CodeIgniter 用户指南

极客学院出版

前言

她是一个小巧但功能强大的 PHP 框架，作为一个简单而“优雅”的工具包，她可以为 PHP 程序员建立功能完善的 Web 应用程序。如果你是一个与人共享主机并且为客户要求的期限而烦恼的开发人员，如果你已经厌倦了那些傻大笨粗的框架，那么 CodeIgniter 就是你所需要的。

CodeIgniter 是一套给 PHP 网站开发者使用的应用程序开发框架和工具包。它提供一套丰富的标准库以及简单的接口和逻辑结构，其目的是使开发人员更快速地进行项目开发。使用 CodeIgniter 可以减少代码的编写量，并将你的精力投入到项目的创造性开发上。

本教程是对官方用户指南的中文译本。

官方原文地址： http://www.codeigniter.com/user_guide/

适用人群

本教程是给那些想深入了解并使用 CodeIgniter 框架的 PHP 初、中级程序员准备的，对高级 PHP 程序员也有一定的帮助。

学习前提

在学习本教程之前，我们假定你已经熟悉 PHP 语言的编写和使用。

版本信息

书中演示代码基于以下版本：

语言/框架	版本信息
CodeIgniter	3.0.0

目录

1

欢迎来到 CodeIgniter

CodeIgniter 是一个应用开发框架和工具包，用于 PHP 开发 Web 网站。它提供了一套丰富的标准库以及简单的接口和逻辑结构，能让你开发的时候速度更快。使用 CodeIgniter 可以减少代码的编写量，并将你的精力投入到项目的创造性开发上。

第 1 章 CodeIgniter 是为谁准备的?

如果你符合下面的条件，那么 CodeIgniter 就是你需要的：

- 你想要一个小巧的框架
- 你想要出色的性能
- 你想要广泛兼容标准主机上的各种 PHP 版本和配置
- 你想要一个几乎只需 0 配置的框架
- 你想要一个不使用命令行的框架
- 你想要一个不需坚守限制性编码规则的框架
- 你不希望被迫学习一门模板语言（虽然可以选择你喜欢的模板解析器）。
- 你不喜欢复杂，热爱简单
- 你需要清晰、完整的文档

2

下载 CodeIgniter

- CodeIgniter v3.0.0 (当前版本)
- CodeIgniter v2.2.1
- CodeIgniter v2.2.0
- CodeIgniter v2.1.4
- CodeIgniter v2.1.3
- CodeIgniter v2.1.2
- CodeIgniter v2.1.1
- CodeIgniter v2.1.0

第 2 章 GitHub

Git 是分布式版本控制系统。

Git 访问地址：GitHub。请注意，尽管大家都想让这个代码功能更多，但我们不能保证任何 branch 代码功能性

从 2.0.3 版本开始, 也可以通过访问 GitHub Releases 稳定版。

3

安装指南

CodeIgniter 的安装分为四个步骤:

- 解压缩安装包
- 把 CodeIgniter 文件夹和文件上传到你的服务器。通常把 `index.php` 放在根目录。
- 用任何文本编辑器打开 `application/config/config.php`, 并设置网站根 URL。如果你打算使用加密或 Session，需要设置加密密钥。
- 如果你打算使用数据库，用任何文本编辑器打开 `application/config/database.php`, 并设置数据库参数。

如果你希望通过隐藏 CodeIgniter 文件的路径来增加安全性，可以通过修改 system 和 application 目录的名字来实现，把它改成任何你喜欢的名字。如果已经修改了名字，你需要打开主目录下面的 index.php 文件设置里面的 `$system_path` 和 `$application_folder` 变量，把它改成之前修改的新名字。

为了安全考虑,system 和 application 两个文件夹应放到网站根目录以外的地方，这样浏览器就不能够直接访问它们。在默认设置下, 在每个文件夹中都有一个 `.htaccess` 配置文件以拒绝直接访问, 但是当把代码部署到生产环境时最好移除他们，因为生产环境的 Web 服务可能会不支持 `.htaccess` 的配置。

如果你移动了以上两个文件夹,请打开主目录下的 index.php 文件并编辑 `$system_path 和$application_folder` 两个变量, 最好使用绝对路径进行替换, 例如： `/www/MyUser/system`.

另外有一个附加的考虑就是，如果要在生产环境中使用，最好关闭 PHP 的错误报告和其他任何与开发相关的功能，在 CodeIgniter 中，可以设置 `ENVIRONMENT` 常量来实现这个功能。详细的文档请参阅安全章节。

以上就是全部安装过程!

如果你刚刚接触 CodeIgniter，请阅读用户指南的开始部分，学习如何构造动态的 PHP 应用。让我们享受这个过程吧!

4

版本升级

请阅读你所对应版本的升级注意事项

5

疑难解答

如果发现无论你在 URL 里面写什么，都只是出现默认页面的话，有可能是你的服务器不支持 PATH_INFO 变量，它用来给搜索引擎提供友好的 URL。解决这个问题的第一步是打开 `application/config/config.php` 文件，查找 URI Protocol 信息,你可以试试其他的设置方法。如果这些方法都无效，你需要让 CodeIgniter 去强行加一个问号去标记你的 URL。为了做到这点，打开你的 `**application/config/config.php**` 文件，把:

```
$config['index_page'] = "index.php";
```

改成这样：

```
$config['index_page'] = "index.php?";
```

开始 CodeIgniter

学习任何软件都需要努力。我们尽力让大家学习的时候少走弯路，并能享受这一过程。

第一步是安装 CodeIgniter，之后需要阅读**介绍**部分的内容。

下一步，需要按顺序阅读**通用主题**。每个内容都和前一部分有关联，并且包含了实例，你可以试试。

掌握了基础知识后，你可以研究**类库参考**和**帮助参考**，以便使用本地库和帮助文件。

如果你遇到了问题，可以到社区来寻求帮助，也可以看看其他人的例子。

CodeIgniter 特性

开发框架是否优秀与它的特性没有太大的关系。从特性中你不知道用户的体验、不能体验到框架设计是否直接了当，是否智能。特性也不能告诉你框架代码的质量如何、性能如何、细节处理的如何、安全性如何。真正的判断一个框架的唯一办法是使用它。CodeIgniter 的安装很简单，所以请使用它。CodeIgniter的主要特性如下：

- 系统基于 MVC 模型（Model）– 视图（View）– 控制器（Controllers）
- 超轻量级
- 对多种数据库平台的全特性支持的数据库类
- 支持查询组件数据库
- 表单与数据验证
- 安全性与 XSS 过滤
- Session 管理
- 邮件发送类，支持附件，HTML 或文本邮件，多协议(sendmail, SMTP 和 Mail)及其他
- 图像处理类库(剪裁，缩放，旋转等)。支持 GD，ImageMagick 和 BetPBM
- 文件上传类
- FTP 类
- 本地化
- 分页
- 数据加密
- 基准测试
- 全页面缓存
- 错误日志
- 应用程序评测
- 日历类
- User–Agent 类
- Zip 编码类
- 模板引擎类
- Trackback 类
- XML–RPC 类库
- 单元测试类
- 对搜索引擎友好的 URL

- 灵活的 URI 路由
- 支持钩子和类扩展
- 大量的辅助函数

8

初探 CodeIgniter

# #

CodeIgniter 是一个应用框架

CodeIgniter 是 PHP 开发 web 应用的工具集。通过提供一套丰富常用库，简单的接口，和访问这些库的逻辑结构，它能让你从零开始开发的时候速度更快。CodeIgniter 可以让任务的代码量减少，这样你就可以将精力放在开发上。

# #

CodeIgniter 免费

CodeIgniter 是经过 MIT 开源许可授权的，只要你愿意就可以使用它。更多的信息参考 许可协议 。

# #

CodeIgniter 是轻量级的

真正的轻量级。系统核心仅需要非常小的库。它和其他需要很多资源的库明显不同。同时，它的附加库是运行时加载，根据你的进程的需求来定，所以核心库非常的轻且快。

# #

CodeIgniter 非常快

真的非常快。你可以试试找找比 CodeIgniter 性能更好的快。

# #

CodeIgniter 使用 M-V-C

CodeIgniter 使用了模型（Model）- 视图（View）- 控制器（Controllers）的方法，这样可以让逻辑层和表现层分离。这对工程的模板设计者非常有利，它能让代码量变少。更多 MVC 细节参考模型（Model）- 视图（View）- 控制器（Controllers)。

# #

CodeIgniter 生成干净的 URLs

CodeIgniter 生成 URLs 是干净的，并且对搜索引擎友好。不同于标准的“字符串查询”方法，CodeIgniter 使用了基于段（segment-based）的方法：

```
example.com/news/article/345
```

注意: 默认情况下，index.php 文件包含在 URL，但是可以通过一个简单的 `.htaccess` 文件移除。

# #

CodeIgniter 功能强大

CodeIgniter 拥有全范围的类库，可以满足大多数网络开发任务的需求，包括： 读取数据库、发送电子邮件、数据确认、保存 session 、对图片的操作，以及支持 XML-RPC 数据传输等。

# #

CodeIgniter 可扩展

CodeIgniter 系统可以通过自定义的类库，辅助函数，类扩展，或系统钩子，简单的实现系统扩展。

# #

CodeIgniter 不需要模板引擎

虽然 CodeIgniter 自带了一个可选的模板解析器程序，但并不强制你使用。模板引擎与本地 PHP 性能不匹配，使用模板引擎我们要学习其语法，这最低限度只比学PHP基础要容易一点点。看看以下PHP 代码：

```
<ul>
<?php foreach ($addressbook as $name):?>
   <li><?=$name?></li>
<?php endforeach; ?>
</ul>
```

再来对比模板引擎所使用的伪代码：

```
<ul>
{foreach from=$addressbook item="name"}
   <li>{$name}</li>
{/foreach}
</ul>
```

模板引擎的例子更加干净一些，但是性能更差，因为它需要先转为 PHP 代码才能运行。因为我们的目标是**最佳性能**，所以我们不用模板引擎。

# #

CodeIgniter 已经彻底文档化

开发者一般喜欢写代码，而不喜欢写文档。我们当然也一样，不过既然文档和代码**一样重要**，我们就要完成它。我们的代码非常干净同时注释也非常优秀。

# #

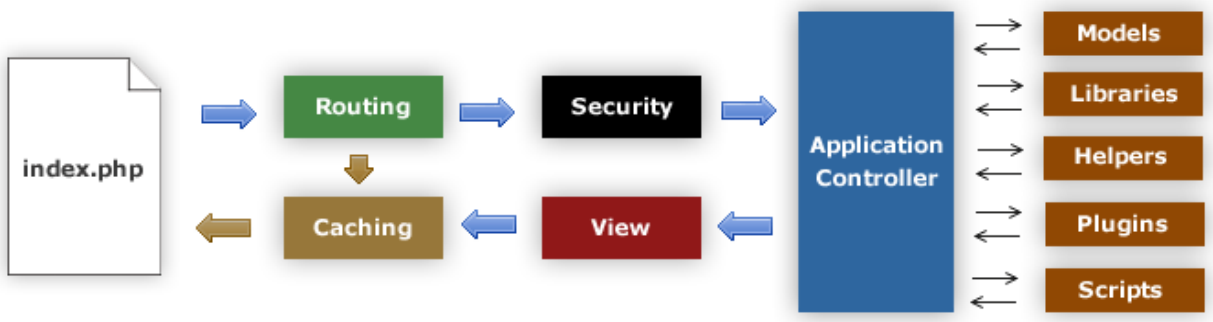
CodeIgniter 的用户社区非常友好

我们的社区非常的活跃，大家参与的积极性非常高。

9

应用程序流程图

下图展示了数据如何如何贯穿系统：



图片 9.1 应用程序流程图

1. index.php 作为前端控制器，初始化运行 CodeIgniter 所需的资源。
2. Router 检查 HTTP 请求，以确定谁来处理它
3. 如果缓存文件已经存在，将会直接发送给浏览器，不需要系统执行
4. 安全性。在应用控制器加载前， HTTP 请求和任何用户请求的数据将会被过滤。
5. 控制器加载模型，核心库，辅助函数，和其他处理某个请求需要的任何资源。
6. 最终视图(View)经过渲染，发送到 Web 浏览器。如果开启缓存(Caching)，视图首先被缓存，以便用于以后的请求。

10

模式–视图–控制

CodeIgniter 基于模式–视图–控制这一模式设计。MVC 能将逻辑层和展示层分离。实际上，它能让你的 web 页面包含最少的代码，因为显示模块从 PHP 代码中分离出来了。

- **模式** 表示你的数据结构。通常你的模式类包含的函数，能帮你检索，插入，更新数据块里的数据。
- **视图** 是展示给用户看的信息。视图通常是一个 web 页面，但是在 CodeIgniter 中，视图也可以是页面的一部分，比如头部，底部。也可以是 RSS 页面，或者其他任何类型的页面。
- **控制** 是**模式**，**视图**，需要处理的资源的桥梁，并生成 web 页面。

CodeIgniter 在使用 MVC 方面非常的宽松，因为**模式**并不是必须的。如果你觉得自己不需要分离，或者觉得维护**模式**，需要花费更多的精力，你可以不用管**模式**，仅使用**控制**和**视图**。

CodeIgniter 也可以和你现有的脚本合并使用，或者允许自行开发此系统的核心库，可以让你找到最适合你的方式工作。

11

设计和架构目标

CodeIgniter 的目标是在最小化，最轻量级的开发包中得到最高的执行效率、功能和灵活性。

为了实现这个目标，我们在开发过程的每一步都致力于基准测试、重构和简化工作，拒绝加入任何对实现目标没有帮助的东西。

从技术和架构角度看，CodeIgniter 按照下列目标创建：

- **动态实例化**。在 CodeIgniter 中，只有在需要的时候，才导入组件，执行函数，而不是在全局范围。除了最小的核心资源外，不假设系统需要任何资源，因此缺省的系统非常轻量级。被 HTTP 请求所触发的事件，以及你设计的控制器和视图将决定什么时候触发他们。
- **松耦合**。耦合是指系统里的组件之间的关联程度。组件相互依赖越少，这个系统的重用性和灵活性就越好。我们的目标是一个松耦合的系统。
- **组件专一性**。专一性是指组件有一个非常小的专注目标。在 CodeIgniter 里，为了达到最大的用途，每个类和它的功能都是高度自治的。

