
CodeIgniter 中文手册

发布 **3.0.2**

不列颠哥伦比亚理工学院
CodeIgniter 中国开发者社区

2016 年 04 月 22 日

常规主题

7.1 常规主题

7.1.1 CodeIgniter URL

默认情况下，CodeIgniter 中的 URL 被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，CodeIgniter 使用基于段的方法：

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

注解： 在 CodeIgniter 中也可以使用查询字符串的方法，参见下文。

URI 分段

如果遵循模型 - 视图 - 控制器模式，那么 URI 中的每一段通常表示下面的含义：

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

1. 第一段表示要调用的控制器 **类** ；
2. 第二段表示要调用的类中的 **函数** 或 **方法** ；
3. 第三段以及后面的段代表传给控制器的参数，如 ID 或其他任何变量；

[URI 类](#) 和 [URL 辅助函数](#) 包含了一些函数可以让你更容易的处理 URI 数据，另外，你的 URL 可以通过 [URI 路由](#) 进行重定向从而得到更大的灵活性。

移除 URL 中的 index.php

默认情况，你的 URL 中会包含 **index.php** 文件：

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

如果你的 Apache 服务器启用了 *mod_rewrite* , 你可以简单的通过一个 .htaccess 文件再加上一些简单的规则就可以移除 index.php 了。下面是这个文件的一个例子, 其中使用了“否定条件”来排除某些不需要重定向的项目:

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

在上面的例子中, 除已存在的目录和文件, 其他的 HTTP 请求都会经过你的 index.php 文件。

注解: 这些规则并不是对所有 Web 服务器都有效。

注解: 确保使用上面的规则排除掉你希望能直接访问到的资源。

添加 URL 后缀

在你的 **config/config.php** 文件中你可以指定一个后缀, CodeIgniter 生成 URL 时会自动添加上它。例如, 一个像这样的 URL:

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

你可以添加一个后缀, 如: **.html** , 这样页面看起来就是这个样子:

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

启用查询字符串

有些时候, 你可能更喜欢使用查询字符串格式的 URL:

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

CodeIgniter 也支持这个格式, 你可以在 **application/config.php** 配置文件中启用它。打开你的配置文件, 查找下面这几项:

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

你只要把“enable_query_strings”参数设为 TRUE 即可启用该功能。然后通过你设置的 trigger 关键字来访问你的控制器和方法:

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

注解: 如果你使用查询字符串格式的 URL, 你就必须自己手工构造 URL 而不能使用 URL 辅助函数了 (以及其他生成 URL 相关的库, 例如表单辅助函数), 这是由于这些库只能处理分段格式的 URL 。

7.1.2 控制器

控制器是你整个应用的核心，因为它们决定了 HTTP 请求将被如何处理。

目录

- 控制器
 - 什么是控制器?
 - 让我们试试看: Hello World!
 - 方法
 - 通过 URI 分段向你的方法传递参数
 - 定义默认控制器
 - 重映射方法
 - 处理输出
 - 私有方法
 - 将控制器放入子目录中
 - 构造函数
 - 保留方法名
 - 就这样了!

什么是控制器?

简而言之，一个控制器就是一个类文件，是以一种能够和 URI 关联在一起的方式来命名的。

考虑下面的 URI:

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

上例中，CodeIgniter 将会尝试查询一个名为 `Blog.php` 的控制器并加载它。

当控制器的名称和 URI 的第一段匹配上时，它将会被加载。

让我们试试看: **Hello World!**

接下来你会看到如何创建一个简单的控制器，打开你的文本编辑器，新建一个文件 `Blog.php`，然后放入以下代码:

```
<?php
class Blog extends CI_Controller {

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

然后将文件保存到 `application/controllers/` 目录下。

重要: 文件名必须是大写字母开头, 如: 'Blog.php'。

现在使用类似下面的 URL 来访问你的站点:

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

如果一切正常, 你将看到:

Hello World!

重要: 类名必须以大写字母开头。

这是有效的:

```
<?php
class Blog extends CI_Controller {

}
```

这是无效的:

```
<?php
class blog extends CI_Controller {

}
```

另外, 一定要确保你的控制器继承了父控制器类, 这样它才能使用父类的方法。

方法

上例中, 方法名为 `index()`。"index" 方法总是在 URI 的 **第二段** 为空时被调用。另一种显示 "Hello World" 消息的方法是:

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

URI 中的第二段用于决定调用控制器中的哪个方法。

让我们试一下, 向你的控制器添加一个新的方法:

```
<?php
class Blog extends CI_Controller {

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

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

```
    }
}
```

现在, 通过下面的 URL 来调用 comments 方法:

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

你应该能看到你的新消息了。

通过 **URI** 分段向你的方法传递参数

如果你的 URI 多于两个段, 多余的段将作为参数传递到你的方法中。

例如, 假设你的 URI 是这样:

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

你的方法将会收到第三段和第四段两个参数 (“sandals” 和 “123”) :

```
<?php
class Products extends CI_Controller {

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

重要: 如果你使用了 **URI 路由**, 传递到你的方法的参数将是路由后的参数。

定义默认控制器

CodeIgniter 可以设置一个默认的控制器的, 当 URI 没有分段参数时加载, 例如当用户直接访问你网站的首页时。打开 `application/config/routes.php` 文件, 通过下面的参数指定一个默认的控制器的:

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

其中, “Blog” 是你想加载的控制器类名, 如果你现在通过不带任何参数的 `index.php` 访问你的站点, 你将看到你的 “Hello World” 消息。

For more information, please refer to the “Reserved Routes” section of the **URI 路由** documentation.

重映射方法

正如上文所说, URI 的第二段通常决定控制器的哪个方法被调用。CodeIgniter 允许你使用 `remap()` 方法来重写该规则:

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

重要: 如果你的控制包含一个 `_remap()` 方法, 那么无论 URI 中包含什么参数时都会调用该方法。它允许你定义你自己的路由规则, 重写默认的使用 URI 中的分段来决定调用哪个方法这种行为。

被重写的方法 (通常是 URI 的第二段) 将被作为参数传递到 `_remap()` 方法:

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

方法名之后的所有其他段将作为 `_remap()` 方法的第二个参数, 它是可选的。这个参数可以使用 PHP 的 `call_user_func_array()` 函数来模拟 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 有一个输出类, 它可以自动的将最终数据发送到你的浏览器。更多信息可以阅读[视图](#)和[输出类](#)页面。但是, 有时候, 你可能希望对最终的数据进行某种方式的后处理, 然后你自己手工发送到浏览器。CodeIgniter 允许你向你的控制器中添加一个 `_output()` 方法, 该方法可以接受最终的输出数据。

重要: 如果你的控制器含有一个 `_output()` 方法, 输出类将会调用该方法来显示数据, 而不是直接显示数据。该方法的第一个参数包含了最终输出的数据。

这里是个例子:

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

注解: 请注意, 当数据传到 `_output()` 方法时, 数据已经是最终状态。这时基准测试和计算内存占用都已经完成, 缓存文件也已经写到文件 (如果你开启缓存的话), HTTP 头也已经发送 (如果用到了该特性)。为了使你的控制器能正确处理缓存, `_output()` 可以这样写:

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

如果你在使用 `_output()` 时, 希望获取页面执行时间和内存占用情况, 结果可能会不准确, 因为并没有统计你后加的处理代码。另一个可选的方法是在所有最终输出之前来进行处理, 请参阅[输出类](#)。

私有方法

有时候你可能希望某些方法不能被公开访问, 要实现这点, 只要简单的将方法声明为 `private` 或 `protected`, 这样这个方法就不能被 URL 访问到了。例如, 如果你有一个下面这个方法:

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

使用下面的 URL 尝试访问它, 你会发现是无法访问的:

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

注解: 为了向后兼容原有的功能, 在方法名前加上一个下划线前缀也可以让该方法无法访问。

将控制器放入子目录中

如果你正在构建一个比较大的应用, 那么将控制器放到子目录下进行组织可能会方便一点。CodeIgniter 也可以实现这一点。

你只需要简单的在 `application/controllers/` 目录下创建新的目录, 并将控制器文件放到子目录下。

注解: 当使用该功能时, URI 的第一段必须制定目录, 例如, 假设你在如下位置有一个控制器:

application/controllers/products/Shoes.php

为了调用该控制器, 你的 URI 应该像下面这样:

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

Each of your sub-directories may contain a default controller which will be called if the URL contains *only* the sub-directory. Simply put a controller in there that matches the name of your 'default_controller' as specified in your *application/config/routes.php* file.

你也可以使用 CodeIgniter 的 [URI 路由](#) 功能来重定向 URI 。

构造函数

如果你打算在你的控制器中使用构造函数, 你 **必须** 将下面这行代码放在里面:

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

原因是你的构造函数将会覆盖父类的构造函数, 所以我们要手工的调用它。

例如:

```
<?php
class Blog extends CI_Controller {

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

如果你需要在你的类被初始化时设置一些默认值, 或者进行一些默认处理, 构造函数将很有用。构造函数没有返回值, 但是可以执行一些默认操作。

保留方法名

因为你的控制器将继承主程序的控制器, 在新建方法时你必须要小心不要使用和父类一样的方法名, 要不然你的方法将覆盖它们, 参见[保留名称](#)。

重要: 另外, 你也绝对不要新建一个和类名称一样的方法。如果你这样做了, 而且你的控制器又没有 `__construct()` 构造函数, 那么这个和类名同名的方法 `Index::index()` 将会作为类的构造函数被执行! 这个是 PHP4 的向前兼容的一个特性。

就这样了!

OK, 总的来说, 这就是关于控制器的所有内容了。

7.1.3 保留名称

为了便于编程, CodeIgniter 使用了一些函数、方法、类和变量名来实现。因此, 这些名称不能被开发者所使用, 下面是不能使用的保留名称列表。

控制器名称

因为你的控制器类将继承主程序控制器, 所以你的方法命名一定不能和主程序控制器类中的方法名相同, 否则你的方法将会覆盖他们。下面列出了已经保留的名称, 请不要将你的控制器命名为这些:

- CLController
- Default
- index

函数

- `is_php()`
- `is_really_writable()`
- `load_class()`
- `is_loaded()`
- `get_config()`
- `config_item()`
- `show_error()`
- `show_404()`
- `log_message()`
- `set_status_header()`
- `get_mimes()`
- `html_escape()`
- `remove_invisible_characters()`
- `is_https()`
- `function_usable()`
- `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

7.1.4 视图

简单来说, 一个视图其实就是一个 Web 页面, 或者页面的一部分, 像页头、页脚、侧边栏等。实际上, 视图可以很灵活的嵌在另一个视图里, 然后这个视图再嵌在另一个视图里, 等等, 如果你想使用这种层次结构的话, 可以这样做。

视图不是直接被调用的, 它必须通过[控制器](#)来加载。在 MVC 框架里, 控制器扮演着类似于交警的角色, 它专门负责读取特定的视图。如果你还没有读过[控制器](#)页面, 你应该先看下这个。

使用在[控制器](#)页面中创建的控制器例子, 让我们再添加一个视图。

创建视图

使用你的文本编辑器, 创建一个 blogview.php 文件, 代码如下:

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

然后保存到你的 `application/views/` 目录下。

加载视图

使用下面的方法来加载指定的视图:

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

name 参数为你的视图文件名。

注解: 文件的扩展名.php 可以省略, 除非你使用了其他的扩展名。

现在, 打开你之前创建的控制器文件 Blog.php, 然后将 echo 语句替换成加载视图的代码:

```
<?php
class Blog extends CI_Controller {

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

跟之前一样, 通过类似于下面的 URL 来访问你的网站, 你将看到新的页面:

example.com/index.php/blog/

加载多个视图

CodeIgniter 可以智能的处理在控制器中多次调用 `$this->load->view()` 方法。如果出现了多次调用, 视图会被合并到一起。例如, 你可能希望有一个页头视图、一个菜单视图, 一个内容视图以及一个页脚视图。代码看起来应该这样:

```
<?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');
    }
}
```

在上面的例子中, 我们使用了“添加动态数据”, 我们会在后面讲到。

在子目录中存储视图

如果你喜欢的话, 你的视图文件可以放到子目录下组织存储, 当你这样做, 加载视图时需要包含子目录的名字, 例如:

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

向视图添加动态数据

通过视图加载方法的第二个参数可以从控制器中动态的向视图传入数据, 这个参数可以是一个 **数组** 或者一个 **对象**。这里是使用数组的例子:

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

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

这里是使用对象的例子:

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

注解: 当你使用对象时, 对象中的变量会转换为数组元素。

让我们在你的控制器文件中尝试一下, 添加如下代码:

```
<?php
class Blog extends CI_Controller {

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

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

再打开你的视图文件, 将文本修改为传入的数组对应的变量:

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

然后通过刚刚的 URL 重新加载页面, 你应该可以看到变量被替换了。

使用循环

传入视图文件的数据不仅仅限制为普通的变量, 你还可以传入多维数组, 这样你就可以在视图中生成多行了。例如, 如果你从数据库中获取数据, 一般情况下数据都是一个多维数组。

这里是个简单的例子, 将它添加到你的控制器中:

```
<?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);
    }
}
```

然后打开你的视图文件, 创建一个循环:

```
<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>
```

注解: 你会发现在上例中, 我们使用了 PHP 的替代语法, 如果你对其还不熟悉, 可以阅读[这里](#)。

将视图作为数据返回

加载视图方法有一个可选的第三个参数可以让你修改它的默认行为，它让视图作为字符串返回而不是显示到浏览器中，这在你想对视图数据做某些处理时很有用。如果你将该参数设置为 `TRUE`，该方法返回字符串，默认情况下为 `FALSE`，视图将显示到浏览器。如果你需要返回的数据，记住将它赋值给一个变量：

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

7.1.5 模型

模型对于那些想使用更传统的 MVC 模式的人来说是可选的。

目录

- 模型
 - 什么是模型?
 - 剖析模型
 - 加载模型
 - 模型的自动加载
 - 连接数据库

什么是模型?

模型是专门用来和数据库打交道的 PHP 类。例如，假设你使用 CodeIgniter 管理一个博客，那么你应该会有一个用于插入、更新以及获取博客数据的模型类。这里是一个模型类的例子：

```
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']));
}
}
```

注解: 上面的例子中使用了[查询构造器](#) 数据库方法。

注解: 为了保证简单, 我们在这个例子中直接使用了 `$_POST` 数据, 这其实是个不好的实践, 一个更通用的做法是使用[输入库](#) 的 `$this->input->post('title')`。

剖析模型

模型类位于你的 `application/models/` 目录下, 如果你愿意, 也可以在里面创建子目录。

模型类的基本原型如下:

```
class Model_name extends CI_Model {

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

}
```

其中, `Model_name` 是类的名字, 类名的第一个字母 **必须** 大写, 其余部分小写。确保你的类继承 `CI_Model` 基类。

文件名和类名应该一致, 例如, 如果你的类是这样:

```
class User_model extends CI_Model {

    public function __construct()
    {
```

```

        parent::__construct();
    }
}

```

那么你的文件名应该是这样:

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

加载模型

你的模型一般会在你的**控制器**的方法中加载并调用, 你可以使用下面的方法来加载模型:

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

如果你的模型位于一个子目录下, 那么加载时要带上你的模型所在目录的相对路径, 例如, 如果你的模型位于 *application/models/blog/Queries.php*, 你可以这样加载它:

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

加载之后, 你可以通过一个和你的类同名的对象访问模型中的方法。

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

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

如果你想将你的模型对象赋值给一个不同名字的对象, 你可以使用 `$this->load->model()` 方法的第二个参数:

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

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

这里是一个例子, 该控制器加载一个模型, 并处理一个视图:

```

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);
    }
}

```

模型的自动加载

如果你发现你有一个模型需要在整个应用程序中使用, 你可以让 CodeIgniter 在系统初始化时自动加载它。打开 `application/config/autoload.php` 文件, 并将该模型添加到 `autoload` 数组中。

连接数据库

当模型加载之后, 它 **并不会** 自动去连接你的数据库, 下面是一些关于数据库连接的选项:

- 你可以在控制器或模型中使用 [标准的数据库方法](#) 连接数据库。
- 你可以设置第三个参数为 `TRUE` 让模型在加载时自动连接数据库, 会使用你的数据库配置文件中的配置:

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

- 你还可以通过第三个参数传一个数据库连接配置:

```
$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);
```

7.1.6 辅助函数

辅助函数, 顾名思义, 是帮助我们完成特定任务的函数。每个辅助函数文件都是某一类函数的集合。例如, **URL 辅助函数** 帮助我们创建链接, **表单辅助函数** ** 帮助我们创建表单元素, ** **本文辅助函数** 帮助我们处理文本的格式化, **Cookie 辅助函数** 帮助我们读取或设置 Cookie, **文件辅助函数** 帮助我们处理文件, 等等等等。

不同于 CodeIgniter 中的大多数系统, 辅助函数没有使用面向对象的方式来实现的。它们是简单的过程式函数, 每个函数处理一个特定的任务, 不依赖于其他的函数。

CodeIgniter 默认不会自己加载辅助函数, 所以使用辅助函数的第一步就是加载它。一旦加载了, 它就可以在你的 **控制器** 和 **视图** 中全局访问了。

一般情况下, 辅助函数位于 `system/helpers` 或者 `application/helpers` 目录目录下。CodeIgniter 首先会查找 `application/helpers` 目录, 如果该目录不存在, 或者你加载的辅助函数没有在该目录下找到, CodeIgniter 就会去 `system/helpers/` 目录查找。

加载辅助函数

可以使用下面的方法简单的加载辅助函数:

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

name 参数为辅助函数的文件名, 去掉.php 文件后缀以及 _helper 部分。

例如, 要加载 **URL 辅助函数**, 它的文件名为 **url_helper.php**, 你可以这样加载它:

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

辅助函数可以在你的控制器方法的任何地方加载 (甚至可以在你的视图文件中加载, 尽管这不是个好的实践), 只要确保在使用之前加载它就可以了。你可以在你的控制器的构造函数中加载它, 这样就可以在该控制器的任何方法中使用它, 你也可以在某个需要它的函数中单独加载它。

注解: 上面的加载辅助函数的方法没有返回值, 所以不要将它赋值给变量, 直接调用就好了。

加载多个辅助函数

如果你需要加载多个辅助函数, 你可以使用一个数组, 像下面这样:

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

自动加载辅助函数

如果你需要在你的整个应用程序中使用某个辅助函数, 你可以将其设置为在 CodeIgniter 初始化时自动加载它。打开 **application/config/autoload.php** 文件然后将你想加载的辅助函数添加到 **autoload** 数组中。

使用辅助函数

一旦你想要使用的辅助函数被加载, 你就可以像使用标准的 PHP 函数一样使用它们。

例如, 要在你的视图文件中使用 **anchor()** 函数创建一个链接, 你可以这样做:

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

其中, "Click Here" 是链接的名称, "blog/comments" 是你希望链接到 controller/method 的 URI。

扩展辅助函数

为了扩展辅助函数, 你需要在 `application/helpers/` 目录下新建一个文件, 文件名和已存在的辅助函数文件名一样, 但是要加上 **MY_** 前缀 (这个可以配置, 见下文)。

如果你只是想往现有类中添加一些功能, 例如增加一两个方法, 或者修改辅助函数中的某个函数, 这时替换整个类感觉就有点杀鸡用牛刀了。在这种情况下, 最好的方法是扩展类。

注解: “扩展”一词在这里可能不是很恰当, 因为辅助函数都是过程式的独立函数, 在传统编程中并不能被扩展。不过在 CodeIgniter 中, 你可以向辅助函数中添加函数, 或者使用你自己的函数替代辅助函数中的函数。

例如, 要扩展原始的 **数组辅助函数**, 首先你要创建一个文件 `application/helpers/MY_array_helper.php`, 然后像下面这样添加或重写函数:

```
// 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);
}
```

设置自定义前缀

用于扩展辅助函数的文件名前缀和扩展类库和核心类是一样的。要自定义这个前缀, 你可以打开 `application/config/config.php` 文件然后找到这项:

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

请注意所有原始的 CodeIgniter 类库都以 **CI_** 开头, 所以请不要使用这个作为你的自定义前缀。

然后?

在目录里你可以找到所有的辅助函数清单, 你可以浏览下它们看看它们都是做什么的。

7.1.7 使用 CodeIgniter 类库

所有的系统类库都位于 *system/libraries/* 目录下, 大多数情况下, 在使用之前, 你要先在控制器中初始化它, 使用下面的方法:

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

'class_name' 是你想要调用的类库名称, 例如, 要加载[表单验证类库](#), 你可以这样做:

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

一旦类库被载入, 你可以根据该类库的用户指南中介绍的方法去使用它了。

另外, 多个类库可以通过一个数组来同时加载。

例如:

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

创建你自己的类库

请阅读用户指南中关于[创建你自己的类库](#)部分。

7.1.8 创建类库

当我们使用“类库”这个词的时候, 通常我们指的是位于 *libraries* 这个目录下的那些类, 在我们这份用户手册的类库参考部分有详细的介绍。但是在这篇文章中, 我们将介绍如何在 *application/libraries* 目录下创建你自己的类库, 和全局的框架类库独立开来。

另外, 如果你希望在现有的类库中添加某些额外功能, CodeIgniter 允许你扩展原生的类, 或者你甚至可以在你的 *application/libraries* 目录下放置一个和原生的类库同名的文件完全替代它。

总结起来:

- 你可以创建一个全新的类库,
- 你可以扩展原生的类库,
- 你可以替换掉原生的类库。

下面将详细讲述这三点。

注解: 除了数据库类不能被扩展或被你的类替换外, 其他的类都可以。

存储位置

你的类库文件应该放置在 *application/libraries* 目录下, 当你初始化类时, CodeIgniter 会在这个目录下寻找这些类。

命名约定

- 文件名首字母必须大写, 例如: Myclass.php
- 类名定义首字母必须大写, 例如: class Myclass
- 类名和文件名必须一致

类文件

类应该定义成如下原型:

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

class Someclass {

    public function some_method()
    {
    }
}
```

注解: 这里的 Someclass 名字只是个例子。

使用你的类

在你的控制器 的任何方法中使用如下代码初始化你的类:

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

其中, *someclass* 为文件名, 不包括.php 文件扩展名。文件名可以写成首字母大写, 也可以写成全小写, CodeIgniter 都可以识别。

一旦加载, 你就可以使用小写字母名称来访问你的类:

```
$this->someclass->some_method(); // Object instances will always be lower case
```

初始化类时传入参数

在加载类库的时候, 你可以通过第二个参数动态的传递一个数组数据, 该数组将被传到你的类的构造函数中:

```
$params = array('type' => 'large', 'color' => 'red');

$this->load->library('someclass', $params);
```

如果你使用了该功能, 你必须在定义类的构造函数时加上参数:

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

class Someclass {

    public function __construct($params)
    {
        // Do something with $params
    }
}
```

你也可以将参数保存在配置文件中来传递, 只需简单的创建一个和类文件同名的配置文件, 并保存到你的 `application/config/` 目录下。要注意的是, 如果你使用了上面介绍的方法动态的传递参数, 配置文件将不可用。

在你的类库中使用 CodeIgniter 资源

在你的类库中使用 `get_instance()` 函数来访问 CodeIgniter 的原生资源, 这个函数返回 CodeIgniter 超级对象。

通常情况下, 在你的控制器方法中你会使用 `$this` 来调用所有可用的 CodeIgniter 方法:

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

但是 `$this` 只能在你的控制器、模型或视图中直接使用, 如果你想在你的自己的类中使用 CodeIgniter 类, 你可以像下面这样做:

首先, 将 CodeIgniter 对象赋值给一个变量:

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

一旦你把 CodeIgniter 对象赋值给一个变量之后, 你就可以使用这个变量来代替 `$this`

```
$CI =& get_instance();

$CI->load->helper('url');
$CI->load->library('session');
```



```
$CI->config->item('base_url');  
// etc.
```

注解: 你会看到上面的 `get_instance()` 函数通过引用来传递:

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

这是非常重要的, 引用赋值允许你使用原始的 CodeIgniter 对象, 而不是创建一个副本。

既然类库是一个类, 那么我们最好充分的使用 OOP 原则, 所以, 为了让类中的所有方法都能使用 CodeIgniter 超级对象, 建议将其赋值给一个属性:

```
class Example_library {  
  
    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()  
    {  
        echo $this->CI->config->item('base_url');  
    }  
}
```

使用你自己的类库替换原生类库

简单的将你的类文件名改为和原生的类库文件一致, CodeIgniter 就会使用它替换掉原生的类库。要使用该功能, 你必须将你的类库文件和类定义改成和原生的类库完全一样, 例如, 要替换掉原生的 Email 类的话, 你要新建一个 `application/libraries/Email.php` 文件, 然后定义定义你的类:

```
class CI_Email {  
  
}
```

注意大多数原生类都以 CI_ 开头。

要加载你的类库，和标准的方法一样：

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

注解： 注意数据库类不能被你自己的类替换掉。

扩展原生类库

如果你只是想往现有的类库中添加一些功能，例如增加一两个方法，这时替换整个类感觉就有点杀鸡用牛刀了。在这种情况下，最好的方法是扩展类库。扩展一个类和替换一个类差不多，除了以下几点：

- 类在定义时必须继承自父类。
- 你的新类名和文件名必须以 MY_ 为前缀（这个可配置，见下文）

例如，要扩展原生的 Email 类你需要新建一个文件命名为 *application/libraries/MY_Email.php*，然后定义你的类：

```
class MY_Email extends CI_Email {

}
```

如果你需要在你的类中使用构造函数，确保你调用了父类的构造函数：

```
class MY_Email extends CI_Email {

    public function __construct($config = array())
    {
        parent::__construct($config);
    }

}
```

注解： 并不是所有的类库构造函数的参数都是一样的，在对类库扩展之前先看看它是怎么实现的。

加载你的扩展类

要加载你的扩展类，还是使用和通常一样的语法。不用包含前缀。例如，要加载上例中你扩展的 Email 类，你可以使用：

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

一旦加载，你还是和通常一样使用类变量来访问你扩展的类，以 email 类为例，访问它的方法如下：

```
$this->email->some_method();
```

设置自定义前缀

要设置你自己的类的前缀, 你可以打开 `application/config/config.php` 文件, 找到下面这项:

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

请注意所有原始的 CodeIgniter 类库都以 **CI_** 开头, 所以请不要使用这个作为你的自定义前缀。

7.1.9 使用 CodeIgniter 驱动器

驱动器是一种特殊类型的类库, 它有一个父类和任意多个子类。子类可以访问父类, 但不能访问兄弟类。在你的 **控制器** 中, 驱动器为你的类库提供了一种优雅的语法, 从而不用将它们拆成很多离散的类。

驱动器位于 `system/libraries/` 目录, 每个驱动器都有一个独立的目录, 目录名和驱动器父类的类名一致, 在该目录下还有一个子目录, 命名为 `drivers`, 用于存放所有子类的文件。

要使用一个驱动器, 你可以在控制器中使用下面的方法来进行初始化:

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

`class_name` 是你想要调用的驱动器类名, 例如, 你要加载名为 `Some_parent` 的驱动器, 可以这样:

```
$this->load->driver('some_parent');
```

然后就可以像下面这样调用该类的方法:

```
$this->some_parent->some_method();
```

而对于那些子类, 我们不用初始化, 可以直接通过父类调用了:

```
$this->some_parent->child_one->some_method();  
$this->some_parent->child_two->another_method();
```

创建你自己的驱动器

请阅读用户指南中关于如何 **创建你自己的驱动器** 部分。

7.1.10 创建驱动器

驱动器目录及文件结构

下面是驱动器目录和文件结构布局的简单例子:

- `/application/libraries/Driver_name`

- Driver_name.php
- drivers
 - * Driver_name_subclass_1.php
 - * Driver_name_subclass_2.php
 - * Driver_name_subclass_3.php

注解: 为了在大小写敏感的文件系统下保证兼容性, Driver_name 目录必须以 `ucfirst()` 函数返回的结果格式进行命名。

注解: 由于驱动器的架构是子驱动器并不继承主驱动器, 因此在子驱动器里无法访问主驱动器中的属性或方法。

译者补充

鉴于这篇文档并没有详细介绍什么是驱动器 (driver), 驱动器的用途, 以及如何创建驱动器, 下面列出一些外部资源供参考:

- [Usage of drivers in CodeIgniter](#)
- [Guide to CodeIgniter Drivers](#)
- [Codeigniter Drivers Tutorial](#)

7.1.11 创建核心系统类

每次 CodeIgniter 运行时, 都有一些基础类伴随着核心框架自动的被初始化。你也可以使用你自己类来替代这些核心类或者扩展这些核心类。

大多数用户一般不会有这种需求, 但对于那些想较大幅度的改变 CodeIgniter 的人来说, 我们依然提供了替换和扩展核心类的选择。

注解: 改变系统核心类会产生很大影响, 所以在你做之前必须清楚地知道自己正在做什么。

系统类清单

以下是系统核心文件的清单, 它们在每次 CodeIgniter 启动时被调用:

- Benchmark
- Config
- Controller
- Exceptions

- Hooks
- Input
- Language
- Loader
- Log
- Output
- Router
- Security
- URI
- Utf8

替换核心类

要使用你自己的系统类替换默认的系统类只需简单的将你自己的文件放入目录 *application/core* 下:

`application/core/some_class.php`

如果这个目录不存在, 你可以创建一个。

任何一个和上面清单中同名的文件将被替换成核心类。

要注意的是, 你的类名必须以 CI 开头, 例如, 你的文件是 `Input.php`, 那么类应该命名为:

```
class CI_Input {  
  
}
```

扩展核心类

如果你只是想往现有类中添加一些功能, 例如增加一两个方法, 这时替换整个类感觉就有点杀鸡用牛刀了。在这种情况下, 最好是使用扩展类的方法。扩展一个类和替换一个类的做法几乎是一样的, 除了要注意以下几点:

- 你定义的类必须继承自父类。
- 你的类名和文件名必须以 MY_ 开头。(这是可配置的, 见下文)

举个例子, 要扩展原始的 `Input` 类, 你需要新建一个文件 `application/core/MY_Input.php`, 然后像下面这样定义你的类:

```
class MY_Input extends CI_Input {  
  
}
```

注解: 如果在你的类中需要使用构造函数, 记得要调用父类的构造函数:

```
class MY_Input extends CI_Input {

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

提示: 任何和父类同名的方法将会取代父类中的方法 (这又被称作 “方法覆盖”), 这让你可以充分的利用并修改 CodeIgniter 的核心。

如果你扩展了控制器核心类, 那么记得在你的应用程序控制器里继承你扩展的新类。

```
class Welcome extends MY_Controller {

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

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

自定义前缀

要想自定义你自己的类的前缀, 打开文件 `application/config/config.php` 然后找到这项:

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

请注意所有原始的 CodeIgniter 类库都以 CI_ 开头, 所以请不要使用这个作为你的自定义前缀。

7.1.12 创建附属类

有些时候, 你可能想在你的控制器之外新建一些类, 但同时又希望这些类还能访问 CodeIgniter 的资源。下面你会看到, 这其实很简单。

get_instance()

```
get_instance()
```

返回 Reference to your controller's instance

返回类型 CI_Controller

任何在你的控制器方法中初始化的类都可以简单的通过 `get_instance()` 函数来访问 CodeIgniter 资源。这个函数返回一个 CodeIgniter 对象。

通常来说, 调用 CodeIgniter 的方法需要使用 `$this`

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

但是 `$this` 只能在你的控制器、模型或视图中使用, 如果你想在你的类中使用 CodeIgniter 类, 你可以像下面这样做:

首先, 将 CodeIgniter 对象赋值给一个变量:

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

一旦你把 CodeIgniter 对象赋值给一个变量之后, 你就可以使用这个变量来 代替 `$this`

```
$CI =& get_instance();

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

如果你在类中使用 “`get_instance()`” 函数, 最好的方法是将它赋值给一个属性, 这样你就不用在每个方法里都调用 `get_instance()` 了。

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');
}
}

```

在上面的例子中, `foo()` 和 `bar()` 方法在初始化 `Example` 类之后都可以正常工作, 而不需要在每个方法里都调用 `get_instance()` 函数。

7.1.13 钩子 - 扩展框架核心

CodeIgniter 的钩子特性提供了一种方法来修改框架的内部运作流程, 而无需修改核心文件。CodeIgniter 的运行遵循着一个特定的流程, 你可以参考这个页面的[应用程序流程图](#)。但是, 有些时候你可能希望在执行流程中的某些阶段添加一些动作, 例如在控制器加载之前或之后执行一段脚本, 或者在其他的某些位置触发你的脚本。

启用钩子

钩子特性可以在 `application/config/config.php` 文件中全局的启用或禁用, 设置下面这个参数:

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

定义钩子

钩子是在 `application/config/hooks.php` 文件中被定义的, 每个钩子可以定义为下面这样的数组格式:

```

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

```

注意:

数组的索引为你想使用的挂钩点名称, 例如上例中挂钩点为 `pre_controller`, 下面会列出所有可用的挂钩点。钩子数组是一个关联数组, 数组的键值可以是下面这些:

- **class** 你希望调用的类名, 如果你更喜欢使用过程式的函数的话, 这一项可以留空。
- **function** 你希望调用的方法或函数的名称。
- **filename** 包含你的类或函数的文件名。

- **filepath** 包含你的脚本文件的目录名。注意：你的脚本必须放在 *application/* 目录里面，所以 **filepath** 是相对 *application/* 目录的路径，举例来说，如果你的脚本位于 *application/hooks/*，那么 **filepath** 可以简单的设置为 'hooks'，如果你的脚本位于 *application/hooks/utilities/*，那么 **filepath** 可以设置为 'hooks/utilities'，路径后面不用加斜线。
- **params** 你希望传递给你脚本的任何参数，可选。

如果你使用 PHP 5.3 以上的版本，你也可以使用 lambda 表达式（匿名函数或闭包）作为钩子，这样语法更简单：

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

多次调用同一个挂钩点

如果你想在同一个挂钩点处添加多个脚本，只需要将钩子数组变成二维数组即可，像这样：

```
$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')
);
```

注意数组索引后面多了个中括号：

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

这可以让你在同一个挂钩点处执行多个脚本，多个脚本执行顺序就是你定义数组的顺序。

挂钩点

以下是所有可用挂钩点的一份列表：

- **pre_system** 在系统执行的早期调用，这个时候只有基准测试类和钩子类被加载了，还没有执行到路由或其他流程。

- **pre_controller** 在你的控制器调用之前执行，所有的基础类都已加载，路由和安全检查也已经完成。
- **post_controller_constructor** 在你的控制器实例化之后立即执行，控制器的任何方法都还尚未调用。
- **post_controller** 在你的控制器完全运行结束时执行。
- **display_override** 覆盖 `_display()` 方法，该方法用于在系统执行结束时向浏览器发送最终的页面结果。这可以让你有自己的显示页面的方法。注意你可能需要使用 `$this->CI =& get_instance()` 方法来获取 CI 超级对象，以及使用 `$this->CI->output->get_output()` 方法来获取最终的显示数据。
- **cache_override** 使用你自己的方法来替代输出类中的 `_display_cache()` 方法，这让你有自己的缓存显示机制。
- **post_system** 在最终的页面发送到浏览器之后、在系统的最后期被调用。

7.1.14 自动加载资源

CodeIgniter 的”自动加载”特性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。如果你需要在整个应用程序中全局使用某些资源，为方便起见可以考虑自动加载它们。

支持自动加载的有下面这些：

- *libraries/* 目录下的核心类
- *helpers/* 目录下的辅助函数
- *config/* 目录下的用户自定义配置文件
- *system/language/* 目录下的语言文件
- *models/* 目录下的模型类

要实现自动加载资源，你可以打开 **application/config/autoload.php** 文件，然后将你需要自动加载的项添加到 `autoload` 数组中。你可以在该文件中的每种类型的 `autoload` 数组的注释中找到相应的提示。

注解： 添加 `autoload` 数组时不用包含文件扩展名（.php）

另外，如果你想让 CodeIgniter 使用 **Composer** 的自动加载，只需将 **application/config/config.php** 配置文件中的 `$config['composer_autoload']` 设置为 `TRUE` 或者设置为你自定义的路径。

7.1.15 公共函数

CodeIgniter 定义了一些全局的函数，你可以在任何地方使用它们，并且不需要加载任何类库或辅助函数。

is_php() (*\$version*)

参数

- **\$version** (*string*) – Version number

返回 TRUE if the running PHP version is at least the one specified or FALSE if not

返回类型 bool

判断当前运行的 PHP 版本是否高于或等于你提供的版本号。

例如:

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

如果当前运行的 PHP 版本等于或高于提供的版本号，该函数返回布尔值 TRUE，反之则返回 FALSE。

is_really_writable(\$file)

参数

- **\$file** (*string*) – File path

返回 TRUE if the path is writable, FALSE if not

返回类型 bool

在 Windows 服务器上只有当文件标志了只读属性时，PHP 的 `is_writable()` 函数才返回 FALSE，其他情况都是返回 TRUE，即使文件不是真的可写也返回 TRUE。

这个函数首先尝试写入该文件，以此来判断该文件是不是真的可写。通常只在 `is_writable()` 函数返回的结果不准确的平台下才推荐使用该函数。

例如:

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

注解: 更多信息，参看 [PHP bug #54709](#)。

config_item(\$key)

参数

- **\$key** (*string*) – Config item key

返回 Configuration key value or NULL if not found

返回类型 mixed

访问配置信息最好的方式是使用配置类，但是，你也可以通过 `config_item()` 函数来访问单个配置项，更多信息，参看配置类

```
set_status_header($code[, $text = ''])
```

参数

- **\$code** (*int*) – HTTP Reponse status code
- **\$text** (*string*) – A custom message to set with the status code

返回类型 void

用于手动设置服务器的 HTTP 状态码，例如：

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

查看[这里](#) 有一份状态码的完整清单。

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

参数

- **\$str** (*string*) – Input string
- **\$url_encoded** (*bool*) – Whether to remove URL-encoded characters as well

返回 Sanitized string

返回类型 string

这个函数防止在 ASCII 字符串中插入空字符，例如：Java\0script 。

举例：

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

```
html_escape($var)
```

参数

- **\$var** (*mixed*) – Variable to escape (string or array)

返回 HTML escaped string(s)

返回类型 mixed

这个函数类似于 PHP 原生的 `htmlspecialchars()` 函数，只是它除了可以接受字符串参数外，还可以接受数组参数。

它在防止 XSS 攻击时很有用。

```
get_mimes()
```

返回 An associative array of file types

返回类型 array

这个函数返回 *application/config/mimes.php* 文件中定义的 MIME 数组的引用。

is_https()

返回 TRUE if currently using HTTP-over-SSL, FALSE if not

返回类型 bool

该函数在使用 HTTPS 安全连接时返回 TRUE，没有使用 HTTPS（包括非 HTTP 的请求）则返回 FALSE。

is_cli()

返回 TRUE if currently running under CLI, FALSE otherwise

返回类型 bool

当程序在命令行下运行时返回 TRUE，反之返回 FALSE。

注解： 该函数会检查 `PHP_SAPI` 的值是否是 'cli'，或者是否定义了 `STDIN` 常量。

function_usable(\$function_name)

参数

- **\$function_name** (*string*) – Function name

返回 TRUE if the function can be used, FALSE if not

返回类型 bool

检查一个函数是否可用，可用返回 TRUE，否则返回 FALSE。

该函数直接调用 `function_exists()` 函数，并检查当前是否加载了 *Suhosin* 扩展 <http://www.hardened-php.net/suhosin/>，如果加载了 *Suhosin*，检查函数有没有被它禁用。

这个函数在你需要检查某些函数的可用性时非常有用，例如 `eval()` 和 `exec()` 函数是非常危险的，可能会由于服务器的安全策略被禁用。

注解： 之所以引入这个函数，是由于 *Suhosin* 的某个 bug 可能会终止脚本的执行，虽然这个 bug 已经被修复了（版本 0.9.34），但可惜的是还没发布。

7.1.16 兼容性函数

CodeIgniter 提供了一系列兼容性函数可以让你使用，它们只有在高版本的 PHP 中才有，或者需要依赖其他的扩展才有。

由于是自己实现的，这些函数本身也可能有它自己的依赖性，但如果你的 PHP 中不提供这些函数时，这些函数还是有用的。

注解: 和 公用函数 `<common_functions>` 一样, 兼容性函数也一直可以访问, 只要满足了他们的依赖条件。

- 密码哈希
 - 依赖性
 - 常量
 - 函数参考
- 哈希 (信息摘要)
 - 依赖性
 - 函数参考
- 多字节字符串
 - 依赖性
 - 函数参考
- 标准函数
 - 依赖性
 - 函数参考

密码哈希

这几个兼容性函数移植了 PHP 标准的 密码哈希扩展 的实现, 这些函数只有在 PHP 5.5 以后的版本中才有。

依赖性

- PHP 5.3.7
- `crypt()` 函数需支持 CRYPT_BLOWFISH

常量

- PASSWORD_BCRYPT
- PASSWORD_DEFAULT

函数参考

`password_get_info($hash)`

参数

- **\$hash** (*string*) – Password hash

返回 Information about the hashed password

返回类型 array

更多信息, 请参考 PHP 手册中的 `password_get_info()` 函数

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

参数

- **\$password** (*string*) – Plain-text password
- **\$algo** (*int*) – Hashing algorithm
- **\$options** (*array*) – Hashing options

返回 Hashed password or FALSE on failure

返回类型 string

更多信息, 请参考 PHP 手册中的 `password_hash()` 函数

注解: 除非提供了你自己的有效盐值, 该函数会依赖于一个可用的 CSPRNG 源 (密码学安全的伪随机数生成器), 下面列表中的每一个都可以满足这点: - `mcrypt_create_iv()` with `MCRYPT_DEV_URANDOM` - `openssl_random_pseudo_bytes()` - `/dev/arandom` - `/dev/urandom`

`password_needs_rehash()`

参数

- **\$hash** (*string*) – Password hash
- **\$algo** (*int*) – Hashing algorithm
- **\$options** (*array*) – Hashing options

返回 TRUE if the hash should be rehashed to match the given algorithm and options, FALSE otherwise

返回类型 bool

更多信息, 请参考 PHP 手册中的 `password_needs_rehash()` 函数

`password_verify($password, $hash)`

参数

- **\$password** (*string*) – Plain-text password
- **\$hash** (*string*) – Password hash

返回 TRUE if the password matches the hash, FALSE if not

返回类型 bool

更多信息, 请参考 PHP 手册中的 `password_verify()` 函数

哈希 (信息摘要)

兼容性函数移植了 `hash_equals()` 和 `hash_pbkdf2()` 的实现, 这两函数分别在 PHP 5.6 和 PHP 5.5 以后的版本中才有。

依赖性

- 无

函数参考

hash_equals(*\$known_string*, *\$user_string*)

参数

- **\$known_string** (*string*) – Known string
- **\$user_string** (*string*) – User-supplied string

返回 TRUE if the strings match, FALSE otherwise

返回类型 string

更多信息, 请参考 PHP 手册中的 [hash_equals\(\)](#) 函数

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

参数

- **\$algo** (*string*) – Hashing algorithm
- **\$password** (*string*) – Password
- **\$salt** (*string*) – Hash salt
- **\$iterations** (*int*) – Number of iterations to perform during derivation
- **\$length** (*int*) – Output string length
- **\$raw_output** (*bool*) – Whether to return raw binary data

返回 Password-derived key or FALSE on failure

返回类型 string

更多信息, 请参考 PHP 手册中的 [hash_pbkdf2\(\)](#) 函数

多字节字符串

这一系列兼容性函数提供了对 PHP 的 [多字节字符串扩展](#) 的有限支持, 由于可选的解决方法有限, 所以只有几个函数是可用的。

注解: 如果没有指定字符集参数, 默认使用 `$config['charset']` 配置。

依赖性

- [iconv](#) 扩展

重要: 这个依赖是可选的, 无论 iconv 扩展是否存在, 这些函数都已经定义了, 如果 iconv 扩展不可用, 它们会降级到非多字节字符串的函数版本。

重要: 当设置了字符集时, 该字符集必须被 iconv 支持, 并且要设置成它可以识别的格式。

注解: 如果你需要判断是否支持真正的多字节字符串扩展, 可以使用 `MB_ENABLED` 常量。

函数参考

mb_strlen(*\$str* [, *\$encoding* = *NULL*])

参数

- **\$str** (*string*) – Input string
- **\$encoding** (*string*) – Character set

返回 Number of characters in the input string or FALSE on failure

返回类型 string

更多信息, 请参考 [PHP 手册中的 mb_strlen\(\) 函数](#)

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

参数

- **\$haystack** (*string*) – String to search in
- **\$needle** (*string*) – Part of string to search for
- **\$offset** (*int*) – Search offset
- **\$encoding** (*string*) – Character set

返回 Numeric character position of where \$needle was found or FALSE if not found

返回类型 mixed

更多信息, 请参考 [PHP 手册中的 mb_strpos\(\) 函数](#)

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

参数

- **\$str** (*string*) – Input string

- **\$start** (*int*) – Position of first character
- **\$length** (*int*) – Maximum number of characters
- **\$encoding** (*string*) – Character set

返回 Portion of \$str specified by \$start and \$length or FALSE on failure

返回类型 string

更多信息, 请参考 PHP 手册中的 [mb_substr\(\)](#) 函数

标准函数

这一系列兼容性函数提供了一些高版本的 PHP 中才有的标准函数。

依赖性

- None

函数参考

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

参数

- **\$array** (*array*) – Array to fetch results from
- **\$column_key** (*mixed*) – Key of the column to return values from
- **\$index_key** (*mixed*) – Key to use for the returned values

返回 An array of values representing a single column from the input array

返回类型 array

更多信息, 请参考 PHP 手册中的 [array_column\(\)](#) 函数

array_replace(*array* \$array1[, ...])

参数

- **\$array1** (*array*) – Array in which to replace elements
- **...** (*array*) – Array (or multiple ones) from which to extract elements

返回 Modified array

返回类型 array

更多信息, 请参考 PHP 手册中的 [array_replace\(\)](#) 函数

array_replace_recursive(*array* \$array1[, ...])

参数

- **\$array1** (*array*) – Array in which to replace elements
- ... (*array*) – Array (or multiple ones) from which to extract elements

返回 Modified array

返回类型 array

更多信息, 请参考 PHP 手册中的 [array_replace_recursive\(\)](#) 函数

重要: 只有 PHP 原生的函数才可以检测到无穷递归, 如果你使用的是 PHP 5.3+, 使用时要担心引用!

hex2bin(\$data)

参数

- **\$data** (*array*) – Hexadecimal representation of data

返回 Binary representation of the given data

返回类型 string

更多信息, 请参考 PHP 手册中的 [hex2bin\(\)](#) 函数

quoted_printable_encode(\$str)

参数

- **\$str** (*string*) – Input string

返回 8bit-encoded string

返回类型 string

更多信息, 请参考 PHP 手册中的 [quoted_printable_encode\(\)](#) 函数

7.1.17 URI 路由

一般情况下, 一个 URL 字符串和它对应的控制器中类和方法是一一对应的关系。URL 中的每一段通常遵循下面的规则:

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

但是有时候, 你可能想改变这种映射关系, 调用一个不同的类和方法, 而不是 URL 中对应的那样。

例如, 假设你希望你的 URL 变成下面这样:

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

URL 的第二段通常表示方法的名称，但在上面的例子中，第二段是一个商品 ID，为了实现这一点，CodeIgniter 允许你重新定义 URL 的处理流程。

设置你自己的路由规则

路由规则定义在 `application/config/routes.php` 文件中，在这个文件中你会发现一个名为 `$route` 的数组，利用它你可以设置你自己的路由规则。在路由规则中你可以使用通配符或正则表达式。

通配符

一个典型的使用通配符的路由规则如下：

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

在一个路由规则中，数组的键表示要匹配的 URI，而数组的值表示要重定向的位置。上面的例子中，如果 URL 的第一段是字符串“product”，第二段是个数字，那么，将调用“catalog”类的“product_lookup”方法。

你可以使用纯字符串匹配，或者使用下面两种通配符：

(:num) 匹配只含有数字的一段。**(:any)** 匹配含有任意字符的一段。（除了‘/’字符，因为它是段与段之间的分隔符）

注解： 通配符实际上是正则表达式的别名，**:any** 会被转换为 `[^/]+`，**:num** 会被转换为 `[0-9]+`。

注解： 路由规则将按照它们定义的顺序执行，前面的规则优先级高于后面的规则。

注解： 路由规则并不是过滤器！设置一个这样的路由：`'foo/bar/(:num)'`，`Foo` 控制器的 `bar` 方法还是有可能通过一个非数字的参数被调用（如果这个路由也是合法的话）。

例子

这里是一些路由的例子：

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

URL 的第一段是单词“journals”时，将重定向到“blogs”类。

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

URL 包含 `blog/joe` 的话，将重定向到“blogs”类和“users”方法。ID 参数设为“34”。

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

URL 的第一段是“product”，第二段是任意字符时，将重定向到“catalog”类的“product_lookup”方法。

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

URL 的第一段是“product”，第二段是数字时，将重定向到“catalog”类的“product_lookup_by_id”方法，并将第二段的数字作为参数传递给它。

重要： 不要在前面或后面加反斜线（'/'）。

正则表达式

如果你喜欢，你可以在路由规则中使用正则表达式。任何有效的正则表达式都是允许的，包括逆向引用。

注解： 如果你使用逆向引用，你需要使用美元符号代替双斜线语法。

一个典型的使用正则表达式的路由规则看起来像下面这样：

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

上例中，一个类似于 products/shirts/123 这样的 URL 将会重定向到“shirts”控制器的“id_123”方法。

使用正则表达式，你还可以匹配含有反斜线字符（'/'）的段，它通常来说是多个段之间的分隔符。

例如，当一个用户访问你的 Web 应用中的某个受密码保护的页面时，如果他没有登陆，会先跳转到登陆页面，你希望他们在成功登陆后重定向回刚才那个页面，那么这个例子会很有用：

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

如果你还不知道正则表达式，可以访问 *regular-expressions.info* <<http://www.regular-expressions.info/>> 开始学习一下。

注解： 你也可以在你的路由规则中混用通配符和正则表达式。

回调函数

如果你正在使用的 PHP 版本高于或等于 5.3，你还可以在路由规则中使用回调函数来处理逆向引用。例如：

```
$route['products/([a-zA-Z]+)/edit/(\d+)'] = function ($product_type, $id)
{
```

```
return 'catalog/product_edit/' . strtolower($product_type) . '/' . $id;
};
```

在路由中使用 HTTP 动词

还可以在你的路由规则中使用 HTTP 动词（请求方法），当你在创建 RESTful 应用时特别有用。你可以使用标准的 HTTP 动词（GET、PUT、POST、DELETE、PATCH），也可以使用自定义的动词（例如：PURGE），不区分大小写。你需要做的就是路由数组后面再加一个键，键名为 HTTP 动词。例如：

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

上例中，当发送 PUT 请求到 “products” 这个 URI 时，将会调用 `Product::insert()` 方法。

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

当发送 DELETE 请求到第一段为 “products”，第二段为数字这个 URL 时，将会调用 `Product::delete()` 方法，并将数字作为第一个参数。

当然，使用 HTTP 动词是可选的。

保留路由

有下面三个保留路由：

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

This route points to the action that should be executed if the URI contains no data, which will be the case when people load your root URL. The setting accepts a **controller/method** value and `index()` would be the default method if you don't specify one. In the above example, it is `Welcome::index()` that would be called.

注解： You can NOT use a directory as a part of this setting!

You are encouraged to always have a default route as otherwise a 404 page will appear by default.

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

这个路由表示当用户请求了一个不存在的页面时该加载哪个控制器，它将会覆盖默认的 404 错误页面。Same per-directory rules as with ‘default_controller’ apply here as well. `show_404()` 函数不会受影响，它还是会继续加载 `application/views/errors/` 目录下的默认的 `error_404.php` 文件。

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

从它的布尔值就能看出来这其实并不是一个路由，这个选项可以自动的将 URL 中的控制器和方法中的连字符（‘-’）转换为下划线（‘_’），当你需要这样时，它可以让你

少写很多路由规则。由于连字符不是一个有效的类名或方法名，如果你不使用它的话，将会引起一个严重错误。

重要： 保留的路由规则必须位于任何一般的通配符或正则路由的前面。

7.1.18 错误处理

CodeIgniter 可以通过下面介绍的方法来在你的应用程序中生成错误报告。另外，还有一个错误日志类用来将错误或调试信息保存到文本文件中。

注解： CodeIgniter 默认将显示所有的 PHP 错误，你可能在开发结束之后改变该行为。在你的 `index.php` 文件的顶部有一个 `error_reporting()` 函数，通过它可以修改错误设置。当发生错误时，禁用错误报告并不会阻止向日志文件写入错误信息。

和 CodeIgniter 中的大多数系统不同，错误函数是一个可以在整个应用程序中使用的简单接口，这让你在使用该函数时不用担心类或方法的作用域的问题。

当任何一处核心代码调用 `exit()` 时，CodeIgniter 会返回一个状态码。这个状态码和 HTTP 状态码不同，是用来通知其他程序 PHP 脚本是否成功运行的，如果运行不成功，又是什么原因导致了脚本退出。状态码的值被定义在 `application/config/constants.php` 文件中。状态码在 CLI 形式下非常有用，可以帮助你的服务器跟踪并监控你的脚本。

下面的函数用于生成错误信息：

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

参数

- **\$message** (*mixed*) – Error message
- **\$status_code** (*int*) – HTTP Response status code
- **\$heading** (*string*) – Error page heading

返回类型 void

该函数使用下面的错误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 `$status_code` 将决定发送什么 HTTP 状态码。如果 `$status_code` 小于 100，HTTP 状态码将被置为 500，退出状态码将被置为 `$status_code + EXIT_AUTO_MIN`。如果它的值大于 `EXIT_AUTO_MAX` 或者如果 `$status_code` 大于等于 100，退出状态码将被置为 `EXIT_ERROR`。详情可查看 `application/config/constants.php` 文件。

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

参数

- **\$page** (*string*) – URI string
- **\$log_error** (*bool*) – Whether to log the error

返回类型 void

该函数使用下面的错误模板来显示 404 错误信息:

application/views/errors/html/error_404.php

或:

application/views/errors/cli/error_404.php

传递给该函数的字符串代表的是找不到的文件路径。退出状态码将设置为 `EXIT_UNKNOWN_FILE`。注意如果找不到控制器 CodeIgniter 将自动显示 404 错误信息。

默认 CodeIgniter 会自动将 `show_404()` 函数调用记录到错误日志中。将第二个参数设置为 `FALSE` 将跳过记录日志。

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

参数

- **\$level** (*string*) – Log level: 'error', 'debug' or 'info'
- **\$message** (*string*) – Message to log
- **\$php_error** (*bool*) – Whether we're logging a native PHP error message

返回类型 void

该函数用于向你的日志文件中写入信息，第一个参数你必须提供三个信息级别中的一个，用于指定记录的是什么类型的信息（调试，错误和一般信息），第二个参数为信息本身。

示例:

```
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.');
```

有三种信息类型:

1. 错误信息。这些是真正的错误，例如 PHP 错误或用户错误。

2. 调试信息。这些信息帮助你调试程序, 例如, 你可以在一个类初始化的地方记录下来作为调试信息。
3. 一般信息。这些是最低级别的信息, 简单的给出程序运行过程中的一些信息。

注解: 为了保证日志文件被正确写入, `logs/` 目录必须设置为可写的。此外, 你必须设置 `application/config/config.php` 文件中的 “threshold” 参数, 举个例子, 例如你只想记录错误信息, 而不想记录另外两种类型的信息, 可以通过这个参数来控制。如果你将该参数设置为 0, 日志就相当于被禁用了。

7.1.19 网页缓存

CodeIgniter 可以让你通过缓存页面来达到更好的性能。

尽管 CodeIgniter 已经相当高效了, 但是网页中的动态内容、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。依靠网页缓存, 你的网页可以达到近乎静态网页的加载速度, 因为程序的输出结果已经保存下来了。

缓存是如何工作的?

可以针对到每个独立的页面进行缓存, 并且你可以设置每个页面缓存的更新时间。当页面第一次加载时, 缓存将被写入到 `application/cache` 目录下的文件中去。之后请求这个页面时, 就可以直接从缓存文件中读取内容并输出到用户的浏览器。如果缓存过期, 会在输出之前被删除并重新刷新。

开启缓存

将下面的代码放到任何一个控制器的方法内, 你就可以开启缓存了:

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

其中 `$n` 是缓存更新的时间 (单位分钟)。

上面的代码可以放在方法的任何位置, 它出现的顺序对缓存没有影响, 所以你可以把它放到任何你认为合理的地方。一旦该代码被放在方法内, 你的页面就开始被缓存了。

重要: 由于 CodeIgniter 存储缓存的方式, 只有通过 `view` 输出的页面才能缓存。

重要: 如果你修改了可能影响页面输出的配置, 你需要手工删除掉你的缓存文件。

注解: 在写入缓存文件之前, 你需要把 `application/cache/` 目录的权限设置为可写。

删除缓存

如果你不再需要缓存某个页面, 你可以删除掉该页面上的缓存代码, 这样它在过期之后就不会刷新了。

注解: 删除缓存代码之后并不是立即生效, 必须等到缓存过期才会生效。

如果你需要手工删除缓存, 你可以使用 `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');
```

7.1.20 程序分析

分析器类会在页面下方显示基准测试结果, 运行过的 SQL 语句, 以及 `$_POST` 数据。这些信息有助于开发过程中的调试和优化。

初始化类

重要: 这个类无须初始化, 如果已按照下面的方式启用, 他将被输出类自动加载。

启用分析器

要启用分析器, 你可以在你的控制器方法的任何位置添加一行下面的代码:

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

当启用之后, 将会生成一份报告插入到页面的最底部。

使用下面的方法禁用分析器:

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

设置基准测试点

为了让分析器编译并显示你的基准测试数据, 你必须使用特定的语法来命名基准点。

请阅读[基准测试类](#)中关于设置基准点的资料。

启用和禁用分析器中的字段

分析器中的每个字段都可以通过设置相应的控制变量为 TRUE 或 FALSE 来启用或禁用。有两种方法来实现，其中的一种方法是：在 *application/config/profiler.php* 文件里设置全局的默认值。

例如：

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

另一种方法是：在你的控制器里通过调用输出类的 `set_profiler_sections()` 函数来覆盖全局设置和默认设置：

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

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

下表列出了可用的分析器字段和用来访问这些字段的 key 。

Key	Description	De- fault
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	被请求的控制器类和调用的方法	TRUE
get	请求中的所有 GET 数据	TRUE
http_headers	本次请求的 HTTP 头部	TRUE
memory_usage	本次请求消耗的内存（单位字节）	TRUE
post	请求中的所有 POST 数据	TRUE
queries	列出所有执行的数据库查询，以及执行时间	TRUE
uri_string	本次请求的 URI	TRUE
session_data	当前会话中存储的数据	TRUE
query_toggle_count	指定显示多少个数据库查询，剩下的则默认折叠起来	25

注解： 在你的数据库配置文件中禁用 *save_queries* 参数也可以禁用数据库查询相关的分析器，上面说的 ‘queries’ 字段就没用了。你可以通过 `$this->db->save_queries = TRUE`；来覆写该设置。另外，禁用这个设置也会导致你无法查看查询语句以及 *last_query <database/helpers>* 。

7.1.21 以 CLI 方式运行

除了从浏览器中通过 URL 来调用程序的控制器之外，你也可以通过 CLI（命令行界面）的方式来调用。

目录

- 以 CLI 方式运行
 - 什么是 CLI ?
 - 为什么使用命令行?
 - 让我们试一试: Hello World!
 - 就这么简单!

什么是 CLI ?

CLI (命令行界面) 是一种基于文本的和计算机交互的方式。更多信息, 请查看[维基百科](#)。

为什么使用命令行?

虽然不是很明显, 但是有很多情况下我们需要使用命令行来运行 CodeIgniter。

- 使用 `cron` 定时运行任务, 而不需要使用 `wget` 或 `curl`
- 通过函数 `is_cli()` 的返回值来让你的 `cron` 页面不能通过 URL 访问到
- 制作交互式的任务, 例如: 设置权限, 清除缓存, 备份等等
- 与其他语言进行集成, 例如可以通过 C++ 调用一条指令来运行你模型中的代码。

让我们试一试: Hello World!

让我们先创建一个简单的控制器, 打开你的文本编辑器, 新建一个文件并命名为 `Tools.php`, 然后输入如下的代码:

```
<?php
class Tools extends CI_Controller {

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

然后将文件保存到 `application/controllers/` 目录下。

现在你可以通过类似下面的 URL 来访问它:

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

或者, 我们可以通过 CLI 来访问。在 Mac/Linux 下你可以打开一个终端, 在 Windows 下你可以打开“运行”, 然后输入“`cmd`”, 进入 CodeIgniter 项目所在的目录。

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

如果你操作正确, 你应该会看到 *Hello World!* 。

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

这里我们传一个参数给它, 这和使用 URL 参数是一样的。"John Smith" 被作为参数传入并显示出:

```
Hello John Smith!
```

就这么简单!

简单来说, 这就是你需要知道的关于如何在命令行中使用控制器的所有事情了。记住, 这只是一个普通的控制器, 所以路由和 `remap` 也照样工作。

7.1.22 管理你的应用程序

默认情况下, CodeIgniter 假设你只有一个应用程序, 被放置在 *application/* 目录下。但是, 你完全可以拥有多个程序并让它们共享一份 CodeIgniter。你甚至也可以对你的应用程序目录改名, 或将其移到其他的位置。

重命名应用程序目录

如果你想重命名应用程序目录, 你只需在重命名之后打开 `index.php` 文件将 `$application_folder` 变量改成新的名字:

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

移动应用程序目录

你可以将你的应用程序目录移动到除 Web 根目录之外的其他位置, 移到之后你需要打开 `index.php` 文件将 `$application_folder` 变量改成新的位置 (使用 `**` 绝对路径 `**`) :

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

在一个 CodeIgniter 下运行多个应用程序

如果你希望在一个 CodeIgniter 下管理多个不同的应用程序, 只需简单的将 `application` 目录下的所有文件放置到每个应用程序独立的子目录下即可。

例如, 你要创建两个应用程序: "foo" 和 "bar", 你可以像下面这样组织你的目录结构:

```

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/

```

要选择使用某个应用程序时，你需要打开 `index.php` 文件然后设置 `$application_folder` 变量。例如，选择使用“foo”这个应用，你可以这样：

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

注解： 你的每一个应用程序都需要一个它自己的 `index.php` 文件来调用它，你可以随便对 `index.php` 文件进行命名。

7.1.23 处理多环境

开发者常常希望当系统运行在开发环境或生产环境中时能有不同的行为，例如，在开发环境如果程序能输出详细的错误信息将非常有用，但是在生产环境这将造成一些安全问题。

ENVIRONMENT 常量

CodeIgniter 默认使用 `$_SERVER['CI_ENV']` 的值作为 `ENVIRONMENT` 常量，如果 `$_SERVER['CI_ENV']` 的值没有设置，则设置为 `'development'`。在 `index.php` 文件的顶部，你可以看到：

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

`$_SERVER['CI_ENV']` 的值可以在 `.htaccess` 文件或 Apache 的配置文件中使 `SetEnv` 命令进行设置，Nginx 或其他 Web 服务器也有类似的设置方法。或者你可以直接删掉这个逻辑，根据服务器的 IP 地址来设置该常量。

使用这个常量，除了会影响到一些基本的框架行为外（见下一节），你还可以在开发过程中使用它来区分当前运行的是什么环境。

对默认框架行为的影响

CodeIgniter 系统中有几个地方用到了 `ENVIRONMENT` 常量。这一节将描述它对框架行为有哪些影响。

错误报告

如果将 `ENVIRONMENT` 常量设置为 ‘development’，当发生 PHP 错误时错误信息会显示到浏览器上。与之相对的，如果将常量设置为 ‘production’ 错误输出则会被禁用。在生产环境禁用错误输出是个不错的安全实践。

配置文件

另外，CodeIgniter 还可以根据不同的环境加载不同的配置文件，这在处理例如不同环境下有着不同的 API Key 的情况时相当有用。这在 [配置类](#) 文档中的“环境”一节有着更详细的介绍。

7.1.24 在视图文件中使用 PHP 替代语法

如果你不使用 CodeIgniter 的模板引擎，那么你就只能在视图文件中使用纯 PHP 语法了。为了精简视图文件，使其更可读，建议你在写控制结构或 `echo` 语句时使用 PHP 的替代语法。如果你还不熟悉这个语法，下面将介绍如何通过这个语法来消灭你代码中的大括号和 `echo` 语句。

自动短标记支持

注解： 如果你发现本页所介绍的语法在你的服务器上行不通，那么有可能是你的 `PHP.ini` 文件中禁用了“短标记”。CodeIgniter 可以动态的重写所有的短标记，这样即使你的服务器不支持你也可以使用短标记语法。这个特性可以在 `config/config.php` 文件中启用。

请注意，如果你使用了这个特性，当你的视图文件发生 PHP 错误时，错误信息和行号将无法准确显示，因为所有的错误都显示成 `eval()` 错误。

Echo 替代语法

通常情况，你会使用下面的方法来打印一个变量：

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

使用替代语法，你可以写成这样：

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

控制结构的替代语法

像 `if`、`for`、`foreach`、`while` 这样的控制结构也可以写成精简的格式。下面以 `foreach` 举例：

```
<ul>

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

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

<?php endforeach; ?>

</ul>
```

注意这里没有任何括号。所有的结束括号被替换成了 `endforeach`。上面说的那些控制结构也都有这相似的结束标志：`endif`、`endfor`、`endforeach` 和 `endwhile`。

另外要注意的一点是，每个分支结构的后面都要跟一个冒号，而不是分号（除最后一个），这是非常重要的一点！

这里是另一个例子，使用了 `if/elseif/else`，注意看冒号的位置：

```
<?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; ?>
```

7.1.25 安全

这篇文章将介绍一些基本的关于 Web 安全的“最佳实践”，并详细说明了 CodeIgniter 内部的安全特性。

URI 安全

CodeIgniter 严格限制 URI 中允许出现的字符，以此来减少恶意数据传到你的应用程序的可能性。URI 中只允许包含一些字符：

- 字母和数字
- 波浪符：~
- 百分号：%
- 句号：.
- 分号：:

- 下划线: -
- 连字号: -
- 空格

Register_globals

在系统初始化期间, 如果发现任何 `$_GET`、`$_POST`、`$_REQUEST` 和 `$_COOKIE` 数组中的键值变成了全局变量, 则删除该变量。

这个过程和设置 `register_globals = off` 效果是一样的。(译注: 阅读[这里](#)了解 `register_globals` 设置)

display_errors

在生产环境下, 一般都是通过将 `display_errors` 标志设置为 0 来禁用 PHP 的错误报告。这可以阻止原生的 PHP 错误被显示到页面上, 错误中可能会包含潜在的敏感信息。

在 CodeIgniter 中, 可以将 `index.php` 文件中的 **ENVIRONMENT** 常量设置为 `'production'`, 这样也可以关闭这些错误信息。在开发模式下, 建议将它设置为 `'development'`。关于不同环境之间的区别可以阅读[处理多环境](#)页面了解更多。

magic_quotes_runtime

在系统初始化期间, `magic_quotes_runtime` 指令会被禁用, 这样当你在从数据库中获取数据时就不用再去除反斜线了。

最佳实践

在你的应用程序处理任何数据之前, 无论这些数据是来自于提交的表单 POST, 还是来自 COOKIE、URI、XML-RPC, 或者甚至是来自于 SERVER 数组, 你都应该使用下面这三步来处理:

1. 验证数据类型是否正确, 以及长度、大小等等
2. 过滤不良数据
3. 在提交到数据库或者显示到浏览器之前对数据进行转义

CodeIgniter 提供了以下的方法和技巧来帮你处理该过程:

XSS 过滤

CodeIgniter 自带有一个 XSS 过滤器, 这个过滤器可以查找一些 XSS 的常用技术, 例如向你的数据中嵌入恶意的 JavaScript 脚本, 劫持 cookie 信息或其他一些技术。XSS 过滤器在[这里](#)有更详细的描述。

注解: XSS 过滤 只应该在输出数据时使用。对输入的数据进行过滤可能会在无意中
对数据造成修改, 例如过滤密码中的特殊字符, 这样会降低安全性, 而不是提高安全
性。

CSRF 保护

CSRF (Cross-Site Request Forgery, 跨站请求伪造) 是攻击者骗取受害者在不知情的
情况下提交请求的攻击方式。

CodeIgniter 提供了对 CSRF 的保护, 会在每个非 GET HTTP 请求时自动触发,
当然前提是你使用某种方式来创建表单, 这在[安全类](#)文档中有进一步的解释。

密码处理

在你的应用程序中正确处理密码是非常关键的。

但是不幸的是, 许多开发者并不知道怎么去做, 而且网络上充斥着大量过时的甚至
错误的建议, 提供不了任何帮助。

我们提供了一个清单来帮助你, 告诉你什么该做, 什么不该做。

- 绝不要以明文存储密码。

永远使用 **哈希算法** 来处理密码。

- 绝不要使用 Base64 或其他编码方式来存储密码。

这和以明文存储密码是一样的, 使用 **哈希**, 而不要使用 **编码**。

编码以及加密, 都是双向的过程, 而密码是保密的, 应该只被它的所有者知道,
这个过程必须是单向的。哈希正是用于做这个的, 从来没有解哈希这种说法, 但
是编码就存在解码, 加密就存在解密。

- 绝不要使用弱哈希或已被破解的哈希算法, 像 MD5 或 SHA1。

这些算法太老了, 而且被证明存在缺陷, 它们一开始就并不是为了保存密码而设
计的。

另外, 绝不要自己发明算法。

只使用强密码哈希算法, 例如 BCrypt, 在 PHP 自己的 **密码哈希** 函数中也是使
用它。

即使你的 PHP 版本不是 5.5+, 也请使用它们, CodeIgniter 为你提供了这些算
法, 只要你的 PHP 版本是 5.3.7 以上都可以使用。(如果不满足这点要求, 那么
请升级你的 PHP)

如果你连升级 PHP 也无法做到, 那么使用 `hash_pbkdf()` <[http://php.net/
hash_pbkdf2](http://php.net/hash_pbkdf2)> 吧, 为实现兼容性我们提供了这个函数。

- 绝不要以明文形式显示或发送密码。

即使是对密码的所有者也应该这样。如果你需要“忘记密码”的功能，可以随机生成一个新的一次性的（这点很重要）密码，然后把这个密码发送给用户。

- 绝不要对用户的密码做一些没必要的限制。

如果你使用除 BCrypt（它有最多 72 字符的限制）之外的其他哈希算法，你应该设置一个相对长一点的密码长度（例如 1024 字符），这样可以缓解 DoS 攻击。

但是除此之外，对密码的其他限制诸如密码中只允许使用某些字符，或者密码中不允许包含某些字符，就没有任何意义了。

这样做不仅不会提高安全性，反而 **降低了** 安全性，而且真的没有任何理由需要这样做。只要你对密码进行哈希处理了，那么无论是技术上，还是在存储上都没有任何限制。

验证输入数据

CodeIgniter 有一个 [表单验证类](#) 用于帮助你验证、过滤以及预处理你的数据。

就算这个类不适用于你的使用场景，那么你也应该确保对输入数据进行验证过滤。例如，你希望接受一个数字型的参数，你可以使用 `is_numeric()` 或 `ctype_digit()` 函数来检查一下。永远将数据限制在你运行的范围内。

记住，不仅要验证 `$_POST` 和 `$_GET` 变量，而且也不要放过 cookie、user-agent 以及 **其他所有的不是直接由你的代码生成的数据**。

插入数据库之前对数据进行转义

永远不要不做转义就将数据插入到数据库，更多信息，可以阅读[数据库查询](#)这一节。

隐藏你的文件

另一个很好的安全实践是，在你的 `webroot` 目录（通常目录名为“`htdocs/`”）下只保留 `index.php` 文件和“`assets`”目录（用于存放 js、css、图片等静态资源）。只需要这些文件能从 Web 上访问就可以了。

允许你的访问者访问其他位置可能潜在的导致他们访问一些敏感数据或者执行脚本等等。

如果你不允许这样做，你可以使用 `.htaccess` 文件来限制对这些资源的访问。

CodeIgniter 在每个目录下放置了一个 `index.html` 文件，试图隐藏这些敏感数据，但是要记住的是，这对于防止一个真正的攻击者来说并不够。

7.1.26 PHP 开发规范

CodeIgniter 的开发遵循本页所描述的编码规范，我们也推荐在你自己的应用程序开发中使用这些规范，但不做强求。

目录

- PHP 开发规范
 - 文件格式
 - * TextMate
 - * BBEdit
 - PHP 结束标签
 - 文件的命名
 - 类和方法的命名
 - 变量的命名
 - 注释
 - 常量
 - TRUE、FALSE 和 NULL
 - 逻辑操作符
 - 对返回值进行比较以及类型转换
 - 调试代码
 - 文件中的空格
 - 兼容性
 - 一个类一个文件
 - 空格
 - 换行
 - 代码缩进
 - 中括号和小括号内的空格
 - 本地化文本
 - 私有方法和变量
 - PHP 错误
 - 短标记
 - 每行只有一条语句
 - 字符串
 - SQL 查询
 - 缺省的函数参数

文件格式

文件应该保存为 Unicode (UTF-8) 编码格式, 不要使用 字节序标记 (BOM), 和 UTF-16 和 UTF-32 不一样, UTF-8 编码格式的文件不需要指定字节序。而且 BOM 会在 PHP 的输出中产生副作用, 它会阻止应用程序设置它的头信息。另外, 所有的换行符应该使用 Unix 格式换行符 (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

1. 打开应用程序设置

2. 点击“高级”，切换到“保存”标签页。
3. 在“文件编码”中，选择“UTF-8（推荐）”
4. 在“换行符”中，选择“LF（推荐）”
5. 可选：如果你想对现有文件也能自动作此设置，勾选“同时应用到已有文件”选项

BBEdit

1. 打开应用程序设置
2. 选择左侧的“文本编码”
3. 在“新文档的默认编码”，选择“Unicode (UTF-8, no BOM)”
4. 可选：在“如果无法检测文件编码，使用...”，选择“Unicode (UTF-8, no BOM)”
5. 选择左侧的“文本文件”
6. 在“默认的换行符”中，选择“Mac OS X and Unix (LF)”

PHP 结束标签

PHP 结束标签 `?>` 对于 PHP 解析器来说是可选的，但是只要使用了，结束标签之后的空格有可能导致不想要的输出，这个空格可能是由开发者或者用户又或者 FTP 应用程序引入的，甚至可能导致出现 PHP 错误，如果配置了不显示 PHP 错误，就会出现空白页面。基于这个原因，所有的 PHP 文件将不使用结束标签，而是以一个空行代替。

文件的命名

类文件的命名必须以大写字母开头，其他文件（配置文件，视图，一般的脚本文件等）的命名是全小写。

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, 类文件的名称必须和类的名称保持一致, 例如, 如果你有一个类名为 *Myclass*, 那么文件名应该是 **Myclass.php**。

类和方法的命名

类名必须以大写字母开头, 多个单词之间使用下划线分割, 不要使用驼峰命名法。

错误的:

```
class superclass
class SuperClass
```

正确的:

```
class Super_class

class Super_class {

    public function __construct()
    {

    }

}
```

类的方法应该使用全小写, 并且应该明确指出该方法的功能, 最好包含一个动词。避免使用冗长的名称, 多个单词之间使用下划线分割。

错误的:

```
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
```

正确的:

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

变量的命名

变量的命名规则和类方法的命名规则非常接近, 使用全小写, 使用下划线分割, 并且应该明确指出该变量的用途。非常短的无意义的变量只应该在 for 循环中作为迭代器使用。

错误的:

```
$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
```

正确的:

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

注释

通常情况下, 应该多写点注释, 这不仅可以向那些缺乏经验的程序员描述代码的流程和意图, 而且当你几个月后再回过头来看自己的代码时仍能帮你很好的理解。注释并没有强制规定的格式, 但是我们建议以下的形式。

DocBlock 风格的注释, 写在类、方法和属性定义的前面, 可以被 IDE 识别:

```
/**
 * 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();
```

单行注释应该和代码合在一起, 大块的注释和代码之间应该留一个空行。

```
// 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

常量遵循和变量一样的命名规则，除了它需要全部大写。尽量使用 CodeIgniter 已经定义好的常量，如：SLASH、LD、RD、PATH_CACHE 等。

错误的:

```
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
```

正确的:

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

TRUE、FALSE 和 NULL

TRUE、FALSE 和 NULL 这几个关键字全部使用大写。

错误的:

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

正确的:

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


逻辑操作符

不要使用 `||` 操作符，它在一些设备上看不清（可能看起来像是数字 11），使用 `&&` 操作符比使用 `AND` 要好一点，但是两者都可以接受。另外，在 `!` 操作符的前后都应该加一个空格。

错误的:

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

正确的:

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

对返回值进行比较以及类型转换

有一些 PHP 函数在失败时返回 `FALSE`，但是也可能会返回 “” 或 0 这样的有效值，这些值在松散类型比较时和 `FALSE` 是相等的。所以当你在条件中使用这些返回值作比较时，一定要使用严格类型比较，确保返回值确实是你想要的，而不是松散类型的其他值。

在检查你自己的返回值和变量时也要遵循这种严格的方式，必要时使用 `===` 和 `!`。

错误的:

```
// 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)
```

正确的:

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

错误的:

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

    }
}
```

正确的:

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

    }
}
```

另外关于 [类型转换](#) 的信息也将很有用。类型转换会对变量产生一点轻微的影响，但可能也是期望的。例如 NULL 和布尔值 FALSE 会转换为空字符串，数字 0（和其他数字）将会转换为数字字符串，布尔值 TRUE 会变成“1”：

```
$str = (string) $str; // cast $str as a string
```

调试代码

不要在你的提交中包含调试代码，就算是注释掉了也不行。像 `var_dump()`、`print_r()`、`die()` 和 `exit()` 这样的函数，都不应该包含在你的代码里，除非它们用于除调试之外的其他特殊用途。

文件中的空格

PHP 起始标签的前面和结束标签的后面都不要留空格，输出是被缓存的，所以如果你的文件中有空格的话，这些空格会在 CodeIgniter 输出它的内容之前被输出，从而导致错误，而且也会导致 CodeIgniter 无法发送正确的头信息。

兼容性

CodeIgniter 推荐使用 PHP 5.4 或更新版本，但是它还得同时兼容 PHP 5.2.4。你的代码要么提供适当的回退来兼容这点，要么提供一些可选的功能，当不兼容时能安静的退出而不影响用户的程序。

另外，不要使用那些需要额外安装的库的 PHP 函数，除非你能给出当该函数不存在时，有其他的函数能替代它。

一个类一个文件

除非几个类是 * 紧密相关的 *，否则每个类应该单独使用一个文件。在 CodeIgniter 中一个文件包含多个类的一个例子是 Xmlrpc 类文件。

空格

在代码中使用制表符（tab）来代替空格，这虽然看起来是一件小事，但是使用制表符代替空格，可以让开发者阅读你代码的时候，可以根据他们的喜好在他们的程序中自定义缩进。此外还有一个好处是，这样文件可以更紧凑一点，也就是本来是四个空格字符，现在只要一个制表符就可以了。

换行

文件必须使用 Unix 的换行格式保存。这对于那些在 Windows 环境下的开发者可能是个问题, 但是不管在什么环境下, 你都应该确认下你的文本编辑器已经配置好使用 Unix 换行符了。

代码缩进

使用 Allman 代码缩进风格。除了类的定义之外, 其他的所有大括号都应该独占一行, 并且和它对应的控制语句保持相同的缩进。

错误的:

```
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() {
    // ...
}
```

正确的:

```
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()
{
    // ...
}

```

中括号和小括号内的空格

一般情况下，使用中括号和小括号的时候不应该使用多余的空格。唯一的例外是，在那些接受一个括号和参数的 PHP 的控制结构（declare、do-while、elseif、for、foreach、if、switch、while）的后面应该加一个空格，这样做可以和函数区分开来，并增加可读性。

错误的:

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

正确的:

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

错误的:

```
function foo ( $bar )
{
}

```

正确的:

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

}
```

错误的:

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

正确的:

```
foreach ( $query->result() as $row) // single space following PHP control structures, but no
```

本地化文本

CodeIgniter 的类库应该尽可能的使用相应的语言文件。

错误的:

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

正确的:

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

私有方法和变量

那些只能在内部访问的方法和变量, 例如供共有方法使用的那些工具方法或辅助函数, 应该以下划线开头。

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

PHP 错误

运行代码时不应该出现任何错误信息, 并不是把警告和提示信息关掉来满足这一点。例如, 绝不要直接访问一个你没设置过的变量 (例如, `$_POST` 数组), 你应该先使用 `isset()` 函数判断下。

确保你的开发环境对所有人都开启了错误报告, PHP 环境的 `display_errors` 参数也开启了, 你可以通过下面的代码来检查:

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

有些服务器上 `display_errors` 参数可能是禁用的, 而且你没有权限修改 `php.ini` 文件, 你可以使用下面的方法来启用它:

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

注解: 使用 `ini_set()` 函数在运行时设置 `display_errors` 参数和通过 `php.ini` 配置文件来设置是不一样的, 换句话说, 当出现致命错误 (fatal errors) 时, 这种方法没用。

短标记

使用 PHP 的完整标记, 防止服务器不支持短标记 (`short_open_tag`) 参数。

错误的:

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

```
<?=$foo?>
```

正确的:

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

注解: PHP 5.4 下 `<?=>` 标记是永远可用的。

每行只有一条语句

切记不要在同一行内写多条语句。

错误的:

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

正确的:

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

字符串

字符串使用单引号引起来, 当字符串中有变量时使用双引号, 并且使用大括号将变量包起来。另外, 当字符串中有单引号时, 也应该使用双引号, 这样就不用使用转义符。

错误的:

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

正确的:

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

SQL 查询

SQL 关键字永远使用大写: SELECT、INSERT、UPDATE、WHERE、AS、JOIN、ON、IN 等。

考虑到易读性, 把长的查询分成多行, 最好是每行只有一个从句或子从句。

错误的:

```
// 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_
...where foo != 'oof' and baz != 'zab' order by foobaz limit 5, 100");
```

正确的:

```
$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");
```

缺省的函数参数

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)
```

8.1 类库参考

8.1.1 基准测试类

CodeIgniter 有一个一直都是启用状态的基准测试类，用于计算两个标记点之间的时间差。

注解： 该类是由系统自动加载，无需手动加载。

另外，基准测试总是在框架被调用的那一刻开始，在输出类向浏览器发送最终的视图之前结束。这样可以显示出整个系统执行的精确时间。

- 使用基准测试类
 - 在性能分析器中使用基准测试点
 - 显示总执行时间
 - 显示内存占用
- 类参考

使用基准测试类

基准测试类可以在你的控制器、视图 以及模型 中使用。

使用流程如下：

1. 标记一个起始点
2. 标记一个结束点
3. 使用 `elapsed_time` 函数计算时间差。

这里是个真实的代码示例：


```
$this->benchmark->mark('code_start');

// Some code happens here

$this->benchmark->mark('code_end');

echo $this->benchmark->elapsed_time('code_start', 'code_end');
```

注解: “code_start” 和 “code_end” 这两个单词是随意的，它们只是两个用于标记的单词而已，你可以任意使用其他你想使用的单词，另外，你也可以设置多个标记点。看如下示例：

```
$this->benchmark->mark('dog');

// Some code happens here

$this->benchmark->mark('cat');

// More code happens here

$this->benchmark->mark('bird');

echo $this->benchmark->elapsed_time('dog', 'cat');
echo $this->benchmark->elapsed_time('cat', 'bird');
echo $this->benchmark->elapsed_time('dog', 'bird');
```

在性能分析器中使用基准测试点

如果你希望你的基准测试数据显示在性能分析器中，那么你的标记点就需要成对出现，而且标记点名称需要以 `_start` 和 `_end` 结束，每一对的标记点名称应该一致。例如：

```
$this->benchmark->mark('my_mark_start');

// Some code happens here...

