# CodeIgniter4 中文手册

## 版本 *4.0.0*

CodeIgniter 基金会
CodeIgniter 中国开发者社区

2020 年 11 月 21 日

# 目录

# 章节 1

# 欢迎使用 CodeIgniter4

CodeIgniter 是一套给 PHP 网站开发者使用的应用程序开发框架和工具包。它的目标是让你能够更快速的开发，它提供了日常任务中所需的大量类库，以及简单的接口和逻辑结构。通过减少代码量，CodeIgniter 让你更加专注于你的创造性工作。

CodeIgniter 将尽可能的保持其灵活性，以允许你以喜欢的方式工作，而不是被迫以其它方式工作。框架可以轻松扩展或替换核心部件，使系统按你期望的方式工作。简而言之，CodeIgniter 是一个可扩展的框架，它试图提供你所需的工具，同时让你避免踩坑。

## 1.1 CodeIgniter 是为谁准备的？

CodeIgniter 就是你所需要的，如果…

- 你想要一个小巧的框架；
- 你需要出色的性能；
- 你想要一个几乎零配置的框架；
- 你想要一个不需使用命令行的框架；
- 你想要一个不想被编码规则的条条框框限制住的框架；
- 你对 PEAR 这种庞然大物不感兴趣；
- 你不想被迫学习一种新的模板语言（当然如果你喜欢，你可以选择一个模板解析器）；
- 你不喜欢复杂，追求简单；
- 你需要清晰、完整的文档。

### 1.1.1 服务器安装要求

需要 7.2 或更新版本的 PHP 同时需要安装 *intl* 扩展 。

在服务器同时需要启用以下 PHP 扩展: `php-json`, `php-mbstring`, `php-mysqlnd`, `php-xml`

为了能够使用*CURLRequest* ，你需要安装 libcurl 。

对于大多数 web 应用程序来说，一个数据库是不可或缺的。以下是我们支持的数据库:

- MySQL (5.1+) 通过 *MySQLi* 驱动使用
- PostgreSQL 通过 *Postgre* 驱动使用
- SQLite3 通过 *SQLite3* 驱动使用

CodeIgniter4 并没有转换或重写所有的驱动，下列显示了几个很棒的数据库与它们对应的驱动

- MySQL (5.1+) 通过 *pdo* 驱动
- Oracle 通过 *oci8* 和 *pdo* 驱动使用
- PostgreSQL 通过 *pdo* driver
- MS SQL 通过 *mssql*, *sqlsrv* (2005 版本及以上使用) 和 *pdo* 驱动使用
- SQLite 通过 *sqlite* (version 2) 和 *pdo* 驱动使用
- CUBRID 通过 *cubrid* 和 *pdo* 驱动使用
- Interbase/Firebird 通过 *ibase* and *pdo* 驱动使用
- ODBC 通过 *odbc* 和 *pdo* 驱动使用（你应该知道 ODBC 实际上只是一个抽象层）

### 1.1.2 发展历程与贡献者

CodeIgniter 原本是由 EllisLab 所开发的。本框架的开发目的是为了提升在现实使用过程中的性能，同时实现了许多原创的类库，辅助函数和子系统（借鉴了 ExpressionEngine 的代码结构）。在过去的几年中，是由 EllisLab，ExpressionEngine 开发团队以及被称为 the Reactor Team 的社区成员所开发并维护的。

而在 2014 年，CodeIgniter 转由 British Columbia Institute of Technology 进行维护，并正式宣布其将作为一个社区维护的项目。

本框架的前沿开发过程中，CodeIgniter 委员会中的优秀贡献者们体现了先锋的精神（译者注：实际大部分的功能与测试都是由几位开源贡献者鼎力实现的）。

### 1.1.3 PSR 规范

PHP-FIG 创建于 2009 年，旨在帮助各个框架之间更自由的协作标准，遵循统一的编码和风格规范。CodeIgniter 虽然并非 FIG 的成员之一，但我们的宗旨是一致的。这份文档主要是用来列出现有我们所遵循已被提案通过和一些草案的情况。

**PSR-1: 基础编码规范**

这份规范覆盖了基本类，方法和文件的命名标准。我们的 开发规范符合 PSR-1，并且在它的基础上添加了自己的标准。

**PSR-2: 编码风格规范**

这份 PSR 的争议性是比较大的，在它第一次出现的时候。CodeIgniter 在其中遇到了许多建议，但不会完全符合这些规范。

**PSR-3: 日志接口规范**

CodeIgniter 的*Logger* 实现了该 PSR 提供的所有接口。

**PSR-4: 自动加载规范**

这份 PSR 提供了组织文件和命名空间以允许自动加载类的标准方法的方法。我们的自动加载类 符合 PSR-4 规范。

**PSR-6: 缓存接口规范**

CodeIgniter 不会尝试符合这份 PSR ，因为我们相信它超越了它的需求。我们会考虑新提出的 SimpleCache 接口 。

**PSR-7: HTTP 消息接口规范**

这份 PSR 标准化了表示 HTTP 交互的方式。虽然许多概念成为我们的 HTTP 层的一部分，但 CodeIgniter 并不力求与此规范兼容。

—

如果你发现任何我们声称实现 PSR 但未能正确执行的地方，请通知我们，我们会将其修正，或提交需要更改的拉动请求。

# 章节 2

# 开始

## 2.1 安装

CodeIgniter4 可以通过多种不同的方式安装：手动，使用 Composer 或使用 Git。你喜欢哪个?

- 如果你希望像 CodeIgniter3 那样简单的“下载并使用”，那么你应该选择手动安装。
- 如果你打算在项目中添加其他软件包，我们建议你使用 Composer 方式安装。
- 如果你想为框架添砖加瓦，那么你应该选择 Git 方式安装。

### 2.1.1 手动安装

CodeIgniter 4 框架 目录保存了所有已发布的框架版本。这是为那些不想用 Composer 的开发者准备的。

在 `app` 文件夹里开发你的项目，而 `public` 文件夹就将成为你的对外根目录。与此同时，请勿更改 `system` 文件夹下面的任何内容!

**Note**: 这是和 CodeIgniter 3 描述的安装方式最相近的。

#### 安装

下载 最新版本, 并将其解压到你的项目根目录。

#### 设置

无

#### 升级

下载一份最新的框架，并根据发布通知或更新日志里的升级教程来将最新版本的内容合并进你的项目。

通常来说，替换掉 system 目录，并且检查 app/Config 文件夹下受影响的变更内容即可。

#### 优点

下载就可以运行

#### 缺点

当更新时，你要自己合并冲突

#### 结构

在你的项目设置完成后，新建以下文件夹: app, public, system, writable

#### 安装翻译

如果你想充分利用系统信息的翻译，可以类似地把这些翻译加入到项目中。

下载 最新版本的翻译 解压下载的 zip 文件并将 Language 文件夹的内容复制到你的 PROJECT_ROOT/app/Languages 文件夹下。

在进行任何翻译的升级时都需要重复以上步骤。

### 2.1.2 通过 Composer 安装

可以通过多种方式在你的系统中来使用 Composer 安装 CodeIgniter。

前两种方法描述了使用 CodeIgniter4 来创建一个项目的骨架结构，从而让你可以在一个新的 webapp 中作为基础来使用。而第三种技术，如下所述，使得你可以将 CodeIgniter4 加入进一个现存的 webapp 中。

**Note**: 如果你正使用一个 Git 仓库来存储代码或与他人写作，那么 `vendor` 目录就需要添加到 gitignore 文件中。在这种情况下，当你克隆仓库到新系统中，就需要执行“composer update“指令

### 启动应用

CodeIgniter 4 应用启动 仓库里通过 composer 依赖最新版本的框架来维护了一个基础骨架的应用。

以下安装教程适用于每一位希望启动一个新的基于 CodeIgniter4 的项目的开发者。

### 安装和设置

在你的项目根目录执行以下命令:

```
composer create-project codeigniter4/appstarter project-root
```

该指令将会创建一个“project-root”目录。

如果你忽略了”project-root”参数，该命令就会创建一个”appstarter”目录，该目录当需要时可以被重命名。

如果你不需要或不想安装 PHPUnit 以及跟它相关的任何 Composer 依赖，请在该命令的尾部增加”–no-dev”选项。这一操作将只会使用 Composer 安装框架本体以及三个我们打包过的可信赖的外部依赖包。

下面是一个这样的安装指令的示例，使用默认的项目根目录”APPstarter”：

```
composer create-project codeigniter4/appstarter --no-dev
```

安装完成后你应该根据“升级”这节里的步骤继续进行。

### 升级

每当有新的发布时，在你项目的根目录运行以下指令:

```
composer update
```

如果在你创建项目时使用了”–no-dev”选项，那么在这里也一样适合这样做。`composer update --no-dev`

阅读升级指南，并检查指定的 `app/Config` 目录是否有内容变更。

### 优点

便于安装，便于升级。

### 缺点

你仍需要在更新后检查 `app/Config` 的变更。

### 结构

设置完成后你的项目中会有以下目录:

- app, public, tests, writable
- vendor/codeigniter4/framework/system
- vendor/codeigniter4/framework/app & public (compare with yours after updating)

### 最新的开发版本

App Start 仓库里有着 `builds` 脚本，在框架当前稳定发布版本和最新的开发版本间进行选择。对于开发者而言，可以选择使用该脚本来获取最新的变更，不过这些变更可能是不稳定的。

开发者用户手册 可以在线访问。请注意与当前发布版本的用户手册有所不同，并独立维护一个开发的分支。

在你的项目根目录执行以下指令:

```
php builds development
```

以上的指令将会更新 **composer.json** 文件并将当前的工作仓库指向 `develop` 分支，并在配置和 XML 文件中更新对应的路径。如果要回退以上变更，请执行:

```
php builds release
```

在使用完 `builds` 命令后，请确保运行 `composer update` 来将你的 vendor 目录与最新版本的同步。

### 将 CodeIgniter4 添加到现存项目中

在”手动安装”这章中描述过的 CodeIgniter 4 framework 仓库同样也可使用 Composer 来被添加到现存的项目中。

在 `app` 目录下开发你的应用，`public` 目录作为文档的根目录。

在你的项目根目录下:

```
composer require codeigniter4/framework
```

与前面两个 composer 安装方式类似，你也可以在” composer require” 命令中使用” –no-dev” 参数来忽略安装 PHPunit。

### 设置

从 vendor/codeigniter4/framework 中复制 app, public, tests 和 writable 目录到你的项目根目录下。

从 vendor/codeigniter4/framework 中复制 env, phpunit.xml.dist and spark 文件到你的项目根目录下。

你需要设置指向 vendor/codeigniter/framework 的目录——通过修改 app/Config/Paths.php 中的 $systemDirectory 变量

### 升级

每当有新的发布时，在你项目的根目录运行以下指令:

```
composer update
```

如果在你创建项目时使用了” –no-dev” 选项，那么在这里也一样适合这样做。composer update --no-dev

阅读升级指南，并检查指定的 app/Config 目录是否有内容变更。

### 专业人士

相当简单的安装方式；便于升级

### 贡献者

你仍需要在更新后检查 app/Config 的变更。

### 结构

设置完成后你的项目结构如下:

- app, public, tests, writable
- vendor/codeigniter4/framework/system

#### 安装翻译

如果你想充分利用系统信息的翻译，可以类似地把这些翻译加入到项目中。

在项目根目录运行以下指令:

```
composer require codeigniter4/translations @rc
```

当你每次运行 `composer update` 时这些翻译文件也同样会被更新。

### 2.1.3 运行你的应用程序



一个 CodeIgniter 4 的程序能够通过以下几种方式来运行：部署在一台 web 服务器上，使用虚拟化，或者使用 CodeIgniter 的命令行工具以便测试。本节阐述了如何使用以上技术来进行部署，以及介绍了一些高级用法与如何做出贡献。

如果 CodeIgniter 对你来说相当陌生，请阅读用户手册中的准备开始 这节来学习如何构建一个动态的 PHP 应用，祝你玩得开心!

#### 初始化配置与设置

1. 用一个文本编辑器打开 **app/Config/App.php** 文件并设置你的 baseURL（网站基础 URL）。如果你希望更灵活点，也可以通过编辑 `.env` 文件，新增或更改其中的 **app.baseURL=" http://example.com"** 来更改 baseURL。
2. 如果你想要使用数据库，用文本编辑器打开 **app/Config/Database.php** 文件并进行数据库设置。同样的，也可以在 `.env` 文件里进行如上设置。

在生产环境里需要做的另一个件事就是关闭 PHP 的错误报告以及其他所有的只在开发环境里的功能。在 CodeIgniter 中，可以通过设置 `ENVIRONMENT` 常量来进行。关于这一特性，在文档:doc: 环境 </general/environments>‘ 中进行了更为详尽的介绍。在默认情况下，应用程序会使用” production”（生产）环境。为了更为充分地使用所提供的 debug（问题定位）工具，你需要将环境设置为 “develop”（开发环境）

---

**注解:** 如果你想要在一台 web 服务器上运行你的网站。你需要修改项目线下的 `writable` 文件夹的权限，从而使得你的 web 服务器的当前用户可以对它进行写入。

---

### 本地开发主机

CodeIgniter4 中内置了一个本地开发用的主机，利用了 PHP 内置的 web 服务器并实现了 CodeIgniter 的路由机制。你可以使用主目录下的如下如下命令行中的 `serve` 脚本来启动:

```
php spark serve
```

这将会启动服务器，与此同时，你可以在浏览器中输入以下 http://localhost:8080 地址来浏览你的应用。

---

**注解:** 内置的开发服务器只应该在本地开发机器上使用。绝对不要将其用于生产服务器中

---

如果你想在服务器上运行一个不仅仅是 localhost，而是其他站点名的网站，你需要首先将该站点加入到你的 `hosts` 文件中。该文件实际所处的位置根据不同的操作系统会存在差异，不过对于所有 Unix 类型的系统（包括 OS X），该文件都是位于 **/etc/hosts** 。

本地开发主机可以通过三个命令行选项来进行自定义化:

- 你可以使用 `--host` 命令行选项来指定当前应用所位于的主机名:

  ```
  php spark serve --host=example.dev
  ```

- 默认情况下，服务器在 8080 端口上运行；不过如果你可能会需要多个站点同时运行，或者在 8080 端口上已有一个应用正在部署。那么就可以通过 `--port` 选项来指定另一个端口:

  ```
  php spark serve --port=8081
  ```

- 你也可以指定不同的 PHP 版本，通过 `--php` 选项，同时指定你想使用的对应的 PHP 版本所处的路径:

  ```
  php spark serve --php=/usr/bin/php7.6.5.4
  ```

### 在 Apache 上部署主机

CodeIgniter4 的 webapp 通常部署在一个网站主机上。在本文档中我们将 Apache 的 `httpd` 进程假定为标准使用的平台。

Apache 在许多平台中默认集成，不过也能够以一个安装包（其中包含数据库引擎和 PHP 执行文件）从 [Bitnami] 上下载 (https://bitnami.com/stacks/infrastructure)

#### .htaccess

本文档中假定你可以使用 “mod_rewrite” 模块，该模块可以在 URL 中移除 “index.php” 这一部分。

确保该模块（重写模块）在主配置文件中已被启用（未注释），例如 apache2/conf/httpd.conf:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

与此同时，确保默认的文档根目录 <Directory> 元素中也启用了该模块，在“AllowOverride”设置中:

```
<Directory "/opt/lamp7.2/apache2/htdocs">
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

### 虚拟主机

我们推荐使用虚拟主机的配置来运行你的应用。你可以为每个应用设置不同的别名。

确保虚拟主机模块在主配置文件中启用（未注释），例如 apache2/conf/httpd.conf:

```
LoadModule vhost_alias_module modules/mod_vhost_alias.so
```

在你的主机配置文件（hosts 文件）里增加一个主机别名，在 unix 类型的平台上通常是 /etc/hosts ，而在 Windows 上通常是 c:/Windows/System32/drivers/etc/hosts 。在该文件中增加一行，比如“myproject.local”或“myproject.test”，举例来说:

```
127.0.0.1 myproject.local
```

在虚拟主机配置中，为你的 webapp 增加一个 <VirtualHost> 元素，例如在 apache2/conf/extra/httpd-vhost.conf 中:

```
<VirtualHost *:80>
    DocumentRoot "/opt/lamp7.2/apache2/htdocs/myproject/public"
    ServerName myproject.local
    ErrorLog "logs/myproject-error_log"
    CustomLog "logs/myproject-access_log" common
</VirtualHost>
```

如果你的项目目录并不位于 Apache 的文档根目录下，你的 <VirtualHost> 元素就需要一个嵌套的 <Directory> 元素来为服务器访问这些文件提供授权。

### 测试

上述配置完成后，你的 webapp 应该就可以通过在浏览器上输入 http://myproject.local 的 URL 来进行访问了。

每当你更改了它的配置后，Apache 都需要被重新启动

#### 通过 Vagrant 部署主机

虚拟化也是一个有效地测试你希望部署的环境中的 webapp 的实现情况的方式，即使你是在一个不同环境中进行部署的话。即使你为两个环境使用了相同的平台，虚拟化也可以为测试提供独立的环境。

相关的代码位于 `VagrantFile.dist` 中，该文件也可以被复制到 `VagrantFile` 里，并根据你的系统的情况来进行增减。例如为特定的数据库或缓存引擎提供访问。

#### 设置

我们假设了你已经安装了 VirtualBox 和 Vagrant 的指定平台版本。

我们的 Vagrant 配置文件默认你在系统中使用 ubuntu/bionic64 Vagrant box 。

Vagrant 配置文件假定你是这样进行安装的:

```
vagrant box add ubuntu/bionic64
```

#### 测试

设置完成后，你就可以用以下命令在虚拟机中部署你的 webapp

```
vagrant up
```

你的 webapp 就可以通过 `http://localhost:8080` 来访问，而当次构建的代码覆盖率测试报告可以通过 `http://localhost:8081` ，用户指南通过 `http://localhost:8082` 进行访问。

### 2.1.4 从老版本升级

请阅读你需要升级的版本所对应的升级指南

#### 从 CodeIgniter 3 系列版本升级到 4 系列版本

CodeIgniter 4 是对该框架的重写，并且不向前兼容（对以前的版本不兼容）。比起升级你的应用，更为合适的是转换和重写它。当你做完了这一步（即已经升级到 CodeIgniter4）之后，在 CodeIgniter4 的不同版本间进行升级就会轻而易举。

The "lean, mean and simple" philosophy has been retained, but the implementation has a lot of differences, compared to CodeIgniter 3.

升级过程中并没有 12 步检查列表之类的东西。取而代之的是，在一个新的项目文件夹里开始 CodeIgniter 4 的重新部署开始与使用本框架 ，并开始转换和整合你的应用部件。下面我们会试着指出最需要注意的点。

CI4 中我们并没有完全迁移和重写全部的 CI3 库！参考 CodeIgniter 4 路线图 中的最新列表！

在项目转换之前 **请务必阅读用户指南**!

**下载**

- CI4 同样可以通过解压-运行的 zip 或 tarball 压缩文件的格式进行使用，其中包含有用户指南（在 *docs* 子目录中）
- 它也可以通过 Composer 进行安装

**命名空间**

- CI4 构建基于 PHP7.2+ 的版本，框架中除了辅助函数的所有部分都进行了命名空间标注。

**应用结构**

- `application` 目录被重命名为 `app` ，而框架中仍旧存在着 `system` 文件夹，有着与以往版本一样的功能。
- 本框架现在提供了一个 `public` 目录，希望你可以将其用于项目的根目录
- There is also a `writable` folder, to hold cache data, logs, and session data
- `app` 目录与 CI3 中的 `application` 目录类似，不过有着一些命名的变更，以及将一些子目录移动到 `writable` 目录下。
- 如今已经没有一个嵌套的 `application/core` 目录了，由于我们已经提供了一套不同的机制来扩展框架核心（如下所示）。

**加载类文件**

- 由于对框架组件的引用如今已作为属性动态注入到你的控制器中，现在已经不存在一个 CodeIgniter 的”超级对象”了。
- 类如今已经按需加载了，并且组件也是通过 `Services` 进行维护（服务）
- 类加载器自动处理 PSR4 风格的类文件定位，对于那些以 `App`` (`application` 目录) 和 ``CodeIgniter` system 目录) 为顶级命名空间的类。而通过对 composer 自动加载的支持与智能假设机制，框架甚至可以定位你的那些并未命名空间声明的模型和库文件。
- 你可以配置类的自动加载来支持任何你喜欢的应用结构，包括”HMVC”风格的（译者注：按等级划分的 MVC 模式，简单的解释就是把 MVC 又细分成了多个子 MVC，每个模块就分成一个 MVC）

**控制器**

- 控制器继承了 `\\CodeIgniter\\Controller` 类，而非 `CI_Controller` 类
- 控制器不再需要一个构造函数了（用于调用 CI 魔术方法），除非这是你自己定义的基类控制器的一部分
- CI 提供了 `Request` （请求）and `Response` （响应）对象供你使用，比起 CI3 的风格来说更为强大

- 如果你需要一个基类控制器（比如 CI3 中的 MY_Controller），那么请在你需要的地方使用就行。比如 `BaseController extends Controller`，并使用你自己的类来继承 `BaseController`

**模型**

- 模型继承了 `\\CodeIgniter\\Model` 而非 `CI_Model`
- CI4 的模型拥有更多的功能，包括动态数据库连接，基础的 CRUD（增删改查），模型内验证和自动分页功能。
- CI4 同样拥有可供你构建的 `Entity`（实体）类，用于实现更为丰富的数据表映射功能
- 取消使用 CI3 的 `$this->load->model(x);`，而是使用模型的命名空间模式来调用 `$this->x = new X();`。

**视图**

- 你的视图看起来与从前类似，但是却是以完全不同的方式调用……取消使用 `$this->load->view(x);`，而是通过 `echo view(x);`。
- CI4 支持视图单元，以构建分片响应
- 模板处理器一如过往，但是在功能上有了显著的提升

**库**

- 你的应用类仍旧可以深入访问 `app/Libraries`，但这不是必须的。
- 取消使用 CI3 的 `$this->load->library(x);` 调用方式，如今你可以使用组件的命名空间模式来调用 `$this->x = new X();`

**辅助函数**

- 辅助函数与以往大致相似，不过有一部分被简化了

**事件**

- Hooks（钩子）如今已经被 Events（事件）所取代
- 取消使用 CI3 的 `$hook['post_controller_constructor']` 调用方式，如今你可以使用命名空间 `CodeIgniter\Events\Events;` 下的 `Events::on('post_controller_constructor', ['MyClass', 'MyFunction']);`。
- 事件保持启用状态并全局可用。

**扩展框架**

- 你不需要一个 `core` 目录来保存类似 `MY_...` 的框架组件扩展或替代品。
- 在库目录下，你不需要类似 `MY_x` 之类的类来继承或取代 CI4 的框架部分。
- 你可以在任何地方创建这样的类，并加入适当的自定义组件来代替默认的那些组件。

### 2.1.5 故障排除

以下是一些常见的安装问题，以及建议的解决方法。

#### 我必须在我的 URL 中包含 index.php

如果 “/mypage/find/apple“类似的 URL“/index.php/mypage/find/apple“不起作用，但类似的 URL ，则你的 “.htaccess“规则（对于 Apache）未正确设置。

#### 仅加载默认页面

如果你发现无论你在 URL 中放入什么内容，只会加载默认页面，可能是你的服务器不支持提供搜索引擎友好 URL 所需的 REQUEST_URI 变量。首先，打开 *application/Config/App.php* 文件并查找 URI 协议信息。它会建议你尝试一些备用设置。如果在你尝试此操作后仍然无效，则需要强制 CodeIgniter 向你的网址添加问号。为此，请打开 *application/Config/App.php* 文件并更改

```
public $indexPage = 'index.php';
```

To this:

```
public $indexPage = 'index.php?';
```

#### 该教程给出了 404 错误:(

你无法使用 PHP 的内置 Web 服务器来学习本教程。它不处理正确路由请求所需的 ‘.htaccess‘文件。

解决方案：使用 Apache 为你的站点提供服务。

### 2.1.6 CodeIgniter 仓库

The CodeIgniter 4 开源项目有着属于它自己的 Github 组织.

对于感兴趣的开源贡献者来说，有以下几个开发仓库。

| 仓库 | 受众 | 描述 |
|---|---|---|
| CodeIgniter4 | 贡献者 | 项目核心仓库，包括测试和源文档结构 |
| translations | 开发者 | 系统信息的翻译 |
| coding-standard | 贡献者 | 代码风格的统一和相关规约 |
| | | |

以下是几个开发的仓库，在安装指南里提到过。当新版本发布时，development 仓库将执行自动构建，该仓库不能被直接提交代码。

| 仓库 | 受众 | 描述 |
|---|---|---|
| framework | 开发者 | 框架的已发布版本 |
| appstarter | 开发者 | 项目启动器（app/public/writable）依赖于 “framework” |
| userguide | 任何人 | 预构建的用户手册 |
| | | |

在上述所有仓库中，仓库的最新版本可以通过选择 Github 仓库页面的第二层导航条的 Code（代码）导航块中的” release”（发布）链接来进行下载。

### Composer 包

我们同样在 packagist.org 维护着一个通过 Composer 进行安装的包以下对应着上文提及的相关仓库:

- codeigniter4/framework
- codeigniter4/appstarter
- codeigniter4/translations
- codeigniter4/coding-standard

参照文档安装 页面来获取更多信息.

### CodeIgniter 4 项目

我们在 Github 上也维护着 codeigniter4projects 组织，用于管理那些不属于框架的一部分，但是可以用来更为轻松地使用框架的项目！

| 仓库 | 受众 | 描述 |
|---|---|---|
| website2 | 开发者 | codeigniter.com 网站，通过 CodeIgniter 4 来进行构建 |
| playground | 开发者 | Basic code examples in project form. Still growing. |

这些仓库不能通过 composer 来进行安装

不论你选择何种方式安装并运行 CodeIgniter4，你都可以随时访问我们的在线 用户指南。

---

**注解:** 在使用 CodeIgniter 4 之前，请确保你的服务器满足要求，尤其是所需的 PHP 版本和 PHP 扩展。例如，你可能会发现必须取消注释 `php.ini` 的 “extension” 部分才能启用 “curl” 和 “intl”。

---

# 章节 3

# 构建你的第一个应用

## 3.1 构建你的第一个应用

### 3.1.1 概述

本教程旨在向你介绍 CodeIgniter4 框架以及 MVC 架构的基本原理，并将会手把手的向你讲解如何开发一个基本的 CodeIgniter 应用。

如果你不熟悉 PHP，建议你先阅读 W3Schools PHP 教程，然后再继续阅读下面的内容。

在本教程中，你将创建一个 **基本的新闻应用程序**。你将从编写可加载静态页面的代码开始。接下来，你将创建一个新闻展示页面，用于从数据库中读取新闻内容。最后，我们再编写一个表单，用于在数据库中创建新闻内容。

本教程将主要关注：

- 模型-视图-控制器的基础知识
- 路由相关的基础知识
- 表单验证
- 使用 CodeIgniter 的“Query Builder”执行基本的数据库查询

整个教程分为几个章节，每个章节仅解释 CodeIgniter 框架功能的一小部分。你将会阅读到以下几个章节：

- 介绍章节，也就是本文，本章将概括性的为你讲解如何下载框架并运行自带的默认应用。
- 加载静态页，本章将为你讲解控制器，视图和路由的基础知识。
- 新闻展示功能，你将在这里开始使用模型，并将进行一些基本的数据库操作。

- 创建新闻项目，本章将介绍更高级的数据库操作和表单验证。
- 结束语，本章将为你提供进一步阅读的指导和一些其他资源。

享受 CodeIgniter 框架的探索之旅。

### 加载静态页

**Note:** 本教程假设你已经下载好 CodeIgniter，并将其安装 到你的开发环境。

首先你需要新建一个 **控制器**来处理静态页。控制器就是用来帮助你完成工作的一个简单的类，它是你整个 Web 应用程序的”粘合剂”。

例如，当访问下面这个 URL 时:

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

根据此 URL 我们可以推测出有一个名称为“news”的控制器，被调用的方法为“latest”,” latest” 方法的作用应该是查询 10 条新闻条目并展示在页面上。在 MVC 模式里，你会经常看到下面格式的 URL:

> http://example.com/{[}controller-class{]}/{[}controller-method{]}/ {[}arguments]

在正式环境下 URL 的格式可能会更复杂，但现在，我们只需要知道这些就够了。

新建一个文件 *application/Controllers/Pages.php*，然后添加如下代码:

```
<?php
class Pages extends CodeIgniter\Controller {

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

你刚创建了一个 Pages 类，有一个方法 view 并可接受一个 $page 的参数。Pages 类继承自 CodeIgniter\Controller 类，这意味着它可以访问 CodeIgniter\Controller 类 (*system/Controller.php*) 中定义的方法和变量。

控制器将是你 Web 应用程序中处理请求的核心。和其他的 PHP 类一样，可以在你的控制器中使用 $this 来访问它。

现在，你已经创建了你的第一个方法，是时候创建一些基本的页面模板了。我们将新建两个 “views” (页面模板) 分别作为我们的页头和页脚。

新建页头文件 *application/Views/Templates/Header.php* 并添加以下代码:

```
<!doctype html>
<html>
<head>
        <title>CodeIgniter Tutorial</title>
</head>
```

(下页继续)

(续上页)

```
<body>

        <h1><?= $title; ?></h1>
```

页头包含了一些基本的 HTML 代码，用于展示页面主视图之前的内容。同时，它还打印出了 `$title` 变量，这个我们之后讲控制器的时候再细说。现在，再新建个页脚文件 *application/Views/Templates/Footer.php*，然后添加以下代码：

```
                <em>&copy; 2016</em>
        </body>
</html>
```

### 在控制器中添加逻辑

你刚新建的控制器中有一个 `view()` 方法，这个方法可接受一个用于指定要加载页面的参数。静态页面的模板目录为：*application/Views/Pages/*。

在该目录中，新建 *Home.php* 和 *About.php* 模板文件。在每个文件中任意输入一些文本然后保存它们。如果你不知道写什么，那就写“Hello World!”吧。

为了加载这些界面，你需要检查下请求的页面是否存在：

```
public function view($page = 'home')
{
    if ( ! file_exists(APPPATH.'/Views/Pages/'.$page.'.php'))
        {
                // Whoops, we don't have a page for that!
                throw new \CodeIgniter\PageNotFoundException($page);
        }

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

        echo view('Templates/Header', $data);
        echo view('Pages/'.$page, $data);
        echo view('Templates/Footer', $data);
}
```

当请求的页面存在时，将给用户加载并展示出一个包含页头页脚的页面。如果不存在，会显示“404 Page not found”的错误页面。

此事例方法中，第一行用以检查界面是否存在，`file_exists()` 是原生的 PHP 函数，用于检查某个文件是否存在。`PageNotFoundException` 是 CodeIgniter 的内置函数，用来展示默认的错误页面。

在页头模板文件中，`$title` 变量代表页面的自定义标题，它是在方法中被赋值的，但并不是直接赋值给 title 变量，而是赋值给 `$data` 数组中的 title 元素。

最后要做的就是按顺序加载所需的视图，`view()` 方法中的参数代表要展示的视图文件

名称。$data 数组中的每一个元素将被赋值给一个变量，这个变量的名字就是数组的键值。所以控制器中 $data['title'] 的值，就等于视图中 $title 的值。

### 路由

控制器已经开始工作了！在你的浏览器中输入 [your-site-url]index.php/pages/view 来查看你的页面。当你访问 index.php/pages/view/about 时你将看到包含页头和页脚的 about 页面。

使用自定义的路由规则，你可以将任意的 URL 映射到任意的控制器和方法上，从而打破默认的规则：http://example.com/[controller-class]/[controller-method]/[arguments]

让我们来试试。打开路由文件 *application/Config/Routes.php* 然后添加如下两行代码，并删除掉其它对 $route 数组赋值的代码。

```
$routes->setDefaultController('Pages/view');
$routes->add('(:any)', 'Pages::view/$1');
```

CodeIgniter 读取路由的规则为从上到下，并将请求映射到第一个匹配的规则。每个规则都是一个正则表达式（左侧）映射到一个控制器和方法（右侧）。当获取到请求时，CodeIgniter 首先查找能匹配到的第一条规则，然后调用相应的可能存在参数的控制器和方法。

你可以在关于 URL 路由的文档中找到更多信息。

路由事例的第二条规则 $routes 数组中使用了通配符（:any）来匹配所有的请求，然后将参数传递给 Pages 类的 view() 方法。

为请求默认的控制器，你必须确定当前路由未被定义或重新编写过。默认的路由文件 **does** 下存在一个处理网站根目录的路由 (/) 规则. 删除以下的路由来确保 Pages 控制器可以访问到我们的 home 页面：

$routes->add( '/' , 'Home::index' );

现在访问 index.php/about。路由规则是不是正确的将你带到了控制器中的 view() 方法？太酷了！

### 新闻展示功能

在上一个章节里，我们写了一个用于展示静态页面的类文件，通过这个简单的例子我们把 CI4 框架里的一些基本的概念讲解了一下。我们还简单的使用了 CI4 里面的路由功能，通过自定义路由规则实现了页面的链接地址净化，使页面的访问地址看起来更整洁和搜索引擎友好。现在，我们要开始进行一些基于数据库的动态内容的开发了。

### 建立教程所需的数据库

我们假设你已经安装和配置 好了用于 CodeIgniter4 运行所需的数据库软件。我们也假设你会使用数据库管理的客户端工具（mysql，MySQL Workbench 或者 phpMyAdmin

等）来运行稍后教程里提供的创建数据库表和插入测试数据所需的 SQL 代码。

下面我们就来教你如何为本教程创建一个数据库，并且正确配置 CodeIgniter 来使用这个数据库。

用你安装好的数据库客户端工具打开数据库，然后运行下面的两段 SQL 代码（MySQL 适用）来创建一个数据表和插入一些测试数据。随着你对 CodeIgniter 的熟悉程度越来越高，这些数据库相关的操作也可以通过程序代码在 CodeIgniter 框架下完成，你可以阅读:doc: *数据迁移 <../dbmgmt/migration>* 和:doc: ‘数据填充 <../dbmgmt/seeds>‘这两个章节来了解相关内容，以便掌握用程序代码操作数据库的相关技能。

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

**Note:** 数据表里的 **slug** 字段在基于互联网访问的网站上是个非常有用的字段。一般在这个字段里存放简短的词语来概括性描述数据内容，这将提升用户打开网址时候的访问体验，并且这种网址也是搜索引擎友好的网址，有利于网站内容的 SEO 优化。

然后我们在数据表里插入如下测试数据：

```
INSERT INTO news VALUES
(1,'Elvis sighted','elvis-sighted','Elvis was sighted at the Podunk␣
↪internet cafe. It looked like he was writing a CodeIgniter app.'),
(2,'Say it isn\'t so!','say-it-isnt-so','Scientists conclude that some␣
↪programmers have a sense of humor.'),
(3,'Caffeination, Yes!','caffeination-yes','World\'s largest coffee shop␣
↪open onsite nested coffee shop for staff only.');
```

### 连接到你的数据库

**CodeIgniter** 安装时会自动生成一个 .env 文件，确保里面的配置信息没有被注释掉，并且和你本地的数据库实际情况相吻合：

```
database.default.hostname = localhost
database.default.database = ci4tutorial
database.default.username = root
database.default.password = root
database.default.DBDriver = MySQLi
```

### 创建你的数据模型文件

我们要求你将数据库的操作代码写在模型（Model）文件里面，以便以后代码重用，不要将这些代码写在控制器（Controller）里。你的模型文件们应该成为你处理数据库相关的增、删、改、查操作的默认地方。交由模型文件来操作数据库或者其他格数的数据文件。

打开 **app/Models/** 目录，在这个目录下面创建一个名字为 **NewsModel.php** 的文件，并在文件里加入如下代码。为了保证代码运行顺利，你需要确认一下已经正确的完成了数据库的相关配置:doc: 这里 *<../database/configuration>*。

```
<?php

    namespace App\Models;

    class NewsModel extends \CodeIgniter\Model
    {
            protected $table = 'news';
    }
```

这段代码和我们上一个章节里创建的控制器里的代码类似。我们通过继承 “CodeIgniter-Model“创建了一个新的模型文件，并加载了 CI4 内置的数据库操作类库。后面我们可以在代码里通过 “$this->db“来调用数据库的相关操作类库。

现在我们的数据库和数据模型文件已经建立好了。我们首先写一个方法从数据库中获取所有的新闻文章。为实现这点，我们将使用 **CodeIgniter** 的数据库抽象层工具查询构建器，通过它你可以编写你的查询代码，并在:doc:‘所有支持的数据库平台 <../intro/requirements>‘上运行。数据模型文件可以方便的和 **查询构建器**一起工作，并且提供了一些方法让你操作数据的时候更加简单。现在向你的模型中添加如下代码。

```
public function getNews($slug = false)
{
        if ($slug === false)
        {
                return $this->findAll();
        }

        return $this->asArray()
                    ->where(['slug' => $slug])
                    ->first();
}
```

通过这段代码，你可以执行两种不同的查询，一种是获取所有的新闻条目，另一种是根据特定的 *slug* 来获取指定的新闻条目。你可能注意到了，我们直接的进行了基于 “$slug“ 变量的数据对比命令，并不需要预先执行相应字段的查询操作，因为:doc: 查询构建器 *<../database/query_builder>* 自动帮我们完成了这个工作。

我们在这里用到的 `findAll()` 和 `first()` 都是 *CodeIgniter4* 的数据模型（Model）基础类里面内置的方法。他们根据我们在数据模型文件里（本例中是 **NewsModel** 文件）声明的 “$table“变量而知道该对哪个数据表进行操作。这些方法通过 **查询构建器**

运行指令操作当前数据表，并且会以数组的形式返回数据查询结果。在这个例子里面，“findAll()“的返回值是包含了指定数据表中的所有数据对象的一个数组。

### 显示新闻

现在，查询已经在数据模型文件里写好了，接下来我们需要将数据模型绑定到视图上，向用户显示新闻条目了。这可以在之前写的 Pages 控制器里来做，但为了更清楚的阐述，我们定义了一个新的 News 控制器，创建在 *app/controllers/News.php* 文件中。

```
<?php namespace App\Controllers;

use App\Models\NewsModel;

class News extends \CodeIgniter\Controller
{
        public function index()
        {
                $model = new NewsModel();

                $data['news'] = $model->getNews();
        }

        public function view($slug = null)
        {
                $model = new NewsModel();

                $data['news'] = $model->getNews($slug);
        }
}
```

阅读上面的代码你会发现，这和之前写的代码有些相似之处。首先，它继承了 *CodeIgniter* 的一个核心类，Controller，这个核心类提供了很多非常有用的方法，它确保你可以操作当前的 Request 和 Response 对象，也可以操作 “Logger“ 类，方便你把日志文件写到磁盘里。

其次，有两个方法用来显示新闻条目，一个显示所有的，另一个显示特定的。你可以看到第二个方法中调用模型方法时传入了 $slug 参数，模型根据这个 *slug* 返回特定的新闻条目。

现在，通过模型，控制器已经获取到数据了，但还没有显示出来。下一步要做的就是将数据传递给视图。我们修改 index() 方法成下面的样子：：

```
public function index()
{
        $model = new NewsModel();

        $data = [
                'news'  => $model->getNews(),
```

(下页继续)

(续上页)

```
                'title' => 'News archive',
        ];

        echo view('templates/header', $data);
        echo view('news/index', $data);
        echo view('templates/footer');
}
```

上面的代码从模型中获取所有的新闻条目，并赋值给一个变量（*news*）。另外页面的标题赋值给了 `$data['title']` 元素，然后所有的数据被传递给视图。现在你需要创建一个视图文件来显示新闻条目了，新建 *app/Views/news/index.php* 文件并添加如下代码。

```
<h2><?= $title ?></h2>

<?php if (! empty($news) && is_array($news)) : ?>

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

                <h3><?= $news_item['title'] ?></h3>

                <div class="main">
                        <?= $news_item['text'] ?>
                </div>
                <p><a href="<?= '/news/'.$news_item['slug'] ?>">View␣
↪article</a></p>

        <?php endforeach; ?>

<?php else : ?>

        <h3>No News</h3>

        <p>Unable to find any news for you.</p>

<?php endif ?>
```

这里，我们通过一个循环将所有的新闻条目显示给用户，你可以看到我们直接采用了 *HTML* 和 *PHP* 混用的写法创建了一个视图页面。如果你希望使用一种模板语言，你可以使用 CodeIgniter 的 视图模版解析类 *<../outgoing/view_parser>* ，或其他的第三方解析器。

新闻的列表页就做好了，但是我们还缺少一个显示特定新闻条目的页面。我们可以调用之前创建的模型里的数据来实现这个功能，你只需要向控制器中添加一些代码，然后再新建一个视图就可以了。回到 `News` 控制器，使用下面的代码替换掉 `view()` 方法：

```
public function view($slug = NULL)
{
```

(下页继续)

(续上页)

```
        $model = new NewsModel();

        $data['news'] = $model->getNews($slug);

        if (empty($data['news']))
        {
                throw new \CodeIgniter\PageNotFoundException('Cannot␣
↪find the page: '. $slug);
        }

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

        echo view('templates/header', $data);
        echo view('news/view', $data);
        echo view('templates/footer');
}
```

我们并没有直接调用 getNews() 方法，而是传入了一个 $slug 参数，所以它会返回相应的新闻条目。最后剩下的事是创建视图文件 app/Views/news/view.php 并添加如下代码。

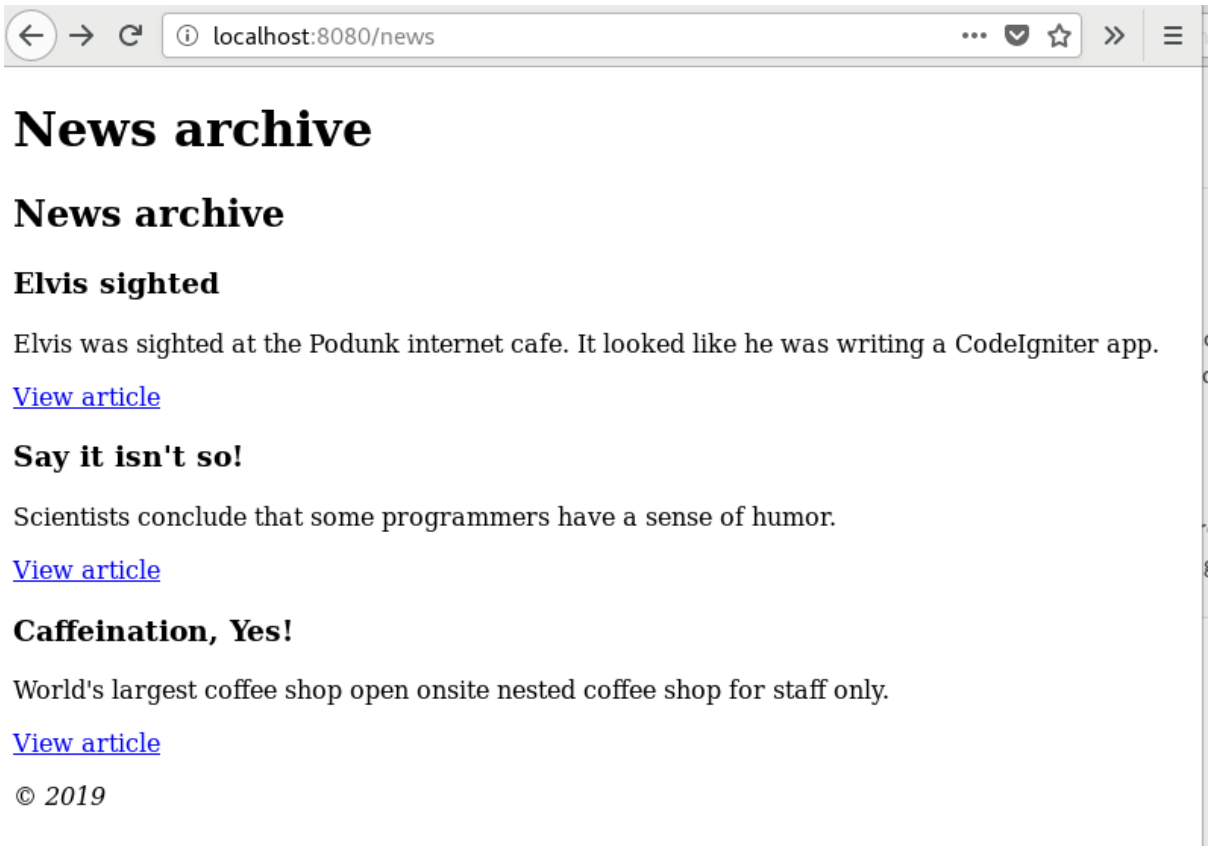
<?php echo ‘<h2>’ .$news[ ‘title’ ].’ </h2>’ ; echo $news[ ‘body’ ];

### 路由

由于之前我们创建了基于通配符的路由规则，所以现在需要新增一条路由以便能访问到你刚刚创建的控制器。修改路由配置文件（**app/config/routes.php**）添加类似下面的代码。该规则可以让地址中带 *news* 的请求访问 News 控制器而不是去访问之前默认的 Pages 控制器。第一行代码可以让访问 *news/slug* 地址的 URI 重定向到 News 控制器的 view() 方法。

```
$routes->get('news/(:segment)', 'News::view/$1');
$routes->get('news', 'News::index');
$routes->get('(:any)', 'Pages::view/$1');
```

在地址栏里输入 *localhost:8080/news* 来访问你创建好的新闻列表页面吧。你将会看到如下图一样的一个展示新闻列表的网页，列表里的每个文章都带一个可以打开该条新闻详情页面的超级链接。




### 创建新闻项目

你现在知道如何使用 CodeIgniter 从数据库读取数据，但你尚未向数据库写入任何信息。在本节中，你将扩展之前创建新的控制器和模型以包含此功能。

### 创建表格

要将数据输入数据库，你需要创建一个表格，你可以在其中输入要存储的信息。这意味着你将需要一个包含两个字段的表格，一个用于标题，另一个用于文本。你将从模型中的标题中获得 slug。在 *application / Views / news / create.php* 创建新视图

```
<h2><?= esc($title); ?></h2>

<?= \Config\Services::validation()->listErrors(); ?>

<?= form_open('news/create'); ?>

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

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

(下页继续)

(续上页)

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

</form>
```

这里只有两个函数你可能不熟悉:*form_open()*‘函数和 ConfigServices::validation()->listErrors()‘函数。第一个函数由:doc:*form helper <../helpers/form_helper>*‘提供，并呈现表格元素并添加额外的功能，例如添加一个隐藏的:doc:*‘CSRF prevention field <../libraries/security>*。后者用于报告与表格验证相关的错误。

回到你的新闻控制器。你将在此处执行两项操作，检查表格是否已提交以及提交的数据是否通过了验证规则。你将使用:doc:‘form validation <../libraries/validation>‘库来执行此操作。

```
public function create()
{
    helper('form');
    $model = new NewsModel();

    if (! $this->validate($this->request, [
        'title' => 'required|min[3]|max[255]',
        'text'  => 'required'
    ]))
    {
        echo view('templates/header', ['title' => 'Create a news item']);
        echo view('news/create');
        echo view('templates/footer');

    }
    else
    {
        $model->save([
            'title' => $this->request->getVar('title'),
            'slug'  => url_title($this->request->getVar('title')),
            'text'  => $this->request->getVar('text'),
        ]);
        echo view('news/success');
    }
}
```

上面的代码添加了很多功能。前几行加载表格 helper 和 NewsModel。之后，Controller 提供的辅助函数用于验证 $_POST 字段。在这种情况下，标题和文本字段是必需的。

如上所示，CodeIgniter 具有强大的验证库。你可以阅读:doc:*more about this library here <../libraries/validation>*。

接下来，你可以看到检查表格验证是否成功运行的条件。如果没有，则显示表格，如果提交并传递了所有规则，则调用模型。这将负责将新闻项传递到模型中。这包含一个新函数 url_title()。这个函数由:doc:‘URL helper <../helpers/url_helper>‘提供，它将你

传递的字符串剥离出来，用短划线 (-) 替换所有空格，并确保所有内容都是小写字符。这给你留下了一个漂亮的 slug，非常适合创建 URI。

在此之后，加载视图以显示成功消息。在 **application/Views/news/success.php** 创建一个视图并写一条成功消息。

### 模型

最适合剩下的就是确保你的模型设置为允许正确保存数据。使用的 “save()“方法将根据主键的存在来确定是否应插入信息，或者行是否已存在且应更新。在这种情况下，没有 “id“传递给它的字段，所以它会在它的表中插入一个新行，即 **news**。

但是，默认情况下，模型中的插入和更新方法实际上不会保存任何数据，因为它不知道哪些字段可以安全更新。编辑模型以在 “$allowedFields“属性中为其提供可更新字段的列表。

```
<?php
class NewsModel extends \CodeIgniter\Model
{
    protected $table = 'news';

    protected $allowedFields = ['title', 'slug', 'text'];
}
```

此新属性现在包含我们允许保存到数据库的字段。请注意，我们遗漏了 “id“？那是因为你几乎不需要这样做，因为它是数据库中的自动递增字段。这有助于防止批量分配漏洞。如果你的模型正在处理你的时间戳，你也可以将其保留。

### 路由

在开始将新闻项添加到 CodeIgniter 应用程序之前，必须向 *Config/Routes.php* 文件添加额外的规则。确保你的文件包含以下内容。这可以确保 CodeIgniter 将’ create’ 视为一种方法，而不是新闻项目的 slug。

```
$routes->post('news/create', 'News::create');
$routes->add('news/(:segment)', 'News::view/$1');
$routes->get('news', 'News::index');
$routes->add('(:any)', 'Pages::view/$1');
```

现在将浏览器指向安装 CodeIgniter 的本地开发环境，并将 index.php/news/create 添加到 URL。恭喜，你刚刚创建了第一个 CodeIgniter 应用程序！添加一些新闻并查看你制作的不同页面。

### 结束语

本教程没有涵盖你对完整内容管理系统可能期望的所有内容，但它向你介绍了路由，编写控制器和模型等更重要的主题。我们希望本教程能让你深入了解 CodeIgniter 的一些

基本设计模式，你可以对其进行扩展。

现在你已完成本教程，我们建议你查看其余文档。CodeIgniter 因其全面的文档而受到称赞。使用它有利于你，并彻底阅读“简介”和“一般主题”部分。你应该在需要时阅读类和帮助程序引用。

每个中级 PHP 程序员都应该能够在几天内获得 CodeIgniter 的支持。

如果你仍然对框架或你自己的 CodeIgniter 代码有疑问，你可以：

- 看看我们的 论坛
- 访问我们的 IRC 聊天室
- 探索 Wiki

### 3.1.2 开始并运行

你可以从官网手动下载框架文件，但是对于本教程，我们将使用推荐的方式并通过 Composer 安装 AppStarter 软件包。在命令行中输入以下内容：

```
composer create-project codeigniter4/appstarter ci-blog -s rc
```

这将创建一个新文件夹 ci-blog，其中包含你的应用程序代码，并且在 vendor 文件夹中安装了 CodeIgniter。

默认情况下，CodeIgniter 以生产模式启动。这是一项安全措施，可防止你的网站在上线后马上被攻击，从而使网站更加安全。所以首先让我们先修改一下这部分内容。将 `env` 文件复制或重命名为 `.env`，并打开它。

该文件包含服务器特定的设置。这意味着你无需将任何敏感信息提交到代码库中。它包括一些常用的公共配置项，尽管它们都已被注释掉。因此，取消注释 CI_ENVIRONMENT 所在的代码行，然后将 `production` 更改为 `development`：

```
CI_ENVIRONMENT = development
```

这样一来，你就可以在浏览器中查看这个应用了。你可以通过你喜欢的任何 Web 服务器（Apache 或 Nginx 等等）为它提供服务，但是 CodeIgniter 也附带了一个简单的命令，该命令使用 PHP 的内建服务器来帮助你在开发机上快速启动并运行。在项目的根目录中，在命令行中输入以下内容：

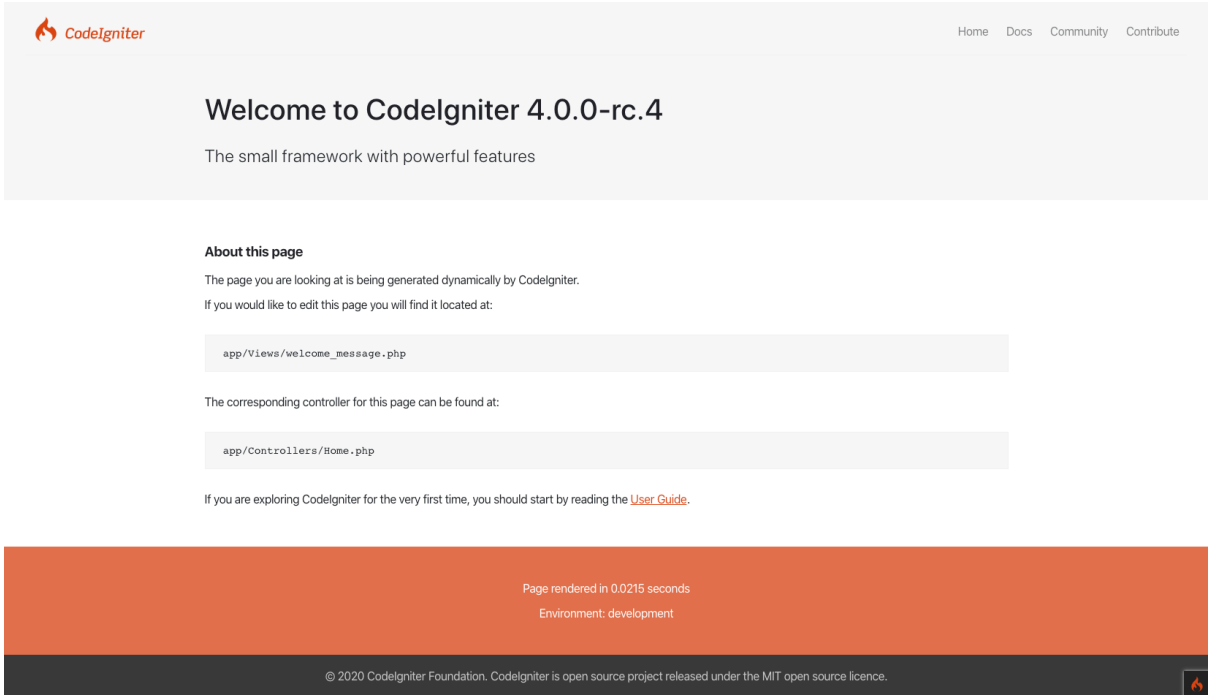
```
php spark serve
```

### 3.1.3 欢迎页

现在，在浏览器地址栏中输入正确的 URL，你将看到框架的欢迎页面。立即尝试以下 URL：

```
http://localhost:8080
```

你将看到类似这样的页面：

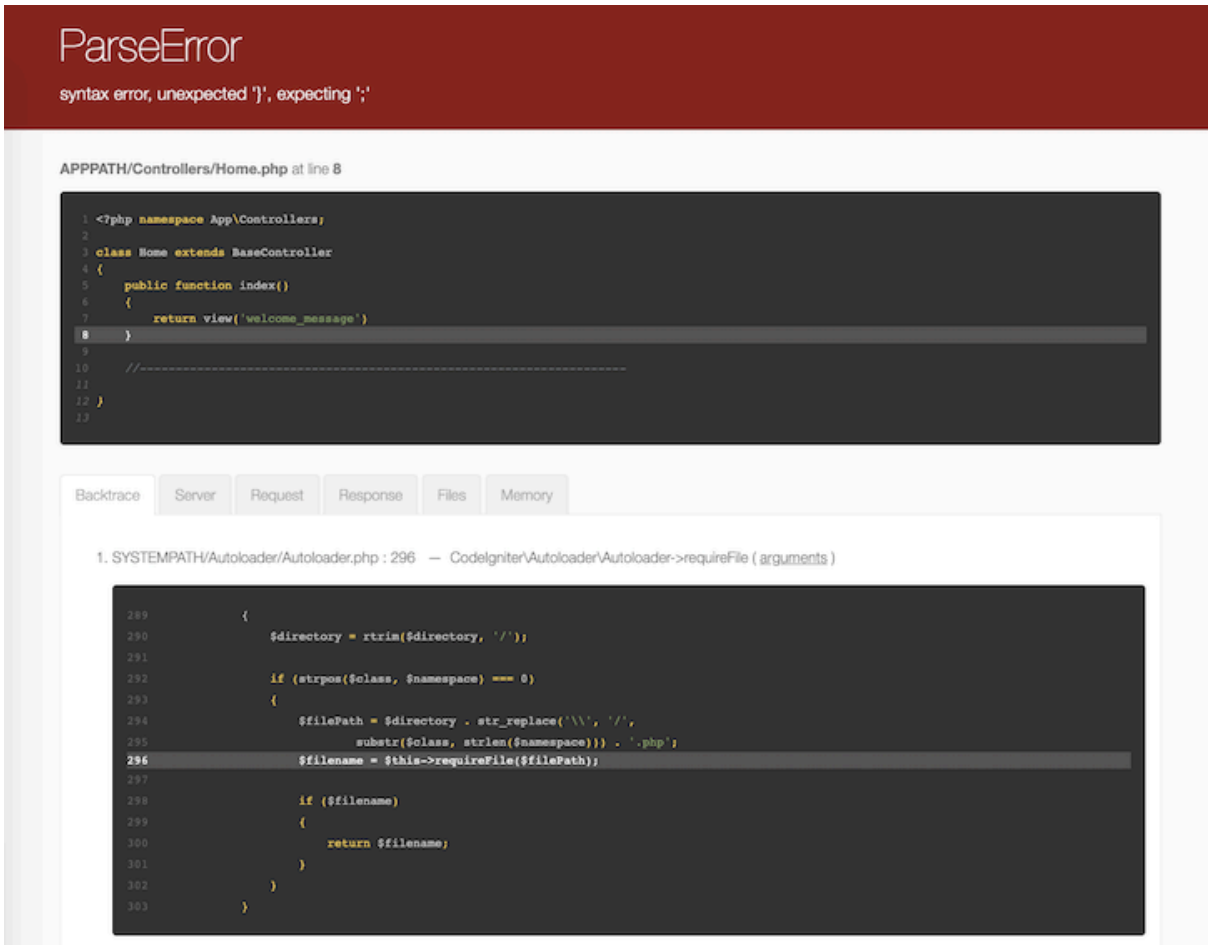



这意味着你的应用程序已经正常运行了，你可以开始编写你自己的代码了。

### 3.1.4 调试

现在你处于开发模式，你将在应用程序底部看到一个工具栏。该工具栏包含许多有用的功能，你可以在开发过程中使用这些功能。这个工具栏不会出现在生产环境中。单击底部的任何选项卡将显示附加信息。单击工具栏右侧的 X 可以将其最小化为带有 CodeIgniter 图标的小方块。如果单击该工具栏将再次显示。

除此之外，当你在程序中遇到异常或其他错误时，CodeIgniter 还会提供一些有用的错误页面。打开 `app/Controllers/Home.php` 并修改一些内容以制造一个错误（删除分号或花括号试试 ~）。屏幕将显示类似以下内容的页面：




这里有几件事要注意：

1. 将鼠标悬停在顶部的红色标题上会显示一个 *搜索* 链接，该链接将在新标签页中打开 Google.com 并搜索此错误的相关信息。
2. 单击 Backtrace 中任意行上的 `arguments` 链接将展开传递到该函数调用中的参数列表。

页面中的其他内容都是简明易懂的。

现在，我们已经入门了，并了解了一点调试程序的方法，下面让我们开始构建这个小型新闻应用吧。

# 章节 4

# 概览和常规主题

## 4.1 CodeIgniter4 概览

以下内容描述了 CodeIgniter4 背后的架构理念：

### 4.1.1 应用结构

为了可以充分利用 CodeIgniter，你需要了解应用程序的结构，默认情况下，你可以更改内容以满足你的应用程序的需求。

#### 默认目录

新安装的应用程序中有六个目录：`/app`，`/system`，`/public`，`/writable`，`/tests` 和 `/docs`。这些目录中的每一个都有一个非常具体的使用规范。

#### app

`app` 目录是你所有应用程序代码所在的目录。它带有一个默认的目录结构，适用于许多应用程序。以下文件夹构成基本内容：

```
/app
        /Config         Stores the configuration files
        /Controllers    Controllers determine the program flow
        /Database       Stores the database migrations and seeds files
        /Filters        Stores filter classes that can run before and␣
↪after controller
```

(下页继续)

(续上页)

```
        /Helpers        Helpers store collections of standalone functions
        /Language       Multiple language support reads the language␣
↪strings from here
        /Libraries      Useful classes that don't fit in another category
        /Models         Models work with the database to represent the␣
↪business entities.
        /ThirdParty     ThirdParty libraries that can be used in␣
↪application
        /Views          Views make up the HTML that is displayed to the␣
↪client.
```

由于 app 目录已经是命名空间，因此你可以随意修改此目录的结构以满足应用程序的需要。例如，你可能决定开始使用存储库模式和实体模型来处理数据。在这种情况下，你可以将 Models 目录重命名为 Repositories，并添加新 Entities 目录。

---

**注解:** 如果重命名 `Controllers` 目录，则无法使用路由到控制器的自动方法，并且需要在你的路由文件中定义所有路由。

---

此目录中的所有文件都位于 `App` 命名空间下，你可以在 **app/Config/Constants.php** 文件中自由更改。

### system

该目录存储构成框架的文件本身。虽然你在使用应用程序目录方面具有很大的灵活性，但系统目录中的文件永远不应该被修改。相反，你应该扩展类或创建新类，以提供所需的相应功能。

此目录中的所有文件都位于 `CodeIgniter` 命名空间下。

### public

**public** 文件夹包含 Web 应用程序的浏览器可以直接访问的地址，防止源代码的直接访问。它包含主要的 **.htaccess** 文件，**index.php** 以及其它你想要添加的样式文件地址，比如 CSS，javascript 或图像。

这个文件夹将成为你站点的”Web 根目录”，并且你的 Web 服务器配置将指向它。

### writable

此目录包含在应用程序生命周期中可能需要写入的所有目录。包括用于存储缓存文件，日志和任何用户可能发送使用的目录。你可以在此处添加应用程序需要写入的任何其他目录。这允许你将其他主目录保持为不可写，作为附加的安全措施。

#### tests

此目录设置为测试文件的存储地址。`_support` 目录包含各种模拟类和其他在编写测试时可以使用的实用程序。该目录请在生产环境中忽略提交/传输到生产环境中。

#### docs

如果此目录是你项目中的一部分，那么此目录包含 CodeIgniter4 用户指南的本地副本。

#### 修改目录位置

如果你需要重定位任何主目录位置，可以在 `app/Config/Paths` 更改配置。

详情请参考 管理你的应用

### 4.1.2 模型，视图和控制器

当创建一个应用的时候，我们需要找到一种组织代码的方法，使其易于找到正确的文件并且易于维护。和很多 Web 框架类似，CodeIgnite 框架也使用了模型、视图、控制器结构，即 MVC 模式，来组织接着代码文件。这种方式可以将数据，展示部分和流程部分分别作为单独的部分存放在我们的应用中。需要注意的是，对于每一个元素的所担任的角色会有很多种看法，本文描述了我们对此的看法。如果你有不同的看法，则可以根据需要随意修改使用方式。

**模型**主要用来管理应用的数据, 根据应用的特殊业务规则获取数据。

**视图**是一个没有或者少量逻辑的简单的文件, 它只负责将数据展示给用户。

**控制器**主要承担了胶水代码的功能, 它主要在视图层和数据存储之间来回的处理并整合数据。

在最简单的情况下，控制器和模型只是一个完成特定工作的类。他们虽然不是你可以使用的唯一类的类型，但他们是构成整个框架的核心。你也可以将控制器和模型文件存储在任何你需要的位置，但是 CodeIgnite 框架在 **/app** 目录中为我们指定了存储目录。我们将在之后进行详细讨论。

下面我们就来看一下这三个主要组成部分。

#### 组成

#### 视图

视图是最简单的文件，一个视图文件通常是一个 HTML 文件加入少量的 PHP 代码。视图中的 PHP 代码应该尽可能的简单，一般只是显示一个变量内容，或者通过循环语句将数据输出在表格中展示出来。

视图从控制器中获取数据并展示——控制器将数据发送给视图，视图通过简单的 `echo` 调用将数据展示出来。你也可以在一个视图中插入展示其他视图，这样可以很简单的在每个页面上展示出公共的页眉和页脚。

视图文件通常存放在 **/app/Views** 目录下，如果在创建文件时不按照一定的规则创建的话，会显得我们的代码杂乱无章。CodeIgnite 框架虽然没有规定任何的规则，但通过经验我们规定在 **Views** 目录下创建一个新的目录对应每个控制器。然后通过方法名来命名视图。这样就会使我们之后查找起来更加容易。例如：用户配置可能会显示在一个名为 `User` 的控制器中，并且方法名称为 `profile` ，你就可以将该视图文件保存在 **/app/Views/User/Profile.php** 这个路径下，并这样命名。

这种良好的组织代码方式建议养成一个习惯。可能有些时候，你有一些其他需求需要以其他方式来组织代码，没关系，只要 CodeIgnite 框架可以找到这个文件，这个视图就会被显示。

点击这里阅读更多关于视图的相关内容

### 模型

对于许多开发人员而言，在确定执行哪些业务规则时会困惑。模型的主要任务是给应用维护单一类型的数据。比如：用户，博客内容，交易信息等。所以，模型的工作有以下两种，对数据进行采集或者放入数据库中执行业务规则；检索数据并将数据库中的数据读取出来。也就是进行数据的增删改查的操作。

数据的任何限制和要求都由模型层承担，包括在保存数据前将原始数据初始化，或者在数据传给控制器前将数据格式化。这样可以保证你可以不用在多个控制器中出现重复代码，或者出错。

模型类型的文件保存在 **/app/Models** 这个目录下，虽然他们也可以使用一个命名空间分组，但是还是建议你将模型文件放在这个目录下。

点击这里阅读更多关于模型的相关内容

### 控制器

控制器主要承担了几个不同的角色。最常见的就是他们会接收用户的请求，然后判断这个请求应该执行什么样的操作。而这一过程通常会涉及到将数据发送给模型层保存，或者去请求模型层的数据返回给视图。控制器也会用来加载其他应用程序请求的除模型参与的任务。

控制器的林外的任务就是用来处理和 HTTP 请求相关的所有事情——重定向、认证，Web 安全，编码等。总之，控制器是你的应用程序的入口，通过控制器访问你的应用的用户才可以到达指定的地方并获取他们想要的数据使用格式。

控制器通常会保存在 **/app/Controllers** 这个路径下，虽然你也可以使用命名空间分组，但是还是建议你将控制器存放在该目录下。

点击这里阅读更多关于控制器的相关内容

### 4.1.3 自动加载文件

每个应用都在不同的位置包含有大量的类文件。框架提供了实现核心功能的类，而你的应用将会由大量的库，模型，以及其他实体文件以运行。你也可能需要第三方的类库以供项目使用。记录每个单独的文件的位置，并硬编码一系列的 `requires()` 在文件中，这是一个非常头疼且容易出错的事情。这就是自动加载器的用武之地。

CodeIgniter 提供了一个非常灵活且需要极少配置的自动加载器。它可以定位单个的非命名空间标注的类，符合命名空间规范 PSR4 目录加载结构的类，甚至可以在常规目录下定位类文件（例如控制器，模型等）。

为了提升性能，CodeIgniter 的核心组件已被添加到类映射文件中。

自动加载器可以单独运行，如果你需要的话，可以和其他自动加载器协同运行，例如 Composer 或者是你自己的自定义加载器。因为它们都是通过 spl_autoload_register 来注册运行的，所以可以依次运行，互不打扰。

自动加载器总是处于激活状态，并通过 `spl_autoload_register()` 在框架运行开始时进行注册挂载。

#### 配置

初始配置是在 **/application/Config/Autoload.php** 文件中进行。该文件包含两个主要的数组，一个用于类映射图，一个用于符合 PSR-4 规范的命名空间。

#### 命名空间

我们推荐通过在应用文件里创建一个或多个命名空间来管理你的类。而这一点对于业务逻辑相关联的类，实体类等也是最为重要的。配置文件中的 `psr4` 数组允许你将命名空间和对应的类所存在的目录进行映射:

```
$psr4 = [
        'App'          => APPPATH,
        'CodeIgniter' => SYSTEMPATH,
];
```

数组的每一行的键就是命名空间本身，不需要反斜杠 ()。如果你需要在定义数组时使用双引号，确保使用反斜杠进行转义。这意味着应当如同 `My\\App` 而不是 `My\App` 这样。对应的值就是这些类所存在的目录，而这些需要包括反斜杠。

默认来说，应用文件夹对应着 `App` 命名空间。尽管你不一定非得给应用目录下的控制器，库和模型声明命名空间，但是如果你这样做了的话，这些文件就会在 `App` 命名空间下被找到。你可以通过编辑 **/application/Config/Constants.php** 文件来改变这个命名空间，并且通过更改 `APP_NAMESPACE` 选项来设置新的命名空间值:

```
define('APP_NAMESPACE', 'App');
```

你需要修改所有现存的指向当前命名空间的文件。

**重要：** 配置文件的命名空间是 Config ，而不是如你所想的 App\Config。这一特性使得核心系统文件可被准确定位，甚至在应用的命名空间被更改的情况下。

#### 类映射图

类映射图是 CodeIgniter 用来榨干系统最后一分性能的手段，通过不使用额外的 file_exists() 调用来查询文件系统来实现。你可以利用类映射图来链接到第三方库，即使它们并没有命名空间:

```
$classmap = [
        'Markdown' => APPPATH .'third_party/markdown.php'
];
```

每一行的键就是你所需要定位的类名。值就是需要定位的路径

#### 支持从前版本

如果以上的所有方法都找不到对应的类文件，且这个类没有对应的命名空间，自动加载器将会查找 **/application/Libraries** 和 **/application/Models** 目录来尝试定位文件。这为从以前版本升级提供了一个简洁的方式。

对于支持从前版本而言，没有额外的配置选项。

#### 支持 Composer

默认情况下 CodeIgniter 自动初始化支持 Composer. 默认情况下，它将在 ROOTPATH. 'vendor / autoload.php' 中查找 Composer 的自动加载文件。如果因为一些原因需要更改这个文件的位置，你可以修改定义在 Config\Constants.php 的值

**注解：** 如果在 CodeIgniter 和 Composer 中都定义了相同的名称空间，则 CodeIgniter 的自动加载器将首先定位文件。

### 4.1.4 服务

- 共享类
- 服务发现

## 引言

在 CodeIgniter 内部的所有类实际上都是以”服务”的形式呈现的。这意味着，所有的类都是以定义在一个简单的配置文件里，而非硬编码所需要加载的类名，来进行加载的。该配置文件实际上扮演了一种为所需类创建新的实例的工厂的角色。

一个简单的例子可能会讲得更清楚，比如请设想你需要获得一个 Timer （计时器）类的实例，最简单的方法就是为该类创建一个新的实例:

```
$timer = new \CodeIgniter\Debug\Timer();
```

这种方式运行得相当不错，直到你决定需要在该位置上使用另一个计时器类时。可能这个类比起默认的计时器类提供了更高级的报告方法。为了实现这一目标，你可能会查找应用中的所有位置来定位哪些地方使用了定时器类。由于你可能在很多地方都设置了该类的实例，以获取应用日常运行的性能日志，这种查找-替换的工作可能会变得相当的耗时并且错误频发。这就是服务的用武之地。

取代了手动创建实例的操作，我们保留了一个中央控制类来为我们新建实例。该类的结构相当简单，仅仅包含了一个方法用于调度我们需要用作服务的每个类。该方法只是返回了指定类的一个共享实例，用于所有依赖该类的地方以服务的形式来调用。从而我们可以用以下代码来取代每次都新建一个实例的方式:

```
$timer = \Config\Services::timer();
```

当你想要更改这一实现时，只需要更改服务的配置文件，从而在应用中就可以自动地进行变更替换，不需要任何其他操作。现在你所需要的只是使用新的替换上来的类的特性，非常地简单且不易出错。

---

**注解:** 我们推荐只在控制器里创建服务。在其他文件例如模型和库，应当依赖于构造函数或者 setter 方法的传参来实例化。

---

## 便利的方法

有两个方法被用于获取一个服务，这些方法都非常的方便。

第一个就是 `service()` 方法，该方法返回了指定服务的新的实例。唯一需要的参数就是服务名。该方法与服务文件内部返回共享实例的方式是一样的，因此对该方法的多次调用总是会返回一个相同的实例:

```
$logger = service('logger');
```

如果创建的方法需要额外的参数，那么这些参数就应该在服务名后传递:

```
$renderer = service('renderer', APPPATH.'views/');
```

第二个方法 single_service() ，和 service() 一样调用，不过每次都会返回一个指定类的新的实例:

```
$logger = single_service('logger');
```

### 定义服务

为了保证服务的正常运行，你需要能够对每个拥有常量 API，或者实现了 接口 的类建立依赖。CodeIgniter 的大部分类都提供了一个它们所应当提供的接口。当你需要扩展或者替代核心类时，你只需要确保自己符合这些接口的要求并且确定这些类的功能是完善的。

举例来说，RouterCollection 类实现了 RouterCollectionInterface 接口。当你想要替代该类，并实现不同的路由管理方法时，只需要创建一个实现了 RouterCollectionInterface 接口的类即可:

```
class MyRouter implements \CodeIgniter\Router\RouteCollectionInterface
{
        // Implement required methods here.
}
```

最后，修改 **/app/Config/Services.php** 以创建 MyRouter 类的实例，来替代 CodeIgniter\Router\RouterCollection

```
public static function routes()
{
        return new \App\Router\MyRouter();
}
```

### 允许使用参数

在某些情况下，你可能想要使用某个选项来为一个类在实例化的时候传递配置信息。由于服务文件只是简单的类文件，如上操作非常方便。

renderer 服务就是一个不错的例子。默认情况下，我们需要该类能够找到 APPPATH.views/ 目录下的视图文件。我们同时也想为开发者提供改变路径的选项（如果他们需要的话）。因此该类接受 $viewPath 变量作为构造函数的参数。该服务的调用方法可能如下所示:

```
public static function renderer($viewPath=APPPATH.'views/')
{
        return new \CodeIgniter\View\View($viewPath);
}
```

这一过程在构造函数方法里设置了默认路径，同时也可以轻松地改变其所使用的路径:

```
$renderer = \Config\Services::renderer('/shared/views');
```

### 共享类

某些情况下你可能只需要创建一个类的单实例。该操作可以通过工厂方法内部调用的 getSharedInstance() 方法来轻松地处理。该方法检查了该类是否已创建了存储于内部的单个实例，如果没有的话，就会创建一个新的。所有的工厂方法都会提供一个 $getShared = true 的值作为最后的参数。你可以像这样操作该方法:

```
class Services
{
    public static function routes($getShared = false)
    {
        if (! $getShared)
        {
            return new \CodeIgniter\Router\RouteCollection();
        }

        return static::getSharedInstance('routes');
    }
}
```

### 服务发现

CodeIgniter 可以自动发现所有你在其他命名空间里可能定义的 Config\Services.php 文件。这一功能允许了任何模块服务化文件的简单使用。为了这些定制化的服务文件可以被自动发现，他们需要满足这些要求

- 它们的命名空间必须在 Config\Autoload.php 中已定义
- 在命名空间内部，该文件必须可以在 Config\Services.php 里被找到
- 它们必须继承 CodeIgniter\Config\BaseService 类

一个小例子可以帮助我们更好地理解。

假设你创建了一个新的目录，比如在根目录下的一个叫做 Blog 的目录。该目录中里有一个 **博客模块**，并含有控制器，模型等文件。如果你愿意的话也可以将某些类作为服务而使用。第一步就是创建一个新的文件: Blog\Config\Services.php ，该文件结构应当如下所示:

```
<?php namespace Blog\Config;

use CodeIgniter\Config\BaseService;

class Services extends BaseService
{
```

(下页继续)

(续上页)

```
    public static function postManager()
    {
        ...
    }
}
```

现在你可以使用如上描述的文件。每当你想要调用其他控制器的 posts 服务时，就可以简单地使用该框架的 Config\Services 类来获取你所需要的服务:

```
$postManager = Config\Services::postManager();
```

**注解:** 如果多个服务文件拥有相同的方法名，那么第一个被发现的服务实例就会作为返回值。

### 4.1.5 处理 HTTP 请求

为了充分地使用 CodeIgniter，你需要对 HTTP 请求和响应的工作方式有基本的了解。因为在开发 Web 应用时需要处理 HTTP 请求，所以对于所有想要成功的开发者来说，理解 HTTP 背后的概念是 **必须**的。

本章的第一部分会给出一些关于 HTTP 的概述，接着我们会讨论怎样用 CodeIgniter 来处理 HTTP 请求与响应。

#### 什么是 HTTP ？

HTTP 是两台计算机相互通信的一种基于文本的协议。当浏览器请求页面时，它会询问服务器是否可以获取该页面。然后，服务器准备页面并将响应发送回发送请求的浏览器。就是这样简单，也可以说复杂些，但基本就是这样。

HTTP 是用于描述该交换约定的术语。它代表超文本传输协议（Hypertext Transfer Protocol）。开发 web 应用程序时，你的目标只是了解浏览器的要求，并能够做出适当的响应。

#### HTTP 请求

当客户端（浏览器，手机软件等）尝试发送 HTTP 请求时，客户端会向服务器发出一条文本消息然后等待响应。

这条文本消息会像这样:

```
GET / HTTP/1.1
Host codeigniter.com
Accept: text/html
User-Agent: Chrome/46.0.2490.80
```

这条消息包含了所有服务器可能需要的信息。比如它请求的 method（GET，POST，DELETE 等）、它所支持的 HTTP 版本。

该请求还包括许多可选的请求头字段，这些头字段可以包含各种信息，例如客户端希望内容显示为哪种语言，客户端接受的格式类型等等。Wikipedia 上有一篇文章，列出了 所有的请求头字段 （译者注：国内用户如果无法访问的话，可以查看 在 MDN 上的页面 ）。

### HTTP 响应

服务器收到请求后，你的 web 应用程序会处理这条信息然后输出一些响应结果。服务器会将你的响应结果打包为对客户端的的你的响应结果打包为对客户端的响应的一部分。服务器对客户端的响应消息看起来会像这样：

```
HTTP/1.1 200 OK
Server: nginx/1.8.0
Date: Thu, 05 Nov 2015 05:33:22 GMT
Content-Type: text/html; charset=UTF-8

<html>
        . . .
</html>
```

响应消息告诉客户端服务器正在使用的 HTTP 版本规范，以及响应状态码（200）。状态码是标准化的对客户端具有非常特定含义的代码。它可以告诉客户端响应成功（200），或者找不到页面（404）等等。在 IANA 可以找到 完整的响应状态码列表 。

### 对 HTTP 请求和响应的处理

虽然 PHP 提供了与 HTTP 请求和响应进行交互的原生方式，但 CodeIgniter 像大多数框架一样，将它们抽象化，让你拥有一个一致、简单的接口。*IncomingRequest 类* 类是 HTTP 请求的面向对象的表示形式。它提供你所需要的一切：

```
use CodeIgniter\HTTP\IncomingRequest;

$request = service('request');

// 请求的 uri（如 /about ）
$request->uri->getPath();

// 检索 $_GET 与 $_POST 变量
$request->getGet('foo');
$request->getPost('foo');

// 从 $_REQUEST 检索，其中应同时包含 $_GET 和 $_POST 内容
$request->getVar('foo');
```

（下页继续）

(续上页)

```
// 从 AJAX 调用中检索 JSON
$request->getJSON();

// 检索 server 变量
$request->getServer('Host');

// 检索 HTTP 请求头，使用不区分大小写的名称
$request->getHeader('host');
$request->getHeader('Content-Type');

$request->getMethod();  // GET, POST, PUT 等等
```

request 类会在后台为你做很多工作，你无需担心。`isAJAX()` 和 `isSecure()` 函数会自动检查几种不同的 method 来最后确定正确的答案。

---

**注解：** `isAJAX()` 函数依赖于 `X-Requested-With` 头部，这个头部在一些情况下，不会在 XHR 请求中通过 JavaScript 默认发送。想要了解如何避免这个问题，请参考*AJAX Requests* 章节

---

CodeIgniter 还提供了*Response 类*，它是 HTTP 响应的面向对象式表示。它为你提供一种简单而强大的方法来构造对客户的响应：

```
use CodeIgniter\HTTP\Response;

$response = service('response');

$response->setStatusCode(Response::HTTP_OK);
$response->setBody($output);
$response->setHeader('Content-type', 'text/html');
$response->noCache();

// 把响应结果发给浏览器
$response->send();
```

另外，*Response 类* 还允许你处理 HTTP 缓存层以获得最佳性能。

### 4.1.6 安全指南

我们需要认真对待安全问题。CodeIgniter 有多项功能和技术来执行良好的安全习惯，这样你需要做的就比较简单。

我们尊重 开放式 Web 应用程序安全项目 (OWASP) 组织并且尽可能遵循他们的建议。

以下是来自 OWASP Top Ten Cheat Sheet，确定 Web 应用程序上的漏洞。针对每一个漏洞，我们提供了一个简短的描述和 OWASP 建议，然后根据 CodeIgniter 的规定来解

决这个漏洞。

### A1 注入

注入攻击是通过客户端的输入向应用程序发送部分或全部不适当的插入数据。攻击向量包括 SQL、XML、ORM、代码和缓冲区溢出。

#### OWASP 建议

- 说明：设置正确的内容类型、字符集和区域
- 提交：验证字段并且提供反馈
- 控制器：净化输入；使用正确的字符集验证输入
- 模型：参数化检查

#### CodeIgniter 规定

- HTTP library 提供输入字段和内容元数据的过滤
- 表格验证库

### A2 不严谨的的身份认证和会话管理

不充分的身份验证或不恰当的会话管理会导致用户获得比他们权限更大的权限。

#### OWASP 建议

- 说明：验证认证和角色；用表格发送 CSRF token
- 设计：只使用内置会话管理
- 控制器：验证用户、角色、CSRF token
- 模型：验证角色
- 提示：考虑使用 request 管理器

#### CodeIgniter 规定

- Session 库
- HTTP library 提供对 CSRF 的验证
- 方便添加第三方认证

### A3 跨站脚本 (XSS)

输入验证不足导致其中一个用户可以将内容添加到一个网站，当其他用户查看该网站时，该网站可能是恶意的。

#### OWASP 建议

- 说明：根据输出环境对所有用户数据进行转义；设置输入约束
- 控制器：正确的输入验证
- 提示：只处理可信数据；不要将 HTML 转义数据存入数据库中。

#### CodeIgniter 规定

- esc 函数
- 表格验证库

### A4 直接引用不安全的对象

当应用程序根据用户提供的输入提供直接访问时，就会发生不安全的直接对象引用。由于此漏洞，攻击者可以绕过系统中的授权直接访问资源，例如数据库记录或文件。

#### OWASP 建议

- 说明：不要暴露内部数据；使用随机的参考图
- 控制器：获得的数据来自可信任的来源或随机的参考图
- 模型：更新数据之前验证用户角色

#### CodeIgniter 规定

- 表格验证库
- 容易添加第三方认证

### A5 安全配置错误

应用程序体系结构配置不当会导致可能危及整个架构安全性的错误。

### OWASP 建议

- 说明：强化 Web 和应用服务器；使用 HTTP 严格传输安全
- 控制器：强化 Web 和应用服务器；保护 XML 堆栈
- 模型：强化数据库服务器

### CodeIgniter 规定

- bootstrap 合理的检查

### A6 敏感信息泄露

敏感数据在通过网络传输时必须受到保护。敏感数据可以包括用户凭证和信用卡。根据经验，如果数据在存储时必须受到保护，那么它在传输过程中也必须受到保护。

### OWASP 建议

- 说明：使用 TLS1.2 （安全传输层协议）；使用强密码和哈希；不要把 keys 或哈希发送到浏览器
- 控制器：使用强密码和哈希
- 模型：加密和服务器的通信和授权

### CodeIgniter 规定

- 存储加密的会话密钥

### A7 缺少功能级访问控制

敏感数据在通过网络传输时必须受到保护。敏感数据可以包括用户凭证和信用卡。根据经验，如果数据在存储时必须受到保护，那么它在传输过程中也必须受到保护。

### OWASP 建议

- 说明：确保非 Web 数据在 Web 根目录之外；验证用户和角色；发送 CSRF tokens
- 控制器：验证用户和角色；验证 CSRF tokens
- 模块：验证角色

### CodeIgniter 规定

- 公共文件夹，放在 application 和 system 外面
- HTTP library 提供 CSRF 验证

### A8 跨站请求伪造（CSRF）

CSRF 是一种攻击，它迫使最终用户在当前已通过身份验证的 Web 应用程序上执行不必要的操作。

### OWASP 建议

- 说明：验证用户和角色；发送 CSRF tokens
- 控制器：验证用户和角色；验证 CSRF tokens
- 模型：验证角色

### CodeIgniter 规定

- HTTP library 提供 CSRF 验证

### A9 使用含有已知漏洞的组件

许多应用程序都可以利用漏洞和已知的攻击策略，获得远程控制或者得到数据。

### OWASP 建议

- 不要使用这些有漏洞的组件

### CodeIgniter 规定

- 添加第三方库时必须审查

### A10 未验证的重定向和转发

错误的业务逻辑或注入可操作的代码可能会错误地重定向用户。

#### OWASP 建议

- 说明：不要使用 URL 重定向；使用随机的间接引用
- 控制器：不要使用 URL 重定向；使用随机的间接引用
- 模型：验证角色

#### CodeIgniter 规定

- HTTP library 提供…
- Session library provides flashdata

## 4.2 常规主题

### 4.2.1 配置文件

每一个项目，都需要一种方法来定义不同的全局配置项，而这通常是借助配置文件来实现的。而配置文件，一般来说，是通过声明一个将所有的配置项作为公开属性的类，来实现这一配置过程的。

Unlike many other frameworks, CodeIgniter configurable items aren't contained in a single file. Instead, each class that needs configurable items will have a configuration file with the same name as the class that uses it. You will find the application configuration files in the **/app/Config** folder.