CodeIgniter 动态实例化，松耦合，组件高度专一。它用一个很小的开发包，实现了简单，灵活和高性能。

12

静态页面

注意: 这个文档假设在你的开发环境中已经下载了 CodeIgniter 和文档。

首先，你需要创建一个能处理静态页面的控制器类。控制器类可以帮助代理工作。它是 Web 应用程序的粘合剂。

例如, 假设调用了以下 URL 请求:

```
http://example.com/news/latest/10
```

我们可以假设有一个控制器 "news"。调用此类下的 “latest” 方法，获取最新的 10 条内容，并显示到页面上。基于 MVC 架构，通常我们可以看到 URL 模式如下：

```
http://example.com/[controller-class]/[controller-method]/[arguments]
```

实际上 URL 语法可能会更复杂，这个模式也会发生变化。但是到目前为止，我们仍然需要了解这个模式。

创建一个文件位于 `application/controllers/Pages.php` ,代码如下：

```
<?php
class Pages extends CI_Controller {

   public function view($page = 'home')
   {
      }
}
```

你已经创建了一个 `pages` 类，包含一个名为 `view` 的方法，参数为 `$page` 。 `page` 类继承自 `CI_Controller` 类。这个新的 pages 类可以继承 `CI_Controller` (system/core/Controller.php) 类里面定义的方法和变量。

这个控制器将会成为 **web 应用请求的中心**。在 CodeIgniter 技术讨论中，它被称为超级对象。和任何 php 类一样，可以通过 `$this` 来调用这个控制器。通过 `$this` ，可以加载库，视图，和常用的框架。

现在你已经有了第一个方法，可以开始创建基础页面模板了。我们将会创建两个视图（页面模板），页头和页尾。

创建头文件 `*application/views/templates/header.php*` ，添加以下代码：

```
<html>
   <head>
      <title>CodeIgniter Tutorial</title>
   </head>
   <body>

      <h1><?php echo $title; ?></h1>
```

头文件包含加载主视图前的 HTML 基础代码，。同时也会输出稍后我们会在控制器里定义的 `$title` 变量。接着，创建页尾 *application/views/templates/footer.php* 包含以下代码。

```
      <em>&copy; 2015</em>
   </body>
</html>
```

# #

给控制器添加逻辑代码

之前你创建了一个包含 `view()` 方法的控制器。这个方法包含一个参数，它是将要加载的页面。静态页面路径：*application/views/pages/* 。

这个文件夹里，创建两个文件 *home.php* 和 *about.php*。在这两个文件里，输入一些文字，任何文字都可以，并保存。如果你喜欢不寻常的内容，可以试试 "Hello World"。

为了加载这些页面，你需要检查请求页面是否真的存在：

```
public function view($page = 'home')
{
        if ( ! file_exists(APPPATH.'/views/pages/'.$page.'.php'))
    {
        // Whoops, we don't have a page for that!
        show_404();
    }

    $data['title'] = ucfirst($page); // Capitalize the first letter

    $this->load->view('templates/header', $data);
    $this->load->view('pages/'.$page, $data);
    $this->load->view('templates/footer', $data);
}
```

现在，如果页面存在，将会被加载并显示给用户，包含页头，页尾。如果页面不存在，将会显示 `404` 页面。

第一行代码检查页面文字是否存在。PHP 代码 `file_exists()` 检查文件是否存在。 `show_404()` 是 CodeIgniter 内置函数指向默认的错误页面。

在头文件模板中， `$title` 变量用来初始化页面标题。标题的内容在这个方法中定义，不过我们并没有将值直接赋值给变量，而是把它塞入了 `$data` 数组。

最后需要按显示顺序加载视图。 `view()` 方法的第二个参数用来给视图传值。 `$data` 数组中的每个值都被定义成与它关键字相同的一个变量，如控制器中 `$data['title']` 的值就等同于视图中变量 `$title` 。

# #

路由

控制器现在可以工作了！在浏览器中输入 `[your-site-url]index.php/pages/view` 可以看到你的页面，输入 `index.php/pages/view/about` 可以看到你的关于页面，它也包含页头和页尾。

使用自定义的路由规则，你可以映射任意 URI 到任意的控制器和方法，这样就可以摆脱常用的转换了：`http://example.com/[controller-class]/[controller-method]/[arguments]`

让我们来试试。打开路由文件 *application/config/routes.php*，并且添加以下代码。删除设置 `$route` 数组的相关代码：

```
$route['default_controller'] = 'pages/view';
$route['(:any)'] = 'pages/view/$1';
```

CodeIgniter 从头到尾读取路由规则，并且路由到一个匹配规则上。每一个规则都是一个正则表达式（左侧）映射到一个由斜线分隔的控制器和方法名（右侧）。当有请求时，CodeIgniter 查找第一个匹配规则，并且调用合适的控制器和方法，可能还会带参数。

更多和 URI 路由相关的信息可以查看路由文档.

上面的代码第二行是指利用通配符 (:any) 可以使任何请求都能匹配到 `$routes` 数组，并且通过参数传递给 pages 类的 `view()` 方法。

现在访问：`index.php/about`。看看是否已经能正确地显示页面了呢？真帅！

13

读取新闻

上一个章节中，我们通过写一个包含静态页面的类，了解了这个架构的基本概念。通过添加自定义路由规则我们也重新梳理了 URI。现在我们开始介绍动态内容，并开始使用数据库

#

创建数据模型

数据库操作并不在控制器中（controller），而是在数据模型里，这样可以很容易的被复用。一般我们在数据模型中查询，插入，更新数据库信息。它表示你的数据。

打开 *application/models/* 文件夹并创建一个新的文件 *News_model.php* ，添加以下代码。请确认你已经按照文档数据库配置了数据库。

```
<?php
class News_model extends CI_Model {

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

这段代码和之前的控制器代码很相似。通过继承 CI_Model 并加载数据库，创建了一个数据模型。通过 $this->db 对象可以使得数据库类可用。

在查询数据库前，需要创建一个数据表。连接数据库并运行 SQL 命令（MySQL），并在里面添加内容。

```
CREATE TABLE news (
    id int(11) NOT NULL AUTO_INCREMENT,
    title varchar(128) NOT NULL,
    slug varchar(128) NOT NULL,
    text text NOT NULL,
    PRIMARY KEY (id),
    KEY slug (slug)
);
```

现在数据库和数据模型已经创建好了，我们需要一个方法把我们的文章从数据库中提取出来。为了完成这个目标，CodeIgniter 中包含了的抽象层：查询生成器。它能让你的查询语句只写一次并让它们工作（支持的所有数据库）。添加以下代码到你的数据模型中。

```
public function get_news($slug = FALSE)
{
    if ($slug === FALSE)
    {
        $query = $this->db->get('news');
```

```
        return $query->result_array();
    }

    $query = $this->db->get_where('news', array('slug' => $slug));
    return $query->row_array();
}
```

通过这段代码，你可以执行 2 种不同的查询。你可以得到所有新的记录，或者通过 `slug <#>` 得到新的记录。你可能已经注意到 `$slug` 变量在运行前并没有被验证过，因为查询生成器已经把这事弄完了。

#

显示新闻

既然已经写好了查询，我们就需要把这个数据模型和视图相关联。其实这个事情在我们之前写的 `Pages` 控制器中可以实现，但是为了更清楚的展示给大家，我们创建一个新的 `News` 控制器，这个控制器位于 *application/controllers/News.php*。

```
<?php
class News extends CI_Controller {

    public function __construct()
    {
        parent::__construct();
        $this->load->model('news_model');
        $this->load->helper('url_helper');
    }

    public function index()
    {
        $data['news'] = $this->news_model->get_news();
    }

    public function view($slug = NULL)
    {
        $data['news_item'] = $this->news_model->get_news($slug);
    }
}
```

这段代码和我们之前写过的类似。首先， `__construct()` 方法调用了父类(`CI_Controller`) 的构造函数并加载了数据模型。这样它就能在这个控制器的其他方法中使用。同时也加载了 `URL Helper` 函数集合，它会在之后的视图中用到。

接着，有两个方法可以显示所有新闻和某个新闻。在第二个方法的参数是 `$slug` 。这个方法用 `$slug` 来确定返回哪个新闻数据。

现在，使用我们的数据模型检索除了数据，但是还没有显示任何内容。下一步就是将数据传给视图。

```
public function index()
{
    $data['news'] = $this->news_model->get_news();
    $data['title'] = 'News archive';
```

```
    $this->load->view('templates/header', $data);
    $this->load->view('news/index', $data);
    $this->load->view('templates/footer');
}
```

上面的代码从数据模型中获取到所有的新闻，并将它赋值给变了。文章标题为 `$data['title']` ，所有的数据都会传递给视图。现在你需要创建一个视图来显示这些内容。创建 *application/views/news/index.php* 并添加以下代码。

```
<h2><?php echo $title; ?></h2>

<?php foreach ($news as $news_item): ?>

    <h3><?php echo $news_item['title']; ?></h3>
    <div class="main">
        <?php echo $news_item['text']; ?>
    </div>
    <p><a href="<?php echo site_url('news/'.$news_item['slug']); ?>">View article</a></p>

<?php endforeach; ?>
```

这里，遍历所有的新闻并显示给用户。从上面的代码可以看出我们的模板是 PHP 和 HTML 的混合体。如果你想用模板语言，可以使用 CodeIgniter 模板解析器或者第三方解析器。

新闻列表页面已经完成，但是还没有新闻的独立显示页面。早些时候创建的数据模型可以很容易的实现这个功能。仅需要你添加以下代码到控制器并创建一个视图。回到 `News` 控制器并使用以下代码更新 `view()` 。

```
public function view($slug = NULL)
{
    $data['news_item'] = $this->news_model->get_news($slug);

    if (empty($data['news_item']))
    {
        show_404();
    }

    $data['title'] = $data['news_item']['title'];

    $this->load->view('templates/header', $data);
    $this->load->view('news/view', $data);
    $this->load->view('templates/footer');
}
```

现在这里将 $slug 变量作为参数传给了 get_news() 方法，这样就可以方法想要的文章了。剩下需要做的事情仅仅是创建一个相应的视图 *application/views/news/view.php* 。添加以下代码：

```
<?php
echo '<h2>'.$news_item['title'].'</h2>';
echo $news_item['text'];
```

#

路由

因为之前设置了通配符路由规则，现在你需要额外的路由来显示刚刚写的控制器。按照以下代码修改你的路由文件 `*application/config/routes.php*` 。它让你的请求到 `News` 控制器，而不是直达 `Pages` 控制器。第一行代码表示的是 `news` 控制器中通过 `slug` 读取的那条新闻。

```
$route['news/(:any)'] = 'news/view/$1';
$route['news'] = 'news';
$route['(:any)'] = 'pages/view/$1';
$route['default_controller'] = 'pages/view';
```

把浏览器地址定位到你的根目录，后面加上 `index.php/news` ，然后看看你的新闻页面。

14

创建新闻

现在你已经知道如何使用 CodeIgniter 从数据库里读取数据，但你还没往数据库写任何东西。这个章节将会扩展你的之前创建的新闻控制器和模式包含这个功能。

#

创建一个表单

为了给数据库输入数据，你需要创建一个表单来存储信息。你的表单需要两个字段，标题和正文。你可以从数据模型中得标题来获得 slug。在 `application/views/news/create.php` 创建一个新的视图。

```
<h2><?php echo $title; ?></h2>

<?php echo validation_errors(); ?>

<?php echo form_open('news/create'); ?>

    <label for="title">Title</label>
    <input type="input" name="title" /><br />

    <label for="text">Text</label>
    <textarea name="text"></textarea><br />

    <input type="submit" name="submit" value="Create news item" />

</form>
```

这里可能有 2 个事情你不太了解： `form_open()` 函数和 `validation_errors()` 函数。

第一个函数由表单辅助函数提供,用来提供表单元素和额外的附加函数，比如添加隐藏的安全类。另外一个用来报告验证表单正确性过程中的错误。

回到你的新闻控制器。在这里你需要做 2 件事情，一是检查表单是否已经提交，二是检查提交的数据能通过验证，你需要用到表单验证来坐这些事。

```
public function create()
{
    $this->load->helper('form');
    $this->load->library('form_validation');

    $data['title'] = 'Create a news item';

    $this->form_validation->set_rules('title', 'Title', 'required');
    $this->form_validation->set_rules('text', 'text', 'required');

    if ($this->form_validation->run() === FALSE)
    {
```

```
        $this->load->view('templates/header', $data);
        $this->load->view('news/create');
        $this->load->view('templates/footer');

    }
    else
    {
        $this->news_model->set_news();
        $this->load->view('news/success');
    }
}
```

上面的代码添加了一些功能，前几行代码载入了表单辅助函数和表单验证库。这样就设置好了表单验证规则。`set_rules()` 方法有 3 个参数：输入域的名称，错误信息名称，规则。这里的规则是标题和正文不能为空。

如上所示，CodeIgniter 拥有强大的表单验证库。详情可以阅读表单验证文档。

继续看，你可以发现一个用来检查表单验证是否运行成功的条件。如果失败，显示表单。如果成功提交并通过所有验证，视图将会显示消息。创建一个视图 `application/views/news/success.php`，并写入成功消息。

#

数据模型

最后需要做的事情就是将数据写到数据库中。你将会用到 Query Builder 类来插入一条信息，并使用输入库来获得 post 数据。打开之前创建的数据模型，并添加以下代码：

```
public function set_news()
{
    $this->load->helper('url');

    $slug = url_title($this->input->post('title'), 'dash', TRUE);

    $data = array(
        'title' => $this->input->post('title'),
        'slug' => $slug,
        'text' => $this->input->post('text')
    );

    return $this->db->insert('news', $data);
}
```

这新方法用来维护插入到数据库的信息。第三行包含了一个新的函数 `url_title()` 。这个函数由 URL 辅助函数提供，用来组织你输入的字符串，将空格的内容换成横线（-），确保其中全都是小写字母。这样就可以拥有一个漂亮的 slug，可以完美的创建 URI。

让我们继续准备将要向 $data 数组输入的记录。每个元素都和数据库表里的列相对应。你可能已经注意到了一个新方法 post()。这个方法可以保证数据是过滤过的，从而保护你不受其他人的恶意攻击。这个输入类默认加载。最后将 $data 数据插入到数据库中。

#

路由

在你添加新闻到 CodeIgniter 应用前，你必须给 `config/routes.php` 添加一个附件的规则。它需要保证你的文件包含以下代码。它能让 CodeIgniter 将 “create” 看做一个方法，而不是一个新闻的 slug。

```
$route['news/create'] = 'news/create';
$route['news/(:any)'] = 'news/view/$1';
$route['news'] = 'news';
$route['(:any)'] = 'pages/view/$1';
$route['default_controller'] = 'pages/view';
```

现在可以将你的浏览器定位到安装了 CodeIgniter 本地开发环境，并添加 `index.php/news/create` 到URL。恭喜你创建了第一个 CodeIgniter 程序！添加一些新闻，然后看看以创建的不同页面吧。

15

CodeIgniter URL

默认情况下，CodeIgniter 的 URL 设计成对搜索引擎和人类友好。不同于使用标准的“查询字符串”方法（它和动态系统同步），CodeIgniter 使用的是基于段的方法：