$this->benchmark->mark('my_mark_end');

$this->benchmark->mark('another_mark_start');

// Some more code happens here...

$this->benchmark->mark('another_mark_end');
```

阅读性能分析器 页面了解更多信息。

显示总执行时间

如果你想显示从 CodeIgniter 运行开始到最终结果输出到浏览器之间花费的总时间，只需简单的将下面这行代码放入你的视图文件中：

```
<?php echo $this->benchmark->elapsed_time();?>
```

你大概也注意到了，这个方法和上面例子中的介绍的那个计算两个标记点之间时间差的方法是一样的，只是不带任何参数。当不设参数时，CodeIgniter 在向浏览器输出最终结果之前不会停止计时，所以无论你在哪里使用该方法，输出的计时结果都是总执行时间。

如果你不喜欢纯 PHP 语法的话，也可以在你的视图中使用另一种伪变量的方式来显示总执行时间：

```
{elapsed_time}
```

注解： 如果你想在你的控制器方法中进行基准测试，你需要设置你自己的标记起始点和结束点。

显示内存占用

如果你的 PHP 在安装时使用了 `-enable-memory-limit` 参数进行编译，你就可以在你的视图文件中使用下面这行代码来显示整个系统所占用的内存大小：

```
<?php echo $this->benchmark->memory_usage();?>
```

注解： 这个方法只能在视图文件中使用，显示的结果代表整个应用所占用的内存大小。

如果你不喜欢纯 PHP 语法的话，也可以在你的视图中使用另一种伪变量的方式来显示占用的内存大小：

```
{memory_usage}
```

类参考

class CI_Benchmark

mark(\$name)

参数

- **\$name** (*string*) – the name you wish to assign to your marker

返回类型 void

设置一个基准测试的标记点。

```
elapsed_time([$point1 = '', $point2 = '', $decimals = 4]))
```

参数

- **\$point1** (*string*) – a particular marked point
- **\$point2** (*string*) – a particular marked point
- **\$decimals** (*int*) – number of decimal places for precision

返回 Elapsed time**返回类型** string

计算并返回两个标记点之间的时间差。

如果第一个参数为空, 方法将返回 {elapsed_time} 伪变量。这用于在视图中显示整个系统的执行时间, 输出类将在最终输出时使用真实的总执行时间替换掉这个伪变量。

```
memory_usage()
```

返回 Memory usage info**返回类型** string

只是简单的返回 {memory_usage} 伪变量。

该方法可以在视图的任意位置使用, 直到最终输出页面时输出类 才会将真实的值替换掉这个伪变量。

8.1.2 缓存驱动器

CodeIgniter 提供了几种最常用的快速缓存的封装, 除了基于文件的缓存, 其他的缓存都需要对服务器进行特殊的配置, 如果配置不正确, 将会抛出一个致命错误异常 (Fatal Exception)。

- 使用示例
- 类参考
- 驱动器
 - 可选 PHP 缓存 (APC)
 - 基于文件的缓存
 - Memcached 缓存
 - WinCache 缓存
 - Redis 缓存
 - 虚拟缓存 (Dummy Cache)

使用示例

下面的示例代码用于加载缓存驱动器, 使用 APC 作为缓存, 如果 APC 在服务器环境下不可用, 将降级到基于文件的缓存。

```

$this->load->driver('cache', array('adapter' => 'apc', 'backup' => 'file'));

if ( ! $foo = $this->cache->get('foo'))
{
    echo 'Saving to the cache!<br />';
    $foo = 'foobarbaz!';

    // Save into the cache for 5 minutes
    $this->cache->save('foo', $foo, 300);
}

echo $foo;

```

你也可以设置 **key_prefix** 参数来给缓存名添加前缀，当你在同一个环境下运行多个应用时，它可以避免冲突。

```

$this->load->driver('cache',
    array('adapter' => 'apc', 'backup' => 'file', 'key_prefix' => 'my_')
);

$this->cache->get('foo'); // Will get the cache entry named 'my_foo'

```

类参考

class CI_Cache

is_supported(\$driver)

参数

- **\$driver** (*string*) – the name of the caching driver

返回 TRUE if supported, FALSE if not

返回类型 bool

当使用 `$this->cache->get()` 方法来访问驱动器时该方法会被自动调用，但是，如果你使用了某些个人的驱动器，应该先调用该方法确保这个驱动器在服务器环境下是否被支持。

```

if ($this->cache->apc->is_supported())
{
    if ($data = $this->cache->apc->get('my_cache'))
    {
        // do things.
    }
}

```

get(\$id)

参数

- **\$id** (*string*) – Cache item name

返回 Item value or FALSE if not found

返回类型 mixed

该方法用于从缓存中获取一项条目，如果获取的条目不存在，方法返回 FALSE 。

```
$foo = $this->cache->get('my_cached_item');
```

```
save($id, $data[, $ttl = 60[, $raw = FALSE]])
```

参数

- **\$id** (*string*) – Cache item name
- **\$data** (*mixed*) – the data to save
- **\$ttl** (*int*) – Time To Live, in seconds (default 60)
- **\$raw** (*bool*) – Whether to store the raw value

返回 TRUE on success, FALSE on failure

返回类型 string

该方法用于将一项条目保存到缓存中，如果保存失败，方法返回 FALSE 。

```
$this->cache->save('cache_item_id', 'data_to_cache');
```

注解： 参数 **\$raw** 只有在使用 APC 和 Memcache 缓存时才有用，它用于 `increment()` 和 `decrement()` 方法。

```
delete($id)
```

参数

- **\$id** (*string*) – name of cached item

返回 TRUE on success, FALSE on failure

返回类型 bool

该方法用于从缓存中删除一项指定条目，如果删除失败，方法返回 FALSE 。

```
$this->cache->delete('cache_item_id');
```

```
increment($id[, $offset = 1])
```

参数

- **\$id** (*string*) – Cache ID
- **\$offset** (*int*) – Step/value to add

返回 New value on success, FALSE on failure

返回类型 mixed

对缓存中的值执行原子自增操作。

```
// 'iterator' has a value of 2
```

```
$this->cache->increment('iterator'); // 'iterator' is now 3
```

```
$this->cache->increment('iterator', 3); // 'iterator' is now 6
```

decrement(\$id[, \$offset = 1])

参数

- \$id (*string*) – Cache ID
- \$offset (*int*) – Step/value to reduce by

返回 New value on success, FALSE on failure

返回类型 mixed

对缓存中的值执行原子自减操作。

```
// 'iterator' has a value of 6
```

```
$this->cache->decrement('iterator'); // 'iterator' is now 5
```

```
$this->cache->decrement('iterator', 2); // 'iterator' is now 3
```

clean()

返回 TRUE on success, FALSE on failure

返回类型 bool

该方法用于清空整个缓存, 如果清空失败, 方法返回 FALSE。

```
$this->cache->clean();
```

cache_info()

返回 Information on the entire cache database

返回类型 mixed

该方法返回整个缓存的信息。

```
var_dump($this->cache->cache_info());
```

注解: 返回的信息以及数据结构取决于使用的缓存驱动器。

get_metadata(\$id)

参数

- \$id (*string*) – Cache item name

返回 Metadata for the cached item

返回类型 mixed

该方法用于获取缓存中某个指定条目的详细信息。

```
var_dump($this->cache->get_metadata('my_cached_item'));
```

注解: 返回的信息以及数据结构取决于使用的缓存驱动器。

驱动器

可选 PHP 缓存 (APC)

上述所有方法都可以直接使用, 而不用在加载驱动器时指定 adapter 参数, 如下所示:

```
$this->load->driver('cache');  
$this->cache->apc->save('foo', 'bar', 10);
```

关于 APC 的更多信息, 请参阅 <http://php.net/apc>

基于文件的缓存

和输出类的缓存不同的是, 基于文件的缓存支持只缓存视图的某一部分。使用这个缓存时要注意, 确保对你的应用程序进行基准测试, 因为当磁盘 I/O 频繁时可能对缓存有负面影响。

上述所有方法都可以直接使用, 而不用在加载驱动器时指定 adapter 参数, 如下所示:

```
$this->load->driver('cache');  
$this->cache->file->save('foo', 'bar', 10);
```

Memcached 缓存

可以在 memcached.php 配置文件中指定多个 Memcached 服务器, 配置文件位于 *application/config/* 目录。

上述所有方法都可以直接使用, 而不用在加载驱动器时指定 adapter 参数, 如下所示:

```
$this->load->driver('cache');  
$this->cache->memcached->save('foo', 'bar', 10);
```

关于 Memcached 的更多信息, 请参阅 <http://php.net/memcached>

WinCache 缓存

在 Windows 下, 你还可以使用 WinCache 缓存。

上述所有方法都可以直接使用, 而不用在加载驱动器时指定 adapter 参数, 如下所示:

```
$this->load->driver('cache');
$this->cache->wincache->save('foo', 'bar', 10);
```

关于 WinCache 的更多信息, 请参阅 <http://php.net/wincache>

Redis 缓存

Redis 是一个在内存中以键值形式存储数据的缓存, 使用 LRU (最近最少使用算法) 缓存模式, 要使用它, 你需要先安装 Redis 服务器和 `phpredis` 扩展。

连接 Redis 服务器的配置信息必须保存到 `application/config/redis.php` 文件中, 可用参数有:

```
$config['socket_type'] = 'tcp'; // `tcp` or `unix`
$config['socket'] = '/var/run/redis.sock'; // in case of `unix` socket type
$config['host'] = '127.0.0.1';
$config['password'] = NULL;
$config['port'] = 6379;
$config['timeout'] = 0;
```

上述所有方法都可以直接使用, 而不用在加载驱动器时指定 adapter 参数, 如下所示:

```
$this->load->driver('cache');
$this->cache->redis->save('foo', 'bar', 10);
```

关于 Redis 的更多信息, 请参阅 <http://redis.io>

虚拟缓存 (Dummy Cache)

这是一个永远不会命中的缓存, 它不存储数据, 但是它允许你在当使用的缓存在你的环境下不被支持时, 仍然保留使用缓存的代码。

8.1.3 日历类

使用日历类可以让你动态的创建日历, 并且可以使用日历模板来格式化显示你的日历, 允许你 100% 的控制它设计的每个方面。另外, 你还可以向日历的单元格传递数据。


```

    3 => 'http://example.com/news/article/2006/03/',
    7 => 'http://example.com/news/article/2006/07/',
    13 => 'http://example.com/news/article/2006/13/',
    26 => 'http://example.com/news/article/2006/26/'
);

```

```
echo $this->calendar->generate(2006, 6, $data);
```

使用上面的例子，天数为 3、7、13 和 26 将变成链接指向你提供的 URL。

注解： 默认情况下，系统假定你的数组中包含了链接。在下面介绍日历模板的部分，你会看到你可以自定义处理传入日历单元格的数据，所以你可以传不同类型的信息。

设置显示参数

有 8 种不同的参数可以让你设置日历的各个方面，你可以通过加载函数的第二个参数来设置参数。例如：

```

$prefs = array(
    'start_day'    => 'saturday',
    'month_type'   => 'long',
    'day_type'     => 'short'
);

$this->load->library('calendar', $prefs);

echo $this->calendar->generate();

```

上面的代码将显示一个日历从礼拜六开始，使用完整的月份标题和缩写的天数格式。更多参数信息请看下面。

参数	默认值	选项	描述
template	None	None	字符串或数组，包含了你的日历模板，见下面的模板部分。
local_time	time()	None	当前时间的 Unix 时间戳。
start_day	sunday	Any week day (sunday, monday, tuesday, etc.)	指定每周的第一天是周几。
month_type	long	long, short	月份的显示样式 (long = January, short = Jan)
day_type	abr	long, short, abr	星期的显示样式 (long = Sunday, short = Sun, abr = Su)
show_next_prev	FALSE	TRUE/FALSE (boolean)	是否显示“上个月”和“下个月”链接，见下文。
next_prev_url	url- troller/ method	A URL	设置“上个月”和“下个月”的链接地址。
show_other_days	FALSE	TRUE/FALSE (boolean)	是否显示第一周和最后一周相邻月份的日期。

显示下一月/上一月链接

要让你的日历通过下一月/上一月链接动态的减少/增加，可以仿照下面的例子建立你的日历：

```
$prefs = array(
    'show_next_prev' => TRUE,
    'next_prev_url'  => 'http://example.com/index.php/calendar/show/'
);

$this->load->library('calendar', $prefs);

echo $this->calendar->generate($this->uri->segment(3), $this->uri->segment(4));
```

在上面的例子中，你会注意到这几点：

- “show_next_prev” 参数必须设置为 TRUE
- “next_prev_url” 参数必须设置一个 URL，如果不设置，会默认使用当前的 **控制器/方法**
- 通过 URI 的段将“年”和“月”参数传递给日历生成函数（日历类会自动添加“年”和“月”到你提供的 URL）

创建一个日历模板

通过创建一个日历模板你能够 100% 的控制你的日历的设计。当使用字符串方式设置模板时，日历的每一部分都要被放在一对伪变量中，像下面这样：

```

$prefs['template'] = '

{table_open}<table border="0" cellpadding="0" cellspacing="0">{/table_open}

{heading_row_start}<tr>{/heading_row_start}

{heading_previous_cell}<th><a href="{previous_url}">&lt;&lt;</a></th>{/heading_previous_cell}
{heading_title_cell}<th colspan="{colspan}">{heading}</th>{/heading_title_cell}
{heading_next_cell}<th><a href="{next_url}">&gt;&gt;</a></th>{/heading_next_cell}

{heading_row_end}</tr>{/heading_row_end}

{week_row_start}<tr>{/week_row_start}
{week_day_cell}<td>{week_day}</td>{/week_day_cell}
{week_row_end}</tr>{/week_row_end}

{cal_row_start}<tr>{/cal_row_start}
{cal_cell_start}<td>{/cal_cell_start}
{cal_cell_start_today}<td>{/cal_cell_start_today}
{cal_cell_start_other}<td class="other-month">{/cal_cell_start_other}

{cal_cell_content}<a href="{content}">{day}</a>{/cal_cell_content}
{cal_cell_content_today}<div class="highlight"><a href="{content}">{day}</a></div>{/cal_cell_content}

{cal_cell_no_content}{day}</cal_cell_no_content}
{cal_cell_no_content_today}<div class="highlight">{day}</div>{/cal_cell_no_content_today}

{cal_cell_blank}&nbsp;&nbsp;&{/cal_cell_blank}

{cal_cell_other}{day}</cal_cell_other}

{cal_cell_end}</td>{/cal_cell_end}
{cal_cell_end_today}</td>{/cal_cell_end_today}
{cal_cell_end_other}</td>{/cal_cell_end_other}
{cal_row_end}</tr>{/cal_row_end}

{table_close}</table>{/table_close}

';

$this->load->library('calendar', $prefs);

echo $this->calendar->generate();

```

当使用数组方式设置模板时, 你需要传递 *key => value* 键值对, 你可以只设置你想设置的参数, 其他没有设置的参数会使用日历类的默认值代替。

例子:

```

$prefs['template'] = array(
    'table_open'          => '<table class="calendar">',
    'cal_cell_start'      => '<td class="day">',

```

```
'cal_cell_start_today' => '<td class="today">'
);

$this->load->library('calendar', $prefs);

echo $this->calendar->generate();
```

类参考

class CI_Calendar

```
initialize([$config = array()])
```

参数

- **\$config** (*array*) – Configuration parameters

返回 CI_Calendar instance (method chaining)

返回类型 CI_Calendar

初始化日历类参数，输入参数为一个关联数组，包含了日历的显示参数。

```
generate([$year = '[', $month = '[', $data = array()])])
```

参数

- **\$year** (*int*) – Year
- **\$month** (*int*) – Month
- **\$data** (*array*) – Data to be shown in the calendar cells

返回 HTML-formatted calendar

返回类型 string

生成日历。

```
get_month_name($month)
```

参数

- **\$month** (*int*) – Month

返回 Month name

返回类型 string

提供数字格式的月份，返回月份的名称。

```
get_day_names($day_type = '')
```

参数

- **\$day_type** (*string*) – 'long', 'short', or 'abr'

返回 Array of day names

返回类型 array

根据类型返回一个包含星期名称 (Sunday、Monday 等等) 的数组, 类型有: long、short 和 abr。如果没有指定 `$day_type` 参数 (或该参数无效), 方法默认使用 abr (缩写) 格式。

adjust_date(*\$month*, *\$year*)

参数

- **\$month** (*int*) – Month
- **\$year** (*int*) – Year

返回 An associative array containing month and year

返回类型 array

该方法调整日期确保日期有效。例如, 如果你将月份设置为 13, 年份将自动加 1, 并且月份变为一月:

```
print_r($this->calendar->adjust_date(13, 2014));
```

输出:

```
Array
(
    [month] => '01'
    [year] => '2015'
)
```

get_total_days(*\$month*, *\$year*)

参数

- **\$month** (*int*) – Month
- **\$year** (*int*) – Year

返回 Count of days in the specified month

返回类型 int

获取指定月的天数:

```
echo $this->calendar->get_total_days(2, 2012);
// 29
```

注解: 该方法是日期辅助函数 `days_in_month()` 函数的别名。

default_template()

返回 An array of template values

返回类型 array

默认的模板, 当你没有使用你自己的模板时将会使用该方法。

parse_template()

返回 CI_Calendar instance (method chaining)

返回类型 CI_Calendar

解析模板中的伪变量 {pseudo-variables} 显示日历。

8.1.4 购物车类

购物车类允许项目被添加到 session 中, session 在用户浏览你的网站期间都保持有效状态。这些项目能够以标准的“购物车”格式被检索和显示, 并允许用户更新数量或者从购物车中移除项目。

重要: 购物车类已经废弃, 请不要使用。目前保留它只是为了向前兼容。

请注意购物车类只提供核心的“购物车”功能。它不提供配送、信用卡授权或者其它处理组件。

- 使用购物车类
 - 初始化购物车类
 - 将一个项目添加到购物车
 - 将多个项目添加到购物车
 - 显示购物车
 - 更新购物车
 - * 什么是 Row ID?
- 类参考

使用购物车类

初始化购物车类

重要: 购物车类利用 CodeIgniter 的 *Session* 类把购物车信息保存到数据库中, 所以在使用购物车类之前, 你必须根据 *Session* 类文档 中的说明来创建数据库表, 并且在 application/config/config.php 文件中把 Session 相关参数设置为使用数据库。

为了在你的控制器构造函数中初始化购物车类, 请使用 `$this->load->library` 函数:

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

一旦加载, 就可以通过调用 `$this->cart` 来使用购物车对象了:

```
$this->cart
```

注解: 购物车类会自动加载和初始化 Session 类, 因此除非你在别处要用到 session, 否则你不需要再次加载 Session 类。

将一个项目添加到购物车

要添加项目到购物车, 只需将一个包含了商品信息的数组传递给 `$this->cart->insert()` 函数即可, 就像下面这样:

```
$data = array(
    'id'      => 'sku_123ABC',
    'qty'     => 1,
    'price'   => 39.95,
    'name'    => 'T-Shirt',
    'options' => array('Size' => 'L', 'Color' => 'Red')
);

$this->cart->insert($data);
```

重要: 上面的前四个数组索引 (id、qty、price 和 name) 是 **必需的**。如果缺少其中的任何一个, 数据将不会被保存到购物车中。第 5 个索引 (options) 是可选的。当你的商品包含一些相关的选项信息时, 你就可以使用它。正如上面所显示的那样, 请使用一个数组来保存选项信息。

五个保留的索引分别是:

- **id** - 你的商店里的每件商品都必须有一个唯一的标识符。典型的标识符是库存量单位 (SKU) 或者其它类似的标识符。
- **qty** - 购买的数量。
- **price** - 商品的价格。
- **name** - 商品的名称。
- **options** - 标识商品的任何附加属性。必须通过数组来传递。

除以上五个索引外, 还有两个保留字: rowid 和 subtotal。它们是购物车类内部使用的, 因此, 往购物车中插入数据时, 请不要使用这些词作为索引。

你的数组可能包含附加的数据。你的数组中包含的所有数据都会被存储到 session 中。然而, 最好的方式是标准化你所有商品的数据, 这样更方便你在表格中显示它们。

```
$data = array(
    'id'      => 'sku_123ABC',
    'qty'     => 1,
    'price'   => 39.95,
    'name'    => 'T-Shirt',
    'coupon'  => 'XMAS-50OFF'
);

$this->cart->insert($data);
```

如果成功的插入一条数据后, `insert()` 方法将会返回一个 id 值 (`$rowid`)。

将多个项目添加到购物车

通过下面这种多维数组的方式，可以一次性添加多个产品到购物车中。当你希望允许用户选择同一页面中的多个项目时，这就非常有用。

```
$data = array(
    array(
        'id'      => 'sku_123ABC',
        'qty'     => 1,
        'price'   => 39.95,
        'name'    => 'T-Shirt',
        'options' => array('Size' => 'L', 'Color' => 'Red')
    ),
    array(
        'id'      => 'sku_567ZYX',
        'qty'     => 1,
        'price'   => 9.95,
        'name'    => 'Coffee Mug'
    ),
    array(
        'id'      => 'sku_965QRS',
        'qty'     => 1,
        'price'   => 29.95,
        'name'    => 'Shot Glass'
    )
);

$this->cart->insert($data);
```

显示购物车

为了显示购物车的数据，你得创建一个视图文件，它的代码类似于下面这个。

请注意这个范例使用了[表单辅助函数](#)。

```
<?php echo form_open('path/to/controller/update/method'); ?>

<table cellpadding="6" cellspacing="1" style="width:100%" border="0">

<tr>
    <th>QTY</th>
    <th>Item Description</th>
    <th style="text-align:right">Item Price</th>
    <th style="text-align:right">Sub-Total</th>
</tr>

<?php $i = 1; ?>

<?php foreach ($this->cart->contents() as $items): ?>
```

```

<?php echo form_hidden($i.'[rowid]', $items['rowid']); ?>

<tr>
    <td><?php echo form_input(array('name' => $i.'[qty]', 'value' => $items['qty'], 'max'
    <td>
        <?php echo $items['name']; ?>

        <?php if ($this->cart->has_options($items['rowid']) == TRUE): ?>

            <p>
                <?php foreach ($this->cart->product_options($items['rowid']) as $option:
                    <strong><?php echo $option_name; ?>:</strong> <?php echo $option_value; ?>
                <?php endforeach; ?>
            </p>

            <?php endif; ?>

        </td>
        <td style="text-align:right"><?php echo $this->cart->format_number($items['price']); ?>
        <td style="text-align:right">$<?php echo $this->cart->format_number($items['subtotal']); ?>
    </tr>

<?php $i++; ?>

<?php endforeach; ?>

<tr>
    <td colspan="2"> </td>
    <td class="right"><strong>Total</strong></td>
    <td class="right">$<?php echo $this->cart->format_number($this->cart->total()); ?></td>
</tr>

</table>

<p><?php echo form_submit('', 'Update your Cart'); ?></p>

```

更新购物车

为了更新购物车中的信息, 你必须将一个包含了 Row ID 和数量的数组传递给 `$this->cart->update()` 函数。

注解: 如果数量被设置为 0 , 那么购物车中对应的项目会被移除。

```

$data = array(
    'rowid' => 'b99ccdf16028f015540f341130b6d8ec',
    'qty'   => 3
);

```

```
);

$this->cart->update($data);

// Or a multi-dimensional array

$data = array(
    array(
        'rowid'    => 'b99ccdf16028f015540f341130b6d8ec',
        'qty'      => 3
    ),
    array(
        'rowid'    => 'xw82g9q3r495893iajdh473990rikw23',
        'qty'      => 4
    ),
    array(
        'rowid'    => 'fh4kdkkkaoe30njgoe92rkdkkobec333',
        'qty'      => 2
    )
);

$this->cart->update($data);
```

你也可以更新任何一个在新增购物车时定义的属性，如：options、price 或其他用户自定义字段。

```
$data = array(
    'rowid' => 'b99ccdf16028f015540f341130b6d8ec',
    'qty'   => 1,
    'price' => 49.95,
    'coupon' => NULL
);

$this->cart->update($data);
```

什么是 Row ID? 当一个项目被添加到购物车时，程序所生成的那个唯一的标识符就是 row ID。创建唯一 ID 的理由是，当购物车中相同的商品有不同的选项时，购物车就能够对它们进行管理。

比如说，有人购买了两件相同的 T-shirt（相同的商品 ID），但是尺寸不同。商品 ID（以及其它属性）都会完全一样，因为它们是相同的 T-shirt，它们唯一的差别就是尺寸不同。因此购物车必须想办法来区分它们，这样才能独立地管理这两件尺寸不同的 T-shirt。而基于商品 ID 和其它相关选项信息来创建一个唯一的“row ID”就能解决这个问题。

在几乎所有情况下，更新购物车都将是用户通过“查看购物车”页面来实现的，因此对开发者来说，不必太担心“row ID”，只要保证你的“查看购物车”页面中的一个隐藏表单字段包含了这个信息，并且确保它能被传递给表单提交时所调用的更新函数就行了。请仔细分析上面的“查看购物车”页面的结构以获取更多信息。

类参考

class CI_Cart

\$product_id_rules = '.a-z0-9_-'

用于验证商品 ID 有效性的正则表达式规则，默认是：字母、数字、连字符 (-)、下划线 (_)、句点 (.)

\$product_name_rules = 'w -.:'

用于验证商品 ID 和商品名有效性的正则表达式规则，默认是：字母、数字、连字符 (-)、下划线 (_)、冒号 (:)、句点 (.)

\$product_name_safe = TRUE

是否只接受安全的商品名称，默认为 TRUE。

insert(*\$items = array()*)

参数

- **\$items** (*array*) – Items to insert into the cart

返回 TRUE on success, FALSE on failure

返回类型 bool

将项目添加到购物车并保存到 session 中，根据成功或失败返回 TRUE 或 FALSE。

update(*\$items = array()*)

参数

- **\$items** (*array*) – Items to update in the cart

返回 TRUE on success, FALSE on failure

返回类型 bool

该方法用于更新购物车中某个项目的属性。一般情况下，它会在“查看购物车”页面被调用，例如用户在下单之前修改商品数量。参数是个数组，数组的每一项必须包含 rowid。

remove(*\$rowid*)

参数

- **\$rowid** (*int*) – ID of the item to remove from the cart

返回 TRUE on success, FALSE on failure

返回类型 bool

根据 \$rowid 从购物车中移除某个项目。

total()

返回 Total amount

返回类型 int

显示购物车总额。

total_items()

返回 Total amount of items in the cart

返回类型 int

显示购物车中商品数量。

contents() (*\$newest_first = FALSE*)

参数

- **\$newest_first** (*bool*) – Whether to order the array with newest items first

返回 An array of cart contents

返回类型 array

返回一个数组，包含购物车的所有信息。参数为布尔值，用于控制数组的排序方式。TRUE 为按购物车里的项目从新到旧排序，FALSE 为从旧到新。

get_item() (*\$row_id*)

参数

- **\$row_id** (*int*) – Row ID to retrieve

返回 Array of item data

返回类型 array

根据指定的 *\$rowid* 返回购物车中该项的信息，如果不存在，返回 FALSE。

has_options() (*\$row_id = ''*)

参数

- **\$row_id** (*int*) – Row ID to inspect

返回 TRUE if options exist, FALSE otherwise

返回类型 bool

如果购物车的某项包含 options 则返回 TRUE。该方法可以用在针对 **contents()** 方法的循环中，你需要指定项目的 rowid，正如上文“显示购物车”的例子中那样。

product_options() (*\$row_id = ''*)

参数

- **\$row_id** (*int*) – Row ID

返回 Array of product options

返回类型 array

该方法返回购物车中某个商品的 options 数组。该方法可以用在针对 contents() 方法的循环中, 你需要指定项目的 rowid , 正如上文“显示购物车”的例子中那样。

destroy()

返回类型 void

清空购物车。该函数一般在用户订单处理完成之后调用。

8.1.5 配置类

配置类用于获取配置参数, 这些参数可以来自于默认的配置文件的 (application/config/config.php), 也可以来自你自定义的配置文件。

注解: 该类由系统自动初始化, 你无需手工加载。

- 使用配置类
 - 配置文件剖析
 - 加载配置文件
 - * 手工加载
 - * 自动加载
 - 获取配置项
 - 设置配置项
 - 多环境
- 类参考

使用配置类

配置文件剖析

CodeIgniter 默认有一个主要的配置文件, 位于 application/config/config.php 。如果你使用文本编辑器打开它的话, 你会看到配置项都存储在一个叫做 \$config 的数组中。

你可以往这个文件中添加你自己的配置项, 或者你喜欢将你的配置项和系统的分开的话, 你也可以创建你自己的配置文件并保存到配置目录下。

注解: 如果你要创建你自己的配置文件, 使用 and 主配置文件相同的格式, 将配置项保存到名为 \$config 的数组中。CodeIgniter 会智能的管理这些文件, 所以就算数组名都一样也不会冲突 (假设数组的索引没有相同的)。

加载配置文件

注解: CodeIgniter 会自动加载主配置文件 (application/config/config.php), 所以你需要加载你自己创建的配置文件就可以了。

有两种加载配置文件的方式:

手工加载 要加载你自定义的配置文件, 你需要在 **控制器** 中使用下面的方法:

```
$this->config->load('filename');
```

其中, filename 是你的配置文件的名称, 无需.php 扩展名。

如果你需要加载多个配置文件, 它们会被合并成一个大的 config 数组里。尽管你是在不同的配置文件中定义的, 但是, 如果有两个数组索引名称一样的话还是会发生名称冲突。为了避免冲突, 你可以将第二个参数设置为 TRUE, 这样每个配置文件中的配置会被存储到以该配置文件名为索引的数组中去。例如:

```
// Stored in an array with this prototype: $this->config['blog_settings'] = $config
$this->config->load('blog_settings', TRUE);
```

请阅读下面的“获取配置项”一节, 学习如何获取通过这种方式加载的配置。

第三个参数用于抑制错误信息, 当配置文件不存在时, 不会报错:

```
$this->config->load('blog_settings', FALSE, TRUE);
```

自动加载 如果你发现有一个配置文件你需要在全局范围内使用, 你可以让系统自动加载它。要实现这点, 打开位于 application/config/ 目录下的 **autoload.php** 文件, 将你的配置文件添加到自动加载数组中。

获取配置项

要从你的配置文件中获取某个配置项, 使用如下方法:

```
$this->config->item('item_name');
```

其中 item_name 是你希望获取的 \$config 数组的索引名, 例如, 要获取语言选项, 你可以这样:

```
$lang = $this->config->item('language');
```

如果你要获取的配置项不存在, 方法返回 NULL。

如果你在使用 \$this->config->load 方法时使用了第二个参数, 每个配置文件中的配置被存储到以该配置文件名为索引的数组中, 要获取该配置项, 你可以将 \$this->config->item() 方法的第二个参数设置为这个索引名 (也就是配置文件名)。例如:

```
// Loads a config file named blog_settings.php and assigns it to an index named "blog_settings"
$this->config->load('blog_settings', TRUE);
```

```
// Retrieve a config item named site_name contained within the blog_settings array
```

```
$site_name = $this->config->item('site_name', 'blog_settings');

// An alternate way to specify the same item:
$blog_config = $this->config->item('blog_settings');
$site_name = $blog_config['site_name'];
```

设置配置项

如果你想动态的设置一个配置项, 或修改某个已存在的配置项, 你可以这样:

```
$this->config->set_item('item_name', 'item_value');
```

其中, item_name 是你希望修改的 \$config 数组的索引名, item_value 为要设置的值。

多环境

你可以根据当前的环境来加载不同的配置文件, index.php 文件中定义了 ENVIRONMENT 常量, 在[处理多环境](#)中有更详细的介绍。

要创建特定环境的配置文件, 新建或复制一个配置文件到 application/config/{ENVIRONMENT}/{FILENAME}.php。

例如, 要新建一个生产环境的配置文件, 你可以:

1. 新建目录 application/config/production/
2. 将已有的 config.php 文件拷贝到该目录
3. 编辑 application/config/production/config.php 文件, 使用生产环境下配置

当你将 ENVIRONMENT 常量设置为 'production' 时, 你新建的生产环境下的 config.php 里的配置将会加载。

你可以放置以下配置文件到特定环境的目录下:

- 默认的 CodeIgniter 配置文件
- 你自己的配置文件

注解: CodeIgniter 总是先加载全局配置文件 (例如, application/config/ 目录下的配置文件), 然后再去尝试加载当前环境的配置文件。这意味着你没必要将所有的配置文件都放到特定环境的配置目录下, 只需要放那些在每个环境下不一样的配置文件就可以了。另外, 你也不用拷贝所有的配置文件内容到特定环境的配置文件中, 只需要将那些在每个环境下不一样的配置项拷进去就行了。定义在环境目录下的配置项, 会覆盖掉全局的配置。

类参考

class CI_Config**\$config**

所有已加载的配置项组成的数组。

\$is_loaded

所有已加载的配置文件组成的数组。

item(\$item[, \$index=''])

参数

- **\$item** (*string*) – Config item name
- **\$index** (*string*) – Index name

返回 Config item value or NULL if not found

返回类型 mixed

获取某个配置项。

set_item(\$item, \$value)

参数

- **\$item** (*string*) – Config item name
- **\$value** (*string*) – Config item value

返回类型 void

设置某个配置项的值。

slash_item(\$item)

参数

- **\$item** (*string*) – config item name

返回 Config item value with a trailing forward slash or NULL if not found

返回类型 mixed

这个方法和 **item()** 一样，只是在获取的配置项后面添加一个斜线，如果配置项不存在，返回 NULL。

load([\$file = ''], \$use_sections = FALSE, \$fail_gracefully = FALSE)]])

参数

- **\$file** (*string*) – Configuration file name
- **\$use_sections** (*bool*) – Whether config values should be loaded into their own section (index of the main config array)

- **\$fail_gracefully** (*bool*) – Whether to return FALSE or to display an error message

返回 TRUE on success, FALSE on failure

返回类型 bool

加载配置文件。

site_url()

返回 Site URL

返回类型 string

该方法返回你的网站的 URL , 包括你在配置文件中设置的 “index” 值。

这个方法通常通过 [URL 辅助函数](#) 中函数来访问。

base_url()

返回 Base URL

返回类型 string

该方法返回你的网站的根 URL , 你可以在后面加上样式和图片的路径来访问它们。

这个方法通常通过 [URL 辅助函数](#) 中函数来访问。

system_url()

返回 URL pointing at your CI system/ directory

返回类型 string

该方法返回 CodeIgniter 的 system 目录的 URL 。

注解: 该方法已经废弃, 因为这是一个不安全的编码实践。你的 *system/* 目录不应该被公开访问。

8.1.6 Email 类

CodeIgniter 拥有强大的 Email 类支持以下特性:

- 多协议: Mail、Sendmail 和 SMTP
- SMTP 协议支持 TLS 和 SSL 加密
- 多个收件人
- 抄送 (CC) 和密送 (BCC)
- HTML 格式邮件或纯文本邮件
- 附件
- 自动换行

- 优先级
- 密送批处理模式 (BCC Batch Mode), 大邮件列表将被分成小批次密送
- Email 调试工具

- 使用 Email 类
 - 发送 Email
 - 设置 Email 参数
 - * 在配置文件中设置 Email 参数
 - Email 参数
 - 取消自动换行
- 类参考

使用 **Email** 类

发送 **Email**

发送邮件不仅很简单, 而且你可以通过参数或通过配置文件设置发送邮件的不同选项。

下面是个简单的例子, 用于演示如何发送邮件。注意: 这个例子假设你是在某个[控制器](#) 里面发送邮件。

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

$this->email->from('your@example.com', 'Your Name');
$this->email->to('someone@example.com');
$this->email->cc('another@another-example.com');
$this->email->bcc('them@their-example.com');

$this->email->subject('Email Test');
$this->email->message('Testing the email class.');
```

```
$this->email->send();
```

设置 **Email** 参数

有 21 种不同的参数可以用来对你发送的邮件进行配置。你可以像下面这样手工设置它们, 或者通过配置文件自动加载, 见下文:

通过向邮件初始化函数传递一个包含参数的数组来设置参数, 下面是个如何设置参数的例子:

```
$config['protocol'] = 'sendmail';
$config['mailpath'] = '/usr/sbin/sendmail';
$config['charset'] = 'iso-8859-1';
$config['wordwrap'] = TRUE;
```

```
$this->email->initialize($config);
```

注解: 如果你不设置, 大多数参数将使用默认值。

在配置文件中设置 Email 参数 如果你不喜欢使用上面的方法来设置参数, 你可以将它们放到配置文件中。你只需要简单的创建一个新文件 email.php, 将 \$config 数组放到该文件, 然后保存到 config/email.php, 这样它将会自动被加载。如果你使用配置文件的方式来设置参数, 你就不需要使用 `$this->email->initialize()` 方法了。

Email 参数

下表为发送邮件时所有可用的参数。

参数	默认值	选项	描述
user-agent	CodeIgniter	None	用户代理 (user agent)
protocol	mail	mail, sendmail, or smtp	邮件发送协议
mail-path	/usr/sbin/sendmail	None	服务器上 Sendmail 的实际路径
smtp_host	No Default	None	SMTP 服务器地址
smtp_user	No Default	None	SMTP 用户名
smtp_pass	No Default	None	SMTP 密码
smtp_port	25	None	SMTP 端口
smtp_timeout	5	None	SMTP 超时时间 (单位: 秒)
smtp_keepalive	FALSE	TRUE or FALSE (boolean)	是否启用 SMTP 持久连接
smtp_crypto	No Default	tls or ssl	SMTP 加密方式
word-wrap	TRUE	TRUE or FALSE (boolean)	是否启用自动换行
wrapchars	76		自动换行时每行的最大字符数
mailtype	text	text or html	邮件类型。如果发送的是 HTML 邮件, 必须是一个完整的网页, 确保网页中没有使用相对路径的链接和图片地址, 它们在邮件中不能正确显示。
charset	\$config['charset']		字符集 (utf-8, iso-8859-1 等)
validate	FALSE	TRUE or FALSE (boolean)	是否验证邮件地址
priority	3	1, 2, 3, 4, 5	Email 优先级 (1 = 最高. 5 = 最低. 3 = 正常)
crlf	\n	“\r\n” or “\n” or “\r”	换行符 (使用 “rn” 以遵守 RFC 822)
new-line	\n	“\r\n” or “\n” or “\r”	换行符 (使用 “rn” 以遵守 RFC 822)
bcc_batch_mode	FALSE	TRUE or FALSE (boolean)	是否启用密送批处理模式 (BCC Batch Mode)
bcc_batch_size	200	None	使用密送批处理时每一批邮件的数量
dsn	FALSE	TRUE or FALSE (boolean)	是否启用服务器提示消息

取消自动换行

如果你启用了自动换行（推荐遵守 RFC 822），然后你的邮件中又有一个超长的链接，那么它也会被自动换行，会导致收件人无法点击该链接。CodeIgniter 允许你禁用部分内容的自动换行，像下面这样：

```
The text of your email that
gets wrapped normally.
```

```
{unwrap}http://example.com/a_long_link_that_should_not_be_wrapped.html{/unwrap}
```

```
More text that will be
wrapped normally.
```

在你不想自动换行的内容前后使用 `{unwrap}` `{/unwrap}` 包起来。

类参考

class CI_Email

```
from($from[, $name = '[', $return_path = NULL])
```

参数

- **\$from** (*string*) – “From” e-mail address
- **\$name** (*string*) – “From” display name
- **\$return_path** (*string*) – Optional email address to redirect undelivered e-mail to

返回 CI_Email instance (method chaining)

返回类型 CI_Email

设置发件人 email 地址和名称：

```
$this->email->from('you@example.com', 'Your Name');
```

你还可以设置一个 Return-Path 用于重定向未收到的邮件：

```
$this->email->from('you@example.com', 'Your Name', 'returned_emails@example.com');
```

注解： 如果你使用的是 ‘smtp’ 协议，不能使用 Return-Path 。

```
reply_to($replyto[, $name = ''])
```

参数

- **\$replyto** (*string*) – E-mail address for replies
- **\$name** (*string*) – Display name for the reply-to e-mail address

返回 CI_Email instance (method chaining)

返回类型 CI_Email

设置邮件回复地址, 如果没有提供这个信息, 将会使用:meth:from 函数中的值。例如:

```
$this->email->reply_to('you@example.com', 'Your Name');
```

`to($to)`

参数

- **\$to** (*mixed*) – Comma-delimited string or an array of e-mail addresses

返回 CI_Email instance (method chaining)

返回类型 CI_Email

设置收件人 email 地址, 地址可以是单个、一个以逗号分隔的列表或是一个数组:

```
$this->email->to('someone@example.com');
```

```
$this->email->to('one@example.com, two@example.com, three@example.com');
```

```
$this->email->to(
    array('one@example.com', 'two@example.com', 'three@example.com')
);
```

`cc($cc)`

参数

- **\$cc** (*mixed*) – Comma-delimited string or an array of e-mail addresses

返回 CI_Email instance (method chaining)

返回类型 CI_Email

设置抄送 (CC) 的 email 地址, 和 “to” 方法一样, 地址可以是单个、一个以逗号分隔的列表或是一个数组。

`bcc($bcc[, $limit = ''])`

参数

- **\$bcc** (*mixed*) – Comma-delimited string or an array of e-mail addresses
- **\$limit** (*int*) – Maximum number of e-mails to send per batch

返回 CI_Email instance (method chaining)

返回类型 CI_Email

设置密送 (BCC) 的 email 地址, 和 “to” 方法一样, 地址可以是单个、一个以逗号分隔的列表或是一个数组。

如果设置了 `$limit` 参数, 将启用批处理模式, 批处理模式可以同时发送一批邮件, 每一批不超过设置的 `$limit` 值。

subject(*\$subject*)

参数

- **\$subject** (*string*) – E-mail subject line

返回 CI_Email instance (method chaining)

返回类型 CI_Email

设置 email 主题:

```
$this->email->subject('This is my subject');
```

message(*\$body*)

参数

- **\$body** (*string*) – E-mail message body

返回 CI_Email instance (method chaining)

返回类型 CI_Email

设置 email 正文部分:

```
$this->email->message('This is my message');
```

set_alt_message(*\$str*)

参数

- **\$str** (*string*) – Alternative e-mail message body

返回 CI_Email instance (method chaining)

返回类型 CI_Email

设置可选的 email 正文部分:

```
$this->email->set_alt_message('This is the alternative message');
```

如果你发送的是 HTML 格式的邮件, 可以设置一个可选的正文部分。对于那些设置了不接受 HTML 格式的邮件的人来说, 可以显示一段备选的不包含 HTML 格式的文本。如果你没有设置该参数, CodeIgniter 会自动从 HTML 格式邮件中删掉 HTML 标签。

set_header(*\$header*, *\$value*)

参数

- **\$header** (*string*) – Header name
- **\$value** (*string*) – Header value

返回 CI_Email instance (method chaining)

返回类型 CI_Email

向 email 添加额外的头:

```
$this->email->set_header('Header1', 'Value1');
$this->email->set_header('Header2', 'Value2');
```

```
clear([$clear_attachments = FALSE])
```

参数

- **\$clear_attachments** (*bool*) – Whether or not to clear attachments

返回 CI_Email instance (method chaining)

返回类型 CI_Email

将所有的 email 变量清空, 当你在一个循环中发送邮件时, 这个方法可以让你在每次发邮件之前将变量重置。

```
foreach ($list as $name => $address)
{
    $this->email->clear();

    $this->email->to($address);
    $this->email->from('your@example.com');
    $this->email->subject('Here is your info '.$name);
    $this->email->message('Hi '.$name.' Here is the info you requested.');
```

如果将参数设置为 TRUE , 邮件的附件也会被清空。

```
$this->email->clear(TRUE);
```

```
send([$auto_clear = TRUE])
```

参数

- **\$auto_clear** (*bool*) – Whether to clear message data automatically

返回 TRUE on success, FALSE on failure

返回类型 bool

发送 email , 根据成功或失败返回布尔值 TRUE 或 FALSE , 可以在条件语句中使用:

```
if ( ! $this->email->send() )
{
    // Generate error
}
```

如果发送成功, 该方法将会自动清除所有的参数。如果不想清除, 可以将参数置为 FALSE

```
if ($this->email->send(FALSE))
{
    // Parameters won't be cleared
}
```

注解: 为了使用 `print_debugger()` 方法, 你必须避免清空 `email` 的参数。

```
attach($filename[, $disposition = '[', $newname = NULL[, $mime = ']]])
```

参数

- **\$filename** (*string*) – File name
- **\$disposition** (*string*) – ‘disposition’ of the attachment. Most email clients make their own decision regardless of the MIME specification used here. <https://www.iana.org/assignments/cont-disp/cont-disp.xhtml>
- **\$newname** (*string*) – Custom file name to use in the e-mail
- **\$mime** (*string*) – MIME type to use (useful for buffered data)

返回 CI_Email instance (method chaining)

返回类型 CI_Email

添加附件, 第一个参数为文件的路径。要添加多个附件, 可以调用该方法多次。例如:

```
$this->email->attach('/path/to/photo1.jpg');
$this->email->attach('/path/to/photo2.jpg');
$this->email->attach('/path/to/photo3.jpg');
```

要让附件使用默认的 Content-Disposition (默认为: attachment) 将第二个参数留空, 你也可以使用其他的 Content-Disposition

```
$this->email->attach('image.jpg', 'inline');
```

另外, 你也可以使用 URL:

```
$this->email->attach('http://example.com/filename.pdf');
```

如果你想自定义文件名, 可以使用第三个参数:

```
$this->email->attach('filename.pdf', 'attachment', 'report.pdf');
```

如果你想使用一段字符串来代替物理文件, 你可以将第一个参数设置为该字符串, 第三个参数设置为文件名, 第四个参数设置为 MIME 类型:

```
$this->email->attach($buffer, 'attachment', 'report.pdf', 'application/pdf');
```

```
attachment_cid($filename)
```

参数

- **\$filename** (*string*) – Existing attachment filename

返回 Attachment Content-ID or FALSE if not found

返回类型 string

设置并返回一个附件的 Content-ID , 可以让你将附件 (图片) 内联显示到 HTML 正文中去。第一个参数必须是一个已经添加到附件中的文件名。

```
$filename = '/img/photo1.jpg';
$this->email->attach($filename);
foreach ($list as $address)
{
    $this->email->to($address);
    $cid = $this->email->attachment_cid($filename);
    $this->email->message('');
    $this->email->send();
}
```

注解: 每个 email 的 Content-ID 都必须重新创建, 为了保证唯一性。

```
print_debugger([$include = array('headers', 'subject', 'body')])
```

参数

- **\$include** (*array*) – Which parts of the message to print out

返回 Formatted debug data

返回类型 string

返回一个包含了所有的服务器信息、email 头部信息、以及 email 信息的字符串。用于调试。

你可以指定只返回消息的哪个部分, 有效值有: **headers** 、 **subject** 和 **body** 。

例如:

```
// You need to pass FALSE while sending in order for the email data
// to not be cleared - if that happens, print_debugger() would have
// nothing to output.
$this->email->send(FALSE);

// Will only print the email headers, excluding the message subject and body
$this->email->print_debugger(array('headers'));
```

注解: 默认情况, 所有的数据都会被打印出来。

8.1.7 加密类

加密类提供了双向数据加密的方式，它依赖于 PHP 的 Mcrypt 扩展，所以要有 Mcrypt 扩展才能运行。

重要： 这个类库已经废弃，保留只是为了向前兼容。请使用新的[加密类](#)。

- 使用加密类
 - 设置你的密钥
 - 消息长度
 - 初始化类
- 类参考

使用加密类

设置你的密钥

密钥 是对字符串进行加密或解密的一段信息片段。实际上，你设置的密钥是 **唯一** 能解密通过该密钥加密的数据，所以一定要慎重选择你的密钥，而且如果你打算对持久数据进行加密的话，你最好不要修改密钥。

不用说，你应该小心保管好你的密钥，如果有人得到了你的密钥，那么数据就能很容易的被解密。如果你的服务器不在你的控制之下，想保证你的密钥绝对安全是不可能的，所以在在你使用密钥对敏感数据（例如信用卡号码）进行加密之前，请再三斟酌。

为了最大程度的利用加密算法，你的密钥最好使用 32 位长度（256 字节），为了保证安全性，密钥字符串应该越随机越好，包含数字、大写和小写字符，**不应该** 直接使用一个简单的字符串。

你的密钥可以保存在 `application/config/config.php` 文件中，或者使用你自己设计的保存机制也行，然后在加解密时动态的取出密钥。

如果要通过文件 `application/config/config.php` 来保存你的密钥，那么打开该文件然后设置：

```
$config['encryption_key'] = "YOUR KEY";
```

消息长度

有一点很重要，你应该知道，通过加密方法生成的消息长度大概会比原始的消息长 2.6 倍。举个例子来说，如果你要加密的字符串是 “my super secret data”，它的长度为 21 字符，那么加密之后的字符串的长度大约为 55 字符（这里之所以说“大约”是因为加密字符串以 64 位为单位非线性增长）。当你选择数据保存机制时请记住这一点，例如 Cookie 只能存储 4k 的信息。

初始化类

正如 CodeIgniter 中的其他类一样, 在你的控制器中使用 `$this->load->library()` 方法来初始化加密类:

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

初始化之后, 加密类的对象就可以这样访问:

```
$this->encrypt
```

类参考

class CI_Encrypt

```
encode($string[, $key = ''])
```

参数

- **\$string** (*string*) – Data to encrypt
- **\$key** (*string*) – Encryption key

返回 Encrypted string

返回类型 string

执行数据加密, 并返回加密后的字符串。例如:

```
$msg = 'My secret message';

$encrypted_string = $this->encrypt->encode($msg);
```

如果你不想使用配置文件中的密钥, 你也可以将你的密钥通过第二个可选参数传入:

```
$msg = 'My secret message';
$key = 'super-secret-key';

$encrypted_string = $this->encrypt->encode($msg, $key);
```

```
decode($string[, $key = ''])
```

参数

- **\$string** (*string*) – String to decrypt
- **\$key** (*string*) – Encryption key

返回 Plain-text string

返回类型 string

解密一个已加密的字符串。例如:

```
$encrypted_string = 'APANtByIGI1BpVXZTJgcsAG8GZl8pdwwa84';

$plaintext_string = $this->encrypt->decode($encrypted_string);
```

如果你不想使用配置文件中的密钥, 你也可以将你的密钥通过第二个可选参数传入:

```
$msg = 'My secret message';
$key = 'super-secret-key';

$encrypted_string = $this->encrypt->decode($msg, $key);
```

set_cipher(*\$cipher*)

参数

- **\$cipher** (*int*) – Valid PHP MCrypt cypher constant

返回 CI_Encrypt instance (method chaining)

返回类型 CI_Encrypt

设置一个 Mcrypt 加密算法, 默认情况下, 使用的是 MCrypt_RIJNDAEL_256, 例如:

```
$this->encrypt->set_cipher(MCRYPT_BLOWFISH);
```

访问 php.net 获取一份 可用的加密算法清单。

如果你想测试下你的服务器是否支持 MCrypt, 你可以:

```
echo extension_loaded('mcrypt') ? 'Yup' : 'Nope';
```

set_mode(*\$mode*)

参数

- **\$mode** (*int*) – Valid PHP MCrypt mode constant

返回 CI_Encrypt instance (method chaining)

返回类型 CI_Encrypt

设置一个 Mcrypt 加密模式, 默认情况下, 使用的是 MCrypt_MODE_CBC, 例如:

```
$this->encrypt->set_mode(MCRYPT_MODE_CFB);
```

访问 php.net 获取一份 可用的加密模式清单。

```
encode_from_legacy($string[, $legacy_mode = MCRYPT_MODE_ECB[, $key
= '']])
```

参数

- **\$string** (*string*) – String to encrypt

- **\$legacy_mode** (*int*) – Valid PHP MCrypt cipher constant
- **\$key** (*string*) – Encryption key

返回 Newly encrypted string

返回类型 string

允许你重新加密在 CodeIgniter 1.x 下加密的数据，这样可以和 CodeIgniter 2.x 的加密类库保持兼容。只有当你的加密数据是永久的保存在诸如文件或数据库中时，并且你的服务器支持 Mcrypt ，你才可能需要使用这个方法。如果你只是在诸如会话数据或其他临时性的数据中使用加密的话，那么你根本用不到它。尽管如此，使用 2.x 版本之前的加密库加密的会话数据由于不能被解密，会话会被销毁。

重要： 为什么只是提供了一个重新加密的方法，而不是继续支持原有的加密方法呢？这是因为 CodeIgniter 2.x 中的加密库不仅在性能和安全性上有所提高，而且我们并不提倡继续使用老版本的加密方法。当然如果你愿意的话，你完全可以扩展加密库，使用老的加密方法来替代新的加密方法，无缝的兼容 CodeIgniter 1.x 加密数据。但是作为一个开发者，作出这样的决定还是应该小心谨慎。

```
$new_data = $this->encrypt->encode_from_legacy($old_encrypted_string);
```

参数	默认值	描述
\$orig_data		使用 CodeIgniter 1.x 加密过的原始数据
\$legacy_mode	<code>MCRYPT_MODE_ECB</code>	加密原始数据时使用的 Mcrypt 加密模式，CodeIgniter 1.x 默认使用的是 <code>MCRYPT_MODE_ECB</code> ，如果不指定该参数的话，将默认使用该方式。
\$key	n/a	加密密钥，这个值通常在上面所说的配置文件里。

8.1.8 加密类（新版）

重要： 绝不要使用这个类或其他任何加密类来进行密码处理！密码应该是被 哈希 ，你应该使用 PHP 自带的 密码哈希扩展 。

加密类提供了双向数据加密的方式，为了实现密码学意义上的安全，它使用了一些并非在所有系统上都可用的 PHP 的扩展，要使用这个类，你的系统上必须安装了下面的扩展：

- **OpenSSL** （以及 PHP 5.3.3）
- **MCrypt** （要支持 `MCRYPT_DEV_URANDOM`）

只要有一点不满足，我们就无法为你提供足够高的安全性。

- 使用加密类
 - 初始化类
 - 默认行为
 - 设置 `encryption_key` 参数
 - 支持的加密算法和模式
 - * 可移植的算法 (Portable ciphers)
 - * 特定驱动算法 (Driver-specific ciphers)
 - * 加密模式
 - 消息长度
 - 配置类库
 - 对数据进行加密与解密
 - * 实现原理
 - * 使用自定义参数
 - * 支持的 HMAC 认证算法
- 类参考

使用加密类

初始化类

正如 CodeIgniter 中的其他类一样, 在你的控制器中使用 `$this->load->library()` 方法来初始化加密类:

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

初始化之后, 加密类的对象就可以这样访问:

```
$this->encryption
```

默认行为

默认情况下, 加密类会通过你配置的 `encryption_key` 参数和 SHA512 HMAC 认证, 使用 AES-128 算法的 CBC 模式。

注解: 选择使用 AES-128 算法不仅是因为它已经被证明相当强壮, 而且它也已经不同的加密软件和编程语言 API 中广泛的使用了。

但是要注意的是, `encryption_key` 参数的用法可能并不是你想的那样。

如果你对密码学有点熟悉的话, 你应该知道, 使用 HMAC 算法认证也需要使用一个密钥, 而在加密的过程和认证的过程中使用相同的密钥可不是个好的做法。

正因为此, 程序会从你的配置的 `encryption_key` 参数中派生出两个密钥来: 一个用于加密, 另一个用于认证。这其实是通过一种叫做 **HKDF** (HMAC-based Key Derivation Function) 的技术实现的。

设置 `encryption_key` 参数

加密密钥 *encryption key* 是用于控制加密过程的一小段信息，使用它可以对普通文本进行加密和解密。这个过程可以保证只有你能对数据进行解密，其他人是看不到你的数据的，这其中的关键就是加密密钥。如果你使用了一个密钥来加密数据，那么就只能通过这个密钥来解密，所以 you 不仅应该仔细选择你的密钥，还应该好好的保管好它，不要忘记了。

还有一点要注意的是，为了确保最高的安全性，这个密钥不仅 *应该* 越强壮越好，而且 *应该* 经常修改。不过这在现实中很难做到，也不好实现，所以 CodeIgniter 提供了一个配置参数用于设置你的密钥，这个密钥（几乎）每次都会用到。

不用说，你应该小心保管好你的密钥，如果有人得到了你的密钥，那么数据就能很容易的被解密。如果你的服务器不在你的控制之下，想保证你的密钥绝对安全是不可能的，所以在在你使用密钥对敏感数据（例如信用卡号码）进行加密之前，请再三斟酌。

你的加密密钥的长度 **必须** 满足正在使用的加密算法允许的长度。例如，AES-128 算法最长支持 128 位（16 字节）。下面有一个表列出了不同算法支持的密钥长度。

你所使用的密钥应该越随机越好，它不能是一个普通的文本字符串，经过哈希函数处理过也不行。为了生成一个合适的密钥，你应该使用加密类提供的 `create_key()` 方法：

```
// $key will be assigned a 16-byte (128-bit) random key
$key = $this->encryption->create_key(16);
```

密钥可以保存在 `application/config/config.php` 配置文件中，或者你也可以设计你自己的存储机制，然后加密解密的时候动态的去获取它。

如果要保存在配置文件 `application/config/config.php` 中，可以打开该文件，然后设置：

```
$config['encryption_key'] = 'YOUR KEY';
```

你会发现 `create_key()` 方法返回的是二进制数据，没办法复制粘贴，所以你可能还需要使用 `bin2hex()`、`hex2bin()` 或 Base64 编码来更好的处理密钥数据。例如：

```
// Get a hex-encoded representation of the key:
$key = bin2hex($this->encryption->create_key(16));

// Put the same value in your config with hex2bin(),
// so that it is still passed as binary to the library:
$config['encryption_key'] = hex2bin(<your hex-encoded key>);
```

支持的加密算法和模式

可移植的算法 (Portable ciphers) 因为 MCrypt 和 OpenSSL（我们也称之为“驱动”）支持的加密算法不同，而且实现方式也不太一样，CodeIgniter 将它们设计成一种可移植的方式来使用，换句话说，你可以交换使用它们两个，至少对它们两个驱动都支持的算法来说是这样。

而且 CodeIgniter 的实现也和其他编程语言和类库的标准实现一致。

下面是可移植算法的清单，其中“CodeIgniter 名称”一栏就是你在使用加密类的时候使用的名称：

算法名称	CodeIgniter 名称	密钥长度（位/ 字节）	支持的模式
AES-128 / Rijndael-128	aes-128	128 / 16	CBC, CTR, CFB, CFB8, OFB, ECB
AES-192	aes-192	192 / 24	CBC, CTR, CFB, CFB8, OFB, ECB
AES-256	aes-256	256 / 32	CBC, CTR, CFB, CFB8, OFB, ECB
DES	des	56 / 7	CBC, CFB, CFB8, OFB, ECB
TripleDES	tripledes	56 / 7, 112 / 14, 168 / 21	CBC, CFB, CFB8, OFB
Blowfish	blowfish	128-448 / 16-56	CBC, CFB, OFB, ECB
CAST5 / CAST-128	cast5	88-128 / 11-16	CBC, CFB, OFB, ECB
RC4 / ARCFour	rc4	40-2048 / 5-256	Stream

重要： 由于 MCrypt 的内部实现，如果你提供了一个长度不合适的密钥，它会使用另一种不同的算法来加密，这将和你配置的算法不一致，所以要特别注意这一点！

注解： 上表中还有一点要澄清，Blowfish、CAST5 和 RC4 算法支持可变长度的密钥，也就是说，只要密钥的长度在指定范围内都是可以的。

注解： 尽管 CAST5 支持的密钥的长度可以小于 128 位（16 字节），其实实际上，根据 [RFC 2144](#) 我们知道，它会用 0 进行补齐到最大长度。

注解： Blowfish 算法支持最短 32 位（4 字节）的密钥，但是经过我们的测试发现，只有密钥长度大于等于 128 位（16 字节）时，才可以很好的同时支持 MCrypt 和 OpenSSL，再说，设置这么短的密钥也不是好的做法。

特定驱动算法（Driver-specific ciphers） 正如前面所说，MCrypt 和 OpenSSL 支持不同的加密算法，所以你也可以选择下面这些只针对某一特定驱动的算法。但是为了移植性考虑，而且这些算法也没有经过彻底测试，我们并不建议你使用这些算法。

算法名称	驱动	密钥长度 (位/ 字节)	支持的模式
AES-128	OpenSSL	128 / 16	CBC, CTR, CFB, CFB8, OFB, ECB, XTS
AES-192	OpenSSL	192 / 24	CBC, CTR, CFB, CFB8, OFB, ECB, XTS
AES-256	OpenSSL	256 / 32	CBC, CTR, CFB, CFB8, OFB, ECB, XTS
Rijndael-128	MCrypt	128 / 16, 192 / 24, 256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
Rijndael-192	MCrypt	128 / 16, 192 / 24, 256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
Rijndael-256	MCrypt	128 / 16, 192 / 24, 256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
GOST	MCrypt	256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
Twofish	MCrypt	128 / 16, 192 / 24, 256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
CAST-128	MCrypt	40-128 / 5-16	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
CAST-256	MCrypt	128 / 16, 192 / 24, 256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
Loki97	MCrypt	128 / 16, 192 / 24, 256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
SaferPlus	MCrypt	128 / 16, 192 / 24, 256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
Serpent	MCrypt	128 / 16, 192 / 24, 256 / 32	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
XTEA	MCrypt	128 / 16	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
RC2	MCrypt	8-1024 / 1-128	CBC, CTR, CFB, CFB8, OFB, OFB8, ECB
RC2	OpenSSL	8-1024 / 1-128	CBC, CFB, OFB, ECB
Camellia-128	OpenSSL	128 / 16	CBC, CFB, CFB8, OFB, ECB
Camellia-192	OpenSSL	192 / 24	CBC, CFB, CFB8, OFB, ECB
Camellia-256	OpenSSL	256 / 32	CBC, CFB, CFB8, OFB, ECB
Seed	OpenSSL	128 / 16	CBC, CFB, OFB, ECB

注解: 如果你要使用这些算法, 你只需将它的名称以小写形式传递给加密类即可。

注解: 你可能已经注意到, 所有的 AES 算法 (以及 Rijndael-128 算法) 也在上面的可移植算法列表中出现了, 这是因为这些算法支持不同的模式。还有很重要的一点是, 在使用 128 位的密钥时, AES-128 和 Rijndael-128 算法其实是一样的。

注解: CAST-128 / CAST-5 算法也在两个表格都出现了, 这是因为当密钥长度小于等于 80 位时, OpenSSL 的实现貌似有问题。

注解: 列表中可以看到 RC2 算法同时被 MCrypt 和 OpenSSL 支持, 但是两个驱动对它的实现方式是不一样的, 而且也是不能移植的。我们只找到了一条关于这个的不确定的消息可能是 MCrypt 的实现有问题。

加密模式 加密算法的不同模式有着不同的特性, 它们有着不同的目的, 有的可能比另一些更强壮, 有的可能速度更快, 有的可能提供了额外的功能。我们并不打算深入研究这个, 这应该是密码学专家做的事。下表将向我们普通的用户列出一些简略的参考信息。如果你是个初学者, 直接使用 CBC 模式就可以了, 一般情况下它已经足够强壮和安全, 并且已经被广泛接受。

模式名称	CodeIgniter 名称	支持的驱动	备注
CBC	cbc	MCrypt, OpenSSL	安全的默认选择
CTR	ctr	MCrypt, OpenSSL	理论上比 CBC 更好, 但并没有广泛使用
CFB	cfb	MCrypt, OpenSSL	N/A
CFB8	cfb8	MCrypt, OpenSSL	和 CFB 一样, 但是使用 8 位模式 (不推荐)
OFB	ofb	MCrypt, OpenSSL	N/A
OFB8	ofb8	MCrypt	和 OFB 一样, 但是使用 8 位模式 (不推荐)
ECB	ecb	MCrypt, OpenSSL	忽略 IV (不推荐)
XTS Stream	xts stream	OpenSSL, MCrypt, OpenSSL	通常用来加密可随机访问的数据, 如 RAM 或硬盘 这其实并不是一种模式, 只是表明使用了流加密, 通常在算法 + 模式的初始化过程中会用到。

消息长度

有一点对你来说可能很重要, 加密的字符串通常要比原始的文本字符串要长 (取决于算法)。

这个会取决于加密所使用的算法, 添加到密文上的 IV, 以及添加的 HMAC 认证信息。另外, 为了保证传输的安全性, 加密消息还会被 Base64 编码。

当你选择数据保存机制时请记住这一点, 例如 Cookie 只能存储 4k 的信息。

配置类库

考虑到可用性, 性能, 以及一些历史原因, 加密类使用了和老的加密类一样的驱动、加密算法、模式和密钥。

上面的“默认行为”一节已经提到，系统将自动检测驱动（OpenSSL 优先级要高一点），使用 CBC 模式的 AES-128 算法，以及 `$config['encryption_key']` 参数。

如果你想改变这点，你需要使用 `initialize()` 方法，它的参数为一个关联数组，每一项都是可选：

选项	可能的值
driver	'mcrypt', 'openssl'
cipher	算法名称（参见 支持的加密算法和模式 ）
mode	加密模式（参见 加密模式 ）
key	加密密钥

例如，如果你想将加密算法和模式改为 AES-126 CTR，可以这样：

```
$this->encryption->initialize(
    array(
        'cipher' => 'aes-256',
        'mode' => 'ctr',
        'key' => '<a 32-character random string>'
    )
);
```

另外，我们也可以设置一个密钥，如前文所说，针对所使用的算法选择一个合适的密钥非常重要。

我们还可以修改驱动，如果你两种驱动都支持，但是出于某种原因你想使用 MCrypt 来替代 OpenSSL

```
// Switch to the MCrypt driver
$this->encryption->initialize(array('driver' => 'mcrypt'));

// Switch back to the OpenSSL driver
$this->encryption->initialize(array('driver' => 'openssl'));
```

对数据进行加密与解密

使用已配置好的参数来对数据进行加密和解密是非常简单的，你只要将字符串传给 `encrypt()` 和/或 `decrypt()` 方法即可：

```
$plain_text = 'This is a plain-text message!';
$ciphertext = $this->encryption->encrypt($plain_text);

// Outputs: This is a plain-text message!
echo $this->encryption->decrypt($ciphertext);
```

这样就行了！加密类会为你完成所有必须的操作并确保安全，你根本不用关系细节。

重要： 两个方法在遇到错误时都会返回 FALSE，如果是 `encrypt()` 返回 FALSE，那么只可能是配置参数错了。在生产代码中一定要对 `decrypt()` 方法进行检查。

实现原理 如果你非要知道整个过程的实现步骤, 下面是内部的实现:

- `$this->encryption->encrypt($plain_text)`
 1. 通过 HKDF 和 SHA-512 摘要算法, 从你配置的 `encryption_key` 参数中获取两个密钥: 加密密钥和 HMAC 密钥。
 2. 生成一个随机的初始向量 (IV)。
 3. 使用上面的加密密钥和 IV , 通过 AES-128 算法的 CBC 模式 (或其他你配置的算法和模式) 对数据进行加密。
 4. 将 IV 附加到密文后。
 5. 对结果进行 Base64 编码, 这样就可以安全的保存和传输它, 而不用担心字符集问题。
 6. 使用 HMAC 密钥生成一个 SHA-512 HMAC 认证消息, 附加到 Base64 字符串后, 以保证数据的完整性。
- `$this->encryption->decrypt($ciphertext)`
 1. 通过 HKDF 和 SHA-512 摘要算法, 从你配置的 `encryption_key` 参数中获取两个密钥: 加密密钥和 HMAC 密钥。由于 `encryption_key` 不变, 所以生成的结果和上面 `encrypt()` 方法生成的结果是一样的, 否则你没办法解密。
 2. 检查字符串的长度是否足够长, 并从字符串中分离出 HMAC , 然后验证是否一致 (这可以防止时序攻击 (timing attack)), 如果验证失败, 返回 FALSE 。
 3. 进行 Base64 解码。
 4. 从密文中分离出 IV , 并使用 IV 和加密密钥对数据进行解密。

使用自定义参数 假设你需要和另一个系统交互, 这个系统不受你的控制, 而且它使用了其他的方法来加密数据, 加密的方式和我们上面介绍的流程不一样。

在这种情况下, 加密类允许你修改它的加密和解密的流程, 这样你就可以简单的调整成自己的解决方案。

注解: 通过这种方式, 你可以不用在配置文件中配置 `encryption_key` 就能使用加密类。

你所需要做的就是传一个包含一些参数的关联数组到 `encrypt()` 或 `decrypt()` 方法, 下面是个例子:

```
// Assume that we have $ciphertext, $key and $hmac_key
// from an outside source

$message = $this->encryption->decrypt(
    $ciphertext,
    array(
        'cipher' => 'blowfish',
        'mode' => 'cbc',
    )
);
```



```
        'key' => $key,
        'hmac_digest' => 'sha256',
        'hmac_key' => $hmac_key
    )
);
```

在上面的例子中，我们对一段使用 CBC 模式的 Blowfish 算法加密的消息进行解密，并使用 SHA-256 HMAC 认证方式。

重要： 注意在这个例子中 ‘key’ 和 ‘hmac_key’ 参数都要指定，当使用自定义参数时，加密密钥和 HMAC 密钥不再是默认的那样从配置参数中自动获取的了。

下面是所有可用的选项。

但是，除非你真的需要这样做，并且你知道你在做什么，否则我们建议你不要修改加密的流程，因为这会影响安全性，所以请谨慎对待。

选项	默认值	必须的/ 可选的	描述
cipher	N/A	Yes	加密算法（参见 支持的加密算法和模式 ）
mode	N/A	Yes	加密模式（参见 加密模式 ）
key	N/A	Yes	加密密钥
hmac	TRUE	No	是否使用 HMAC 布尔值，如果为 FALSE ， <i>hmac_digest</i> 和 <i>hmac_key</i> 将被忽略
hmac_digest	sha512	No	HMAC 消息摘要算法（参见 支持的 HMAC 认证算法 ）
hmac_key	N/A	Yes, 除非 <i>hmac</i> 设为 FALSE	HMAC 密钥
raw_data	FALSE	No	加密文本是否保持原样布尔值，如果为 TRUE ，将不 执行 Base64 编码和解码操作 HMAC 也不会是十六进 制字符串

重要： `encrypt()` and `decrypt()` will return FALSE if a mandatory parameter is not provided or if a provided value is incorrect. This includes *hmac_key*, unless *hmac* is set to FALSE.

