为 application 目录下的 config 目录下的 database.php 文件, 我们修改相应的配置项, 比如这里是我的配置情况:

```
$db['default'] = array(
    'dsn' => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => 'xin',
    'database' => 'ci',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'encrypt' => FALSE,
    'compress' => FALSE,
    'stricton' => FALSE,
    'failover' => array(),
    'save_queries' => TRUE
);
```

通常我们在操作数据库之前, 首先需要进行数据库的加载, 通常代码如下:

```
$this->load->database();
```

在连接之后, 我们可以使用 `$this->db` 来获取当前的数据库对象, 然后我们就可以对它进行数据库的操作了。

需要特殊说明的是, 对于查询操作, 一般我们得到的都是一个结果集, 我们还需要调用一下 `result()` 这个方法来获取结果集。

*****查询范例*****

我们来做一次数据的查询吧, 这里我们要获取 user 表的数据, 我们可以创建一个 Db 控制器, 然后写一个 demo 方法, 整个文件的代码如下:

```

<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Db extends CI_Controller {

    public function demo()
    {
        $this->load->database();


        $sql = "select * from user";

        $data = $this->db
                ->query($sql)
                ->result('array');

        var_dump($data);
    }
}

```

需要说明的是，当前数据库中是有 user 表的，那么我们就可以看到它数据取出后的结果了：



```

array (size=3)
  0 =>
    array (size=3)
      'id' => string '1' (length=1)
      'name' => string '陆杰' (length=6)
      'passwd' => string '111' (length=3)
  1 =>
    array (size=3)
      'id' => string '2' (length=1)
      'name' => string '辛星' (length=6)
      'passwd' => string '222' (length=3)
  2 =>
    array (size=3)
      'id' => string '3' (length=1)
      'name' => string '倩倩' (length=6)
      'passwd' => string '333' (length=3)

```

我们在上述代码中的 query() 用于进行查询操作，而 result() 则是从结果集中获取对应的数据，需要说明的是，它传递的参数 'array' 则是说明了我们要数组格式的数据。如果我们不传递参数，则默认是获取的一个个的对象。

如果我们在使用 result() 的时候使用默认的方式，也就是说如果我们的代码这么写：

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Db extends CI_Controller {

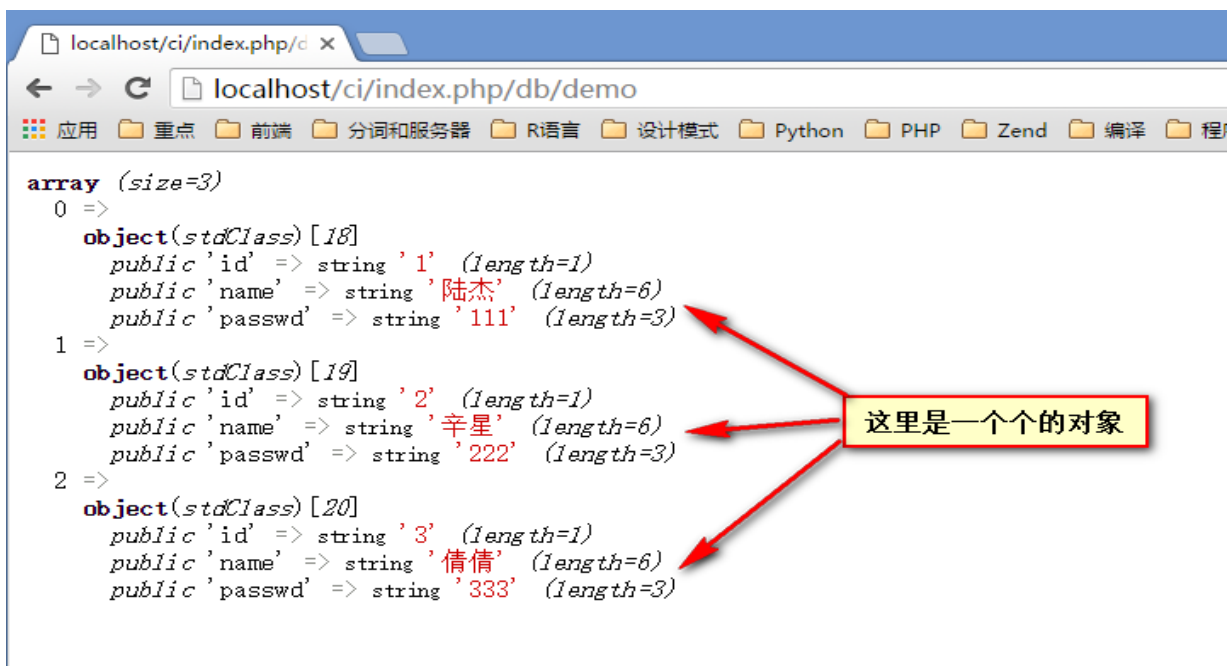
    public function demo()
    {
        $this->load->database();