```
example.com/news/article/my_article
```

注意：查询字符串 URL 是可以选的，如下所示

# #

URI 段

根据模型-视图-控制器模式，此 URL 段一般按以下形式表示：

```
example.com/class/function/ID
```

1. 第一段表示调用控制器类
2. 第二段表示将要调用的类函数或类方法
3. 第三及更多的段表示的是传递给控制器的参数，如 ID 或其它各种变量

# #

删除 index.php 文件

默认情况下：index.php 文件包含在你的 URLS 里：

```
example.com/index.php/news/article/my_article
```

如果你的 Apache 服务器允许 `mod_rewrite` 。你可以很容易的移除这个文件。下面是一个例子，使用"negative"方法将非指定内容重新定向：

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)$ index.php/$1 [L]
```

在上面的例子中，任何指向不存在的文件夹和文件的 HTTP 请求都被定向到 index.php。

注意：这些特定的规则不一定适合所有的服务器配置。

# #

添加一个 URL 后缀

config/config.php 文件里，你可以指定一个后缀，这样 CodeIgniter 生成的所有 URL 都会添加上。例如 URL 如下：

```
example.com/index.php/products/view/shoes
```

你可以选择增加一个后缀，比如 *.html,* ，这样 URL 就会变成：

```
example.com/index.php/products/view/shoes.html
```

# #

允许查询字符串

某些情况下，你可能会选择使用查询字符串 URL：

```
index.php?c=products&m=view&id=345
```

CodeIgniter 可以有选择的支持这个功能，在 *application/config.php* 文件里可以打开，打开后，内容如下：

```
$config['enable_query_strings'] = FALSE;
$config['controller_trigger'] = 'c';
$config['function_trigger'] = 'm';
```

如果将 "enable_query_strings" 改为 TRUE，这个功能就可用。通过 `触发器` 关键字来调用你的控制器和函数：

```
index.php?c=controller&m=method
```

注意：如果你使用了查询字符串，你就需要创建自己的 URLs，而不是使用 URL 辅助函数（以及其他辅助生成 URLs 的函数，比如表单辅助函数），因为设计的时候他们是基于段的 URLs。

16

控制器

控制器是应用程序的核心，因为它决定了如何处理 HTTP 请求。

# #

什么是控制器

控制器是一个简单的类文件，它的命名方式能和 URI 相关联。

看看这个 URI：

```
example.com/index.php/blog/
```

在上述的例子中，CodeIgniter 尝试寻找一个名为 Blog.php 的控制器并加载。

当一个控制器名字和 URI 第一个部分相符合时，将会被加载

#

让我试试: Hello World!

让我创建一个简单的控制器以便观察他是如何工作的。使用你的文本编辑器，创建一个 Blog.php 的文件，并加入以下代码：

```
<?php
class Blog extends CI_Controller {

    public function index()
    {
        echo 'Hello World!';
    }
}
```

保存到 *application/controllers/* 文件夹.

注意: 文件名必须是'Blog.php'，首字母 'B' 大写.

现在，可以使用和下面类似 URL 访问你的站点：

```
example.com/index.php/blog/
```

如果一切正确，你将会看到：

```
Hello World!
```

注意: 类名必须以大写字母开头

这个是有效的:

```
<?php
class Blog extends CI_Controller {

}
```

这个是无效的：

```
<?php
class blog extends CI_Controller {

}
```

同时要保证你的控制器继承自父控制器，它能继承所有的方法。

#

方法

上面的列子，方法名 `index()` 。如果第二部分为空，将会载入 "index" 方法。第二种显示 "Hello World" 消息的方法如下：

```
example.com/index.php/blog/index/
```

URI 的第二段决定了将会调用控制器里的哪个方法。

让我们试试添加一个新方法到你的控制器：

```
<?php
class Blog extends CI_Controller {

    public function index()
    {
        echo 'Hello World!';
    }

    public function comments()
    {
        echo 'Look at this!';
    }
}
```

现在用下面的 URL 试着调用第二个方法：

```
example.com/index.php/blog/comments/
```

你将会看到新的消息。

# #

将 URI 段作为参数传递给你的方法

如果你的 URI 的段落超过 2 个，他们将会作为参数传递。

例如，我们有一个 URI 如下：

```
example.com/index.php/products/shoes/sandals/123
```

你的方法将会受到参数 3 和 4（"sandals" 和 "123"）：

```
<?php
class Products extends CI_Controller {

    public function shoes($sandals, $id)
    {
        echo $sandals;
        echo $id;
    }
}
```

注意：如果你使用了 URI 路由特性，传递给你的方法的参数将会是重新路由过的对象。

#

定义一个默认控制器

当 URI 不存在，或者直接访问根目录的时候，CodeIgniter 将会载入默认控制器。打开 *application/config/routes.php* 文件，并输入以下内容，可以改变默认控制器：

```
$route['default_controller'] = 'blog';
```

当 Blog 正是你想用使用的控制器类时。现在如果你加载 index.php，而没有指定任何 URI 段，默认情况下将会看到 "Hello World" 消息。

#

重新映射调用方法

如上所述，URI 的第二个段通常会确定调用控制器里的哪个方法。CodeIgniter 允许你重写这个行为，通过使用 `_remap()` 方法。

```
public function _remap()
{
    // Some code here...
}
```

注意: 如果你的控制器包含一个 _remap() 方法，它将会始终被调用，无论你的 URI 包含什么。它重写了正常的 URI 决定调用哪个方法的行为，允许你来定义自己的路由规则。

被重新定义的方法调用方式（一般是 URI 中的第二段）将作为一个参数传递给 `_remap()` ：

```
public function _remap($method)
{
    if ($method === 'some_method')
    {
        $this->$method();
    }
    else
    {
        $this->default_method();
    }
}
```

任何在该方法名称之后的附加段都会被视为 `_remap()` 的第二个参数（可选）。这个可选的数组参数可以与 PHP 的 `call_user_func_array` 联用，模拟 CodeIgniter 的默认行为。

例如:

```
public function _remap($method, $params = array())
{
    $method = 'process_'.$method;
    if (method_exists($this, $method))
    {
        return call_user_func_array(array($this, $method), $params);
    }
    show_404();
}
```

#

处理输出

CodeIgniter 拥有一个输出类用来确保你修改的数据会自动被传递给浏览器。关于这个的更多信息可以在视图和输出类里找到。有些时候，你可能想要自己发布修改一些最终的数据或是自己把它传递给浏览器。CodeIgniter 允许你给你的控制器增加一个名为 `_output()` 的方法来接收最终的数据。

注意：如果你的控制器包含一个 `_output()` 方法，那么它将总是被调用，而不是直接输出最终的数据。这个方法类似于 OO 里的析构函数，不管你调用任何方法这个方法总是会被执行。例如：

```
public function _output($output)
{
    echo $output;
}
```

注意：你的 _output() 将接收最终的数据。 Benchmark 和内存的使用率数据将被渲染，缓存文件会被写入（如果已启用缓存），并且 HTTP 头也将被发送（如果您使用该功能），然后交给 _output() 函数。

为了让你的控制器输出缓存正确, 它的 _output() 函数可以这样来写:

```
    if ($this->output->cache_expiration > 0)
    {
        $this->output->_write_cache($output);
    }
```

如果您正在使用页面执行时间和内存使用统计的功能，这可能不完全准确，因为他们不会考虑到你所做的任何进一步的动作。请在输出类参用可用的方法，来控制输出以使其在任何最终进程完成之前执行。

# #

私有方法

在某些情况下，你可能想要隐藏一些方法使之无法对外查阅。将方法私有化很简单，只要在方法名字前面加一个下划线（“_”）做前缀就无法通过 URL 访问到了。例如，如果你有一个像这样的方法：

```
private function _utility()
{
    // some code
}
```

那么，通过下面这样的 URL 进行访问是无法访问到的：

```
example.com/index.php/blog/_utility/
```

注意:前缀方法名称使用下划线，是为了避免被调用。这是原本就有的功能，目的是向后兼容。

# #

如何将控制器放入子文件夹中

如果你在建立一个大型的应用程序，你会发现 CodeIgniter 可以很方便的将控制器放到一些子文件夹中。

只要在 application/controllers 目录下创建文件夹并放入你的控制器就可以了。

注意：如果你要使用某个子文件夹下的功能，就要保证 URI 的第一个片段是用于描述这个文件夹的。例如说你有一个控制器在这里：

```
application/controllers/products/Shoes.php
```

调用这个控制器的时候你的 URI 要这么写

```
example.com/index.php/products/shoes/show/123
```

你的每个子文件夹中需要包含一个默认的控制器，这样如果 URI 中只有子文件夹而没有具体功能的时候它将被调用。只要将你作为默认的控制器名称在 application/config/routes.php 文件中指定就可以了。

CodeIgniter 也允许你使用 URI 路由功能来重新定向 URI。

# #

构造函数

如果要在你的任意控制器中使用构造函数的话，那么必须在里面加入下面这行代码：

```
parent::__construct();
```

这行代码的必要性在于，你此处的构造函数会覆盖掉这个父控制器类中的构造函数，所以我们要手动调用它。例如:

```
<?php
class Blog extends CI_Controller {

    public function __construct()
    {
        parent::__construct();
        // Your own constructor code
    }
}
```

如果你需要设定某些默认的值或是在实例化类的时候运行一个默认的程序，那么构造函数在这方面就非常有用了。

构造函数并不能返回值，但是可以用来设置一些默认的功能。

#

保留的方法名称

因为你添加的控制器类继承了主要的应用程序控制器，所以你要小心你的方法名不要和那个类中的方法名一样了，否则你的方法会覆盖原有的。详细信息请查看保留字部分。

注意: 你不要让方法名和类名相同。如果你这么做，将不会有构造函数，然后 `Index::index()` 方法将会作为类的构造函数执行。这是 PHP 4 向后兼容的特性。

# #

就这样

好，以上就是有关控制器的所有内容了。

17 保留字

为了便于编码，CodeIgniter 使用了一系列的函数，方法，类和变量名来完成操作。因此，有一些名字开发中不能使用。底下的列表中的保留字不能被使用。

# #

控制器名

因为你的控制器类继承自主程序控制器，你方法名一定不要和主程序控制器类中得函数名相同，否则你的局部方法将会覆盖他们。底下列出了保留的名字。不能在你的控制器中使用：

- CI_Controller
- Default
- index

# #

函数

- :php:func: is_php()
- :php:func: is_really_writable()
- load_class()
- is_loaded()
- get_config()
- :php:func: config_item()
- :php:func: show_error()
- :php:func: show_404()
- :php:func: log_message()
- :php:func: set_status_header()
- :php:func: get_mimes()
- :php:func: html_escape()
- :php:func: remove_invisible_characters()
- :php:func: is_https()
- :php:func: function_usable()
- :php:func: get_instance()
- _error_handler()
- _exception_handler()
- _stringify_attributes()

# #

变量

- `$config`
- `$db`
- `$lang`

# #

常量

- ENVIRONMENT
- FCPATH
- SELF
- BASEPATH
- APPPATH
- VIEWPATH
- CI_VERSION
- MB_ENABLED
- ICONV_ENABLED
- UTF8_ENABLED
- FILE_READ_MODE
- FILE_WRITE_MODE
- DIR_READ_MODE
- DIR_WRITE_MODE
- FOPEN_READ
- FOPEN_READ_WRITE
- FOPEN_WRITE_CREATE_DESTRUCTIVE
- FOPEN_READ_WRITE_CREATE_DESTRUCTIVE
- FOPEN_WRITE_CREATE
- FOPEN_READ_WRITE_CREATE
- FOPEN_WRITE_CREATE_STRICT
- FOPEN_READ_WRITE_CREATE_STRICT
- SHOW_DEBUG_BACKTRACE
- EXIT_SUCCESS

- EXIT_ERROR
- EXIT_CONFIG
- EXIT_UNKNOWN_FILE
- EXIT_UNKNOWN_CLASS
- EXIT_UNKNOWN_METHOD
- EXIT_USER_INPUT
- EXIT_DATABASE
- EXIT__AUTO_MIN
- EXIT__AUTO_MAX

18 视图

视图是一个简单的 Web 页面，或者页面的部分，如页头，页尾，侧边栏等等。实际上，如果你需要这种树状类型，视图可以灵活的嵌入到其他视图（或者再嵌入其他视图）。

视图从不会被直接调用，必须通过控制器来调用。记住，在 MVC 架构中，控制器扮演了交通警察的角色，那么它就得负责取回一个特殊视图。如果你没有阅读过控制器文档，那么你需要先阅读一下。

使用之前创建的控制页面，并加入一个视图。

# #

创建一个视图

使用你的文本编辑器，创建一个 blogview.php 文件，并输入：

```
<html>
<head>
   <title>My Blog</title>
</head>
<body>
   <h1>Welcome to my Blog!</h1>
</body>
</html>
```

并保存到 *application/views/* 文件夹。

#

加载一个视图

加载一个特点的视图文件，需要使用以下方法：

```
$this->load->view('name');
```

上面的 name 就是你的视图文件。

注意： .php 后缀名不用特别指定，除非你使用了非 .php 的文件。

现在，打开之前你创建的 Blog.php，使用视图加载方法覆盖 echo 语句。

```
<?php
class Blog extends CI_Controller {

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

如果你使用之前的 URL 浏览网站，你将会看到你的新视图，URL 与下面类似：

```
example.com/index.php/blog/
```

#

载入多个视图

CodeIgniter 将会智能的处理多个从控制器发起的 `$this->load->view()` 调用。如果超过一个调用，它们将会合并到一起。例如，你可能希望有一个页头视图，菜单视图，内容视图，页脚视图。它可能长成这样：

```
<?php

class Page extends CI_Controller {

    public function index()
    {
        $data['page_title'] = 'Your title';
        $this->load->view('header');
        $this->load->view('menu');
        $this->load->view('content', $data);
        $this->load->view('footer');
    }

}
```

上述的列子中，我们使用动态添加数据方法，稍后你将会看到。

# #

用子文件夹保存视图

如果你想让文件系统更有组织性，可以使用子文件夹来保存视图。当你载入视图的时候，必须带有子文件夹名。

```
$this->load->view('directory_name/file_name');
```

#

添加动态数据到视图

控制器将数组或者对象作为第二个参数传递给视图加载方法。底下是使用数组的例子：

```
$data = array(
   'title' => 'My Title',
   'heading' => 'My Heading',
   'message' => 'My Message'
);

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

这个是使用对象作为参数：

```
$data = new Someclass();
$this->load->view('blogview', $data);
```

注意:如果你使用一个对象，类变量将会变为数组元素。

让我们使用控制器文件试试，打开它并添加以下代码：

```
<?php
class Blog extends CI_Controller {

   public function index()
   {
      $data['title'] = "My Real Title";
      $data['heading'] = "My Real Heading";

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

现在打开你的视图文件，将文本改为和数组 key 对应的变量：

```
<html>
<head>
   <title><?php echo $title;?></title>
</head>
<body>
   <h1><?php echo $heading;?></h1>
```

```
</body>
</html>
```

接着使用之前的 URL 加载页面，你将看到变量已经替换。

#

创建循环

你传给视图的变量，不仅是一个简单变量。你也可以传递多维数组，他可以循环生成多行。例如，如果你从你的数据库里获取数据，它既是典型的多维数组。

这里有一个简单的例子，添加到你的控制器：

```
<?php
class Blog extends CI_Controller {

   public function index()
   {
      $data['todo_list'] = array('Clean House', 'Call Mom', 'Run Errands');

      $data['title'] = "My Real Title";
      $data['heading'] = "My Real Heading";

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

现在打开你的视图文件，并创建循环：

```
<html>
<head>
   <title><?php echo $title;?></title>
</head>
<body>
   <h1><?php echo $heading;?></h1>

   <h3>My Todo List</h3>

   <ul>
   <?php foreach ($todo_list as $item):?>

      <li><?php echo $item;?></li>

   <?php endforeach;?>
   </ul>
```

```
</body>
</html>
```

注意:你也许已经注意到了，上面的例子我们使用了 PHP 的替代语法。如果你不熟悉可以阅读 alternative_php 文档.