支持的 HMAC 认证算法 对于 HMAC 消息认证，加密类支持使用 SHA-2 家族的算法：

算法	原始长度（字节）	十六进制编码长度（字节）
sha512	64	128
sha384	48	96
sha256	32	64
sha224	28	56

之所以没有包含一些其他的流行算法，例如 MD5 或 SHA1，是因为这些算法目前已被证明不够安全，我们并不鼓励使用它们。如果你非要使用这些算法，简单的使用 PHP 的原生函数 `hash_hmac()` 也可以。

当未来出现广泛使用的更好的算法时，我们自然会将其添加进去。

类参考

class CI_Encryption

initialize(\$params)

参数

- **\$params** (*array*) – Configuration parameters

返回 CI_Encryption instance (method chaining)

返回类型 CI_Encryption

初始化加密类的配置，使用不同的驱动，算法，模式或密钥。

例如:

```
$this->encryption->initialize(
    array('mode' => 'ctr')
);
```

请参考[配置类库](#)一节了解详细信息。

encrypt(\$data[, \$params = NULL])

参数

- **\$data** (*string*) – Data to encrypt
- **\$params** (*array*) – Optional parameters

返回 Encrypted data or FALSE on failure

返回类型 string

对输入数据进行加密，并返回密文。

例如:

```
$ciphertext = $this->encryption->encrypt('My secret message');
```

请参考[使用自定义参数](#)一节了解更多参数信息。

decrypt(\$data[, \$params = NULL])

参数

- **\$data** (*string*) – Data to decrypt
- **\$params** (*array*) – Optional parameters

返回 Decrypted data or FALSE on failure

返回类型 string

对输入数据进行解密, 并返回解密后的文本。

例如:

```
echo $this->encryption->decrypt($ciphertext);
```

请参考[使用自定义参数](#)一节了解更多参数信息。

create_key(*\$length*)

参数

- **\$length** (*int*) – Output length

返回 A pseudo-random cryptographic key with the specified length, or FALSE on failure

返回类型 string

从操作系统获取随机数据 (例如/dev/urandom), 并生成加密密钥。

```
hkdf($key[, $digest = 'sha512', $salt = NULL[, $length = NULL[, $info =  
, ]]]])
```

参数

- **\$key** (*string*) – Input key material
- **\$digest** (*string*) – A SHA-2 family digest algorithm
- **\$salt** (*string*) – Optional salt
- **\$length** (*int*) – Optional output length
- **\$info** (*string*) – Optional context/application-specific info

返回 A pseudo-random key or FALSE on failure

返回类型 string

从一个密钥生成另一个密钥 (较弱的密钥)。

这是内部使用的一个方法, 用于从配置的 *encryption_key* 参数生成一个加密密钥和 HMAC 密钥。

将这个方法公开, 是为了可能会在其他地方使用到。关于这个算法的描述可以看 [RFC 5869](#)。

和 RFC 5869 描述不同的是, 这个方法不支持 SHA1。

例如:

```
$hmac_key = $this->encryption->hkdf(  
    $key,  
    'sha512',  
    NULL,
```

```

        NULL,
        'authentication'
    );

    // $hmac_key is a pseudo-random key with a length of 64 bytes

```

8.1.9 文件上传类

CodeIgniter 的文件上传类用于上传文件，你可以设置参数限制上传文件的类型和大小。

- 处理流程
 - 创建上传表单
 - 上传成功页面
 - 控制器
 - 上传文件目录
 - 尝试一下!
- 参考指南
 - 初始化文件上传类
 - 参数设置
 - 参数
 - 在配置文件中设置参数
- 类参考

处理流程

上传一个文件通常涉及以下步骤：

- 显示一个上传表单，用户选择文件并上传。
- 当提交表单时，文件将被上传到你指定的目录。
- 同时，根据你设置的参数对文件进行校验是否允许上传。
- 上传成功后，向用户显示成功消息。

下面是个简单的教程来示范该过程，然后会列出一些其他的参考信息。

创建上传表单

使用文本编辑器新建一个文件 `upload_form.php`，放入如下代码，并保存到 `application/views/` 目录下：

```

<html>
<head>
<title>Upload Form</title>
</head>

```

```
<body>

<?php echo $error;?>

<?php echo form_open_multipart('upload/do_upload');?>

<input type="file" name="userfile" size="20" />

<br /><br />

<input type="submit" value="upload" />

</form>

</body>
</html>
```

你会注意到我们使用了表单辅助函数来创建 form 的起始标签, 文件上传需要使用 multipart 表单, 辅助函数可以帮你正确生成它。还要注意的, 代码里有一个 \$error 变量, 当发生错误时, 可以用它来显示错误信息。

上传成功页面

使用文本编辑器新建一个文件 upload_success.php, 放入如下代码, 并保存到 **application/views/** 目录下:

```
<html>
<head>
<title>Upload Form</title>
</head>
<body>

<h3>Your file was successfully uploaded!</h3>

<ul>
<?php foreach ($upload_data as $item => $value):?>
<li><?php echo $item;?>: <?php echo $value;?></li>
<?php endforeach; ?>
</ul>

<p><?php echo anchor('upload', 'Upload Another File!'); ?></p>

</body>
</html>
```

控制器

使用文本编辑器新建一个控制器 Upload.php, 放入如下代码, 并保存到 **application/controllers/** 目录下:

```

<?php

class Upload extends CI_Controller {

    public function __construct()
    {
        parent::__construct();
        $this->load->helper(array('form', 'url'));
    }

    public function index()
    {
        $this->load->view('upload_form', array('error' => ' ' ));
    }

    public function do_upload()
    {
        $config['upload_path']      = './uploads/';
        $config['allowed_types']    = 'gif|jpg|png';
        $config['max_size']         = 100;
        $config['max_width']        = 1024;
        $config['max_height']       = 768;

        $this->load->library('upload', $config);

        if ( ! $this->upload->do_upload('userfile'))
        {
            $error = array('error' => $this->upload->display_errors());

            $this->load->view('upload_form', $error);
        }
        else
        {
            $data = array('upload_data' => $this->upload->data());

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

```

上传文件目录

你需要一个目录来保存上传的图片，在 CodeIgniter 的安装根目录下创建一个 uploads 目录，并将它的权限设置为 777 。

尝试一下!

使用类似于下面的 URL 来访问你的站点:

```
example.com/index.php/upload/
```

你应该能看到一个上传文件的表单, 尝试着上传一个图片文件 (jpg、gif 或 png 都可以), 如果你的控制器中路径设置正确, 你就可以成功上传文件了。

参考指南

初始化文件上传类

正如 CodeIgniter 中的其他类一样, 在你的控制器中使用 `$this->load->library()` 方法来初始化文件上传类:

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

初始化之后, 文件上传类的对象就可以这样访问:

```
$this->upload
```

参数设置

和其他的类库一样, 你可以通过你配置的参数来控制允许上传什么类型的文件。在上面的控制器中, 你设置了下面的这些参数:

```
$config['upload_path'] = './uploads/';  
$config['allowed_types'] = 'gif|jpg|png';  
$config['max_size'] = '100';  
$config['max_width'] = '1024';  
$config['max_height'] = '768';
```

```
$this->load->library('upload', $config);
```

```
// Alternately you can set preferences by calling the ``initialize()`` method. Useful if you  
$this->upload->initialize($config);
```

上面的参数根据它的名称就能很容易理解, 下表列出了所有可用的参数。

参数

下表列出了所有可用的参数, 当没有指定参数时程序会使用默认值。

参数	默认值	选项	描述
upload_path	None	None	文件上传的位置，必须是可写的，可以是相对路径或绝对路径
allowed_types	None	None	允许上传的文件 MIME 类型，通常文件的后缀名可作为 MIME 类型可以是数组，也可以是以管道符 () 分割的字符串
file_name	None	Desired file name	如果设置了，CodeIgniter 将会使用该参数重命名上传的文件设置的文件名后缀必须也是允许的文件类型如果没有设置后缀，将使用原文件的后缀名
file_ext_tolower	FALSE	TRUE/FALSE (boolean)	如果设置为 TRUE，文件后缀名将转换为小写
overwrite	FALSE	TRUE/FALSE (boolean)	如果设置为 TRUE，上传的文件如果和已有的文件同名，将会覆盖已存在文件如果设置为 FALSE，将会在文件名后加上一个数字
max_size	0	None	允许上传文件大小的最大值（单位 KB），设置为 0 表示无限制注意：大多数 PHP 会有它们自己的限制值，定义在 php.ini 文件中通常是默认 2 MB（2048 KB）。
max_width	0	None	图片的最大宽度（单位为像素），设置为 0 表示无限制
max_height	0	None	图片的最大高度（单位为像素），设置为 0 表示无限制
min_width	0	None	图片的最小宽度（单位为像素），设置为 0 表示无限制
min_height	0	None	图片的最小高度（单位为像素），设置为 0 表示无限制
max_filename	None	None	文件名的最大长度，设置为 0 表示无限制
max_filename_increment	10	None	当 overwrite 参数设置为 FALSE 时，将会在同名文件的后面加上一个自增的数字这个参数用于设置这个数字的最大值
encrypt_name	FALSE	TRUE/FALSE (boolean)	如果设置为 TRUE，文件名将会转换为一个随机的字符串如果你不希望上传文件的人知道保存后的文件名，这个参数会很有用
remove_spaces	TRUE	TRUE/FALSE (boolean)	如果设置为 TRUE，文件名中的所有空格将转换为下划线，推荐这样做
detect_mime	TRUE	TRUE/FALSE (boolean)	如果设置为 TRUE，将会在服务端对文件类型进行检测，可以预防代码注入攻击除非不得已，请不要禁用该选项，这将导致安全风险
mod_mime_fix	TRUE	TRUE/FALSE (boolean)	如果设置为 TRUE，那么带有多个后缀名的文件将会添加一个下划线后缀这样可以避免触发 Apache mod_mime 。如果你的上传目录是公开的，请不要关闭该选项，这将导致安全风险

在配置文件中设置参数

如果你不喜欢使用上面的方法来设置参数，你可以将参数保存到配置文件中。你只需简单的创建一个文件 `upload.php` 并将 `$config` 数组放到该文件中，然后保存文件到

`config/upload.php` , 这些参数将会自动被使用。如果你在配置文件中设置参数, 那么你就不需要使用 `$this->upload->initialize()` 方法了。

类参考

class CI_Upload

```
initialize([array $config = array(), $reset = TRUE])
```

参数

- **\$config** (*array*) – Preferences
- **\$reset** (*bool*) – Whether to reset preferences (that are not provided in \$config) to their defaults

返回 CI_Upload instance (method chaining)

返回类型 CI_Upload

```
do_upload([$field = 'userfile'])
```

参数

- **\$field** (*string*) – Name of the form field

返回 TRUE on success, FALSE on failure

返回类型 bool

根据你设置的参数上传文件。

注解: 默认情况下上传文件是来自于表单的 `userfile` 字段, 而且表单应该是“multipart”类型。

```
<form method="post" action="some_action" enctype="multipart/form-data" />
```

如果你想设置你自己的字段, 可以将字段名传给 `do_upload()` 方法:

```
$field_name = "some_field_name";  
$this->upload->do_upload($field_name);
```

```
display_errors([$open = '<p>', $close = '</p>'])
```

参数

- **\$open** (*string*) – Opening markup
- **\$close** (*string*) – Closing markup

返回 Formatted error message(s)

返回类型 string

如果 `do_upload()` 方法返回 `FALSE`，可以使用该方法来获取错误信息。该方法返回所有的错误信息，而不是直接显示出来。

格式化错误信息

默认情况下该方法会将错误信息包在 `<p>` 标签中，你可以设置你自己的标签：

```
$this->upload->display_errors('<p>', '</p>');
```

```
data([$index = NULL])
```

参数

- **\$data** (*string*) – Element to return instead of the full array

返回 Information about the uploaded file

返回类型 mixed

该方法返回一个数组，包含你上传的文件的所有信息，下面是数组的原型：

```
Array
(
    [file_name] => mypic.jpg
    [file_type] => image/jpeg
    [file_path] => /path/to/your/upload/
    [full_path] => /path/to/your/upload/jpg.jpg
    [raw_name] => mypic
    [orig_name] => mypic.jpg
    [client_name] => mypic.jpg
    [file_ext] => .jpg
    [file_size] => 22.2
    [is_image] => 1
    [image_width] => 800
    [image_height] => 600
    [image_type] => jpeg
    [image_size_str] => width="800" height="200"
)
```

你也可以只返回数组中的一项：

```
$this->upload->data('file_name'); // Returns: mypic.jpg
```

下表解释了上面列出的所有数组项：

项	描述
file_name	上传文件的文件名, 包含后缀名
file_type	文件的 MIME 类型
file_path	文件的绝对路径
full_path	文件的绝对路径, 包含文件名
raw_name	文件名, 不含后缀名
orig_name	原始的文件名, 只有在使用了 encrypt_name 参数时该值才 有用
client_name	用户提交过来的文件名, 还没有对该文件名做任何处理
file_ext	文件后缀名, 包括句点
file_size	文件大小 (单位 kb)
is_image	文件是否为图片 (1 = image. 0 = not.)
image_width	图片宽度
image_height	图片高度
image_type	图片类型 (通常是不带句点的文件后缀名)
image_size_str	一个包含了图片宽度和高度的字符串 (用于放在 image 标 签中)

8.1.10 表单验证类

CodeIgniter 提供了一个全面的表单验证和数据预处理类可以帮你少写很多代码。

Page Contents

- 表单验证类
 - 概述
 - 表单验证指南
 - * 表单
 - * 成功页面
 - * 控制器
 - * 试一下!
 - * 解释
 - * 设置验证规则
 - * 使用数组来设置验证规则
 - * 级联规则 (Cascading Rules)
 - * 预处理数据
 - * 重新填充表单
 - * 回调: 你自己的验证函数
 - * 使用任何可调用的方法作为验证规则
 - * 设置错误信息
 - * 翻译表单域名称
 - * 更改错误定界符
 - * 单独显示错误
 - * 验证数组 (除 \$_POST 数组)
 - 将一系列验证规则保存到一个配置文件
 - * 如何保存你的规则
 - * 创建规则集
 - * 调用某组验证规则
 - * 将控制器方法和规则集关联在一起
 - 使用数组作为域名称
 - 规则参考
 - 预处理参考
 - 类参考
 - 辅助函数参考

概述

在解释 CodeIgniter 的数据验证处理之前, 让我们先描述一下一般的情况:

1. 显示一个表单。
2. 你填写并提交了它。
3. 如果你提交了一些无效的信息, 或者可能漏掉了一个必填项, 表单将会重新显示你的数据, 并提示一个错误信息。
4. 这个过程将继续, 直到你提交了一个有效的表单。

在接收端, 脚本必须:

1. 检查必填的数据。

2. 验证数据类型是否为正确, 条件是否满足。例如, 如果提交一个用户名, 必须验证它是否只包含了允许的字符, 必须有一个最小长度, 不能超过最大长度。用户名不能和已存在的其他人名字相同, 或者不能是某个保留字, 等等。
3. 为确保安全性对数据进行过滤。
4. 如果需要, 预格式化数据 (数据需要清除空白吗? 需要 HTML 编码? 等等)
5. 准备数据, 插入数据库。

尽管上面的过程并不是很复杂, 但是通常需要编写很多代码, 而且为了显示错误信息, 在网页中经常要使用多种不同的控制结构。表单验证虽然简单, 但是实现起来却非常枯燥。

表单验证指南

下面是实现 CodeIgniter 表单验证的一个简易教程。

为了进行表单验证, 你需要这三样东西:

1. 一个包含表单的视图文件。
2. 一个包含“成功”信息的视图文件, 在成功提交后将被显示。
3. 一个接收并处理所提交数据的控制器方法。

让我们以一个会员注册表单为例来创建这三样东西。

表单

使用文本编辑器创建一个名为 myform.php 的文件, 在它里面插入如下代码, 并把它保存到你的 applications/views/ 目录下:

```
<html>
<head>
<title>My Form</title>
</head>
<body>

<?php echo validation_errors(); ?>

<?php echo form_open('form'); ?>

<h5>Username</h5>
<input type="text" name="username" value="" size="50" />

<h5>Password</h5>
<input type="text" name="password" value="" size="50" />

<h5>Password Confirm</h5>
<input type="text" name="passconf" value="" size="50" />
```

```

<h5>Email Address</h5>
<input type="text" name="email" value="" size="50" />

<div><input type="submit" value="Submit" /></div>

</form>

</body>
</html>

```

成功页面

使用文本编辑器创建一个名为 formsuccess.php 的文件，在它里面插入如下代码，并把它保存到你的 applications/views/ 目录下：

```

<html>
<head>
<title>My Form</title>
</head>
<body>

<h3>Your form was successfully submitted!</h3>

<p><?php echo anchor('form', 'Try it again!'); ?></p>

</body>
</html>

```

控制器

使用文本编辑器创建一个名为 Form.php 的控制器文件，在它里面插入如下代码，并把它保存到你的 application/controllers/ 目录下：

```

<?php

class Form extends CI_Controller {

    public function index()
    {
        $this->load->helper(array('form', 'url'));

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

        if ($this->form_validation->run() == FALSE)
        {
            $this->load->view('myform');
        }
        else

```

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

试一下!

访问类似于下面这样的 URL 来体验一下你的表单:

`example.com/index.php/form/`

如果你提交表单, 你会看到表单只是简单重新加载了, 这是因为你还没有设置任何验证规则。

由于你还没有告诉表单验证类验证什么东西, 它默认返回 `FALSE`, “`run()`” 方法只在全部成功匹配了你的规则后才会返回 `TRUE`。

解释

在这个页面上你会注意到以下几点:

例子中的表单 (`myform.php`) 是一个标准的 Web 表单, 除了以下两点:

1. 它使用了一个表单辅助函数来创建表单的起始标签。严格来说这并不是必要的, 你完全可以使用标准的 HTML 来创建, 使用辅助函数的好处是它生成 `action` 的时候, 是基于你配置文件来生成 URL 的, 这使得你的应用在更改 URL 时更具移植性。
2. 在表单的顶部你将注意到如下函数调用:

```
<?php echo validation_errors(); ?>
```

这个函数将会返回验证器返回的所有错误信息。如果没有错误信息, 它将返回空字符串。

控制器 (`Form.php`) 有一个方法: `index()`。这个方法初始化验证类, 并加载你视图中用到的表单辅助函数和 URL 辅助函数, 它也会执行验证流程, 基于验证是否成功, 它会重新显示表单或显示成功页面。

设置验证规则

CodeIgniter 允许你为单个表单域创建多个验证规则, 按顺序层叠在一起, 你也可以同时对表单域的数据进行预处理。要设置验证规则, 可以使用 `set_rules()` 方法:

```
$this->form_validation->set_rules();
```

上面的方法有 **三个** 参数:

1. 表单域名 - 就是你给表单域取的那个名字。

2. 表单域的“人性化”名字，它将被插入到错误信息中。例如，如果你有一个表单域叫做“user”，你可能会给它一个人性化的名字叫做“用户名”。
3. 为此表单域设置的验证规则。
4. （可选的）当此表单域设置自定义的错误信息，如果没有设置该参数，将使用默认的。

注解： 如果你想让表单域的名字保存在一个语言文件里，请参考[翻译表单域名称](#)

下面是个例子，在你的控制器（Form.php）中紧接着验证初始化函数之后，添加这段代码：

```
$this->form_validation->set_rules('username', 'Username', 'required');
$this->form_validation->set_rules('password', 'Password', 'required');
$this->form_validation->set_rules('passconf', 'Password Confirmation', 'required');
$this->form_validation->set_rules('email', 'Email', 'required');
```

你的控制器现在看起来像这样：

```
<?php
```

```
class Form extends CI_Controller {

    public function index()
    {
        $this->load->helper(array('form', 'url'));

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

        $this->form_validation->set_rules('username', 'Username', 'required');
        $this->form_validation->set_rules('password', 'Password', 'required',
            array('required' => 'You must provide a %s.'));
        $this->form_validation->set_rules('passconf', 'Password Confirmation', 'required');
        $this->form_validation->set_rules('email', 'Email', 'required');

        if ($this->form_validation->run() == FALSE)
        {
            $this->load->view('myform');
        }
        else
        {
            $this->load->view('formsuccess');
        }
    }
}
```

现在如果你不填写表单就提交，你将会看到错误信息。如果你填写了所有的表单域并提交，你会看到成功页。

注解： 当出现错误时表单页将重新加载，所有的表单域将会被清空，并没有被重新填

充。稍后我们再去处理这个问题。

使用数组来设置验证规则

在继续之前请注意, 如果你更喜欢通过一个操作设置所有规则的话, 你也可以使用一个数组来设置验证规则, 如果你使用这种方式, 你必须像下面这样来定义你的数组:

```
$config = array(
    array(
        'field' => 'username',
        'label' => 'Username',
        'rules' => 'required'
    ),
    array(
        'field' => 'password',
        'label' => 'Password',
        'rules' => 'required',
        'errors' => array(
            'required' => 'You must provide a %s.',
        ),
    ),
    array(
        'field' => 'passconf',
        'label' => 'Password Confirmation',
        'rules' => 'required'
    ),
    array(
        'field' => 'email',
        'label' => 'Email',
        'rules' => 'required'
    )
);

$this->form_validation->set_rules($config);
```

级联规则 (Cascading Rules)

CodeIgniter 允许你将多个规则连接在一起。让我们试一试, 修改规则设置函数中的第三个参数, 如下:

```
$this->form_validation->set_rules(
    'username', 'Username',
    'required|min_length[5]|max_length[12]|is_unique[users.username]',
    array(
        'required' => 'You have not provided %s.',
        'is_unique' => 'This %s already exists.'
    )
);
```

```
$this->form_validation->set_rules('password', 'Password', 'required');
$this->form_validation->set_rules('passconf', 'Password Confirmation', 'required|matches[password]');
$this->form_validation->set_rules('email', 'Email', 'required|valid_email|is_unique[users.email]');
```

上面的代码设置了以下规则：

1. 用户名表单域长度不得小于 5 个字符、不得大于 12 个字符。
2. 密码表单域必须跟密码确认表单域的数据一致。
3. 电子邮件表单域必须是一个有效邮件地址。

马上试试看！提交不合法的数据后你会看到新的错误信息，跟你设置的新规则相符。还有很多其他的验证规则，你可以阅读验证规则参考。

注解： 你也可以传一个包含规则的数组给 `set_rules()` 方法来替代字符串，例如：

```
$this->form_validation->set_rules('username', 'Username', array('required', 'min_length[5]'));
```

预处理数据

除了上面我们使用的那些验证函数，你还可以以多种方式来预处理你的数据。例如，你可以设置像这样的规则：

```
$this->form_validation->set_rules('username', 'Username', 'trim|required|min_length[5]|max_length[12]');
$this->form_validation->set_rules('password', 'Password', 'trim|required|min_length[8]');
$this->form_validation->set_rules('passconf', 'Password Confirmation', 'trim|required|matches[password]');
$this->form_validation->set_rules('email', 'Email', 'trim|required|valid_email');
```

在上面的例子里，我们去掉字符串两端空白（trimming），检查字符串的长度，确保两次输入的密码一致。

任何只有一个参数的 PHP 原生函数都可以被用作一个规则，比如“`htmlspecialchars`”，“`trim`”等等。

注解： 你一般会在验证规则 `**` 之后 `**` 使用预处理功能，这样如果发生错误，原数据将会被显示在表单。

重新填充表单

目前为止我们只是在处理错误，是时候用提交的数据重新填充表单了。CodeIgniter 为此提供了几个辅助函数，你最常用到的一个是：

```
set_value('field name')
```

打开 `myform.php` 视图文件并使用 `set_value()` 函数更新每个表单域的 值：

不要忘记在 `php:func:'set_value()'` 函数中包含每个表单域的名字！


```
<html>
<head>
<title>My Form</title>
</head>
<body>

<?php echo validation_errors(); ?>

<?php echo form_open('form'); ?>

<h5>Username</h5>
<input type="text" name="username" value="<?php echo set_value('username'); ?>" size="50" />

<h5>Password</h5>
<input type="text" name="password" value="<?php echo set_value('password'); ?>" size="50" />

<h5>Password Confirm</h5>
<input type="text" name="passconf" value="<?php echo set_value('passconf'); ?>" size="50" />

<h5>Email Address</h5>
<input type="text" name="email" value="<?php echo set_value('email'); ?>" size="50" />

<div><input type="submit" value="Submit" /></div>

</form>

</body>
</html>
```

现在刷新你的页面并提交表单触发一个错误, 你的表单域应该被重新填充了。

注解: 下面的 [类参考](#) 节包含了可以让你重填下拉菜单, 单选框和复选框的函数。

重要: 如果你使用一个数组作为一个表单域的名字, 那么函数的参数也应该是一个数组。例如:

```
<input type="text" name="colors[]" value="<?php echo set_value('colors[]'); ?>" size="50" />
```

更多信息请参考下面的[使用数组作为域名称](#)一节。

回调: 你自己的验证函数

验证系统支持设置你自己的验证函数, 这样你可以扩展验证类以适应你自己的需求。例如, 如果你需要查询数据库来检查用户名是否唯一, 你可以创建一个回调函数, 让我们来新建一个例子。

在你的控制器中, 将用户名的规则修改为:

```
$this->form_validation->set_rules('username', 'Username', 'callback_username_check');
```

然后在你的控制器中添加一个新的方法 `username_check()`。你的控制器现在看起来是这样:

```
<?php
```

```
class Form extends CI_Controller {

    public function index()
    {
        $this->load->helper(array('form', 'url'));

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

        $this->form_validation->set_rules('username', 'Username', 'callback_username_check');
        $this->form_validation->set_rules('password', 'Password', 'required');
        $this->form_validation->set_rules('passconf', 'Password Confirmation', 'required');
        $this->form_validation->set_rules('email', 'Email', 'required|is_unique[users.email]');

        if ($this->form_validation->run() == FALSE)
        {
            $this->load->view('myform');
        }
        else
        {
            $this->load->view('formsuccess');
        }
    }

    public function username_check($str)
    {
        if ($str == 'test')
        {
            $this->form_validation->set_message('username_check', 'The {field} field can not be test');
            return FALSE;
        }
        else
        {
            return TRUE;
        }
    }
}
```

重新载入表单并以“test”作为用户名提交数据, 你会看到表单域数据被传递到你的回调函数中处理了。

要调用一个回调函数只需把函数名加一个“callback_”前缀 ** 并放在验证规则里。如果你需要在你的回调函数中调用一个额外的参数, 你只需要在回调函数后面用 [] 把参数 (这个参数只能是字符串类型) 括起来, 例如: “callback_foo**[bar]”, 其中

bar 将成为你的回调函数中的第二个参数。

注解: 你也可以对传给你的表单数据进行处理并返回, 如果你的回调函数返回了除布尔型的 TRUE 或 FALSE 之外的任何值, 它将被认为是你新处理过的表单数据。

使用任何可调用的方法作为验证规则

如果回调的规则对你来说还不够好 (例如, 它们被限制只能定义在控制器中), 别失望, 还有一种方法来创建自定义的规则: 任何 `is_callable()` 函数返回 TRUE 的东西都可以作为规则。

看下面的例子:

```
$this->form_validation->set_rules(
    'username', 'Username',
    array(
        'required',
        array($this->users_model, 'valid_username')
    )
);
```

上面的代码将使用 `Users_model` 模型的 `valid_username()` 方法来作为验证规则。

当然, 这只是个例子, 规则不只限于使用模型的方法, 你可以使用任何对象和方法来接受域值作为第一个参数。如果你使用 PHP 5.3+ , 还可以使用匿名方法:

```
$this->form_validation->set_rules(
    'username', 'Username',
    array(
        'required',
        function($value)
        {
            // Check $value
        }
    )
);
```

但是, 由于可调用的规则并不是一个字符串, 也没有一个规则名, 所以当你需要为它们设置相应的错误消息时会有麻烦。为了解决这个问题, 你可以将这样的规则放到一个数组的第二个值, 第一个值放置规则名:

```
$this->form_validation->set_rules(
    'username', 'Username',
    array(
        'required',
        array('username_callable', array($this->users_model, 'valid_username'))
    )
);
```

下面是使用匿名方法 (PHP 5.3+) 的版本:

```

$this->form_validation->set_rules(
    'username', 'Username',
    array(
        'required',
        array(
            'username_callable',
            function($str)
            {
                // Check validity of $str and return TRUE or FALSE
            }
        )
    )
);

```

设置错误信息

所有原生的错误信息都位于下面的语言文件中: **language/english/form_validation_lang.php**

To set your own global custom message for a rule, you can either extend/override the language file by creating your own in **application/language/english/form_validation_lang.php** (read more about this in the [Language Class](#) documentation), or use the following method:

```

$this->form_validation->set_message('rule', 'Error Message');

```

如果你要为某个域的某个规则设置你的自定义信息, 可以使用 `set_rules()` 方法:

```

$this->form_validation->set_rules('field_name', 'Field Label', 'rule1|rule2|rule3',
    array('rule2' => 'Error Message on rule2 for this field_name')
);

```

其中, rule 是该规则的名称, Error Message 为该规则显示的错误信息。

如果你希望在错误信息中包含域的人性化名称, 或者某些规则设置的一个可选参数 (例如: `max.length`), 你可以在消息中使用 **{field}** 和 **{param}** 标签:

```

$this->form_validation->set_message('min_length', '{field} must have at least {param} characters');

```

如果域的人性化名称为 Username , 并有一个规则 `min_length[5]` , 那么错误信息会显示: “Username must have at least 5 characters.”

注解: 老的 `sprintf()` 方法和在字符串使用 `%s` 也还可以工作, 但是会覆写掉上面的标签。所以你同时只应该使用两个中的一个。

在上面回调的例子中, 错误信息是通过方法的名称 (不带 “callback_” 前缀) 来设置的:

```

$this->form_validation->set_message('username_check')

```

翻译表单域名称

如果你希望将传递给 `set_rules()` 方法的人性化名称存储在一个语言文件中, 使它们能被翻译成其他语言, 你可以这么做:

首先, 给人性化名称添加一个前缀: **lang:**, 如下:

```
$this->form_validation->set_rules('first_name', 'lang:first_name', 'required');
```

然后, 将该名称保存到你的某个语言文件数组中 (不带前缀):

```
$lang['first_name'] = 'First Name';
```

注解: 如果你保存的语言文件没有自动被 CI 加载, 你要记住在你的控制器中使用下面的方法手工加载:

```
$this->lang->load('file_name');
```

关于语言文件的更多信息, 参看[语言类](#)。

更改错误定界符

在默认情况下, 表单验证类会使用 `<p>` 标签来分割每条错误信息。你可以通过全局的, 单独的, 或者通过配置文件对其进行自定义。

1. **全局的修改定界符** 要在全局范围内修改错误定界符, 你可以在控制器方法中加载表单验证类之后, 使用下面的代码:

```
$this->form_validation->set_error_delimiters('<div class="error">', '</div>');
```

在这个例子中, 我们改成使用 `<div>` 标签来作为定界符。

2. **单独的修改定界符** 有两个错误生成方法可以用于设置它们自己的定界符, 如下:

```
<?php echo form_error('field name', '<div class="error">', '</div>'); ?>
```

或者:

```
<?php echo validation_errors('<div class="error">', '</div>'); ?>
```

3. **在配置文件中设置定界符** 你还可以在配置文件 `application/config/form_validation.php` 中定义错误定界符, 如下:

```
$config['error_prefix'] = '<div class="error_prefix">';  
$config['error_suffix'] = '</div>';
```

单独显示错误

如果你喜欢紧挨着每个表单域显示错误信息而不是显示为一个列表, 你可以使用 `form_error()` 方法。

尝试一下! 修改你的表单如下:

```
<h5>Username</h5>
<?php echo form_error('username'); ?>
<input type="text" name="username" value="<?php echo set_value('username'); ?>" size="50" />

<h5>Password</h5>
<?php echo form_error('password'); ?>
<input type="text" name="password" value="<?php echo set_value('password'); ?>" size="50" />

<h5>Password Confirm</h5>
<?php echo form_error('passconf'); ?>
<input type="text" name="passconf" value="<?php echo set_value('passconf'); ?>" size="50" />

<h5>Email Address</h5>
<?php echo form_error('email'); ?>
<input type="text" name="email" value="<?php echo set_value('email'); ?>" size="50" />
```

如果没有错误信息, 将不会显示。如果有错误信息, 将会在输入框的旁边单独显示。

重要: 如果你使用一个数组作为一个表单域的名字, 那么函数的参数也应该是一个数组。例如:

```
<?php echo form_error('options[size]'); ?>
<input type="text" name="options[size]" value="<?php echo set_value("options[size]"); ?>" size="50" />
```

更多信息, 请参考下面的[使用数组作为域名称](#)一节。

验证数组 (除 `$_POST` 数组)

有时你可能希望对一个单纯的数组进行验证, 而不是对 `$_POST` 数组。

在这种情况下, 你可以先定义要验证的数组:

```
$data = array(
    'username' => 'johndoe',
    'password' => 'mypassword',
    'passconf' => 'mypassword'
);

$this->form_validation->set_data($data);
```

Creating validation rules, running the validation, and retrieving error messages works the same whether you are validating `$_POST` data or another array of your choice.

重要: You have to call the `set_data()` method *before* defining any validation rules.

重要: 如果你想验证多个数组, 那么你应该在验证下一个新数组之前先调用 `reset_validation()` 方法。

更多信息, 请参数下面的[类参考](#)一节。

将一系列验证规则保存到一个配置文件

表单验证类的一个不错的特性是, 它允许你将整个应用的所有验证规则存储到一个配置文件中。你可以对这些规则进行分组, 这些组既可以在匹配控制器和方法时自动加载, 也可以在需要时手动调用。

如何保存你的规则

如果要保存验证规则, 你需要在 `application/config/` 目录下创建一个名为 `form_validation.php` 的文件。然后在该文件中, 将验证规则保存在数组 `$config` 中即可。和之前介绍的一样, 验证规则数组格式如下:

```
$config = array(
    array(
        'field' => 'username',
        'label' => 'Username',
        'rules' => 'required'
    ),
    array(
        'field' => 'password',
        'label' => 'Password',
        'rules' => 'required'
    ),
    array(
        'field' => 'passconf',
        'label' => 'Password Confirmation',
        'rules' => 'required'
    ),
    array(
        'field' => 'email',
        'label' => 'Email',
        'rules' => 'required'
    )
);
```

你的验证规则会被自动加载, 当用户触发 `run()` 方法时被调用。

请务必将数组名称定义成 `$config`。

创建规则集

为了将你的多个规则组织成规则集, 你需要将它们放置到子数组中。请参考下面的例子, 在此例中我们设置了两组规则集, 我们分别命名为 “signup” 和 “email”, 你可以根据自己的需求任意命名:

```
$config = array(
    'signup' => array(
        array(
            'field' => 'username',
            'label' => 'Username',
            'rules' => 'required'
        ),
        array(
            'field' => 'password',
            'label' => 'Password',
            'rules' => 'required'
        ),
        array(
            'field' => 'passconf',
            'label' => 'Password Confirmation',
            'rules' => 'required'
        ),
        array(
            'field' => 'email',
            'label' => 'Email',
            'rules' => 'required'
        )
    ),
    'email' => array(
        array(
            'field' => 'emailaddress',
            'label' => 'EmailAddress',
            'rules' => 'required|valid_email'
        ),
        array(
            'field' => 'name',
            'label' => 'Name',
            'rules' => 'required|alpha'
        ),
        array(
            'field' => 'title',
            'label' => 'Title',
            'rules' => 'required'
        ),
        array(
            'field' => 'message',
            'label' => 'MessageBody',
            'rules' => 'required'
        )
    )
);
```


调用某组验证规则

为了调用特定组的验证规则, 你可以将它的名称传给 `run()` 方法。例如, 使用 `signup` 规则你可以这样:

```
if ($this->form_validation->run('signup') == FALSE)
{
    $this->load->view('myform');
}
else
{
    $this->load->view('formsuccess');
}
```

将控制器方法和规则集关联在一起

调用一组规则的另一种方法是将控制器方法和规则集关联在一起 (这种方法也更自动), 例如, 假设你有一个控制器类 `Member` 和一个方法 `signup`, 你的类如下:

```
<?php

class Member extends CI_Controller {

    public function signup()
    {
        $this->load->library('form_validation');

        if ($this->form_validation->run() == FALSE)
        {
            $this->load->view('myform');
        }
        else
        {
            $this->load->view('formsuccess');
        }
    }
}
```

在你的验证规则配置文件中, 使用 `member/signup` 来给这组规则集命名:

```
$config = array(
    'member/signup' => array(
        array(
            'field' => 'username',
            'label' => 'Username',
            'rules' => 'required'
        ),
        array(
            'field' => 'password',
            'label' => 'Password',
```

```

        'rules' => 'required'
    ),
    array(
        'field' => 'passconf',
        'label' => 'PasswordConfirmation',
        'rules' => 'required'
    ),
    array(
        'field' => 'email',
        'label' => 'Email',
        'rules' => 'required'
    )
);

```

当一组规则的名称和控制器类/方法名称完全一样时，它会在该控制器类/方法中自动被 `run()` 方法调用。

使用数组作为域名称

表单验证类支持使用数组作为域名称，比如：

```
<input type="text" name="options[]" value="" size="50" />
```

如果你将域名称定义为数组，那么在使用域名称作为参数的[辅助函数函数](#)时，你必须传递给他们与域名称完全一样的数组名，对这个域名称的验证规则也一样。

例如，为上面的域设置验证规则：

```
$this->form_validation->set_rules('options[]', 'Options', 'required');
```

或者，为上面的域显示错误信息：

```
<?php echo form_error('options[]'); ?>
```

或者，重新填充该域的值：

```
<input type="text" name="options[]" value="<?php echo set_value('options[]'); ?>" size="50"
```

你也可以使用多维数组作为域的名称，例如：

```
<input type="text" name="options[size]" value="" size="50" />
```

甚至：

```
<input type="text" name="sports[nba][basketball]" value="" size="50" />
```

和上面的例子一样，你必须在辅助函数中使用完全一样的数组名：

```
<?php echo form_error('sports[nba][basketball]'); ?>
```

如果你正在使用复选框（或其他拥有多个选项的域），不要忘了在每个选项后加个空的方括号，这样，所有的选择才会被添加到 POST 数组中：

```
<input type="checkbox" name="options[]" value="red" />
<input type="checkbox" name="options[]" value="blue" />
<input type="checkbox" name="options[]" value="green" />
```

或者, 使用多维数组:

```
<input type="checkbox" name="options[color][]" value="red" />
<input type="checkbox" name="options[color][]" value="blue" />
<input type="checkbox" name="options[color][]" value="green" />
```

当你使用辅助函数时, 也要添加方括号:

```
<?php echo form_error('options[color][]'); ?>
```

规则参考

下表列出了所有可用的原生规则:

规则	参数	描述	例子
required	No	如果表单元素为空, 返回 FALSE	
matches	Yes	如果表单元素值与参数中对应的表单字段的值不相等, 返回 FALSE	matches[form_item]
regex_match	Yes	如果表单元素不匹配正则表达式, 返回 FALSE	regex_match[/regex/]
differs	Yes	如果表单元素值与参数中对应的表单字段的值相等, 返回 FALSE	differs[form_item]
is_unique	Yes	如果表单元素值在指定的表和字段中并不唯一, 返回 FALSE 注意: 这个规则需要启用 查询构造器	is_unique[table.field]
min_length	Yes	如果表单元素值的长度小于参数值, 返回 FALSE	min_length[3]
max_length	Yes	如果表单元素值的长度大于参数值, 返回 FALSE	max_length[12]
exact_length	Yes	如果表单元素值的长度不等于参数值, 返回 FALSE	exact_length[8]
greater_than	Yes	如果表单元素值小于或等于参数值或非数字, 返回 FALSE	greater_than[8]
greater_than_or_equal_to	Yes	如果表单元素值小于参数值或非数字, 返回 FALSE	greater_than_equal_to[8]
less_than	Yes	如果表单元素值大于或等于参数值或非数字, 返回 FALSE	less_than[8]
less_than_equal_to	Yes	如果表单元素值大于参数值或非数字, 返回 FALSE	less_than_equal_to[8]
in_list	Yes	如果表单元素值不在规定的列表中, 返回 FALSE	in_list[red,blue,green]
alpha	No	如果表单元素值包含除字母以外的其他字符, 返回 FALSE	
alpha_numeric	No	如果表单元素值包含除字母和数字以外的其他字符, 返回 FALSE	
alpha_numeric_spaces	No	如果表单元素值包含除字母、数字和空格以外的其他字符, 返回 FALSE 应该在 trim 之后使用, 避免首尾的空格	
alpha_dash	No	如果表单元素值包含除字母/数字/下划线/破折号以外的其他字符, 返回 FALSE	
numeric	No	如果表单元素值包含除数字以外的字符, 返回 FALSE	
integer	No	如果表单元素包含除整数以外的字符, 返回 FALSE	
decimal	No	如果表单元素包含非十进制数字时, 返回 FALSE	
is_natural	No	如果表单元素值包含了非自然数的其他数值 (不包括零), 返回 FALSE 自然数形如: 0、1、2、3 等等。	
is_natural_not_zero	No	如果表单元素值包含了非自然数的其他数值 (包括零), 返回 FALSE 非零的自然数: 1、2、3 等等。	
valid_url	No	如果表单元素值包含不合法的 URL, 返回 FALSE	
valid_email	No	如果表单元素值包含不合法的 email 地址, 返回 FALSE	
valid_emails	No	如果表单元素值包含不合法的 email 地址 (地址之间用逗号分割), 返回 FALSE	
valid_ip	No	如果表单元素值不是一个合法的 IP 地址, 返回 FALSE 通过可选参数 “ipv4” 或 “ipv6” 来指定 IP 地址格式。	
valid_base64	No	如果表单元素值包含除了 base64 编码字符之外的其他字符, 返回 FALSE	

注解: 这些规则也可以作为独立的函数被调用, 例如:

```
$this->form_validation->required($string);
```

注解: 你也可以使用任何一个接受两个参数的原生 PHP 函数 (其中至少有一个参数是必须的, 用于传递域值)

预处理参考

下表列出了所有可用的预处理方法:

名称	参数	描述
prep_for_form	No	将特殊字符的转换, 以便可以在表单域中显示 HTML 数据, 而不会破坏它
prep_url	No	当 URL 丢失 “http://” 时, 添加 “http://”
strip_image_tags	No	移除 HTML 中的 image 标签, 只保留 URL
en-code_php_tags	No	将 PHP 标签转成实体

注解: 你也可以使用任何一个接受一个参数的原生 PHP 函数。例如: `trim()`、`htmlspecialchars()`、`urldecode()` 等

类参考

class CI_Form_validation

```
set_rules($field[, $label = '', $rules = ''])
```

参数

- **\$field** (*string*) – Field name
- **\$label** (*string*) – Field label
- **\$rules** (*mixed*) – Validation rules, as a string list separated by a pipe “|”, or as an array or rules

返回 CI_Form_validation instance (method chaining)

返回类型 CI_Form_validation

允许您设置验证规则, 如在本教程上面描述的:

- 设置验证规则
- 将一系列验证规则保存到一个配置文件

```
run([$group = ''])
```

参数

- **\$group** (*string*) – The name of the validation group to run

返回 TRUE on success, FALSE if validation failed

返回类型 bool

运行验证程序。成功返回 TRUE，失败返回 FALSE。您也可以传一个验证规则集的名称作为参数，参考[将一系列验证规则保存到一个配置文件](#)

set_message(\$lang[, \$val = ''])

参数

- **\$lang** (*string*) – The rule the message is for
- **\$val** (*string*) – The message

返回 CI_Form_validation instance (method chaining)

返回类型 CI_Form_validation

允许您设置自定义错误消息，参考[设置错误信息](#)

set_error_delimiters([\$prefix = '<p>', \$suffix = '</p>'])

参数

- **\$prefix** (*string*) – Error message prefix
- **\$suffix** (*string*) – Error message suffix

返回 CI_Form_validation instance (method chaining)

返回类型 CI_Form_validation

设置错误消息的前缀和后缀。

set_data(\$data)

参数

- **\$data** (*array*) – Array of data validate

返回 CI_Form_validation instance (method chaining)

返回类型 CI_Form_validation

允许你设置一个数组来进行验证，取代默认的 \$_POST 数组

reset_validation()

返回 CI_Form_validation instance (method chaining)

返回类型 CI_Form_validation

当你验证多个数组时，该方法可以重置验证规则，当验证下一个新数组时应该调用它。

error_array()

返回 Array of error messages

返回类型 array

返回错误信息数组。

```
error_string([$prefix = '[', $suffix = ''])
```

参数

- **\$prefix** (*string*) – Error message prefix
- **\$suffix** (*string*) – Error message suffix

返回 Error messages as a string

返回类型 string

返回所有的错误信息（和 `error_array()` 返回结果一样），并使用换行符分割格式化成为字符串

```
error($field[, $prefix = '[', $suffix = ''])
```

参数

- **\$field** (*string*) – Field name
- **\$prefix** (*string*) – Optional prefix
- **\$suffix** (*string*) – Optional suffix

返回 Error message string

返回类型 string

返回特定域的错误消息，也可以添加一个前缀和/或后缀（通常是 HTML 标签）

```
has_rule($field)
```

参数

- **\$field** (*string*) – Field name

返回 TRUE if the field has rules set, FALSE if not

返回类型 bool

检查某个域是否有验证规则。

辅助函数参考

请参考[表单辅助函数](#)手册了解以下函数：

- `form_error()`
- `validation_errors()`
- `set_value()`
- `set_select()`
- `set_checkbox()`

- `set_radio()`

注意这些都是过程式的函数，所以 **不需要** 添加 `$this->form_validation` 就可以直接调用它们。

8.1.11 FTP 类

CodeIgniter 的 FTP 类允许你传输文件至远程服务器，也可以对远程文件进行移动、重命名或删除操作。FTP 类还提供了一个“镜像”功能，允许你将你本地的一个目录通过 FTP 整个的同步到远程服务器上。

注解： 只支持标准的 FTP 协议，不支持 SFTP 和 SSL FTP 。

- 使用 FTP 类
 - 初始化类
 - 使用示例
- 类参考

使用 FTP 类

初始化类

正如 CodeIgniter 中的其他类一样，在你的控制器中使用 `$this->load->library()` 方法来初始化 FTP 类：

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

初始化之后，FTP 类的对象就可以这样访问：

```
$this->ftp
```

使用示例

在这个例子中，首先建立一个到 FTP 服务器的连接，接着读取一个本地文件然后以 ASCII 模式上传到服务器上。文件的权限被设置为 755 。

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

$config['hostname'] = 'ftp.example.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug']    = TRUE;

$this->ftp->connect($config);
```



```
$this->ftp->upload('/local/path/to/myfile.html', '/public_html/myfile.html', 'ascii', 0775);

$this->ftp->close();
```

下面的例子从 FTP 服务器上获取文件列表。

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

$config['hostname'] = 'ftp.example.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug']    = TRUE;

$this->ftp->connect($config);

$list = $this->ftp->list_files('/public_html/');

print_r($list);

$this->ftp->close();
```

下面的例子在 FTP 服务器上创建了一个本地目录的镜像。

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

$config['hostname'] = 'ftp.example.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['debug']    = TRUE;

$this->ftp->connect($config);

$this->ftp->mirror('/path/to/myfolder/', '/public_html/myfolder/');

$this->ftp->close();
```

类参考

class CI_FTP

connect(*[\$config = array()]*)

参数

- **\$config** (*array*) – Connection values

返回 TRUE on success, FALSE on failure

返回类型 bool

连接并登录到 FTP 服务器，通过向函数传递一个数组来设置连接参数，或者你可以把这些参数保存在一个配置文件中。

下面例子演示了如何手动设置参数：

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

$config['hostname'] = 'ftp.example.com';
$config['username'] = 'your-username';
$config['password'] = 'your-password';
$config['port']     = 21;
$config['passive']  = FALSE;
$config['debug']    = TRUE;

$this->ftp->connect($config);
```

在配置文件中设置 FTP 参数

如果你喜欢，你可以把 FTP 参数保存在一个配置文件中，只需创建一个名为 ftp.php 的文件，然后把 \$config 数组添加到该文件中，然后将文件保存到 `application/config/ftp.php`，它就会自动被读取。

可用的连接选项

选项名称	默认值	描述
host-name	n/a	FTP 主机名（通常类似于这样：ftp.example.com）
user-name	n/a	FTP 用户名
pass-word	n/a	FTP 密码
port	21	FTP 服务端口
debug	FALSE	TRUE/FALSE (boolean): 是否开启调试模式，显示错误信息
passive	TRUE	TRUE/FALSE (boolean): 是否使用被动模式

```
upload($lospath, $rempath[, $mode = 'auto'[, $permissions = NULL]])
```

参数

- **\$lospath** (*string*) – Local file path
- **\$rempath** (*string*) – Remote file path
- **\$mode** (*string*) – FTP mode, defaults to 'auto' (options are: 'auto', 'binary', 'ascii')
- **\$permissions** (*int*) – File permissions (octal)

返回 TRUE on success, FALSE on failure

返回类型 bool

将一个文件上传到你的服务器上。必须指定本地路径和远程路径这两个参数，而传输模式和权限设置这两个参数则是可选的。例如：

```
$this->ftp->upload('/local/path/to/myfile.html', '/public_html/my-file.html', 'ascii', 0775);
```

如果使用了 auto 模式，将根据源文件的扩展名来自动选择传输模式。

设置权限必须使用一个八进制的权限值。

```
download($rempath, $lospath[, $mode = 'auto'])
```

参数

- **\$rempath** (*string*) – Remote file path
- **\$lospath** (*string*) – Local file path
- **\$mode** (*string*) – FTP mode, defaults to 'auto' (options are: 'auto', 'binary', 'ascii')

返回 TRUE on success, FALSE on failure

返回类型 bool

从你的服务器下载一个文件。必须指定远程路径和本地路径，传输模式是可选的。例如：

```
$this->ftp->download('/public_html/myfile.html', '/local/path/to/myfile.html', 'ascii');
```

如果使用了 auto 模式，将根据源文件的扩展名来自动选择传输模式。

如果下载失败（包括 PHP 没有写入本地文件的权限）函数将返回 FALSE。

```
rename($old_file, $new_file[, $move = FALSE])
```

参数

- **\$old_file** (*string*) – Old file name
- **\$new_file** (*string*) – New file name
- **\$move** (*bool*) – Whether a move is being performed

返回 TRUE on success, FALSE on failure

返回类型 bool

允许你重命名一个文件。需要指定原文件的文件路径和名称，以及新的文件路径和名称。

```
// Renames green.html to blue.html
```

```
$this->ftp->rename('/public_html/foo/green.html', '/public_html/foo/blue.html');
```

```
move($old_file, $new_file)
```

参数

- **\$old_file** (*string*) – Old file name

- **\$new_file** (*string*) – New file name

返回 TRUE on success, FALSE on failure

返回类型 bool

允许你移动一个文件。需要指定原路径和目的路径:

```
// Moves blog.html from "joe" to "fred"
$this->ftp->move('/public_html/joe/blog.html', '/public_html/fred/blog.html');
```

注解: 如果目的文件名和原文件名不同, 文件将会被重命名。

delete_file(\$filepath)

参数

- **\$filepath** (*string*) – Path to file to delete

返回 TRUE on success, FALSE on failure

返回类型 bool

用于删除一个文件。需要提供原文件的路径。

```
$this->ftp->delete_file('/public_html/joe/blog.html');
```

delete_dir(\$filepath)

参数

- **\$filepath** (*string*) – Path to directory to delete

返回 TRUE on success, FALSE on failure

返回类型 bool

用于删除一个目录以及该目录下的所有文件。需要提供目录的路径（以斜线结尾）。

重要: 使用该方法要非常小心! 它会递归的删除目录下的所有内容, 包括子目录和所有文件。请确保你提供的路径是正确的。你可以先使用 `list_files()` 方法来验证下路径是否正确。

```
$this->ftp->delete_dir('/public_html/path/to/folder/');
```

list_files(\$path = '.')

参数

- **\$path** (*string*) – Directory path

返回 An array list of files or FALSE on failure

返回类型 array

用于获取服务器上某个目录的文件列表, 你需要指定目录路径。

```
$list = $this->ftp->list_files('/public_html/');  
print_r($list);
```

mirror(\$lospath, \$rempath)

参数

- **\$lospath** (*string*) – Local path
- **\$rempath** (*string*) – Remote path

返回 TRUE on success, FALSE on failure

返回类型 bool

递归的读取文本的一个目录和它下面的所有内容（包括子目录），然后通过 FTP 在远程服务器上创建一个镜像。无论原文件的路径和目录结构是什么样的，都会在远程服务器上一模一样的重建。你需要指定一个原路径和目的路径：

```
$this->ftp->mirror('/path/to/myfolder/', '/public_html/myfolder/');
```

mkdir(\$path[, \$permissions = NULL])

参数

- **\$path** (*string*) – Path to directory to create
- **\$permissions** (*int*) – Permissions (octal)

返回 TRUE on success, FALSE on failure

返回类型 bool

用于在服务器上创建一个目录。需要指定目录的路径并以斜线结尾。

还可以通过第二个参数传递一个八进制的值设置权限。

```
// Creates a folder named "bar"  
$this->ftp->mkdir('/public_html/foo/bar/', 0755);
```

chmod(\$path, \$perm)

参数

- **\$path** (*string*) – Path to alter permissions for
- **\$perm** (*int*) – Permissions (octal)

返回 TRUE on success, FALSE on failure

返回类型 bool

用于设置文件权限。需要指定你想修改权限的文件或目录的路径：

```
// Chmod "bar" to 755  
$this->ftp->chmod('/public_html/foo/bar/', 0755);
```

chANGEDIR(\$path[, \$suppress_debug = FALSE])

参数

- **\$path** (*string*) – Directory path
- **\$suppress_debug** (*bool*) – Whether to turn off debug messages for this command

返回 TRUE on success, FALSE on failure

返回类型 bool

用于修改当前工作目录到指定路径。

如果你希望使用这个方法作为 `is_dir()` 的一个替代, `$suppress_debug` 参数将很有用。

close()

返回 TRUE on success, FALSE on failure

返回类型 bool

断开和服务器的连接。当你上传完毕时, 建议使用这个函数。

8.1.12 图像处理类

CodeIgniter 的图像处理类可以使你完成以下的操作:

- 调整图像大小
- 创建缩略图
- 图像裁剪
- 图像旋转
- 添加图像水印

可以很好的支持三个主流的图像库: GD/GD2、NetPBM 和 ImageMagick 。

注解: 添加水印操作仅仅在使用 GD/GD2 时可用。另外, 即使支持其他的图像处理库, 但是为了计算图像的属性, GD 仍是必需的。然而在进行图像处理操作时, 还是会使用你指定的库。

- 初始化类
 - 处理图像
 - 处理函数
 - 参数
 - 在配置文件中设置参数
- 添加图像水印
 - 水印的两种类型
 - 给图像添加水印
 - 水印处理参数
 - * Text 参数
 - * Overlay 参数
- 类参考

初始化类

跟 CodeIgniter 中的其他类一样, 可以在你的控制器中使用 `$this->load->library()` 方法加载图像处理类:

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

一旦加载, 图像处理类就可以像下面这样使用:

```
$this->image_lib
```

处理图像

不管你想进行何种图像处理操作 (调整大小, 图像裁剪, 图像旋转, 添加水印), 通常过程都是一样的。你会先设置一些你想进行的图像操作的参数, 然后调用四个可用方法中的一个。例如, 创建一个图像缩略图:

```
$config['image_library'] = 'gd2';
$config['source_image'] = '/path/to/image/mypic.jpg';
$config['create_thumb'] = TRUE;
$config['maintain_ratio'] = TRUE;
$config['width']        = 75;
$config['height']       = 50;

$this->load->library('image_lib', $config);

$this->image_lib->resize();
```

以上代码告诉 `image_resize` 函数去查找位于 `source_image` 目录下的名为 `mypic.jpg` 的图片, 然后运用 GD2 图像库创建一个 75 X 50 像素的缩略图。当 `maintain_ratio` 选项设为 `TRUE` 时, 生成的缩略图将保持图像的纵横比例, 同时尽可能的在宽度和高度上接近所设定的 `width` 和 `height`。缩略图将被命名为类似 `mypic_thumb.jpg` 的形式。

注解: 为了让图像类能进行所有操作, 包含图片的文件夹必须开启可写权限。

注解: 图像处理的某些操作可能需要大量的服务器内存。如果在处理图像时, 你遇到了内存不足错误, 您可能需要限制图像大小的最大值, 和/或调整 PHP 的内存限制。

处理函数

有五个处理函数可以调用:

- `$this->image_lib->resize()`
- `$this->image_lib->crop()`
- `$this->image_lib->rotate()`
- `$this->image_lib->watermark()`

当调用成功时, 这些函数会返回 `TRUE`, 否则会返回 `FALSE`。如果调用失败时, 用以下函数可以获取错误信息:

```
echo $this->image_lib->display_errors();
```

下面是一个好的做法, 将函数调用放在条件判断里, 当调用失败时显示错误的信息:

```
if ( ! $this->image_lib->resize() )
{
    echo $this->image_lib->display_errors();
}
```

注解: 你也可以给错误信息指定 HTML 格式, 像下面这样添加起始和结束标签:

```
$this->image_lib->display_errors('<p>', '</p>');
```

参数

你可以用下面的参数来对图像处理进行配置, 满足你的要求。

注意, 不是所有的参数都可以应用到每一个函数中。例如, x/y 轴参数只能被图像裁剪使用。但是, 宽度和高度参数对裁剪函数是无效的。下表的“可用性”一栏将指明哪些函数可以使用对应的参数。

“可用性”符号说明:

- R - 调整图像大小
- C - 图像裁剪
- X - 图像旋转
- W - 添加图像水印

参数	默认值	选项	描述	可用性
image_library	GD2	GD, GD2, ImageMagick, NetPBM	设置要使用的图像库	R, C, X, W
library_path	None	None	设置 ImageMagick 或 NetPBM 库在服务器上的路径。要使用它们中的任何一个, 你都需要设置它们的路径。	R, C, X
source_image	image	None	设置原始图像的名称和路径。路径只能是相对或绝对的服务器路径, 不能使用 URL。	R, C, S, W
dynamic_output	FALSE	TRUE/ FALSE (boolean)	决定新生成的图像是要写入硬盘还是内存中。注意, 如果是生成到内存的话, 一次只能显示一副图像, 而且不能调整它在你页面中的位置, 它只是简单的将图像数据以及图像的 HTTP 头发送到浏览器。	R, C, X, W
file_permissions	0644	(integer)	设置生成图像文件的权限。注意: 权限值为八进制表示法。	R, C, X, W
quality	90%	1 - 100%	设置图像的品质。品质越高, 图像文件越大。	R, C, X, W
new_image	None	None	设置目标图像的名称和路径。创建图像副本时使用该参数, 路径只能是相对或绝对的服务器路径, 不能使用 URL。	R, C, X, W
width	None	None	设置你想要的图像宽度。	R, C
height	None	None	设置你想要的图像高度。	R, C
create_thumb	FALSE	TRUE/ FALSE (boolean)	告诉图像处理函数生成缩略图。	R
thumb_name	thumb	None	指定缩略图后缀, 它会被插入到文件扩展名的前面, 所以 mypic.jpg 文件会变成 mypic_thumb.jpg	R
maintain_ratio	TRUE	TRUE/ FALSE (boolean)	指定是否在缩放或使用硬值的时候使图像保持原始的纵横比例。	R, C
master_dim	auto	auto, width, height	指定一个选项作为缩放和创建缩略图时的主轴。例如, 你想要将一张图片缩放到 100×75 像素。如果原来的图像的大小不能完美的缩放到这个尺寸, 那么由这个参数决定把哪个轴作为硬值。“auto”依据图片到底是过高还是过长自动设定轴。	R
rotation_angle	None	90, 180, 270, vrt, hor	指定图片旋转的角度。注意, 旋转是逆时针的, 如果想向右转 90 度, 就得把这个参数定义为 270。	X
x_axis	None	None	为图像的裁剪设定 X 轴上的长度。例如, 设为 30 就是将图片左边的 30 像素裁去。	C

在配置文件中设置参数

如果你不喜欢使用上面的方法来设置参数，你可以将参数保存到配置文件中。你只需简单的创建一个文件 `image_lib.php` 并将 `$config` 数组放到该文件中，然后保存文件到 **`config/image_lib.php`**，这些参数将会自动被使用。如果你在配置文件中设置参数，那么你就不需要使用 `$this->image_lib->initialize()` 方法了。

添加图像水印

水印处理功能需要 GD/GD2 库的支持。

水印的两种类型

你可以使用以下两种图像水印处理方式：

- **Text**：水印信息将以文字方式生成，要么使用你所指定的 TrueType 字体，要么使用 GD 库所支持的内部字体。如果你要使用 TrueType 版本，那么你安装的 GD 库必须是以支持 TrueType 的形式编译的（大多数都是，但不是所有）。
- **Overlay**：水印信息将以图像方式生成，新生成的水印图像（通常是透明的 PNG 或者 GIF）将覆盖在原图像上。

给图像添加水印

类似使用其他类型的图像处理函数（resizing、cropping 和 rotating），你也要对水印处理函数进行参数设置来生成你要的结果，例子如下：

```
$config['source_image'] = '/path/to/image/mypic.jpg';
$config['wm_text'] = 'Copyright 2006 - John Doe';
$config['wm_type'] = 'text';
$config['wm_font_path'] = './system/fonts/texb.ttf';
$config['wm_font_size'] = '16';
$config['wm_font_color'] = 'ffffff';
$config['wm_vrt_alignment'] = 'bottom';
$config['wm_hor_alignment'] = 'center';
$config['wm_padding'] = '20';

$this->image_lib->initialize($config);

$this->image_lib->watermark();
```

上面的例子是使用 16 像素 True Type 字体来生成文本水印“Copyright 2006 - John Doe”，该水印将出现在离图像底部 20 像素的中下部位置。

注解： 当调用图像类处理图像时，所有的目标图片必须有“写入”权限，例如：777

水印处理参数

下表列举的参数对于两种水印处理方式（text 或 overlay）都适用。

参数	默认值	选项	描述
wm_type	text	text, overlay	设置想要使用的水印处理类型。
source_image	None	None	设置原图像的名称和路径，路径必须是相对或绝对路径，不能是 URL。
dynamic_output	FALSE	TRUE/FALSE (boolean)	决定新生成的图像是要写入硬盘还是内存中。注意，如果是生成到内存的话，一次只能显示一副图像，而且不能调整它在你页面中的位置，它只是简单的将图像数据以及图像的 HTTP 头发送到浏览器。
quality	90%	1 - 100%	设置图像的品质。品质越高，图像文件越大。
wm_padding	None	A number	内边距，以像素为单位，是水印与图片边缘之间的距离。
wm_vrt_alignment	top	middle, bottom	设置水印图像的垂直对齐方式。
wm_hor_alignment	center	center, right	设置水印图像的水平对齐方式。
wm_hor_offset	None	None	你可以指定一个水平偏移量（以像素为单位），用于设置水印的位置。偏移量通常是向右移动水印，除非你把水平对齐方式设置为“right”，那么你的偏移量将会向左移动水印。
wm_vrt_offset	None	None	你可以指定一个垂直偏移量（以像素为单位），用于设置水印的位置。偏移量通常是向下移动水印，除非你把垂直对齐方式设置为“bottom”，那么你的偏移量将会向上移动水印。

Text 参数 下表列举的参数只适用于 text 水印处理方式。

参数	默认值	选项	描述
wm_text	None	None	你想作为水印显示的文本。通常是一份版权声明。
wm_font_path	None	None	你想使用的 TTF 字体 (TrueType) 在服务器上的路径。如果你没有使用这个选项, 系统将使用原生的 GD 字体。
wm_font_size	16	None	字体大小。说明: 如果你没有使用上面的 TTF 字体选项, 那么这个数值必须是 1-5 之间的一个数字, 如果使用了 TTF, 你可以使用任意有效的字体大小。
wm_font_color	fff	None	字体颜色, 以十六进制给出。注意, 你必须给出完整的 6 位数的十六进制值 (如: 993300), 而不能使用 3 位数的简化值 (如: fff)。
wm_shadow_color	None	None	阴影的颜色, 以十六进制给出。如果此项为空, 将不使用阴影。注意, 你必须给出完整的 6 位数的十六进制值 (如: 993300), 而不能使用 3 位数的简化值 (如: fff)。
wm_shadow_distance	3	None	阴影与文字之间的距离 (以像素为单位)。

Overlay 参数 下表列举的参数只适用于 overlay 水印处理方式。

参数	默认值	选项	描述
wm_overlay_path	None	None	你想要用作水印的图片在你服务器上的路径。只在你使用了 overlay 方法时需要。
wm_opacity	70	1 - 100	图像不透明度。你可以指定你的水印图片的不透明度。这将使水印模糊化, 从而不会掩盖住底层原始图片, 通常设置为 50。
wm_x_transp	None	A number	如果你的水印图片是一个 PNG 或 GIF 图片, 你可以指定一种颜色用来使图片变得“透明”。这项设置 (以及下面那项) 将允许你指定这种颜色。它的原理是, 通过指定 “X” 和 “Y” 坐标值 (从左上方开始测量) 来确定图片上对应位置的某个像素, 这个像素所代表的颜色就是你要设置为透明的颜色。
wm_y_transp	None	A number	与前一个选项一起, 允许你指定某个像素的坐标值, 这个像素所代表的颜色就是你要设置为透明的颜色。

类参考

class CI_Image_lib

```
initialize([ $props = array() ])
```

参数

- **\$props** (*array*) – Image processing preferences

返回 TRUE on success, FALSE in case of invalid settings

返回类型 bool

初始化图像处理类。

resize()

返回 TRUE on success, FALSE on failure

返回类型 bool

该函数让你能调整原始图像的大小, 创建一个副本 (调整或未调整过的), 或者创建一个缩略图。

创建一个副本和创建一个缩略图之间没有实际上的区别, 除了缩略图的文件名会有一个自定义的后缀 (如: mypic_thumb.jpg)。

所有列在上面 [参数](#) 表中的参数对这个函数都可用, 除了这三个: *rotation_angle*、*x_axis* 和 *y_axis*。

创建一个缩略图

resize 函数能用来创建缩略图 (并保留原图), 只要你把这个参数设为 TRUE

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

这一个参数决定是否创建一个缩略图。

创建一个副本

resize 函数能创建一个图像的副本 (并保留原图), 只要你通过以下参数设置一个新的路径或者文件名:

```
$config['new_image'] = '/path/to/new_image.jpg';
```

注意以下规则:

- 如果只指定新图像的名字, 那么它会被放在原图像所在的文件夹下。
- 如果只指定路径, 新图像会被放在指定的文件夹下, 并且名字和原图像相同。
- 如果同时定义了路径和新图像的名字, 那么新图像会以指定的名字放在指定的文件夹下。

调整原图像的大小

如果上述两个参数 (create_thumb 和 new_image) 均未被指定, resize 函数的处理将直接作用于原图像。

crop()

返回 TRUE on success, FALSE on failure

返回类型 bool

crop 函数的用法与 resize 函数十分接近, 除了它需要你设置用于裁剪的 X 和 Y 值 (单位是像素), 如下:

```
$config['x_axis'] = 100;  
$config['y_axis'] = 40;
```

前面那张 [参数](#) 表中所列的所有参数都可以用于这个函数, 除了这些: *rotation_angle*、*width*、*height*、*create_thumb*、*new_image*。

这是一个如何裁剪一张图片的示例:

```
$config['image_library'] = 'imagemagick';
$config['library_path'] = '/usr/X11R6/bin/';
$config['source_image'] = '/path/to/image/mypic.jpg';
$config['x_axis'] = 100;
$config['y_axis'] = 60;

$this->image_lib->initialize($config);

if ( ! $this->image_lib->crop() )
{
    echo $this->image_lib->display_errors();
}
```

注解: 如果没有一个可视化的界面, 是很难裁剪一张图片的。因此, 除非你打算制作这么一个界面, 否则这个函数并不是很有用。事实上我们在自己开发的 CMS 系统 ExpressionEngine 的相册模块中添加的一个基于 JavaScript 的用户界面来选择裁剪的区域。

rotate()

返回 TRUE on success, FALSE on failure

返回类型 bool

rotate 函数需要通过参数设置旋转的角度:

```
$config['rotation_angle'] = '90';
```

以下是 5 个可选项:

- 1.90 - 逆时针旋转 90 度。
- 2.180 - 逆时针旋转 180 度。
- 3.270 - 逆时针旋转 270 度。
- 4.hor - 水平翻转。
- 5.vrt - 垂直翻转。

下面是旋转图片的一个例子:

```
$config['image_library'] = 'netpbm';
$config['library_path'] = '/usr/bin/';
$config['source_image'] = '/path/to/image/mypic.jpg';
$config['rotation_angle'] = 'hor';

$this->image_lib->initialize($config);

if ( ! $this->image_lib->rotate() )
```

```
{
    echo $this->image_lib->display_errors();
}
```

watermark()

返回 TRUE on success, FALSE on failure

返回类型 bool

在图像上添加一个水印，更多信息请参考[给图像添加水印](#)。

clear()

返回类型 void

clear 函数重置所有之前用于处理图片的值。当你用循环来处理一批图片时，你可能会想使用它。

```
$this->image_lib->clear();
```

display_errors() (*\$open* = '<p>', *\$close* = '</p>')

参数

- **\$open** (*string*) – Error message opening tag
- **\$close** (*string*) – Error message closing tag

返回 Error messages

返回类型 string

返回所有检测到的错误信息。

```
echo $this->image_lib->diplay_errors();
```

8.1.13 输入类

输入类有两个用途：

1. 为了安全性，对输入数据进行预处理
2. 提供了一些辅助方法来获取输入数据并处理

注解： 该类由系统自动加载，你无需手工加载

- 对输入进行过滤
 - 安全性过滤
 - XSS 过滤
- 访问表单数据
 - 使用 POST、GET、COOKIE 和 SERVER 数据
 - 使用 `php://input` 流
- 类参考

对输入进行过滤

安全性过滤

当访问控制器时，安全过滤方法会自动被调用，它做了以下几件事情：

- 如果 `$config['allow_get_array']` 设置为 FALSE（默认是 TRUE），销毁全局的 GET 数组。
- 当开启 `register_globals` 时，销毁所有的全局变量。
- 过滤 GET/POST/COOKIE 数据的键值，只允许出现字母和数字（和其他一些）字符。
- 提供了 XSS（跨站脚本攻击）过滤，可全局启用，或按需启用。
- 将换行符统一为 `PHP_EOL`（基于 UNIX 的系统下为 `\n`，Windows 系统下为 `\r\n`），这个是可配置的。

XSS 过滤

输入类可以自动的对输入数据进行过滤，来阻止跨站脚本攻击。如果你希望在每次遇到 POST 或 COOKIE 数据时自动运行过滤，你可以在 `application/config/config.php` 配置文件中设置如下参数：

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

关于 XSS 过滤的信息，请参考[安全类](#)文档。

重要： 参数 `'global_xss_filtering'` 已经废弃，保留它只是为了实现向前兼容。XSS 过滤应该在 * 输出 * 的时候进行，而不是 * 输入 * 的时候！

访问表单数据

使用 POST、GET、COOKIE 和 SERVER 数据

CodeIgniter 提供了几个辅助方法来从 POST、GET、COOKIE 和 SERVER 数组中获取数据。使用这些方法来获取数据而不是直接访问数组 (`$POST['something']`) 的最大的好处是，这些方法会检查获取的数据是否存在，如果不存在则返回 NULL。

这使用起来将很方便, 你不再需要去检查数据是否存在。换句话说, 通常你需要像下面这样做:

```
$something = isset($_POST['something']) ? $_POST['something'] : NULL;
```

使用 CodeIgniter 的方法, 你可以简单的写成:

```
$something = $this->input->post('something');
```

主要有下面几个方法:

- `$this->input->post()`
- `$this->input->get()`
- `$this->input->cookie()`
- `$this->input->server()`

使用 **php://input** 流

如果你需要使用 PUT、DELETE、PATCH 或其他请求方法, 你只能通过一个特殊的输入流来访问, 这个流只能被读一次, 这和从诸如 `$_POST` 数组中读取数据相比起来要复杂一点, 因为 POST 数组可以被访问多次来获取多个变量, 而不用担心它会消失。

CodeIgniter 为你解决了这个问题, 你只需要使用下面的 `$raw_input_stream` 属性即可, 就可以在任何时候读取 **php://input** 流中的数据:

```
$this->input->raw_input_stream;
```

另外, 如果输入流的格式和 `$_POST` 数组一样, 你也可以通过 `input_stream()` 方法来访问它的值:

```
$this->input->input_stream('key');
```

和其他的 `get()` 和 `post()` 方法类似, 如果请求的数据不存在, 则返回 NULL。你也可以将第二个参数设置为 TRUE, 来让数据经过 `xss_clean()` 的检查:

```
$this->input->input_stream('key', TRUE); // XSS Clean
$this->input->input_stream('key', FALSE); // No XSS filter
```

注解: 你可以使用 `method()` 方法来获取你读取的是什么数据, PUT、DELETE 还是 PATCH。

类参考

class CI_Input

\$raw_input_stream

返回只读的 php://input 流数据。

该属性可以被多次读取。

```
post([$index = NULL[, $xss_clean = NULL]])
```

参数

- **\$index** (*mixed*) – POST parameter name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 \$_POST if no parameters supplied, otherwise the POST value if found or NULL if not