- *访问配置文件*
- *创建配置文件*
- *环境变量*
- *Environment Variables and CodeIgniter*
- *嵌套变量*
- *命名空间中的变量*
- *Configuration Classes and Environment Variables*
- *以数组的方式调用环境变量*
- *Handling Different Environments*
- *注册器*

### 访问配置文件

You can access configuration files for your classes in several different ways.

- By using the `new` keyword to create an instance:

```
// Creating new configuration object by hand
$config = new \Config\Pager();
```

- By using the `config()` function:

```
// Get shared instance with config function
$config = config('Pager');

// Access config class with namespace
$config = config( 'Config\\Pager' );

// Creating a new object with config function
$config = config('Pager', false);
```

All configuration object properties are public, so you access the settings like any other property:

```
$config = config('Pager');
// Access settings as object properties
$pageSize = $config->perPage;
```

若没有给定 namespace(命名空间)，框架会在所有可用的、已被定义的命名空间中搜寻所需的文件，就如同 **/app/Config/** 一样。

All of the configuration files that ship with CodeIgniter are namespaced with `Config`. Using this namespace in your application will provide the best performance since it knows exactly where to find the files.

我们也可以通过使用一个不同的命名空间，从而在服务器的任意位置上部署所需的配置文件。这一举措可以让我们将生产环境的服务器中的配置文件移动到一个不能通过 Web 访问的位置；而在开发环境中，将其放置在 **/app** 目录下以便访问。

### 创建配置文件

When you need a new configuration, first you create a new file at your desired location. The default file location (recommended for most cases) is **/app/Config**. The class should use the appropriate namespace, and it should extend `CodeIgniter\Config\BaseConfig` to ensure that it can receive environment-specific settings.

Define the class and fill it with public properties that represent your settings.:

```
<?php namespace Config;
```

(下页继续)

(续上页)

```
use CodeIgniter\Config\BaseConfig;

class CustomClass extends BaseConfig
{
    public $siteName  = 'My Great Site';
    public $siteEmail = 'webmaster@example.com';

}
```

### 环境变量

One of today' s best practices for application setup is to use Environment Variables. One reason for this is that Environment Variables are easy to change between deploys without changing any code. Configuration can change a lot across deploys, but code does not. For instance, multiple environments, such as the developer' s local machine and the production server, usually need different configuration values for each particular setup.

Environment Variables should also be used for anything private such as passwords, API keys, or other sensitive data.

### Environment Variables and CodeIgniter

CodeIgniter makes it simple and painless to set Environment Variables by using a "dotenv" file. The term comes from the file name, which starts with a dot before the text "env" .

CodeIgniter expects **.env** to be at the root of your project alongside the `system` and `app` directories. There is a template file distributed with CodeIgniter that' s located at the project root named **env** (Notice there' s no dot (.) at the start?). It has a large collection of variables your project might use that have been assigned empty, dummy, or default values. You can use this file as a starting place for your application by either renaming the template to **.env**, or by making a copy of it named **.env**.

---

**重要:** Make sure the **.env** file is NOT tracked by your version control system. For *git* that means adding it to **.gitignore**. Failure to do so could result in sensitive credentials being exposed to the public.

---

Settings are stored in **.env** files as a simple a collection of name/value pairs separated by an equal sign.

```
S3_BUCKET = dotenv
SECRET_KEY = super_secret_key
CI_ENVIRONMENT = development
```

When your application runs, **.env** will be loaded automatically, and the variables put into the environment. If a variable already exists in the environment, it will NOT be overwritten. The loaded Environment variables are accessed using any of the following: `getenv()`, `$_SERVER`, or `$_ENV`.

```
$s3_bucket = getenv('S3_BUCKET');
$s3_bucket = $_ENV['S3_BUCKET'];
$s3_bucket = $_SERVER['S3_BUCKET'];
```

### 嵌套变量

为了减少输入，我们也可以用将变量名包裹在 `${...}` 的形式，来重用先前定义过的变量:

```
BASE_DIR="/var/webroot/project-root"
CACHE_DIR="${BASE_DIR}/cache"
TMP_DIR="${BASE_DIR}/tmp"
```

### 命名空间中的变量

有时候，我们会遇到多个变量具有相同名字的情况。当这种情况发生时，系统将没有办法获知这个变量所对应的确切的值。我们可以通过将这些变量放入”命名空间“中，来放置这一情况的出现。

在配置文件中，点号 (.) 通常被用来表示一个变量是命名空间变量。这种变量通常是由一个独立前缀，后接一个点号 (.) 然后才是变量名称本身所组成的:

```
// 非命名空间变量
name = "George"
db=my_db

// 命名空间变量
address.city = "Berlin"
address.country = "Germany"
frontend.db = sales
backend.db = admin
BackEnd.db = admin
```

### Configuration Classes and Environment Variables

When you instantiate a configuration class, any *namespaced* environment variables are considered for merging into the configuration object’ s properties.

If the prefix of a namespaced variable exactly matches the namespace of the configuration class, then the trailing part of the setting (after the dot) is treated as a configuration property. If it matches an existing configuration property, the environment variable’ s

value will replace the corresponding value from the configuration file. If there is no match, the configuration class properties are left unchanged. In this usage, the prefix must be the full (case-sensitive) namespace of the class.

```
Config\App.CSRFProtection  = true
Config\App.CSRFCookieName = csrf_cookie
Config\App.CSPEnabled = true
```

---

**注解：** Both the namespace prefix and the property name are case-sensitive. They must exactly match the full namespace and property names as defined in the configuration class file.

---

The same holds for a *short prefix*, which is a namespace using only the lowercase version of the configuration class name. If the short prefix matches the class name, the value from **.env** replaces the configuration file value.

```
app.CSRFProtection  = true
app.CSRFCookieName = csrf_cookie
app.CSPEnabled = true
```

---

**注解：** When using the *short prefix* the property names must still exactly match the class defined name.

---

### 以数组的方式调用环境变量

从更长远的角度来看，一个命名空间环境变量也可以以数组的方式被调用。如果一个命名空间环境变量的前缀与某个配置类所匹配，那么这个变量的剩余部分，若同样包含点号，则将会被当做一个数组的引用来调用:

```
// 常规的命名空间变量
Config\SimpleConfig.name = George

// 数组化的命名空间变量
Config\SimpleConfig.address.city = "Berlin"
Config\SimpleConfig.address.country = "Germany"
```

如果这个变量是对 SimpleConfig 配置类的成员的引用，上述例子将会如下图所示:

```
$address['city'] = "Berlin";
$address['country'] = "Germany";
```

而 `$address` 属性的其他部分将不会被改动。

我们同样可以将数组属性名作为前缀来使用，当配置文件如下所示时:

```
// array namespaced variables
Config\SimpleConfig.address.city = "Berlin"
address.country = "Germany"
```

### Handling Different Environments

Configuring multiple environments is easily accomplished by using a separate **.env** file with values modified to meet that environment's needs.

The file should not contain every possible setting for every configuration class used by the application. In truth, it should include only those items that are specific to the environment or are sensitive details like passwords and API keys and other information that should not be exposed. But anything that changes between deployments is fair-game.

In each environment, place the **.env** file in the project's root folder. For most setups, this will be the same level as the `system` and `app` directories.

Do not track **.env** files with your version control system. If you do, and the repository is made public, you will have put sensitive information where everybody can find it.

### 注册器

一个配置文件可以指定任意数量的”注册器“；这里所指的注册器为其他类可能提供的额外的配置属性。这一行为通常通过在配置文件中增加一个 `registrars` 属性来实现，这一属性存有一个可选的注册器数组。:

```
protected $registrars = [
    SupportingPackageRegistrar::class
];
```

为了实现“注册器”的功能，这些类中必须声明一个与配置类同名的静态方法，而这一方法应当返回一个包含有属性配置项的关联数组。

当我们实例化了一个配置类的对象后，系统将自动循环搜索在 `$registrars` 中指定的类。对于这些类而言，当其中包含有与该配置类同名的方法时，框架将调用这一方法，并将其返回的所有属性，如同上节所述的命名空间变量一样，并入到配置项中。

配置类举例如下:

```
<?php namespace App\Config;

use CodeIgniter\Config\BaseConfig;

class MySalesConfig extends BaseConfig
{
    public $target        = 100;
    public $campaign      = "Winter Wonderland";
    protected $registrars = [
```

(下页继续)

(续上页)

```
        '\App\Models\RegionalSales';
    ];
}
```

…所关联的地区销售模型将如下所示:

```
<?php namespace App\Models;

class RegionalSales
{
    public static function MySalesConfig()
    {
        return ['target' => 45, 'actual' => 72];
    }
}
```

如上所示，当 *MySalesConfig* 被实例化后，它将以两个属性的被声明而结束，然而 *$target* 属性将会被 *RegionalSalesModel* 的注册器所覆盖，故而最终的配置属性为:

```
$target = 45;
$campaign = "Winter Wonderland";
```

### 4.2.2 CodeIgniter URL

在默认情况下，CodeIgniter 中的 URL 被设计成对搜索引擎和用户友好的样式。不同于使用传统的在动态系统中使用代词的标准“查询字符串”的方式，CodeIgniter 使用基于段的方法:

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

#### 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/method/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。此外，可以通过*URI 路由* 的方式进行重定向你的 URL 从而使得程序更加灵活。

### 移除 index.php 文件

默认情况，你的 URL 中会包含 **index.php** 文件:

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

如果你的服务器支持重写 URL ，那么通过 URL 重写，我们可以轻易的去除这个文件。在不同的服务器中，处理方式各异，故而如下我们主要展示两个最为通用的 Web 服务器。

### Apache 服务器

Apache 需要开启 *mod_rewrite* 扩展。当开启时，我们可以使用一个 `.htaccess` 文件以及一些简单的规则来实现 URL 重写。如下为这个文件的一个样例，其中使用了”否定“方法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外,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服务器配置都有效。

**注解:** 确保使用上面的规则时，排除掉那些你希望能直接访问到的资源。

### NGINX

在 NGINX 中，我们可以定义一个 location 块并用 `try_files` 导向来取得如上文中 Apache 配置一样的效果:

```
location / {
    try_files $uri $uri/ /index.php/$args;
}
```

服务器将会首先寻找符合对应 URI 的文件或目录（对于每个文件，通过根目录和别名目录来构建其完整的路径），然后再将其他的请求发送至 index.php 文件中。

### 4.2.3 辅助函数

辅助函数正如其名，是用于辅助你处理任务的。每个辅助文件都只是一个特殊分类下一系列函数的集合。例如 **URL 辅助函数**用于创建链接，**表单辅助函数**用于创建表单元素，**文本辅助函数**用于不同的文本编排方式，**Cookie 辅助函数**用于设置和获取 cookies，**文件辅助函数**用于处理文件等。



与 CodeIgniter 中其他大多数的系统部件有所不同，辅助函数不是以面向对象的方式实现的，而仅仅是简单的，程序化的函数方法。每个辅助函数都只执行一个特定的功能，并与其他的函数不产生依赖。

CodeIgniter 在默认情况下不加载辅助函数文件，因此使用它的第一步就是加载它。而当辅助函数被加载后，该函数就在你的控制器 and 视图 中全局可用。

辅助函数通常储存在 **system/Helpers** 或 **app/Helpers directory** 下。CodeIgniter 首先会在 **app/Helpers** 目录下进行查找，但是如果该目录不存在，或者指定的辅助函数文件不在当前位置。CI 就会在你的全局 *system/Helpers/* 目录下进行搜索。

#### 加载一个辅助文件

可以使用如下方法来轻易地加载辅助文件:

```
helper('name');
```

其中 **name** 是辅助文件的文件名，并不需要带有.php 的文件扩展名或者是” helper” 这部分。

举例来说，为了加载名为 **cookie_helper.php** 的 **Cookie 辅助文件 **，你需要这样做:

```
helper('cookie');
```

如果你需要加载多个辅助文件，可以通过传值一个文件名构成的数组，这样一来这些文件就都会被加载了:

```
helper(['cookie', 'date']);
```

一个辅助函数文件可以在你的控制器方法里的任意位置被加载（甚至在你的视图文件里，虽然这样做并不符合最佳实践），只要你在使用前加载了该文件就行。你也可以在

控制器的构造函数里加载辅助函数，从而在当前控制器的任何方法里都可以使用，或者在需要调用的方法里加载这个辅助函数。

---

**注解：** 辅助函数的加载方法并不会返回值，所以请不要尝试将它赋值给一个变量。请如上所示调用即可。

---

---

**注解：** URL 辅助函数文件总是会自动加载，因此你不需要手动加载它。

---

### 从非标准位置加载

辅助函数文件可以从 **app/Helpers** 目录和 **system/Helpers** 目录外被加载，只要它们所处的路径可以通过自动加载器配置文件 中的 PSR-4 这节所配置的命名空间定位。你可以为辅助函数文件用命名空间作为前缀来起名，从而使其可被定位。在命名空间所处的目录下，加载器期望该文件存在于命名空间中一个名为 `Helpers` 的子目录下。以下是一个帮助理解的例子。

举例来说，假设我们把所有博客相关的代码都放在了我们的命名空间 `Example\Blog` 中。而文件存在于我们服务器的 **/Modules/Blog/**命名空间下**。**那么我们就应该把博客模块对应的辅助文件放在**/Modules/Blog/Helpers/**命名空间里**。****blog_helper** 文件就应该位于 **/Modules/Blog/Helpers/blog_helper.php** 的位置。在控制器中我们可以使用如下指令来加载该辅助文件:

```
helper('Modules\Blog\blog');
```

---

**注解：** 通过该方式加载的文件中包含的函数并不是真正带有命名空间的。这里的命名空间只是用于简单快捷地定位文件而已。

---

### 使用一个辅助函数

当你加载完包含有你想要使用的辅助函数的文件后，就可以像调用一个标准的 PHP 函数一样调用它。

举例来说，在你的视图文件中使用 `anchor` 函数来创建一个链接，可以这样做:

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

其中” Click Here” 是链接名，而” blogs/comments” 是你希望链接到的控制器/方法名所对应的的 URI

### “继承”辅助函数

为了实现对辅助函数文件的”继承”，在你的 **app/Helpers/** 目录下创建一个和现存的辅助函数文件同名的文件。

如果你你想做的只是对已有的辅助函数文件加一些新功能——那么在里面加一两个函数或者改某个特殊的辅助函数的代码——比起用你的版本来完全替换整个辅助函数文件来说要实用的多。在这种情况下，最好只是”继承”这个辅助函数文件。

---

**注解:** “继承”这个词语在这里用起来太过于泛化了，因为辅助函数是面向过程且离散的，并不能像传统的场景一样被继承。在这种情况下，这里的”继承”的意思是，让你可以添加或替换原本的辅助文件所提供的函数。

---

举个例子，为了继承原生的 **数组辅助函数**，你可以创建一个名为 **app/Helpers/array_helper.php** 的文件并增加或重载以下函数:

```
// any_in_array() 并不是数组辅助函数，因为被定义为一个新的函数
function any_in_array($needle, $haystack)
{
        $needle = is_array($needle) ? $needle : [$needle];

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

        return FALSE;
}

// random_element() 在数组辅助函数中被定义了，所以在这里重载了原生的函数
function random_element($array)
{
        shuffle($array);
        return array_pop($array);
}
```

**helper()** 方法会扫描所有 **app/Config/Autoload.php** 里定义的 PSR-4 命名空间并同时加载所有匹配的辅助文件这一行为将使得所有模块的辅助文件都会被加载，包括所有你所创建用于该程序的对应辅助文件。加载顺序如下:

1. app/Helpers - 这里的文件总是首先被加载
2. {namespace}/Helpers - 所有的命名空间都会通过所定义的顺序遍历
3. system/Helpers - 系统辅助文件在最后加载

### 然后呢？

在内容表里你可以找到所有可用的辅助函数文件*Helpers* 。请逐一浏览它们的用途吧。

## 4.2.4 公共函数和全局常量

CodeIgniter 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。



### 公共函数

### 服务访问器函数

`cache`(*[$key]*)

**参数**

- **$key** (*string*) – 需从缓存中检索的参数名 (可选)

**返回** 缓存对象或从缓存取回的变量

**返回类型** mixed

若 $key 不存在, 则返回缓存引擎实例. 若 $key 有值存在, 则返回 $key 当前存储在缓存中的值，若值不存在则返回 null。

Examples:

```
$foo = cache('foo');
$cache = cache();
```

`env`(*$key*[, *$default=null*])

**参数**

- **$key** (*string*) – 需检索的环境变量中的参数名
- **$default** (*mixed*) – 如参数值不存在则返回默认值.

**返回** 运行环境变量, 默认值, 或者 null.

**返回类型** mixed

用于检索事前设置在环境变量中的变量值, 若无设置则返回默认值. 若没有找到健值则返回一个布尔值结果（false）.

在特定的运行环境中利用.env 文件设置环境变量非常有用, 例如数据库设置, API 健值等.

**esc**(*$data*, *$context='html'*[, *$encoding*])

**参数**

- **$data** (*string|array*) – 被输出的信息.
- **$context** (*string*) – 被输出内容的上下文. 默认值‘html’.
- **$encoding** (*string*) – 编码字符串.

**返回** 输出的数据（The escaped data）.

**返回类型** mixed

页面中包含的输出数据, 它在防止 XSS 攻击时很有用。使用 Laminas Escaper 库来处理实际的数据过滤。

若 $data 为字符串, 则简单转义并且返回。若 $data 为数组, 则遍历数组，转义 key/value 键值对中的‘value’。

有效的 context 值: html, js, css, url, attr, raw, null

**helper**(*$filename*)

**参数**

- **$filename** (*string|array*) – 加载的辅助类文件的名称，或一个包含类文件名的数组。

加载辅助类文件。

详情参照辅助函数 页.

**lang**(*$line*[, *$args*[, *$locale*]])

**参数**

- **$line** (*string*) – 检索文本的行
- **$args** (*array*) – 一组数组数据，用于替代占位符.
- **$locale** (*string*) – 使用不同的地区，而不是默认的地区设置。

检索一个基于某个别名字符串的本地特定文件。

更多详细信息请见*Localization* 页。

**model**(*$name*[, *$getShared = true*[, *&$conn = null*]])

**参数**

- **$name** (*string*) –
- **$getShared** (*boolean*) –

- **$conn** (*ConnectionInterface|null*) –

**返回** More simple way of getting model instances

**返回类型** mixed

**old**(*$key*[, *$default = null*[, *$escape = 'html'*]])

**参数**

- **$key** (*string*) – 需要使用的原有的表单提交的键。
- **$default** (*mixed*) – 如果当 $key 不存在时返回的默认值。
- **$escape** (*mixed*) – 一个*escape* 的上下文，或传值 false 来禁用该功能。

**返回** 给定的键对应的值，或设置的默认值

**返回类型** mixed

提供了一个简易的方式，在表单提交时访问 “原有的输入数据”。

示例:

```
// 在控制器中查看表单提交
if (! $model->save($user))
{
    // 'withInput' 方法意味着 " 原有的数据 " 需要被存储。
    return redirect()->back()->withInput();
}

// 视图中
<input type="email" name="email" value="<?= old('email') ?>">
// 以数组的形式
<input type="email" name="user[email]" value="<?= old('user.email')␣
↪?>">
```

**注解:** 如果你正使用:doc: *form helper </helpers/form_helper>* , 这个特性就是内置的。只有在你不使用 form helper 的时候才需要手动调用。

**session**([*$key*])

**变量 string $key** 在 session 中查找的健值名称.

**返回** $key 的值或者 null，若 $key 不存在则返回一个 session object 实例。

**返回类型** mixed

提供一个访问 session 类和检索存储值的便捷方法。更多信息详见 the *Sessions* 页.

**timer**([*$name*])

**参数**

- **$name** (*string*) – 检测点的名称.

**返回** Timer 实例

**返回类型** CodeIgniterDebugTimer

提供一个快速访问 Timer class 的便捷的方法。你可以将基准点的名称作为唯一参数传递。这将从这一点开始计时，如果这个名称的计时器已经运行，则停止计时。

示例:

```
// 获取一个 timer 实例
$timer = timer();

// 设置计时器的开始与结束点
timer('controller_loading');     // 开始计时器
. . .
timer('controller_loading');     // 停止计时器运行
```

**view**(*$name*[, *$data*[, *$options*]])

**参数**

- **$name** (*string*) – 被加载的文件名
- **$data** (*array*) – 键值对数组，在视图中能被获取。
- **$options** (*array*) – 可选的参数数组，用于传递值给渲染类.

**返回** 视图的输出.

**返回类型** string

抓取当前的 RendererInterface-compatible 类（界面渲染类），告诉它展示特定的视图。给控制器、库、路由闭包提供了一种便捷的方法。

目前，在 $options 数组里只有一个选项是可用的，saveData 指定在同一个请求中，在多次调用 view() 时数据将连续。默认情况下，在显示该单一视图文件之后，该视图的数据被丢弃。

$option 数组主要用于与第三方库整合，例如 Twig。

示例:

```
$data = ['user' => $user];

echo view('user_profile', $data);
```

详情参见 the *Views* 页。

**view_cell**(*$library*[, *$params = null*[, *$ttl = 0*[, *$cacheName = null*]]])

**参数**

- **$library** (*string*) –

- **$params** (*null*) –
- **$ttl** (*integer*) –
- **$cacheName** (*string/null*) –

**返回** View cells are used within views to insert HTML chunks that are managed by other classes.

**返回类型** string

For more details, see the *View Cells* page.

### 其他函数

**app_timezone()**

**返回** The timezone the application has been set to display dates in.

**返回类型** string

Returns the timezone the application has been set to display dates in.

**csrf_token()**

**返回** 当前 CSRF token 名称。

**返回类型** string

返回当前 CSRF token 名称。

**csrf_header()**

**returns** The name of the header for current CSRF token.

**rtype** string

The name of the header for current CSRF token.

**csrf_hash()**

**返回** 当前 CSRF hash 值.

**返回类型** string

返回当前 CSRF hash 的值.

**csrf_field()**

**返回** 带有全部请求 CSRF 信息的隐藏 input 的 HTML 字符串。

**返回类型** string

返回已插入 CSRF 信息的隐藏 input:

<input type=" hidden" name=" {csrf_token}" value=" {csrf_hash}" >

**csrf_meta()**

**返回** A string with the HTML for meta tag with all required CSRF information.

**返回类型** string

Returns a meta tag with the CSRF information already inserted:

<meta name=" {csrf_header}" content=" {csrf_hash}" >

**force_https**(*$duration = 31536000*[, *$request = null*[, *$response = null*]])

**参数**

- **$duration** (*int*) – 浏览器的秒数应该将此资源的链接转换为 HTTPS 。
- **$request** (*RequestInterface*) – 当前请求对象的实例。
- **$response** (*ResponseInterface*) – 当前响应对象的实例。

检查页面当前是否通过 HTTPS 访问，如果不是，则用户通过 HTTPS 重定向回当前 URI。将设置 HTTP 严格的传输安全标头，该命令指示现代浏览器自动将 HTTP 请求修改为 $duration 参数时间的 HTTPS 请求。

**function_usable**(*$function_name*)

**参数**

- **$function_name** (*string*) – Function to check for

**返回** TRUE if the function exists and is safe to call, FALSE otherwise.

**返回类型** bool

**is_cli**()

**返回** 如果脚本是从命令行执行的，则为 true，否则为 false。

**返回类型** bool

**is_really_writable**(*$file*)

**参数**

- **$file** (*string*) – The filename being checked.

**返回** TRUE if you can write to the file, FALSE otherwise.

**返回类型** bool

**log_message**(*$level*, *$message*[, *$context*])

**参数**

- **$level** (*string*) – 级别程度
- **$message** (*string*) – 写入日志的信息.
- **$context** (*array*) – 一个标记和值的联合数组被替换到 $message

**返回** 如果写入日志成功则为 TRUE ，如果写入日志出现问题则为 FALSE 。

**返回类型** bool

使用 **app/Config/Logger.php** 中定义的日志处理程序记录日志。

级别可为以下值: **emergency**, **alert**, **critical**, **error**, **warning**, **notice**, **info**, or **debug**.

Context 可用于替换 message 字符串中的值。详情参见 the:doc:*Logging Information <logging>* 页。

**redirect**(*string $uri*)

**参数**

- **$uri** (*string*) – 需要引导用户重定向到的页面.

返回以后 RedirectResponse 的实例以便创建重定向:

```
// 回到上一个页面
return redirect()->back();

// 跳转至具体的 URL
return redirect()->to('/admin');

// 跳转到一个命名路由或反向路由 URI
return redirect()->route('named_route');

// 在跳转中保持原有的输入值，使得它们可以被 `old()` 函数调用。
return redirect()->back()->withInput();

// 显示一个消息
return redirect()->back()->with('foo', 'message');
```

当将 URI 传给这个函数时。它将会被作为一个反向路由请求，而不是一个完整的 URI ，就像使用 redirect()->route() 一样:

```
// 跳转到一个命名路由或反向路由 URI
return redirect('named_route');
```

**remove_invisible_characters**(*$str*[, *$urlEncoded = TRUE*])

**参数**

- **$str** (*string*) – 输入字符串
- **$urlEncoded** (*bool*) – 是否移除 URL 编码字符

**返回** 已过滤的字符串

**返回类型** string

这个函数防止在 ASCII 字符之间插入空字符 (NULL)，例如 Java\0script。

示例:

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

**route_to**(*$method*[, *...$params*])

**参数**

- **$method** (*string*) – 命名路由别名，或匹配 controller/method 名称。
- **$params** (*mixed*) – 一个或更多参数被传递到路由中匹配。

以指定的路由别名或 controller::method 组合为依据生成一个相对 URI 。如果提供参数，将执行参数。

详情参见 the *URI 路由* 页。

**service**(*$name*[, *...$params*])

**参数**

- **$name** (*string*) – 加载的服务名称
- **$params** (*mixed*) – 一个或多个参数传递到服务方法。

**返回** 指定的服务类的实例。

**返回类型** mixed

提供简易访问任何在系统中定义的服务，详见 the *Services* 。这将总是返回类的共享实例，因此不管在单个请求中调用多少次，都只会创建一个类实例。

示例:

```
$logger = service('logger');
$renderer = service('renderer', APPPATH.'views/');
```

**single_service**(*$name*[, *...$params*])

**参数**

- **$name** (*string*) – 加载的服务名称
- **$params** (*mixed*) – 一个或多个参数传递到服务方法。

**返回** 指定的服务类的实例。

**返回类型** mixed

**等同于前面所描述的 service() 函数，除了所有调用该函数将返回一个类的新实例。** service 返回的是相同的实例。

**slash_item**(*$item*)

**参数**

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

**返回** The configuration item or NULL if the item doesn’t exist

**返回类型** string|null

Fetch a config file item with slash appended (if not empty)

**stringify_attributes**(*$attributes*[, *$js*])

**参数**

- **$attributes** (*mixed*) – 字符串, 键值对数组, 或者对象
- **$js** (*boolean*) – TRUE 若值不需要引用 (Javascript 风格)

**返回** 字符串包含键值对属性, 逗号分隔

**返回类型** string

辅助函数用于转换字符串, 数组, 或者字符串的对象属性。

### 全局常量

以下的常量在你的应用中的任何地方有效。

### 核心常量

**constant APPPATH**
app 目录的路径。

**constant ROOTPATH**
项目根目录，APPPATH 目录的上层目录。

**constant SYSTEMPATH**
system 目录的路径。

**constant FCPATH**
保存的前端控制器目录的路径。

**constant WRITEPATH**
writable 目录的路径。

### 时间常量

**constant SECOND**
等于 1.

**constant MINUTE**
等于 60.

**constant HOUR**
等于 3600.

**constant DAY**
等于 86400.

**constant WEEK**
等于 604800.

**constant MONTH**
等于 2592000.

**constant YEAR**
等于 31536000.

**constant DECADE**
等于 315360000.

### 4.2.5 记录日志信息



你可以通过 `log_message()` 方法将信息记录在本地日志文件中，并且必须在第一个参数中指定错误的”级别”，来表明这个信息的类型（debug，error 等）。第二个参数就是信息本身:

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

总共有八种不同的事件报错级别，与 RFC 5424 中所定义的错误级别一一对应，它们是:

| 级别 | 描述 |
|---|---|
| debug | 详细的 debug 信息。 |
| info | 你的应用中的一些有意义的事件，例如用户登录，记录 SQL 语句等。 |
| notice | 你的应用中的一些正常但明显有价值的事件。 |
| warning | 出现了异常，但不是错误，例如使用了被废弃的 APIs ，某个 API 的调用异常，或其他不期望出现的，但不是错误的情况。 |
| error | 运行时错误，不需要立即被处理但通常需要被记录或者监控。 |
| critical | 危险情况，例如某个程序组件不可用，或出现未被捕获的异常等。 |
| alert | 告警，必须采取行动来修复，例如整个网站宕机或数据库无法访问等。 |
| emergency | 系统不可用。 |

日志系统不提供警告系统管理员或网站管理者的方法，只是单纯的记录信息。对于诸多更为危险的错误级别，日志就会被异常调度器自动抛出，如上所述。

### 配置

你可以修改 `/app/Config/Logger.php` 配置文件来修改哪些级别的事件会被实际记录，以及为不同的事件等级分配不同的日志记录器等。

配置文件中的 `threshold` （报错阈值）决定了从哪个级别开始的事件将会在整个应用中记录下来。如果应用中有任何低于报错阈值的事件记录被记录时，这些请求将会被忽略。最为简单的使用阈值的方法就是将其设为你希望记录的报错等级的最低值。举例来说，如果你想记录 warning 信息，而不是 information 信息，就需要将报错阈值设为 5 。所有报错等级低于 5 的日志记录请求（包括运行时错误，系统错误等）将会被记录，而 info, notice 和 debug 级别的错误就会被忽略:

```
public $threshold = 5;
```

关于报错级别和对应的阈值的列表列举在配置文件中以供参阅。

你可以通过给报错阈值赋值一个包含报错等级数字的数组，来选择特定的报错级别:

```
// 只记录 debug 和 info 类型的报错
public $threshold = [5, 8];
```

### 使用多个日志调度器

日志系统支持同时使用多种调度器来处理日志记录。每一种调度器可以独立地设置用于特定的错误等级，并忽略其他的。现状而言，我们默认安装了两种调度器以供使用:

- **文件调度器**是默认的调度器，它将会每天在本地创建一个独立的日志文件，同时这也是较为被推荐的日志记录方式。
- **ChromeLogger 调度器**如果你在 Chrome 浏览器上安装了 ChromeLogger 扩展 ，你可以使用这种调度器将日志输出到 Chrome 的控制台窗口中。

调度器配置于主配置文件中的 $handlers 属性中，这一属性的格式为一个包含一组调度器和它们对应的配置的数组。每个调度器被定义数组的键，格式为完整命名空间格式的类名，而对应的值就是一个数组。每个调度器配置块中都会有一个通用的属性: handle ，对应着该调度器将要记录的报错级别的 名字

```
public $handlers = [

        //--------------------------------------------------------------------
        // 文件调度器
        //--------------------------------------------------------------------

        'CodeIgniter\Log\Handlers\FileHandler' => [

                'handles' => ['critical', 'alert', 'emergency', 'debug',
'error', 'info', 'notice', 'warning'],
        ]
];
```

### 根据上下文修改记录信息

我们经常会根据上下文来修改记录信息的某些细节。比如说，可能会记录用户 ID，IP 地址，当前的 POST 变量等。你可以通过在信息中使用通配符来实现。每个通配符必须被大括号（{}）包裹起来。在第三个参数中，你需要提供一个包含有通配符名，与其对应值的数组。这些内容将会插入到记录信息字符串中:

```
// 生成一条例如这样的信息: 用户 123 登录系统，登录 IP 为 127.0.0.1
$info = [
        'id' => $user->id,
        'ip_address' => $this->request->ip_address()
];

log_message('info', 'User {id} logged into the system from {ip_address}',
 $info);
```

如果你想记录一条异常或一个错误，你可以使用”exception”作为键，对应的值就是这条异常或错误本身。这样一来这个异常或错误对象包含的错误信息，文件名和对应行号就会生成一条字符串。你需要在记录信息中中提供 exception 通配符:

```
try
{
        ... 一些能抛出异常的代码
}
catch (\Exception $e)
{
```

(下页继续)

(续上页)

```
        log_message('error', '[ERROR] {exception}', ['exception' => $e]);
}
```

以下是几个核心通配符，它们将会在请求页面时自动被替换成指定的数据:

| 通配符 | 对应的替换数据 |
| --- | --- |
| {post_vars} | $_POST 变量 |
| {get_vars} | $_GET 变量 |
| {session_vars} | $_SESSION 变量 |
| {env} | 当前环境名，例如 development |
| {file} | 生成日志的文件名 |
| {line} | {file} 中生成日志的指定行号 |
| {env:foo} | 在 $_ENV 数组中 foo 这个 key 对应的 value |

#### 使用第三方日志器

你可以使用任何自己喜欢的日志器，只要它继承了 `Psr\Log\LoggerInterface` 并符合 PSR3 规范。这意味着你可以使用任何符合 PSR-3 规范的日志器，或者造一个自己的。

你需要将第三方日志器放入 `/app/Config/Autoload.php` 配置文件中或者通过某个自动加载器，比如 Composer，来保证第三方日志器在系统中可被找到。接下来你需要修改 `/app/Config/Services.php` ，将 `logger` 的别名设置为新的日志器的类名。

现在开始，对 `log_message()` 的所有调用都会使用你自定义的日志器进行日志记录。

#### LoggerAware Trait（代码复用）

当你需要将你的日志库以框架不感知的形式调用时，你可以使用实现了 `setLogger()` 方法的 `CodeIgniter\Log\LoggerAwareTrait` 。从而当在不同框架环境下使用日志库时，你的日志器依旧可如同预期一般运行，只要它能找到一个符合 PSR3 的日志器。

### 4.2.6 错误处理

CodeIgniter 通过 SPL collection 和一些框架内自定义异常来生成系统错误报告。错误处理的行为取决于你部署环境的设置，当一个错误或异常被抛出时，只要应用不是在 `production` 环境下运行，就会默认展示出详细的错误报告。在这种情况下，应为用户显示一个更为通用的信息来保证最佳的用户体验。

### 使用异常处理

本节为新手程序员或没有多少异常处理使用经验的开发人员做一个简单概述。

异常处理是在异常被”抛出”的时候产生的事件。它会暂停当前脚本的执行，并将捕获到的异常发送到错误处理程序后显示适当的错误提示页

```
throw new \Exception("Some message goes here");
```

如果你调用了一个可能会产生异常的方法，你可以使用 `try/catch block` 去捕获异常

```
try {
        $user = $userModel->find($id);
}
catch (\Exception $e)
{
        die($e->getMessage());
}
```

如果 `$userModel` 抛出了一个异常，那么它就会被捕获，并执行 catch 代码块内的语句。在这个样例中，脚本终止并输出了 `UserModel` 定义的错误信息。

在这个例子中，我们可以捕捉任意类型的异常。如果我们仅仅想要监视特定类型的异常，比如 UnknownFileException，我们就可以把它在 catch 参数中指定出来。这样一来，其它异常和非监视类型子类的异常都会被传递给错误处理程序

```
catch (\CodeIgniter\UnknownFileException $e)
{
        // do something here...
}
```

这便于你自己进行错误处理或是在脚本结束前做好清理工作。如果你希望错误处理程序正常运行，可以在 catch 语句块中再抛出一个新的异常

```
catch (\CodeIgniter\UnknownFileException $e)
{
        // do something here...

        throw new \RuntimeException($e->getMessage(), $e->getCode(), $e);
}
```

### 配置

默认情况下，CodeIgniter 将在 `development` 和 `testing` 环境中展示所有的错误，而在 `production` 环境中不展示任何错误。你可以在主 `index.php` 文件的顶部找到环境配置部分来更改此设置。

---

**重要：** 如果发生错误，禁用错误报告将不会阻止日志的写入。

---

### 记录异常

By default, all Exceptions other than 404 - Page Not Found exceptions are logged. This can be turned on and off by setting the **$log** value of `Config\Exceptions`:

```
class Exceptions
{
    public $log = true;
}
```

To ignore logging on other status codes, you can set the status code to ignore in the same file:

```
class Exceptions
{
    public $ignoredCodes = [ 404 ];
}
```

---

**注解：** It is possible that logging still will not happen for exceptions if your current Log settings are not set up to log **critical** errors, which all exceptions are logged as.

---

### 自定义异常

下列是可用的自定义异常:

### PageNotFoundException

这是用来声明 404 ，页面无法找到的错误。当异常被抛出时，系统将显示后面的错误模板 /application/views/errors/html/error_404.php。你应为你的站点自定义所有错误视图。如果在 Config/Routes.php 中，你指定了 404 的重写规则，那么它将代替标准的 404 页来被调用

```
if (! $page = $pageModel->find($id))
{
        throw␣
↪\CodeIgniter\Exceptions\PageNotFoundException::forPageNotFound();
}
```

你可以通过异常传递消息，它将在 404 页默认消息位置被展示。

### ConfigException

当配置文件中的值无效或 class 类不是正确类型等情况时，请使用此异常

```
throw new \CodeIgniter\Exceptions\ConfigException();
```

它将 HTTP 状态码置为 500，退出状态码被置为 3.

### UnknownFileException

在文件没有被找到时，请使用此异常

```
throw new \CodeIgniter\UnknownFileException();
```

它将 HTTP 状态码置为 500，退出状态码被置为 4.

### UnknownClassException

当一个类没有被找到时，请使用此异常

```
throw new \CodeIgniter\UnknownClassException($className);
```

它将 HTTP 状态码置为 500，退出状态码被置为 5.

### UnknownMethodException

当一个类的方法不存在时，请使用此异常

```
throw new \CodeIgniter\UnknownMethodException();
```

它将 HTTP 状态码置为 500，退出状态码被置为 6.

#### UserInputException

当用户的输入无效时，请使用此异常

```
throw new \CodeIgniter\UserInputException();
```

它将 HTTP 状态码置为 500，退出状态码被置为 7.

#### DatabaseException

当产生如连接不能建立或连接临时丢失的数据库错误时，请使用此异常

```
throw new \CodeIgniter\DatabaseException();
```

它将 HTTP 状态码置为 500，退出状态码被置为 8.

#### RedirectException

This exception is a special case allowing for overriding of all other response routing and forcing a redirect to a specific route or URL:

```
throw new \CodeIgniter\Router\Exceptions\RedirectException($route);
```

`$route` may be a named route, relative URI, or a complete URL. You can also supply a redirect code to use instead of the default (`302`, "temporary redirect"):

```
throw new \CodeIgniter\Router\Exceptions\RedirectException($route, 301);
```

### 4.2.7 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，文件将会被当前的缓存引擎所配置的方式缓存起来（译者注：例如文件缓存，memcache 缓存等）。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。

**注解：** 基准标记没有缓存，所以当缓存启用时，仍然可以查看页面加载速度。

#### 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:

```
$this->cachePage($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:** 如果你修改了可能影响页面输出的配置，你需要手动删除你的缓存文件。

**注解:** 在写入缓存文件之前，必须通过编辑 **app/Config/Cache.php** 文件来设置缓存引擎。

#### 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。

### 4.2.8 AJAX 请求

`IncomingRequest::isAJAX()` 方法使用了 `X-Requested-With` 请求头来确定一个请求是否是 XHR(XML Http Request) 或者是一个正常的请求。然后最新的 JavaScript 实现 (例如 fetch 方法) 中不再随着请求发送这个头，因此使用 `IncomingRequest::isAJAX()` 就不那么可靠了，因为没有这个头，该方法就不能识别一个请求是否是一个 XHR。

为了解决这个问题，最有效的解决方式（至今）就是人为定义一个请求头，迫使这个请求信息发送的服务器从而识别这个请求是否是一个 XHR。

以下就是如何迫使在 Fetch API 和其他 JavaScript 库中发送 `X-Requested-With` 请求头。

#### Fetch API

```
fetch(url, {
    method: "get",
    headers: {
      "Content-Type": "application/json",
```

(下页继续)

(续上页)

```
        "X-Requested-With": "XMLHttpRequest"
    }
});
```

#### jQuery

对于类似 jQuery 之类的库来说，不需要额外发送这个头，因为根据 官方文档 ，对于所有 `$.ajax()` 请求来说，这都是一个标准头。但是如果你还是不想担风险并强制发送这个头，就像下面这样做吧:

```
$.ajax({
    url: "your url",
    headers: {'X-Requested-With': 'XMLHttpRequest'}
});
```

#### VueJS

在 VueJS 中你只需要在 `created` 函数中增加以下代码，只要你在这类请求时使用 Axios:

```
axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';
```

#### React

```
axios.get("your url", {headers: {'Content-Type': 'application/json'}})
```

### 4.2.9 代码模块

CodeIgniter 支持代码模块化组合，以便于你构建可重用的代码。模块通常来说是以一个特定主题为中心而构建的，并可被认为是在大型的程序中的一系列微型程序。我们支持框架中所有标准的文件类型，例如控制器，模型，视图，配置文件，辅助函数，语言文件等。模块可能包含着或多或少的你所需要的以上这些类型中。

### 命名空间

CodeIgniter 所使用的模块功能的核心组件来自于*与 PSR4 相适应的自动加载* 。虽然所有的代码都可以使用 PSR4 的自动加载和命名空间，最主要的充分使用模块优势的方式还是为你的代码加上命名空间，并将其添加到 **app/Config/Autoload.php** 中，在 psr4 这节中。

举例而言，比如我们需要维护一个在应用间复用的简单的博客模块。我们可能会创建一个带有公司名（比如 acme）的文件夹来保存所有的模块。我们可能会将其置于我们的 **application** 目录旁边，在主项目目录下:

```
/acme        // 新的模块目录
/application
/public
/system
/tests
/writable
```

打开 **app/Config/Autoload.php** 并将 **Acme** 命名空间加入到 psr4 数组成员中:

```
$psr4 = [
    'Config'         => APPPATH . 'Config',
    APP_NAMESPACE    => APPPATH,                // 自定义命名空间
    'App'            => APPPATH,                // 确保筛选器等组件可找到,
    'Acme'           => ROOTPATH.'acme'
];
```

当我们设置完以上流程后，就可以通过 Acme 命名空间来访问 **acme** 目录下的文件夹内容。这已经完成了 80% 的模块工作所需要的内容，所以你可以通过熟悉命名空间来适应这种使用方式。这样多种文件类型将会被自动扫描并在整个定义的命名空间中使用——这也是使用模块的关键。

在模块中的常见目录结构和主程序目录类似:

```
/acme
    /Blog
        /Config
        /Controllers
        /Database
            /Migrations
            /Seeds
        /Helpers
        /Language
            /en
        /Libraries
        /Models
        /Views
```

当然了，不强制使用这样的目录结构，你也可以自定义目录结构来更好地符合你的模块要求，去掉那些你不需要的目录并增加一些新的目录，例如实体（Entites），接口（Interfaces），仓库（Repository）等。

### 自动发现

很多情况下，你需要指名你所需要包含进来的文件的命名空间全称，但是 CodeIgniter 可以通过配置自动发现的文件类型，来将模块更方便地整合进你的项目中:

- *Events*
- *Registrars*
- *Route files*
- *Services*

这些是在 **app/Config/Modules.php** 文件中配置的。

自动发现系统通过扫描所有在 **Config/Autoload.php** 中定义的 PSR4 类型的命名空间来实现对于目录/文件的识别。

To make auto-discovery work for our **Blog** namespace, we need to make one small adjustment. **Acme** needs to be changed to **Acme\Blog** because each "module" within the namespace needs to be fully defined. Once your module folder path is defined, the discovery process would look for discoverable items on that path and should, for example, find the routes file at **/acme/Blog/Config/Routes.php**.

### 开启/关闭自动发现

你可以开启或关闭所有的系统中的自动发现，通过 **$enabled** 类变量。False 的话就会关闭所有的自动发现，优化性能，但却会让你的模块可用性相对下降。

### 明确目录项目

通过 **$activeExplorers** 选项，你可以明确哪些项目是自动发现的。如果这个项目不存在，就不会对它进行自动发现流程，而数组中的其他成员仍旧会被自动发现。

### 自动发现与 Composer

通过 Composer 安装的包将会默认被自动发现。这只需要 Composer 识别所需要加载的命名空间是符合 PSR4 规范的命名空间，PSR0 类型的命名空间将不会被发现。

如果在定位文件时，你不想扫描所有 Composer 已识别的的目录，可以通过编辑 `Config\Modules.php` 中的 `$discoverInComposer` 变量来关闭这一功能:

```
public $discoverInComposer = false;
```

### 处理文件

这节将会详细介绍每种文件类型（控制器，视图，语言文件等）以及在模块中如果使用它们。其中的某些信息在用户手册中将会更为详细地描述，不过在这里重新介绍一下以便了解全局的情况。

### 路由

默认情况下，路由 将会在模块内部自动扫描，而这一特性可在 **Modules** 配置文件中被关闭，如上所述。

---

**注解:** 由于在当前域内包含了路由文件，`$routes` 实例已经被定义了，所以当你尝试重新定义类的时候可能会引起错误。

---

### 控制器

在主 **app/Controller** 目录下定义的控制器不会自动被 URI 路由自动调用，所以需要在路由文件内部手动声明:

```
// Routes.php
$routes->get('blog', 'Acme\Blog\Controllers\Blog::index');
```

为了减少不必要的输入，**group** 路由特性（译者注：分组路由 </incoming/routing# 分组路由 > ）是一个不错的选择:

```
$routes->group('blog', ['namespace' => 'Acme\Blog\Controllers'],␣
↪function($routes)
{
```

（下页继续）

(续上页)

```
    $routes->get('/', 'Blog::index');
});
```

### 配置文件

No special change is needed when working with configuration files. These are still namespaced classes and loaded with the `new` command:

```
$config = new \Acme\Blog\Config\Blog();
```

Config files are automatically discovered whenever using the **config()** function that is always available.

### 迁移

迁移文件将通过定义的命名空间自动发现。所有命名空间里找到的迁移每次都会被自动运行。

### 种子

种子文件可在 CLI 或其他种子文件里使用，只要提供了完整的命名空间名。如果通过 CLI 调用，就需要提供双反斜杠定义的类名格式 (\):

```
> php public/index.php migrations seed␣
↪Acme\\Blog\\Database\\Seeds\\TestPostSeeder
```

### 辅助函数

当使用 `helper()` 方法时，辅助函数将会通过定义的命名空间自动定位。只要它存在于 **Helpers** 命名空间目录下:

```
helper('blog');
```

### 语言文件

当使用 `lang()` 方法时，语言文件是通过定义的命名空间来自动定位的。只要这个文件是遵循主程序目录一样的目录结构来放置的。

#### 库

库总是通过完全命名空间化的类名进行实例化，所以不需要额外的操作:

```
$lib = new \Acme\Blog\Libraries\BlogLib();
```

#### 模型

模型总是通过完全命名空间化的类名进行实例化，所以不需要额外的操作:

```
$model = new \Acme\Blog\Models\PostModel();
```

#### 视图

视图文件可通过视图 文档中所述的类命名空间进行加载:

```
echo view('Acme\Blog\Views\index');
```

### 4.2.10 管理多个应用

默认情况下，我们假设你只是用 CodeIgniter 来管理一个应用，并将该应用在 **application** 目录下进行构建。然而也存在这样的可能性：多个应用共享一个 CodeIgniter 的安装目录，甚至开发者会将 application 目录进行重命名或移动位置。

#### 重命名或迁移应用程序目录

如果你想重命名你的应用文件夹或者移动 it to a different location on your server, other than your project root, open your main **app/Config/Paths.php** and set a *full server path* in the `$appDirectory` variable (at about line 38):

```
public $appDirectory = '/path/to/your/application';
```

You will need to modify two additional files in your project root, so that they can find the `Paths` configuration file:

- `/spark` runs command line apps; the path is specified on or about line 36:

  ```
  require 'app/Config/Paths.php';
  // ^^^ Change this if you move your application folder
  ```

- `/public/index.php` is the front controller for your webapp; the config path is specified on or about line 16:

```
$pathsPath = FCPATH . '../app/Config/Paths.php';
// ^^^ Change this if you move your application folder
```

### 单个 CodeIgniter 对应运行多个应用

如果你想要让多个不同的应用来共享一次 CodeIgniter 的安装文件，只需要将你的应用目录下的所有目录都移动到他们对应的子目录中即可。

举例而言，加入你想要创建两个应用程序，命名为”foo”和”bar”，你可以将你的应用目录排列如下:

```
/foo
    /app
    /public
    /tests
    /writable
/bar
    /app
    /public
    /tests
    /writable
/codeigniter
    /system
    /docs
```

This would have two apps, “foo” and “bar”, both having standard application directories and a `public` folder, and sharing a common codeigniter framework.

The `index.php` inside each application would refer to its own configuration, `../app/Config/Paths.php`, and the `$systemDirectory` variable inside each of those would be set to refer to the shared common “system” folder.

If either of the applications had a command-line component, then you would also modify `spark` inside each application’s project folder, as directed above.

## 4.2.11 处理多环境

开发者常常希望根据是生产环境还是开发环境能够区分不同的定制行为，例如，如果在开发环境的程序当中输出详细的错误信息这样做对开发者来说是非常有帮助的，但是这样做的话在生产环境中会造成一些安全问题。In development environments, you might want additional tools loaded that you don’t in production environments, etc.

### 环境常量

CodeIgniter 默认使用 $_SERVER[‘CI_ENVIRONMENT’] 的值作为 ENVIRONMENT 常量，否则默认就是‘production’。这样能够根据不同服务器安装环境定制不同的环境依赖。

### .env

最简单的方式是在你的 .env 配置文件里设置。

```
CI_ENVIRONMENT = development
```

### Apache

如果要获取 $_SERVER[ ‘CI_ENVIRONMENT’ ] 的值可以在.htaccess 的文件里，或者可以在 Apache 的配置文件里使用 SetEnv 命令进行设置

```
SetEnv CI_ENVIRONMENT development
```

### nginx

在 nginx 下，为了能够在 $_SERVER 里显示环境变量的值你必须通过 fastcgi_params 来传递。这样允许它在虚拟主机上工作来替代使用 env 去为整个服务器设置它，即使在专用服务器上运行良好。你可以修改该服务器的配置为:

```
server {
    server_name localhost;
    include     conf/defaults.conf;
    root        /var/www;

    location    ~* "\.php$" {
        fastcgi_param CI_ENVIRONMENT "production";
        include conf/fastcgi-php.conf;
    }
}
```

可选方法适用于 nginx 和其它服务器，或者你也可以完全移除这部分逻辑，并根据服务器的 IP 地址设置常量 (实例)。

使用这个常量，除了会影响到一些基本的框架行为外 (见下一章节)，在开发过程中你还可以使用常量来区分当前运行的是什么环境。

### 引导文件

CodeIgnite 要求在 **APPPATH/Config/Boot** 下放置一个与环境名称匹配的 PHP 脚本文件。这些文件包含你想为你的环境所做的符合要求的任何定制，无论是更新对错误显示的设置，还是加载附加开发工具，或者是添加其他东西。系统会自动加载这些文件。在新的版本中为你创建好了以下文件:

```
* development.php
* production.php
* testing.php
```

### 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

将 ENVIRONMENT 常量值设置为‘development’，这将导致所有发生的 PHP 错误在客户端请求页面时显示在浏览器上。相反，如果将常量设置为‘production’将禁用所有错误输出。在生产环境禁用错误输出是 良好的安全实践。

### 配置文件

另外，CodeIgnite 还可以根据不同的环境自动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。

章节 5

# 请求处理

## 5.1 控制器和路由

控制器用于处理收到的请求。

### 5.1.1 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。

### 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 \CodeIgniter\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 \CodeIgniter\Controller {

}
```

这是 **无效**的:

```
<?php
class blog extends \CodeIgniter\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 \CodeIgniter\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 \CodeIgniter\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，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:

```
$routes->setDefaultController('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。

想要了解更多信息，请参阅 ./source/general/routing.rst 部分文档。

### 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)
{
        $method = 'process_'.$method;
        if (method_exists($this, $method))
        {
                return $this->$method(...$params);
        }
        show_404();
}
```

### 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:

```
protected 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/
```

### 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 的第一段必须指定目录，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
```

---

每个子目录包含一个默认控制器，将在 URL 只包含子目录的时候被调用。默认控制器在 *application/Config/Routes.php* 中定义。

你也可以使用 CodeIgniter 的 ./source/general/routing.rst 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(…$params);
```

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 \CodeIgniter\Controller
{
        public function __construct(...$params)
        {
                parent::__construct(...$params);

                // 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。

### 包含属性

你创建的每一个 controller 都应该继承 `CodeIgniter\Controller` 类。这个类提供了适合所有控制器的几个属性。

### Request 对象

`$this->request` 作为应用程序的主要属性 ./source/libraries/request.rst 是可以一直被使用的类属性。

### Response 对象

`$this->response` 作为应用程序的主要属性 ./source/libraries/response.rst 是可以一直被使用的类属性。

### Logger 对象

`$this->logger` 类实例 ./source/general/logging.rst 是可以一直被使用的类属性。

### forceHTTPS

一种强制通过 HTTPS 访问方法的便捷方法，在所有控制器中都是可用的:

```
if (! $this->request->isSecure())
{
        $this->forceHTTPS();
}
```

默认情况下，在支持 HTTP 严格传输安全报头的现代浏览器中，此调用应强制浏览器将非 HTTPS 调用转换为一年的 HTTPS 调用。你可以通过将持续时间（以秒为单位）作为第一个参数来修改。

```
if (! $this->request->isSecure())
{
        $this->forceHTTPS(31536000);     // one year
}
```

---

**注解:** 你可以使用更多全局变量和函数 ./source/general/common_functions.rst ，包括年、月等等。

---

### 辅助函数

你可以定义一个辅助文件数组作为类属性。每当控制器被加载时，这些辅助文件将自动加载到内存中，这样就可以在控制器的任何地方使用它们的方法。:

```
class MyController extends \CodeIgniter\Controller
{
        protected $helpers = ['url', 'form'];
}
```

### 验证 $_POST 数据

控制器还提供了一个简单方便的方法来验证 $_POST 数据，将一组规则作为第一个参数进行验证，如果验证不通过，可以选择显示一组自定义错误消息。你可以通过 **$this->request** 这个用法获取 POST 数据。Validation Library docs 是有关规则和消息数组的格式以及可用规则的详细信息。

```
public function updateUser(int $userID)
{
    if (! $this->validate([
        'email' => "required|is_unique[users.email,id,{$userID}]",
        'name' => 'required|alpha_numeric_spaces'
    ]))
    {
        return view('users/update', [
            'errors' => $this->errors
        ]);
    }

    // do something here if successful...
}
```

如果你觉得在配置文件中保存规则更简单，你可以通过在 Config\Validation.php 中定义代替 $rules 数组

```
public function updateUser(int $userID)
{
    if (! $this->validate('userRules'))
    {
        return view('users/update', [
            'errors' => $this->errors
        ]);
    }

    // do something here if successful...
}
```

**注解：** 验证也可以在模型中自动处理。你可以在任何地方处理，你会发现控制器中的一些情况比模型简单，反之亦然。

**就这样了！**

OK，总的来说，这就是关于控制器的所有内容了。

### 5.1.2 URI 路由

- 默认控制器
- 默认方法
- 连字符 (-) 转换
- 仅使用定义路由
- 404 重载

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，调用一个不同的类或方法，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 **app/config/Routes.php** 文件中。你将会在其中看到，该文件创建了一个 RouteCollection 类的实例，这一实例允许你定义自己的路由规则。路由中可使用通配符和正则表达式。

路由通常将 URI 置于左侧，而将控制器和对应的方法以及任何可能存在的，并需要传递给控制器的参数映射在右侧。控制器与其方法的列出形式就像你调用一个类的静态方法一样，用双冒号来分隔一个充分命名空间化形式的类与其方法，例如 `Users::list`。如果这个方法需要被传递参数，这些参数应被以正斜杠分割的形式在方法名后列出，如:

```
// 调用 $Users->list()
Users::list
// 调用 $Users->list(1, 23)
Users::list/1/23
```

### 通配符

一个典型的路由规则看上去就像这样:

```
$routes->add('product/(:num)', 'App\Catalog::productLookup');
```

在一个路由中，第一个参数包含需要被匹配到的 URI，而第二个参数包含着这个路由应被定位到的目标位置。在上述例子中，当单词”product”在 URL 的第一个分段中被发现，同时在第二个分段中出现了一个数字，那么 App\Catalog 类与 productLookup 方法就会调用。

通配符是一系列简单的正则表达式类型的字符串。在路由处理过程中，通配符会被正则表达式的值所取代，故而这些通配符主要是为了可读性而设计的。

当在你的路由处理过程中，可使用如下通配符:

- **(:any)** 将会从当前位置开始到 URI 结束，匹配任何字符。这一通配符可能会包括多个 URI 分段。
- **(:segment)** 将会匹配除了斜杠 (/) 以外的任何字符，从而将匹配结果限制在一个单独的分段中。
- **(:num)** 将会匹配任何整数。
- **(:alpha)** 将会匹配任何英文字母字符。
- **(:alphanum)** 将会匹配任何英文字母或整数，或者是这两者的组合。
- **(:hash)** 与 **:segment** 相同，但可用于方便地查看那个路由正在使用哈希 id(参照*Model* )。

---

**注解:** 因为 **{locale}** 是一个系统保留关键词，用于*localization* ，所以不可用于通配符或路由的其他部分。

---

### 示例

以下是一些路由示例:

```
$routes->add('journals', 'App\Blogs');
```

一个第一个分段包含有单词”journals”的 URL 将会被映射于 App\Blogs 类，这个类的默认方法通常将会是 index():

```
$routes->add('blog/joe', 'Blogs::users/34');
```

一个包含有“blog/joe”的分段的 URL 将会被映射于 \Blogs 类和 users 方法，而其 ID 参数将会被置为 34:

```
$routes->add('product/(:any)', 'Catalog::productLookup');
```

一个第一个分段为”product”，并且第二个分段是任意字符的 URl，将会被映射于 \Catalog 类的 productLookup 方法:

```
$routes->add('product/(:num)', 'Catalog::productLookupByID/$1';
```

一个第一个分段为” product”，并且第二个分段是数字的 URl，将会被映射于 \Catalog 类的 productLookup 方法，并将这一数字传递为方法的一个变量参数。

---

**重要：** 尽管 add() 方法是相当方便的，我们还是推荐使用基于 HTTP 动词的路由结构，如下所述，并且这也更为安全。

---

与此同时，这样也会带来轻微的性能提升，因为只有匹配当前请求方法的路由会被保存，从而在搜索路由时会减少搜索次数。

### 自定义通配符

你也可以在路由文件中创建自己的通配符从而实现用户体验和可读性的定制需求。

你可以使用 addPlaceholder 方法来增加新的通配符。第一个参数是一个被用来作为通配符的字符串，第二个是该通配符应当被替换成的正则表达式。这一方法操作需要在你增加路由之前被调用:

```
$routes->addPlaceholder('uuid', '[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-
↪9a-f]{4}-[0-9a-f]{12}');
$routes->add('users/(:uuid)', 'Users::show/$1');
```

### 正则表达式

如果你更倾向于使用正则表达式的话，也可以用它来定义路由规则。允许任何有效的正则表达式，例如反向引用。

---

**重要：** Note: 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:

```
$routes->add('products/([a-z]+)/(\d+)', 'Products::show/$1/id_$2');
```

---

上例中，一个类似于 products/shirts/123 这样的 URL 将会重定向到 Products 控制器的 show 方法。并且将原来的第一个第二个 URI 分段作为参数传递给它。通过正则表达式，你也可以捕获一个带有斜杠（’ /’）的分段，而通常来说斜杠是用于多个分段时间的分隔符。

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，你希望在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:

```
$routes->add('login/(.+)', 'Auth::login/$1');
```

对于诸位虽然不熟悉正则表达式而又想了解更多关于正则表达式的，‘regular-expressions.info <http://www.regular-expressions.info/>‘_ 可能是一个不错的起点。

**重要：** 注意：你也可以在你的路由规则中混用通配符和正则表达式。

### 闭包

你可以使用一个匿名函数，或者闭包，作为路由的映射目标位置。这一函数将会在用户访问指定 URI 时执行。以上操作在执行小功能，或只是显示一个简单的视图时，是相当方便的:

```
$routes->add('feed', function()
{
    $rss = new RSSFeeder();
    return $rss->feed('general');
});
```

### 映射多个路由

虽然 add() 方法非常简单易用，但是调用 map() 方法来同时处理多个路由通常更为方便。你可以通过定义一个路由的数组，并将其作为 map() 方法的第一个参数的批量处理的方式，来取代每次都要用 add() 方法来添加所需要路由:

```
$routes = [];
$routes['product/(:num)']       = 'Catalog::productLookupById';
$routes['product/(:alphanum)'] = 'Catalog::productLookupByName';

$collection->map($routes);
```

### 重定向路由

任何存在了足够长时间的网站都肯定存在移动过的页面。你可以通过 addRedirect() 方法来重定向需要跳转到其他路由的路由规则。第一个参数是原有的路由的 URI 规则，第二个参数是新的 URI，或者是一个命名路由的名称。第三个参数是随着重定向一起发送的状态码，默认值 302 ，这也是通常情况下用的比较多的，意味着暂时的重定向:

```
$routes->add('users/profile', 'Users::profile', ['as' => 'profile']);

// 重定向至命名路由
$routes->addRedirect('users/about', 'profile');
// 重定向至 URI
$routes->addRedirect('users/about', 'users/profile');
```

当页面加载时，若匹配到重定向路由，则用户将会在加载原有控制器之前被重定向到新页面。

### 分组路由

你可以使用 group() 将你的路由分组并设定一个通用的名字。分组名将作为 URI 的一个分段，用于组内所有定义的路由之前。这一方式可以帮助你在定义一大组有相同前缀的路由时，减少额外的打字输入，例如设置一个管理分组时:

```
$routes->group('admin', function($routes)
{
        $routes->add('users', 'Admin\Users::index');
        $routes->add('blog', 'Admin\Blog::index');
});
```

如上，'users' 和 'blog' 这些 URI 就会加上"amdin"的前缀，从而处理例如 /admin/users 和 /admin/blog 的 URI。如果你需要的话，同样也可以嵌套分组以便管理:

```
$routes->group('admin', function($routes)
{
        $routes->group('users', function($routes)
        {
                $routes->add('list', 'Admin\Users::list');
        });

});
```

这将用于处理例如 admin/users/list 的 URI。

如果你需要为一个分组指定指定选项，类似 *namespace* ，请在回调前使用:

```
$routes->group('api', ['namespace' => 'App\API\v1'], function($routes)
{
        $routes->resource('users');
});
```

这将能够使得如同 /api/users/ 一样 resource 的路由映射于 App\API\v1\Users 控制器上。你也可以对一组路由使用一个特定的 过滤器 。过滤器总是会在控制器的调用前或调用后运行，这一操作在认证或 api 日志时格外有用:

```
$routes->group('api', ['filter' => 'api-auth'], function($routes)
{
    $routes->resource('users');
});
```

控制器的值必须与定义在 app/Config/Filters.php 中的一系列别名中的至少一个所匹配。

### 环境约束

你可以设置一组在特定环境下运行的路由。这方便了你创建一组只有开发者在本地环境中可使用，而在测试和生产环境不可见的工具。以上操作可通过 environment() 方

法来实现。第一个参数是环境名。在这个闭包中的定义的所有路由，仅在当前环境下可访问:

```
$routes->environment('development', function($routes)
{
        $routes->add('builder', 'Tools\Builder::index');
});
```

### 反向路由

反向路由允许你定义一个链接与它需要查找的当前路由所需要使用的控制器和方法以及参数。这可以不需要改变程序代码而定义路由规则。通常用于视图内部以创建链接地址。

举例来说，如果你需要一个跳转到图片相册的路由，你可以使用 route_to() 辅助函数以获取当前应该使用的路由。第一个参数是完整的控制器类名与方法名以双英文冒号(::)区分，就像你在写一条原生的路由规则的格式一样。其他所有需要传递给这个路由的参数都将在后面被传递:

```
// 该路由定义为:
$routes->add('users/(:id)/gallery(:any)',
↪'App\Controllers\Galleries::showUserGallery/$1/$2');

// 生成对应连接到用户 ID1: 5, 图片 ID: 12 的指定 URL
// 生成:/users/15/gallery/12
<a href="<?= route_to('App\Controllers\Galleries::showUserGallery', 15,␣
↪12) ?>"> 查看相册</a>
```

### 使用命名路由

你可以为路由命名，从而提高系统健壮性（鲁棒性），这一操作可通过给一个路由命名从而在后面调用来实现。即使路由定义改变了，所有在系统中通过 route_to 创建的的连接将仍旧可用并且不需要进行任何变动。命名一个路由，通过与路由名一起传递 as 选项来实现:

```
// 路由定义为:
$routes->add('users/(:id)/gallery(:any)', 'Galleries::showUserGallery/$1/
↪$2', ['as' => 'user_gallery');

// 生成对应连接到用户 ID1: 5, 图片 ID: 12 的指定 URL
    // 生成:/users/15/gallery/12
<a href="<?= route_to('user_gallery', 15, 12) ?>">View Gallery</a>
```

这同样使得视图更具有可读性。

### 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。你可以使用所有标准的 HTTP 动词（GET、PUT、POST、DELETE 等），每个动词都拥有自己对应的方法供你使用:

```
$routes->get('products', 'Product::feature');
$routes->post('products', 'Product::feature');
$routes->put('products/(:num)', 'Product::feature');
$routes->delete('products/(:num)', 'Product::feature');
```

你可以指定一个路由可以匹配多个动词，将其传递 `match()` 方法作为一个数组:

```
$routes->match(['get', 'put'], 'products', 'Product::feature');
```

### 命令行专用的路由

你可以使用 `cli()` 方法来创建命令行专用，浏览器不可访问的路由。这一方法中创建 crojobs(定时任务) 或命令行工具时相当有效。而基于 HTTP 动词的路由同样对于命令行也是不可访问的，除了通过 `any()` 方法创建的路由之外:

```
$routes->cli('migrate', 'App\Database::migrate');
```

### 全局选项

所有用于创建路由的方法（例如 add, get, post, resource 等）都可以调用一个选项数组来修改已生成的路由或限制它们的规则。而这一数组 `$options` 就是这些方法的最后一个参数:

```
$routes->add('from', 'to', $options);
$routes->get('from', 'to', $options);
$routes->post('from', 'to', $options);
$routes->put('from', 'to', $options);
$routes->head('from', 'to', $options);
$routes->options('from', 'to', $options);
$routes->delete('from', 'to', $options);
$routes->patch('from', 'to', $options);
$routes->match(['get', 'put'], 'from', 'to', $options);
$routes->resource('photos', $options);
$routes->map($array, $options);
$routes->group('name', $options, function());
```

### 应用过滤器

你可以通过指定一个过滤器在控制器调用前或调用后运行的方式来改变指定路由的行为，这一操作通常在鉴权或 API 记录日志时非常有用:

```
$routes->add('admin',' AdminController::index', ['filter' => 'admin-auth
↪']);
```

过滤器的值必须至少匹配 app/Config/Filters.php 中的一个别名。你也可以指定过滤器的 before() 和 after() 方法的参数:

```
$routes->add('users/delete/(:segment)', 'AdminController::index', [
↪'filter' => 'admin-auth:dual,noreturn']);
```

浏览 Controller filters 来获取更多有关设置筛选过滤器的信息。

### 指定命名空间

尽管默认的命名空间会在生成的控制器前自动附加（如下），你也可以通过 namespace 选项来指定一个别的命名空间在选项数组中。选项值应该与你想指定的命名空间一致:

```
// 路由指定至 \Admin\Users::index()
    $routes->add('admin/users', 'Users::index', ['namespace' => 'Admin
↪']);
```

新的命名空间仅应用于创建一个单独路由的方法调用中，例如 get, post 等。对于创建多个路由的方法，新的命名空间将会被附在所有被这个方法锁生成的路由之前，例如在 group() 中，所有的路由都是在闭包中生成的。

### 限制域名

你可以通过给选项数组的”hostname”选项传一个域名作为值的形式来限制一组路由只在你的应用的特定域名或子域名下生效:

```
$collection->get('from', 'to', ['hostname' => 'accounts.example.com']);
```

这个例子仅允许当前访问的路由在域名为”accounts.example.com”时生效，而在其主域名”example.com”下无法生效。

### 限制子域名

当 subdomain 选项开启时，系统将会限制路由仅在此子域名生效。只有在访问该子域名时系统才会匹配这组路由规则:

```
// 限制子域名为 media.example.com
$routes->add('from', 'to', ['subdomain' => 'media']);
```

你可以通过设置该选项值为星号 (*) 的方式来对所有子域名生效。当你访问的 URL 不匹配任何子域名时，这项路由将不会被匹配到:

```
// 限制所有子域名访问
$routes->add('from', 'to', ['subdomain' => '*']);
```

**重要：** 系统不是完美无缺的，所以在部署生产环境前需要在特定的子域名下进行测试。大多数域名都没有问题，但在一些边缘情况下，特别是某些域名本身中就含有点号 (.)，而这个点号又不是拿来区分前缀或者后缀时，就可能会出错。

### Offsetting the Matched Parameters

你可以向后推移在路由中匹配到的参数的位置，通过在 `offset` 选项中传递任何数字值，该值指名了推移匹配的 URI 分段的数量。

这将会为开发 API 带来好处，当 URI 第一个分段是版本号时，同样可以用于第一个参数是一个语言标识（例如 en，fr 等，译者注）：

```
$routes->get('users/(:num)', 'users/show/$1', ['offset' => 1]);

// 创建：
$routes['users/(:num)'] = 'users/show/$2';
```

(译者注：实质就是将匹配的位置向后推移，由于第一个分段的位置可能会被其他参数占用，所以通配符的位置需要后移，例如/en/users/(:num)，这里/en/是第一个分段，不需要作为路由使用，所以 (:num) 实际上通过 offset 后移到了 $2 的位置。)

### 路由配置选项

路由集合类提供了多个可影响到所有路由的选项配置，并可被修改以符合程序要求，这些选项可在 */app/Config/Routes/php*‘ 文件的顶部被更改。

### 默认命名空间

当匹配到了一个需要路由的控制器，路由将会为该控制器增加一个默认的命名空间。默认设置下，这个命名空间的值为空，从而每个每个路由都需要完全对应到的带有命名空间的控制器类名:

```
$routes->setDefaultNamespace('');

// 控制器为 \Users
$routes->add('users', 'Users::index');

// 控制器为 \Admin\Users
$routes->add('users', 'Admin\Users::index');
```

如果你的控制器不是严格遵从命名空间的话，就没有更改的必要。如果你为控制器指定了命名空间，就可以通过更改默认命名空间的值来减少打字输入:

```
$routes->setDefaultNamespace('App');

// 控制器为 \App\Users
$routes->add('users', 'Users::index');

// 控制器为 \App\Admin\Users
$routes->add('users', 'Admin\Users::index');
```

### 默认控制器

当用户直接访问你的站点的根路径时（例如 example.com），所调用的控制器将会由 setDefaultController() 方法所设置的参数决定，除非有一个路由是显式声明过（默认控制器）。这一方法的默认值是 Home，对应的控制器是 /app/Controllers/Home.php

```
// example.com 对应的路由是 app/Controllers/Welcome.php
$routes->setDefaultController('Welcome');
```

默认控制器同样也在找不到对应的路由规则，URI 对应到控制器的对应目录下的情况下被用到。例如有个用户访问了 example.com/admin ，如果有个控制器被命名为 /app/Controllers/admin/Home.php ，那么就被调用到。

### 默认方法

与默认控制器的设置类似，用于设置设置默认方法。其应用场景是，找到了 URI 对应的控制器，但是 URI 分段对应不上控制器的方法时。默认值是 index

```
$routes->setDefaultMethod('listAll');
```

在这个例子中，当用户访问 example.com/products 时，Products 控制器存在，从而执行 Products::listAll() 方法。

### 连字符 (-) 转换

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:

```
$routes->setTranslateURIDashes(true);
```

#### 仅使用定义路由

当指定的 URI 映射不到定义的路由时，系统将会将 URI 映射到如上所述的控制器和方法。你可以通过设置 setAutoRoute() 选项为 false 的方式来关闭这一自动映射，并限制系统仅使用你定义的路由:

```
$routes->setAutoRoute(false);
```

#### 404 重载

如果当前 URI 匹配不到对应的页面，系统将输出一个通用的 404 视图。你可以通过使用 set404Override() 方法，定义一个操作来改变以上行为。这一方法的参数可以是一个合法的类/方法的组合，就如同你在任何路由或者闭包中定义的一样:

```
// 将执行 App\Errors 类的 show404 方法
$routes->set404Override('App\Errors::show404');

// 将会输出一个自定义的视图
$routes->set404Override(function()
{
    echo view('my_errors/not_found.html');
});
```

### 5.1.3 控制器过滤器



控制器过滤器可以是在控制器运行前或者运行后执行相应的操作，与事件 不同，你可以非常简单、方便的选择在应用程序的哪个 URI 上应用过滤器。过滤器可以修改传入的请求，也可以对响应做出修改，从而具有很大的灵活性和功能性。我们可以使用过滤器执行一些共同的常见的任务，例如:

- 对于传入的请求执行 CSRF 验证
- 根据用户角色控制显示的功能
- 在某些功能或接口执行请求速率限制
- 显示“停机维护”页面
- 自动执行内容协商操作（例如设置 Accept-Language 值）
- 更多

### 创建过滤器

过滤器类必须实现 CodeIgniter\Filters\FilterInterface 接口。过滤器类必须有 2 个方法：before() 和 after()，它们会在控制器运行之前和之后执行。如果你的业务只需要其中一个方法，那另外的方法留空即可，不可以删除。一个标准的过滤器类模板如下:

```
<?php namespace App\Filters;

use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
use CodeIgniter\Filters\FilterInterface;

class MyFilter implements FilterInterface
{
    public function before(RequestInterface $request)
    {
        // Do something here
    }

    //--------------------------------------------------------------------
↪---

    public function after(RequestInterface $request, ResponseInterface
↪$response)
    {
        // Do something here
    }
}
```

### 前置过滤器

任何过滤器，你都可以返回 $request 对象并且可以对当前的请求进行更改替换，这些更改在后续的控制器执行时，仍然有效。

因为是前置过滤器，它会在控制器被执行前触发，所以你有时会希望做一些验证操作，不执行后续的控制器，例如登录验证。那么你可以通过返回不是请求对象的任何形式

来做到这一点。通常是执行重定向。例如以下的示例:

```
public function before(RequestInterface $request)
{
    $auth = service('auth');

    if (! $auth->isLoggedIn())
    {
        return redirect('login');
    }
}
```

如果返回了 Response 对象，那么 Response 对象会发送到客户端，并且程序会停止运行。这对实现 API 速率限制很有作用，详细可以参考 **app/Filters/Throttle.php** 相关示例。

### 后置过滤器

后置过滤器与前置过滤器几乎一样，不同的是后置过滤器只返回 $response 对象。并且，你无法停止程序的运行。你只能对 $response 对象做一些修改，比如为了确保客户端可以正常识别而设置某些安全选项，或者使用缓存输出，甚至可以使用错别字过滤器过滤最终的输出内容。

### 配置过滤器

创建完过滤器后，你需要在 app/Config/Filters.php 配置它的运行时机。该文件包含了 4 个属性，可以精确控制过滤器的运行时机。

#### $aliases

$aliases 数组可以将一个简单的名称与一个或多个完整类的路径进行绑定关联，这些完整的类就是需要运行的过滤器:

```
public $aliases = [
    'csrf' => \CodeIgniter\Filters\CSRF::class
];
```

别名是强制性的，如果你尝试使用完整的类名，系统会触发一个错误。以别名方式定义，可以很容易的切换实现类。例如当你需要替换其他过滤器时，只需要更改别名对应的类即可。

当然，你也可以将多个过滤器绑定到一个别名中，这样可以使复杂的过滤器组变得简单:

```
public $aliases = [
    'apiPrep' => [
```

(下页继续)

（续上页）

```
        \App\Filters\Negotiate::class,
        \App\Filters\ApiAuth::class
    ]
];
```

你可以在 $aliases 中定义多个别名以满足系统需求。

### $globals

这部分允许你定义应用程序中每个请求需要经过的过滤器。请一定要注意过滤器的数量，因为所有的请求都将经过这些过滤器，过多会导致影响性能。可以在 before 和 after 中添加别名来指定过滤器:

```
public $globals = [
        'before' => [
                'csrf'
        ],
        'after'  => []
];
```

有时候你希望对绝大多数请求都使用过滤器处理，但个别请求需要单独处理时，这样的情况很常见。一个常见的场景，你需要在 CSRF 预防过滤器中排除一些请求，例如来自第三方的请求或者特定的 URI 地址，其他请求则必须经过 CSRF 验证。那么，我们可以通过 except 来实现，可以定义一个或多个排除的 URI 地址:

```
public $globals = [
        'before' => [
                'csrf' => ['except' => 'api/*']
        ],
        'after'  => []
];
```

可以设置任意完整的 URI，也可以使用正则表达式，或者像本示例一样，设置星号 * 通配符的形式来设置。这样以 api/ 开头的所有请求都将不受 CSRF 过滤器的保护。但该应用程序的其他请求不受影响。如果你需要指定多个 URI，可以使用数组的形式即可，具体可以参考示例:

```
public $globals = [
        'before' => [
                'csrf' => ['except' => ['foo/*', 'bar/*']]
        ],
        'after'  => []
];
```

#### $methods

你可以将过滤器应用于请求的某些方法，例如 POST、GET、PUT 等，在数组中使用全部小写的形式指定过滤器名称，与 `$globals` 或 `$filters` 属性设置目的不同，这些过滤器全部都是前置过滤器，也就是说都在控制器运行前执行:

```
public $methods = [
    'post' => ['foo', 'bar'],
    'get'  => ['baz']
]
```

除标准的 HTTP 方法外，还支持两种特殊的方法：' cli' 和 'ajax'。它们是所有的 'cli' 命令行运行的请求和 AJAX 请求。

---

**注解:** AJAX 请求的界定在 `X-Requested-With` 标志，在某些情况下，`X-Requested-With` 不会通过 JavaScript 的 XHR 请求发送到后端，从而导致过滤器无法执行。如何避免此类问题，请参照文档的*AJAX 请求* 章节。

---

#### $filters

这个属性是过滤器别名数组，每个别名可以定义指定 URI 的前置或后置过滤器:

```
public filters = [
    'foo' => ['before' => ['admin/*'], 'after' => ['users/*']],
    'bar' => ['before' => ['api/*', 'admin/*']]
];
```

#### 默认提供的过滤器

CodeIgniter4 默认绑定了三个过滤器：Honeypot、Security 和 DebugToolbar。

### 5.1.4 HTTP 消息

消息类为 HTTP 消息中的请求和响应的实现提供了一个通用的接口，包括消息体，协议版本，处理消息头的工具和一些进行内容协商的方法。

该类是请求类 和响应类 共有的父类。因此某些方法，例如内容协商方法等，可能只适用于请求和响应，而对于其他方法不适用，但是我们将其聚合在该类中从而使得处理头的方法可以定义在相同的位置。

#### 什么是内容协商

内容协商的核心机制，其实只是 HTTP 实现的一个简单的部分，即允许单个资源适用于不止一种内容类型，从而允许客户端选择适合自己的数据类型。

一个典型的案例就是，该浏览器无法播放 PNG 格式的文件，并且只能请求 GIF 或者 JPEG 格式的图片文件。而当资源服务器接收到该请求时，它将会检查可用的文件类型是否能满足客户端所请求的，并选择本身所支持的格式中，最为适合的图片格式，在本例中就可能会返回一个 JPEG 的图片文件。

同样的协商方式会在以下四种数据类型中提现:

- **媒体/文档类型** - 可以是图片的类型格式，或者是 HTML、XML 或 JSON.
- **字符集 Character Set** - 该文档所属的字符集，通常是 UTF-8
- **文档编码 Document Encoding** - 通常是返回结果的压缩类型（译者注：例如 gzip）
- **文档语言 Document Language** - 对于支持多语言的站点，有助于决定返回哪种语言格式

### 类的参考文档

`CodeIgniter\HTTP\Message`

`body()`

**返回** 当前的消息体

**返回类型** string

返回当前消息的实体部分，如果实体不存在或者已经被发送过，返回 null:

```
echo $message->body();
```

`setBody(`[*$str*]`)`

**参数**

- **$str** (*string*) – 消息体对应的字符串.

**返回** 该消息实例，用于链式调用

**返回类型**

CodeIgniter\HTTP\Message 实例.

设置当前请求的实体内容

`populateHeaders()`

**返回** void

扫描并处理 SERVER 数据中找到的请求头内容，并将其存储以供下次使用。该方法用于请求类 *Class* 从而使得当前的请求头内容可被使用.

这里所指的请求头实际上是所有以 `HTTP_` 开头的 SERVER 数据，例如 `HTTP_POST` 。每个消息都会从被从标准的大小写格式转换为首字母大写并以横线 (-) 连接的格式。并移除了开头的 `HTTP_` 部分，故而 `HTTP_ACCEPT_LANGUAGE` 变成了 `Accept-Language` 。

getHeaders()

**返回** 一个包括了所有能确定的头部的数组.

**返回类型** array

返回所有能确定或者是先前设定过的头

getHeader([*$name*[, *$filter = null*]])

**参数**

- **$name** (*string*) – 你想要获取对应的值的头的名字
- **$filter** (*int*) – 所需要使用的过滤器类型。可供使用的过滤器见表 过滤器 。

**返回** 当前头的值。如果该头有多个值，就会以数组的形式返回

**返回类型** string|array|null

使你可以获取单个消息头的当前值。$name 对应的是大小写敏感的头名。由于在上述例子中已经对头进行了内部转换，你可以通过任何大小写方式格式来传值:

```
// 这些都等同：
$message->getHeader('HOST');
$message->getHeader('Host');
$message->getHeader('host');
```

如果该头有多个值，就会以数组的形式返回. 你可以使用 headerLine() 方法来将这些数据转换为字符串的形式来返回:

```
echo $message->getHeader('Accept-Language');

// 输出如下：
[
        'en',
        'en-US'
]
```

你可以通过将过滤器的值作为第二个参数传递给该函数:

```
$message->getHeader('Document-URI', FILTER_SANITIZE_URL);
```

headerLine(*$name*)

**参数**

- **$name** (*string*) – 需要获取的头的名字.

**返回** 头所对应的值（字符串形式）

**返回类型** string

将该头对应的值以字符串形式返回。该方法使得你可以在该头对应多个值时，将头对应的值轻松地以字符串形式返回。值以逗号分隔形式:

```
echo $message->headerLine('Accept-Language');

// 输出:
en, en-US
```

**setHeader**([*$name*[, *$value*]])

**参数**

- **$name** (*string*) – 需要设置值的头的名字
- **$value** (*mixed*) – 需要设置的值

**返回** 当前消息实例

**返回类型** CodeIgniter\HTTP\Message

为单个头赋值。$name 是该头所对应的大小写敏感的命名。如果该头部当前不存在就会被创建。$value 可以是字符串或者一个字符串数组:

```
$message->setHeader('Host', 'codeigniter.com');
```

**removeHeader**([*$name*])

**参数**

- **$name** (*string*) – 需要移除的头的名字.

**返回** 当前消息实例

**返回类型** CodeIgniter\HTTP\Message

从消息中移除指定头. $name 是该头所对应的大小写敏感的命名:

```
$message->remove('Host');
```

**appendHeader**(*[$name[, $value]]*)

**参数**

- **$name** (*string*) – 需要修改的头的名字
- **$value** (*mixed*) – 需要为该头增加的值

**返回** 当前消息实例

**返回类型** CodeIgniter\HTTP\Message

为一个现存的头增加值。该头的值不可以是单个字符串，必须是一个数组。如果是单个字符串的话会抛出一个 LogicException 异常

```
$message->appendHeader('Accept-Language', 'en-US; q=0.8');
```

**protocolVersion**()

**返回** 当前 HTTP 协议版本

**返回类型** string

返回当前消息对应的 HTTP 协议版本，如果没有设定过的话就会返回 `null`，可选值为 `1.0` 和 `1.1` 。

`setProtocolVersion`(*$version*)

**参数**

- **$version** (*string*) – HTTP 协议版本

**返回** 当前消息实例

**返回类型** CodeIgniter\HTTP\Message

为当前消息所使用的 HTTP 协议设定版本。可赋值为 `1.0` 或 `1.1`:

```
$message->setProtocolVersion('1.1');
```

`negotiateMedia`(*$supported*[, *$strictMatch=false*])

**参数**

- **$supported** (*array*) – 系统所支持的媒体类型构成的数组
- **$strictMatch** (*bool*) – 是否需要严格匹配

**返回** 对于所请求的媒体格式，返回程序支持的媒体类型

**返回类型** string

用于处理 `Accept` 请求头并将其与应用程序所支持的媒体类型进行对比来给出最合适的类型。本方法会返回一个合适的媒体类型，第一个参数是应用程序所支持的类型，用于和客户端所请求的类型进行比对:

```
$supported = [
        'image/png',
        'image/jpg',
        'image/gif'
];
$imageType = $message->negotiateMedia($supported);
```

`$supported` 数组里成员的顺序应该以程序优先返回的顺序进行定义，其中第一个成员应该是应用程序所期待的返回类型，其余降序排列。如果和请求的类型匹配不上，就默认返回数组里的第一个成员。

根据 RFC，协商匹配可以选择以返回一个默认值（就如该方法所做的那样），或者是返回一个空字符串。如果你希望进行严格匹配并返回一个空字符串的话，请为第二个参数传值 `true`

```
// 如果匹配不到就返回一个空字符串
$imageType = $message->negotiateMedia($supported, true);
```

匹配流程实际上同时考虑到了请求类型的优先级和在 RFC 中的明确性。这就意味着请求头的值越明确，所对应的优先级就越高，（除非通过 q 的值来修改）更多细节请阅读 appropriate section of the RFC

**negotiateCharset**(*$supported*)

**参数**

- **$supported** (*array*) – 系统所支持的字符集构成的数组

**返回** 对于所请求的字符集类型，所能匹配到的最优先的字符集

**返回类型** string

和 negotiateMedia() 方法一样，只是用于匹配 Accept-Charset 请求头:

```
$supported = [
        'utf-8',
        'iso-8895-9'
];
$charset = $message->negotiateCharset($supported);
```

匹配不到的情况下，返回默认的 utf-8 字符集。

**negotiateEncoding**(*$supported*)

**参数**

- **$supported** (*array*) – 系统所支持的字符编码构成的数组

**返回** 对于所请求的字符编码，所能匹配到的最优先的字符编码

**返回类型** string

与上述两个方法类似，用于匹配 Accept-Encoding 请求头；无法匹配时返回 $supported 数组的第一个元素:

```
$supported = [
        'gzip',
        'compress'
];
$encoding = $message->negotiateEncoding($supported);
```

**negotiateLanguage**(*$supported*)

**参数**

- **$supported** (*array*) – 系统所支持的语言构成的数组

**返回** 对于所请求的语言，所能匹配到的最优先的语言

**返回类型** string

与上述三个个方法类似，用于匹配 Accept-Language 请求头；无法匹配时返回 $supported 数组的第一个元素:

```
$supported = [
        'en',
        'fr',
        'x-pig-latin'
];
$language = $message->negotiateLanguage($supported);
```

关于语言标记的更多信息，请参阅 RFC 1766 。

### 5.1.5 Request 类

请求类是 HTTP 请求的面向对象表现形式。这意味着它可以用于传入请求，例如来自浏览器的请求，以及将请求从应用程序发到到第三方应用的传出请求。

这个类提供了它们需要的共同的功能，但是这两种情况都有自定义的类，它们继承请求类，然后添加特定的功能。

从 传入请求类和*CURL 请求类* 了解更多信息。

#### 类参考

**CodeIgniter\HTTP\IncomingRequest**

**getIPAddress()**

**返回** 可以检测到的用户 IP 地址，否则为 NULL ，如果 IP 地址无效，则返回 0.0.0.0

**返回类型** string

返回当前用户的 IP 地址。如果 IP 地址无效，返回 '0.0.0.0'

```
echo $request->getIPAddress();
```

---

**重要:** 此方法会根据 App->proxy_ips 的配置，来返回 HTTP_X_FORWARDED_FOR、HTTP_CLIENT_IP、HTTP_X_CLIENT_IP 或 HTTP_X_CLUSTER_CLIENT_IP 。

---

**validIP**(*$ip*[, *$which* = ''])

**参数**

- **$ip** (*string*) – IP 地址
- **$which** (*string*) – IP 协议 ( 'ipv4' 或 'ipv6' )

**返回** IP 有效返回 true，否则返回 false

**返回类型** bool

传入一个 IP 地址，根据 IP 是否有效返回 true 或 false

**注解：** $request->getIPAddress() 自动检测 IP 地址是否有效

```
if ( ! $request->validIP($ip))
{
    echo 'Not Valid';
}
else
{
    echo 'Valid';
}
```

第二个参数可选，可以为‘ipv4’或‘ipv6’。默认这两种格式会全部检查。

**method**([*$upper = FALSE*])

**参数**

- **$upper** (*bool*) – 以大写还是小写返回方法名，TRUE 表示大写

**返回** HTTP 请求方法

**返回类型** string

返回 $_SERVER['REQUEST_METHOD']，并且转换字母到指定大写或小写

```
echo $request->method(TRUE); // Outputs: POST
echo $request->method(FALSE); // Outputs: post
echo $request->method(); // Outputs: post
```

**getServer**([*$index = null*[, *$filter = null*[, *$flags = null*]]])

**参数**

- **$index** (*mixed*) – 要过滤的变量
- **$filter** (*int*) – 要过滤的类型，过滤类型列表 见此.
- **$flags** (*int*) – 过滤器 ID. 完整列表 见此.

**返回** $_SERVER 值，如果不存在则返回 NULL

**返回类型** mixed

该方法与 IncomingRequest 类中的 post()，get() 和 cookie() 方法相同。只是它只获取 getServer 数据 ($_SERVER)

```
$request->getServer('some_data');
```

要返回多个 $_SERVER 值的数组，请将所有键作为数组传递。

```
$require->getServer(array('SERVER_PROTOCOL', 'REQUEST_URI'));
```

### 5.1.6 IncomingRequest 类

IncomingRequest 类提供了一个客户端（比如浏览器）HTTP 请求的面向对象封装。基于它可以访问所有 Request 和 Message 中的方法，以及以下列出的方法。



#### 获得请求

如果当前控制器继承了 CodeIgniter\Controller，则一个 Request 类的实例已被初始化并可作为属性被使用:

```
class UserController extends CodeIgniter\Controller
{
        public function index()
        {
                if ($this->request->isAJAX())
                {
                        . . .
                }
        }
}
```

如果在控制器外使用 Request 对象，可以通过*Services class* 获得实例:

```
$request = \Config\Services::request();
```

推荐将 Request 对象作为一个依赖注入到当前类中并保存为一个属性:

```
use CodeIgniter\HTTP\RequestInterface;

class SomeClass
{
        protected $request;

        public function __construct(RequestInterface $request)
        {
                $this->request = $request;
        }
}

$someClass = new SomeClass(\Config\Services::request());
```

### 判断请求类型

请求有多种来源，包含使用 AJAX 发起和使用 CLI 发起的。可通过 `isAJAX()` and `isCLI()` 来检测:

```
// Check for AJAX request.
if ($request->isAJAX())
{
        . . .
}

// Check for CLI Request
if ($request->isCLI())
{
        . . .
}
```

你可以检测请求的 HTTP 类型:

```
// Returns 'post'
$method = $request->getMethod();
```

该方法默认返回类型是小写的字符串（比如‘get’，‘post’等等），你可以通过传递 `true` 参数来获得大写的返回结果:

```
// Returns 'GET'
$method = $request->getMethod(true);
```

还可以通过 `isSecure()` 方法检测请求是否是 HTTPS:

```
if (! $request->isSecure())
{
```

(下页继续)

(续上页)

```
        force_https();
}
```

### 数据读取

你可以通过 Request 对象读取 $_SERVER, $_GET, $_POST, $_ENV, $_SESSION 内的信息。因为输入数据不会自动过滤，只会返回请求时的原始数据。而使用这些方法去替代直接获取数据的（比如 $_POST[ 'something' ]）主要优点是当参数不存在时会返回 null ，而且你还能做数据过滤。这可以使你很方便的直接使用数据而不需要先去判断某个参数是否存在。换句话说，一般情况下你以前会这么做:

```
$something = isset($_POST['foo']) ? $_POST['foo'] : NULL;
```

而使用 CodeIgniter 的内建方法你可以很简单的做到同样的事:

```
$something = $request->getVar('foo');
```

因为 `getVar()` 方法从 $_REQUEST 获得数据，所以使用它可以获得 $_GET, $POST, $_COOKIE 内的数据。虽然这很方便，但是你有时也需要使用一些特定的方法，比如:

- `$request->getGet()`
- `$request->getPost()`
- `$request->getServer()`
- `$request->getCookie()`

另外，还有一些实用的方法可以同时获取 $_GET 或者 $_POST 的数据，因为有获取顺序的问题，我们提供了以下方法:

- `$request->getPostGet()` - 先 $_POST, 后 $_GET
- `$request->getGetPost()` - 先 $_GET, 后 $_POST

**获取 JSON 数据**

你可以使用 `getJSON()` 去获取 php://input 传递的 JSON 格式的数据。

---

**注解:** 因为无法检测来源数据是否具有有效的 JSON 格式，所以只有当你确认数据来源格式是 JSON 后才可使用。

---

```
$json = $request->getJSON();
```

默认情况下，这会返回一个 JSON 数据对象。如果你需要一个数据，请传递 `true` 作为第一个参数。

该方法的第二和第三个参数则分别对应 json_decode 方法的 `depth` 和 `options` 参数.

**获取原始数据（获取 Method 为 PUT, PATCH, DELETE 传递的数据）**

最后，你可以通过 getRawInput() 去获取 php://input 传递的原始数据。

$data = $request->getRawInput();

这会返回数据并转换为数组。比如:

```
var_dump($request->getRawInput());

[
        'Param1' => 'Value1',
        'Param2' => 'Value2'
]
```

### 数据过滤

为了保证应用程序的安全，必须过滤所有输入的数据。你可以传递过滤类型到方法的最后一个参数里。会调用系统方法 filter_var() 去过滤。具体过滤类型可以参考 PHP 手册里的列表 valid filter types.

过滤一个 POST 变量可以这么做:

```
$email = $request->getVar('email', FILTER_SANITIZE_EMAIL);
```

以上提到的方法中除了 getJSON() 和 getRawInput() ，都支持给最后一个参数传递类型来实现过滤。

### 获取数据头

你可以通过 getHeaders() 方法获得请求的数据头，该方法会以数组形式返回所有的数据头信息，数据的键值为数据头名称，值则为一个 CodeIgniter\HTTP\Header 的实例:

```
var_dump($request->getHeaders());

[
        'Host' => CodeIgniter\HTTP\Header,
        'Cache-Control' => CodeIgniter\HTTP\Header,
        'Accept' => CodeIgniter\HTTP\Header,
]
```

如果你只是想获得某个头的信息，你可以将数据头名称作为参数传递给 getHeader() 方法。数据头名称无视大小写，如果存在则返回指定头信息。如果不存在则返回 null

```
// 以下这些效果一样
$host = $request->getHeader('host');
$host = $request->getHeader('Host');
$host = $request->getHeader('HOST');
```

你可以使用 hasHeader() 去判断请求头是否存在:

```
if ($request->hasHeader('DNT'))
{
        // Don't track something...
}
```

如果你需要某个头的值并在一行字符串内输出，可以使用 getHeaderLine() 方法:

```
// Accept-Encoding: gzip, deflate, sdch
echo 'Accept-Encoding: '.$request->getHeaderLine('accept-encoding');
```

如果你需要完整头信息，输出包括全部名称和值的字符串，可以使用如下方法做转换:

```
echo (string)$header;
```

### 请求地址

你可以通过访问 $request->uri 属性获取代表当前访问信息的 doc:*URI <uri>* 对象。通过以下方法获取当前请求的完整访问地址:

```
$uri = (string)$request->uri;
```

该对象赋予了你访问全部请求信息的能力:

```
$uri = $request->uri;

echo $uri->getScheme();          // http
echo $uri->getAuthority();       // snoopy:password@example.com:88
echo $uri->getUserInfo();        // snoopy:password
echo $uri->getHost();            // example.com
echo $uri->getPort();            // 88
echo $uri->getPath();            // /path/to/page
echo $uri->getQuery();           // foo=bar&bar=baz
echo $uri->getSegments();        // ['path', 'to', 'page']
echo $uri->getSegment(1);        // 'path'
echo $uri->getTotalSegments();   // 3
```

### 上传文件

所有上传文件的信息可以通过 $request->getFiles() 方法获得，该方法会返回一个*FileCollection* 实例。这会有助于减少处理文件上传的工作量，以及使用最佳方案去降低安全风险。

```
$files = $request->getFiles();

// Grab the file by name given in HTML form
```

(下页继续)

(续上页)

```
if ($files->hasFile('uploadedFile')
{
        $file = $files->getFile('uploadedfile');

        // Generate a new secure name
        $name = $file->getRandomName();

        // Move the file to it's new home
        $file->move('/path/to/dir', $name);

        echo $file->getSize('mb');      // 1.23
        echo $file->getExtension();     // jpg
        echo $file->getType();          // image/jpg
}
```

你也可以通过 HTML 中提交的文件名去获取单个上传文件:

```
$file = $request->getFile('uploadedfile');
```

### 内容协商

你可以很轻松的通过 negotiate() 方法来完成信息内容类型的协商:

```
$language    = $request->negotiate('language', ['en-US', 'en-GB', 'fr',
↪'es-mx']);
$imageType   = $request->negotiate('media', ['image/png', 'image/jpg']);
$charset     = $request->negotiate('charset', ['UTF-8', 'UTF-16']);
$contentType = $request->negotiate('media', ['text/html', 'text/xml']);
$encoding    = $request->negotiate('encoding', ['gzip', 'compress']);
```

查看 Content Negotiation 获得更多细节。

### 类信息参考

**注解:** 除了这里列出的，本类还继承了 Request Class 和 Message Class 的方法。

以下方法由父类提供:

- CodeIgniter\HTTP\Request::getIPAddress()
- CodeIgniter\HTTP\Request::validIP()
- CodeIgniter\HTTP\Request::getMethod()
- CodeIgniter\HTTP\Request::getServer()

- CodeIgniter\HTTP\Message::body()
- CodeIgniter\HTTP\Message::setBody()
- CodeIgniter\HTTP\Message::populateHeaders()
- CodeIgniter\HTTP\Message::headers()
- CodeIgniter\HTTP\Message::header()
- CodeIgniter\HTTP\Message::headerLine()
- CodeIgniter\HTTP\Message::setHeader()
- CodeIgniter\HTTP\Message::removeHeader()
- CodeIgniter\HTTP\Message::appendHeader()
- CodeIgniter\HTTP\Message::protocolVersion()
- CodeIgniter\HTTP\Message::setProtocolVersion()
- CodeIgniter\HTTP\Message::negotiateMedia()
- CodeIgniter\HTTP\Message::negotiateCharset()
- CodeIgniter\HTTP\Message::negotiateEncoding()
- CodeIgniter\HTTP\Message::negotiateLanguage()
- CodeIgniter\HTTP\Message::negotiateLanguage()

**CodeIgniter\HTTP\IncomingRequest**

**isCLI()**

**返回** 由命令行发起的请求会返回 true ， 其他返回 false。

**返回类型** bool

**isAJAX()**

**返回** AJAX 请求返回 true ， 其他返回 false。

**返回类型** bool

**isSecure()**

**返回** HTTPS 请求返回 true ， 其他返回 false。

**返回类型** bool

**getVar(**[*$index = null*[, *$filter = null*[, *$flags = null*]]]**)**

**参数**

- **$index** (*string*) – 需要查找的数据名。
- **$filter** (*int*) – 过滤类型。参见列表 查看。
- **$flags** (*int*) – 过滤器名，值为过滤器的预定义变量名。参见列表 查看。

**返回** 不传参数会返回 REQUEST 中的所有元素，传参并且参数存在则返回对应的 REQUEST 值，不存在返回 null

**返回类型** mixed|null

第一个参数包含需要查找的数据名

```
$request->getVar('some_data');
```

如数据不存在则返回 null 。

只需传递期望的过滤类型到第二个参数，就可以帮助你完成数据过滤

```
$request->getVar('some_data', FILTER_SANITIZE_STRING);
```

不传任何参数会得到一个包含全部 REQUEST 数据的数组。

第一个参数 null ，第二个参数设置过滤类型，可获得一个被过滤的包涵全部 REQUEST 数据的数组

```
$request->getVar(null, FILTER_SANITIZE_STRING); // returns all␣
↪POST items with string sanitation
```

获取多个键值的信息，可以将需要的键值以数组形式传递给第一个参数

```
$request->getVar(['field1', 'field2']);
```

与之前一样，此时传递过滤类型给第二个参数，也可获得过滤后的数据

```
$request->getVar(['field1', 'field2'], FILTER_SANITIZE_STRING);
```

**getGet**([*$index = null*[, *$filter = null*[, *$flags = null*]]])

**参数**

- **$index** (*string*) – 需要查找的数据名。
- **$filter** (*int*) – 过滤类型。参见列表 查看。
- **$flags** (*int*) – 过滤器名，值为过滤器的预定义变量名。参见列表 查看。

**返回** 不传参数会返回 GET 中的所有元素，传参并且参数存在则返回对应的 GET 值，不存在返回 null

**返回类型** mixed|null

该方法与 getVar() 类似，只返回 GET 的数据。

**getPost**([*$index = null*[, *$filter = null*[, *$flags = null*]]])

**参数**

- **$index** (*string*) – 需要查找的数据名。
- **$filter** (*int*) – 过滤类型。参见列表 查看。

- **$flags** (*int*) – 过滤器名，值为过滤器的预定义变量名。参见列表 查看。

**返回** 不传参数会返回 POST 中的所有元素，传参并且参数存在则返回对应的 POST 值，不存在返回 null

**返回类型** mixed|null

该方法与 `getVar()` 类似，只返回 POST 的数据。

**getPostGet**([*$index = null*[, *$filter = null*[, *$flags = null*]]])

**参数**

- **$index** (*string*) – 需要查找的数据名。
- **$filter** (*int*) – 过滤类型。参见列表 查看。
- **$flags** (*int*) – 过滤器名，值为过滤器的预定义变量名。参见列表 查看。

**返回** 不传参数会返回 POST / GET 中的所有元素，传参并且参数存在则返回对应的 POST / GET 值，不存在返回 null

**返回类型** mixed|null

该方法和 `getPost()`，`getGet()` 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:

```
$request->getPostGet('field1');
```

**getGetPost**([*$index = null*[, *$filter = null*[, *$flags = null*]]])

**参数**

- **$index** (*string*) – 需要查找的数据名。
- **$filter** (*int*) – 过滤类型。参见列表 查看。
- **$flags** (*int*) – 过滤器名，值为过滤器的预定义变量名。参见列表 查看。

**返回** 不传参数会返回 POST / GET 中的所有元素，传参并且参数存在则返回对应的 POST / GET 值，不存在返回 null

**返回类型** mixed|null

该方法和 `getPost()`，`getGet()` 类似，它会同时查找 POST 和 GET 两个数组来获取数据，先查找 GET ，再查找 POST:

```
$request->getGetPost('field1');
```

**getCookie**([*$index = null*[, *$filter = null*[, *$flags = null*]]])

**参数**

- **$index** (*string*) – COOKIE 名。
- **$filter** (*int*) – 过滤类型。参见列表 查看。

- **$flags** (*int*) – 过滤器名，值为过滤器的预定义变量名。参见列表 查看。

**返回** 不传参数会返回 COOKIE 中的所有元素，传参并且参数存在则返回对应的 COOKIE 值，不存在返回 null

**返回类型** mixed

该方法与 getPost()，getGet() 类似，只返回 COOKIE 的数据

```
$request->getCookie('some_cookie');
$request->getCookie('some_cookie', FILTER_SANITIZE_STRING); //␣
↪with filter
```

获取多个键值的信息，可以将需要的键值以数组形式传递给第一个参数

```
$request->getCookie(array('some_cookie', 'some_cookie2'));
```

**注解:** 与*Cookie Helper* function *get_cookie()* 不同，该方法不会自动添加配置中 $config['cookie_prefix'] 的值。

**getServer**([*$index = null*[, *$filter = null*[, *$flags = null*]]])

**参数**

- **$index** (*string*) – 服务器信息名。
- **$filter** (*int*) – 过滤类型。参见列表 查看。
- **$flags** (*int*) – 过滤器名，值为过滤器的预定义变量名。参见列表 查看。

**返回** 不传参数会返回 SERVER 中的所有元素，传参并且参数存在则返回对应的 SERVER 值，不存在返回 null

**返回类型** mixed

该方法与 getPost()，getGet() ，getCookie() 类似，只返回 SERVER 的数据

```
$request->getServer('some_data');
```

获取多个键值的信息，可以将需要的键值以数组形式传递给第一个参数

```
$request->getServer(['SERVER_PROTOCOL', 'REQUEST_URI']);
```

**getUserAgent**([*$filter = null*])

**参数**

- **$filter** (*int*) – 过滤类型。参见列表 查看。

**返回** 包含 User Agent 信息的字符串，不存在返回 null

**返回类型** mixed

该方法从服务器信息哪查找并以字符串形式返回 User Agent

```
$request->getUserAgent();
```

### 5.1.7 内容协商

内容协商是一种用来根据客户端和服务端可处理的资源类型，来决定返回给客户端哪种类型的内容的机制。该机制可用来决定客户端是想要 HTML 还是想要 JSON ，一个图片是应该以 JPG 还是以 PNG 格式返回，或者支持哪种类型的压缩方法等。这些决策是通过分析四个不同的请求头，而这些请求头里支持多个带有优先级的值选项。手动对这些值选项进行优先级匹配通常是较有挑战性的，因此 CodeIgniter 提供了 `Negotiator` 来处理以上过程。

#### 加载类文件

你可以通过 Service 类来手动加载一个该类的实例:

```
$negotiator = \Config\Services::negotiator();
```

以上操作会获取所有的请求实例并自动将其自动注入到 Negotiator （协商，下同）类中。

该类并不需要主动加载。而是通过请求的 `IncomingRequest` 实例来进行范文。尽管你并不能通过这一过程直接访问该实例，你可以通过 `negotiate()` 方法来调用它的所有方法:

```
$request->negotiate('media', ['foo', 'bar']);
```

当通过该方法访问实例时，第一个参数是你需要匹配的内容的类型，第二个是所支持的类型值构成的数组。

#### 协商

本节中，我们将讨论四种可以用来协商的类型，并展示如何通过上述两种方法来进行内容协商。

#### 媒体

第一层首先要看的就是媒体协商。该协商方式是通过 `Accept` 请求头进行的，并且是可用的请求头中最为复杂的类型之一。一个常见的例子就是客户端告诉服务端其所需要的数据格式，而这种操作在 API 中最为常见。例如，一个客户端可能从一个 API 终点请求 JSON 编码的数据:

```
GET /foo HTTP/1.1
Accept: application/json
```

该服务器需要提供一个所支持的该内容的类型列表。在本例中，API 可能需要返回像原生 HTML ，JSON 或者是 XML 格式的数据。而根据客户端偏好，该列表应顺序返回:

```
$supported = [
        'application/json',
        'text/html',
        'application/xml'
];

$format = $request->negotiate('media', $supported);
// 或者是
$format = $negotiate->media($supported);
```

在本例中，客户端和服务器协商一致，将数据以 JSON 的格式返回，因此‘json’就会从协商方法中返回。默认情况下，如果没有匹配到，在 $support 数组中的第一个成员就会返回。尽管在某些情况下，你可能会强制要求服务端进行严格匹配格式。因此如果你将 true 作为最后参数传入时，在匹配不到时就会返回空字符串:

```
$format = $request->negotiate('media', $supported, true);
// 或
$format = $negotiate->media($supported, true);
```

### 语言

另一个常见的用法就是用于决定需要返回的内容的语言。如果你运行的是一个单语言网站，该功能显然并没有什么影响。但是如果对于那些提供多语言内容的网站来说，该功能就会变得非常有用，基于浏览器将通常会在 Accept-Language 请求头中发送偏好的语言类型:

```
GET /foo HTTP/1.1
Accept-Language: fr; q=1.0, en; q=0.5
```

本例中，浏览器偏好法语，并次偏好英语。如果你的网站支持英语或德语，那么你就会如下操作:

```
$supported = [
        'en',
        'de'
];

$lang = $request->negotiate('language', $supported);
// 或
$lang = $negotiate->language($supported);
```

本例中，”en”将作为当前语言返回。如果没有产生匹配，就会返回 $supported 数组的第一个成员，因此该成员将会一直作为偏好语言。

### 编码

Accept-Encoding 请求头包含了客户端所期望接收到的字符集，用于确定客户端支持哪种类型的压缩方式:

```
GET /foo HTTP/1.1
Accept-Encoding: compress, gzip
```

你的 web 服务器将会定义可以使用的压缩类型。某些服务器，例如 Apache，只支持了 **gzip**

```
$type = $request->negotiate('encoding', ['gzip']);
// 或
$type = $negotiate->encoding(['gzip']);
```

更多信息，参阅 Wikipedia.

### 字符集

所期待的字符集类型会通过 Accept-Charset 请求头来传值:

```
GET /foo HTTP/1.1
Accept-Charset: utf-16, utf-8
```

默认情况下，如果没有匹配的话就会返回 **utf-8**

```
$charset = $request->negotiate('charset', ['utf-8']);
// 或者是
$charset = $negotiate->charset(['utf-8']);
```

## 5.1.8 HTTP 类型伪装

当处理 HTML 表单时，你只可以使用 GET 或 POST 这两个 HTTP 动词。在大多数情况下，这种情况是没有问题的。然而为了支持 REST-ful 格式的路由，你需要支持其他更为正确的路由动词。例如 DELETE 或 PUT。由于浏览器不支持这种方式，CodeIgniter 提供了一种正在使用的伪装请求类型的方法。这种方法允许你发起一个 POST 请求，但是告诉程序这个请求应该被作为另一个请求类型而处理。

为了伪装请求类型，一个名为 _method 的隐藏输入字段需要被添加到表单中。这个字段的值应当是你希望发送的请求类型:

```
<form action="" method="post">
    <input type="hidden" name="_method" value="PUT" />

</form>
```

这个表单就会被转化成一个 PUT 请求，并且只要路由和 IncomingRequest 类能识别的话，这就是一个真正的 PUT 请求。

你所使用的表单必须得是一个 POST 请求，GET 请求无法被伪装。

---

**注解:** 请确认你的 Web 服务器的配置，因为有些服务器默认没有支持所有的 HTTP 动词，所以必须添加一些额外的包文件来开启这项功能。

---

### 5.1.9 处理 RESTful 请求资源



表现层状态转移 (REST) 是一种对于分布式应用的架构风格，最初由 Roy Fielding 在他的 2000 年博士论文 Architectural Styles and the Design of Network-based Software Architectures 所提出。上文可能读起来有些枯燥，你也可以参照 Martin Fowler 的 Richardson Maturity Model 以获得更方便的教程。

对于 REST 的架构方式，比起大多数软件架构体系，大家理解和误解得会更多。或者可以这样说，当你将 Roy Fielding 的原则越多得使用在一个架构中，你的应用就会变得越”RESTful”。

CodeIgniter 实现了一种简易的方式来创建 RESTful API 从而访问你的资源，通过其自带的资源路由和 *ResourceController* （资源控制器）。

#### 资源路由

对单个资源，你可以通过 `resource()` 方法来创建一个方便的 RESTful 路由。这种方式可以创建五种对于操作资源的 CRUD 方式，最为常见的比如：创建一个资源，更新一个已存在的资源，列出所有这类资源，获取一个单独资源以及删除一个单独的资源。第一个参数就是这个资源的名字:

```
$routes->resource('photos');

// 与以下方式等同:
$routes->get('photos/new',              'Photos::new');
$routes->post('photos',                 'Photos::create');
$routes->get('photos',                  'Photos::index');
$routes->get('photos/(:segment)',       'Photos::show/$1');
$routes->get('photos/(:segment)/edit', 'Photos::edit/$1');
$routes->put('photos/(:segment)',       'Photos::update/$1');
$routes->patch('photos/(:segment)',     'Photos::update/$1');
$routes->delete('photos/(:segment)',    'Photos::delete/$1');
```

**注解:** 上述的排序方式是为了排版清晰起见，而实际上这些路由在 RouteCollection 中的创建顺序已经确保了路由可以被正确地解析。

**重要:** 路由是由它们所定义的顺序而进行匹配的，因此如果像上面一样，如果你有一个 GET 资源图片请求，类似 “photos/poll” 一样，那么 show 的请求路由会比 get 请求优先被匹配。为了解决这个问题，将 get 请求这行移到资源行的顶部，从而它可以被首先匹配。

第二个参数接受的是一个包含着用于修改生成路由方式的选项数组。尽管这些路由通过 API 调用的，且这里支持更多的请求类型，你还是可以通过传递 “websafe” 选项来生成 update 和 delete 请求类型来处理 HTML 表单:

```
$routes->resource('photos', ['websafe' => 1]);

// 将会创建以下的同等效应的路由:
$routes->post('photos/(:segment)/delete', 'Photos::delete/$1');
$routes->post('photos/(:segment)',        'Photos::update/$1');
```

### 更改所使用的控制器

你可以通过指定所使用的控制器，通过为 `controller` 选项传值:

```
$routes->resource('photos', ['controller' =>'App\Gallery']);

// 将会创建如下路由
$routes->get('photos', 'App\Gallery::index');
```

### 更改使用的通配符

默认情况下，在需要资源 ID 时，我们需要使用 segment 占位符。你可以通过为 placeholder 选项传值一个新的字符串来实现这一操作:

```
$routes->resource('photos', ['placeholder' => '(:id)']);

// 将会创建如下路由：
$routes->get('photos/(:id)', 'Photos::show/$1');
```

### 限制生成的路由

你可以通过 only 选项来限制所生成的路由。这个选项的传值可以是一个数组或者是一个由逗号分隔的列表，其中包含着需要创建的类型名。而剩余的将会被忽略:

```
$routes->resource('photos', ['only' => ['index', 'show']]);
```

反过来你也可以通过 except 选项来移除那些不使用的路由。该选项就在 only 后运行:

```
$routes->resource('photos', ['except' => 'new,edit']);
```

合理的请求类型为: index, show, create, update, new, edit and delete.

### 资源控制器

*ResourceController* 为开始你的 RESTful API 的构建提供了一个非常便利的起点，实现了上述列举的资源路由的请求类型。

继承或重载 *modelName* 和 *format* 属性，并实现你想要处理的请求类型:

```
<?php namespace App\Controllers;

use CodeIgniter\RESTful\ResourceController;

class Photos extends ResourceController
{

        protected $modelName = 'App\Models\Photos';
        protected $format    = 'json';

        public function index()
        {
                return $this->respond($this->model->findAll());
        }

        // ...
}
```

上述路由结构如下:

```
$routes->resource('photos');
```

### 表现层路由

你可以使用 presenter() 方法来创建一个表现层路由，并分配给对应的资源控制器。这将会为那些给你的资源返回视图的的控制器方法创建路由，或者处理从这些控制器所创建的视图里发送的表单请求。

由于表现层惯例是由一个通用控制器来处理，这个功能不是必需的。它的用法与一个资源路由类似:

```
$routes->presenter('photos');

// 与如下等同:
$routes->get('photos/new',                 'Photos::new');
$routes->post('photos/create',             'Photos::create');
$routes->post('photos',                    'Photos::create');    // alias
$routes->get('photos',                     'Photos::index');
$routes->get('photos/show/(:segment)',     'Photos::show/$1');
$routes->get('photos/(:segment)',          'Photos::show/$1');   // alias
$routes->get('photos/edit/(:segment)',     'Photos::edit/$1');
$routes->post('photos/update/(:segment)', 'Photos::update/$1');
$routes->get('photos/remove/(:segment)',   'Photos::remove/$1');
$routes->post('photos/delete/(:segment)', 'Photos::update/$1');
```

**注解:** 上述的排序方式是为了排版清晰起见，而实际上这些路由在 RouteCollection 中的创建顺序已经确保了路由可以被正确地解析

可能对于 *photos* 你可能并不准备同时构建资源和表现层控制器，因此你需要对它们进行区分，例如:

```
$routes->resource('api/photo');
$routes->presenter('admin/photos');
```

第二个参数接受一个选项数组，用于修改生成的路由

### 更改所使用的控制器

你可以通过为 controller 选项传递需要更改的控制器的名字来指定实际用到的控制器:

```
$routes->presenter('photos', ['controller' =>'App\Gallery']);
```

(下页继续)

(续上页)

```
// 创建的路由如下:
$routes->get('photos', 'App\Gallery::index');
```

### 更改所使用的通配符

默认情况下, 当需要资源 ID 时, 我们使用 segment 通配符。你可以通过为 placeholder 选项传值一个新的字符串来指定新的通配符:

```
$routes->presenter('photos', ['placeholder' => '(:id)']);

// 生成路由如下:
$routes->get('photos/(:id)', 'Photos::show/$1');
```

限制生成的路由 Limit the Routes Made ————————

你可以通过 only 选项来限制生成的路由。该选项的传值应当是一个数组或者是一个逗号分隔的, 由所需要创建的方法的名字构成的列表。只有匹配上述方法的路由会被创建而其余的会被忽略:

```
$routes->presenter('photos', ['only' => ['index', 'show']]);
```

反过来你也可以通过 except 选项来去除那些无用的路由, 该选项位于 only 之后:

```
$routes->presenter('photos', ['except' => 'new,edit']);
```

可使用的方法为: index, show, new, create, edit, update, remove 和 delete.

ResourcePresenter(资源表现器) ResourcePresenter ==========

*ResourcePresenter* 为你输出一个资源对应的视图提供了一个便捷的起点, 而它同样也可利用属于该资源路由的方法来处理这些视图里提交的表单。

继承或重载 *modelName* 属性, 并实现那些你所需要调用的方法:

```
<?php namespace App\Controllers;

use CodeIgniter\RESTful\ResourcePresenter;

class Photos extends ResourcePresenter
{

        protected $modelName = 'App\Models\Photos';

        public function index()
        {
                return view('templates/list',$this->model->findAll());
        }
```

(下页继续)

(续上页)

```
        // ...
}
```

上述路由如下所示:

```
$routes->presenter('photos');
```

### 表现器/控制器对比

下表对比了用 *resource()* 和 *presenter()* 分别创建的默认路由以及对应的控制器方法。

| 操作 | 方法 | 控制器路由 | 表现层路由 | 控制器方法 | 表现层方法 |
|---|---|---|---|---|---|
| **New** | GET | photos/new | photos/new | `new()` | `new()` |
| **Create** | POST | photos | photos | `create()` | `create()` |
| Create (alias) | POST | | photos/create | | `create()` |
| **List** | GET | photos | photos | `index()` | `index()` |
| **Show** | GET | photos/(:segment) | photos/(:segment) | `show($id = null)` | `show($id = null)` |
| Show (alias) | GET | | photos/show/(:segment) | | `show($id = null)` |
| **Edit** | GET | photos/(:segment)/edit | photos/edit/(:segment) | `edit($id = null)` | `edit($id = null)` |
| **Update** | PUT/PATCH | photos/(:segment) | | `update($id = null)` | |
| Update (websafe) | POST | photos/(:segment) | photos/update/(:segment) | `update($id = null)` | `update($id = null)` |
| **Remove** | GET | | photos/remove/(:segment) | | `remove($id = null)` |
| **Delete** | DELETE | photos/(:segment) | | `delete($id = null)` | |
| Delete (websafe) | POST | | photos/delete/(:segment) | `delete($id = null)` | `delete($id = null)` |

## 5.2 构建响应

视图组件用于构建返回给用户的内容。

### 5.2.1 视图

视图只是一个网页或页面片段，例如页眉，页脚，侧边栏等。实际上，视图可以灵活地嵌入其他视图中（在其他视图内部）。

视图不会被直接调用，它必须通过控制器加载。请记住，在 MVC 框架中，控制器充当交通警察的作用，因此它专门负责读取特定的视图。如果你还没有阅读过控制器 页面，建议你应该先看下这个。

使用在控制器这一章里我们所创建的样例控制器，让我们为它创建一个视图。

### 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>
```

将文件保存到 **app/Views** 文件夹。

### 显示视图

要加载并且显示指定的视图文件，你需要用到下面的方法:

```
echo view('name');
```

*name* 是视图文件的名称。

**重要:** 如果你省略了文件的扩展名，那么框架会默认该文件以.php 扩展名结尾。

现在，打开你之前创建的 Blog.php 这个控制器文件，并将 echo 语句替换为 view 方法，以完成显示视图的功能:

```
<?php namespace App\Controllers;

class Blog extends \CodeIgniter\Controller
{
        public function index()
        {
                echo view('BlogView');
        }
}
```

如果你使用之前访问网站的 URL 来重新访问站点，你应该会看到新的视图。这个 URL 类似以下的内容:

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

**注解:** 尽管所有示例都是直接显示视图内容，但是你也可以让视图内容的结果返回给控制器；并将其添加到所有已捕获的输出内容中。（译者注：即不是直接输出而是返回一个字符串用作后续使用）

### 加载多个视图

CodeIgniter 可以智能的处理在控制器中多次调用 view() 方法。如果出现了多次调用，它们将被合并到一起。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 namespace App\Controllers;

class Page extends \CodeIgniter\Controller
{
        public function index()
        {
                $data = [
                        'page_title' => 'Your title'
                ];

                echo view('header');
                echo view('menu');
                echo view('content', $data);
                echo 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

如果你喜欢这样的组织形式，则视图文件可以保存到子目录中。当你这样做时，加载视图时需要包含子目录的名字，例如:

```
echo view('directory_name/file_name');
```

### 命名空间视图

您可以将视图存储在已命名空间的 **View** 目录下，并像加载加载命名空间一样加载视图。虽然 PHP 不支持在命名空间下加载非类文件，但是 CodeIgniter 提供了此功能，使你可以将它们以类似于模块的方式打包在一起，以便于重用或分发。

如果您在*自动加载* 文件 PSR-4 数组中设置 `Blog` 目录在 `Example\Blog` 命名空间下，则可以像使用命名空间一样找到视图文件。下面的示例就是通过在名称空间前添加视图名称来从 **/blog/views** 目录下加载 **BlogView** 文件:

```
echo view('Example\Blog\Views\BlogView');
```

---

**注解:** **译者注**这段有点难懂，需要和*模块* 章节一起看会比较容易懂。我的理解：框架中视图文件默认在 **app/Views** 目录下，当然这个也是可以通过 **app/Config/Paths.php** 类的 `$viewDirectory` 属性进行更改的。那么如果我们使用了 modules 功能把 Blog 模块独立出来，视图文件也是可以正常加载的，那么就需要在 **app/Config/Autoload.php** 文件中设定好映射目录，然后就可以通过命名空间的形式来加载视图文件了。

---

### 缓存视图

你可以通过 `view` 方法的第三个参数 `cache` 选项来实现视图缓存功能，缓存的实际单位是秒:

```
// 视图会缓存 60 秒
echo view('file_name', $data, ['cache' => 60]);
```

默认情况下，缓存视图的文件名与视图文件名相同。不过，你可以通过传递 `cache_name` 参数对缓存文件名进行自定义:

```
// 视图会缓存 60 秒
echo view('file_name', $data, ['cache' => 60, 'cache_name' => 'my_cached_
↪view']);
```

### 视图中显示动态数据

数据通过视图方法的第二个参数从控制器传递到视图，这是一个例子:

```
$data = [
        'title'   => 'My title',
        'heading' => 'My Heading',
        'message' => 'My Message'
];

echo view('blogview', $data);
```

让我们打开你的控制器文件，并添加一下代码:

```
<?php namespace App\Controllers;

class Blog extends \CodeIgniter\Controller
{
        public function index()
        {
                $data['title']   = "My Real Title";
                $data['heading'] = "My Real Heading";

                echo view('blogview', $data);
        }
}
```

现在打开视图文件，并将文本更改为与数据中的数组键对应的变量:

```
<html>
<head>
    <title><?= $title ?></title>
</head>
<body>
    <h1><?= $heading ?></h1>
</body>
</html>
```

现在重新刷新页面，你应该会看到变量已经替换成数据中的值。

默认情况下，传递的数据只在当前调用 *view* 中可用。如果在一次请求中多次调用该方法，则必须将所需的数据传递给每个视图。这样可以防止数据显示/覆盖到其他视图中的数据而导致出现问题。如果你想保留数据，则可以将 *saveData* 选项传递到第三个参数的 *$option* 数组中:

```
$data = [
        'title'   => 'My title',
        'heading' => 'My Heading',
        'message' => 'My Message'
];

echo view('blogview', $data, ['saveData' => true]);
```

另外，如果您希望 view 方法的默认功能是在调用之间保存数据，则可以在 **app/Config/Views.php** 中将 $saveData 设置为 **true**。

### 创建循环

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 namespace App\Controllers;

class Blog extends \CodeIgniter\Controller
{
        public function index()
        {
                $data = [
                        'todo_list' => ['Clean House', 'Call Mom', 'Run␣
↪Errands'],
                        'title'     => "My Real Title",
                        'heading'   => "My Real Heading"
                ];

                echo view('blogview', $data);
        }
}
```

现在打开视图文件并创建一个循环:

```
<html>
<head>
        <title><?= $title ?></title>
</head>
<body>
        <h1><?= $heading ?></h1>

        <h3>My Todo List</h3>

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

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

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

(下页继续)

(续上页)

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

### 5.2.2 子视图

子视图允许你插入在控制器以外生成的 HTML 片段，但它只能调用指定类的方法，且该方法只能返回有效的 HTML 字符串内容。这个可调用方法可以是在项目中自动加载器可以定位到的任何可访问类的任何方法，唯一的限制是该类的构造方法不可有必须传入的参数。使用这个功能后，对模块化代码有很好的帮助。

```
<?= view_cell('\App\Libraries\Blog::recentPosts') ?>
```

在这个示例中，会自动运行 App\Libraries\Blog 类的 recentPosts() 方法，该方法必须返回有效的 HTML 字符串。该方法可以是静态方法，也可以是非静态方法。

#### 子视图参数

可以通过 view_cell 方法的第二个参数向方法进行传值来进一步优化调用方式。参数支持键/值对的数组或键/值对的字符串（已逗号分隔）：

```
// 数组形式的参数
<?= view_cell('\App\Libraries\Blog::recentPosts', ['category' =>
↪'codeigniter', 'limit' => 5]) ?>

// 字符串形式的参数
<?= view_cell('\App\Libraries\Blog::recentPosts', 'category=codeigniter,␣
↪limit=5') ?>

public function recentPosts(array $params=[])
{
    $posts = $this->blogModel->where('category', $params['category'])
                             ->orderBy('published_on', 'desc')
                             ->limit($params['limit'])
                             ->get();

    return view('recentPosts', ['posts' => $posts]);
}
```

此外，可以在方法中使用与参数变量匹配的参数名称，以提高可读性。当以这种方式使用它时，必须始终在视图调用方法中指定所有参数:

```
<?= view_cell('\App\Libraries\Blog::recentPosts', 'category=codeigniter,␣
↪limit=5') ?>
```

(下页继续)

(续上页)

```
public function recentPosts(int $limit, string $category)
{
    $posts = $this->blogModel->where('category', $category)
                             ->orderBy('published_on', 'desc')
                             ->limit($limit)
                             ->get();

    return view('recentPosts', ['posts' => $posts]);
}
```

#### 子视图缓存

您可以通过传递缓存数据的秒数作为第三个参数来缓存子视图的调用结果，默认将使用当前配置的缓存引擎。

```
// 视图将缓存 5 分钟
<?= view_cell('\App\Libraries\Blog::recentPosts', 'limit=5', 300) ?>
```

当然，你也可以自定义缓存视图的文件名已替换默认的缓存文件名，通过第 4 个参数来自定义:

```
// 视图将缓存 5 分钟，将缓存文件重新命名为 newcacheid
<?= view_cell('\App\Libraries\Blog::recentPosts', 'limit=5', 300,
↪'newcacheid') ?>
```

### 5.2.3 视图渲染器



#### 使用视图渲染器

view() 方法是一种便捷功能，可以获取 renderer 服务实例，然后可以设置数据并显示视图。这种方式是我们常用的方式，但有时候我们需要一种更为直接的使用方式；在这种情况下你可以直接以视图服务的形式调用它:

```
$view = \Config\Services::renderer();
```

如果不使用 View 类作为默认渲染器，则可以直接实例化它:

```
$view = new \CodeIgniter\View\View();
```

---

**重要:** 你应该只在控制器内创建、使用服务。如果你想在其他类（Library）中使用，则应在类的构造函数中将其设置为依赖项。

---

然后，您可以使用它提供的三种标准方法中的任何一种: **render(viewpath, options, save)**, **setVar(name, value, context)** 和 **setData(data, context)**.

### 它是做什么的

View 类将视图的参数提取到可在脚本内部访问的 PHP 变量后，处理存储在应用程序视图路径中的标准 HTML/PHP 脚本。这意味着视图中的参数名称必须是合法的 PHP 变量名。

View 类在内部使用一个关联数组，以保存视图参数，直到调用 render() 方法为止。这意味着视图参数（或变量）名称必须是唯一的，否则后面变量的值将覆盖前面的变量。

同时，这还会影响脚本中不同上下文的参数值，你将必须为每个值赋予唯一的参数名称。

数组类型的值没有任何特殊含义，可以根据自己的 PHP 代码处理数组。

### 链式调用方法

*setVar()* 和 *setData()* 方法支持链式调用，允许将多个不同的调用组合到一个方法链中使用:

```
$view->setVar('one', $one)
     ->setVar('two', $two)
     ->render('myView');
```

### 转义数据

当你将数据传递给 setVar() 和 setData() 方法时，可以选择转义数据以防止跨站点脚本攻击。作为这两种方法中的最后一个参数，你可以传递所需的上下文，以选择是否对数据进行转义。

如果你不想对数据进行转义，你可以向每个方法的最后一个参数传递 *null* 或 *raw*，这样将不会对数据进行转义:

```
$view->setVar('one', $one, 'raw');
```

如果选择不转义数据，或者要传递对象实例，则可以使用 esc() 辅助方法在视图中手动转义数据。第一个参数是要转义的字符串，第二个参数是用于转义数据的上下文（请参见下文）：

```
<?= \esc($object->getStat()) ?>
```

**注解：** 译者注：框架内部使用 \Zend\Escaper\Escaper 类中以 escape 开头的相关方法对数据进行的转义处理。

### 转义上下文

默认情况下，esc() 方法认为要转义的数据会在 HTML 中使用。如果数据打算用于 Javascript、CSS 或 href 属性时，需要不同的转义规则才能生效。你可以传入转义类型名称作为第二个参数，选择合适的规则。规则支持‘html’，‘js’，‘css’，‘url’和‘attr’：

```
<a href="<?= esc($url, 'url') ?>" data-foo="<?= esc($bar, 'attr') ?>">
↪Some Link</a>

<script>
        var siteName = '<?= esc($siteName, 'js') ?>';
</script>

<style>
        body {
                background-color: <?= esc('bgColor', 'css') ?>
        }
</style>
```

### 视图渲染器选项

可以将多个选项信息传递给 render() 或 renderString() 方法：

- cache - 缓存视图结果的时间（以秒为时间单位），renderString() 方法中会忽略
- cache_name - 保存缓存视图结果的文件名，默认是 viewpath，renderString() 方法中会忽略
- saveData - 如果要保留视图的参数，并在后续调用中使用，应设置为 true

### 类参考

CodeIgniter\View\View

render(*$view[, $options[, $saveData=false]]*)

**参数**

- **$view** (*string*) – 源视图文件的文件名
- **$options** (*array*) – 以键值对传递的选项数组
- **$saveData** (*boolean*) – 如果该值为 true，该方法会保留该数据并为其他调用使用；反之就会在渲染视图后清除该数据

**返回** 指定视图文件所渲染的文字内容

**返回类型** string

根据传入的文件名和预先设置的数据来渲染输出:

```
echo $view->render('myview');
```

renderString(*$view[, $options[, $saveData=false]]*)

**参数**

- **$view** (*string*) – 需要渲染的视图的内容，例如从数据库里返回的内容等
- **$options** (*array*) – 以键值对传递的选项数组
- **$saveData** (*boolean*) – 如果该值为 true，该方法会保留该数据并为其他调用使用；反之就会在渲染视图后清除该数据

**返回** 指定视图文件所渲染的文字内容

**返回类型** string

根据给定的视图分块和预先设置的数据来渲染输出:

```
echo $view->renderString('<div>My Sharona</div>');
```

该方法可以用于输出一些例如数据库中存储的内容。但是你需要意识到这一操作可能是有安全风险的，并且 ** 必须 ** 对这些数据进行验证以及尽可能进行转义！

setData(*[$data[, $context=null]]*)

**参数**

- **$data** (*array*) – 以键值对传递的视图数据数组
- **$context** (*string*) – 使用数据转义的上下文

**返回** 用于方法链式调用的渲染器

**返回类型** CodeIgniter\View\RendererInterface.

同时设置多组视图数据:

```
$view->setData(['name'=>'George', 'position'=>'Boss']);
```

支持以下的上下文: html, css, js, url, or attr or raw, 如果是‘raw’的话，就不进行转义

每个调用都会为对象附加一个属性数据，直到视图被渲染

**setVar**(*$name*[, *$value=null*[, *$context=null*]])

**参数**

- **$name** (*string*) – 视图数据变量的变量名
- **$value** (*mixed*) – 该变量的变量值
- **$context** (*string*) – 使用数据转义的上下文

**返回** 用于方法链式调用的渲染器

**返回类型** CodeIgniter\View\RendererInterface.

设置单个数据变量:

```
$view->setVar('name','Joe','html');
```

支持以下的上下文: html, css, js, url, or attr or raw, 如果是‘raw’的话，就不进行转义

如果你想要使用对该对象已使用过的视图数据变量，新的值就会取代老的值。

### 5.2.4 视图布局



CodeIgniter 提供了一个简单但非常灵活的布局系统，使你可以轻松地在整个 web 应用程序中使用一个或多个基本页面布局。布局支持在任何渲染视图中插入内容节。你可以通过创建不同的布局来支持一栏、两栏或博客存档页面等。布局不会直接被渲染，但可以通过渲染一个视图（View），而该视图可以指定要扩展的布局（Layout）来实现（渲染布局）。

### 创建布局

布局和其他视图一样。它们唯一的区别是它们的用途。布局就是使用 renderSection() 方法的视图文件。这个方法会充当内容的占位符。

```
<!doctype html>
<html>
<head>
    <title>My Layout</title>
</head>
<body>
    <?= $this->renderSection('content') ?>
</body>
</html>
```

renderSection() 方法只有一个参数，那就是节的名称，这样所有子视图就都可以知道节的名称。

### 在视图中使用布局

无论何时需要把视图插入到布局中时，都必须在文件开头使用 extend() 方法：

```
<?= $this->extend('default') ?>
```

extend() 方法采用你所希望使用的视图文件的名称。由于它们也是视图，因此它们的位置就像视图一样。默认情况下，会在应用程序的 View 目录中查找它们，但还会扫描其他 PSR-4 定义的命名空间。你还可以加上一个命名空间以在特定名称空间的 View 目录中定位视图：

```
<?= $this->extend('Blog\Views\default') ?>
```

拓展布局所有内容时，必须包含 section($name) 和 endSection() 方法的调用。这些调用之间的任何内容都将插入到与节名称匹配的 renderSection($name) 调用所在的布局中：：

```
<?= $this->extend('default') ?>

<?= $this->section('content') ?>
    <h1>Hello World!</h1>
<?= $this->endSection() ?>
```

endSection() 不需要节的名称，它会自动结束需要结束的节。

### 渲染视图

渲染视图及其布局的方法与在控制器中显示的任何其他视图的方法完全相同：

```
public function index()
{
    echo view('some_view');
}
```

渲染器足够强大，它可以检测视图是需要单独渲染还是需要布局。

#### 引用局部视图

局部视图是不扩展任何布局的视图文件。它们通常是可以在视图之间重复使用的内容。使用视图布局时，必须使用 `$this->include()` 来引用。

```
<?= $this->extend('default') ?>

<?= $this->section('content') ?>
    <h1>Hello World!</h1>

    <?= $this->include('sidebar') ?>
<?= $this->endSection() ?>
```

调用 include() 方法时，可以将渲染普通视图时可以使用的所有选项都传递给它，包括缓存指令等。

### 5.2.5 View Parser

The View Parser can perform simple text substitution for pseudo-variables contained within your view files. It can parse simple variables or variable tag pairs.

Pseudo-variable names or control constructs are enclosed in braces, like this:

```
<html>
<head>
        <title>{blog_title}</title>
</head>
<body>
        <h3>{blog_heading}</h3>

        {blog_entries}
                <h5>{title}</h5>
                <p>{body}</p>
        {/blog_entries}

</body>
</html>
```

These variables are not actual PHP variables, but rather plain text representations that allow you to eliminate PHP from your templates (view files).

**注解:** CodeIgniter does **not** require you to use this class since using pure PHP in your view pages (for instance using the *View renderer* ) lets them run a little faster. However, some developers prefer to use some form of template engine if they work with designers who they feel would find some confusion working with PHP.

### Using the View Parser Class

The simplest method to load the parser class is through its service:

```
$parser = \Config\Services::parser();
```

Alternately, if you are not using the `Parser` class as your default renderer, you can instantiate it directly:

```
$parser = new \CodeIgniter\View\Parser();
```

Then you can use any of the three standard rendering methods that it provides: **render(viewpath, options, save)**, **setVar(name, value, context)** and **setData(data, context)**. You will also be able to specify delimiters directly, through the **setDelimiters(left,right)** method.

Using the `Parser`, your view templates are processed only by the Parser itself, and not like a conventional view PHP script. PHP code in such a script is ignored by the parser, and only substitutions are performed.

This is purposeful: view files with no PHP.

### What It Does

The `Parser` class processes "PHP/HTML scripts" stored in the application's view path. These scripts can not contain any PHP.

Each view parameter (which we refer to as a pseudo-variable) triggers a substitution, based on the type of value you provided for it. Pseudo-variables are not extracted into PHP variables; instead their value is accessed through the pseudo-variable syntax, where its name is referenced inside braces.

The Parser class uses an associative array internally, to accumulate pseudo-variable settings until you call its `render()`. This means that your pseudo-variable names need to be unique, or a later parameter setting will over-ride an earlier one.

This also impacts escaping parameter values for different contexts inside your script. You will have to give each escaped value a unique parameter name.

### Parser templates

You can use the `render()` method to parse (or render) simple templates, like this:

```
$data = [
        'blog_title'   => 'My Blog Title',
        'blog_heading' => 'My Blog Heading'
];

echo $parser->setData($data)
             ->render('blog_template');
```

View parameters are passed to `setData()` as an associative array of data to be replaced in the template. In the above example, the template would contain two variables: {blog_title} and {blog_heading} The first parameter to `render()` contains the name of the *view file*, Where *blog_template* is the name of your view file.

---

**重要:** If the file extension is omitted, then the views are expected to end with the .php extension.

---

### Parser Configuration Options

Several options can be passed to the `render()` or `renderString()` methods.

- `cache` - the time in seconds, to save a view' s results; ignored for renderString()
- **`cache_name` - the ID used to save/retrieve a cached view result; defaults to the view** ignored for renderString()
- **`saveData` - true if the view data parameters should be retained for subsequent calls;** default is **false**
- **`cascadeData` - true if pseudo-variable settings should be passed on to nested** substitutions; default is **true**

```
echo $parser->render('blog_template', [
        'cache'      => HOUR,
        'cache_name' => 'something_unique',
]);
```

### Substitution Variations

There are three types of substitution supported: simple, looping, and nested. Substitutions are performed in the same sequence that pseudo-variables were added.

The **simple substitution** performed by the parser is a one-to-one replacement of pseudo-variables where the corresponding data parameter has either a scalar or string value, as in this example:

```
$template = '<head><title>{blog_title}</title></head>';
$data     = ['blog_title' => 'My ramblings'];

echo $parser->setData($data)->renderString($template);

// Result: <head><title>My ramblings</title></head>
```

The `Parser` takes substitution a lot further with "variable pairs" , used for nested substitutions or looping, and with some advanced constructs for conditional substitution.

When the parser executes, it will generally

- handle any conditional substitutions
- handle any nested/looping substitutions
- handle the remaining single substitutions

### Loop Substitutions

A loop substitution happens when the value for a pseudo-variable is a sequential array of arrays, like an array of row settings.

The above example code allows simple variables to be replaced. What if you would like an entire block of variables to be repeated, with each iteration containing new values? Consider the template example we showed at the top of the page:

```
<html>
<head>
        <title>{blog_title}</title>
</head>
<body>
        <h3>{blog_heading}</h3>

        {blog_entries}
                <h5>{title}</h5>
                <p>{body}</p>
        {/blog_entries}

</body>
</html>
```

In the above code you'll notice a pair of variables: {blog_entries} data··· {/blog_entries}. In a case like this, the entire chunk of data between these pairs would be repeated multiple times, corresponding to the number of rows in the "blog_entries" element of the parameters array.

Parsing variable pairs is done using the identical code shown above to parse single variables, except, you will add a multi-dimensional array corresponding to your variable pair data. Consider this example:

```
$data = [
        'blog_title'   => 'My Blog Title',
        'blog_heading' => 'My Blog Heading',
        'blog_entries' => [
                ['title' => 'Title 1', 'body' => 'Body 1'],
                ['title' => 'Title 2', 'body' => 'Body 2'],
                ['title' => 'Title 3', 'body' => 'Body 3'],
                ['title' => 'Title 4', 'body' => 'Body 4'],
                ['title' => 'Title 5', 'body' => 'Body 5']
        ]
];

echo $parser->setData($data)
            ->render('blog_template');
```

The value for the pseudo-variable `blog_entries` is a sequential array of associative arrays. The outer level does not have keys associated with each of the nested "rows".

If your "pair" data is coming from a database result, which is already a multi-dimensional array, you can simply use the database `getResultArray()` method:

```
$query = $db->query("SELECT * FROM blog");

$data = [
        'blog_title'   => 'My Blog Title',
        'blog_heading' => 'My Blog Heading',
        'blog_entries' => $query->getResultArray()
];

echo $parser->setData($data)
            ->render('blog_template');
```

If the array you are trying to loop over contains objects instead of arrays, the parser will first look for an `asArray` method on the object. If it exists, that method will be called and the resulting array is then looped over just as described above. If no `asArray` method exists, the object will be cast as an array and its public properties will be made available to the Parser.

This is especially useful with the Entity classes, which has an asArray method that returns all public and protected properties (minus the _options property) and makes them available to the Parser.

### Nested Substitutions

A nested substitution happens when the value for a pseudo-variable is an associative array of values, like a record from a database:

```
$data = [
        'blog_title'   => 'My Blog Title',
        'blog_heading' => 'My Blog Heading',
        'blog_entry'   => [
                'title' => 'Title 1', 'body' => 'Body 1'
        ]
];

echo $parser->setData($data)
            ->render('blog_template');
```

The value for the pseudo-variable `blog_entry` is an associative array. The key/value pairs defined inside it will be exposed inside the variable pair loop for that variable.

A `blog_template` that might work for the above:

```
<h1>{blog_title} - {blog_heading}</h1>
{blog_entry}
        <div>
                <h2>{title}</h2>
                <p>{body}</p>
```

(下页继续)

(续上页)

```
        </div>
{/blog_entry}
```

If you would like the other pseudo-variables accessible inside the "blog_entry" scope, then make sure that the "cascadeData" option is set to true.

### Comments

You can place comments in your templates that will be ignored and removed during parsing by wrapping the comments in a `{# #}` symbols.

```
{# This comment is removed during parsing. #}
{blog_entry}
        <div>
                <h2>{title}</h2>
                <p>{body}</p>
        </div>
{/blog_entry}
```

### Cascading Data

With both a nested and a loop substitution, you have the option of cascading data pairs into the inner substitution.

The following example is not impacted by cascading:

```
$template = '{name} lives in {location}{city} on {planet}{/location}.';

$data = [
        'name'     => 'George',
        'location' => [ 'city' => 'Red City', 'planet' => 'Mars' ]
];

echo $parser->setData($data)->renderString($template);
// Result: George lives in Red City on Mars.
```

This example gives different results, depending on cascading:

```
$template = '{location}{name} lives in {city} on {planet}{/location}.';

$data = [
        'name'     => 'George',
        'location' => [ 'city' => 'Red City', 'planet' => 'Mars' ]
];
```

(下页继续)

(续上页)

```
echo $parser->setData($data)->renderString($template, ['cascadeData'=>
↪false]);
// Result: {name} lives in Red City on Mars.

echo $parser->setData($data)->renderString($template, ['cascadeData'=>
↪true]);
// Result: George lives in Red City on Mars.
```

### Preventing Parsing

You can specify portions of the page to not be parsed with the `{noparse}{/noparse}` tag pair. Anything in this section will stay exactly as it is, with no variable substitution, looping, etc, happening to the markup between the brackets.

```
{noparse}
        <h1>Untouched Code</h1>
{/noparse}
```

### Conditional Logic

The Parser class supports some basic conditionals to handle `if`, `else`, and `elseif` syntax. All `if` blocks must be closed with an `endif` tag:

```
{if $role=='admin'}
        <h1>Welcome, Admin!</h1>
{endif}
```

This simple block is converted to the following during parsing:

```
<?php if ($role=='admin'): ?>
        <h1>Welcome, Admin!</h1>
<?php endif ?>
```

All variables used within if statements must have been previously set with the same name. Other than that, it is treated exactly like a standard PHP conditional, and all standard PHP rules would apply here. You can use any of the comparison operators you would normally, like ==, ===, !==, <, >, etc.

```
{if $role=='admin'}
        <h1>Welcome, Admin</h1>
{elseif $role=='moderator'}
        <h1>Welcome, Moderator</h1>
{else}
```

(下页继续)

(续上页)

```
        <h1>Welcome, User</h1>
{endif}
```

**注解:** In the background, conditionals are parsed using an **eval()**, so you must ensure that you take care with the user data that is used within conditionals, or you could open your application up to security risks.

### Escaping Data

By default, all variable substitution is escaped to help prevent XSS attacks on your pages. CodeIgniter's `esc` method supports several different contexts, like general **html**, when it's in an HTML **attr*, in **css**, etc. If nothing else is specified, the data will be assumed to be in an HTML context. You can specify the context used by using the **esc** filter:

```
{ user_styles | esc(css) }
<a href="{ user_link | esc(attr) }">{ title }</a>
```

There will be times when you absolutely need something to used and NOT escaped. You can do this by adding exclamation marks to the opening and closing braces:

```
{! unescaped_var !}
```

### Filters

Any single variable substitution can have one or more filters applied to it to modify the way it is presented. These are not intended to drastically change the output, but provide ways to reuse the same variable data but with different presentations. The **esc** filter discussed above is one example. Dates are another common use case, where you might need to format the same data differently in several sections on the same page.

Filters are commands that come after the pseudo-variable name, and are separated by the pipe symbol, `|`:

```
// -55 is displayed as 55
{ value|abs  }
```

If the parameter takes any arguments, they must be separated by commas and enclosed in parentheses:

```
{ created_at|date(Y-m-d) }
```

Multiple filters can be applied to the value by piping multiple ones together. They are processed in order, from left to right:

```
{ created_at|date_modify(+5 days)|date(Y-m-d) }
```

### Provided Filters

The following filters are available when using the parser:

| Filter | Arguments | Description | Example |
|---|---|---|---|
| abs | | Displays the absolute value of a number. | { v\|abs } |
| capitalize | | Displays the string in sentence case: all lowercase with firstletter capitalized. | { v\|capitalize} |
| date | format (Y-m-d) | A PHP **date**-compatible formatting string. | { v\|date(Y-m-d) } |
| date_modify | value to add / subtract | A **strtotime** compatible string to modify the date, like `+5 day` or `-1 week`. | { v\|date_modify(+1 day) } |
| default | default value | Displays the default value if the variable is empty or undefined. | { v\|default(just in case) } |
| esc | html, attr, css, js | Specifies the context to escape the data. | { v\|esc(attr) } |
| excerpt | phrase, radius | Returns the text within a radius of words from a given phrase. Same as **excerpt** helper function. | { v\|excerpt(green giant, 20) } |
| highlight | phrase | Highlights a given phrase within the text using '<mark></mark>' tags. | { v\|highlight(view parser) } |
| highlight_code | | Highlights code samples with HTML/CSS. | { v\|highlight_code } |

下页继续

表 1 – 续上页

| | | | |
|---|---|---|---|
| limit_chars | limit | Limits the number of characters to $limit. | { v\|limit_chars(100) } |
| limit_words | limit | Limits the number of words to $limit. | { v\|limit_words(20) } |
| local_currency | currency, locale | Displays a localized version of a currency. "currency" valueis any 3-letter ISO 4217 currency code. | { v\|local_currency(EUR,en_US) } |
| local_number | type, precision, locale | Displays a localized version of a number. "type" can be one of: decimal, currency, percent, scientific, spellout, ordinal, duration. | { v\|local_number(decimal,2,en_US) } |
| lower | | Converts a string to lowercase. | { v\|lower } |
| nl2br | | Replaces all newline characters (n) to an HTML <br/> tag. | { v\|nl2br } |
| number_format | places | Wraps PHP **number_format** function for use within the parser. | { v\|number_format(3) } |
| prose | | Takes a body of text and uses the **auto_typography()** method to turn it into prettier, easier-to-read, prose. | { v\|prose } |
| round | places, type | Rounds a number to the specified places. Types of **ceil** and **floor** can be passed to use those functions instead. | { v\|round(3) } { v\|round(ceil) } |
| strip_tags | allowed chars | Wraps PHP **strip_tags**. Can accept a string of allowed tags. | { v\|strip_tags(<br>) } |

下页继续

表 1 – 续上页

| | | | |
|---|---|---|---|
| title | | Displays a "title case" version of the string, with all lowercase, and each word capitalized. | { v\|title } |
| upper | | Displays the string in all uppercase. | { v\|upper } |
| | | | |

See PHP's NumberFormatter for details relevant to the "local_number" filter.

### Custom Filters

You can easily create your own filters by editing **app/Config/View.php** and adding new entries to the `$filters` array. Each key is the name of the filter is called by in the view, and its value is any valid PHP callable:

```
public $filters = [
        'abs'        => '\CodeIgniter\View\Filters::abs',
        'capitalize' => '\CodeIgniter\View\Filters::capitalize',
];
```

### PHP Native functions as Filters

You can use native php function as filters by editing **app/Config/View.php** and adding new entries to the `$filters` array.Each key is the name of the native PHP function is called by in the view, and its value is any valid native PHP function prefixed with:

```
public $filters = [
        'str_repeat' => '\str_repeat',
];
```

### Parser Plugins

Plugins allow you to extend the parser, adding custom features for each project. They can be any PHP callable, making them very simple to implement. Within templates, plugins are specified by `{+ +}` tags:

```
{+ foo +} inner content {+ /foo +}
```

This example shows a plugin named **foo**. It can manipulate any of the content between its opening and closing tags. In this example, it could work with the text " inner content ". Plugins are processed before any pseudo-variable replacements happen.

While plugins will often consist of tag pairs, like shown above, they can also be a single tag, with no closing tag:

```
{+ foo +}
```

Opening tags can also contain parameters that can customize how the plugin works. The parameters are represented as key/value pairs:

```
{+ foo bar=2 baz="x y" }
```

Parameters can also be single values:

```
{+ include somefile.php +}
```

### Provided Plugins

The following plugins are available when using the parser:

| Plugin | Arguments | Description | Example |
|---|---|---|---|
| current_url | | Alias for the current_url helper function. | {+ current_url +} |
| previous_url | | Alias for the previous_url helper function. | {+ previous_url +} |
| siteURL | | Alias for the site_url helper function. | {+ siteURL "login" +} |
| mailto | email, title, attributes | Alias for the mailto helper function. | {+ mailto email=foo@example.com title=" Stranger Things" +} |
| safe_mailto | email, title, attributes | Alias for the safe_mailto helper function. | {+ safe_mailto email=foo@example.com title=" Stranger Things" +} |
| lang | language string | Alias for the lang helper function. | {+ lang number.terabyteAbbr +} |
| validation_errors | field-name(optional) | Returns either error string for the field (if specified) or all validation errors. | {+ validation_errors +} , {+ validation_errors field=" email" +} |
| route | route name | Alias for the route_to helper function. | {+ route "login" +} |

### Registering a Plugin

At its simplest, all you need to do to register a new plugin and make it ready for use is to add it to the **app/Config/View.php**, under the **$plugins** array. The key is the name of the plugin that is used within the template file. The value is any valid PHP callable, including static class methods, and closures:

```
public $plugins = [
        'foo'   => '\Some\Class::methodName',
        'bar'   => function($str, array $params=[]) {
                return $str;
        },
];
```

Any closures that are being used must be defined in the config file' s constructor:

```
class View extends \CodeIgniter\Config\View
{
    public $plugins = [];

    public function __construct()
    {
        $this->plugins['bar'] = function(array $params=[]) {
            return $params[0] ?? '';
        };

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

If the callable is on its own, it is treated as a single tag, not a open/close one. It will be replaced by the return value from the plugin:

```
public $plugins = [
        'foo'   => '\Some\Class::methodName'
];

// Tag is replaced by the return value of Some\Class::methodName␣
↪static function.
{+ foo +}
```

If the callable is wrapped in an array, it is treated as an open/close tag pair that can operate on any of the content between its tags:

```
public $plugins = [
        'foo' => ['\Some\Class::methodName']
];
```

(下页继续)

(续上页)

```
{+ foo +} inner content {+ /foo +}
```

### Usage Notes

If you include substitution parameters that are not referenced in your template, they are ignored:

```
$template = 'Hello, {firstname} {lastname}';
$data = [
        'title' => 'Mr',
        'firstname' => 'John',
        'lastname' => 'Doe'
];
echo $parser->setData($data)
             ->renderString($template);

// Result: Hello, John Doe
```

If you do not include a substitution parameter that is referenced in your template, the original pseudo-variable is shown in the result:

```
$template = 'Hello, {firstname} {initials} {lastname}';
$data = [
        'title'     => 'Mr',
        'firstname' => 'John',
        'lastname'  => 'Doe'
];
echo $parser->setData($data)
             ->renderString($template);

// Result: Hello, John {initials} Doe
```

If you provide a string substitution parameter when an array is expected, i.e. for a variable pair, the substitution is done for the opening variable pair tag, but the closing variable pair tag is not rendered properly:

```
$template = 'Hello, {firstname} {lastname} ({degrees}{degree} {/degrees})
↪';
$data = [
        'degrees'   => 'Mr',
        'firstname' => 'John',
        'lastname'  => 'Doe',
        'titles'    => [
                ['degree' => 'BSc'],
                ['degree' => 'PhD']
```

(下页继续)

(续上页)

```
        ]
];
echo $parser->setData($data)
             ->renderString($template);

// Result: Hello, John Doe (Mr{degree} {/degrees})
```

### View Fragments

You do not have to use variable pairs to get the effect of iteration in your views. It is possible to use a view fragment for what would be inside a variable pair, and to control the iteration in your controller instead of in the view.

An example with the iteration controlled in the view:

```
$template = '<ul>{menuitems}
        <li><a href="{link}">{title}</a></li>
{/menuitems}</ul>';

$data = [
        'menuitems' => [
                ['title' => 'First Link', 'link' => '/first'],
                ['title' => 'Second Link', 'link' => '/second'],
        ]
];
echo $parser->setData($data)
             ->renderString($template);
```

Result:

```
<ul>
        <li><a href="/first">First Link</a></li>
        <li><a href="/second">Second Link</a></li>
</ul>
```

An example with the iteration controlled in the controller, using a view fragment:

```
$temp = '';
$template1 = '<li><a href="{link}">{title}</a></li>';
$data1 = [
        ['title' => 'First Link', 'link' => '/first'],
        ['title' => 'Second Link', 'link' => '/second'],
];

foreach ($data1 as $menuItem)
{
```

(下页继续)

(续上页)

```
        $temp .= $parser->setData($menuItem)->renderString($template1);
}

$template2 = '<ul>{menuitems}</ul>';
$data = [
        'menuitems' => $temp
];
echo $parser->setData($data)
            ->renderString($template2);
```

Result:

```
<ul>
        <li><a href="/first">First Link</a></li>
        <li><a href="/second">Second Link</a></li>
</ul>
```

### Class Reference

**CodeIgniter\View\Parser**

**render**(*$view[, $options[, $saveData=false]]]*)

**参数**

- **$view** (*string*) – File name of the view source
- **$options** (*array*) – Array of options, as key/value pairs
- **$saveData** (*boolean*) – If true, will save data for use with any other calls, if false, will clean the data after rendering the view.

**返回** The rendered text for the chosen view

**返回类型** string

Builds the output based upon a file name and any data that has already been set:

```
echo $parser->render('myview');
```

Options supported:

- `cache` - the time in seconds, to save a view's results
- `cache_name` - the ID used to save/retrieve a cached view result; defaults to the viewpath
- `cascadeData` - true if the data pairs in effect when a nested or loop substitution occurs should be propagated

- `saveData` - true if the view data parameter should be retained for subsequent calls
- `leftDelimiter` - the left delimiter to use in pseudo-variable syntax
- `rightDelimiter` - the right delimiter to use in pseudo-variable syntax

Any conditional substitutions are performed first, then remaining substitutions are performed for each data pair.

**renderString**(*$template[, $options[, $saveData=false]]]*)

**参数**

- **$template** (*string*) – View source provided as a string
- **$options** (*array*) – Array of options, as key/value pairs
- **$saveData** (*boolean*) – If true, will save data for use with any other calls, if false, will clean the data after rendering the view.

**返回** The rendered text for the chosen view

**返回类型** string

Builds the output based upon a provided template source and any data that has already been set:

```
echo $parser->render('myview');
```

Options supported, and behavior, as above.

**setData**(*[$data[, $context=null]]*)

**参数**

- **$data** (*array*) – Array of view data strings, as key/value pairs
- **$context** (*string*) – The context to use for data escaping.

**返回** The Renderer, for method chaining

**返回类型** CodeIgniter\View\RendererInterface.

Sets several pieces of view data at once:

```
$renderer->setData(['name'=>'George', 'position'=>'Boss']);
```

**Supported escape contexts: html, css, js, url, or attr or raw.** If 'raw', no escaping will happen.

**setVar**(*$name[, $value=null[, $context=null]]*)

**参数**

- **$name** (*string*) – Name of the view data variable

- **$value** (*mixed*) – The value of this view data
- **$context** (*string*) – The context to use for data escaping.

**返回** The Renderer, for method chaining

**返回类型** CodeIgniter\View\RendererInterface.

Sets a single piece of view data:

```
$renderer->setVar('name','Joe','html');
```

**Supported escape contexts: html, css, js, url, attr or raw.** If 'raw', no escaping will happen.

**setDelimiters**(*$leftDelimiter = '{', $rightDelimiter = '}'*)

**参数**

- **$leftDelimiter** (*string*) – Left delimiter for substitution fields
- **$rightDelimiter** (*string*) – right delimiter for substitution fields

**返回** The Renderer, for method chaining

**返回类型** CodeIgniter\View\RendererInterface.

Over-ride the substitution field delimiters:

```
$renderer->setDelimiters('[',']');
```

### 5.2.6 HTML Table Class

The Table Class provides methods that enable you to auto-generate HTML tables from arrays or database result sets.



#### Using the Table Class

### Initializing the Class

The Table class is not provided as a service, and should be instantiated "normally", for instance:

```
$table = new \CodeIgniter\View\Table();
```

### Examples

Here is an example showing how you can create a table from a multi-dimensional array. Note that the first array index will become the table heading (or you can set your own headings using the `setHeading()` method described in the function reference below).

```
$table = new \CodeIgniter\View\Table();

$data = [
        ['Name', 'Color', 'Size'],
        ['Fred', 'Blue',  'Small'],
        ['Mary', 'Red',   'Large'],
        ['John', 'Green', 'Medium'],
];

echo $table->generate($data);
```

Here is an example of a table created from a database query result. The table class will automatically generate the headings based on the table names (or you can set your own headings using the `setHeading()` method described in the class reference below).

```
$table = new \CodeIgniter\View\Table();

$query = $db->query('SELECT * FROM my_table');

echo $table->generate($query);
```

Here is an example showing how you might create a table using discrete parameters:

```
$table = new \CodeIgniter\View\Table();

$table->setHeading('Name', 'Color', 'Size');

$table->addRow('Fred', 'Blue', 'Small');
$table->addRow('Mary', 'Red', 'Large');
$table->addRow('John', 'Green', 'Medium');

echo $table->generate();
```

Here is the same example, except instead of individual parameters, arrays are used:

```
$table = new \CodeIgniter\View\Table();

$table->setHeading(array('Name', 'Color', 'Size'));

$table->addRow(['Fred', 'Blue', 'Small']);
$table->addRow(['Mary', 'Red', 'Large']);
$table->addRow(['John', 'Green', 'Medium']);

echo $table->generate();
```

### Changing the Look of Your Table

The Table Class permits you to set a table template with which you can specify the design of your layout. Here is the template prototype:

```
$template = [
        '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>',

        'tfoot_open'          => '<tfoot>',
        'tfoot_close'         => '</tfoot>',

        'footing_row_start'   => '<tr>',
        'footing_row_end'     => '</tr>',
        'footing_cell_start'  => '<td>',
        'footing_cell_end'    => '</td>',

        '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>',
```

(下页继续)

(续上页)

```
        'cell_alt_start'     => '<td>',
        'cell_alt_end'       => '</td>',

        'table_close'        => '</table>'
];

$table->setTemplate($template);
```

**注解：** You' ll notice there are two sets of "row" blocks in the template. These permit you to create alternating row colors or design elements that alternate with each iteration of the row data.

You are NOT required to submit a complete template. If you only need to change parts of the layout you can simply submit those elements. In this example, only the table opening tag is being changed:

```
$template = [
        'table_open' => '<table border="1" cellpadding="2" cellspacing="1
↪" class="mytable">'
];

$table->setTemplate($template);
```

You can also set defaults for these by passing an array of template settings to the Table constructor.:

```
$customSettings = [
        'table_open' => '<table border="1" cellpadding="2" cellspacing="1
↪" class="mytable">'
];

$table = new \CodeIgniter\View\Table($customSettings);
```

### Class Reference

**class Table**

**$function = NULL**

Allows you to specify a native PHP function or a valid function array object to be applied to all cell data.

```
$table = new \CodeIgniter\View\Table();
```

(下页继续)

(续上页)

```
$table->setHeading('Name', 'Color', 'Size');
$table->addRow('Fred', '<strong>Blue</strong>', 'Small');

$table->function = 'htmlspecialchars';
echo $table->generate();
```

In the above example, all cell data would be run through PHP's `htmlspecialchars()` function, resulting in:

```
<td>Fred</td><td>&lt;strong&gt;Blue&lt;/strong&gt;</td><td>
↪Small</td>
```

**generate**([*$tableData = NULL*])

**参数**

- **$tableData** (*mixed*) – Data to populate the table rows with

**返回** HTML table

**返回类型** string

Returns a string containing the generated table. Accepts an optional parameter which can be an array or a database result object.

**setCaption**(*$caption*)

**参数**

- **$caption** (*string*) – Table caption

**返回** Table instance (method chaining)

**返回类型** *Table*

Permits you to add a caption to the table.

```
$table->setCaption('Colors');
```

**setHeading**([*$args = []*[, ...]])

**参数**

- **$args** (*mixed*) – An array or multiple strings containing the table column titles

**返回** Table instance (method chaining)

**返回类型** *Table*

Permits you to set the table heading. You can submit an array or discrete params:

```
$table->setHeading('Name', 'Color', 'Size'); // or

$table->setHeading(['Name', 'Color', 'Size']);
```

**setFooting**([*$args = []*[, ...]])

**参数**

- **$args** (*mixed*) – An array or multiple strings containing the table footing values

**返回** Table instance (method chaining)

**返回类型** *Table*

Permits you to set the table footing. You can submit an array or discrete params:

```
$table->setFooting('Subtotal', $subtotal, $notes); // or

$table->setFooting(['Subtotal', $subtotal, $notes]);
```

**addRow**([*$args = array()*[, ...]])

**参数**

- **$args** (*mixed*) – An array or multiple strings containing the row values

**返回** Table instance (method chaining)

**返回类型** *Table*

Permits you to add a row to your table. You can submit an array or discrete params:

```
$table->addRow('Blue', 'Red', 'Green'); // or

$table->addRow(['Blue', 'Red', 'Green']);
```

If you would like to set an individual cell' s tag attributes, you can use an associative array for that cell. The associative key **data** defines the cell' s data. Any other key => val pairs are added as key=' val' attributes to the tag:

```
$cell = ['data' => 'Blue', 'class' => 'highlight', 'colspan' =>␣
↪2];
$table->addRow($cell, 'Red', 'Green');

// generates
// <td class='highlight' colspan='2'>Blue</td><td>Red</td><td>
↪Green</td>
```

**makeColumns**([*$array = []*[, *$columnLimit = 0*]])

**参数**

- **$array** (*array*) – An array containing multiple rows' data
- **$columnLimit** (*int*) – Count of columns in the table

**返回** An array of HTML table columns

**返回类型** array

This method takes a one-dimensional array as input and creates a multi-dimensional array with a depth equal to the number of columns desired. This allows a single array with many elements to be displayed in a table that has a fixed column count. Consider this example:

```
$list = ['one', 'two', 'three', 'four', 'five', 'six', 'seven',
↪'eight', 'nine', 'ten', 'eleven', 'twelve'];

$newList = $table->makeColumns($list, 3);

$table->generate($newList);

// 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>
```

**setTemplate**(*$template*)

**参数**

- **$template** (*array*) – An associative array containing template values

**返回** TRUE on success, FALSE on failure

**返回类型** bool

Permits you to set your template. You can submit a full or partial template.