# #

返回视图

view 函数第三个可选参数可以改变函数的行为，让数据作为字符串返回而不是发送到浏览器。如果你想用其他方法处理数据，这会很有用。如果你将参数设置为 TRUE (boolean)，它将会返回数据。默认为 false，这样会发送给浏览器。如果你想要返回数据，记得将他赋值给变量。

```
$string = $this->load->view('myfile', '', TRUE);
```

19

模型

模型对于那些想使用传统的 MVC 方法的人来说是可选的。

# #

什么是模型

模型是一个 PHP 类，是用来和数据库打交道的。例如，我们假设你使用 CodeIgniter 来管理你的博客。你应该会有一个模型类来插入，更新，检索博客数据。下面的例子将向你展示一个普通的模型类。

```
class Blog_model extends CI_Model {

    public $title;
    public $content;
    public $date;

    public function __construct()
    {
        // Call the CI_Model constructor
        parent::__construct();
    }

    public function get_last_ten_entries()
    {
        $query = $this->db->get('entries', 10);
        return $query->result();
    }

    public function insert_entry()
    {
        $this->title    = $_POST['title']; // please read the below note
        $this->content    = $_POST['content'];
        $this->date    = time();

        $this->db->insert('entries', $this);
    }

    public function update_entry()
    {
        $this->title    = $_POST['title'];
        $this->content    = $_POST['content'];
        $this->date    = time();

        $this->db->update('entries', $this, array('id' => $_POST['id']));
    }
```

```
}
```

注意: 上述例子使用的是 `查询生成器` 数据库方法。

注意: 为了方便，我们直接使用了 `$_POST` 方法。这不是一个好方法，平时我们应该使用输入库方法 `$this->input->post('title')` 。

# #

剖析模型

模型类文件存放在 *application/models/* 文件夹里。如果你想要这种类型的组织，可以在里面创建子文件夹。

最基本的模型类如下：

```
class Model_name extends CI_Model {

    public function __construct()
    {
        parent::__construct();
    }

}
```

Model_name 是你的类名。类名首字母必须大写其他字母小写。确保你的类继承了基本模型类。

文件名必须和类名匹配。例如，假如这是你的类：

```
class User_model extends CI_Model {

    public function __construct()
    {
        parent::__construct();
    }

}
```

类的文件名应该是：

```
application/models/User_model.php
```

# #

加载一个模型

模型可以在控制类中加载并调用。可以使用以下方法来加载模型类：

```
$this->load->model('model_name');
```

如果你的模型在子文件夹中，需要包含模型文件夹的相对路径。例如，如果你在 *application/models/blog/Queries.php* 拥有一个模型，可以这么加载：

```
$this->load->model('blog/queries');
```

一旦加载，你可以使用一个和你的类同名的对象来访问模型方法：

```
$this->load->model('model_name');

$this->model_name->method();
```

如果你想让你的模型有不同的对象名，可以通过指定加载方法第二个参数：

```
$this->load->model('model_name', 'foobar');

$this->foobar->method();
```

这里有个控制器的例子，加载一个模型，然后通过视图显示出来：

```
class Blog_controller extends CI_Controller {

   public function blog()
   {
      $this->load->model('blog');

      $data['query'] = $this->blog->get_last_ten_entries();

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

# #

自动加载模型

如果你觉得需要一个特殊的模型，贯穿整个应用，可以告诉 CodeIgniter 初始化的时候自动加载。实现的方法是打开 *application/config/autoload.php* 文件，然后添加到自动加载数组。

# #

连接到数据库

如果一个模型已经加载，但是没有自动连接到数据库。下面的连接选项可用：

- 您可以使用标准方法来连接数据库, 也可以通过控制器或者您的模型类。
- 您可以把第三个参数设置为 TRUE，来使模型加载函数自动连接数据库，连接配置可以在您的数据库配置文件中可以定义

  ```
  $this->load->model('model_name', '', TRUE);
  ```

- 您可以手动设定第三个参数，来加载您的自定义数据库配置：

  ```
  $config['hostname'] = 'localhost';
  $config['username'] = 'myusername';
  $config['password'] = 'mypassword';
  $config['database'] = 'mydatabase';
  $config['dbdriver'] = 'mysqli';
  $config['dbprefix'] = '';
  $config['pconnect'] = FALSE;
  $config['db_debug'] = TRUE;

  $this->load->model('model_name', '', $config);
  ```

20

辅助函数

辅助函数能帮助你完成任务。每个辅助函数文件都是某个分类的函数集。其中 URL Helpers 帮助我们创建链接，Form Helpers 帮助我们创建元素，Text Helpers 提供一些列的格式化输出，Cookie Helpers 设置并读取 cookies，File Helpers 帮助我们处理文件，等等。

和 CodeIgniter 中的其他部分不同，辅助函数不是通过面向对象方法实现。它们仅仅是简单的过程处理函数，每个辅助函数处理一个特定的任务，并且和其他函数没有依赖性。

CodeIgniter 默认不载入辅助函数文件，所以使用前需要加载。加载后就全局可用。

辅助函数通常存储在 system/helpers，或者 application/helpers 文件夹里。CodeIgniter 首先会查找 application/helpers 文件夹。如果文件夹里不存在，或者文件夹里没有找到，CodeIgniter 将会到 system/helpers 里查找。

# #

加载辅助函数

加载辅助函数文件非常简单，如下所示：

```
$this->load->helper('name');
```

其中 name 是辅助函数的文件名，没有 .php 文件扩展名。

例如，加载 URL Helper 文件，命名为 url_helper.php，可以这么做：

```
$this->load->helper('url');
```

辅助函数可以在你的控制器方法里的任何地方加载（也可以在你的视图文件里加载，不过这不是好方法），建议你在使用辅助函数前加载它。你可以在控制器的构造函数中加载辅助函数，这样你可以在控制器的任何函数中都可以使用，你也可以在某个需要它的函数里加载。

注意: 辅助函数并不返回任何值，所以不要将它赋值给任何变量，直接想例子中那样用就可以了。

#

加载多个辅助函数

如果你需要加载多个辅助函数，你可以将他们加入到数组中，如下：

```
$this->load->helper(
   array('helper1', 'helper2', 'helper3')
);
```

# #

自动加载辅助函数

如果需要某个辅助函数在你的应用任何地方都可以使用，你可以告诉 CodeIgniter 在系统初始化的时候自动加载。打开 application/config/autoload.php 文件，并加入需要的辅助函数。

# #

使用辅助函数

一旦加载了想要用到的辅助函数文件，你就可以用标准的函数方法来调用它。

例如，在你的某个视图文件中使用 anchor() 函数创建链接，可以这么做：

```
<?php echo anchor('blog/comments', 'Click Here');?>
```

"Click Here" 是连接名称，"blog/comments" 是链接到控制器或方法的 URI。

# #

”扩展”辅助函数

如果你想**扩展**辅助函数，在 application/helpers/ 文件夹里创建一个文件，它的名字和需要扩展的辅助函数一致，但是前缀是 MY_（这是可以配置的，如下）。

如果你仅仅是想给原有的辅助函数添加一些方法，或者改变一个方法，就不需要重写自己的辅助函数，仅需要**扩展**它。

注意:“扩展” 这个词其实不太恰当，因为 Helper 的方法是过程式和离散式的，在传统的语言环境中无法被**扩展**，不过在 CodeIgniter 中，你可以添加或者修改 helper 方法。

例如，扩展本地已有的 Array Helper 你应该建立一个文件：application/helpers/MY_array_helper.php，并添加或重写函数:

```
// any_in_array() is not in the Array Helper, so it defines a new function
function any_in_array($needle, $haystack)
{
   $needle = is_array($needle) ? $needle : array($needle);

   foreach ($needle as $item)
   {
      if (in_array($item, $haystack))
      {
         return TRUE;
      }
      }

   return FALSE;
}

// random_element() is included in Array Helper, so it overrides the native function
function random_element($array)
{
   shuffle($array);
   return array_pop($array);
}
```

# #

设置你自己的前缀

为扩展辅助函数的而加的文件名前缀，同时也是对库和核心类的扩展。打开文件 application/config/config.php，然后找到以下内容：

```
$config['subclass_prefix'] = 'MY_';
```

请注意所有本地的 CodeIgniter 库的前缀都是 CI_，所以你不要用这个。

# #

现在可以做什么?

你可以看看在目录里所有的辅助函数它们都是做什么的。

21

使用 CodeIgniter 库

所有可用的库文件都位于 *system/libraries/* 文件夹里。多数情况下，你需要在控制器中初始化后才能使用它们，方法如下：

```
$this->load->library('class_name');
```

'class_name' 是你想要用的类名。例如加载“表单验证类”，可以这样做

```
$this->load->library('form_validation');
```

一旦类库初始化后，你就可以按照用户手册中的方法来使用它们。

此外，给加载函数传递需要加载的多个类库数组，就可以同时加载多个类库。

例如:

```
$this->load->library(array('email', 'table'));
```

# #

创建你自己的类库

请阅读用户手册中关于创建自己类库文档。

22

创建适配器

#

适配器文件夹和文件结构

适配器目录和文件结构如下:

- `/application/libraries/Driver_name`
    - Driver_name.php
    - drivers
        - Driver_name_subclass_1.php
        - Driver_name_subclass_2.php
        - Driver_name_subclass_3.php

注意: 为了和文件名大小写敏感的文件系统兼容，Driver_name 必须用 `ucfirst()` 处理一下

注意: 从适配器架构开说，子类不会扩展并且不会继承主驱动器的属性或方法。

23

创建核心系统类

每次 CodeIgniter 运行时，都有很多基础类作为核心架构的一部分被初始化。你也可以使用你自己的类替换核心系统类，或者扩展核心系统类。

多数用户没有这个需求，但对于那些想较大幅修复 CodeIgniter 的人来说,我们依然提供了替换和扩展核心系统类的方法

注意: 改变核心系统类会有很大的影响，所以在你修改前需要清楚自己在做什么。

#

系统类清单

下面是核心系统类的列表，他们在每次 CodeIgniter 启动时被调用：

- Benchmark
- Config
- Controller
- Exceptions
- Hooks
- Input
- Language
- Loader
- Log
- Output
- Router
- Security
- URI
- Utf8

# #

替换核心系统类

要使用你自己的系统类替换默认类，只需将你的文件放到 *application/core/* 文件夹里：

```
application/core/some_class.php
```

如果文件夹不存在，你可以创建一个。

只要你的自定义文件名和默认的完全一样，它就会自动替换原有的类。

需要注意的时，你的自定义类必须以 CI 作为前缀，例如你的文件名是 Input.php，那么类名就必须是：

```
class CI_Input {

}
```

#

扩展核心系统类

如果你仅仅是需要给存在的库添加一些函数，那完全没有必要替换整个库，仅需要扩展原生库。扩展一个类就像在一个类中添加例外：

- 类声明必须扩展自父类
- 你的新类名和文件名前缀必须是 MY_（这是可配置的，见下面的说明）

例如，想要扩展原生 Input 类，你必须创建一个文件 application/core/MY_Input.php，并且声明类：

```
class MY_Input extends CI_Input {

}
```

注意: 如果在你的类里需要使用构造函数，必须在构造函数中显示的继承父类：

```
class MY_Input extends CI_Input {

    public function __construct()
    {
        parent::__construct();
    }
}
```

提示: 在你的新类中定义的所有函数，如果与父类中函数的命名完全一样,这些函数就能取代父类中原有的函数 (这也被称为"方法覆盖")。这允许你在本质上改变 CodeIgniter 的核心。

如果你扩展了控制器核心类，那么也要在你的应用程序控制器的构造函数里使用这个新类：

```
class Welcome extends MY_Controller {

    public function __construct()
    {
        parent::__construct();
    }

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

# #

自定义前缀

要设定你自己的子类前缀，请打开 *application/config/config.php* 文件，并找到：

```
$config['subclass_prefix'] = 'MY_';
```

请注意：CodeIgniter 所有的库文件前缀都是 CI_，所以你不要用。

24

创建辅助类

在某些情况下，你可能需要开发一个类，这个类能使用控制器一部分功能。

#

get_instance()

- 返回: 你的控制器实例的引用
- 返回类型: CI_Controller

在你的控制器方法中初始化的任何类，都可以访问 CodeIgniter 原生资源，简单通过使用 `get_instance()` 函数。这个函数回传 CodeIgniter 对象。

通常来说，调用任何 CodeIgniter 方法需要你使用 `$this` 结构：

```
$this->load->helper('url');
$this->load->library('session');
$this->config->item('base_url');
// etc.
```

然而， `$this` 仅能再你的控制器，数据模型，视图中使用。如果你想在你的类中使用 CodeIgniter 类，可以参考以下做法：

首先，将 CodeIgniter 对象赋值给变量：

```
$CI =& get_instance();
```

一旦你将对象赋值给变量，你可以使用变量实例取代 `$this` ：

```
$CI =& get_instance();

$CI->load->helper('url');
$CI->load->library('session');
$CI->config->item('base_url');
// etc.
```