返回类型 mixed

第一个参数为你想要获取的 POST 数据名:

```
$this->input->post('some_data');
```

如果获取的数据不存在, 该方法返回 NULL 。

第二个参数可选, 用于决定是否使用 XSS 过滤器对数据进行过滤。要使用过滤器, 可以将第二个参数设置为 TRUE , 或者将 \$config['global_xss_filtering'] 参数设置为 TRUE 。

```
$this->input->post('some_data', TRUE);
```

如果不带任何参数该方法将返回 POST 中的所有元素。

如果希望返回 POST 所有元素并将它们通过 XSS 过滤器进行过滤, 可以将第一个参数设为 NULL , 第二个参数设为 TRUE

```
$this->input->post(NULL, TRUE); // returns all POST items with XSS filter
$this->input->post(NULL, FALSE); // returns all POST items without XSS filter
```

如果要返回 POST 中的多个元素, 将所有需要的键值作为数组传给它:

```
$this->input->post(array('field1', 'field2'));
```

和上面一样, 如果希望数据通过 XSS 过滤器进行过滤, 将第二个参数设置为 TRUE:

```
$this->input->post(array('field1', 'field2'), TRUE);
```

```
get([$index = NULL[, $xss_clean = NULL]])
```

参数

- **\$index** (*mixed*) – GET parameter name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 \$_GET if no parameters supplied, otherwise the GET value if found or NULL if not

返回类型 mixed

该函数和 `post()` 一样, 只是它用于获取 GET 数据。

```
$this->input->get('some_data', TRUE);
```

如果不带任何参数该方法将返回 GET 中的所有元素。

如果希望返回 GET 所有元素并将它们通过 XSS 过滤器进行过滤, 可以将第一个参数设为 NULL, 第二个参数设为 TRUE

```
$this->input->get(NULL, TRUE); // returns all GET items with XSS filter
$this->input->get(NULL, FALSE); // returns all GET items without XSS filtering
```

如果要返回 GET 中的多个元素, 将所有需要的键值作为数组传给它:

```
$this->input->get(array('field1', 'field2'));
```

和上面一样, 如果希望数据通过 XSS 过滤器进行过滤, 将第二个参数设置为 TRUE:

```
$this->input->get(array('field1', 'field2'), TRUE);
```

```
post_get($index[, $xss_clean = NULL])
```

参数

- **\$index** (*string*) – POST/GET parameter name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 POST/GET value if found, NULL if not

返回类型 mixed

该方法和 `post()` 和 `get()` 方法类似, 它会同时查找 POST 和 GET 两个数组来获取数据, 先查找 POST, 再查找 GET:

```
$this->input->post_get('some_data', TRUE);
```

```
get_post($index[, $xss_clean = NULL])
```

参数

- **\$index** (*string*) – GET/POST parameter name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 GET/POST value if found, NULL if not

返回类型 mixed

该方法和 `post_get()` 方法一样, 只是它先查找 GET 数据:

```
$this->input->get_post('some_data', TRUE);
```

注解: 这个方法在之前的版本中和 `post_get()` 方法是完全一样的, 在 CodeIgniter 3.0 中有所修改。

```
cookie([$index = NULL[, $xss_clean = NULL]])
```

参数

- **\$index** (*mixed*) – COOKIE name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 \$_COOKIE if no parameters supplied, otherwise the COOKIE value if found or NULL if not

返回类型 mixed

该方法和 `post()` 和 `get()` 方法一样, 只是它用于获取 COOKIE 数据:

```
$this->input->cookie('some_cookie');
$this->input->cookie('some_cookie', TRUE); // with XSS filter
```

如果要返回 COOKIE 中的多个元素, 将所有需要的键值作为数组传给它:

```
$this->input->cookie(array('some_cookie', 'some_cookie2'));
```

注解: 和 [Cookie 辅助函数](#) 中的 `get_cookie()` 函数不同的是, 这个方法不会根据 `$config['cookie_prefix']` 来添加前缀。

```
server([$index[, $xss_clean = NULL]])
```

参数

- **\$index** (*mixed*) – Value name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 \$_SERVER item value if found, NULL if not

返回类型 mixed

该方法和 `post()`、`get()` 和 `cookie()` 方法一样, 只是它用于获取 SERVER 数据:

```
$this->input->server('some_data');
```

如果要返回 SERVER 中的多个元素, 将所有需要的键值作为数组传给它:

```
$this->input->server(array('SERVER_PROTOCOL', 'REQUEST_URI'));
```

```
input_stream([$index = NULL[, $xss_clean = NULL]])
```

参数

- **\$index** (*mixed*) – Key name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 Input stream array if no parameters supplied, otherwise the specified value if found or NULL if not

返回类型 `mixed`

该方法和 `get()`、`post()` 和 `cookie()` 方法一样, 只是它用于获取 `php://input` 流数据。

```
set_cookie($name = '[', $value = '[', $expire = '[', $domain = '[', $path =  
    '/', $prefix = '[', $secure = FALSE, $httponly = FALSE]]]]]  
    ])
```

参数

- **\$name** (*mixed*) – Cookie name or an array of parameters
- **\$value** (*string*) – Cookie value
- **\$expire** (*int*) – Cookie expiration time in seconds
- **\$domain** (*string*) – Cookie domain
- **\$path** (*string*) – Cookie path
- **\$prefix** (*string*) – Cookie name prefix
- **\$secure** (*bool*) – Whether to only transfer the cookie through HTTPS
- **\$httponly** (*bool*) – Whether to only make the cookie accessible for HTTP requests (no JavaScript)

返回类型 `void`

设置 COOKIE 的值, 有两种方法来设置 COOKIE 值: 数组方式和参数方式。

数组方式

使用这种方式, 可以将第一个参数设置为一个关联数组:

```
$cookie = array(  
    'name'    => 'The Cookie Name',  
    'value'   => 'The Value',  
    'expire'  => '86500',  
    'domain'  => '.some-domain.com',  
    'path'    => '/',  
    'prefix'  => 'myprefix_',  
    'secure'  => TRUE  
);
```

```
$this->input->set_cookie($cookie);
```

注意

只有 `name` 和 `value` 两项是必须的, 要删除 COOKIE 的话, 将 `expire` 设置为空。

COOKIE 的过期时间是 **秒**, 将它加到当前时间上就是 COOKIE 的过期时间。记住不要把它设置成时间了, 只要设置成距离当前时间的秒数即可, 那

么在这段时间内，COOKIE 都将保持有效。如果将过期时间设置为 0，那么 COOKIE 只在浏览器打开的期间是有效的，关闭后就失效了。

如果需要设置一个全站范围内的 COOKIE，而不关心用户是如何访问你的站点的，可以将 **domain** 参数设置为你的 URL 前面以句点开头，如：
.your-domain.com

path 参数通常不用设，上面的例子设置为根路径。

prefix 只在你想避免和其他相同名称的 COOKIE 冲突时才需要使用。

secure 参数只有当你需要使用安全的 COOKIE 时使用。

参数方式

如果你喜欢，你也可以使用下面的方式来设置 COOKIE:

```
$this->input->set_cookie($name, $value, $expire, $domain, $path, $prefix, $secure)
```

ip_address()

返回 Visitor's IP address or '0.0.0.0' if not valid

返回类型 string

返回当前用户的 IP 地址，如果 IP 地址无效，则返回 '0.0.0.0':

```
echo $this->input->ip_address();
```

重要: 该方法会根据 `$config['proxy_ips']` 配置，来返回 HTTP_X_FORWARDED_FOR、HTTP_CLIENT_IP、HTTP_X_CLIENT_IP 或 HTTP_X_CLUSTER_CLIENT_IP。

valid_ip(\$ip[, \$which = ''])

参数

- **\$ip** (*string*) – IP address
- **\$which** (*string*) – IP protocol ('ipv4' or 'ipv6')

返回 TRUE if the address is valid, FALSE if not

返回类型 bool

判断一个 IP 地址是否有效，返回 TRUE/FALSE。

注解: 上面的 `$this->input->ip_address()` 方法会自动验证 IP 地址的有效性。

```
if ( ! $this->input->valid_ip($ip))
{
    echo 'Not Valid';
}
else
{

```

```
        echo 'Valid';
    }
```

第二个参数可选，可以是字符串 'ipv4' 或 'ipv6' 用于指定 IP 的格式，默认两种格式都会检查。

user_agent(*[\$xss_clean = NULL]*)

返回 User agent string or NULL if not set

参数

- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回类型 mixed

返回当前用户的用户代理字符串（Web 浏览器），如果不可用则返回 FALSE。

```
echo $this->input->user_agent();
```

关于用户代理的相关方法请参考[用户代理类](#)。

request_headers(*[\$xss_clean = FALSE]*)

参数

- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 An array of HTTP request headers

返回类型 array

返回 HTTP 请求头数组。当在非 Apache 环境下运行时，`apache_request_headers()` 函数不可用，这个方法将很有用。

```
$headers = $this->input->request_headers();
```

get_request_header(*\$index*, *[\$xss_clean = FALSE]*)

参数

- **\$index** (*string*) – HTTP request header name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering

返回 An HTTP request header or NULL if not found

返回类型 string

返回某个指定的 HTTP 请求头，如果不存在，则返回 NULL。

```
$this->input->get_request_header('some-header', TRUE);
```

is_ajax_request()

返回 TRUE if it is an Ajax request, FALSE if not

返回类型 bool

检查服务器头中是否含有 HTTP_X_REQUESTED_WITH , 如果有返回 TRUE , 否则返回 FALSE 。

is_cli_request()

返回 TRUE if it is a CLI request, FALSE if not

返回类型 bool

检查程序是否从命令行界面运行。

注解: 该方法检查当前正在使用的 PHP SAPI 名称, 同时检查是否定义了 STDIN 常量, 来判断当前 PHP 是否从命令行运行。

```
$this->input->is_cli_request()
```

注解: 该方法已经被废弃, 现在只是 is_cli() 函数的一个别名而已。

method(*[\$upper = FALSE]*)

参数

- **\$upper** (*bool*) – Whether to return the request method name in upper or lower case

返回 HTTP request method

返回类型 string

返回 \$_SERVER['REQUEST_METHOD'] 的值, 它有一个参数用于设置返回大写还是小写。

```
echo $this->input->method(TRUE); // Outputs: POST
echo $this->input->method(FALSE); // Outputs: post
echo $this->input->method(); // Outputs: post
```

8.1.14 Javascript 类

CodeIgniter 提供一个类库和一些共用的方法来处理 Javascript 。要注意的是, CodeIgniter 并不是只能用于 jQuery , 其他脚本库也可以。jQuery 仅仅是作为一个方便的工具, 如果你选择使用它的话。

重要: 这个类库已经废弃, 不要使用它。它将永远处于“实验”版本, 而且现在也已经不提供支持了。保留它只是为了向前兼容。

- 使用 Javascript 类
 - 初始化类
 - 初始化配置
 - * 在视图文件中设置变量
 - * 设置库路径
 - jQuery 类
 - jQuery 事件
 - 特效
 - * hide() / show()
 - * toggle()
 - * animate()
 - * toggleClass()
 - * fadeIn() / fadeOut()
 - * slideUp() / slideDown() / slideToggle()
 - 插件
 - * corner()
 - * tablesorter()
 - * modal()
 - * calendar()

使用 Javascript 类

初始化类

要初始化 Javascript 类，你可以在控制器的构造函数中使用 `$this->load->library()` 函数。目前，唯一可用的库是 jQuery，可以使用下面的方法加载：

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

Javascript 类也可以接受参数：

- js_library_driver (string) *default: 'jquery'*
- autoload (bool) *default: TRUE*

你可以通过一个关联数组覆盖默认的参数：

```
$this->load->library(
    'javascript',
    array(
        'js_library_driver' => 'scripto',
        'autoload' => FALSE
    )
);
```

再次说明，目前只有 jQuery 是可用的，如果你不想让 jQuery 脚本文件自动的包含在 script 标签中，你可以设置 autoload 参数为 FALSE。这在当你在 CodeIgniter 之外加载它时，或者 script 标签已经有了的时候很有用。

一旦加载完成, jQuery 类对象就可以通过下面的方式使用:

```
$this->javascript
```

初始化配置

在视图文件中设置变量 作为一个 Javascript 库, 源文件必须能被应用程序访问到。

由于 Javascript 是一种客户端语言, 库必须能写入内容到最终的输出中去, 这通常就是视图。你需要在输出的 `<head>` 中包含下面的变量。

```
<?php echo $library_src;?>
<?php echo $script_head;?>
```

`$library_src` 是要载入的库文件的路径, 以及之后所有插件脚本的路径; `$script_head` 是需要显示的具体的一些事件、函数和其他的命令。

设置库路径 在 Javascript 类库中有一些配置项, 它们可以在 `application/config.php` 文件中设置, 也可以在它们自己的配置文件 `config/javascript.php` 中设置, 还可以通过在控制器中使用 `set_item()` 方法来设置。

例如, 有一个“加载中”的图片, 或者进度条指示, 如果没有它的话, 当调用 Ajax 请求时, 将会显示“加载中”这样的文本。

```
$config['javascript_location'] = 'http://localhost/codeigniter/themes/js/jquery/';
$config['javascript_ajax_img'] = 'images/ajax-loader.gif';
```

如果你把文件留在与图片下载路径相同的目录里, 那么你不需要设置这个配置项。

jQuery 类

要在你的控制器构造函数中手工初始化 jQuery 类, 使用 `$this->load->library()` 方法:

```
$this->load->library('javascript/jquery');
```

你可以提供一个可选的参数来决定加载该库时是否将其自动包含到 script 标签中。默认情况下会包含, 如果不需要, 可以像下面这样来加载:

```
$this->load->library('javascript/jquery', FALSE);
```

加载完成后, jQuery 类对象可以使用下面的代码来访问:

```
$this->jquery
```

jQuery 事件

使用下面的语法来设置事件。

```
$this->jquery->event('element_path', code_to_run());
```

在上面的例子中：

- “event” 可以是 blur、change、click、dblclick、error、focus、hover、keydown、keyup、load、mousedown、mouseup、mouseover、mouseout、resize、scroll 或者 unload 中的任何一个事件。
- “element_path” 可以是任何的 jQuery 选择器。使用 jQuery 独特的选择器语法，通常是一个元素 ID 或 CSS 选择器。例如，“#notice_area” 会影响到 `<div id="notice_area">`，`#content a.notice` 会影响到 ID 为 “content” 的元素下的所有 class 为 “notice” 的链接。
- “code_to_run()” 为你自己写的脚本，或者是一个 jQuery 动作，例如下面所介绍的特效。

特效

jQuery 库支持很多强大的 特效，在使用特效之前，必须使用下面的方法加载：

```
$this->jquery->effect([optional path] plugin name); // for example $this->jquery->effect('b
```

hide() / show() 这两个函数会影响你的页面上元素的可见性，hide() 函数用于将元素隐藏，show() 则相反。

```
$this->jquery->hide(target, optional speed, optional extra information);  
$this->jquery->show(target, optional speed, optional extra information);
```

- “target” 是任何有效的 jQuery 选择器。
- “speed” 可选，可以设置为 slow、normal、fast 或你自己设置的毫秒数。
- “extra information” 可选，可以包含一个回调，或者其他的附加信息。

toggle() toggle() 用于将元素的可见性改成和当前的相反，将可见的元素隐藏，将隐藏的元素可见。

```
$this->jquery->toggle(target);
```

- “target” 是任何有效的 jQuery 选择器。

animate()

```
$this->jquery->animate(target, parameters, optional speed, optional extra information);
```

- “target” 是任何有效的 jQuery 选择器。

- “parameters” 通常是我想改变元素的一些 CSS 属性。
- “speed” 可选，可以设置为 slow、normal、fast 或你自己设置的毫秒数。
- “extra information” 可选，可以包含一个回调，或者其他的附加信息。

更完整的说明，参见 <http://api.jquery.com/animate/>

下面是个在 ID 为 “note” 的一个 div 上使用 animate() 的例子，它使用了 jQuery 库的 click 事件，通过 click 事件触发。

```
$params = array(
    'height' => 80,
    'width' => '50%',
    'marginLeft' => 125
);
$this->jquery->click('#trigger', $this->jquery->animate('#note', $params, 'normal'));
```

toggleClass() 该函数用于往目标元素添加或移除一个 CSS 类。

```
$this->jquery->toggleClass(target, class)
```

- “target” 是任何有效的 jQuery 选择器。
- “class” 是任何 CSS 类名，注意这个类必须是在某个已加载的 CSS 文件中定义的。

fadeIn() / fadeOut() 这两个特效会使某个元素渐变的隐藏和显示。

```
$this->jquery->fadeIn(target, optional speed, optional extra information);
$this->jquery->fadeOut(target, optional speed, optional extra information);
```

- “target” 是任何有效的 jQuery 选择器。
- “speed” 可选，可以设置为 slow、normal、fast 或你自己设置的毫秒数。
- “extra information” 可选，可以包含一个回调，或者其他的附加信息。

slideUp() / slideDown() / slideToggle() 这些特效可以让元素滑动。

```
$this->jquery->slideUp(target, optional speed, optional extra information);
$this->jquery->slideDown(target, optional speed, optional extra information);
$this->jquery->slideToggle(target, optional speed, optional extra information);
```

- “target” 是任何有效的 jQuery 选择器。
- “speed” 可选，可以设置为 slow、normal、fast 或你自己设置的毫秒数。
- “extra information” 可选，可以设置为 slow、normal、fast 或你自己设置的毫秒数。

插件

使用这个库时还有几个 jQuery 插件可用。

corner() 用于在页面的某个元素四周添加不同样式的边角。更多详细信息, 参考 <http://malsup.com/jquery/corner/>

```
$this->jquery->corner(target, corner_style);
```

- “target” 是任何有效的 jQuery 选择器。
- “corner_style” 可选, 可以设置为任何有效的样式, 例如: round、sharp、bevel、bite、dog 等。如果只想设置某个边角的样式, 可以在样式后添加一个空格, 然后使用 “tl” (左上), “tr” (右上), “bl” (左下), 和 “br” (右下)。

```
$this->jquery->corner("#note", "cool tl br");
```

tablesorter() 待添加

modal() 待添加

calendar() 待添加

8.1.15 语言类

语言类提供了一些方法用于获取语言文件和不同语言的文本来实现国际化。

在你的 CodeIgniter 的 **system** 目录, 有一个 **language** 子目录, 它包含了一系列 **英文** 的语言文件。在 **system/language/english/** 这个目录下的这些文件定义了 CodeIgniter 框架的各个部分使用到的一些常规消息, 错误消息, 以及其他一些通用的单词或短语。

如果需要的话, 你可以创建属于你自己的语言文件, 用于提供应用程序的错误消息和其他消息, 或者将核心部分的消息翻译为其他的语言。翻译的消息或你另加的消息应该放在 **application/language/** 目录下, 每种不同的语言都有相应的一个子目录 (例如, ‘french’ 或者 ‘german’)。

CodeIgniter 框架自带了一套 “英语” 语言文件, 另外可以在 [CodeIgniter 3 翻译仓库](#) 找到其他不同的语言, 每个语言都有一个独立的目录。

当 CodeIgniter 加载语言文件时, 它会先加载 **system/language/** 目录下的, 然后再加载你的 **application/language/** 目录下的来覆盖它。

注解: 每个语言都有它自己的目录, 例如, 英语语言文件位于: **system/language/english**

- 处理多语言
 - 语言文件的例子
 - 切换语言
- 国际化
- 使用语言类
 - 创建语言文件
 - 加载语言文件
 - 读取语言文本
 - * 使用语言行作为表单的标签
 - 自动加载语言文件
- 类参考

处理多语言

如果你想让你的应用程序支持多语言，你就需要在 **application/language/** 目录下提供不同语言的文件，然后在 **application/config/config.php** 配置文件中指定默认语言。

application/language/english/ 目录可以包含你的应用程序需要的额外语言文件，例如错误消息。

每个语言对应的目录中都应该包含从翻译仓库中获取到的核心文件，或者你自己翻译它们，你也可以添加你的程序需要的其他文件。

你应该将你正在使用的语言保存到一个会话变量中。

语言文件的例子

```
system/
  language/
    english/
      ...
      email_lang.php
      form_validation_lang.php
      ...

application/
  language/
    english/
      error_messages_lang.php
    french/
      ...
      email_lang.php
      error_messages_lang.php
      form_validation_lang.php
      ...
```

切换语言

```
$idiom = $this->session->get_userdata('language');
$this->lang->load('error_messages', $idiom);
$oops = $this->lang->line('message_key');
```

国际化

CodeIgniter 的语言类给你的应用程序提供了一种简单轻便的方式来实现多语言，它并不是通常我们所说的 **国际化与本地化** 的完整实现。

我们可以给每一种语言一个别名，一个更通用的名字，而不是使用诸如 “en”、“en-US”、“en-CA-x-ca” 这种国际标准的缩写名字。

注解： 当然，你完全可以在你的程序中使用国际标准的缩写名字。

使用语言类

创建语言文件

语言文件的命名必须以 **lang.php** 结尾，例如，你想创建一个包含错误消息的文件，你可以把它命名为：error_lang.php。

在此文件中，你可以在每行把一个字符串赋值给名为 \$lang 的数组，例如：

```
$lang['language_key'] = 'The actual message to be shown';
```

注解： 在每个文件中使用一个通用的前缀来避免和其他文件中的相似名称冲突是个好方法。例如，如果你在创建错误消息你可以使用 error_ 前缀。

```
$lang['error_email_missing'] = 'You must submit an email address';
$lang['error_url_missing'] = 'You must submit a URL';
$lang['error_username_missing'] = 'You must submit a username';
```

加载语言文件

在使用语言文件之前，你必须先加载它。可以使用下面的代码：

```
$this->lang->load('filename', 'language');
```

其中 filename 是你要加载的语言文件名（不带扩展名），language 是要加载哪种语言（比如，英语）。如果没有第二个参数，将会使用 **application/config/config.php** 中设置的默认语言。

你也可以通过传一个语言文件的数组给第一个参数来同时加载多个语言文件。

```
$this->lang->load(array('filename1', 'filename2'));
```

注解: `language` 参数只能包含字母。

读取语言文本

当你的语言文件已经加载, 你可以通过下面的方法来访问任何一行语言文本:

```
$this->lang->line('language_key');
```

其中, `language_key` 参数是你想显示的文本行所对应的数组的键名。

万一你不确定你想读取的那行文本是否存在, 你还可以将第二个参数设置为 `FALSE` 禁用错误日志:

```
$this->lang->line('misc_key', FALSE);
```

注解: 该方法只是简单的返回文本行, 而不是显示出它。

使用语言行作为表单的标签 这一特性已经从语言类中废弃, 并移到了 [语言辅助函数](#) 的 `lang()` 函数。

自动加载语言文件

如果你发现你需要在整个应用程序中使用某个语言文件, 你可以让 CodeIgniter 在系统初始化的时候 **自动加载** 该语言文件。可以打开 `application/config/autoload.php` 文件, 把语言放在 `autoload` 数组中。

类参考

class CI_Lang

```
load($langfile[, $idiom = ''], $return = FALSE[, $add_suffix = TRUE[,  
    $alt_path = '']])
```

参数

- **\$langfile** (*mixed*) – Language file to load or array with multiple files
- **\$idiom** (*string*) – Language name (i.e. 'english')
- **\$return** (*bool*) – Whether to return the loaded array of translations

- **\$add_suffix** (*bool*) – Whether to add the ‘_lang’ suffix to the language file name
- **\$alt_path** (*string*) – An alternative path to look in for the language file

返回 Array of language lines if \$return is set to TRUE, otherwise void

返回类型 mixed

加载一个语言文件。

```
line($line[, $log_errors = TRUE])
```

参数

- **\$line** (*string*) – Language line key name
- **\$log_errors** (*bool*) – Whether to log an error if the line isn’t found

返回 Language line string or FALSE on failure

返回类型 string

从一个已加载的语言文件中，通过行名获取一行该语言的文本。

8.1.16 加载器类

加载器，顾名思义，是用于加载元素的，加载的元素可以是库（类），视图文件，驱动器，辅助函数，模型或其他你自己的文件。

注解： 该类由系统自动加载，你无需手工加载。

- 应用程序”包”
 - 包的视图文件
 - 类参考

应用程序”包”

应用程序包（Package）可以很便捷的将你的应用部署在一个独立的目录中，以实现自己整套的类库，模型，辅助函数，配置，文件和语言包。建议将这些应用程序包放置在 application/third_party 目录下。下面是一个简单应用程序包的目录结构。

下面是一个名为“Foo Bar”的应用程序包目录的例子。

```
/application/third_party/foo_bar
```

```
config/  
helpers/  
language/
```

```
libraries/
models/
```

无论应用程序包是为了实现什么样的目的，它都包含了属于自己的配置文件、辅助函数、语言包、类库和模型。如果要在你的控制器里使用这些资源，你首先需要告知加载器 (Loader) 从应用程序包加载资源，使用 `add_package_path()` 方法来添加包的路径。

包的视图文件

默认情况下，当调用 `add_package_path()` 方法时，包的视图文件路径就设置好了。视图文件的路径是通过一个循环来查找的，一旦找到第一个匹配的即加载该视图。

在这种情况下，它可能在包内产生视图命名冲突，并可能导致加载错误的包。为了确保不会发生此类问题，在调用 `add_package_path()` 方法时，可以将可选的第二个参数设置为 `FALSE`。

```
$this->load->add_package_path(APPPATH.'my_app', FALSE);
$this->load->view('my_app_index'); // Loads
$this->load->view('welcome_message'); // Will not load the default welcome_message b/c the

// Reset things
$this->load->remove_package_path(APPPATH.'my_app');

// Again without the second parameter:
$this->load->add_package_path(APPPATH.'my_app');
$this->load->view('my_app_index'); // Loads
$this->load->view('welcome_message'); // Loads
```

类参考

class CI_Loader

```
library($library[, $params = NULL[, $object_name = NULL]])
```

参数

- **\$library** (*mixed*) – Library name as a string or an array with multiple libraries
- **\$params** (*array*) – Optional array of parameters to pass to the loaded library's constructor
- **\$object_name** (*string*) – Optional object name to assign the library to

返回 CI_Loader instance (method chaining)

返回类型 CI_Loader

该方法用于加载核心类。

注解: 我们有时候说“类”，有时候说“库”，这两个词不做区分。

例如，如果你想使用 CodeIgniter 发送邮件，第一步就是在控制器中加载 email 类：

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

加载完之后，email 类就可以使用 `$this->email` 来访问使用了。

类库文件可以被保存到主 `libraries` 目录的子目录下面，或者保存到个人的 `application/libraries` 目录下。要载入子目录下的文件，只需将路径包含进来就可以了，注意这里说的路径是指相对于 `libraries` 目录的路径。例如，当你有一个文件保存在下面这个位置：

```
libraries/flavors/Chocolate.php
```

你应该使用下面的方式来载入它：

```
$this->load->library('flavors/chocolate');
```

你可以随心所欲地将文件保存到多层的子目录下。

另外，你可以同时加载多个类，只需给 `library` 方法传入一个包含所有要载入的类名的数组即可：

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

设置选项

第二个参数是可选的，用于选择性地传递配置参数。一般来说，你可以将参数以数组的形式传递过去：

```
$config = array (
    'mailtype' => 'html',
    'charset'  => 'utf-8',
    'priority' => '1'
);
```

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

配置参数通常也可以保存在一个配置文件中，在每个类库自己的页面中有详细的说明，所以在使用类库之前，请认真阅读说明。

请注意，当第一个参数使用数组来同时载入多个类时，每个类将获得相同的参数信息。

给类库分配不同的对象名

第三个参数也是可选的，如果为空，类库通常就会被赋值给一个与类库同名的对象。例如，如果类库名为 `Calendar`，它将会被赋值给一个名为 `$this->calendar` 的变量。

如果你希望使用你的自定义名称，你可以通过第三个参数把它传递过去：

```
$this->load->library('calendar', NULL, 'my_calendar');

// Calendar class is now accessed using:
$this->my_calendar
```

请注意, 当第一个参数使用数组来同时载入多个类时, 第三个参数将不起作用。

```
driver($library[, $params = NULL[, $object_name]])
```

参数

- **\$library** (*mixed*) – Library name as a string or an array with multiple libraries
- **\$params** (*array*) – Optional array of parameters to pass to the loaded library's constructor
- **\$object_name** (*string*) – Optional object name to assign the library to

返回 CI.Loader instance (method chaining)

返回类型 CI.Loader

该方法用于加载驱动器类, 和 `library()` 方法非常相似。

例如, 如果你想在 CodeIgniter 中使用会话, 第一步就是在控制器中加载 session 驱动器:

```
$this->load->driver('session');
```

加载完之后, session 驱动器就可以使用 `$this->session` 来访问使用了。

驱动器文件可以被保存到主 `libraries` 目录的子目录下面, 或者保存到个人的 `application/libraries` 目录下。子目录的名称必须和驱动器父类的名称一致, 你可以阅读[驱动器](#)了解详细信息。

另外, 你可以同时加载多个驱动器, 只需给 `driver` 方法传入一个包含所有要载入的驱动器名的数组即可::

```
$this->load->driver(array('session', 'cache'));
```

设置选项

第二个参数是可选的, 用于选择性地传递配置参数。一般来说, 你可以将参数以数组的形式传递过去:

```
$config = array(
    'sess_driver' => 'cookie',
    'sess_encrypt_cookie' => true,
    'encryption_key' => 'mysecretkey'
);

$this->load->driver('session', $config);
```

配置参数通常也可以保存在一个配置文件中, 在每个类库自己的页面中有详细的说明, 所以在使用类库之前, 请认真阅读说明。

给类库分配不同的对象名

第三个参数也是可选的, 如果为空, 驱动器通常就会被赋值给一个与它同名的对象。例如, 如果驱动器名为 `Session`, 它将会被赋值给一个名为 `$this->session` 的变量。

如果你希望使用你的自定义名称, 你可以通过第三个参数把它传递过去:

```
$this->load->driver('session', '', 'my_session');
```

```
// Session class is now accessed using:
```

```
$this->my_session
```

```
view($view[, $vars = array(), return = FALSE])
```

参数

- **\$view** (*string*) – View name
- **\$vars** (*array*) – An associative array of variables
- **\$return** (*bool*) – Whether to return the loaded view

返回 View content string if \$return is set to TRUE, otherwise CI Loader instance (method chaining)

返回类型 mixed

该方法用于加载你的视图文件。如果你尚未阅读本手册的[视图](#) 章节的话, 建议你先去阅读那里的内容, 会有更详细的函数使用说明。

第一个参数是必须的, 指定你要载入的视图文件的名称。

注解: 无需加上.php 扩展名, 除非你使用了其他的扩展名。

第二个参数是 **** 可选的 ****, 允许你传入一个数组或对象参数, 传入的参数将使用 PHP 的 `extract()` 函数进行提取, 提取出来的变量可以在视图中使用。再说一遍, 请阅读[视图](#) 章节了解该功能的更多用法。

第三个参数是 **** 可选的 ****, 用于改变方法的行为, 将数据以字符串的形式返回, 而不是发送给浏览器。当你希望对数据进行一些特殊处理时, 这个参数就非常有用。如果你将这个参数设置为 TRUE, 方法就会返回数据。这个参数的默认值是 FALSE, 也就是数据将会被发送给浏览器。如果你希望数据被返回, 记得要将它赋值给一个变量:

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

```
vars($vars[, $val = ''])
```

参数

- **\$vars** (*mixed*) – An array of variables or a single variable name
- **\$val** (*mixed*) – Optional variable value

返回 CI.Loader instance (method chaining)

返回类型 CI.Loader

这个方法以一个关联数组作为输入参数, 将这个数组用 PHP 的 `extract()` 函数转化成与之对应的变量。这个方法的结果与上面的 `$this->load->view()` 方法使用第二个参数的结果一样。假如你想在控制器的构造函数中定义一些全局变量, 并希望这些变量在控制器的每一个方法加载的视图文件中都可用, 这种情况下你可能想单独使用这个函数。你可以多次调用该方法, 数据将被缓存, 并被合并为一个数组, 以便转换成变量。

`get_var($key)`

参数

- **\$key** (*string*) – Variable name key

返回 Value if key is found, NULL if not

返回类型 mixed

该方法检查关联数组中的变量对你的视图是否可用。当一个变量在一个类或者控制器的另一个方法里被以这样的方式定义时: `$this->load->vars()`, 会做这样的检查。

`get_vars()`

返回 An array of all assigned view variables

返回类型 array

该方法返回所有对视图可用的变量。

`clear_vars()`

返回 CI.Loader instance (method chaining)

返回类型 CI.Loader

清除缓存的视图变量。

`model($model[, $name = '', $db_conn = FALSE])`

参数

- **\$model** (*mixed*) – Model name or an array containing multiple models
- **\$name** (*string*) – Optional object name to assign the model to
- **\$db_conn** (*string*) – Optional database configuration group to load

返回 CI.Loader instance (method chaining)

返回类型 CI.Loader

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

如果你的模型位于子目录下, 加载时将路径包含进来即可。例如, 如果你有一个模型位于 `application/models/blog/Queries.php`, 你可以使用下面的方法来加载:

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

如果你希望将你的模型赋值给一个不同的变量, 你可以在第二个参数中指定:

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

```
database([$params = '', $return = FALSE, $query_builder = NULL]))
```

参数

- **\$params** (*mixed*) – Database group name or configuration options
- **\$return** (*bool*) – Whether to return the loaded database object
- **\$query_builder** (*bool*) – Whether to load the Query Builder

返回 Loaded CI_DB instance or FALSE on failure if \$return is set to TRUE, otherwise CI_Loader instance (method chaining)

返回类型 mixed

该方法用于加载数据库类, 有两个可选的参数。更多信息, 请阅读[数据库](#)。

```
dbforge([$db = NULL, $return = FALSE]))
```

参数

- **\$db** (*object*) – Database object
- **\$return** (*bool*) – Whether to return the Database Forge instance

返回 Loaded CI_DB_forge instance if \$return is set to TRUE, otherwise CI_Loader instance (method chaining)

返回类型 mixed

加载[数据库工厂类](#), 更多信息, 请参考该页面。

```
dbutil([$db = NULL, $return = FALSE]))
```

参数

- **\$db** (*object*) – Database object
- **\$return** (*bool*) – Whether to return the Database Utilities instance

返回 Loaded CI_DB_utility instance if \$return is set to TRUE, otherwise CI_Loader instance (method chaining)

返回类型 mixed

加载数据库工具类，更多信息，请参考该页面。

helper(\$helpers)

参数

- **\$helpers** (*mixed*) – Helper name as a string or an array containing multiple helpers

返回 CI.Loader instance (method chaining)

返回类型 CI.Loader

该方法用于加载辅助函数文件，其中 `file_name` 为加载的文件名，不带 `_helper.php` 后缀。

file(\$path[, \$return = FALSE])

参数

- **\$path** (*string*) – File path
- **\$return** (*bool*) – Whether to return the loaded file

返回 File contents if \$return is set to TRUE, otherwise CI.Loader instance (method chaining)

返回类型 mixed

这是一个通用的文件载入方法，在第一个参数中给出文件所在的路径和文件名，将会打开并读取对应的文件。默认情况下，数据会被发送给浏览器，就如同视图文件一样，但如果你将第二个参数设置为 TRUE，那么数据就会以字符串的形式被返回，而不是发送给浏览器。

language(\$files[, \$lang = ''])

参数

- **\$files** (*mixed*) – Language file name or an array of multiple language files
- **\$lang** (*string*) – Language name

返回 CI.Loader instance (method chaining)

返回类型 CI.Loader

该方法是语言加载方法 `$this->lang->load()` 的一个别名。

config(\$file[, \$use_sections = FALSE[, \$fail_gracefully = FALSE]])

参数

- **\$file** (*string*) – Configuration file name
- **\$use_sections** (*bool*) – Whether configuration values should be loaded into their own section
- **\$fail_gracefully** (*bool*) – Whether to just return FALSE in case of failure

返回 TRUE on success, FALSE on failure

返回类型 bool

该方法是配置文件加载方法 `$this->config->load()` 的一个别名。

`is_loaded($class)`

参数

- **\$class** (*string*) – Class name

返回 Singleton property name if found, FALSE if not

返回类型 mixed

用于检查某个类是否已经被加载。

注解: 这里的类指的是类库和驱动器。

如果类已经被加载, 方法返回它在 CodeIgniter 超级对象中被赋值的变量的名称, 如果没有加载, 返回 FALSE:

```
$this->load->library('form_validation');
$this->load->is_loaded('Form_validation'); // returns 'form_validation'

$this->load->is_loaded('Nonexistent_library'); // returns FALSE
```

重要: 如果你有类的多个实例 (被赋值给多个不同的属性), 那么将返回第一个的名称。

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

$this->load->is_loaded('Form_validation'); // returns 'fv'
```

`add_package_path($path[, $view_cascade = TRUE])`

参数

- **\$path** (*string*) – Path to add
- **\$view_cascade** (*bool*) – Whether to use cascading views

返回 CI.Loader instance (method chaining)

返回类型 CI.Loader

添加一个包路径, 用于告诉加载器类使用给定的路径来加载后续请求的资源。例如, "Foo Bar" 应用程序包里有一个名为 Foo_bar.php 的类, 在控制器中, 我们可以按照如下的方法调用:

```
$this->load->add_package_path(APPPATH.'third_party/foo_bar/')
->library('foo_bar');
```

`remove_package_path([$path = ''])`

参数

- **\$path** (*string*) – Path to remove

返回 CI.Loader instance (method chaining)

返回类型 CI.Loader

当你的控制器完成从应用程序包中读取资源，如果你还需要读取其他的应用程序包的资源，你会希望删除当前使用的包路径来让加载器不再使用这个文件夹中的资源。要删除最后一次使用的包路径，你可以直接不带参数的调用该方法。

或者你也可以删除一个特定的包路径，指定与之前使用 `add_package_path()` 方法时所加载的包相同的路径：

```
$this->load->remove_package_path(APPPATH.'third_party/foo_bar/');
```

```
get_package_paths([ $include_base = TRUE ])
```

参数

- **\$include_base** (*bool*) – Whether to include BASEPATH

返回 An array of package paths

返回类型 array

返回当前所有可用的包路径。

8.1.17 迁移类

迁移是一种非常方便的途径来组织和管理你的数据库变更，当你编写了一小段 SQL 对数据库做了修改之后，你就需要告诉其他的开发者他们也需要运行这段 SQL，而且当你将应用程序部署到生产环境时，你还需要记得对数据库已经做了哪些修改，需要执行哪些 SQL。

在 CodeIgniter 中，**migration** 表记录了当前已经执行了哪些迁移，所以你需要做的就是，修改你的应用程序文件然后调用 `$this->migration->current()` 方法迁移到当前版本，当前版本可以在 `application/config/migration.php` 文件中进行设置。

- 迁移文件命令规则
- 创建一次迁移
- 使用范例
- 迁移参数
- 类参考

迁移文件命令规则

每个迁移都是根据文件名中的数字顺序向前或向后运行，有两种不同的数字格式：

- **序列格式:** 每个迁移文件以数字序列格式递增命名, 从 **001** 开始, 每个数字都需要占三位, 序列之间不能有间隙。(这是 CodeIgniter 3.0 版本之前的命令方式)
- **时间戳格式:** 每个迁移文件以创建时间的时间戳来命名, 格式为: **YYYYMMDDHHIISS** (例如: **20121031100537**), 这种方式可以避免在团队环境下以序列命名可能造成的冲突, 而且也是 CodeIgniter 3.0 之后版本中推荐的命名方式。

可以在 `application/config/migration.php` 文件中的 `$config['migration_type']` 参数设置命名规则。

无论你选择了哪种规则, 将这个数字格式作为迁移文件的前缀, 并在后面添加一个下划线, 再加上一个描述性的名字。如下所示:

- `001_add_blog.php` (sequential numbering)
- `20121031100537_add_blog.php` (timestamp numbering)

创建一次迁移

这里是一个新博客站点的第一次迁移的例子, 所有的迁移文件位于 `application/migrations/` 目录, 并命名为这种格式: `20121031100537_add_blog.php`。

```
<?php

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

class Migration_Add_blog extends CI_Migration {

    public function up()
    {
        $this->dbforge->add_field(array(
            'blog_id' => array(
                'type' => 'INT',
                'constraint' => 5,
                'unsigned' => TRUE,
                'auto_increment' => TRUE
            ),
            'blog_title' => array(
                'type' => 'VARCHAR',
                'constraint' => '100',
            ),
            'blog_description' => array(
                'type' => 'TEXT',
                'null' => TRUE,
            ),
        ));
        $this->dbforge->add_key('blog_id', TRUE);
        $this->dbforge->create_table('blog');
    }
}
```

```

public function down()
{
    $this->dbforge->drop_table('blog');
}
}

```

然后在 `application/config/migration.php` 文件中设置:
`$config['migration_version'] = 20121031100537;`

使用范例

在这个例子中, 我们在 `application/controllers/Migrate.php` 文件中添加如下的代码来更新数据库:

```

<?php

class Migrate extends CI_Controller
{

    public function index()
    {
        $this->load->library('migration');

        if ($this->migration->current() === FALSE)
        {
            show_error($this->migration->error_string());
        }
    }

}

```

迁移参数

下表为所有可用的迁移参数。

参数	默认值	可选项	描述
migration_enabled	FALSE	TRUE / FALSE	启用或禁用迁移
migration_path	APP-PATH.'migrations/'	None	迁移目录所在位置
migration_version	0	None	当前数据库所使用版本
migration_table	migrations	None	用于存储当前版本的数据库表名
migration_auto_latest	FALSE	TRUE / FALSE	启用或禁用自动迁移
migration_type	'timestamp'	'timestamp' / 'sequential'	迁移文件的命名规则

类参考

class CI_Migration**current()**

返回 TRUE if no migrations are found, current version string on success, FALSE on failure

返回类型 mixed

迁移至当前版本。(当前版本通过 *application/config/migration.php* 文件的 `$config['migration_version']` 参数设置)

error_string()

返回 Error messages

返回类型 string

返回迁移过程中发生的错误信息。

find_migrations()

返回 An array of migration files

返回类型 array

返回 `migration_path` 目录下的所有迁移文件的数组。

latest()

返回 Current version string on success, FALSE on failure

返回类型 mixed

这个方法 和 `current()` 类似, 但是它并不是迁移到 `$config['migration_version']` 参数所对应的版本, 而是迁移到迁移文件中的最新版本。

version(*\$target_version*)**参数**

- **`$target_version`** (*mixed*) – Migration version to process

返回 TRUE if no migrations are found, current version string on success, FALSE on failure

返回类型 mixed

迁移到特定版本 (回退或升级都可以), 这个方法和 `current()` 类似, 但是忽略 `$config['migration_version']` 参数, 而是迁移到用户指定版本。

```
$this->migration->version(5);
```

8.1.18 输出类

输出类是个核心类，它的功能只有一个：发送 Web 页面内容到请求的浏览器。如果你开启缓存，它也负责缓存你的 Web 页面。

注解： 这个类由系统自动加载，你无需手工加载。

在一般情况下，你可能根本就不会注意到输出类，因为它无需你的干涉，对你来说完全是透明的。例如，当你使用加载器加载一个视图文件时，它会自动传入到输出类，并在系统执行的最后由 CodeIgniter 自动调用。尽管如此，在你需要时，你还是可以对输出进行手工处理。

- 类参考

类参考

class CI_Output

`$parse_exec_vars = TRUE;`

启用或禁用解析伪变量（{elapsed_time} 和 {memory_usage}）。

CodeIgniter 默认会在输出类中解析这些变量。要禁用它，可以在你的控制器中设置这个属性为 FALSE。

`$this->output->parse_exec_vars = FALSE;`

`set_output($output)`

参数

- `$output (string)` – String to set the output to

返回 CI_Output instance (method chaining)

返回类型 CI_Output

允许你手工设置最终的输出字符串。使用示例：

`$this->output->set_output($data);`

重要： 如果你手工设置输出，这必须放在方法的最后一步。例如，如果你正在某个控制器的方法中构造页面，将 `set_output` 这句代码放在方法的最后。

`set_content_type($mime_type[, $charset = NULL])`

参数

- `$mime_type (string)` – MIME Type identifier string
- `$charset (string)` – Character set

返回 CI-Output instance (method chaining)

返回类型 CI-Output

允许你设置你的页面的 MIME 类型, 可以很方便的提供 JSON 数据、JPEG、XML 等等格式。

```
$this->output
    ->set_content_type('application/json')
    ->set_output(json_encode(array('foo' => 'bar')));

$this->output
    ->set_content_type('jpeg') // You could also use ".jpeg" which will have the j
    ->set_output(file_get_contents('files/something.jpg'));
```

重要: 确保你传入到这个方法的 MIME 类型在 *application/config/mimes.php* 文件中能找到, 要不然方法不起任何作用。

你也可以通过第二个参数设置文档的字符集。

```
$this->output->set_content_type('css', 'utf-8');
```

get_content_type()

返回 Content-Type string

返回类型 string

获取当前正在使用的 HTTP 头 Content-Type , 不包含字符集部分。

```
$mime = $this->output->get_content_type();
```

注解: 如果 Content-Type 没有设置, 默认返回 'text/html' 。

get_header(\$header)

参数

- **\$header** (*string*) – HTTP header name

返回 HTTP response header or NULL if not found

返回类型 mixed

返回请求的 HTTP 头, 如果 HTTP 头还没设置, 返回 NULL 。例如:

```
$this->output->set_content_type('text/plain', 'UTF-8');
echo $this->output->get_header('content-type');
// Outputs: text/plain; charset=utf-8
```

注解: HTTP 头名称是不区分大小写的。

注解: 返回结果中也包括通过 PHP 原生的 `header()` 函数发送的原始 HTTP 头。

get_output()

返回 Output string

返回类型 string

允许你手工获取存储在输出类中的待发送的内容。使用示例:

```
$string = $this->output->get_output();
```

注意, 只有通过 CodeIgniter 输出类的某个方法设置过的数据, 例如 `$this->load->view()` 方法, 才可以使用该方法获取到。

append_output(\$output)

参数

- **\$output** (*string*) – Additional output data to append

返回 CI_Output instance (method chaining)

返回类型 CI_Output

向输出字符串附加数据。

```
$this->output->append_output($data);
```

set_header(\$header[, \$replace = TRUE])

参数

- **\$header** (*string*) – HTTP response header
- **\$replace** (*bool*) – Whether to replace the old header value, if it is already set

返回 CI_Output instance (method chaining)

返回类型 CI_Output

允许你手工设置服务器的 HTTP 头, 输出类将在最终显示页面时发送它。例如:

```
$this->output->set_header('HTTP/1.0 200 OK');
$this->output->set_header('HTTP/1.1 200 OK');
$this->output->set_header('Last-Modified: '.gmdate('D, d M Y H:i:s', $last_update));
$this->output->set_header('Cache-Control: no-store, no-cache, must-revalidate');
$this->output->set_header('Cache-Control: post-check=0, pre-check=0');
$this->output->set_header('Pragma: no-cache');
```

set_status_header([\$code = 200[, \$text = '']])

参数

- **\$code** (*int*) – HTTP status code
- **\$text** (*string*) – Optional message

返回 CI-Output instance (method chaining)

返回类型 CI-Output

允许你手工设置服务器的 HTTP 状态码。例如:

```
$this->output->set_status_header(401);  
// Sets the header as: Unauthorized
```

阅读[这里](#) 得到一份完整的 HTTP 状态码列表。

注解: 这个方法是[通用方法](#) 中的 `set_status_header()` 的别名。

enable_profiler(*[\$val = TRUE]*)

参数

- **\$val** (*bool*) – Whether to enable or disable the Profiler

返回 CI-Output instance (method chaining)

返回类型 CI-Output

允许你启用或禁用[程序分析器](#) , 它可以在你的页面底部显示基准测试的结果或其他一些数据帮助你调试和优化程序。

要启用分析器, 将下面这行代码放到你的[控制器](#) 方法的任何位置:

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

当启用它时, 将生成一份报告并插入到你的页面的最底部。

要禁用分析器, 你可以这样:

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

set_profiler_sections(*\$sections*)

参数

- **\$sections** (*array*) – Profiler sections

返回 CI-Output instance (method chaining)

返回类型 CI-Output

当程序分析器启用时, 该方法允许你启用或禁用程序分析器的特定字段。请参考[程序分析器](#) 文档获取详细信息。

cache(*\$time*)

参数

- **\$time** (*int*) – Cache expiration time in seconds

返回 CI-Output instance (method chaining)

返回类型 CI-Output

将当前页面缓存一段时间。

更多信息, 请阅读[文档缓存](#)。

```
_display([$output = ''])
```

参数

- **\$output** (*string*) – Output data override

返回 void

返回类型 void

发送最终输出结果以及服务器的 HTTP 头到浏览器, 同时它也会停止基准测试的计时器。

注解: 这个方法会在脚本执行的最后自动被调用, 你无需手工调用它。除非你在你的代码中使用了 `exit()` 或 `die()` 结束了脚本执行。

例如:

```
$response = array('status' => 'OK');
```

```
$this->output
```

```
    ->set_status_header(200)
```

```
    ->set_content_type('application/json', 'utf-8')
```

```
    ->set_output(json_encode($response, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE
```

```
    ->_display());
```

```
exit;
```

注解: 手工调用该方法而不结束脚本的执行, 会导致重复输出结果。

8.1.19 分页类

CodeIgniter 的分页类非常容易使用, 而且它 100% 可定制, 可以通过动态的参数, 也可以通过保存在配置文件中的参数。

- 例子
 - 说明
 - 在配置文件中设置参数
- 自定义分页
- 添加封装标签
- 自定义第一个链接
- 自定义最后一个链接
- 自定义下一页链接
- 自定义上一页链接
- 自定义当前页面链接
- 自定义数字链接
- 隐藏数字链接
- 给链接添加属性
- 禁用 “rel” 属性
- 类参考

如果你还不熟悉“分页”这个词，它指的是用于你在页面之间进行导航的链接。像下面这样：

```
« First < 1 2 3 4 5 > Last »
```

例子

下面是一个简单的例子，如何在你的控制器方法中创建分页：

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

$config['base_url'] = 'http://example.com/index.php/test/page/';
$config['total_rows'] = 200;
$config['per_page'] = 20;

$this->pagination->initialize($config);

echo $this->pagination->create_links();
```

说明

如上所示，`$config` 数组包含了你的配置参数，被传递到 `$this->pagination->initialize()` 方法。另外还有二十几个配置参数你可以选择，但是最少你只需要这三个配置参数。下面是这几个参数的含义：

- **base_url** 这是一个指向你的分页所在的控制器类/方法的完整的 URL，在上面的这个例子里，它指向了一个叫“Test”的控制器和它的一个叫“Page”的方法。记住，如果你需要一个不同格式的 URL，你可以[重新路由](#)。
- **total_rows** 这个数字表示你需要做分页的数据的总行数。通常这个数值是你查询数据库得到的数据总量。

- **per_page** 这个数字表示每个页面中希望展示的数量，在上面的那个例子中，每页显示 20 个项目。

当你没有分页需要显示时，`create_links()` 方法会返回一个空的字符串。

在配置文件中设置参数

如果你不喜欢用以上的方法进行参数设置，你可以将参数保存到配置文件中。简单地创建一个名为 `pagination.php` 的文件，把 `$config` 数组加到这个文件中，然后将文件保存到 `application/config/pagination.php`。这样它就可以自动被调用。用这个方法，你不再需要使用 `$this->pagination->initialize` 方法。

自定义分页

下面是所有的参数列表，可以传递给 `initialization` 方法来定制你喜欢的显示效果。

```
$config['uri_segment'] = 3;
```

分页方法自动检测你 URI 的哪一段包含页数，如果你的情况不一样，你可以明确指定它。

```
$config['num_links'] = 2;
```

放在你当前页码的前面和后面的“数字”链接的数量。比方说值为 2 就会在每一边放置两个数字链接，就像此页顶端的示例链接那样。

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

默认分页的 URL 中显示的是你当前正在从哪条记录开始分页，如果你希望显示实际的页数，将该参数设置为 `TRUE`。

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

默认情况下，分页类假设你使用 [URI 段](#)，并像这样构造你的链接：

```
http://example.com/index.php/test/page/20
```

如果你把 `$config['enable_query_strings']` 设置为 `TRUE`，你的链接将自动地被重写成查询字符串格式。这个选项也可以被明确地设置，把 `$config['page_query_string']` 设置为 `TRUE`，分页链接将变成：

```
http://example.com/index.php?c=test&m=page&per_page=20
```

请注意，“per_page”是默认传递的查询字符串，但也可以使用 `$config['query_string_segment'] = ' 000000'` 来配置。

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

默认情况下你的查询字符串参数会被忽略，将这个参数设置为 `TRUE`，将会将查询字符串参数添加到 URI 分段的后面以及 URL 后缀的前面。：

```
http://example.com/index.php/test/page/20?query=search%term
```

这可以让你混合使用 *URI 分段* 和查询字符串参数, 这在 3.0 之前的版本中是不行的。

```
$config['prefix'] = '';
```

给路径添加一个自定义前缀, 前缀位于偏移段的前面。

```
$config['suffix'] = '';
```

给路径添加一个自定义后缀, 后缀位于偏移段的后面。

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

当该参数设置为 TRUE 时, 会使用 `application/config/config.php` 配置文件中定义的 `$config['url_suffix']` 参数 **重写** `$config['suffix']` 的值。

添加封装标签

如果你希望在整个分页的周围用一些标签包起来, 你可以通过下面这两个参数:

```
$config['full_tag_open'] = '<p>;'
```

起始标签放在所有结果的左侧。

```
$config['full_tag_close'] = '</p>;'
```

结束标签放在所有结果的右侧。

自定义第一个链接

```
$config['first_link'] = 'First';
```

左边第一个链接显示的文本, 如果你不想显示该链接, 将其设置为 FALSE 。

注解: 该参数的值也可以通过语言文件来翻译。

```
$config['first_tag_open'] = '<div>;'
```

第一个链接的起始标签。

```
$config['first_tag_close'] = '</div>;'
```

第一个链接的结束标签。

```
$config['first_url'] = '';
```

可以为第一个链接设置一个自定义的 URL 。

自定义最后一个链接

```
$config['last_link'] = 'Last';
```

右边最后一个链接显示的文本, 如果你不想显示该链接, 将其设置为 FALSE 。

注解: 该参数的值也可以通过语言文件来翻译。

```
$config['last_tag_open'] = '<div>;
```

最后一个链接的起始标签。

```
$config['last_tag_close'] = '</div>;
```

最后一个链接的结束标签。

自定义下一页链接

```
$config['next_link'] = '&gt;;
```

下一页链接显示的文本, 如果你不想显示该链接, 将其设置为 FALSE 。

注解: 该参数的值也可以通过语言文件来翻译。

```
$config['next_tag_open'] = '<div>;
```

下一页链接的起始标签。

```
$config['next_tag_close'] = '</div>;
```

下一页链接的结束标签。

自定义上一页链接

```
$config['prev_link'] = '&lt;;
```

上一页链接显示的文本, 如果你不想显示该链接, 将其设置为 FALSE 。

注解: 该参数的值也可以通过语言文件来翻译。

```
$config['prev_tag_open'] = '<div>;
```

上一页链接的起始标签。

```
$config['prev_tag_close'] = '</div>;
```

上一页链接的结束标签。

自定义当前页面链接

```
$config['cur_tag_open'] = '<b>;
```

当前页链接的起始标签。

```
$config['cur_tag_close'] = '</b>;
```

当前页链接的结束标签。

自定义数字链接

```
$config['num_tag_open'] = '<div>';
```

数字链接的起始标签。

```
$config['num_tag_close'] = '</div>';
```

数字链接的结束标签。

隐藏数字链接

如果你不想显示数字链接（例如你只想显示上一页和下一页链接），你可以通过下面的代码来阻止它显示：

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

给链接添加属性

如果你想为分页类生成的每个链接添加额外的属性，你可以通过键值对设置 “attributes” 参数：

```
// Produces: class="myclass"
$config['attributes'] = array('class' => 'myclass');
```

注解： 以前的通过 “anchor_class” 参数来设置 class 属性的方法已经废弃。

禁用 “rel” 属性

默认 rel 属性会被自动的被添加到合适的链接上，如果由于某些原因，你想禁用它，你可以用下面的方法：

```
$config['attributes']['rel'] = FALSE;
```

类参考

class CI_Pagination

```
initialize([$params = array()])
```

参数

- **\$params** (*array*) – Configuration parameters

返回 CI_Pagination instance (method chaining)

返回类型 CI_Pagination

使用提供的参数初始化分页类。

`create_links()`

返回 HTML-formatted pagination

返回类型 string

返回分页的代码，包含生成的链接。如果只有一个页面，将返回空字符串。

8.1.20 模板解析类

模板解析类可以对你视图文件中的伪变量进行简单的替换，它可以解析简单的变量和变量标签对。

如果你从没使用过模板引擎，下面是个例子，伪变量名称使用大括号括起来：

```
<html>
  <head>
    <title>{blog_title}</title>
  </head>
  <body>
    <h3>{blog_heading}</h3>

    {blog_entries}
      <h5>{title}</h5>
      <p>{body}</p>
    {/blog_entries}

  </body>
</html>
```

这些变量并不是真正的 PHP 变量，只是普通的文本，这样能让你的模板（视图文件）中没有任何 PHP 代码。

注解： CodeIgniter **并没有** 让你必须使用这个类，因为直接在视图中使用纯 PHP 可能速度会更快点。尽管如此，一些开发者还是喜欢使用模板引擎，他们可能是和一些其他的不熟悉 PHP 的设计师共同工作。

重要： 模板解析类 **不是** 一个全面的模板解析方案，我们让它保持简洁，为了达到更高的性能。

- 使用模板解析类
 - 初始化类
 - 解析模板
 - 变量对
 - 使用说明
 - 视图片段
- 类参考

使用模板解析类

初始化类

跟 CodeIgniter 中的其他类一样, 可以在你的控制器中使用 `$this->load->library()` 方法加载模板解析类:

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

一旦加载, 模板解析类就可以像下面这样使用:

```
$this->parser
```

解析模板

你可以使用 `parse()` 方法来解析 (或显示) 简单的模板, 如下所示:

```
$data = array(
    'blog_title' => 'My Blog Title',
    'blog_heading' => 'My Blog Heading'
);

$this->parser->parse('blog_template', $data);
```

第一个参数为视图文件的名称 (在这个例子里, 文件名为 `blog_template.php`), 第二个参数为一个关联数组, 它包含了要对模板进行替换的数据。上例中, 模板将包含两个变量: `{blog_title}` 和 `{blog_heading}`。

没有必要对 `$this->parser->parse()` 方法返回的结果进行 `echo` 或其他处理, 它会自动的保存到输出类, 以待发送给浏览器。但是, 如果你希望它将数据返回而不是存到输出类里去, 你可以将第三个参数设置为 `TRUE`

```
$string = $this->parser->parse('blog_template', $data, TRUE);
```

变量对

上面的例子可以允许替换简单的变量, 但是如果你想重复某一块代码, 并且每次重复的值都不同又该怎么办呢? 看下我们一开始的时候展示那个模板例子:

```
<html>
  <head>
    <title>{blog_title}</title>
  </head>
  <body>
    <h3>{blog_heading}</h3>

    {blog_entries}
    <h5>{title}</h5>
    <p>{body}</p>
  </blog_entries>

</body>
</html>
```

在上面的代码中, 你会发现一对变量: {blog_entries} data... {/blog_entries}。这个例子的意思是, 这个变量对之间的整个数据块将重复多次, 重复的次数取决于“blog_entries”参数中元素的个数。

解析变量对和上面的解析单个变量的代码完全一样, 除了一点, 你需要根据变量对的数据使用一个多维的数组, 像下面这样:

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

$data = array(
    'blog_title'    => 'My Blog Title',
    'blog_heading' => 'My Blog Heading',
    'blog_entries' => array(
        array('title' => 'Title 1', 'body' => 'Body 1'),
        array('title' => 'Title 2', 'body' => 'Body 2'),
        array('title' => 'Title 3', 'body' => 'Body 3'),
        array('title' => 'Title 4', 'body' => 'Body 4'),
        array('title' => 'Title 5', 'body' => 'Body 5')
    )
);

$this->parser->parse('blog_template', $data);
```

如果你的变量对数据来自于数据库查询结果, 那么它已经是一个多维数组了, 你可以简单的使用数据库的 `result_array()` 方法:

```
$query = $this->db->query("SELECT * FROM blog");

$this->load->library('parser');

$data = array(
    'blog_title'    => 'My Blog Title',
    'blog_heading' => 'My Blog Heading',
    'blog_entries' => $query->result_array()
);

$this->parser->parse('blog_template', $data);
```

使用说明

如果你传入的某些参数在模板中没用到, 它们将被忽略:

```
$template = 'Hello, {firstname} {lastname}';
$data = array(
    'title' => 'Mr',
    'firstname' => 'John',
    'lastname' => 'Doe'
);
$this->parser->parse_string($template, $data);

// Result: Hello, John Doe
```

如果你的模板中用到了某个变量, 但是你传入的参数中没有, 将直接显示出原始的伪变量:

```
$template = 'Hello, {firstname} {initials} {lastname}';
$data = array(
    'title' => 'Mr',
    'firstname' => 'John',
    'lastname' => 'Doe'
);
$this->parser->parse_string($template, $data);

// Result: Hello, John {initials} Doe
```

如果你的模板中需要使用某个数组变量, 但是你传入的参数是个字符串类型, 那么变量对的起始标签将会被替换, 但是结束标签不会被正确显示:

```
$template = 'Hello, {firstname} {lastname} ({degrees}{degree} {/degrees})';
$data = array(
    'degrees' => 'Mr',
    'firstname' => 'John',
    'lastname' => 'Doe',
    'titles' => array(
        array('degree' => 'BSc'),
        array('degree' => 'PhD')
    )
);
$this->parser->parse_string($template, $data);

// Result: Hello, John Doe (Mr{degree} {/degrees})
```

如果你的某个单一变量的名称和变量对中的某个变量名称一样, 显示结果可能会不对:

```
$template = 'Hello, {firstname} {lastname} ({degrees}{degree} {/degrees})';
$data = array(
```

```

        'degree' => 'Mr',
        'firstname' => 'John',
        'lastname' => 'Doe',
        'degrees' => array(
            array('degree' => 'BSc'),
            array('degree' => 'PhD')
        )
    );
    $this->parser->parse_string($template, $data);

    // Result: Hello, John Doe (Mr Mr )

```

视图片段

你没必要在你的视图文件中使用变量对来实现重复，你也可以在变量对之间使用一个视图片段，在控制器，而不是视图文件中，来控制重复。

下面是一个在视图中实现重复的例子：

```

$template = '<ul>{menuitems}
    <li><a href="{link}">{title}</a></li>
{/menuitems}</ul>';

$data = array(
    'menuitems' => array(
        array('title' => 'First Link', 'link' => '/first'),
        array('title' => 'Second Link', 'link' => '/second'),
    )
);
$this->parser->parse_string($template, $data);

```

结果：

```

<ul>
    <li><a href="/first">First Link</a></li>
    <li><a href="/second">Second Link</a></li>
</ul>

```

下面是一个在控制器中利用视图片段来实现重复的例子：

```

$temp = '';
$template1 = '<li><a href="{link}">{title}</a></li>';
$data1 = array(
    array('title' => 'First Link', 'link' => '/first'),
    array('title' => 'Second Link', 'link' => '/second'),
);

foreach ($data1 as $menuitem)
{
    $temp .= $this->parser->parse_string($template1, $menuitem, TRUE);
}

```

```
$template = '<ul>{menuitems}</ul>';
$data = array(
    'menuitems' => $temp
);
$this->parser->parse_string($template, $data);
```

结果:

```
<ul>
  <li><a href="/first">First Link</a></li>
  <li><a href="/second">Second Link</a></li>
</ul>
```

类参考

class CI_Parser

`parse($template, $data[, $return = FALSE])`

参数

- **\$template** (*string*) – Path to view file
- **\$data** (*array*) – Variable data
- **\$return** (*bool*) – Whether to only return the parsed template

返回 Parsed template string

返回类型 string

根据提供的路径和变量解析一个模板。

`parse_string($template, $data[, $return = FALSE])`

参数

- **\$template** (*string*) – Path to view file
- **\$data** (*array*) – Variable data
- **\$return** (*bool*) – Whether to only return the parsed template

返回 Parsed template string

返回类型 string

该方法和 `parse()` 方法一样, 只是它接受一个字符串作为模板, 而不是去加载视图文件。

`set_delimiters([$l = '{', $r = '}'])`

参数

- **\$l** (*string*) – Left delimiter

- **\$r** (*string*) – Right delimiter

返回类型 void

设置模板中伪变量的分割符（起始标签和结束标签）。

8.1.21 安全类

安全类包含了一些方法，用于安全的处理输入数据，帮助你创建一个安全的应用。

- XSS 过滤
- 跨站请求伪造 (CSRF)
- 类参考

XSS 过滤

CodeIgniter comes with a Cross Site Scripting prevention filter, which looks for commonly used techniques to trigger JavaScript or other types of code that attempt to hijack cookies or do other malicious things. If anything disallowed is encountered it is rendered safe by converting the data to character entities.

使用 XSS 过滤器过滤数据可以使用 `xss_clean()` 方法:

```
$data = $this->security->xss_clean($data);
```

它还有一个可选的第二个参数 `is_image`，允许此函数对图片进行检测以发现那些潜在的 XSS 攻击，这对于保证文件上传的安全非常有用。当此参数被设置为 `TRUE` 时，函数的返回值将是一个布尔值，而不是一个修改过的字符串。如果图片是安全的则返回 `TRUE`，相反，如果图片中包含有潜在的、可能会被浏览器尝试运行的恶意信息，函数将返回 `FALSE`。

```
if ($this->security->xss_clean($file, TRUE) === FALSE)
{
    // file failed the XSS test
}
```

跨站请求伪造 (CSRF)

打开你的 `application/config/config.php` 文件，进行如下设置，即可启用 CSRF 保护:

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

如果你使用 **表单辅助函数**，`form_open()` 函数将会自动地在你的表单中插入一个隐藏的 CSRF 字段。如果没有插入这个字段，你可以手工调用 `get_csrf_token_name()` 和 `get_csrf_hash()` 这两个函数。

```
$csrf = array(
    'name' => $this->security->get_csrf_token_name(),
    'hash' => $this->security->get_csrf_hash()
);

...