        $sql = "select * from user";

        $data = $this->db
            ->query($sql)
            ->result();

        var_dump($data);
    }
}
```

那么我们这里看到每条记录都会变成对象的形式，代码效果如下：



The screenshot shows a web browser window with the URL `localhost/ci/index.php/db/demo`. The output is a PHP var_dump of an array containing three objects. Each object is a `stdClass` with properties `id`, `name`, and `passwd`. Red arrows point from a yellow box labeled "这里是一个个的对象" to each of the three objects in the array.

```
array (size=3)
  0 =>
    object(stdClass) [18]
      public 'id' => string '1' (length=1)
      public 'name' => string '陆杰' (length=6)
      public 'passwd' => string '111' (length=3)
  1 =>
    object(stdClass) [19]
      public 'id' => string '2' (length=1)
      public 'name' => string '辛星' (length=6)
      public 'passwd' => string '222' (length=3)
  2 =>
    object(stdClass) [20]
      public 'id' => string '3' (length=1)
      public 'name' => string '倩倩' (length=6)
      public 'passwd' => string '333' (length=3)
```

可能从其他框架转过来的朋友们会不习惯每次查询后都需要写一个result()来获取结果集，但是不得不承认的是这种方式也有不少优点的，而且它本身也有比较丰富的方法可供我们调用：

(1)num_rows()用来获取总的行数。

(2)list_fields()用来获取所有的字段数。

(3)result_array()用数组的方式来获取数据。

(4)result_object()用对象的方式来获取数据。

(5)row()用来获取一行数据。

(6)next_row()用来获取下一行数据。

(7)first_row()用来获取第一行数据。

(8)previous_row()用来获取上一行数据。

比如我们这里可以来使用一下，我们在 Db 控制器中新建一个result()方法，然后书写如下代码：

```
public function result()
{
    $this->load->database();

    $sql = "select * from user";

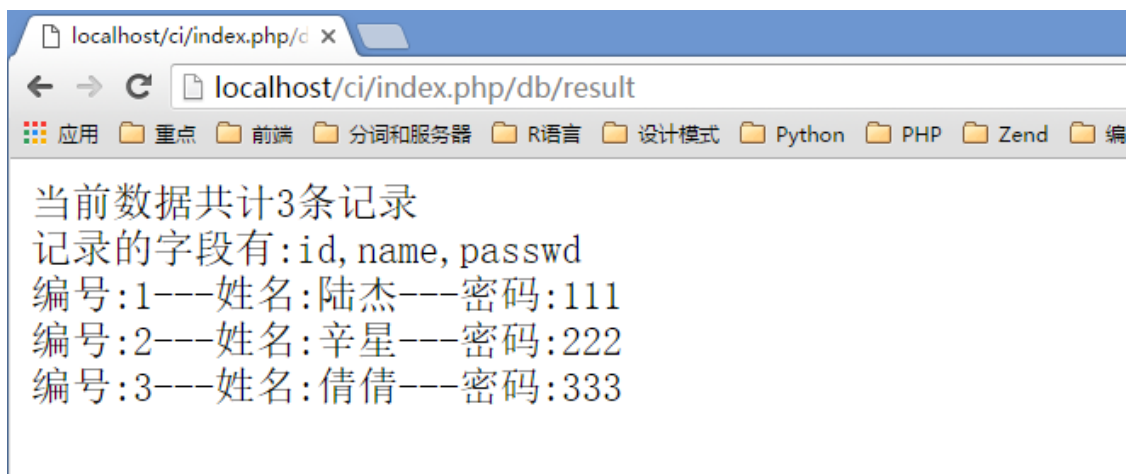
    $re = $this->db->query($sql);

    //获取行数
    echo "当前数据共计".$re->num_rows(). "条记录"<br/>;