如果在其他类中使用 `get_instance()` ，可以将它赋给一个属性。通过这个方法，你不需要在每个方法中调用 `get_instance()` 。例如：

```
class Example {

    protected $CI;

    // We'll use a constructor, as you can't directly call a function
    // from a property definition.
```

```
    public function __construct()
    {
        // Assign the CodeIgniter super-object
        $this->CI =& get_instance();
    }

    public function foo()
    {
        $this->CI->load->helper('url');
        redirect();
    }

    public function bar()
    {
        $this->CI->config->item('base_url');
    }
}
```

在上述的例子中， `foo()` 和 `bar()` 方法将会在你实例化的子类工作，不需要每次都调用 `get_instance()` 。

25

钩子 – 扩展系统核心

CodeIgniter 的钩子功能，让你可以在不改变系统核心文件的基础上，改变或增加系统的核心运行功能。当 CodeIgniter 运行后，它会产生一个特殊的进程，这个进程在项目文件里有说明。当然，您可以自定义一些动作来替代程序运行过程中的某些阶段。例如，你可以在控制器加载前运行一段脚本，或者在加载后运行，或者你想在其他地方触发脚本。

#

启用钩子

通过设定 application/config/config.php 文件里的选项，钩子功能可以在全局范围内打开或者关闭：

```
$config['enable_hooks'] = TRUE;
```

#

定义钩子

钩子定义在 application/config/hooks.php 里。每个钩子可以用下面格式的数组来定义：

```
$hook['pre_controller'] = array(
    'class'    => 'MyClass',
    'function' => 'Myfunction',
    'filename' => 'Myclass.php',
    'filepath' => 'hooks',
    'params'   => array('beer', 'wine', 'snacks')
);
```

说明:

数组的索引与你要用的钩子名相关。上述的例子中，挂钩点是 `pre_controller` 。挂钩点参数列表如下所示，以下各项将定义在你相关联的钩子数组里：

- class 你希望调用的类。如果想使用过程函数来代替类的话，此项为空。
- function 你想调用的函数或方法的名字
- filename 包含你的类或函数的文件名
- filepath 包含你的脚本的文件夹。注意:你的脚本必须放在 *application/* 下的目录里，所以，文件路径是相对于目录的。例如，如果你的脚本放在 *application/hooks/*里，你可以简单的使用 'hooks' 作为文件路径。如果脚本位于 *application/hooks/utilities/*，你可以把 'hooks/utilities' 作为你的文件路径。注意后面没有 "/"。
- params 你希望传递给脚本的任何参数。这个参数是可选的。

如果你用的是 PHP 5.3+, 你也能使用 lambda/anoymous 函数 (或闭包（ closures ）) 作为钩子:

```
$hook['post_controller'] = function()
{
    /* do something here */
};
```

#

多次调用同一个钩子

如果你在多个脚本中用同一个钩子，最简单的方法就是把你的数组定位成二维的，比如：

```
$hook['pre_controller'][] = array(
    'class'    => 'MyClass',
    'function' => 'MyMethod',
    'filename' => 'Myclass.php',
    'filepath' => 'hooks',
    'params'   => array('beer', 'wine', 'snacks')
);

$hook['pre_controller'][] = array(
    'class'    => 'MyOtherClass',
    'function' => 'MyOtherMethod',
    'filename' => 'Myotherclass.php',
    'filepath' => 'hooks',
    'params'   => array('red', 'yellow', 'blue')
);
```

注意每个数组后的中括号：

```
$hook['pre_controller'][]
```

这样你就可以在多个脚本中使用同一个钩子。你定义的数组顺序就是程序的执行顺序。

#

挂钩点

以下是可用的挂钩点:

- pre_system 系统执行的早期执行，仅仅在 benchmark 和 hooks 类加载完毕的时候可用。而且没有路由或其他进程发生。
- pre_controller 在调用控制器之前可以挂钩。所有的基础类，路由，和安全性检测完成后。
- post_controller_constructor 初始化控制器成功后，以及其他方法调用前挂钩。
- post_controller 在控制器完全执行后调用。
- display_override 覆盖 `_display()` 方法，在系统执行的最后发送页面给浏览器。允许你使用自己的显示方法。注意，你需要使用 `$this->CI =& get_instance()` 引用 CI 的 superobject，然后最终数据可以通过调用 `$this->CI->output->get_output()` 获得。
- cache_override 可以让你调用自己的函数来取代 output 类中的 `_display_cache()` 函数。这可以让你使用自己的缓存显示机制。
- post_system 在最终渲染页面发送到浏览器后，浏览器接收完最终数据的系统末尾调用。

26

自动载入资源

CodeIgniter 的 “自动加载” 功能，可以允许系统每次运行时自动初始化类库，辅助函数和模型。如果你想让某些资源在系统中各处可用，可以考虑使用自动加载功能。

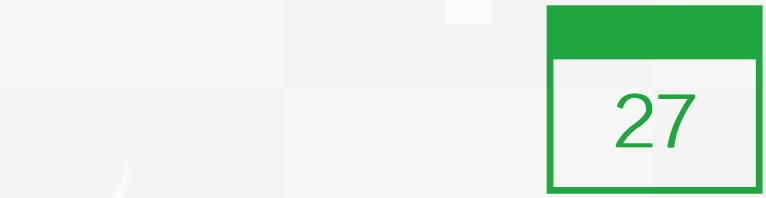
以下内容可以自动加载：

- *libraries/* 文件夹里的类
- *helpers/* 文件夹里的帮助文件
- *config/* 文件夹里的自定义配置文件
- *system/language/* 文件夹里的语言包
- *models/* 文件夹里的模型

要自动加载，打开 application/config/autoload.php 文件并将他们添加到自动加载数组，你会发现该文件中对应于上面每个项目类型。

注意: 将内容添加到自动加载数组，不要添加 (.php) 扩展名。

另外，如果你想要让 CodeIgniter 使用 `Composer` 自动加载，可以设置 `$config['composer_autoload']` 为 `TRUE`，或者 application/config/ 路径。



27

公共函数

CodeIgniter 使用了一些全局定义的函数来完成某些操作，你在任何情况下都可以使用，而不需要加载库或辅助函数。

# #

is_php($version)

- 参数 字符串 `$version` : 版本号
- 返回：如果运行中得 PHP 版本大于等于 version，返回 TRUE，否则返回 FALSE
- 返回类型：bool

检测 PHP 版本是否大于参数的版本号。例如:

```
if (is_php('5.3'))
{
    $str = quoted_printable_encode($str);
}
```

如果安装的 PHP 版本号大于等于参数 version，返回 TRUE，如果小于参数 version，则返回 FALSE。

# #

`is_really_writable($file)`

- 参数 字符串 $file: 文件路径
- 返回：如果路径可写，返回 TRUE，否则返回 FALSE
- 返回类型：bool

在Windows平台，实际上 `is_writable()` 函数在没有文件写权限时也返回 TRUE。那是因为，只有文件是只读属性时，操作系统才向 PHP 报告为 FALSE。这个函数依靠对文件的先行写入来判断是否真的具有写权限。通常情况下，只有在这个信息不可靠的平台上才推荐使用。例如:

```
if (is_really_writable('file.txt'))
{
    echo "I could write to this if I wanted to";
}
else
{
    echo "File is not writable";
}
```

#

config_item($key)

- 参数 字符串 $key: 配置选项键值
- 返回：配置值或者 NULL
- 返回类型：混合

尽管使用 config_item() 函数能够取得单个配置信息，但是 `配置库` 是访问这些信息的优选方式。更多信息请见类库参考。

# #

show_error($message, $status_code[, $heading = 'An Error Was Encountered’])

- 参数 混合 $message: 错误信息
- 参数 整数 $status_code: HTTP 响应状态码
- 参数 字符串 $heading: 错误页面头
- 返回类型：void

这个函数调用 CI_Exception::show_error() 。更多细节参考文档“错误处理”文档。

# #

show_404([$page = '', $log_error = TRUE])

- 参数 字符串 $page: URI 字符串
- 参数 bool $log_error: 是否将错误写入日志
- 返回类型：void

这个函数调用 CI_Exception::show_404() 。更多细节参考文档“错误处理”文档。

# #

`log_message($level, $message)`

- 参数 字符串 $level: 日志级别：'error', 'debug' 或 'info'
- 参数 字符串 $message: 写入日志的消息
- 返回类型：void

这是 `CI_Log::write_log()` 函数的别名。更多细节参考文档“错误处理”文档。

# #

set_status_header($code[, $text = ‘’])

- 参数 整数 $code: HTTP 响应状态码
- 参数 字符串 $text: 状态码伴随的消息
- 返回类型：void

允许你手工设置服务器状态头，例如：

```
set_status_header(401);
// Sets the header as:  Unauthorized
```

# #

`remove_invisible_characters($str[, $url_encoded = TRUE])`

- 参数 字符串 $str: 输入字符串
- 参数 bool $url_encoded: 是否移除 URL-encoded 字符串
- 返回：过滤后的字符串
- 返回类型：string

这个函数不允许在 ASCII 字符间插入 NULL 字符，比如 `Java\\0script` 。例如：

```
remove_invisible_characters('Java\\0script');
// Returns: 'Javascript'
```

# #

`html_escape($var)`

- 参数 混合 $var: 需要转义的字符串或数组
- 返回：转义过的 HTML 字符串
- 返回类型：混合

这是原生 `htmlspecialchars()` 函数的别名，它能接受字符串数组。有助于防止跨站脚本攻击（XSS）。

#

`get_mimes()`

- 返回：和文件类型相关的数组
- 返回类型：数组

这个函数返回 *application/config/mimes.php* 里的 MIMEs 数组的引用。

# #

`is_https()`

- 返回：如果当前使用 HTTP-over-SSL，返回 TRUE，否则返回 FALSE
- 返回类型：bool

如果使用安全连接（HTTPS），返回 TRUE，否则返回 FALSE

# #

`is_cli()`

- 返回：如果当前运行环境是 CLI，返回 TRUE，否则返回 FALSE。
- 返回类型：bool

如果应用程序运行在命令行，返回 TRUE，否则返回 FALSE。这个函数同时检查 `PHP_SAPI` 是否是 'cli'，或者如果 `STDIN` 常量已经定义过。

# #

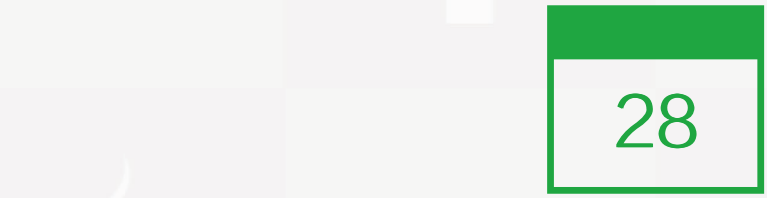
`function_usable($function_name)`

- 参数 字符串 $function_name：函数名
- 返回：如果函数可用返回 TRUE，否则返回 FALSE
- 返回类型：bool

如果一个函数存在并且可用，返回 TRUE，否则返回 FALSE。 这个使用 `function_exists()` 函数检查，并且如果已经加载 `Suhosin extension` ，检查是否它有没有关闭这个函数的检查功能。

如果你想检查诸如 `eval()` 和 `exec()` 函数是否可用，这个函数非常有用。这个函数也非常的危险，因此在高度严格的安全策略服务器可能被禁用。

注意：此功能被引入，是为了 Suhosin 终止脚本运行，但事实证明这是一个错误。这个错误修复了有一段时间了（ 版本 0.9.34 ），但是很遗憾它还没发布。



28

兼容性函数

CodeIgniter 提供了一套兼容性函数，让你可以使用非原生的 PHP 函数，但是只有在更高的版本或者依赖于某个扩展插件。

作为定制化实现，这些函数也有一定的依赖性，但是如果你的安装的 PHP 没有提供原生函数，它们还是很有用的。

注意: 大部分和 `通用函数` 很像, 只要依赖性满足，兼容性函数一直可用。 `<div class="custom-index container"></div>`

# #

哈希密码

这套函数提供一个 PHP 标准“哈希密码扩展” 的 “反向移植”，仅在 PHP 5.5 版本之后可用。

#

依赖性

- PHP 5.3.7
- `CRYPT_BLOWFISH` 支持 `crypt()`

#

常量

- `PASSWORD_BCRYPT`
- `PASSWORD_DEFAULT`

#

函数参考

#

`password_get_info($hash)`

- 参数 字符串 $hash: 哈希密码
- 返回：哈希过的密码信息
- 返回类型：数组

更多信息，可以参考 `PHP password_get_info() 使用手册`

#

`password_hash($password, $algo[, $options = array()])`

- 参数 字符串 $password: 文本密码
- 参数 整数 $algo: Hashing 算法
- 参数 数组 $options: Hashing 参数
- 返回 哈希过的密码，失败时返回 FALSE
- 返回类型 字符串

更多信息，可用参考 `PHP password_get_info() 使用手册` 。

注意: 除非你提供自己的（有效的）salt，这个函数可用进一步的提供依赖于可用的 CSPRNG 源。满足底下的每个条件:

```
- ``mcrypt_create_iv()`` with ``MCRYPT_DEV_URANDOM``
- ``openssl_random_pseudo_bytes()``
- /dev/arandom
- /dev/urandom
```

#

`password_needs_rehash()`

- 参数 字符串 $hash: 哈希密码
- 参数 整数 $algo: 哈希算法
- 参数 数组 $options: 哈希参数
- 返回 TRUE 如果哈希符合给定的算法和参数，返回 TRUE，否则返回 FALSE
- 返回类型 bool

更多信息，可以参考 `PHP password_needs_rehash() 使用手册` 。

#

`password_verify($password, $hash)`

- 参数 字符串 $password: 纯文本密码

- 参数 字符串 $hash: 哈希密码
- 返回 TRUE 如果密码和哈希匹配返回 TRUE，否则返回 FALSE。
- 返回类型 bool

更多信息，可以参考 PHP password_verify() 使用手册

# #

哈希 (消息摘要)

这个兼容性层包含了 `hash_equals()` 和 `hash_pbkdf2()` 函数的公钥，除此之外还分别要求 PHP 5.6 和/或 5.5。

#

依赖

- 无

#

函数参考

#

`hash_equals($known_string, $user_string)`

- 参数 字符串 $known_string: 已知字符串
- 参数 字符串 $user_string: 用户提供的字符串
- 返回 如果字符串匹配返回 TRUE，否则返回 FALSE
- 返回类型 字符串

更多信息，可以参考 `PHP hash_equals() 使用手册` 。

#

`hash_pbkdf2($algo, $password, $salt, $iterations[, $length = 0[, $raw_output = FALSE]])`