```
$template = [
        'table_open'   => '<table border="1" cellpadding="2"␣
↪cellspacing="1" class="mytable">'
```

(下页继续)

(续上页)

```
];

$table->setTemplate($template);
```

**setEmpty**(*$value*)

**参数**

- **$value** (*mixed*) – Value to put in empty cells

**返回** Table instance (method chaining)

**返回类型** *Table*

Lets you set a default value for use in any table cells that are empty. You might, for example, set a non-breaking space:

```
$table->setEmpty(" ");
```

**clear**()

**返回** Table instance (method chaining)

**返回类型** *Table*

Lets you clear the table heading, row data and caption. If you need to show multiple tables with different data you should to call this method after each table has been generated to clear the previous table information.

Example

```
$table = new \CodeIgniter\View\Table();


$table->setCaption('Preferences')
    ->setHeading('Name', 'Color', 'Size')
    ->addRow('Fred', 'Blue', 'Small')
    ->addRow('Mary', 'Red', 'Large')
    ->addRow('John', 'Green', 'Medium');

echo $table->generate();

$table->clear();

$table->setCaption('Shipping')
    ->setHeading('Name', 'Day', 'Delivery')
    ->addRow('Fred', 'Wednesday', 'Express')
    ->addRow('Mary', 'Monday', 'Air')
    ->addRow('John', 'Saturday', 'Overnight');

echo $table->generate();
```

### 5.2.7 HTTP 响应

响应类扩展了*HTTP 消息类*，只适用于服务器返回响应给调用它的客户端。



#### 使用响应类

响应类被实例化并传递到控制器。可以通过 `$this->response` 访问它。很多时候不需要直接使用它，因为 CodeIgniter 会为你发送标头和正文。如果一切正常，页面会成功创建被请求的内容。但是当出现问题时，或者当你需要发送指定的状态码，或者想要使用强大的 HTTP 缓存，可以立即使用它。

#### 设置输出内容

当需要直接设置脚本的输出内容时，不要依赖 CodeIgniter 来自动获取它，应该手动调用 `setBody` 方法。通常用于设置响应的状态码。

```
$this->response->setStatusCode(404)
               ->setBody($body);
```

响应中的原因短语（‘OK’，‘Created’，‘Moved Permenantly’）将被自动添加，但也可以通过为 `setStatusCode()` 方法设置第二个参数来添加自定义的原因。

```
$this->response->setStatusCode(404, 'Nope. Not here.');
```

#### 设置 HTTP 头

通常，你需要为响应设置 HTTP 头。响应类通过 `setHeader()` 方法简化了这个操作。

`setHeader()` 方法的第一个参数是 HTTP 头的名称，第二个参数是值，它可以是字符串或值的数组，当发送到客户端时将被正确组合。

使用这些函数而不是使用 PHP 原生函数，可以确保不会过早发送 HTTP 头导致错误，并使测试成为可能。

```
$response->setHeader('Location', 'http://example.com')
                ->setHeader('WWW-Authenticate', 'Negotiate');
```

如果 HTTP 头已经存在并且可以有多个值，可以使用 appendHeader() prependHeader() 方法分别将值添加到值列表的结尾或开头。

第一个参数是 HTTP 头的名称，第二个参数是添加到结尾或开头的值。

```
$response->setHeader('Cache-Control', 'no-cache')
                ->appendHeader('Cache-Control', 'must-revalidate');
```

HTTP 头可以用 removeHeader() 方法移除，此方法只接受 HTTP 头的名称作为唯一参数。并且不区分大小写。

```
$response->removeHeader('Location');
```

### 文件下载

响应类提供了一个简单地将文件发送给客户端的方法，提示浏览器下载文件。会设置适当的标题来实现。

第一个参数是 **下载文件的名称**，第二个参数是文件内容。

如果将第二个参数设为 NULL，并且 $filename 是一个已存在的，可读的文件路径，那么将会使用这个路径下的内容作为文件内容。

如果将第三个参数设置为布尔值 TRUE，那么实际的文件的 MIME 类型 (基于文件扩展名) 将被发送，这样当浏览器拥有该类型的处理程序 - 可以使用到它。

示例:

```
$data = 'Here is some text!';
$name = 'mytext.txt';
$response->download($name, $data);
```

如果要从服务器下载现有的文件，你需要这样做:

```
// photo.jpg 的内容将被自动读取
$response->download('/path/to/photo.jpg', NULL);
```

### HTTP 缓存

内置的 HTTP 规范是帮助客户端 (通常是 web 浏览器) 缓存结果的工具。

正确使用它，可以为应用程序带来巨大的性能提升，因为它会告诉客户端不需要联系服务器，因为没有任何改变。你不会比这更快。

这些都通过 Cache-Control 和 Etag 头来处理。本指南并不适合完整介绍缓存的功能，但你可以在 Google Developers 和 Mobify Blog 中了解更多。

默认情况下，所有通过 CodeIgniter 发送的响应都是关闭了 HTTP 缓存的。但在实际应用中，情况千变万化，无法简单的设置一个合适的默认值，除非关闭它，不过，可以通过 setCache() 方法设置你需要的缓存的值。这非常简单

```
$options = [
        'max-age'  => 300,
        's-maxage' => 900,
        'etag'     => 'abcde',
];
$this->response->setCache($options);
```

$options 是一个简单的键值对数组，它们被分配给 Cache-Control 头。你也可以根据具体情况自由设定所有选项。

虽然大多数选项都应用于 Cache-Control 头，但它会智能地处理 etag 和 last-modified 选项到适当的头。

### 内容安全策略 (CSP)

对 XSS 攻击的最佳保护方式之一是在站点上实施内容安全策略。

这迫使你将从你网站的 HTML 中载入的每一个内容来源列入白名单中，包括图片，样式表，JavaScript 文件等。浏览器将拒绝白名单外的的内容。这个白名单在响应的 Content-Security-Policy 标头中创建，并且有多种配置方式。

这听起来很复杂，在某些网站上肯定会有挑战性。对于很多简单的网站，所有的内容由相同的域名 (http://example.com) 提供，整合起来非常简单。

由于这是一个复杂的主题，本用户指南将不会覆盖所有细节。有关更多信息，你应该访问以下网站:

- Content Security Policy main site
- W3C Specification
- Introduction at HTML5Rocks
- Article at SitePoint

### 启用 CSP

默认情况下，CSP 策略是禁用的。想要在应用程序中启用 CSP，修改 **application/Config/App.php** 中的 CSPEnabled 的值

```
public $CSPEnabled = true;
```

当开启后，响应对象将包含一个 CodeIgniter\HTTP\ContentSecurityPolicy 的实例。

在 application/Config/ContentSecurityPolicy.php 中设置的值应用于这个实例，如果在运行时没有修改，那么将会发送正确的格式化后的标题，并且完成所有操作。

### 运行时配置

如果你的应用需要在运行时进行更改，则可以访问 $response->CSP 实例。该类拥有很多方法，可以很清晰地映射到你需要设置的 header 头

```
$reportOnly = true;

$response->CSP->reportOnly($reportOnly);
$response->CSP->setBaseURI('example.com', true);
$response->CSP->setDefaultSrc('cdn.example.com', $reportOnly);
$response->CSP->setReportURI('http://example.com/csp/reports');
$response->CSP->setSandbox(true, ['allow-forms', 'allow-scripts']);
$response->CSP->upgradeInsecureRequests(true);
$response->CSP->addChildSrc('https://youtube.com', $reportOnly);
$response->CSP->addConnectSrc('https://*.facebook.com', $reportOnly);
$response->CSP->addFontSrc('fonts.example.com', $reportOnly);
$response->CSP->addFormAction('self', $reportOnly);
$response->CSP->addFrameAncestor('none', $reportOnly);
$response->CSP->addImageSrc('cdn.example.com', $reportOnly);
$response->CSP->addMediaSrc('cdn.example.com', $reportOnly);
$response->CSP->addObjectSrc('cdn.example.com', $reportOnly);
$response->CSP->addPluginType('application/pdf', $reportOnly);
$response->CSP->addScriptSrc('scripts.example.com', $reportOnly);
$response->CSP->addStyleSrc('css.example.com', $reportOnly);
```

### 内联内容

可以设置一个网站不保护自己的页面上的内联脚本和样式，因为这可能是用户生成的内容的结果。为了防止这种情况，CSP 允许你再 <style> 和 <script> 标记中指定一个随机数，并将这些值添加到响应头中。这样处理很痛苦，但是却是最安全的。为了简单起见，你可以在代码中包含 {csp-style-nonce} 或 {csp-script-nonce} 占位符，程序将会自动为你处理

```
// Original
<script {csp-script-nonce}>
    console.log("Script won't run as it doesn't contain a nonce attribute
↪");
</script>

// Becomes
<script nonce="Eskdikejidojdk978Ad8jf">
    console.log("Script won't run as it doesn't contain a nonce attribute
↪");
```

(下页继续)

(续上页)

```
</script>

// OR
<style {csp-style-nonce}>
        . . .
</style>
```

### 类参考

**注解:** 除了这里列出的方法，响应类还继承了 消息类的方法。

父类提供的可用的方法:

- CodeIgniter\HTTP\Message::body()
- CodeIgniter\HTTP\Message::setBody()
- CodeIgniter\HTTP\Message::populateHeaders()
- CodeIgniter\HTTP\Message::headers()
- CodeIgniter\HTTP\Message::header()
- CodeIgniter\HTTP\Message::headerLine()
- CodeIgniter\HTTP\Message::setHeader()
- CodeIgniter\HTTP\Message::removeHeader()
- CodeIgniter\HTTP\Message::appendHeader()
- CodeIgniter\HTTP\Message::protocolVersion()
- CodeIgniter\HTTP\Message::setProtocolVersion()
- CodeIgniter\HTTP\Message::negotiateMedia()
- CodeIgniter\HTTP\Message::negotiateCharset()
- CodeIgniter\HTTP\Message::negotiateEncoding()
- CodeIgniter\HTTP\Message::negotiateLanguage()
- CodeIgniter\HTTP\Message::negotiateLanguage()

**CodeIgniter\HTTP\Response**

**statusCode()**

**返回** 此次响应的 HTTP 状态码

**返回类型** int

返回此响应的当前状态码，如果没有设置状态码，则会抛出 BadMethodCallException 异常。:

```
echo $response->statusCode();
```

**setStatusCode**(*$code*[, *$reason=''*])

**参数**

- **$code** (*int*) – HTTP 状态码
- **$reason** (*string*) – 一个可选的原因短语

**返回** 当前的响应实例

**返回类型** CodeIgniter\HTTP\Response

设置此次响应的 HTTP 状态码

```
$response->setStatusCode(404);
```

原因短语将会根据协议规定自动的生成。如果你需要为自定义状态码设置自己的愿意短语，你可以将原因短语作为第二个参数传递

```
$response->setStatusCode(230, "Tardis initiated");
```

**reason**()

**返回** 当前的原因短语。

**返回类型** string

返回此响应的当前状态码。如果没有设置状态，将返回一个空字符串

```
echo $response->reason();
```

**setDate**(*$date*)

**参数**

- **$date** (*DateTime*) – 一个设置了此响应的时间的 DateTime 实例。

**返回** 当前的响应类实例

**返回类型** CodeIgniter\HTTP\Response

设置响应的时间。$date 参数必须是一个 DateTime 实例

```
$date = DateTime::createFromFormat('j-M-Y', '15-Feb-2016');
$response->setDate($date);
```

**setContentType**(*$mime*[, *$charset='UTF-8'*])

**参数**

- **$mime** (*string*) – 响应的内容类型

- **$charset** (*string*) – 此响应使用的字符集。

**返回** 当前的响应类实例

**返回类型** CodeIgniter\HTTP\Response

设置此响应的内容类型

```
$response->setContentType('text/plain');
$response->setContentType('text/html');
$response->setContentType('application/json');
```

默认情况下，该方法将字符集设置为 UTF-8。如果你需要修改，可以将字符集作为第二个参数传递

```
$response->setContentType('text/plain', 'x-pig-latin');
```

**noCache**()

**返回** 当前的响应类实例

**返回类型** CodeIgniter\HTTP\Response

设置 Cache-Control 标头来关闭所有的 HTTP 缓存。这是所有响应消息的默认设置

```
$response->noCache();

// Sets the following header:
Cache-Control: no-store, max-age=0, no-cache
```

**setCache**(*$options*)

**参数**

- **$options** (*array*) – 一组缓存设置的键值

**返回** 当前的响应类实例

**返回类型** CodeIgniter\HTTP\Response

设置 Cache-Control 标头，包括 ETags 和 Last-Modified 。典型的键有:

- etag
- last-modified
- max-age
- s-maxage
- private
- public
- must-revalidate
- proxy-revalidate

- no-transform

当设置了 last-modified 选项时，它的值可以是一个 date 字符串，或一个 DateTime 对象。

**setLastModified**(*$date*)

**参数**

- **$date** (*string|DateTime*) – 设置 Last-Modified 的时间

**返回** 当前的响应类实例

**返回类型** CodeIgniter\HTTP\Response

设置 Last-Modified 头。$date 可以是一个字符串或一个 DateTime 实例

```
$response->setLastModified(date('D, d M Y H:i:s'));
$response->setLastModified(DateTime::createFromFormat('u',
↪$time));
```

**send**()

**返回** 当前的响应类实例

**返回类型** CodeIgniter\HTTP\Response

通知响应类发送内容给客户端。这将首先发送 HTTP 头，然后是响应的主体内容。对于主应用程序的响应，你不需要调用它，因为它由 CodeIgniter 自动处理。

**setCookie**(*$name = ''*[, *$value = ''*[, *$expire = ''*[, *$domain = ''*[, *$path = '/'*[, *$prefix = ''*[, *$secure = FALSE*[, *$httponly = FALSE*]]]]]]])

**参数**

- **$name** (*mixed*) – Cookie 名称或参数数组
- **$value** (*string*) – Cookie 值
- **$expire** (*int*) – Cookie 过期时间，单位：秒
- **$domain** (*string*) – Cookie 作用域
- **$path** (*string*) – Cookie 可用的路径
- **$prefix** (*string*) – Cookie 前缀
- **$secure** (*bool*) – 是否只通过 HTTPS 传输 Cookie
- **$httponly** (*bool*) – 是否只允许 HTTP 请求读取 cookie，JavaScript 不可以读取

**返回类型** void

设置一个包含你指定的值的 Cookie 。有两种将信息传递给该方法的方式: 数组和独立参数:

**数组方式**

使用此方法，将关联数组传递给第一个参数

```
$cookie = array(
        'name'   => 'The Cookie Name',
        'value'  => 'The Value',
        'expire' => '86500',
        'domain' => '.some-domain.com',
        'path'   => '/',
        'prefix' => 'myprefix_',
        'secure' => TRUE
);

$response->setCookie($cookie);
```

**注意事项**

只需要名称和值。要删除 Cookie ，将其设置为过期即可。

过期时间使用 **秒数**, 将从当前时间开始计算。

不要设置为一个具体的时间，而只是从 *now* 开始的你希望 Cookie 有效的秒数。

如果过期时间设置为零，Cookie 将只在浏览器打开时有效，浏览器关闭时则被清除。

对于整站的 Cookie ，无论你的网站是被如何请求的，请将你的网址添加到到 **domain** 中并且以. 开始，例如: .your-domain.com

通常不需要该路径，因为默认已经设置了根目录。

仅当你需要避免与服务器的其他相同命名的 Cookie 冲突时，才需要前缀。

仅当你想要加密 Cookie 时才需要设置 secure 项为 TRUE。

**独立参数**

如果你愿意，也可以使用单个参数传递数据来设置 Cookie。

```
$response->setCookie($name, $value, $expire, $domain, $path,
↪$prefix, $secure);
```

### 5.2.8 API 响应特性

现代化的 PHP 开发都需要构建 API ，不管它只是为了给 javascript 单页应用提供数据还是作为独立的产品。CodeIgniter 提供了一个 API 响应特性，可用于任何控制器，使公共响应类型简单，无需记住它的 HTTP 状态代码应返回的响应类型。

### 使用示例

下面的示例显示了控制器中常见的使用模式。

```
<?php namespace App\Controllers;

class Users extends \CodeIgniter\Controller
{
    use CodeIgniter\API\ResponseTrait;

    public function createUser()
    {
        $model = new UserModel();
        $user = $model->save($this->request->getPost());

        // 响应 201 状态码
        return $this->respondCreated();
    }
}
```

在这个例子中，响应了 201 的 HTTP 状态码，并使用 “创建” 的通用状态消息返回。方法存在于最常见的用例中

```
// 通用响应方式
respond($data, 200);
// 通用错误响应
fail($errors, 400);
// 项目创建响应
respondCreated($data);
// 项目成功删除
respondDeleted($data);
// 客户端未授权
failUnauthorized($description);
// 禁止动作
failForbidden($description);
// 找不到资源
failNotFound($description);
// Data 数据没有验证
failValidationError($description);
// 资源已存在
failResourceExists($description);
```

(下页继续)

(续上页)

```
// 资源早已被删除
failResourceGone($description);
// 客户端请求数过多
failTooManyRequests($description);
```

### 处理响应类型

当您通过以下任何一种方法传递数据时，它们将决定基于数据类型来格式化结果:

- 如果 $data 是一个字符串，它将被当作 HTML 发送回客户端。
- 如果 $data 是一个数组，它将尝试请求内容类型与客户端进行协商，默认为 JSON。如果没有在 ConfigAPI.php 中配置内容。默认使用 `$supportedResponseFormats` 属性。

需要使用格式化，请修改 **Config/Format.php** 文件配置。`$supportedResponseFormats` 包含了一个格式化响应类型列表。默认情况下，系统将会自动判断并响应 XML 和 JSON 格式:

```
public $supportedResponseFormats = [
    'application/json',
    'application/xml'
];
```

这是在 Content Negotiation 中使用的数组，以确定返回的响应类型。如果在客户端请求的内容和您支持的内容之间没有匹配，则返回第一个该数组中的格式。

接下来，需要定义用于格式化数据数组的类。这必须是一个完全合格的类名，类名必须实现 **CodeIgniterAPIFormatterInterface**。格式化支持 JSON 和 XML

```
public $formatters = [
    'application/json' => \CodeIgniter\API\JSONFormatter::class,
    'application/xml'  => \CodeIgniter\API\XMLFormatter::class
];
```

因此，如果您的请求在 **Accept** 头中请求 JSON 格式的数据，那么您传递的数据数组就可以通过其中任何一个 `respond*` 或 `fail*` 方法将由 **CodeIgniterAPIJSONFormatter** 格式化。由此产生的 JSON 数据将被发送回客户端。

### 引用类

**respond**(*$data*[, *$statusCode=200*[, *$message=''*]])

**参数**

- **$data** (*mixed*) – 返回客户端的数据。字符串或数组。
- **$statusCode** (*int*) – 返回的 HTTP 状态码。默认为 200。

- **$message** (*string*) – 返回的自定义 “reason” 消息。

这是该特征中所有其他方法用于将响应返回给客户端的方法。

$data 元素可以是字符串或数组。默认情况下，一个字符串将作为 HTML 返回，而数组将通过 json_encode 运行并返回为 JSON，除非 Content Negotiation 确定它应该以不同的格式返回。

如果一个 $message 字符串被传递，它将被用来替代标准的 IANA 标准码回应状态。但不是每个客户端都会遵守自定义代码，并将使用 IANA 标准

匹配状态码。

---

**注解:** 由于它在活动的响应实例上设置状态码和正文，所以应该一直作为脚本执行中的最终方法。

---

**fail**(*$messages*[, *int $status=400*[, *string $code=null*[, *string $message=”*]]])

**参数**

- **$messages** (*mixed*) – 包含遇到错误消息的字符串或字符串数组。
- **$status** (*int*) – 返回的 HTTP 状态码。默认为 400。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** 以客户端的首选格式进行多部分响应。

这是用于表示失败的响应的通用方法，并被所有其他 “fail” 方法使用。

该 $messages 元素可以是字符串或字符串数 组。该 $status 参数是应返回的 HTTP 状态码。

由于使用自定义错误代码更好地提供了许多 API，因此可以在第三个参数中传递自定义错误代码。如果没有值，它将是一样的 $status 【状态码】。

如果一个 $message 字符串被传递，它将被用于代替响应状态的标准 IANA 码。不是每个客户端都会遵守自定义代码，并且将使用与状态代码相匹配的 IANA 标准。

这个响应是一个包含两个元素的数组：error 和 messages 。error 元素包含错误的状态代码。messages 元素包含一组错误消息。它看起来像:

```
$response = [
    'status' => 400,
    'code' => '321a',
    'messages' => [
        'Error message 1',
        'Error message 2'
    ]
];
```

**respondCreated**(*$data*[, *string $message = ''*])

**参数**

- **$data** (*mixed*) – 返回给客户端的数据。字符串或数组。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

设置创建新资源时使用的相应状态代码，通常为 201:

```
$user = $userModel->insert($data);
return $this->respondCreated($user);
```

**respondDeleted**(*$data*[, *string $message = ''*])

**参数**

- **$data** (*mixed*) – 返回给客户端的数据。字符串或数组
- **$message** (*string*) – 自定义的 “原因” 消息返回。

**返回** Response 对象的 send() 方法的值。

设置当通过此 API 调用的结果删除新资源时使用的相应状态代码（通常为 200）。

```
$user = $userModel->delete($id);
return $this->respondDeleted(['id' => $id]);
```

**failUnauthorized**(*string $description*[, *string $code=null*[, *string $message = ''*]])

**参数**

- **$description** (*mixed*) – 显示用户的错误信息。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

设置当用户未被授权或授权不正确时使用的相应状态代码。状态码为 401。

```
return $this->failUnauthorized('Invalid Auth token');
```

**failForbidden**(*string $description*[, *string $code=null*[, *string $message = ''*]])

**参数**

- **$description** (*mixed*) – 显示用户的错误信息。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

不像 failUnauthorized，当请求 API 路径决不允许采用这种方法。未经授权意味着客户端被鼓励再次尝试使用不同的凭据。禁止意味着客户端不应该再次尝试，因为它不会有帮助。状态码为 403。

```
return $this->failForbidden('Invalid API endpoint.');
```

**failNotFound**(*string $description*[, *string $code=null*[, *string $message = ''*]])

**参数**

- **$description** (*mixed*) – 显示用户的错误信息。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

设置于在找不到请求的资源时使用的状态码。状态码为 404。

```
return $this->failNotFound('User 13 cannot be found.');
```

**failValidationError**(*string $description*[, *string $code=null*[, *string $message = ''*]])

**参数**

- **$description** (*mixed*) – 显示用户的错误信息。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

设置于客户端发送的数据未通过验证规则时使用的状态码。状态码通常为 400。

```
return $this->failValidationError($validation->getErrors());
```

**failResourceExists**(*string $description*[, *string $code=null*[, *string $message = ''*]])

**参数**

- **$description** (*mixed*) – 显示用户的错误信息。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

设置于当客户端尝试创建的资源已经存在时使用的状态码。状态码通常为 409。

```
return $this->failResourceExists('A user already exists with that␣
↪email.');
```

failResourceGone(*string $description*[, *string $code=null*[, *string $message = ''*]])

**参数**

- **$description** (*mixed*) – 显示用户的错误信息。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

设置于当请求的资源先前被删除并且不再使用时使用的状态码。状态码通常为 410。

```
return $this->failResourceGone('That user has been previously␣
↪deleted.');
```

failTooManyRequests(*string $description*[, *string $code=null*[, *string $message = ''*]])

**参数**

- **$description** (*mixed*) – 显示用户的错误信息。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

设置于当客户端调用 API 路径次数过多时使用的状态码。这可能是由于某种形式的节流或速率限制。状态码通常为 400。

```
return $this->failTooManyRequests('You must wait 15 seconds before␣
↪making another request.');
```

failServerError(*string $description*[, *string $code = null*[, *string $message = ''*]])

**参数**

- **$description** (*mixed*) – 显示用户的错误信息。
- **$code** (*string*) – 一个自定义的 API 特定的错误代码。
- **$message** (*string*) – 返回的自定义 “reason” 消息。

**返回** Response 对象的 send() 方法的值。

设置于当存在服务器错误时使用的状态码。

```
return $this->failServerError('Server error.');
```

### 5.2.9 本地化



#### 处理不同地域

CodeIgniter 提供了一系列帮助你处理多语言环境下将应用本土化的工具。尽管一个应用完全地本土化是一个复杂的问题，在你的应用中将一些字符串根据不同的语言进行替换，是相当简单的。

语言字符串存储于 **app/Language** 目录下，其下的每个子目录都代表着一种所支持的语言:

```
/app
    /Language
        /en
            app.php
        /fr
            app.php
```

**重要:** 地区的识别仅对于使用了 IncomingRequest 类的基于 web 的请求起效，命令行请求无法使用这些功能。

#### 配置地区

每个站点都拥有默认的语言/地区属性，可以通过 **Config/App.php** 进行设置:

```
public $defaultLocale = 'en';
```

该变量的值，可以是任何字符串值，用于你的应用程序处理文本字符串和其他格式用。我们推荐使用 BCP 47 类型的语言代号。该说明中，语言编码是例如用于美国英语的

en-US 或者是用于法国法语的 fr-FR 的格式。或者也可以参照 W3C’ s site 以获取可读性更高的说明。

如果不能找到绝对匹配的语言代码时，该系统将足够灵活地使用更为泛化的语言代码。如果地区代码被设为 **en-US** ，而只有 **en** 语言的语言文件，那么因为没有更为精确地匹配 **en-US** 的语言，我们就会使用这些语言文件。但是如果有一个语言文件目录 **app/Language/en-US** 存在的话，该目录里的语言文件就会被首先使用。

### 地区识别

我们有两种方式用于在请求中识别正确的地区。第一种方式是”设后即忘”的方式，并会自动执行内容协商 以决定使用正确的地区。第二种方式使得你可以在路由中给定一个特定的分段并用于设置地区。

### 内容协商

你可以通过在 Config/App 中设置两个额外的参数来自动开启内容协商。第一个参数用于告诉 Request 类我们需要开启内容协商，因此只要将其设为 true 即可:

```
public $negotiateLocale = true;
```

当该参数启用时，系统会自动根据你在 $suppoertLocales 中定义的语言数组来协商使用正确的语言。如果在你提供的语言和所请求的语言中中匹配不到的话，该数组的第一个成员就会被使用。在下例中，在不匹配时，**en** 地区就会被使用:

```
public $supportedLocales = ['en', 'es', 'fr-FR'];
```

### 在路由中

第二种方法是用一个自定义的通配符来检测所需要的地区，并将其用于当前请求中。在你的路由中，通配符 {locale}} 可以被替换为一个路由分段。如果该分段存在的话，所匹配到的路由分段就是你的地区:

```
$routes->get('{locale}/books', 'App\Books::index');
```

在本例中，如果用户尝试访问 http://example.com/fr/books ，地区就会被设置为 fr ，并假设这是一个合理的地区参数。

---

**注解:** 如果该路由分段值匹配不到 App 配置文件中合理的地区值的话，就会用默认的地区来代替。

---

### 获取当前地区

当前地区默认从 IncomingRequest 实例中获取，通过 getLocale() 方法。如果你的控制器继承了 CodeIgniter\Controller，以上操作也可以通过 $this->request 来实现:

```
<?php namespace App\Controllers;

class UserController extends \CodeIgniter\Controller
{
    public function index()
    {
        $locale = $this->request->getLocale();
    }
}
```

或者你也可以用服务类 来获取当前的请求:

```
$locale = service('request')->getLocale();
```

## 语言本土化

### 创建语言文件

Languages do not have any specific naming convention that are required. The file should be named logically to describe the type of content it holds. For example, let's say you want to create a file containing error messages. You might name it simply: **Errors.php**.

Within the file, you would return an array, where each element in the array has a language key and the string to return:

```
'language_key' => 'The actual message to be shown.'
```

---

**注解:** It's good practice to use a common prefix for all messages in a given file to avoid collisions with similarly named items in other files. For example, if you are creating error messages you might prefix them with error_

---

```
return [
    'errorEmailMissing'    => 'You must submit an email address',
    'errorURLMissing'      => 'You must submit a URL',
    'errorUsernameMissing' => 'You must submit a username',
];
```

### 基本用途

你可以使用 lang() 辅助函数从所有语言文件中获取文本值，通过将文件名和语言键作为第一个参数，以点号 (.) 分隔。举例来说，从 Errors 语言文件中加载 errorEmailMissing 字符串，你可以如下操作:

```
echo lang('Errors.errorEmailMissing');
```

如果所请求的语言键对于当前的地区来说不存在的话，就会不做修改的返回请求的参数。在本例中，如果‘Errors.errorEmailMissing’对应的翻译不存在的话，就会直接被返回。

### 参数替换

---

**注解:** 以下函数需要加载并启用 intl 扩展。如果该扩展未加载，则不会进行替换操作。可参阅 Sitepoint.

---

你可以在语言字符串中，通过对 lang() 函数的第二个参数传递一个值数组来替代通配符中的内容。这一操作对于简单的数字翻译和格式化来说非常方便:

```
// 语言文件, Tests.php:
return [
    "apples"      => "I have {0, number} apples.",
    "men"         => "I have {1, number} men out-performed the remaining
↪{0, number}",
    "namedApples" => "I have {number_apples, number, integer} apples.",
];

// 输出 "I have 3 apples."
echo lang('Tests.apples', [ 3 ]);
```

通配符中的第一项对应着数组的索引下标（如果该下标是数字格式的话):

```
// 输出 "The top 23 men out-performed the remaining 20"
echo lang('Tests.men', [20, 23]);
```

如果希望的话，你也可以使用命名数组来更为直接地传递参数:

```
// 显示 "I have 3 apples."
echo lang("Tests.namedApples", ['number_apples' => 3]);
```

显然你可以实现比起数字替换更为高级的功能。根据标准库 official ICU docs 所示，以下类型的数据可被替换:

- numbers - 整数，汇率，百分比
- dates - 短，中，长，完整格式

- time - 短，中，长，完整格式
- spellout - 大写数字 (例如 34 变成 thirty-four)
- ordinal
- duration

Here are a few examples:

```
// The language file, Tests.php
return [
    'shortTime'  => 'The time is now {0, time, short}.',
    'mediumTime' => 'The time is now {0, time, medium}.',
    'longTime'   => 'The time is now {0, time, long}.',
    'fullTime'   => 'The time is now {0, time, full}.',
    'shortDate'  => 'The date is now {0, date, short}.',
    'mediumDate' => 'The date is now {0, date, medium}.',
    'longDate'   => 'The date is now {0, date, long}.',
    'fullDate'   => 'The date is now {0, date, full}.',
    'spelledOut' => '34 is {0, spellout}',
    'ordinal'    => 'The ordinal is {0, ordinal}',
    'duration'   => 'It has been {0, duration}',
];

// Displays "The time is now 11:18 PM"
echo lang('Tests.shortTime', [time()]);
// Displays "The time is now 11:18:50 PM"
echo lang('Tests.mediumTime', [time()]);
// Displays "The time is now 11:19:09 PM CDT"
echo lang('Tests.longTime', [time()]);
// Displays "The time is now 11:19:26 PM Central Daylight Time"
echo lang('Tests.fullTime', [time()]);

// Displays "The date is now 8/14/16"
echo lang('Tests.shortDate', [time()]);
// Displays "The date is now Aug 14, 2016"
echo lang('Tests.mediumDate', [time()]);
// Displays "The date is now August 14, 2016"
echo lang('Tests.longDate', [time()]);
// Displays "The date is now Sunday, August 14, 2016"
echo lang('Tests.fullDate', [time()]);

// Displays "34 is thirty-four"
echo lang('Tests.spelledOut', [34]);

// Displays "It has been 408,676:24:35"
echo lang('Tests.ordinal', [time()]);
```

你需要阅读 MessageFormatter 类以及 ICU 编码格式以充分使用这一功能的特性，例如

执行条件替换，多元素替换等。以上两者的链接都在上文中有所提及，希望可以可以帮助你充分利用这一特性。

### 确定地区

为了在替换参数时显式调用一个不同的地区，你可以通过将地区作为 lang() 方法的第三个参数来实现:

```
// Displays "The time is now 23:21:28 GMT-5"
echo lang('Test.longTime', [time()], 'ru-RU');

// Displays "£7.41"
echo lang('{price, number, currency}', ['price' => 7.41], 'en-GB');
// Displays "$7.41"
echo lang('{price, number, currency}', ['price' => 7.41], 'en-US');
```

### 嵌套数组

语言文件可以接受嵌套数组作为参数，以更为方便地处理列表类型的数据等:

```
// Language/en/Fruit.php

return [
    'list' => [
        'Apples',
        'Bananas',
        'Grapes',
        'Lemons',
        'Oranges',
        'Strawberries'
    ]
];

// Displays "Apples, Bananas, Grapes, Lemons, Oranges, Strawberries"
echo implode(', ', lang('Fruit.list'));
```

### 语言回滚

如果对于一个给定的地区，你有多种语言文件类型，例如对于 Language/en.php ，你可以通过为这一地区增加一个语言变量，例如 Language/en-US/app.php

你唯一需要为这些信息提供的就是它们在不同地区里的值。如果对应的信息翻译不存在的话，就会从主地区设置中获取并赋值。

本土化功能可以将所有翻译信息回滚为英语，以防止在新的信息增加到框架中时，你没办法为所在地区实现翻译。

因此，如果你在使用地区 fr-CA ，那么翻译信息会首先从 Language/fr/CA 中搜索，然后在 Language/fr ， 最后在 Language/en 中。

#### 信息翻译

在我们的 仓库 . 中，有一份”正式的”翻译集

你可以下载该仓库并复制其中的 Language 目录到你的 app 中。因为 App 命名空间映射到了你的 app 目录，对应的翻译就会被自动使用。

不过更好的使用方式是在你的项目中使用 composer require codeigniter4/translations ，因为翻译目录自动映射之后，这样被翻译过的信息就会自动被使用。

### 5.2.10 在视图文件中使用 PHP 替代语法

如果你不使用模板引擎来简化输出，那么意味着你将在视图文件中使用纯 PHP 语法。为了精简视图文件中的 PHP 代码同时增强代码的可读性，建议你在写控制结构和 echo 语句时使用 PHP 的替代语法。如果你对这个语法还不熟悉，下面将介绍如何通过这个语法来消除你代码中的大括号和 echo 语句。

#### Echo 的替代语法

通常来说，你在输出或打印一个变量的时候会这样做:

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

而使用替代语法，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简化格式。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，结束括号被 endforeach 取而代之。上面列举出的那些控制结构都有相似的结束标志: endif, endfor, endforeach 和 endwhile。

同时要注意的是，每个结构分支后面都要跟一个冒号 (除了最后一个)，而不是分号, 这很重要!

这是另外一个样例，使用了 if/elseif/else，注意看分支语句后的冒号:

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

        <h3>Hi Sally</h3>

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

        <h3>Hi Joe</h3>

<?php else : ?>

        <h3>Hi unknown user</h3>

<?php endif ?>
```

# 章节 6

# 数据库

## 6.1 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 SQL 语句以及查询构造器模式。数据库方法的语法简单明了。

### 6.1.1 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。更完整的信息请参考每个函数/类单独的介绍页面。

#### 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:

```
$db = \Config\Database::connect();
```

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 = $db->query('SELECT name, title, email FROM my_table');
$results = $query->getResult();
```

(下页继续)

(续上页)

```
foreach ($results as $row)
{
        echo $row->title;
        echo $row->name;
        echo $row->email;
}

echo 'Total Results: ' . count($results);
```

上面的 getResult() 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 = $db->query('SELECT name, title, email FROM my_table');
$results = $query->getResultArray();

foreach ($results as $row)
{
        echo $row['title'];
        echo $row['name'];
        echo $row['email'];
}
```

上面的 getResultArray() 函数返回一个 **二维数组**。例如：$row[‘title’]

### 单结果标准查询（对象形式）

```
$query = $db->query('SELECT name FROM my_table LIMIT 1');
$row = $query->getRow();
echo $row->name;
```

上面的 getRow() 函数返回一个 **对象**。例如：$row->name

### 单结果标准查询（数组形式）

```
$query = $db->query('SELECT name FROM my_table LIMIT 1');
$row = $query->getRowArray();
echo $row['name'];
```

上面的 getRowArray() 函数返回一个 **一维数组**。例如：$row[‘name’]

#### 标准插入

```
$sql = "INSERT INTO mytable (title, name) VALUES (".$db->escape($title).
↪", ".$db->escape($name).")";
$db->query($sql);
echo $db->getAffectedRows();
```

#### 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 = $db->table('table_name')->get();

foreach ($query->getResult() as $row)
{
        echo $row->title;
}
```

上面的 get() 函数从给定的表中查询出所有结果。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
);

$db->table('mytable')->insert($data);  // 生成: INSERT INTO mytable␣
↪(title, name, date) VALUES ('{$title}', '{$name}', '{$date}')
```

### 6.1.2 数据库配置

- 配置.env 文件
- 参数解释:

CodeIgniter 有一个用来保存数据库配置的文件（用户名，密码，数据库名等），这个配置文件位于 application/Config/Database.php。你也可以在.env 文件里配置数据库连接参数。下面来看看详细配置信息。

配置信息是一个数组，存储在类的属性里面，原型如下:

```
public $default = [
        'DSN'   => '',
        'hostname' => 'localhost',
        'username' => 'root',
        'password' => '',
        'database' => 'database_name',
        'DBDriver' => 'MySQLi',
        'DBPrefix' => '',
        'pConnect' => TRUE,
        'DBDebug'  => TRUE,
        'cacheOn'  => FALSE,
        'cacheDir' => '',
        'charset'  => 'utf8',
        'DBCollat' => 'utf8_general_ci',
        'swapPre'  => '',
        'encrypt'  => FALSE,
        'compress' => FALSE,
        'strictOn' => FALSE,
        'failover' => array(),
];
```

类的属性名称就是连接名称，并且在连接时可以作为指定配置组名称使用。

有些数据库驱动（例如：PDO，PostgreSQL，Oracle，ODBC）可能需要提供完整的 DNS 信息。在这种情况下，你需要使用 DNS 配置参数，就像使用该驱动的原生 PHP 扩展一样，例如:

```
// PDO
$default['DSN'] = 'pgsql:host=localhost;port=5432;dbname=database_name';

// Oracle
$default['DSN'] = '//localhost/XE';
```

**注解:** 如果你没有指定 DNS 驱动需要的参数信息，CodeIgniter 将使用你提供的其它配置信息自动构造它。

**注解:** 如果你提供了一个 DNS 参数，但是缺少了某些配置（例如：数据库的字符集），若该配置存在在其它的配置项中，CodeIgniter 将自动在 DNS 上附加上该配置。

当主数据库由于某些原因无法连接时，你可以配置多个灾备数据库。例如可以像下面这样为一个连接配置灾备数据库:

```
$default['failover'] = [
                [
                        'hostname' => 'localhost1',
```

(下页继续)

(续上页)

```
                'username' => '',
                'password' => '',
                'database' => '',
                'DBDriver' => 'MySQLi',
                'DBPrefix' => '',
                'pConnect' => TRUE,
                'DBDebug'  => TRUE,
                'cacheOn'  => FALSE,
                'cacheDir' => '',
                'charset'  => 'utf8',
                'DBCollat' => 'utf8_general_ci',
                'swapPre'  => '',
                'encrypt'  => FALSE,
                'compress' => FALSE,
                'strictOn' => FALSE
        ],
        [
                'hostname' => 'localhost2',
                'username' => '',
                'password' => '',
                'database' => '',
                'DBDriver' => 'MySQLi',
                'DBPrefix' => '',
                'pConnect' => TRUE,
                'DBDebug'  => TRUE,
                'cacheOn'  => FALSE,
                'cacheDir' => '',
                'charset'  => 'utf8',
                'DBCollat' => 'utf8_general_ci',
                'swapPre'  => '',
                'encrypt'  => FALSE,
                'compress' => FALSE,
                'strictOn' => FALSE
        ]
    ];
```

你可以指定任意多个灾备数据库配置。

你可以选择性地存储多组连接信息。例如，在一个安装实例里面运行多个环境（开发、生产、测试等），你可以为每个环境配置连接组，然后在组之间进行切换。举个例子：若要设置一个‘test’环境，你可以这么做:

```
public $test = [
        'DSN'   => '',
        'hostname' => 'localhost',
        'username' => 'root',
        'password' => '',
```

(下页继续)

(续上页)

```
        'database' => 'database_name',
        'DBDriver' => 'MySQLi',
        'DBPrefix' => '',
        'pConnect' => TRUE,
        'DBDebug'  => TRUE,
        'cacheOn'  => FALSE,
        'cacheDir' => '',
        'charset'  => 'utf8',
        'DBCollat' => 'utf8_general_ci',
        'swapPre'  => '',
        'compress' => FALSE,
        'encrypt'  => FALSE,
        'strictOn' => FALSE,
        'failover' => array()
);
```

然后，修改该配置文件中的属性值，告知系统使用该组信息:

```
$defaultGroup = 'test';
```

**注解:** 组名称‘test’是任意的。它可以是你想要的任意名称。默认情况下，主连接使用‘default’这个名称，但你也可以起一个与你项目更加相关的名称。

你可以修改配置文件里面类的构造函数，让它自动检测运行环境并将‘defaultGroup’更新为正确的值:

```
class Database
{
    public $development = [...];
    public $test        = [...];
    public $production  = [...];

        public function __construct()
        {
                $this->defaultGroup = ENVIRONMENT;
        }
}
```

### 配置.env 文件

你也可以将当前服务器的数据库配置保存到 .env 文件中。你只需要在默认配置组中输入你想要变更的值。该值在 default 组中的格式为:

```
database.default.username = 'root';
database.default.password = '';
database.default.database = 'ci4';
```

其它信息

**参数解释:**

| 配置名 | 描述 |
| --- | --- |
| **dsn** | DNS 连接字符串（该字符串包含了连接数据库的全部配置信息） |
| **hostname** | 数据库的主机名，通常为本机的‘localhost’ |
| **username** | 连接数据库的用户名 |
| **password** | 连接数据库的密码 |
| **database** | 需要连接的数据库名 |
| **DBDriver** | 数据库类型，如：MySQLi、Postgre 等。大小写必须与驱动名匹配 |
| **DBPrefix** | 当使用查询构造器 查询时，可以选择性的为表加个前缀，它允许多个 CodeIgniter 程序共用一个数据库 |
| **pConnect** | TRUE/FALSE (boolean) - 是否使用持续连接 |
| **DBDebug** | TRUE/FALSE (boolean) - 是否显示数据库错误信息 |
| **cacheOn** | TRUE/FALSE (boolean) - 是否开启数据库查询缓存 |
| **cacheDir** | 数据库查询缓存目录，服务器绝对路径 |
| **charset** | 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集规则<br>**注解:** 只用于‘MySQLi’数据库驱动 |
| **swapPre** | 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** | 默认数据库模式为‘public’，用于 PostgreSQL 和 ODBC 驱动 |
| **encrypt** | 是否是用加密连接<br>• ‘sqlsrv’和‘pdo/sqlsrv’驱动接受 TRUE/FALSE<br>• ‘MySQLi’和‘pdo/mysql’驱动接受一个数组，选项如下:<br>• ‘ssl_key’- 私钥文件存放路径<br>• ‘ssl_cert’- 公钥证书文件存放路径<br>• ‘ssl_ca’- CA 证书授权文件路径<br>• ‘ssl_capath’- PEM 格式的受信任 CA 证书存放目录<br>• ‘ssl_cipher’- 允许使用的加密算法列表，多项用（‘:’）分割<br>• ‘ssl_verify’- TRUE/FALSE; 是否验证服务器的证书 (仅限 MySQLi) |
| **compress** | 是否使用客户端压缩协议（只用于 MySQL） |
| **strictOn** | TRUE/FALSE (boolean) - 是否强制使用“Strict Mode”连接。在程序开发时，使 |

**注解：** 根据你使用的数据库平台（MySQL、PostgreSQL 等）不是所有参数都要配置。例如，当你使用 SQLite 时，你无需指定用户名和密码，数据库名称是你的数据库文件路径。以上内容假设你使用的是 MySQL 数据库。

### 6.1.3 连接你的数据库

你可以在任意你需要的方法中添加以下代码来连接你的数据库，或在类的构造函数中添加这段代码让其在类里全局可用。

```
$db = \Config\Database::connect();
```

如果上面的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默认配置组来连接数据库，对于大多数人而言，这是首选的方案。

有一个简便的、纯粹是封装上一段代码的方法，亦可以让你便捷的连接数据库:

```
$db = db_connect();
```

#### 可用的参数

1. 数据库组名，一个必须与配置类的属性名匹配的字符串。默认值为 $config->defaultGroup;
2. TRUE/FALSE (boolean). 是否返回共享连接（参考下文的连接多个数据库）。

#### 手动连接数据库

这个函数的第一个参数是 **可选的**，用来从你的配置文件中选取某个配置组（建立连接）。例如:

从配置文件中选择一个特定的配置组，你可以这样做:

```
$db = \Config\Database::connect('group_name');
```

其中 group_name 是配置文件中配置组的名字。

#### 用多个链接连同一个数据库

默认情况下，connect() 方法每次返回数据库连接的同一实例。若你需要一个单独的连接到相同数据库，使用 false 作为第二个参数:

```
$db = \Config\Database::connect('group_name', false);
```

### 连接多个数据库

如果你需要同时连接到多个不同的数据库，你可以这样做:

```
$db1 = \Config\Database::connect('group_one');
$db = \Config\Database::connect('group_two');
```

注意: 将“group_one”和“group_two”修改为你想要连接的配置组名称

---

**注解:** 如果只是在同一连接上使用不同的数据库，你不需要创建单独的数据库配置。当你需要时，可以切换到不同的数据库，例如:

$db->dbSelect($database2_name);

---

### 使用自定义配置连接数据库

你可以传入一个数据库配置数组参数替代配置组名称，以此获得一个自定义的数据库连接。数组的格式必须与数据库配置文件的配置组格式相同:

```
$custom = [
            'DSN'      => '',
            'hostname' => 'localhost',
            'username' => '',
            'password' => '',
            'database' => '',
            'DBDriver' => 'MySQLi',
            'DBPrefix' => '',
            'pConnect' => false,
            'DBDebug'  => (ENVIRONMENT !== 'production'),
            'cacheOn'  => false,
            'cacheDir' => '',
            'charset'  => 'utf8',
            'DBCollat' => 'utf8_general_ci',
            'swapPre'  => '',
            'encrypt'  => false,
            'compress' => false,
            'strictOn' => false,
            'failover' => [],
            'port'     => 3306,
    ];
$db = \Config\Database::connect($custom);
```

#### 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（例如处理图像），若超过了数据库的超时值，你应该考虑在执行后续查询前先调用 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重新建立连接。

---

**重要：** 若你使用 MySQLi 数据库驱动，reconnect() 方法并不能 ping 通服务器但它可以关闭连接然后再次连接。

---

```
$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，你仍可以显式关闭连接。

```
$db->close();
```

### 6.1.4 执行查询

### 基础查询

#### 执行常规查询

使用 **query** 方法提交一个查询:

```
$db->query('YOUR QUERY HERE');
```

执行“读取”类型查询时，query() 方法返回一个数据库查询结果 **对象**，如何使用参考显示查询结果；执行“写入”类查询时，只返回 TRUE 或 FALSE，表示执行成功或失败。检索数据时，你通常需要自行编写查询语句，例如:

```
$query = $db->query('YOUR QUERY HERE');
```

#### 执行简单查询

**simpleQuery** 方法是 $db->query() 的简化版本。它不会返回查询结果，不记录查询耗时，不会绑定变量，也不保存查询语句（用于调试）。它只是简单的让你执行一个查询语句，可能大多数用户鲜有使用。

它只返回“execute”方法执行的返回值，无关数据库类型。典型的返回值是 TRUE/FALSE，当执行类型是写入型时，它表示写操作的成功或失败（插入、删除或修改，实际就如此使用）；执行读取类型时，表示能否成功获取查询结果资源/对象。

```
if ($db->simpleQuery('YOUR QUERY'))
{
        echo "Success!";
}
else
{
        echo "Query failed!";
}
```

---

**注解:** PostgreSQL 的 pg_exec() 方法 (举例) 执行成功时总是返回一个资源值，即使执行写入型查询也是如此。所以当你判断布尔值时要切记此点。

---

#### 手动指定数据表前缀

当你配置了数据库的表前缀，执行原生 SQL 查询等类似操作，应当给数据表增加前缀，你可以使用如下操作:

```
$db->prefixTable('tablename'); // 输出 prefix_tablename
```

出于某些原因，你想以编程的方式修改表前缀，且不想创建新数据库连接时，可以这样做:

```
$db->setPrefix('newprefix');
$db->prefixTable('tablename'); // 输出 newprefix_tablename
```

你可以用此方法随时随地获取当前的表前缀:

```
$DBPrefix = $db->getPrefix();
```

### 保护标识符

许多数据库建议保护表名和字段名 - 比如 MySQL 使用反引号。**查询构造器会自动保护它们**, 但如果你需要手动保护标识符时，可以这么做:

```
$db->protectIdentifiers('table_name');
```

---

**重要:** 尽管查询构造器会尽可能且适当的引用你所需的字段名和表名，但请注意它不适用于恶意用户输入，勿将其用于未处理的用户数据。

---

当你在数据库配置文件里配有表前缀，这个方法还能给表加上前缀，开启这个功能请在第二个参数填写 TRUE（布尔值）:

```
$db->protectIdentifiers('table_name', TRUE);
```

### 查询转义

执行数据库查询前做数据转义是又好又安全的实践，CodeIgniter 有三种方法帮到你:

1. **$db->escape()** 这个方法会判断数据类型，对字符串数据做转义，它也会自动给数据加单引号，你无需额外处理:

   ```
   $sql = "INSERT INTO table (title) VALUES(".$db->escape($title).")";
   ```

2. **$db->escapeString()** 这个方法对传入数据做强制转义，且无关类型，多数时候你会用上面的方法而非这个。此方法使用举例:

   ```
   $sql = "INSERT INTO table (title) VALUES('".$db->escapeString(
   ↪$title)."')";
   ```

3. **$db->escapeLikeString() 这个方法用于 LIKE 条件字符串转义，** 以确保 LIKE 的通配符（‘%’，‘_’）也能正确的转义。

```
$search = '20% raise';
$sql = "SELECT id FROM table WHERE column LIKE '%" .
$db->escapeLikeString($search)."%' ESCAPE '!'";
```

**重要：** escapeLikeString() 方法使用 '!' (感叹号) 转义 *LIKE* 条件中的特殊字符，因为这个方法只转义引号里的字符串，它不能自动添加 ESCAPE '!' 条件，因此你必须手动添加。

### 查询绑定

绑定可以让你用简单的查询语法，让系统将查询语句合在一起，考虑下这个例子:

```
$sql = "SELECT * FROM some_table WHERE id = ? AND status = ? AND author␣
↪= ?";
$db->query($sql, [3, 'live', 'Rick']);
```

查询语句的问号会被方法第二个参数的数组顺次替换。

使用 IN 条件时，绑定用多维数组搞定集合:

```
$sql = "SELECT * FROM some_table WHERE id IN ? AND status = ? AND author␣
↪= ?";
$db->query($sql, [[3, 6], 'live', 'Rick']);
```

转化后的语句是:

```
SELECT * FROM some_table WHERE id IN (3,6) AND status = 'live' AND␣
↪author = 'Rick'
```

使用绑定的第二个好处是，它会自动转义输入值，生成安全的查询语句。你无需记住要手动转义数据这件事 - 引擎会自动帮你完成。

### 命名绑定

你可以用命名绑定，而不用问号标记绑定值的位置，从而允许在查询中使用键名匹配占位符:

```
$sql = "SELECT * FROM some_table WHERE id = :id: AND status = :status:␣
↪AND author = :name:";
$db->query($sql, [
        'id'     => 3,
        'status' => 'live',
        'name'   => 'Rick'
]);
```

**注解：** 查询语句中的每个键名前后【必须】加英文冒号。

### 错误处理

**$db->error();**

如果你需要获取最近一次发生的数据库报错，error() 方法会返回一个数组，包含错误号和错误信息，来看下用例:

```
if ( ! $db->simpleQuery('SELECT `example_field` FROM `example_table`'))
{
        $error = $db->error(); // Has keys 'code' and 'message'
}
```

### 预编译查询

大部分数据库引擎支持某种形式的预编译语句，使你仅做一次预编译，然后在新数据集上多次查询。它消除了 SQL 注入的可能性，因为数据是以另一种形式传给数据库而非查询语句。当你需要多次执行相同查询时，它也相当快速。然而，若你想应用于所有查询，这会极大影响性能，因为它通常要访问数据库两次。由于查询构造器和数据库连接已经处理了转义数据，所以，安全方面已经为你解决了，但有时候，你也需要通过预编译语句或预编译查询来优化查询。

### 编译查询语句

使用 prepare() 方法可轻松完成编译，它有一个参数，是函数闭包，返回一个查询对象。查询对象由任一“最终”类型的查询自动生成，包括 **insert** , **update** , **delete** , **replace** 和 **get** 。使用查询构造器执行查询可以最轻松地处理此问题。查询实际没有执行，传入的值不重要也不会被处理，仅做占位使用。这样会返回一个预编译查询对象:

```
$pQuery = $db->prepare(function($db)
{
    return $db->table('user')
              ->insert([
                   'name'    => 'x',
                   'email'   => 'y',
                   'country' => 'US'
              ]);
});
```

如果你不想使用查询构造器，你可以手动创建查询对象，用问号做占位符:

```
use CodeIgniter\Database\Query;

$pQuery = $db->prepare(function($db)
{
    $sql = "INSERT INTO user (name, email, country) VALUES (?, ?, ?)";
```

(下页继续)

(续上页)

```
    return (new Query($db))->setQuery($sql);
});
```

如果数据库要求在预编译阶段提供选项数组，可以将数组放到第二个参数:

```
use CodeIgniter\Database\Query;

$pQuery = $db->prepare(function($db)
{
    $sql = "INSERT INTO user (name, email, country) VALUES (?, ?, ?)";

    return (new Query($db))->setQuery($sql);
}, $options);
```

### 执行预编译查询

一旦你有了一个预编译查询，你可以使用 execute() 方法真正的执行查询。你可以传递多个你需要的查询参数，参数的个数必须与占位符个数相同，参数的顺序也要与原始占位符保持一致:

```
// 编译查询语句
$pQuery = $db->prepare(function($db)
{
    return $db->table('user')
              ->insert([
                  'name'    => 'x',
                  'email'   => 'y',
                  'country' => 'US'
              ]);
});

// 准备数据
$name    = 'John Doe';
$email   = 'j.doe@example.com';
$country = 'US';

// 执行查询
$results = $pQuery->execute($name, $email, $country);
```

这会返回标准的结果集.

### 其他方法

除了上述两个主要方法，预编译查询还有以下方法可用:

**close()**

虽然 PHP 在（自动）关闭所有打开的查询资源时做的非常好，但手动关闭执行完的预编译查询同样也是好的主意:

```
$pQuery->close();
```

**getQueryString()**

返回预编译查询的字符串。

**hasError()**

返回布尔值 true/false ，表示调用最近一次是否有执行错误。

**getErrorCode() getErrorMessage()**

如果有报错，可以用这两个方法获取错误号和错误信息。

### 使用查询对象

在内部，所有查询的处理和存储都在 CodeIgniterDatabaseQuery 的实例中进行。这个类负责绑定参数、也做预编译查询、还能保存查询时的性能数据。

**getLastQuery()**

当你需要获取最近一次的查询对象，请使用 getLastQuery() 方法:

```
$query = $db->getLastQuery();
echo (string)$query;
```

### 查询类

每个查询对象都保存了此次查询的一些信息，它有部分被时间线功能使用，但你也可以使用（译者注：此处时间线指数据库执行 SQL 过程，记录它们方便调试和优化性能）。

**getQuery()**

返回各种编译构造之后的最终查询语句，也就是发送到数据库执行的语句:

```
$sql = $query->getQuery();
```

将查询对象做字符串转换也能获得相同的值:

```
$sql = (string)$query;
```

**getOriginalQuery()**

返回初始传入对象里的 SQL 语句，没有任何绑定或前缀修饰等等:

```
$sql = $query->getOriginalQuery();
```

#### hasError()

如果执行时有任何错误，这个方法将返回 true:

```
if ($query->hasError())
{
        echo 'Code: '. $query->getErrorCode();
        echo 'Error: '. $query->getErrorMessage();
}
```

#### isWriteType()

如果当前查询是写入型 (例如 INSERT, UPDATE, DELETE, 等)，此方法返回 true:

```
if ($query->isWriteType())
{
        ... do something
}
```

#### swapPrefix()

替换最终执行的 SQL 里的表前缀，第一个参数是原始你想替换的前缀，第二个参数是替换之后你想要的前缀:

```
$sql = $query->swapPrefix('ci3_', 'ci4_');
```

#### getStartTime()

获取查询执行时间，以秒为单位，精确到毫秒级:

```
$microtime = $query->getStartTime();
```

#### getDuration()

返回执行查询的时长（秒），浮点数，精确到毫秒:

```
$microtime = $query->getDuration();
```

### 6.1.5 生成查询结果

有几种方式生成查询结果:

### 结果数组

#### getResult()

这个方法返回 **对象数组**类型的查询结果，或在失败时返回 **一个空数组**。典型的用法是使用 foreach 循环，像这样:

```
$query = $db->query("YOUR QUERY");

foreach ($query->getResult() as $row)
{
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

上面的方法是 getResultObject() 的别名。

如果你想返回一个二维数组，可以给第一个参数传字符串 ‘array’

```
$query = $db->query("YOUR QUERY");

foreach ($query->getResult('array') as $row)
{
    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

上面的方法是 getResultArray() 的别名。

你也可以传字符串参数到 getResult() 方法，表示要为每个结果对象实例化的类

```
$query = $db->query("SELECT * FROM users;");

foreach ($query->getResult('User') as $user)
{
    echo $user->name; // 获取属性
    echo $user->reverseName(); // 或访问 'User' 类定义的方法
}
```

上面的方法是 getCustomResultObject() 的别名。

#### getResultArray()

这个方法返回一个纯数组的查询结果，查无结果时为空数组。典型的用法是使用 foreach 循环，像这样:

```
$query = $db->query("YOUR QUERY");

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

(下页继续)

(续上页)

```
{
    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

### 单行结果

#### getRow()

这个方法返回一个单行结果，如果你的查询有多行结果，它仅返回第一条。返回结果是一个 **对象**。使用示例:

```
$query = $db->query("YOUR QUERY");

$row = $query->getRow();

if (isset($row))
{
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

如果你想返回指定行的结果，可以在第一个参数里提供这个数字:

```
$row = $query->getRow(5);
```

你也可以传一个字符串到第二个参数，表示该结果实例化的对象类:

```
$query = $db->query("SELECT * FROM users LIMIT 1;");
$row = $query->getRow(0, 'User');

echo $row->name; // 获取属性
echo $row->reverse_name(); // 或访问 'User' 类定义的方法
```

#### getRowArray()

这个与上面的 row() 方法基本相同，区别是它返回的是一个数组。示例:

```
$query = $db->query("YOUR QUERY");

$row = $query->getRowArray();

if (isset($row))
{
    echo $row['title'];
```

(下页继续)

(续上页)

```
    echo $row['name'];
    echo $row['body'];
}
```

如果你想返回指定行的结果，可以在第一个参数里提供这个数字:

```
$row = $query->getRowArray(5);
```

另外，你可以用这些方法在结果集里做前进/后退/首行/尾行的游标操作:

**$row = $query->getFirstRow()**
**$row = $query->getLastRow()**
**$row = $query->getNextRow()**
**$row = $query->getPreviousRow()**

默认他们返回一个对象，除非第一个参数是字符串“array”才会返回数组:

**$row = $query->getFirstRow( ‘array’ )**
**$row = $query->getLastRow( ‘array’ )**
**$row = $query->getNextRow( ‘array’ )**
**$row = $query->getPreviousRow( ‘array’ )**

---

**注解:** 以上所有方法都会把整个查询结果载入内存（预加载）。请使用 `getUnbufferedRow()` 方法处理大型结果集。

---

### getUnbufferedRow()

这个方法返回单个结果，不会像 `row()` 把整个结果预加载到内存里。如果你的查询结果多于一个，它返回当前行并将内部数据指针向前移动。

```
$query = $db->query("YOUR QUERY");

while ($row = $query->getUnbufferedRow())
{
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

你可以选择性的传参‘object’(默认) 或‘array’来指定返回数据的类型:

```
$query->getUnbufferedRow();          // 对象
$query->getUnbufferedRow('object'); // 对象
$query->getUnbufferedRow('array');  // 关联数组
```

### 自定义结果对象

**你可以用一个自定义的类实例作为返回结果，代替原来的 stdClass 对象或数组，**
getResult() 和 getResultArray() 允许如此操作。

如果该类（文件）尚未加载到内存，自动加载器会尝试载入它。对象的属性值会设置为数据库的返回数据，如果是非公开属性，你需要提供一个 __set() 方法以允许他们被赋予值。

示例:

```
class User
{
        public $id;
        public $email;
        public $username;

        protected $last_login;

        public function lastLogin($format)
        {
                return $this->lastLogin->format($format);
        }

        public function __set($name, $value)
        {
                if ($name === 'lastLogin')
                {
                        $this->lastLogin = DateTime::createFromFormat('U
↪', $value);
                }
        }

        public function __get($name)
        {
                if (isset($this->$name))
                {
                        return $this->$name;
                }
        }
}
```

除了下面列出的两个方法之外，这些方法也可以指定类名返回类实例的结果集: getFirstRow(), getLastRow(), getNextRow() 和 getPreviousRow() 。

#### getCustomResultObject()

以要求的类实例数组的形式返回整个结果集。唯一的参数是要实例化的类的名称。

示例:

```
$query = $db->query("YOUR QUERY");

$rows = $query->getCustomResultObject('User');

foreach ($rows as $row)
{
        echo $row->id;
        echo $row->email;
        echo $row->last_login('Y-m-d');
}
```

#### getCustomRowObject()

以要求的类实例形式返回单个结果，第一个参数是它在结果集里的序号，第二个参数是要实例化的类的名称。

示例:

```
$query = $db->query("YOUR QUERY");

$row = $query->getCustomRowObject(0, 'User');

if (isset($row))
{
        echo $row->email;                    // 获取属性
        echo $row->last_login('Y-m-d');      // 或访问 'User' 类定义的方法
}
```

你也可以用 getRow() 方法达到相同效果。

示例:

```
$row = $query->getCustomRowObject(0, 'User');
```

### 结果处理辅助方法

#### getFieldCount()

返回查询结果的字段个数（列数），确保你是使用查询结果对象调用此方法:

```
$query = $db->query('SELECT * FROM my_table');

echo $query->getFieldCount();
```

#### getFieldNames()

返回查询结果的字段名（列名）的数组，确保你是使用查询结果对象调用此方法:

```
$query = $db->query('SELECT * FROM my_table');

    echo $query->getFieldNames();
```

### freeResult()

它会释放查询结果占用的内存并删除资源 ID。通常 PHP 会在脚本结束时自动释放内存，然而，如果你在某个脚本里执行了很多查询，你也许想处理完每个查询后即刻释放内存，以此减少内存消耗。

举例:

```
$query = $thisdb->query('SELECT title FROM my_table');

foreach ($query->getResult() as $row)
{
        echo $row->title;
}

$query->freeResult();  // $query 的结果对象不再可用

$query2 = $db->query('SELECT name FROM some_table');

$row = $query2->getRow();
echo $row->name;
$query2->freeResult(); // $query2 的结果对象不再可用
```

### dataSeek()

该方法设置一个内部指针，用来获取下一个结果行，它仅和 getUnbufferedRow() 一起使用才有作用。

它接受一个正整数值，默认是 0，返回 TRUE 表示成功，FALSE 表示失败。

```
$query = $db->query('SELECT `field_name` FROM `table_name`');
$query->dataSeek(5); // Skip the first 5 rows
$row = $query->getUnbufferedRow();
```

---

**注解:** 不是所有数据库驱动支持这个特性，（不支持的）会返回 FALSE。最值得注意的是 - 你无法在 PDO 中使用它。

---

## 类库参考

CodeIgniter\Database\BaseResult

getResult([$type = 'object'])

参数

- **$type** (*string*) – 要求的结果类型 - array, object, 或类名

**返回** 包含查询到的行的数组

**返回类型** array

它是这几种方法的包装：`getResultArray()`, `getResultObject()` 和 `getCustomResultObject()` 。

用法: 详见结果数组.

`getResultArray()`

**返回** 包含查询到的行的数组

**返回类型** array

返回查询结果行的数组，每行都是关联数组。

用法: 详见结果数组.

`getResultObject()`

**返回** 包含查询到的行的数组

**返回类型** array

返回查询结果行的数组，每行都是 `stdClass` 类的实例。

用法: 详见结果数组.

`getCustomResultObject(`*$class_name*`)`

**参数**

- **$class_name** (*string*) – 结果行的类实例名

**返回** 包含查询到的行的数组

**返回类型** array

返回查询结果行的数组，每行都是指定类的实例。

`getRow(`*$n = 0*[, *$type = 'object'*]`)`

**参数**

- **$n** (*int*) – 想要返回的结果行的序号
- **$type** (*string*) – 要求的结果类型 - array, object, 或类名

**返回** 要求的行数据，不存在时返回 NULL

**返回类型** mixed

它是这几种方法的包装：`getRowArray()`, `getRowObject()` 和 `getCustomRowObject()` 。

用法: 详见单行结果.

`getUnbufferedRow(`*$type = 'object'*`)`

**参数**

- **$type** (*string*) – 要求的结果类型 - array, object, 或类名

**返回** 结果集的下一行，不存在时返回 NULL

**返回类型** mixed

按要求的格式返回结果集的下一行。

用法: 详见单行结果.

`getRowArray(`*$n = 0*`)`

**参数**
- **$n** (*int*) – 想要返回的结果行的序号

**返回** 要求的行数据，不存在时返回 NULL
**返回类型** array

返回结果行，格式为关联数组。

用法: 详见单行结果.

`getRowObject`([*$n = 0*])
**参数**
- **$n** (*int*) – 想要返回的结果行的序号

**返回** 要求的行数据，不存在时返回 NULL
**返回类型** stdClass

返回结果行，格式为 `stdClass` 的类实例。

用法: 详见单行结果.

`getCustomRowObject`(*$n*, *$type*)
**参数**
- **$n** (*int*) – 想要返回的结果行的序号
- **$class_name** (*string*) – 结果行的类实例名

**返回** 要求的行数据，不存在时返回 NULL
**返回类型** $type

返回结果行，格式为要求的的类实例。

`dataSeek`([*$n = 0*])
**参数**
- **$n** (*int*) – 即将返回的结果行的序号

**返回** TRUE 表示成功，FALSE 表示失败
**返回类型** bool

移动结果集的内部指针到指定位置。

用法: 详见结果处理辅助方法.

`setRow`(*$key*[, *$value = NULL*])
**参数**
- **$key** (*mixed*) – 列名或键值数组
- **$value** (*mixed*) – 分配给列的值，$key 是单个字段名

**返回类型** void

为特定列分配值。

`getNextRow`([*$type = 'object'*])
**参数**
- **$type** (*string*) – 要求的结果类型 - array, object, 或类名

**返回** 结果集的下一行，不存在时返回 NULL
**返回类型** mixed

返回结果集的下一行。

`getPreviousRow`([*$type = 'object'*])
**参数**

- **$type** (*string*) – 要求的结果类型 - array, object, 或类名

**返回** 结果集的上一行，不存在时返回 NULL
**返回类型** mixed

返回结果集的上一行。

`getFirstRow`([*$type = 'object'*])

**参数**

- **$type** (*string*) – 要求的结果类型 - array, object, 或类名

**返回** 结果集的第一行，不存在时返回 NULL
**返回类型** mixed

返回结果集的第一行。

`getLastRow`([*$type = 'object'*])

**参数**

- **$type** (*string*) – 要求的结果类型 - array, object, 或类名

**返回** 结果集的最后一行，不存在时返回 NULL
**返回类型** mixed

返回结果集的最后一行。

`getFieldCount`()

**返回** 结果集中字段的个数
**返回类型** int

返回结果集中字段的个数。

用法: 详见**'结果处理辅助函数 '**__.

`getFieldNames`()

**returns** 列名称的数组

**rtype** array

返回一个包含结果集中字段名的数组。

`getFieldData`()

**返回** 包含字段元数据的数组
**返回类型** array

生成一个包含字段元数据的 `stdClass` 对象的数组。

`freeResult`()

**返回类型** void

释放一个结果集。

用法: 详见**'结果处理辅助函数 '**__.

### 6.1.6 查询语句辅助函数

### 从语句的执行中获取信息

#### $db->insertID()

当执行插入语句时，插入行的 ID

---

**注解：** 如果使用 PDO 驱动来操作 PostgreSQL，或使用 Interbase 驱动，该函数需要一个 $name 参数，用于在查找插入 ID 时使用正确的顺序（译者注：v4.0.3 代码里并不处理这个入参，可能是英文手册勘误 2020-07-14）。

---

#### $db->affectedRows()

执行”写入”类型的语句（insert，update 等）时返回有多少行受影响

---

**注解：** 在 MYSQL 中“DELETE FROM TABLE”会返回 0 行受影响。所以数据库类做了一个小的 hack，使其可以返回受影响的正确行数。这个功能默认是启用的，不过可以通过修改数据库驱动文件关闭。

---

#### $db->getLastQuery()

返回最近一次执行的查询语句（查询语句字符串，而非查询结果）

### 关于数据库的信息

#### $db->countAll()

帮你确认一张数据表的总共行数，第一个参数是表名。这也是查询构建器的一部分，例如:

```
echo $db->table('my_table')->countAll();

// 输出一个整数，例如 25
```

#### $db->getPlatform()

输出当前运行的数据库平台 (MySQL, MS SQL, Postgres 等):

```
echo $db->getPlatform();
```

#### $db->getVersion()

输出当前运行的数据库版本:

```
echo $db->getVersion();
```

### 6.1.7 查询构造器类

CodeIgniter 提供了查询构造器类，它允许你用较少的代码量获取数据库的信息、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 不要求每个数据表有一个类文件，它使用了一种更简单的接口。

除了简单，使用查询构造器的主要好处是可以让你创建跨数据库的应用程序，因为查询语句是由每种数据库适配器生成的。它也允许用于更安全的查询，因为系统会自动转义传入数据。



#### 加载查询构造器

查询构造器通过数据库连接对象的 `table()` 方法加载，这会设置查询语句 `FROM` 的部分并且返回一个查询构造器的新实例:

```
$db      = \Config\Database::connect();
$builder = $db->table('users');
```

查询构造器仅在你明确请求类时才加载到内存中，因此默认不使用（消耗）任何资源。

#### 选择数据

下面的方法用来构建 SQL **SELECT** 语句。

**$builder->get()**

执行选择查询并返回结果，可用于获取一个表的所有记录:

```
$builder = $db->table('mytable');
$query   = $builder->get();  // 生成: SELECT * FROM mytable
```

第一个和第二个参数用于设置 limit 和 offset 子句:

```
$query = $builder->get(10, 20);

// 执行: SELECT * FROM mytable LIMIT 20, 10
// (在 MySQL 里的情况，其他数据库的语法略有不同)
```

你应该已经注意到了，上面方法的结果赋值给了一个 $query 变量，我们可以用它输出查询结果:

```
$query = $builder->get();

foreach ($query->getResult() as $row)
{
        echo $row->title;
}
```

请访问结果方法 页面获得结果生成的完整论述。

**$builder->getCompiledSelect()**

和 **$builder->get()** 方法一样编译选择查询但是并不执行，此方法只是将 SQL 查询语句作为字符串返回。

例如:

```
$sql = $builder->getCompiledSelect();
echo $sql;

// 输出字符串: SELECT * FROM mytable
```

第一个参数使你能设置是否重置查询构造器（默认重置，就像使用 *$builder->get()* 时一样):

```
echo $builder->limit(10,20)->getCompiledSelect(false);

// 输出字符串: SELECT * FROM mytable LIMIT 20, 10
// (在 MySQL 里的情况，其他数据库的语法略有不同)

echo $builder->select('title, content, date')->getCompiledSelect();

// 输出字符串: SELECT title, content, date FROM mytable LIMIT 20, 10
```

最值得注意的是，上例第二个查询并没有用到 **$builder->from()** 方法，也没有为查询指定表名参数。因为这个查询没有被可重置值的 **$builder->get()** 方法执行，或是使用 **$builder->resetQuery()** 方法直接重置。

**$builder->getWhere()**

与 get() 函数相同，只是它允许你用第一个参数中添加“where”子句，而不是使用 db->where() 功能:

```
$query = $builder->getWhere(['id' => $id], $limit, $offset);
```

请阅读下面 *where* 方法获得更多信息。

**$builder->select()**

允许你编写查询的 SELECT 部分:

```
$builder->select('title, content, date');
$query = $builder->get();

// 执行: SELECT title, content, date FROM mytable
```

**注解:** 如果要从表中选择全部字段 (*) ，不需要使用这个函数。当省略它时，CodeIgniter 假定你希望选择所有字段并自动添加‘SELECT *’。

$builder->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。当你编写复合查询语句时很有用，它不会因为自动转义而搞坏你的语句。

```
$builder->select('(SELECT SUM(payments.amount) FROM payments WHERE␣
↪payments.invoice_id=4) AS amount_paid', FALSE);
$query = $builder->get();
```

**$builder->selectMax()**

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重命名结果字段（可选）。

```
$builder->selectMax('age');
$query = $builder->get();  // 生成: SELECT MAX(age) as age FROM mytable

$builder->selectMax('age', 'member_age');
$query = $builder->get(); // 生成: SELECT MAX(age) as member_age FROM␣
↪mytable
```

**$builder->selectMin()**

该方法用于编写查询语句中的“SELECT MIN(field)”部分，和 selectMax() 一样，你可以使用第二个参数重命名结果字段（可选）。

```
$builder->selectMin('age');
$query = $builder->get(); // 生成: SELECT MIN(age) as age FROM mytable
```

**$builder->selectAvg()**

该方法用于编写查询语句中的“SELECT AVG(field)”部分，和 selectMax() 一样，你可以使用第二个参数重命名结果字段（可选）。

```
$builder->selectAvg('age');
$query = $builder->get(); // 生成: SELECT AVG(age) as age FROM mytable
```

### $builder->selectSum()

该方法用于编写查询语句中的 “SELECT SUM(field)” 部分，和 selectMax() 一样，你可以使用第二个参数重命名结果字段（可选）。

```
$builder->selectSum('age');
$query = $builder->get(); // 生成: SELECT SUM(age) as age FROM mytable
```

### $builder->selectCount()

该方法用于编写查询语句中的 “SELECT COUNT(field)” 部分，和 selectMax() 一样，你可以使用第二个参数重命名结果字段（可选）。

---

**注解:** 该方法在使用 `groupBy()` 时特别有用。用于一般的结果计数详见 `countAll()` 或 `countAllResults()` 。

---

```
$builder->selectCount('age');
$query = $builder->get(); // 生成: SELECT COUNT(age) as age FROM mytable
```

### $builder->from()

该方法用于编写查询语句中的 FROM 子句:

```
$builder->select('title, content, date');
$builder->from('mytable');
$query = $builder->get();  // 生成: 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 $db->table() 中指定。额外调用 from() 将向查询的 FROM 部分添加更多表。

---

### $builder->join()

该方法用于编写查询语句中的 JOIN 子句:

```
$builder->db->table('blog');
$builder->select('*');
$builder->join('comments', 'comments.id = blogs.id');
$query = $builder->get();

// 生成:
// SELECT * FROM blogs JOIN comments ON comments.id = blogs.id
```

如果你的查询有多个连接，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，可选: left，right, outer, inner, left outer 和 right outer 。

```
$builder->join('comments', 'comments.id = blogs.id', 'left');
// 生成: LEFT JOIN comments ON comments.id = blogs.id
```

### 查找具体数据

**$builder->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:** 所有传入数据将会自动转义，生成安全的查询语句。

---

1. **简单的 key/value 方式:**

   ```
   $builder->where('name', $name); // 生成: WHERE name = 'Joe'
   ```

   注意它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:

   ```
   $builder->where('name', $name);
   $builder->where('title', $title);
   $builder->where('status', $status);
   // WHERE name = 'Joe' AND title = 'boss' AND status =
   ↪'active'
   ```

2. **自定义 key/value 方式:**

   你可以在第一个参数中包含一个比较运算符，用来控制比较条件:

   ```
   $builder->where('name !=', $name);
   $builder->where('id <', $id); // 生成: WHERE name != 'Joe'␣
   ↪AND id < 45
   ```

3. **关联数组方式:**

   ```
   $array = ['name' => $name, 'title' => $title, 'status' =>
   ↪$status];
   $builder->where($array);
   // 生成: 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运算符:

   ```
   $array = ['name !=' => $name, 'id <' => $id, 'date >' =>
   ↪$date];
   $builder->where($array);
   ```

4. **自定义字符串：**你可以手动编写子句：

```
$where = "name='Joe' AND status='boss' OR status='active'";
$builder->where($where);
```

$builder->where() 的第三个参数（可选），如果设置为 FALSE，CodeIgniter 将不保护你的表名和字段名。

```
$builder->where('MATCH (field) AGAINST ("value")', NULL, FALSE);
```

1. **子查询：**你可以使用匿名函数生成一个子查询。

```
$builder->where('advance_amount <', function(BaseBuilder
↪$builder) {
    return $builder->select('MAX(advance_amount)', false)->
↪from('orders')->where('id >', 2);
});
// 生成: WHERE "advance_amount" < (SELECT MAX(advance_amount)␣
↪FROM "orders" WHERE "id" > 2)
```

### $builder->orWhere()

这个方法和上面的方法一样，只是多个条件之间使用 OR 进行连接

```
$builder->where('name !=', $name);
$builder->orWhere('id >', $id);  // 生成: WHERE name != 'Joe'␣
↪OR id > 50
```

### $builder->whereIn()

该方法用于生成 WHERE IN（‘item’，‘item’）子句，多个子句之间使用 AND 连接

```
$names = ['Frank', 'Todd', 'James'];
$builder->whereIn('username', $names);
// 生成: WHERE username IN ('Frank', 'Todd', 'James')
```

你可以用子查询替代数组值。

```
$builder->whereIn('id', function(BaseBuilder $builder) {
    return $builder->select('job_id')->from('users_jobs')->
↪where('user_id', 3);
});
// 生成: WHERE "id" IN (SELECT "job_id" FROM "users_jobs"␣
↪WHERE "user_id" = 3)
```

### $builder->orWhereIn()

该方法用于生成 WHERE IN（‘item’，‘item’）子句，多个子句之间使用 OR 连接

```
$names = ['Frank', 'Todd', 'James'];
$builder->orWhereIn('username', $names);
// 生成: OR username IN ('Frank', 'Todd', 'James')
```

你可以用子查询替代数组值。

```
$builder->orWhereIn('id', function(BaseBuilder $builder) {
    return $builder->select('job_id')->from('users_jobs')->
↪where('user_id', 3);
});

// 生成: OR "id" IN (SELECT "job_id" FROM "users_jobs" WHERE
↪"user_id" = 3)
```

### $builder->whereNotIn()

该方法用于生成 WHERE NOT IN(‘item’, ‘item’) 子句，多个子句之间使用 AND 连接

```
$names = ['Frank', 'Todd', 'James'];
$builder->whereNotIn('username', $names);
// 生成: WHERE username NOT IN ('Frank', 'Todd', 'James')
```

你可以用子查询替代数组值。

```
$builder->whereNotIn('id', function(BaseBuilder $builder) {
    return $builder->select('job_id')->from('users_jobs')->
↪where('user_id', 3);
});

// 生成: WHERE "id" NOT IN (SELECT "job_id" FROM "users_jobs"␣
↪WHERE "user_id" = 3)
```

### $builder->orWhereNotIn()

该方法用于生成 WHERE NOT IN(‘item’, ‘item’) 子句，多个子句之间使用 OR 连接

```
$names = ['Frank', 'Todd', 'James'];
$builder->orWhereNotIn('username', $names);
// 生成: OR username NOT IN ('Frank', 'Todd', 'James')
```

你可以用子查询替代数组值。

```
$builder->orWhereNotIn('id', function(BaseBuilder $builder) {
    return $builder->select('job_id')->from('users_jobs')->
↪where('user_id', 3);
});
```

(下页继续)

(续上页)

```
// 生成: OR "id" NOT IN (SELECT "job_id" FROM "users_jobs"␣
↪WHERE "user_id" = 3)
```

### 查找相似的数据

**$builder->like()**

这个方法使您能够生成类似 **LIKE** 子句，做搜索时非常有用。

---

**注解:** 所有传入数据将被自动转义。

---

**注解:** like* 通过传第五个参数传递值 true 可以强制在执行查询时不区分大小写。这项特性可用性跟平台相关，否则将强制值转为小写，例如 WHERE LOWER(column) LIKE '%search%'，让其生效可能需要在制作索引时用 LOWER(column) 而不是 column 。

---

1. **简单 key/value 方式:**

   ```
   $builder->like('title', 'match');
   // 生成: 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:

   ```
   $builder->like('title', 'match');
   $builder->like('body', 'match');
   // WHERE `title` LIKE '%match%' ESCAPE '!' AND  `body`␣
   ↪LIKE '%match% ESCAPE '!'
   ```

   如果你想控制通配符通配符（%）的位置，可以指定第三个参数，可用选项：’ before’，’ after’ 和 ‘both’（默认）。

   ```
   $builder->like('title', 'match', 'before');     // 生成:␣
   ↪WHERE `title` LIKE '%match' ESCAPE '!'
   $builder->like('title', 'match', 'after');      // 生成:␣
   ↪WHERE `title` LIKE 'match%' ESCAPE '!'
   $builder->like('title', 'match', 'both');       // 生成:␣
   ↪WHERE `title` LIKE '%match%' ESCAPE '!'
   ```

2. **关联数组方式:**

   ```
   $array = ['title' => $match, 'page1' => $match, 'page2' =>
   ↪$match];
   $builder->like($array);
   ```

(下页继续)

(续上页)

```
// WHERE `title` LIKE '%match%' ESCAPE '!' AND  `page1`␣
↪LIKE '%match%' ESCAPE '!' AND  `page2` LIKE '%match%'␣
↪ESCAPE '!'
```

### $builder->orLike()

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:

```
$builder->like('title', 'match'); $builder->orLike('body', $match);
// WHERE `title` LIKE '%match%' ESCAPE '!' OR  `body` LIKE '%match%'␣
↪ESCAPE '!'
```

### $builder->notLike()

这个方法和 like() 方法一样，只是生成 NOT LIKE 子句:

```
$builder->notLike('title', 'match');    // WHERE `title` NOT LIKE '
↪%match% ESCAPE '!'
```

### $builder->orNotLike()

这个方法和 notLike() 方法一样，只是多个条件之间使用 OR 连接:

```
$builder->like('title', 'match');
$builder->orNotLike('body', 'match');
// WHERE `title` LIKE '%match% OR  `body` NOT LIKE '%match%' ESCAPE '!'
```

### $builder->groupBy()

该方法用于生成 GROUP BY 子句:

```
$builder->groupBy("title"); // 生成: GROUP BY title
```

你也可以通过一个数组传入多个值:

```
$builder->groupBy(["title", "date"]);  // 生成: GROUP BY title, date
```

### $builder->distinct()

该方法用于向查询中添加“DISTINCT”关键字

```
$builder->distinct();
$builder->get(); // 生成: SELECT DISTINCT * FROM mytable
```

### $builder->having()

该方法用于生成 HAVING 子句，有下面两种不同的语法。有两种可用语法，单参数或双参数:

```
$builder->having('user_id = 45');  // 生成: HAVING user_id = 45
$builder->having('user_id',  45);  // 生成: HAVING user_id = 45
```

你还可以传递一个包含多个值的数组:

```
$builder->having(['title =' => 'My Title', 'id <' => $id]);
// 生成: HAVING title = 'My Title', id < 45
```

如果你正在使用 CodeIgniter 为其转义查询的数据库，你可以传第三个可选参数来防止转义内容，设为 FALSE 。

```
$builder->having('user_id',  45);  // 生成: HAVING `user_id` = 45 in␣
↪some databases such as MySQL
$builder->having('user_id',  45, FALSE);  // 生成: HAVING user_id = 45
```

### $builder->orHaving()

该方法和 having() 方法一样，只是多个条件之间使用 “OR” 进行连接。

### $builder->havingIn()

生成一个 HAVING 字段的 IN ( ‘item’ , ‘item’ ) SQL 查询子句，多个条件之间使用 AND 连接

```
$groups = [1, 2, 3];
$builder->havingIn('group_id', $groups);
// 生成: HAVING group_id IN (1, 2, 3)
```

你可以用子查询代替数组。

```
$builder->havingIn('id', function(BaseBuilder $builder) {
    return $builder->select('user_id')->from('users_jobs')->
↪where('group_id', 3);
});
// 生成: HAVING "id" IN (SELECT "user_id" FROM "users_jobs"␣
↪WHERE "group_id" = 3)
```

### $builder->orHavingIn()

生成一个 HAVING 字段的 IN ( ‘item’ , ‘item’ ) SQL 查询子句，多个条件之间使用 OR 连接

```
$groups = [1, 2, 3];
$builder->orHavingIn('group_id', $groups);
// 生成: OR group_id IN (1, 2, 3)
```

你可以用子查询代替数组。

```
$builder->orHavingIn('id', function(BaseBuilder $builder) {
    return $builder->select('user_id')->from('users_jobs')->
↪where('group_id', 3);
});

// 生成: OR "id" IN (SELECT "user_id" FROM "users_jobs" WHERE
↪"group_id" = 3)
```

(下页继续)

（续上页）

```
```

### $builder->havingNotIn()

生成一个 HAVING 字段的 NOT IN（‘item’，‘item’）SQL 查询子句，多个条件之间使用 AND 连接

```
$groups = [1, 2, 3];
$builder->havingNotIn('group_id', $groups);
// 生成: HAVING group_id NOT IN (1, 2, 3)
```

你可以用子查询代替数组。

```
$builder->havingNotIn('id', function(BaseBuilder $builder) {
    return $builder->select('user_id')->from('users_jobs')->
↪where('group_id', 3);
});

// 生成: HAVING "id" NOT IN (SELECT "user_id" FROM "users_jobs
↪" WHERE "group_id" = 3)
```

### $builder->orHavingNotIn()

生成一个 HAVING 字段的 NOT IN（‘item’，‘item’）SQL 查询子句，多个条件之间使用 OR 连接

```
$groups = [1, 2, 3];
$builder->havingNotIn('group_id', $groups);
// 生成: OR group_id NOT IN (1, 2, 3)
```

你可以用子查询代替数组。

```
$builder->orHavingNotIn('id', function(BaseBuilder $builder) {
    return $builder->select('user_id')->from('users_jobs')->
↪where('group_id', 3);
});

// 生成: OR "id" NOT IN (SELECT "user_id" FROM "users_jobs"␣
↪WHERE "group_id" = 3)
```

### $builder->havingLike()

该方法让你能够在 HAVING 查询部分生成 **LIKE** 子句，常用于搜索。

**注解：** 该方法所有传入参数会被自动转义。

**注解：** havingLike* 通过传第五个参数传递值 true 可以强制在执行查询时不区分大小写。这项特性可用性跟平台相关，否则将强制值转为小写，例如 HAVING LOWER(column)

LIKE '%search%'，让其生效可能需要在制作索引时用 LOWER(column) 而不是 column 。

1. **简单 key/value 方式:**

```
$builder->havingLike('title', 'match');
// 生成: HAVING `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:

```
$builder->havingLike('title', 'match');
$builder->havingLike('body', 'match');
// HAVING `title` LIKE '%match%' ESCAPE '!' AND  `body`␣
↪LIKE '%match% ESCAPE '!'
```

如果你想控制通配符通配符（%）的位置，可以指定第三个参数，可用选项：’before’,’after’ 和 ‘both’（默认）。

```
$builder->havingLike('title', 'match', 'before');       //␣
↪生成: HAVING `title` LIKE '%match' ESCAPE '!'
$builder->havingLike('title', 'match', 'after');        //␣
↪生成: HAVING `title` LIKE 'match%' ESCAPE '!'
$builder->havingLike('title', 'match', 'both'); // 生成:␣
↪HAVING `title` LIKE '%match%' ESCAPE '!'
```

2. **关联数组方式:**

```
$array = ['title' => $match, 'page1' => $match, 'page2' =>
↪$match];
$builder->havingLike($array);
// HAVING `title` LIKE '%match%' ESCAPE '!' AND  `page1`␣
↪LIKE '%match%' ESCAPE '!' AND  `page2` LIKE '%match%'␣
↪ESCAPE '!'
```

### $builder->orHavingLike()

这个方法和上面的方法一样，只是多个条件之间使用 OR 进行连接:

```
$builder->havingLike('title', 'match'); $builder->orHavingLike('body',
↪$match);
// HAVING `title` LIKE '%match%' ESCAPE '!' OR  `body` LIKE '%match%'␣
↪ESCAPE '!'
```

### $builder->notHavingLike()

这个方法和 havingLike() 一样，只是它生成的是 NOT LIKE 子句:

```
$builder->notHavingLike('title', 'match');      // HAVING `title` NOT␣
↪LIKE '%match% ESCAPE '!'
```

**$builder->orNotHavingLike()**

这个方法和 notHavingLike() 一样，只是多个条件之间使用 OR 进行连接:

```
$builder->havingLike('title', 'match');
$builder->orNotHavingLike('body', 'match');
// HAVING `title` LIKE '%match% OR  `body` NOT LIKE '%match%' ESCAPE '!
↪'
```

### 结果排序

**$builder->orderBy()**

该方法用于生成 ORDER BY 子句。

第一个参数包含你要排序的列名。

第二个参数用于设置排序的方向，可选项有：**ASC** ，**DESC** 和 **RANDOM** 。

```
$builder->orderBy('title', 'DESC');
// 生成: ORDER BY `title` DESC
```

第一个参数也可以是你自己的排序字符串:

```
$builder->orderBy('title DESC, name ASC');
// 生成: 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。

```
$builder->orderBy('title', 'DESC');
$builder->orderBy('name', 'ASC');
// 生成: ORDER BY `title` DESC, `name` ASC
```

如果你选择了 **RANDOM** 选项，第一个参数会被忽略，除非你指定第一个参数作为随机数的种子。

```
$builder->orderBy('title', 'RANDOM');
// 生成: ORDER BY RAND()

$builder->orderBy(42, 'RANDOM');
// 生成: ORDER BY RAND(42)
```

---

**注解:** Oracle 目前还不支持随机排序，会默认使用 ASC 替代。

---

### 结果分页与计数

**$builder->limit()**

该方法可以让你限制查询结果的返回行数:

```
$builder->limit(10);  // 生成: LIMIT 10
```

第二个参数可以用来设置偏移。

```
$builder->limit(10, 20);  // 生成: LIMIT 20, 10 (在 MySQL 里的情况，其他
数据库的语法略有不同)
```

**$builder->countAllResults()**

**该方法用于获取指定构造器查询返回的结果数量，接受的构造器方法有** `where()` ，`orWhere()` , `like()` , `orLike()` 等，例如:

```
echo $builder->countAllResults('my_table');  // 生成一个整数，比如␣
↪25
$builder->like('title', 'match');
$builder->from('my_table');
echo $builder->countAllResults(); // 生成一个整数，比如 17
```

然而，这个方法会重置你在 `select()` 里设置的所有值，如果你要保留它们，可以将第一个参数设置为 FALSE:

```
echo $builder->countAllResults(false); // 生成一个整数，比如 17
```

**$builder->countAll()**

该方法用于获取指定表的总行数，例如:

```
echo $builder->countAll();  // 生成一个整数，比如 25
```

与 countAllResult 方法一样，该方法也会重置你在 `select()` 里设置的所有值，如果你要保留它们，可以将第一个参数设置为 FALSE。

### 查询分组

查询分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:

```
$builder->select('*')->from('my_table')
        ->groupStart()
                ->where('a', 'a')
                ->orGroupStart()
                        ->where('b', 'b')
                        ->where('c', 'c')
                ->groupEnd()
        ->groupEnd()
        ->where('d', 'd')
->get();
```

(下页继续)

(续上页)

```
// 生成:
// SELECT * FROM (`my_table`) WHERE ( `a` = 'a' OR ( `b` = 'b' AND `c`␣
↪= 'c' ) ) AND `d` = 'd'
```

**注解：** 条件组必须要配对，确保每个 groupStart() 方法都有一个 groupEnd() 方法与之配对。

**$builder->groupStart()**

开始一个新的条件组，为查询中的 WHERE 条件添加一个左括号。

**$builder->orGroupStart()**

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”。

**$builder->notGroupStart()**

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”。

**$builder->orNotGroupStart()**

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”。

**$builder->groupEnd()**

结束当前的条件组，为查询中的 WHERE 条件添加一个右括号。

**$builder->groupHavingStart()**

开始一个新的条件组，为查询中的 HAVING 条件添加一个左括号。

**$builder->orGroupHavingStart()**

开始一个新的条件组，为查询中的 HAVING 条件添加一个左括号，并在前面加上 “OR”。

**$builder->notGroupHavingStart()**

开始一个新的条件组，为查询中的 HAVING 条件添加一个左括号，并在前面加上 “NOT”。

**$builder->orNotGroupHavingStart()**

开始一个新的条件组，为查询中的 HAVING 条件添加一个左括号，并在前面加上 “OR NOT”。

**$builder->groupHavingEnd()**

结束当前的条件组，为查询中的 HAVING 条件添加一个右括号。

### 插入数据

#### $builder->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'
);

$builder->insert($data);
// 生成: INSERT INTO mytable (title, name, date) VALUES ('My title',
↪'My name', 'My date')
```

第一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;
$builder->insert($object);
// 生成: INSERT INTO mytable (title, content, date) VALUES ('My Title',
↪'My Content', 'My Date')
```

第一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。

---

#### $builder->ignore()

该方法根据你提供的数据生成一条 INSERT IGNORE 语句并执行，如果已经存在相同主键，该数据不会被插入。你可以给该方法传入一个可选参数，类型是 **boolean** 。下面是使用数组的例子:

```
$data = [
        'title' => 'My title',
        'name'  => 'My Name',
        'date'  => 'My date'
```

(下页继续)

(续上页)

```
];

$builder->ignore(true)->insert($data);
// 生成: INSERT OR IGNORE INTO mytable (title, name, date) VALUES ('My␣
↪title', 'My name', 'My date')
```

### $builder->getCompiledInsert()

该方法和 $builder->insert() 方法一样编译插入查询，但是 并不执行。此方法只是将 SQL 查询作为字符串返回。

例如:

```
$data = array(
        'title' => 'My title',
        'name'  => 'My Name',
        'date'  => 'My date'
);

$sql = $builder->set($data)->getCompiledInsert('mytable');
echo $sql;

// 生成字符串: 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（默认会重置，如 $builder->insert() 方法一样）：

```
echo $builder->set('title', 'My Title')->getCompiledInsert('mytable',␣
↪FALSE);

// 生成字符串: INSERT INTO mytable (`title`) VALUES ('My Title')

echo $builder->set('content', 'My Content')->getCompiledInsert();

// 生成字符串: INSERT INTO mytable (`title`, `content`) VALUES ('My␣
↪Title', 'My Content')
```

最值得注意的是，上例第二个查询并没有用到 **$builder->from()** 方法，也没有为查询指定表名参数。因为这个查询没有被可重置值的 **$builder->insert()** 方法执行，或是使用 **$builder->resetQuery()** 方法直接重置。

---

**注解:** 这个方法不支持批量插入。

---

### $builder->insertBatch()

该方法根据你提供的数据生成一条 INSERT 语句并执行，它的参数可以是一个 **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'
        )
);

$builder->insertBatch($data);
// 生成: INSERT INTO mytable (title, name, date) VALUES ('My title',
↪'My name', 'My date'),  ('Another title', 'Another name', 'Another␣
↪date')
```

第一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

#### $builder->replace()

该方法用于执行一条 REPLACE 语句，基本上是（可选）DELETE + INSERT 的 SQL 标准，使用 *PRIMARY* 和 *UNIQUE* 键作为决定因素。在我们的例子中，它可以使你免于实现各种不同逻辑的组合：`select()` ，`update()` ，`delete()` 和 `insert()` 。

例如:

```
$data = array(
        'title' => 'My title',
        'name'  => 'My Name',
        'date'  => 'My date'
);

$builder->replace($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* 字段是主键，那么如果我们数据库里有一行包含‘My title’为标题的数据，那行将被删除并被我们的新数据取代。

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。

### $builder->set()

该方法可以设置 insert 或 update 用到的数据。

**它可以用来代替直接将数据数组传递给 insert 或 update 方法:**

```
$builder->set('name', $name);
$builder->insert();  // 生成: INSERT INTO mytable (`name`) VALUES ('{
↪$name}')
```

如果你多次调用该方法，它会正确组装出 insert 或 update 语句来:

```
$builder->set('name', $name);
$builder->set('title', $title);
$builder->set('status', $status);
$builder->insert();
```

set() 将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。为了说明区别，这里有一个带转义的 set() 方法和不带转义的例子。

```
$builder->set('field', 'field+1', FALSE);
$builder->where('id', 2);
$builder->update(); // 生成 UPDATE mytable SET field = field+1 WHERE␣
↪`id` = 2

$builder->set('field', 'field+1');
$builder->where('id', 2);
$builder->update(); // 生成 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
);

$builder->set($array);
$builder->insert();
```

或者一个对象:

```
/*
class Myclass {
        public $title   = 'My Title';
        public $content = 'My Content';
        public $date    = 'My Date';
}
```

(下页继续)

(续上页)

```
*/

$object = new Myclass;
$builder->set($object);
$builder->insert();
```

### $builder->update()

该方法根据你提供的数据生成更新字符串并执行，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
);

$builder->where('id', $id);
$builder->update($data);
// 生成:
//
//      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;
$builder->where('id', $id);
$builder->update($object);
// 生成:
//
// 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。

你应该注意到用 $builder->where() 方法可以为你设置 WHERE 子句。你可以选择性的将这些（条件）信息直接以字符串传入 update 方法:

```
$builder->update($data, "id = 4");
```

或者使用一个数组:

```
$builder->update($data, array('id' => $id));
```

当执行更新操作时，你还可以使用上面介绍的 $builder->set() 方法。

**$builder->updateBatch()**

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'
   )
);

$builder->updateBatch($data, 'title');

// 生成:
// 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数据，是个二维数组，第二个参数是 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:** 由于该方法的内部实现，在这之后调用 `affectedRows()` 方法的返回值可能不

正确，替代办法是用 updateBatch() 的返回值，表示受影响的行数。

#### $builder->getCompiledUpdate()

该方法和 $builder->getCompiledInsert() 方法完全一样，除了生成的 SQL 语句是 UPDATE 而不是 INSERT。

查看 *$builder->getCompiledInsert()* 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

#### $builder->delete()

该方法生成删除 SQL 语句并执行。

```
$builder->delete(array('id' => $id));  // 生成：// DELETE FROM mytable
↪// WHERE id = $id
```

第一个参数为 where 子句。你也可以使用 where() 或 or_where() 方法替代第一个参数:

```
$builder->where('id', $id);
$builder->delete();

// 生成：
// DELETE FROM mytable
// WHERE id = $id
```

如果你想删除一个表中的全部数据，可以使用 truncate() 或 emptyTable() 方法。

#### $builder->emptyTable()

该方法生成删除 SQl 语句并执行:

```
$builder->emptyTable('mytable'); // 生成：DELETE FROM mytable
```

#### $builder->truncate()

该方法生截断 SQL 语句并执行。

```
$builder->truncate();

// 生成：
// TRUNCATE mytable
```

**注解：** 如果 TRUNCATE 命令不可用，truncate() 方法将执行 “DELETE FROM table”。

#### $builder->getCompiledDelete()

该方法和 `$builder->getCompiledInsert()` 方法完全一样，除了生成的 SQL 语句是 DELETE 而不是 INSERT。

查看 $builder->getCompiledInsert() 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，链式方法可以大大简化你的语法。感受一下这个例子:

```
$query = $builder->select('title')
                 ->where('id', $id)
                 ->limit(10, 20)
                 ->get();
```

### 重置查询构造器

#### $builder->resetQuery()

该方法使你可以重置查询构造器，而无需先执行例如 $builder->get() 或 $builder->insert() 这类方法。

当你要用查询构造器生成 SQL 语句（如：`$builder->getCompiledSelect()`），之后再执行它，这种情况下，不重置查询构造器很有用:

```
    // 注意 get_compiled_select 方法的第二个参数为 FALSE
$sql = $builder->select(['field1','field2'])
               ->where('field3',5)
               ->getCompiledSelect(false);

// ...
// 用 SQL 代码做一些疯狂的事情... 比如将它添加到 cron 脚本中
// 以后执行还是什么...
// ...