    //获取都有哪些字段信息
    echo "记录的字段有:".implode(", ", $re->list_fields())."<br/>;

    //获取具体信息
    foreach($re->result() as $key=>$val){
        echo "编号:".$val->id."---";
        echo "姓名:".$val->name."---";
        echo "密码:".$val->passwd."<br/>";
    }
}
```

在上面的代码中，我们直接执行了一条SQL语句，然后我们用一个循环的方式来输出每条记录的内容，需要说明的是，这里的每条记录默认返回的格式是对象格式。然后我们就会看到具体的输出如下：



对于我们使用 SQL 语句进行查询操作的情形，我们就介绍到这里啦。

第八节:数据库进阶

*****SQL 注入*****

对于商业应用来说，安全和性能是绝对不能忽视的两个要点，说到安全，需要注意的地方简直太多了，即使是对于 MySQL 本身，也是有很多值得注意的地方，我们这里不讨论这么大的话题，我们仅仅局限在防 SQL 注入这个问题上。

对于防止 SQL 注入方面，使用预编译和绑定的机制无疑是很重要的，在 ci 中，我们也可以在 sql 中使用问号作为占位符，然后传递需要绑定的数据数组就可以了。

我们还是来看一个范例吧，我们的在 Db 控制器中新建一个 bind() 方法，代码如下：

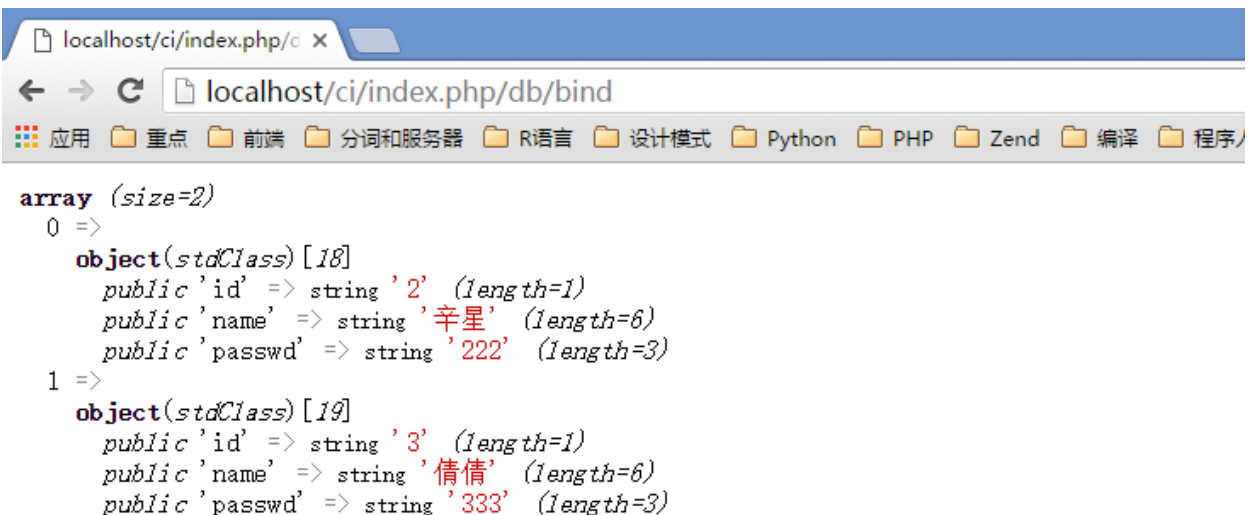
```
public function bind()
{
    $this->load->database();

    $sql = "select * from user where id > ? and id < ?";
    $param = array(1,4);

    $data = $this->db->query($sql,$param)->result();

    var_dump($data);
}
```

在上述代码中，我们的 \$sql 是使用问号作为占位符的 sql 语句，而 \$param 则是会绑定的数据，其中 1 对应第一个问号，4 对应第二个问号，我们来看一下运行效果吧，如下：