<input type="hidden" name="<?=$csrf['name'];?>" value="<?=$csrf['hash'];?>" />
```

令牌 (tokens) 默认会在每一次提交时重新生成, 或者你也可以设置成在 CSRF cookie 的生命周期内一直有效。默认情况下令牌重新生成提供了更严格的安全机制, 但可能会对可用性带来一定的影响, 因为令牌很可能会变得失效 (例如使用浏览器的返回前进按钮、使用多窗口或多标签页浏览、异步调用等等)。你可以修改下面这个参数来改变这一点。

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

另外, 你可以添加一个 URI 的白名单, 跳过 CSRF 保护 (例如某个 API 接口希望接受原始的 POST 数据), 将这些 URI 添加到 'csrf_exclude_uris' 配置参数中:

```
$config['csrf_exclude_uris'] = array('api/person/add');
```

URI 中也支持使用正则表达式 (不区分大小写):

```
$config['csrf_exclude_uris'] = array(
    'api/record/[0-9]+',
    'api/title/[a-z]+'
);
```

类参考

class CI_Security

```
xss_clean($str[, $is_image = FALSE])
```

参数

- **\$str** (*mixed*) – Input string or an array of strings

返回 XSS-clean data

返回类型 mixed

尝试移除输入数据中的 XSS 代码, 并返回过滤后的数据。如果第二个参数设置为 TRUE, 将检查图片中是否含有恶意数据, 是的话返回 TRUE, 否则返回 FALSE。

```
sanitize_filename($str[, $relative_path = FALSE])
```

参数

- **\$str** (*string*) – File name/path

- **\$relative_path** (*bool*) – Whether to preserve any directories in the file path

返回 Sanitized file name/path

返回类型 string

尝试对文件名进行净化, 防止目录遍历尝试以及其他的安全威胁, 当文件名作为用户输入的参数时格外有用。

```
$filename = $this->security->sanitize_filename($this->input->post('filename'));
```

如果允许用户提交相对路径, 例如 *file/in/some/approved/folder.txt*, 你可以将第二个参数 **\$relative_path** 设置为 TRUE。

```
$filename = $this->security->sanitize_filename($this->input->post('filename'), TRUE);
```

get_csrf_token_name()

返回 CSRF token name

返回类型 string

返回 CSRF 的令牌名 (token name), 也就是 `$config['csrf_token_name']` 的值。

get_csrf_hash()

返回 CSRF hash

返回类型 string

返回 CSRF 哈希值 (hash value), 在和 `get_csrf_token_name()` 函数一起使用时很有用, 用于生成表单里的 CSRF 字段以及发送有效的 AJAX POST 请求。

entity_decode(\$str[, \$charset = NULL])

参数

- **\$str** (*string*) – Input string
- **\$charset** (*string*) – Character set of the input string

返回 Entity-decoded string

返回类型 string

该方法和 ENT_COMPAT 模式下的 PHP 原生函数 `html_entity_decode()` 差不多, 只是它除此之外, 还会检测不以分号结尾的 HTML 实体, 因为有些浏览器允许这样。

如果没有设置 **\$charset** 参数, 则使用你配置的 `$config['charset']` 参数作为编码格式。

get_random_bytes(\$length)

参数

- **\$length** (*int*) – Output length

返回 A binary stream of random bytes or FALSE on failure

返回类型 string

这是一种生成随机字符串的简易方法，该方法通过按顺序调用 `mcrypt_create_iv()`、`/dev/urandom` 和 `openssl_random_pseudo_bytes()` 这三个函数，只要有一个函数是可用的，都可以返回随机字符串。

用于生成 CSRF 和 XSS 的令牌。

注解： 输出并不能保证绝对安全，只是尽量做到更安全。

8.1.22 Session 类

Session（会话）类可以让你保持一个用户的“状态”，并跟踪他在浏览你的网站时的活动。

CodeIgniter 自带了几个存储 session 的驱动：

- 文件（默认的，基于文件系统）
- 数据库
- Redis
- Memcached

另外，你也可以基于其他的存储机制来创建你自己的自定义 session 存储驱动，使用自定义的驱动，同样也可以使用 Session 类提供的那些功能。

- 使用 Session 类
 - 初始化 Session 类
 - Session 是如何工作的?
 - * 关于并发的注意事项
 - 什么是 Session 数据?
 - 获取 Session 数据
 - 添加 Session 数据
 - 删除 Session 数据
 - Flashdata
 - Tempdata
 - 销毁 Session
 - 访问 session 元数据
 - Session 参数
 - Session 驱动
 - * 文件驱动
 - 小提示
 - * 数据库驱动
 - * Redis 驱动
 - * Memcached 驱动
 - 小提示
 - * 自定义驱动
- 类参考

使用 Session 类

初始化 Session 类

Session 通常会在每个页面载入的时候全局运行, 所以 Session 类必须首先被初始化。您可以在[控制器](#)的构造函数中初始化它, 也可以在系统中[自动加载](#)。Session 类基本上都是在后台运行, 你不会注意到。所以当初始化 session 之后, 系统会自动读取、创建和更新 session 数据。

要手动初始化 Session 类, 你可以在控制器的构造函数中使用 `$this->load->library()` 方法:

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

初始化之后, 就可以使用下面的方法来访问 Session 对象了:

```
$this->session
```

重要: 由于[加载类](#)是在 CodeIgniter 的控制器基类中实例化的, 所以如果要在你的控制器构造函数中加载类库的话, 确保先调用 `parent::__construct()` 方法。

Session 是如何工作的?

当页面载入后, Session 类就会检查用户的 cookie 中是否存在有效的 session 数据。如果 session 数据不存在 (或者与服务端不匹配, 或者已经过期), 那么就会创建一个新的 session 并保存起来。

如果 session 数据存在并且有效, 那么就会更新 session 的信息。根据你的配置, 每一次更新都会生成一个新的 Session ID 。

有一点非常重要, 你需要了解一下, Session 类一旦被初始化, 它就会自动运行。上面所说的那些, 你完全不用做任何操作。正如接下来你将看到的那样, 你可以正常的使用 session 数据, 至于读、写和更新 session 的操作都是自动完成的。

注解: 在 CLI 模式下, Session 类将自动关闭, 这种做法完全是基于 HTTP 协议的。

关于并发的注意事项 如果你开发的网站并不是大量的使用 AJAX 技术, 那么你可以跳过这一节。如果你的网站是大量的使用了 AJAX, 并且遇到了性能问题, 那么下面的注意事项, 可能正是你需要的。

在 CodeIgniter 之前的版本中, Session 类并没有实现锁机制, 这也就意味着, 两个 HTTP 请求可能会同时使用同一个 session 。说的更专业点就是, 请求是非阻塞的。(requests were non-blocking)

在处理 session 时使用非阻塞的请求同样意味着不安全, 因为在一个请求中修改 session 数据 (或重新生成 Session ID) 会对并发的第二个请求造成影响。这是导致很多问题的根源, 同时也是为什么 CodeIgniter 3.0 对 Session 类完全重写的原因。

那么为什么要告诉你这些呢? 这是因为在你查找性能问题的原因时, 可能会发现加锁机制正是导致性能问题的罪魁祸首, 因此就想着如何去掉锁...

请不要这样做! 去掉加锁机制是完全错误的, 它会给你带来更多的问题!

锁并不是问题, 它是一种解决方案。你的问题是当 session 已经处理完毕不再需要时, 你还将 session 保持是打开的状态。所以, 你需要做的其实是, 当结束当前请求时, 将不再需要的 session 关闭掉。

简单来说就是: 当你不再需要使用某个 session 变量时, 就使用 `session_write_close()` 方法来关闭它。

什么是 Session 数据?

Session 数据是个简单的数组, 带有一个特定的 session ID (cookie)。

如果你之前在 PHP 里使用过 session , 你应该对 PHP 的 `$_SESSION` 全局变量 很熟悉 (如果没有, 请阅读下链接中的内容)。

CodeIgniter 使用了相同的方式来访问 session 数据, 同时使用了 PHP 自带的 session 处理机制, 使用 session 数据和操作 `$_SESSION` 数组一样简单 (包括读取, 设置, 取消设置)。

另外, CodeIgniter 还提供了两种特殊类型的 session 数据:flashdata 和 tempdata, 在下面将有介绍。

注解: 在之前的 CodeIgniter 版本中, 常规的 session 数据被称之为 ‘userdata’, 当文档中出现这个词时请记住这一点。大部分都是用于解释自定义 ‘userdata’ 方法是如何工作的。

获取 Session 数据

session 数组中的任何信息都可以通过 `$_SESSION` 全局变量获取:

```
$_SESSION['item']
```

或使用下面的方法 (magic getter) :

```
$this->session->item
```

同时, 为了和之前的版本兼容, 也可以使用 `userdata()` 方法:

```
$this->session->userdata('item');
```

其中, item 是你想获取的数组的键值。例如, 将 ‘name’ 键值对应的项赋值给 `$name` 变量, 你可以这样:

```
$name = $_SESSION['name'];
```

```
// or:
```

```
$name = $this->session->name
```

```
// or:
```

```
$name = $this->session->userdata('name');
```

注解: 如果你访问的项不存在, `userdata()` 方法返回 NULL 。

如果你想获取所有已存在的 userdata , 你可以忽略 item 参数:

```
$_SESSION
```

```
// or:
```

```
$this->session->userdata();
```

添加 Session 数据

假设某个用户访问你的网站, 当他完成认证之后, 你可以将他的用户名和 email 地址添加到 session 中, 这样当你需要的时候你就可以直接访问这些数据, 而不用查询数据库了。

你可以简单的将数据赋值给 `$_SESSION` 数组, 或赋值给 `$this->session` 的某个属性。

同时, 老版本中的通过 “userdata” 来赋值的方法也还可以用, 只不过是传递一个包含你的数据的数组给 `set_userdata()` 方法:

```
$this->session->set_userdata($array);
```

其中, `$array` 是包含新增数据的一个关联数组, 下面是个例子:

```
$newdata = array(
    'username' => 'johndoe',
    'email'    => 'johndoe@some-site.com',
    'logged_in' => TRUE
);

$this->session->set_userdata($newdata);
```

如果你想一次只添加一个值, `set_userdata()` 也支持这种语法:

```
$this->session->set_userdata('some_name', 'some_value');
```

如果你想检查某个 session 值是否存在, 可以使用 `isset()`:

```
// returns FALSE if the 'some_name' item doesn't exist or is NULL,
// TRUE otherwise:
isset($_SESSION['some_name'])
```

或者, 你也可以使用 `has_userdata()`:

```
$this->session->has_userdata('some_name');
```

删除 Session 数据

和其他的变量一样, 可以使用 `unset()` 方法来删除 `$_SESSION` 数组中的某个值:

```
unset($_SESSION['some_name']);

// or multiple values:

unset(
    $_SESSION['some_name'],
    $_SESSION['another_name']
);
```

同时, 正如 `set_userdata()` 方法可用于向 session 中添加数据, `unset_userdata()` 方法可用于删除指定键值的数据。例如, 如果你想从你的 session 数组中删除 ‘some_name’:

```
$this->session->unset_userdata('some_name');
```

这个方法也可以使用一个数组来同时删除多个值:

```
$array_items = array('username', 'email');

$this->session->unset_userdata($array_items);
```

注解: 在 CodeIgniter 之前的版本中, `unset_userdata()` 方法接受一个关联数组, 包含 `key => 'dummy value'` 这样的键值对, 这种方式不再支持。

Flashdata

CodeIgniter 支持 “flashdata”, 它指的是一种只对下一次请求有效的 session 数据, 之后将会自动被清除。

这用于一次性的信息时特别有用, 例如错误或状态信息 (诸如 “第二条记录删除成功” 这样的信息)。

要注意的是, flashdata 就是常规的 session 变量, 只不过以特殊的方式保存在 ‘`__ci_vars`’ 键下 (警告: 请不要乱动这个值)。

将已有的值标记为 “flashdata”:

```
$this->session->mark_as_flash('item');
```

通过传一个数组, 同时标记多个值为 flashdata:

```
$this->session->mark_as_flash(array('item', 'item2'));
```

使用下面的方法来添加 flashdata:

```
$_SESSION['item'] = 'value';
$this->session->mark_as_flash('item');
```

或者, 也可以使用 `set_flashdata()` 方法:

```
$this->session->set_flashdata('item', 'value');
```

你还可以传一个数组给 `set_flashdata()` 方法, 和 `set_userdata()` 方法一样。

读取 flashdata 和读取常规的 session 数据一样, 通过 `$_SESSION` 数组:

```
$_SESSION['item']
```

重要: `userdata()` 方法不会返回 flashdata 数据。

如果你要确保你读取的就是 “flashdata” 数据, 而不是其他类型的数据, 可以使用 `flashdata()` 方法:

```
$this->session->flashdata('item');
```

或者不传参数, 直接返回所有的 flashdata 数组:

```
$this->session->flashdata();
```

注解: 如果读取的值不存在, `flashdata()` 方法返回 `NULL` 。

如果你需要在另一个请求中还继续保持 `flashdata` 变量, 你可以使用 `keep_flashdata()` 方法。可以传一个值, 或包含多个值的一个数组。

```
$this->session->keep_flashdata('item');  
$this->session->keep_flashdata(array('item1', 'item2', 'item3'));
```

Tempdata

CodeIgniter 还支持 “tempdata”, 它指的是一种带有有效时间的 `session` 数据, 当它的有效时间已过期, 或在有效时间内被删除, 都会自动被清除。

和 `flashdata` 一样, `tempdata` 也是常规的 `session` 变量, 只不过以特殊的方式保存在 `‘_ci_vars’` 键下 (再次警告: 请不要乱动这个值)。

将已有的值标记为 “tempdata”, 只需简单的将要标记的键值和过期时间 (单位为秒) 传给 `mark_as_temp()` 方法即可:

```
// 'item' will be erased after 300 seconds  
$this->session->mark_as_temp('item', 300);
```

你也可以同时标记多个值为 `tempdata`, 有下面两种不同的方式, 这取决于你是否要将所有的值都设置成相同的过期时间:

```
// Both 'item' and 'item2' will expire after 300 seconds  
$this->session->mark_as_temp(array('item', 'item2'), 300);  
  
// 'item' will be erased after 300 seconds, while 'item2'  
// will do so after only 240 seconds  
$this->session->mark_as_temp(array(  
    'item' => 300,  
    'item2' => 240  
));
```

使用下面的方法来添加 `tempdata`:

```
$_SESSION['item'] = 'value';  
$this->session->mark_as_temp('item', 300); // Expire in 5 minutes
```

或者, 也可以使用 `set_tempdata()` 方法:

```
$this->session->set_tempdata('item', 'value', 300);
```

你还可以传一个数组给 `set_tempdata()` 方法:

```
$tempdata = array('newuser' => TRUE, 'message' => 'Thanks for joining!');  
  
$this->session->set_tempdata($tempdata, NULL, $expire);
```

注解: 如果没有设置 `expiration` 参数, 或者设置为 0 , 将默认使用 300 秒 (5 分钟) 作为生存时间 (time-to-live)。

要读取 `tempdata` 数据, 你可以再一次通过 `$SESSION` 数组:

```
$_SESSION['item']
```

重要: `userdata()` 方法不会返回 `tempdata` 数据。

如果你要确保你读取的就是“`tempdata`”数据, 而不是其他类型的数据, 可以使用 `tempdata()` 方法:

```
$this->session->tempdata('item');
```

或者不传参数, 直接返回所有的 `tempdata` 数组:

```
$this->session->tempdata();
```

注解: 如果读取的值不存在, `tempdata()` 方法返回 `NULL` 。

如果你需要在某个 `tempdata` 过期之前删除它, 你可以直接通过 `$SESSION` 数组来删除:

```
unset($_SESSION['item']);
```

但是, 这不会删除这个值的 `tempdata` 标记 (会在下一次 HTTP 请求时失效), 所以, 如果你打算在相同的请求中重用这个值, 你可以使用 `unset_tempdata()`:

```
$this->session->unset_tempdata('item');
```

销毁 Session

要清除当前的 `session` (例如: 退出登录时), 你可以简单的使用 PHP 自带的 `session_destroy()` 函数或者 `sess_destroy()` 方法。两种方式效果完全一样:

```
session_destroy();
```

```
// or
```

```
$this->session->sess_destroy();
```

注解: 这必须是同一个请求中关于 `session` 的最后一次操作, 所有的 `session` 数据 (包括 `flashdata` 和 `tempdata`) 都被永久性销毁, 销毁之后, 关于 `session` 的方法将不可用。

访问 session 元数据

在之前的 CodeIgniter 版本中, session 数据默认包含 4 项: 'session_id'、'ip_address'、'user_agent'、'last_activity'。

这是由 session 具体的工作方式决定的, 但是我们现在的实现没必要这样做了。尽管如此, 你的应用程序可能还依赖于这些值, 所以下面提供了访问这些值的替代方法:

- session_id: `session_id()`
- ip_address: `$_SERVER['REMOTE_ADDR']`
- user_agent: `$this->input->user_agent()` (unused by sessions)
- last_activity: 取决于 session 的存储方式, 没有直接的方法, 抱歉!

Session 参数

在 CodeIgniter 中通常所有的东西都是拿来直接就可以用的, 尽管如此, session 对于所有的程序来说, 都是一个非常敏感的部分, 所以必须要小心的配置它。请花点时间研究下下面所有的选项以及每个选项的作用。

你可以在你的配置文件 `application/config/config.php` 中找到下面的关于 session 的配置参数:

参数	默认值	选项	描述
<code>sess_driver</code>	<code>files</code>	<code>files/</code> <code>database/</code> <code>redis/</code> <code>memcached/</code> <code>custom</code>	使用的存储 session 的驱动
<code>sess_cookie_name</code>	<code>ci_session</code>	<code>[A-Za-z_-]</code> characters only	session cookie 的名称
<code>sess_expire_on_close</code>	<code>TRUE</code>	<code>TRUE/</code> <code>FALSE</code> (boolean)	你希望 session 持续的秒数如果你希望 session 不过期 (直到浏览器关闭), 将其设置为 0
<code>sess_save_path</code>	<code>NULL</code>	<code>None</code>	指定存储位置, 取决于使用的存储 session 的驱动
<code>sess_match_ip</code>	<code>FALSE</code>	<code>TRUE/</code> <code>FALSE</code> (boolean)	读取 session cookie 时, 是否验证用户的 IP 地址 注意有些 ISP 会动态的修改 IP, 所以如果你想要一个不过期的 session, 将其设置为 <code>FALSE</code>
<code>sess_time_to_update</code>	<code>300</code>	<code>Time in</code> seconds (integer)	该选项用于控制过多久将重新生成一个新 session ID 设置为 0 将禁用 session ID 的重新生成
<code>sess_regenerate_destroy</code>	<code>FALSE</code>	<code>TRUE/</code> <code>FALSE</code> (boolean)	当自动重新生成 session ID 时, 是否销毁老的 session ID 对应的数据 如果设置为 <code>FALSE</code> , 数据之后将自动被垃圾回收器删除

注解: 如果上面的某个参数没有配置, Session 类将会试图读取 `php.ini` 配置文件中的 session 相关的配置 (例如 `'sess_expire_on_close'`)。但是, 请不要依赖于这个行为, 因为

这可能会导致不可预期的结果，而且这也有可能在未来的版本中修改。请合理的配置每一个参数。

除了上面的这些参数之外，cookie 和 session 原生的驱动还会公用下面这些由[输入类](#)和[安全类](#)提供的配置参数。

参数	默认值	描述
cookie_domain	'	session 可用的域
cookie_path	/	session 可用的路径
cookie_secure	FALSE	是否只在加密连接 (HTTPS) 时创建 session cookie

注解： 'cookie_httponly' 配置对 session 没有影响。出于安全原因，HttpOnly 参数将一直启用。另外，'cookie_prefix' 参数完全可以忽略。

Session 驱动

正如上面提到的，Session 类自带了 4 种不同的驱动（或叫做存储引擎）可供使用：

- files
- database
- redis
- memcached

默认情况下，初始化 session 时将使用[文件驱动](#)，因为这是最安全的选择，可以在所有地方按预期工作（几乎所有的环境下都有文件系统）。

但是，你也可以通过 **application/config/config.php** 配置文件中的 `$config['sess_driver']` 参数来使用任何其他的驱动。特别提醒的是，每一种驱动都有它自己的注意事项，所以在你选择之前，确定你熟悉它们。

另外，如果默认提供的这些不能满足你的需求，你也可以创建和使用[自定义驱动](#)。

注解： 在之前版本的 CodeIgniter 中，只有“cookie 驱动”这唯一的一种选择，因为这个我们收到了大量的负面的反馈。因此，我们吸取了社区的反馈意见，同时也要提醒你，因为它 **** 不安全 ****，所以已经被废弃了，建议你不要试着通过自定义驱动来重新实现它。

文件驱动 文件驱动利用你的文件系统来存储 session 数据。

可以说，文件驱动和 PHP 自带的默认 session 实现非常类似，但是有一个很重要的细节要注意的是，实际上它们的代码并不相同，而且有一些局限性（以及优势）。

说的更具体点，它不支持 PHP 的 `session.save_path` 参数的目录分级（directory level）和 `mode` 格式，另外为了安全性大多数的参数都被硬编码。只提供了 `$config['sess_save_path']` 参数用于设置绝对路径。

另一个很重要的事情是，确保存储 session 文件的目录不能被公开访问到或者是共享目录，确保 **只有你** 能访问并查看配置的 `sess_save_path` 目录中的内容。否则，如

果任何人都能访问，他们就可以从中窃取到当前的 session（这也被称为 session 固定（session fixation）攻击）

在类 UNIX 操作系统中，这可以通过在该目录上执行 `chmod` 命令，将权限设置为 0700 来实现，这样就可以只允许目录的所有者执行读取和写入操作。但是要注意的是，脚本的执行者通常不是你自己，而是类似于 ‘www-data’ 这样的用户，所以只设置权限可能会破坏你的程序。

根据你的环境，你应该像下面这样来操作。

```
mkdir /<path to your application directory>/sessions/  
chmod 0700 /<path to your application directory>/sessions/  
chown www-data /<path to your application directory>/sessions/
```

小提示 有些人可能会选择使用其他的 session 驱动，他们认为文件存储通常比较慢。其实这并不总是对的。

执行一些简单的测试可能会让你真的相信 SQL 数据库更快一点，但是在 99% 的情况下，这只是当你的 session 并发非常少的时候是对的。当 session 的并发数越来越大，服务器的负载越来越高，这时就不一样了，文件系统将会胜过几乎所有的关系型数据库。

另外，如果性能是你唯一关心的，你可以看下 `tmpfs`（注意：外部资源），它可以让你的 session 非常快。

数据库驱动 数据库驱动使用诸如 MySQL 或 PostgreSQL 这样的关系型数据库来存储 session，这是一个非常常见的选择，因为它可以让开发者非常方便的访问应用中的 session 数据，因为它只是你的数据库中的一个表而已。

但是，还是有几点要求必须满足：

- 只有设置为 **default** 的数据库连接可以使用（或者在控制器中使用 `$this->db` 来访问的连接）
- 你必须启用[查询构造器](#)
- 不能使用持久连接
- 使用的数据库连接不能启用 `cache_on` 参数

为了使用数据库驱动，你还需要创建一个我们刚刚已经提到的数据表，然后将 `$config['sess_save_path']` 参数设置为表名。例如，如果你想使用 ‘ci_sessions’ 这个表名，你可以这样：

```
$config['sess_driver'] = 'database';  
$config['sess_save_path'] = 'ci_sessions';
```

注解： 如果你从 CodeIgniter 之前的版本中升级过来的，并且没有配置 ‘sess_save_path’ 参数，Session 类将查找并使用老的 ‘sess_table_name’ 参数替代。请不要依赖这个行为，因为它可能会在以后的版本中移除。

然后，新建数据表。

对于 MySQL:

```
CREATE TABLE IF NOT EXISTS `ci_sessions` (
  `id` varchar(40) NOT NULL,
  `ip_address` varchar(45) NOT NULL,
  `timestamp` int(10) unsigned DEFAULT 0 NOT NULL,
  `data` blob NOT NULL,
  KEY `ci_sessions_timestamp` (`timestamp`)
);
```

对于 PostgreSQL:

```
CREATE TABLE "ci_sessions" (
  "id" varchar(40) NOT NULL,
  "ip_address" varchar(45) NOT NULL,
  "timestamp" bigint DEFAULT 0 NOT NULL,
  "data" text DEFAULT '' NOT NULL
);

CREATE INDEX "ci_sessions_timestamp" ON "ci_sessions" ("timestamp");
```

You will also need to add a PRIMARY KEY depending on your 'sess_match_ip' setting. The examples below work both on MySQL and PostgreSQL:

```
// When sess_match_ip = TRUE
ALTER TABLE ci_sessions ADD PRIMARY KEY (id, ip_address);

// When sess_match_ip = FALSE
ALTER TABLE ci_sessions ADD PRIMARY KEY (id);

// To drop a previously created primary key (use when changing the setting)
ALTER TABLE ci_sessions DROP PRIMARY KEY;
```

重要: 只有 MySQL 和 PostgreSQL 数据库是被正式支持的, 因为其他数据库平台都缺乏合适的锁机制。在没锁的情况下使用 session 可能会导致大量的问题, 特别是使用了大量的 AJAX, 所以我们并不打算支持这种情况。如果你遇到了性能问题, 请你在完成 session 数据的处理之后, 调用 `session_write_close()` 方法。

Redis 驱动

注解: 由于 Redis 没有锁机制, 这个驱动的锁是通过一个保持 300 秒的值来模拟的 (emulated by a separate value that is kept for up to 300 seconds)。

Redis 是一种存储引擎, 通常用于缓存, 并由于他的高性能而流行起来, 这可能也正是你使用 Redis 驱动的原因。

缺点是它并不像关系型数据库那样普遍, 需要你的系统中安装了 `phpredis` 这个 PHP 扩展, 它并不是 PHP 程序自带的。可能的情况是, 你使用 Redis 驱动的原因是你已经非常熟悉 Redis 了并且你使用它还有其他的目的。

和文件驱动和数据库驱动一样, 你必须通过 `$config['sess_save_path']` 参数来配置存储 session 的位置。这里的格式有些不同, 同时也要复杂一点, 这在 `phpredis` 扩

展的 README 文件中有很好的解释, 链接如下:

<https://github.com/phpredis/phpredis#php-session-handler>

警告: CodeIgniter 的 Session 类并没有真的用到 'redis' 的 `session.save_handler`, 只是采用了它的路径的格式而已。

最常见的情况是, 一个简单 `host:port` 对就可以了:

```
$config['sess_driver'] = 'redis';
$config['sess_save_path'] = 'tcp://localhost:6379';
```

Memcached 驱动

注解: 由于 Memcache 没有锁机制, 这个驱动的锁是通过一个保持 300 秒的值来模拟的 (emulated by a separate value that is kept for up to 300 seconds)。

Memcached 驱动和 Redis 驱动非常相似, 除了它的可用性可能要好点, 因为 PHP 的 `Memcached` 扩展已经通过 PECL 发布了, 并且在某些 Linux 发行版本中, 可以非常方便的安装它。

除了这一点, 以及排除任何对 Redis 的偏见, 关于 Memcached 要说的真的没什么区别, 它也是一款通常用于缓存的产品, 而且以它的速度而闻名。

不过, 值得注意的是, 使用 Memcached 设置 X 的过期时间为 Y 秒, 它只能保证 X 会在 Y 秒过后被删除 (但不会早于这个时间)。这个是非常少见的, 但是应该注意一下, 因为它可能会导致 session 的丢失。

`$config['sess_save_path']` 参数的格式相当简单, 使用 `host:port` 对即可:

```
$config['sess_driver'] = 'memcached';
$config['sess_save_path'] = 'localhost:11211';
```

小提示 也可以使用一个可选的 `权重` 参数来支持多服务器的配置, 权重参数使用冒号分割 (`:weight`), 但是我们并没有测试这是绝对可靠的。

如果你想体验这个特性 (风险自负), 只需简单的将多个服务器使用逗号分隔:

```
// localhost will be given higher priority (5) here,
// compared to 192.0.2.1 with a weight of 1.
$config['sess_save_path'] = 'localhost:11211:5,192.0.2.1:11211:1';
```

自定义驱动 你也可以创建你自己的自定义 session 驱动, 但是要记住的是, 这通常来说都不是那么简单, 因为需要用到很多知识来正确实现它。

你不仅要知道 session 一般的工作原理, 而且要知道它在 PHP 中是如何实现的, 还要知道它的内部存储机制是如何工作的, 如何去处理并发, 如何去避免死锁 (不是通过去掉锁机制), 以及最后一点但也是很很重要的一点, 如何去处理潜在的安全问题。

总的来说, 如果你不知道怎么在原生的 PHP 中实现这些, 那么你也不应该在 CodeIgniter 中尝试实现它。我已经警告过你了。

如果你只想给你的 session 添加一些额外的功能, 你只要扩展 Session 基类就可以了, 这要容易的多。要学习如何实现这点, 请阅读[创建你的类库](#)这一节。

言归正传, 当你为 CodeIgniter 创建 session 驱动时, 有三条规则你必须遵循:

- 将你的驱动文件放在 `application/libraries/Session/drivers/` 目录下, 并遵循 Session 类所使用的命名规范。

例如, 如果你想创建一个名为 'dummy' 的驱动, 那么你需要创建一个名为 `Session_dummy_driver` 的类, 并将其放在 `application/libraries/Session/drivers/Session_dummy_driver.php` 文件中。

- 扩展 `CI_Session_driver` 类。

这只是一个拥有几个内部辅助方法的基本类, 同样可以和其他类库一样被扩展。如果你真的需要这样做, 我们并不打算在这里多做解释, 因为如果你知道如何在 CI 中扩展或覆写类, 那么你已经知道这样做的方法了。如果你还不知道, 那么可能你根本就不应该这样做。

- 实现 `SessionHandlerInterface` 接口。

注解: 你可能已经注意到 `SessionHandlerInterface` 接口已经在 PHP 5.4.0 之后的版本中提供了。CodeIgniter 会在你运行老版本的 PHP 时自动声明这个接口。

参考连接中的内容, 了解为什么以及如何实现。

所以, 使用我们上面的 'dummy' 驱动的例子, 你可能会写如下代码:

```
// application/libraries/Session/drivers/Session_dummy_driver.php:
```

```
class CI_Session_dummy_driver extends CI_Session_driver implements SessionHandlerInterface
{

    public function __construct(&$params)
    {
        // DO NOT forget this
        parent::__construct($params);

        // Configuration & other initializations
    }

    public function open($save_path, $name)
    {
        // Initialize storage mechanism (connection)
    }

    public function read($session_id)
    {
        // Read session data (if exists), acquire locks
    }

    public function write($session_id, $session_data)
```

```
{
    // Create / update session data (it might not exist!)
}

public function close()
{
    // Free locks, close connections / streams / etc.
}

public function destroy($session_id)
{
    // Call close() method & destroy data for current session (order may differ)
}

public function gc($maxlifetime)
{
    // Erase data for expired sessions
}
}
```

如果一切顺利，现在你就可以将 `sess_driver` 参数设置为 ‘dummy’，来使用你自定义的驱动。恭喜你！

类参考

class CI_Session

userdata(*[\$key = NULL]*)

参数

- **\$key** (*mixed*) – Session item key or NULL

返回 Value of the specified item key, or an array of all userdata

返回类型 mixed

从 `$_SESSION` 数组中获取指定的项。如果没有指定参数，返回所有 “userdata” 的数组。

注解： 这是个遗留方法，只是为了和老的应用程序向前兼容而保留。你可以直接使用 `$_SESSION` 替代它。

all_userdata()

返回 An array of all userdata

返回类型 array

返回所有 “userdata” 的数组。

注解: 该方法已废弃, 使用不带参数的 `userdata()` 方法来代替。

&get_userdata()

返回 A reference to `$_SESSION`

返回类型 array

返回一个 `$_SESSION` 数组的引用。

注解: 这是个遗留方法, 只是为了和老的应用程序向前兼容而保留。

has_userdata(*\$key*)

参数

- **\$key** (*string*) – Session item key

返回 TRUE if the specified key exists, FALSE if not

返回类型 bool

检查 `$_SESSION` 数组中是否存在某项。

注解: 这是个遗留方法, 只是为了和老的应用程序向前兼容而保留。它只是 `isset($_SESSION[$key])` 的一个别名, 请使用这个来替代它。

set_userdata(*\$data* [, *\$value* = NULL])

参数

- **\$data** (*mixed*) – An array of key/value pairs to set as session data, or the key for a single item
- **\$value** (*mixed*) – The value to set for a specific session item, if \$data is a key

返回类型 void

将数据赋值给 `$_SESSION` 全局变量。

注解: 这是个遗留方法, 只是为了和老的应用程序向前兼容而保留。

unset_userdata(*\$key*)

参数

- **\$key** (*mixed*) – Key for the session data item to unset, or an array of multiple keys

返回类型 void

从 `$_SESSION` 全局变量中删除某个值。

注解: 这是个遗留方法, 只是为了和老的应用程序向前兼容而保留。它只是 `unset($_SESSION[$key])` 的一个别名, 请使用这个来替代它。

mark_as_flash(\$key)

参数

- **\$key** (*mixed*) – Key to mark asflashdata, or an array of multiple keys

返回 TRUE on success, FALSE on failure

返回类型 bool

将 \$_SESSION 数组中的一项（或多项）标记为“flashdata”。

get_flash_keys()

返回 Array containing the keys of all “flashdata” items.

返回类型 array

获取 \$_SESSION 数组中所有标记为“flashdata”的一个列表。

unmark_flash(\$key)

参数

- **\$key** (*mixed*) – Key to be un-marked asflashdata, or an array of multiple keys

返回类型 void

将 \$_SESSION 数组中的一项（或多项）移除“flashdata”标记。

flashdata([\$key = NULL])

参数

- **\$key** (*mixed*) – Flashdata item key or NULL

返回 Value of the specified item key, or an array of all flashdata

返回类型 mixed

从 \$_SESSION 数组中获取某个标记为“flashdata”的指定项。如果没有指定参数，返回所有“flashdata”的数组。

注解： 这是个遗留方法，只是为了和老的应用程序向前兼容而保留。你可以直接使用 `$_SESSION` 替代它。

keep_flashdata(\$key)

参数

- **\$key** (*mixed*) – Flashdata key to keep, or an array of multiple keys

返回 TRUE on success, FALSE on failure

返回类型 bool

将某个指定的“flashdata”设置为在下一次请求中仍然保持有效。

注解: 这是个遗留方法, 只是为了和老的应用程序向前兼容而保留。它只是 `mark_as_flash()` 方法的一个别名。

```
set_flashdata($data[, $value = NULL])
```

参数

- **\$data** (*mixed*) – An array of key/value pairs to set as flashdata, or the key for a single item
- **\$value** (*mixed*) – The value to set for a specific session item, if \$data is a key

返回类型 void

将数据赋值给 `$_SESSION` 全局变量, 并标记为“flashdata”。

注解: 这是个遗留方法, 只是为了和老的应用程序向前兼容而保留。

```
mark_as_temp($key[, $ttl = 300])
```

参数

- **\$key** (*mixed*) – Key to mark as tempdata, or an array of multiple keys
- **\$ttl** (*int*) – Time-to-live value for the tempdata, in seconds

返回 TRUE on success, FALSE on failure

返回类型 bool

将 `$_SESSION` 数组中的一项 (或多项) 标记为“tempdata”。

```
get_temp_keys()
```

返回 Array containing the keys of all “tempdata” items.

返回类型 array

获取 `$_SESSION` 数组中所有标记为“tempdata”的一个列表。

```
unmark_temp($key)
```

参数

- **\$key** (*mixed*) – Key to be un-marked as tempdata, or an array of multiple keys

返回类型 void

将 `$_SESSION` 数组中的一项 (或多项) 移除“tempdata”标记。

```
tempdata([$key = NULL])
```

参数

- **\$key** (*mixed*) – Tempdata item key or NULL

返回 Value of the specified item key, or an array of all tempdata

返回类型 mixed

从 `$_SESSION` 数组中获取某个标记为“tempdata”的指定项。如果没有指定参数，返回所有“tempdata”的数组。

注解: 这是个遗留方法，只是为了和老的应用程序向前兼容而保留。你可以直接使用 `$_SESSION` 替代它。

set_tempdata(\$data[, \$value = NULL])

参数

- **\$data** (*mixed*) – An array of key/value pairs to set as tempdata, or the key for a single item
- **\$value** (*mixed*) – The value to set for a specific session item, if \$data is a key
- **\$ttl** (*int*) – Time-to-live value for the tempdata item(s), in seconds

返回类型 void

将数据赋值给 `$_SESSION` 全局变量，并标记为“tempdata”。

注解: 这是个遗留方法，只是为了和老的应用程序向前兼容而保留。

sess_regenerate([\$destroy = FALSE])

参数

- **\$destroy** (*bool*) – Whether to destroy session data

返回类型 void

重新生成 session ID，\$destroy 参数可选，用于销毁当前的 session 数据。

注解: 该方法只是 PHP 原生的 `session_regenerate_id()` 函数的一个别名而已。

sess_destroy()

返回类型 void

销毁当前 session。

注解: 这个方法必须在处理 session 相关的操作的 **** 最后 **** 调用。如果调用这个方法，所有的 session 数据都会丢失。

注解: 该方法只是 PHP 原生的 `session_destroy()` 函数的一个别名而已。

`--get($key)`

参数

- **\$key** (*string*) – Session item key

返回 The requested session data item, or NULL if it doesn't exist

返回类型 mixed

魔术方法，根据你的喜好，使用 `$this->session->item` 这种方式来替代 `$_SESSION['item']`。

如果你访问 `$this->session->session_id` 它也会调用 `session_id()` 方法来返回 session ID。

`--set($key, $value)`

参数

- **\$key** (*string*) – Session item key
- **\$value** (*mixed*) – Value to assign to the session item key

返回 void

魔术方法，直接赋值给 `$this->session` 属性，以此来替代赋值给 `$_SESSION` 数组：

```
$this->session->foo = 'bar';

// Results in:
// $_SESSION['foo'] = 'bar';
```

8.1.23 HTML 表格类

表格类提供了一些方法用于根据数组或数据库结果集自动生成 HTML 的表格。

- 使用表格类
 - 初始化类
 - 例子
 - 修改表格样式
- 类参考

使用表格类

初始化类

跟 CodeIgniter 中的其他类一样，可以在你的控制器中使用 `$this->load->library()` 方法加载表格类：

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

一旦加载, 表格类就可以像下面这样使用:

```
$this->table
```

例子

下面这个例子向你演示了如何通过多维数组来创建一个表格。注意数组的第一行将会变成表格的表头（或者你也可以通过下面介绍的 `set_heading()` 方法来设置你自己的表头）。

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

$data = array(
    array('Name', 'Color', 'Size'),
    array('Fred', 'Blue', 'Small'),
    array('Mary', 'Red', 'Large'),
    array('John', 'Green', 'Medium')
);

echo $this->table->generate($data);
```

下面这个例子是通过数据库查询结果来创建一个表格。表格类将使用查询结果的列名自动生成表头（或者你也可以通过下面介绍的 `set_heading()` 方法来设置你自己的表头）。

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

$query = $this->db->query('SELECT * FROM my_table');

echo $this->table->generate($query);
```

下面这个例子演示了如何使用分开的参数来生成表格:

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

$this->table->set_heading('Name', 'Color', 'Size');

$this->table->add_row('Fred', 'Blue', 'Small');
$this->table->add_row('Mary', 'Red', 'Large');
$this->table->add_row('John', 'Green', 'Medium');

echo $this->table->generate();
```

下面这个例子和上面的一样, 但是它不是使用分开的参数, 而是使用了数组:

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

$this->table->set_heading(array('Name', 'Color', 'Size'));
```

```

$this->table->add_row(array('Fred', 'Blue', 'Small'));
$this->table->add_row(array('Mary', 'Red', 'Large'));
$this->table->add_row(array('John', 'Green', 'Medium'));

echo $this->table->generate();

```

修改表格样式

表格类可以让你设置一个表格的模板, 你可以通过它设计表格的样式, 下面是模板的原型:

```

$template = array(
    'table_open'          => '<table border="0" cellpadding="4" cellspacing="0">',

    'thead_open'          => '<thead>',
    'thead_close'         => '</thead>',

    'heading_row_start'   => '<tr>',
    'heading_row_end'     => '</tr>',
    'heading_cell_start'  => '<th>',
    'heading_cell_end'    => '</th>',

    'tbody_open'          => '<tbody>',
    'tbody_close'         => '</tbody>',

    'row_start'           => '<tr>',
    'row_end'             => '</tr>',
    'cell_start'          => '<td>',
    'cell_end'            => '</td>',

    'row_alt_start'       => '<tr>',
    'row_alt_end'         => '</tr>',
    'cell_alt_start'      => '<td>',
    'cell_alt_end'        => '</td>',

    'table_close'         => '</table>'
);

$this->table->set_template($template);

```

注解: 你会发现模板中有两个“row”代码块, 它可以让你的表格每行使用交替的颜色, 或者其他的这种隔行的设计元素。

你不用设置整个模板, 只需要设置你想修改的部分即可。在下面这个例子中, 只有 table 的起始标签需要修改:

```

$template = array(
    'table_open' => '<table border="1" cellpadding="2" cellspacing="1" class="mytable">'

```

```
);
```

```
$this->table->set_template($template);
```

你也可以在配置文件中设置默认模板。

类参考

class CI_Table

\$function = NULL

允许你指定一个原生的 PHP 函数或一个有效的函数数组对象，该函数会作用于所有的单元格数据。

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

$this->table->set_heading('Name', 'Color', 'Size');
$this->table->add_row('Fred', '<strong>Blue</strong>', 'Small');

$this->table->function = 'htmlspecialchars';
echo $this->table->generate();
```

上例中，所有的单元格数据都会先通过 PHP 的 htmlspecialchars() 函数，结果如下：

```
<td>Fred</td><td>&lt;strong&gt;Blue&lt;/strong&gt;</td><td>Small</td>
```

generate(*[\$table_data = NULL]*)

参数

- **\$table_data** (*mixed*) – Data to populate the table rows with

返回 HTML table

返回类型 string

返回生成的表格的字符串。接受一个可选的参数，该参数可以是一个数组或是从数据库获取的结果对象。

set_caption(*\$caption*)

参数

- **\$caption** (*string*) – Table caption

返回 CI_Table instance (method chaining)

返回类型 CI_Table

允许你给表格添加一个标题。

```
$this->table->set_caption('Colors');
```

```
set_heading([$args = array(), ...])
```

参数

- **\$args** (*mixed*) – An array or multiple strings containing the table column titles

返回 CI_Table instance (method chaining)

返回类型 CI_Table

允许你设置表格的表头。你可以提交一个数组或分开的参数：

```
$this->table->set_heading('Name', 'Color', 'Size');
```

```
$this->table->set_heading(array('Name', 'Color', 'Size'));
```

```
add_row([$args = array(), ...])
```

参数

- **\$args** (*mixed*) – An array or multiple strings containing the row values

返回 CI_Table instance (method chaining)

返回类型 CI_Table

允许你在你的表格中添加一行。你可以提交一个数组或分开的参数：

```
$this->table->add_row('Blue', 'Red', 'Green');
```

```
$this->table->add_row(array('Blue', 'Red', 'Green'));
```

如果你想要单独设置一个单元格的属性，你可以使用一个关联数组。关联数组的键名 **data** 定义了这个单元格的数据。其它的键值对 `key => val` 将会以 `key='val'` 的形式被添加为该单元格的属性里：

```
$cell = array('data' => 'Blue', 'class' => 'highlight', 'colspan' => 2); $this->table->add_row($cell, 'Red', 'Green');
```

```
// generates // <td class='highlight' colspan='2'>Blue</td><td>Red</td><td>Green</td>
```

```
make_columns([$array = array(), $col_limit = 0])
```

参数

- **\$array** (*array*) – An array containing multiple rows' data
- **\$col_limit** (*int*) – Count of columns in the table

返回 An array of HTML table columns

返回类型 array

这个函数以一个一维数组为输入，创建一个多维数组，它的深度（译注：不是行数，而是每一行的元素个数）和列数一样。这个函数可以把一个含有多个元素的数组按指定列在表格中显示出来。参考下面的例子：


```

$list = array('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight',
'nine', 'ten', 'eleven', 'twelve');

$new_list = $this->table->make_columns($list, 3);

$this->table->generate($new_list);

// Generates a table with this prototype

<table border="0" cellpadding="4" cellspacing="0"> <tr>
<td>one</td><td>two</td><td>three</td> </tr><tr>
<td>four</td><td>five</td><td>six</td> </tr><tr>
<td>seven</td><td>eight</td><td>nine</td> </tr><tr>
<td>ten</td><td>eleven</td><td>twelve</td></tr> </
table>

```

set_template(\$template)

参数

- **\$template** (*array*) – An associative array containing template values

返回 TRUE on success, FALSE on failure

返回类型 bool

允许你设置你的模板。你可以提交整个模板或部分模板。

```

$template = array(
    'table_open' => '<table border="1" cellpadding="2" cellspacing="1" class="myt
);

```

```

$this->table->set_template($template);

```

set_empty(\$value)

参数

- **\$value** (*mixed*) – Value to put in empty cells

返回 CI_Table instance (method chaining)

返回类型 CI_Table

用于设置当表格中的单元格为空时要显示的默认值。例如，设置一个不换行空格 (NBSP, non-breaking space) :

```

$this->table->set_empty("&nbsp;");

```

clear()

返回 CI_Table instance (method chaining)

返回类型 CI_Table

使你能清除表格的表头和行中的数据。如果你需要显示多个有不同数据的表格，那么你需要在每个表格生成之后调用这个函数来清除之前表格的信息。例如：

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

$this->table->set_heading('Name', 'Color', 'Size'); $this->table-
>add_row('Fred', 'Blue', 'Small'); $this->table->add_row('Mary',
'Red', 'Large'); $this->table->add_row('John', 'Green', 'Medium');

echo $this->table->generate();

$this->table->clear();

$this->table->set_heading('Name', 'Day', 'Delivery'); $this-
>table->add_row('Fred', 'Wednesday', 'Express'); $this->table-
>add_row('Mary', 'Monday', 'Air'); $this->table->add_row('John',
'Saturday', 'Overnight');

echo $this->table->generate();
```

8.1.24 引用通告类

引用通告类提供了一些方法用于发送和接受引用通告数据。

如果你还不知道什么是引用通告，可以在 [这里](#) 找到更多信息。

- 使用引用通告类
 - 初始化类
 - 发送引用通告
 - 接受引用通告
 - 你的 Ping URL
 - 新建 Trackback 表
 - 处理引用通告
 - * 说明
- 类参考

使用引用通告类

初始化类

正如 CodeIgniter 中的其他类一样，在你的控制器中使用 `$this->load->library()` 方法来初始化引用通告类：

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

初始化之后，引用通告类的对象就可以这样访问：

```
$this->trackback
```

发送引用通告

可以在你的控制器的任何方法中使用类似于如下代码来发送引用通告:

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

$tb_data = array(
    'ping_url'  => 'http://example.com/trackback/456',
    'url'       => 'http://www.my-example.com/blog/entry/123',
    'title'     => 'The Title of My Entry',
    'excerpt'   => 'The entry content.',
    'blog_name' => 'My Blog Name',
    'charset'   => 'utf-8'
);

if ( ! $this->trackback->send($tb_data))
{
    echo $this->trackback->display_errors();
}
else
{
    echo 'Trackback was sent!';
}
```

数组中每一项的解释:

- **ping_url** - 你想发送引用通告到该站点的 URL , 你可以同时向发送多个 URL 发送, 多个 URL 之间使用逗号分割
- **url** - 对应的是你的博客的 URL
- **title** - 你的博客标题
- **excerpt** - 你的博客内容 (摘要)
- **blog_name** - 你的博客的名称
- **charset** - 你的博客所使用的字符编码, 如果忽略, 默认使用 UTF-8

注解: 引用通告类会自动发送你的博客的前 500 个字符, 同时它也会去除所有的 HTML 标签。

发送引用通告的方法会根据成功或失败返回 TRUE 或 FALSE , 如果失败, 可以使用下面的代码获取错误信息:

```
$this->trackback->display_errors();
```

接受引用通告

在接受引用通告之前, 你必须先创建一个博客, 如果你还没有博客, 那么接下来的内容对你来说就没什么意义。

接受引用通告比发送要复杂一点, 这是因为你需要一个数据库表来保存它们, 而且你还需要对接受到的引用通告数据进行验证。我们鼓励你实现一个完整的验证过程, 来防止垃圾信息和重复数据。你可能还希望限制一段时间内从某个 IP 发送过来的引用通告的数量, 以此减少垃圾信息。接受引用通告的过程很简单, 验证才是难点。

你的 Ping URL

为了接受引用通告, 你必须在你的每篇博客旁边显示一个引用通告 URL, 人们使用这个 URL 来向你发送引用通告 (我们称其为 Ping URL)。

你的 Ping URL 必须指向一个控制器方法, 在该方法中写接受引用通告的代码, 而且该 URL 必须包含你博客的 ID, 这样当接受到引用通告时你就可以知道是针对哪篇博客的。

例如, 假设你的控制器类叫 Trackback, 接受方法叫 receive, 你的 Ping URL 将类似于下面这样:

```
http://example.com/index.php/trackback/receive/entry_id
```

其中, entry_id 代表你每篇博客的 ID。

新建 Trackback 表

在接受引用通告之前, 你必须创建一个数据库表来储存它。下面是表的一个基本原型:

```
CREATE TABLE trackbacks (
    tb_id int(10) unsigned NOT NULL auto_increment,
    entry_id int(10) unsigned NOT NULL default 0,
    url varchar(200) NOT NULL,
    title varchar(100) NOT NULL,
    excerpt text NOT NULL,
    blog_name varchar(100) NOT NULL,
    tb_date int(10) NOT NULL,
    ip_address varchar(45) NOT NULL,
    PRIMARY KEY `tb_id` (`tb_id`),
    KEY `entry_id` (`entry_id`)
);
```

在引用通告的规范中只有四项信息是发送一个引用通告所必须的: url、title、excerpt 和 blog_name。但为了让数据更有用, 我们还在表中添加了几个其他的字段 (date、ip_address 等)。

处理引用通告

下面是一个如何接受并处理引用通告的例子。下面的代码将放在你的接受引用通告的控制器方法中:

```
$this->load->library('trackback');
$this->load->database();

if ($this->uri->segment(3) == FALSE)
{
    $this->trackback->send_error('Unable to determine the entry ID');
}

if ( ! $this->trackback->receive())
{
    $this->trackback->send_error('The Trackback did not contain valid data');
}

$data = array(
    'tb_id'      => '',
    'entry_id'   => $this->uri->segment(3),
    'url'        => $this->trackback->data('url'),
    'title'      => $this->trackback->data('title'),
    'excerpt'    => $this->trackback->data('excerpt'),
    'blog_name'  => $this->trackback->data('blog_name'),
    'tb_date'    => time(),
    'ip_address' => $this->input->ip_address()
);

$sql = $this->db->insert_string('trackbacks', $data);
$this->db->query($sql);

$this->trackback->send_success();
```

说明 entry_id 将从你的 URL 的第三段获取, 这是基于我们之前例子中的 URL:
`http://example.com/index.php/trackback/receive/entry_id`

注意 entry_id 是第三段, 你可以这样获取:

```
$this->uri->segment(3);
```

在我们上面的接受引用通告的代码中, 如果第三段为空, 我们将发送一个错误信息。如果没有有效的 entry_id, 没必要继续处理下去。

`$this->trackback->receive()` 是个简单的验证方法, 它检查接受到的数据并确保包含了我们所需的四种信息: url、title、excerpt 和 blog_name。该方法成功返回 TRUE, 失败返回 FALSE。如果失败, 也发送一个错误信息。

接受到的引用通告数据可以通过下面的方法来获取:

```
$this->trackback->data('item')
```

其中, item 代表四种信息中的一种: url、title、excerpt 和 blog_name。

如果引用通告数据成功接受, 你可以使用下面的代码发送一个成功消息:

```
$this->trackback->send_success();
```

注解: 上面的代码中不包含数据校验, 我们建议你添加。

类参考

class CI_Trackback

```
$data = array('url' => '', 'title' => '', 'excerpt' => '', 'blog_name' => '',
```

引用通告数据的数组。

```
$convert_ascii = TRUE
```

是否将高位 ASCII 和 MS Word 特殊字符转换为 HTML 实体。

```
send($tb_data)
```

参数

- **\$tb_data** (*array*) – Trackback data

返回 TRUE on success, FALSE on failure

返回类型 bool

发送引用通告。

```
receive()
```

返回 TRUE on success, FALSE on failure

返回类型 bool

该方法简单的检验接受到的引用通告数据, 成功返回 TRUE, 失败返回 FALSE。如果数据是有效的, 将添加到 `$this->data` 数组, 以便保存到数据库。

```
send_error([$message = 'Incomplete information'])
```

参数

- **\$message** (*string*) – Error message

返回类型 void

向引用通告请求返回一条错误信息。

注解: 该方法将会终止脚本的执行。

```
send_success()
```

返回类型 void

向引用通告请求返回一条成功信息。

注解: 该方法将会终止脚本的执行。

data(\$item)

参数

- **\$item** (*string*) – Data key

返回 Data value or empty string if not found

返回类型 string

从引用通告数据中获取一项记录。

process(\$url, \$data)

参数

- **\$url** (*string*) – Target url
- **\$data** (*string*) – Raw POST data

返回 TRUE on success, FALSE on failure

返回类型 bool

打开一个 socket 连接，并将数据传送到服务器。成功返回 TRUE ，失败返回 FALSE 。

extract_urls(\$urls)

参数

- **\$urls** (*string*) – Comma-separated URL list

返回 Array of URLs

返回类型 array

该方法用于发送多条引用通告，它接受一个包含多条 URL 的字符串（以逗号或空格分割），将其转换为一个数组。

validate_url(~~\$~~\$url)

参数

- **\$url** (*string*) – Trackback URL

返回类型 void

如果 URL 中没有包括协议部分，该方法简单将 *http://* 前缀添加到 URL 前面。

get_id(\$url)

参数

- **\$url** (*string*) – Trackback URL

返回 URL ID or FALSE on failure

返回类型 string

查找并返回一个引用通告 URL 的 ID , 失败返回 FALSE 。

convert_xml (*\$str*)

参数

- **\$str** (*string*) – Input string

返回 Converted string

返回类型 string

将 XML 保留字符转换为实体。

limit_characters (*\$str* [, *\$n* = 500 [, *\$end_char* = '…']])

参数

- **\$str** (*string*) – Input string
- **\$n** (*int*) – Max characters number
- **\$end_char** (*string*) – Character to put at end of string

返回 Shortened string

返回类型 string

将字符串裁剪到指定字符个数, 会保持单词的完整性。

convert_ascii (*\$str*)

参数

- **\$str** (*string*) – Input string

返回 Converted string

返回类型 string

将高位 ASCII 和 MS Word 特殊字符转换为 HTML 实体。

set_error (*\$msg*)

参数

- **\$msg** (*string*) – Error message

返回类型 void

设置一个错误信息。

display_errors ([*\$open* = '<p>' [, *\$close* = '</p>']])

参数

- **\$open** (*string*) – Open tag

- **\$close** (*string*) – Close tag

返回 HTML formatted error messages

返回类型 string

返回 HTML 格式的错误信息，如果没有错误，返回空字符串。

8.1.25 排版类

排版类提供了一些方法用于帮助你格式化文本。

- 使用排版类
 - 初始化该类
- 类参考

使用排版类

初始化该类

跟 CodeIgniter 中的其他类一样，可以在你的控制器中使用 `$this->load->library()` 方法加载排版类：

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

一旦加载，排版类就可以像下面这样使用：

```
$this->typography
```

类参考

class CI_Typography

\$protect_braced_quotes = FALSE

当排版类和模板解析器类同时使用时，经常需要保护大括号中的单引号和双引号不被转换。要保护这个，将 `protect_braced_quotes` 属性设置为 `TRUE`。

使用示例：

```
$this->load->library('typography');  
$this->typography->protect_braced_quotes = TRUE;
```

format_characters(\$str)

参数

- **\$str** (*string*) – Input string

返回 Formatted string

返回类型 string

该方法和上面的 `auto_typography()` 类似, 但是它只对字符进行处理:

- 除了出现在标签中的引号外, 引号会被转换成正确的实体。
- 撇号被转换为相应的实体。
- 双破折号 (像 – 或 -) 被转换成 em 破折号。
- 三个连续的点也会被转换为省略号…。
- 句子后连续的多个空格将被转换为 ` `; 以便在网页中显示。

使用示例:

```
$string = $this->typography->format_characters($string);
```

`nl2br_except_pre($str)`

参数

- `$str (string)` – Input string

返回 Formatted string

返回类型 string

将换行符转换为 `
` 标签, 忽略 `<pre>` 标签中的换行符。除了对 `<pre>` 标签中的换行处理有所不同之外, 这个函数和 PHP 函数 `nl2br()` 是完全一样的。

使用示例:

```
$string = $this->typography->nl2br_except_pre($string);
```

8.1.26 单元测试类

单元测试是一种为你的应用程序中的每个函数编写测试的软件开发方法。如果你还不熟悉这个概念, 你应该先去 Google 一下。

CodeIgniter 的单元测试类非常简单, 由一个测试方法和两个显示结果的方法组成。它没打算成为一个完整的测试套件, 只是提供一个简单的机制来测试你的代码是否生成了正确的数据类型和结果。

- 使用单元测试类库
 - 初始化类
 - 运行测试
 - 生成报告
 - 严格模式
 - 启用/禁用单元测试
 - 单元测试结果显示
 - * 自定义显示测试结果
 - * 创建模板
- 类参考

使用单元测试类库

初始化类

正如 CodeIgniter 中的其他类一样, 在你的控制器中使用 `$this->load->library()` 方法来初始化单元测试类:

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

初始化之后, 单元测试类的对象就可以这样访问:

```
$this->unit
```

运行测试

要运行一个测试用例, 需要提供一个测试和一个期望结果, 像下面这样:

```
$this->unit->run('test', 'expected result', 'test name', 'notes');
```

其中, `test` 是你希望测试的代码的结果, `expected result` 是期望返回的结果, `test name` 是可选的, 你可以为你的测试取一个名字, `notes` 是可选的, 可以填些备注信息。例如:

```
$test = 1 + 1;
```

```
$expected_result = 2;
```

```
$test_name = 'Adds one plus one';
```

```
$this->unit->run($test, $expected_result, $test_name);
```

期望的结果可以是字面量匹配 (a literal match), 也可以是数据类型匹配 (a data type match)。下面是字面量匹配的例子:

```
$this->unit->run('Foo', 'Foo');
```

下面是数据类型匹配的例子:

```
$this->unit->run('Foo', 'is_string');
```

注意第二个参数“is_string”，这让方法测试返回的结果是否是字符串类型。以下是可用的数据类型的列表：

- is_object
- is_string
- is_bool
- is_true
- is_false
- is_int
- is_numeric
- is_float
- is_double
- is_array
- is_null
- is_resource

生成报告

你可以在每个测试之后显示出测试的结果，也可以先运行几个测试，然后在最后生成一份测试结果的报告。要简单的显示出测试结果，可以直接在 run 方法的前面使用 echo：

```
echo $this->unit->run($test, $expected_result);
```

要显示一份所有测试的完整报告，使用如下代码：

```
echo $this->unit->report();
```

这份报告会以 HTML 的表格形式显示出来，如果你喜欢获取原始的数据，可以通过下面的代码得到一个数组：

```
echo $this->unit->result();
```

严格模式

默认情况下，单元测试类在字面量匹配时是松散的类型匹配。看下面这个例子：

```
$this->unit->run(1, TRUE);
```

正在测试的结果是一个数字，期望的结果是一个布尔型。但是，由于 PHP 的松散数据类型，如果使用常规的比较操作符的话，上面的测试结果将会是 TRUE。

```
if (1 == TRUE) echo 'This evaluates as true';
```

如果愿意的话, 你可以将单元测试设置为严格模式, 它不仅会比较两个数据的值, 而且还会比较两个数据的数据类型:

```
if (1 === TRUE) echo 'This evaluates as FALSE';
```

使用如下代码启用严格模式:

```
$this->unit->use_strict(TRUE);
```

启用/禁用单元测试

如果你希望在你的代码中保留一些测试, 只在需要的时候才被执行, 可以使用下面的代码禁用单元测试:

```
$this->unit->active(FALSE);
```

单元测试结果显示

单元测试的结果默认显示如下几项:

- Test Name (test_name)
- Test Datatype (test_datatype)
- Expected Datatype (res_datatype)
- Result (result)
- File Name (file)
- Line Number (line)
- Any notes you entered for the test (notes)

你可以使用 `$this->unit->set_test_items()` 方法自定义要显示哪些结果, 例如, 你只想显示出测试名和测试的结果:

自定义显示测试结果

```
$this->unit->set_test_items(array('test_name', 'result'));
```

创建模板 如果你想让你的测试结果以不同于默认的格式显示出来, 你可以设置你自己的模板, 这里是一个简单的模板例子, 注意那些必须的伪变量:

```
$str = '  
<table border="0" cellpadding="4" cellspacing="1">  
{rows}  
  <tr>  
    <td>{item}</td>
```

```

        <td>{result}</td>
    </tr>
{/rows}
</table>';

$this->unit->set_template($str);

```

注解: 你的模板必须在运行测试 **之前** 被定义。

类参考

class CI_Unit_test

set_test_items(*\$items*)

参数

- **\$items** (*array*) – List of visible test items

返回 void

设置要在测试的结果中显示哪些项，有效的选项有：

- test_name
- test_datatype
- res_datatype
- result
- file
- line
- notes

run(*\$test* [, *\$expected* = TRUE] [, *\$test_name* = 'undefined'] [, *\$notes* = ''])

参数

- **\$test** (*mixed*) – Test data
- **\$expected** (*mixed*) – Expected result
- **\$test_name** (*string*) – Test name
- **\$notes** (*string*) – Any notes to be attached to the test

返回 Test report

返回类型 string

运行单元测试。

report([*\$result* = array()])

参数

- **\$result** (*array*) – Array containing tests results

返回 Test report

返回类型 string

根据已运行的测试生成一份测试结果的报告。

```
use_strict([$state = TRUE])
```

参数

- **\$state** (*bool*) – Strict state flag

返回类型 void

在测试中启用或禁用严格比较模式。

```
active([$state = TRUE])
```

参数

- **\$state** (*bool*) – Whether to enable testing

返回类型 void

启用或禁用单元测试。

```
result([$results = array()])
```

参数

- **\$results** (*array*) – Tests results list

返回 Array of raw result data

返回类型 array

返回原始的测试结果数据。

```
set_template($template)
```

参数

- **\$template** (*string*) – Test result template

返回类型 void

设置显示测试结果数据的模板。

8.1.27 URI 类

URI 类用于帮助你从 URI 字符串中获取信息，如果你使用 URI 路由，你也可以从路由后的 URI 中获取信息。

注解： 该类由系统自己加载，无需手工加载。

- 类参考

类参考

class CI_URI

segment(\$n[, \$no_result = NULL])

参数

- **\$n** (*int*) – Segment index number
- **\$no_result** (*mixed*) – What to return if the searched segment is not found

返回 Segment value or \$no_result value if not found

返回类型 mixed

用于从 URI 中获取指定段。参数 n 为你希望获取的段序号，URI 的段从左到右进行编号。例如，如果你的完整 URL 是这样的：

`http://example.com/index.php/news/local/metro/crime_is_up`

那么你的每个分段如下：

```
#. news
#. local
#. metro
#. crime_is_up
```

第二个参数为可选的，默认为 NULL，它用于设置当所请求的段不存在时的返回值。例如，如下代码在失败时将返回数字 0

```
$product_id = $this->uri->segment(3, 0);
```

它可以避免你写出类似于下面这样的代码：

```
if ($this->uri->segment(3) === FALSE)
{
    $product_id = 0;
}
else
{
    $product_id = $this->uri->segment(3);
}
```

rsegment(\$n[, \$no_result = NULL])

参数

- **\$n** (*int*) – Segment index number

- **\$no_result** (*mixed*) – What to return if the searched segment is not found

返回 Routed segment value or \$no_result value if not found

返回类型 *mixed*

当你使用 CodeIgniter 的 [URI 路由](#) 功能时, 该方法和 `segment()` 类似, 只是它用于从路由后的 URI 中获取指定段。

```
slash_segment($n[, $where = 'trailing'])
```

参数

- **\$n** (*int*) – Segment index number
- **\$where** (*string*) – Where to add the slash ('trailing' or 'leading')

返回 Segment value, prepended/suffixed with a forward slash, or a slash if not found

返回类型 *string*

该方法和 `segment()` 类似, 只是它会根据第二个参数在返回结果的前面或/和后面添加斜线。如果第二个参数未设置, 斜线会添加到后面。例如:

```
$this->uri->slash_segment(3);  
$this->uri->slash_segment(3, 'leading');  
$this->uri->slash_segment(3, 'both');
```

返回结果:

- 1.segment/
- 2./segment
- 3./segment/

```
slash_rsegment($n[, $where = 'trailing'])
```

参数

- **\$n** (*int*) – Segment index number
- **\$where** (*string*) – Where to add the slash ('trailing' or 'leading')

返回 Routed segment value, prepended/suffixed with a forward slash, or a slash if not found

返回类型 *string*

当你使用 CodeIgniter 的 [URI 路由](#) 功能时, 该方法和 `slash_segment()` 类似, 只是它用于从路由后的 URI 返回结果的前面或/和后面添加斜线。

```
uri_to_assoc([ $n = 3[, $default = array() ] ])
```

参数

- **\$n** (*int*) – Segment index number
- **\$default** (*array*) – Default values

返回 Associative URI segments array

返回类型 array

该方法用于将 URI 的段转换为一个包含键值对的关联数组。如下 URI:

`index.php/user/search/name/joe/location/UK/gender/male`

使用这个方法你可以将 URI 转为如下的数组原型:

```
[array]
(
    'name'      => 'joe'
    'location'  => 'UK'
    'gender'    => 'male'
)
```

你可以通过第一个参数设置一个位移, 默认值为 3, 这是因为你的 URI 的前两段通常都是控制器和方法。例如:

```
$array = $this->uri->uri_to_assoc(3);
echo $array['name'];
```

第二个参数用于设置默认的键名, 这样即使 URI 中缺少某个键名, 也能保证返回的数组中包含该索引。例如:

```
$default = array('name', 'gender', 'location', 'type', 'sort');
$array = $this->uri->uri_to_assoc(3, $default);
```

如果某个你设置的默认键名在 URI 中不存在, 数组中的该索引值将设置为 NULL。

另外, 如果 URI 中的某个键没有相应的值与之对应 (例如 URI 的段数为奇数), 数组中的该索引值也将设置为 NULL。

`ruri_to_assoc([$n = 3], $default = array())`

参数

- **\$n** (*int*) – Segment index number
- **\$default** (*array*) – Default values

返回 Associative routed URI segments array

返回类型 array

当你使用 CodeIgniter 的 *URI 路由* 功能时, 该方法和 `uri_to_assoc()` 类似, 只是它用于将路由后的 URI 的段转换为一个包含键值对的关联数组。

`assoc_to_uri($array)`

参数

- `$array` (*array*) – Input array of key/value pairs

返回 URI string

返回类型 string

根据输入的关联数组生成一个 URI 字符串，数组的键将包含在 URI 的字符串中。例如：

```
$array = array('product' => 'shoes', 'size' => 'large', 'color' => 'red');  
$str = $this->uri->assoc_to_uri($array);  
  
// Produces: product/shoes/size/large/color/red
```

`uri_string()`

返回 URI string

返回类型 string

返回一个相对的 URI 字符串，例如，如果你的完整 URL 为：

`http://example.com/index.php/news/local/345`

该方法返回：

`news/local/345`

`ruri_string()`

返回 Routed URI string

返回类型 string

当你使用 CodeIgniter 的 [URI 路由](#) 功能时，该方法和 `uri_string()` 类似，只是它用于返回路由后的 URI。

`total_segments()`

返回 Count of URI segments

返回类型 int

返回 URI 的总段数。

`total_rsegments()`

返回 Count of routed URI segments

返回类型 int

当你使用 CodeIgniter 的 [URI 路由](#) 功能时，该方法和 `total_segments()` 类似，只是它用于返回路由后的 URI 的总段数。

`segment_array()`

返回 URI segments array

返回类型 array

返回 URI 所有的段组成的数组。例如:

```
$segs = $this->uri->segment_array();

foreach ($segs as $segment)
{
    echo $segment;
    echo '<br />';
}
```

`rsegment_array()`

返回 Routed URI segments array

返回类型 array

当你使用 CodeIgniter 的 [URI 路由](#) 功能时, 该方法和 `segment_array()` 类似, 只是它用于返回路由后的 URI 的所有的段组成的数组。

8.1.28 用户代理类

用户代理 (User Agent) 类提供了一些方法来帮助你识别正在访问你的站点的浏览器、移动设备或机器人的信息。另外, 你还可以通过它获取 referrer 信息, 以及支持的语言和字符集信息。

- 使用用户代理类
 - 初始化类
 - 用户代理的定义
 - 例子
- 类参考

使用用户代理类

初始化类

正如 CodeIgniter 中的其他类一样, 在你的控制器中使用 `$this->load->library()` 方法来初始化用户代理类:

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

初始化之后, 用户代理类的对象就可以这样访问:

```
$this->agent
```

用户代理的定义

用户代理的名称定义在 `application/config/user_agents.php` 配置文件中。你也可以根据需要向相应的数组中添加你自己的用户代理。

例子

当用户代理类初始化之后，它会尝试判断正在访问你的站点的是 Web 浏览器，还是移动设备，或者是机器人。它还可以获取平台的相关信息。

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

if ($this->agent->is_browser())
{
    $agent = $this->agent->browser().' '.$this->agent->version();
}
elseif ($this->agent->is_robot())
{
    $agent = $this->agent->robot();
}
elseif ($this->agent->is_mobile())
{
    $agent = $this->agent->mobile();
}
else
{
    $agent = 'Unidentified User Agent';
}

echo $agent;

echo $this->agent->platform(); // Platform info (Windows, Linux, Mac, etc.)
```

类参考

class `CI_User_agent`

is_browser(`[$key = NULL]`)

参数

- **\$key** (*string*) – Optional browser name

返回 TRUE if the user agent is a (specified) browser, FALSE if not

返回类型 bool

判断用户代理是否为某个已知的 Web 浏览器，返回布尔值 TRUE 或 FALSE

。

```

if ($this->agent->is_browser('Safari'))
{
    echo 'You are using Safari.';
}
elseif ($this->agent->is_browser())
{
    echo 'You are using a browser.';
}

```

注解: 这个例子中的“Safari”字符串是配置文件中定义的 `browser` 数组的一个元素，你可以在 `application/config/user_agents.php` 文件中找到它，如果需要的话，你可以对其进行添加或修改。

`is_mobile([$key = NULL])`

参数

- **\$key** (*string*) – Optional mobile device name

返回 TRUE if the user agent is a (specified) mobile device, FALSE if not

返回类型 bool

判断用户代理是否为某个已知的移动设备，返回布尔值 TRUE 或 FALSE 。

```

if ($this->agent->is_mobile('iphone'))
{
    $this->load->view('iphone/home');
}
elseif ($this->agent->is_mobile())
{
    $this->load->view('mobile/home');
}
else
{
    $this->load->view('web/home');
}

```

`is_robot([$key = NULL])`

参数

- **\$key** (*string*) – Optional robot name

返回 TRUE if the user agent is a (specified) robot, FALSE if not

返回类型 bool

判断用户代理是否为某个已知的机器人，返回布尔值 TRUE 或 FALSE 。

注解: 用户代理类只定义了一些常见的机器人，它并不是完整的机器人列表，因为可能存在上百个不同的机器人，遍历这个列表效率会很低。如果你

发现某个机器人经常访问你的站点, 并且它不在这个列表中, 你可以将其添加到文件 `application/config/user_agents.php` 中。

is_referral()

返回 TRUE if the user agent is a referral, FALSE if not

返回类型 bool

判断用户代理是否为从另一个网站跳过来的 (Referer 为另一个网站), 返回布尔值 TRUE 或 FALSE 。

browser()

返回 Detected browser or an empty string

返回类型 string

返回当前正在浏览你的站点的浏览器名称。

version()

返回 Detected browser version or an empty string

返回类型 string

返回当前正在浏览你的站点的浏览器版本号。

mobile()

返回 Detected mobile device brand or an empty string

返回类型 string

返回当前正在浏览你的站点的移动设备名称。

robot()

返回 Detected robot name or an empty string

返回类型 string

返回当前正在浏览你的站点的机器人名称。

platform()

返回 Detected operating system or an empty string

返回类型 string

返回当前正在浏览你的站点的平台 (Linux、Windows、OSX 等)。

referrer()

返回 Detected referrer or an empty string

返回类型 string

如果用户代理引用了另一个站点, 返回 referrer 。一般你会像下面这样做:

```
if ($this->agent->is_referral())
{
    echo $this->agent->referrer();
}
```

agent_string()

返回 Full user agent string or an empty string

返回类型 string

返回完整的用户代理字符串，一般字符串的格式如下：

Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.0.4) Gecko/20060613 Camino

accept_lang([\$lang = 'en'])

参数

- \$lang (*string*) – Language key

返回 TRUE if provided language is accepted, FALSE if not

返回类型 bool

判断用户代理是否支持某个语言。例如：

```
if ($this->agent->accept_lang('en'))
{
    echo 'You accept English!';
}
```

注解： 这个方法一般不太可靠，因为有些浏览器并不提供语言信息，甚至在那些提供了语言信息的浏览器中，也并不是准确。

languages()

返回 An array list of accepted languages

返回类型 array

返回一个数组，包含用户代理支持的所有语言。

accept_charset([\$charset = 'utf-8'])

参数

- \$charset (*string*) – Character set

返回 TRUE if the character set is accepted, FALSE if not

返回类型 bool

判断用户代理是否支持某个字符集。例如：

```
if ($this->agent->accept_charset('utf-8'))
{
    // ...
}
```



```
        echo 'Your browser supports UTF-8!';
    }
```

注解: 这个方法一般不太可靠, 因为有些浏览器并不提供字符集信息, 甚至在那些提供了字符集信息的浏览器中, 也并不一定准确。

charsets()

返回 An array list of accepted character sets

返回类型 array

返回一个数组, 包含用户代理支持的所有字符集。

parse(\$string)

参数

- **\$string** (*string*) – A custom user-agent string

返回类型 void

解析一个自定义的用户代理字符串, 而不是当前正在访问站点的用户代理。

8.1.29 XML-RPC 与 XML-RPC 服务器类

CodeIgniter 的 XML-RPC 类允许你向另一个服务器发送请求, 或者建立一个你自己的 XML-RPC 服务器来接受请求。

- 什么是 XML-RPC ?
- 使用 XML-RPC 类
 - 初始化类
 - 发送 XML-RPC 请求
 - * 解释
 - 请求解析
 - 创建一个 XML-RPC 服务器
 - 处理服务器请求
 - * 注意
 - 格式化响应
 - 发送错误信息
 - 创建你自己的客户端与服务端
 - * 客户端
 - * 服务端
 - * 尝试一下
 - 在请求参数中使用关联数组
 - 数据类型
- 类参考

什么是 XML-RPC ?

这是一种在两台计算机之间使用 XML 通过互联网进行通信的简单方法。一台计算机, 我们称之为客户端, 发送一个 XML-RPC 请求给另外一台计算机, 我们称之为服务器, 当服务器收到请求时, 对其进行处理然后将结果返回给客户端。

例如, 使用 MetaWeblog API 时, XML-RPC 客户端 (通常是桌面发布工具) 将会发送请求到你站点上的 XML-RPC 服务器, 这个请求可能是发布一篇新博客, 或者编辑一篇已有的博客。当 XML-RPC 服务器收到该请求时, 它会决定使用哪个类和方法来处理该请求, 请求处理完成后, 服务器将发送一条回复消息。

关于 XML-RPC 的规范, 你可以查看 [XML-RPC](#) 的网站。

使用 XML-RPC 类

初始化类

跟 CodeIgniter 中的其他类一样, 可以在你的控制器中使用 `$this->load->library()` 方法加载 XML-RPC 类和 XML-RPC 服务器类。

加载 XML-RPC 类如下:

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

一旦加载, XML-RPC 类就可以像下面这样使用:

```
$this->xmlrpc
```

加载 XML-RPC 服务器类如下:

```
$this->load->library('xmlrpc');
$this->load->library('xmlrpcs');
```

一旦加载, XML-RPC 服务器类就可以像下面这样使用:

```
$this->xmlrpcs
```

注解: 当使用 XML-RPC 服务器类时, `xmlrpc` 和 `xmlrpcs` 都需要加载。

发送 XML-RPC 请求

向 XML-RPC 服务器发送一个请求, 你需要指定以下信息:

- 服务器的 URL
- 你想要调用的服务器方法
- **请求** 数据 (下面解释)

下面是个基本的例子, 向 [Ping-o-Matic](#) 发送一个简单的 Weblogs.com ping 请求。

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

$this->xmlrpc->server('http://rpc.pingomatic.com/', 80);
$this->xmlrpc->method('weblogUpdates.ping');

$request = array('My Photoblog', 'http://www.my-site.com/photoblog/');
$this->xmlrpc->request($request);

if ( ! $this->xmlrpc->send_request())
{
    echo $this->xmlrpc->display_error();
}
```

解释 上面的代码初始化了一个 XML-RPC 类，并设置了服务器 URL 和要调用的方法（weblogUpdates.ping）。然后通过 request() 方法编译请求，例子中请求是一个数组（标题和你网站的 URL）。最后，使用 send_request() 方法发送完整的请求。如果发送请求方法返回 FALSE，我们会显示出 XML-RPC 服务器返回的错误信息。

请求解析

XML-RPC 请求就是你发送给 XML-RPC 服务器的数据，请求中的每一个数据也被称为请求参数。上面的例子中有两个参数：你网站的 URL 和标题。当 XML-RPC 服务器收到请求后，它会查找它所需要的参数。

请求参数必须放在一个数组中，且数组中的每个参数都必须是 7 种数据类型中的一种（string、number、date 等），如果你的参数不是 string 类型，你必须在请求数组中指定它的数据类型。

下面是三个参数的简单例子：

```
$request = array('John', 'Doe', 'www.some-site.com');
$this->xmlrpc->request($request);
```

如果你的数据类型不是 string，或者你有几个不同类型的数据，那么你需要将每个参数放到它单独的数组中，并在数组的第二位声明其数据类型：

```
$request = array(
    array('John', 'string'),
    array('Doe', 'string'),
    array(FALSE, 'boolean'),
    array(12345, 'int')
);
$this->xmlrpc->request($request);
```

下面的数据类型一节列出了所有支持的数据类型。

创建一个 XML-RPC 服务器

XML-RPC 服务器扮演着类似于交通警察的角色，等待进入的请求，并将它们转到恰当的函数进行处理。

要创建你自己的 XML-RPC 服务器，你需要先在负责处理请求的控制器中初始化 XML-RPC 服务器类，然后设置一个映射数组，用于将请求转发到合适的类和方法，以便进行处理。

下面是个例子：

```
$this->load->library('xmlrpc');
$this->load->library('xmlrpcs');

$config['functions']['new_post'] = array('function' => 'My_blog.new_entry');
$config['functions']['update_post'] = array('function' => 'My_blog.update_entry');
$config['object'] = $this;

$this->xmlrpcs->initialize($config);
$this->xmlrpcs->serve();
```

上例中包含了两个服务器允许的请求方法，数组的左边是允许的方法名，数组的右边是当请求该方法时，将会映射到的类和方法。

其中，'object' 是个特殊的键，用于传递一个实例对象，当映射的方法无法使用 CodeIgniter 超级对象时，它将是必须的。

换句话说，如果 XML-RPC 客户端发送一个请求到 new_post 方法，你的服务器会加载 My_blog 类并调用 new_entry 函数。如果这个请求是到 update_post 方法的，那么你的服务器会加载 My_blog 类并调用 update_entry 方法。

上面例子中的函数名是任意的。你可以决定这些函数在你的服务器上叫什么名字，如果你使用的是标准的 API，比如 Blogger 或者 MetaWeblog 的 API，你必须使用标准的函数名。

这里还有两个附加的配置项，可以在服务器类初始化时配置使用。debug 设为 TRUE 以便调试，xss_clean 可被设置为 FALSE 以避免数据被安全类库的 xss_clean 函数过滤。

处理服务器请求

当 XML-RPC 服务器收到请求并加载类与方法来处理时，它会接收一个包含客户端发送的数据参数。

在上面的例子中，如果请求的是 new_post 方法，服务器请求的类与方法会像这样：

```
class My_blog extends CI_Controller {

    public function new_post($request)
    {
```

```
}  
}
```

`$request` 变量是一个由服务端汇集的对象, 包含由 XML-RPC 客户端发送来的数据。使用该对象可以让你访问到请求参数以便处理请求。请求处理完成后, 发送一个响应返回给客户端。

下面是一个实际的例子, 使用 Blogger API。Blogger API 中的一个方法是 `getUserInfo()`, XML-RPC 客户端可以使用该方法发送用户名和密码到服务器, 在服务器返回的数据中, 会包含该用户的信息 (昵称, 用户 ID, Email 地址等等)。下面是处理的代码:

```
class My_blog extends CI_Controller {  
  
    public function getUserInfo($request)  
    {  
        $username = 'smitty';  
        $password = 'secretsmittypass';  
  
        $this->load->library('xmlrpc');  
  
        $parameters = $request->output_parameters();  
  
        if ($parameters[1] != $username && $parameters[2] != $password)  
        {  
            return $this->xmlrpc->send_error_message('100', 'Invalid Access');  
        }  
  
        $response = array(  
            array(  
                'nickname' => array('Smitty', 'string'),  
                'userid'   => array('99', 'string'),  
                'url'      => array('http://yoursite.com', 'string'),  
                'email'    => array('jsmith@yoursite.com', 'string'),  
                'lastname' => array('Smith', 'string'),  
                'firstname' => array('John', 'string')  
            ),  
            'struct'  
        );  
  
        return $this->xmlrpc->send_response($response);  
    }  
}
```

注意 `output_parameters()` 函数获取一个由客户端发送的请求参数数组。上面的例子中输出参数将会是用户名和密码。

如果客户端发送的用户名和密码无效的话, 将使用 `send_error_message()` 函数返回错误信息。

如果操作成功, 客户端会收到包含用户信息的响应数组。

格式化响应

和请求一样，响应也必须被格式化为数组。然而不同于请求信息，响应数组 **只包含一项**。该项可以是一个包含其他数组的数组，但是只能有一个主数组，换句话说，响应的结果大概是下面这个样子：

```
$response = array('Response data', 'array');
```

但是，响应通常会包含多个信息。要做到这样，我们必须把各个信息放到他们自己的数组中，这样主数组就始终只有一个数据项。下面是一个例子展示如何实现这样的效果：

```
$response = array(
    array(
        'first_name' => array('John', 'string'),
        'last_name' => array('Doe', 'string'),
        'member_id' => array(123435, 'int'),
        'todo_list' => array(array('clean house', 'call mom', 'water plants'), 'array'),
    ),
    'struct'
);
```

注意：上面的数组被格式化为 struct，这是响应最常见的数据类型。

如同请求一样，响应可以是七种数据类型中的一种，参见[数据类型](#)一节。

发送错误信息

如果你需要发送错误信息给客户端，可以使用下面的代码：

```
return $this->xmlrpc->send_error_message('123', 'Requested data not available');
```

第一个参数为错误编号，第二个参数为错误信息。

创建你自己的客户端与服务端

为了帮助你理解目前为止讲的这些内容，让我们来创建两个控制器，演示下 XML-RPC 的客户端和服务端。你将用客户端来发送一个请求到服务端并从服务端收到一个响应。

客户端 使用文本编辑器创建一个控制器 Xmlrpc_client.php，在这个控制器中，粘贴以下的代码并保存到 applications/controllers/ 目录：

```
<?php

class Xmlrpc_client extends CI_Controller {

    public function index()
    {
```

```
$this->load->helper('url');
$server_url = site_url('xmlrpc_server');

$this->load->library('xmlrpc');

$this->xmlrpc->server($server_url, 80);
$this->xmlrpc->method('Greetings');

$request = array('How is it going?');
$this->xmlrpc->request($request);

if ( ! $this->xmlrpc->send_request())
{
    echo $this->xmlrpc->display_error();
}
else
{
    echo '<pre>';
    print_r($this->xmlrpc->display_response());
    echo '</pre>';
}
}
?>
```

注解: 上面的代码中我们使用了一个 URL 辅助函数, 更多关于辅助函数的信息, 你可以阅读[这里](#)。

服务端 使用文本编辑器创建一个控制器 Xmlrpc_server.php , 在这个控制器中, 粘贴以下的代码并保存到 applications/controllers/ 目录:

```
<?php

class Xmlrpc_server extends CI_Controller {

    public function index()
    {
        $this->load->library('xmlrpc');
        $this->load->library('xmlrpcs');

        $config['functions']['Greetings'] = array('function' => 'Xmlrpc_server.process');

        $this->xmlrpcs->initialize($config);
        $this->xmlrpcs->serve();
    }

    public function process($request)
    {
```

```

        $parameters = $request->output_parameters();

        $response = array(
            array(
                'you_said' => $parameters[0],
                'i_respond' => 'Not bad at all.'
            ),
            'struct'
        );

        return $this->xmlrpc->send_response($response);
    }
}

```

尝试一下 现在使用类似于下面这样的链接访问你的站点:

example.com/index.php/xmlrpc_client/

你应该能看到你发送到服务端的信息，以及服务器返回的响应信息。

在客户端，你发送了一条消息（"How's is going?"）到服务端，随着一个请求发送到 "Greetings" 方法。服务端收到这个请求并映射到 "process" 函数，然后返回响应信息。