$data = $builder->get()->getResultArray();

// 会执行并返回以下查询的结果数组吗:
// SELECT field1, field1 from mytable where field3 = 5;
```

### 类库参考

```
class CodeIgniterDatabaseBaseBuilder
```

```
        resetQuery()
```

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

重置当前查询构造器状态。当你需要构建一个可在某些情况下取消的查询时有用。

`countAllResults`(*[$reset = TRUE]*)

**参数**

- **$reset** (*bool*) – 是否重置 SELECT 的值

**返回** 查询结果中的行数

**返回类型** int

生成特定于平台的查询语句，用于计数查询构造器返回的行数。

`countAll`(*[$reset = TRUE]*)

**参数**

- **$reset** (*bool*) – 是否重置 SELECT 的值

**返回** 查询结果中的行数

**返回类型** int

生成特定于平台的查询语句，用于计数查询构造器返回的行数。

`get`(*[$limit = NULL[, $offset = NULL]]*)

**参数**

- **$limit** (*int*) – LIMIT 子句
- **$offset** (*int*) – OFFSET 子句

**返回** CodeIgniterDatabaseResultInterface instance (方法链)

**返回类型** CodeIgniterDatabaseResultInterface

基于已经调用过的查询构造器方法，编译执行 SELECT 查询。

`getWhere`(*[$where = NULL[, $limit = NULL[, $offset = NULL]]]*)

**参数**

- **$where** (*string*) – WHERE 子句
- **$limit** (*int*) – LIMIT 子句
- **$offset** (*int*) – OFFSET 子句

**返回** CodeIgniterDatabaseResultInterface instance (方法链)

**返回类型** CodeIgniterDatabaseResultInterface

与 `get()` 相同，但也允许直接添加 WHERE 。

`select`(*[$select = '*'[, $escape = NULL]]*)

**参数**

- **$select** (*string*) – 查询的 SELECT 部分
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 SELECT 子句。

`selectAvg`(*[$select = ''[, $alias = '']]*)

**参数**

- **$select** (*string*) – 用于计算平均值的字段
- **$alias** (*string*) – 结果值名称的别名

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 SELECT AVG(field) 子句。

**selectMax**(*[$select = '', $alias = '']*)

**参数**

- **$select** (*string*) – 用于计算最大值的字段
- **$alias** (*string*) – 结果值名称的别名

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 SELECT MAX(field) 子句。

**selectMin**(*[$select = '', $alias = '']*)

**参数**

- **$select** (*string*) – 用于计算最小值的字段
- **$alias** (*string*) – 结果值名称的别名

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 SELECT MIN(field) 子句。

**selectSum**(*[$select = '', $alias = '']*)

**参数**

- **$select** (*string*) – 字段来计算总和
- **$alias** (*string*) – 结果值名称的别名

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 SELECT SUM(field) 子句。

**selectCount**(*[$select = '', $alias = '']*)

**参数**

- **$select** (*string*) – 用于计算记录总和的字段
- **$alias** (*string*) – 结果值名称的别名

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 SELECT COUNT(field) 子句。

**distinct**(*[$val = TRUE]*)

**参数**

- **$val** (*bool*) – 预期的 “distinct” 标志值

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

设置一个标志，告诉查询构建器给 SELECT 部分添加 DISTINCT 子句。

**from**(*$from[, $overwrite = FALSE]*)

**参数**

- **$from** (*mixed*) – Table name(s); 字符串或数组
- **$overwrite** (*bool*) – 是否移除第一个设置的表?

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

指定查询的 FROM 子句。

join(*$table*, *$cond*[, *$type* = ''[, *$escape* = *NULL*]])

参数
- **$table** (*string*) – 要 join 的表名
- **$cond** (*string*) – JOIN ON 条件
- **$type** (*string*) – JOIN 类型
- **$escape** (*bool*) – 是否转义值和标识符

返回 BaseBuilder instance (方法链)

返回类型 BaseBuilder

向查询添加 JOIN 子句。

where(*$key*[, *$value* = *NULL*[, *$escape* = *NULL*]])

参数
- **$key** (*mixed*) – 要比较的字段名称或关联数组
- **$value** (*mixed*) – 如果是单个键，则与此值相比
- **$escape** (*bool*) – 是否转义值和标识符

返回 BaseBuilder instance

返回类型 *object*

生成查询的 WHERE 部分，用 'AND' 分隔多个调用。

orWhere(*$key*[, *$value* = *NULL*[, *$escape* = *NULL*]])

参数
- **$key** (*mixed*) – 要比较的字段名称或关联数组
- **$value** (*mixed*) – 如果是单个键，则与此值相比
- **$escape** (*bool*) – 是否转义值和标识符

返回 BaseBuilder instance

返回类型 *object*

生成查询的 WHERE 部分，用 'OR' 分隔多个调用。

orWhereIn([*$key* = *NULL*[, *$values* = *NULL*[, *$escape* = *NULL*]]])

参数
- **$key** (*string*) – 要搜索的字段
- **$values** (*array|Closure*) – 目标值的数组，或子查询的匿名函数
- **$escape** (*bool*) – 是否转义值和标识符

返回 BaseBuilder instance

返回类型 *object*

生成一个 WHERE 字段 IN( 'item' , 'item' ) SQL 查询，多个用 'OR' 连接。

orWhereNotIn([*$key* = *NULL*[, *$values* = *NULL*[, *$escape* = *NULL*]]])

参数
- **$key** (*string*) – 要搜索的字段
- **$values** (*array|Closure*) – 目标值的数组，或子查询的匿名函数
- **$escape** (*bool*) – 是否转义值和标识符

返回 BaseBuilder instance

返回类型 *object*

生成一个 WHERE 字段 NOT IN(‘item’，‘item’) SQL 查询，多个用‘OR’连接。

**whereIn**([*$key = NULL*[, *$values = NULL*[, *$escape = NULL*]]])

**参数**

- **$key** (*string*) – 要检查的字段的名称
- **$values** (*array|Closure*) – 目标值的数组，或子查询的匿名函数
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance

**返回类型** *object*

生成一个 WHERE 字段 IN(‘item’，‘item’) SQL 查询，多个用‘AND’连接。

**whereNotIn**([*$key = NULL*[, *$values = NULL*[, *$escape = NULL*]]])

**参数**

- **$key** (*string*) – 要检查的字段的名称
- **$values** (*array|Closure*) – 目标值的数组，或子查询的匿名函数
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance

**返回类型** *object*

生成一个 WHERE 字段 NOT IN(‘item’，‘item’) SQL 查询，多个用‘AND’连接。

**groupStart**()

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

启动组表达式，使用 AND 连接其中的条件。

**orGroupStart**()

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

启动组表达式，使用 OR 连接其中的条件。

**notGroupStart**()

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

启动组表达式，使用 AND NOT 连接其中的条件。

**orNotGroupStart**()

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

启动组表达式，使用 OR NOT 连接其中的条件。

**groupEnd**()

**返回** BaseBuilder instance

**返回类型** *object*

完成一个组表达式。

like(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *NULL*[, *$insensitiveSearch* = *FALSE*]]]])

**参数**

- **$field** (*string*) – 字段名
- **$match** (*string*) – 匹配的文本部分
- **$side** (*string*) – 将‘%’通配符放在表达式的哪一侧
- **$escape** (*bool*) – 是否转义值和标识符
- **$insensitiveSearch** (*bool*) – 是否强制大小写不敏感检索

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 LIKE 子句，用 AND 分隔多个调用。

orLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *NULL*[, *$insensitiveSearch* = *FALSE*]]]])

**参数**

- **$field** (*string*) – 字段名
- **$match** (*string*) – 匹配的文本部分
- **$side** (*string*) – 将‘%’通配符放在表达式的哪一侧
- **$escape** (*bool*) – 是否转义值和标识符
- **$insensitiveSearch** (*bool*) – 是否强制大小写不敏感检索

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 LIKE 子句，用 OR 分隔多个调用。

notLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *NULL*[, *$insensitiveSearch* = *FALSE*]]]])

**参数**

- **$field** (*string*) – 字段名
- **$match** (*string*) – 匹配的文本部分
- **$side** (*string*) – 将‘%’通配符放在表达式的哪一侧
- **$escape** (*bool*) – 是否转义值和标识符
- **$insensitiveSearch** (*bool*) – 是否强制大小写不敏感检索

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询添加 NOT LIKE 子句，用 AND 分隔多个调用。

orNotLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *NULL*[, *$insensitiveSearch* = *FALSE*]]]])

**参数**

- **$field** (*string*) – 字段名
- **$match** (*string*) – 匹配的文本部分
- **$side** (*string*) – 将‘%’通配符放在表达式的哪一侧
- **$escape** (*bool*) – 是否转义值和标识符
- **$insensitiveSearch** (*bool*) – 是否强制大小写不敏感检索

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder
向查询添加 NOT LIKE 子句，用 OR 分隔多个调用。

**having**(*$key*[, *$value = NULL*[, *$escape = NULL*]])
**参数**
- **$key** (*mixed*) – 标识符（字符串）或 field/value 对的关联数组
- **$value** (*string*) – 如果 $key 是标识符，则寻求此值
- **$escape** (*string*) – 是否转义值和标识符

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder
向查询添加 HAVING 子句，用 AND 分隔多个调用。

**orHaving**(*$key*[, *$value = NULL*[, *$escape = NULL*]])
**参数**
- **$key** (*mixed*) – 标识符（字符串）或 field/value 对的关联数组
- **$value** (*string*) – 如果 $key 是标识符，则寻求此值
- **$escape** (*string*) – 是否转义值和标识符

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder
向查询添加 HAVING 子句，用 OR 分隔多个调用。

**orHavingIn**([*$key = NULL*[, *$values = NULL*[, *$escape = NULL*]]])
**参数**
- **$key** (*string*) – 要检索的字段名
- **$values** (*array|Closure*) – 目标值的数组，或子查询的匿名函数
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance
**返回类型** *object*
向查询添加 HAVING 字段 IN(‘item’, ‘item’) 子句，多个用 OR 连接。

**orHavingNotIn**([*$key = NULL*[, *$values = NULL*[, *$escape = NULL*]]])
**参数**
- **$key** (*string*) – 要检索的字段名
- **$values** (*array|Closure*) – 目标值的数组，或子查询的匿名函数
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance
**返回类型** *object*
向查询添加 HAVING 字段 NOT IN(‘item’, ‘item’) 子句，多个用 OR 连接。

**havingIn**([*$key* = *NULL*[, *$values* = *NULL*[, *$escape* = *NULL*]]])

**参数**

- **$key** (*string*) – 要检索的字段名
- **$values** (*array|Closure*) – 目标值的数组，或子查询的匿名函数
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance

**返回类型** *object*

向查询添加 HAVING 字段 IN(‘item’, ‘item’) 子句，多个用 AND 连接。

**havingNotIn**([*$key* = *NULL*[, *$values* = *NULL*[, *$escape* = *NULL*]]])

**参数**

- **$key** (*string*) – 要检索的字段名
- **$values** (*array|Closure*) – 目标值的数组，或子查询的匿名函数
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance

**返回类型** *object*

向查询添加 HAVING 字段 NOT IN(‘item’, ‘item’) 子句，多个用 AND 连接。

**havingLike**(*$field*[, *$match* = ”[, *$side* = *’both’*[, *$escape* = *NULL*[, *$insensitiveSearch* = *FALSE*]]]])

**参数**

- **$field** (*string*) – 字段名
- **$match** (*string*) – 匹配的文本部分
- **$side** (*string*) – 将‘%’通配符放在表达式的哪一侧
- **$escape** (*bool*) – 是否转义值和标识符
- **$insensitiveSearch** (*bool*) – 是否强制大小写不敏感检索

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询的 HAVING 部分添加 LIKE 子句，用 AND 分隔多个调用。

**orHavingLike**(*$field*[, *$match* = ”[, *$side* = *’both’*[, *$escape* = *NULL*[, *$insensitiveSearch* = *FALSE*]]]])

**参数**

- **$field** (*string*) – 字段名
- **$match** (*string*) – 匹配的文本部分
- **$side** (*string*) – 将‘%’通配符放在表达式的哪一侧
- **$escape** (*bool*) – 是否转义值和标识符
- **$insensitiveSearch** (*bool*) – 是否强制大小写不敏感检索

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询的 HAVING 部分添加 LIKE 子句，用 OR 分隔多个调用。

**notHavingLike**(*$field*[, *$match = ''*[, *$side = 'both'*[, *$escape = NULL*[, *$insensitiveSearch = FALSE*]]]])

**参数**

- **$field** (*string*) – 字段名
- **$match** (*string*) – 匹配的文本部分
- **$side** (*string*) – 将‘%’通配符放在表达式的哪一侧
- **$escape** (*bool*) – 是否转义值和标识符
- **$insensitiveSearch** (*bool*) – 是否强制大小写不敏感检索

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询的 HAVING 部分添加 NOT LIKE 子句，用 AND 分隔多个调用。

**orNotHavingLike**(*$field*[, *$match = ''*[, *$side = 'both'*[, *$escape = NULL*[, *$insensitiveSearch = FALSE*]]]])

**参数**

- **$field** (*string*) – 字段名
- **$match** (*string*) – 匹配的文本部分
- **$side** (*string*) – 将‘%’通配符放在表达式的哪一侧
- **$escape** (*bool*) – 是否转义值和标识符
- **$insensitiveSearch** (*bool*) – 是否强制大小写不敏感检索

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

向查询的 HAVING 部分添加 NOT LIKE 子句，用 OR 分隔多个调用。

**havingGroupStart**()

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

启动 HAVING 子句的组表达式，使用 AND 连接其中的条件。

**orHavingGroupStart**()

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

启动 HAVING 子句的组表达式，使用 OR 连接其中的条件。

**notHavingGroupStart**()

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

启动 HAVING 子句的组表达式，使用 AND NOT 连接其中的条件。

**orNotHavingGroupStart**()

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

启动 HAVING 子句的组表达式，使用 OR NOT 连接其中的条件。

**havingGroupEnd**()

**返回** BaseBuilder instance
**返回类型** *object*

完成一个 HAVING 子句的组表达式。

`groupBy`(*$by*[, *$escape = NULL*])

**参数**
- **$by** (*mixed*) – 根据字段分组; 字符串或数组

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder

向查询添加 GROUP BY 子句。

`orderBy`(*$orderby*[, *$direction = ''*[, *$escape = NULL*]])

**参数**
- **$orderby** (*string*) – 根据字段排序
- **$direction** (*string*) – 要求的排序 - ASC，DESC 或 RANDOM
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder

向查询添加 ORDER BY 子句。

`limit`(*$value*[, *$offset = 0*])

**参数**
- **$value** (*int*) – 限制返回行数
- **$offset** (*int*) – 偏移行数

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder

向查询添加 LIMIT 和 OFFSET 子句。

`offset`(*$offset*)

**参数**
- **$offset** (*int*) – 偏移行数

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder

向查询添加 OFFSET 子句。

`set`(*$key*[, *$value = ''*[, *$escape = NULL*]])

**参数**
- **$key** (*mixed*) – 字段名或 field/value 对的关联数组
- **$value** (*string*) – 字段值，如果 $key 是单个字段
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder

添加 field/value 键值对，稍后用于传递给 `insert()`，`update()` 或 `replace()`。

`insert`([*$set = NULL*[, *$escape = NULL*]])

**参数**
- **$set** (*array*) – field/value 对的关联数组
- **$escape** (*bool*) – 是否转义值和标识符

**返回** 成功时为 TRUE，失败时为 FALSE
**返回类型** bool

编译并执行 INSERT 语句。

**insertBatch**(*$set = NULL*[, *$escape = NULL*[, *$batch_size = 100*]])

**参数**
- **$set** (*array*) – 要插入的数据
- **$escape** (*bool*) – 是否转义值和标识符
- **$batch_size** (*int*) – 要一次插入的行数

**返回** 插入的行数或失败时的 FALSE
**返回类型** mixed

编译并执行批量的 INSERT 语句。

---

**注解:** 当数据超过 $batch_size 行时，将执行多个 INSERT 查询，每次尝试插入最多为 $batch_size 行。

---

**setInsertBatch**(*$key*[, *$value = ''*[, *$escape = NULL*]])

**参数**
- **$key** (*mixed*) – 字段名或 field/value 对应的关联数组
- **$value** (*string*) – 字段值，如果 $key 是单个字段
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance (方法链)
**返回类型** BaseBuilder

添加 field/value 键值对，稍后通过 insertBatch() 向一个表插入。

**update**([*$set = NULL*[, *$where = NULL*[, *$limit = NULL*]]])

**参数**
- **$set** (*array*) – field/value 对应的关联数组
- **$where** (*string*) – WHERE 子句
- **$limit** (*int*) – LIMIT 子句

**返回** TRUE 为成功, FALSE 为失败
**返回类型** bool

编译并执行 UPDATE 语句。

**updateBatch**([*$set = NULL*[, *$value = NULL*[, *$batch_size = 100*]]])

**参数**
- **$set** (*array*) – 字段名，或 field/value 对的关联数组
- **$value** (*string*) – 字段值，如果 $set 是单个字段
- **$batch_size** (*int*) – 在单个查询中分组的条件计数

**返回** 更新的行数或失败时的 FALSE
**返回类型** mixed

编译并执行批量的 UPDATE 语句。

---

**注解:** 当数据超过 $batch_size 行时，将执行多个 INSERT 查询，

每次最多处理 $batch_size 行。

---

**setUpdateBatch**(*$key*[, *$value = ''*[, *$escape = NULL*]])

**参数**

- **$key** (*mixed*) – 字段名，或 field/value 对的关联数组
- **$value** (*string*) – 字段值，如果 $key 是单个字段
- **$escape** (*bool*) – 是否转义值和标识符

**返回** BaseBuilder instance (方法链)

**返回类型** BaseBuilder

添加 field/value 键值对，稍后通过 updateBatch() 更新一个表。

**replace**([*$set = NULL*])

**参数**

- **$set** (*array*) – field/value 对应的关联数组

**返回** TRUE 为成功, FALSE 为失败

**返回类型** bool

编译并执行 REPLACE 语句。

**delete**([*$where = ''*[, *$limit = NULL*[, *$reset_data = TRUE*]]])

**参数**

- **$where** (*string*) – WHERE 子句
- **$limit** (*int*) – LIMIT 子句
- **$reset_data** (*bool*) – TRUE 会重置查询 “write” 子句

**返回** BaseBuilder instance (方法链) 或者失败时为 FALSE

**返回类型** mixed

编译并执行 DELETE 查询。

**increment**(*$column*[, *$value = 1*])

**参数**

- **$column** (*string*) – 要递增的列的名称
- **$value** (*int*) – 要给列增加的数值

给一个字段增加指定量的数值，如果该字段不是数字型字段，比如如 VARCHAR ，它可能会被新的 $value 值替换。

**decrement**(*$column*[, *$value = 1*])

**参数**

- **$column** (*string*) – 要减少的列的名称
- **$value** (*int*) – 要给列减少的数值

给一个字段减去指定量的数值，如果该字段不是数字型字段，比如如 VARCHAR ，它可能会被新的 $value 值替换。

**truncate**()

**返回** TRUE 为成功, FALSE 为失败

**返回类型** bool

在表上执行 TRUNCATE 语句。

**注解:** 如果所用的数据库平台不支持 TRUNCATE ，将使用 DELETE 语句替代。

`emptyTable()`
**返回** TRUE 为成功, FALSE 为失败
**返回类型** bool
通过 DELETE 语句删除表中所有记录。

`getCompiledSelect(`$reset = TRUE`)`
**参数**
- **$reset** (*bool*) – 是否重置当前查询构造器（QB）的值

**返回** 已编译的 SQL 语句为字符串
**返回类型** string
编译 SELECT 语句并将其作为字符串返回。

`getCompiledInsert(`$reset = TRUE`)`
**参数**
- **$reset** (*bool*) – 是否重置当前查询构造器（QB）的值

**返回** 已编译的 SQL 语句为字符串
**返回类型** string
编译 INSERT 语句并将其作为字符串返回。

`getCompiledUpdate(`$reset = TRUE`)`
**参数**
- **$reset** (*bool*) – 是否重置当前查询构造器（QB）的值

**返回** 已编译的 SQL 语句为字符串
**返回类型** string
编译 UPDATE 语句并将其作为字符串返回。

`getCompiledDelete(`$reset = TRUE`)`
**参数**
- **$reset** (*bool*) – 是否重置当前查询构造器（QB）的值

**返回** 已编译的 SQL 语句为字符串
**返回类型** string
编译 DELETE 语句并将其作为字符串返回。

### 6.1.8 事务

CodeIgniter 的数据库抽象类允许你将事务和支持事务安全表类型的数据库一起使用。在 MySQL 中，你需要将表设置为 InnoDB 或者 BDB 类型，而不是更常见的 MyISAM。大多数的数据库本身支持事务。

如果你不熟悉事务，我们建议找个好的在线资源学习下，了解你正用的数据库。以下的信息假定你对事务有基本的了解。

### CodeIgniter 的事务方法

CodeIgniter 采用一种与流行的 ADODB 数据库类很相似的方式处理事务。我们选用这种方法因为它极大地简化了运行事务的过程。在大多数情况下，所需要的只是两行代码。

传统的事务需要相当多的工序才能实现，因为它要求你跟踪查询并根据查询的成功或失败来决定提交还是回滚，这在嵌套查询时尤为麻烦。相比之下，我们已经实现了一个智能事务系统，可以自动为你完成这些工作（如果你想要手动管理你的事务也可以，但这实际上没有任何好处）。

### 运行事务

要使用事务运行查询，你将使用 $this->db->transStart() 和 $this->db->transComplete() 方法，如下所示:

```
$this->db->transStart();
$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->query('AND YET ANOTHER QUERY...');
$this->db->transComplete();
```

你可以在启动/完成方法之间运行任意多的查询，并且根据任何给定查询的成功或失败结果，他们都能被提交或回滚。

### 严格模式

默认情况下，CodeIgniter 以严格模式运行所有事务。启用严格模式时，如果你正在运行多组事务，假如一个组失败，所有组都将被回滚。如果禁用严格模式，则会独立处理每个组，这意味着一个组的失败不会影响其他组。

可以按如下方式禁用严格模式:

```
$this->db->transStrict(false);
```

### 错误处理

如果在 Config / Database.php 文件中启用了错误报告，在提交失败时会看到标准错误消息。如果关闭调试，你可以像下面这样处理自己的错误:

```
$this->db->transStart();
$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->transComplete();

if ($this->db->transStatus() === FALSE)
```

(下页继续)

(续上页)

```
{
        // 生成错误... 或使用 log_message() 函数记录错误
}
```

### 禁用事务

事务功能是默认开启的，如果要禁用事务，可以执行 $this->db->transOff() 操作:

```
$this->db->transOff();

$this->db->transStart();
$this->db->query('AN SQL QUERY...');
$this->db->transComplete();
```

禁用事务时，你的查询将自动提交，就像平时没事务那样的执行查询。

### 测试模式

你可以选择将事务系统置于“测试模式”，这将导致你的查询被回滚 – 即使查询产生有效结果。要使用测试模式，只需将 $this->db->transStart() 方法的第一个参数设置为 TRUE:

```
$this->db->transStart(true); // 查询将被回滚
$this->db->query('AN SQL QUERY...');
$this->db->transComplete();
```

### 手动运行事务

如果你想手动运行事务，可以按如下方式执行:

```
$this->db->transBegin();

$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->query('AND YET ANOTHER QUERY...');

if ($this->db->transStatus() === FALSE)
{
        $this->db->transRollback();
}
else
{
        $this->db->transCommit();
}
```

**注解:** 确保在手动运行事务时使用 $this->db->transBegin()，**而不是** $this->db->transStart()。

### 6.1.9 数据库元数据



#### 表元数据

下面这些方法用于获取表信息。

#### 列出数据库中的所有表

**$db->listTables();**

返回一个数组，包含当前连接数据库的全部表名称。例如:

```
$tables = $db->listTables();

foreach ($tables as $table)
{
        echo $table;
}
```

#### 检查表是否存在

**$db->tableExists();**

有时先检查某个表是否存在再进行操作会比较有用，返回布尔值 TRUE/FALSE. 例如:

```
if ($db->tableExists('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

#### 列出表的所有字段

**$db->getFieldNames()**

返回包含字段名称的数组，有两种不同的调用方式：

1. 你可以调用 $db->object 的方法获取表的字段:

```
$fields = $db->getFieldNames('table_name');

foreach ($fields as $field)
{
        echo $field;
}
```

2. 你可以调用任何查询结果对象的方法获取所有字段:

```
$query = $db->query('SELECT * FROM some_table');

foreach ($query->getFieldNames() as $field)
{
        echo $field;
}
```

#### 检查表中是否存在某字段

**$db->fieldExists()**

有时先确定某个字段是否存在再进行操作也比较有用，该方法返回布尔值 TRUE/FALSE。使用示例:

```
if ($db->fieldExists('field_name', 'table_name'))
{
        // some code...
}
```

**注解：** 将 *field_name* 替换为你要查找的字段名，并且将 *table_name* 替换为你要查找的表的名称

### 获取字段的元数据

**$db->getFieldData()**

该方法返回一个包含字段信息的对象数组。

有时，收集字段名称或相关的元数据会很有用，例如数据类型，最大长度等。

**注解：** 并非所有的数据库都支持元数据。

使用示例:

```
$fields = $db->getFieldData('table_name');

foreach ($fields as $field)
{
        echo $field->name;
        echo $field->type;
        echo $field->max_length;
        echo $field->primary_key;
}
```

如果你已经进行了查询，则可以使用结果对象而且不用提供表名:

```
$query = $db->query("YOUR QUERY");
$fields = $query->fieldData();
```

如果你的数据库支持，则可以用此方法获得以下数据:

- name - 字段名
- max_length - 字段的最大长度
- primary_key - 等于 1 的话表示此字段是主键
- type - 字段的数据类型

### 获取表的索引

**$db->getIndexData()**

返回一个包含索引信息的对象数组。

使用示例:

```
$keys = $db->getIndexData('table_name');

foreach ($keys as $key)
{
        echo $key->name;
        echo $key->type;
        echo $key->fields;  // 字段名的数组
}
```

根据数据库不同 type 会有所区别。例如，MySQL 会返回 primary、fulltext、spatial、index 或 unique 其中之一，每个（索引）关联一张表。

**$db->getForeignKeyData()**

返回一个包含外键信息的对象数组。

使用示例:

```
$keys = $db->getForeignKeyData('table_name');

foreach ($keys as $key)
{
        echo $key->constraint_name;
        echo $key->table_name;
        echo $key->column_name;
        echo $key->foreign_table_name;
        echo $key->foreign_column_name;
}
```

对象字段根据你用的数据库会有不同，例如 SQLite3 不返回 column_name 字段，但会附加 *sequence* 字段用于解释复合外键。

### 6.1.10 自定义函数调用

#### $db->callFunction();

该函数可以用平台无关的形式来调用 CodeIgniter 中没有原生包含的 PHP 数据库方法。举例来说，假如你想调用 `mysql_get_client_info` 函数，但是这一方法 CodeIgniter **并没有**原生支持。你可以这样做:

```
$db->callFunction('get_client_info');
```

第一个参数是函数名（必填），且 **不应该**带有 `mysql\_` 的前缀。该函数会根据当前数据库自动附加前缀。这个机制可确保在不同数据库平台运行相同的函数。当然，各数据库的函数调用并不完全一致，因此，就可移植性而言，此函数的实用性有限。

调用这个函数所需的任何参数可添加到第二、第三个参数，以此类推:

```
$db->callFunction('some_function', $param1, $param2, etc..);
```

这里，你经常要提供数据库连接 ID 或是查询结果 ID 作为参数，当前 DB 连接 ID 可以用该方法获得:

```
$db->connID;
```

查询结果 ID 可以用 QUERY 结果对象来获得，例如:

```
$query = $db->query("SOME QUERY");

$query->resultID;
```

### 6.1.11 数据库事件

数据库类包括着一些你可以用来了解关于数据库执行过程的事件 相关的内容。这些事件可以用来收集数据以供分析和报告。*Debug 工具条* 类使用了这一特性来收集用于工具条中展示的查询语句。

#### 事件

**DBQuery**

该事件会在一个新的查询语句运行完毕时触发，无论成功与否。唯一的参数就是一个当前查询语句*Query* 的实例。你可以使用该方法在标准输出流、日志文件中输出所有的查询语句，甚至创建工具自动化地分析查询语句，帮你发现潜在的索引丢失、慢查询等情况。可行的用例如下:

```
// 在 Config\Events.php 文件中
Events::on('DBQuery',
↪'CodeIgniter\Debug\Toolbar\Collectors\Database::collect');

// 收集所有的查询语句以备后来所需
public static function collect(CodeIgniter\Database\Query $query)
{
    static::$queries[] = $query;
}
```

### 6.1.12 实用工具

数据库实用工具类包含一系列可以帮助你管理数据库的方法。

- *从结果中获取 XML*

#### 从结果中获取 XML

**getXMLFromResult()**

该方法从数据库查询结果中返回 xml 结果，可以如下进行:

```
$model = new class extends \CodeIgniter\Model {
    protected $table      = 'foo';
    protected $primaryKey = 'id';
};
$db = \Closure::bind(function ($model) {
    return $model->db;
}, null, $model)($model);

$util = (new \CodeIgniter\Database\Database())->loadUtils($db);
echo $util->getXMLFromResult($model->get());
```

将会返回如下的 XML 结果:

```
<root>
    <element>
        <id>1</id>
        <name>bar</name>
    </element>
</root>
```

## 6.2 数据建模

CodeIgniter 具备丰富的工具，可用于对数据库表和记录进行建模和处理。

### 6.2.1 Using CodeIgniter’ s Model

## Models

Models provide a way to interact with a specific table in your database. They come out of the box with helper methods for much of the standard ways you would need to interact with a database table, including finding records, updating records, deleting records, and more.

## Accessing Models

Models are typically stored in the `app/Models` directory. They should have a namespace that matches their location within the directory, like `namespace App\Models`.

You can access models within your classes by creating a new instance or using the `model()` helper function.

```
// Create a new class manually
$userModel = new \App\Models\UserModel();

// Create a new class with the model function
$userModel = model('App\Models\UserModel', false);

// Create a shared instance of the model
$userModel = model('App\Models\UserModel');
```

(下页继续)

(续上页)

```
// Create shared instance with a supplied database connection
// When no namespace is given, it will search through all namespaces
// the system knows about and attempt to located the UserModel class.
$db = db_connect('custom');
$userModel = model('UserModel', true, $db);
```

### CodeIgniter’s Model

CodeIgniter does provide a model class that provides a few nice features, including:

- automatic database connection
- basic CRUD methods
- in-model validation
- automatic pagination
- and more

This class provides a solid base from which to build your own models, allowing you to rapidly build out your application’s model layer.

### Creating Your Model

To take advantage of CodeIgniter’s model, you would simply create a new model class that extends `CodeIgniter\Model`:

```
<?php namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{

}
```

This empty class provides convenient access to the database connection, the Query Builder, and a number of additional convenience methods.

### Connecting to the Database

When the class is first instantiated, if no database connection instance is passed to the constructor, it will automatically connect to the default database group, as set in the configuration. You can modify which group is used on a per-model basis by adding the DBGroup property to your class. This ensures that within the model any references to `$this->db` are made through the appropriate connection.

```
<?php namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    protected $DBGroup = 'group_name';
}
```

You would replace "group_name" with the name of a defined database group from the database configuration file.

### Configuring Your Model

The model class has a few configuration options that can be set to allow the class' methods to work seamlessly for you. The first two are used by all of the CRUD methods to determine what table to use and how we can find the required records:

```
<?php namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    protected $table      = 'users';
    protected $primaryKey = 'id';

    protected $returnType     = 'array';
    protected $useSoftDeletes = true;

    protected $allowedFields = ['name', 'email'];

    protected $useTimestamps = false;
    protected $createdField  = 'created_at';
    protected $updatedField  = 'updated_at';
    protected $deletedField  = 'deleted_at';

    protected $validationRules    = [];
    protected $validationMessages = [];
    protected $skipValidation     = false;
}
```

#### $table

Specifies the database table that this model primarily works with. This only applies to the built-in CRUD methods. You are not restricted to using only this table in your own queries.

**$primaryKey**

This is the name of the column that uniquely identifies the records in this table. This does not necessarily have to match the primary key that is specified in the database, but is used with methods like `find()` to know what column to match the specified value to.

---

**注解:** All Models must have a primaryKey specified to allow all of the features to work as expected.

---

**$returnType**

The Model' s CRUD methods will take a step of work away from you and automatically return the resulting data, instead of the Result object. This setting allows you to define the type of data that is returned. Valid values are 'array', 'object', or the fully qualified name of a class that can be used with the Result object' s getCustomResultObject() method.

**$useSoftDeletes**

If true, then any delete* method calls will set `deleted_at` in the database, instead of actually deleting the row. This can preserve data when it might be referenced elsewhere, or can maintain a "recycle bin" of objects that can be restored, or even simply preserve it as part of a security trail. If true, the find* methods will only return non-deleted rows, unless the withDeleted() method is called prior to calling the find* method.

This requires either a DATETIME or INTEGER field in the database as per the model' s $dateFormat setting. The default field name is `deleted_at` however this name can be configured to any name of your choice by using $deletedField property.

**$allowedFields**

This array should be updated with the field names that can be set during save, insert, or update methods. Any field names other than these will be discarded. This helps to protect against just taking input from a form and throwing it all at the model, resulting in potential mass assignment vulnerabilities.

**$useTimestamps**

This boolean value determines whether the current date is automatically added to all inserts and updates. If true, will set the current time in the format specified by $dateFormat. This requires that the table have columns named 'created_at' and 'updated_at' in the appropriate data type.

**$createdField**

Specifies which database field should use for keep data record create timestamp. Leave it empty to avoid update it (even useTimestamps is enabled)

**$updatedField**

Specifies which database field should use for keep data record update timestamp. Leave it empty to avoid update it (even useTimestamps is enabled)

**$dateFormat**

This value works with $useTimestamps and $useSoftDeletes to ensure that the correct type of date value gets inserted into the database. By default, this creates DATETIME values, but valid options are: datetime, date, or int (a PHP timestamp). Using 'useSoftDeletes' or 'useTimestamps' with an invalid or missing dateFormat will cause an exception.

**$validationRules**

Contains either an array of validation rules as described in 如何保存你的规则 or a string containing the name of a validation group, as described in the same section. Described in more detail below.

**$validationMessages**

Contains an array of custom error messages that should be used during validation, as described in 设置自定义错误消息. Described in more detail below.

**$skipValidation**

Whether validation should be skipped during all `inserts` and `updates`. The default value is false, meaning that data will always attempt to be validated. This is primarily used by the `skipValidation()` method, but may be changed to `true` so this model will never validate.

**$beforeInsert $afterInsert $beforeUpdate $afterUpdate $afterFind $afterDelete**

These arrays allow you to specify callback methods that will be run on the data at the time specified in the property name.

### Working With Data

#### Finding Data

Several functions are provided for doing basic CRUD work on your tables, including find(), insert(), update(), delete() and more.

**find()**

Returns a single row where the primary key matches the value passed in as the first parameter:

```
$user = $userModel->find($user_id);
```

The value is returned in the format specified in $returnType.

You can specify more than one row to return by passing an array of primaryKey values instead of just one:

```
$users = $userModel->find([1,2,3]);
```

If no parameters are passed in, will return all rows in that model's table, effectively acting like findAll(), though less explicit.

### findColumn()

Returns null or an indexed array of column values:

```
$user = $userModel->findColumn($column_name);
```

$column_name should be a name of single column else you will get the DataException.

### findAll()

Returns all results:

```
$users = $userModel->findAll();
```

This query may be modified by interjecting Query Builder commands as needed prior to calling this method:

```
$users = $userModel->where('active', 1)
                   ->findAll();
```

You can pass in a limit and offset values as the first and second parameters, respectively:

```
$users = $userModel->findAll($limit, $offset);
```

### first()

Returns the first row in the result set. This is best used in combination with the query builder.

```
$user = $userModel->where('deleted', 0)
                  ->first();
```

### withDeleted()

If $useSoftDeletes is true, then the find* methods will not return any rows where 'deleted_at IS NOT NULL'. To temporarily override this, you can use the withDeleted() method prior to calling the find* method.

```
// Only gets non-deleted rows (deleted = 0)
$activeUsers = $userModel->findAll();

// Gets all rows
$allUsers = $userModel->withDeleted()
                      ->findAll();
```

### onlyDeleted()

Whereas withDeleted() will return both deleted and not-deleted rows, this method modifies the next find* methods to return only soft deleted rows:

```
$deletedUsers = $userModel->onlyDeleted()
                          ->findAll();
```

## Saving Data

### insert()

An associative array of data is passed into this method as the only parameter to create a new row of data in the database. The array's keys must match the name of the columns in a $table, while the array's values are the values to save for that key:

```
$data = [
    'username' => 'darth',
    'email'    => 'd.vader@theempire.com'
];

$userModel->insert($data);
```

### update()

Updates an existing record in the database. The first parameter is the $primaryKey of the record to update. An associative array of data is passed into this method as the second parameter. The array's keys must match the name of the columns in a $table, while the array's values are the values to save for that key:

```
$data = [
    'username' => 'darth',
    'email'    => 'd.vader@theempire.com'
];

$userModel->update($id, $data);
```

Multiple records may be updated with a single call by passing an array of primary keys as the first parameter:

```
$data = [
    'active' => 1
];

$userModel->update([1, 2, 3], $data);
```

When you need a more flexible solution, you can leave the parameters empty and it functions like the Query Builder's update command, with the added benefit of validation, events, etc:

```
$userModel
    ->whereIn('id', [1,2,3])
    ->set(['active' => 1])
    ->update();
```

### save()

This is a wrapper around the insert() and update() methods that handle inserting or

updating the record automatically, based on whether it finds an array key matching the $primaryKey value:

```
// Defined as a model property
$primaryKey = 'id';

// Does an insert()
$data = [
    'username' => 'darth',
    'email'    => 'd.vader@theempire.com'
];

$userModel->save($data);

// Performs an update, since the primary key, 'id', is found.
$data = [
    'id'       => 3,
    'username' => 'darth',
    'email'    => 'd.vader@theempire.com'
];
$userModel->save($data);
```

The save method also can make working with custom class result objects much simpler by recognizing a non-simple object and grabbing its public and protected values into an array, which is then passed to the appropriate insert or update method. This allows you to work with Entity classes in a very clean way. Entity classes are simple classes that represent a single instance of an object type, like a user, a blog post, job, etc. This class is responsible for maintaining the business logic surrounding the object itself, like formatting elements in a certain way, etc. They shouldn' t have any idea about how they are saved to the database. At their simplest, they might look like this:

```
namespace App\Entities;

class Job
{
    protected $id;
    protected $name;
    protected $description;

    public function __get($key)
    {
        if (property_exists($this, $key))
        {
            return $this->$key;
        }
    }

    public function __set($key, $value)
```

(下页继续)

(续上页)

```
    {
        if (property_exists($this, $key))
        {
            $this->$key = $value;
        }
    }
}
```

A very simple model to work with this might look like:

```
use CodeIgniter\Model;

class JobModel extends Model
{
    protected $table = 'jobs';
    protected $returnType = '\App\Entities\Job';
    protected $allowedFields = [
        'name', 'description'
    ];
}
```

This model works with data from the `jobs` table, and returns all results as an instance of `App\Entities\Job`. When you need to persist that record to the database, you will need to either write custom methods, or use the model's `save()` method to inspect the class, grab any public and private properties, and save them to the database:

```
// Retrieve a Job instance
$job = $model->find(15);

// Make some changes
$job->name = "Foobar";

// Save the changes
$model->save($job);
```

**注解:** If you find yourself working with Entities a lot, CodeIgniter provides a built-in *Entity class* that provides several handy features that make developing Entities simpler.

## Deleting Data

### delete()

Takes a primary key value as the first parameter and deletes the matching record from the model's table:

```
$userModel->delete(12);
```

If the model's $useSoftDeletes value is true, this will update the row to set `deleted_at` to the current date and time. You can force a permanent delete by setting the second parameter as true.

An array of primary keys can be passed in as the first parameter to delete multiple records at once:

```
$userModel->delete([1,2,3]);
```

If no parameters are passed in, will act like the Query Builder's delete method, requiring a where call previously:

```
$userModel->where('id', 12)->delete();
```

#### purgeDeleted()

Cleans out the database table by permanently removing all rows that have 'deleted_at IS NOT NULL'.

```
$userModel->purgeDeleted();
```

### Validating Data

For many people, validating data in the model is the preferred way to ensure the data is kept to a single standard, without duplicating code. The Model class provides a way to automatically have all data validated prior to saving to the database with the `insert()`, `update()`, or `save()` methods.

The first step is to fill out the `$validationRules` class property with the fields and rules that should be applied. If you have custom error message that you want to use, place them in the `$validationMessages` array:

```
class UserModel extends Model
{
    protected $validationRules    = [
        'username'     => 'required|alpha_numeric_space|min_length[3]',
        'email'        => 'required|valid_email|is_unique[users.email]',
        'password'     => 'required|min_length[8]',
        'pass_confirm' => 'required_with[password]|matches[password]'
    ];

    protected $validationMessages = [
        'email'        => [
            'is_unique' => 'Sorry. That email has already been taken.␣
↪Please choose another.'
        ]
```

(下页继续)

(续上页)

```
    ];
}
```

The other way to set the validation message to fields by functions,

**setValidationMessage**(*$field*, *$fieldMessages*)

**参数**

- **$field** (*string*) –
- **$fieldMessages** (*array*) –

This function will set the field wise error messages.

Usage example:

```
$fieldName = 'name';
$fieldValidationMessage = [
    'required' => 'Your name is required here',
];
$model->setValidationMessage($fieldName, $fieldValidationMessage);
```

**setValidationMessages**(*$fieldMessages*)

**参数**

- **$fieldMessages** (*array*) –

This function will set the field messages.

Usage example:

```
$fieldValidationMessage = [
    'name' => [
        'required'   => 'Your baby name is missing.',
        'min_length' => 'Too short, man!',
    ],
];
$model->setValidationMessages($fieldValidationMessage);
```

Now, whenever you call the `insert()`, `update()`, or `save()` methods, the data will be validated. If it fails, the model will return boolean **false**. You can use the `errors()` method to retrieve the validation errors:

```
if ($model->save($data) === false)
{
    return view('updateUser', ['errors' => $model->errors()];
}
```

This returns an array with the field names and their associated errors that can be used to either show all of the errors at the top of the form, or to display them individually:

```
<?php if (! empty($errors)) : ?>
    <div class="alert alert-danger">
    <?php foreach ($errors as $field => $error) : ?>
        <p><?= $error ?></p>
    <?php endforeach ?>
    </div>
<?php endif ?>
```

If you' d rather organize your rules and error messages within the Validation configuration file, you can do that and simply set `$validationRules` to the name of the validation rule group you created:

```
class UserModel extends Model
{
    protected $validationRules = 'users';
}
```

### Retrieving Validation Rules

You can retrieve a model' s validation rules by accessing its `validationRules` property:

```
$rules = $model->validationRules;
```

You can also retrieve just a subset of those rules by calling the accessor method directly, with options:

```
$rules = $model->getValidationRules($options);
```

The `$options` parameter is an associative array with one element, whose key is either "except" or "only" , and which has as its value an array of fieldnames of interest.:

```
// get the rules for all but the "username" field
$rules = $model->getValidationRules(['except' => ['username']]);
// get the rules for only the "city" and "state" fields
$rules = $model->getValidationRules(['only' => ['city', 'state']]);
```

### Validation Placeholders

The model provides a simple method to replace parts of your rules based on data that' s being passed into it. This sounds fairly obscure but can be especially handy with the `is_unique` validation rule. Placeholders are simply the name of the field (or array key) that was passed in as $data surrounded by curly brackets. It will be replaced by the **value** of the matched incoming field. An example should clarify this:

```
protected $validationRules = [
    'email' => 'required|valid_email|is_unique[users.email,id,{id}]'
];
```

In this set of rules, it states that the email address should be unique in the database, except for the row that has an id matching the placeholder's value. Assuming that the form POST data had the following:

```
$_POST = [
    'id' => 4,
    'email' => 'foo@example.com'
];
```

then the `{id}` placeholder would be replaced with the number **4**, giving this revised rule:

```
protected $validationRules = [
    'email' => 'required|valid_email|is_unique[users.email,id,4]'
];
```

So it will ignore the row in the database that has `id=4` when it verifies the email is unique.

This can also be used to create more dynamic rules at runtime, as long as you take care that any dynamic keys passed in don't conflict with your form data.

### Protecting Fields

To help protect against Mass Assignment Attacks, the Model class **requires** that you list all of the field names that can be changed during inserts and updates in the `$allowedFields` class property. Any data provided in addition to these will be removed prior to hitting the database. This is great for ensuring that timestamps, or primary keys do not get changed.

```
protected $allowedFields = ['name', 'email', 'address'];
```

Occasionally, you will find times where you need to be able to change these elements. This is often during testing, migrations, or seeds. In these cases, you can turn the protection on or off:

```
$model->protect(false)
      ->insert($data)
      ->protect(true);
```

### Working With Query Builder

You can get access to a shared instance of the Query Builder for that model's database connection any time you need it:

```
$builder = $userModel->builder();
```

This builder is already set up with the model's $table.

You can also use Query Builder methods and the Model's CRUD methods in the same chained call, allowing for very elegant use:

```
$users = $userModel->where('status', 'active')
                   ->orderBy('last_login', 'asc')
                   ->findAll();
```

---

**注解:** You can also access the model's database connection seamlessly:

```
$user_name = $userModel->escape($name);
```

---

### Runtime Return Type Changes

You can specify the format that data should be returned as when using the find*() methods as the class property, $returnType. There may be times that you would like the data back in a different format, though. The Model provides methods that allow you to do just that.

---

**注解:** These methods only change the return type for the next find*() method call. After that, it is reset to its default value.

---

#### asArray()

Returns data from the next find*() method as associative arrays:

```
$users = $userModel->asArray()->where('status', 'active')->findAll();
```

#### asObject()

Returns data from the next find*() method as standard objects or custom class intances:

```
// Return as standard objects
$users = $userModel->asObject()->where('status', 'active')->findAll();

// Return as custom class instances
$users = $userModel->asObject('User')->where('status', 'active')->
↪findAll();
```

### Processing Large Amounts of Data

Sometimes, you need to process large amounts of data and would run the risk of running out of memory. To make this simpler, you may use the chunk() method to get smaller chunks of data that you can then do your work on. The first parameter is the number of rows to retrieve in a single chunk. The second parameter is a Closure that will be called for each row of data.

This is best used during cronjobs, data exports, or other large tasks.

```
$userModel->chunk(100, function ($data)
{
    // do something.
    // $data is a single row of data.
});
```

### Model Events

There are several points within the model' s execution that you can specify multiple callback methods to run. These methods can be used to normalize data, hash passwords, save related entities, and much more. The following points in the model' s execution can be affected, each through a class property: **$beforeInsert**, **$afterInsert**, **$beforeUpdate**, **afterUpdate**, **afterFind**, and **afterDelete**.

### Defining Callbacks

You specify the callbacks by first creating a new class method in your model to use. This class will always receive a $data array as its only parameter. The exact contents of the $data array will vary between events, but will always contain a key named **data** that contains the primary data passed to the original method. In the case of the insert* or update* methods, that will be the key/value pairs that are being inserted into the database. The main array will also contain the other values passed to the method, and be detailed later. The callback method must return the original $data array so other callbacks have the full information.

```
protected function hashPassword(array $data)
{
    if (! isset($data['data']['password'])) return $data;

    $data['data']['password_hash'] = password_hash($data['data'][
↪'password'], PASSWORD_DEFAULT);
    unset($data['data']['password'];

    return $data;
}
```

### Specifying Callbacks To Run

You specify when to run the callbacks by adding the method name to the appropriate class property (beforeInsert, afterUpdate, etc). Multiple callbacks can be added to a single event and they will be processed one after the other. You can use the same callback in multiple events:

```
protected $beforeInsert = ['hashPassword'];
protected $beforeUpdate = ['hashPassword'];
```

### Event Parameters

Since the exact data passed to each callback varies a bit, here are the details on what is in the $data parameter passed to each event:

| Event | $data contents |
|---|---|
| beforeInsert | **data** = the key/value pairs that are being inserted. If an object or Entity class is passed to the insert method, it is first converted to an array. |
| afterInsert | **id** = the primary key of the new row, or 0 on failure. **data** = the key/value pairs being inserted. **result** = the results of the insert() method used through the Query Builder. |
| beforeUpdate | **id** = the primary key of the row being updated. **data** = the key/value pairs that are being inserted. If an object or Entity class is passed to the insert method, it is first converted to an array. |
| afterUpdate | **id** = the primary key of the row being updated. **data** = the key/value pairs being updated. **result** = the results of the update() method used through the Query Builder. |
| afterFind | Varies by find* method. See the following: |
| • find() | **id** = the primary key of the row being searched for. **data** = The resulting row of data, or null if no result found. |
| • findAll() | **data** = the resulting rows of data, or null if no result found. **limit** = the number of rows to find. **offset** = the number of rows to skip during the search. |
| • first() | **data** = the resulting row found during the search, or null if none found. |
| beforeDelete | Varies by delete* method. See the following: |
| • delete() | **id** = primary key of row being deleted. **purge** = boolean whether soft-delete rows should be hard deleted. |
| afterDelete | Varies by delete* method. See the following: |
| • delete() | **id** = primary key of row being deleted. **purge** = boolean whether soft-delete rows should be hard deleted. **result** = the result of the delete() call on the Query Builder. **data** = unused. |

#### Manual Model Creation

You do not need to extend any special class to create a model for your application. All you need is to get an instance of the database connection and you' re good to go. This allows you to bypass the features CodeIgniter' s Model gives you out of the box, and create a fully custom experience.

```
<?php namespace App\Models;

use CodeIgniter\Database\ConnectionInterface;

class UserModel
{
    protected $db;

    public function __construct(ConnectionInterface &$db)
    {
        $this->db =& $db;
    }
}
```

### 6.2.2 Working With Entities

CodeIgniter supports Entity classes as a first-class citizen in it' s database layer, while keeping them completely optional to use. They are commonly used as part of the Repository pattern, but can be used directly with the *Model* if that fits your needs better.

### Entity Usage

At its core, an Entity class is simply a class that represents a single database row. It has class properties to represent the database columns, and provides any additional methods to implement the business logic for that row. The core feature, though, is that it doesn't know anything about how to persist itself. That's the responsibility of the model or the repository class. That way, if anything changes on how you need to save the object, you don't have to change how that object is used throughout the application. This makes it possible to use JSON or XML files to store the objects during a rapid prototyping stage, and then easily switch to a database when you've proven the concept works.

Let's walk through a very simple User Entity and how we'd work with it to help make things clear.

Assume you have a database table named `users` that has the following schema:

```
id          - integer
username    - string
email       - string
password    - string
created_at  - datetime
```

### Create the Entity Class

Now create a new Entity class. Since there's no default location to store these classes, and it doesn't fit in with the existing directory structure, create a new directory at **app/Entities**. Create the Entity itself at **app/Entities/User.php**.

```
<?php namespace App\Entities;

use CodeIgniter\Entity;

class User extends Entity
{
    //
}
```

At its simplest, this is all you need to do, though we'll make it more useful in a minute.

### Create the Model

Create the model first at **app/Models/UserModel.php** so that we can interact with it:

```
<?php namespace App\Models;
```

(下页继续)

(续上页)

```
use CodeIgniter\Model;

class UserModel extends Model
{
    protected $table         = 'users';
    protected $allowedFields = [
        'username', 'email', 'password'
    ];
    protected $returnType    = 'App\Entities\User';
    protected $useTimestamps = true;
}
```

The model uses the `users` table in the database for all of its activities. We' ve set the `$allowedFields` property to include all of the fields that we want outside classes to change. The `id`, `created_at`, and `updated_at` fields are handled automatically by the class or the database, so we don' t want to change those. Finally, we' ve set our Entity class as the `$returnType`. This ensures that all methods on the model that return rows from the database will return instances of our User Entity class instead of an object or array like normal.

### Working With the Entity Class

Now that all of the pieces are in place, you would work with the Entity class as you would any other class:

```
$user = $userModel->find($id);

// Display
echo $user->username;
echo $user->email;

// Updating
unset($user->username);
if (! isset($user->username)
{
    $user->username = 'something new';
}
$userModel->save($user);

// Create
$user = new \App\Entities\User();
$user->username = 'foo';
$user->email    = 'foo@example.com';
$userModel->save($user);
```

You may have noticed that the User class has not set any properties for the columns, but

you can still access them as if they were public properties. The base class, **CodeIgniter-Entity**, takes care of this for you, as well as providing the ability to check the properties with **isset()**, or **unset()** the property, and keep track of what columns have changed since the object was created or pulled from the database.

When the User is passed to the model' s **save()** method, it automatically takes care of reading the properties and saving any changes to columns listed in the model' s **$allowedFields** property. It also knows whether to create a new row, or update an existing one.

### Filling Properties Quickly

The Entity class also provides a method, `fill()` that allows you to shove an array of key/value pairs into the class and populate the class properties. Any property in the array will be set on the Entity. However, when saving through the model, only the fields in $allowedFields will actually be saved to the database, so you can store additional data on your entities without worrying much about stray fields getting saved incorrectly.

```
$data = $this->request->getPost();

$user = new \App\Entities\User();
$user->fill($data);
$userModel->save($user);
```

You can also pass the data in the constructor and the data will be passed through the *fill()* method during instantiation.

```
$data = $this->request->getPost();

$user = new \App\Entities\User($data);
$userModel->save($user);
```

### Handling Business Logic

While the examples above are convenient, they don' t help enforce any business logic. The base Entity class implements some smart `__get()` and `__set()` methods that will check for special methods and use those instead of using the attributes directly, allowing you to enforce any business logic or data conversion that you need.

Here' s an updated User entity to provide some examples of how this could be used:

```
<?php namespace App\Entities;

use CodeIgniter\Entity;
use CodeIgniter\I18n\Time;

class User extends Entity
```

(下页继续)

(续上页)

```
{
    public function setPassword(string $pass)
    {
        $this->attributes['password'] = password_hash($pass, PASSWORD_
↪BCRYPT);

        return $this;
    }

    public function setCreatedAt(string $dateString)
    {
        $this->attributes['created_at'] = new Time($dateString, 'UTC');

        return $this;
    }

    public function getCreatedAt(string $format = 'Y-m-d H:i:s')
    {
        // Convert to CodeIgniter\I18n\Time object
        $this->attributes['created_at'] = $this->mutateDate($this->
↪attributes['created_at']);

        $timezone = $this->timezone ?? app_timezone();

        $this->attributes['created_at']->setTimezone($timezone);

        return $this->attributes['created_at']->format($format);
    }
}
```

The first thing to notice is the name of the methods we' ve added. For each one, the class expects the snake_case column name to be converted into PascalCase, and prefixed with either `set` or `get`. These methods will then be automatically called whenever you set or retrieve the class property using the direct syntax (i.e. $user->email). The methods do not need to be public unless you want them accessed from other classes. For example, the `created_at` class property will be accessed through the `setCreatedAt()` and `getCreatedAt()` methods.

---

**注解:** This only works when trying to access the properties from outside of the class. Any methods internal to the class must call the `setX()` and `getX()` methods directly.

---

In the `setPassword()` method we ensure that the password is always hashed.

In `setCreatedAt()` we convert the string we receive from the model into a DateTime object, ensuring that our timezone is UTC so we can easily convert the viewer' s current timezone. In `getCreatedAt()`, it converts the time to a formatted string in the

application’ s current timezone.

While fairly simple, these examples show that using Entity classes can provide a very flexible way to enforce business logic and create objects that are pleasant to use.

```
// Auto-hash the password - both do the same thing
$user->password = 'my great password';
$user->setPassword('my great password');
```

### Data Mapping

At many points in your career, you will run into situations where the use of an application has changed and the original column names in the database no longer make sense. Or you find that your coding style prefers camelCase class properties, but your database schema required snake_case names. These situations can be easily handled with the Entity class’ data mapping features.

As an example, imagine you have the simplified User Entity that is used throughout your application:

```
<?php namespace App\Entities;

use CodeIgniter\Entity;

class User extends Entity
{
    protected $attributes = [
        'id' => null,
        'name' => null,         // Represents a username
        'email' => null,
        'password' => null,
        'created_at' => null,
        'updated_at' => null,
    ];
}
```

Your boss comes to you and says that no one uses usernames anymore, so you’re switching to just use emails for login. But they do want to personalize the application a bit, so they want you to change the name field to represent a user’ s full name now, not their username like it does currently. To keep things tidy and ensure things continue making sense in the database you whip up a migration to rename the *name* field to *full_name* for clarity.

Ignoring how contrived this example is, we now have two choices on how to fix the User class. We could modify the class property from `$name` to `$full_name`, but that would require changes throughout the application. Instead, we can simply map the `full_name` column in the database to the `$name` property, and be done with the Entity changes:

```
<?php namespace App\Entities;

use CodeIgniter\Entity;

class User extends Entity
{
    protected $attributes = [
        'id' => null,
        'name' => null,         // Represents a username
        'email' => null,
        'password' => null,
        'created_at' => null,
        'updated_at' => null,
    ];

    protected $datamap = [
        'full_name' => 'name'
    ],
}
```

By adding our new database name to the `$datamap` array, we can tell the class what class property the database column should be accessible through. The key of the array is the name of the column in the database, where the value in the array is class property to map it to.

In this example, when the model sets the `full_name` field on the User class, it actually assigns that value to the class' `$name` property, so it can be set and retrieved through `$user->name`. The value will still be accessible through the original `$user->full_name`, also, as this is needed for the model to get the data back out and save it to the database. However, `unset` and `isset` only work on the mapped property, `$name`, not on the original name, `full_name`.

### Mutators

### Date Mutators

By default, the Entity class will convert fields named *created_at*, *updated_at*, or *deleted_at* into *Time* instances whenever they are set or retrieved. The Time class provides a large number of helpful methods in an immutable, localized way.

You can define which properties are automatically converted by adding the name to the **options[ 'dates' ]** array:

```
<?php namespace App\Entities;

use CodeIgniter\Entity;
```

(下页继续)

(续上页)

```
class User extends Entity
{
    protected $dates = ['created_at', 'updated_at', 'deleted_at'];
}
```

Now, when any of those properties are set, they will be converted to a Time instance, using the application's current timezone, as set in **app/Config/App.php**:

```
$user = new \App\Entities\User();

// Converted to Time instance
$user->created_at = 'April 15, 2017 10:30:00';

// Can now use any Time methods:
echo $user->created_at->humanize();
echo $user->created_at->setTimezone('Europe/London')->toDateString();
```

### Property Casting

You can specify that properties in your Entity should be converted to common data types with the **casts** property. This option should be an array where the key is the name of the class property, and the value is the data type it should be cast to. Casting only affects when values are read. No conversions happen that affect the permanent value in either the entity or the database. Properties can be cast to any of the following data types: **integer**, **float**, **double**, **string**, **boolean**, **object**, **array**, **datetime**, and **timestamp**. Add a question mark at the beginning of type to mark property as nullable, i.e. **?string**, **?integer**.

For example, if you had a User entity with an **is_banned** property, you can cast it as a boolean:

```
<?php namespace App\Entities;

use CodeIgniter\Entity;

class User extends Entity
{
    protected $casts = [
        'is_banned' => 'boolean',
        'is_banned_nullable' => '?boolean'
    ],
}
```

### Array/Json Casting

Array/Json casting is especially useful with fields that store serialized arrays or json in them. When cast as:

- an **array**, they will automatically be unserialized,
- a **json**, they will automatically be set as an value of json_decode($value, false),
- a **json-array**, they will automatically be set as an value of json_decode($value, true),

when you read the property' s value. Unlike the rest of the data types that you can cast properties into, the:

- **array** cast type will serialize,
- **json** and **json-array** cast will use json_encode function on

the value whenever the property is set:

```
<?php namespace App\Entities;

use CodeIgniter\Entity;

class User extends Entity
{
    protected $casts => [
        'options' => 'array',
                'options_object' => 'json',
                'options_array' => 'json-array'
    ];
}

$user    = $userModel->find(15);
$options = $user->options;

$options['foo'] = 'bar';

$user->options = $options;
$userModel->save($user);
```

### Checking for Changed Attributes

You can check if an Entity attribute has changed since it was created. The only parameter is the name of the attribute to check:

```
$user = new User();
$user->hasChanged('name');      // false
```

(下页继续)

(续上页)

```
$user->name = 'Fred';
$user->hasChanged('name');      // true
```

Or to check the whole entity for changed values omit the parameter:

```
$user->hasChanged();            // true
```

## 6.3 管理数据库

CodeIgniter 用于重建或查看数据库的工具。

### 6.3.1 数据库工厂类

数据库工厂类（Database Forge）包含了帮助你管理你的数据库的一些相关方法。



#### 初始化 Forge 类

**重要：** 为了初始化 forge 类，你的数据库驱动程序必须已经在运行，因为 forge 类是依赖它运行的。

加载 Forge 类的代码如下:

```
$forge = \Config\Database::forge();
```

你可以将另外一个数据可组名传递给 DB Forge 加载程序，以防要管理的数据库不是默认数据库:

```
$this->myforge = $this->load->dbforge('other_db');
```

在上面的示例中，我们传递的是另一个数据库的名称作为第一个参数来连接。

### 创建和删除数据库

**$forge->createDatabase( 'db_name' )**

用于创建指定数据库，根据成败返回 TRUE 或 FALSE:

```
if ($forge->createDatabase('my_db'))
{
        echo 'Database created!';
}
```

**$forge->dropDatabase( 'db_name' )**

用于删除指定数据库，根据成败返回 TRUE 或 FALSE:

```
if ($forge->dropDatabase('my_db'))
{
        echo 'Database deleted!';
}
```

### 创建和删除数据表

在创建表时，你可能希望做一些事情。如添加字段，向表中添加键，更改列。CodeIgniter 为此提供了一种机制。

### 添加字段

字段是通过关联数组创建的。在数组中，必须包括与字段的数据类型相关的’type’键。例如，int、varchar、text 等。许多数据类型（例如 varchar）还需要“约束”键。

```
$fields = array(
        'users' => array(
                'type'       => 'VARCHAR',
                'constraint' => '100',
        ),
);
// 添加字段时将转换为"users VARCHAR(100)"。
```

此外，可以使用以下键/值:

- unsigned/true：在字段定义中生成“UNSIGNED”。
- default/value：在字段定义中生成默认值。
- null/true：在字段定义中生成”NULL”。如果没有这个，该字段将默认为”NOT NULL”。
- auto_increment/true：在字段上生成 auto_increment 标志。请注意，字段类型必须是支持此类型的类型，例如整数。
- unique/true：为字段定义生成唯一键。

```
$fields = array(
        'blog_id'          => array(
                'type'           => 'INT',
                'constraint'     => 5,
                'unsigned'       => TRUE,
                'auto_increment' => TRUE
        ),
        'blog_title'       => array(
                'type'           => 'VARCHAR',
                'constraint'     => '100',
                'unique'         => TRUE,
        ),
        'blog_author'      => array(
                'type'           =>'VARCHAR',
                'constraint'     => '100',
                'default'        => 'King of Town',
        ),
        'blog_description' => array(
                'type'           => 'TEXT',
                'null'           => TRUE,
        ),
);
```

定义字段后，可以使用 `$forge->addField($fields);` 然后调用 `createTable()` 方法。

**$forge->addField()**

add fields 方法将接受上述数组。

### 将字符串作为字段传递

如果你确切知道要如何创建字段，可以使用 addField() 方法将字符串传递给字段定义

```
$forge->addField("label varchar(100) NOT NULL DEFAULT 'default label'");
```

**注解：** 将原始字符串作为字段传递后，不能用 add_key() 对这些字段进行调用。

**注解：** 对 add_field() 的多次调用是累积的。

### 创建一个 id 字段

创建 id 字段有一个特殊例外。具有类型 id 的字段将自动分配为 INT(9) auto_incrementing 主键。

```
$forge->addField('id');
// 提出 id INT(9) NOT NULL AUTO_INCREMENT
```

### 添加键

通常来说，表都会有键。这可以使用 $forge->addKey( ‘field’ ) 方法来实现。第二个参数设置是可选的，设置为 TRUE 将使其成为主键，第三个参数设置为 TRUE 将使其成为唯一键。注意 addKey() 方法必须紧跟在 createTable() 方法后面。

包含多列的非主键必须使用数组来添加，下面是 MySQL 的例子。

```
$forge->addKey('blog_id', TRUE);
// gives PRIMARY KEY `blog_id` (`blog_id`)

$forge->addKey('blog_id', TRUE);
$forge->addKey('site_id', TRUE);
// gives PRIMARY KEY `blog_id_site_id` (`blog_id`, `site_id`)

$forge->addKey('blog_name');
// gives KEY `blog_name` (`blog_name`)

$forge->addKey(array('blog_name', 'blog_label'));
// gives KEY `blog_name_blog_label` (`blog_name`, `blog_label`)

$forge->addKey(array('blog_id', 'uri'), FALSE, TRUE);
// gives UNIQUE KEY `blog_id_uri` (`blog_id`, `uri`)
```

为了使代码读取更加客观，还可以使用特定的方法添加主键和唯一键。:

```
$forge->addPrimaryKey('blog_id');
// gives PRIMARY KEY `blog_id` (`blog_id`)
```

外键有助于跨表强制执行关系和操作。对于支持外键的表，可以直接在 forge 中添加它们。:

```
$forge->addUniqueKey(array('blog_id', 'uri'));
// gives UNIQUE KEY `blog_id_uri` (`blog_id`, `uri`)
```

### 添加外键

```
$forge->addForeignKey('users_id','users','id');
// gives CONSTRAINT `TABLENAME_users_foreign` FOREIGN KEY(`users_id`)␣
↪REFERENCES `users`(`id`)
```

你可以为约束的“on delete”和“on update”属性指定所需的操作:

```
$forge->addForeignKey('users_id','users','id','CASCADE','CASCADE');
// gives CONSTRAINT `TABLENAME_users_foreign` FOREIGN KEY(`users_id`)␣
↪REFERENCES `users`(`id`) ON DELETE CASCADE ON UPDATE CASCADE
```

### 创建表格

声明字段和键后，你可以根据如下代码创建一张新表

```
$forge->createTable('table_name');
// gives CREATE TABLE table_name
```

可选的第二个参数设置为 TRUE 时会在定义中添加”IF NOT EXISTS”子句

```
$forge->createTable('table_name', TRUE);
// gives CREATE TABLE IF NOT EXISTS table_name
```

你还可以传递可选的表属性，例如 MySQL 的 ENGINE:

```
$attributes = array('ENGINE' => 'InnoDB');
$forge->createTable('table_name', FALSE, $attributes);
// produces: CREATE TABLE `table_name` (...) ENGINE = InnoDB DEFAULT␣
↪CHARACTER SET utf8 COLLATE utf8_general_ci
```

**注解:** 除非你指定 CHARACTER SET 和/或 COLLATE 属性, createTable() 否则将始终使用你配置的 *charset* 和 *DBCollat* 值, 只要它们不为空 (仅限 MySQL).

### 删除表

执行 DROP TABLE 语句时，可以选择添加一个 IF EXISTS 子句。

```
// Produces: DROP TABLE table_name
$forge->dropTable('table_name');

// Produces: DROP TABLE IF EXISTS table_name
$forge->dropTable('table_name',TRUE);
```

### 删除外键

执行一个删除外键语句。

```
// Produces: ALTER TABLE 'tablename' DROP FOREIGN KEY 'users_foreign'
$forge->dropForeignKey('tablename','users_foreign');
```

**注解：** SQLite 数据库驱动程序不支持删除外键。

### 重命名表

执行表重命名

```
$forge->renameTable('old_table_name', 'new_table_name');
// gives ALTER TABLE old_table_name RENAME TO new_table_name
```

### 修改表

#### 向表中添加列

**$forge->addColumn()**

使用 addColumn() 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，并且可以用于无限数量的附加字段。

```
$fields = array(
        'preferences' => array('type' => 'TEXT')
);
$forge->addColumn('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

#### $forge->dropColumn()

该语句用于从表中删除列。

```
$forge->dropColumn('table_name', 'column_to_drop');
```

### 从表中的修改列

#### $forge->modifyColumn()

此方法的用法与“add_column()“相同，只是它是更改现有列，而不是添加新列。为了更改名称，可以将“名称”键添加到字段定义数组中。

```
$fields = array(
        'old_name' => array(
                'name' => 'new_name',
                'type' => 'TEXT',
        ),
);
$forge->modifyColumn('table_name', $fields);
// gives ALTER TABLE table_name CHANGE old_name new_name TEXT
```

### 类引用

class CodeIgniterDatabaseForge

addColumn($table[, $field = array()])

**参数**

- **$table** (*string*) – Table name to add the column to

- **$field** (*array*) – Column definition(s)

**返回** TRUE on success, FALSE on failure

**返回类型** bool

Adds a column to a table. Usage: See **'Adding a Column to a Table'_**.

**addField**(*$field*)

**参数**

- **$field** (*array*) – Field definition to add

**返回** CodeIgniterDatabaseForge instance (method chaining)

**返回类型** *CodeIgniterDatabaseForge*

Adds a field to the set that will be used to create a table. Usage: See **'Adding fields'_**.

**addKey**(*$key*[, *$primary* = *FALSE*[, *$unique* = *FALSE*]])

**参数**

- **$key** (*mixed*) – Name of a key field or an array of fields
- **$primary** (*bool*) – Set to TRUE if it should be a primary key or a regular one
- **$unique** (*bool*) – Set to TRUE if it should be a unique key or a regular one

**返回** CodeIgniterDatabaseForge instance (method chaining)

**返回类型** *CodeIgniterDatabaseForge*

Adds a key to the set that will be used to create a table. Usage: See **'Adding Keys'_**.

**addPrimaryKey**(*$key*)

**参数**

- **$key** (*mixed*) – Name of a key field or an array of fields

**返回** CodeIgniterDatabaseForge instance (method chaining)

**返回类型** *CodeIgniterDatabaseForge*

Adds a primary key to the set that will be used to create a table. Usage: See **'Adding Keys'_**.

**addUniqueKey**(*$key*)

**参数**

- **$key** (*mixed*) – Name of a key field or an array of fields

**返回** CodeIgniterDatabaseForge instance (method chaining)

**返回类型** *CodeIgniterDatabaseForge*

Adds an unique key to the set that will be used to create a table. Usage: See ‘Adding Keys‘__.

`createDatabase`(*$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

Creates a new database. Usage: See ‘Creating and Dropping Databases‘__.

`createTable`(*$table*[, *$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

Creates a new table. Usage: See ‘Creating a table‘__.

`dropColumn`(*$table*, *$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

Drops a column from a table. Usage: See ‘Dropping a Column From a Table‘__.

`dropDatabase`(*$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

Drops a database. Usage: See ‘Creating and Dropping Databases‘__.

`dropTable`(*$table_name*[, *$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

Drops a table. Usage: See **'Dropping a table'_**.

`modifyColumn`(*$table*, *$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

Modifies a table column. Usage: See **'Modifying a Column in a Table'_**.

`renameTable`(*$table_name*, *$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

Renames a table. Usage: See **'Renaming a table'_**.

### 6.3.2 数据库迁移

迁移是一种有条理、有组织的方式更改数据库的便捷方式。你可以手动编辑 SQL 的片段，然后你要负责告诉其他开发人员他们也需要去运行这段 SQL。你还必须跟踪下次部署时需要对生产机器运行哪些更改。

数据库表 ** 迁移 ** 会跟踪已经运行的迁移，因此您只需更新应用程序文件并调用 $migration->current() 以确定应运行哪些迁移。当前版本位于 **application/Config/Migrations.php** 中。

- 迁移文件名
- 创建迁移
- **使用 $currentVersion**

    - 数据库组
    - 命名空间
    - 用法示例
- 命令行工具
- 迁移参数
- 类参考

### 迁移文件名

每个迁移都按数字顺序向前或向后运行，具体取决于所采用的方法。有两种编号样式可供选择：

- **顺序**：每个迁移按顺序编号，从 001 开始。每个数字必须是三位数，并且序列中不得有任何间隙。(这是 CodeIgniter 3.0 之前的编号方案。)
- **时间戳**：使用创建迁移时的时间戳对每个迁移进行编号，格式为 **YYYYMMDDHHIES** 格式（例如 **20121031100537**）。这有助于防止在团队环境中工作时出现编号冲突，并且是 CodeIgniter 3.0 及更高版本中的首选方案。

可以使用 *application/Config/Migrations.php* 文件中的 $type 设置选择所需的样式。默认设置为时间戳。

无论您选择使用哪种编号样式，请在迁移文件前加上迁移编号，后跟下划线和迁移的描述性名称。例如：

- 001_add_blog.php（顺序编号）
- 20121031100537_add_blog.php（时间戳编号）

### 创建迁移

这将是新博客站点的首次迁移。所有迁移都在 **application/Database/Migrations/** 目录中，并命名，如 *20121031100537_Add_blog.php*。

```
<?php namespace App\Database\Migrations;

use CodeIgniter\Database\Migration;

class AddBlog extends Migration
{

        public function up()
        {
                $this->forge->addField([
                        'blog_id'           => [
                                'type'           => 'INT',
```

(下页继续)

(续上页)

```
                                'constraint'     => 5,
                                'unsigned'       => true,
                                'auto_increment' => true,
                        ],
                        'blog_title'       => [
                                'type'           => 'VARCHAR',
                                'constraint'     => '100',
                        ],
                        'blog_description' => [
                                'type'           => 'TEXT',
                                'null'           => true,
                        ],
                ]);
                $this->forge->addKey('blog_id', true);
                $this->forge->createTable('blog');
        }

        public function down()
        {
                $this->forge->dropTable('blog');
        }
}
```

然后在 **application/Config/Migrations.php** 中设置 $currentVersion = 20121031100537;。

数据库连接和数据库 Forge 类都可以通过 $this->db 和 $this->forge 分别使用。

或者，你可以使用命令行调用来生成框架迁移文件。请参阅下面的更多细节。

### 使用 $currentVersion

$currentVersion 设置允许你标记应用程序命名空间应设置的位置。这对于在生产环境中使用尤其有用。在你的应用程序中，你始终可以将迁移更新到当前版本，而不是最新版本，以确保生产和登台服务器正在运行正确的架构。在开发服务器上，你可以为尚未准备好生产的代码添加其他迁移。通过使用该 latest() 方法，你可以确保你的开发机器始终运行前沿架构。

### 数据库组

只能针对单个数据库组运行迁移。如果 **在 application/Config/Database.php** 中定义了多个组，则它将针对该 $defaultGroup 同一配置文件中指定的组运行。有时你可能需要为不同的数据库组使用不同的模式。也许你有一个用于所有常规站点信息的数据库，而另一个数据库用于关键任务数据。通过 $DBGroup 在迁移上设置属性，可以确保仅针对正确的组运行迁移。此名称必须与数据库组的名称完全匹配:

```
class Migration_Add_blog extends \CodeIgniter\Database\Migration
{
  protected $DBGroup = 'alternate_db_group';

  public function up() { . . . }

  public function down() { . . . }
}
```

### 命名空间

迁移库可以自动扫描你在 **application/Config/Autoload.php** 中定义的所有名称空间及其 $psr4 属性以匹配目录名称。它将包括它在 Database/Migrations 中找到的所有迁移。

每个命名空间都有自己的版本序列，这将帮助您升级和降级每个模块（命名空间），而不会影响其他命名空间。

例如，假设我们在 Autoload 配置文件中定义了以下命名空间:

```
$psr4 = [
        'App'        => APPPATH,
        'MyCompany' => ROOTPATH.'MyCompany'
];
```

这将查找位于 **APPPATH/Database/Migrations** 和 **ROOTPATH/Database/Migrations** 的任何迁移。这使得在可重用的模块化代码套件中包含迁移变得简单。

### 用法示例

在此示例中，一些简单的代码放在 **application/Controllers/Migrate.php** 中以更新架构:

```
<?php

class Migrate extends \CodeIgniter\Controller
{

        public function index()
        {
                $migrate = \Config\Services::migrations();

                try
                {
                $migrate->current();
```

(下页继续)

(续上页)

```
                }
                catch (\Exception $e)
                {
                  // Do something with the error here...
                }
        }

}
```

### 命令行工具

CodeIgniter 附带了几个:doc:*commands </cli/cli_commands>*，它们可以从命令行获得，以帮助你处理迁移。这些工具不需要使用迁移，但可能会使那些希望使用它们的人更容易。这些工具主要提供对 MigrationRunner 类中可用的相同方法的访问。

**migrate**

Migrates a database group with all available migrations:

```
> php spark migrate
```

You can use (migrate) with the following options:

- `-g` - to chose database group, otherwise default database group will be used.
- `-n` - to choose namespace, otherwise (App) namespace will be used.
- `-all` - to migrate all namespaces to the latest migration

This example will migrate Blog namespace with any new migrations on the test database group:

```
> php spark migrate -g test -n Blog
```

When using the `-all` option, it will scan through all namespaces attempting to find any migrations that have not been run. These will all be collected and then sorted as a group by date created. This should help to minimize any potential conflicts between the main application and any modules.

**rollback**

回滚所有迁移，将所有数据库组转为空白平板，有效迁移 0:

```
> php spark migrate:rollback
```

你可以使用（rollback）以下选项:

- (-g）选择数据库组，否则将使用默认数据库组。
- (-n）选择名称空间，否则将使用（App）名称空间。
- (all）将所有名称空间迁移到最新的迁移

**refresh**

首先回滚所有迁移，然后迁移到最新版本，刷新数据库状态:

```
> php spark migrate:refresh
```

你可以使用（refresh）以下选项：

- （-g）选择数据库组，否则将使用默认数据库组。
- （-n）选择名称空间，否则将使用（App）名称空间。
- （all）将所有名称空间迁移到最新的迁移

**status**

显示所有迁移的列表及其运行的日期和时间，如果尚未运行，则显示’–’：

```
> php spark migrate:status
Filename                Migrated On
First_migration.php     2016-04-25 04:44:22
```

你可以使用（status）以下选项：

- （-g）选择数据库组，否则将使用默认数据库组。

**make:migration**

Creates a skeleton migration file in **app/Database/Migrations**. It automatically prepends the current timestamp. The class name it creates is the Pascal case version of the filename.