```
array (size=2)
  0 =>
    object(stdClass) [18]
      public 'id' => string '2' (length=1)
      public 'name' => string '辛星' (length=6)
      public 'passwd' => string '222' (length=3)
  1 =>
    object(stdClass) [19]
      public 'id' => string '3' (length=1)
      public 'name' => string '倩倩' (length=6)
      public 'passwd' => string '333' (length=3)
```

这种写法还是挺常规的，它主要用于我们手写 sql 语句的时候，这样会增强安全性，需要说明的是，如果我们用 ci 提供的方法的话，则 ci 已经自动帮我们

实现了预编译和绑定了。当然需要说的是，像 mysql 这种扩展是无法支持的，因此我们建议使用 pdo 扩展。

*****事务*****

对于事务的重要性，我想没有人会怀疑它的意义，而 ci 也对事务提供了不错的支持，其中 ci 对我们常用的事务操作进行了分类，它按照自动提交和手动提交提出了两套执行方案。

第一套方案的基本思路是这样的：

```
$this->db->trans_start(); //开启事务
... //其他数据库操作
$this->db->trans_complete();//结束事务
```

在 trans_start() 和 trans_complete() 之间的操作会被当做一个事务，系统会自动提交或者回滚，就是说如果里面的每一次操作都成功了，那么就会提交事务，如果里面有失败的操作，那么就会自动回滚。

第二套方案的基本思路是这样的：

```
$this->db->trans_begin();//开始一个事务
... //其他操作
if($this->db->trans_status === FALSE){
    $this->db->trans_rollback();//事务回滚
    ... //反馈给前端或日志
}else{
    $this->db->trans_commit();//事务提交
    ... //反馈给前端或日志
}
```

这里我们需要手动使用 trans_commit() 来提交事务或者使用 trans_rollback() 来回滚事务，这种情况下我们的处理更加自由。

其实我们仔细读一下相应的代码，我们会发现其实 trans_start() 在内部是调用了 trans_begin() 的，具体的代码执行机制可以阅读 system 目录中的 database 目录中的 DB_driver.php 文件。

第九节:增删改查

*****概述*****

对于数据库来说，增删改查无疑还是最重要的操作，我们这里就以增删改查为例来介绍一下。

首先是增加数据，我们可以使用insert()方法来插入数据，它的第一个参数是表名，第二个参数是一个数组。我们还是来看一个具体的范例吧，代码范例如下：

```
public function insertOne()
{
    $this->load->database();

    $arr = array();
    $arr['name'] = "耿雪";
    $arr['passwd'] = "xxx";

    $flag = $this->db->insert('user',$arr);

    var_dump($flag);
}
```

在上面，我们向user表中插入了一条数据，我们来看一下界面的反馈情况吧：



我们也可以进行批量插入，这个时候我们可以使用insert_batch()方法，它的第一个参数也是一个表名，第二个参数可以是一个多维数组。代码范例如下：

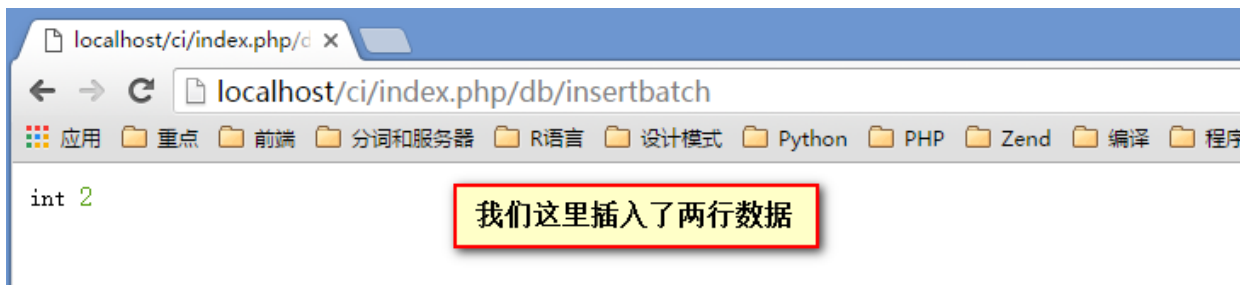
```
public function insertBatch()
{
    $this->load->database();

    $arr = array();
    $arr[] = array('name'=>"小楠","passwd"=>"nnn");
    $arr[] = array('name'=>"李茜","passwd"=>"xxx");