在请求参数中使用关联数组

如果你希望在你的方法参数中使用关联数组，那么你需要使用 `struct` 数据类型:

```

$request = array(
    array(
        // Param 0
        array('name' => 'John'),
        'struct'
    ),
    array(
        // Param 1
        array(
            'size' => 'large',
            'shape' => 'round'
        ),
        'struct'
    )
);

$this->xmlrpc->request($request);

```

你可以在服务端处理请求信息时获取该关联数组。


```
$parameters = $request->output_parameters();  
$name = $parameters[0]['name'];  
$size = $parameters[1]['size'];  
$shape = $parameters[1]['shape'];
```

数据类型

根据 XML-RPC 规范 一共有七种不同的数据类型可以在 XML-RPC 中使用:

- *int* or *i4*
- *boolean*
- *string*
- *double*
- *dateTime.iso8601*
- *base64*
- *struct* (contains array of values)
- *array* (contains array of values)

类参考

class CI_Xmlrpc

```
initialize([$config = array()])
```

参数

- **\$config** (*array*) – Configuration data

返回类型 void

初始化 XML-RPC 类, 接受一个包含你设置的参数的关联数组。

```
server($url[, $port = 80[, $proxy = FALSE[, $proxy_port = 8080]]])
```

参数

- **\$url** (*string*) – XML-RPC server URL
- **\$port** (*int*) – Server port
- **\$proxy** (*string*) – Optional proxy
- **\$proxy_port** (*int*) – Proxy listening port

返回类型 void

用于设置 XML-RPC 服务器端的 URL 和端口:

```
$this->xmlrpc->server('http://www.sometimes.com/pings.php', 80);
```

支持基本的 HTTP 身份认证, 只需简单的将其添加到 URL 中:

```
$this->xmlrpc->server('http://user:pass@localhost/', 80);
```

```
timeout($seconds = 5)
```

参数

- **\$seconds** (*int*) – Timeout in seconds

返回类型 void

设置一个超时时间 (单位为秒), 超过该时间, 请求将被取消:

```
$this->xmlrpc->timeout(6);
```

```
method($function)
```

参数

- **\$function** (*string*) – Method name

返回类型 void

设置 XML-RPC 服务器接受的请求方法:

```
$this->xmlrpc->method('method');
```

其中 method 参数为请求方法名。

```
request($incoming)
```

参数

- **\$incoming** (*array*) – Request data

返回类型 void

接受一个数组参数, 并创建一个发送到 XML-RPC 服务器的请求:

```
$request = array(array('My Photoblog', 'string'), 'http://www.yoursite.com/photoblog');
$this->xmlrpc->request($request);
```

```
send_request()
```

返回 TRUE on success, FALSE on failure

返回类型 bool

发送请求的方法, 成功返回 TRUE, 失败返回 FALSE, 可以用在条件判断里。

```
display_error()
```

返回 Error message string

返回类型 string

当请求失败后, 返回错误信息。

```
echo $this->xmlrpc->display_error();
```

display_response()

返回 Response

返回类型 mixed

远程服务器接收请求后返回的响应, 返回的数据通常是一个关联数组。

```
$this->xmlrpc->display_response();
```

send_error_message(\$number, \$message)

参数

- **\$number** (*int*) – Error number
- **\$message** (*string*) – Error message

返回 XMLRPC_Response instance

返回类型 XMLRPC_Response

这个方法允许你从服务器发送一个错误消息到客户端。第一个参数是错误编号, 第二个参数是错误信息。

```
return $this->xmlrpc->send_error_message(123, 'Requested data not available');
```

8.1.30 Zip 编码类

CodeIgniter 的 Zip 编码类允许你创建 Zip 压缩文档, 文档可以被下载到你的桌面或者保存到某个文件夹里。

- 使用 Zip 编码类
 - 初始化类
 - 使用示例
- 类参考

使用 **Zip** 编码类

初始化类

正如 CodeIgniter 中的其他类一样, 在你的控制器中使用 `$this->load->library()` 方法来初始化 Zip 编码类:

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

初始化之后, Zip 编码类的对象就可以这样访问:

```
$this->zip
```

使用示例

下面这个例子演示了如何压缩一个文件，将其保存到服务器上的一个目录下，并下载到你的桌面上。

```
$name = 'mydata1.txt';
$data = 'A Data String!';

$this->zip->add_data($name, $data);

// Write the zip file to a folder on your server. Name it "my_backup.zip"
$this->zip->archive('/path/to/directory/my_backup.zip');

// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

类参考

class CI_Zip

\$compression_level = 2

使用的压缩等级。

压缩等级的范围为 0 到 9，9 为最高等级，0 为禁用压缩：

```
$this->zip->compression_level = 0;
```

add_data(\$filepath[, \$data = NULL])

参数

- **\$filepath** (*mixed*) – A single file path or an array of file => data pairs
- **\$data** (*array*) – File contents (ignored if \$filepath is an array)

返回类型 void

向 Zip 文档中添加数据，可以添加单个文件，也可以添加多个文件。

当添加单个文件时，第一个参数为文件名，第二个参数包含文件的内容：

```
$name = 'mydata1.txt';
$data = 'A Data String!';
$this->zip->add_data($name, $data);

$name = 'mydata2.txt';
$data = 'Another Data String!';
$this->zip->add_data($name, $data);
```

当添加多个文件时，第一个参数为包含 *file => contents* 这样的键值对的数组，第二个参数被忽略：

```
$data = array(
    'mydata1.txt' => 'A Data String!',
    'mydata2.txt' => 'Another Data String!'
);

$this->zip->add_data($data);
```

如果你想要将你压缩的数据组织到一个子目录下，只需简单的将文件路径包含到文件名中即可：

```
$name = 'personal/my_bio.txt';
$data = 'I was born in an elevator...';

$this->zip->add_data($name, $data);
```

上面的例子将会把 my_bio.txt 文件放到 personal 目录下。

add_dir(*\$directory*)

参数

- **\$directory** (*mixed*) – Directory name string or an array of multiple directories

返回类型 void

允许你往压缩文档中添加一个目录，通常这个方法是不必要的，因为你完全可以使用 `$this->zip->add_data()` 方法将你的数据添加到特定的目录下。但是如果你想创建一个空目录，你可以使用它：

```
$this->zip->add_dir('myfolder'); // Creates a directory called "myfolder"
```

read_file(*\$path*[, *\$archive_filepath* = FALSE])

参数

- **\$path** (*string*) – Path to file
- **\$archive_filepath** (*mixed*) – New file name/path (string) or (boolean) whether to maintain the original filepath

返回 TRUE on success, FALSE on failure

返回类型 bool

允许你压缩一个已经存在于你的服务器上的文件。该方法的参数为一个文件路径，Zip 类会读取该文件的内容并添加到压缩文档中：

```
$path = '/path/to/photo.jpg';

$this->zip->read_file($path);
```

```
// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

如果你希望 Zip 文档中的文件保持它原有的目录结构, 将第二个参数设置为布尔值 TRUE。例如:

```
$path = '/path/to/photo.jpg';

$this->zip->read_file($path, TRUE);

// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

在上面的例子中, photo.jpg 文件将会被放在 *path/to/* 目录下。

你也可以为新添加的文件指定一个新的名称 (包含文件路径):

```
$path = '/path/to/photo.jpg';
$new_path = '/new/path/some_photo.jpg';

$this->zip->read_file($path, $new_path);

// Download ZIP archive containing /new/path/some_photo.jpg
$this->zip->download('my_archive.zip');
```

```
read_dir($path[, $preserve_filepath = TRUE[, $root_path = NULL]])
```

参数

- **\$path** (*string*) – Path to directory
- **\$preserve_filepath** (*bool*) – Whether to maintain the original path
- **\$root_path** (*string*) – Part of the path to exclude from the archive directory

返回 TRUE on success, FALSE on failure

返回类型 bool

允许你压缩一个已经存在于你的服务器上的目录 (包括里面的内容)。该方法的参数为目录的路径, Zip 类会递归的读取它里面的内容并重建成一个 Zip 文档。指定目录下的所有文件以及子目录下的文件都会被压缩。例如:

```
$path = '/path/to/your/directory/';

$this->zip->read_dir($path);

// Download the file to your desktop. Name it "my_backup.zip"
$this->zip->download('my_backup.zip');
```

默认情况下, Zip 文档中会保留第一个参数中指定的目录结构, 如果你希望忽略掉这一大串的树形目录结构, 你可以将第二个参数设置为布尔值

FALSE 。例如:

```
$path = '/path/to/your/directory/';  
  
$this->zip->read_dir($path, FALSE);
```

上面的代码将会创建一个 Zip 文档, 文档里面直接是 “directory” 目录, 然后是它下面的所有的子目录, 不会包含 `/path/to/your` 路径在里面。

archive(\$filepath)

参数

- **\$filepath** (*string*) – Path to target zip archive

返回 TRUE on success, FALSE on failure

返回类型 bool

向你的服务器指定目录下写入一个 Zip 编码文档, 该方法的参数为一个有效的目录路径, 后加一个文件名, 确保这个目录是可写的 (权限为 755 通常就可以了)。例如:

```
$this->zip->archive('/path/to/folder/myarchive.zip'); // Creates a file named myar
```

download(\$filename = 'backup.zip')

参数

- **\$filename** (*string*) – Archive file name

返回类型 void

从你的服务器上下载 Zip 文档, 你需要指定 Zip 文档的名称。例如:

```
$this->zip->download('latest_stuff.zip'); // File will be named "latest_stuff.zip"
```

注解: 在调用这个方法的控制里不要显示任何数据, 因为这个方法会发送多个服务器 HTTP 头, 让文件以二进制的格式被下载。

get_zip()

返回 Zip file content

返回类型 string

返回使用 Zip 编码压缩后的文件数据, 通常情况你无需使用该方法, 除非你要对压缩后的数据做些特别的操作。例如:

```
$name = 'my_bio.txt';  
$data = 'I was born in an elevator...';  
  
$this->zip->add_data($name, $data);  
  
$zip_file = $this->zip->get_zip();
```

clear_data()

返回类型 void

Zip 类会缓存压缩后的数据，这样就不用在调用每个方法的时候重新压缩一遍了。但是，如果你需要创建多个 Zip 文档，每个 Zip 文档有着不同的数据，那么你可以在多次调用之间把缓存清除掉。例如：

```
$name = 'my_bio.txt';
$data = 'I was born in an elevator...';

$this->zip->add_data($name, $data);
$zip_file = $this->zip->get_zip();

$this->zip->clear_data();

$name = 'photo.jpg';
$this->zip->read_file("/path/to/photo.jpg"); // Read the file's contents

$this->zip->download('myphotos.zip');
```

数据库参考

9.1 数据库参考

CodeIgniter 内置了一个快速强大的数据库抽象类，支持传统的查询架构以及查询构造器模式。数据库方法的语法简单明了。

9.1.1 数据库快速入门：示例代码

这个页面包含的示例代码将简单介绍如何使用数据库类。更详细的信息请参考每个函数单独的介绍页面。

初始化数据库类

下面的代码将根据你的 [数据库配置](#) 加载并初始化数据库类：

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

数据库类一旦载入，你就可以像下面介绍的那样使用它。

注意：如果你所有的页面都需要连接数据库，你可以让其自动加载。参见 [数据库连接](#)。

多结果标准查询（对象形式）

```
$query = $this->db->query('SELECT name, title, email FROM my_table');

foreach ($query->result() as $row)
{
    echo $row->title;
    echo $row->name;
    echo $row->email;
}

echo 'Total Results: ' . $query->num_rows();
```

上面的 `result()` 函数返回一个 ** 对象数组 **。例如: `$row->title`

多结果标准查询 (数组形式)

```
$query = $this->db->query('SELECT name, title, email FROM my_table');

foreach ($query->result_array() as $row)
{
    echo $row['title'];
    echo $row['name'];
    echo $row['email'];
}
```

上面的 `result_array()` 函数返回一个 ** 数组的数组 **。例如: `$row['title']`

单结果标准查询 (对象形式)

```
$query = $this->db->query('SELECT name FROM my_table LIMIT 1');
$row = $query->row();
echo $row->name;
```

上面的 `row()` 函数返回一个 ** 对象 **。例如: `$row->name`

单结果标准查询 (数组形式)

```
$query = $this->db->query('SELECT name FROM my_table LIMIT 1');
$row = $query->row_array();
echo $row['name'];
```

上面的 `row_array()` 函数返回一个 ** 数组 **。例如: `$row['name']`

标准插入

```
$sql = "INSERT INTO mytable (title, name) VALUES (". $this->db->escape($title). ", ". $this->db->escape($name). ")";
$this->db->query($sql);
echo $this->db->affected_rows();
```

使用查询构造器查询数据

查询构造器模式 提供给我们一种简单的查询数据的途径:

```
$query = $this->db->get('table_name');

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

```

        echo $row->title;
    }

```

上面的 `get()` 函数从给定的表中查询出所有的结果。[查询构造器](#) 提供了所有数据库操作的快捷函数。

使用查询构造器插入数据

```

$data = array(
    'title' => $title,
    'name' => $name,
    'date' => $date
);

//
// 生成 SQL 语句:
//  INSERT INTO mytable (title, name, date) VALUES ('{$title}', '{$name}', '{$date}')
```

```

$this->db->insert('mytable', $data);

```

9.1.2 数据库配置

CodeIgniter 有一个配置文件用来保存数据库连接值（用户名、密码、数据库名等等），这个配置文件位于 `application/config/database.php`。你也可以放置不同的 `database.php` 文件到特定的环境配置文件夹里来设置[特定环境](#)的数据库连接值。

配置存放在一个多维数组里，原型如下：

```

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

```

有些数据库驱动（例如：PDO，PostgreSQL，Oracle，ODBC）可能需要提供完整的 DSN 字符串。在这种情况下，你需要使用 ‘dsn’ 配置参数，就好像使用该驱动的 PHP 原生扩展一样。例如：

```
// PDO
$db['default']['dsn'] = 'pgsql:host=localhost;port=5432;dbname=database_name';

// Oracle
$db['default']['dsn'] = '//localhost/XE';
```

注解： 如果你没有为需要 DSN 参数的驱动指定 DSN 字符串，CodeIgniter 将使用你提供的其他配置信息自动生成它。

注解： 如果你提供了一个 DSN 字符串，但是缺少了某些配置（例如：数据库的字符集），如果该配置存在其他的配置项中，CodeIgniter 将自动在 DSN 上附加上该配置。

当主数据库由于某些原因无法连接时，你还可以配置故障转移（failover）。可以像下面这样为一个连接配置故障转移：

```
$db['default']['failover'] = array(
    array(
        'hostname' => 'localhost1',
        'username' => '',
        'password' => '',
        'database' => '',
        'dbdriver' => 'mysqli',
        'dbprefix' => '',
        'pconnect' => TRUE,
        'db_debug' => TRUE,
        'cache_on' => FALSE,
        'cachedir' => '',
        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE
    ),
    array(
        'hostname' => 'localhost2',
        'username' => '',
        'password' => '',
        'database' => '',
        'dbdriver' => 'mysqli',
        'dbprefix' => '',
        'pconnect' => TRUE,
        'db_debug' => TRUE,
        'cache_on' => FALSE,
        'cachedir' => '',
```

```

        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE
    )
);

```

你可以指定任意多个故障转移。

我们使用多维数组的原因是为了让你随意的存储多个连接值的设置，例如：如果你有多个环境（开发、生产、测试等等），你能为每个环境建立独立的连接组，并在组之间进行切换。举个例子，如果要设置一个“test”环境，你可以这样做：

```

$db['test'] = array(
    'dsn' => '',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'database_name',
    'dbdriver' => 'mysqli',
    'dbprefix' => '',
    'pconnect' => TRUE,
    'db_debug' => TRUE,
    'cache_on' => FALSE,
    'cachedir' => '',
    'char_set' => 'utf8',
    'dbcollat' => 'utf8_general_ci',
    'swap_pre' => '',
    'compress' => FALSE,
    'encrypt' => FALSE,
    'stricton' => FALSE,
    'failover' => array()
);

```

然后，设置位于配置文件中的 `$active_group` 变量，告诉系统要使用“test”组：

```
$active_group = 'test';
```

注解： 分组的名称“test”是任意的，你可以取任意的名字。默认情况下，主连接使用“default”这个名称。当然，您可以基于您的项目为它起一个更有意义的名字。

查询构造器

可以通过数据库配置文件里的 `$query_builder` 变量对[查询构造器类](#)进行全局的设定（启用设成 TRUE，禁用设成 FALSE，默认是 TRUE）。如果你不用这个类，那么你可以通过将这个变量值设置成 FALSE 来减少在数据库类初始化时对电脑资源的消耗。

```
$query_builder = TRUE;
```

注解: 一些 CodeIgniter 的类, 例如 Sessions, 在执行一些函数的时候需要查询构造器的支持。

参数解释:

配置名	描述
dsn	DSN 连接字符串 (该字符串包含了所有的数据库配置信息)
hostname	数据库的主机名, 通常位于本机, 可以表示为 "localhost"
username	需要连接到数据库的用户名
password	登录数据库的密码
database	你需要连接的数据库名
dbdriver	数据库类型。如: mysql、postgres、odbc 等。必须为小写字母。
dbprefix	当使用 查询构造器 查询时, 可以选择性的为表加个前缀, 它允许在一个数据库上安装多个 CodeIgniter 程序。
pconnect	TRUE/FALSE (boolean) - 是否使用持续连接
db_debug	TRUE/FALSE (boolean) - 是否显示数据库错误信息
cache_on	TRUE/FALSE (boolean) - 是否开启数据库查询缓存, 详情请见 数据库缓存类 。
cachedir	数据库查询缓存目录所在的服务器绝对路径
char_set	与数据库通信时所使用的字符集
dbcollat	与数据库通信时所使用的字符规则
<hr/>	
	注解: 只使用于 'mysql' 和 'mysqli' 数据库驱动
<hr/>	
swap_pre	替换默认的 dbprefix 表前缀, 该项设置对于分布式应用是非常有用的, 你可以在查询中使用由最终用户定制的表前缀。
schema	数据库模式, 默认为 'public', 用于 PostgreSQL 和 ODBC 驱动
encrypt	是否使用加密连接。 <ul style="list-style-type: none">'mysql' (deprecated), 'sqlsrv' and 'pdo/sqlsrv' drivers accept TRUE/FALSE'mysqli' and 'pdo/mysql' drivers accept an array with the following options:<ul style="list-style-type: none">'ssl_key' - Path to the private key file'ssl_cert' - Path to the public key certificate file'ssl_ca' - Path to the certificate authority file'ssl_capath' - Path to a directory containing trusted CA certificates in PEM format'ssl_cipher' - List of <i>allowed</i> ciphers to be used for the encryption, separated by colons (':')

注解: 根据你使用的数据库平台 (MySQL, PostgreSQL 等), 并不是所有的参数都是必须的。例如, 当你使用 SQLite 时, 你无需指定用户名和密码, 数据库名称直接是你的数据库文件的路径。以上内容假设你使用的是 MySQL 数据库。

9.1.3 连接你的数据库

有两种方法连接数据库:

自动连接

“自动连接”特性将在每一个页面加载时自动实例化数据库类。要启用“自动连接”, 可在 `application/config/autoload.php` 中的 `library` 数组里添加 `database`:

```
$autoload['libraries'] = array('database');
```

手动连接

如果你只有一部分页面需要数据库连接, 你可以在那些有需要的函数里手工添加如下代码来连接数据库, 或者写在类的构造函数里, 让整个类都可以访问:

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

如果 `database()` 函数没有指定第一个参数, 它将使用数据库配置文件中指定的组连接数据库。对大多数人而言, 这是首选方案。

可用的参数

1. 数据库连接值, 用数组或 DSN 字符串传递;
2. TRUE/FALSE (boolean) - 是否返回连接 ID (参考下文的“连接多数据库”);
3. TRUE/FALSE (boolean) - 是否启用查询构造器类, 默认为 TRUE。

手动连接到数据库

这个函数的第一个参数是 **** 可选的 ****, 被用来从你的配置文件中指定一个特定的数据库组, 甚至可以使用没有在配置文件中定义的数据库连接值。下面是例子:

从你的配置文件中选择一个特定分组:

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

其中 `group_name` 是你的配置文件中连接组的名字。

连接一个完全手动指定的数据库, 可以传一个数组参数:

```
$config['hostname'] = 'localhost';
$config['username'] = 'myusername';
$config['password'] = 'mypassword';
$config['database'] = 'mydatabase';
$config['dbdriver'] = 'mysqli';
$config['dbprefix'] = '';
$config['pconnect'] = FALSE;
$config['db_debug'] = TRUE;
$config['cache_on'] = FALSE;
$config['cachedir'] = '';
$config['char_set'] = 'utf8';
$config['dbcollat'] = 'utf8_general_ci';
$this->load->database($config);
```

这些值的详细信息请参考: doc: 数据库配置 <configuration> 页面。

注解: 对于 PDO 驱动, 你应该使用 `$config['dsn']` 取代 ‘hostname’ 和 ‘database’ 参数:

```
$config['dsn'] = 'mysql:host=localhost;dbname=mydatabase';
```

或者你可以使用数据源名称 (DSN, Data Source Name) 作为参数, DSN 的格式必须类似于下面这样:

```
$dsn = 'dbdriver://username:password@hostname/database';
$this->load->database($dsn);
```

当用 DSN 字符串连接时, 要覆盖默认配置, 可以像添加查询字符串一样添加配置变量。

```
$dsn = 'dbdriver://username:password@hostname/database?char_set=utf8&dbcollat=utf8_general_ci';
$this->load->database($dsn);
```

连接到多个数据库

如果你需要同时连接到多个不同的数据库, 可以这样:

```
$DB1 = $this->load->database('group_one', TRUE);
$DB2 = $this->load->database('group_two', TRUE);
```

注意: 将 “group_one” 和 “group_two” 修改为你要连接的组名称 (或者像上面介绍的那样传入连接值数组)

第二个参数 TRUE 表示函数将返回数据库对象。

注解: 当你使用这种方式连接数据库时, 你将通过你的对象名来执行数据库命令, 而

不再是通过这份指南中通篇介绍的，就像下面这样的语法了：

```
$this->db->query();  
$this->db->result();  
etc...
```

取而代之的，你将这样执行数据库命令：

```
$DB1->query();  
$DB1->result();  
etc...
```

注解： 如果你只是需要切换到同一个连接的另一个不同的数据库，你没必要创建独立的数据库配置，你可以像下面这样切换到另一个数据库：

```
$this->db->db_select($database2_name);
```

重新连接/ 保持连接有效

当你在处理一些重量级的 PHP 操作时（例如处理图片），如果超过了数据库的超时值，你应该考虑在执行后续查询之前先调用 `reconnect()` 方法向数据库发送 ping 命令，这样可以优雅的保持连接有效或者重新建立起连接。

```
$this->db->reconnect();
```

手动关闭连接

虽然 CodeIgniter 可以智能的管理并自动关闭数据库连接，你仍可以用下面的方法显式的关闭连接：

```
$this->db->close();
```

9.1.4 查询

基本查询

常规查询

要提交一个查询, 使用 **query** 函数:

```
$this->db->query('YOUR QUERY HERE');
```

当你执行读类型的查询 (如: SELECT) 时, **query()** 函数将以 **** 对象 **** 形式返回一个结果集, 参考[这里来显示你的结果](#)。当你执行写类型的查询 (如: INSERT、DELETE、UPDATE) 时, 函数将简单的返回 TRUE 或 FALSE 来表示操作是否成功。你可以将函数返回的结果赋值给一个变量, 这样你就可以根据这个变量来获取数据了, 像下面这样:

```
$query = $this->db->query('YOUR QUERY HERE');
```

简化查询

simple_query 函数是 **\$this->db->query()** 的简化版。它不会返回查询的结果集, 不会去设置查询计数器, 不会去编译绑定的数据, 不会去存储查询的调试信息。它只是用于简单的提交一个查询, 大多数用户并不会用到这个函数。

simple_query 函数直接返回数据库驱动器的 “execute” 方法的返回值。对于写类型的查询 (如: INSERT、DELETE、UPDATE), 返回代表操作是否成功的 TRUE 或 FALSE; 而对于读类型的成功查询, 则返回代表结果集的对象。

```
if ($this->db->simple_query('YOUR QUERY'))
{
    echo "Success!";
}
else
{
    echo "Query failed!";
}
```

注解: 对于所有的查询, 如果成功执行的话, PostgreSQL 的 **pg_exec()** 函数都会返回一个结果集对象, 就算是写类型的查询也是这样。如果你想判断查询执行是否成功或失败, 请记住这一点。

指定数据库前缀

如果你配置了一个数据库前缀参数, 想把它加上你的 SQL 语句里的表名前面, 你可以调用下面的方法:

```
$this->db->dbprefix('tablename'); // outputs prefix_tablename
```

如果你想动态的修改这个前缀，而又不希望创建一个新的数据库连接，可以使用这个方法：

```
$this->db->set_dbprefix('newprefix');
$this->db->dbprefix('tablename'); // outputs newprefix_tablename
```

保护标识符

在很多数据库里，保护表名和字段名是可取的，例如在 MySQL 数据库里使用反引号。使用查询构造器会自动保护标识符，尽管如此，你还是可以像下面这样手工来保护：

```
$this->db->protect_identifiers('table_name');
```

重要： 尽管查询构造器会尽力保护好你输入的表名和字段名，但值得注意的是，它并不是被设计来处理任意用户输入的，所以，请不要传未处理的数据给它。

这个函数也可以为你的表名添加一个前缀，如果你在数据库配置文件中定义了 `dbprefix` 参数，通过将这个函数的第二个参数设置为 `TRUE` 来启用前缀：

```
$this->db->protect_identifiers('table_name', TRUE);
```

转义查询

在提交数据到你的数据库之前，确保先对其进行转义是个非常不错的做法。CodeIgniter 有三个方法来帮你做到这一点：

1. `$this->db->escape()` 这个函数会检测数据类型，仅转义字符串类型的数据。它会自动用单引号将你的数据括起来，你不用手动添加：

```
$sql = "INSERT INTO table (title) VALUES('.$this->db->escape($title).')";
```

2. `$this->db->escape_str()` 这个函数忽略数据类型，对传入的数据进行转义，这个方法并不常用，一般情况都是使用上面的那个方法。方法的使用代码如下：

```
$sql = "INSERT INTO table (title) VALUES('.$this->db->escape_str($title).')";
```

3. `$this->db->escape_like_str()` 这个函数用于处理 `LIKE` 语句中的字符串，这样，`LIKE` 通配符（`'%'`，`'_'`）可以被正确的转义。

```
$search = '20% raise';
$sql = "SELECT id FROM table WHERE column LIKE '%" .
    $this->db->escape_like_str($search)."%";
```

查询绑定

查询绑定可以简化你的查询语法, 它通过系统自动的为你将各个查询组装在一起。参考下面的例子:

```
$sql = "SELECT * FROM some_table WHERE id = ? AND status = ? AND author = ?";
$this->db->query($sql, array(3, 'live', 'Rick'));
```

查询语句中的问号将会自动被第二个参数位置的数组的相应的值替代。

也可以使用数组的数组进行绑定, 里面的数组会被转换成 IN 语句的集合:

```
$sql = "SELECT * FROM some_table WHERE id IN ? AND status = ? AND author = ?";
$this->db->query($sql, array(array(3, 6), 'live', 'Rick'));
```

上面的例子会被转换为这样的查询:

```
SELECT * FROM some_table WHERE id IN (3,6) AND status = 'live' AND author = 'Rick'
```

使用查询绑定的第二个好处是: 所有的值会被自动转义, 生成安全的查询语句。你不再需要手工进行转义, 系统会自动进行。

错误处理

```
$this->db->error();
```

要获取最近一次发生的错误, 使用 `error()` 方法可以得到一个包含错误代码和错误消息的数组。这里是一个简单例子:

```
if ( ! $this->db->simple_query('SELECT `example_field` FROM `example_table`'))
{
    $error = $this->db->error(); // Has keys 'code' and 'message'
}
```

9.1.5 生成查询结果

有几种不同方法可以生成查询结果:

- 结果数组
- 结果行
- 自定义结果对象
- 结果辅助方法
- Class Reference

结果数组

`result()` 方法

该方法以 **** 对象数组 **** 形式返回查询结果, 如果查询失败返回 **** 空数组 ****。一般情况下, 你会像下面这样在一个 foreach 循环中使用它:

```
$query = $this->db->query("YOUR QUERY");

foreach ($query->result() as $row)
{
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

该方法是 `result_object()` 方法的别名。

你还可以传一个字符串参数给 `result()` 方法, 这个字符串参数代表你想把每个结果转换成某个类的类名 (这个类必须已经加载)

```
$query = $this->db->query("SELECT * FROM users;");

foreach ($query->result('User') as $user)
{
    echo $user->name; // access attributes
    echo $user->reverse_name(); // or methods defined on the 'User' class
}
```

`result_array()` 方法

这个方法以 **一个纯粹的数组** 形式返回查询结果, 如果无结果, 则返回一个空数组。一般情况下, 你会像下面这样在一个 foreach 循环中使用它:

```
$query = $this->db->query("YOUR QUERY");

foreach ($query->result_array() as $row)
{
    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

结果行

`row()` 方法

这个方法返回单独一行结果。如果你的查询不止一行结果, 它只返回第一行。返回的结果是 **对象** 形式, 这里是用例:

```
$query = $this->db->query("YOUR QUERY");

$row = $query->row();

if (isset($row))
{
```



```
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

如果你要返回特定行的数据, 你可以将行号作为第一个参数传给这个方法:

```
$row = $query->row(5);
```

你还可以加上第二个参数, 该参数为字符串类型, 代表你想要把结果转换成某个类的类名:

```
$query = $this->db->query("SELECT * FROM users LIMIT 1;");
$row = $query->row(0, 'User');

echo $row->name; // access attributes
echo $row->reverse_name(); // or methods defined on the 'User' class
```

row_array() 方法

这个方法除了返回结果是一个数组而不是一个对象之外, 其他的和上面的 `row()` 方法完全一样。举例:

```
$query = $this->db->query("YOUR QUERY");

$row = $query->row_array();

if (isset($row))
{
    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

如果你要返回特定行的数据, 你可以将行号作为第一个参数传给这个方法:

```
$row = $query->row_array(5);
```

另外, 你可以使用下面这些方法从你的结果集中获取前一个、后一个、第一个或者最后一个结果:

```
$row = $query->first_row()
$row = $query->last_row()
$row = $query->next_row()
$row = $query->previous_row()
```

这些方法默认返回对象, 如果需要返回数组形式, 将单词 “array” 作为参数传入方法即可:

```
$row = $query->first_row('array')
$row = $query->last_row('array')
$row = $query->next_row('array')
```

```
$row = $query->previous_row('array')
```

注解: 上面所有的这些方法都会把所有的结果加载到内存里 (预读取), 当处理大结果集时最好使用 `unbuffered_row()` 方法。

`unbuffered_row()` 方法

这个方法和 `row()` 方法一样返回单独一行结果, 但是它不会预读取所有的结果数据到内存中。如果你的查询结果不止一行, 它将返回当前一行, 并通过内部实现的指针来移动到下一行。

```
$query = $this->db->query("YOUR QUERY");
```

```
while ($row = $query->unbuffered_row())
{
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

为了指定返回值的类型, 可以传一个字符串参数 'object' (默认值) 或者 'array' 给这个方法:

```
$query->unbuffered_row();           // object
$query->unbuffered_row('object');   // object
$query->unbuffered_row('array');    // associative array
```

自定义结果对象

You can have the results returned as an instance of a custom class instead of a `stdClass` or array, as the `result()` and `result_array()` methods allow. This requires that the class is already loaded into memory. The object will have all values returned from the database set as properties. If these have been declared and are non-public then you should provide a `__set()` method to allow them to be set.

Example:

```
class User {

    public $id;
    public $email;
    public $username;

    protected $last_login;

    public function last_login($format)
    {
        return $this->last_login->format($format);
    }

    public function __set($name, $value)
```

```
{
    if ($name === 'last_login')
    {
        $this->last_login = DateTime::createFromFormat('U', $value);
    }
}

public function __get($name)
{
    if (isset($this->$name))
    {
        return $this->$name;
    }
}
}
```

In addition to the two methods listed below, the following methods also can take a class name to return the results as: `first_row()`, `last_row()`, `next_row()`, and `previous_row()`.

custom_result_object()

Returns the entire result set as an array of instances of the class requested. The only parameter is the name of the class to instantiate.

Example:

```
$query = $this->db->query("YOUR QUERY");

$rows = $query->custom_result_object('User');

foreach ($rows as $row)
{
    echo $row->id;
    echo $row->email;
    echo $row->last_login('Y-m-d');
}
```

custom_row_object()

Returns a single row from your query results. The first parameter is the row number of the results. The second parameter is the class name to instantiate.

Example:

```
$query = $this->db->query("YOUR QUERY");

$row = $query->custom_row_object(0, 'User');

if (isset($row))
{
    echo $row->email;    // access attributes
    echo $row->last_login('Y-m-d');    // access class methods
}
```

```
}
```

You can also use the `row()` method in exactly the same way.

Example:

```
$row = $query->custom_row_object(0, 'User');
```

结果辅助方法

`num_rows()` 方法

该方法返回查询结果的行数。注意：在这个例子中，`$query` 变量为查询结果对象：

```
$query = $this->db->query('SELECT * FROM my_table');
```

```
echo $query->num_rows();
```

注解： 并不是所有的数据库驱动器都有原生的方法来获取查询结果的总行数。当遇到这种情况时，所有的数据会被预读取到内存中，并调用 `count()` 函数来取得总行数。

`num_fields()` 方法

该方法返回查询结果的字段数（列数）。在你的查询结果对象上调用该方法：

```
$query = $this->db->query('SELECT * FROM my_table');
```

```
echo $query->num_fields();
```

`free_result()` 方法

该方法释放掉查询结果所占的内存，并删除结果的资源标识。通常来说，PHP 会在脚本执行结束后自动释放内存。但是，如果你在某个脚本中执行大量的查询，你可能需要在每次查询之后释放掉查询结果，以此来降低内存消耗。

举例：

```
$query = $this->db->query('SELECT title FROM my_table');
```

```
foreach ($query->result() as $row)
{
    echo $row->title;
}
```

```
$query->free_result(); // The $query result object will no longer be available
```

```
$query2 = $this->db->query('SELECT name FROM some_table');
```

```
$row = $query2->row();
```

```
echo $row->name;
```

```
$query2->free_result(); // The $query2 result object will no longer be available
```

`data_seek()` 方法

这个方法用来设置下一个结果行的内部指针，它只有在和 `unbuffered_row()` 方法一起使用才有效果。

它接受一个正整数参数（默认值为 0）表示想要读取的下一行，返回值为 TRUE 或 FALSE 表示成功或失败。

```
$query = $this->db->query('SELECT `field_name` FROM `table_name`');
$query->data_seek(5); // Skip the first 5 rows
$row = $query->unbuffered_row();
```

注解：并不是所有的数据库驱动器都支持这一特性，调用这个方法将会返回 FALSE，例如你无法在 PDO 上使用它。

Class Reference

class `CI_DB_result`

`result([$type = 'object'])`

参数

- **`$type`** (*string*) – Type of requested results - array, object, or class name

返回 Array containing the fetched rows

返回类型 array

A wrapper for the `result_array()`, `result_object()` and `custom_result_object()` methods.

Usage: see [结果数组](#).

`result_array()`

返回 Array containing the fetched rows

返回类型 array

Returns the query results as an array of rows, where each row is itself an associative array.

Usage: see [结果数组](#).

`result_object()`

返回 Array containing the fetched rows

返回类型 array

Returns the query results as an array of rows, where each row is an object of type `stdClass`.

Usage: see [结果数组](#).

custom_result_object(\$class_name)

参数

- **\$class_name** (*string*) – Class name for the resulting rows

返回 Array containing the fetched rows

返回类型 array

Returns the query results as an array of rows, where each row is an instance of the specified class.

row(\$n = 0, \$type = 'object')

参数

- **\$n** (*int*) – Index of the query results row to be returned
- **\$type** (*string*) – Type of the requested result - array, object, or class name

返回 The requested row or NULL if it doesn't exist

返回类型 mixed

A wrapper for the `row_array()`, `row_object()` and `custom_row_object()` methods.

Usage: see [结果行](#).

unbuffered_row(\$type = 'object')

参数

- **\$type** (*string*) – Type of the requested result - array, object, or class name

返回 Next row from the result set or NULL if it doesn't exist

返回类型 mixed

Fetches the next result row and returns it in the requested form.

Usage: see [结果行](#).

row_array(\$n = 0)

参数

- **\$n** (*int*) – Index of the query results row to be returned

返回 The requested row or NULL if it doesn't exist

返回类型 array

Returns the requested result row as an associative array.

Usage: see [结果行](#).

`row_object($n = 0)`

参数

- **\$n** (*int*) – Index of the query results row to be returned :returns:
The requested row or NULL if it doesn't exist

返回类型 stdClass

Returns the requested result row as an object of type `stdClass`.

Usage: see [结果行](#).

`custom_row_object($n, $type)`

参数

- **\$n** (*int*) – Index of the results row to return
- **\$class_name** (*string*) – Class name for the resulting row

返回 The requested row or NULL if it doesn't exist

返回类型 \$type

Returns the requested result row as an instance of the requested class.

`data_seek($n = 0)`

参数

- **\$n** (*int*) – Index of the results row to be returned next

返回 TRUE on success, FALSE on failure

返回类型 bool

Moves the internal results row pointer to the desired offset.

Usage: see [结果辅助方法](#).

`set_row($key[, $value = NULL])`

参数

- **\$key** (*mixed*) – Column name or array of key/value pairs
- **\$value** (*mixed*) – Value to assign to the column, \$key is a single field name

返回类型 void

Assigns a value to a particular column.

`next_row($type = 'object')`

参数

- **\$type** (*string*) – Type of the requested result - array, object, or class name

返回 Next row of result set, or NULL if it doesn't exist

返回类型 mixed

Returns the next row from the result set.

`previous_row($type = 'object')`

参数

- **\$type** (*string*) – Type of the requested result - array, object, or class name

返回 Previous row of result set, or NULL if it doesn't exist

返回类型 mixed

Returns the previous row from the result set.

`first_row($type = 'object')`

参数

- **\$type** (*string*) – Type of the requested result - array, object, or class name

返回 First row of result set, or NULL if it doesn't exist

返回类型 mixed

Returns the first row from the result set.

`last_row($type = 'object')`

参数

- **\$type** (*string*) – Type of the requested result - array, object, or class name

返回 Last row of result set, or NULL if it doesn't exist

返回类型 mixed

Returns the last row from the result set.

`num_rows()`

返回 Number of rows in the result set

返回类型 int

Returns the number of rows in the result set.

Usage: see [结果辅助方法](#).

`num_fields()`

返回 Number of fields in the result set

返回类型 int

Returns the number of fields in the result set.

Usage: see [结果辅助方法](#).

field_data()

返回 Array containing field meta-data

返回类型 array

Generates an array of stdClass objects containing field meta-data.

free_result()

返回类型 void

Frees a result set.

Usage: see [结果辅助方法](#).

list_fields()

返回 Array of column names

返回类型 array

Returns an array containing the field names in the result set.

9.1.6 查询辅助函数

关于执行查询的信息

`$this->db->insert_id()`

当执行 INSERT 语句时, 这个方法返回新插入行的 ID。

注解: 如果你使用的是 PostgreSQL 的 PDO 驱动器, 或者 Interbase 驱动器, 这个方法需要一个 `$name` 参数来指定合适的顺序。(什么意思?)

`$this->db->affected_rows()`

当执行 INSERT、UPDATE 等写类型的语句时, 这个方法返回受影响的行数。

注解: 在 MySQL 中执行 “DELETE FROM TABLE” 语句返回受影响的行数为 0。为了让这个方法返回正确的受影响行数, 数据库类对此做了一点小 hack。默认情况下, 这个 hack 是启用的, 你可以在数据库驱动文件中关闭它。

`$this->db->last_query()`

该方法返回上一次执行的查询语句 (是查询语句, 不是结果)。举例:

```
$str = $this->db->last_query();
```

```
// Produces: SELECT * FROM sometable....
```

注解: 将数据库配置文件中的 `save_queries` 设置为 FALSE 可以让这个方法无效。

关于数据库的信息

`$this->db->count_all()`

该方法用于获取数据表的总行数，第一个参数为表名，例如：

```
echo $this->db->count_all('my_table');
```

// Produces an integer, like 25

`$this->db->platform()`

该方法输出你正在使用的数据库平台（MySQL，MS SQL，Postgres 等）：

```
echo $this->db->platform();
```

`$this->db->version()`

该方法输出你正在使用的数据库版本：

```
echo $this->db->version();
```

让你的查询更简单

`$this->db->insert_string()`

这个方法简化了 INSERT 语句的书写，它返回一个正确格式化的 INSERT 语句。
举例：

```
$data = array('name' => $name, 'email' => $email, 'url' => $url);
```

```
$str = $this->db->insert_string('table_name', $data);
```

第一个参数为表名，第二个参数是一个关联数组，表示待插入的数据。上面的例子生成的 SQL 语句如下：

```
INSERT INTO table_name (name, email, url) VALUES ('Rick', 'rick@example.com', 'example.com');
```

注解：所有的值自动被转义，生成安全的查询语句。

`$this->db->update_string()`

这个方法简化了 UPDATE 语句的书写，它返回一个正确格式化的 UPDATE 语句。
举例：

```
$data = array('name' => $name, 'email' => $email, 'url' => $url);
```

```
$where = "author_id = 1 AND status = 'active'";
```

```
$str = $this->db->update_string('table_name', $data, $where);
```

第一个参数是表名，第二个参数是一个关联数组，表示待更新的数据，第三个参数是个 WHERE 子句。上面的例子生成的 SQL 语句如下：

```
UPDATE table_name SET name = 'Rick', email = 'rick@example.com', url = 'example.com' WHERE a
```

注解: 所有的值自动被转义, 生成安全的查询语句。

9.1.7 查询构造器类

CodeIgniter 提供了查询构造器类, 查询构造器允许你使用较少的代码来在数据库中获取、新增或更新数据。有时只需要一两行代码就能完成数据库操作。CodeIgniter 并不需要为每个数据表提供一个类, 而是使用了一种更简单的接口。

除了简单, 使用查询构造器的另一个好处是可以让你创建数据库独立的应用程序, 这是因为查询语句是由每个独立的数据库适配器生成的。另外, 由于系统会自动对数据进行转义, 所以它还能提供更安全的查询。

注解: 如果你想要编写你自己的查询语句, 你可以在数据库配置文件中禁用这个类, 这样数据库核心类库和适配器将使用更少的资源。

- 查询
- 搜索
- 模糊搜索
- 排序
- 分页与计数
- 查询条件组
- 插入数据
- 更新数据
- 删除数据
- 链式方法
- 查询构造器缓存
- 重置查询构造器
- Class Reference

查询

下面的方法用来构建 **SELECT** 语句。

```
$this->db->get()
```

该方法执行 **SELECT** 语句并返回查询结果, 可以得到一个表的所有数据:

```
$query = $this->db->get('mytable'); // Produces: SELECT * FROM mytable
```

第二和第三个参数用于设置 **LIMIT** 子句:

```
$query = $this->db->get('mytable', 10, 20);
```

```
// Executes: SELECT * FROM mytable LIMIT 20, 10
// (in MySQL. Other databases have slightly different syntax)
```

你应该已经注意到了, 上面的方法的结果都赋值给了一个 `$query` 变量, 通过这个变量, 我们可以得到查询的结果:

```
$query = $this->db->get('mytable');

foreach ($query->result() as $row)
{
    echo $row->title;
}
```

参考[生成查询结果](#) 页面获取关于生成结果的更多信息。

`$this->db->get_compiled_select()`

该方法和 `$this->db->get()` 方法一样编译 SELECT 查询并返回查询的 SQL 语句, 但是, 该方法并不执行它。

例子:

```
$sql = $this->db->get_compiled_select('mytable');
echo $sql;
```

```
// Prints string: SELECT * FROM mytable
```

第二个参数用于设置是否重置查询 (默认会重置, 和使用 `$this->db->get()` 方法时一样) :

```
echo $this->db->limit(10,20)->get_compiled_select('mytable', FALSE);
```

```
// Prints string: SELECT * FROM mytable LIMIT 20, 10
// (in MySQL. Other databases have slightly different syntax)
```

```
echo $this->db->select('title, content, date')->get_compiled_select();
```

```
// Prints string: SELECT title, content, date FROM mytable LIMIT 20, 10
```

上面的例子中, 最值得注意的是, 第二个查询并没有用到 `$this->db->from()` 方法, 也没有为查询指定表名参数, 但是它生成的 SQL 语句中有 FROM mytable 子句。这是因为查询并没有被重置 (使用 `$this->db->get()` 方法查询会被执行并被重置, 使用 `$this->db->reset_query()` 方法直接重置)。

`$this->db->get_where()`

这个方法基本上和上面的方法一样, 但它提供了第二个参数可以让你添加一个 WHERE 子句, 而不是使用 `db->where()` 方法:

```
$query = $this->db->get_where('mytable', array('id' => $id), $limit, $offset);
```

阅读下面的 `db->where()` 方法获取更多信息。

注解: `get_where()` 方法的前身为 `getwhere()`, 已废除

`$this->db->select()`

该方法用于编写查询语句中的 SELECT 子句:

```
$this->db->select('title, content, date');  
$query = $this->db->get('mytable');  
  
// Executes: SELECT title, content, date FROM mytable
```

注解: 如果你要查询表的所有列, 可以不用写这个函数, CodeIgniter 会自动查询所有列 (SELECT *).

`$this->db->select()` 方法的第二个参数可选, 如果设置为 FALSE, CodeIgniter 将不保护你的表名和字段名, 这在当你编写复合查询语句时很有用, 不会破坏你编写的语句。

```
$this->db->select('(SELECT SUM(payments.amount) FROM payments WHERE payments.invoice_id=4')  
$query = $this->db->get('mytable');
```

`$this->db->select_max()`

该方法用于编写查询语句中的 SELECT MAX(field) 部分, 你可以使用第二个参数 (可选) 重命名结果字段。

```
$this->db->select_max('age');  
$query = $this->db->get('members'); // Produces: SELECT MAX(age) as age FROM members  
  
$this->db->select_max('age', 'member_age');  
$query = $this->db->get('members'); // Produces: SELECT MAX(age) as member_age FROM members
```

`$this->db->select_min()`

该方法用于编写查询语句中的 SELECT MIN(field) 部分, 和 select_max() 方法一样, 你可以使用第二个参数 (可选) 重命名结果字段。

```
$this->db->select_min('age');  
$query = $this->db->get('members'); // Produces: SELECT MIN(age) as age FROM members
```

`$this->db->select_avg()`

该方法用于编写查询语句中的 SELECT AVG(field) 部分, 和 select_max() 方法一样, 你可以使用第二个参数 (可选) 重命名结果字段。

```
$this->db->select_avg('age');  
$query = $this->db->get('members'); // Produces: SELECT AVG(age) as age FROM members
```

`$this->db->select_sum()`

该方法用于编写查询语句中的 SELECT SUM(field) 部分, 和 select_max() 方法一样, 你可以使用第二个参数 (可选) 重命名结果字段。

```
$this->db->select_sum('age');
$query = $this->db->get('members'); // Produces: SELECT SUM(age) as age FROM members
```

`$this->db->from()`

该方法用于编写查询语句中的 FROM 子句:

```
$this->db->select('title, content, date');
$this->db->from('mytable');
$query = $this->db->get(); // Produces: SELECT title, content, date FROM mytable
```

注解: 正如前面所说, 查询中的 FROM 部分可以在方法 `$this->db->get()` 中指定, 所以, 你可以选择任意一种你喜欢的方式。

`$this->db->join()`

该方法用于编写查询语句中的 JOIN 子句:

```
$this->db->select('*');
$this->db->from('blogs');
$this->db->join('comments', 'comments.id = blogs.id');
$query = $this->db->get();

// Produces:
// SELECT * FROM blogs JOIN comments ON comments.id = blogs.id
```

如果你的查询中有多个连接, 你可以多次调用这个方法。

你可以传入第三个参数指定连接的类型, 有这样几种选择: left, right, outer, inner, left outer 和 right outer。

```
$this->db->join('comments', 'comments.id = blogs.id', 'left');
// Produces: LEFT JOIN comments ON comments.id = blogs.id
```

搜索

`$this->db->where()`

该方法提供了 4 中方式让你编写查询语句中的 WHERE 子句:

注解: 所有的数据将会自动转义, 生成安全的查询语句。

1. 简单的 key/value 方式:

```
$this->db->where('name', $name); // Produces: WHERE name = 'Joe'
```

注意自动为你加上了等号。

如果你多次调用该方法, 那么多个 WHERE 条件将会使用 AND 连接起来:

```
$this->db->where('name', $name);
$this->db->where('title', $title);
$this->db->where('status', $status);
// WHERE name = 'Joe' AND title = 'boss' AND status = 'active'
```

2. 自定义 key/value 方式:

为了控制比较, 你可以在第一个参数中包含一个比较运算符:

```
$this->db->where('name !=', $name);
$this->db->where('id <', $id); // Produces: WHERE name != 'Joe' AND id < 45
```

3. 关联数组方式:

```
$array = array('name' => $name, 'title' => $title, 'status' => $status);
$this->db->where($array);
// Produces: WHERE name = 'Joe' AND title = 'boss' AND status = 'active'
```

你也可以在这个方法里包含你自己的比较运算符:

```
$array = array('name !=' => $name, 'id <' => $id, 'date >' => $date);
$this->db->where($array);
```

4. 自定义字符串:

你可以完全手工编写 WHERE 子句:

```
$where = "name='Joe' AND status='boss' OR status='active'";
$this->db->where($where);
```

`$this->db->where()` 方法有一个可选的第三个参数, 如果设置为 `FALSE`, CodeIgniter 将不保护你的表名和字段名。

```
$this->db->where('MATCH (field) AGAINST ("value")', NULL, FALSE);
```

`$this->db->or_where()`

这个方法和上面的方法一样, 只是多个 WHERE 条件之间使用 OR 进行连接:

```
$this->db->where('name !=', $name);
$this->db->or_where('id >', $id); // Produces: WHERE name != 'Joe' OR id > 50
```

注解: `or_where()` 方法的前身为 `orWhere()`, 已废除

`$this->db->where_in()`

该方法用于生成 WHERE IN 子句, 多个子句之间使用 AND 连接

```
$names = array('Frank', 'Todd', 'James');
$this->db->where_in('username', $names);
// Produces: WHERE username IN ('Frank', 'Todd', 'James')
```

`$this->db->or_where_in()`

该方法用于生成 WHERE IN 子句, 多个子句之间使用 OR 连接


```
$names = array('Frank', 'Todd', 'James');
$this->db->or_where_in('username', $names);
// Produces: OR username IN ('Frank', 'Todd', 'James')
```

\$this->db->where_not_in()

该方法用于生成 WHERE NOT IN 子句, 多个子句之间使用 AND 连接

```
$names = array('Frank', 'Todd', 'James');
$this->db->where_not_in('username', $names);
// Produces: WHERE username NOT IN ('Frank', 'Todd', 'James')
```

\$this->db->or_where_not_in()

该方法用于生成 WHERE NOT IN 子句, 多个子句之间使用 OR 连接

```
$names = array('Frank', 'Todd', 'James');
$this->db->or_where_not_in('username', $names);
// Produces: OR username NOT IN ('Frank', 'Todd', 'James')
```

模糊搜索

\$this->db->like()

该方法用于生成 LIKE 子句, 在进行搜索时非常有用。

注解: 所有数据将会自动被转义。

1. 简单 key/value 方式:

```
$this->db->like('title', 'match');
// Produces: WHERE `title` LIKE '%match%' ESCAPE '!'
```

如果你多次调用该方法, 那么多个 WHERE 条件将会使用 AND 连接起来:

```
$this->db->like('title', 'match');
$this->db->like('body', 'match');
// WHERE `title` LIKE '%match%' ESCAPE '!' AND `body` LIKE '%match%' ESCAPE '!'
```

可以传入第三个可选的参数来控制 LIKE 通配符 (%) 的位置, 可用选项有: 'before', 'after' 和 'both' (默认为 'both')。

```
$this->db->like('title', 'match', 'before'); // Produces: WHERE `title` LIKE '%match%' ESCAPE '!'
$this->db->like('title', 'match', 'after'); // Produces: WHERE `title` LIKE '%match%' ESCAPE '!'
$this->db->like('title', 'match', 'both'); // Produces: WHERE `title` LIKE '%match%' ESCAPE '!'
```

2. 关联数组方式:

```
$array = array('title' => $match, 'page1' => $match, 'page2' => $match);
$this->db->like($array);
// WHERE `title` LIKE '%match%' ESCAPE '!' AND `page1` LIKE '%match%' ESCAPE '!'
```


\$this->db->or_like()

这个方法和上面的方法一样, 只是多个 WHERE 条件之间使用 OR 进行连接:

```
$this->db->like('title', 'match'); $this->db->or_like('body', $match);  
// WHERE `title` LIKE '%match%' ESCAPE '!' OR `body` LIKE '%match%' ESCAPE '!'
```

注解: or_like() 方法的前身为 orlike(), 已废除

\$this->db->not_like()

这个方法和 like() 方法一样, 只是生成 NOT LIKE 子句:

```
$this->db->not_like('title', 'match'); // WHERE `title` NOT LIKE '%match%' ESCAPE '!'
```

\$this->db->or_not_like()

这个方法和 not_like() 方法一样, 只是多个 WHERE 条件之间使用 OR 进行连接:

```
$this->db->like('title', 'match');  
$this->db->or_not_like('body', 'match');  
// WHERE `title` LIKE '%match%' OR `body` NOT LIKE '%match%' ESCAPE '!'
```

\$this->db->group_by()

该方法用于生成 GROUP BY 子句:

```
$this->db->group_by("title"); // Produces: GROUP BY title
```

你也可以通过一个数组传入多个值:

```
$this->db->group_by(array("title", "date")); // Produces: GROUP BY title, date
```

注解: group_by() 方法前身为 groupby(), 已废除

\$this->db->distinct()

该方法用于向查询中添加 DISTINCT 关键字:

```
$this->db->distinct();  
$this->db->get('table'); // Produces: SELECT DISTINCT * FROM table
```

\$this->db->having()

该方法用于生成 HAVING 子句, 有下面两种不同的语法:

```
$this->db->having('user_id = 45'); // Produces: HAVING user_id = 45  
$this->db->having('user_id', 45); // Produces: HAVING user_id = 45
```

你也可以通过一个数组传入多个值:

```
$this->db->having(array('title =' => 'My Title', 'id <' => $id));  
// Produces: HAVING title = 'My Title', id < 45
```

如果 CodeIgniter 自动转义你的查询, 为了避免转义, 你可以将第三个参数设置为 FALSE。

```
$this->db->having('user_id', 45); // Produces: HAVING `user_id` = 45 in some databases su
$this->db->having('user_id', 45, FALSE); // Produces: HAVING user_id = 45
```

\$this->db->or_having()

该方法和 `having()` 方法一样, 只是多个条件之间使用 OR 进行连接。

排序

\$this->db->order_by()

该方法用于生成 ORDER BY 子句。

第一个参数为你想要排序的字段名, 第二个参数用于设置排序的方向, 可选项有: ASC (升序), DESC (降序) 和 RANDOM (随机)。

```
$this->db->order_by('title', 'DESC');
// Produces: ORDER BY `title` DESC
```

第一个参数也可以是你自己的排序字符串:

```
$this->db->order_by('title DESC, name ASC');
// Produces: ORDER BY `title` DESC, `name` ASC
```

如果需要根据多个字段进行排序, 可以多次调用该方法。

```
$this->db->order_by('title', 'DESC');
$this->db->order_by('name', 'ASC');
// Produces: ORDER BY `title` DESC, `name` ASC
```

如果你选择了 **RANDOM** (随机排序), 第一个参数会被忽略, 但是你可以传入一个数字值, 作为随机数的 seed。

```
$this->db->order_by('title', 'RANDOM');
// Produces: ORDER BY RAND()

$this->db->order_by(42, 'RANDOM');
// Produces: ORDER BY RAND(42)
```

注解: `order_by()` 方法的前身为 `orderby()`, 已废除

注解: Oracle 暂时还不支持随机排序, 会默认使用升序

分页与计数

\$this->db->limit()

该方法用于限制你的查询返回结果的数量:

```
$this->db->limit(10); // Produces: LIMIT 10
```

第二个参数可以用来设置偏移。

```
// Produces: LIMIT 20, 10 (in MySQL. Other databases have slightly different syntax)
$this->db->limit(10, 20);
```

`$this->db->count_all_results()`

该方法用于获取特定查询返回结果的数量, 也可以使用查询构造器的这些方法: `where()`, `or_where()`, `like()`, `or_like()` 等等。举例:

```
echo $this->db->count_all_results('my_table'); // Produces an integer, like 25
$this->db->like('title', 'match');
$this->db->from('my_table');
echo $this->db->count_all_results(); // Produces an integer, like 17
```

但是, 这个方法会重置你在 `select()` 方法里设置的所有值, 如果你希望保留它们, 可以将第二个参数设置为 `FALSE`

```
echo $this->db->count_all_results('my_table', FALSE);
```

`$this->db->count_all()`

该方法用于获取某个表的总行数, 第一个参数为表名:

```
echo $this->db->count_all('my_table'); // Produces an integer, like 25
```

查询条件组

查询条件组可以让你生成用括号括起来的一组 `WHERE` 条件, 这能创造出非常复杂的 `WHERE` 子句, 支持嵌套的条件组。例如:

```
$this->db->select('*')->from('my_table')
    ->group_start()
        ->where('a', 'a')
        ->or_group_start()
            ->where('b', 'b')
            ->where('c', 'c')
        ->group_end()
    ->group_end()
    ->where('d', 'd')
->get();

// Generates:
// SELECT * FROM (`my_table`) WHERE ( `a` = 'a' OR ( `b` = 'b' AND `c` = 'c' ) ) AND `d` =
```

注解: 条件组必须要配对, 确保每个 `group_start()` 方法都有一个 `group_end()` 方法与之配对。

`$this->db->group_start()`

开始一个新的条件组, 为查询中的 WHERE 条件添加一个左括号。

```
$this->db->or_group_start()
```

开始一个新的条件组, 为查询中的 WHERE 条件添加一个左括号, 并在前面加上 OR 。

```
$this->db->not_group_start()
```

开始一个新的条件组, 为查询中的 WHERE 条件添加一个左括号, 并在前面加上 NOT 。

```
$this->db->or_not_group_start()
```

开始一个新的条件组, 为查询中的 WHERE 条件添加一个左括号, 并在前面加上 OR NOT 。

```
$this->db->group_end()
```

结束当前的条件组, 为查询中的 WHERE 条件添加一个右括号。

插入数据

```
$this->db->insert()
```

该方法根据你提供的数据生成一条 INSERT 语句并执行, 它的参数是一个 ** 数组 ** 或一个 ** 对象 **, 下面是使用数组的例子:

```
$data = array(
    'title' => 'My title',
    'name' => 'My Name',
    'date' => 'My date'
);

$this->db->insert('mytable', $data);
// Produces: INSERT INTO mytable (title, name, date) VALUES ('My title', 'My name', 'My date')
```

第一个参数为要插入的表名, 第二个参数为要插入的数据, 是个关联数组。

下面是使用对象的例子:

```
/*
class MyClass {
    public $title = 'My Title';
    public $content = 'My Content';
    public $date = 'My Date';
}
*/

$object = new MyClass;
$this->db->insert('mytable', $object);
// Produces: INSERT INTO mytable (title, content, date) VALUES ('My Title', 'My Content', 'My Date')
```

第一个参数为要插入的表名, 第二个参数为要插入的数据, 是个对象。

注解: 所有数据会被自动转义, 生成安全的查询语句。

`$this->db->get_compiled_insert()`

该方法和 `$this->db->insert()` 方法一样根据你提供的数据生成一条 INSERT 语句, 但是并不执行。

例如:

```
$data = array(
    'title' => 'My title',
    'name'  => 'My Name',
    'date'  => 'My date'
);

$sql = $this->db->set($data)->get_compiled_insert('mytable');
echo $sql;
```

// Produces string: INSERT INTO mytable (`title`, `name`, `date`) VALUES ('My title', 'My name', 'My date')

第二个参数用于设置是否重置查询 (默认情况下会重置, 正如 `$this->db->insert()` 方法一样):

```
echo $this->db->set('title', 'My Title')->get_compiled_insert('mytable', FALSE);
```

// Produces string: INSERT INTO mytable (`title`) VALUES ('My Title')

```
echo $this->db->set('content', 'My Content')->get_compiled_insert();
```

// Produces string: INSERT INTO mytable (`title`, `content`) VALUES ('My Title', 'My Content')

上面的例子中, 最值得注意的是, 第二个查询并没有用到 `$this->db->from()` 方法, 也没有为查询指定表名参数, 但是它生成的 SQL 语句中有 INTO mytable 子句。这是因为查询并没有被重置 (使用 `$this->db->insert()` 方法会被执行并被重置, 使用 `$this->db->reset_query()` 方法直接重置)。

注解: 这个方法不支持批量插入。

`$this->db->insert_batch()`

该方法根据你提供的数据生成一条 INSERT 语句并执行, 它的参数是一个 ** 数组 ** 或一个 ** 对象 **, 下面是使用数组的例子:

```
$data = array(
    array(
        'title' => 'My title',
        'name'  => 'My Name',
        'date'  => 'My date'
    ),
    array(
        'title' => 'Another title',
        'name'  => 'Another Name',
    )
);
```

```

        'date' => 'Another date'
    )
);

$this->db->insert_batch('mytable', $data);
// Produces: INSERT INTO mytable (title, name, date) VALUES ('My title', 'My name', 'My date')

```

第一个参数为要插入的表名，第二个参数为要插入的数据，是个二维数组。

注解: 所有数据会被自动转义，生成安全的查询语句。

更新数据

`$this->db->replace()`

该方法用于执行一条 REPLACE 语句，REPLACE 语句根据表的 **** 主键 **** 和 **** 唯一索引 **** 来执行，类似于标准的 DELETE + INSERT。使用这个方法，你不用再手工去实现 `select()`，`update()`，`delete()` 以及 `insert()` 这些方法的不同组合，为你节约大量时间。

例如:

```

$data = array(
    'title' => 'My title',
    'name'  => 'My Name',
    'date'  => 'My date'
);

$this->db->replace('table', $data);

// Executes: REPLACE INTO mytable (title, name, date) VALUES ('My title', 'My name', 'My date')

```

上面的例子中，我们假设 `title` 字段是我们的主键，那么如果我们数据库里有一行的 `title` 列的值为 'My title'，这一行将会被删除并被我们的新数据所取代。

也可以使用 `set()` 方法，而且所有字段都被自动转义，正如 `insert()` 方法一样。

`$this->db->set()`

该方法用于设置新增或更新的数据。

该方法可以取代直接传递数据数组到 `insert` 或 `update` 方法:

```

$this->db->set('name', $name);
$this->db->insert('mytable'); // Produces: INSERT INTO mytable (`name`) VALUES ('{$name}')

```

如果你多次调用该方法，它会正确组装出 INSERT 或 UPDATE 语句来:

```

$this->db->set('name', $name);
$this->db->set('title', $title);
$this->db->set('status', $status);
$this->db->insert('mytable');

```

`set()` 方法也接受可选的第三个参数 (`$escape`), 如果设置为 `FALSE`, 数据将不会自动转义。为了说明两者之间的区别, 这里有一个带转义的 `set()` 方法和不带转义的例子。

```
$this->db->set('field', 'field+1', FALSE);
$this->db->where('id', 2);
$this->db->update('mytable'); // gives UPDATE mytable SET field = field+1 WHERE id = 2

$this->db->set('field', 'field+1');
$this->db->where('id', 2);
$this->db->update('mytable'); // gives UPDATE `mytable` SET `field` = 'field+1' WHERE `id` = 2
```

你也可以传一个关联数组作为参数:

```
$array = array(
    'name' => $name,
    'title' => $title,
    'status' => $status
);

$this->db->set($array);
$this->db->insert('mytable');
```

或者一个对象:

```
/*
class MyClass {
    public $title = 'My Title';
    public $content = 'My Content';
    public $date = 'My Date';
}
*/

$object = new MyClass;
$this->db->set($object);
$this->db->insert('mytable');
```

`$this->db->update()`

该方法根据你提供的数据生成一条 `UPDATE` 语句并执行, 它的参数是一个 **数组** 或一个 **对象**, 下面是使用数组的例子:

```
$data = array(
    'title' => $title,
    'name' => $name,
    'date' => $date
);

$this->db->where('id', $id);
$this->db->update('mytable', $data);
// Produces:
//
// UPDATE mytable
```

```
// SET title = '{$title}', name = '{$name}', date = '{$date}'
// WHERE id = $id
```

或者你可以使用一个对象:

```
/*
class MyClass {
    public $title = 'My Title';
    public $content = 'My Content';
    public $date = 'My Date';
}
*/

$object = new MyClass;
$this->db->where('id', $id);
$this->db->update('mytable', $object);
// Produces:
//
// UPDATE `mytable`
// SET `title` = '{$title}', `name` = '{$name}', `date` = '{$date}'
// WHERE id = `id`
```

注解: 所有数据会被自动转义, 生成安全的查询语句。

你应该注意到 `$this->db->where()` 方法的使用, 它可以为你设置 WHERE 子句。你也可以直接使用字符串形式设置 WHERE 子句:

```
$this->db->update('mytable', $data, "id = 4");
```

或者使用一个数组:

```
$this->db->update('mytable', $data, array('id' => $id));
```

当执行 UPDATE 操作时, 你还可以使用上面介绍的 `$this->db->set()` 方法。

`$this->db->update_batch()`

该方法根据你提供的数据生成一条 UPDATE 语句并执行, 它的参数是一个 ** 数组 ** 或一个 ** 对象 **, 下面是使用数组的例子:

```
$data = array(
    array(
        'title' => 'My title' ,
        'name' => 'My Name 2' ,
        'date' => 'My date 2'
    ),
    array(
        'title' => 'Another title' ,
        'name' => 'Another Name 2' ,
        'date' => 'Another date 2'
    )
);
```



```
$this->db->update_batch('mytable', $data, 'title');
```

```
// Produces:
// UPDATE `mytable` SET `name` = CASE
// WHEN `title` = 'My title' THEN 'My Name 2'
// WHEN `title` = 'Another title' THEN 'Another Name 2'
// ELSE `name` END,
// `date` = CASE
// WHEN `title` = 'My title' THEN 'My date 2'
// WHEN `title` = 'Another title' THEN 'Another date 2'
// ELSE `date` END
// WHERE `title` IN ('My title','Another title')
```

第一个参数为要更新的表名, 第二个参数为要更新的数据, 是个二维数组, 第三个参数是 WHERE 语句的键。

注解: 所有数据会被自动转义, 生成安全的查询语句。

注解: 取决于该方法的内部实现, 在这个方法之后调用 `affected_rows()` 方法返回的结果可能会不正确。但是你可以直接使用该方法的返回值, 代表了受影响的行数。

```
$this->db->get_compiled_update()
```

该方法和 `$this->db->get_compiled_insert()` 方法完全一样, 除了生成的 SQL 语句是 UPDATE 而不是 INSERT。

查看 `$this->db->get_compiled_insert()` 方法的文档获取更多信息。

注解: 该方法不支持批量更新。

删除数据

```
$this->db->delete()
```

该方法生成 DELETE 语句并执行。

```
$this->db->delete('mytable', array('id' => $id)); // Produces: // DELETE FROM mytable // WHERE id = $id
```

第一个参数为表名, 第二个参数为 WHERE 条件。你也可以不用第二个参数, 使用 `where()` 或者 `or_where()` 函数来替代它:

```
$this->db->where('id', $id);
$this->db->delete('mytable');
```

```
// Produces:
// DELETE FROM mytable
// WHERE id = $id
```

如果你想要从多个表中删除数据, 你也可以将由多个表名构成的数组传给 `delete()` 方法。

```
$tables = array('table1', 'table2', 'table3');
$this->db->where('id', '5');
$this->db->delete($tables);
```

如果你想要删除一个表中的所有数据, 可以使用 `truncate()` 或 `empty_table()` 方法。

```
$this->db->empty_table()
```

该方法生成 DELETE 语句并执行:

```
$this->db->empty_table('mytable'); // Produces: DELETE FROM mytable
```

```
$this->db->truncate()
```

该方法生成 TRUNCATE 语句并执行。

```
$this->db->from('mytable');
$this->db->truncate();
```

```
// or
```

```
$this->db->truncate('mytable');
```

```
// Produce:
```

```
// TRUNCATE mytable
```

注解: 如果 TRUNCATE 语句不可用, `truncate()` 方法将执行 “DELETE FROM table”。

```
$this->db->get_compiled_delete()
```

该方法和 `$this->db->get_compiled_insert()` 方法完全一样, 除了生成的 SQL 语句是 DELETE 而不是 INSERT。

查看 `$this->db->get_compiled_insert()` 方法的文档获取更多信息。

链式方法

通过将多个方法连接在一起, 链式方法可以大大的简化你的语法。感受一下这个例子:

```
$query = $this->db->select('title')
    ->where('id', $id)
    ->limit(10, 20)
    ->get('mytable');
```

查询构造器缓存

尽管不是“真正的”缓存，查询构造器允许你将查询的某个特定部分保存（或“缓存”）起来，以便在你的脚本执行之后重用。一般情况下，当查询构造器的一次调用结束后，所有已存储的信息都会被重置，以便下一次调用。如果开启缓存，你就可以使信息避免被重置，方便你进行重用。

缓存调用是累加的。如果你调用了两次有缓存的 `select()`，然后再调用两次没有缓存的 `select()`，这会导致 `select()` 被调用 4 次。

有三个可用的缓存方法方法：

`$this->db->start_cache()`

如需开启缓存必须先调用此方法，所有支持的查询类型（见下文）都会被存储起来供以后使用。

`$this->db->stop_cache()`

此方法用于停止缓存。

`$this->db->flush_cache()`

此方法用于清空缓存。

这里是一个使用缓存的例子：

```
$this->db->start_cache();
$this->db->select('field1');
$this->db->stop_cache();
$this->db->get('tablename');
//Generates: SELECT `field1` FROM (`tablename`)

$this->db->select('field2');
$this->db->get('tablename');
//Generates: SELECT `field1`, `field2` FROM (`tablename`)

$this->db->flush_cache();
$this->db->select('field2');
$this->db->get('tablename');
//Generates: SELECT `field2` FROM (`tablename`)
```

注解： 支持缓存的语句有：select, from, join, where, like, group_by, having, order_by, set

重置查询构造器

`$this->db->reset_query()`

该方法无需执行就能重置查询构造器中的查询，`$this->db->get()` 和 `$this->db->insert()` 方法也可以用于重置查询，但是必须先执行它。和这两个方法一样，使用‘查询构造器缓存’- 缓存下来的查询不会被重置。

当你在使用查询构造器生成 SQL 语句 (如: `$this->db->get_compiled_select()`), 之后再执行它。这种情况下, 不重置查询缓存将非常有用:

```
// Note that the second parameter of the get_compiled_select method is FALSE
$sql = $this->db->select(array('field1','field2'))
    ->where('field3',5)
    ->get_compiled_select('mytable', FALSE);

// ...
// Do something crazy with the SQL code... like add it to a cron script for
// later execution or something...
// ...

$data = $this->db->get()->result_array();

// Would execute and return an array of results of the following query:
// SELECT field1, field1 from mytable where field3 = 5;
```

注解: 如果你正在使用查询构造器缓存功能, 连续两次调用 `get_compiled_select()` 方法并且不重置你的查询, 这将会导致缓存被合并两次。举例来说, 例如你正在缓存 `select()` 方法, 那么会查询两个相同的字段。

Class Reference

class CI_DB_query_builder

`reset_query()`

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Resets the current Query Builder state. Useful when you want to build a query that can be cancelled under certain conditions.

`start_cache()`

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Starts the Query Builder cache.

`stop_cache()`

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Stops the Query Builder cache.

`flush_cache()`

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Empties the Query Builder cache.

set_dbprefix(*[\$prefix = '']*)

参数

- **\$prefix** (*string*) – The new prefix to use

返回 The DB prefix in use

返回类型 string

Sets the database prefix, without having to reconnect.

dbprefix(*[\$table = '']*)

参数

- **\$table** (*string*) – The table name to prefix

返回 The prefixed table name

返回类型 string

Prepends a database prefix, if one exists in configuration.

count_all_results(*[\$table = '']*, *\$reset = TRUE*)

参数

- **\$table** (*string*) – Table name
- **\$reset** (*bool*) – Whether to reset values for SELECTs

返回 Number of rows in the query result

返回类型 int

Generates a platform-specific query string that counts all records returned by an Query Builder query.

get(*[\$table = '']*, *\$limit = NULL*, *\$offset = NULL*)

参数

- **\$table** (*string*) – The table to query
- **\$limit** (*int*) – The LIMIT clause
- **\$offset** (*int*) – The OFFSET clause

返回 CI_DB_result instance (method chaining)

返回类型 CI_DB_result

Compiles and runs SELECT statement based on the already called Query Builder methods.

```
get_where([$table = '', $where = NULL[, $limit = NULL[, $offset = NULL
]]])
```

参数

- **\$table** (*mixed*) – The table(s) to fetch data from; string or array
- **\$where** (*string*) – The WHERE clause
- **\$limit** (*int*) – The LIMIT clause
- **\$offset** (*int*) – The OFFSET clause

返回 CI_DB_result instance (method chaining)

返回类型 CI_DB_result

Same as `get()`, but also allows the WHERE to be added directly.

```
select([$select = '*', $escape = NULL])
```

参数

- **\$select** (*string*) – The SELECT portion of a query
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a SELECT clause to a query.

```
select_avg([$select = '', $alias = ''])
```

参数

- **\$select** (*string*) – Field to compute the average of
- **\$alias** (*string*) – Alias for the resulting value name

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a SELECT AVG(field) clause to a query.

```
select_max([$select = '', $alias = ''])
```

参数

- **\$select** (*string*) – Field to compute the maximum of
- **\$alias** (*string*) – Alias for the resulting value name

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a SELECT MAX(field) clause to a query.

```
select_min([$select = '', $alias = ''])
```

参数

- **\$select** (*string*) – Field to compute the minimum of
- **\$alias** (*string*) – Alias for the resulting value name

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a SELECT MIN(field) clause to a query.

```
select_sum([$select = '', $alias = ''])
```

参数

- **\$select** (*string*) – Field to compute the sum of
- **\$alias** (*string*) – Alias for the resulting value name

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a SELECT SUM(field) clause to a query.

```
distinct([$val = TRUE])
```

参数

- **\$val** (*bool*) – Desired value of the “distinct” flag

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Sets a flag which tells the query builder to add a DISTINCT clause to the SELECT portion of the query.

```
from($from)
```

参数

- **\$from** (*mixed*) – Table name(s); string or array

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Specifies the FROM clause of a query.

```
join($table, $cond[, $type = '', $escape = NULL])
```

参数

- **\$table** (*string*) – Table name to join
- **\$cond** (*string*) – The JOIN ON condition
- **\$type** (*string*) – The JOIN type

- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a JOIN clause to a query.

where(\$key[, \$value = NULL[, \$escape = NULL]])

参数

- **\$key** (*mixed*) – Name of field to compare, or associative array
- **\$value** (*mixed*) – If a single key, compared to this value
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 DB_query_builder instance

返回类型 object

Generates the WHERE portion of the query. Separates multiple calls with 'AND'.

or_where(\$key[, \$value = NULL[, \$escape = NULL]])

参数

- **\$key** (*mixed*) – Name of field to compare, or associative array
- **\$value** (*mixed*) – If a single key, compared to this value
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 DB_query_builder instance

返回类型 object

Generates the WHERE portion of the query. Separates multiple calls with 'OR'.

or_where_in(\$key = NULL[, \$values = NULL[, \$escape = NULL]])

参数

- **\$key** (*string*) – The field to search
- **\$values** (*array*) – The values searched on
- **\$escape** (*bool*) – Whether to escape identifiers

返回 DB_query_builder instance

返回类型 object

Generates a WHERE field IN('item', 'item') SQL query, joined with 'OR' if appropriate.