- 参数 字符串 $algo: 哈希算法
- 参数 字符串 $password: 密码

- 参数 字符串 $salt: 哈希 salt
- 参数 整数 $iterations: 迭代过程中执行的次数
- 参数 整数 $length: 输出字符串的长度
- 参数 bool $raw_output: 是否返回原始二级制数据
- 返回 成功返回加密过的值，失败返回 FALSE
- 返回类型 字符串

更多信息，可以参考 `PHP hash_pbkdf2() 使用手册` 。

# #

多字节字符串

这套兼容性函数提供为多字节字符串扩展提供有限支持。因为有限的替代方法，只有几个函数可用。

注意: 当一个字符参数忽略，可以使用 `$config['charset']` 。

# #

依赖性

- iconv 扩展

注意: 这个依赖性是可选的，这个函数总是被声明。如果 iconv 不可用，他们将会回退到 non-mbstring 版本。

注意: 当提供字符设置时，必须通过 iconv 和它认识的格式来支持。

注意: 你再检查 mbstring 扩展时，可以使用 `MB_ENABLED` 常量。

#

函数参考

#

`mb_strlen($str[, $encoding = NULL])`

- 参数 字符串 $str: 输入字符串
- 参数 字符串 $encoding: 字符集
- 返回 输入字符串的字符数，失败返回 FALSE
- 返回类型 字符串

更多信息，可以参考 PHP mb_strlen() 使用手册。

#

`mb_strpos($haystack, $needle[, $offset = 0[, $encoding = NULL]])`

- 参数 字符串 $haystack: 需要搜索的字符串
- 参数 字符串 $needle: 需要搜索部分字符串
- 参数 整数 $offset: 搜索偏移量
- 参数 字符串 $encoding: 字符集
- 返回 $needle 字符所在的位置，失败返回 FALSE
- 返回类型 混合

更多信息，可以参考 PHP mb_strpos() 使用手册。

#

`mb_substr($str, $start[, $length = NULL[, $encoding = NULL]])`

- 参数 字符串 $str: 输入字符串
- 参数 整数 $start: 第一个字符的位置

- 参数 整数 $length: 字符的最大数量
- 参数 字符串 $encoding: 字符集
- 返回 返回从 $start 开始，长度为 $lengthPortion 的字符串，失败返回 FALSE。
- 返回类型 字符串

更多信息，可以参考 PHP mb_substr() 使用手册。

#

标准函数

这套兼容性函数提供一些标准的 PHP 函数支持，不过需要新版本的 PHP。

#

依赖性

- 无

#

函数参考

#

`array_column(array $array, $column_key[, $index_key = NULL])`

- 参数 数组 $array: 取结果的源数组
- 参数 混合 $column_key: 取结果的列的 key
- 参数 混合 $index_key: 返回值的 key
- 返回 从多维数组中取回某列数组
- 返回类型 数组

更多信息，可以参考 PHP array_column() 使用手册。

#

`array_replace(array $array1[, ...])`

- 参数 数组 $array1: 将要替换的数组
- 参数 array ...: 需要提前内容的数组

- 返回 修改后的数组
- 返回类型 数组

更多信息，可以参考 PHP array_replace() 使用手册。

#

`array_replace_recursive(array $array1[, ...])`

- 参数 数组 $array1: 将要替换内容的数组
- 参数 array...: 将要提取内容的数组
- 返回 修改后的数组
- 返回类型 数组

更多信息，可以参考 PHP array_replace_recursive() 使用手册。

注意: 只有 PHP 原生函数可以检测无穷递归。除非你的 PHP 版本是 5.3+,否则你要消息使用。

#

`hex2bin($data)`

- 参数 数组 $data: 16 进制的数据
- 返回 参数的二级制表示
- 返回类型 字符串

更多信息，可以参考 PHP hex2bin() 使用手册。

#

`quoted_printable_encode($str)`

- 参数 字符串 $str: 输入字符串
- 返回 8bit-encoded 字符串
- 返回类型 字符串

更多信息，可以参考 PHP quoted_printable_encode() 使用手册。

29

URI 路由

一般来说，URL 字符串和相应的控制器类/方法一一对应。URI里的内容通常都是这个模式：

```
example.com/class/function/id/
```

然而在一些例子中，你也许想要重新隐射这个关系来调用一个不同的类/方法（class/function),而不是与 URL 一一对应。

例如, 比如你想要 URL 长成这样:

```
example.com/product/1/
example.com/product/2/
example.com/product/3/
example.com/product/4/
```

通常 URL 的第二段表示方法名，但是上面的列子第二段表示产品 ID。为了完成这个目标，CodeIgniter 允许你重新映射。

#

设置你自己的路由规则

路由规则定义在 *application/config/routes.php* 文件里。在这个文件里，你可以找到一个名为 $route 的数组，它可以让你定义自己的路由规则。定义可以用两种方法：通配符和正则表达式。

#

通配符

一个典型的通配符通常长成这样：

```
$route['product/:num'] = 'catalog/product_lookup';
```

在一个路由中，数组中的键需要匹配的 URI，而数组值包含将被重定向的目的地。在上述的例子中，如果在 URL 第一段中有 "product" ，第二段中是数字，将会用 "catalog" 类和 "product_lookup" 方法替换。

你可以匹配文字的值或者使用以下两种通配类型：

(:num) 将匹配一个只包含数字的段

(:any) 将会匹配一个包含任何字符的段（除了 '/' 字符，因为它是段分割器）

注意: 通配符实际是常规表达式的别名，:any 被翻译成 [^/]+，:num 被翻译成 [0-9]+。

注意: 路由将会按照定义的顺序来运行.高层的路由总是优先于低层的路由.

注意: 路由规则不是过滤器！例如：设定规则 'foo/bar/(:num)' 不会阻止控制器 *Foo* 和方法 bar* 被直接调用。

#

例子

以下是一些路由的例子：

```
$route['journals'] = 'blogs';
```

如果 URL 的第一个分段（类名）是关键字 “journals”，将会被定向到 "blogs" 类。

```
$route['blog/joe'] = 'blogs/users/34';
```

如果 URL 前两个分段是 "blog" 和 "joe"，那么将会重定向到 "blogs" 类的 "users" 方法中处理。并将 ID "34" 设为参数.

```
$route['product/(:any)'] = 'catalog/product_lookup';
```

如果 URL 第一段是 "product"，第二段是任意字符，它将会被映射到 "catalog" 类的 "product_lookup" 方法。

```
$route['product/(:num)'] = 'catalog/product_lookup_by_id/$1';
```

如果 URL 第一段是 "product"，第二段是任意数字，它将会被映射到 "catalog" 类的 "product_lookup_by_id" 方法，参数为 $1。

注意: 不要在前面或后面加 "/"。

#

正则表达式

你可以用正则表达式来自定义你的路由规则，任何有效的正则表达式都是运行的，甚至逆向引用。

注意: 如果你使用逆向引用，需要用双反斜线语法替换卫美元符语法（ \1 替换为 $1 ）

一个典型的正则表达式通常长成这样：

```
$route['products/([a-z]+)/(\d+)'] = '$1/id_$2';
```

上例中，类似 `products/shirts/123` 的 URI，将会换成调用 "shirts" 控制器类和 "id_123" 方法。

使用正则表达式，你也可以获取斜线('/')中的段，这个代表中间多段的分隔符。

例如，如果一个用户访问你的 web 应用的被保护区域的密码，你希望他们在登陆后重新定向到同一个页面，你可以参考以下例子：

```
$route['login/(.+)'] = 'auth/login/$1';
```

如果你不了解正则表达式，并想学习，可以参考 regular-expressions.info.

注意: 你也可以混合使用正则表达式混合通配符。

#

回调

如果你的 PHP 版本 >= 5.3，你可以使用回调函数来取代一般的路由规则来处理 back-references，例如：

```
$route['products/([a-zA-Z]+)/edit/(\d+)'] = function ($product_type, $id)
{
    return 'catalog/product_edit/' . strtolower($product_type) . '/' . $id;
};
```

#

路由中使用 HTTP 动作

可以使用 HTTP 动作（请求方法）来重新定义你的路由规则。当你建立 RESTful 应用的时候特别有用。你可以使用标准的 HTTP 动作（GET, PUT, POST, DELETE, PATCH），或自定义动作（比如 PURGE）。HTTP 动作 规则大小写敏感。所有你需要做的事情就是添加动作数组 key 到你的路由中。例如:

```
$route['products']['put'] = 'product/insert';
```

在上述的例子中，PUT 请求 URI "products" 将会调用 `Product::insert()` 控制器方法.

```
$route['products/(:num)']['DELETE'] = 'product/delete/$1';
```

DELETE 请求到 URL，第一段是 "products"，第二段数字将会映射到 `Product::delete()` 方法，传入数字到第一个参数上。

使用 HTTP 动作是可选的。

#

保留的路由

有三个保留的路由：

```
$route['default_controller'] = 'welcome';
```

这个路由指定在 URI 里没有任何数据时，加载哪个控制器，大家载入根 URL 时就是这个情况。在上述的例子中，"welcome" 类将会被载入。你要尽量有一个预设路由，否则预设会出现一个 404 页面。

```
$route['404_override'] = '';
```

如果请求的控制器没有找到，这个路由将会指示加载哪个控制器。它将会重写默认的 404 错误页面。它不会影响 `show_404()` 函数，这个函数将会继续加载默认的 *application/views/errors/error_404.php* 里的 *error_404.php* 文件。

```
$route['translate_uri_dashes'] = FALSE;
```

很显然这是 boolean 值，这不是真的路由。这个选项让你自动的在控制器和方法中 URI 片段将 '–' 替换成下划线，从而让你节省更多的路由项目。这是必须的，因为破折号不是一个有效的类或方法名，如果你使用破折号，将会导致重大错误。

注意: 保留的路由必须在任何通配符或正则表达式前定义。

30

错误处理

CodeIgniter 允许你在应用中使用以下的函数建立错误报告。另外，它有一个错误日志类，允许错误和调试信息保存为文本文件。

注意:默认情况下，CodeIgniter 显示所有 PHP 错误。你可能想要在开发完成后改变这个行为。你可以在 index.php 顶部找到 `error_reporting()` 函数。即使禁用错误报告，发生错误时，错误日志也不回停止。

CodeIgniter 和其他系统不太一样，错误报告函数是一个简单的程序接口，可以在整个应用程序里使用。不用考虑类或者是函数的范围，这种办法可以直接触发错误通知。

无论系统核心何时调用 `exit()`， CodeIgniter 也会返回一个状态码。退出状态码独立于 HTTP 状态码，这个服务监视其他应用程序是否成功的执行完成，如果没有成功，是什么原因导致的。这些值定义在 *application/config/constants.php*。退出状态码在 CLI 设置中很有用，返回的状态码能让服务器软件追踪脚本代码，让你的程序更健壮。

以下函数能让你产生错误：

# #

`show_error($message, $status_code, $heading = 'An Error Was Encountered’)`

- 参数 混合 $message: 错误消息
- 参数 整数 $status_code: HTTP 状态码
- 参数 字符串 $heading: 错误页面头
- 返回类型: void

这个函数将会使用以下模板来显示错误信息：

```
application/views/errors/html/error_general.php
```

或

```
application/views/errors/cli/error_general.php
```

可选参数 `$status_code` 确定将会发生什么 HTTP 状态码到错误。如 `$status_code` 少于 100，HTTP 状态码将会设置为 500，退出状态码将会设置为：`$status_code + EXIT__AUTO_MIN` 。如果这个值大于 `EXIT__AUTO_MAX` ，或者如果 `$status_code` 为大于等于 100 的值，退出码将会设置为 `EXIT_ERROR` 。更多细节参考 *application/config/constants.php*。

#

show_404($page = '', $log_error = TRUE)

- 参数 字符串 $page: URI 字符串
- 参数 bool $log_error: 是否将错误写入日志
- 返回类型: void

这个函数将会用下列的模板显示 404 错误信息：

```
application/views/errors/html/error_404.php
```

或

```
application/views/errors/cli/error_404.php
```

这个函数希望传入的字符串是没有找到的文件路径。退出状态码将会设置为 EXIT_UNKNOWN_FILE 。注意如没有找到控制器 CodeIgniter 将会自动的显示 404 消息。CodeIgniter 自动将任何 show_404() 调用写入日志。将第二个参数设置为 FALSE，将会跳过日志。

#

log_message($level, $message, $php_error = FALSE)

- 参数 字符串 $level: 日志级别 'error', 'debug' 或 'info'
- 参数 字符串 $message: 写入日志的消息
- 参数 bool $php_error: 是否将原生的 PHP 错误写入日志
- 返回类型: void

这个函数能让你将消息写入到日志。你必须提供第一个参数的三个日志级别之一，它表示写的日志是哪种消息，第二个参数就是要写入的消息。例如：

```
if ($some_var == '')
{
    log_message('error', 'Some variable did not contain a value.');
}
else
{
    log_message('debug', 'Some variable was correctly set');
}

log_message('info', 'The purpose of some variable is to provide some value.');
```

有三种消息类型:

- 错误消息. 比如 PHP 错误或用户错误
- 调试信息. 辅助调试的信息。例如，如果一个类已经初始化，你可以记录这个信息。
- 普通消息. 这是最低级别的消息，只是简单了提供了一些运行时的信息

注意: 想要写日志，需要确保 *logs/* 文件夹可写。另外，必须设置 *application/config/config.php* 里的 "threshold"。例如，你可以只写错误日志，而其他两种日志不写。如果你设置为 0，将禁用日志。

31

web 页面缓存

CodeIgniter 可以缓存网页，这样可以最大化性能。

虽然 CodeIgniter 已经足够快，但是网页中得动态内容，主机的内存 CPU 和数据库读取速度等因素直接影响了网页的加载速度。通过网页缓存，你的网页可以达到静态页面的加载速度，因为已经将所有输出都保存好了。

# #

缓存是如何工作的?