    $num = $this->db->insert_batch('user',$arr);
}
```

```
var_dump($num);  
}
```

这里表示成功插入的行数，我们来看一下效果截图吧：



对于插入数据，一般这也是我们比较常用的两种方式。

*****数据查询*****

对于查询数据来说，则有比较多的方法，下面是几个常用的方法：

(1) **from()** 表示选择的表

(2) **select()** 表示要选择哪些字段，可以用数组表示，也可以用逗号分隔多个字段的字符串，如果为空则表示选择所有字段，相当于填写了 "*"

(3) **distinct()** 表示去除重复的记录

(4) **limit()** 表示要获取多少条记录

(5) **offset()** 表示选择的偏移量

(6) **where()** 表示 where 条件，一般第一个参数是字段名，第二个参数是值，也可以把第一个参数设置为整个条件

(7) **group_by()** 表示按哪些字段进行分组

(8) **order_by()** 表示按哪些字段进行排序，第一个参数为字段名，第二个参数用 'asc' 表示升序，用 'desc' 表示降序

(9) **join()** 表示进行表的连接，第一个参数为连接的表名，第二个参数为连接的条件

(10) **get()** 表示进行获取操作，在它之后通常跟 **result()** 来检索出具体的结果

其实具体的方法还有很多，这里只是列举了比较常用的几个，我们来看一个具体的范例吧，代码范例如下：

```
public function select()  
{  
    $this->load->database();  
  
    $data = $this->db  
        ->from("user")  
        ->where("id > 2")
```

```

        ->limit(3)
        ->select("id,name")
        ->order_by("id","desc")
        ->get()
        ->result();

    var_dump($data);
}

```

然后我们来看一下我们检索的结果吧，截图如下：



```

array (size=3)
  0 =>
    object(stdClass) [18]
      public 'id' => string '6' (length=1)
      public 'name' => string '李茜' (length=6)
  1 =>
    object(stdClass) [19]
      public 'id' => string '5' (length=1)
      public 'name' => string '小楠' (length=6)
  2 =>
    object(stdClass) [20]
      public 'id' => string '4' (length=1)
      public 'name' => string '耿雪' (length=6)

```

不过对于查询来说，它所涉及的方法还是有点太多了，很多功能我们还是需要去查看手册。

*****数据的删除*****

对于数据的删除，我们可以使用 `delete()` 来执行，我们可以用 `from()` 来选择表，用 `where()` 来表示条件，用 `delete()` 来表示最后的删除。我们这里来看一个范例吧：

```

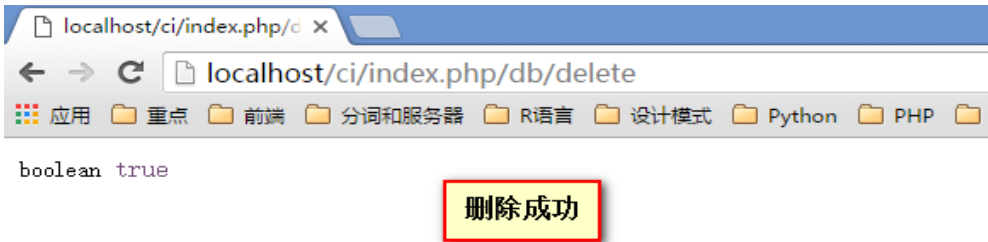
public function delete()
{
    $this->load->database();

    $flag = $this->db
        ->from("user")
        ->where("id > 5")
        ->delete();
}

```

```
var_dump($flag);  
}
```

我们来看一下删除的结果返回吧，截图如下：



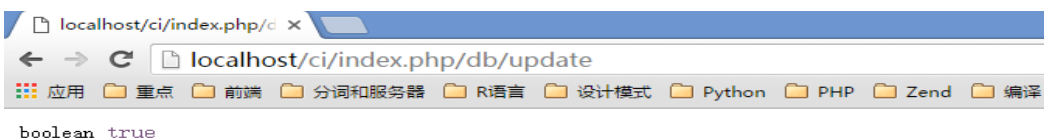
其实ci还提供了注入empty_table()和truncate()等操作，但是在我们的开发中，用的并不是很多，因此这里就略过了。

*****数据修改*****

对于数据的修改，我们可以使用update()来进行修改，我们可以使用from()来选择要更新的表，我们可以用where来表示更新的条件，我们可以用set()表示要更新的数据。我们这里来看一个具体的范例吧：

```
public function update()  
{  
    $this->load->database();  
    $arr = array();  
    $arr['name'] = "star";  
    $arr['passwd'] = 'sss';  
  