```
or_where_not_in([$key = NULL[, $values = NULL[, $escape = NULL]]])
```

参数

- **\$key** (*string*) – The field to search
- **\$values** (*array*) – The values searched on
- **\$escape** (*bool*) – Whether to escape identifiers

返回 DB_query_builder instance

返回类型 object

Generates a WHERE field NOT IN('item', 'item') SQL query, joined with 'OR' if appropriate.

```
where_in([$key = NULL[, $values = NULL[, $escape = NULL]]])
```

参数

- **\$key** (*string*) – Name of field to examine
- **\$values** (*array*) – Array of target values
- **\$escape** (*bool*) – Whether to escape identifiers

返回 DB_query_builder instance

返回类型 object

Generates a WHERE field IN('item', 'item') SQL query, joined with 'AND' if appropriate.

```
where_not_in([$key = NULL[, $values = NULL[, $escape = NULL]]])
```

参数

- **\$key** (*string*) – Name of field to examine
- **\$values** (*array*) – Array of target values
- **\$escape** (*bool*) – Whether to escape identifiers

返回 DB_query_builder instance

返回类型 object

Generates a WHERE field NOT IN('item', 'item') SQL query, joined with 'AND' if appropriate.

```
group_start()
```

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Starts a group expression, using ANDs for the conditions inside it.

or_group_start()

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Starts a group expression, using ORs for the conditions inside it.

not_group_start()

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Starts a group expression, using AND NOTs for the conditions inside it.

or_not_group_start()

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Starts a group expression, using OR NOTs for the conditions inside it.

group_end()

返回 DB_query_builder instance

返回类型 object

Ends a group expression.

like(\$field[, \$match = '[', \$side = 'both', \$escape = NULL])

参数

- **\$field** (*string*) – Field name
- **\$match** (*string*) – Text portion to match
- **\$side** (*string*) – Which side of the expression to put the '%' wildcard on
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a LIKE clause to a query, separating multiple calls with AND.

or_like(\$field[, \$match = '[', \$side = 'both', \$escape = NULL])

参数

- **\$field** (*string*) – Field name
- **\$match** (*string*) – Text portion to match
- **\$side** (*string*) – Which side of the expression to put the '%' wildcard on
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a LIKE clause to a query, separating multiple class with OR.

```
not_like($field[, $match = '[', $side = 'both', $escape = NULL]]])
```

参数

- **\$field** (*string*) – Field name
- **\$match** (*string*) – Text portion to match
- **\$side** (*string*) – Which side of the expression to put the '%' wildcard on
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a NOT LIKE clause to a query, separating multiple calls with AND.

```
or_not_like($field[, $match = '[', $side = 'both', $escape = NULL]]])
```

参数

- **\$field** (*string*) – Field name
- **\$match** (*string*) – Text portion to match
- **\$side** (*string*) – Which side of the expression to put the '%' wildcard on
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a NOT LIKE clause to a query, separating multiple calls with OR.

```
having($key[, $value = NULL[, $escape = NULL]])
```

参数

- **\$key** (*mixed*) – Identifier (string) or associative array of field/value pairs
- **\$value** (*string*) – Value sought if \$key is an identifier
- **\$escape** (*string*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a HAVING clause to a query, separating multiple calls with AND.

```
or_having($key[, $value = NULL[, $escape = NULL]])
```

参数

- **\$key** (*mixed*) – Identifier (string) or associative array of field/value pairs
- **\$value** (*string*) – Value sought if \$key is an identifier
- **\$escape** (*string*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a HAVING clause to a query, separating multiple calls with OR.

group_by(\$by[, \$escape = NULL])

参数

- **\$by** (*mixed*) – Field(s) to group by; string or array

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds a GROUP BY clause to a query.

order_by(\$orderby[, \$direction = '[', \$escape = NULL])

参数

- **\$orderby** (*string*) – Field to order by
- **\$direction** (*string*) – The order requested - ASC, DESC or random
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds an ORDER BY clause to a query.

limit(\$value[, \$offset = 0])

参数

- **\$value** (*int*) – Number of rows to limit the results to
- **\$offset** (*int*) – Number of rows to skip

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds LIMIT and OFFSET clauses to a query.

offset(\$offset)

参数

- **\$offset** (*int*) – Number of rows to skip

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds an OFFSET clause to a query.

```
set($key[, $value = '', $escape = NULL])
```

参数

- **\$key** (*mixed*) – Field name, or an array of field/value pairs
- **\$value** (*string*) – Field value, if \$key is a single field
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds field/value pairs to be passed later to `insert()`, `update()` or `replace()`.

```
insert([ $table = '', $set = NULL[, $escape = NULL] ])
```

参数

- **\$table** (*string*) – Table name
- **\$set** (*array*) – An associative array of field/value pairs
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 TRUE on success, FALSE on failure

返回类型 bool

Compiles and executes an INSERT statement.

```
insert_batch([ $table = '', $set = NULL[, $escape = NULL] ])
```

参数

- **\$table** (*string*) – Table name
- **\$set** (*array*) – Data to insert
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 Number of rows inserted or FALSE on failure

返回类型 mixed

Compiles and executes batch INSERT statements.

```
set_insert_batch($key[, $value = '', $escape = NULL])
```

参数

- **\$key** (*mixed*) – Field name or an array of field/value pairs
- **\$value** (*string*) – Field value, if \$key is a single field
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds field/value pairs to be inserted in a table later via `insert_batch()`.

`update([$table = '', $set = NULL[, $where = NULL[, $limit = NULL]]])`

参数

- **\$table** (*string*) – Table name
- **\$set** (*array*) – An associative array of field/value pairs
- **\$where** (*string*) – The WHERE clause
- **\$limit** (*int*) – The LIMIT clause

返回 TRUE on success, FALSE on failure

返回类型 bool

Compiles and executes an UPDATE statement.

`update_batch([$table = '', $set = NULL[, $value = NULL]]])`

参数

- **\$table** (*string*) – Table name
- **\$set** (*array*) – Field name, or an associative array of field/value pairs
- **\$value** (*string*) – Field value, if \$set is a single field

返回 Number of rows updated or FALSE on failure

返回类型 mixed

Compiles and executes batch UPDATE statements.

`set_update_batch($key[, $value = '', $escape = NULL])`

参数

- **\$key** (*mixed*) – Field name or an array of field/value pairs
- **\$value** (*string*) – Field value, if \$key is a single field
- **\$escape** (*bool*) – Whether to escape values and identifiers

返回 CI_DB_query_builder instance (method chaining)

返回类型 CI_DB_query_builder

Adds field/value pairs to be updated in a table later via `update_batch()`.

`replace([$table = '', $set = NULL])`

参数

- **\$table** (*string*) – Table name

- **\$set** (*array*) – An associative array of field/value pairs

返回 TRUE on success, FALSE on failure

返回类型 bool

Compiles and executes a REPLACE statement.

```
delete([$table = '', $where = '', $limit = NULL, $reset_data = TRUE])
```

参数

- **\$table** (*mixed*) – The table(s) to delete from; string or array
- **\$where** (*string*) – The WHERE clause
- **\$limit** (*int*) – The LIMIT clause
- **\$reset_data** (*bool*) – TRUE to reset the query “write” clause

返回 CI_DB_query_builder instance (method chaining) or FALSE on failure

返回类型 mixed

Compiles and executes a DELETE query.

```
truncate([$table = ''])
```

参数

- **\$table** (*string*) – Table name

返回 TRUE on success, FALSE on failure

返回类型 bool

Executes a TRUNCATE statement on a table.

注解: If the database platform in use doesn't support TRUNCATE, a DELETE statement will be used instead.

```
empty_table([$table = ''])
```

参数

- **\$table** (*string*) – Table name

返回 TRUE on success, FALSE on failure

返回类型 bool

Deletes all records from a table via a DELETE statement.

```
get_compiled_select([$table = '', $reset = TRUE])
```

参数

- **\$table** (*string*) – Table name

- **\$reset** (*bool*) – Whether to reset the current QB values or not

返回 The compiled SQL statement as a string

返回类型 string

Compiles a SELECT statement and returns it as a string.

```
get_compiled_insert([$table = '', $reset = TRUE])
```

参数

- **\$table** (*string*) – Table name
- **\$reset** (*bool*) – Whether to reset the current QB values or not

返回 The compiled SQL statement as a string

返回类型 string

Compiles an INSERT statement and returns it as a string.

```
get_compiled_update([$table = '', $reset = TRUE])
```

参数

- **\$table** (*string*) – Table name
- **\$reset** (*bool*) – Whether to reset the current QB values or not

返回 The compiled SQL statement as a string

返回类型 string

Compiles an UPDATE statement and returns it as a string.

```
get_compiled_delete([$table = '', $reset = TRUE])
```

参数

- **\$table** (*string*) – Table name
- **\$reset** (*bool*) – Whether to reset the current QB values or not

返回 The compiled SQL statement as a string

返回类型 string

Compiles a DELETE statement and returns it as a string.

9.1.8 事务

CodeIgniter 允许你在支持事务安全的表上使用事务。在 MySQL 中，你需要将表的存储引擎设置为 InnoDB 或 BDB，而不是通常我们使用的 MyISAM。大多数其他数据库平台都原生支持事务。

如果你对事务还不熟悉，我们推荐针对你正在使用的数据库，先在网上寻找一些在线资源学习一下。下面将假设你已经明白事务的基本概念。

CodeIgniter 的事务方法

CodeIgniter 使用的事务处理方法与流行的数据库类 ADODB 的处理方法非常相似。我们选择这种方式是因为它极大的简化了事务的处理过程。大多数情况下, 你只需编写两行代码就行了。

传统的事务处理需要实现大量的工作, 你必须随时跟踪你的查询, 并根据查询的成功或失败来决定提交还是回滚。当遇到嵌套查询时将会更加麻烦。相比之下, 我们实现了一个智能的事务系统, 它将自动的为你做这些工作。(你仍然可以选择手工管理你的事务, 但这实在是没啥好处)

运行事务

要使用事务来运行你的查询, 你可以使用 `$this->db->trans_start()` 和 `$this->db->trans_complete()` 两个方法, 像下面这样:

```
$this->db->trans_start();
$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->query('AND YET ANOTHER QUERY...');
$this->db->trans_complete();
```

在 `start` 和 `complete` 之间, 你可以运行任意多个查询, 根据查询执行成功或失败, 系统将自动提交或回滚。

严格模式 (Strict Mode)

CodeIgniter 默认使用严格模式运行所有的事务, 在严格模式下, 如果你正在运行多组事务, 只要有一组失败, 所有组都会被回滚。如果禁用严格模式, 那么每一组都被视为独立的组, 这意味着其中一组失败不会影响其他的组。

严格模式可以用下面的方法禁用:

```
$this->db->trans_strict(FALSE);
```

错误处理

如果你的数据库配置文件 `config/database.php` 中启用了错误报告 (`db_debug = TRUE`), 当提交没有成功时, 你会看到一条标准的错误信息。如果没有启用错误报告, 你可以像下面这样来管理你的错误:

```
$this->db->trans_start();
$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->trans_complete();

if ($this->db->trans_status() === FALSE)
{
```

```
// generate an error... or use the log_message() function to log your error
}
```

启用事务

当执行 `$this->db->trans_start()` 方法时, 事务将自动启用, 如果你要禁用事务, 可以使用 `$this->db->trans_off()` 方法来实现:

```
$this->db->trans_off();

$this->db->trans_start();
$this->db->query('AN SQL QUERY...');
$this->db->trans_complete();
```

当事务被禁用时, 你的查询会自动提交, 就跟没有使用事务一样。

测试模式 (Test Mode)

你可以选择性的将你的事务系统设置为“测试模式”, 这将导致你的所有查询都被回滚, 就算查询成功执行也一样。要使用“测试模式”, 你只需要将 `$this->db->trans_start()` 函数的第一个参数设置为 `TRUE` 即可:

```
$this->db->trans_start(TRUE); // Query will be rolled back
$this->db->query('AN SQL QUERY...');
$this->db->trans_complete();
```

手工运行事务

如果你想手工运行事务, 可以像下面这样做:

```
$this->db->trans_begin();

$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->query('AND YET ANOTHER QUERY...');

if ($this->db->trans_status() === FALSE)
{
    $this->db->trans_rollback();
}
else
{
    $this->db->trans_commit();
}
```

注解: 手动运行事务时, 请务必使用 `$this->db->trans_begin()` 方法, 而不是 `$this->db->trans_start()` 方法。

9.1.9 数据库元数据

表元数据

下面这些方法用于获取表信息：

列出数据库的所有表

```
$this->db->list_tables();
```

该方法返回一个包含你当前连接的数据库的所有表名称的数组。例如：

```
$tables = $this->db->list_tables();  
  
foreach ($tables as $table)  
{  
    echo $table;  
}
```

检测表是否存在

```
$this->db->table_exists();
```

有时候，在对某个表执行操作之前先判断该表是否存在将是很有用的。该函数返回一个布尔值：TRUE / FALSE。使用示例：

```
if ($this->db->table_exists('table_name'))  
{  
    // some code...  
}
```

注解： 使用你要查找的表名替换掉 *table_name*

字段元数据

列出表的所有列

```
$this->db->list_fields()
```

该方法返回一个包含字段名称的数组。有两种不同的调用方式：

1. 将表名陈作为参数传入 `$this->db->list_fields()`：

```
$fields = $this->db->list_fields('table_name');  
  
foreach ($fields as $field)  
{
```

```
        echo $field;
    }
}
```

2. 你可以从任何查询结果对象上调用该方法，获取查询返回的所有字段:

```
$query = $this->db->query('SELECT * FROM some_table');

foreach ($query->list_fields() as $field)
{
    echo $field;
}
```

检测表中是否存在某字段

`$this->db->field_exists()`

有时候，在执行一个操作之前先确定某个字段是否存在将会有用。该方法返回一个布尔值：TRUE / FALSE。使用示例:

```
if ($this->db->field_exists('field_name', 'table_name'))
{
    // some code...
}
```

注解: 使用你要查找的字段名替换掉 *field_name*，然后使用你要查找的表名替换掉 *table_name*。

获取字段的元数据

`$this->db->field_data()`

该方法返回一个包含了字段信息的对象数组。

获取字段名称或相关的元数据，如数据类型，最大长度等等，在有些时候也是非常有用的。

注解: 并不是所有的数据库都支持元数据。

使用示例:

```
$fields = $this->db->field_data('table_name');

foreach ($fields as $field)
{
    echo $field->name;
    echo $field->type;
    echo $field->max_length;
    echo $field->primary_key;
}
```

如果你已经执行了一个查询，你也可以在查询结果对象上调用该方法获取返回结果中的所有字段的元数据：

```
$query = $this->db->query("YOUR QUERY");  
$fields = $query->field_data();
```

如果你的数据库支持，该函数获取的字段信息将包括下面这些：

- name - 列名称
- max.length - 列的最大长度
- primary_key - 等于 1 的话表示此列是主键
- type - 列的数据类型

9.1.10 自定义函数调用

```
$this->db->call_function();
```

这个方法用于执行一些 CodeIgniter 中没有定义的 PHP 数据库函数，而且使用了一种平台独立的方式。举个例子，假设你要调用 `mysql_get_client_info()` 函数，这个函数 CodeIgniter 并不是原生支持的，你可以这样做：

```
$this->db->call_function('get_client_info');
```

你必须提供一个不带 `mysql_` 前缀的函数名来作为第一个参数，这个前缀会根据当前所使用的数据库驱动自动添加。这让你可以在不同的数据库平台执行相同的函数。但是很显然，并不是所有的数据库平台函数都是一样的，所以就移植性而言，它的作用非常有限。

任何你需要的其它参数都放在第一个参数后面。

```
$this->db->call_function('some_function', $param1, $param2, etc..);
```

经常的，你会需要提供一个数据库的 connection ID 或是一个 result ID，connection ID 可以这样来获得：

```
$this->db->conn_id;
```

result ID 可以从查询返回的结果对象获取，像这样：

```
$query = $this->db->query("SOME QUERY");  
  
$query->result_id;
```

9.1.11 数据库缓存类

数据库缓存类允许你把数据库查询结果保存在文本文件中以减少数据库访问。

重要： 当缓存启用时，本类会被数据库驱动自动加载，切勿手动加载。

重要: 并非所有查询结果都能被缓存, 请仔细阅读本页内容。

启用缓存

启用缓存需要三步:

- 在服务器上创建一个可写的目录以便保存缓存文件;
- 通过文件 `application/config/database.php` 中的 `cachedir` 参数设置其目录路径;
- 通过将文件 `application/config/database.php` 中的 `cache_on` 参数设置为 `TRUE`, 也可以用下面的方法手动配置。

缓存一旦启用, 每一次加载页面时, 只要该页面含有数据库查询就会自动缓存起来。

缓存是如何工作的?

当你在访问页面时, CodeIgniter 的查询缓存系统会自动运行。如果缓存被启用, 当页面第一次加载时, 查询结果对象会被序列化并保存到服务器上的一个文本文件中。当下次再访问该页面时, 会直接使用缓存文件而不用访问数据库了, 这样, 在已缓存的页面, 你的数据库访问会降为 0。

只有读类型 (SELECT) 的查询可以被缓存, 因为只有这类查询才会产生结果。写类型的查询 (INSERT、UPDATE 等) 并不会生成结果, 所以不会被缓存。

缓存文件永不过期, 所有的查询只要缓存下来以后除非你删除它们否则将一直可用。你可以针对特定的页面来删除缓存, 或者也可以清空掉所有的缓存。一般来说, 你可以在某些事件发生时 (如数据库中添加了数据) 用下面的函数来清除缓存。

缓存能够提升站点的性能吗?

缓存能否获得性能增益, 取决于很多因素。如果你有一个低负荷而高度优化的数据库, 你可能不会看到性能的提升。而如果你的数据库正在被大量访问, 您可能会看到缓存后的性能有所提升, 前提是你的文件系统并没有太多的开销。要记住一点的是, 缓存只是简单的改变了数据获取的途径而已, 从访问数据库变成了访问文件系统。

例如, 在一些集群服务器环境中, 由于文件系统的操作太过频繁, 缓存其实是有害的。在共享的单一服务器环境中, 缓存才可能有益。不幸的是, 关于是否需要缓存你的数据库这个问题并没有唯一的答案, 这完全取决于你的情况。

缓存文件是如何存储的?

CodeIgniter 将每个查询都缓存到它单独的缓存文件中, 根据调用的控制器方法缓存文件被进一步组织到各自的子目录中。更准确的说, 子目录是使用你 URI 的前两段 (控制器名和方法名) 命名的。

例如, 你有一个 blog 控制器和一个 comments 方法, 并含有三个不同的查询。缓存系统将创建一个名为 blog+comments 的目录, 并在该目录下生成三个缓存文件。

如果你的 URI 中含有动态查询时 (例如使用分页时), 每个查询实例都会生成它单独的缓存文件, 因此, 最终可能会出现缓存文件数是你页面中的查询次数的好几倍这样的情况。

管理你的缓存文件

由于缓存文件不会过期, 那么你的应用程序中应该有删除缓存的机制, 例如, 我们假设你有一个博客并允许用户评论, 每当提交一个新评论时, 你都应该删除掉关于显示评论的那个控制器方法对应的缓存文件。下面将介绍有两种不同的方法用来删除缓存数据。

不是所有的数据库方法都兼容缓存

最后, 我们必须得指出被缓存的结果对象只是一个简化版的结果对象, 正因为这样, 有几个查询结果的方法无法使用。

下面列出的方法是无法在缓存的结果对象上使用的:

- num_fields()
- field_names()
- field_data()
- free_result()

同时, result_id 和 conn_id 这两个 id 也无法使用, 因为这两个 id 只适用于实时的数据库操作。

函数参考

`$this->db->cache_on()` / `$this->db->cache_off()`

用于手工启用/禁用缓存, 当你不想缓存某些查询时, 这两个方法会很有用。例子:

```
// Turn caching on
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM mytable");

// Turn caching off for this one query
$this->db->cache_off();
$query = $this->db->query("SELECT * FROM members WHERE member_id = '$current_user'");

// Turn caching back on
$this->db->cache_on();
$query = $this->db->query("SELECT * FROM another_table");
```

`$this->db->cache_delete()`

删除特定页面的缓存文件，这当你更新你的数据库之后需要清除缓存时很有用。

缓存系统根据你访问页面的 URI 来将缓存写入到相应的缓存文件中，例如，如果你在访问 `example.com/index.php/blog/comments` 这个页面，缓存系统会将缓存文件保存到 `blog+comments` 目录下，要删除这些缓存文件，你可以使用：

```
$this->db->cache_delete('blog', 'comments');
```

如果你没提供任何参数，将会清除当前 URI 对应的缓存文件。

`$this->db->cache_delete_all()`

清除所有的缓存文件，例如：

```
$this->db->cache_delete_all();
```

9.1.12 数据库工厂类

数据库工厂类提供了一些方法来帮助你管理你的数据库。

Table of Contents

- 数据库工厂类
 - 初始化数据库工厂类
 - 创建和删除数据库
 - 创建和删除数据表
 - * 添加字段
 - * 添加键
 - * 创建表
 - * 删除表
 - * 重命名表
 - 修改表
 - * 给表添加列
 - * 从表中删除列
 - * 修改表中的某个列
 - 类参考

初始化数据库工厂类

重要： 由于数据库工厂类依赖于数据库驱动器，为了初始化该类，你的数据库驱动器必须已经运行。

加载数据库工厂类的代码如下：


```
$this->load->dbforge()
```

如果你想管理的不是你正在使用的数据库, 你还可以传另一个数据库对象到数据库工具类的加载方法:

```
$this->myforge = $this->load->dbforge($this->other_db, TRUE);
```

上例中, 我们通过第一个参数传递了一个自定义的数据库对象, 第二个参数表示方法将返回 dbforge 对象, 而不是直接赋值给 `$this->dbforge`。

注解: 两个参数都可以独立使用, 如果你只想传第二个参数, 可以将第一个参数置空。

一旦初始化结束, 你就可以使用 `$this->dbforge` 对象来访问它的方法:

```
$this->dbforge->some_method();
```

创建和删除数据库

```
$this->dbforge->create_database('db_name')
```

用于创建指定数据库, 根据成败返回 TRUE 或 FALSE

```
if ($this->dbforge->create_database('my_db'))
{
    echo 'Database created!';
}
```

```
$this->dbforge->drop_database('db_name')
```

用于删除指定数据库, 根据成败返回 TRUE 或 FALSE

```
if ($this->dbforge->drop_database('my_db'))
{
    echo 'Database deleted!';
}
```

创建和删除数据表

创建表涉及到这样几件事: 添加字段、添加键、修改字段。CodeIgniter 提供了这几个方法。

添加字段

字段通过一个关联数组来创建, 数组中必须包含一个 'type' 索引, 代表字段的数据类型。例如, INT、VARCHAR、TEXT 等, 有些数据类型 (例如 VARCHAR) 还需要加一个 'constraint' 索引。

```
$fields = array(
    'users' => array(
        'type' => 'VARCHAR',
        'constraint' => '100',
    ),
);
// will translate to "users VARCHAR(100)" when the field is added.
```

另外, 还可以使用下面的键值对:

- unsigned/true : 在字段定义中生成 “UNSIGNED”
- default/value : 在字段定义中生成一个默认值
- null/true : 在字段定义中生成 “NULL”, 如果没有这个, 字段默认为 “NOT NULL”
- auto_increment/true : 在字段定义中生成自增标识, 注意数据类型必须支持这个, 例如整型

```
$fields = array(
    'blog_id' => array(
        'type' => 'INT',
        'constraint' => 5,
        'unsigned' => TRUE,
        'auto_increment' => TRUE
    ),
    'blog_title' => array(
        'type' => 'VARCHAR',
        'constraint' => '100',
    ),
    'blog_author' => array(
        'type' => 'VARCHAR',
        'constraint' => '100',
        'default' => 'King of Town',
    ),
    'blog_description' => array(
        'type' => 'TEXT',
        'null' => TRUE,
    ),
);
```

字段定义好了之后, 就可以在调用 `create_table()` 方法的后面使用 `$this->dbforge->add_field($fields);` 方法来添加字段了。

`$this->dbforge->add_field()`

添加字段方法的参数就是上面介绍的数组。

使用字符串参数添加字段 如果你非常清楚的知道你要添加的字段, 你可以使用字段的定义字符串来传给 `add_field()` 方法

```
$this->dbforge->add_field("label varchar(100) NOT NULL DEFAULT 'default label'");
```

注解: Passing raw strings as fields cannot be followed by `add_key()` calls on those fields.

注解: 多次调用 `add_field()` 将会累积。

创建 id 字段 创建 id 字段和创建其他字段非常不一样, id 字段将会自动定义成类型为 INT(9) 的自增主键。

```
$this->dbforge->add_field('id');  
// gives id INT(9) NOT NULL AUTO_INCREMENT
```

添加键

通常来说, 表都会有键。这可以使用 `$this->dbforge->add_key('field')` 方法来实现。第二个参数可选, 可以将其设置为主键。注意 `add_key()` 方法必须紧跟在 `create_table()` 方法的后面。

包含多列的非主键必须使用数组来添加, 下面是 MySQL 的例子。

```
$this->dbforge->add_key('blog_id', TRUE);  
// gives PRIMARY KEY `blog_id` (`blog_id`)  
  
$this->dbforge->add_key('blog_id', TRUE);  
$this->dbforge->add_key('site_id', TRUE);  
// gives PRIMARY KEY `blog_id_site_id` (`blog_id`, `site_id`)  
  
$this->dbforge->add_key('blog_name');  
// gives KEY `blog_name` (`blog_name`)  
  
$this->dbforge->add_key(array('blog_name', 'blog_label'));  
// gives KEY `blog_name_blog_label` (`blog_name`, `blog_label`)
```

创建表

字段和键都定义好了之后, 你可以使用下面的方法来创建表:

```
$this->dbforge->create_table('table_name');  
// gives CREATE TABLE table_name
```

第二个参数设置为 TRUE, 可以在定义中添加 “IF NOT EXISTS” 子句。

```
$this->dbforge->create_table('table_name', TRUE);  
// gives CREATE TABLE IF NOT EXISTS table_name
```

你还可以指定表的属性, 例如 MySQL 的 ENGINE

```
$attributes = array('ENGINE' => 'InnoDB');
$this->dbforge->create_table('table_name', FALSE, $attributes);
// produces: CREATE TABLE `table_name` (...) ENGINE = InnoDB DEFAULT CHARACTER SET utf8 COLLATE = utf8_bin
```

注解: 除非你指定了 CHARACTER SET 或 COLLATE 属性, create_table() 方法默认会使用配置文件中 char_set 和 dbcollat 的值 (仅针对 MySQL)。

删除表

执行一个 DROP TABLE 语句, 可以选择添加 IF EXISTS 子句。

```
// Produces: DROP TABLE table_name
$this->dbforge->drop_table('table_name');

// Produces: DROP TABLE IF EXISTS table_name
$this->dbforge->drop_table('table_name', TRUE);
```

重命名表

执行一个重命名表语句。

```
$this->dbforge->rename_table('old_table_name', 'new_table_name');
// gives ALTER TABLE old_table_name RENAME TO new_table_name
```

修改表

给表添加列

\$this->dbforge->add_column()

add_column() 方法用于对现有数据表进行修改, 它的参数和上面介绍的字段数组一样。

```
$fields = array(
    'preferences' => array('type' => 'TEXT')
);
$this->dbforge->add_column('table_name', $fields);
// Executes: ALTER TABLE table_name ADD preferences TEXT
```

如果你使用 MySQL 或 CUBIRD, 你可以使用 AFTER 和 FIRST 语句来为新添加的列指定位置。

例如:

```
// Will place the new column after the `another_field` column:
$fields = array(
    'preferences' => array('type' => 'TEXT', 'after' => 'another_field')
```

```
);

// Will place the new column at the start of the table definition:
$fields = array(
    'preferences' => array('type' => 'TEXT', 'first' => TRUE)
);
```

从表中删除列

`$this->dbforge->drop_column()`

用于从表中删除指定列。

```
$this->dbforge->drop_column('table_name', 'column_to_drop');
```

修改表中的某个列

`$this->dbforge->modify_column()`

该方法的用法和 `add_column()` 一样，只是它用于对现有的列进行修改，而不是添加新列。如果要修改列的名称，你可以在列的定义数组中添加一个“name”索引。

```
$fields = array(
    'old_name' => array(
        'name' => 'new_name',
        'type' => 'TEXT',
    ),
);
$this->dbforge->modify_column('table_name', $fields);
// gives ALTER TABLE table_name CHANGE old_name new_name TEXT
```

类参考

`class CI_DB_forge`

`add_column($table[, $field = array(), $after = NULL])`

参数

- **`$table`** (*string*) – Table name to add the column to
- **`$field`** (*array*) – Column definition(s)
- **`$after`** (*string*) – Column for AFTER clause (deprecated)

返回 TRUE on success, FALSE on failure

返回类型 bool

给表添加列。用法参见给表添加列。

`add_field($field)`

参数

- **\$field** (*array*) – Field definition to add

返回 CI_DB_forge instance (method chaining)

返回类型

CI_DB_forge

添加字段到集合, 用于创建一个表。用法参见[添加字段](#)。

`add_key($key[, $primary = FALSE])`

参数

- **\$key** (*array*) – Name of a key field
- **\$primary** (*bool*) – Set to TRUE if it should be a primary key or a regular one

返回 CI_DB_forge instance (method chaining)

返回类型 CI_DB_forge

添加键到集合, 用于创建一个表。用法参见: [添加键](#)。

`create_database($db_name)`

参数

- **\$db_name** (*string*) – Name of the database to create

返回 TRUE on success, FALSE on failure

返回类型 bool

创建数据库。用法参见: [创建和删除数据库](#)。

`create_table($table[, $if_not_exists = FALSE[, array $attributes = array()]])`

参数

- **\$table** (*string*) – Name of the table to create
- **\$if_not_exists** (*string*) – Set to TRUE to add an 'IF NOT EXISTS' clause
- **\$attributes** (*string*) – An associative array of table attributes

返回 TRUE on success, FALSE on failure

返回类型 bool

创建表。用法参见: [创建表](#)。

`drop_column($table, $column_name)`

参数

- **\$table** (*string*) – Table name
- **\$column_name** (*array*) – The column name to drop

返回 TRUE on success, FALSE on failure

返回类型 bool

删除某个表的字段。用法参见：[从表中删除列](#)。

drop_database(\$db_name)

参数

- **\$db_name** (*string*) – Name of the database to drop

返回 TRUE on success, FALSE on failure

返回类型 bool

删除数据库。用法参见：[创建和删除数据库](#)。

drop_table(\$table_name[, \$if_exists = FALSE])

参数

- **\$table** (*string*) – Name of the table to drop
- **\$if_exists** (*string*) – Set to TRUE to add an 'IF EXISTS' clause

返回 TRUE on success, FALSE on failure

返回类型 bool

删除表。用法参见：[删除表](#)。

modify_column(\$table, \$field)

参数

- **\$table** (*string*) – Table name
- **\$field** (*array*) – Column definition(s)

返回 TRUE on success, FALSE on failure

返回类型 bool

修改表的某个列。用法参见：[修改表中的某个列](#)。

rename_table(\$table_name, \$new_table_name)

参数

- **\$table** (*string*) – Current of the table
- **\$new_table_name** (*string*) – New name of the table

返回 TRUE on success, FALSE on failure

返回类型 bool

重命名表。用法参见：[重命名表](#)。

9.1.13 数据库工具类

数据库工具类提供了一些方法用于帮助你管理你的数据库。

- 初始化工具类
- 使用数据库工具类
 - 获取数据库名称列表
 - 判断一个数据库是否存在
 - 优化表
 - 修复表
 - 优化数据库
 - 将查询结果导出到 CSV 文档
 - 将查询结果导出到 XML 文档
- 备份你的数据库
 - 数据备份说明
 - 使用示例
 - 设置备份参数
 - 备份参数说明
- 类参考

初始化工具类

重要: 由于工具类依赖于数据库驱动器，为了初始化工具类，你的数据库驱动器必须已经运行。

加载工具类的代码如下：

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

如果你想管理的不是你正在使用的数据库，你还可以传另一个数据库对象到数据库工具类的加载方法：

```
$this->myutil = $this->load->dbutil($this->other_db, TRUE);
```

上例中，我们通过第一个参数传递了一个自定义的数据库对象，第二个参数表示方法将返回 dbutil 对象，而不是直接赋值给 `$this->dbutil`。

注解: 两个参数都可以独立使用，如果你只想传第二个参数，可以将第一个参数置空。

一旦初始化结束，你就可以使用 `$this->dbutil` 对象来访问它的方法：

```
$this->dbutil->some_method();
```


使用数据库工具类

获取数据库名称列表

返回一个包含所有数据库名称的列表:

```
$dbs = $this->dbutil->list_databases();

foreach ($dbs as $db)
{
    echo $db;
}
```

判断一个数据库是否存在

有时我们需要判断某个数据库是否存在, 可以使用该方法。方法返回布尔值 TRUE/FALSE 。例如:

```
if ($this->dbutil->database_exists('database_name'))
{
    // some code...
}
```

注解: 使用你自己的数据库名替换 *database_name* , 该方法区分大小写。

优化表

根据你指定的表名来优化表, 根据成败返回 TRUE 或 FALSE

```
if ($this->dbutil->optimize_table('table_name'))
{
    echo 'Success!';
}
```

注解: 不是所有的数据库平台都支持表优化, 通常使用在 MySQL 数据库上。

修复表

根据你指定的表名来修复表, 根据成败返回 TRUE 或 FALSE

```
if ($this->dbutil->repair_table('table_name'))
{
    echo 'Success!';
}
```

注解: 不是所有的数据库平台都支持表修复。

优化数据库

允许你优化数据库类当前正在连接的数据库。返回一个数组，包含数据库状态信息，失败时返回 FALSE。

```
$result = $this->dbutil->optimize_database();

if ($result !== FALSE)
{
    print_r($result);
}
```

注解: 不是所有的数据库平台都支持数据库优化，通常使用在 MySQL 数据库上。

将查询结果导出到 CSV 文档

允许你从查询结果生成 CSV 文档，第一个参数必须是查询的结果对象。例如：

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

$query = $this->db->query("SELECT * FROM mytable");

echo $this->dbutil->csv_from_result($query);
```

第二、三、四个参数分别为分隔符、换行符和每个字段包围字符，默认情况下，分隔符为逗号，换行符为“n”，包围字符为双引号。例如：

```
$delimiter = ",";
$newline = "\r\n";
$enclosure = '"';

echo $this->dbutil->csv_from_result($query, $delimiter, $newline, $enclosure);
```

重要: 该方法并不写入 CSV 文档，它只是简单的返回 CSV 内容，如果你需要写入到文件中，你可以使用[文件辅助函数](#)。

将查询结果导出到 XML 文档

允许你从查询结果生成 XML 文档，第一个参数为查询的结果对象，第二个参数可选，可以包含一些的配置参数。例如：

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

$query = $this->db->query("SELECT * FROM mytable");

$config = array (
    'root'      => 'root',
    'element'   => 'element',
    'newline'   => "\n",
    'tab'       => "\t"
);

echo $this->dbutil->xml_from_result($query, $config);
```

重要: 该方法并不写入 XML 文档, 它只是简单的返回 XML 内容, 如果你需要写入到文件中, 你可以使用[文件辅助函数](#)。

备份你的数据库

数据备份说明

允许你备份完整的数据库或指定的表。备份的数据可以压缩成 Zip 或 Gzip 格式。

注解: 该功能只支持 MySQL 和 Interbase/Firebird 数据库。

注解: 对于 Interbase/Firebird 数据库, 只能提供一个备份文件名参数。

```
$this->dbutil->backup('db_backup_filename');
```

注解: 限于 PHP 的执行时间和内存限制, 备份非常大的数据库应该不行。如果你的数据库非常大, 你可以直接使用命令行进行备份, 如果你没有 root 权限的话, 让你的管理员来帮你备份。

使用示例

```
// Load the DB utility class
$this->load->dbutil();

// Backup your entire database and assign it to a variable
$backup = $this->dbutil->backup();

// Load the file helper and write the file to your server
$this->load->helper('file');
write_file('/path/to/mybackup.gz', $backup);
```

```
// Load the download helper and send the file to your desktop
$this->load->helper('download');
force_download('mybackup.gz', $backup);
```

设置备份参数

备份参数为一个数组，通过第一个参数传递给 backup() 方法，例如：

```
$prefs = array(
    'tables'      => array('table1', 'table2'),    // Array of tables to backup.
    'ignore'      => array(),                      // List of tables to omit from the backup
    'format'      => 'txt',                        // gzip, zip, txt
    'filename'    => 'mybackup.sql',               // File name - NEEDED ONLY WITH ZIP FILES
    'add_drop'    => TRUE,                        // Whether to add DROP TABLE statements to backup file
    'add_insert'  => TRUE,                        // Whether to add INSERT data to backup file
    'newline'     => "\n"                         // Newline character used in backup file
);

$this->dbutil->backup($prefs);
```

备份参数说明

参数	默认值选项	描述	
tables	empty array	None	你要备份的表，如果留空将备份所有的表。
ignore	empty array	None	你要忽略备份的表。
format	gzip	gzip, zip, txt	导出文件的格式。
filename	the current date/time	None	备份文件名。如果你使用了 zip 压缩这个参数是必填的。
add_drop	TRUE	TRUE/ FALSE	是否在导出的 SQL 文件里包含 DROP TABLE 语句
add_insert	TRUE	TRUE/ FALSE	是否在导出的 SQL 文件里包含 INSERT 语句
newline	"\n"	"\n", "\r", "\r\n"	导出的 SQL 文件使用的换行符
foreign_key_checks	TRUE	TRUE/ FALSE	导出的 SQL 文件中是否继续保持外键约束

类参考

class CI_DB_utility

```
backup([ $params = array() ])
```

参数

- **\$params** (*array*) – An associative array of options

返回 raw/(g)zipped SQL query string

返回类型 string

根据用户参数执行数据库备份。

database_exists(\$database_name)

参数

- **\$database_name** (*string*) – Database name

返回 TRUE if the database exists, FALSE otherwise

返回类型 bool

判断数据库是否存在。

list_databases()

返回 Array of database names found

返回类型 array

获取所有的数据库名称列表。

optimize_database()

返回 Array of optimization messages or FALSE on failure

返回类型 array

优化数据库。

optimize_table(\$table_name)

参数

- **\$table_name** (*string*) – Name of the table to optimize

返回 Array of optimization messages or FALSE on failure

返回类型 array

优化数据库表。

repair_table(\$table_name)

参数

- **\$table_name** (*string*) – Name of the table to repair

返回 Array of repair messages or FALSE on failure

返回类型 array

修复数据库表。

```
csv_from_result($query[, $delim = ',', $newline = "\n", $enclosure = '"']
                ]])
```

参数

- **\$query** (*object*) – A database result object
- **\$delim** (*string*) – The CSV field delimiter to use
- **\$newline** (*string*) – The newline character to use
- **\$enclosure** (*string*) – The enclosure delimiter to use

返回 The generated CSV file as a string

返回类型 string

将数据库结果对象转换为 CSV 文档。

```
xml_from_result($query[, $params = array()])
```

参数

- **\$query** (*object*) – A database result object
- **\$params** (*array*) – An associative array of preferences

返回 The generated XML document as a string

返回类型 string

将数据库结果对象转换为 XML 文档。

9.1.14 数据库驱动器参考

这是一个平台无关的数据库实现基类，该类不会被直接调用，而是通过特定的数据库适配器类来继承和实现该类。

关于数据库驱动器，已经在其他几篇文档中介绍过，这篇文档将作为它们的一个参考。

重要： 并不是所有的方法都被所有的数据库驱动器所支持，当不支持的时候，有些方法可能会失败（返回 FALSE）。

```
class CI_DB_driver
```

```
    initialize()
```

返回 TRUE on success, FALSE on failure

返回类型 bool

初始化数据库配置，建立对数据库的连接。

```
    db_connect($persistent = TRUE)
```

参数

- **\$persistent** (*bool*) – Whether to establish a persistent connection or a regular one

返回 Database connection resource/object or FALSE on failure

返回类型 mixed

建立对数据库的连接。

注解: 返回值取决于当前使用的数据库驱动器, 例如 `mysqli` 实例将会返回 `'mysqli'` 驱动器。

db_pconnect()

返回 Database connection resource/object or FALSE on failure

返回类型 mixed

建立对数据库的连接, 使用持久连接。

注解: 该方法其实就是调用 `db_connect(TRUE)` 。

reconnect()

返回 TRUE on success, FALSE on failure

返回类型 bool

如果超过服务器的超时时间都没有发送任何查询请求, 使用该方法可以让数据库连接保持有效, 或重新连接数据库。

db_select([*\$database* = ''])**参数**

- **\$database** (*string*) – Database name

返回 TRUE on success, FALSE on failure

返回类型 bool

切换到某个数据库。

db_set_charset(*\$charset*)**参数**

- **\$charset** (*string*) – Character set name

返回 TRUE on success, FALSE on failure

返回类型 bool

设置客户端字符集。

platform()

返回 Platform name

返回类型 string

当前使用的数据库平台 (mysql、mssql 等)。

version()

返回 The version of the database being used

返回类型 string

数据库版本。

query(\$sql[, \$binds = FALSE[, \$return_object = NULL]])

参数

- **\$sql** (*string*) – The SQL statement to execute
- **\$binds** (*array*) – An array of binding data
- **\$return_object** (*bool*) – Whether to return a result object or not

返回 TRUE for successful “write-type” queries, CI_DB_result instance (method chaining) on “query” success, FALSE on failure

返回类型 mixed

执行一个 SQL 查询。

如果是读类型的查询，执行 SQL 成功后将返回结果对象。

有以下几种可能的返回值：

- 如果是写类型的查询，执行成功返回 TRUE
- 执行失败返回 FALSE
- 如果是读类型的查询，执行成功返回 CI_DB_result 对象

simple_query(\$sql)

参数

- **\$sql** (*string*) – The SQL statement to execute

返回 Whatever the underlying driver’s “query” function returns

返回类型 mixed

query() 方法的简化版，当你只需要简单的执行一个查询，并不关心查询的结果时，可以使用该方法。

trans_strict(\$mode = TRUE)

参数

- **\$mode** (*bool*) – Strict mode flag

返回类型 void

启用或禁用事务的严格模式。

在严格模式下, 如果你正在运行多组事务, 只要有一组失败, 所有组都会被回滚。

如果禁用严格模式, 那么每一组都被视为独立的组, 这意味着其中一组失败不会影响其他的组。

trans_off()

返回类型 void

实时的禁用事务。

trans_start(*[\$test_mode = FALSE]*)

参数

- **\$test_mode** (*bool*) – Test mode flag

返回类型 void

开启一个事务。

trans_complete()

返回类型 void

结束事务。

trans_status()

返回 TRUE if the transaction succeeded, FALSE if it failed

返回类型 bool

获取事务的状态, 用来判断事务是否执行成功。

compile_binds(*\$sql, \$binds*)

参数

- **\$sql** (*string*) – The SQL statement
- **\$binds** (*array*) – An array of binding data

返回 The updated SQL statement

返回类型 string

根据绑定的参数值编译 SQL 查询。

is_write_type(*\$sql*)

参数

- **\$sql** (*string*) – The SQL statement

返回 TRUE if the SQL statement is of “write type”, FALSE if not

返回类型 bool

判断查询是写类型 (INSERT、UPDATE、DELETE), 还是读类型 (SELECT)。

elapsed_time(*[\$decimals = 6]*)

参数

- **\$decimals** (*int*) – The number of decimal places

返回 The aggregate query elapsed time, in microseconds

返回类型 string

计算查询所消耗的时间。

total_queries()

返回 The total number of queries executed

返回类型 int

返回当前已经执行了多少次查询。

last_query()

返回 The last query executed

返回类型 string

返回上一次执行的查询。

escape(*\$str*)

参数

- **\$str** (*mixed*) – The value to escape, or an array of multiple ones

返回 The escaped value(s)

返回类型 mixed

根据输入数据的类型进行数据转义, 包括布尔值和空值。

escape_str(*\$str*, *\$like = FALSE*)

参数

- **\$str** (*mixed*) – A string value or array of multiple ones
- **\$like** (*bool*) – Whether or not the string will be used in a LIKE condition

返回 The escaped string(s)

返回类型 mixed

转义字符串。

警告: 返回的字符串没有用引号引起来。

escape_like_str(*\$str*)

参数

- **\$str** (*mixed*) – A string value or array of multiple ones

返回 The escaped string(s)

返回类型 mixed

转义 LIKE 字符串。

和 `escape_str()` 方法类似, 但同时也对 LIKE 语句中的 % 和 - 通配符进行转义。

primary(\$table)

参数

- **\$table** (*string*) – Table name

返回 The primary key name, FALSE if none

返回类型 string

获取一个表的主键。

注解: 如果数据库不支持主键检测, 将假设第一列就是主键。

count_all([\$table = ''])

参数

- **\$table** (*string*) – Table name

返回 Row count for the specified table

返回类型 int

返回表中的总记录数。

list_tables([\$constrain_by_prefix = FALSE])

参数

- **\$constrain_by_prefix** (*bool*) – TRUE to match table names by the configured dbprefix

返回 Array of table names or FALSE on failure

返回类型 array

返回当前数据库的所有表。

table_exists(\$table_name)

参数

- **\$table_name** (*string*) – The table name

返回 TRUE if that table exists, FALSE if not

返回类型 bool

判断某个数据库表是否存在。

list_fields(\$table)

参数

- **\$table** (*string*) – The table name

返回 Array of field names or FALSE on failure

返回类型 array

返回某个表的所有字段名。

field_exists(\$field_name, \$table_name)

参数

- **\$table_name** (*string*) – The table name
- **\$field_name** (*string*) – The field name

返回 TRUE if that field exists in that table, FALSE if not

返回类型 bool

判断某个字段是否存在。

field_data(\$table)

参数

- **\$table** (*string*) – The table name

返回 Array of field data items or FALSE on failure

返回类型 array

获取某个表的所有字段信息。

escape_identifiers(\$item)

参数

- **\$item** (*mixed*) – The item or array of items to escape

返回 The input item(s), escaped

返回类型 mixed

对 SQL 标识符进行转义，例如列名、表名、关键字。

insert_string(\$table, \$data)

参数

- **\$table** (*string*) – The target table
- **\$data** (*array*) – An associative array of key/value pairs

返回 The SQL INSERT statement, as a string

返回类型 string

生成 INSERT 语句。

`update_string($table, $data, $where)`

参数

- **\$table** (*string*) – The target table
- **\$data** (*array*) – An associative array of key/value pairs
- **\$where** (*mixed*) – The WHERE statement conditions

返回 The SQL UPDATE statement, as a string

返回类型 string

生成 UPDATE 语句。

`call_function($function)`

参数

- **\$function** (*string*) – Function name

返回 The function result

返回类型 string

使用一种平台无关的方式执行一个原生的 PHP 函数。

`cache_set_path([$path = ''])`

参数

- **\$path** (*string*) – Path to the cache directory

返回类型 void

设置缓存路径。

`cache_on()`

返回 TRUE if caching is on, FALSE if not

返回类型 bool

启用数据库结果缓存。

`cache_off()`

返回 TRUE if caching is on, FALSE if not

返回类型 bool

禁用数据库结果缓存。

`cache_delete([$segment_one = '', $segment_two = ''])`

参数

- **\$segment_one** (*string*) – First URI segment
- **\$segment_two** (*string*) – Second URI segment

返回 TRUE on success, FALSE on failure

返回类型 bool

删除特定 URI 的缓存文件。

cache_delete_all()

返回 TRUE on success, FALSE on failure

返回类型 bool

删除所有缓存文件。

close()

返回类型 void

关闭数据库的连接。

display_error(*[\$error = '']*, *[\$swap = '']*, *[\$native = FALSE]*)

参数

- **\$error** (*string*) – The error message
- **\$swap** (*string*) – Any “swap” values
- **\$native** (*bool*) – Whether to localize the message

返回类型 void

返回 Displays the DB error screensends the application/views/errors/error_db.php template

显示一个错误信息，并终止脚本执行。

错误信息是使用 *application/views/errors/error_db.php* 文件中的模板来显示。

protect_identifiers(*\$item*, *[\$prefix_single = FALSE]*, *[\$protect_identifiers = NULL]*, *[\$field_exists = TRUE]*)

参数

- **\$item** (*string*) – The item to work with
- **\$prefix_single** (*bool*) – Whether to apply the dbprefix even if the input item is a single identifier
- **\$protect_identifiers** (*bool*) – Whether to quote identifiers
- **\$field_exists** (*bool*) – Whether the supplied item contains a field name or not

返回 The modified item

返回类型 string

根据配置的 *dbprefix* 参数，给列名或表名（可能是表别名）添加一个前缀。

为了处理包含路径的列名，必须要考虑一些逻辑。

例如下面的查询:

```
SELECT * FROM hostname.database.table.column AS c FROM hostname.database.table
```

或者下面这个查询，使用了表别名:

```
SELECT m.member_id, m.member_name FROM members AS m
```

由于列名可以包含四段（主机、数据库名、表名、字段名）或者有一个表别名的前缀，我们需要做点工作来判断这一点，才能将 *dbprefix* 插入到正确的位置。

该方法在查询构造器类中被广泛使用。

辅助函数参考

10.1 辅助函数参考

10.1.1 数组辅助函数

数组辅助函数文件包含了一些帮助你处理数组的函数。

- 加载辅助函数
- 可用函数

加载辅助函数

该辅助函数通过下面的代码加载：

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

可用函数

该辅助函数有下列可用函数：

```
element($item, $array[, $default = NULL])
```

参数

- **\$item** (*string*) – Item to fetch from the array
- **\$array** (*array*) – Input array
- **\$default** (*bool*) – What to return if the array isn't valid

返回 NULL on failure or the array item.

返回类型 mixed

该函数通过索引获取数组中的元素。它会测试索引是否设置并且有值，如果有值，函数将返回该值，如果没有值，默认返回 NULL 或返回通过第三个参数设置的默认值。

示例:

```
$array = array(
    'color' => 'red',
    'shape' => 'round',
    'size' => ''
);

echo element('color', $array); // returns "red"
echo element('size', $array, 'foobar'); // returns "foobar"
```

`elements($items, $array[, $default = NULL])`

参数

- **\$item** (*string*) – Item to fetch from the array
- **\$array** (*array*) – Input array
- **\$default** (*bool*) – What to return if the array isn't valid

返回 NULL on failure or the array item.

返回类型 mixed

该函数通过多个索引获取数组中的多个元素。它会测试每一个索引是否设置并且有值，如果其中某个索引没有值，返回结果中该索引所对应的元素将被置为 NULL，或者通过第三个参数设置的默认值。

示例:

```
$array = array(
    'color' => 'red',
    'shape' => 'round',
    'radius' => '10',
    'diameter' => '20'
);

$my_shape = elements(array('color', 'shape', 'height'), $array);
```

上面的函数返回的结果如下:

```
array(
    'color' => 'red',
    'shape' => 'round',
    'height' => NULL
);
```

你可以通过第三个参数设置任何你想要设置的默认值。

```
$my_shape = elements(array('color', 'shape', 'height'), $array, 'foobar');
```

上面的函数返回的结果如下:

```
array(
    'color'    => 'red',
    'shape'    => 'round',
    'height'   => 'foobar'
);
```

当你需要将 `$_POST` 数组传递到你的模型中时这将很有用, 这可以防止用户发送额外的数据被写入到你的数据库。

```
$this->load->model('post_model');
$this->post_model->update(
    elements(array('id', 'title', 'content'), $_POST)
);
```

从上例中可以看出, 只有 id、title、content 三个字段被更新。

```
random_element($array)
```

参数

- `$array (array)` – Input array

返回 A random element from the array

返回类型 mixed

传入一个数组, 并返回数组中随机的一个元素。

使用示例:

```
$quotes = array(
    "I find that the harder I work, the more luck I seem to have. - Thomas Jefferson",
    "Don't stay in bed, unless you can make money in bed. - George Burns",
    "We didn't lose the game; we just ran out of time. - Vince Lombardi",
    "If everything seems under control, you're not going fast enough. - Mario Andretti",
    "Reality is merely an illusion, albeit a very persistent one. - Albert Einstein",
    "Chance favors the prepared mind - Louis Pasteur"
);

echo random_element($quotes);
```

10.1.2 验证码辅助函数

验证码辅助函数文件包含了一些帮助你创建验证码图片的函数。

- 加载辅助函数
- 使用验证码辅助函数
 - 添加到数据库
- 可用函数

加载辅助函数

该辅助函数通过下面的代码加载:

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

使用验证码辅助函数

辅助函数加载之后你可以像下面这样生成一个验证码图片:

```
$vals = array(
    'word'      => 'Random word',
    'img_path'  => './captcha/',
    'img_url'   => 'http://example.com/captcha/',
    'font_path' => './path/to/fonts/texb.ttf',
    'img_width' => '150',
    'img_height' => 30,
    'expiration' => 7200,
    'word_length' => 8,
    'font_size' => 16,
    'img_id'    => 'Imageid',
    'pool'      => '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ',

    // White background and border, black text and red grid
    'colors'    => array(
        'background' => array(255, 255, 255),
        'border'     => array(255, 255, 255),
        'text'       => array(0, 0, 0),
        'grid'       => array(255, 40, 40)
    )
);

$cap = create_captcha($vals);
echo $cap['image'];
```

- 验证码辅助函数需要使用 GD 图像库。
- 只有 `img_path` 和 `img_url` 这两个参数是必须的。
- 如果没有提供 `word` 参数, 该函数将生成一个随机的 ASCII 字符串。你也可以使用自己的词库, 从里面随机挑选。
- 如果你不设置 TRUE TYPE 字体 (译者注: 是主要的三种计算机矢量字体之一) 的路径, 将使用 GD 默认的字體。

- “captcha” 目录必须是可写的。
- **expiration** 参数表示验证码图片在删除之前将保留多久（单位为秒），默认保留 2 小时。
- **word_length** 默认值为 8，**pool** 默认值为 ‘0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ’
- **font_size** 默认值为 16，GD 库的字体对大小有限制，如果字体大小需要更大一点的话可以设置一种 TRUE TYPE 字体。
- **img_id** 将会设置为验证码图片的 “id”。
- **colors** 数组中如果有某个颜色未设置，将使用默认颜色。

添加到数据库

使用验证码函数是为了防止用户胡乱提交，要做到这一点，你需要将 `create_captcha()` 函数返回的信息保存到数据库中。然后，等用户提交表单数据时，通过数据库中保存的数据进行验证，并确保它没有过期。

这里是数据表的一个例子：

```
CREATE TABLE captcha (
    captcha_id bigint(13) unsigned NOT NULL auto_increment,
    captcha_time int(10) unsigned NOT NULL,
    ip_address varchar(45) NOT NULL,
    word varchar(20) NOT NULL,
    PRIMARY KEY `captcha_id` (`captcha_id`),
    KEY `word` (`word`)
);
```

这里是使用数据库的示例。在显示验证码的那个页面，你的代码类似于下面这样：

```
$this->load->helper('captcha');
$vals = array(
    'img_path' => './captcha/',
    'img_url' => 'http://example.com/captcha/'
);

$cap = create_captcha($vals);
$data = array(
    'captcha_time' => $cap['time'],
    'ip_address' => $this->input->ip_address(),
    'word' => $cap['word']
);

$query = $this->db->insert_string('captcha', $data);
$this->db->query($query);

echo 'Submit the word you see below:';
echo $cap['image'];
echo '<input type="text" name="captcha" value="" />';
```

然后在处理用户提交的页面, 处理如下:

```
// First, delete old captchas
$expiration = time() - 7200; // Two hour limit
$this->db->where('captcha_time < ', $expiration)
    ->delete('captcha');

// Then see if a captcha exists:
$sql = 'SELECT COUNT(*) AS count FROM captcha WHERE word = ? AND ip_address = ? AND captcha_time < ?';
$binds = array($_POST['captcha'], $this->input->ip_address(), $expiration);
$query = $this->db->query($sql, $binds);
$row = $query->row();

if ($row->count == 0)
{
    echo 'You must submit the word that appears in the image.';
}
```

可用函数

该辅助函数有下列可用函数:

```
create_captcha([ $data = '[', $img_path = '[', $img_url = '[', $font_path = ''] ]
```

参数

- **\$data** (*array*) – Array of data for the CAPTCHA
- **\$img_path** (*string*) – Path to create the image in
- **\$img_url** (*string*) – URL to the CAPTCHA image folder
- **\$font_path** (*string*) – Server path to font

返回 array('word' => \$word, 'time' => \$now, 'image' => \$img)

返回类型 array

根据你提供的一系列参数生成一张验证码图片, 返回包含此图片信息的数组。

```
array(
    'image' => IMAGE TAG
    'time'   => TIMESTAMP (in microtime)
    'word'   => CAPTCHA WORD
)
```

image 就是一个 image 标签:

```

```

time 是一个毫秒级的时间戳，作为图片的文件名（不带扩展名）。就像这样：1139612155.3422

word 是验证码图片中的文字，如果在函数的参数中没有指定 **word** 参数，这将是一个随机字符串。

10.1.3 Cookie 辅助函数

Cookie 辅助函数文件包含了一些帮助你处理 Cookie 的函数。

- 加载辅助函数
- 可用函数

加载辅助函数

该辅助函数通过下面的代码加载：

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

可用函数

该辅助函数有下列可用函数：

```
set_cookie($name[, $value = '', $expire = '', $domain = '', $path = '/', $prefix
```

参数

- **\$name** (*mixed*) – Cookie name *or* associative array of all of the parameters available to this function
- **\$value** (*string*) – Cookie value
- **\$expire** (*int*) – Number of seconds until expiration
- **\$domain** (*string*) – Cookie domain (usually: .yourdomain.com)
- **\$path** (*string*) – Cookie path
- **\$prefix** (*string*) – Cookie name prefix
- **\$secure** (*bool*) – Whether to only send the cookie through HTTPS
- **\$httponly** (*bool*) – Whether to hide the cookie from JavaScript

返回类型 void

该辅助函数提供给你一种更友好的语法来设置浏览器 Cookie，参考[输入类](#) 获取它的详细用法，另外，它是 `CI_Input::set_cookie()` 函数的别名。

```
get_cookie($index[, $xss_clean = NULL])
```

参数

- **\$index** (*string*) – Cookie name
- **\$xss_clean** (*bool*) – Whether to apply XSS filtering to the returned value

返回 The cookie value or NULL if not found

返回类型 mixed

该辅助函数提供给你一种更友好的语法来获取浏览器 Cookie, 参考[输入类](#) 获取它的详细用法, 同时, 这个函数和 `CI_Input::cookie()` 函数非常类似, 只是它会根据配置文件 `application/config/config.php` 中的 `$config['cookie_prefix']` 参数来作为 Cookie 的前缀。

```
delete_cookie($name[, $domain = '', $path = '/', $prefix = ''])
```

参数

- **\$name** (*string*) – Cookie name
- **\$domain** (*string*) – Cookie domain (usually: .yourdomain.com)
- **\$path** (*string*) – Cookie path
- **\$prefix** (*string*) – Cookie name prefix

返回类型 void

删除一条 Cookie, 只需要传入 Cookie 名即可, 也可以设置路径或其他参数来删除特定 Cookie。

```
delete_cookie('name');
```

这个函数和 `set_cookie()` 比较类似, 只是它并不提供 Cookie 的值和过期时间等参数。第一个参数也可以是个数组, 包含多个要删除的 Cookie。另外, 你也可以像下面这样删除特定条件的 Cookie。

```
delete_cookie($name, $domain, $path, $prefix);
```

10.1.4 日期辅助函数

日期辅助函数文件包含了一些帮助你处理日期的函数。

- [加载辅助函数](#)
- [可用函数](#)
- [时区参考](#)

加载辅助函数

该辅助函数通过下面的代码加载:

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

可用函数

该辅助函数有下列可用函数:

```
now([ $timezone = NULL ])
```

参数

- **\$timezone** (*string*) – Timezone

返回 UNIX timestamp

返回类型 int

根据服务器的本地时间, 以及一个 PHP 支持的时区参数或配置文件中的“基准时间”参数返回当前时间的 UNIX 时间戳, 如果你不打算设置“基准时间”(如果你的站点允许用户设置他们自己的时区, 你通常需要设置这个), 该函数就和 PHP 的 `time()` 函数没什么区别。

```
echo now('Australia/Victoria');
```

如果没有指定时区, 该函数将使用 **time_reference** 参数调用 `time()` 函数。

```
mdate([ $datestr = '[', $time = '']])
```

参数

- **\$datestr** (*string*) – Date string
- **\$time** (*int*) – UNIX timestamp

返回 MySQL-formatted date

返回类型 string

该函数和 PHP 的 `date()` 函数一样, 但是它支持 MySQL 风格的日期格式, 在代码之前使用百分号, 例如: `%Y %m %d`

使用这个函数的好处是你不用关心去转义那些不是日期代码的字符, 如果使用 `date()` 函数时, 你就要这么做。

例如:

```
$datestring = 'Year: %Y Month: %m Day: %d - %h:%i %a';
$time = time();
echo mdate($datestring, $time);
```

如果第二个参数没有提供一个时间, 那么默认会使用当前时间。

```
standard_date([ $fmt = 'DATE_RFC822', $time = NULL ])
```

参数

- **\$fmt** (*string*) – Date format

- **\$time** (*int*) – UNIX timestamp

返回 Formatted date or FALSE on invalid format

返回类型 string

生成标准格式的时间字符串。

例如:

```
$format = 'DATE_RFC822';
$time = time();
echo standard_date($format, $time);
```

注解: 该函数已经废弃, 请使用原生的 `date()` 函数和 [时间格式化常量](#) 替代:

```
echo date(DATE_RFC822, time());
```

支持的格式:

Constant	Description	Example
DATE_ATOM	Atom	2005-08-15T16:13:03+0000
DATE_COOKIE	HTTP Cookies	Sun, 14 Aug 2005 16:13:03 UTC
DATE_ISO8601	ISO-8601	2005-08-14T16:13:03+00:00
DATE_RFC822	RFC 822	Sun, 14 Aug 05 16:13:03 UTC
DATE_RFC850	RFC 850	Sunday, 14-Aug-05 16:13:03 UTC
DATE_RFC1036	RFC 1036	Sunday, 14-Aug-05 16:13:03 UTC
DATE_RFC1123	RFC 1123	Sun, 14 Aug 2005 16:13:03 UTC
DATE_RFC2822	RFC 2822	Sun, 14 Aug 2005 16:13:03 +0000
DATE_RSS	RSS	Sun, 14 Aug 2005 16:13:03 UTC
DATE_W3C	W3C	2005-08-14T16:13:03+0000

```
local_to_gmt([ $time = '' ])
```

参数

- **\$time** (*int*) – UNIX timestamp

返回 UNIX timestamp

返回类型 int

将时间转换为 GMT 时间。

例如:

```
$gmt = local_to_gmt(time());
```

```
gmt_to_local([ $time = '', $timezone = 'UTC', $dst = FALSE ])
```

参数

- **\$time** (*int*) – UNIX timestamp
- **\$timezone** (*string*) – Timezone
- **\$dst** (*bool*) – Whether DST is active

返回 UNIX timestamp

返回类型 int

根据指定的时区和 DST（夏令时，Daylight Saving Time）将 GMT 时间转换为本地时间。

例如:

```
$timestamp = 1140153693;
$timezone = 'UM8';
$daylight_saving = TRUE;
echo gmt_to_local($timestamp, $timezone, $daylight_saving);
```

注解: 时区列表请参见本页末尾。

`mysql_to_unix([$time = ''])`

参数

- **\$time** (*string*) – MySQL timestamp

返回 UNIX timestamp

返回类型 int

将 MySQL 时间戳转换为 UNIX 时间戳。

例如:

```
$unix = mysql_to_unix('20061124092345');
```

`unix_to_human([$time = '', $seconds = FALSE, $fmt = 'us']])`

参数

- **\$time** (*int*) – UNIX timestamp
- **\$seconds** (*bool*) – Whether to show seconds
- **\$fmt** (*string*) – format (us or euro)

返回 Formatted date

返回类型 string

将 UNIX 时间戳转换为方便人类阅读的格式，如下:

YYYY-MM-DD HH:MM:SS AM/PM

这在当你需要在一个表单字段中显示日期时很有用。

格式化后的时间可以带也可以不带秒数，也可以设置成欧洲或美国时间格式。如果只指定了一个时间参数，将使用不带秒数的美国时间格式。

例如:

```
$now = time();  
echo unix_to_human($now); // U.S. time, no seconds  
echo unix_to_human($now, TRUE, 'us'); // U.S. time with seconds  
echo unix_to_human($now, TRUE, 'eu'); // Euro time with seconds
```

```
human_to_unix([ $datestr = '' ])
```

参数

- **\$datestr** (*int*) – Date string

返回 UNIX timestamp or FALSE on failure

返回类型 int

该函数和 `unix_to_human()` 函数相反, 将一个方便人类阅读的时间格式转换为 UNIX 时间戳。这在当你需要在一个表单字段中显示日期时很有用。如果输入的时间不同于上面的格式, 函数返回 FALSE。

例如:

```
$now = time();  
$human = unix_to_human($now);  
$unix = human_to_unix($human);
```

```
nice_date([ $bad_date = '', $format = FALSE ])
```

参数

- **\$bad_date** (*int*) – The terribly formatted date-like string
- **\$format** (*string*) – Date format to return (same as PHP's `date()` function)

返回 Formatted date

返回类型 string

该函数解析一个没有格式化过的数字格式的日期, 并将其转换为格式化的日期。它也能解析格式化好的日期。

默认该函数将返回 UNIX 时间戳, 你也可以提供一个格式化字符串给第二个参数 (和 PHP 的 `date()` 函数一样)。

例如:

```
$bad_date = '199605';  
// Should Produce: 1996-05-01  
$better_date = nice_date($bad_date, 'Y-m-d');  
  
$bad_date = '9-11-2001';  
// Should Produce: 2001-09-11  
$better_date = nice_date($bad_date, 'Y-m-d');
```

```
timespan([ $seconds = 1, $time = '', $units = '' ])
```

参数

- **\$seconds** (*int*) – Number of seconds
- **\$time** (*string*) – UNIX timestamp
- **\$units** (*int*) – Number of time units to display

返回 Formatted time difference

返回类型 string

将一个 UNIX 时间戳转换为以下这种格式:

1 Year, 10 Months, 2 Weeks, 5 Days, 10 Hours, 16 Minutes

第一个参数为一个 UNIX 时间戳, 第二个参数是一个比第一个参数大的 UNIX 时间戳。第三个参数可选, 用于限制要显示的时间单位个数。

如果第二个参数为空, 将使用当前时间。

这个函数最常见的用途是, 显示从过去某个时间点到当前时间经过了多少时间。

例如:

```
$post_date = '1079621429';
$now = time();
$units = 2;
echo timespan($post_date, $now, $units);
```

注解: 该函数生成的本文可以在语言文件 `language/<your_lang>/date_lang.php` 中找到。

days_in_month(`[$month = 0`, `$year = ''`])

参数

- **\$month** (*int*) – a numeric month
- **\$year** (*int*) – a numeric year

返回 Count of days in the specified month

返回类型 int

返回指定某个月的天数, 会考虑闰年。

例如:

```
echo days_in_month(06, 2005);
```

如果第二个参数为空, 将使用今年。

注解: 该函数其实是原生的 `cal_days_in_month()` 函数的别名, 如果它可用的话。

date_range(`[$unix_start = ''`, `$mixed = ''`, `$is_unix = TRUE`, `$format = 'Y-m-d'`
`]]]]`)

参数

- **\$unix_start** (*int*) – UNIX timestamp of the range start date
- **\$mixed** (*int*) – UNIX timestamp of the range end date or interval in days
- **\$is_unix** (*bool*) – set to FALSE if \$mixed is not a timestamp
- **\$format** (*string*) – Output date format, same as in `date()`

返回 An array of dates

返回类型 array

返回某一段时间的日期列表。

例如:

```
$range = date_range('2012-01-01', '2012-01-15');
echo "First 15 days of 2012:";
foreach ($range as $date)
{
    echo $date."\n";
}
```

`timezones`(`[$tz = '']`)

参数

- **\$tz** (*string*) – A numeric timezone

返回 Hour difference from UTC

返回类型 int

根据指定的时区（可用的时区列表参见下文的“时区参考”）返回它的 UTC 时间偏移。

例如:

```
echo timezones('UM5');
```

这个函数和`timezone_menu()` 函数一起使用时很有用。

```
timezone_menu([ $default = 'UTC', $class = '', $name = 'timezones', $attributes = '' ])
```

参数

- **\$default** (*string*) – Timezone
- **\$class** (*string*) – Class name
- **\$name** (*string*) – Menu name
- **\$attributes** (*mixed*) – HTML attributes

返回 HTML drop down menu with time zones

返回类型 string

该函数用于生成一个时区下拉菜单，像下面这样。

当你的站点允许用户选择自己的本地时区时，这个菜单会很有用。

第一个参数为菜单默认选定的时区，例如，要设置太平洋时间为默认值，你可以这样：

```
echo timezone_menu('UM8');
```

菜单中的值请参见下面的时区参考。

第二个参数用于为菜单设置一个 CSS 类名。

第四个参数用于为生成的 select 标签设置一个或多个属性。

注解： 菜单中的文本可以在语言文件 `language/<your_lang>/date_lang.php` 中找到。

时区参考

下表列出了每个时区和它所对应的位置。

注意，为了表述清晰和格式工整，有些位置信息做了适当的删减。

时区	位置
UM12	(UTC - 12:00) 贝克岛、豪兰岛
UM11	(UTC - 11:00) 萨摩亚时区、纽埃
UM10	(UTC - 10:00) 夏威夷 -阿留申标准时间、库克群岛
UM95	(UTC - 09:30) 马克萨斯群岛
UM9	(UTC - 09:00) 阿拉斯加标准时间、甘比尔群岛
UM8	(UTC - 08:00) 太平洋标准时间、克利珀顿岛
UM7	(UTC - 07:00) 山区标准时间
UM6	(UTC - 06:00) 中部标准时间
UM5	(UTC - 05:00) 东部标准时间、西加勒比
UM45	(UTC - 04:30) 委内瑞拉标准时间
UM4	(UTC - 04:00) 大西洋标准时间、东加勒比
UM35	(UTC - 03:30) 纽芬兰标准时间
UM3	(UTC - 03:00) 阿根廷、巴西、法属圭亚那、乌拉圭
UM2	(UTC - 02:00) 南乔治亚岛、南桑威奇群岛
UM1	(UTC -1:00) 亚速尔群岛、佛得角群岛
UTC	(UTC) 格林尼治标准时间、西欧时间
UP1	(UTC +1:00) 中欧时间、西非时间
UP2	(UTC +2:00) 中非时间、东欧时间
UP3	(UTC +3:00) 莫斯科时间、东非时间
UP35	(UTC +3:30) 伊朗标准时间
UP4	(UTC +4:00) 阿塞拜疆标准时间、萨马拉时间
UP45	(UTC +4:30) 阿富汗
UP5	(UTC +5:00) 巴基斯坦标准时间、叶卡捷琳堡时间
下页继续	

Table 10.1 – 续上页

时区	位置
UP55	(UTC +5:30) 印度标准时间、斯里兰卡时间
UP575	(UTC +5:45) 尼泊尔时间
UP6	(UTC +6:00) 孟加拉国标准时间、不丹时间、鄂木斯克时间
UP65	(UTC +6:30) 可可岛、缅甸
UP7	(UTC +7:00) 克拉斯诺亚尔斯克时间、柬埔寨、老挝、泰国、越南
UP8	(UTC +8:00) 澳大利亚西部标准时间、北京时间
UP875	(UTC +8:45) 澳大利亚中西部标准时间
UP9	(UTC +9:00) 日本标准时间、韩国标准时间、雅库茨克
UP95	(UTC +9:30) 澳大利亚中部标准时间
UP10	(UTC +10:00) 澳大利亚东部标准时间、海参崴时间
UP105	(UTC +10:30) 豪勋爵岛
UP11	(UTC +11:00) 中科雷姆斯克时间、所罗门群岛、瓦努阿图
UP115	(UTC +11:30) 诺福克岛
UP12	(UTC +12:00) 斐济、吉尔伯特群岛、堪察加半岛、新西兰
UP1275	(UTC +12:45) 查塔姆群岛标准时间
UP13	(UTC +13:00) 凤凰岛、汤加
UP14	(UTC +14:00) 莱恩群岛

10.1.5 目录辅助函数

目录辅助函数文件包含了一些帮助你处理目录的函数。

- 加载辅助函数
- 可用函数

加载辅助函数

该辅助函数通过下面的代码加载:

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

可用函数

该辅助函数有下列可用函数:

```
directory_map($source_dir[, $directory_depth = 0[, $hidden = FALSE]])
```

参数

- **\$source_dir** (*string*) – Path to the source directory
- **\$directory_depth** (*int*) – Depth of directories to traverse (0 = fully recursive, 1 = current dir, etc)
- **\$hidden** (*bool*) – Whether to include hidden directories

返回 An array of files

返回类型 array

举例:

```
$map = directory_map('./mydirectory/');
```

注解: 路径总是相对于你的 index.php 文件。

如果目录内含有子目录, 也将被列出。你可以使用第二个参数 (整数) 来控制递归的深度。如果深度为 1, 则只列出根目录:

```
$map = directory_map('./mydirectory/', 1);
```

默认情况下, 返回的数组中不会包括那些隐藏文件。如果需要显示隐藏的文件, 你可以设置第三个参数为 true

```
$map = directory_map('./mydirectory/', FALSE, TRUE);
```

每一个目录的名字都将作为数组的索引, 目录所包含的文件将以数字作为索引。下面有个典型的数组示例:

```
Array (
    [libraries] => Array
        (
            [0] => benchmark.html
            [1] => config.html
            ["database/" ] => Array
                (
                    [0] => query_builder.html
                    [1] => binds.html
                    [2] => configuration.html
                    [3] => connecting.html
                    [4] => examples.html
                    [5] => fields.html
                    [6] => index.html
                    [7] => queries.html
                )
            [2] => email.html
            [3] => file_uploading.html
            [4] => image_lib.html
            [5] => input.html
            [6] => language.html
            [7] => loader.html
            [8] => pagination.html
            [9] => uri.html
        )
)
```


10.1.6 下载辅助函数

下载辅助函数文件包含了下载相关的一些函数。

- 加载辅助函数
- 可用函数

加载辅助函数

该辅助函数通过下面的代码加载:

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

可用函数

该辅助函数有下列可用函数:

```
force_download([$filename = '', $data = '', $set_mime = FALSE]])
```

参数

- **\$filename** (*string*) – Filename
- **\$data** (*mixed*) – File contents
- **\$set_mime** (*bool*) – Whether to try to send the actual MIME type

返回类型 void

生成 HTTP 头强制下载数据到客户端，这在实现文件下载时很有用。第一个参数为下载文件名称，第二个参数为文件数据。

如果第二个参数为空，并且 **\$filename** 参数是一个存在并可读的文件路径，那么这个文件的内容将被下载。

如果第三个参数设置为 TRUE，那么将发送文件实际的 MIME 类型（根据文件的扩展名），这样你的浏览器会根据该 MIME 类型来处理。

Example:

```
$data = 'Here is some text!';  
$name = 'mytext.txt';  
force_download($name, $data);
```

下载一个服务器上已存在的文件的例子如下:

```
// Contents of photo.jpg will be automatically read  
force_download('/path/to/photo.jpg', NULL);
```

10.1.7 邮件辅助函数

邮件辅助函数文件包含了用于处理邮件的一些函数。欲了解关于邮件更全面的解决方案，可以参考 CodeIgniter 的 *Email* 类。

重要： 不鼓励继续使用邮件辅助函数，这个库当前仅是为了向前兼容而存在。

- 加载辅助函数
- 可用函数

加载辅助函数

该辅助函数通过下面的代码加载：

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

可用函数

该辅助函数有下列可用函数：

valid_email(\$email)

参数

- **\$email** (*string*) – E-mail address

返回 TRUE if a valid email is supplied, FALSE otherwise

返回类型 bool

检查 Email 地址格式是否正确，注意该函数只是简单的检查它的格式是否正确，并不能保证该 Email 地址能接受到邮件。

Example:

```
if (valid_email('email@somesite.com'))
{
    echo 'email is valid';
}
else
{
    echo 'email is not valid';
}
```

注解： 该函数实际上就是调用 PHP 原生的 `filter_var()` 函数而已：

```
(bool) filter_var($email, FILTER_VALIDATE_EMAIL);
```

send_email(\$recipient, \$subject, \$message)

参数

- **\$recipient** (*string*) – E-mail address
- **\$subject** (*string*) – Mail subject
- **\$message** (*string*) – Message body

返回 TRUE if the mail was successfully sent, FALSE in case of an error

返回类型 bool

使用 PHP 函数 `mail()` 发送邮件。

注解: 该函数实际上就是调用 PHP 原生的 `mail()` 函数而已

```
mail($recipient, $subject, $message);
```

欲了解关于邮件更全面的解决方案, 可以参考 CodeIgniter 的 *Email* 类。

10.1.8 文件辅助函数

文件辅助函数文件包含了一些帮助你处理文件的函数。

- 加载辅助函数
 - 可用函数

加载辅助函数

该辅助函数通过下面的代码加载:

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

可用函数

该辅助函数有下列可用函数:

read_file(\$file)

参数

- **\$file** (*string*) – File path

返回 File contents or FALSE on failure

返回类型 string

返回指定文件的内容。

例如:

```
$string = read_file('./path/to/file.php');
```

可以是相对路径或绝对路径，如果失败返回 FALSE 。

注解： 路径是相对于你网站的 index.php 文件的，而不是相对于控制器或视图文件。这是因为 CodeIgniter 使用的前端控制器，所以所有的路径都是相对于 index.php 所在路径。

注解： 该函数已废弃，使用 PHP 的原生函数 `file_get_contents()` 代替。

重要： 如果你的服务器配置了 `open_basedir` 限制，该函数可能无法访问限制之外的文件。

```
write_file($path, $data[, $mode = 'wb'])
```

参数

- **\$path** (*string*) – File path
- **\$data** (*string*) – Data to write to file
- **\$mode** (*string*) – `fopen()` mode

返回 TRUE if the write was successful, FALSE in case of an error

返回类型 bool

向指定文件中写入数据，如果文件不存在，则创建该文件。

例如：

```
$data = 'Some file data';
if ( ! write_file('./path/to/file.php', $data) )
{
    echo 'Unable to write the file';
}
else
{
    echo 'File written!';
}
```

你还可以通过第三个参数设置写模式：

```
write_file('./path/to/file.php', $data, 'r+');
```

默认的模式为 'wb'，请阅读 [PHP 用户指南](#) 了解写模式的选项。

注解： 路径是相对于你网站的 index.php 文件的，而不是相对于控制器或视图文件。这是因为 CodeIgniter 使用的前端控制器，所以所有的路径都是相对于 index.php 所在路径。

注解: 该函数在写入文件时会申请一个排他性锁。

`delete_files($path[, $del_dir = FALSE[, $htdocs = FALSE]])`

参数

- **\$path** (*string*) – Directory path
- **\$del_dir** (*bool*) – Whether to also delete directories
- **\$htdocs** (*bool*) – Whether to skip deleting .htaccess and index page files

返回 TRUE on success, FALSE in case of an error

返回类型 bool

删除指定路径下的所有文件。

例如:

```
delete_files('./path/to/directory/');
```

如果第二个参数设置为 TRUE , 那么指定路径下的文件夹也一并删除。

例如:

```
delete_files('./path/to/directory/', TRUE);
```

注解: 要被删除的文件必须是当前系统用户所有或者是当前用户对之具有写权限。

`get_filenames($source_dir[, $include_path = FALSE])`

参数

- **\$source_dir** (*string*) – Directory path
- **\$include_path** (*bool*) – Whether to include the path as part of the filenames

返回 An array of file names

返回类型 array

获取指定目录下所有文件名组成的数组。如果需要完整路径的文件名, 可以将第二个参数设置为 TRUE 。

例如:

```
$controllers = get_filenames(APPPATH.'controllers/');
```

`get_dir_file_info($source_dir, $top_level_only)`

参数

- **\$source_dir** (*string*) – Directory path

- **Stop_level_only** (*bool*) – Whether to look only at the specified directory (excluding sub-directories)

返回 An array containing info on the supplied directory's contents

返回类型 array

获取指定目录下所有文件信息组成的数组，包括文件名、文件大小、日期和权限。默认不包含子目录下的文件信息，如有需要，可以设置第二个参数为 FALSE，这可能会是一个耗时的操作。

例如:

```
$models_info = get_dir_file_info(APPPATH.'models/');
```

```
get_file_info($file[, $returned_values = array('name', 'server_path', 'size', 'date')
    ])
```

参数

- **\$file** (*string*) – File path
- **\$returned_values** (*array*) – What type of info to return

返回 An array containing info on the specified file or FALSE on failure

返回类型 array

获取指定文件的信息，包括文件名、路径、文件大小，修改日期等。第二个参数可以用于声明只返回你想要的信息。

第二个参数 **\$returned_values** 有效的值有: *name*、*size*、*date*、*readable*、*writable*、*executable* 和 *fileperms*。

```
get_mime_by_extension($filename)
```

参数

- **\$filename** (*string*) – File name

返回 MIME type string or FALSE on failure

返回类型 string

根据 *config/mimes.php* 文件中的配置将文件扩展名转换为 MIME 类型。如果无法判断 MIME 类型或 MIME 配置文件读取失败，则返回 FALSE。

```
$file = 'somefile.png';
echo $file.' is has a mime type of '.get_mime_by_extension($file);
```

注解: 这个函数只是一种简便的判断 MIME 类型的方法，并不准确，所以请不要用于安全相关的地方。

```
symbolic_permissions($perms)
```

参数

- **\$perms** (*int*) – Permissions

返回 Symbolic permissions string

返回类型 string

将文件权限的数字格式（例如 `fileperms()` 函数的返回值）转换为标准的符号格式。

```
echo symbolic_permissions(fileperms('./index.php')); // -rw-r--r--
```

`octal_permissions($perms)`

参数

- `$perms (int)` – Permissions

返回 Octal permissions string

返回类型 string

将文件权限的数字格式（例如 `fileperms()` 函数的返回值）转换为三个字符的八进制表示格式。

```
echo octal_permissions(fileperms('./index.php')); // 644
```

10.1.9 表单辅助函数

表单辅助函数包含了一些函数用于帮助你处理表单。

- 加载辅助函数
- 对域值转义
- 可用函数

加载辅助函数

使用下面的代码来加载表单辅助函数：

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

对域值转义

你可能会需要在表单元素中使用 HTML 或者诸如引号这样的字符，为了安全性，你需要使用通用函数 `html_escape()`。

考虑下面这个例子：

```
$string = 'Here is a string containing "quoted" text.';
```

```
<input type="text" name="myfield" value="<?php echo $string; ?>" />
```

因为上面的字符串中包含了一对引号，它会破坏表单，使用`html_escape()` 函数可以对 HTML 的特殊字符进行转义，从而可以安全的在域值中使用字符串：

```
<input type="text" name="myfield" value="<?php echo html_escape($string); ?>" />
```

注解： 如果你使用了这个页面上介绍的任何一个函数，表单的域值会被自动转义，所以你无需再调用这个函数。只有在你创建自己的表单元素时需要使用它。

可用函数

该辅助函数有下列可用函数：

```
form_open([ $action = '', $attributes = '', $hidden = array() ] )
```

参数

- **\$action** (*string*) – Form action/target URI string
- **\$attributes** (*array*) – HTML attributes
- **\$hidden** (*array*) – An array of hidden fields' definitions

返回 An HTML form opening tag

返回类型 string

生成一个 form 起始标签，并且它的 action URL 会根据你的配置文件自动生成。你还可以给表单添加属性和隐藏域，另外，它还会根据你配置文件中的字符集参数自动生成 *accept-charset* 属性。

使用该函数来生成标签比你自己写 HTML 代码最大的好处是：当你的 URL 变动时，它可以提供更好的可移植性。

这里是个简单的例子：

```
echo form_open('email/send');
```

上面的代码会创建一个表单，它的 action 为根 URL 加上 “email/send”，向下面这样：

```
<form method="post" accept-charset="utf-8" action="http://example.com/index.php/email/"
```

添加属性

可以通过第二个参数传递一个关联数组来添加属性，例如：

```
$attributes = array('class' => 'email', 'id' => 'myform');
echo form_open('email/send', $attributes);
```

另外，第二个参数你也可以直接使用字符串：

```
echo form_open('email/send', 'class="email" id="myform"');
```

上面的代码会创建一个类似于下面的表单：


```
<form method="post" accept-charset="utf-8" action="http://example.com/index.php/em
```

添加隐藏域

可以通过第三个参数传递一个关联数组来添加隐藏域, 例如:

```
$hidden = array('username' => 'Joe', 'member_id' => '234');
echo form_open('email/send', '', $hidden);
```

你可以使用一个空值跳过第二个参数。

上面的代码会创建一个类似于下面的表单:

```
<form method="post" accept-charset="utf-8" action="http://example.com/index.php/em
    <input type="hidden" name="username" value="Joe" />
    <input type="hidden" name="member_id" value="234" />
```

```
form_open_multipart([$action = '', $attributes = array(), $hidden = array()])
```

参数

- **\$action** (*string*) – Form action/target URI string
- **\$attributes** (*array*) – HTML attributes
- **\$hidden** (*array*) – An array of hidden fields' definitions

返回 An HTML multipart form opening tag

返回类型 string

这个函数和上面的 `form_open()` 函数完全一样, 只是它会给表单添加一个 *multipart* 属性, 在你使用表单上传文件时必须使用它。

```
form_hidden($name[, $value = ''])
```

参数

- **\$name** (*string*) – Field name
- **\$value** (*string*) – Field value

返回 An HTML hidden input field tag

返回类型 string

生成隐藏域。你可以使用名称和值两个参数来创建一个隐藏域:

```
form_hidden('username', 'johndoe');
// Would produce: <input type="hidden" name="username" value="johndoe" />
```

... 或者你可以使用一个关联数组, 来生成多个隐藏域:

```
$data = array(
    'name' => 'John Doe',
    'email' => 'john@example.com',
    'url' => 'http://example.com'
);
```

```

echo form_hidden($data);

/*
    Would produce:
    <input type="hidden" name="name" value="John Doe" />
    <input type="hidden" name="email" value="john@example.com" />
    <input type="hidden" name="url" value="http://example.com" />
*/

```

你还可以向第二个参数传递一个关联数组:

```

$data = array(
    'name' => 'John Doe',
    'email' => 'john@example.com',
    'url' => 'http://example.com'
);

echo form_hidden('my_array', $data);

/*
    Would produce:

    <input type="hidden" name="my_array[name]" value="John Doe" />
    <input type="hidden" name="my_array[email]" value="john@example.com" />
    <input type="hidden" name="my_array[url]" value="http://example.com" />
*/

```

如果你想创建带有其他属性的隐藏域, 可以这样:

```

$data = array(
    'type' => 'hidden',
    'name' => 'email',
    'id' => 'hiddenemail',
    'value' => 'john@example.com',
    'class' => 'hiddenemail'
);

echo form_input($data);

/*
    Would produce:

    <input type="hidden" name="email" value="john@example.com" id="hiddenemail" class=
*/

```

```
form_input([$data = '', $value = '', $extra = ''])
```

参数

- **\$data** (*array*) – Field attributes data
- **\$value** (*string*) – Field value

- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML text input field tag

返回类型 string

用于生成标准的文本输入框，你可以简单的使用文本域的名称和值:

```
echo form_input('username', 'johndoe');
```

或者使用一个关联数组，来包含任何你想要的数据:

```
$data = array(
    'name'      => 'username',
    'id'        => 'username',
    'value'     => 'johndoe',
    'maxlength' => '100',
    'size'      => '50',
    'style'     => 'width:50%'
);

echo form_input($data);

/*
    Would produce:

    <input type="text" name="username" value="johndoe" id="username" maxlength="100" s
*/
```

如果你还希望能包含一些额外的数据，例如 JavaScript，你可以通过第三个参数传一个字符串:

```
$js = 'onClick="some_function()"';
echo form_input('username', 'johndoe', $js);
```

Or you can pass it as an array:

```
$js = array('onClick' => 'some_function()');
echo form_input('username', 'johndoe', $js);
```

```
form_password([ $data = '[', $value = '[', $extra = ''] ])
```

参数

- **\$data** (*array*) – Field attributes data
- **\$value** (*string*) – Field value
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML password input field tag

返回类型 string

该函数和上面的 `form_input()` 函数一样, 只是生成的输入框为 “password” 类型。

```
form_upload([$data = '', $value = '', $extra = ''])
```

参数

- **\$data** (*array*) – Field attributes data
- **\$value** (*string*) – Field value
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML file upload input field tag

返回类型 string

该函数和上面的 `form_input()` 函数一样, 只是生成的输入框为 “file” 类型, 可以用来上传文件。

```
form_textarea([$data = '', $value = '', $extra = ''])
```

参数

- **\$data** (*array*) – Field attributes data
- **\$value** (*string*) – Field value
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML textarea tag

返回类型 string

该函数和上面的 `form_input()` 函数一样, 只是生成的输入框为 “textarea” 类型。

注解: 对于 textarea 类型的输入框, 你可以使用 *rows* 和 *cols* 属性, 来代替上面例子中的 *maxlength* 和 *size* 属性。

```
form_dropdown($name = '', $options = array(), $selected = array(), $extra = '')
```

参数

- **\$name** (*string*) – Field name
- **\$options** (*array*) – An associative array of options to be listed
- **\$selected** (*array*) – List of fields to mark with the *selected* attribute
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML dropdown select field tag

返回类型 string

用于生成一个标准的下拉框域。第一个参数为域的名称，第二个参数为一个关联数组，包含所有的选项，第三个参数为你希望默认选中的值。你也可以把第三个参数设置成一个包含多个值的数组，CodeIgniter 将会为你生成多选下拉框。

例如:

```
$options = array(
    'small'    => 'Small Shirt',
    'med'      => 'Medium Shirt',
    'large'    => 'Large Shirt',
    'xlarge'   => 'Extra Large Shirt',
);

$shirts_on_sale = array('small', 'large');
echo form_dropdown('shirts', $options, 'large');

/*
    Would produce:

    <select name="shirts">
        <option value="small">Small Shirt</option>
        <option value="med">Medium Shirt</option>
        <option value="large" selected="selected">Large Shirt</option>
        <option value="xlarge">Extra Large Shirt</option>
    </select>
*/

echo form_dropdown('shirts', $options, $shirts_on_sale);

/*
    Would produce:

    <select name="shirts" multiple="multiple">
        <option value="small" selected="selected">Small Shirt</option>
        <option value="med">Medium Shirt</option>
        <option value="large" selected="selected">Large Shirt</option>
        <option value="xlarge">Extra Large Shirt</option>
    </select>
*/
```

如果你希望为起始标签 <select> 添加一些额外的数据，例如 id 属性或 JavaScript，你可以通过第四个参数传一个字符串:

```
$js = 'id="shirts" onChange="some_function();"';
echo form_dropdown('shirts', $options, 'large', $js);
```

Or you can pass it as an array:

```
$js = array(
    'id'          => 'shirts',
    'onChange'    => 'some_function();'
```

```
);
echo form_dropdown('shirts', $options, 'large', $js);
```

如果你传递的 `$options` 数组是个多维数组, `form_dropdown()` 函数将会生成带 `<optgroup>` 的下拉框, 并使用数组的键作为 label。

```
form_multiselect([ $name = ' ', $options = array(), $selected = array(), $extra
                  = ' ' ])
```

参数

- **\$name** (*string*) – Field name
- **\$options** (*array*) – An associative array of options to be listed
- **\$selected** (*array*) – List of fields to mark with the *selected* attribute
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML dropdown multiselect field tag

返回类型 string

用于生成一个标准的多选下拉框。第一个参数为域的名称, 第二个参数为一个关联数组, 包含所有的选项, 第三个参数为你希望默认选中的一个或多个值。

参数的用法和上面的 `form_dropdown()` 函数一样, 只是域的名称需要使用数组语法, 例如: `foo[]`

```
form_fieldset([ $legend_text = ' ', $attributes = array() ])
```

参数

- **\$legend_text** (*string*) – Text to put in the `<legend>` tag
- **\$attributes** (*array*) – Attributes to be set on the `<fieldset>` tag

返回 An HTML fieldset opening tag

返回类型 string

用于生成 fieldset 和 legend 域。

例如:

```
echo form_fieldset('Address Information');
echo "<p>fieldset content here</p>\n";
echo form_fieldset_close();
```

```
/*
```

```
Produces:
```

```
<fieldset>
  <legend>Address Information</legend>
  <p>form content here</p>
```

```

        </fieldset>
    */

```

和其他的函数类似, 你也可以通过给第二个参数传一个关联数组来添加额外的属性:

```

$attributes = array(
    'id'      => 'address_info',
    'class'   => 'address_info'
);

echo form_fieldset('Address Information', $attributes);
echo "<p>fieldset content here</p>\n";
echo form_fieldset_close();

/*
    Produces:

    <fieldset id="address_info" class="address_info">
        <legend>Address Information</legend>
        <p>fieldset content here</p>
    </fieldset>
*/

```

```
form_fieldset_close([$extra = ''])
```

参数

- **\$extra** (*string*) – Anything to append after the closing tag, as is

返回 An HTML fieldset closing tag

返回类型 string

用于生成结束标签 `</fieldset>`, 使用这个函数唯一的一个好处是, 它可以在结束标签的后面加上一些其他的数据。例如:

```

$string = '</div></div>';
echo form_fieldset_close($string);
// Would produce: </fieldset></div></div>

```

```
form_checkbox([$data = '', $value = '', $checked = FALSE, $extra = ''])
```

参数

- **\$data** (*array*) – Field attributes data
- **\$value** (*string*) – Field value
- **\$checked** (*bool*) – Whether to mark the checkbox as being checked
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML checkbox input tag

返回类型 string

用于生成一个复选框，例如：

```
echo form_checkbox('newsletter', 'accept', TRUE);
// Would produce: <input type="checkbox" name="newsletter" value="accept" checked="checked" />
```

第三个参数为布尔值 TRUE 或 FALSE，用于指定复选框默认是否为选定状态。

和其他函数一样，你可以传一个属性的数组给它：

```
$data = array(
    'name'      => 'newsletter',
    'id'        => 'newsletter',
    'value'     => 'accept',
    'checked'   => TRUE,
    'style'     => 'margin:10px'
);

echo form_checkbox($data);
// Would produce: <input type="checkbox" name="newsletter" id="newsletter" value="accept" checked="checked" />
```

另外，如果你希望向标签中添加额外的数据如 JavaScript，也可以传一个字符串给第四个参数：

```
$js = 'onClick="some_function()"';
echo form_checkbox('newsletter', 'accept', TRUE, $js)
```

Or you can pass it as an array:

```
$js = array('onClick' => 'some_function()');
echo form_checkbox('newsletter', 'accept', TRUE, $js)
```

```
form_radio([$data = '[', $value = '[', $checked = FALSE, $extra = ']]])
```

参数

- **\$data** (*array*) – Field attributes data
- **\$value** (*string*) – Field value
- **\$checked** (*bool*) – Whether to mark the radio button as being *checked*
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML radio input tag

返回类型 string

该函数和 `form_checkbox()` 函数完全一样，只是它生成的是单选框。

```
form_label([$label_text = '[', $id = '[', $attributes = array()])
```

参数

- **\$label_text** (*string*) – Text to put in the <label> tag
- **\$id** (*string*) – ID of the form element that we're making a label for
- **\$attributes** (*string*) – HTML attributes

返回 An HTML field label tag

返回类型 string

生成 <label> 标签, 例如:

```
echo form_label('What is your Name', 'username');  
// Would produce: <label for="username">What is your Name</label>
```

和其他的函数一样, 如果你想添加额外的属性的话, 可以传一个关联数组给第三个参数。

例如:

```
$attributes = array(  
    'class' => 'mycustomclass',  
    'style' => 'color: #000;' )  
);
```

```
echo form_label('What is your Name', 'username', $attributes);  
// Would produce: <label for="username" class="mycustomclass" style="color: #000;">Wh
```

```
form_submit([ $data = '[', $value = '[', $extra = ''] ])
```

参数

- **\$data** (*string*) – Button name
- **\$value** (*string*) – Button value
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML input submit tag

返回类型 string

用于生成一个标准的提交按钮。例如:

```
echo form_submit('mysubmit', 'Submit Post!');  
// Would produce: <input type="submit" name="mysubmit" value="Submit Post!" />
```

和其他的函数一样, 如果你想添加额外的属性的话, 可以传一个关联数组给第一个参数, 第三个参数可以向表单添加额外的数据, 例如 JavaScript 。

```
form_reset([ $data = '[', $value = '[', $extra = ''] ])
```

参数

- **\$data** (*string*) – Button name
- **\$value** (*string*) – Button value

- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML input reset button tag

返回类型 string

用于生成一个标准的重置按钮。用法和 `form_submit()` 函数一样。

```
form_button([$data = '', $content = '', $extra = ''])
```

参数

- **\$data** (*string*) – Button name
- **\$content** (*string*) – Button label
- **\$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string

返回 An HTML button tag

返回类型 string

用于生成一个标准的按钮，你可以简单的使用名称和内容来生成按钮：

```
echo form_button('name', 'content');
// Would produce: <button name="name" type="button">Content</button>
```

或者使用一个关联数组，来包含任何你想要的数据：

```
$data = array(
    'name'      => 'button',
    'id'        => 'button',
    'value'     => 'true',
    'type'      => 'reset',
    'content'   => 'Reset'
);

echo form_button($data);
// Would produce: <button name="button" id="button" value="true" type="reset">Reset</button>
```

如果你还希望能包含一些额外的数据，例如 JavaScript，你可以通过第三个参数传一个字符串：

```
$js = 'onClick="some_function()"';
echo form_button('mybutton', 'Click Me', $js);
```

```
form_close([$extra = ''])
```

参数

- **\$extra** (*string*) – Anything to append after the closing tag, as is

返回 An HTML form closing tag

返回类型 string

用于生成结束标签 `</form>`，使用这个函数唯一的一个好处是，它可以在结束标签的后面加上一些其他的数据。例如：

```
$string = '</div></div>'; echo form_close($string); // Would produce:  
</form> </div></div>
```

```
set_value($field[, $default = '', $html_escape = TRUE])
```

参数

- **\$field** (*string*) – Field name
- **\$default** (*string*) – Default value
- **\$html_escape** (*bool*) – Whether to turn off HTML escaping of the value

返回 Field value

返回类型 string

用于你显示 `input` 或者 `textarea` 类型的输入框的值。你必须在第一个参数中指定名称，第二个参数是可选的，允许你设置一个默认值，第三个参数也是可选，可以禁用对值的转义，当你在和 `form_input()` 函数一起使用时，可以避免重复转义。

例如：

```
<input type="text" name="quantity" value="<?php echo set_value('quantity', '0'); ?>" s
```

当上面的表单元素第一次加载时将会显示 “0”。

注解： If you’ve loaded the [表单验证类](#) and have set a validation rule for the field name in use with this helper, then it will forward the call to the [表单验证类](#)’s own `set_value()` method. Otherwise, this function looks in `$_POST` for the field value.

```
set_select($field[, $value = '', $default = FALSE])
```

参数

- **\$field** (*string*) – Field name
- **\$value** (*string*) – Value to check for
- **\$default** (*string*) – Whether the value is also a default one

返回 ‘selected’ attribute or an empty string

返回类型 string

如果你使用 `<select>` 下拉菜单，此函数允许你显示选中的菜单项。

第一个参数为下拉菜单的名称，第二个参数必须包含每个菜单项的值。第三个参数是可选的，用于设置菜单项是否为默认选中状态 (`TRUE / FALSE`)。

例如：

```
<select name="myselect">
  <option value="one" <?php echo set_select('myselect', 'one', TRUE); ?> >One</option>
  <option value="two" <?php echo set_select('myselect', 'two'); ?> >Two</option>
  <option value="three" <?php echo set_select('myselect', 'three'); ?> >Three</option>
</select>
```

```
set_checkbox($field[, $value = '', $default = FALSE])
```

参数

- **\$field** (*string*) – Field name
- **\$value** (*string*) – Value to check for
- **\$default** (*string*) – Whether the value is also a default one

返回 'checked' attribute or an empty string

返回类型 string

允许你显示一个处于提交状态的复选框。

第一个参数必须包含此复选框的名称，第二个参数必须包含它的值，第三个参数是可选的，用于设置复选框是否为默认选中状态 (TRUE / FALSE)。

例如:

```
<input type="checkbox" name="mycheck" value="1" <?php echo set_checkbox('mycheck', '1', TRUE); ?>
<input type="checkbox" name="mycheck" value="2" <?php echo set_checkbox('mycheck', '2', FALSE); ?>
```

```
set_radio($field[, $value = '', $default = FALSE])
```

参数

- **\$field** (*string*) – Field name
- **\$value** (*string*) – Value to check for
- **\$default** (*string*) – Whether the value is also a default one

返回 'checked' attribute or an empty string

返回类型 string

允许你显示那些处于提交状态的单选框。该函数和上面的 `set_checkbox()` 函数一样。

例如:

```
<input type="radio" name="myradio" value="1" <?php echo set_radio('myradio', '1', TRUE); ?>
<input type="radio" name="myradio" value="2" <?php echo set_radio('myradio', '2', FALSE); ?>
```

注解: 如果你正在使用表单验证类，你必须为你的每一个表单域指定一个规则，即使是空的，这样可以确保 `set_*`() 函数能正常运行。这是因为如果定义了一个表单验证对象，`set_*`() 函数的控制权将移交到表单验证类，而不是辅助函数函数。

```
form_error([$field = '[', $prefix = '[', $suffix = ''])
```

参数

- **\$field** (*string*) – Field name
- **\$prefix** (*string*) – Error opening tag
- **\$suffix** (*string*) – Error closing tag

返回 HTML-formatted form validation error message(s)

返回类型 string

从[表单验证类](#) 返回验证错误消息，并附上验证出错的域的名称，你可以设置错误消息的起始和结束标签。

例如:

```
// Assuming that the 'username' field value was incorrect:
echo form_error('myfield', '<div class="error">', '</div>');
```

```
// Would produce: <div class="error">Error message associated with the "username" field
```

```
validation_errors([$prefix = '[', $suffix = ''])
```

参数

- **\$prefix** (*string*) – Error opening tag
- **\$suffix** (*string*) – Error closing tag

返回 HTML-formatted form validation error message(s)

返回类型 string

和[form_error\(\)](#) 函数类似，返回所有[表单验证类](#) 生成的错误信息，你可以为每个错误消息设置起始和结束标签。

例如:

```
echo validation_errors('<span class="error">', '</span>');
```

```
/*
```

```
Would produce, e.g.:
```

```
<span class="error">The "email" field doesn't contain a valid e-mail address!</span>
<span class="error">The "password" field doesn't match the "repeat_password" field
```

```
*/
```

```
form_prep($str)
```

参数

- **\$str** (*string*) – Value to escape

返回 Escaped value

返回类型 string

允许你在表元素中安全的使用 HTML 和例如引号这样的字符，而不用担心对表单造成破坏。

注解： 如果你使用了这个页面上介绍的任何一个函数，表单的域值会被自动转义，所以你无需再调用这个函数。只有在你创建自己的表元素时需要使用它。

注解： 该函数已经废弃，现在只是通用函数 `html_escape()` 的一个别名，请使用 `html_escape()` 代替它。

10.1.10 HTML 辅助函数

HTML 辅助函数文件包含了用于处理 HTML 的一些函数。

- 加载辅助函数
- 可用函数

加载辅助函数

该辅助函数通过下面的代码加载：

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

可用函数

该辅助函数有下列可用函数：

```
heading([ $data = '', $h = '1', $attributes = '' ])
```

参数

- **\$data** (*string*) – Content
- **\$h** (*string*) – Heading level
- **\$attributes** (*mixed*) – HTML attributes

返回 HTML heading tag

返回类型 string

用于创建 HTML 标题标签，第一个参数为标题内容，第二个参数为标题大小。例如：

```
echo heading('Welcome!', 3);
```

上面代码将生成: `<h3>Welcome!</h3>`

另外, 为了向标题标签添加属性, 例如 HTML class、id 或内联样式, 可以通过第三个参数传一个字符串或者一个数组:

```
echo heading('Welcome!', 3, 'class="pink"');
echo heading('How are you?', 4, array('id' => 'question', 'class' => 'green'));
```

上面代码将生成:

```
<h3 class="pink">Welcome!</h3>
<h4 id="question" class="green">How are you?</h4>
```

```
img($src = '', $index_page = FALSE, $attributes = '')
```

参数

- `$src` (*string*) – Image source data
- `$index_page` (*bool*) – Whether to treat `$src` as a routed URI string
- `$attributes` (*array*) – HTML attributes

返回 HTML image tag

返回类型 string

用于生成 HTML `` 标签, 第一个参数为图片地址, 例如:

```
echo img('images/picture.jpg'); // gives  'images/picture.jpg',
    'alt'    => 'Me, demonstrating how to eat 4 slices of pizza at one time',
    'class'  => 'post_images',
    'width'  => '200',
    'height'=> '200',
    'title'  => 'That was quite a night',
    'rel'    => 'lightbox'
);

img($image_properties);
//  标签，这在生成样式的 link 标签时很有用，当然也可以生成其他的 link 标签。参数为 *href*，后面的是可选的：*rel*、*type*、*title*、*media* 和 *index\_page*。

*index\_page* 参数是个布尔值，用于指定是否在 *href* 链接中添加由 `$config['index_page']` 所设置的起始页面。

例如:

```
echo link_tag('css/mystyles.css');
// gives <link href="http://site.com/css/mystyles.css" rel="stylesheet" type="text/css"
```

另一个例子:

```
echo link_tag('favicon.ico', 'shortcut icon', 'image/ico');
// <link href="http://site.com/favicon.ico" rel="shortcut icon" type="image/ico" />

echo link_tag('feed', 'alternate', 'application/rss+xml', 'My RSS Feed');
// <link href="http://site.com/feed" rel="alternate" type="application/rss+xml" title="My RSS Feed" />
```

另外，你也可以通过向 `link()` 函数传递一个关联数组来完全控制所有的属性和值:

```
$link = array(
 'href' => 'css/primer.css',
 'rel' => 'stylesheet',
 'type' => 'text/css',
 'media' => 'print'
);

echo link_tag($link);
// <link href="http://site.com/css/primer.css" rel="stylesheet" type="text/css" media="print" />
```

```
ul($list, $attributes = '')
```



### 参数

- `$list (array)` – List entries
- `$attributes (array)` – HTML attributes

返回 HTML-formatted unordered list

返回类型 string

用于生成 HTML 无序列表 (<ul>), 参数为简单的数组或者多维数组。例如:

```
$list = array(
 'red',
 'blue',
 'green',
 'yellow'
);

$attributes = array(
 'class' => 'boldlist',
 'id' => 'mylist'
);

echo ul($list, $attributes);
```

上面的代码将生成:

```
<ul class="boldlist" id="mylist">
 red
 blue
 green
 yellow

```

下面是个更复杂的例子, 使用了多维数组:

```
$attributes = array(
 'class' => 'boldlist',
 'id' => 'mylist'
);

$list = array(
 'colors' => array(
 'red',
 'blue',
 'green'
),
 'shapes' => array(
 'round',
 'square',
 'circles' => array(
 'ellipse',
 'oval',
```

```

 'sphere'
)
),
'moods' => array(
 'happy',
 'upset' => array(
 'defeated' => array(
 'dejected',
 'disheartened',
 'depressed'
),
 'annoyed',
 'cross',
 'angry'
)
)
);

```

```
echo ul($list, $attributes);
```

上面的代码将生成:

```

<ul class="boldlist" id="mylist">
 colors

 red
 blue
 green

 shapes

 round
 square
 circles

 ellipse
 oval
 sphere

 moods

 happy
 upset

 defeated

 dejected

```

```

 disheartened
 depressed

 annoyed
 cross
 angry


```

```
ol($list, $attributes = '')
```

#### 参数

- **\$list** (*array*) – List entries
- **\$attributes** (*array*) – HTML attributes

返回 HTML-formatted ordered list

返回类型 string

和 `ul()` 函数一样, 只是它生成的是有序列表 (`<ol>` )。

```
meta([$name = '', $content = '', $type = 'name', $newline = "n"]))
```

#### 参数

- **\$name** (*string*) – Meta name
- **\$content** (*string*) – Meta content
- **\$type** (*string*) – Meta type
- **\$newline** (*string*) – Newline character

返回 HTML meta tag

返回类型 string

用于生成 meta 标签, 你可以传递一个字符串参数, 或者一个数组, 或者一个多维数组。

例如:

```

echo meta('description', 'My Great site');
// Generates: <meta name="description" content="My Great Site" />

echo meta('Content-type', 'text/html; charset=utf-8', 'equiv');
// Note the third parameter. Can be "equiv" or "name"
// Generates: <meta http-equiv="Content-type" content="text/html; charset=utf-8" />

echo meta(array('name' => 'robots', 'content' => 'no-cache'));
// Generates: <meta name="robots" content="no-cache" />

```

```

$meta = array(
 array(
 'name' => 'robots',
 'content' => 'no-cache'
),
 array(
 'name' => 'description',
 'content' => 'My Great Site'
),
 array(
 'name' => 'keywords',
 'content' => 'love, passion, intrigue, deception'
),
 array(
 'name' => 'robots',
 'content' => 'no-cache'
),
 array(
 'name' => 'Content-type',
 'content' => 'text/html; charset=utf-8', 'type' => 'equiv'
)
);

echo meta($meta);
// Generates:
// <meta name="robots" content="no-cache" />
// <meta name="description" content="My Great Site" />
// <meta name="keywords" content="love, passion, intrigue, deception" />
// <meta name="robots" content="no-cache" />
// <meta http-equiv="Content-type" content="text/html; charset=utf-8" />

```

**doctype**(*\$type = 'xhtml1-strict'*)

#### 参数

- **\$type** (*string*) – Doctype name

**返回** HTML DocType tag

**返回类型** string

用于生成 DTD（文档类型声明, document type declaration），默认使用的是 XHTML 1.0 Strict，但是你也可以选择其他的。

例如:

```

echo doctype(); // <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

echo doctype('html4-trans'); // <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">

```

下表是可选的文档类型，它是可配置的，你可以在 `application/config/doc-types.php` 文件中找到它。

文档类型	选项	结果
XHTML 1.1	xhtml11	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
XHTML 1.0 Strict	xhtml1-strict	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
XHTML 1.0 Transitional	xhtml1-trans	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
XHTML 1.0 Frameset	xhtml1-frame	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
XHTML Basic 1.1	xhtml-basic11	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN" "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
HTML 5	html5	<!DOCTYPE html>
HTML 4 Strict	html4-strict	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
HTML 4 Transitional	html4-trans	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
HTML 4 Frameset	html4-frame	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
MathML 1.01	mathml1	<!DOCTYPE math SYSTEM "http://www.w3.org/Math/DTD/mathml1/mathml.dtd">
MathML 2.0	mathml2	<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN" "http://www.w3.org/Math/DTD/mathml2/mathml2.dtd">
SVG 1.0	svg10	<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
SVG 1.1 Full	svg11	<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
SVG 1.1 Basic	svg11-basic	<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1 Basic//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-basic.dtd">
SVG 1.1 Tiny	svg11-tiny	<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1 Tiny//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-tiny.dtd">
XHTML +MathML +SVG (XHTML host)	xhtml-math-svg-xh	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN" "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">
488 XHTML +MathML +SVG (SVG host)	xhtml-math-svg-sh	<!DOCTYPE svg:svg PUBLIC "-//W3C//DTD XHTML 1.1 plus MathML 2.0 plus SVG 1.1//EN" "http://www.w3.org/2002/04/xhtml-math-svg/xhtml-math-svg.dtd">

```
br($count = 1)
```

#### 参数

- **\$count** (*int*) – Number of times to repeat the tag

返回 HTML line break tag

返回类型 string

根据指定的个数生成多个换行标签 (<br />)。例如:

```
echo br(3);
```

上面的代码将生成:

```



```

---

**注解:** 该函数已经废弃, 请使用原生的 `str_repeat()` 函数代替。

---

```
nbs($num = 1)
```

#### 参数

- **\$num** (*int*) – Number of space entities to produce

返回 A sequence of non-breaking space HTML entities

返回类型 string

根据指定的个数生成多个不换行空格 (&nbsp;)。例如:

```
echo nbs(3);
```

上面的代码将生成:

```

```

---

**注解:** 该函数已经废弃, 请使用原生的 `str_repeat()` 函数代替。

---

### 10.1.11 Inflector 辅助函数

Inflector 辅助函数文件包含了一些帮助你单词转换为单复数或驼峰格式等等的函数。

- 加载辅助函数
- 可用函数

#### 加载辅助函数

该辅助函数通过下面的代码加载:

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

## 可用函数

该辅助函数有下列可用函数:

**singular**(\$str)

### 参数

- **\$str** (*string*) – Input string

**返回** A singular word

**返回类型** string

将一个单词的复数形式变为单数形式。例如:

```
echo singular('dogs'); // Prints 'dog'
```

**plural**(\$str)

### 参数

- **\$str** (*string*) – Input string

**返回** A plural word

**返回类型** string

将一个单词的单数形式变为复数形式。例如:

```
echo plural('dog'); // Prints 'dogs'
```

**camelize**(\$str)

### 参数

- **\$str** (*string*) – Input string

**返回** Camelized string

**返回类型** string

将一个以空格或下划线分隔的单词转换为驼峰格式。例如:

```
echo camelize('my_dog_spot'); // Prints 'myDogSpot'
```

**underscore**(\$str)

### 参数

- **\$str** (*string*) – Input string

**返回** String containing underscores instead of spaces

**返回类型** string

将以空格分隔的多个单词转换为下划线分隔格式。例如:

```
echo underscore('my dog spot'); // Prints 'my_dog_spot'
```

```
humanize($str[, $separator = '_'])
```

参数

- **\$str** (*string*) – Input string
- **\$separator** (*string*) – Input separator

返回 Humanized string

返回类型 string

将以下划线分隔的多个单词转换为以空格分隔, 并且每个单词以大写开头。例如:

```
echo humanize('my_dog_spot'); // Prints 'My Dog Spot'
```

如果单词是以连接符分割的, 第二个参数传入连接符:

```
echo humanize('my-dog-spot', '-'); // Prints 'My Dog Spot'
```

```
is_countable($word)
```

参数

- **\$word** (*string*) – Input string

返回 TRUE if the word is countable or FALSE if not

返回类型 bool

判断某个单词是否有复数形式。例如:

```
is_countable('equipment'); // Returns FALSE
```

### 10.1.12 语言辅助函数

语言辅助函数文件包含了用于处理语言文件的一些函数。

- 加载辅助函数
- 可用函数

加载辅助函数

该辅助函数通过下面的代码加载:

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



## 可用函数

该辅助函数有下列可用函数:

```
lang($line[, $for = '[', $attributes = array()])
```

### 参数

- **\$line** (*string*) – Language line key
- **\$for** (*string*) – HTML “for” attribute (ID of the element we’re creating a label for)
- **\$attributes** (*array*) – Any additional HTML attributes

**返回** HTML-formatted language line label

**返回类型** string

此函数使用简单的语法从已加载的语言文件中返回一行文本。这种简单的写法在视图文件中可能比调用 `CI_Lang::line()` 更顺手。

Example:

```
echo lang('language_key', 'form_item_id', array('class' => 'myClass'));
// Outputs: <label for="form_item_id" class="myClass">Language line</label>
```

## 10.1.13 数字辅助函数

数字辅助函数文件包含了用于处理数字的一些函数。

- 加载辅助函数
  - 可用函数

## 加载辅助函数

该辅助函数通过下面的代码加载:

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

## 可用函数

该辅助函数有下列可用函数:

```
byte_format($num[, $precision = 1])
```

### 参数

- **\$num** (*mixed*) – Number of bytes
- **\$precision** (*int*) – Floating point precision

返回 Formatted data size string

返回类型 string

根据数值大小以字节的形式格式化，并添加适合的缩写单位。例如：

```
echo byte_format(456); // Returns 456 Bytes
echo byte_format(4567); // Returns 4.5 KB
echo byte_format(45678); // Returns 44.6 KB
echo byte_format(456789); // Returns 447.8 KB
echo byte_format(3456789); // Returns 3.3 MB
echo byte_format(12345678912345); // Returns 1.8 GB
echo byte_format(123456789123456789); // Returns 11,228.3 TB
```

可选的第二个参数允许你设置结果的精度：

```
echo byte_format(45678, 2); // Returns 44.61 KB
```

---

**注解：** 这个函数生成的缩写单位可以在 `language/<your_lang>/number_lang.php` 语言文件中找到。

---

### 10.1.14 路径辅助函数

路径辅助函数文件包含了用于处理服务端文件路径的一些函数。

- 加载辅助函数
- 可用函数

#### 加载辅助函数

该辅助函数通过下面的代码加载：

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

#### 可用函数

该辅助函数有下列可用函数：

```
set_realpath($path[, $check_existance = FALSE])
```

参数

- **\$path** (*string*) – Path
- **\$check\_existance** (*bool*) – Whether to check if the path actually exists

返回 An absolute path

返回类型 string

该函数返回指定路径在服务端的绝对路径（不是符号路径或相对路径），可选的第二个参数用于指定当文件路径不存在时是否报错。

Examples:

```
$file = '/etc/php5/apache2/php.ini';
echo set_realpath($file); // Prints '/etc/php5/apache2/php.ini'

$non_existent_file = '/path/to/non-exist-file.txt';
echo set_realpath($non_existent_file, TRUE); // Shows an error, as the path cannot
echo set_realpath($non_existent_file, FALSE); // Prints '/path/to/non-exist-file.txt'

$directory = '/etc/php5';
echo set_realpath($directory); // Prints '/etc/php5/'

$non_existent_directory = '/path/to/nowhere';
echo set_realpath($non_existent_directory, TRUE); // Shows an error, as the path cannot
echo set_realpath($non_existent_directory, FALSE); // Prints '/path/to/nowhere'
```

### 10.1.15 安全辅助函数

安全辅助函数文件包含了一些和安全相关的函数。

- 加载辅助函数
- 可用函数

#### 加载辅助函数

该辅助函数通过下面的代码加载:

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

#### 可用函数

该辅助函数有下列可用函数:

```
xss_clean($str[, $is_image = FALSE])
```

参数

- **\$str** (*string*) – Input data
- **\$is\_image** (*bool*) – Whether we're dealing with an image

返回 XSS-clean string

返回类型 string

该函数提供了 XSS 攻击的过滤。

它是 `CI_Input::xss_clean()` 函数的别名, 更多信息, 请查阅[输入类](#) 文档。

**sanitize\_filename**(*\$filename*)

#### 参数

- **\$filename** (*string*) – Filename

**返回** Sanitized file name

**返回类型** string

该函数提供了目录遍历攻击的防护。

它是 `CI_Security::sanitize_filename()` 函数的别名, 更多信息, 请查阅[安全类](#) 文档。

**do\_hash**(*\$str*[, *\$type* = 'sha1'])

#### 参数

- **\$str** (*string*) – Input
- **\$type** (*string*) – Algorithm

**返回** Hex-formatted hash

**返回类型** string

该函数可计算单向散列, 一般用于对密码进行加密, 默认使用 SHA1 。

你可以前往 [hash\\_algos\(\)](#) 查看所有支持的算法清单。

举例:

```
$str = do_hash($str); // SHA1
$str = do_hash($str, 'md5'); // MD5
```

---

**注解:** 这个函数前身为 `dohash()`, 已废弃。

---



---

**注解:** 这个函数也不建议使用, 使用原生的 `hash()` 函数替代。

---

**strip\_image\_tags**(*\$str*)

#### 参数

- **\$str** (*string*) – Input string

**返回** The input string with no image tags

**返回类型** string

该安全函数从一个字符串中剥除 image 标签, 它将 image 标签转为纯图片的 URL 文本。

举例:

```
$string = strip_image_tags($string);
```

它是 `CI_Security::strip_image_tags()` 函数的别名, 更多信息, 请查阅[安全类文档](#)。

`encode_php_tags($str)`

#### 参数

- **\$str** (*string*) – Input string

**返回** Safely formatted string

**返回类型** string

该安全函数将 PHP 标签转换为实体对象。

---

**注解:** 如果你使用函数 `xss_clean()`, 会自动转换。

---

举例:

```
$string = encode_php_tags($string);
```

### 10.1.16 表情辅助函数

表情辅助函数文件包含了一些让你管理表情的函数。

---

**重要:** 表情辅助函数已经废弃, 不建议使用。现在只是为了向前兼容而保留。

---


- 加载辅助函数
- 概述
- 可点击的表情包教程
  - 控制器
  - 字段别名
- 可用函数

#### 加载辅助函数

该辅助函数通过下面的代码加载:

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

#### 概述

表情辅助函数用于将纯文本的表情转换为图片, 例如: `:-)` 转换为 

另外它还可以显示一组表情图片，当你点击其中的某个表情时将会被插入到一个表单域中。例如，如果你有一个博客并允许用户提交评论，你可以将这组表情图片显示在评论的旁边，这样用户就可以点击想要的表情，然后通过一点点的 Javascript 代码，将该表情插入到用户的评论中去。

### 可点击的表情包教程

这里是一个如何在表单中使用可点击的表情包的示例，这个示例需要你首先下载并安装表情图片，然后按下面的步骤创建一个控制器和视图。

---

**重要：** 开始之前，请先 [下载表情图片](#) 然后将其放置到服务器的一个公共目录，并打开 `application/config/smileys.php` 文件设置表情替换的规则。

---

### 控制器

在 `application/controllers/` 目录下，创建一个文件 `Smileys.php` 然后输入下面的代码。

---

**重要：** 修改下面的 `get_clickable_smileys()` 函数的 URL 参数，让其指向你的表情目录。

---

你会发现我们除了使用到了表情库，还使用到了 [表格类](#)：

```
<?php
```

```
class Smileys extends CI_Controller {

 public function index()
 {
 $this->load->helper('smiley');
 $this->load->library('table');

 $image_array = get_clickable_smileys('http://example.com/images/smileys/', 'comment');
 $col_array = $this->table->make_columns($image_array, 8);

 $data['smiley_table'] = $this->table->generate($col_array);
 $this->load->view('smiley_view', $data);
 }
}
```

然后，在 `application/views/` 目录下新建一个文件 `smiley_view.php` 并输入以下代码：

```
<html>
 <head>
 <title>Smileys</title>
 <?php echo smiley_js(); ?>
```

```
</head>
<body>
 <form name="blog">
 <textarea name="comments" id="comments" cols="40" rows="4"></textarea>
 </form>
 <p>Click to insert a smiley!</p>
 <?php echo $smiley_table; ?> </body> </html>
 When you have created the above controller and view, load it by visiting http://www
</body>
</html>
```

## 字段别名

当修改视图的时候, 会牵扯到控制器中的 id 字段, 带来不便。为了解决这一问题, 你可以在视图中给表情一个别名, 并将其映射到 id 字段。

```
$image_array = get_smiley_links("http://example.com/images/smileys/", "comment_textarea_alias");
```

将别名映射到 id 字段, 可以使用 smiley\_js 函数并传入这两个参数:

```
$image_array = smiley_js("comment_textarea_alias", "comments");
```

## 可用函数

```
get_clickable_smileys($image_url[, $alias = '[', $smileys = NULL]])
```

### 参数

- **\$image\_url** (*string*) – URL path to the smileys directory
- **\$alias** (*string*) – Field alias

**返回** An array of ready to use smileys

**返回类型** array

返回一个已经绑定了可点击表情的数组。你必须提供表情文件夹的 URL , 还有表单域的 ID 或者表单域的别名。

举例:

```
$image_array = get_clickable_smileys('http://example.com/images/smileys/', 'comment');

smiley_js([$alias = '[', $field_id = '[', $inline = TRUE]])
```

### 参数

- **\$alias** (*string*) – Field alias
- **\$field\_id** (*string*) – Field ID
- **\$inline** (*bool*) – Whether we're inserting an inline smiley

返回 Smiley-enabling JavaScript code

返回类型 string

生成可以让图片点击后插入到表单域中的 JavaScript 代码。如果你在生成表情链接的时候提供了一个别名来代替 id，你需要在函数中传入别名和相应的 id，此函数被设计为应放在你 Web 页面的 <head> 部分。

举例:

```
<?php echo smiley_js(); ?>
```

```
parse_smileys([$str = '[', $image_url = '[', $smileys = NULL]])
```

参数

- **\$str** (*string*) – Text containing smiley codes
- **\$image\_url** (*string*) – URL path to the smileys directory
- **\$smileys** (*array*) – An array of smileys

返回 Parsed smileys

返回类型 string

输入一个文本字符串，并将其中的纯文本表情替换为等效的表情图片，第一个参数为你的字符串，第二个参数是你的表情目录对应的 URL。

举例:

```
$str = 'Here are some smileys: :-) ;-)';
$str = parse_smileys($str, 'http://example.com/images/smileys/');
echo $str;
```

### 10.1.17 字符串辅助函数

字符串辅助函数文件包含了一些帮助你处理字符串的函数。

---

**重要:** Please note that these functions are NOT intended, nor suitable to be used for any kind of security-related logic.

---

- 加载辅助函数
- 可用函数

#### 加载辅助函数

该辅助函数通过下面的代码加载:

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



## 可用函数

该辅助函数有下列可用函数:

```
random_string([$type = 'alnum', $len = 8])
```

### 参数

- **\$type** (*string*) – Randomization type
- **\$len** (*int*) – Output string length

**返回** A random string

**返回类型** string

根据你所指定的类型和长度产生一个随机字符串。可用于生成密码或随机字符串。

第一个参数指定字符串类型，第二个参数指定其长度。有下列几种字符串类型可供选择:

- **alpha**: 只含有大小写字母的字符串
- **alnum**: 含有大小写字母以及数字的字符串
- **basic**: 根据 `mt_rand()` 函数生成的一个随机数字
- **numeric**: 数字字符串
- **nozero**: 数字字符串 (不含零)
- **md5**: 根据 `md5()` 生成的一个加密的随机数字 (长度固定为 32)
- **sha1**: 根据 `sha1()` 生成的一个加密的随机数字 (长度固定为 40)

使用示例:

```
echo random_string('alnum', 16);
```

---

**注解:** *unique* 和 *encrypt* 类型已经废弃，它们只是 *md5* 和 *sha1* 的别名。

---

```
increment_string($str[, $separator = '_'[, $first = 1]])
```

### 参数

- **\$str** (*string*) – Input string
- **\$separator** (*string*) – Separator to append a duplicate number with
- **\$first** (*int*) – Starting number

**返回** An incremented string

**返回类型** string

自增字符串是指向字符串尾部添加一个数字，或者对这个数字进行自增。这在生成文件的拷贝时非常有用，或者向数据库中某列（例如 title 或 slug）添加重复的内容，但是这一列设置了唯一索引时。

使用示例:

```
echo increment_string('file', '_'); // "file_1"
echo increment_string('file', '-', 2); // "file-2"
echo increment_string('file_4'); // "file_5"
```

**alternator**(\$args)

**参数**

- **\$args** (*mixed*) – A variable number of arguments

**返回** Alternated string(s)

**返回类型** mixed

当执行一个循环时，让两个或两个以上的条目轮流使用。示例:

```
for ($i = 0; $i < 10; $i++)
{
 echo alternator('string one', 'string two');
}
```

你可以添加任意多个参数，每一次循环后下一个条目将成为返回值。

```
for ($i = 0; $i < 10; $i++)
{
 echo alternator('one', 'two', 'three', 'four', 'five');
}
```

---

**注解:** 如果要多次调用该函数，可以简单的通过不带参数重新初始化下。

---

**repeater**(\$data[, \$num = 1])

**参数**

- **\$data** (*string*) – Input
- **\$num** (*int*) – Number of times to repeat

**返回** Repeated string

**返回类型** string

重复生成你的数据。例如:

```
$string = "\n";
echo repeater($string, 30);
```

上面的代码会生成 30 个空行。

---

**注解:** 该函数已经废弃，使用原生的 `str_repeat()` 函数替代。

---

`reduce_double_slashes($str)`

参数

- **\$str** (*string*) – Input string

返回 A string with normalized slashes

返回类型 string

将字符串中的双斜线（'//'）转换为单斜线（'/'), 但不转换 URL 协议中的双斜线（例如: <http://>）

示例:

```
$string = "http://example.com//index.php";
echo reduce_double_slashes($string); // results in "http://example.com/index.php"
```

`strip_slashes($data)`

参数

- **\$data** (*mixed*) – Input string or an array of strings

返回 String(s) with stripped slashes

返回类型 mixed

移除一个字符串数组中的所有斜线。

示例:

```
$str = array(
 'question' => 'Is your name O'reilly?',
 'answer' => 'No, my name is O'connor.'
);

$str = strip_slashes($str);
```

上面的代码将返回下面的数组:

```
array(
 'question' => "Is your name O'reilly?",
 'answer' => "No, my name is O'connor."
);
```

---

**注解:** 由于历史原因, 该函数也接受一个字符串参数, 这时该函数就相当于 `stripslashes()` 的别名。

---

`trim_slashes($str)`

参数

- **\$str** (*string*) – Input string

返回 Slash-trimmed string

返回类型 string

移除字符串开头和结尾的所有斜线。例如:

```
$string = "/this/that/theother/";
echo trim_slashes($string); // results in this/that/theother
```

---

**注解:** 该函数已废弃, 使用原生的 `trim()` 函数代替: `|| trim($str, '/')`;

---

`reduce_multiples($str[, $character = ',', $trim = FALSE])`

#### 参数

- **\$str** (*string*) – Text to search in
- **\$character** (*string*) – Character to reduce
- **\$trim** (*bool*) – Whether to also trim the specified character

**返回** Reduced string

**返回类型** string

移除字符串中重复出现的某个指定字符。例如:

```
$string = "Fred, Bill,, Joe, Jimmy";
$string = reduce_multiples($string, ","); //results in "Fred, Bill, Joe, Jimmy"
```

如果设置第三个参数为 `TRUE`, 该函数将移除出现在字符串首尾的指定字符。例如:

```
$string = ",Fred, Bill,, Joe, Jimmy,";
$string = reduce_multiples($string, ",", TRUE); //results in "Fred, Bill, Joe, Jimmy"
```

`quotes_to_entities($str)`

#### 参数

- **\$str** (*string*) – Input string

**返回** String with quotes converted to HTML entities

**返回类型** string

将字符串中的单引号和双引号转换为相应的 HTML 实体。例如:

```
$string = "Joe's \"dinner\"";
$string = quotes_to_entities($string); //results in "Joe's "dinner""
```

`strip_quotes($str)`

#### 参数

- **\$str** (*string*) – Input string

**返回** String with quotes stripped

**返回类型** string

移除字符串中出现的单引号和双引号。例如:

```
$string = "Joe's \"dinner\"";
$string = strip_quotes($string); //results in "Joes dinner"
```

### 10.1.18 文本辅助函数

文本辅助函数文件包含了一些帮助你处理文本的函数。

- 加载辅助函数
- 可用函数

#### 加载辅助函数

该辅助函数通过下面的代码加载:

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

#### 可用函数

该辅助函数有下列可用函数:

```
word_limiter($str[, $limit = 100[, $end_char = '…#8230;']])
```

##### 参数

- **\$str** (*string*) – Input string
- **\$limit** (*int*) – Limit
- **\$end\_char** (*string*) – End character (usually an ellipsis)

**返回** Word-limited string

**返回类型** string

根据指定的 单词 个数裁剪字符串。例如:

```
$string = "Here is a nice text string consisting of eleven words."
$string = word_limiter($string, 4);
// Returns: Here is a nice
```

第三个参数用于给裁剪的字符串设置一个可选的后缀, 默认使用省略号。

```
character_limiter($str[, $n = 500[, $end_char = '…#8230;']])
```

##### 参数

- **\$str** (*string*) – Input string
- **\$n** (*int*) – Number of characters
- **\$end\_char** (*string*) – End character (usually an ellipsis)

返回 Character-limited string

返回类型 string

根据指定的 字符 个数裁剪字符串。它会保证单词的完整性，所以最终生成的字符串长度和你指定的长度有可能会出入。

例如:

```
$string = "Here is a nice text string consisting of eleven words.";
$string = character_limiter($string, 20);
// Returns: Here is a nice text string
```

第三个参数用于给裁剪的字符串设置一个可选的后缀，如果没该参数，默认使用省略号。

---

**注解:** 如果你需要将字符串精确的裁剪到指定长度，请参见下面的[ellipsis\(\)](#)函数。

---

**ascii\_to\_entities(\$str)**

参数

- **\$str** (*string*) – Input string

返回 A string with ASCII values converted to entities

返回类型 string

将 ASCII 字符转换为字符实体，包括高位 ASCII 和 Microsoft Word 中的特殊字符，在 Web 页面中使用这些字符可能会导致问题。转换为字符实体后，它们就可以不受浏览器设置的影响正确的显示出来，也能可靠的存储到数据库中。本函数依赖于你的服务器支持的字符集，所以它可能并不能保证 100% 可靠，但在大多数情况下，它都能正确的识别这些特殊字符（例如重音字符）。

例如:

```
$string = ascii_to_entities($string);
```

**convert\_accented\_characters(\$str)**

参数

- **\$str** (*string*) – Input string

返回 A string with accented characters converted

返回类型 string

将高位 ASCII 字符转换为与之相等的普通 ASCII 字符，当你的 URL 中需要使用非英语字符，而你的 URL 又设置了只允许出现普通 ASCII 字符时很有用。

例如:

```
$string = convert_accented_characters($string);
```

---

**注解:** 该函数使用了 `application/config/foreign_chars.php` 配置文件来决定将什么字符转换为什么字符。

---

```
word_censor($str, $censored[, $replacement = ''])
```

#### 参数

- **\$str** (*string*) – Input string
- **\$censored** (*array*) – List of bad words to censor
- **\$replacement** (*string*) – What to replace bad words with

**返回** Censored string

**返回类型** string

对字符串中出现的敏感词进行审查。第一个参数为原始字符串，第二个参数为一个数组，包含你要禁用的单词，第三个参数（可选）可以设置将出现的敏感词替换成什么，如果未设置，默认替换为磅字符：####。

例如:

```
$disallowed = array('darn', 'shucks', 'golly', 'phooey');
$string = word_censor($string, $disallowed, 'Beep!');
```

```
highlight_code($str)
```

#### 参数

- **\$str** (*string*) – Input string

**返回** String with code highlighted via HTML

**返回类型** string

对一段代码（PHP、HTML 等）进行着色。例如:

```
$string = highlight_code($string);
```

该函数使用了 PHP 的 `highlight_string()` 函数，所以着色的颜色是在 `php.ini` 文件中设置的。

```
highlight_phrase($str, $phrase[, $tag_open = '<mark>', $tag_close = '</mark>'])
```

#### 参数

- **\$str** (*string*) – Input string
- **\$phrase** (*string*) – Phrase to highlight
- **\$tag\_open** (*string*) – Opening tag used for the highlight
- **\$tag\_close** (*string*) – Closing tag for the highlight

**返回** String with a phrase highlighted via HTML

**返回类型** string

对字符串内的一个短语进行突出显示。第一个参数是原始字符串，第二个参数是你想要突出显示的短语。如果要用 HTML 标签对短语进行标记，那么第三个和第四个参数分别是你想要对短语使用的 HTML 开始和结束标签。

例如:

```
$string = "Here is a nice text string about nothing in particular.";
echo highlight_phrase($string, "nice text", '', '');
```

上面的代码将输出:

```
Here is a nice text string about nothing in partic
```

---

**注解:** 该函数默认是使用 `<strong>` 标签，老版本的浏览器可能不支持 `<mark>` 这个 HTML5 新标签，所以如果你想支持这些老的浏览器，推荐你在你的样式文件中添加如下 CSS 代码:

```
mark {
 background: #ff0;
 color: #000;
};
```

---

**word\_wrap**(*\$str* [, *\$charlim* = 76])

**参数**

- **\$str** (*string*) – Input string
- **\$charlim** (*int*) – Character limit

**返回** Word-wrapped string

**返回类型** string

根据指定的 字符 数目对文本进行换行操作，并且保持单词的完整性。

例如:

```
$string = "Here is a simple string of text that will help us demonstrate this function";
echo word_wrap($string, 25);
```

```
// Would produce:
// Here is a simple string
// of text that will help us
// demonstrate this
// function.
```

**ellipsesize**(*\$str*, *\$max\_length* [, *\$position* = 1 [, *\$ellipsis* = '…']])

**参数**

- **\$str** (*string*) – Input string
- **\$max\_length** (*int*) – String length limit



- **\$position** (*mixed*) – Position to split at (int or float)
- **\$ellipsis** (*string*) – What to use as the ellipsis character

返回 Ellipsized string

返回类型 string

该函数移除字符串中出现的标签，并根据指定的长度裁剪字符串，并插入省略号。

第一个参数是要处理的字符串，第二个参数为最终处理完后的字符串长度，第三个参数为插入省略号的位置，值为 0-1 表示从左到右。例如设置为 1 省略号将插入到字符串的右侧，0.5 将插入到中间，0 将插入到左侧。

第四个参数是可选的，表示省略号的类型，默认是 &hellip; 。

例如:

```
$str = 'this_string_is_entirely_too_long_and_might_break_my_design.jpg';
echo ellipsize($str, 32, .5);
```

输出结果:

```
this_string_is_e…ak_my_design.jpg
```

### 10.1.19 排版辅助函数

排版辅助函数文件包含了文本排版相关的一些函数。

- 加载辅助函数
- 可用函数

#### 加载辅助函数

该辅助函数通过下面的代码加载:

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

#### 可用函数

该辅助函数有下列可用函数:

```
auto_typography($str[, $reduce_linebreaks = FALSE])
```

参数

- **\$str** (*string*) – Input string
- **\$reduce\_linebreaks** (*bool*) – Whether to reduce multiple instances of double newlines to two

返回 HTML-formatted typography-safe string

返回类型 string

格式化文本以便纠正语义和印刷错误的 HTML 代码。

这个函数是 `CI_Typography::auto_typography` 函数的别名。更多信息, 查看[排版类](#)。

Usage example:

```
$string = auto_typography($string);
```

---

**注解:** 格式排版可能会消耗大量处理器资源, 特别是在排版大量内容时。如果你选择使用这个函数的话, 你可以考虑使用 [缓存](#) <../general/caching>。

---

`nl2br_except_pre($str)`

参数

- `$str (string)` – Input string

返回 String with HTML-formatted line breaks

返回类型 string

将换行符转换为 `<br />` 标签, 忽略 `<pre>` 标签中的换行符。除了对 `<pre>` 标签中的换行处理有所不同之外, 这个函数和 PHP 函数 `nl2br()` 是完全一样的。

使用示例:

```
$string = nl2br_except_pre($string);
```

`entity_decode($str, $charset = NULL)`

参数

- `$str (string)` – Input string
- `$charset (string)` – Character set

返回 String with decoded HTML entities

返回类型 string

这个函数是 `CI_Security::entity_decode()` 函数的别名。更多信息, 查看[安全类](#)。

### 10.1.20 URL 辅助函数

URL 辅助函数文件包含了一些帮助你处理 URL 的函数。

- [加载辅助函数](#)
  - [可用函数](#)

## 加载辅助函数

该辅助函数通过下面的代码加载:

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

## 可用函数

该辅助函数有下列可用函数:

```
site_url([$uri = '', $protocol = NULL])
```

### 参数

- **\$uri** (*string*) – URI string
- **\$protocol** (*string*) – Protocol, e.g. 'http' or 'https'

**返回** Site URL

**返回类型** string

根据配置文件返回你的站点 URL 。index.php （获取其他你在配置文件中设置的 **index\_page** 参数）将会包含在你的 URL 中，另外再加上你传给函数的 URI 参数，以及配置文件中设置的 **url\_suffix** 参数。

推荐在任何时候都使用这种方法来生成你的 URL ，这样在你的 URL 变动时你的代码将具有可移植性。

传给函数的 URI 段参数可以是一个字符串，也可以是个数组，下面是字符串的例子:

```
echo site_url('news/local/123');
```

上例将返回类似于: `http://example.com/index.php/news/local/123`

下面是使用数组的例子:

```
$segments = array('news', 'local', '123');
echo site_url($segments);
```

该函数是 `CI_Config::site_url()` 的别名，更多信息请查阅 [配置类](#) 文档。

```
base_url($uri = '', $protocol = NULL)
```

### 参数

- **\$uri** (*string*) – URI string
- **\$protocol** (*string*) – Protocol, e.g. 'http' or 'https'

**返回** Base URL

**返回类型** string

根据配置文件返回你站点的根 URL ，例如:

```
echo base_url();
```

该函数和[site\\_url\(\)](#) 函数相同, 只是不会在 URL 的后面加上 *index\_page* 或 *url\_suffix*。

另外, 和[site\\_url\(\)](#) 一样的是, 你也可以使用字符串或数组格式的 URI 段。下面是字符串的例子:

```
echo base_url("blog/post/123");
```

上例将返回类似于: *http://example.com/blog/post/123*

跟[site\\_url\(\)](#) 函数不一样的是, 你可以指定一个文件路径 (例如图片或样式文件), 这将很有用, 例如:

```
echo base_url("images/icons/edit.png");
```

将返回类似于: *http://example.com/images/icons/edit.png*

该函数是 `CI_Config::base_url()` 的别名, 更多信息请查阅[配置类](#) 文档。

#### `current_url()`

**返回** The current URL

**返回类型** string

返回当前正在浏览的页面的完整 URL (包括分段)。

---

**注解:** 该函数和调用下面的代码效果是一样的: `|| site_url(uri_string());`

---

#### `uri_string()`

**返回** An URI string

**返回类型** string

返回包含该函数的页面的 URI 分段。例如, 如果你的 URL 是:

```
http://some-site.com/blog/comments/123
```

函数将返回:

```
blog/comments/123
```

该函数是 `CI_Config::uri_string()` 的别名, 更多信息请查阅[配置类](#) 文档。

#### `index_page()`

**返回** 'index\_page' value

**返回类型** mixed

返回你在配置文件中配置的 `index_page` 参数, 例如:

```
echo index_page();
```

```
anchor($uri = '', $title = '', $attributes = '')
```

#### 参数

- **\$uri** (*string*) – URI string
- **\$title** (*string*) – Anchor title
- **\$attributes** (*mixed*) – HTML attributes

**返回** HTML hyperlink (anchor tag)

**返回类型** string

根据你提供的 URL 生成一个标准的 HTML 链接。

第一个参数可以包含任何你想添加到 URL 上的段, 和上面的`site_url()` 函数一样, URL 的段可以是字符串或数组。

---

**注解:** 如果你创建的链接是指向你自己的应用程序, 那么不用包含根 URL (`http&#58;//...`)。这个会根据你的配置文件自动添加到 URL 前面。所以你只需指定要添加的 URL 段就可以了。

---

第二个参数是链接的文本, 如果留空, 将使用链接本身作为文本。

第三个参数为你希望添加到链接的属性, 可以是一个字符串, 也可以是个关联数组。

这里是一些例子:

```
echo anchor('news/local/123', 'My News', 'title="News title"');
// Prints: My

echo anchor('news/local/123', 'My News', array('title' => 'The best news!'));
// Prints: My

echo anchor('', 'Click here');
// Prints: Click Here
```

```
anchor_popup($uri = '', $title = '', $attributes = FALSE)
```

#### 参数

- **\$uri** (*string*) – URI string
- **\$title** (*string*) – Anchor title
- **\$attributes** (*mixed*) – HTML attributes

**返回** Pop-up hyperlink

**返回类型** string

和`anchor()` 函数非常类似, 只是它生成的 URL 将会在新窗口被打开。你可以通过第三个参数指定 JavaScript 的窗口属性, 以此来控制窗口将如何被打开。如果没有设置第三个参数, 将会使用你的浏览器设置打开一个新窗口。

这里是属性的例子:

```

$atts = array(
 'width' => 800,
 'height' => 600,
 'scrollbars' => 'yes',
 'status' => 'yes',
 'resizable' => 'yes',
 'screenx' => 0,
 'screeny' => 0,
 'window_name' => '_blank'
);

echo anchor_popup('news/local/123', 'Click Me!', $atts);

```

---

**注解:** 上面的属性是函数的默认值, 所以你只需要设置和你想要的不一样的参数。如果想使用所有默认的参数, 只要简单的传一个空数组即可: `|| echo anchor_popup('news/local/123', 'Click Me!', array());`

---



---

**注解:** `window_name` 其实并不算一个属性, 而是 Javascript 的 `window.open()` [<http://www.w3schools.com/jsref/met\\_win\\_open.asp>](http://www.w3schools.com/jsref/met_win_open.asp) 函数的一个参数而已, 该函数接受一个窗口名称或一个 window 对象。

---



---

**注解:** 任何不同于上面列出来的其他的属性将会作为 HTML 链接的属性。

---

```
mailto($email, $title = '', $attributes = '')
```

#### 参数

- **\$email** (*string*) – E-mail address
- **\$title** (*string*) – Anchor title
- **\$attributes** (*mixed*) – HTML attributes

**返回** A “mail to” hyperlink

**返回类型** string

创建一个标准的 HTML e-mail 链接。例如:

```
echo mailto('me@my-site.com', 'Click Here to Contact Me');
```

和上面的 `anchor()` 函数一样, 你可以通过第三个参数设置属性:

```

$attributes = array('title' => 'Mail me');
echo mailto('me@my-site.com', 'Contact Me', $attributes);

```

```
safe_mailto($email, $title = '', $attributes = '')
```

#### 参数

- **\$email** (*string*) – E-mail address
- **\$title** (*string*) – Anchor title

- **\$attributes** (*mixed*) – HTML attributes

返回 A spam-safe “mail to” hyperlink

返回类型 string

和[mailto\(\)](#) 函数一样, 但是它的 *mailto* 标签使用了一个混淆的写法, 可以防止你的 e-mail 地址被垃圾邮件机器人爬到。

```
auto_link($str, $type = 'both', $popup = FALSE)
```

参数

- **\$str** (*string*) – Input string
- **\$type** (*string*) – Link type ('email', 'url' or 'both')
- **\$popup** (*bool*) – Whether to create popup links

返回 Linkified string

返回类型 string

将一个字符串中的 URL 和 e-mail 地址自动转换为链接, 例如:

```
$string = auto_link($string);
```

第二个参数用于决定是转换 URL 还是 e-mail 地址, 默认情况不指定该参数, 两者都会被转换。E-mail 地址的链接是使用上面介绍的[safe\\_mailto\(\)](#) 函数生成的。

只转换 URL

```
$string = auto_link($string, 'url');
```

只转换 e-mail 地址:

```
$string = auto_link($string, 'email');
```

第三个参数用于指定链接是否要在新窗口打开。可以是布尔值 TRUE 或 FALSE

```
$string = auto_link($string, 'both', TRUE);
```

```
url_title($str, $separator = '-', $lowercase = FALSE)
```

参数

- **\$str** (*string*) – Input string
- **\$separator** (*string*) – Word separator
- **\$lowercase** (*string*) – Whether to transform the output string to lower-case

返回 URL-formatted string

返回类型 string

将字符串转换为对人类友好的 URL 字符串格式。例如, 如果你有一个博客, 你希望使用博客的标题作为 URL , 这时该函数很有用。例如:

```
$title = "What's wrong with CSS?";
$url_title = url_title($title);
// Produces: Whats-wrong-with-CSS
```

第二个参数指定分隔符，默认使用连字符。一般的选择有：-（连字符）或者 \_（下划线）

例如：

```
$title = "What's wrong with CSS?";
$url_title = url_title($title, 'underscore');
// Produces: Whats_wrong_with_CSS
```

---

**注解：** 第二个参数连字符和下划线的老的用法已经废弃。

---

第三个参数指定是否强制转换为小写。默认不会，参数类型为布尔值 TRUE 或 FALSE。

例如：

```
$title = "What's wrong with CSS?";
$url_title = url_title($title, 'underscore', TRUE);
// Produces: whats_wrong_with_css
```

**prep\_url**(\$str = '')

**参数**

- **\$str** (*string*) – URL string

**返回** Protocol-prefixed URL string

**返回类型** string

当 URL 中缺少协议前缀部分时，使用该函数将会向 URL 中添加 http://。

像下面这样使用该函数：

```
$url = prep_url('example.com');
```

**redirect**(\$uri = '', \$method = 'auto', \$code = NULL)

**参数**

- **\$uri** (*string*) – URI string
- **\$method** (*string*) – Redirect method ('auto', 'location' or 'refresh')
- **\$code** (*string*) – HTTP Response code (usually 302 or 303)

**返回类型** void

通过 HTTP 头重定向到指定 URL。你可以指定一个完整的 URL，也可以指定一个 URL 段，该函数会根据配置文件自动生成改 URL。



第二个参数用于指定一种重定向方法。可用的方法有：**auto**、**location** 和 **refresh**。**location** 方法速度快，但是在 IIS 服务器上不可靠。默认值为 **auto**，它会根据你的服务器环境智能的选择使用哪种方法。

第三个参数可选，允许你发送一个指定的 HTTP 状态码，这个可以用来为搜索引擎创建 301 重定向。默认的状态码为 302，该参数只适用于 **location** 重定向方法，对于 **refresh** 方法无效。例如：

```
if ($logged_in == FALSE)
{
 redirect('/login/form/');
}

// with 301 redirect
redirect('/article/13', 'location', 301);
```

---

**注解：** 为了让该函数有效，它必须在任何内容输出到浏览器之前被调用。因为输出内容会使用服务器 HTTP 头。

---



---

**注解：** 为了更好的控制服务器头，你应该使用 输出类 `</libraries/output>` 的 `set_header()` 方法。

---



---

**注解：** 使用 IIS 的用户要注意，如果你隐藏了 *Server* 这个 HTTP 头，*auto* 方法将无法检测到 IIS。在这种情况下，推荐你使用 **refresh** 方法。

---



---

**注解：** 当使用 HTTP/1.1 的 POST 来访问你的页面时，如果你使用的是 **location** 方法，会自动使用 HTTP 303 状态码。

---



---

**重要：** 该函数会终止脚本的执行。

---

### 10.1.21 XML 辅助函数

XML 辅助函数文件包含了用于处理 XML 数据的一些函数。

- 加载辅助函数
- 可用函数

#### 加载辅助函数

该辅助函数通过下面的代码加载：

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

## 可用函数

该辅助函数有下列可用函数:

**xml\_convert**(\$str[, \$protect\_all = FALSE])

### 参数

- **\$str** (*string*) – the text string to convert
- **\$protect\_all** (*bool*) – Whether to protect all content that looks like a potential entity instead of just numbered entities, e.g. &foo;

**返回** XML-converted string

**返回类型** string

将输入字符串中的下列 XML 保留字符转换为实体 (Entity):

- 和号: &
- 小于号和大于号: < >
- 单引号和双引号: ' "
- 减号: -

如果 & 符号是作为实体编号的一部分, 例如: &#123; , 该函数将不予处理。举例:

```
$string = '<p>Here is a paragraph & an entity ({).</p>';
$string = xml_convert($string);
echo $string;
```

输出结果:

```
<p>Here is a paragraph & an entity ({).</p>
```

## 10.2 Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

1. The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or
2. The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or
3. The contribution was provided directly to me by some other person who certified (1), (2) or (3) and I have not modified it.