CodeIgniter 允许单页缓存，你也可以设置缓存时间。当页面第一次加载时，缓存文件将会保存到 `application/cache` 文件夹里。下次访问的时候，将会提取缓存页面并发送到用户的浏览器中。如果过期，它将会被删除并重新生成。

注意: Benchmark 标签在缓存的页面依然有效。

# #

允许缓存

将下面的代码放到你的控制器方法中就可以启用缓存。

```
$this->output->cache($n);
```

参数 `$n` 就是缓存失效的分钟数。

以上的代码可以放到任何一个方法中。先后的顺序不会影响，所以你可以自己定制位置。一旦标签设置好，你的页面就开始缓存。

> 注意: 由于 CodeIgniter 存储缓存文件的方式，只有通过 view 文件的输出才能被缓存。
>
> 注意: 如果你改变了配置选项将会影响输出，你需要手工的删除缓存文件。
>
> 注意: 在保存缓存文件之前，请确保 application/cache 文件夹可写。

#

删除缓存

如果你不再需要缓存，可以删除标签，这样它再失效后就不再缓存。

> 注意: 删除标签并不会立即删除缓存文件。它将会在失效后才会删除。

如果你需要手工删除缓存，可以使用 `delete_cache()` 方法。

```
// Deletes cache for the currently requested URI
$this->output->delete_cache();

// Deletes cache for /foo/bar
$this->output->delete_cache('/foo/bar');
```

32

应用性能分析

这个分析器类将会显示基准结果，运行的查询，并将 `$_POST` 数据放在你的页尾。这个信息在开发中非常有用，它能帮你调试和优化。

# #

初始化类

注意: 这个类不需要初始化。如果已按照下面的方式激活,他将被输出类自动装载。

# #

启动分析器

在控制器中设置以下方法,可以启动该分析器。

```
$this->output->enable_profiler(TRUE);
```

分析器启动后将产生一个报告并插入您的页面底部.

使用以下方法可以禁用分析器:

```
$this->output->enable_profiler(FALSE);
```

# #

设置基准点

为了让分析器编译并显示你的基准数据，你必须特定的语法命名基准点。

更多细节可以参考文档“基准库”

# #

启用和禁用分析器的段

可以通过设置相应的控制变量 TRUE 或 FALSE 来启用或禁用分析数据中得每个字段。它可以通过下面两种方法之一来实现。其中一个方法是你可以在 *application/config/profiler.php* 配置文件里设置整个程序的全局默认值。例如：

```
$config['config']        = FALSE;
$config['queries']       = FALSE;
```

在你的控制器中，你可以通过调用 `set_profiler_sections()` 方法来重写默认值和配置文件值。

```
$sections = array(
   'config'  => TRUE,
   'queries' => TRUE
);

$this->output->set_profiler_sections($sections);
```

下表列出了可用的分析器数据字段和用来访问这些字段的key。

默认键值	描述	默认值
benchmarks	在各个计时点花费的时间以及总时间	TRUE
config	CodeIgniter 配置变量	TRUE
controller_info	被调用的method及其所属的控制器类	TRUE
get	在request中传递的所有 GET 参数	TRUE
http_headers	本次请求的 HTTP 头	TRUE
memory_usage	本次请求消耗的内存（byte 为单位）	TRUE
post	在request中传递的所有POST参数	TRUE
queries	列出执行的数据库操作语句及其消耗的时间	TRUE
uri_string	本次请求的URI	TRUE
session_data	数据存储在当前 session	TRUE
query_toggle_count	指定显示多少个数据库查询语句，剩下的则默认折叠起来。	25

注意: 在你的数据库配置中禁用 `保存查询` 文档设置，将会有效的禁用数据库查询性能分析。你可以使用 `$this->db->save_queries = TRUE;` 重写这个设置。没有这个设置，你无法查看查询或者 `last_query` 。

33

通过 CLI 执行 CodeIgniter

除了通过浏览器 URL 调用控制器外，也可以通过命令行接口（CLI）调用。

# #

什么是 CLI?

命令行接口是一种基于文本的和计算机交互的方式。更多信息参考维基百科文章。

# #

为什么使用命令行运行?

这里有很多原因通过命令行运行 CodeIgniter，不过总是被忽略。

- 使用 cron 定时运行任务而不需要使用 wget 或 curl
- 通过检查 `$this->input->is_cli_request()` 让你的 cron 任务无法通过网址访问到
- 让交互式任务可以做设置权限、清空缓存、执行备份等操
- 与其他语言进行集成。比如一个 C++ 脚本可以调用一条指令来运行你模型中的代码!

# #

让我们试试: Hello World!

让我们创建一个简单的控制，这样你可以看到他时如工作的。使用你的文本编辑器，创建一个文件 Tools.php，加入以下代码：

```
<?php
class Tools extends CI_Controller {

    public function message($to = 'World')
    {
        echo "Hello {$to}!".PHP_EOL;
    }
}
```

将文件保存到 *application/controllers/* 文件夹。

现在正常情况下你可以通过你的网站的 URL 来访问它：

```
example.com/index.php/tools/message/to
```

除此之外，我们也可以在 mac/linux 中打开终端，或在 Windows 中运行 “cmd”，并进入我们的 CodeIgniter 项目的目录。

```
$ cd /path/to/project;
$ php index.php tools message
```

如果你做的没问题，将会看到 *Hello World!*。

```
$ php index.php tools message "John Smith"
```

这里我们像使用 URL 参数一样给它传递了一个参数。“John Smith” 参数被传入，并且输出也变成：

```
Hello John Smith!
```

# #

就是这样!

简单的来说，这就是全部你需要知道的有关命令行中使用控制器的事情了。记住这只是一个普通的控制器，所以路由和 _remap 也照样工作。

34

管理你的应用程序

默认情况下，你仅会用 CodeIgniter 管理一个应用程序，这个程序位于 *application/* 文件夹中。当然，也有可能多个程序共享一个 CodeIgniter，甚至对 *application/* 重命名或更换路径。

# #

重命名应用文件夹

如果你想重命名应用文件夹，你可以打开 index.php 文件，并使用变量 `$application_folder` 设置它的名字。

```
$application_folder = 'application';
```

# #

更改应用文件夹路径

你可以将应用文件夹移动到服务器上不同的位置，而不是根目录。打开 index.php 并设置 `$application_folder` 变量为服务器的完整路径：

```
$application_folder = '/path/to/your/application';
```

# #

在一个 CodeIgniter 下运行多个应用

如果你想要多个应用程序共享一个 CodeIgniter， 可以将 application 下所有的文件夹放在不同的应用程序的文件夹内。

例如，你要建立两个应用程序 "foo" 和 "bar",你的应用程序文件夹的结构可能如下:

```
applications/foo/
applications/foo/config/
applications/foo/controllers/
applications/foo/libraries/
applications/foo/models/
applications/foo/views/
applications/bar/
applications/bar/config/
applications/bar/controllers/
applications/bar/libraries/
applications/bar/models/
applications/bar/views/
```

要选择某个应用程序，需要打开 index.php 文件，并设置 `$application_folder` 变量。例如，选择使用 "foo" 程序，可以这么做:

```
$application_folder = 'applications/foo';
```

注意: 每个应用程序都需要自己的 index.php 文件，它会调用相应的应用。你可以任意命名 index.php。

35

处理多环境

开发者通常希望在开发环境和生产环境有不同的行为。例如错误信息在开发中有用，而在项目上线后者可能会造成一些安全问题。

# #

ENVIRONMENT 常量

默认情况下，CodeIgniter 把环境常量 `$_SERVER['CI_ENV']` 设置为 'development'，在 index.php 的顶部，你会看到：

```
define('ENVIRONMENT', isset($_SERVER['CI_ENV']) ? $_SERVER['CI_ENV'] : 'development');
```

这个服务器变量可在 .htaccess 文件中设置，或者 Apache 使用 `SetEnv` 设置。这个方法对于 nginx 或其他方法有效，或者你可以整个移除这个逻辑，并根据服务器 IP 设置常量。

除了影响基本框架的行为（参见下个章节），你可以在你的开发环境中使用这个常量，以便区别于不同的环境。

# #

对默认框架行为的影响

CodeIgniter 系统中有哪些地方使用了 ENVIRONMENT 常量。这个部分描述了默认系统框架行为如何受到影响。

#

错误报告

将环境变量设置为 'development' ，会让 PHP 错误都输出到浏览器。相反，如果设置为 'production'，将会禁用错误输出。在产品中禁止错误输出是一个不错的安全策略。

#

配置文件

可选的，你可以让 CodeIgniter 加载特定的环境配置文件。这可能会对管理多环境使用不同 API 密钥这样的事情很有用。这在文档配置类“环境”一节有详细的说明。

36

视图文件的 PHP 替代语法

如果你不使用 CodeIgniter 的模板语法，你可以在视图文件中使用原始的 PHP 代码。要使这些文件里的 PHP 代码最小化，并让他们容易辨认，建议你使用 PHP 替代语法，来控制结构和短标签 echo 语句。如果你不熟悉这个语法，下面内容将会让你消灭大括号和 "echo" 语句。

# #

自动短标签支持

注意: 如果你发现本页描述语法在你的服务器上不能工作，可能是因为你的 PHP ini 文件禁用了 "short tags"。CodeIgniter 将会选择性的重写，运行你使用语法，及时你的服务器不支持。可以在 *config/config.php* 文件中打开这个特性。

请注意，如果你使用这个特性，如果 PHP 错误在你的视图文件中出现，错误消息和行数不会准确的出现。相反，所有的错误将会展示为 `eval()` 错误。

# #

替代 Echo

echo，或者打印一个变量，可以这么写：

```
<?php echo $variable; ?>
```

使用替换语法，你可以这么写：

```
<?=$variable?>
```

# #

替代控制结构

控制结构，像 if，for，foreach，和 while 也可以写成简化的形式。这里是一个用 foreach 的例子：

```
<ul>

<?php foreach ($todo as $item): ?>

    <li><?=$item?></li>

<?php endforeach; ?>

</ul>
```

注意，这里没有大括号，它被 `endforeach` 替换。每个上述的控制结构拥有相同结束语法： `endif` ，`endfor` ，`endforeach` , 和 `endwhile` .

同时也需要注意，每个结构以后不用分号（除了最后一个），用冒号，这很重要！

这有另一个例子，使用 `if` / `elseif` / `else` . 注意冒号 ::

```
<?php if ($username === 'sally'): ?>

    <h3>Hi Sally</h3>

<?php elseif ($username === 'joe'): ?>

    <h3>Hi Joe</h3>

<?php else: ?>

    <h3>Hi unknown user</h3>

<?php endif; ?>
```

37

安全

本章描述了 Web 安全的“最佳实践”，详细说明了 CodeIgniter 的内部安全特性。

# #

URI 安全

CodeIgniter 严格限制 URI 中所包含的字符，让你设计的程序减少被恶意数据入侵的可能。URI 一般只包含以下内容：

- 字母和数字 (仅拉丁字符)
- 波浪号: ~
- 百分号: %
- 点: .
- 冒号: :
- 下划线: _
- 减号: -
- 空格

# #

`Register_globals`

系统初始化的时候，所有的全局变量都被 unset，除了那些 `$_GET`，`$_POST`，`$_REQUEST` and `$_COOKIE` 数组中得内容。实际上 unsetting 实例程序与register_globals = off 作用相同。

# #

display_errors

在生产环境中，通常都通过设置 *display_errors* 值为 0 ，来禁用 PHP 的错误报告。它能禁用 PHP 错误输出，它可能包含敏感信息。

设置 index.php 里的 ENVIRONMENT 常量为 'production'，将会关闭这些错误。在开发模式中，还是尽量用 'development'。更多开发环境和生产环境的细节参考“处理环境”文档。

# #

`magic_quotes_runtime`

在系统初始化时， `magic_quote_runtime` 指令被关闭，以便在数据库检索数据时不必去掉反斜线。

# #

最佳实践

在接收任何数据到你的程序前，不管是表单提交的 POST 数据，COOKIE 数据，URI 数据、XML-RPC 数据、还是 SERVER 数组中的数据，我们都推荐你实践下面的三个步骤：

- 过滤不良数据.
- 验证数据以确保正确的类型, 长度, 大小等. (有时这一步也可取代第一步)
- 在提交数据到数据库前转码

CodeIgniter 提供了以下函数和提示来帮助你完成这个过程：

#

XSS 过滤

CodeIgniter 带有一个跨站脚本过滤器，这个过滤器会查找那些常用手段嵌入到你数据中恶意的Javascript，或其它一些试图欺骗 cookie 做其它恶意事情的代码。 XSS Filter 的详细描述在参见安全性文档。

注意: XSS 过滤仅在输出时起作用。过滤输入数据可能会修改原始数据，这并不是我们想要的，比如从密码中剥离特殊字符，我们宁愿减少安全性而不是替换它。

# #

CSRF 保护

CSRF 表示跨站伪造请求（Cross-Site Request Forgery）,攻击者欺骗受害人到一个他自己不知道的地方提交请求。

CodeIgniter 提供 CSRF 保护立即可用，它会自动触发每个非 Get HTTP 请求，但是也要求你以某种形式创建自己的提交表单。详情参见文档“安全库”。

#

密码处理

在应用中，密码处理是非常关键的。

不幸的是，很多开发者并不知道如何处理，很多网页都过时了，或者充满错误。

我们将会给你的可以做和不可以做的列表来帮助你：

- **不要**将密码存储为文本格式，必须**哈希**密码。
- **不要**使用 Base64 或类似的编码来存储密码。这和以文本格式存储密码一样。一定要**哈希**，而不是编码。编码和加密是两种处理方法。密码必须是本人才能知道，因此只能这么做。哈希算法是不可逆的。
- **不要**使用弱哈希算法，例如 MD5 或者 SHA1。这些算法已经过时，并且是有缺陷的，所以不是好的密码哈希的算法。同时，**不要**发明你自己的哈希算法。必须使用强哈希算法，比如 BCrypt，它是 PHP 自己的哈希函数。请使用他们，即使你的 PHP 版本不是 5.5+，CodeIgniter 为你提供的版本最低是 5.3.7（如果你的版本达不到要求，请升级）。如果你不能升级到新版本，请使用 hash_pbkdf() 函数，它也提供了兼容算法。
- **不要**以明文的形式发送密码！即使对于密码拥有者，如果使用了“忘记密码”功能，重新生成一个随机的一次性密码，将它发送给用户。
- **不要**给密码设置不必的限制。如果你使用的哈希算法不是 BCrypt（它限制长度为 72 位），你必须将密码设置一个相对多一些位数，以抵制 Dos 攻击，比如 1024 字节。不必限制密码必须是字符或者数字，它可以包含特殊字符。

# #

验证输入数据

CodeIgniter 中得“表单验证库”可以帮助你验证，过滤，并准备数据。

即使这些东西不起作用，你也要验证并净化所有输入数据。例如，如果你希望一个输入必须是数字，你可以使用 `is_numeric()` 或 `ctype_digit()` 来检查。

需要注意，这个检查不仅对于 `$_POST` 和 `$_GET` 变量，同时也要检查 cookies，user-agent 字符串和那些并不是你的代码创建的基础数据。

# #

插入数据库前转义所有数据

转义前的数据不要插入到数据库，更多细节参考文档**数据库查询**。

#

隐藏你的文件

另一个安全的实践是，在你的服务器的根目录里仅保留 *index.php* 和 "资源"（比如 .js css 和图片文件）（一般命名为 "htdocs/"）。这些文件都是访问网站所需的文件。