    $ret = $this->db  
        ->from("user")  
        ->where('id','2')  
        ->set($arr)  
        ->update();  
  
    var_dump($ret);  
}
```

根据上面的代码，我们可以知道我们把id为2的那条记录的name字段设置为star，我们把passwd字段设置为sss，然后我们来看一下更新的效果吧，这里我们来个界面截图：



第十节:模型

*****模型概述*****

我们上面介绍了一下如何操作数据库，没错，我们可以在控制器中处理他们，也似乎并没有任何问题，但是当我们的业务越做越大的时候，我们就需要专门抽象出来一个 Model 层就很有必要了，也就是我们的模型层。

一个数据模型，基本上和一个表是对应的，但是并不是一一的完全对应，比如我们这里可以创建一个消息的模型，我们的表用 msg 表示，它有三个字段，一个是 id 表示自增的编号，一个 errcode 表示错误码，一个 errstr 表示错误信息，下面是这个表的所有数据截图：

```
mysql> select * from msg;
```

id	errcode	errstr
1	1001	系统繁忙
2	1002	暂无数据
3	1003	请求参数为空

3 rows in set (0.00 sec)

```
mysql>
```

我们这里的msg表中有三条数据

然后我们可以创建一个 Model，我们在 application 的 models 目录下，我们新建一个 Msg.php 文件，然后我们在里面输入如下代码：

```
<?php
defined('BASEPATH') OR exit('No direct script access allowed');

class Msg extends CI_Model
{
    public function __construct()
    {
        $this->load->database();
        parent::__construct();
    }

    public function get_my()
    {
        $data = $this->db
            ->from('msg')
            ->where('id > 1')
            ->select('errcode,errstr')
            ->get();
    }
}
```



```

        ->result();
    return $data;
}
}

```

需要说明的是，我们的 Model 也可以像控制器那样通过 `$this->load->database()` 来加载数据库，然后它也可以通过 `$this->db` 来引用数据库对象。在上面的代码中，我们在构造函数中加载了数据库对象，那么我们就不用在每个方法中都写这个来加载数据库对象了，然后我们的 `get_my()` 方法就可以直接使用 `$this->db` 来获取数据，然后返回了。

这样我们的一个 Model 就定义完毕了，一般来说和操作消息有关的代码都应该写到这个里面。

*****使用模型*****

在我们定义了模型之后，那么我们该怎么使用这个模型呢？我们一般需要做这么两步工作：

- (1) 通过 `$this->load->model('xxx')` 来加载一个模型。
- (2) 通过 `$this->xxx` 来调用这个模型，包括其方法和属性。

我们这里可以创建一个 Demo 控制器，然后我们在它的 `index()` 方法中调用 Msg 这个模型，代码如下：

```

<?php
defined('BASEPATH') OR exit('No direct script access allowed');

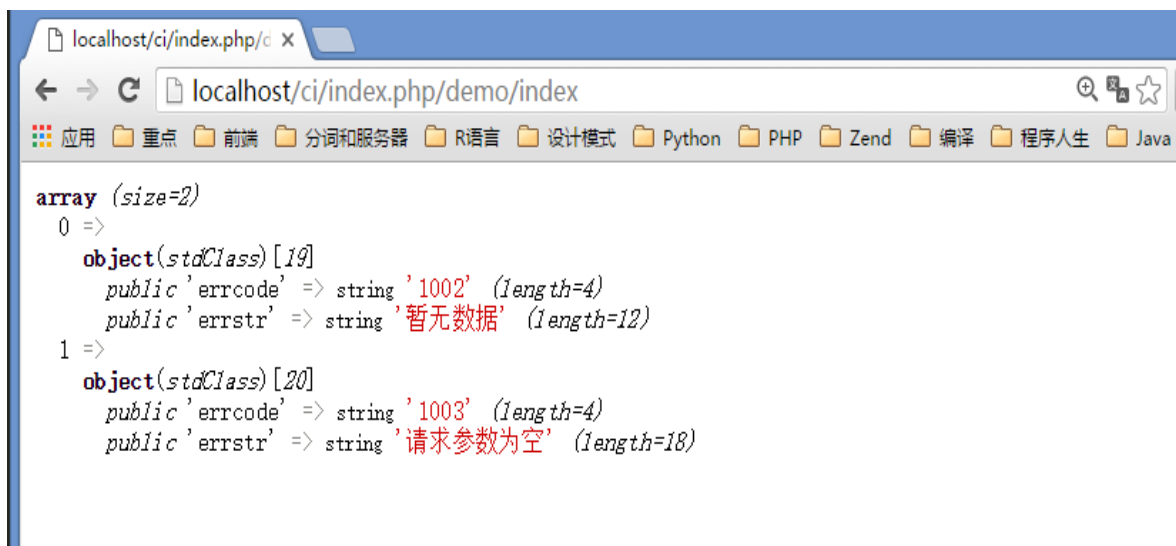
class Demo extends CI_Controller
{
    public function index()
    {
        //加载 Msg 模型
        $this->load->model('msg');