```
> php spark make:migration <class> [options]
```

You can use (make:migration) with the following options:

- `-n` - to choose namespace, otherwise the value of `APP_NAMESPACE` will be used.
- `-force` - If a similarly named migration file is present in destination, this will be overwritten.

### 迁移参数

以下是 **app/Config/Migrations.php** 中提供的所有迁移配置选项的表。

| 参数 | 默认值 | 可选项 | 描述 |
|---|---|---|---|
| **enabled** | true | true / false | 启用或者禁用迁移 |
| **table** | migrations | None | 用于存储当前版本的数据库表名 |
| **timestampFormat** | Y-m-d-His_ | | The format to use for timestamps when creating a migration. |

### 类参考

class CodeIgniterDatabaseMigrationRunner

current(*$group*)

**参数**

- **$group** (*mixed*) – database group name, if null (App) namespace will be used.

**返回** TRUE if no migrations are found, current version string on success, FALSE on failure

**返回类型** mixed

Migrates up to the current version (whatever is set for `$currentVersion` in *application/Config/Migrations.php*).

findMigrations()

**返回** An array of migration files

**返回类型** array

An array of migration filenames are returned that are found in the **path** property.

latest(*$namespace*, *$group*)

**参数**

- **$namespace** (*mixed*) – application namespace, if null (App) namespace will be used.
- **$group** (*mixed*) – database group name, if null default database group will be used.

**返回** Current version string on success, FALSE on failure

**返回类型** mixed

This works much the same way as `current()` but instead of looking for the `$currentVersion` the Migration class will use the very newest migration found in the filesystem.

latestAll(*$group*)

**参数**

- **$group** (*mixed*) – database group name, if null default database group will be used.

**返回** TRUE on success, FALSE on failure

**返回类型** mixed

This works much the same way as `latest()` but instead of looking for one namespace, the Migration class will use the very newest migration found for all namespaces.

version(*$target_version*, *$namespace*, *$group*)

**参数**

- **$namespace** (*mixed*) – application namespace, if null (App) namespace will be used.
- **$group** (*mixed*) – database group name, if null default database group will be used.
- **$target_version** (*mixed*) – Migration version to pro-

cess

**返回** TRUE if no migrations are found, current version string on success, FALSE on failure

**返回类型** mixed

Version can be used to roll back changes or step forwards programmatically to specific versions. It works just like `current()` but ignores `$currentVersion`.

```
$migration->version(5);
```

`setNamespace`(*$namespace*)

**参数**

- **$namespace** (*string*) – application namespace.

**返回** The current MigrationRunner instance

**返回类型** *CodeIgniterDatabaseMigrationRunner*

Sets the path the library should look for migration files:

```
$migration->setNamespace($path)
          ->latest();
```

`setGroup`(*$group*)

**参数**

- **$group** (*string*) – database group name.

**返回** The current MigrationRunner instance

**返回类型** *CodeIgniterDatabaseMigrationRunner*

Sets the path the library should look for migration files:

```
$migration->setNamespace($path)
          ->latest();
```

### 6.3.3 数据填充

数据填充是一种简单的将数据添加到数据库的方式。这在开发的过程中特别有用，你只需要准备开发中所需要的示例数据填充到数据库中，而且不仅如此，这些数据可以包括你不想要包括的迁移的静态数据，例如国家/地区，地理编码表，事件或设置信息等等。

数据填充是必须有 **run()** 方法的简单类，并继承于 **CodeIgniterDatabaseSeeder** 。在 **run()** 中，该类可以创建你所需要的任何类型的数据。该类可以创建需要的任何形式的数据。它可以分别通过建立 `$this->db` 和 `$this->forge` 访问数据库连接。填充文件必须存储在 **application/Database/Seeds** 目录中。文件名和类名必须保持一致。

```
// application/Database/Seeds/SimpleSeeder.php
class SimpleSeeder extends \CodeIgniter\Database\Seeder
{
        public function run()
        {
                $data = [
```

(下页继续)

(续上页)

```
                        'username' => 'darth',
                        'email' => 'darth@theempire.com'
                ];

                // Simple Queries
                $this->db->query("INSERT INTO users (username, email)␣
↪VALUES(:username, :email)",
                        $data
                );

                // Using Query Builder
                $this->db->table('users')->insert($data);
        }
}
```

### 嵌套数据填充

你可以使用 **call()** 方法来运行其他的 seed 类。这允许你更容易使用 seeder，而且同时也将任务分发到各个 seeder 文件当中:

```
class TestSeeder extends \CodeIgniter\Database\Seeder
{
        public function run()
        {
                $this->call('UserSeeder');
                $this->call('CountrySeeder');
                $this->call('JobSeeder');
        }
}
```

你也可以在 **call()** 方法中使用完全合格的类名，使你的 seeder 在任何地方都可以更好的加载。这对于更多模块化代码库来说非常方便:

```
public function run()
{
        $this->call('UserSeeder');
        $this->call('My\Database\Seeds\CountrySeeder');
}
```

### 使用 Seeders

你可以通过数据库配置类获取主 seeder:

```
$seeder = \Config\Database::seeder();
$seeder->call('TestSeeder');
```

### 命令行填充数

如果不想创建专用控制器，也可以从命令行填充数据，作为 Migrations CLI 工具的一部分:

```
> php index.php migrations seed TestSeeder
```

# 章节 7

# 类库和辅助函数

## 7.1 类库参考

### 7.1.1 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）。



#### 示例代码

以下示例代码展示控制器中的常见使用模式。

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

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

echo $foo;
```

你可以通过 Services 类直接获取缓存引擎的实例:

```
$cache = \Config\Services::cache();

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

### 配置缓存

缓存引擎的所有配置都在 **application/Config/Cache.php** 文件中。在该文件中，以下项目可用。

**$handler**

$handler 处理器是启动引擎时应用作主处理程序。可用的名称有：dummy, file, memcached, redis, wincache。

**$backupHandler**

在第一选择 $hanlder 不可用的情况下，这是要加载的下一个缓存处理程序。这通常是**文件**处理程序，因为文件系统始终可用，但可能不适合更复杂的多服务器设置。

**$prefix**

如果您有多个应用程序使用相同的缓存存储，则可以在此处添加一个前缀到所有键名称的自定义前缀。

**$path**

`file` 处理程序使用它来显示应该将缓存文件保存到哪里。

**$memcached**

这是使用 `Memcache(d)` 处理程序时将使用的一系列服务器。

**$redis**

使用 `Redis` 处理程序时要使用的 Redis 服务器的设置。

### 类参考

`isSupported()`

**返回** 如果支持，则为 TRUE，否则为 FALSE

**返回类型** 布尔值

`get(`*$key*`)`

**参数**

- **$key** (*string*) – Cache 缓存项名称

**返回** 项目值或 FALSE 如果没有找到

**返回类型** mixed

此方法将尝试从缓存存储中获取项目。如果该项目不存在，该方法将返回 FALSE。

Example:

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

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

**参数**

- **$key** (*string*) – Cache item name
- **$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

此方法将会将项目保存到缓存存储。如果保存失败，该方法将返回 FALSE。

Example:

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

**注解:** 该 `$raw` 参数仅由 Memcache 使用，以便允许使用 `increment()` 和 `decrement()`。

`delete(`*$key*`)`

**参数**

- **$key** (*string*) – name of cached item

**返回** TRUE on success, FALSE on failure

**返回类型** bool

此方法将从缓存存储中删除特定项目。如果项目删除失败，该方法将返回 FALSE。

Example:

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

**increment**(*$key*[, *$offset = 1*])

**参数**

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

**返回** New value on success, FALSE on failure

**返回类型** mixed

Performs atomic incrementation of a raw stored value. 执行原始存储值的原子增量

Example:

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

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

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

**decrement**(*$key*[, *$offset = 1*])

**参数**

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

**返回** New value on success, FALSE on failure

**返回类型** mixed

执行原始存储值的原子递减。

Example:

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

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

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

**clean**()

**返回** TRUE on success, FALSE on failure

**返回类型** bool

此方法将‘clean’整个缓存。如果缓存文件的删除失败，该方法将返回 FALSE。
Example:

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

**cache_info()**

**返回** Information on the entire cache database

**返回类型** mixed

此方法将返回整个缓存中的信息。

Example:

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

**注解：** 返回的信息和数据的结构取决于正在使用的适配器。

**getMetadata(*$key*)**

**参数**

- **$key** (*string*) – Cache item name

**返回** Metadata for the cached item

**返回类型** mixed

此方法将返回缓存中特定项目的详细信息。

Example:

```
var_dump($cache->getMetadata('my_cached_item'));
```

**注解：** 返回的信息和数据的结构取决于正在使用的适配器。

### 驱动器

#### 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。

#### Memcached 缓存

可以在缓存配置文件中指定多个 Memcached 服务器。

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。

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['host'] = '127.0.0.1';
$config['password'] = NULL;
$config['port'] = 6379;
$config['timeout'] = 0;
```

有关 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。

### 7.1.2 CURLRequest 类

`CURLRequest` 类是一个轻量级的基于 CURL 的 HTTP 客户端，用于同其他网站和服务器进行沟通。该类可用于获取谷歌搜索的内容，抓取一个网站页面或一个图片，或者是用来同 API 进行信息传递等诸多功能。

- *body*（请求体）
- *cert*（证书）
- *connect_timeout*（连接超时）
- *cookie*
- *debug*（调试 *bug*）
- *delay*（延时）
- *form_params*（表单参数）
- *headers*（请求头）
- *http_errors*（*http* 错误）
- *json*
- *multipart*
- *query*（查询语句）
- *timeout*（超时）
- *verify*（鉴权）
- *version*（版本）

该类模仿了 Guzzle HTTP Client 库，因为该库被广泛应用于多方面。我们尽可能地与 Guzzle 保持语法一致，不过如果你需要一些额外的功能的话（比如该类并未提供的功能之类的），可能需要稍微更改一下语法来使用 Guzzle 库。

---

**注解：** 该类需要安装你的 PHP 版本的 cURL 库 。该库是一个在大多数情况下都广泛被使用的库，但不是所有服务器都安装了它。因此请检查你的服务器上安装了该库以解决依赖问题。

---

### 加载该类库

该类库可以通过手动加载或者通过服务类 加载。

通过服务类来加载 `curlrequest()` 方法:

```
$client = \Config\Services::curlrequest();
```

你可以将一个默认选项数组作为参数传递给该方法作为第一个参数，用于修改 cURL 处理请求的方式。选项描述如下:

```
$options = [
        'base_uri' => 'http://example.com/api/v1/',
        'timeout'  => 3
```

(下页继续)

(续上页)

```
];
$client = \Config\Services::curlrequest($options);
```

当手动创建类实例时，你需要传递一些依赖。第一个参数是 Config\App 类的实例。第二个参数是一个 URI 实例。第三个参数是一个 Response 类的对象。第四个参数是一个可选的 $options 数组:

```
$client = new \CodeIgniter\HTTP\CURLRequest(
        new \Config\App(),
        new \CodeIgniter\HTTP\URI(),
        new \CodeIgniter\HTTP\Response(new \Config\App()),
        $options
);
```

### 使用该类库

处理 CURL 请求基本上只是创建一个 Request 请求并获取*Response 对象* 的过程。这一过程就是用来处理数据交换的。这一过程后，你可以对获得的信息进行完全自定义的处理.

### 发送请求

大多数交流会话是通过 request() 方法进行的，该方法触发请求并返回一个 Response 实例。该方法将 HTTP 动词，URL 信息和选项数组作为请求参数。:

```
$client = \Config\Services::curlrequest();

$response = $client->request('GET', 'https://api.github.com/user', [
        'auth' => ['user', 'pass']
]);
```

由于该响应是 CodeIgniter\HTTP\Response 类的一个实例对象，故而可以通过调用该类的对应方法:

```
echo $response->getStatusCode();
echo $response->getBody();
echo $response->getHeader('Content-Type');
$language = $response->negotiateLanguage(['en', 'fr']);
```

尽管 request() 方法非常灵活，你也可以使用以下的简称方法。这些方法将 URL 作为第一个参数并将选项数组作为第二个参数:

```
* $client->get('http://example.com');
* $client->delete('http://example.com');
* $client->head('http://example.com');
```

(下页继续)

（续上页）

```
* $client->options('http://example.com');
* $client->patch('http://example.com');
* $client->put('http://example.com');
* $client->post('http://example.com');
```

### base_uri （基础 URI ）

base_uri 可以在该类实例化时作为一个选项进行设置。该参数使得你可以设置一个基础 URI ，并在该实例对象进行请求时使用相对 URL 路径。这一操作在和 API 通信时特别管用:

```
$client = \Config\Services::curlrequest([
        'base_uri' => 'https://example.com/api/v1/'
]);

// GET http:example.com/api/v1/photos
$client->get('photos');

// GET http:example.com/api/v1/photos/13
$client->delete('photos/13');
```

当 request() 方法或者其他简称方法接受相对 URI 作为参数时，就会将 base_uri 和该相对 URI 根据 RFC 2986, section 2 进行组合以下是一些组合的例子

### 使用响应

每个 request() 函数调用都会返回一个包含有许多有用信息和方法的 Response 实例对象。最通用的方法使得你可以定制化地处理响应对象本身。

你可以获取响应的状态码以及状态原因:

```
$code   = $response->getStatusCode();     // 200
$reason = $response->getReason();       // OK
```

你可以获取响应头:

```
// 获取一个响应头的内容
echo $response->getHeaderLine('Content-Type');

// 获取所有响应头
foreach ($response->getHeaders() as $name => $value)
{
        echo $name .': '. $response->getHeaderLine($name) ."\n";
}
```

响应体可以通过 getBody() 方法来获取:

```
$body = $response->getBody();
```

响应体是远端服务器提供的原生响应内容。如果内容类型需要格式化的话，你需要保证在代码中这样处理:

```
if (strpos($response->getHeader('content-type'), 'application/json') !==␣
↪false)
{
        $body = json_decode($body);
}
```

### 请求选项

本节描述了在构造函数，request() 方法以及所有简称方法中可以传递的所有可用选项。

#### allow_redirects （运行重定向）

默认情况下，CURL 会遵循远端服务器返回的所有的 “Location:” 响应头规则。allow_redirects 选项使得你可以修改这一执行过程。

如果该值被设为 false ，就不会执行任何的重定向规则。

> $client->request( ‘GET’ , ‘http://example.com’ , [ ‘allow_redirects’ => false]);

设为 true 时就会执行请求的默认设置:

```
$client->request('GET', 'http://example.com', ['allow_redirects' =>␣
↪true]);

// 设置以下默认选项:
'max'       => 5, // 终止前最多的重定向次数
'strict'    => true, // 在重定向过程中确保发送的 POST 请求始终保持为 POST␣
↪ (译注: 某些服务器会在重定向时修改请求方法, 例如 304 重定向时修改请求方式
为 GET)
'protocols' => ['http', 'https'] // 限制重定向使用一个或多个协议
```

你可以为 allow_redirects 选项传递一个选项数组用于重定向时使用新的设置，而不是默认设置:

```
$client->request('GET', 'http://example.com', ['allow_redirects' => [
        'max'       => 10,
        'protocols' => ['https'] // Force HTTPS domains only.
]]);
```

**注解：** 当 PHP 在 safe_mode 或者 open_basedir 选项开启时，不会进行重定向。

### auth （认证）

使得你可以为 HTTP Basic 和 Digest 和认证过程提供细节信息。你的脚本文件需要执行额外操作以支持诊断认证——只需要在访问时传递用户名和密码。第三个参数是认证的类型，可以是 basic 或者 digest:

```
$client->request('GET', 'http://example.com', ['auth' => ['username',
↪'password', 'digest']]);
```

### body （请求体）

对于支持请求体的方法，例如 PUT 或者是 POST 来说，有两种方法来设置请求体。第一种是使用 setBody() 方法:

```
$client->setBody($body)
       ->request('put', 'http://example.com');
```

第二种方法是通过传递一个 body 选项。该方式是为了与 Guzzle 兼容起见的，并提供了和上述方式一样的功能。该值必须是一个字符串:

```
$client->request('put', 'http://example.com', ['body' => $body]);
```

### cert（证书）

指定一个 PEM 格式的客户端证书的位置，通过为 cert 选项来传递绝对路径的方式来实现。如果需要密码的话，为该选项数组的第一个元素的值为路径，第二个元素的值设为密码:

```
$client->request('get', '/', ['cert' => ['/path/getServer.pem', 'password
↪']);
```

### connect_timeout （连接超时）

默认情况下，CodeIgniter 并未对 cURL 尝试连接一个网站的时间进行限制。如果你需要修改这个值，可以通过为 connect_timeout 选项提供时间秒数值的方式来进行。传值为 0 时，无限等待:

```
$response->request('GET', 'http://example.com', ['connect_timeout' =>␣
↪0]);
```

### cookie

该选项指定了 CURL 用于存取 cookie 值的文件名。这一过程通过使用 CURL_COOKIEJAR 和 CURL_COOKIEFILE 选项来实现。例如:

```
$response->request('GET', 'http://example.com', ['cookie' => WRITEPATH .
↪'CookieSaver.txt']);
```

### debug （调试 bug）

当 debug 被传递并设为 true 时，就会启动额外的调试模式并在脚本执行时输出标准错误流信息 ( STDERR )。该操作是通过传递 CURLOPT_VERBOSE 并返回输出来实现的。因此当你需要利用 spark serve 运行一个内置服务器时，将会看到命令行中的输出内容。否则输出就会被写入到服务器的错误日志中:

```
$response->request('GET', 'http://example.com', ['debug' => true]);
```

可以通过将文件名作为参数传入的方式，将输出写入到文件中:

```
$response->request('GET', 'http://example.com', ['debug' => '/usr/local/
↪curl_log.txt']);
```

### delay （延时）

使得你可以在发送请求前延迟指定的毫秒时间:

```
// 延时 2 秒
$response->request('GET', 'http://example.com', ['delay' => 2000]);
```

### form_params （表单参数）

你可以通过为 form_params 选项传递关联数组的方式，在一个 application/x-www-form-urlencoded POST 请求里发送表单数据。该操作会将 Content-Type 请求头强制设为 application/x-www-form-urlencoded

```
$client->request('POST', '/post', [
        'form_params' => [
                'foo' => 'bar',
                'baz' => ['hi', 'there']
        ]
]);
```

**注解:** form_params 不能和 multipart 选项一起使用。你可以非此即彼地使用这两个选项。form_params 用于 application/x-www-form-urlencoded 请求，而 multipart

用于 multipart/form-data 请求。

### headers （请求头）

尽管你可以通过 setHeader() 方法来传递任何请求头，你也可以通过为选项传递关联数组作为参数的方式来实现自定义请求头。该关联数组中每个键都是请求头的名字，而值就是一个字符串或者是一个字符串数组，包括着请求头字段的值:

```
$client->request('get', '/', [
        'headers' => [
                'User-Agent' => 'testing/1.0',
                'Accept'     => 'application/json',
                'X-Foo'      => ['Bar', 'Baz']
        ]
]);
```

如果请求头在构造函数中被传入时，就会被设为默认选项。而默认选项会被后续设置的选项或者 setHeader() 的调用所覆盖。

### http_errors （http 错误）

默认情况下，CURLRequest 类会在 HTTP 状态码大于等于 400 时结束请求并报错。你可以通过将 http_errors 选项设为 false 的方式来返回内容:

```
$client->request('GET', '/status/500');
// 自动失败报错

$res = $client->request('GET', '/status/500', ['http_errors' => false]);
echo $res->getStatusCode();
// 500
```

### json

json 选项用于上传 JSON 编码的数据作为请求体。同时会在请求头上加入 Content-Type 为 application/json 。并覆盖先前设置的 Content-Type 请求头。传递给该选项的参数可以是任何 json_encode() 函数所接受的参数:

```
$response = $client->request('PUT', '/put', ['json' => ['foo' => 'bar
↪']]);
```

**注解：** 该选项不允许对 json_encode() 和 Content-Type 请求头进行自定义地修改。如果你需要这一功能，就需要手动编码数据并将其传递给 CURLRequest 类的 setBody() 方法，并通过 setHeader() 方法来设置 Content-Header 请求头。

### multipart

如果你想通过 POST 请求来发送文件或者其他数据时，可以使用 multipart 选项和 CURLFile 类 。该选项的值应当是一个需要关联数组，包含有需要发送的数据。为了安全起见，上传文件时在前缀上加上 @ 的遗留方法已被禁止。你所需要发送的文件应当以 CURLFile 类的实例的方式传递:

```
$post_data = [
        'foo'      => 'bar',
        'userfile' => new \CURLFile('/path/to/file.txt')
];
```

**注解:** multipart 不能和 form_params 选项一起使用。你可以非此即彼地使用这两个选项。form_params 用于 application/x-www-form-urlencoded 请求，而 multipart 用于 multipart/form-data 请求。

### query （查询语句）

你可以通过为 query 选项传递一个关联数组的方式来发送查询字符串信息:

```
// 发送一个 GET 请求来获取 /get?foo=bar 的结果
$client->request('GET', '/get', ['query' => ['foo' => 'bar']]);
```

### timeout（超时）

默认情况下，cURL 函数可以执行任意长的时间，不受时间限制。你可以通过 timeout 选项来修改这一过程。选项值是你需要这个函数运行的时间。使用 0 来无限等待:

```
$response->request('GET', 'http://example.com', ['timeout' => 5]);
```

### verify（鉴权）

该选项描述了 SSL 验证鉴权行为。如果 verify 选项被设为 true ，就开始 SSL 鉴权操作并使用系统提供默认的 CA 包文件。如果设为 false ，就会禁用鉴权操作（这一行为不安全，并可能导致中间人攻击!）。你可以将该参数设为一个 CA 包文件所在的路径，从而进行自定义的鉴权操作。该选项默认值为 true

```
// 使用系统的 CA 包文件（默认设置）
$client->request('GET', '/', ['verify' => true]);

// 使用硬盘上的一个自定义的 SSL 鉴权文件
$client->request('GET', '/', ['verify' => '/path/to/cert.pem']);
```

（下页继续）

(续上页)

```
// 完全禁用鉴权（不安全!）
$client->request('GET', '/', ['verify' => false]);
```

#### version（版本）

你可以通过为版本参数传递一个字符串或者浮点数（特别是 1.0，或 1.1，尚未支持 2.0）的方式来设置协议版本:

```
// 强制使用 HTTP/1.0
$client->request('GET', '/', ['version' => 1.0]);
```

### 7.1.3 Email Class

CodeIgniter’s robust Email Class supports the following features:

- Multiple Protocols: Mail, Sendmail, and SMTP
- TLS and SSL Encryption for SMTP
- Multiple recipients
- CC and BCCs
- HTML or Plaintext email
- Attachments
- Word wrapping
- Priorities
- BCC Batch Mode, enabling large email lists to be broken into small BCC batches.
- Email Debugging tools

### Using the Email Library

### Sending Email

Sending email is not only simple, but you can configure it on the fly or set your preferences in the **app/Config/Email.php** file.

Here is a basic example demonstrating how you might send email:

```
$email = \Config\Services::email();

$email->setFrom('your@example.com', 'Your Name');
$email->setTo('someone@example.com');
$email->setCC('another@another-example.com');
$email->setBCC('them@their-example.com');

$email->setSubject('Email Test');
$email->setMessage('Testing the email class.');

$email->send();
```

### Setting Email Preferences

There are 21 different preferences available to tailor how your email messages are sent. You can either set them manually as described here, or automatically via preferences stored in your config file, described below:

Preferences are set by passing an array of preference values to the email initialize method. Here is an example of how you might set some preferences:

```
$config['protocol'] = 'sendmail';
$config['mailPath'] = '/usr/sbin/sendmail';
$config['charset']  = 'iso-8859-1';
$config['wordWrap'] = true;

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

**注解:** Most of the preferences have default values that will be used if you do not set them.

### Setting Email Preferences in a Config File

If you prefer not to set preferences using the above method, you can instead put them into the config file. Simply open the **app/Config/Email.php** file, and set your configs in the Email properties. Then save the file and it will be used automatically. You will

NOT need to use the `$email->initialize()` method if you set your preferences in the config file.

### Email Preferences

The following is a list of all the preferences that can be set when sending email.

| Preference | Default Value | Options | Description |
|---|---|---|---|
| **userAgent** | CodeIgniter | None | The "user agent". |
| **protocol** | mail | mail, sendmail, or smtp | The mail sending protocol. |
| **mailpath** | /usr/sbin/sendmail | None | The server path to Sendmail. |
| **SMTPHost** | No Default | None | SMTP Server Address. |
| **SMTPUser** | No Default | None | SMTP Username. |
| **SMTPPass** | No Default | None | SMTP Password. |
| **SMTPPort** | 25 | None | SMTP Port. |
| **SMTPTimeout** | 5 | None | SMTP Timeout (in seconds). |
| **SMTPKeepAlive** | FALSE | TRUE or FALSE (boolean) | Enable persistent SMTP connections. |
| **SMTPCrypto** | No Default | tls or ssl | SMTP Encryption |
| **wordWrap** | TRUE | TRUE or FALSE (boolean) | Enable word-wrap. |
| **wrapChars** | 76 | | Character count to wrap at. |
| **mailType** | text | text or html | Type of mail. If you send HTML email you must send it as a complete web page. Make sure you don't have any relative links or relative image paths otherwise they will not work. |
| **charset** | utf-8 | | Character set (utf-8, iso-8859-1, etc.). |
| **validate** | TRUE | TRUE or FALSE (boolean) | Whether to validate the email address. |
| **priority** | 3 | 1, 2, 3, 4, 5 | Email Priority. 1 = highest. 5 = lowest. 3 = normal. |
| **CRLF** | \n | "\r\n" or "\n" | Newline character. (Use "\r\n" to comply with RFC 822). |

### Overriding Word Wrapping

If you have word wrapping enabled (recommended to comply with RFC 822) and you have a very long link in your email it can get wrapped too, causing it to become un-clickable by the person receiving it. CodeIgniter lets you manually override word wrapping within part of your message like this:

```
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.
```

Place the item you do not want word-wrapped between: {unwrap} {/unwrap}

### Class Reference

**CodeIgniter\Email\Email**

**setFrom**(*$from*[, *$name* = ''[, *$returnPath* = *null*]])

**参数**

- **$from** (*string*) – "From" e-mail address
- **$name** (*string*) – "From" display name
- **$returnPath** (*string*) – Optional email address to redirect undelivered e-mail to

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Sets the email address and name of the person sending the email:

```
$email->setFrom('you@example.com', 'Your Name');
```

You can also set a Return-Path, to help redirect undelivered mail:

```
$email->setFrom('you@example.com', 'Your Name', 'returned_
↪emails@example.com');
```

**注解:** Return-Path can't be used if you've configured 'smtp' as your protocol.

setReplyTo(*$replyto*[, *$name* = ''])

**参数**

- **$replyto** (*string*) – E-mail address for replies
- **$name** (*string*) – Display name for the reply-to e-mail address

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Sets the reply-to address. If the information is not provided the information in the *setFrom* method is used. Example:

```
$email->setReplyTo('you@example.com', 'Your Name');
```

setTo(*$to*)

**参数**

- **$to** (*mixed*) – Comma-delimited string or an array of e-mail addresses

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Sets the email address(s) of the recipient(s). Can be a single e-mail, a comma-delimited list or an array:

```
$email->setTo('someone@example.com');
```

```
$email->setTo('one@example.com, two@example.com, three@example.
↪com');
```

```
$email->setTo(['one@example.com', 'two@example.com',
↪'three@example.com']);
```

setCC(*$cc*)

**参数**

- **$cc** (*mixed*) – Comma-delimited string or an array of e-mail addresses

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Sets the CC email address(s). Just like the "to", can be a single e-mail, a comma-delimited list or an array.

setBCC(*$bcc*[, *$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** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Sets the BCC email address(s). Just like the `setTo()` method, can be a single e-mail, a comma-delimited list or an array.

If `$limit` is set, "batch mode" will be enabled, which will send the emails to batches, with each batch not exceeding the specified `$limit`.

`setSubject`(*$subject*)

**参数**

- **$subject** (*string*) – E-mail subject line

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Sets the email subject:

```
$email->setSubject('This is my subject');
```

`setMessage`(*$body*)

**参数**

- **$body** (*string*) – E-mail message body

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Sets the e-mail message body:

```
$email->setMessage('This is my message');
```

`setAltMessage`(*$str*)

**参数**

- **$str** (*string*) – Alternative e-mail message body

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Sets the alternative e-mail message body:

```
$email->setAltMessage('This is the alternative message');
```

This is an optional message string which can be used if you send HTML formatted email. It lets you specify an alternative message with no HTML

formatting which is added to the header string for people who do not accept HTML email. If you do not set your own message CodeIgniter will extract the message from your HTML email and strip the tags.

**setHeader**(*$header*, *$value*)

**参数**

- **$header** (*string*) – Header name
- **$value** (*string*) – Header value

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Appends additional headers to the e-mail:

```
$email->setHeader('Header1', 'Value1');
$email->setHeader('Header2', 'Value2');
```

**clear**(*$clearAttachments = false*)

**参数**

- **$clearAttachments** (*bool*) – Whether or not to clear attachments

**返回** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Initializes all the email variables to an empty state. This method is intended for use if you run the email sending method in a loop, permitting the data to be reset between cycles.

```
foreach ($list as $name => $address)
{
        $email->clear();

        $email->setTo($address);
        $email->setFrom('your@example.com');
        $email->setSubject('Here is your info '.$name);
        $email->setMessage('Hi ' . $name . ' Here is the info␣
↪you requested.');
        $email->send();
}
```

If you set the parameter to TRUE any attachments will be cleared as well:

```
$email->clear(true);
```

**send**(*$autoClear = true*)

**参数**

- **$autoClear** (*bool*) – Whether to clear message data automatically

**返回** TRUE on success, FALSE on failure

**返回类型** bool

The e-mail sending method. Returns boolean TRUE or FALSE based on success or failure, enabling it to be used conditionally:

```
if (! $email->send())
{
        // Generate error
}
```

This method will automatically clear all parameters if the request was successful. To stop this behaviour pass FALSE:

```
if ($email->send(false))
{
        // Parameters won't be cleared
}
```

---

**注解:** In order to use the `printDebugger()` method, you need to avoid clearing the email parameters.

---

---

**注解:** If `BCCBatchMode` is enabled, and there are more than `BCCBatchSize` recipients, this method will always return boolean `TRUE`.

---

**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** CodeIgniter\Email\Email instance (method chaining)

**返回类型** CodeIgniter\Email\Email

Enables you to send an attachment. Put the file path/name in the first parameter. For multiple attachments use the method multiple times. For example:

```
$email->attach('/path/to/photo1.jpg');
$email->attach('/path/to/photo2.jpg');
$email->attach('/path/to/photo3.jpg');
```

To use the default disposition (attachment), leave the second parameter blank, otherwise use a custom disposition:

```
$email->attach('image.jpg', 'inline');
```

You can also use a URL:

```
$email->attach('http://example.com/filename.pdf');
```

If you' d like to use a custom file name, you can use the third parameter:

```
$email->attach('filename.pdf', 'attachment', 'report.pdf');
```

If you need to use a buffer string instead of a real - physical - file you can use the first parameter as buffer, the third parameter as file name and the fourth parameter as mime-type:

```
$email->attach($buffer, 'attachment', 'report.pdf',
↪'application/pdf');
```

**setAttachmentCID**(*$filename*)

**参数**

- **$filename** (*string*) – Existing attachment filename

**返回** Attachment Content-ID or FALSE if not found

**返回类型** string

Sets and returns an attachment's Content-ID, which enables your to embed an inline (picture) attachment into HTML. First parameter must be the already attached file name.

```
$filename = '/img/photo1.jpg';
$email->attach($filename);
foreach ($list as $address)
{
        $email->setTo($address);
        $cid = $email->setAttachmentCID($filename);
        $email->setMessage('<img src="cid:'. $cid .'" alt=
↪"photo1" />');
        $email->send();
}
```

**注解：** Content-ID for each e-mail must be re-created for it to be unique.

`printDebugger`(*$include = ['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

Returns a string containing any server messages, the email headers, and the email message. Useful for debugging.

You can optionally specify which parts of the message should be printed. Valid options are: **headers**, **subject**, **body**.

Example:

```
// You need to pass FALSE while sending in order for the email data
// to not be cleared - if that happens, printDebugger() would have
// nothing to output.
$email->send(false);

// Will only print the email headers, excluding the message subject and body
$email->printDebugger(['headers']);
```

**注解：** By default, all of the raw data will be printed.

### 7.1.4 加密服务

**重要：** 请勿使任何 *encryption* 库来存储密码！密码必须使用 散列，而你应该通过 PHP 的密码散列扩展 进行散列。

加密服务提供双向对称（密钥）数据加密。该服务将实例化或初始化 **加密程序**以适配你的参数，如下所述。

加密处理程序必须实现 CodeIgniter 的 `EncrypterInterface` 接口。使用 PHP 密码扩展或其它第三方库可能需要在服务器上安装其他软件，并且可能需要在 PHP 实例启用。

支持以下拓展：

- OpenSSL

这并不是一套完整的密码解决方案。如果您需要更多功能（例如公钥加密），建议你考虑直接使用 OpenSSL 或其他 密码学扩展 。还有一种更全面的软件包，例如 Halite（基于 libsodium 构建的 O-O 软件包）。

**注解：** 自从 PHP 7.2 起就已弃用了对 MCrypt 扩展的支持。



## 使用加密类库

就像 CodeIgniter 的其他服务，它可以通过 Config\Services 来调用：

```
$encrypter = \Config\Services::encrypter();
```

如果你已设置了启动密钥（请参阅配置加密类库 ），那么加密和解密数据很简单，将适当的字符串传递给 encrypt() 或 decrypt() 方法：

```
$plainText = 'This is a plain-text message!';
$ciphertext = $encrypter->encrypt($plainText);

// 输出: This is a plain-text message!
echo $encrypter->decrypt($ciphertext);
```

就是这样！加密库将为加密整个过程提供开箱即用的加密安全性。你无需担心。

## 配置加密类库

上面的示例将使用 app/Config/Encryption.php 中的配置设置。

它只有两个设置选项

| 选项 | 可能的值 |
| --- | --- |
| key | 加密 |
| 启动器 | |
| driver | 首选加密程序 (默认为 OpenSSL) |

你可以通过将自己的配置对象传递给 Services 调用来替换配置文件的设置。$config 的值必须是 *Config\Encryption* 类的实例或扩展 *CodeIgniter\Config\BaseConfig* 的实例。

```
$config          = new Config\Encryption();
$config->key     = 'aBigsecret_ofAtleast32Characters';
$config->driver = 'OpenSSL';

$encrypter = \Config\Services::encrypter($config);
```

### 默认行为

默认情况下，加密库使用 OpenSSL 加密程序。该处理程序使用 AES-256-CTR 算法、你配置的 **密钥**和 SHA512 HMAC 身份验证进行加密。

### 配置你的密钥

你的加密密钥的长度 **必须**在使用的加密算法允许的范围内。比如对于 AES-256 来说，则为 256 位或 32 个字节（字符）长度。

密钥应该尽可能随机，并且不能是常规文本字符串，也不能是哈希函数的输出等。要创建正确的密钥，可以使用加密库的 createKey() 方法。

```
// $key 将被分配一个 32 字节 (256 位) 随机密钥
$key = Encryption::createKey(32);
```

密钥可以存储在 *app/Config/Encryption.php* 中，或者您可以设计自己的存储机制，并在加解密时动态传递密钥。

要将密钥保存到 *app/Config/Encryption.php* ，请打开文件并进行以下设置：：

```
public $key = 'YOUR KEY';
```

### 对密钥或结果编码

你会注意到 createKey() 方法会输出二进制数据，这是很难解决（即复制粘贴可能会损坏），所以你可以使用 bin2hex() 、hex2bin() 或编码的 Base64 处理以更友好的密钥。例如：：

```
// 获取一个十六进制形式的密钥
$encoded = bin2hex(Encryption::createKey(32));

// 使用 hex2bin() 将相同的值放入配置中，
// 这样它仍会以二进制形式传递给库配置：
$key = hex2bin(<your hex-encoded key>);
```

你可能会发现相同的技术对于加密结果也是有效的：：

```
// Encrypt some text & make the results text
// 加密一些文本并生成密文
$encoded = base64_encode($encrypter->encrypt($plaintext));
```

### 加密处理程序说明

#### OpenSSL 说明

一直以来，OpenSSL 扩展一直是 PHP 的标配。

CodeIgniter 的 OpenSSL 处理程序使用 AES-256-CTR 算法。

你的配置提供的 密钥用于派生另外两个密钥，一个用于加密，另一个用于身份验证。这是通过一种叫做 基于 HMAC 的密钥派生函数 （HKDF）的技术来实现的。

#### 消息长度

加密后的字符串通常长于原始的纯文本字符串（取决于算法）。

这受密码算法本身影响，初始化因子（IV）以及 HMAC 身份验证消息也会加在密码文本之前。此外，加密的消息也会经过 Base64 编码，因此无论使用什么字符集，它都可以安全地存储和传输。

但是选择数据存储机制时，请记住，Cookie 只能保存 4K 信息。

### 直接使用加密服务

除了使用使用加密类库 中 Services 那样的方法外，你还可以直接创建“加密器”，或更改现有实例的设置。

```
// create an Encrypter instance
// 创建一个加密器实例
$encryption = new \Encryption\Encryption();

// reconfigure an instance with different settings
// 用不同的设置重新配置实例
$encrypter = $encryption->initialize($config);
```

请记住，$config 必须是 *ConfigEncryption* 类或扩展 *CodeIgniterConfigBaseConfig* 类的实例。

### 类参考

**CodeIgniter\Encryption\Encryption**

**static createKey**(*$length*)

**参数**

- **$length** (*int*) – 输出密钥的长度

**返回** 具有指定长度的随机密码密钥，创建失败则为 FALSE

**返回类型** string

通过从操作系统的源（即/dev/urandom）获取随机数据来创建加密密钥。

**initialize**(*$config*)

**参数**

- **$config** (*BaseConfig*) – Configuration parameters

**返回** CodeIgniter\Encryption\EncrypterInterface instance

**返回类型** CodeIgniter\Encryption\EncrypterInterface

**Throws** CodeIgniter\Encryption\EncryptionException

初始化（或配置）库以使用不同的设置。

例:

```
$encrypter = $encryption->initialize(['cipher' => '3des']);
```

请参阅配置加密类库 部分以获取详细信息。

**CodeIgniter\Encryption\EncrypterInterface**

**encrypt**(*$data*, *$params = null*)

**参数**

- **$data** (*string*) – 要加密的数据
- **$params** – 配置参数（或键）

**返回** 加密后的数据，加密失败时返回 FALSE

**返回类型** string

**Throws** CodeIgniter\Encryption\EncryptionException

加密输入数据并返回其密文。

将配置参数作为第二个参数传递时，如果 $params 是数组，则 密钥将用作这次加密的起始键；或者也可以把这次加密的密钥作为字符串传递。

例:

```
$ciphertext = $encrypter->encrypt('My secret message');
$ciphertext = $encrypter->encrypt('My secret message', [
↪'key' => 'New secret key']);
$ciphertext = $encrypter->encrypt('My secret message',
↪'New secret key');
```

**decrypt**(*$data*, *$params = null*)

**参数**

- **$data** (*string*) – 要解密的数据
- **$params** – 配置参数（或键）

**返回** 解密后的数据，解密失败时返回 FALSE

**返回类型** string

**Throws** CodeIgniter\Encryption\EncryptionException

加密输入数据并返回其密文。

将配置参数作为第二个参数传递时，如果 $params 是数组，则 密钥将用作这次解密的起始键；或者也可以把这次解密的密钥作为字符串传递。

例:

```
echo $encrypter->decrypt($ciphertext);
echo $encrypter->decrypt($ciphertext, ['key' => 'New␣
↪secret key']);
echo $encrypter->decrypt($ciphertext, 'New secret key');
```

### 7.1.5 使用文件类

CodeIgniter 提供了一个文件类, 它将提供 SplFileInfo class 方法和一些额外的便利方法. 这个类是*uploaded files* 的基类和*images*.

- 获取文件类实例
- 利用 *Spl*
- 新功能
  - 移动文件

### 获取文件类实例

通过传递构造函数中文件的路径来创建新的文件实例。默认情况下, 文件不需要存在。但是您可以传递一个附加参数 “true” , 以检查该文件是否存在, 并在不存在的情况下抛出 FileNotFoundException() 的异常提示.

```
$file = new \CodeIgniter\Files\File($path);
```

### 利用 Spl

一旦你有一个实例, 你就可以完成 SplFileInfo 类的全部功能, 包括:

```
echo $file->getBasename();  // 获取文件的基本名称

echo $file->getMTime();     // 获取上次修改的时间

echo $file->getRealpath();  // 获取真正的实际路径


echo $file->getPerms();     // 获取文件权限


if ($file->isWritable())    // 向 CSV 中写入几行数据.
{
    $csv = $file->openFile('w');

    foreach ($rows as $row)
    {
        $csv->fputcsv($row);
    }
}
```

### 新功能

除了 SplFileInfo 类中的所有方法之外, 还有一些新的方法.

#### getRandomName()

您可以生成一个加密安全的随机文件名, 其中包含当前时间戳, getRandomName() 方法在移动文件时重命名文件很有用:

```
$newName = $file->getRandomName();  // 例如: 1465965676_385e33f741.jpg
```

#### getSize()

返回上传文件的大小 (以字节为单位). 可以将 ‘kb’ 或 ‘mb’ 作为第一个参数传入方法, 将分别返回千字节和兆字节的结果:

```
$bytes     = $file->getSize();       // 256901

$kilobytes = $file->getSize('kb');   // 250.880

$megabytes = $file->getSize('mb');   // 0.245
```

**getMimeType()**

尽可能在确定文件安全的前提下，使用该方法获取文件的类型：

```
$type = $file->getMimeType();

echo $type; // image/png
```

**guessExtension()**

使用 `getMimeType()` 方法确定文件扩展名时. 如果文件类型未知，将返回 null . `guessExtension()` 比使用 `getMimeType()` 来获取扩展名功能强一点. 可以配置 **application/Config/Mimes.php** 中的配置文件来获取文件扩展名：

```
$ext = $file->guessExtension();      // 例如：返回图片类型 'jpg' (没有句点
↪'.')
```

#### 移动文件

每个文件可以使用 `move()` 方法移动到新的位置. 指定文件的目录作作为该方法的第一个参数：

```
$file->move(WRITEPATH.'uploads');
```

默认情况下，使用原始文件名. 您可以通过第二个参数重命名你要移动的文件：

```
$newName = $file->getRandomName();

$file->move(WRITEPATH.'uploads', $newName);
```

### 7.1.6 Honeypot Class

The Honeypot Class makes it possible to determine when a Bot makes a request to a CodeIgniter4 application, if it's enabled in `Application\Config\Filters.php` file. This is done by attaching form fields to any form, and this form field is hidden from a human but accessible to a Bot. When data is entered into the field, it' s assumed the request is coming from a Bot, and you can throw a `HoneypotException`.

### Enabling Honeypot

To enable a Honeypot, changes have to be made to the `app/Config/Filters.php`. Just uncomment honeypot from the `$globals` array, like⋯:

```
public $globals = [
        'before' => [
            'honeypot'
            // 'csrf',
        ],
        'after'  => [
            'toolbar',
            'honeypot'
        ]
    ];
```

A sample Honeypot filter is bundled, as `system/Filters/Honeypot.php`. If it is not suitable, make your own at `app/Filters/Honeypot.php`, and modify the `$aliases` in the configuration appropriately.

### Customizing Honeypot

Honeypot can be customized. The fields below can be set either in `app/Config/Honeypot.php` or in `.env`.

- `hidden` - true|false to control visibility of the honeypot field; default is `true`
- `label` - HTML label for the honeypot field, default is 'Fill This Field'
- `name` - name of the HTML form field used for the template; default is 'honeypot'
- `template` - form field template used for the honeypot; default is '<label>{label}</label><input type=" text" name=" {name}" value=" " />'

## 7.1.7 图像处理类

CodeIgniter 的图像处理类允许你执行以下操作:

- 图像大小调整
- 创建缩略图
- 图像裁剪

- 图像旋转
- 图像水印

图像处理类支持使用以下图像库:GD/GD2 和 ImageMagick



## 初始化类

与 CodeIgniter 中的大多数其他类一样，你可以通过控制器中调用 Services 类的初始化图像处理类:

```
$image = Config\Services::image();
```

你可以将要使用的图像库的别名传递给服务功能:

```
$image = Config\Services::image('imagick');
```

可用的图像库处理程序如下:

- gd 对应调用的是 GD/GD2 图像库。
- imagick 对应调用的是 ImageMagick 图像库。

如果你要使用 ImageMagick 图像库，则必须要在 **application/Config/Images.php** 中设置服务器上该库的所在路径。

---

**注解:** ImageMagick 处理程序不需要在服务器上加载 imagick 扩展。只要你的脚本可以访问该库并且可以使用 `exec()` 运行在服务器上，它就可以工作。

---

## 处理图像

无论你执行何种图像的处理方法函数（调整大小、裁剪、旋转、使用水印），一般调用过程都是相同的。你将根据要执行的操作设置一些首选项，然后调用其中一个你需要的使用的可用处理函数:

```
$image = Config\Services::image()
        ->withFile('/path/to/image/mypic.jpg')
        ->fit(100, 100, 'center')
        ->save('/path/to/image/mypic_thumb.jpg');
```

上面的代码告我们它会查找来自 image 文件夹中的名为 *mypic.jpg* 的图像，然后使用 GD2 image_library 图像库来创建一个 100 x 100 像素的新图像，并将其保存到新文件 (the thumb)。由于它使用 fit() 方法，它将尝试根据所需的宽高比找到要裁剪的图像的最佳部分，然后裁剪并调整结果大小。

在保存新图像之前，可以根据需求来通过许多可用方法来处理图像。原始图像将保持原样，而新图像会通过每个方法传参，将处理结果应用于直接的结果之上:

```
$image = Config\Services::image()
        ->withFile('/path/to/image/mypic.jpg')
        ->reorient()
        ->rotate(90)
        ->crop(100, 100, 0, 0)
        ->save('/path/to/image/mypic_thumb.jpg');
```

此示例将采用相同的图像并首先修复任何移动电话的定向问题，图像将旋转 90 度，然后从左上角开始将结果裁剪为 100x100 像素图像。结果将保存成缩略图。

**注解:** 为了让图像处理类可以进行任何处理，包含图像文件的文件夹必须具有写入权限。

**注解:** 对于某些操作，图像处理时可能需要相当大量的服务器内存。如果在处理图像时遇到内存不足错误，可能需要限制其图像的最大大小，和/或调整 PHP 内存限制。

### 处理方法

有六种可用的处理方法可以调用:

- $image->crop()
- $image->fit()
- $image->flatten()
- $image->flip()
- $image->resize()
- $image->rotate()
- $image->text()

这些方法将会返回类实例，如上所示，它们可以链接在一起。如果失败，它们将抛出包含错误的消息到 `CodeIgniter\Images\ImageException` 。一个好的做法是捕获异常消息，在失败时显示错误，如下所示:

```
try {
$image = Config\Services::image()
```

(下页继续)

(续上页)

```
    ->withFile('/path/to/image/mypic.jpg')
    ->fit(100, 100, 'center')
    ->save('/path/to/image/mypic_thumb.jpg');
}
catch (CodeIgniter\Images\ImageException $e)
{
        echo $e->getMessage();
}
```

**注解：** 你可以选择通过在函数中提交开始/结束标记来指定要应用于错误的 HTML 格式，如下所示:

```
$this->image_lib->display_errors('<p>', '</p>');
```

### 图像裁剪

图像可以被裁剪，只保留原始图像的一部分。通常用于创建特定大小/纵横比匹配的缩略图图像。这是用 crop() 方法处理的:

```
crop(int $width = null, int $height = null, int $x = null, int $y =␣
↪null, bool $maintainRatio = false, string $masterDim = 'auto')
```

- **$width** 是结果图像的所需宽度，以像素为单位。
- **$height** 是结果图像的所需高度，以像素为单位。
- **$x** 是从图像左侧开始裁剪的像素数。
- **$y** 是从图像顶部开始裁剪的像素数。
- **$maintainRatio** 如果为 true，将根据需要调整最终尺寸以保持图像的原始高宽比。
- **$masterDim** 可使其保持不变的维度，当 $maintainRatio 为 true 时。值可以是:’width’,’ height’ 或’ auto’。

要从图像中心取出 50x50 像素的正方形，你需要首先计算适当的 x 和 y 偏移值:

```
$info = Services::image('imagick')
            ->withFile('/path/to/image/mypic.jpg')
            ->getFile()
            ->getProperties(true);

$xOffset = ($info['width'] / 2) - 25;
$yOffset = ($info['height'] / 2) - 25;
```

(下页继续)

(续上页)

```
Services::image('imagick')
            ->withFile('/path/to/image/mypic.jpg')
            ->crop(50, 50, $xOffset, $yOffset)
            ->save('path/to/new/image.jpg');
```

### 拟合图像

使用 `fit()` 方法旨在通过执行以下步骤帮助简化以“智能”方式裁剪图像的一部分:

- 确定要裁剪的原始图像的正确部分，以保持所需的宽高比。
- 裁剪原始图像。
- 调整大小到最终尺寸。

```
fit(int $width, int $height = null, string $position = 'center')
```

- **$width** 是图像的最终宽度。
- **$height** 是图像所需的最终高度。
- **$position** 确定要裁剪的图像部分。允许的位置: ‘top-left’ , ‘top’ , ‘top-right’ , ‘left’ , ‘center’ , ‘right’ , ‘bottom-left’ , ‘bottom’ , ‘bottom-right’。

这里提供一种更简单的裁剪方式，可以始终保持纵横比:

```
Services::image('imagick')
        ->withFile('/path/to/image/mypic.jpg')
        ->fit(100, 150, 'left')
        ->save('path/to/new/image.jpg');
```

### 展平图像

使用 `flatten()` 方法旨在在透明图像（PNG）后面添加背景颜色并将 RGBA 像素转换为 RGB 像素

- 从透明图像转换为 jpgs 格式时指定背景颜色。

```
flatten(int $red = 255, int $green = 255, int $blue = 255)
```

- **$red** 是背景的红色值。
- **$green** 是背景的绿色值。
- **$blue** 是背景的蓝色值。

```
Services::image('imagick')
        ->withFile('/path/to/image/mypic.png')
```

(下页继续)

(续上页)

```
        ->flatten()
        ->save('path/to/new/image.jpg');

Services::image('imagick')
        ->withFile('/path/to/image/mypic.png')
        ->flatten(25,25,112)
        ->save('path/to/new/image.jpg');
```

### 翻转图像

图像可以沿水平轴或垂直轴翻转:

```
flip(string $dir)
```

- **$dir** 指定要翻转的轴。可以是“垂直”或“水平”。

```
Services::image('imagick')
        ->withFile('/path/to/image/mypic.jpg')
        ->flip('horizontal')
        ->save('path/to/new/image.jpg');
```

### 调整图像大小

可以使用 resize() 方法调整图像大小以适合你需要的任何维度:

```
resize(int $width, int $height, bool $maintainRatio = false, string
↪$masterDim = 'auto')
```

- **$width** 是新图像的所需宽度（以像素为单位）
- **$height** 是新图像的所需高度（以像素为单位）
- **$maintainRatio** 确定图像是否被拉伸以适应新尺寸，或者是否保持原始宽高比。
- **$masterDim** 指定在保持比率时哪个轴应该具有其维度。’宽度’,’高度’。

调整图像大小时，你可以选择是保持原始图像的比例，还是拉伸/压缩新图像以适合所需的尺寸。如果 $maintainRatio 为 true，则 $masterDim 指定的尺寸将保持不变，而另一个尺寸将更改为与原始图像的纵横比相匹配。

```
Services::image('imagick')
        ->withFile('/path/to/image/mypic.jpg')
        ->resize(200, 100, true, 'height')
        ->save('path/to/new/image.jpg');
```

### 旋转图像

使用 rotate() 方法允许你以 90 度的增量旋转图像:

```
rotate(float $angle)
```

- **$angle** 是要旋转的度数。’ 90’ ,’ 180’ ,’ 270’ 之一。

---

**注解:** 虽然 $angle 参数接受 float，但它会在进程中将其转换为整数。如果该值不是上面列出的三个值，他会抛出一个自 CodeIgniterImagesImageException 的图像异常错误.

---

### 添加文本水印

你可以使用 text() 方法非常简单地将文本水印叠加到图像上。这对于放置版权声明，摄影师名称或简单地将图像标记为预览非常有用，这会使它们最终不会用于其他人的产品上。

```
text(string $text, array $options = [])
```

第一个参数是你要显示的文本字符串。第二个参数是一个选项数组，允许你指定文本的显示方式:

```
Services::image('imagick')
        ->withFile('/path/to/image/mypic.jpg')
        ->text('Copyright 2017 My Photo Co', [
            'color'      => '#fff',
            'opacity'    => 0.5,
            'withShadow' => true,
            'hAlign'     => 'center',
            'vAlign'     => 'bottom',
            'fontSize'   => 20
        ])
        ->save('path/to/new/image.jpg');
```

可识别的选项如下:

- color 文本颜色 (十六进制数字), 即# ff0000
- opacity 设置一个介于 0 到 1 之间的数字，表示文本的不透明度。
- withShadow 以布尔值是否来显示阴影。
- shadowColor 设定阴影的颜色（十六进制数）。
- shadowOffset 偏移阴影的像素数。适用于垂直和水平值。
- hAlign 水平对齐：左，中，右
- vAlign 垂直对齐：顶部，中间，底部

- hOffset 指定 x 轴上的附加偏移，以像素为单位
- vOffset 指定 y 轴上的附加偏移，以像素为单位
- fontPath 要使用的 TTF 字体的完整服务器路径。如果没有给出系统字体，将使用系统字体。
- fontSize 要使用的字体大小。将 GD 处理程序与系统字体一起使用时，有效值介于 1-5 之间。

**注解：** ImageMagick 驱动程序无法识别 fontPath 的完整服务器路径。相反，需要你提供希望使用的已安装系统字体之一的名称，即如 Calibri。

### 7.1.8 分页类

CodeIgniter 提供了一个非常简单但灵活的分页库，该库主题简单，可以在 Model 中使用，并能够在单个页面上支持多个分页器。

#### 加载库

与 CodeIgniter 中的所有服务一样，它可以通过 Config\Services 进行加载，尽管通常它并不需要手动加载：

```
$pager = \Config\Services::pager();
```

#### 分页数据库结果

通常，可以使用分页器库对从数据库中检索到的结果进行分页。使用*Model* 类时，可以使用其内置的 paginate() 方法来自动检索当前批次的结果，并设置 Pager 库，以便可以在控制器中使用它。它甚至可以通过 page=X 变量从当前 URL 读取当前应显示的页面。

在你的应用程序中提供用户的分页列表时，控制器的方法应类似于：

```
<?php namespace App\Controllers;

use CodeIgniter\Controller;

class UserController extends Controller
{
    public function index()
    {
        $model = new \App\Models\UserModel();

        $data = [
```

(下页继续)

(续上页)

```
            'users' => $model->paginate(10),
            'pager' => $model->pager
        ];

        echo view('users/index', $data);
    }
}
```

在上面的示例中，我们首先创建了 UserModel 的实例。然后，我们对它填充数据来发送到视图。第一个元素是来自数据库 **users** 的结果，这将针对正确的页面进行检索，每页会返回 10 个用户。发送到视图的第二个必须的项是 Pager 实例本身。为了方便起见，Model 将会保留所使用的实例，并将其存储在 public 类变量 **$pager** 中。因此，我们将其获取并将其分配给视图中的 $pager 变量。

然后，在视图内，我们需要告诉它应该在哪里显示结果的链接：

```
<?= $pager->links() ?>
```

就是这样。Pager 类将为当前页面两侧超过两个页面的任何页面呈现“首页”和“末页”的链接，以及“下一页”和“上一页”的链接。

如果你更喜欢简单的输出，则可以使用 `simpleLinks()` 方法，它会输出“较旧”和“较新”链接而不是有着详细信息的分页链接：

```
<?= $pager->simpleLinks() ?>
```

在后台中，库加载了一个视图文件，文件确定链接的格式，从而可以轻松地根据需要进行修改。有关如何完全自定义输出的详细信息，请参见下文。

### 分页多个结果

如果需要提供来自两个不同的结果集的链接，则可以将组名传递给大多数分页方法，以使数据分开：

```
// 在控制器文件中：
public function index()
{
    $userModel = new \App\Models\UserModel();
    $pageModel = new \App\Models\PageModel();

    $data = [
        'users' => $userModel->paginate(10, 'group1'),
        'pages' => $pageModel->paginate(15, 'group2'),
        'pager' => $userModel->pager
    ];

    echo view('users/index', $data);
```

(下页继续)

(续上页)

```
}

// 在视图文件中：
<?= $pager->links('group1') ?>
<?= $pager->simpleLinks('group2') ?>
```

### 手动分页

你可能会发现有时候只需要根据已知数据来创建分页。这时你可以使用 makeLinks() 方法来手动创建链接，这个方法分别将当前页面，每页的结果数和项目总数作为第一个，第二个和第三个参数：

```
<?= $pager->makeLinks($page, $perPage, $total) ?>
```

默认情况下，这将以正常方式将链接显示为一组链接，你还可以通过将模板名称作为第四个参数传入来更改使用的显示模板。在以下各节中可以找到更多详细信息。

```
<?= $pager->makeLinks($page, $perPage, $total, 'template_name') ?>
```

也可以使用 URI 字段（segment）而不是用查询参数来表示页码，只需指定字段号即可用作的第五个参数 makeLinks() 。然后，由分页器生成的 URI 看起来会像 *https://domain.tld/model/*『页码』而不是 *https://domain.tld/model?page=*『页码』。

```
<?= $pager->makeLinks($page, $perPage, $total, 'template_name',
↪$segment) ?>
```

请注意：$segment 的值不能大于 URI 字段的数量加 1。

如果你需要在一页上显示很多分页器，那么定义组的其他参数可能会有所帮助：

```
$pager = service('pager');
$pager->setPath('path/for/my-group', 'my-group'); // 另外，你可以为每个组
定义路径
$pager->makeLinks($page, $perPage, $total, 'template_name', $segment,
↪'my-group');
```

### 仅使用预期查询进行分页

默认情况下，所有 GET 查询都显示在分页链接中。

例如，当访问 URL *http://domain.tld?search=foo&order=asc&hello=i+am+here&page=2* 时，可以生成页面 3 链接以及其他链接，如下所示：

```
echo $pager->links();
// 页面 3 链接：  http://domain.tld?search=foo&order=asc&hello=i+am+here&
↪page=3
```

only() 方法还允许你将其限制为仅已预期的查询：

```
echo $pager->only(['search', 'order'])->links();
// 页面 3 链接：  http://domain.tld?search=foo&order=asc&page=3
```

*page* 查询默认情况下启用。并 only() 在所有分页链接中起作用。

### 自定义链接

### 查看配置

当链接呈现到页面时，它们使用视图文件来渲染 HTML。你可以通过编辑 **app/Config/Pager.php** 来轻松地更改使用的视图：

```
public $templates = [
    'default_full'   => 'CodeIgniter\Pager\Views\default_full',
    'default_simple' => 'CodeIgniter\Pager\Views\default_simple'
];
```

设置存储应使用的视图的别名和命名空间的视图路径 。*default_full* 和 *default_simple* 视图会分别被用于 links() 和 simpleLinks() 方法。要更改在整个应用程序范围内显示的方式，你可以在处分配一个新视图。

例如，假设你创建一个与 Foundation CSS 框架一起使用的新视图文件，然后将文件放在 **app/Views/Pagers/foundation_full.php** 中。由于 **application** 目录的命名空间为 App ，并且其下的所有目录都直接映射到命名空间的各个部分，因此你可以通过其命名空间找到视图文件：

```
'default_full'   => 'App\Views\Pagers\foundation_full',
```

但是，由于它位于标准的 **app/Views** 目录下，因此不需要命名空间，因为 "view()" 方法可以按文件名定位它。在这种情况下，你只需提供子目录和文件名：

```
'default_full'   => 'Pagers/foundation_full',
```

创建视图并将其配置好后，将会自动使用它。你不必替换现有模板。你也可以在配置文件中根据需要创建的任意数量的其他模板。常见的情况是你的应用程序的前端和后端需要不同的样式。

```
public $templates = [
    'default_full'   => 'CodeIgniter\Pager\Views\default_full',
    'default_simple' => 'CodeIgniter\Pager\Views\default_simple',
    'front_full'     => 'App\Views\Pagers\foundation_full',
];
```

配置完成后，你可以指定它作为 links() 、simpleLinks() 以及 makeLinks() 方法的最后的一个参数：

```
<?= $pager->links('group1', 'front_full') ?>
<?= $pager->simpleLinks('group2', 'front_full') ?>
<?= $pager->makeLinks($page, $perPage, $total, 'front_full') ?>
```

### 创建视图

创建新视图时，只需要创建生成分页链接本身所需的代码。你不应该创建不必要的包装 div，因为它可能会在多个地方使用，并且这会限制它们的用途。这里通过向你展示现有的 default_full 模板，来演示创建新视图：

```
<?php $pager->setSurroundCount(2) ?>

<nav aria-label="Page navigation">
    <ul class="pagination">
    <?php if ($pager->hasPrevious()) : ?>
        <li>
            <a href="<?= $pager->getFirst() ?>" aria-label="First">
                <span aria-hidden="true">First</span>
            </a>
        </li>
        <li>
            <a href="<?= $pager->getPrevious() ?>" aria-label="Previous">
                <span aria-hidden="true">&laquo;</span>
            </a>
        </li>
    <?php endif ?>

    <?php foreach ($pager->links() as $link) : ?>
        <li <?= $link['active'] ? 'class="active"' : '' ?>>
            <a href="<?= $link['uri'] ?>">
                <?= $link['title'] ?>
            </a>
        </li>
    <?php endforeach ?>

    <?php if ($pager->hasNext()) : ?>
        <li>
            <a href="<?= $pager->getNext() ?>" aria-label="Previous">
                <span aria-hidden="true">&raquo;</span>
            </a>
        </li>
        <li>
            <a href="<?= $pager->getLast() ?>" aria-label="Last">
                <span aria-hidden="true">Last</span>
            </a>
```

(下页继续)

（续上页）

```
        </li>
    <?php endif ?>
    </ul>
</nav>
```

**setSurroundCount()**

在第一行中，setSurroundCount() 方法指定了我们要显示到当前页面链接两侧的两个链接。它接受的唯一参数是要显示的链接数。

**hasPrevious() & hasNext()**

如果根据传递给 setSurroundCount 的值，如果当前页面的任何一侧上可以显示更多链接，则这些方法将返回布尔值 true。例如，假设我们有 20 页数据，当前页面是第 3 页，如果周围的计数是 2，则以下链接将显示在列表中：1、2、3、4 和 5。由于要显示的第一个链接是第 1 页，但是页面 0 并不存在，因此 hasPrevious() 会返回 **false** 。但是，hasNext() 将返回 **true** ，因为在第 5 页之后还有 15 个额外的结果页。

**getPrevious() & getNext()**

这两个方法返回编号链接两侧上一页或下一页结果的 URL。有关完整说明，请参见上一段。

**getFirst() & getLast()**

与 getPrevious() 和 getNext() 类似，这两个方法返回指向结果集中第一页和最后一页的链接。

**links()**

返回所有有关编号链接的数据数组。每个链接的数组都包含链接的 uri，标题（只是数字）和一个布尔值，布尔值表示链接为当前链接还是活动链接：

```
$link = [
        'active' => false,
        'uri'    => 'http://example.com/foo?page=2',
        'title'  => 1
];
```

### 7.1.9 安全类

安全类包含了一些方法，用于帮助保护你的网站，以免受到跨站请求伪造（CSRF）的攻击。

### 加载类

如果你加载这个类，只是想进行 CSRF 的防护，那就没必要加载它，因为它是作为一个过滤器运行的，而且没有手动调用的接口。

如果你想在某种情况下直接访问这个类，你可以通过服务文件来加载它:

```
$security = \Config\Services::security();
```

### 跨站请求伪造（CSRF）

打开你的 **application/Config/Filters.php** 文件并且全局开启 *csrf* 过滤器，即可开启 CSRF 防护:

```
public $globals = [
        'before' => [
                'csrf'
        ]
];
```

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* 过滤器的‘except’配置参数中:

```
public $globals = [
        'before' => [
                'csrf' => ['except' => ['api/record/save']]
        ]
];
```

同样支持正则表达式（不区分大小写）:

```
public $globals = [
            'before' => [
                    'csrf' => ['except' => ['api/record/[0-9]+']]
            ]
    ];
```

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，你可以手动调用 get_csrf_token_name() 和 get_csrf_hash() 这两个函数。

```
<input type="hidden" name="<?= csrf_token() ?>" value="<?= csrf_hash() ?>
↪" />
```

另外，你可以使用 csrf_field() 方法来帮你生成这个隐藏的 input 字段:

```
// Generates: <input type="hidden" name="{csrf_token}" value="{csrf_
↪hash}" />
<?= csrf_field() ?>
```

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。

```
public $CSRFRegenerate  = true;
```

#### 其它的辅助方法

你将永远不需要直接使用安全类中的大多数方法。下面的一些方法，你可能会觉得有用，这些方法和 CSRF 防护无关。

**sanitizeFilename()**

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，这在文件名作为用户输入的参数时格外有用。第一个参数是需要净化的路径名。

如果用户输入包含相对路径是可以接受的，例如：file/in/some/approved/folder.txt，那么你可以设置第二个可选参数，$relative_path 为 true 。

```
$path = $security->sanitizeFilename($request->getVar('filepath'));
```

### 7.1.10 Session 类

Session 类允许你维护用户的“状态”并跟踪他们在浏览你的网站时的活动。

CodeIgniter 有一些用于会话（session）储存的驱动程序，你可以在目录的最后部分中看到它们：

## 使用 Session 类

### 初始化会话

会话通常会在每次加载页面时在全局范围内运行，因此应该恰当地初始化 Session 类。

访问并初始化会话：

```
$session = \Config\Services::session($config);
```

`$config` 参数是可选的，它是你的应用程序配置。如果未提供，服务将会使用你的默认配置。

初始化成功后，可以用以下方式使用 Session 库对象：

```
$session
```

或者，你可以使用使用默认配置的 helper 方法，这个版本阅读起来会更友好一些，但是不能配置任何配置选项

```
$session = session();
```

### Session 是怎样工作的？

加载页面后，Session 类将检查用户的浏览器是否发送了有效的会话 cookie。如果会话 Cookie **不存在**（或者如果它不匹配一个存储在服务器上的会话 ID 或已过期），新会话将被创建和保存。

如果确实存在有效的会话，则其信息将被更新。对于每次更新，如果配置了会话 ID ，则可以对其进行重新生成。

对你来说很重要的一点是，一旦初始化，Session 类就会自动运行。你无需执行任何操作即可导致上述现象发生。如下所示，你可以使用会话数据，但是读取，写入和更新会话的过程是自动的。

**注解：** 在 CLI 下，Session 库将自动停止运行，因为它只是一个完全基于 HTTP 协议的概念。

### 关于异步的说明

除非你要开发使用 AJAX 的网站，否则可以跳过本节。但是，如果你遇到了性能问题，那么本说明正是你所需要的。

早期版本的 CodeIgniter 中的会话未实现锁定，这意味着使用同一会话的两个 HTTP 请求可以完全同时运行。使用更合适的技术术语就是，请求是非阻塞的。

但是，会话上下文中的非阻塞请求也意味着不安全，因为在一个请求中对会话数据的修改（或会话 ID 再生）可能会干扰第二个并发请求的执行。这个细节是许多问题的根源，也是 CodeIgniter 4 拥有完全重写的 Session 库的主要原因。

我们为什么要告诉你这个？因为在尝试找出性能问题的原因之后，你可能会得出结论，锁定是问题所在，因此研究了如何删除锁定……

不要那样做！删除锁定是 **错误**的，它将给你带来更多问题！

锁定不是问题，而是解决方案。你的问题是，你已经打开了会话，但已经处理了该会话，因此不再需要它。因此，你需要的是在不再需要当前请求后关闭会话。

```
$session->destroy();
```

### 什么是会话数据？

会话数据是与特定会话 ID（cookie）关联的数组。

如果你以前在 PHP 中使用过会话，则应该熟悉 PHP 的 $_SESSION 全局变量 （如果不熟悉，请阅读该链接上的内容）。

CodeIgniter 使用与 PHP 提供的会话处理程序机制相同的方式来访问其会话数据。使用会话数据就像操作（读取，设置和删除）$_SESSION 数组一样简单。

此外，CodeIgniter 还提供 2 种特殊类型的会话数据，下面将进一步说明：闪存数据（flashdata）和临时数据（tempdata）。

### 检索会话数据

会话数组中的任何信息都可以通过 $_SESSION 全局变量获得：

```
$_SESSION['item']
```

或通过常规访问器方法：

```
$session->get('item');
```

或通过魔术方法，例如 getter ：

```
$session->item
```

甚至可以通过会话辅助函数：

```
session('item');
```

item 就是你所要获取的项目所对应的数组的键。例如，要将先前存储的“名称”项分配给 $name 变量，你可以这样做：

```
$name = $_SESSION['name'];

// 或者：
$name = $session->name

// 或者：
$name = $session->get('name');
```

**注解：** 对于 get() 方法，如果你要访问的项目不存在，返回 NULL。

如果要检索所有现有的用户数据，则可以简单地省略 item 键（获取器仅适用于单个属性值）：

```
$_SESSION

// 或者：
$session->get();
```

### 添加会话数据

假设某个特定用户登录到你的网站。身份验证后，你可以将其用户名和电子邮件地址添加到会话中，从而使你可以全局使用该数据，而不必在需要时运行数据库查询。

你可以把 $_SESSION 看作像其他变量一样，将数据简单地分配给数组。或作为 $session 的属性。

以前的 userdata 方法已被废弃，但是你可以将包含新会话数据的数组传递给该 set() 方法：

```
$session->set($array);
```

此处 $array 是一个包含新数据的关联数组，这是一个例子：

```
$newdata = [
    'username'  => 'johndoe',
    'email'     => 'johndoe@some-site.com',
    'logged_in' => TRUE
];

$session->set($newdata);
```

如果要一次为一个会话数据只添加一个值，则 set() 还支持以下语法：

```
$session->set('some_name', 'some_value');
```

如果要验证会话值是否存在，只需使用 isset() 以下命令进行检查：

```
// 如果'some_name' 项目不存在或为 NULL，则返回 FALSE，反之则返回 TRUE
isset($_SESSION['some_name'])
```

或者你可以调用 has()：

```
$session->has('some_name');
```

### 向会话数据推送新值

push 方法用于将新值推送到作为数组的会话值上。例如，如果“兴趣爱好”键包含一个兴趣爱好数组，则可以将新值添加到数组中，如下所示：

```
$session->push('hobbies', ['sport'=>'tennis']);
```

### 删除会话数据

与其他任何变量一样，$_SESSION 使用 unset() 通过以下方式取消设置的值：

```
unset($_SESSION['some_name']);

// 或者同时取消设置多个值

unset(
    $_SESSION['some_name'],
    $_SESSION['another_name']
);
```

同样，就像 set() 可以用来向会话添加信息一样，remove() 也可以通过传递会话数据的键来删除信息。例如，如果要从会话数据数组中删除“some_name”：

```
$session->remove('some_name');
```

此方法还接受要取消设置的项目键数组：

```
$array_items = ['username', 'email'];
$session->remove($array_items);
```

### 闪存数据（Flashdata）

CodeIgniter 支持“flashdata”，这是仅对下一个请求可用的会话数据，然后将其自动清除。

这可能非常有用，特别是对于一次性的信息，错误或状态消息（例如：“记录 2 已删除”）。

应当注意，flashdata 变量是常规会话变量，在 CodeIgniter 会话处理程序内部进行管理。

要将现有条目标记为“flashdata”：

```
$session->markAsFlashdata('item');
```

如果要将多个项目标记为 flashdata，只需将键作为数组传递：

```
$session->markAsFlashdata(['item', 'item2']);
```

要添加闪存数据：

```
$_SESSION['item'] = 'value';
$session->markAsFlashdata('item');
```

或者使用以下 `setFlashdata()` 方法：

```
$session->setFlashdata('item', 'value');
```

你还可以通过与 `set()` 相同的方式，将一个数组传递给 `setFlashdata()` 。

读取 flashdata 变量与通过 `$_SESSION` 以下方式读取常规会话数据相同：

```
$_SESSION['item']
```

---

**重要：** `get()` 当通过键检索单个项时，该方法将返回 flashdata 项。但是，从会话中获取所有用户数据时，它不会返回 flashdata。

---

但是，如果你想确定自己正在读取“flashdata”（而不是其他种类的数据），则也可以使用以下 `getFlashdata()` 方法：

```
$session->getFlashdata('item');
```

或者，要获取包含所有 flashdata 的数组，只需省略 key 参数：

```
$session->getFlashdata();
```

**注解：** getFlashdata() 如果找不到该项目，则该方法返回 NULL。

如果发现需要通过其他请求保留 flashdata 变量，则可以使用 `keepFlashdata()` 方法来实现。你可以传递单个项或一组 flashdata 项来保留。

```
$session->keepFlashdata('item');
$session->keepFlashdata(['item1', 'item2', 'item3']);
```

### 临时数据（Tempdata）

CodeIgniter 还支持“tempdata”这种具有特定到期时间的会话数据。该值过期或会话过期或被删除后，该值将自动删除。

与 flashdata 相似，tempdata 变量由 CodeIgniter 会话处理程序在内部进行管理。

要将现有项目标记为“tempdata”，只需将其密钥和有效时间（以秒为单位）传递给该 `mark_as_temp()` 方法：

```
// 'item' will be erased after 300 seconds
$session->markAsTempdata('item', 300);
```

你可以通过两种方式将多个项目标记为临时数据，具体取决于你是否希望它们都具有相同的到期时间：

```
// “item”和“item2”都将在 300 秒后过期
$session->markAsTempdata(['item', 'item2'], 300);

// 'item' 将在 300 秒后删除，而'item2' 将在 240 秒后删除
$session->markAsTempdata([
    'item'  => 300,
    'item2' => 240
]);
```

添加临时数据：

```
$_SESSION['item'] = 'value';
$session->markAsTempdata('item', 300); // Expire in 5 minutes
```

或者使用以下 `setTempdata()` 方法：

```
$session->setTempdata('item', 'value', 300);
```

你还可以将数组传递给 `set_tempdata()` ：

```
$tempdata = ['newuser' => TRUE, 'message' => 'Thanks for joining!'];
$session->setTempdata($tempdata, NULL, $expire);
```

**注解：** 如果省略了到期时间或将其设置为 0，则将使用默认的生存时间值为 300 秒（或 5 分钟）。

要读取 tempdata 变量，同样可以通过 $_SESSION 超全局数组访问它：

```
$_SESSION['item']
```

**重要：** get() 当通过键检索单个项目时，该方法将返回 tempdata 项目。但是，从会话中获取所有用户数据时，它不会返回 tempdata。

或者，如果你想确保自己正在读取 “tempdata”（而不是其他种类的数据），则也可以使用以下 getTempdata() 方法：

```
$session->getTempdata('item');
```

当然，如果要检索所有现有的临时数据：

```
$session->getTempdata();
```

**注解：** getTempdata() 如果找不到该项目，则该方法返回 NULL。

如果你需要在一个临时数据过期之前删除它，你可以在 $_SESSION 数组里面做到

```
unset($_SESSION['item']);
```

但是，这不会删除使该特定项成为 tempdata 的标记（它将在下一个 HTTP 请求中失效），因此，如果你打算在同一请求中重用同一键，则需要使用 removeTempdata()：

```
$session->removeTempdata('item');
```

### 销毁会话

要清除当前会话（例如，在注销过程中），你可以简单地使用 PHP 的 session_destroy() 函数或库的 destroy() 方法。两者将以完全相同的方式工作：

```
session_destroy();

// 或者

$session->destroy();
```

**注解：** 这必须是你在同一请求期间执行的与会话有关的最后一个操作。销毁会话后，所有会话数据（包括 flashdata 和 tempdata）将被永久销毁，并且在同一请求期间功能将无法使用。

你还可以 `stop()` 通过删除旧的 session_id，销毁所有数据并销毁包含会话 ID 的 cookie，使用该方法完全终止会话：

```
$session->stop();
```

### 访问会话元数据

在以前的 CodeIgniter 版本中，默认情况下，会话数据数组包括 4 个项目："session_id"，"ip_address"，"user_agent"，"last_activity"。

这是由于会话如何工作的细节所致，但现在在我们的新实现中不再需要。但是，你的应用程序可能会依赖这些值，因此下面是访问它们的替代方法：

- session_id: `session_id()`
- ip_address: `$_SERVER['REMOTE_ADDR']`
- user_agent: `$_SERVER['HTTP_USER_AGENT']` (unused by sessions)
- last_activity: Depends on the storage, no straightforward way. Sorry!

### 会话首选项

通常，CodeIgniter 可以使所有工作立即可用。但是，会话是任何应用程序中非常敏感的组件，因此必须进行一些仔细的配置。请花点时间考虑所有选项及其效果。

你将在 **app/Config/App.php** 文件中找到以下与会话相关的首选项：

| 配置项 | 默认 | 选项 | 描述 |
|---|---|---|---|
| **sessionDriver** | CodeIgniter\Session\Handlers\FileHandler | CodeIgniter\Session\Handlers\FileHandler CodeIgniter\Session\Handlers\DatabaseHandler CodeIgniter\Session\Handlers\MemcachedHandler CodeIgniter\Session\Handlers\RedisHandler CodeIgniter\Session\Handlers\ArrayHandler | 使用的会话驱动程序 |
| **sessionCookieName** | ci_session | [A-Za-z_-] characters only | 会话 cookie 的名字 |
| **sessionExpiration** | 7200 (2 hours) | Time in seconds (integer) | 您希望会话持续的秒数。如果您希望会话不过期（直到浏览器关闭），请将值设置为零：0 |
| **sessionSavePath** | NULL | None | 指定存储位置，取决于所使用的驱动程序。 |
| **sessionMatchIP** | FALSE | TRUE/FALSE (boolean) | 读取会话 cookie 时是否验证用户的 IP 地址。请注意，某些 ISP 会动态更改 IP，因此，如果您希望会话不过期，可能会将其设置为 FALSE。 |
| **sessionTimeToUpdate** | 300 | Time in seconds (integer) | 此选项控制会话类重新生成自身并创建新的频率。会话 ID。将其设置为 0 将禁用会话 ID 再生。 |
| **sessionRegenerateDestroy** | FALSE | TRUE/FALSE (boolean) | 自动重新生成时是否销毁与旧会话 ID 相关联的会话 ID。设置为 FALSE 时，垃圾收集器稍后将删除数据。 |

**注解：** 作为最后的选择，如果未配置上述任何项，则会话库将尝试获取 PHP 的与会话相关的 INI 设置以及旧式 CI 设置，例如 “sess_expire_on_close”。但是，你永远不要依赖此行为，因为它可能导致意外的结果或将来被更改。请正确配置所有内容。

除了上述值之外，cookie 和本机驱动程序还应用了*IncomingRequest* 和*Security* 类共享的以下配置值：

| 配置项 | 默认 | 描述 |
|---|---|---|
| **cookieDomain** | '' | 会话适用的域 |
| **cookiePath** | / | 会话适用的路径 |
| **cookieSecure** | FALSE | 是否仅在加密（HTTPS）连接上创建会话 cookie |

**注解：** “cookieHTTPOnly” 设置对会话没有影响。出于安全原因，始终启用 HttpOnly 参数。此外，“cookiePrefix” 设置被完全忽略。

### Session 驱动程序

如前所述，Session 库带有 4 个处理程序或存储引擎，你可以使用它们：

- CodeIgniterSessionHandlersFileHandler
- CodeIgniterSessionHandlersDatabaseHandler
- CodeIgniterSessionHandlersMemcachedHandler
- CodeIgniterSessionHandlersRedisHandler
- CodeIgniterSessionHandlersArrayHandler

默认情况下，在 `FileHandler` 初始化会话时将使用驱动程序，因为它是最安全的选择，并且有望在任何地方都可以使用（实际上每个环境都有一个文件系统）。

但是，可以选择通过 **app/Config/App.php** 文件中的 `public $sessionDriver` 行选择任何其他驱动程序。请记住，每个驾驶员都有不同的警告，因此在做出选择之前，一定要使自己熟悉（如下）。

**注解：** 在测试期间使用 ArrayHandler 并将其存储在 PHP 数组中，同时防止数据被持久保存。

### FileHandler 驱动程序（默认）

“FileHandler” 驱动程序使用你的文件系统来存储会话数据。

可以肯定地说，它的工作原理与 PHP 自己的默认会话实现完全相同，但是如果这对你来说是一个重要的细节，请记住，它实际上不是相同的代码，并且有一些限制（和优点）。

更具体地说，它不支持 directory level and mode formats used in session.save_path ，并且为了安全起见，大多数选项都经过硬编码。相反，`public $sessionSavePath` 仅支持绝对路径。

你还应该知道的另一件事是，确保不要使用公共可读或共享目录来存储会话文件。确保 只有你有权查看所选 *sessionSavePath* 目录的内容。否则，任何能够做到这一点的人都可以窃取当前的任何会话（也称为“会话固定”攻击）。

在类似 UNIX 的操作系统上，这通常是通过使用 *chmod* 命令在该目录上设置 0700 模式权限来实现的，该命令仅允许目录所有者对目录执行读取和写入操作。但是要小心，因为 运行脚本的系统用户通常不是你自己的，而是“www-data”之类的东西，因此仅设置这些权限可能会破坏你的应用程序。

Instead, you should do something like this, depending on your environment 取而代之的是，你应该根据自己的环境执行类似的操作

```
mkdir /<path to your application directory>/Writable/sessions/
chmod 0700 /<path to your application directory>/Writable/sessions/
chown www-data /<path to your application directory>/Writable/sessions/
```

### Bonus Tip

某些人可能会选择其他会话驱动程序，因为文件存储通常较慢。这只有一半是正确的。

一个非常基本的测试可能会让你相信 SQL 数据库更快，但是在 99%的情况下，只有当你只有几个当前会话时，这才是正确的。随着会话数的增加和服务器负载的增加（这很重要），文件系统将始终胜过几乎所有的关系数据库设置。

此外，如果只考虑性能，则可能需要研究使用 tmpfs ，（警告：外部资源），它可以使会话快速发展。

### DatabaseHandler 驱动程序

“DatabaseHandler”驱动程序使用关系数据库（例如 MySQL 或 PostgreSQL）来存储会话。这是许多用户中的一个流行选择，因为它使开发人员可以轻松访问应用程序中的会话数据 - 它只是数据库中的另一个表。

但是，必须满足一些条件：

- 你不能使用持久连接。
- 你不能在启用 *cacheOn* 设置的情况下使用连接。

为了使用“DatabaseHandler”会话驱动程序，你还必须创建我们已经提到的该表，然后将其设置为你的 `$sessionSavePath` 值。例如，如果你想使用“ci_sessions”作为表名，则可以这样做：

```
public $sessionDriver   = 'CodeIgniter\Session\Handlers\DatabaseHandler';
public $sessionSavePath = 'ci_sessions';
```

然后，当然要创建数据库表…

对于 MySQL：

```
CREATE TABLE IF NOT EXISTS `ci_sessions` (
    `id` varchar(128) 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(128) 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");
```

你还需要 **根据你的 "sessionMatchIP" 设置**添加主键。以下示例在 MySQL 和 PostgreSQL 上均可使用：

```
// 当 sessionMatchIP = TRUE 时
ALTER TABLE ci_sessions ADD PRIMARY KEY (id, ip_address);

// 当 sessionMatchIP = FALSE 时
ALTER TABLE ci_sessions ADD PRIMARY KEY (id);

// 删除先前创建的主键（在更改设置时使用）
ALTER TABLE ci_sessions DROP PRIMARY KEY;
```

你可以通过在 **applicationConfigApp.php** 文件中添加新行并使用要使用的组名来选择要使用的数据库组：

```
public $sessionDBGroup = 'groupName';
```

如果你不想手工完成所有这些操作，则可以使用 **session:migrationcli** 中的命令为你生成一个迁移文件：

```
> php spark session:migration
> php spark migrate
```

该命令在生成代码时将考虑 **sessionSavePath** 和 **sessionMatchIP** 设置。

---

**重要：** 由于缺少其他平台上的建议性锁定机制，因此仅正式支持 MySQL 和 PostgreSQL 数据库。使用不带锁的会话会导致各种问题，尤其是在大量使用 AJAX 的情况下，我们不支持这种情况。`session_write_close()` 如果遇到性能问题，请在处理完会话数

据后使用。

### RedisHandler 驱动程序

**注解：** 由于 Redis 没有公开锁定机制，因此该驱动程序的锁定由一个单独的值模拟，该值最多可保留 300 秒。

Redis 是一种存储引擎，由于其高性能而通常用于缓存并广受欢迎，这可能也是你使用'RedisHandler'会话驱动程序的原因。

缺点是它不像关系数据库那样普遍存在，并且需要在系统上安装 phpredis PHP 扩展，并且没有与 PHP 捆绑在一起。很有可能，仅当你已经熟悉 Redis 并将其用于其他目的时，才使用 RedisHandler 驱动程序。

与 "FileHandler" 和 "DatabaseHandler" 驱动程序一样，你还必须通过该 `$sessionSavePath` 设置配置会话的存储位置。此处的格式有些不同，同时又很复杂。最好用 *phpredis* 扩展的 README 文件来解释，所以我们将简单地链接到它：

```
https://github.com/phpredis/phpredis
```

**警告：** CodeIgniter 的会话库不使用实际的'redis'session.save_handler。仅注意上面链接中的路径格式。

但是，对于最常见的情况，一个简单的 `host:port` 配对就足够了：