让你的访问者可以看到任何东西，他们可能会访问到敏感数据，执行脚本等。

如果你不禁止这么做，你可以尝试使用 .htaccess 文件来限定访问这些资源。

CodeIgniter 在所有的文件将会拥有一个 index.html 文件，可以试着隐藏一部分，不过你要注意，这些都不能阻止高级攻击者。

38

PHP 开发规范

本章主要描述了开发 CodeIgniter 框架本身所采用的编码风格。建议在你的代码中也使用这些规范。

# #

文件格式

文件需要保存为 Unicode (UTF-8)。不应该使用 BOM，与 UTF-16 和 UTF-32 不同，UTF-8 编码的文件不需要指明字节序，而且 BOM 在 PHP 中会产生预期之外的输出，阻止了应用程序设置它自己的头信息。应该使用 Unix 格式的行结束符(LF)。

以下是在一些常见的文本编辑器中更改这些设置的方法。针对你的编辑器，方法也许会有所不同；请参考你的编辑器的说明

#

TextMate

- 打开 Application Preferences
- 点击 Advanced, 点击 "Saving" 栏
- 在 "File Encoding", 选择 "UTF-8 (推荐)"
- 在 "Line Endings", 选择 "LF (推荐)"
- *可选* 检查 "Use for existing files as well" ，如果你想修改文件结束符，打开你的新参考。

#

BBEdit

- 打开 the Application Preferences
- 选择左边 "Text Encodings"
- 在 "Default text encoding for new documents", 选择 "Unicode (UTF-8, no BOM)"
- *可选:* 在 "If file's encoding can't be guessed, use", 选择 "Unicode (UTF-8, no BOM)"
- 选择左侧 "Text Files".
- 在 "Default line breaks", 选择 "Mac OS X and Unix (LF)"

# #

PHP 闭合标签

PHP 闭合标签 “?>” 在 PHP 中对 PHP 的分析器是可选的。但是，如果使用闭合标签，任何由开发者，用户，或者 FTP 应用程序插入闭合标签后面的空格都有可能会引起多余的输出、php 错误、之后的输出无法显示、空白页。因此，所有的 php 文件应该省略这个 php 闭合标签，并插入一段注释来标明这是文件的底部并定位这个文件在这个应用的相对路径。这样有利于你确定这个文件已经结束而不是被删节的。

# #

命名文件

类文件命名采用首字母大写，而其他文件命名（配置，视图，脚本等）必须全是小写

错误的:

```
somelibrary.php
someLibrary.php
SOMELIBRARY.php
Some_Library.php

Application_config.php
Application_Config.php
applicationConfig.php
```

正确的:

```
Somelibrary.php
Some_library.php

applicationconfig.php
application_config.php
```

另外，类文件名必须和类名相同。例如，如果你又一个类 `Myclass` ，它的文件名也必须是 Myclass.php

# #

类和方法命名

类名必须大写开头，字与字间使用下划线分割，而不是驼峰风格。

错误:

```
class superclass
class SuperClass
```

正确:

```
class Super_class
class Super_class {

   public function __construct()
   {

   }
}
```

类方法必须全小写，命名能清楚的表达函数的功能，最好包含动词。尽量避免太长太啰嗦的命名。字与字间使用下划线分割。

错误:

```
function fileproperties()       // not descriptive and needs underscore separator
function fileProperties()       // not descriptive and uses CamelCase
function getfileproperties()       // Better!  But still missing underscore separator
function getFileProperties()       // uses CamelCase
function get_the_file_properties_from_the_file()    // wordy
```

正确:

```
function get_file_properties()    // descriptive, underscore separator, and all lowercase letters
```

# #

变量名

变量的命名规则与方法的命名规则十分相似。就是说，变量名应该只包含小写字母，用下划线分隔，并且能适当地指明变量的用途和内容。那些短的、无意义的变量名应该只作为迭代器用在 for() 循环里。

错误:

```
$j = 'foo';       // single letter variables should only be used in for() loops
$Str          // contains uppercase letters
$bufferedText       // uses CamelCasing, and could be shortened without losing semantic meaning
$groupid       // multiple words, needs underscore separator
$name_of_last_city_used    // too long
```

正确:

```
for ($j = 0; $j < 10; $j++)
$str
$buffer
$group_id
$last_city
```

# #

注释

通常来说，代码尽量需要注释。它不仅能帮助新手描述清楚流程和代码目的，也能让你几个月后再看代码能很快入手。注释非强制规范，但是推荐以下形式。

文档块式的注释要写在类和方法的声明前，这样它们就能被集成开发环境(IDE)捕获：

```
/**
 * Super Class
 *
 * @package    Package Name
 * @subpackage    Subpackage
 * @category    Category
 * @author    Author Name
 * @link    http://example.com
 */
class Super_class {



/**
 * Encodes string for use in XML
 *
 * @param    string    $str    Input string
 * @return    string
 */
function xml_encode($str)



/**
 * Data for class manipulation
 *
 * @var    array
 */
public $data = array();
```

使用行注释时，在大的注释块和代码间留一个空行。

```
// break up the string by newlines
$parts = explode("\n", $str);
```

```
// A longer comment that needs to give greater detail on what is
// occurring and why can use multiple single-line comments.  Try to
// keep the width reasonable, around 70 characters is the easiest to
// read.  Don't hesitate to link to permanent external resources
// that may provide greater detail:
//
// http://example.com/information_about_something/in_particular/

$parts = $this->foo($parts);
```

#

常量

常量命名除了要全部用大写外，其他的规则都和变量相同。**在适当的时候，始终使用 CodeIgniter 常量，例如 L**ASH, LD, RD, PATH_CACHE 等等.

错误:

```
myConstant    // missing underscore separator and not fully uppercase
N       // no single-letter constants
S_C_VER       // not descriptive
$str = str_replace('{foo}', 'bar', $str);    // should use LD and RD constants
```

正确:

```
MY_CONSTANT
NEWLINE
SUPER_CLASS_VERSION
$str = str_replace(LD.'foo'.RD, 'bar', $str);
```

# #

TRUE, FALSE, 和 NULL

TRUE, FALSE, 和 NULL 关键字必须全部大写。

错误:

```
if ($foo == true)
$bar = false;
function foo($bar = null)
```

正确:

```
if ($foo == TRUE)
$bar = FALSE;
function foo($bar = NULL)
```

# #

逻辑操作

不建议使用 `||` ，因为在某些设备上辨识度低（很像数字 11）。`&&` 比 `AND` 更好一些，`!` 前后都要加上空格。

错误:

```
if ($foo || $bar)
if ($foo AND $bar)  // okay but not recommended for common syntax highlighting applications
if (!$foo)
if (! is_array($foo))
```

正确:

```
if ($foo OR $bar)
if ($foo && $bar) // recommended
if ( ! $foo)
if ( ! is_array($foo))
```

# #

比较返回值与类型映射

某些 PHP 函数失败时返回 FALSE，但是也有可能返回 "" 或 0，它在松散的比较中会被计算为 FALSE。在条件语句中使用这些返回值的时候，为了确保返回值是你所预期的类型而不是一个有着松散类型的值，请进行显式的比较。

在返回和检查你自己的变量时也要遵循这种严格的方法，必要时使用 === 和 !==。

错误:

```
// If 'foo' is at the beginning of the string, strpos will return a 0,
// resulting in this conditional evaluating as TRUE
if (strpos($str, 'foo') == FALSE)
```

正确:

```
if (strpos($str, 'foo') === FALSE)
```

错误:

```
function build_string($str = "")
{
    if ($str == "")    // uh-oh!  What if FALSE or the integer 0 is passed as an argument?
    {

    }
}
```

正确:

```
function build_string($str = "")
{
    if ($str === "")
    {

    }
}
```

# #

调试代码

提交代码的时候不要遗留调试信息，即使是注释过的。`var_dump()`，`print_r()`，`die()` / `exit()` 不要出现在你的代码中，除非有特殊用途.

# #

文件中的空格

在 PHP 开始标签签名和结束标签后面都不应该有空格。由于输出已经被缓存，所以文件中的空格会导致 CodeIgniter 在输出自己内容前，就开始输出，这会导致 CodeIgniter 出错，并且无法发送正确的头。

# #

兼容性

CodeIgniter 推荐的版本是 5.4+，但是也需要和 PHP 5.2.4 版本兼容。你的代码必须版本兼容，或提供合适的备案，或者做出选择性的功能。

不要使用那些依赖于非默认安装库的 PHP 函数，除非你的代码中包含了该函数不可用时的替代方法。

# #

一个文件一个类

每个类都有独立的文件，除非这些类紧密关联。CodeIgniter 里 Xmlrpc 库就是一个文件多个类。

# #

空格

在你的使用 tab 作为缩进，而不是空格。这是一件很小的事情，但是它能让其他开发者使用他们喜欢的缩进排版开阅读你的代码，并且可以在他们自己的 IDE 中调整。另外还有一个好处，使用一个 tab 至少能取代4个空格，因此文件更小。

# #

换行

文件必须使用 Unix 的换行。这个规则比较偏向于 windows 使用者，总之确认你的编辑器是用的 unix 换行。

# #

代码缩进

除了类声明，使用 Allman 风格缩进，大括号永远都是独自一行，并且与其所属的控制语句有相同的缩进排版。

错误:

```
function foo($bar) {
    // ...
}

foreach ($arr as $key => $val) {
    // ...
}

if ($foo == $bar) {
    // ...
} else {
    // ...
}

for ($i = 0; $i < 10; $i++)
    {
    for ($j = 0; $j < 10; $j++)
        {
        // ...
        }
    }

try {
    // ...
}
catch() {
    // ...
}
```

正确:

```
function foo($bar)
{
    // ...
}
```

```
foreach ($arr as $key => $val)
{
    // ...
}

if ($foo == $bar)
{
    // ...
}
else
{
    // ...
}

for ($i = 0; $i < 10; $i++)
{
    for ($j = 0; $j < 10; $j++)
    {
        // ...
    }
}

try
{
    // ...
}
catch()
{
    // ...
}
```

# #

括号间距

一般来说，括号不应该有额外的空格，但是在一些需要括号来接受参数的控制结构（declare, do-while,elseif, for, foreach, if, switch, while）后面应该加上空格，以便于函数区别，增加可读性。

错误:

```
$arr[ $foo ] = 'foo';
```

正确:

```
$arr[$foo] = 'foo'; // no spaces around array keys
```

错误:

```
function foo ( $bar )
{

}
```

正确:

```
function foo($bar) // no spaces around parenthesis in function declarations
{

}
```

错误:

```
foreach( $query->result() as $row )
```

正确:

```
foreach ($query->result() as $row) // single space following PHP control structures, but not in interior parenthesis
```

# #

本地化文本

CodeIgniter 库需要使用相应的语言文件。

错误:

```
return "Invalid Selection";
```

正确:

```
return $this->lang->line('invalid_selection');
```

#

私有的方法和变量

仅能内部访问的方法和变量，比如工具和辅助函数，必须用下划线开头。

```
public function convert_text()
private function _convert_text()
```

# #

PHP 错误

代码不应该有错误，并且不能通过隐藏警告和提醒来达成目标。用到不是自己创建的变量时（比如，`$_POST` 数组 key），先用 `isset()` 检查是否可用。

确保你的开发环境允许所有的用户报告错误，以及启用 PHP 环境中得 display_errors。你可以这么检查：

```
if (ini_get('display_errors') == 1)
{
   exit "Enabled";
}
```

在某些服务器 *display_errors* 被禁用，而你没有办法修改 php.ini，你通常可以这么启用：

```
ini_set('display_errors', 1);
```

注意: 运行的时候使用 `ini_set()` 来设置 display_errors.也就是说程序发生重大错误的时候，将不会有任何作用。

# #

短标记

一直使用 PHP 完整标记，以避免服务器不支持短标记，也就是未打开 short_open_tag 。

错误:

```
<? echo $foo; ?>

<?=$foo?>
```

正确:

```
<?php echo $foo; ?>
```

注意: PHP 5.4 起，永远可以使用 <?= 标签。

# #

每行一条语句

永远不要在一行里写多条语句

错误:

```
$foo = 'this'; $bar = 'that'; $bat = str_replace($foo, $bar, $bag);
```

正确:

```
$foo = 'this';
$bar = 'that';
$bat = str_replace($foo, $bar, $bag);
```

# #

字符串

一直使用单引号，除非你需要解析变量，如果需要解析变量请使用大括号，以避免变量解析错误。 如果字符串包含单引号的话你可以使用双引号，这样就不用转义了。

错误:

```
"My String"                 // no variable parsing, so no use for double quotes
"My string $foo"              // needs braces
'SELECT foo FROM bar WHERE baz = \'bag\'    // ugly
```

正确:

```
'My String'
"My string {$foo}"
"SELECT foo FROM bar WHERE baz = 'bag'"
```

# #

SQL 查询

SQL 关键字永远大写: SELECT, INSERT, UPDATE, WHERE, AS, JOIN, ON, IN, etc.

将较长的语句拆成多行可以增加可读性，最好每个子句都放一行

错误:

```
// keywords are lowercase and query is too long for
// a single line (... indicates continuation of line)
$query = $this->db->query("select foo, bar, baz, foofoo, foobar as raboof, foobaz from exp_pre_email_addresses
...where foo != 'oof' and baz != 'zab' order by foobaz limit 5, 100");
```

正确:

```
$query = $this->db->query("SELECT foo, bar, baz, foofoo, foobar AS raboof, foobaz
            FROM exp_pre_email_addresses
            WHERE foo != 'oof'
            AND baz != 'zab'
            ORDER BY foobaz
            LIMIT 5, 100");
```

# #

默认的函数参数

适当的时候，提供函数参数的缺省值，这有助于防止因错误的函数调用引起的 PHP 错误，另外提供常见的备选值可以节省几行代码 例如:

```
function foo($bar = '', $baz = FALSE)
```

39 结束语

这篇教程并没有覆盖完整的内容管理系统，但是它向你介绍了更重要的路由，控制器，数据模型。希望这篇教程可以帮助你了解 CodeIgniter 基本的设计模式，以便之后扩展。

现在你已经完成了这个教程，我们推荐你查看剩余的教程。因为 CodeIgniter 文档比较完善，大家经常称赞。利用这个优势，并通读“介绍”和“常规主题”部分，当你需要时可以查阅类库参考和辅助函数参考。

每个中级 PHP 程序员都可以在几天内学会 CodeIgniter。

如果你对架构或者 CodeIgniter 代码仍有疑问，可以：

- 访问论坛
- 访问聊天室
- 浏览wiki

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台

更多信息请访问

http://wiki.jikexueyuan.com/project/codeigniter-user-guide/