        //获取数据
        $data = $this->msg->get_my();

        var_dump($data);
    }
}

```

在我们的 `index` 方法中，我们调用了我们的模型的方法，然后我们来访问一下这个 `index()` 方法吧，我们的访问截图如下：



```
array (size=2)
  0 =>
    object(stdClass) [19]
      public 'errcode' => string '1002' (length=4)
      public 'errstr' => string '暂无数据' (length=12)
  1 =>
    object(stdClass) [20]
      public 'errcode' => string '1003' (length=4)
      public 'errstr' => string '请求参数为空' (length=18)
```

对于我们使用 Model 的例子，我们就介绍到这里啦。

第十一节:MVC

*****MVC 简介*****

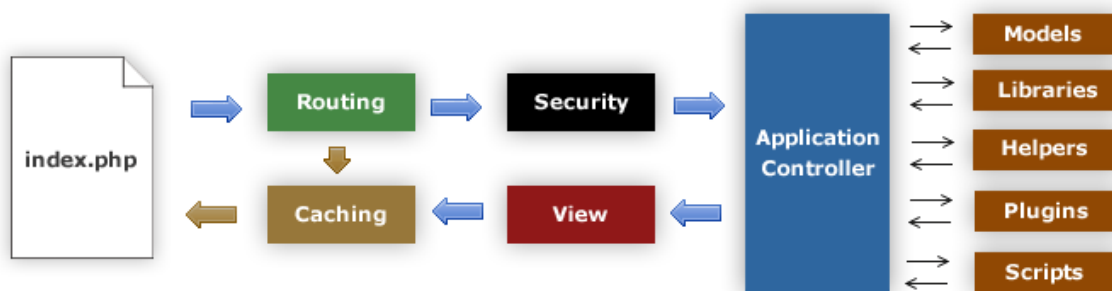
在 web 开发中，MVC 模式是非常常见的一种模式，其全名为"Model View Controller"，其中 Model 即模型，View 即视图，Controller 即控制器，在前面我们已经都介绍过了。

在这种模式下，模型主要处理和数据有关的操作，视图则负责进行界面的展示，控制器则负责进行业务逻辑的处理，比如是否记录日志、是否使用缓存等等操作。通过这种分工，我们的思路更加清晰，当我们的代码量急剧增加的时候，也不会让我们的思路过于混乱。

不过需要说明的是，我们接触的大多数 PHP 框架都是基于 MVC 的模式进行设计的，当然也可能增加了对模块的支持，此时就是 HMVC，其基本思路都是一致的。

*****CI 的实现*****

对于 ci 来说，下面这张图是一个请求的完整的执行方式：



其中 index.php 是我们的统一入口，Routing 则是执行路由操作，Security 则是进行安全过滤，Caching 则是进行缓存，而 View、Application Controller、Models 就分别代表了 V、C、M 了。

它的主要优点就是对不同的部分进行了切分，这样使得多个人进行合作的时候更加方便，比如编写界面的只需要处理 View 部分的东西就可以了，这样可以减少我们的沟通成本。

附录:致敬读者

*****梦之都*****

- 1.梦之都致力于发布优秀的IT原创教程,而且所有教程都提供两种方式:离线下载(通常是PDF格式)和在线阅读(通过网页的方式)。
- 2.在前进的道路上,梦之都希望做您最优秀的得力助手。学海无涯,与君共勉。
- 3.梦之都的官方网站为:www.mengzhidu.com。我是辛星,我在这里等你。

*****加入我们*****

- 1.现在梦之都已经开放注册啦,朋友们可以到梦之都的官网去下载、收藏自己喜欢的教程奥。
- 2.想在前进的道路上找到更多的伙伴,可以加PHP技术交流群:459233896。