```
public $sessionDiver    = 'CodeIgniter\Session\Handlers\RedisHandler';
public $sessionSavePath = 'tcp://localhost:6379';
```

### MemcachedHandler 驱动程序

**注解：** 由于 Memcached 没有公开锁定机制，因此该驱动程序的锁定由一个单独的值模拟，该值最多保留 300 秒。

除了可能的可用性外，"Memcached "驱动程序的所有属性都与 "RedisHandler" 驱动程序非常相似，因为 PHP 的 Memcached 扩展是通过 PECL 分发的，并且某些 Linux 发行版使其可以作为易于安装的软件包使用。

除此之外，对于 Redis 并没有任何故意的偏见，关于 Memcached 的说法没有多大不同 - 它也是一种流行的产品，通常用于缓存并以其速度着称。

但是，值得注意的是，Memcached 给出的唯一保证是将值 X 设置为在 Y 秒后过期将导致在 Y 秒过去之后将其删除（但不一定要在该时间之前过期）。这种情况很少发生，但

是应该考虑，因为这可能会导致会话丢失。

该 $sessionSavePath 格式相当这里简单，仅仅是一对 host:port ：

```
public $sessionDriver    = 'CodeIgniter\Session\Handlers\MemcachedHandler
↪';
public $sessionSavePath = 'localhost:11211';
```

#### Bonus Tip

还支持使用可选的 *weight* 参数作为第三个冒号（ :weight ）值的多服务器配置，但是我们必须注意，我们尚未测试这是否可靠。

如果要尝试使用此功能（后果自负），只需用逗号分隔多个服务器路径：

```
// 相比于 192.0.2.1 权重为 1，本地主机将获得更高的优先级（5）。
public $sessionSavePath = 'localhost:11211:5,192.0.2.1:11211:1';
```

### 7.1.11 限流类



限流类（Throttler）提供了一种非常简单的方法，可以将用户要执行的活动限制为在设定的时间段内只能进行一定次数的尝试。这最常用于对 API 进行速率限制，或限制用户针对表单进行的尝试次数，以帮助防止暴力攻击。该类可用于你根据设置的时间来进行限制的操作。

#### 总览

Throttler 实现了 Token Bucket （令牌桶） 算法的一个简化版本。一般，会将你要执行的每个操作都视为一个存储桶。调用该 check() 方法时，你要告诉它存储桶的大小，可以容纳多少令牌以及时间间隔。在默认情况下，每个 check() 的调用请求将会使用 1 个可用令牌。让我们通过一个例子来阐明这一点。（译注：国内用户可参考 令牌桶 ）

假设我们希望某动作每秒发生一次。对 Throttler 的第一次呼叫将如下所示。第一个参数是存储桶名称，第二个参数是存储桶持有的令牌数量，第三个参数是存储桶重新填充所需的时间：

```
$throttler = \Config\Services::throttler();
$throttler->check($name, 60, MINUTE);
```

我们暂时使用 全局常量 </general/common_functions> 的其中一个，以使其更具可读性。也就是说，这个存储桶每分钟允许执行 60 次操作，或者每秒允许执行 1 次操作。

假设某个第三方脚本试图重复访问 URL 。最初，它能够在不到一秒钟的时间内使用完所有 60 个令牌。但是，在那之后，Throttler 将仅允许每秒执行一次操作，从而有可能减慢请求的速度，以至于让攻击不再有价值。

**注解:** 为了使 Throttler 类可以正常工作，必须将 Cache 库设置为实际可用的缓存对象处理程序。为了获得最佳性能，建议使用像 Redis 或 Memcached 那样的内存缓存。

### 速率限制

Throttler 类不会自发地做任何的请求速率限制或对请求进行限流，但却是上述功能得以实现的关键。这里提供了一个示例过滤器，该过滤器以每个 IP 地址每秒一个请求的速率限制实现了非常简单的速率限制。我们将介绍它的工作原理，以及如何设置它并开始在应用程序中使用它。

### 实现代码

你可以在 **app/Filters/Throttle.php** 上创建自己的 Throttler 过滤器，大致如下：

```
<?php namespace App\Filters;

use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
use Config\Services;

class Throttle implements FilterInterface
{
        /**
         * 这是一个为应用程序使用 Trottler 类来实现速率限制的实例
         *
         * @param RequestInterface|\CodeIgniter\HTTP\IncomingRequest
↪$request
         *
         * @return mixed
         */
        public function before(RequestInterface $request)
        {
            $throttler = Services::throttler();
```

(下页继续)

(续上页)

```
                    // 在整个站点上将 IP 地址限制为每秒不超过 1 个请求
                    if ($throttler->check($request->getIPAddress(), 60,␣
↪MINUTE) === false)
            {
                return Services::response()->setStatusCode(429);
                    }
        }

        //--------------------------------------------------------------------
↪------

        /**
         * 暂时无事可做
         *
         * @param RequestInterface|\CodeIgniter\HTTP\IncomingRequest
↪$request
         * @param ResponseInterface|\CodeIgniter\HTTP\Response
↪$response
         *
         * @return mixed
         */
        public function after(RequestInterface $request,␣
↪ResponseInterface $response)
        {
        }
}
```

运行时，此方法首先获取节流阀的实例。接下来，它将 IP 地址用作存储桶名称，并进行设置以将其限制为每秒一个请求。如果节流阀拒绝检查，返回 false，则我们返回一个状态码为 429（太多尝试的 HTTP Response）的响应，并且脚本执行在调用控制器之前就结束了。本示例将基于对站点的所有请求（而不是每页）中的单个 IP 地址进行限制。

### 应用过滤器

我们不一定需要限制网站上的每个页面。对于许多 Web 应用程序，最有意义的是仅将其应用于 POST 请求，尽管 API 可能希望限制用户发出的每个请求。为了将此应用到传入请求，你需要编辑 **/app/Config/Filters.php** 并首先向过滤器添加别名：

```
public $aliases = [
        ...
        'throttle' => \App\Filters\Throttle::class
];
```

接下来，我们将其分配给网站上的所有 POST 请求：

```
public $methods = [
    'post' => ['throttle', 'CSRF']
];
```

这就是全部。现在，会对网站上发出的所有 POST 请求进行速率限制。

### 类参考

**check**(*string $key*, *int $capacity*, *int $seconds*[, *int $cost = 1*])

**参数**

- **$key** (*string*) – 储存桶的名称
- **$capacity** (*int*) – 储存桶中持有的令牌数量
- **$seconds** (*int*) – 储存桶完全填满的秒数
- **$cost** (*int*) – 此操作将会花费的令牌数量

**返回** 如果可以执行此操作则为 TRUE，否则为 FALSE

**返回类型** bool

检查存储桶中是否还有令牌，或者是否在分配的时间限制内使用了太多令牌。在每次检查期间，如果成功，将根据 $cost 参数来减少可用令牌的数量。

**getTokentime**()

**返回** 直到下一次令牌可用的秒数

**返回类型** int

在 `check()` 运行并返回 FALSE 之后，可以使用此方法确定直到新令牌可用并可以再次尝试操作之前的时间。在这种情况下，最小强制等待时间为一秒。

## 7.1.12 日期与时间类

CodeIgniter 提供了一个完全本地化的，不变的日期与时间类，这个类建立在 PHP 原生的 DateTime 类之上，但使用了 Intl 扩展程序的功能来进行跨时区转换时间并正确显示不同语言环境的输出。这个类就是 **Time** 类，位于 **CodeIgniter\I18n** 命名空间中。

---

**注解：** 由于 Time 类是 DateTime 类的拓展，因此如果您需要此类不提供的功能，可以在 DateTime 类中找到它们。

---

- 处理各个时间的值

### 实例化

有多种创建 Time 类实例的方法。首先是像其他类一样简单地创建一个新实例。当您以这种方式进行操作时，您可以传递一个表示所需时间的字符串。它可以是 PHP 的 strtotime() 函数可以解析的任何字符串：

```
use CodeIgniter\I18n\Time;

$myTime = new Time('+3 week');
$myTime = new Time('now');
```

你可以在参数中分别传递表示时区和语言环境的字符串。时区可以是 PHP 的 DateTimeZone 类可以支持所有时区。语言环境可以是 PHP 的 Locale 类支持的任何语言环境。如果未提供语言环境或时区，则将使用应用程序配置中的默认值。

```
$myTime = new Time('now', 'America/Chicago', 'en_US');
```

### now()

Time 类有几个有用的 helper 方法来实例化这个类，首先是 **now()** 方法，该方法返回设置为当前时间的新实例。您可以在参数中提供表示时区和语言环境的字符串。如果未提供语言环境或时区，则将使用应用程序配置中的默认值。

```
$myTime = Time::now('America/Chicago', 'en_US');
```

### parse()

这个 helper 程序方法是默认的构造函数的 static 版本。它以 DateTime 类构造函数可接受的任何表示时间的字符串为第一个参数，将表示时区的字符串作为第二个参数，将表示语言环境的字符串作为第三个参数：

```
$myTime = Time::parse('next Tuesday', 'America/Chicago', 'en_US');
```

### today()

返回一个新实例，该实例的日期设置为当前日期，时间设置为午夜。它在第一个和第二个参数中分别接受表示时区和语言环境的字符串：

```
$myTime = Time::today('America/Chicago', 'en_US');
```

### yesterday()

返回一个新实例，该实例的日期设置为昨天的日期，时间设置为午夜。它在第一个和第二个参数中分别接受表示时区和语言环境的字符串：

```
$myTime = Time::yesterday('America/Chicago', 'en_US');
```

### tomorrow()

返回一个新实例，该实例的日期设置为明天的日期，时间设置为午夜。它在第一个和第二个参数中分别接受表示时区和语言环境的字符串：

```
$myTime = Time::tomorrow('America/Chicago', 'en_US');
```

### createFromDate()

给定 **年**、**月**、**日**的单独输入，将返回一个新实例。如果未提供它们三个中的任何一个，它将使用当前时间的该值进行填充。在第四和第五个参数中接受时区和语言环境的字符串：

```
$today       = Time::createFromDate();       // 将使用现在时间的年、月、日
$anniversary = Time::createFromDate(2018);  // 将使用现在时间的月、日
$date        = Time::createFromDate(2018, 3, 15, 'America/Chicago', 'en_
↪US');
```

### createFromTime()

与 **createFromDate()** 相似，只不过它只和 **小时**、**分钟**和 **秒**有关。使用当前时间的日期作为 Time 实例的日期部分。在第四个和第五个参数中接受时区和语言环境的字符串：

```
$lunch  = Time::createFromTime(11, 30)       // 11:30 am today
$lunch  = Time::createFromTime(11, 30)      // 今天的 11:30
$dinner = Time::createFromTime(18, 00, 00)  // 6:00 pm today
$dinner = Time::createFromTime(18, 00, 00)  // 今天的 18:00
$time   = Time::createFromTime($hour, $minutes, $seconds, $timezone,
↪$locale);
```

### create()

前面两种方法的组合，将 **年**、**月**、**日**、**小时**、**分钟**和 **秒**作为单独的参数。任何未提供的值将使用当前的日期和时间来确定。在第四个和第五个参数中接受时区和语言环境的字符串：

```
$time = Time::create($year, $month, $day, $hour, $minutes, $seconds,
↪$timezone, $locale);
```

### createFromFormat()

它是替代 DateTime 构造函数的方法。它允许同时设置时区，并返回一个 **Time** 实例，而不是 DateTime 实例：

```
$time = Time::createFromFormat('j-M-Y', '15-Feb-2009', 'America/Chicago
↪');
```

### createFromTimestamp()

该方法使用 UNIX 时间戳以及时区和语言环境（可选）来创建新的 Time 实例：

```
$time = Time::createFromTimestamp(1501821586, 'America/Chicago', 'en_US
↪');
```

### instance()

与提供 DateTime 实例的其他 library 一起使用时，可以使用此方法将其转换为 Time 实例，可以选择设置语言环境。时区将根据传入的 DateTime 实例自动确定：

```
$dt   = new DateTime('now');
$time = Time::instance($dt, 'en_US');
```

### toDateTime()

它不是用来实例化的，此方法与 **实例化**方法相反，它允许您将 Time 实例转换为 DateTime 实例。这样会保留时区设置，但会丢失语言环境，因为 DateTime 并不了解语言环境：

```
$datetime = Time::toDateTime();
```

### 显示时间值

由于 Time 是 DateTime 类的拓展，因此您将获得提供的所有输出方法，包括 format() 方法。但是，DateTime 方法不提供本地化结果。不过，Time 类提供了许多 helper 方法来显示值的本地化版本。

### toLocalizedString()

这是 DateTime 的 format() 方法的本地化版本。但是，必须使用 IntlDateFormatter 类可以接受的值，而不能使用你熟悉的值。完整的值列表可以在 这里 找到。

```
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toLocalizedString('MMM d, yyyy');   // March 9, 2016
```

### toDateTimeString()

这是与 IntlDateFormatter 一起使用的三种辅助方法中的第一种，无需记住它们的值。这将返回一个格式化的字符串，该字符串的格式与数据库中日期时间列的常用格式相同（Y-m-d H:i:s）：

```
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toDateTimeString();     // 2016-03-09 12:00:00
```

### toDateString()

仅返回时间与日期的日期部分：

```
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toDateTimeString();     // 2016-03-09
```

### toTimeString()

仅返回时间与日期的时间部分：

```
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toTimeString();     // 12:00:00
```

### humanize()

此方法返回一个字符串，该字符串以易于理解的人类可读格式显示当前日期或时间与实例之间的差异。它会返回“3 小时前”、“1 个月内”等字符串：

```
// 假设现在的时间是: March 10, 2017 (America/Chicago)
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');

echo $time->humanize();     // 1 year ago
```

通过以下方式确定显示的确切时间：

| 时间差异 | 结果 |
|---|---|
| $time > 1 year && < 2 years | in 1 year / 1 year ago |
| $time > 1 month && < 1 year | in 6 months / 6 months ago |
| $time > 7 days && < 1 month | in 3 weeks / 3 weeks ago |
| $time > today && < 7 days | in 4 days / 4 days ago |
| $time == tomorrow / yesterday | Tomorrow / Yesterday |
| $time > 59 minutes && < 1 day | 1:37pm |
| $time > now && < 1 hour | in 35 minutes / 35 minutes ago |
| $time == now | Now |

返回的结果的语言被语言文件 Time.php 所控制。

### 处理各个时间的值

Time 对象提供了许多方法来获取和设置现有实例的各个项目，例如年、月、时等。通过以下方法检索到的的所有值都会被完全本地化，并遵守创建 Time 实例所使用的语言环境。

以下所有 *getX* 和 *setX* 方法也可以当作类属性使用。因此，对像 *getYear* 这样调用的方法也可以通过 *$time->year* 进行调用，依此类推。

### 获取器

有以下几种基本的获取器：

```
$time = Time::parse('August 12, 2016 4:15:23pm');

echo $time->getYear();      // 2016
echo $time->getMonth();     // 8
echo $time->getDay();       // 12
echo $time->getHour();      // 16
echo $time->getMinute();    // 15
echo $time->getSecond();    // 23

echo $time->year;           // 2016
echo $time->month;          // 8
echo $time->day;            // 12
echo $time->hour;           // 16
echo $time->minute;         // 15
echo $time->second;         // 23
```

除这些之外，还有许多方法可以获取有关日期的其他信息：

```
$time = Time::parse('August 12, 2016 4:15:23pm');
```

(下页继续)

(续上页)

```
echo $time->getDayOfWeek();      // 6 - 但可能会因地区的一个星期的第一天而
有所不同
echo $time->getDayOfYear();      // 225
echo $time->getWeekOfMonth();    // 2
echo $time->getWeekOfYear();     // 33
echo $time->getTimestamp();      // 1471018523 - UNIX 时间戳
echo $time->getQuarter();        // 3

echo $time->dayOfWeek;           // 6
echo $time->dayOfYear;           // 225
echo $time->weekOfMonth;         // 2
echo $time->weekOfYear;          // 33
echo $time->timestamp;           // 1471018523
echo $time->quarter;             // 3
```

### getAge()

返回 Time 实例与当前时间之间的差值（以年为单位）。主要是用于根据某人的生日检查其年龄：

```
$time = Time::parse('5 years ago');

echo $time->getAge();    // 5
echo $time->age;         // 5
```

### getDST()

根据 Time 实例是否正在遵守夏令时，返回布尔值 true 或 false：

```
echo Time::createFromDate(2012, 1, 1)->getDst();      // false
echo Time::createFromDate(2012, 9, 1)->dst;      // true
```

### getLocal()

如果 Time 实例的时区与 web 应用程序当前所在的时区位于同一时区，则返回布尔值 true：

```
echo Time::now()->getLocal();                    // true
echo Time::now('Europe/London')->getLocal();     // false
```

### getUtc()

如果 Time 实例使用 UTC 时间，则返回 true：

```
echo Time::now('America/Chicago')->getUtc();     // false
echo Time::now('UTC')->utc;                       // true
```

### getTimezone()

返回一个新的 DateTimeZone 实例，该实例是 Time 实例的时区：

```
$tz = Time::now()->getTimezone();
$tz = Time::now()->timezone;

echo $tz->getName();
echo $tz->getOffset();
```

### getTimezoneName()

返回 Time 实例的 完整时区字符串 ：

```
echo Time::now('America/Chicago')->getTimezoneName();    // America/
↪Chicago
echo Time::now('Europe/London')->timezoneName;           // Europe/London
```

### 设置器

存在以下的基本设置器。如果设置的任何值超出允许范围，则会抛出 InvalidArgumentExeption 。

**注解：** 所有设置器都将返回一个新的 Time 实例，而原始实例保持不变。

**注解：** 如果值超出范围，则设置器将抛出 InvalidArgumentException。

```
$time = $time->setYear(2017);
$time = $time->setMonthNumber(4);            // April
$time = $time->setMonthLongName('April');
$time = $time->setMonthShortName('Feb');     // February
$time = $time->setDay(25);
$time = $time->setHour(14);                  // 2:00 pm
```

(下页继续)

(续上页)

```
$time = $time->setMinute(30);
$time = $time->setSecond(54);
```

### setTimezone()

将时间从当前时区转换为新时区：

```
$time  = Time::parse('May 10, 2017', 'America/Chicago');
$time2 = $time->setTimezone('Europe/London');           // 将时间从当前时区转换为新时区

echo $time->timezoneName;   // American/Chicago
echo $time2->timezoneName;  // Europe/London
```

### setTimestamp()

返回日期设置为新时间戳的新实例：

```
$time = Time::parse('May 10, 2017', 'America/Chicago');
$time2 = $time->setTimestamp(strtotime('April 1, 2017'));

echo $time->toDateTimeString();     // 2017-05-10 00:00:00
echo $time2->toDateTimeString();     // 2017-04-01 00:00:00
```

### Modifying the Value

通过以下方法，您可以通过在当前时间上增加或减少值来修改日期。这不会修改现有的 Time 实例，只会返回一个新实例。

```
$time = $time->addSeconds(23);
$time = $time->addMinutes(15);
$time = $time->addHours(12);
$time = $time->addDays(21);
$time = $time->addMonths(14);
$time = $time->addYears(5);

$time = $time->subSeconds(23);
$time = $time->subMinutes(15);
$time = $time->subHours(12);
$time = $time->subDays(21);
$time = $time->subMonths(14);
$time = $time->subYears(5);
```

### 比较两个 Time

以下方法使您可以将一个 Time 实例与另一个 Time 实例进行比较。在进行比较之前，首先将所有比较转换为 UTC，以确保不同时区都正确响应。

#### equals()

确定传入的日期时间是否等于当前实例。在这种情况下，相等意味着它们表示同一时间，并且不需要位于同一时区，因为两个时间都转换为 UTC 并以这种方式进行比较：

```
$time1 = Time::parse('January 10, 2017 21:50:00', 'America/Chicago');
$time2 = Time::parse('January 11, 2017 03:50:00', 'Europe/London');

$time1->equals($time2);     // true
```

要作比较的值可以是 Time 实例，DateTime 实例或 DateTime 类可以理解的任何表示时间的字符串。当将字符串作为第一个参数传递时，可以将时区字符串作为第二个参数传递。如果没有给出时区，将使用配置的默认值：

```
$time1->equals('January 11, 2017 03:50:00', 'Europe/London');  // true
```

#### sameAs()

除了只有在日期，时间和时区都相同时才返回 true，这与 **equals** 方法相同：

```
$time1 = Time::parse('January 10, 2017 21:50:00', 'America/Chicago');
$time2 = Time::parse('January 11, 2017 03:50:00', 'Europe/London');

$time1->sameAs($time2);     // false
$time2->sameAs('January 10, 2017 21:50:00', 'America/Chicago');     //␣
↪true
```

#### isBefore()

检查传入的时间是否在当前实例之前。两种情况下都针对 UTC 版本进行了比较：

```
$time1 = Time::parse('January 10, 2017 21:50:00', 'America/Chicago');
$time2 = Time::parse('January 11, 2017 03:50:00', 'America/Chicago');

$time1->isBefore($time2);  // true
$time2->isBefore($time1);  // false
```

要作比较的值可以是 Time 实例，DateTime 实例或 DateTime 类可以理解的任何表示时间的字符串。当将字符串作为第一个参数传递时，可以将时区字符串作为第二个参数传递。如果没有给出时区，将使用配置的默认值：

```
$time1->isBefore('March 15, 2013', 'America/Chicago');  // false
```

### isAfter()

除了检查时间是否在传入的时间之后，其他的与 **isBefore()** 完全相同：

```
$time1 = Time::parse('January 10, 2017 21:50:00', 'America/Chicago');
$time2 = Time::parse('January 11, 2017 03:50:00', 'America/Chicago');

$time1->isAfter($time2);  // false
$time2->isAfter($time1);  // true
```

### 查看差异

要直接比较两个 Times，可以使用 **difference()** 方法，该方法返回 **CodeIgniter\I18n\TimeDifference** 实例。第一个参数可以是 Time 实例、DateTime 实例或带有日期或时间的字符串。如果在第一个参数中传递了表示时间字符串，则第二个参数可以是时区字符串：

```
$time = Time::parse('March 10, 2017', 'America/Chicago');

$diff = $time->difference(Time::now());
$diff = $time->difference(new DateTime('July 4, 1975', 'America/Chicago
↪');
$diff = $time->difference('July 4, 1975 13:32:05', 'America/Chicago');
```

有了 TimeDifference 实例后，您可以使用多种方法来查找有关两个 Time 间的信息。如果比较时间在待比较时间之前，则返回值为负数；反之，如果比较时间在带比较时间之后，则返回的值为正数：

```
$current = Time::parse('March 10, 2017', 'America/Chicago');
$test    = Time::parse('March 10, 2010', 'America/Chicago');

$diff = $current->difference($test);

echo $diff->getYears();      // -7
echo $diff->getMonths();     // -84
echo $diff->getWeeks();      // -365
echo $diff->getDays();       // -2557
echo $diff->getHours();      // -61368
echo $diff->getMinutes();    // -3682080
echo $diff->getSeconds();    // -220924800
```

你可以用 **getX()** 方法，也可以像使用属性一样访问计算值：

```
echo $diff->years;      // -7
echo $diff->months;     // -84
echo $diff->weeks;      // -365
echo $diff->days;       // -2557
echo $diff->hours;      // -61368
echo $diff->minutes;    // -3682080
echo $diff->seconds;    // -220924800
```

#### humanize()

与 Time 的 humanize() 方法非常相似，此方法返回一个字符串，该字符串以易于理解的格式显示时间之间的时差。它可以创建像 “3 小时前”、“1 个月内” 这样的的字符串。它们之间最大的区别在于最近日期的处理方式：

```
// Assume current time is: March 10, 2017 (America/Chicago)
// 假设现在时间是：  March 10, 2017 (America/Chicago)
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');

echo $time->humanize();      // 1 year ago
```

通过以下方式确定显示的确切时间：

| 时间差异 | 结果 |
|---|---|
| $time > 1 year && < 2 years | in 1 year / 1 year ago |
| $time > 1 month && < 1 year | in 6 months / 6 months ago |
| $time > 7 days && < 1 month | in 3 weeks / 3 weeks ago |
| $time > today && < 7 days | in 4 days / 4 days ago |
| $time > 1 hour && < 1 day | in 8 hours / 8 hours ago |
| $time > 1 minute && < 1 hour | in 35 minutes / 35 minutes ago |
| $time < 1 minute | Now |

返回的结果的语言被语言文件 Time.php 所控制。

### 7.1.13 Typography 类

Typography 库包含一些方法用于帮助您以语义相关的方式设置文本格式。

#### 加载类库

与 CodeIgniter 的所有其他服务一样，可以通过 `Config\Services` 来加载，通常不需要手动加载：

```
$typography = \Config\Services::typography();
```

### 可用的静态方法

以下的方法是可用的：

**autoTypography()**

`autoTypography`(*$str*[, *$reduce_linebreaks = FALSE*])

**参数**

- **$str** (*string*) – Input string
- **$reduce_linebreaks** (*bool*) – 是否将多个双重换行减少为两个

**返回** HTML 格式化的排版安全的字符串

**返回类型** string

格式化文本使其成为语义和排版正确的 HTML 。

使用示例:

```
$string = $typography->autoTypography($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，你可以考虑缓存 你的页面。

**formatCharacters()**

`formatCharacters`(*$str*)

**参数**

- **$str** (*string*) – Input string

**返回** 带有格式化字符的字符串

**返回类型** string

将双引号或单引号转成正确的实体，也会转化—破折号，双空格和 & 符号。

使用示例:

```
$string = $typography->formatCharacters($string);
```

**nl2brExceptPre()**

`nl2brExceptPre`(*$str*)

**参数**

- **$str** (*string*) – Input string

**返回** 带有 HTML 格式化换行符的字符串

**返回类型** string

将换行转换为 <br /> 标签，忽略 <pre> 标签中的换行符。这个函数和 PHP 原生的 `nl2br()` 函数是一样的，但忽略 <pre> 标签。

使用示例:

```
$string = $typography->nl2brExceptPre($string);
```

### 7.1.14 使用文件上传类

在 CodeIgniter 中通过表单使用文件上传功能将会比直接使用 PHP 的 `$_FILES` 数组更加简单和安全。其将继承文件类 并获取该类的所有功能。

---

**注解:** 这和 CodeIgniter 的上一版本的文件上传类不同。这次提供了一个原生接口及一些小功能来上传文件。上传类将在最终版的时提供。

---



#### 访问文件

#### 所有文件

当你上传文件时，PHP 可以在本地使用全局数组 `$_FILES` 来访问这些文件。当你同时上传多个文件时，这个数组存在一些不可忽视的缺点和很多开发者没有意识到的安全方面的潜在缺陷。CodeIgniter 通过一个公共接口来规范你对文件的使用，从而改善这些问题。

通过当前的 `IncomingRequest` 实例来访问文件。使用 `getFiles()` 方法来获取本次请求中上传的所有文件。方法将会返回由 `CodeIgniter\HTTP\Files\UploadedFile` 实例表示的文件数组:

```
$files = $this->request->getFiles();
```

当然，有很多种方式来为文件 input 标签命名，除了最简外任何其他任何命名方式都可能产生奇怪的结果。数组将会以你期望的方式返回。使用最简方式，一个单文件提交表单可能会是这样:

```
<input type="file" name="avatar" />
```

其将会返回一个简单的数组像是:

```
[
        'avatar' => // UploadedFile instance
]
```

如果你在标签名称中使用数组表示法，input 标签将看上去像是这样:

```
<input type="file" name="my-form[details][avatar]" />
```

getFiles() 方法返回的数组看上去将像是这样:

```
[
        'my-form' => [
                'details' => [
                        'avatar' => // UploadedFile instance
                ]
        ]
]
```

在某些情况下，你可以指定一组文件元素来上传:

```
Upload an avatar: <input type="file" name="my-form[details][avatars][]" /
↪>
Upload an avatar: <input type="file" name="my-form[details][avatars][]" /
↪>
```

在这种情况下，返回的文件数组将会像是这样:

```
[
        'my-form' => [
                'details' => [
                        'avatar' => [
                                0 => /* UploadedFile instance */,
                                1 => /* UploadedFile instance */
                ]
        ]
]
```

### 单个文件

如果你只需要访问单个文件，你可以使用 getFile() 方法来直接获取文件实例。其将会返回一个 CodeIgniter\HTTP\Files\UploadedFile 实例:

### 最简使用

使用最简方式，一个单文件提交表单可能会是这样:

```
<input type="file" name="userfile" />
```

其将会返回一个简单的文件实例像是:

```
$file = $this->request->getFile('userfile');
```

### 数组表示法

如果你在标签名称中使用数组表示法，input 标签将看上去像是这样:

```
<input type="file" name="my-form[details][avatar]" />
```

这样来获取文件实例:

```
$file = $this->request->getFile('my-form.details.avatar');
```

### 多文件

<input type=" file" name=" images[]" multiple />

**在控制器中::** if($imagefile = $this->request->getFiles()) {

foreach($imagefile[ 'images' ] as $img) {

if ($img->isValid() && ! $img->hasMoved()) {

$newName = $img->getRandomName(); $img->move(WRITEPATH.' uploads' , $newName);

}

}

}

循环中的 **images** 是表单中的字段名称

如果多个文件使用相同名称提交，你可以使用 getFile() 去逐个获取每个文件:: 在控制器中:

```
$file1 = $this->request->getFile('images.0');
$file2 = $this->request->getFile('images.1');
```

另外一个例子:

```
Upload an avatar: <input type="file" name="my-form[details][avatars][]" /
 ↪>
Upload an avatar: <input type="file" name="my-form[details][avatars][]" /
 ↪>
```

在控制器中:

```
$file1 = $this->request->getFile('my-form.details.avatars.0');
$file2 = $this->request->getFile('my-form.details.avatars.1');
```

**注解:** 使用 getFiles() 更合适。

### 使用文件

一旦你获取到了 UploadedFile 实例, 你可以以安全的方式检索到文件的信息，还能将文件移动到新的位置。

### 验证文件

你可以调用 isValid() 方法来检查文件是否是通过 HTTP 无误上传的:

```
if (! $file->isValid())
{
        throw new RuntimeException($file->getErrorString().'('.$file->
 ↪getError().')');
}
```

如这个例子所见，如果一个文件产生一个上传错误，你可以通过 getError() 和 getErrorString() 方法获取错误码（一个整数）和错误消息。通过此方法可以发现以下错误:

- 文件大小超过了 upload_max_filesize 配置的值。
- 文件大小超过了表单定义的上传限制。
- 文件仅部分被上传。
- 没有文件被上传。
- 无法将文件写入磁盘。
- 无法上传文件：缺少临时目录。

- PHP 扩展阻止了文件上传。

### 文件名称

#### getName()

你可以通过 getName() 提取到客户端提供的文件的原始名称。其通常是由客户端发送的文件名，不应受信。如果文件已经被移动，将返回移动文件的最终名称:

```
$name = $file->getName();
```

#### getClientName()

总是返回由客户端发送的上传文件的原始名称，即使文件已经被移动了:

```
$originalName = $file->getClientName();
```

#### getTempName()

要获取在上传期间产生的临时文件的全路径，你可以使用 getTempName() 方法:

```
$tempfile = $file->getTempName();
```

### 其他文件信息

#### getClientExtension()

基于上传文件的名称，返回原始文件扩展名。这不是一个值得信赖的来源。对于可信的版本，请使用 getExtension() 来代替:

```
$ext = $file->getClientExtension();
```

#### getClientType()

返回由客户端提供的文件的媒体类型 (mime type)。这不是一个值得信赖的值，对于可信的版本，请使用 getType() 来代替:

```
$type = $file->getClientType();

echo $type; // image/png
```

### 移动文件

每个文件都可以使用恰如其名的“move()‘ 方法来移动到新的位置。使用第一个参数为目标目录来移动文件:

```
$file->move(WRITEPATH.'uploads');
```

默认的，将使用文件原始名称。你可以指定一个新的文件名称作为第二个参数传递给方法。

```
$newName = $file->getRandomName(); $file->move(WRITEPATH.'uploads', $newName);
```

一旦文件被移除，将删除临时文件。你可以通过 `hasMoved()` 方法来检查文件是否已经被移动了，返回布尔值:

```
if ($file->isValid() && ! $file->hasMoved())
{
    $file->move($path);
}
```

### 7.1.15 使用 URI 类

CodeIngiter 为你在应用中使用 URI 类提供了一个面向对象的解决方案。使用这种方式可以轻易地确保结构始终准确，无论 URI 的复杂程度如何，也能将相对 URI 添加到现有应用中，并保证其可以被安全、准确地解析。



#### 创建 URI 实例

就像创建一个普通类实例一样去创建一个 URI 实例:

```
$uri = new \CodeIgniter\HTTP\URI();
```

或者，你可以使用 service() 方法来返回一个 URI 实例:

```
$uri = service('uri');
```

当创建新实例的时候，你可以将完整或部分 URL 传递给构造函数，其将会被解析为相应的分段:

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com/some/path');
$uri = service('uri', 'http://www.example.com/some/path');
```

### 当前 URI

很多时候，你真正想要的是一个表示着当前请求 URL 的对象。可以有两种不同的方式来获取。第一，直接从当前请求对象中提取。假设你所在的控制器已继承自 CodeIgniter\Controller，可以这样做:

```
$uri = $this->request->uri;
```

第二，你可以使用 **url_helper** 中的一个可用函数来获取:

```
helper('url');
$uri = current_url(true);
```

你必须在第一个参数中传递 true, 否则该函数将仅返回表示当前 URL 的字符串。

### URI 字符串

很多时候，你真正想要的是得到一个表示 URI 的字符串。那直接将 URI 对象转换为字符串就可以了:

```
$uri = current_url(true);
echo (string)$uri;  // http://example.com
```

如果你知道 URI 的各个部分，同时还想确保其格式准确无误，你可以通过使用 URI 类的静态方法 createURIString() 来生成字符串:

```
$uriString = URI::createURIString($scheme, $authority, $path, $query,
↪$fragment);

// Creates: http://exmample.com/some/path?foo=bar#first-heading
echo URI::createURIString('http', 'example.com', 'some/path', 'foo=bar',
↪'first-heading');
```

### URI 的组成

一旦你得到了一个 URI 实例，你就可以设置或检索这个 URI 的任意部分。本节将详细介绍这些部分的内容及如何使用它们。

#### Scheme

最常见的传输协议是‘http’或‘https’，同时也支持如‘file’，‘mailto’等其他协议。

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com/some/path');

echo $uri->getScheme(); // 'http'
$uri->setScheme('https');
```

#### Authority

许多 URI 内装载着被统称为‘authority’的数个元素，包括用户信息，主机地址和端口号。你可以通过 `getAuthority()` 方法来获取一个包含了所有相关元素的字符串，也可以对独立的元素进行操作。

```
$uri = new \CodeIgniter\HTTP\URI('ftp://user:password@example.com:21/
↪some/path');

echo $uri->getAuthority();  // user@example.com:21
```

默认情况下，因为你不希望向别人展示密码，所以它不会被显示出来。如你想展示密码，可以使用 `showPassword()` 方法。URI 实例会在你再次关掉显示之前一直保持密码部分地展示，所以你应在使用完成后立刻关闭它:

```
echo $uri->getAuthority();  // user@example.com:21
echo $uri->showPassword()->getAuthority();   // user:password@example.
↪com:21

// Turn password display off again.
$uri->showPassword(false);
```

如果你不想显示端口，可以传递唯一参数 `true`:

```
echo $uri->getAuthority(true);  // user@example.com
```

**注解：** 如果当前端口值是传输协议的默认端口值，那它将永远不会被显示。

### Userinfo

用户信息部分是在使用 FTP URI 时你看到的用户名和密码。当你能在 Authority 中得到它时，你也可以通过方法直接获取它:

```
echo $uri->getUserInfo();    // user
```

默认情况下，它将不会展示密码，但是你可以通过 showPassword() 方法来重写它:

```
echo $uri->showPassword()->getUserInfo();    // user:password
$uri->showPassword(false);
```

### Host

URI 的主机部分通常是 URL 的域名。可以通过 getHost() 和 setHost() 方法很容易地设置和获取:

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com/some/path');

echo $uri->getHost();    // www.example.com
echo $uri->setHost('anotherexample.com')->getHost();     //␣
↪anotherexample.com
```

### Port

端口值是一个在 0 到 65535 之间的整数。每个协议都会有一个与之关联的默认端口值。

```
$uri = new \CodeIgniter\HTTP\URI('ftp://user:password@example.com:21/
↪some/path');

echo $uri->getPort();    // 21
echo $uri->setPort(2201)->getPort(); // 2201
```

当使用 setPort() 方法时，端口值会在通过可用范围值检查后被设置。

### Path

路径是站点自身的所有分段。如你所料，可以使用 getPath() 和 setPath() 方法来操作它:

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com/some/path');

echo $uri->getPath();    // 'some/path'
echo $uri->setPath('another/path')->getPath();  // 'another/path'
```

**注解：** 以这种方式或类允许的其他方式设置 path 的时候，将会对危险字符进行编码，并移除点分段来确保安全。

### Query

查询变量可以通过类使用简单的字符串来调整。Query 的值通常只能设定为一个字符串。

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com?foo=bar');

echo $uri->getQuery();  // 'foo=bar'
$uri->setQuery('foo=bar&bar=baz');
```

**注解：** Query 值不能包含片段，否则会抛出一个 InvalidArgumentException 异常。

你可以使用一个数组来设置查询值:

```
$uri->setQueryArray(['foo' => 'bar', 'bar' => 'baz']);
```

`setQuery()` 和 `setQueryArray()` 方法会重写已经存在的查询变量。你可以使用 `addQuery()` 方法在不销毁已存在查询变量的前提下追加值。第一个参数是变量名，第二个参数是值:

```
$uri->addQuery('foo', 'bar');
```

#### 过滤查询值

你可以对 `getQuery()` 方法传递一个选项数组来过滤查询返回值，使用关键字 *only* 或 *except*:

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com?foo=bar&bar=baz&
↪baz=foz');

// Returns 'foo=bar'
echo $uri->getQuery(['only' => ['foo']);

// Returns 'foo=bar&baz=foz'
echo $uri->getQuery(['except' => ['bar']]);
```

这样只是对调用方法后的返回值进行更改。如果你需要对 URI 对象的查询值进行永久地更改，可以使用 `stripQuery()` 和 `keepQuery()` 方法来更改真实对象的查询变量:

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com?foo=bar&bar=baz&
↪baz=foz');
```

(下页继续)

(续上页)

```
// Leaves just the 'baz' variable
$uri->stripQuery('foo', 'bar');

// Leaves just the 'foo' variable
$uri->keepQuery('foo');
```

### Fragment

片段是 URL 的结尾部分，前面是英镑符号 (#)。在 HTML 中，它们是指向页面锚点的链接。媒体 URI 可以用其他各种方法来使用它们。

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com/some/path#first-
↪heading');

echo $uri->getFragment();   // 'first-heading'
echo $uri->setFragment('second-heading')->getFragment();    // 'second-
↪heading'
```

### URI 分段

路径中，斜杠之间的每一节都是一个单独的分段。URI 类提供一个简单的方式去界定段值。路径最左侧的段为起始段 1。

```
// URI = http://example.com/users/15/profile

// Prints '15'
if ($request->uri->getSegment(1) == 'users')
{
        echo $request->uri->getSegment(2);
}
```

你能得到总分段数量:

```
$total = $request->uri->getTotalSegments(); // 3
```

最后，你能获取到一个包含着所有分段的数组:

```
$segments = $request->uri->getSegments();

// $segments =
[
        0 => 'users',
        1 => '15',
        2 => 'profile'
]
```

### 7.1.16 User Agent Class

The User Agent Class provides functions that help identify information about the browser, mobile device, or robot visiting your site.



#### Using the User Agent Class

#### Initializing the Class

The User Agent class is always available directly from the current *IncomingRequest* instance. By default, you will have a request instance in your controller that you can retrieve the User Agent class from:

```
$agent = $this->request->getUserAgent();
```

#### User Agent Definitions

The user agent name definitions are located in a config file located at: **app/Config/UserAgents.php**. You may add items to the various user agent arrays if needed.

#### Example

When the User Agent class is initialized it will attempt to determine whether the user agent browsing your site is a web browser, a mobile device, or a robot. It will also gather the platform information if it is available:

```
$agent = $this->request->getUserAgent();

if ($agent->isBrowser())
{
        $currentAgent = $agent->getBrowser().' '.$agent->getVersion();
}
elseif ($agent->isRobot())
```

(下页继续)

(续上页)

```
{
        $currentAgent = $this->agent->robot();
}
elseif ($agent->isMobile())
{
        $currentAgent = $agent->getMobile();
}
else
{
        $currentAgent = 'Unidentified User Agent';
}

echo $currentAgent;

echo $agent->getPlatform(); // Platform info (Windows, Linux, Mac, etc.)
```

### Class Reference

**CodeIgniter\HTTP\UserAgent**

**isBrowser**(*[$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

Returns TRUE/FALSE (boolean) if the user agent is a known web browser.

```
if ($agent->isBrowser('Safari'))
{
        echo 'You are using Safari.';
}
elseif ($agent->isBrowser())
{
        echo 'You are using a browser.';
}
```

**注解:** The string "Safari" in this example is an array key in the list of browser definitions. You can find this list in **app/Config/UserAgents.php** if you want to add new browsers or change the strings.

**isMobile**(*[$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

Returns TRUE/FALSE (boolean) if the user agent is a known mobile device.

```
if ($agent->isMobile('iphone'))
{
        echo view('iphone/home');
}
elseif ($agent->isMobile())
{
        echo view('mobile/home');
}
else
{
        echo view('web/home');
}
```

`isRobot`([*$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

Returns TRUE/FALSE (boolean) if the user agent is a known robot.

---

**注解:** The user agent library only contains the most common robot definitions. It is not a complete list of bots. There are hundreds of them so searching for each one would not be very efficient. If you find that some bots that commonly visit your site are missing from the list you can add them to your **app/Config/UserAgents.php** file.

---

`isReferral`()

**返回** TRUE if the user agent is a referral, FALSE if not

**返回类型** bool

Returns TRUE/FALSE (boolean) if the user agent was referred from another site.

`getBrowser`()

**返回** Detected browser or an empty string

**返回类型** string

Returns a string containing the name of the web browser viewing your site.

`getVersion()`

**返回** Detected browser version or an empty string

**返回类型** string

Returns a string containing the version number of the web browser viewing your site.

`getMobile()`

**返回** Detected mobile device brand or an empty string

**返回类型** string

Returns a string containing the name of the mobile device viewing your site.

`getRobot()`

**返回** Detected robot name or an empty string

**返回类型** string

Returns a string containing the name of the robot viewing your site.

`getPlatform()`

**返回** Detected operating system or an empty string

**返回类型** string

Returns a string containing the platform viewing your site (Linux, Windows, OS X, etc.).

`getReferrer()`

**返回** Detected referrer or an empty string

**返回类型** string

The referrer, if the user agent was referred from another site. Typically you'll test for this as follows:

```
if ($agent->isReferral())
{
        echo $agent->referrer();
}
```

`getAgentString()`

**返回** Full user agent string or an empty string

**返回类型** string

Returns a string containing the full user agent string. Typically it will be something like this:

```
Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.0.4)␣
↪Gecko/20060613 Camino/1.0.2
```

**parse**(*$string*)

**参数**

- **$string** (*string*) – A custom user-agent string

**返回类型** void

Parses a custom user-agent string, different from the one reported by the current visitor.

### 7.1.17 验证类

CodeIgniter 提供了全面的数据验证类，最大程度减少你需要编写的代码量。

## 概述

在解释 CodeIgniter 的数据验证之前，我们先介绍理想的状况：

1. 显示一个表单。
2. 你填写并提交。
3. 如果你提交的表单数据无效，或者丢失了必填项，将重新显示包含了你的数据和错误消息的表单。
4. 这个过长将一直持续到你提交的表单数据有效为止。

在接收端，脚本必须：

1. 检查需要的数据。
2. 验证数据的类型是否正确，并且符合要求的标准。例如，如果提交了用户名，则必须仅包含允许的字符。它必须大于最小长度，小于最大长度。用户名不能是系统中已经存在的用户名，甚至是保留的关键词。等等。
3. 处理数据以保证安全。
4. 如果有必要，对数据进行格式化 (是否需要裁剪数据? HTML 编码? 等等。)
5. 准备要插入数据库的数据。

尽管上述过程没有什么非常复杂的，但是通常需要编写大量的代码，并且显示各种错误消息，在 HTML 表单中放置各种控制结构。表单验证虽然容易创建，但是通常十分混乱，实现起来很繁琐。

## 表单验证教程

以下是实现 CodeIgniter 表单验证的“动手”教程。

为了实现表单验证，你需要做三件事：

1. 一个包含表单的*View* 文件。
2. 一个提交成功后显示 “success” 的 View 文件。
3. 一个*controller* 方法用来接收和处理提交的数据。

让我们以会员注册表单为例来做这三件事。

### 表单

使用编辑器创建一个名为 **Signup.php** 的视图文件，将代码复制到文件中，并保存到 **app/Views/** 文件夹：

```
<html>
<head>
    <title>My Form</title>
</head>
<body>

    <?= $validation->listErrors() ?>

    <?= 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

使用编辑器创建一个名为 **Success.php** 的视图文件，将代码复制到文件中，并保存到 **app/Views/** 文件夹：

```
<html>
<head>
    <title>My Form</title>
</head>
<body>

    <h3>Your form was successfully submitted!</h3>

    <p><?= anchor('form', 'Try it again!') ?></p>

</body>
</html>
```

### 控制器

使用编辑器创建一个名为 **Form.php** 的控制器文件，将代码复制到文件中，并保存到 **app/Controllers/** 文件夹：

```
<?php namespace App\Controllers;

use CodeIgniter\Controller;

class Form extends Controller
{
    public function index()
    {
        helper(['form', 'url']);

        if (! $this->validate([]))
        {
            echo view('Signup', [
                'validation' => $this->validator
            ]);
        }
        else
        {
            echo view('Success');
        }
    }
}
```

### 试一试!

要尝试使用表单，请使用与此网址相似的网址访问你的网站

```
example.com/index.php/form/
```

如果你提交表单，则应该只看到表单重新加载。那是因为你没有设置任何验证规则。

---

**注解:** 由于你没有告诉 **Validation 类**进行任何验证，它在 **默认情况**下 **返回 false** (boolean false)。`validate()` 方法仅在验证你设置的 **所有规则**并且没有 **任何失败**的情况下返回 true 。

---

### 说明

你会注意到上述页面的几件事情：

表单 (Signup.php) 是一个标准的 web 表单，但有一些例外：

1. 它使用表单辅助类来创建表单。从技术上讲这没必要，你可以使用标准的 HTML 代码来创建表单。但是，使用表单辅助类可以根据配置文件中的 URL 来生成表单的 action URL。当你的网址发生更改时，则你的程序更容易进行移植。
2. 在表单的顶部，你会注意到调用了以下函数：

   ```
   <?= $validation->listErrors() ?>
   ```

   该函数将返回 validator 发送的所有错误消息。如果没有消息，则返回一个空字符串。

控制器 (Form.php) 拥有一个方法：`index()`。这个方法使用控制器提供的 validate 方法，并加载表单辅助类和 URL 辅助类。它还运行验证程序，根据验证程序是否验证成功，它将显示表单或成功页。

### 加载 validation 库

该库通过名叫 **validation** 的服务进行加载：

```
$validation =  \Config\Services::validation();
```

这将自动加载 `Config\Validation` 文件，文件中包含了多个规则类，以及便于重用的规则集合。

---

**注解:** 你可用永远都不会使用该方法，因为*Controller* 和*Model* 中都提供了更简便的验证方法。

---

### 设置验证规则

CodeIgniter 允许你为给定字段设置多个验证规则，并按顺序执行它们。要设置验证规则，你将使用 `setRule()`，`setRules()` 方法。

#### setRule()

该方法设置单个规则，它使用 **字段名称**作为第一个参数，第二个参数是一个可选的标签，第三个参数是以竖线分隔的规则列表的字符串：

```
$validation->setRule('username', 'Username', 'required');
```

**字段名称**必须与需要验证的任何数据数组的键匹配。如果直接从 $_POST 获取数组，则它必须与表单的 input name 完全匹配。

#### setRules()

与 `setRule()` 类似，但其接受字段名称与其规则所组成的数组：

```
$validation->setRules([
    'username' => 'required',
    'password' => 'required|min_length[10]'
]);
```

想设置带标签的错误消息，你可以像这样设置：

```
$validation->setRules([
    'username' => ['label' => 'Username', 'rules' => 'required'],
    'password' => ['label' => 'Password', 'rules' => 'required|min_
↪length[10]']
]);
```

#### withRequest()

使用验证库最常见的场景之一是验证从 HTTP 请求输入的数据。如果需要，你可以传递当前的 Request 对象的实例，它将接收所有输入数据，并将其设置为待验证的数据：

```
$validation->withRequest($this->request)
           ->run();
```

### 处理 Validation

### 验证数组的键

如果需要验证的数据在嵌套的关联数组中，则可以使用“点数组语法”轻松验证数据：

```
// The data to test:
'contacts' => [
    'name' => 'Joe Smith',
    'friends' => [
        [
            'name' => 'Fred Flinstone'
        ],
        [
            'name' => 'Wilma'
        ]
    ]
]

// Joe Smith
$validation->setRules([
    'contacts.name' => 'required'
]);

// Fred Flintsone & Wilma
$validation->setRules([
    'contacts.friends.name' => 'required'
]);
```

你可以使用通配符“*”来匹配数组的任何一个层级：

```
// Fred Flintsone & Wilma
$validation->setRules([
    'contacts.*.name' => 'required'
]);
```

“点数组语法”也通常用于一维数组。例如，多选下拉列表返回的数据：

```
// The data to test:
'user_ids' => [
    1,
    2,
    3
]
// Rule
$validation->setRules([
    'user_ids.*' => 'required'
]);
```

### 验证单个值

根据规则验证单个值：

```
$validation->check($value, 'required');
```

### 将验证规则集合保存到配置文件

Validation 类一个好的功能是，它允许你将整个程序的验证规则存储在配置文件中。将规则组合成一个“group”，可以在运行验证时指定不同的组。

### 如何保存你的规则

要存储你的验证规则，只需在 Config\Validation 类中使用 group 名创建一个新的公共属性，该元素将包含你的验证规则数组。验证规则数组的原型如下所示：

```
class Validation
{
    public $signup = [
        'username'     => 'required',
        'password'     => 'required',
        'pass_confirm' => 'required|matches[password]',
        'email'        => 'required|valid_email'
    ];
}
```

你可以在调用 run() 方法时指定要使用的组：

```
$validation->run($data, 'signup');
```

你也可以将自定义错误消息存储在配置文件中，属性名称与组名相同并添加 _errors。当使用该组时，默认的错误消息将被替换：

```
class Validation
{
    public $signup = [
        'username'     => 'required',
        'password'     => 'required',
        'pass_confirm' => 'required|matches[password]',
        'email'        => 'required|valid_email'
    ];

    public $signup_errors = [
        'username' => [
            'required'    => 'You must choose a username.',
```

（下页继续）

(续上页)

```
        ],
        'email'    => [
            'valid_email' => 'Please check the Email field. It does not␣
↪appear to be valid.'
        ]
    ];
}
```

或者在组中传递所有的设置：

```
class Validation
{
    public $signup = [
        'username' => [
            'rules'  => 'required',
            'errors' => [
                'required' => 'You must choose a Username.'
            ]
        ],
        'email'    => [
            'rules'  => 'required|valid_email',
            'errors' => [
                'valid_email' => 'Please check the Email field. It does␣
↪not appear to be valid.'
            ]
        ],
    ];
}
```

有关数组格式的详细信息请查看下文。

### 获取与设置规则组

#### 获取规则组

该方法从验证配置中获取规则组：

```
$validation->getRuleGroup('signup');
```

#### 设置规则组

该方法设置将规则组从验证配置设置到验证服务：

```
$validation->setRuleGroup('signup');
```

### 运行多个 Validation

**注解：** run() 方法不会重置错误状态。如果上次运行失败，run() 方法将始终返回 false，getErrors() 方法将始终返回上次的所有错误，直至状态被显式重置。

如果需要运行多个验证，例如在不同的数据集上运行或者一个接一个的运行不同的规则，你应该在每次运行前调用 $validation->reset() 清除上次运行产生的错误。需要注意的是 reset() 将重置之前的所有数据、规则或是自定义错误消息。所以需要重复 setRules()，setRuleGroup() 等方法：

```
for ($userAccounts as $user) {
    $validation->reset();
    $validation->setRules($userAccountRules);
    if (!$validation->run($user)) {
        // handle validation errors
    }
}
```

### Validation 占位符

Validation 类提供了一个简单的方法，可以根据传入的数据替换部分规则。这听起来十分晦涩，但在使用 is_unique 进行验证时十分方便。占位符是字段的名称（或数组的键），该字段名称（或数组的键）将用花括号包起来作为 $data 传入。它将被替换为匹配的传入字段的 **值**。以下例子可以解释这些：

```
$validation->setRules([
    'email' => 'required|valid_email|is_unique[users.email,id,{id}]'
]);
```

在这组规则中，它声明 email 在数据库中是唯一的，除了具有与占位符匹配的 id 信息，假设表单 POST 数据中有以下内容：

```
$_POST = [
    'id' => 4,
    'email' => 'foo@example.com'
];
```

那么占位符 {id} 将被修改为数字 4，以下是修改后的规则：

```
$validation->setRules([
    'email' => 'required|valid_email|is_unique[users.email,id,4]'
]);
```

因此，在验证 email 唯一时，将忽略数据库中 id=4 的行。

这也可以用于在运行时动态创建更多的规则，只要你确保传入的任何动态键都不会与表单数据产生冲突。

### 处理错误

Validation 库提供了几种方法帮助你设置错误消息，提供自定义错误消息，以及显示一个或多个错误消息。

默认情况下，错误消息来自 system/Language/en/Validation.php 中的语言字符串，其中每个规则都有一个条目。

### 设置自定义错误消息

setRule() 和 setRules() 允许自定义错误消息数据作为最后一个参数传入。每一个错误的错误消息都是定制的，这将带来愉快的用户体验。如果没有设置自定义错误消息，则提供默认值。

这是两种设置错误消息的方式。

作为最后一个参数：

```
$validation->setRules([
        'username' => 'required|is_unique[users.username]',
        'password' => 'required|min_length[10]'
    ],
    [   // Errors
        'username' => [
            'required' => 'All accounts must have usernames provided',
        ],
        'password' => [
            'min_length' => 'Your password is too short. You want to get␣
↪hacked?'
        ]
    ]
);
```

或者作为标签样式：

```
$validation->setRules([
        'username' => [
            'label'  => 'Username',
            'rules'  => 'required|is_unique[users.username]',
            'errors' => [
                'required' => 'All accounts must have {field} provided'
            ]
        ],
        'password' => [
```

(下页继续)

(续上页)

```
            'label'  => 'Password',
            'rules'  => 'required|min_length[10]',
            'errors' => [
                'min_length' => 'Your {field} is too short. You want to␣
↪get hacked?'
            ]
        ]
    ]
);
```

如果你希望包含字段的“human”名称，或者某些规则允许的可选参数（比如 max_length），或当前参与验证的值，则可以分别将 {field}，{param}，{value} 标记添加到你的消息中：

```
'min_length' => 'Supplied value ({value}) for {field} must have at least
↪{param} characters.'
```

在一个用户名字段为 Username ，验证规则为 min_length[6] ，字段值为“Pizza”的验证中，将显示错误消息“Supplied value (Pizza) for Username must have at least 6 characters”

**注解：** 如果你传递最后一个参数，则标签样式的错误信息将被忽略。

### 消息和验证标签的翻译

要使用语言文件中的翻译字符串，可以简单的使用点语法。假设我们有一个包含翻译的文件位于 app/Languages/en/Rules.php。我们可以简单的使用定义在文件中的语言行，如下：

```
$validation->setRules([
        'username' => [
            'label'  => 'Rules.username',
            'rules'  => 'required|is_unique[users.username]',
            'errors' => [
                'required' => 'Rules.username.required'
            ]
        ],
        'password' => [
            'label'  => 'Rules.password',
            'rules'  => 'required|min_length[10]',
            'errors' => [
                'min_length' => 'Rules.password.min_length'
            ]
        ]
```

(下页继续)

(续上页)

```
    ]
);
```

### 获取所有错误

如果你需要检索所有验证失败字段的错误消息，你可以使用 getErrors() 方法：

```
$errors = $validation->getErrors();

// Returns:
[
    'field1' => 'error message',
    'field2' => 'error message',
]
```

如果没有错误，则返回空数组。

### 获取单个错误

你可以使用 getError() 方法检索单个字段的错误消息。参数名是唯一的参数：

```
$error = $validation->getError('username');
```

如果没有错误，则返回空字符串。

### 检查是否存在错误

你可以使用 hasError() 方法检查字段是否存在错误。字段名是唯一的参数：

```
if ($validation->hasError('username'))
{
    echo $validation->getError('username');
}
```

### 自定义错误显示

当你调用 $validation->listErrors() 或 $validation->showError() ，它将在后台加载一个视图文件，该文件确定错误的显示方法。默认情况下，它在经过包装的 div 上显示 errors 。你可以轻松的创建视图并在整个程序中使用它。

### 创建视图

第一步是创建视图文件，它可以放在 view() 方法可以加载的任何地方。这意味着标准的 View 目录，或者任何命名空间下的 View 目录都可以正常工作。例如，可以在 **/app/Views/_errors_list.php** 创建新的视图文件：

```
<div class="alert alert-danger" role="alert">
    <ul>
    <?php foreach ($errors as $error) : ?>
        <li><?= esc($error) ?></li>
    <?php endforeach ?>
    </ul>
</div>
```

$errors 数组可以在包含错误列表的视图中使用，其中键是发生错误的字段名，值是错误消息，如下所示：

```
$errors = [
    'username' => 'The username field must be unique.',
    'email'    => 'You must provide a valid email address.'
];
```

实际上可以创建两种类型的视图文件。第一种包含所有错误消息，这就是我们刚才看到的。另一种更简单，只包含一个错误消息变量 $error。它与指定字段名的 showError() 方法一起使用。

```
<span class="help-block"><?= esc($error) ?></span>
```

### 配置

创建视图后，需要让 Validation 库知道它们。打开 Config/Validation.php，在里面找到 $templates 属性。你可以在其中列出任意多个自定义视图，并提供一个可以引用他们的短别名。我们将添加上边的示例文件，它看起来像：

```
public $templates = [
    'list'    => 'CodeIgniter\Validation\Views\list',
    'single'  => 'CodeIgniter\Validation\Views\single',
    'my_list' => '_errors_list'
];
```

### 指定模板

通过将别名作为 listErrors 方法的第一个参数，来指定要使用的模板：

```
<?= $validation->listErrors('my_list') ?>
```

当显示特定字段错误时，你可以将别名作为第二个参数传递给 showError 方法，别名参数应该在字段名称之后：

```
<?= $validation->showError('username', 'my_single') ?>
```

### 创建自定义规则

规则简单的存储在命名空间类中。只要自动加载器能找到它们，你可将他们存储到任何位置。这些文件称作规则集。要添加新的规则集，请编辑 **Config/Validation.php** 并将新文件添加到 $ruleSets 数组：

```
public $ruleSets = [
    \CodeIgniter\Validation\Rules::class,
    \CodeIgniter\Validation\FileRules::class,
    \CodeIgniter\Validation\CreditCardRules::class,
];
```

你可以将其添加为具有完全限定类的简单字符串，或者使用 ::class 后缀进行添加。如上所示，这里的好处是，它在更高级的 IED 中提供了额外的一些导航功能。

在文件中，每一个方法都是一个规则，它必须接受字符串作为第一个字符串，并且必须返回布尔值 true 或 false 。如果通过测试则返回 true ，否则返回 false 。

```
class MyRules
{
    public function even(string $str): bool
    {
        return (int)$str % 2 == 0;
    }
}
```

默认情况下，系统将在 CodeIgniter\Language\en\Validation.php 中查找错误要使用语言字符串。在自定义规则中，你可以通过第二个参数 $error 的引用来提供错误消息：

```
public function even(string $str, string &$error = null): bool
{
    if ((int)$str % 2 != 0)
    {
        $error = lang('myerrors.evenError');
        return false;
    }

    return true;
}
```

现在你可像其他规则一样使用新的自定义规则：

```
$this->validate($request, [
    'foo' => 'required|even'
]);
```

### 允许参数

如果你的方法需要使用参数，则该函数至少需要三个参数：要验证的字符串、参数字符串以及包含提交表单所有数据的数组。$data 数组对于像 require_with 这样需要检查另一个提交字段的值作为其结果基础的规则来说十分方便：

```
public function required_with($str, string $fields, array $data): bool
{
    $fields = explode(',', $fields);

    // If the field is present we can safely assume that
    // the field is here, no matter whether the corresponding
    // search field is present or not.
    $present = $this->required($str ?? '');

    if ($present)
    {
        return true;
    }

    // Still here? Then we fail this test if
    // any of the fields are present in $data
    // as $fields is the lis
    $requiredFields = [];

    foreach ($fields as $field)
    {
        if (array_key_exists($field, $data))
        {
            $requiredFields[] = $field;
        }
    }

    // Remove any keys with empty values since, that means they
    // weren't truly there, as far as this is concerned.
    $requiredFields = array_filter($requiredFields, function ($item) use␣
↪($data) {
        return ! empty($data[$item]);
    });

    return empty($requiredFields);
}
```

自定义错误可以通过第四个参数传递，如上所述。

### 可用规则

以下是可供使用的所有本地规则的列表：

**注解:** 规则是一个字符串；参数之间 **不能有空格**，尤其是 `is_unique` 规则。`ignore_value` 前后不能有空格。

```
// is_unique[table.field,ignore_field,ignore_value]

$validation->setRules([
    'name' => "is_unique[supplier.name,uuid, $uuid]",  // is not ok
    'name' => "is_unique[supplier.name,uuid,$uuid ]",  // is not ok
    'name' => "is_unique[supplier.name,uuid,$uuid]",   // is ok
    'name' => "is_unique[supplier.name,uuid,{uuid}]",  // is ok - see
↪"Validation Placeholders"
]);
```

| Rule | Parameter | Description |
|---|---|---|
| alpha | No | Fails if field has anything other than alphabetic characters. |
| alpha_space | No | Fails if field contains anything other than alphabetic character |
| alpha_dash | No | Fails if field contains anything other than alphanumeric charac |
| alpha_numeric | No | Fails if field contains anything other than alphanumeric charac |
| alpha_numeric_space | No | Fails if field contains anything other than alphanumeric or spa |
| alpha_numeric_punct | No | Fails if field contains anything other than alphanumeric, space |
| decimal | No | Fails if field contains anything other than a decimal number. A |
| differs | Yes | Fails if field does not differ from the one in the parameter. |
| exact_length | Yes | Fails if field is not exactly the parameter value. One or more c |
| greater_than | Yes | Fails if field is less than or equal to the parameter value or not |
| greater_than_equal_to | Yes | Fails if field is less than the parameter value, or not numeric. |
| hex | No | Fails if field contains anything other than hexadecimal charact |
| if_exist | No | If this rule is present, validation will only return possible error |
| in_list | Yes | Fails if field is not within a predetermined list. |
| integer | No | Fails if field contains anything other than an integer. |
| is_natural | No | Fails if field contains anything other than a natural number: 0 |
| is_natural_no_zero | No | Fails if field contains anything other than a natural number, ex |
| is_not_unique | Yes | Checks the database to see if the given value exist. Can ignore |
| is_unique | Yes | Checks if this field value exists in the database. Optionally set |
| less_than | Yes | Fails if field is greater than or equal to the parameter value or |
| less_than_equal_to | Yes | Fails if field is greater than the parameter value or not numeri |
| matches | Yes | The value must match the value of the field in the parameter. |
| max_length | Yes | Fails if field is longer than the parameter value. |
| min_length | Yes | Fails if field is shorter than the parameter value. |

| Rule | Parameter | Description |
|---|---|---|
| numeric | No | Fails if field contains anything other than numeric characters. |
| regex_match | Yes | Fails if field does not match the regular expression. |
| permit_empty | No | Allows the field to receive an empty array, empty string, null c |
| required | No | Fails if the field is an empty array, empty string, null or false. |
| required_with | Yes | The field is required when any of the other required fields are |
| required_without | Yes | The field is required when all of the other fields are present in |
| string | No | A generic alternative to the alpha* rules that confirms the ele |
| timezone | No | Fails if field does match a timezone per `timezone_identifier` |
| valid_base64 | No | Fails if field contains anything other than valid Base64 charact |
| valid_json | No | Fails if field does not contain a valid JSON string. |
| valid_email | No | Fails if field does not contain a valid email address. |
| valid_emails | No | Fails if any value provided in a comma separated list is not a |
| valid_ip | No | Fails if the supplied IP is not valid. Accepts an optional paran |
| valid_url | No | Fails if field does not contain a valid URL. |
| valid_date | No | Fails if field does not contain a valid date. Accepts an optiona |
| valid_cc_number | Yes | Verifies that the credit card number matches the format used |

### 文件上传规则

这些验证规则可以让你进行基本的检查，验证上传的文件是否满足你的业务需求。由于文件上传字段在 HTML 字段中不存在，并且存储在 $_FILES 全局变量中，因此字段名需要输入两次，第一个用于指定验证的字段，像其他规则一样，第二次作为所有文件上传规则的第一个参数：

```
// In the HTML
<input type="file" name="avatar">

// In the controller
$this->validate([
    'avatar' => 'uploaded[avatar]|max_size[avatar,1024]'
]);
```

| Rule | Parameter | Description | Example |
|---|---|---|---|
| uploaded | Yes | Fails if the name of the parameter does not match the name of any uploaded files. | uploaded[field_name] |
| max_size | Yes | Fails if the uploaded file named in the parameter is larger than the second parameter in kilobytes (kb). | max_size[field_name,2048 |
| max_dims | Yes | Fails if the maximum width and height of an uploaded image exceed values. The first parameter is the field name. The second is the width, and the third is the height. Will also fail if the file cannot be determined to be an image. | max_dims[field_name,300 |
| mime_in | Yes | Fails if the file' s mime type is not one listed in the parameters. | mime_in[field_name,image |
| ext_in | Yes | Fails if the file' s extension is not one listed in the parameters. | ext_in[field_name,png,jpg |
| is_image | Yes | Fails if the file cannot be determined to be an image based on the mime type. | is_image[field_name] |

文件验证规则适用于单个和多个文件上传。

**注解：** 你也可以使用任何最多允许两个参数的本地 PHP 函数，其中至少需要一个参数（传递字段数据）。

## 7.2 辅助函数

辅助函数是一些程序功能的集合。

### 7.2.1 Array Helper

The array helper provides several functions to simplify more complex usages of arrays. It is not intended to duplicate any of the existing functionality that PHP provides - unless it is to vastly simplify their usage.



#### Loading this Helper

This helper is loaded using the following code:

```
helper('array');
```

### Available Functions

The following functions are available:

**dot_array_search**(*string $search*, *array $values*)

**参数**

- **$search** (*string*) – The dot-notation string describing how to search the array
- **$values** (*array*) – The array to search

**返回** The value found within the array, or null

**返回类型** mixed

This method allows you to use dot-notation to search through an array for a specific-key, and allows the use of a the '*' wildcard. Given the following array:

```
$data = [
    'foo' => [
        'buzz' => [
            'fizz' => 11
        ],
        'bar' => [
            'baz' => 23
        ]
    ]
]
```

We can locate the value of 'fizz' by using the search string "foo.buzz.fizz". Likewise, the value of baz can be found with "foo.bar.baz":

```
// Returns: 11
$fizz = dot_array_search('foo.buzz.fizz', $data);

// Returns: 23
$baz = dot_array_search('foo.bar.baz', $data);
```

You can use the asterisk as a wildcard to replace any of the segments. When found, it will search through all of the child nodes until it finds it. This is handy if you don't know the values, or if your values have a numeric index:

```
// Returns: 23
$baz = dot_array_search('foo.*.baz', $data);
```

### 7.2.2 Cookie 辅助函数

Cookie 辅助函数文件包含一些协助 Cookie 运行的函数。



#### 加载 Cookie 辅助函数

Cookie 辅助函数文件使用下面的代码加载:

```
helper('cookie');
```

#### 函数参考

该辅助函数有下列可用函数:

**set_cookie**(*$name*[, *$value* = ''[, *$expire* = ''[, *$domain* = ''[, *$path* = '/'[, *$prefix* = ''[, *$secure* = *false*[, *$httpOnly* = *false*]]]]]]])

**参数**

- **$name** (*mixed*) – Cookie 名称 或对这函数所有通用参数的关联数组
- **$value** (*string*) – Cookie 值
- **$expire** (*int*) – 直到截止时的秒数
- **$domain** (*string*) – Cookie 域名 (通常是: .yourdomain.com)
- **$path** (*string*) – Cookie 路径
- **$prefix** (*string*) – Cookie 名称前缀
- **$secure** (*bool*) – 是否仅仅通过 HTTPS 发送 Cookie
- **$httpOnly** (*bool*) – 是否从 JavaScript 中隐藏 Cookie

**返回类型** void

辅助函数给你更友好的语法去 设置浏览器的 Cookies. 辅助函数使用的说明参考响应库 , 同时对 `Response::setCookie()` 来说 Cookie 辅助函数是别称.

**get_cookie**(*$index*[, *$xssClean* = *false*])

**参数**

- **$index** (*string*) – Cookie 名称
- **$xss_clean** (*bool*) – 返回值是否应用在 XSS 过滤中

**返回** 返回 Cookie 值而如果没有则为空

**返回类型** mixed

辅助函数给你更友好的语法去 获取浏览器的 Cookies. 辅助函数详细的使用说明参考传入请求库 同时辅助函数的作用非常近似于 `IncomingRequest::getCookie()`, 你也许已经设置在你的 *application/Config/App.php* 文件里除了它也预置了 `$cookiePrefix` .

`delete_cookie`(*$name*[, *$domain* = ''[, *$path* = '/'[, *$prefix* = '']]])

**参数**

- **$name** (*string*) – Cookie 名称
- **$domain** (*string*) – Cookie 域名 (通常是: .yourdomain.com)
- **$path** (*string*) – Cookie 路径
- **$prefix** (*string*) – Cookie 名称前缀

**返回类型** void

该函数让你删除一个 Cookie. 除非你已经设置了一个定制路径或者其他值，仅仅 Cookie 的名字是必须的。

```
delete_cookie('name');
```

这个函数除了没有值和截止参数，它对 `set_cookie()` 来说在其他方面是恒等的。你能在第一参数里确定数组值或者你要设置离散参数。

```
delete_cookie($name, $domain, $path, $prefix);
```

### 7.2.3 Date Helper

The Date Helper file contains functions that assist in working with dates.



#### Loading this Helper

This helper is loaded using the following code:

```
helper('date');
```

### Available Functions

The following functions are available:

**now**([*$timezone = NULL*])

**参数**

- **$timezone** (*string*) – Timezone

**返回** UNIX timestamp

**返回类型** int

Returns the current time as a UNIX timestamp, referenced either to your server's local time or any PHP supported timezone, based on the "time reference" setting in your config file. If you do not intend to set your master time reference to any other PHP supported timezone (which you'll typically do if you run a site that lets each user set their own timezone settings) there is no benefit to using this function over PHP's `time()` function.

```
echo now('Australia/Victoria');
```

If a timezone is not provided, it will return `time()` based on the **time_reference** setting.

**timezone_select**([*$class = ''*, *$default = ''*, *$what = DateTimeZone::ALL*, *$country = null*])

**参数**

- **$class** (*string*) – Optional class to apply to the select field
- **$default** (*string*) – Default value for initial selection
- **$what** (*int*) – DateTimeZone class constants (see listIdentifiers)
- **$country** (*string*) – A two-letter ISO 3166-1 compatible country code (see listIdentifiers)

**返回** Preformatted HTML select field

**返回类型** string

Generates a *select* form field of available timezones (optionally filtered by *$what* and *$country*). You can supply an option class to apply to the field to make formatting easier, as well as a default selected value.

```
echo timezone_select('custom-select', 'America/New_York');
```

Many functions previously found in the CodeIgniter 3 `date_helper` have been moved to the `I18n` module in CodeIgniter 4.

### 7.2.4 文件系统辅助函数

目录辅助函数文件包含的函数协助目录运行。



#### 加载文件系统辅助函数

文件系统辅助函数使用下面的代码加载:

```
helper('filesystem');
```

#### 通用函数

接下来的函数是通用的:

**directory_map**(*$source_dir*[, *$directory_depth = 0*[, *$hidden = FALSE*]])

**参数**

- **$source_dir** (*string*) – 资源目录路径
- **$directory_depth** (*int*) – 遍历目录量度 (0 = 完全递归, 1 = 最近目录, 等等)
- **$hidden** (*bool*) – 是否包含隐藏目录

**返回** 文件数组

**返回类型** array

例如:

```
$map = directory_map('./mydirectory/');
```

---

**注解:** 路径几乎常常与你的主要 index.php 文件有关系。

子文件夹包含的目录还会被映射。如果你希望控制递归量度，你会使用秒参数（整型）。1 的量度将仅仅映射根层目录:

```
$map = directory_map('./mydirectory/', 1);
```

---

默认情况下，在返回数组里将不会被包含隐藏文件。推翻这个运转状态，你也许要设置第三个参数为真（boolean）：

```
$map = directory_map('./mydirectory/', FALSE, TRUE);
```

每一个文件名将是数组索引，它包含的文件将会被用数值编入索引。下面是一个典型数组:

```
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
                )
```

如果没有找到结果，将会返回空数组。

**write_file**(*$path*, *$data*[, *$mode = 'wb'*])

**参数**

- **$path** (*string*) – File 路径
- **$data** (*string*) – 数据写入 file
- **$mode** (*string*) – fopen() 模式

**返回** 如果写入成功为 TRUE , 万一错误是 FALSE

**返回类型** bool

将数据写入指定路径中的文件。如果文件不存在，这个函数将创建文件。

例如:

```
$data = 'Some file data';
if ( ! write_file('./path/to/file.php', $data))
{
        echo 'Unable to write the file';
}
else
{
        echo 'File written!';
}
```

你能随意地通过第三个参数设置写模式:

```
       write_file('./path/to/file.php', $data, 'r+');

默认模式是'wb'. 模式选项请查看 `PHP 用户指导 <http://php.net/manual/
↪en/function.fopen.php>`_ .
```

**注解:** 这个函数向文件里写入数据要按顺序，它的权限必须被设置成可写的。如果文件已经不存在，那么目录下的文件必须是可写的。

**注解:** 路径关联你的主站的 index.php 文件，不是你的 controller 或者 view 文件。CodeIgniter 用前端 controller 因此路径常常关联主站的 index.

**注解:** 当写入文件时函数捕获了文件上独占的锁定。

**delete_files**(*$path*[, *$del_dir = FALSE*[, *$htdocs = FALSE*]])

**参数**

- **$path** (*string*) – 目录路径
- **$del_dir** (*bool*) – 是否也删除目录
- **$htdocs** (*bool*) – 是否跳过删除.htaccess 和 index page 文件

**返回** 万一为 FALSE，TRUE 为真

**返回类型** bool

删除所有包含在备用路径里的文件。

例如:

```
delete_files('./path/to/directory/');
```

如果第二个参数设置为 TRUE，包含备用根路径的任何目录将也会被删除。

例如:

```
delete_files('./path/to/directory/', TRUE);
```

**注解:** 文件必须是可写的而已经归属至系统的文件原则上已被删除。

**get_filenames**(*$source_dir*[, *$include_path = FALSE*])

**参数**

- **$source_dir** (*string*) – 目录路径
- **$include_path** (*bool*) – 作为文件名的部分是否包含路径

**返回** 文件名数组

**返回类型** array

函数里取服务器路径输入并返回包含所有文件名的数组。设置第二参数为 TRUE 文件路径能很随意的被添加到文件名里。

例如:

```
$controllers = get_filenames(APPPATH.'controllers/');
```

**get_dir_file_info**(*$source_dir*, *$top_level_only*)

**参数**

- **$source_dir** (*string*) – 目录路径
- **$top_level_only** (*bool*) – 是否仅仅查看特殊目录 (不包含子目录)

**返回** 数组涵盖的信息在备用目录的内容中

**返回类型** array

阅读指定的目录并建立包含文件名，文件大小，日期和权限的数组。如果传送第二个参数被阻止成 FALSE 包含指定目录的子文件夹一定是只读的，如同这是个强调操作。

事例:

```
$models_info = get_dir_file_info(APPPATH.'models/');
```

**get_file_info**(*$file*[, *$returned_values = array('name', 'server_path', 'size', 'date')*])

**参数**

- **$file** (*string*) – File 路径
- **$returned_values** (*array*) – 任何返回的信息类型

**返回** 在指定文件上的数组包含的信息或失效的 FALSE

**返回类型** array

约定的文件和路径，文件返回（随意地）the *name*, *path*, *size* and *date modified* 属性信息。第二参数允许你明确地声明任何你想返回的信息。

有效的 `$returned_values` 选项是: *name*, *size*, *date*, *readable*, *writeable*, *executable* 和 *fileperms*.

**symbolic_permissions**(*$perms*)

**参数**

- **$perms** (*int*) – 权限

**返回** 象征权限的 string

**返回类型** string

抓取数值权限（就像是被 `fileperms()` 返回的）并且返回文件权限的标准符号记号。

```
echo symbolic_permissions(fileperms('./index.php'));  // -rw-r--r--
```

**octal_permissions**(*$perms*)

**参数**

- **$perms** (*int*) – 权限

**返回** 八进制权限的 string

**返回类型** string

抓取数值权限（就像是被 `fileperms()` 返回的）并且返回文件权限的一个由三个字母组成的八进制记号。

```
echo octal_permissions(fileperms('./index.php')); // 644
```

**set_realpath**(*$path*[, *$check_existance = FALSE*])

**参数**

- **$path** (*string*) – 路径
- **$check_existance** (*bool*) – 如果路径确实存在是否要去检查

**返回** 绝对路径

**返回类型** string

函数会返回不带符号链接的服务器路径或者有关联的目录结构。如果路径不能决定选项的次一级争议将触发一个错误。

例如:

```
$file = '/etc/php5/apache2/php.ini';
echo set_realpath($file); //  输出 '/etc/php5/apache2/php.ini'

$non_existent_file = '/path/to/non-exist-file.txt';
```

(下页继续)

(续上页)

```
echo set_realpath($non_existent_file, TRUE);    // 显示错误，如同路
径不能决定
echo set_realpath($non_existent_file, FALSE);   // 输出 '/path/to/
↪non-exist-file.txt'

$directory = '/etc/php5';
echo set_realpath($directory);  // 输出 '/etc/php5/'

$non_existent_directory = '/path/to/nowhere';
echo set_realpath($non_existent_directory, TRUE);       // 显示错误，
如同路径不能决定
echo set_realpath($non_existent_directory, FALSE);      // 输出 '/
↪path/to/nowhere'
```

### 7.2.5 表单辅助函数

表单辅助函数包含的函数辅助表单运行.



#### 加载表单辅助函数

表单辅助函数使用下文的代码加载:

```
helper('form');
```

#### 换码（转义）字段值

你也许需要使用 HTML 和字符像在你的表单内部的元素里引用。为了安全地执行，你将需要使用:doc:*common function <../general/common_functions>* `esc()`.

考虑下文的示例:

```
$string = 'Here is a string containing "quoted" text.';

<input type="text" name="myfield" value="<?= $string; ?>" />
```

由于上面字符串包含一套引用，那将导致表单中断。The *esc()* 函数转换 HTML 特殊字节以便它能安全地使用:

```
<input type="text" name="myfield" value="<?= esc($string); ?>" />
```

**注解：** 如果你在页面使用任意表单辅助函数列举，并且你传达像组合的数组一样的值，表单值将会被自动换码，所以不需要调用这个函数。使用它只有你要创建你自己的将要传达作为字符串的表单元素。

### 通用函数

接下来的函数是通用的:

**form_open**([*$action = ''*[, *$attributes = ''*[, *$hidden = array()*]]])

**参数**

- **$action** (*string*) – 表单行为/目标 URI 字符串
- **$attributes** (*mixed*) – HTML 属性，就像数组或者换码字符串
- **$hidden** (*array*) – 隐藏字段的定义的一组数组 An array of hidden fields' definitions

**返回** HTML 表单随时可用的 tag

**返回类型** string

创建一个带着基地址 URL 的随时可用的表单标签 ** 从你的配置优先选择营造 **. 它将随意地让你添加表单属性和隐藏输入字段，并且会常常在你的配置文件里添加基于 charset 值的 *accept-charset* 属性。

宁可使用标签的绝对好处也不要艰苦的编码你自己的 HTML 是由于在事件里你的 URLs 曾改变而标签容许你的网址是更便携的。

下面是一则简单的例子:

```
echo form_open('email/send');
```

上面的例子将创建一个指向你的基地址 URL 和 "email/send" URL 部分的表单，像这样:

```
<form method="post" accept-charset="utf-8" action="http://example.
↪com/index.php/email/send">
```

You can also add {locale} like the following:

```
echo form_open('{locale}/email/send');
```

The above example would create a form that points to your base URL plus the current request locale with "email/send" URI segments, like this:

```
<form method="post" accept-charset="utf-8" action="http://example.
↪com/index.php/en/email/send">
```

**添加属性**

由正传达组合的数组到第二个参数的属性能被加入，像这样:

```
$attributes = ['class' => 'email', 'id' => 'myform'];
echo form_open('email/send', $attributes);
```

二选一地，你能明确的像字符串一样说明第二个参数:

```
echo form_open('email/send', 'class="email" id="myform"');
```

上文的例子将会创建一个同样的表单相似于下文这个事例:

```
<form method="post" accept-charset="utf-8" action="http://
↪example.com/index.php/email/send" class="email" id=
↪"myform">
```

If CSRF filter is turned on *form_open()* will generate CSRF field at the beginning of the form. You can specify ID of this field by passing csrf_id as one of the $attribute array:

form_open( ‘/u/sign-up’ , [ ‘csrf_id’ => ‘my-id’ ]);

will return:

<form action=” /u/sign-up” method=” post” accept-charset=” utf-8” > <input type=” hidden” id=” my-id” name=” csrf_field” value=” 964ede6e0ae8a680f7b8eab69136717d” />

**添加隐藏输入字段**

由正传达组合的数组到第三个参数的隐藏字段能被添加，像这样:

```
$hidden = ['username' => 'Joe', 'member_id' => '234'];
echo form_open('email/send', '', $hidden);
```

由正传达的任何 false 值到隐藏字段，你能忽略第二个参数.

上面的事例将创建类似于下面的句子:

```
<form method="post" accept-charset="utf-8" action="http://
↪example.com/index.php/email/send">
        <input type="hidden" name="username" value="Joe" />
        <input type="hidden" name="member_id" value="234" /
↪>
```

**form_open_multipart**([*$action = ”*[, *$attributes = ”*[, *$hidden = array()*]]])

**参数**

- **$action** (*string*) – 表单行为/目标 URI 字符串
- **$attributes** (*mixed*) – HTML 属性，就像数组或者换码字符串
- **$hidden** (*array*) – 隐藏字段的定义的一组数组

**返回** HTML 多部件的表单随时可用的 tag

**返回类型** string

这个函数对上文的*form_open()* 来说是类似的，除了它附加了一个 *multipart* 属性，如果你喜欢使用表单上传文件这个属性是必须的。

**form_hidden**(*$name*[, *$value* = ''])

**参数**

- **$name** (*string*) – 字段名
- **$value** (*string*) – 字段值

**返回** HTML 隐藏输入字段 tag

**返回类型** string

让你生成隐藏输入字段。你也能提交名称/值字符串去创建一个字段:

```
form_hidden('username', 'johndoe');
// 将产生: <input type="hidden" name="username" value="johndoe" />
```

…或者你能提交组合数组去创建复合字段:

```
$data = [
        'name'  => 'John Doe',
        'email' => 'john@example.com',
        'url'   => 'http://example.com'
];

echo form_hidden($data);

/*
        将产生:
        <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"␣
↪/>
*/
```

你也能传达组合的数组给字段值:

```
$data = [
        'name'  => 'John Doe',
```

(下页继续)

(续上页)

```
        'email' => 'john@example.com',
        'url'   => 'http://example.com'
];

echo form_hidden('my_array', $data);

/*
        将产生:

        <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" />
*/
```

倘若你想创建额外属性的隐藏输入字段:

```
$data = [
        'type'  => 'hidden',
        'name'  => 'email',
        'id'    => 'hiddenemail',
        'value' => 'john@example.com',
        'class' => 'hiddenemail'
];

echo form_input($data);

/*
        将产生:

        <input type="hidden" name="email" value="john@example.com"␣
↪id="hiddenemail" class="hiddenemail" />
*/
```

**form_input**([*$data* = ''[, *$value* = ''[, *$extra* = ''[, *$type* = 'text']]]])

**参数**

- **$data** (*array*) – 字段属性数据
- **$value** (*string*) – 字段值
- **$extra** (*mixed*) – 额外属性被添加到 tag 任何一方像数组或者文字字符串
- **$type** (*string*) – 输入字段类型。例如: 'text', 'email', 'number', 等等.

**返回** HTML 文本输入字段 tag

**返回类型** string

让你生成标准的文本输入字段。你能最低程度地在第一和第二参数里传达字段名和值:

```
echo form_input('username', 'johndoe');
```

或者你能传达包含你希望你的表单要包含的任何数据的组合的数组:

```
$data = [
        'name'      => 'username',
        'id'        => 'username',
        'value'     => 'johndoe',
        'maxlength' => '100',
        'size'      => '50',
        'style'     => 'width:50%'
];

echo form_input($data);

/*
        将产生:

        <input type="text" name="username" value="johndoe" id=
↪"username" maxlength="100" size="50" style="width:50%"  />
*/
```

如果你想要你的表单包含一些额外的数据，像 JavaScript ，你能在第三参数里像字符串一样传达参数:

```
$js = 'onClick="some_function()"';
echo form_input('username', 'johndoe', $js);
```

或者你能像数组一样传达参数:

```
        $js = ['onClick' => 'some_function();'];
        echo form_input('username', 'johndoe', $js);

支持 HTML5 输入字段扩充范围，你能像第四个参数一样传达一个输入键入信息::

        echo form_input('email', 'joe@example.com', ['placeholder' =>
↪ 'Email Address...'], 'email');

        /*
                将产生:
```

(下页继续)

(续上页)

```
                <input type="email" name="email" value="joe@example.
↪com" placeholder="Email Address..." />
        */
```

**form_password**([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

**参数**

- **$data** (*array*) – 字段属性数据
- **$value** (*string*) – 字段值
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** HTML 密码输入字段 tag

**返回类型** string

此函数除了函数使用的“password”输入类型在完全关系到上文所述的*form_input()* 函数是完全相似的。

**form_upload**([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

:param array $data: 字段属性数据:param string $value: 字段值:param mixed $extra: 额外的属性被添加到 tag 任何一方像数组或者文字的字符串:returns: HTML 文件上传输入字段 tag :rtype: string

此函数除了使用“file”输入类型在完全关系到上文所述的*form_input()* 函数是完全相似的，接受函数适用于上传文件。

**form_textarea**([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

**参数**

- **$data** (*array*) – 字段属性数据
- **$value** (*string*) – 字段值
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** HTML 文本区域 tag

**返回类型** string

此函数除了产生“textarea”类型外在完全关系到上文所述的*form_input()* 函数是完全相似的。

---

**注解:** 上文的例子里代替 *maxlength* 和 *size* 属性，你会更换具体指定的 *rows* 和 *cols* 。

---

**form_dropdown**([*$name* = ''[, *$options* = *array()*[, *$selected* = *array()*[, *$extra* = '']]]])

**参数**

- **$name** (*string*) – 字段名
- **$options** (*array*) – 选项的组合的数组被列举
- **$selected** (*array*) – 字段的列表要标明 *selected* 属性
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** HTML 下拉菜单选择字段 tag

**返回类型** string

让你创建一个下拉菜单字段。第一个参数会包含字段名，第二个参数会包含一个组合的数组选项，而第三参数会包含你希望被选择的值。你也能通过第三参数传达一个符合选项数组，并且辅助函数会为你创建一个复合选项。

例如:

```
        $options = [
                'small'  => 'Small Shirt',
                'med'    => 'Medium Shirt',
                'large'  => 'Large Shirt',
                'xlarge' => 'Extra Large Shirt',
        ];

        $shirts_on_sale = ['small', 'large'];
        echo form_dropdown('shirts', $options, 'large');

        /*
                将产生:

                <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);

        /*
                将产生:

                <select name="shirts" multiple="multiple">
                        <option value="small" selected="selected">
↪Small Shirt</option>
```

(下页继续)

(续上页)

```
                        <option value="med">Medium  Shirt</option>
                        <option value="large" selected="selected">
↪Large Shirt</option>
                        <option value="xlarge">Extra Large Shirt</
↪option>
                </select>
        */

如果你想要开始部分的 <select> 包含额外的数据，像 id 属性或者␣
↪JavaScript ，你能在第四个参数里像字符串一样传达它::

        $js = 'id="shirts" onChange="some_function();"';
        echo form_dropdown('shirts', $options, 'large', $js);
```

或者你能像传达数组一样传达参数:

```
$js = [
        'id'       => 'shirts',
        'onChange' => 'some_function();'
];
echo form_dropdown('shirts', $options, 'large', $js);
```

如果数组被传达象 $options 一样是一个多维数组，那么 form_dropdown() 将会产生一个像 label 一样带着数组键码的 <optgroup> 。

**form_multiselect**([*$name = ''*[, *$options = array()*[, *$selected = array()*[, *$extra = ''*]]]])

**参数**

- **$name** (*string*) – 字段名
- **$options** (*array*) – 选项的组合数组被列举
- **$selected** (*array*) – 字段的列表要标明 *selected* 属性
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** HTML 下拉菜单混合选项字段 tag

**返回类型** string

让你创建一个标准的混合字段。第一个参数将包含字段名，第二个参数会包含选项的一个组合的数组，而第三个参数会包含值或者你想要被选择的值。

参数用法是完全相似于上文去使用的*form_dropdown()* ，除了当然地字段名将需要去用 POST 数组语法，例如：foo[].

**form_fieldset**([*$legend_text = ''*[, *$attributes = array()*]])

**参数**

- **$legend_text** (*string*) – Text 放进 <legend> tag
- **$attributes** (*array*) – 属性被置位在 <fieldset> tag 上

**返回** HTML 字段置位开始 tag

**返回类型** string

让你生成 fieldset/legend 字段。

例如:

```
echo form_fieldset('Address Information');
echo "<p>fieldset content here</p>\n";
echo form_fieldset_close();

/*
        生成:

                <fieldset>
                        <legend>Address Information</legend>
                                <p>form content here</p>
                </fieldset>
*/
```

相似于其他函数，如果你更喜欢设置额外属性你能在第二参数里提交一个组合的数组:

```
$attributes = [
        'id'    => 'address_info',
        'class' => 'address_info'
];

echo form_fieldset('Address Information', $attributes);
echo "<p>fieldset content here</p>\n";
echo form_fieldset_close();

/*
        生成:

        <fieldset id="address_info" class="address_info">
                <legend>Address Information</legend>
                <p>form content here</p>
        </fieldset>
*/
```

`form_fieldset_close`([*$extra = ''*])

**参数**

- **$extra** (*string*) – 闭合 tag 附加的任何字段, *as is*

**返回** HTML 字段置位关闭 tag

**返回类型**

string

产生一个正关闭的 </fieldset> tag. 使用这个函数仅有的优势是它允许你传达数据给将被添加的下文关联的 tag 。例如

```
$string = '</div></div>';
echo form_fieldset_close($string);
// 将生成: </fieldset></div></div>
```

**form_checkbox**([*$data = ''*[, *$value = ''*[, *$checked = FALSE*[, *$extra = ''*]]]])

**参数**

- **$data** (*array*) – 字段属性数据
- **$value** (*string*) – 字段值
- **$checked** (*bool*) – 是否去标明 checkbox 在 *checked* 状态
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** HTML checkbox 输入 tag

**返回类型** string

让你产生一个 checkbox 字段. 简单的例子:

```
echo form_checkbox('newsletter', 'accept', TRUE);
// 将生成:  <input type="checkbox" name="newsletter" value="accept
↪" checked="checked" />
```

第三个参数包含一个布尔值 TRUE/FALSE 去决定是否 box 应该被记号或者未记号。

在这个辅助函数里类似的对于其他的表单函数来说，你也能传达属性的数组给函数:

```
$data = [
        'name'    => 'newsletter',
        'id'      => 'newsletter',
        'value'   => 'accept',
        'checked' => TRUE,
        'style'   => 'margin:10px'
];

echo form_checkbox($data);
// 将生成: <input type="checkbox" name="newsletter" id="newsletter
↪" value="accept" checked="checked" style="margin:10px" />
```

也跟其他函数一样，如果你想要 tag 去包含像 JavaScript 的额外数据，你能在第四个参数里像传达字符串一样传达它:

```
$js = 'onClick="some_function()"';
echo form_checkbox('newsletter', 'accept', TRUE, $js);
```

或者你能像数组一样传达它:

```
$js = ['onClick' => 'some_function();'];
echo form_checkbox('newsletter', 'accept', TRUE, $js);
```

**form_radio**([*$data = ''*[, *$value = ''*[, *$checked = FALSE*[, *$extra = ''*]]]])

**参数**

- **$data** (*array*) – 字符串属性数据
- **$value** (*string*) – 字符串值
- **$checked** (*bool*) – 是否标明 radio 按钮是 *checked* 状态
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** HTML radio 输入 tag

**返回类型** string

除了函数使用“radio”输入类型此函数在完全关系到上文所述的*form_checkbox()*函数是完全类似的。

**form_label**([*$label_text = ''*[, *$id = ''*[, *$attributes = array()*]]])

**参数**

- **$label_text** (*string*) – Text 提交 <label> tag
- **$id** (*string*) – 我们正在制作的一个 label 表单元素的 ID
- **$attributes** (*string*) – HTML 属性

**返回** HTML 字段 label tag

**返回类型** string

让你产生一个 <label>. 简单事例:

```
echo form_label('What is your Name', 'username');
// 将生成:  <label for="username">What is your Name</label>
```

相似于其他函数，如果你更喜欢设置额外的属性你能在第三个参数里提交一个组合的数组.

例如:

```
$attributes = [
        'class' => 'mycustomclass',
        'style' => 'color: #000;'
];
```

(下页继续)

(续上页)

```
echo form_label('What is your Name', 'username', $attributes);
// 将生成:  <label for="username" class="mycustomclass" style=
↪"color: #000;">What is your Name</label>
```

**form_submit**([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

**参数**

- **$data** (*string*) – Button 名
- **$value** (*string*) – Button 值
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** HTML 输入 submit tag

**返回类型** string

让你产生一个标准的 submit 按钮。简单事例:

```
echo form_submit('mysubmit', 'Submit Post!');
// 将生成:  <input type="submit" name="mysubmit" value="Submit␣
↪Post!" />
```

相似于其他函数，如果你更喜欢设置你的本身的属性你能在第一个参数里提交一个组合数组。第三个参数让你添加额外的数据到你的表单，象 JavaScript.

**form_reset**([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

**参数**

- **$data** (*string*) – Button 名
- **$value** (*string*) – Button 值
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** HTML 输入重新设定 button tag

**返回类型** string

让你生成标准重新设定 button 。使用习惯对 `form_submit()` 是完全相似的.

**form_button**([*$data* = ''[, *$content* = ''[, *$extra* = '']]])

**参数**

- **$data** (*string*) – Button 名
- **$content** (*string*) – Button label
- **$extra** (*mixed*) – 额外的属性被添加到 tag 任何一方像数组或者文字的字符串

**返回** An HTML button tag

**返回类型** string

让你生成标准 button 元素. 你能在第一和第二参数里最低程度地传达 button 名称和内容:

```
echo form_button('name','content');
// 将生成: <button name="name" type="button">Content</button>
```

或者你能传达你的表单去包含你希望包含任何数据的一个组合的数组:

```
$data = [
        'name'    => 'button',
        'id'      => 'button',
        'value'   => 'true',
        'type'    => 'reset',
        'content' => 'Reset'
];

echo form_button($data);
// 将生成: <button name="button" id="button" value="true" type=
↪"reset">Reset</button>
```

如果你想要你的表单包含一些额外的数据，例如 JavaScript ，你能在第三个参数里像字符串一样传达它:

```
$js = 'onClick="some_function()"';
echo form_button('mybutton', 'Click Me', $js);
```

**form_close**([*$extra = ''*])

**参数**

- **$extra** (*string*) – 在关闭 tag 后任何事要追加的, *as is*

**返回** HTML 表单关闭 tag

**返回类型** string

生成正关闭的 </form> tag. 最佳的优势去使用这个函数容许你去传达数据给它，它将会被添加如下文的 tag 。例如:

```
$string = '</div></div>';
echo form_close($string);
// 将生成:  </form> </div></div>
```

**set_value**(*$field*[, *$default = ''*[, *$html_escape = TRUE*]])

**参数**

- **$field** (*string*) – 字段名
- **$default** (*string*) – 默认值

- **$html_escape** (*bool*) – 是否关闭 HTML 值的转义

**返回** 字段值

**返回类型** string

容许你去设置输入表单或者文本区域的值。你必须经过函数的第一个参数提供字段名。第二个操作参数允许你为表单设置一个默认值。第三个操作参数允许你去关闭 HTML 值的转义，万一你需要使用此函数联合，即*form_input()* 并规避双层转义。

例如:

```
<input type="text" name="quantity" value="<?php echo set_value(
↪'quantity', '0'); ?>" size="50" />
```

当第一次加载时下文的表单将显示 “0” .

**set_select**(*$field*[, *$value* = ''[, *$default* = *FALSE*]])

**参数**

- **$field** (*string*) – 字段名
- **$value** (*string*) – 检测的值
- **$default** (*string*) – 是否值也是默认的

**返回** ‘selected’ 属性或者一个空字符串

**返回类型** string

如果你使用 <select> 菜单, 此函数允许你显示已经被选择的菜单题目。.

第一个参数必须包含选择菜单的包含名，第二个参数必须包含选择菜单包含值，而第三个操作参数仍你设置像默认值 (use boolean TRUE/FALSE) 的一个项.

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**

- **$field** (*string*) – 字段名
- **$value** (*string*) – 检测的值
- **$default** (*string*) – 是否值也是默认的

**返回** ‘checked’ 属性或者一个空字符串

**返回类型** string

容许你在已经提交状况下显示一个 checkbox.

第一个参数必须包含 checkbox 的名，第二个参数必须包含它的值，并且第三个操作参数让你设置一个像默认值 (use boolean TRUE/FALSE) 的项.

例如:

```
<input type="checkbox" name="mycheck" value="1" <?php echo set_
↪checkbox('mycheck', '1'); ?> />
<input type="checkbox" name="mycheck" value="2" <?php echo set_
↪checkbox('mycheck', '2'); ?> />
```

**set_radio**(*$field*[, *$value = ''*[, *$default = FALSE*]])

**参数**

- **$field** (*string*) – 字段名
- **$value** (*string*) – 检测的值
- **$default** (*string*) – 是否值也是默认的

**返回** ‘checked’ 属性或者空字符串

**返回类型** string

容许你去显示它们已经提交状态下的 radio buttons . 此函数对于上文*set_checkbox()* 函数是完全相似的。

事例:

```
<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'); ?> />
```

---

**注解:** 如果你正在使用表单验证类，你必须常常为你的字段明确说明一个规范，即使空的，适当的为了 set_*() 函数去工作。这是因为如果表单验证对象已经定义了，控制器为了 set_*() 已经送交了类方法替代一般的辅助函数。

---

### 7.2.6 HTML 辅助函数

HTML 辅助函数包含的函数辅助 HTML 运行。

- *加载 HTML 辅助函数*

- 通用函数

### 加载 HTML 辅助函数

HTML 辅助函数使用下面的代码加载:

```
helper('html');
```

### 通用函数

下面的函数是通用的:

**img**([*$src* = ''[, *$indexPage* = *false*[, *$attributes* = '']]])

**param mixed $src** Image 原始码数据

**param bool $indexPage** 是否像路由的 URI 字符串处理 $src

**param mixed $attributes** HTML 属性

**returns** HTML image tag

**rtype** string

让你创建 HTML <img /> tags. 第一个参数包含 image 原始码。事例:

```
echo img('images/picture.jpg');
// <img src="http://site.com/images/picture.jpg" />
```

有一个可选择的第二参数是特定的 true/false 值并规定如果 *src* 将经由 `$config['indexPage']` 被添加到地址并创建有明确说明的页面。推测起来，假如你正在使用一个 media 控制器那将是自以为是的:

```
echo img('images/picture.jpg', true);
// <img src="http://site.com/index.php/images/picture.jpg" alt="" /
↪>
```

此外，组合数组能被作为第一参数传达，为了完成控制额外的所有属性和值。如果不提供 *alt* 属性，CodeIgniter 将产生空字符串。

例如:

```
$imageProperties = [
    'src'    => 'images/picture.jpg',
    'alt'    => 'Me, demonstrating how to eat 4 slices of␣
↪pizza at one time',
    'class'  => 'post_images',
    'width'  => '200',
```

(下页继续)

(续上页)

```
    'height' => '200',
    'title'  => 'That was quite a night',
    'rel'    => 'lightbox'
];

img($imageProperties);
// <img src="http://site.com/index.php/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" />
```

**link_tag**([*$href* = ''[, *$rel* = 'stylesheet'[, *$type* = 'text/css'[, *$title* = ''[, *$media* = ''[, *$indexPage* = *false*]]]]])

**参数**

- **$href** (*string*) – 链接文件的原始码
- **$rel** (*string*) – 关系类型
- **$type** (*string*) – 关系文件夹的类型
- **$title** (*string*) – 链接主题
- **$media** (*string*) – 媒体类型
- **$indexPage** (*bool*) – 是否像路由的 URI 字符串处理 $src

**返回** HTML link tag

**返回类型** string

让你创建 HTML <link /> tags. 这对样式表链接是有用的, 和其他链接一样。参数是 *href* ， 带着可选择的 *rel*, *type*, *title*, *media* 和 *indexPage*.

*indexPage* 是 boolean 值并规定如果 *href* 将经由 `$config['indexPage']` 被添加到地址并创建有明确说明的页面。

例如:

```
echo link_tag('css/mystyles.css');
// <link href="http://site.com/css/mystyles.css" rel="stylesheet"␣
↪type="text/css" />
```

更多示例:

```
echo link_tag('favicon.ico', 'shortcut icon', 'image/ico');
// <link href="http://site.com/favicon.ico" rel="shortcut icon"␣
↪type="image/ico" />
```

(下页继续)

(续上页)

```
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" />
```

间隔地，为了完全控制额外的所有属性和值组合数组能被传达到 `link_tag()` 函数:

```
$link = [
    'href'  => 'css/printer.css',
    'rel'   => 'stylesheet',
    'type'  => 'text/css',
    'media' => 'print'
];

echo link_tag($link);
// <link href="http://site.com/css/printer.css" rel="stylesheet"␣
↪type="text/css" media="print" />
```

**script_tag**(*$src = ''[, $indexPage = false]*)

**参数**

- **$src** (*mixed*) – JavaScript 文件的原始码名称
- **$indexPage** (*bool*) – 是否像路由的 URI 字符串处理 $src

**返回** HTML script tag

**返回类型** string

让你创建 HTML <script></script> tags. 参数是 *src*, 与可选的 *indexPage* 一起.

*indexPage* 是 boolean 值并规定如果 *src* 将经由 `$config['indexPage']` 被添加到地址并创建有明确说明的页面。

例如:

```
echo script_tag('js/mystyles.js');
// <script src="http://site.com/js/mystyles.js" type="text/
↪javascript"></script>
```

间隔地，为了完全控制额外的所有属性和值组合数组能被通过 `script_tag()` 函数:

```
$script = ['src'  => 'js/printer.js'];

echo script_tag($script);
// <script src="http://site.com/js/printer.js" type="text/
↪javascript"></script>
```

**ul**(*$list[, $attributes = '']*)

**param array $list** 目录登录

**param array $attributes** HTML 属性

**returns** HTML-formatted 无序目录

**rtype** string

容许你从简单或者多倍空间的数组产生无序 HTML 目录。事例::

```
    $list = [
        'red',
        'blue',
        'green',
        'yellow'
    ];

    $attributes = [
        'class' => 'boldlist',
        'id'    => 'mylist'
    ];

    echo ul($list, $attributes);
```

上文的代码将产生下文这样地 HTML 代码:

```
.. code-block:: html

    <ul class="boldlist" id="mylist">
        <li>red</li>
        <li>blue</li>
        <li>green</li>
        <li>yellow</li>
    </ul>
```

下面是更复杂的事例, 使用多维空间的数组::

```
    $attributes = [
        'class' => 'boldlist',
        'id'    => 'mylist'
    ];

    $list = [
        'colors' => [
            'red',
            'blue',
            'green'
        ],
        'shapes' => [
```

(下页继续)

(续上页)

```
            'round',
            'square',
            'circles' => [
                'ellipse',
                'oval',
                'sphere'
            ]
        ],
        'moods'  => [
            'happy',
            'upset'   => [
                'defeated' => [
                    'dejected',
                    'disheartened',
                    'depressed'
                ],
                'annoyed',
                'cross',
                'angry'
            ]
        ]
    ];

    echo ul($list, $attributes);
```

上文的代码将产生这样的 HTML 前端代码:

```
.. code-block:: html

    <ul class="boldlist" id="mylist">
        <li>colors
            <ul>
                <li>red</li>
                <li>blue</li>
                <li>green</li>
            </ul>
        </li>
        <li>shapes
            <ul>
                <li>round</li>
                <li>suare</li>
                <li>circles
                    <ul>
                        <li>elipse</li>
                        <li>oval</li>
                        <li>sphere</li>
```

(下页继续)

(续上页)

```
                    </ul>
                </li>
            </ul>
        </li>
        <li>moods
            <ul>
                <li>happy</li>
                <li>upset
                    <ul>
                        <li>defeated
                            <ul>
                                <li>dejected</li>
                                <li>disheartened</li>
                                <li>depressed</li>
                            </ul>
                        </li>
                        <li>annoyed</li>
                        <li>cross</li>
                        <li>angry</li>
                    </ul>
                </li>
            </ul>
        </li>
    </ul>
```

**ol**(*$list*, *$attributes = ''*)

**参数**

- **$list** (*array*) – 目录登录
- **$attributes** (*array*) – HTML 属性

**返回** HTML-formatted 有序目录

**返回类型** string

完全相似于*ul()* , 为了代替有序目录 <ul> 它仅产生 <ol> tag.

**video**(*$src*[, *$unsupportedMessage = ''*[, *$attributes = ''*[, *$tracks = []*[, *$indexPage = false*]]]])

**参数**

- **$src** (*mixed*) – 任一原始码字符串或者原始码的数组. 参看*source()* 函数
- **$unsupportedMessage** (*string*) – 如果 media tag 不支持由浏览器提供的消息会显示
- **$attributes** (*string*) – HTML 属性

- **$tracks** (*array*) – 在数组里使用追踪函数。参看 *track()* 函数
- **$indexPage** (*bool*) –

**返回** HTML-formatted 影像元素

**返回类型** string

容许你从简单的或者原始码数组产生 HTML 影像元素。事例:

```
$tracks =
[
    track('subtitles_no.vtt', 'subtitles', 'no', 'Norwegian No'),
    track('subtitles_yes.vtt', 'subtitles', 'yes', 'Norwegian Yes')
];

echo video('test.mp4', 'Your browser does not support the video tag.
↪', 'controls');

echo video
(
    'http://www.codeigniter.com/test.mp4',
    'Your browser does not support the video tag.',
    'controls',
    $tracks
);

echo video
(
    [
      source('movie.mp4', 'video/mp4', 'class="test"'),
      source('movie.ogg', 'video/ogg'),
      source('movie.mov', 'video/quicktime'),
      source('movie.ogv', 'video/ogv; codecs=dirac, speex')
    ],
    'Your browser does not support the video tag.',
    'class="test" controls',
    $tracks
 );
```

上文的编码将产生这样地 HTML 前端代码:

```
<video src="test.mp4" controls>
  Your browser does not support the video tag.
</video>

<video src="http://www.codeigniter.com/test.mp4" controls>
  <track src="subtitles_no.vtt" kind="subtitles" srclang="no" label=
↪"Norwegian No" />
  <track src="subtitles_yes.vtt" kind="subtitles" srclang="yes"␣
↪label="Norwegian Yes" />
```

(下页继续)

(续上页)

```
  Your browser does not support the video tag.
</video>

<video class="test" controls>
  <source src="movie.mp4" type="video/mp4" class="test" />
  <source src="movie.ogg" type="video/ogg" />
  <source src="movie.mov" type="video/quicktime" />
  <source src="movie.ogv" type="video/ogv; codecs=dirac, speex" />
  <track src="subtitles_no.vtt" kind="subtitles" srclang="no" label=
↪"Norwegian No" />
  <track src="subtitles_yes.vtt" kind="subtitles" srclang="yes"␣
↪label="Norwegian Yes" />
  Your browser does not support the video tag.
</video>
```

**audio**(*$src*[, *$unsupportedMessage* = ''[, *$attributes* = ''[, *$tracks* = *[]*[, *$indexPage* = *false*]]]])

**参数**

- **$src** (*mixed*) – 任一原始码字符串或者原始码数组。参看*source()* 函数
- **$unsupportedMessage** (*string*) – 如果 media tag 不支持由浏览器提供的消息会显示
- **$attributes** (*string*) –
- **$tracks** (*array*) – 在数组里用追踪函数. 参看*track()* 函数
- **$indexPage** (*bool*) –

**返回** HTML-formatted 音频元素

**返回类型** string

完全相似于*video()*, 它仅仅产生 <audio> tag 代替 <video>.

**source**(*$src* = ''[, *$type* = *false*[, *$attributes* = '']])

**param string $src** media source 的路径

**param bool $type** 以可选择的编码参数的资源 MIME（多用途的网络邮件扩充协议）类型

**param array $attributes** HTML 属性

**returns** HTML source tag

**rtype** string

让你创建 HTML <source /> tags. 第一个参数包含起源 source. 例如:

```
echo source('movie.mp4', 'video/mp4', 'class="test"');
// <source src="movie.mp4" type="video/mp4" class="test" />
```

**embed**(*$src = ''*[, *$type = false*[, *$attributes = ''*[, *$indexPage = false*]]])

**param string $src** 资源的路径 embed

**param bool $type** MIME（多用途的网络邮件扩充协议）类型

**param array $attributes** HTML 属性

**param bool $indexPage**

**returns** HTML embed tag

**rtype** string

让你创建 HTML <embed /> tags. 第一参数包含 embed source. 例如:

```
echo embed('movie.mov', 'video/quicktime', 'class="test"');
// <embed src="movie.mov" type="video/quicktime" class="test"/>
```

**object**(*$data = ''*[, *$type = false*[, *$attributes = ''*]])

**参数**

- **$data** (*string*) – 资源 URL
- **$type** (*bool*) – 资源的内容类型
- **$attributes** (*array*) – HTML 属性
- **$params** (*array*) – 在数组里使用 param 函数。参看*param()* 函数

**返回** HTML object tag

**返回类型** string

让你创建 HTML <object /> tags. 第一参数包含 object data. 事例:

```
echo object('movie.swf', 'application/x-shockwave-flash', 'class=
↪"test"');

echo object
(
    'movie.swf',
    'application/x-shockwave-flash',
    'class="test"',
    [
        param('foo', 'bar', 'ref', 'class="test"'),
        param('hello', 'world', 'ref', 'class="test"')
    ]
);
```

上文编码将产生这样的 HTML 前端代码:

```
<object data="movie.swf" class="test"></object>

<object data="movie.swf" class="test">
  <param name="foo" type="ref" value="bar" class="test" />
  <param name="hello" type="ref" value="world" class="test" />
</object>
```

**param**(*$name = ''*[, *$type = false*[, *$attributes = ''*]])

**参数**

- **$name** (*string*) – 参数的名字
- **$value** (*string*) – 参数的值
- **$attributes** (*array*) – HTML 属性

**返回** HTML param tag

**返回类型** string

让你创建 HTML <param /> tags. 第一个参数包含 param source. 事例:

```
echo param('movie.mov', 'video/quicktime', 'class="test"');
// <param src="movie.mov" type="video/quicktime" class="test"/>
```

**track**(*$name = ''*[, *$type = false*[, *$attributes = ''*]])

**参数**

- **$name** (*string*) – 参数的名称
- **$value** (*string*) – 参数的值
- **$attributes** (*array*) – HTML 属性

**返回** HTML track tag

**返回类型** string

产生一个跟踪元素去具体指定时间的轨迹。在 WebVVT 格式里轨迹已被格式化。事例:

```
echo track('subtitles_no.vtt', 'subtitles', 'no', 'Norwegian No');
// <track src="subtitles_no.vtt" kind="subtitles" srclang="no"␣
↪label="Norwegian No" />
```

**doctype**([*$type = 'html5'*])

**参数**

- **$type** (*string*) – Doctype 名字

**返回** HTML DocType tag

**返回类型** string

帮助你产生 document type 声明，而 DTD’s. HTML 5 是默认使用的，但是许多 doctypes 是通用的。

事例:

```
echo doctype();
// <!DOCTYPE html>

echo doctype('html4-trans');
// <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.
↪w3.org/TR/html4/strict.dtd">
```

接下来的是重定义 doctype 选择的目录。这些是可设置的，被从 *application/Config/DocTypes.php* 出栈，或者在你的 *.env* 结构里它们能被加载。

| 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" > |
| 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" > |
| XHTML+RDFa | xhtml- | <!DOCTYPE html PUBLIC "-//W3C//DTD |

### 7.2.7 偏转辅助函数

偏转辅助函数文件包含的函数容许你改变 ** 英文 ** 词汇到复数，单数，驼峰式大小写，等等。



#### 加载偏转辅助函数

偏转辅助函数使用下面的代码加载:

```
helper('inflector');
```

#### 通用函数

下面的函数是通用的:

**singular**(*$string*)

**参数**

- **$string** (*string*) – 输入 string

**返回** 单数单词

**返回类型** string

改变复数单词为单数。例如:

```
echo singular('dogs'); //  打印出  'dog'
```

**plural**(*$string*)

**参数**

- **$string** (*string*) – 输入 string

**返回** 复数单词

**返回类型**

string

改变单数单词为复数。例如:

```
echo plural('dog'); // 打印出  'dogs'
```

**counted**(*$count*, *$string*)

**参数**

- **$count** (*int*) – Number of items
- **$string** (*string*) – Input string

**返回** A singular or plural phrase

**返回类型** string

Changes a word and its count to a phrase. 例如:

```
echo counted(3, 'dog'); // 打印出 '3 dogs'
```

**camelize**(*$string*)

**参数**

- **$string** (*string*) – 输入 string

**返回** 驼峰化 string

**返回类型** string

由空格或者下划线改变单词分割的字符串为驼峰式大小写。例如:

```
echo camelize('my_dog_spot'); // 打印出 'myDogSpot'
```

**pascalize**(*$string*)

**参数**

- **$string** (*string*) – Input string

**返回** Pascal case string

**返回类型** string

Changes a string of words separated by spaces or underscores to Pascal case, which is camel case with the first letter capitalized. 例如:

```
echo pascalize('my_dog_spot'); // 打印出 'MyDogSpot'
```

**underscore**(*$string*)

**参数**

- **$string** (*string*) – 输入 string

**返回** 字符串包含下划线代替空格

**返回类型** string

由多空格和下划线带来多样的单词分割。事例:

```
echo underscore('my dog spot'); // 打印出 'my_dog_spot'
```

**humanize**(*$string*[, *$separator* = '_'])

**参数**

- **$string** (*string*) – 输入 string

- **$separator** (*string*) – 输入分隔符 Input separator

**返回** 人性化的 string

**返回类型** string

由空格带来复合单词的分割并在他们中间添加空格。每个单词用大写书写。

事例:

```
echo humanize('my_dog_spot'); // 打印出 'My Dog Spot'
```

使用波折号代替下划线:

```
echo humanize('my-dog-spot', '-'); // 打印出 'My Dog Spot'
```

**is_pluralizable**(*$word*)

**参数**

- **$word** (*string*) – 输入 string

**返回** 如果单词为可数的则 TRUE 否则 FALSE

**返回类型** bool

多次核对假设约定的单词已经有一个复数版本。事例:

```
is_pluralizable('equipment'); // 返回 FALSE
```

**dasherize**(*$string*)

**参数**

- **$string** (*string*) – 输入 string

**返回** 底线转换 string

**返回类型** string

在 string 里取代带着波折号的下划线。事例:

```
dasherize('hello_world'); // 返回 'hello-world'
```

**ordinal**(*$integer*)

**参数**

- **$integer** (*int*) – integer 决定词尾

**返回** 顺序的词尾

**返回类型** string

返回的词尾应该添加一个数目去表示位置例如 1st, 2nd, 3rd, 4th. 事例:

```
ordinal(1); // 返回 'st'
```

**ordinalize**(*$integer*)

**参数**

- **$integer** (*int*) – integer 序号

**返回** 序数化 integer

**返回类型** string

转换数目为顺序的字符串过去总是指示位置例如 1st, 2nd, 3rd, 4th. 事例:

```
ordinalize(1); // 返回 '1st'
```

### 7.2.8 数字辅助函数

在本地化识别风格里数字辅助函数文件包含的函数帮助你与数字化的数据工作。



#### 加载数字辅助函数

数字辅助函数使用下面的代码加载:

```
helper('number');
```

#### 当某些事情出岔子

如果 PHP 的国际化和本地化不能分给被提供的值，由于赋予了区域和选项，那么 `BadFunctionCallException()` 函数将会被掷出。

#### 通用函数

下面的函数是通用的:

**number_to_size**(*$num[, $precision = 1[, $locale = null]]*)

**参数**

- **$num** (*mixed*) – 字节的数目
- **$precision** (*int*) – 浮点精确度

**返回** 格式化数据大小 string, 要不然如果提供的值不是数字的则是错误的

**返回类型** string

像字节一样格式化数字，以大小为基础，并添加适事例当的词尾。事例:

```
echo number_to_size(456); // 返回 456 Bytes
echo number_to_size(4567); // 返回 4.5 KB
echo number_to_size(45678); // 返回 44.6 KB
echo number_to_size(456789); // 返回 447.8 KB
echo number_to_size(3456789); // 返回 3.3 MB
echo number_to_size(12345678912345); // 返回 1.8 GB
echo number_to_size(123456789123456789); // 返回 11,228.3 TB
```

第二个可选的参数允许你设置结果的精确度:

```
echo number_to_size(45678, 2); // 返回 44.61 KB
```

第三个可选的参数当产生数字时应该常被使用，并能对格式化产生作用，它允许你去具体指定区域。如果没有区域被具体指定，请求将会被解析并且适当区域会减少头文件或者本地应用默认程序:

```
// 产生 11.2 TB
echo number_to_size(12345678912345, 1, 'en_US');
// 产生 11,2 TB
echo number_to_size(12345678912345, 1, 'fr_FR');
```

---

**注解:** 由本段函数产生文本在接下来的语言文件中被找到: *language/<your_lang>/Number.php*

---

**number_to_amount**(*$num[, $precision = 1[, $locale = null]]*)

**参数**

- **$num** (*mixed*) – 数字格式
- **$precision** (*int*) – 浮点精确度
- **$locale** (*string*) – 为了格式化区域使用

**返回** string 的可读版本，要不然如果提供的值不是数字的为错误的

**返回类型** string

为了计数能达到百万的四次方，转换数字格式为人类可读版本，像 **123.4 trillion**. 事例:

```
echo number_to_amount(123456); // 返回 123 thousand
echo number_to_amount(123456789); // 返回 123 million
echo number_to_amount(1234567890123, 2); // 返回 1.23 trillion
echo number_to_amount('123,456,789,012', 2); // 返回 123.46 billion
```

一个可选择的第二参数允许你去设置结果的精确度:

```
echo number_to_amount(45678, 2); // 返回 45.68 thousand
```

一个可选择的第三参数允许区域被具体指定:

```
echo number_to_amount('123,456,789,012', 2, 'de_DE'); // 返回 123,
↪46 billion
```

**number_to_currency**(*$num*, *$currency*[, *$locale = null*])

**参数**

- **$num** (*mixed*) – 数字格式
- **$currency** (*string*) – 货币类型, 例如 USD, EUR, 等等
- **$locale** (*string*) – 为了格式化区域使用
- **$fraction** (*integer*) – Number of fraction digits after decimal point

**返回** 为了本地化数字应与货币相称

**返回类型** string

在公用的通货格式里转换数字, 例如 USD, EUR, GBP, 等等:

```
echo number_to_currency(1234.56, 'USD');  // 返回 $1,234.56
echo number_to_currency(1234.56, 'EUR');  // 返回 £1,234.56
echo number_to_currency(1234.56, 'GBP');  // 返回 £1,234.56
echo number_to_currency(1234.56, 'YEN');  // 返回 YEN1,234.56
```

**number_to_roman**(*$num*)

**参数**

- **$num** (*string*) – 想要转换的数字

**返回** 来自赋予参数的被转换的 roman 数字

**返回类型** string

转换数字为 roman:

```
echo number_to_roman(23);  // 返回 XXIII
echo number_to_roman(324);  // 返回 CCCXXIV
echo number_to_roman(2534);  // 返回 MMDXXXIV
```

函数仅处理 1 到 3999 之间的数字。超出范围的任何值它将返回空。

### 7.2.9 安全辅助函数

安全辅助函数文件包含安全相关联的函数。

## 加载安全辅助函数

安全辅助函数使用下面的代码加载:

```
helper('security');
```

## 通用函数

下面是通用函数:

**sanitize_filename**(*$filename*)

**参数**

- **$filename** (*string*) – 文件名

**返回** 净化文件名

**返回类型** string

提供保护来应对磁盘遍历。

对于 \CodeIgniter\Security::sanitize_filename() 本函数仅是别名。更多信息，请查看文档*Security Library* 。

**strip_image_tags**(*$str*)

**参数**

- **$str** (*string*) – 输入 string

**返回** 无成像标签的输入

**返回类型** string

这是一个将无成像标签从 string 中剥去的安全函数。它留下成像 URL 就像清楚的文本一样。

例如:

```
$string = strip_image_tags($string);
```

**encode_php_tags**(*$str*)

**参数**

- **$str** (*string*) – 输入 string

**返回** 安全地格式化 string

**返回类型** string

这是一个安全函数去转换 PHP 标签为实体。

例如:

```
$string = encode_php_tags($string);
```

### 7.2.10 文本辅助函数

文本辅助函数文件包括了一系列有助于处理文本的函数



#### 加载辅助函数

该系列函数通过以下方式加载:

```
helper('text');
```

#### 可用函数列表

以下函数可用:

**random_string**([*$type = 'alnum'*[, *$len = 8*]])

**参数**

- **$type** (*string*) – 需要随机输出的类型
- **$len** (*int*) – 输出的字符串长度

**返回** 一个随机字符串

**返回类型** string

基于类型和长度生成一个随机字符串。对于创建密码或随机哈希等非常有用。

第一个参数给定字符串类型，第二个参数给定字符串长度，可使用以下类型:

- **alpha**: 仅有大小写字母构成的字符串
- **alnum**: 含有大小写字母和数字的字符串
- **basic**: 基于 `mt_rand()` 方法组成的随机数（忽略长度）
- **numeric**: 数字类型的字符串
- **nozero**: 数字类型字符串，其中不含有零
- **md5**: 基于 `md5()` 的加密随机数（固定长度 32 位）

- **sha1**: 基于 sha1() 的加密随机数（固定长度 40 位）
- **crypto**: 基于 random_bytes() 的随机字符串

用例如下:

```
echo random_string('alnum', 16);
```

**increment_string**(*$str*[, *$separator = '_'*[, *$first = 1*]])

**参数**

- **$str** (*string*) – 输入的字符串
- **$separator** (*string*) – 用于增加一个数字的分隔符
- **$first** (*int*) – 起始数字

**返回** 递增字符串

**返回类型** string

通过将一个每次在尾部递增数字的方式，递增一个字符串。用于创建”拷贝”或者用于拥有唯一标题或简介的文件或数据库内容。

用例如下:

```
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*) – 参数的一个变量数字

**返回** 变化后的字符串

**返回类型** mixed

允许在进行循环时，两个或多个项目之间交换变化，例如:

```
for ($i = 0; $i < 10; $i++)
{
        echo alternator('string one', 'string two');
}
```

如果你需要的话也可以增加尽可能多的参数，在下一次迭代时，下一个项目将会被返回。

```
for ($i = 0; $i < 10; $i++)
{
        echo alternator('one', 'two', 'three', 'four', 'five');
}
```

**注解：** 多个独立函数调用时，只需要不传参，不用重新初始化直接调用即可。

`reduce_double_slashes`(*$str*)

**参数**

- **$str** (*string*) – 输入字符串

**返回** 格式化斜线后的字符串

**返回类型** string

将一个字符串中的双斜线转变为单斜线，除了在 URL 协议前缀中的，比如 http://

例如:

```
$string = "http://example.com//index.php";
echo reduce_double_slashes($string); // 返回 "http://example.com/
↪index.php"
```

`strip_slashes`(*$data*)

**参数**

- **$data** (*mixed*) – 输入的字符串或者字符串数组

**返回** 去除斜杠后的字符串（数组）

**返回类型** mixed

从一组字符串中去除所有斜杠

例如:

```
$str = [
        'question' => 'Is your name O\'reilly?',
        'answer'   => 'No, my name is O\'connor.'
];

$str = strip_slashes($str);
```

以上会返回数组:

```
[
        'question' => "Is your name O'reilly?",
        'answer'   => "No, my name is O'connor."
];
```

**注解：** 基于历史原因，该函数也接受字符串类型的输入。这样看起来就跟 `stripslashes()` 函数的别名一样 alias for `stripslashes()`.

reduce_multiples(*$str*[, *$character* = ''[, *$trim* = *FALSE*]])

**参数**

- **$str** (*string*) – 需要搜索的文本
- **$character** (*string*) – 需要简化的字符
- **$trim** (*bool*) – 是否在字符串首位同时去除指定的字符

**返回** 简化后的字符串

**返回类型** string

将多个连续出现的相同字符简化为一个，例如:

```
$string = "Fred, Bill,, Joe, Jimmy";
$string = reduce_multiples($string,","); //结果 "Fred, Bill, Joe,␣
↪Jimmy"
```

如果第三个参数被设为 TRUE 的话，该函数就会将首部和尾部出现的该字符串同时去除，例如:

```
$string = ",Fred, Bill,, Joe, Jimmy,";
$string = reduce_multiples($string, ", ", TRUE); //结果是 "Fred,␣
↪Bill, Joe, Jimmy"
```

quotes_to_entities(*$str*)

**参数**

- **$str** (*string*) – 输入的字符串

**返回** 拥有转义符号的字符串转换后的 HTML 实体

**返回类型** string

将一个单引号或双引号转换为对应的 HTML 实体，例如:

```
$string = "Joe's \"dinner\"";
$string = quotes_to_entities($string); //结果是 "Joe&#39;s &quot;
↪dinner&quot;"
```

strip_quotes(*$str*)

**参数**

- **$str** (*string*) – 输入字符串

**返回** 去除了引号的字符串

**返回类型** string

从字符串中去除单双引号，例如:

```
$string = "Joe's \"dinner\"";
$string = strip_quotes($string); //结果是 "Joes dinner"
```

word_limiter(*$str*[, *$limit = 100*[, *$end_char = '…'*]])

**参数**

- **$str** (*string*) – 输入字符串
- **$limit** (*int*) – 限制
- **$end_char** (*string*) – 结尾字符（通常是省略号）

**返回** 限制了单词的字符串

**返回类型** string

根据 单词的长度截断字符串，例如:

```
$string = "Here is a nice text string consisting of eleven words.";
$string = word_limiter($string, 4);
// Returns:  Here is a nice
```

第三个参数是一个可选的字符串后缀。默认是一个省略号。

character_limiter(*$str*[, *$n = 500*[, *$end_char = '…'*]])

**参数**

- **$str** (*string*) – 输入字符串
- **$n** (*int*) – 字符数量
- **$end_char** (*string*) – 结尾字符

**返回** 限定了字符的字符串

**返回类型** string

根据给定的 字符的数量截断字符串。该方法将会保持单词的完整性，因此字符串长度可能会比你给定的略多或略少

例如:

```
$string = "Here is a nice text string consisting of eleven words.";
$string = character_limiter($string, 20);
// 返回:  Here is a nice text string
```

第三个参数是一个可选的字符串后缀，未定义则默认使用省略号

---

**注解:** 如果你想截断完全一致长度的字符串，参照下方的函数*ellipsize()*

---

ascii_to_entities(*$str*)

**参数**

- **$str** (*string*) – 输入字符串

**返回** 一个将 ASCII 值转化为实体的字符串

**返回类型** string

将 ASCII 码转化为字符实体，包括可能导致 web 页面中出现问题的高位 ASCII 码以及一些 Word 字符串。通过这一方法可以使得这些字符无论是浏览器设置或是存储于数据库中都可以正确地显示。不过该方法依赖于你浏览器所支持的字符集，因此不一定 100% 可靠。不过在大多数情况下，该方法可以正确识别非正常类型的字符（例如方言字符等）

例如:

```
$string = ascii_to_entities($string);
```

**entities_to_ascii**(*$str*[, *$all = TRUE*])

**参数**

- **$str** (*string*) – 输入字符串
- **$all** (*bool*) – 是否同样转换非安全的实体

**返回** 将 HTML 实体转化为 ASCII 码的字符串

**返回类型** string

该函数与*ascii_to_entities()* 相反，将字符实体转换为 ASCII 码

**convert_accented_characters**(*$str*)

**参数**

- **$str** (*string*) – 输入字符串

**返回** 一个字符串，其中方言字符已进行过转换

**返回类型** string

将高位 ASCII 码转化为等同功能的低位 ASCII 码。当面对只有标准 ASCII 码可以安全使用的情况，将非英语的字符进行转换，比如在 URL 中

例如:

```
$string = convert_accented_characters($string);
```

---

**注解:** 该函数利用配置文件 *app/Config/ForeignCharacters.php* 来定义并进行数组翻译。

---

**word_censor**(*$str*, *$censored*[, *$replacement = ''*])

**参数**

- **$str** (*string*) – 输入字符串
- **$censored** (*array*) – 一系列需要被探测的有问题的单词
- **$replacement** (*string*) – 用于替换问题单词的字符串

**返回** 探测后的字符串

**返回类型** string

用于检测文本字符串中的敏感词。第一个参数为原有的字符串，第二个是一个含有你需要拦截的敏感词的数组。第三个参数（可选）为需要用于替换的单词。如果不声明的话就会用井号替换: ###

例如:

```
$disallowed = ['darn', 'shucks', 'golly', 'phooey'];
$string     = word_censor($string, $disallowed, 'Beep!');
```

**highlight_code**(*$str*)

**参数**

- **$str** (*string*) – 输入字符串

**返回** HTML 格式代码高亮的字符串

**返回类型** string

将一个代码字符串（PHP, HTML, 等）加上颜色。例如:

```
$string = highlight_code($string);
```

该函数使用了 PHP 的 highlight_string() 方法，因此使用的颜色是在你的 php.ini 文件中定义的。

**highlight_phrase**(*$str*, *$phrase*[, *$tag_open* = *'<mark>'*[, *$tag_close* = *'</mark>'*]])

**参数**

- **$str** (*string*) – 输入字符串
- **$phrase** (*string*) – 高亮的片段
- **$tag_open** (*string*) – 用于高亮的开括号
- **$tag_close** (*string*) – 用于高亮的闭括号

**返回** 通过 HTML 进行片段高亮后的字符串

**返回类型** string

在一个文本字符串中高亮一个片段。第一个参数是原本的字符串，第二个参数是你需要高亮的片段。第三个第四个参数包含你需要用于包裹高亮片段的 HTML 标签。

例如:

```
$string = "Here is a nice text string about nothing in particular.";
echo highlight_phrase($string, "nice text", '<span style="color:
↪#990000;">', '</span>');
```

以上将会输出:

```
Here is a <span style="color:#990000;">nice text</span> string␣
↪about nothing in particular.
```

**注解：** 该函数默认使用  标签。旧版本的浏览器可能不支持新型 HTML5 的格式标签，因此我们推荐你将下述 CSS 加入到你的样式表中，如果你需要支持这类浏览器的话:

```
mark {
        background: #ff0;
        color: #000;
};
```

**word_wrap**(*$str*[, *$charlim = 76*])

**参数**

- **$str** (*string*) – 输入字符串
- **$charlim** (*int*) – 字符限制

**返回** 单词换行过的字符串

**返回类型** string

将一个文本以指定的字符长度进行换行，并保持单词完整性

例如:

```
$string = "Here is a simple string of text that will help us␣
↪demonstrate this function.";
echo word_wrap($string, 25);

// 输出如下:
// Here is a simple string
// of text that will help us
// demonstrate this
// function.
```

过长的单词会被截断，不过 URL 不会

**ellipsize**(*$str*, *$max_length*[, *$position = 1*[, *$ellipsis = '…'*]])

**参数**

- **$str** (*string*) – 输入字符串
- **$max_length** (*int*) – 字符串长度限制
- **$position** (*mixed*) – 需要截断的位置（整数或浮点数）
- **$ellipsis** (*string*) – 作为省略的标记符

**返回** 省略后的字符串

**返回类型** string

该函数将去除字符串中的标记并将其截断为指定长度，同时加上一个省略标记符

第一个参数是需要省略的字符串，第二个是在输出的字符串中的字符长度。第三个参数是在省略后的字符串中，省略标记符号是否需要从 0-1，从左到右的方式出现。例如，值为 1 时，就会在右边，0.5 就是中间，0 就是在左边。

第四个可选的参数是省略符号类型，默认情况下会插入一个 …

例如:

```
$str = 'this_string_is_entirely_too_long_and_might_break_my_design.
↪jpg';
echo ellipsize($str, 32, .5);
```

结果:

```
this_string_is_e&hellip;ak_my_design.jpg
```

**excerpt**(*$text*, *$phrase = false*, *$radius = 100*, *$ellipsis = '...'*)

**参数**

- **$text** (*string*) – 需要截取摘要的文本
- **$phrase** (*string*) – 需要截取的文本附近的片段或单词
- **$radius** (*int*) – 在片段前后截取的字符数量
- **$ellipsis** (*string*) – 省略标记符

**返回** 摘要.

**返回类型** string

该函数会取出指定 $phrase 前后各 $radius 个数量的字符。

第一个参数是需要截取摘要的文本，第二个是需要截取的中心单词或片段。第三个参数是需要截取的数量。如果不传 $phrase 参数的话就会从头开始获取 $radius 个字符并加上省略标记符

例如:

```
$text = 'Ut vel faucibus odio. Quisque quis congue libero. Etiam␣
↪gravida
eros lorem, eget porttitor augue dignissim tincidunt. In eget risus␣
↪eget
mauris faucibus molestie vitae ultricies odio. Vestibulum id␣
↪ultricies diam.
Curabitur non mauris lectus. Phasellus eu sodales sem. Integer␣
↪dictum purus
ac enim hendrerit gravida. Donec ac magna vel nunc tincidunt␣
↪molestie sed
vitae nisl. Cras sed auctor mauris, non dictum tortor. Nulla vel␣
↪scelerisque
```

(下页继续)

(续上页)

```
arcu. Cras ac ipsum sit amet augue laoreet laoreet. Aenean a risus␣
↪lacus.
Sed ut tortor diam.';

echo excerpt($str, 'Donec');
```

输出:

```
... non mauris lectus. Phasellus eu sodales sem. Integer dictum␣
↪purus ac
enim hendrerit gravida. Donec ac magna vel nunc tincidunt molestie␣
↪sed
vitae nisl. Cras sed auctor mauris, non dictum ...
```

### 7.2.11 URL 辅助函数

URL 辅助函数文件包含的函数辅助 URLs 运行。



#### 加载 URL 辅助函数

在每个请求中 URL 辅助函数由框架自动地加载。

#### 通用函数

下文函数是通用的:

**site_url**([*$uri = ''*[, *$protocol = NULL*[, *$altConfig = NULL*]]])

**参数**

- **$uri** (*string*) – URI string
- **$protocol** (*string*) – 协议，处理资料传送的标准，例如 'http' 或者 'https'
- **$altConfig** (*\Config\App*) – 使用更替配置

**返回** Site URL

**返回类型** string

返回你的 site URL，就像在你的配置文件里说明的。index.php 文件（或者在你的配置文件里任何你已经设置在你网站的 **index_page**）将会添加到 URL，如同你通过函数程序段的一些 URL，外加在你的配置文件中已经设置的 **url_suffix**.

在你的 URL 改变的事件中，你被鼓励在任何时间使用函数生成本地 URL 以便你的页面将变得更加便携。

程序段能随意地像 string 或者 array 通过函数。下文是 string 事例:

```
echo site_url('news/local/123');
```

上文的事例返回的地址如下: *http://example.com/index.php/news/local/123*

这里是一个通过数组程序段的事例:

```
$segments = ['news', 'local', '123'];
echo site_url($segments);
```

对不同的网站如果生成 URLs 你或许会找到比你的配置更有用的更替配置，该函数包含不同配置优先权。我们为单元测试框架本身使用这个函数。

**base_url**([*$uri = ''*[, *$protocol = NULL*]])

**参数**

- **$uri** (*string*) – URI string
- **$protocol** (*string*) – 协议，处理资料传送的标准，例如 'http' 或者 'https'

**返回** 基地址 URL

**返回类型** string

返回你网站的基地址 URL, 如同在你配置文件里具体说明的。事例:

```
echo base_url();
```

如同*site_url()* 该函数返回相同的事件, 排除 *index_page* 或者 *url_suffix* 被附加的情况。

也如函数*site_url()*, 你能提供程序段如 string 或者 array. 这里是 string 事例:

```
echo base_url("blog/post/123");
```

上文事例返回的地址如下: *http://example.com/blog/post/123*

因为不同的*site_url()* 函数是有用的, 你能提供 string 值到文件里，譬如图片或者层叠式样式表。例如:

```
echo base_url("images/icons/edit.png");
```

上文的输出函数将给你如下面的链接: *http://example.com/images/icons/edit.png*

**current_url**([*$returnObject = false*])

**参数**

- **$returnObject** (*boolean*) – True 如果你想要 URI 事例返回，代替 string。

**返回** 最近的 URL

**返回类型** string|URI

返回最近被浏览过的页面的正确的 URL (包括程序段)。

---

**注解:** 引用下面的函数是同样的:

---

```
base_url(uri_string());
```

**previous_url**([*$returnObject = false*])

**参数**

- **$returnObject** (*boolean*) – True 如果你想要 URI 事例返回，代替 string.

**返回** URL 用户以前通过的

**返回类型** string|URI

返回完整页面的 URL （包含程序段）是用户以前通过的。

由于安全问题造成盲目的信任 HTTP_REFERER 系统变量，在对话里如果它是有用的 CodeIgniter 将储存以前浏览的页面。这保证我们将常常使用已知且可信的源，如果对话已经被加载了，或者是别的方式不能得到的，那么 HTTP_REFERER 的净化版本将会被应用。

**uri_string**()

**返回** An URI string

**返回类型** string

返回你的最近 URL 的路径部分。例如，如果你的 URL 是这样的:

```
http://some-site.com/blog/comments/123
```

函数将返回:

```
blog/comments/123
```

**index_page**([*$altConfig = NULL*])

**参数**

- **$altConfig** (*\Config\App*) – 使用更替配置

**返回** 'index_page' 值

**返回类型** mixed

返回你网站的 **index_page**, 如同在你的配置文件里明确说明的。事例:

```
echo index_page();
```

如同用*site_url()*, 你也许要具体制定一个更替配置。对不同的网站如果生成 URLs 你或许会找到比你现有的更有用的更替配置，函数包含不同配置优先权。我们为单元测试框架本身使用这个函数。

**anchor**([*$uri* = ''[, *$title* = ''[, *$attributes* = ''[, *$altConfig* = *NULL*]]]])

**参数**

- **$uri** (*mixed*) – URI 程序段的 URI string 或者 array
- **$title** (*string*) – 锚定 title
- **$attributes** (*mixed*) – HTML 属性
- **$altConfig** (*ConfigApp*) – 使用更替配置

**返回** HTML 超连结 (锚定 tag)

**返回类型** string

基于你本地网站 URL 创建标准 HTML 锚定链接。

第一个参数能包含任意你希望应用到 URL 的程序段。如同上文用*site_url()* 函数，程序段可以是 string 或者 array.

---

**注解:** 如果你正在构造的链接对于你的应用是内部的则不包含基地址 URL (http://…). 在你的配置文件里函数将会明确说明的从信息里被自动添加。你希望附加到的 URL 仅仅包含 URI 的程序段。

---

第二参数是你想要链接表达的正文。如果你留下第二个程序为空，URL 将会被应用。

第三个参数包含你想要添加到链接里的的属性列表。属性可以是简单的 string 或者组合数组。

这里是一些示例

```
echo anchor('news/local/123', 'My News', 'title="News title"');
// Prints: <a href="http://example.com/index.php/news/local/123"␣
↪title="News title">My News</a>

echo anchor('news/local/123', 'My News', array('title' => 'The best␣
↪news!'));
// Prints: <a href="http://example.com/index.php/news/local/123"␣
↪title="The best news!">My News</a>

echo anchor('', 'Click here');
// Prints: <a href="http://example.com/index.php">Click here</a>
```

如同上文阐述的，你也许可以明确说明更替配置。如果对不同网站生成链接你也许会发现更替配置比你的配置是更有用的，它包含不同的配置优先权。我们为单元测试框架自身使用这个函数。

---

**注解:** 属性载入锚定函数是自动地退出对 XSS 攻击不利的保护。

---

**anchor_popup**([*$uri* = ''[, *$title* = ''[, *$attributes* = *FALSE*[, *$altConfig* = *NULL*]]]])

**参数**

- **$uri** (*string*) – URI string
- **$title** (*string*) – 锚定 title
- **$attributes** (*mixed*) – HTML 属性
- **$altConfig** (*ConfigApp*) – 使用更替配置

**返回** 自动跳起的 hyperlink

**返回类型** string

几乎同源于*anchor()* 函数，除了在新窗口里它是开放的 URL。在第三个参数中你能明确说明 JavaScript 窗口属性去控制窗口如何被打开。如果第三个参数没有设定，它将会带着你自身的浏览器设定去简单地打开一个新窗口。

这里是带着属性的事例:

```
$atts = [
    '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);

As above, you may specify an alternate configuration. You may find the alternate configuration useful if generating links for a different site than yours, which contains different configuration preferences. We use this for unit testing the framework itself.

---

**注解:** 上文属性是默认函数因此你仅仅需要去设置哪些个不同于你需要的属性。在第三个参数里如果你想要函数去简单地通过空数组使用所有它的默认值:

---

:: echo anchor_popup( ‘news/local/123’ , ‘Click Me!’ , []);

**注解:** **window_name** 不是真实的属性，但是对于 JavaScript 争论 window.open() 方法，它接受任何一方的窗口名或者窗口目标。

**注解:** 任何超过上文列表的其他属性将会被分列就像 HTML 属性对于锚定 tag. 如同上文描述的，你也许可以明确说明更替配置。你也许会发现如果正生成的链接对不同的网站更替配置比你的配置更有用，他包含不同的配置优先权。我们为单元测试框架自身使用这个函数。

**注解:** 属性载入锚定自动跳起函数是自动地退出对 XSS 攻击不利的保护。

**mailto**(*$email*[, *$title* = ''[, *$attributes* = '']])

**参数**

- **$email** (*string*) – E-mail 地址
- **$title** (*string*) – 锚定 title
- **$attributes** (*mixed*) – HTML 属性

**返回** “mail to” 超连结

**返回类型** string

创建标准的 HTML 邮件链接。用法事例:

```
        echo mailto('me@my-site.com', 'Click Here to Contact Me');

如同用上文 :php:func:`anchor()` tab 函数，
你可以使用第三个参数设定属性::

        $attributes = array('title' => 'Mail me');
        echo mailto('me@my-site.com', 'Contact Me', $attributes);
```

**注解:** 属性载入锚定 mailto 函数是自动地退出对 XSS 攻击不利的保护。

**safe_mailto**(*$email*[, *$title* = ''[, *$attributes* = '']])

**参数**

- **$email** (*string*) – E-mail 地址
- **$title** (*string*) – 锚定 title
- **$attributes** (*mixed*) – HTML 属性

**返回** 安全垃圾邮件 “mail to” 超连结

**返回类型** string

完全相似于*mailto()* 函数除了 *mailto* tag 的模糊版本，由于垃圾邮件群聊程序用 JavaScript 写了该函数正使用序数数字用以从保护已经收获的 e-mail 地址。

**auto_link**(*$str*[, *$type = 'both'*[, *$popup = FALSE*]])

**参数**

- **$str** (*string*) – 输入 string
- **$type** (*string*) – 链接类型（‘email’，‘url’或者‘both’）
- **$popup** (*bool*) – 是否创建自动跳起链接

**返回** 链接化的 string

**返回类型** string

在字符到链接里自动地转换包含 URLs 和 e-mail 地址。事例:

```
$string = auto_link($string);
```

第二参数决定是否 URLs 和 e-mail 是转换了仅仅一个或者其他什么的。如果参数不是明确的说明默认行为是兼有的。E-mail 链接编码如同上文显示的*safe_mailto()* 一样。

仅转换 URLs:

```
$string = auto_link($string, 'url');
```

仅转换 e-mail 地址:

```
$string = auto_link($string, 'email');
```

第三个参数决定是否链接在新窗口被显示。值是 TRUE 或者 FALSE（boolean）:

```
$string = auto_link($string, 'both', TRUE);
```

**注解:** 仅有的被普遍承认的 URLs 这些链接用“www.”或者用“://”开始。

**url_title**(*$str*[, *$separator = '-'*[, *$lowercase = FALSE*]])

**参数**

- **$str** (*string*) – 输入 string
- **$separator** (*string*) – 字符分隔符
- **$lowercase** (*bool*) – 是否转换输出 string 为小写字型

**返回** 已经格式化的 string

**返回类型** string

取 string 作为输入值并创建友好人性化的 URL string. 这是有用的，例如，在 URL 里你有个 blog，在 blog 里你想要使用你的整个主题。事例:

```
$title     = "What's wrong with CSS?";
$url_title = url_title($title);
// Produces: Whats-wrong-with-CSS
```

第二个参数决定词汇的定义符号。默认的破折号被使用。更好的选项是: - (破折号) 或者 _ (下划线)。

例如:

```
$title     = "What's wrong with CSS?";
$url_title = url_title($title, 'underscore');
// Produces: Whats_wrong_with_CSS
```

第三个参数决定是或者不是小写字符是被强迫的。默认他们不是。选项是 boolean TRUE/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** 协议前缀 URL string

**返回类型** string

在事件里这个函数正从一个 URL 错过，它将添加 *http://* 协议前缀。通过 URL string 的函数像下文这样:

```
$url = prep_url('example.com');
```

## 7.2.12 XML 辅助函数

XML 辅助函数文件包含一些用于处理 XML 数据的函数。



### 加载辅助函数

辅助函数是通过以下代码加载的

```
helper('xml');
```

### 可用的函数

可使用以下函数:

**xml_convert**(*$str*[, *$protect_all = FALSE*])

**param string $str** 所需要转换的文本字符串

**param bool $protect_all** 是否保持那些看起来是一个潜在实体的结构而非将其转化为数字标识的实体，例如 $foo。

**returns** 转化成 XML 结构的字符串

**rtype** string

将一个字符串作为输入并将以下的保留 XML 字符转化为实体:

- 与操作符: &
- 大于小于号: < >
- 单双引号: ‘“
- 横杠: -

该函数将忽略作为数字字符实体的一部分而存在的 & 符号，例如 { 。如下所示:

```
        $string = '<p>Here is a paragraph & an entity (&#123;).</p>
↪';
        $string = xml_convert($string);
        echo $string;

输出:

.. code-block:: html

        &lt;p&gt;Here is a paragraph &amp; an entity (&#123;).&lt;/
↪p&gt;
```

# 章节 8

# 高级主题

## 8.1 测试

CodeIgniter 具备许多工具，可帮助你彻底测试和调试应用程序。以下各章节将帮助你快速测试应用程序。

### 8.1.1 Testing

CodeIgniter has been built to make testing both the framework and your application as simple as possible. Support for `PHPUnit` is built in, and the framework provides a number of convenient helper methods to make testing every aspect of your application as painless as possible.



#### System Set Up

### Installing phpUnit

CodeIgniter uses phpUnit as the basis for all of its testing. There are two ways to install phpUnit to use within your system.

### Composer

The recommended method is to install it in your project using Composer. While it' s possible to install it globally we do not recommend it, since it can cause compatibility issues with other projects on your system as time goes on.

Ensure that you have Composer installed on your system. From the project root (the directory that contains the application and system directories) type the following from the command line:

```
> composer require --dev phpunit/phpunit
```

This will install the correct version for your current PHP version. Once that is done, you can run all of the tests for this project by typing:

```
> ./vendor/bin/phpunit
```

### Phar

The other option is to download the .phar file from the phpUnit site. This is a standalone file that should be placed within your project root.

### Testing Your Application

### PHPUnit Configuration

The framework has a `phpunit.xml.dist` file in the project root. This controls unit testing of the framework itself. If you provide your own `phpunit.xml`, it will over-ride this.

Your `phpunit.xml` should exclude the `system` folder, as well as any `vendor` or `ThirdParty` folders, if you are unit testing your application.

### The Test Class

In order to take advantage of the additional tools provided, your tests must extend `CIUnitTestCase`. All tests are expected to be located in the **tests/app** directory by default.

To test a new library, **Foo**, you would create a new file at **tests/app/Libraries/FooTest.php**:

```
<?php namespace App\Libraries;

use CodeIgniter\Test\CIUnitTestCase;

class FooTest extends CIUnitTestCase
{
    public function testFooNotBar()
    {
        . . .
    }
}
```

To test one of your models, you might end up with something like this in `tests/app/Models/OneOfMyModelsTest.php`:

```
<?php namespace App\Models;

use CodeIgniter\Test\CIUnitTestCase;

class OneOfMyModelsTest extends CIUnitTestCase
{
    public function testFooNotBar()
    {
        . . .
    }
}
```

You can create any directory structure that fits your testing style/needs. When namespacing the test classes, remember that the **app** directory is the root of the `App` namespace, so any classes you use must have the correct namespace relative to `App`.

---

**注解:** Namespaces are not strictly required for test classes, but they are helpful to ensure no class names collide.

---

When testing database results, you must use the CIDatabaseTestClass class.

### Additional Assertions

`CIUnitTestCase` provides additional unit testing assertions that you might find useful.

**assertLogged($level, $expectedMessage)**

Ensure that something you expected to be logged actually was:

```
$config = new LoggerConfig();
$logger = new Logger($config);
```

(下页继续)

(续上页)

```
... do something that you expect a log entry from
$logger->log('error', "That's no moon");

$this->assertLogged('error', "That's no moon");
```

**assertEventTriggered($eventName)**

Ensure that an event you expected to be triggered actually was:

```
Events::on('foo', function($arg) use(&$result) {
    $result = $arg;
});

Events::trigger('foo', 'bar');

$this->assertEventTriggered('foo');
```

**assertHeaderEmitted($header, $ignoreCase=false)**

Ensure that a header or cookie was actually emitted:

```
$response->setCookie('foo', 'bar');

ob_start();
$this->response->send();
$output = ob_get_clean(); // in case you want to check the actual body

$this->assertHeaderEmitted("Set-Cookie: foo=bar");
```

Note: the test case with this should be run as a separate process in PHPunit.

**assertHeaderNotEmitted($header, $ignoreCase=false)**

Ensure that a header or cookie was not emitted:

```
$response->setCookie('foo', 'bar');

ob_start();
$this->response->send();
$output = ob_get_clean(); // in case you want to check the actual body

$this->assertHeaderNotEmitted("Set-Cookie: banana");
```

Note: the test case with this should be run as a separate process in PHPunit.

**assertCloseEnough($expected, $actual, $message='' , $tolerance=1)**

For extended execution time testing, tests that the absolute difference between expected and actual time is within the prescribed tolerance.:

```
$timer = new Timer();
$timer->start('longjohn', strtotime('-11 minutes'));
$this->assertCloseEnough(11 * 60, $timer->getElapsedTime('longjohn'));
```

The above test will allow the actual time to be either 660 or 661 seconds.

**assertCloseEnoughString($expected, $actual, $message='' , $tolerance=1)**

For extended execution time testing, tests that the absolute difference between expected and actual time, formatted as strings, is within the prescribed tolerance.:

```
$timer = new Timer();
$timer->start('longjohn', strtotime('-11 minutes'));
$this->assertCloseEnoughString(11 * 60, $timer->getElapsedTime('longjohn
↪'));
```

The above test will allow the actual time to be either 660 or 661 seconds.

### Accessing Protected/Private Properties

When testing, you can use the following setter and getter methods to access protected and private methods and properties in the classes that you are testing.

**getPrivateMethodInvoker($instance, $method)**

Enables you to call private methods from outside the class. This returns a function that can be called. The first parameter is an instance of the class to test. The second parameter is the name of the method you want to call.

```
// Create an instance of the class to test
$obj = new Foo();

// Get the invoker for the 'privateMethod' method.
    $method = $this->getPrivateMethodInvoker($obj, 'privateMethod');

// Test the results
    $this->assertEquals('bar', $method('param1', 'param2'));
```

**getPrivateProperty($instance, $property)**

Retrieves the value of a private/protected class property from an instance of a class. The first parameter is an instance of the class to test. The second parameter is the name of the property.

```
// Create an instance of the class to test
$obj = new Foo();

// Test the value
$this->assertEquals('bar', $this->getPrivateProperty($obj, 'baz'));
```

#### setPrivateProperty($instance, $property, $value)

Set a protected value within a class instance. The first parameter is an instance of the class to test. The second parameter is the name of the property to set the value of. The third parameter is the value to set it to:

```
// Create an instance of the class to test
$obj = new Foo();

// Set the value
$this->setPrivateProperty($obj, 'baz', 'oops!');

// Do normal testing...
```

### Mocking Services

You will often find that you need to mock one of the services defined in **app/Config/Services.php** to limit your tests to only the code in question, while simulating various responses from the services. This is especially true when testing controllers and other integration testing. The **Services** class provides two methods to make this simple: `injectMock()`, and `reset()`.

#### injectMock()

This method allows you to define the exact instance that will be returned by the Services class. You can use this to set properties of a service so that it behaves in a certain way, or replace a service with a mocked class.

```
public function testSomething()
{
    $curlrequest = $this->getMockBuilder('CodeIgniter\HTTP\CURLRequest')
                        ->setMethods(['request'])
                        ->getMock();
    Services::injectMock('curlrequest', $curlrequest);

    // Do normal testing here....
}
```

The first parameter is the service that you are replacing. The name must match the function name in the Services class exactly. The second parameter is the instance to replace it with.

#### reset()

Removes all mocked classes from the Services class, bringing it back to its original state.

### Stream Filters

**CITestStreamFilter** provides an alternate to these helper methods.

You may need to test things that are difficult to test. Sometimes, capturing a stream, like PHP' s own STDOUT, or STDERR, might be helpful. The `CITestStreamFilter` helps you capture the output from the stream of your choice.

An example demonstrating this inside one of your test cases:

```
public function setUp()
{
    CITestStreamFilter::$buffer = '';
    $this->stream_filter = stream_filter_append(STDOUT,
↪'CITestStreamFilter');
}

public function tearDown()
{
    stream_filter_remove($this->stream_filter);
}

public function testSomeOutput()
{
    CLI::write('first.');
    $expected = "first.\n";
    $this->assertEquals($expected, CITestStreamFilter::$buffer);
}
```

### 8.1.2 Testing Your Database



#### The Test Class

In order to take advantage of the built-in database tools that CodeIgniter provides for testing, your tests must extend `CIDatabaseTestCase`:

```
<?php namespace App\Database;

use CodeIgniter\Test\CIDatabaseTestCase;

class MyTests extends CIDatabaseTestCase
```

(下页继续)

(续上页)

```
{
    . . .
}
```

Because special functionality executed during the `setUp()` and `tearDown()` phases, you must ensure that you call the parent' s methods if you need to use those methods, otherwise you will lose much of the functionality described here:

```
<?php namespace App\Database;

use CodeIgniter\Test\CIDatabaseTestCase;

class MyTests extends CIDatabaseTestCase
{
    public function setUp()
    {
        parent::setUp();

        // Do something here....
    }

    public function tearDown()
    {
        parent::tearDown();

        // Do something here....
    }
}
```

### Setting Up a Test Database

When running database tests, you need to provide a database that can be used during testing. Instead of using the PHPUnit built-in database features, the framework provides tools specific to CodeIgniter. The first step is to ensure that you have set up a `tests` database group in **app/Config/Database.php**. This specifies a database connection that is only used while running tests, to keep your other data safe.

If you have multiple developers on your team, you will likely want to keep your credentials stored in the **.env** file. To do so, edit the file to ensure the following lines are present and have the correct information:

```
database.tests.dbdriver = 'MySQLi';
database.tests.username = 'root';
database.tests.password = '';
database.tests.database = '';
```

### Migrations and Seeds

When running tests, you need to ensure that your database has the correct schema set up and that it is in a known state for every test. You can use migrations and seeds to set up your database, by adding a couple of class properties to your test.

```
<?php namespace App\Database;

use CodeIgniter\Test\CIDatabaseTestCase;

class MyTests extends\CIDatabaseTestCase
{
    protected $refresh  = true;
    protected $seed     = 'TestSeeder';
    protected $basePath = 'path/to/database/files';
}
```

**$refresh**

This boolean value determines whether the database is completely refreshed before every test. If true, all migrations are rolled back to version 0, then the database is migrated to the latest available migration.

**$seed**

If present and not empty, this specifies the name of a Seed file that is used to populate the database with test data prior to every test running.

**$basePath**

By default, CodeIgniter will look in **tests/_support/Database/Seeds** to locate the seeds that it should run during testing. You can change this directores by specifying the `$basePath` property. This should not include the **seeds** directory, but the path to the single directory that holds the sub-directory.

**$namespace**

By default, CodeIgniter will look in **tests/_support/DatabaseTestMigrations/Database/Migrat** to locate the migrations that it should run during testing. You can change this location by specifying a new namespace in the `$namespace` properties. This should not include the **Database/Migrations** path, just the base namespace.

### Helper Methods

The **CIDatabaseTestCase** class provides several helper methods to aid in testing your database.

**seed($name)**

Allows you to manually load a Seed into the database. The only parameter is the name of the seed to run. The seed must be present within the path specified in `$basePath`.

### dontSeeInDatabase($table, $criteria)

Asserts that a row with criteria matching the key/value pairs in `$criteria` DOES NOT exist in the database.

```
$criteria = [
    'email'  => 'joe@example.com',
    'active' => 1
];
$this->dontSeeInDatabase('users', $criteria);
```

### seeInDatabase($table, $criteria)

Asserts that a row with criteria matching the key/value pairs in `$criteria` DOES exist in the database.

```
$criteria = [
    'email'  => 'joe@example.com',
    'active' => 1
];
$this->seeInDatabase('users', $criteria);
```

### grabFromDatabase($table, $column, $criteria)

Returns the value of `$column` from the specified table where the row matches `$criteria`. If more than one row is found, it will only test against the first one.

```
$username = $this->grabFromDatabase('users', 'username', ['email' =>
↪'joe@example.com']);
```

### hasInDatabase($table, $data)

Inserts a new row into the database. This row is removed after the current test runs. `$data` is an associative array with the data to insert into the table.

```
$data = [
    'email' => 'joe@example.com',
    'name'  => 'Joe Cool'
];
$this->hasInDatabase('users', $data);
```

### seeNumRecords($expected, $table, $criteria)

Asserts that a number of matching rows are found in the database that match `$criteria`.

```
$criteria = [
    'active' => 1
];
$this->seeNumRecords(2, 'users', $criteria);
```

### 8.1.3 Testing Controllers

Testing your controllers is made convenient with a couple of new helper classes and traits. When testing controllers, you can execute the code within a controller, without first running through the entire application bootstrap process. Often times, using the Feature Testing tools will be simpler, but this functionality is here in case you need it.

---

**注解:** Because the entire framework has not been bootstrapped, there will be times when you cannot test a controller this way.

---

#### The Helper Trait

You can use either of the base test classes, but you do need to use the `ControllerTester` trait within your tests:

```
<?php namespace CodeIgniter;

use CodeIgniter\Test\ControllerTester;

class TestControllerA extends \CIDatabaseTestCase
{
    use ControllerTester;
}
```

Once the trait has been included, you can start setting up the environment, including the request and response classes, the request body, URI, and more. You specify the controller to use with the `controller()` method, passing in the fully qualified class name of your controller. Finally, call the `execute()` method with the name of the method to run as the parameter:

```
<?php namespace CodeIgniter;

use CodeIgniter\Test\ControllerTester;

class TestControllerA extends \CIDatabaseTestCase
{
    use ControllerTester;

    public function testShowCategories()
    {
        $result = $this->withURI('http://example.com/categories')
                       ->
↪controller(\App\Controllers\ForumController::class)
                       ->execute('showCategories');
```

(下页继续)

(续上页)

```
        $this->assertTrue($result->isOK());
    }
}
```

### Helper Methods

#### controller($class)

Specifies the class name of the controller to test. The first parameter must be a fully qualified class name (i.e. include the namespace):

```
$this->controller(\App\Controllers\ForumController::class);
```

#### execute($method)

Executes the specified method within the controller. The only parameter is the name of the method to run:

```
$results = $this->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');
```

This returns a new helper class that provides a number of routines for checking the response itself. See below for details.

#### withConfig($config)

Allows you to pass in a modified version of **ConfigApp.php** to test with different settings:

```
$config = new Config\App();
$config->appTimezone = 'America/Chicago';

$results = $this->withConfig($config)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');
```

If you do not provide one, the application's App config file will be used.

#### withRequest($request)

Allows you to provide an **IncomingRequest** instance tailored to your testing needs:

```
$request = new CodeIgniter\HTTP\IncomingRequest(new Config\App(), new␣
↪URI('http://example.com'));
$request->setLocale($locale);

$results = $this->withRequest($request)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');
```

If you do not provide one, a new IncomingRequest instance with the default application values will be passed into your controller.

**withResponse($response)**

Allows you to provide a **Response** instance:

```
$response = new CodeIgniter\HTTP\Response(new Config\App());

$results = $this->withResponse($response)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');
```

If you do not provide one, a new Response instance with the default application values will be passed into your controller.

**withLogger($logger)**

Allows you to provide a **Logger** instance:

```
$logger = new CodeIgniter\Log\Handlers\FileHandler();

$results = $this->withResponse($response)
                ->withLogger($logger)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');
```

If you do not provide one, a new Logger instance with the default configuration values will be passed into your controller.

**withURI($uri)**

Allows you to provide a new URI that simulates the URL the client was visiting when this controller was run. This is helpful if you need to check URI segments within your controller. The only parameter is a string representing a valid URI:

```
$results = $this->withURI('http://example.com/forums/categories')
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');
```

It is a good practice to always provide the URI during testing to avoid surprises.

**withBody($body)**

Allows you to provide a custom body for the request. This can be helpful when testing API controllers where you need to set a JSON value as the body. The only parameter is a string that represents the body of the request:

```
$body = json_encode(['foo' => 'bar']);

$results = $this->withBody($body)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');
```

### Checking the Response

When the controller is executed, a new **ControllerResponse** instance will be returned that provides a number of helpful methods, as well as direct access to the Request and Response that were generated.

#### isOK()

This provides a simple check that the response would be considered a "successful" response. This primarily checks that the HTTP status code is within the 200 or 300 ranges:

```
$results = $this->withBody($body)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');

if ($results->isOK())
{
    . . .
}
```

#### isRedirect()

Checks to see if the final response was a redirection of some sort:

```
$results = $this->withBody($body)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');

if ($results->isRedirect())
{
    . . .
}
```

#### request()

You can access the Request object that was generated with this method:

```
$results = $this->withBody($body)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');

$request = $results->request();
```

#### response()

This allows you access to the response object that was generated, if any:

```
$results = $this->withBody($body)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');
```

(下页继续)

(续上页)

```
$response = $results->response();
```

### getBody()

You can access the body of the response that would have been sent to the client with the **getBody()** method. This could be generated HTML, or a JSON response, etc.:

```
$results = $this->withBody($body)
                ->controller(\App\Controllers\ForumController::class)
                ->execute('showCategories');

$body = $results->getBody();
```

## Response Helper methods

The response you get back contains a number of helper methods to inspect the HTML output within the response. These are useful for using within assertions in your tests.

The **see()** method checks the text on the page to see if it exists either by itself, or more specifically within a tag, as specified by type, class, or id:

```
// Check that "Hello World" is on the page
$results->see('Hello World');
// Check that "Hello World" is within an h1 tag
$results->see('Hello World', 'h1');
// Check that "Hello World" is within an element with the "notice"␣
↪class
$results->see('Hello World', '.notice');
// Check that "Hello World" is within an element with id of "title"
$results->see('Hellow World', '#title');
```

The **dontSee()** method is the exact opposite:

```
// Checks that "Hello World" does NOT exist on the page
$results->dontSee('Hello World');
// Checks that "Hellow World" does NOT exist within any h1 tag
$results->dontSee('Hello World', 'h1');
```

The **seeElement()** and **dontSeeElement()** are very similar to the previous methods, but do not look at the values of the elements. Instead, they simply check that the elements exist on the page:

```
// Check that an element with class 'notice' exists
$results->seeElement('.notice');
// Check that an element with id 'title' exists
```

(下页继续)

(续上页)

```
$results->seeElement('#title')
// Verify that an element with id 'title' does NOT exist
$results->dontSeeElement('#title');
```

You can use **seeLink()** to ensure that a link appears on the page with the specified text:

```
// Check that a link exists with 'Upgrade Account' as the text::
$results->seeLink('Upgrade Account');
// Check that a link exists with 'Upgrade Account' as the text, AND a␣
↪class of 'upsell'
$results->seeLink('Upgrade Account', '.upsell');
```

The **seeInField()** method checks for any input tags exist with the name and value:

```
// Check that an input exists named 'user' with the value 'John Snow'
$results->seeInField('user', 'John Snow');
// Check a multi-dimensional input
$results->seeInField('user[name]', 'John Snow');
```

Finally, you can check if a checkbox exists and is checked with the **seeCheckboxIsChecked()** method:

```
// Check if checkbox is checked with class of 'foo'
$results->seeCheckboxIsChecked('.foo');
// Check if checkbox with id of 'bar' is checked
$results->seeCheckboxIsChecked('#bar');
```

### 8.1.4 HTTP Feature Testing

Feature testing allows you to view the results of a single call to your application. This might be returning the results of a single web form, hitting an API endpoint, and more. This is handy because it allows you to test the entire life-cycle of a single request, ensuring that the routing works, the response is the correct format, analyze the results, and more.

- *Session Assertions*
- *Header Assertions*
- *Cookie Assertions*
- *DOM Assertions*
- *Working With JSON*
- *Working With XML*

### The Test Class

Feature testing requires that all of your test classes extend the `CodeIgniter\Test\FeatureTestCase` class. Since this extends CIDatabaseTestCase you must always ensure that `parent::setUp()` and `parent::tearDown()` are called before you take your actions.

```
<?php namespace App;

use CodeIgniter\Test\FeatureTestCase;

class TestFoo extends FeatureTestCase
{
    public function setUp()
    {
        parent::setUp();
    }

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

### Requesting A Page

Essentially, the FeatureTestCase simply allows you to call an endpoint on your application and get the results back. to do this, you use the `call()` method. The first parameter is the HTTP method to use (most frequently either GET or POST). The second parameter is the path on your site to test. The third parameter accepts an array that is used to populate the superglobal variables for the HTTP verb you are using. So, a method of **GET** would have the **$_GET** variable populated, while a **post** request would have the **$_POST** array populated.

```
// Get a simple page
$result = $this->call('get', site_url());

// Submit a form
$result = $this->call('post', site_url('contact'), [
    'name' => 'Fred Flintstone',
    'email' => 'flintyfred@example.com'
]);
```

Shorthand methods for each of the HTTP verbs exist to ease typing and make things clearer:

```
$this->get($path, $params);
$this->post($path, $params);
$this->put($path, $params);
$this->patch($path, $params);
$this->delete($path, $params);
$this->options($path, $params);
```

**注解：** The $params array does not make sense for every HTTP verb, but is included for consistency.

### Setting Different Routes

You can use a custom collection of routes by passing an array of "routes" into the `withRoutes()` method. This will override any existing routes in the system:

```
$routes = [
   [ 'get', 'users', 'UserController::list' ]
];

$result = $this->withRoutes($routes)
    ->get('users');
```

Each of the "routes" is a 3 element array containing the HTTP verb (or "add" for all), the URI to match, and the routing destination.

### Setting Session Values

You can set custom session values to use during a single test with the `withSession()` method. This takes an array of key/value pairs that should exist within the $_SESSION variable when this request is made. This is handy for testing authentication and more.

```
$values = [
    'logged_in' => 123
];

$result = $this->withSession($values)
    ->get('admin');
```

### Bypassing Events

Events are handy to use in your application, but can be problematic during testing. Especially events that are used to send out emails. You can tell the system to skip any event handling with the `skipEvents()` method:

```
$result = $this->skipEvents()
    ->post('users', $userInfo);
```

### Testing the Response

Once you' ve performed a `call()` and have results, there are a number of new assertions that you can use in your tests.

---

**注解:** The Response object is publicly available at `$result->response`. You can use that instance to perform other assertions against, if needed.

---

### Checking Response Status

#### isOK()

Returns a boolean true/false based on whether the response is perceived to be "ok". This is primarily determined by a response status code in the 200 or 300' s.

```
if ($result->isOK())
{
    ...
}
```

#### assertOK()

This assertion simply uses the **isOK()** method to test a response.

```
$this->assertOK();
```

#### isRedirect()

Returns a boolean true/false based on whether the response is a redirected response.

```
if ($result->isRedirect())
{
    ...
}
```

### assertRedirect()

Asserts that the Response is an instance of RedirectResponse.

```
$this->assertRedirect();
```

### assertStatus(int $code)

Asserts that the HTTP status code returned matches $code.

```
$this->assertStatus(403);
```

## Session Assertions

### assertSessionHas(string $key, $value = null)

Asserts that a value exists in the resulting session. If $value is passed, will also assert that the variable' s value matches what was specified.

```
$this->assertSessionHas('logged_in', 123);
```

### assertSessionMissing(string $key)

Asserts that the resulting session does not include the specified $key.

```
$this->assertSessionMissin('logged_in');
```

## Header Assertions

### assertHeader(string $key, $value = null)

Asserts that a header named **$key** exists in the response. If **$value** is not empty, will also assert that the values match.

```
$this->assertHeader('Content-Type', 'text/html');
```

### assertHeaderMissing(string $key)

Asserts that a header name **$key** does not exist in the response.

```
$this->assertHeader('Accepts');
```

### Cookie Assertions

**assertCookie(string $key, $value = null, string $prefix = '')**

Asserts that a cookie named **$key** exists in the response. If **$value** is not empty, will also assert that the values match. You can set the cookie prefix, if needed, by passing it in as the third parameter.

```
$this->assertCookie('foo', 'bar');
```

**assertCookieMissing(string $key)**

Asserts that a cookie named **$key** does not exist in the response.

```
$this->assertCookieMissing('ci_session');
```

**assertCookieExpired(string $key, string $prefix = '')**

Asserts that a cookie named **$key** exists, but has expired. You can set the cookie prefix, if needed, by passing it in as the second parameter.

```
$this->assertCookieExpired('foo');
```

### DOM Assertions

You can perform tests to see if specific elements/text/etc exist with the body of the response with the following assertions.

**assertSee(string $search = null, string $element = null)**

Asserts that text/HTML is on the page, either by itself or - more specifically - within a tag, as specified by type, class, or id:

```
// Check that "Hello World" is on the page
$this->assertSee('Hello World');
// Check that "Hello World" is within an h1 tag
$this->assertSee('Hello World', 'h1');
// Check that "Hello World" is within an element with the "notice"␣
↪class
$this->assertSee('Hello World', '.notice');
// Check that "Hello World" is within an element with id of "title"
$this->assertSee('Hellow World', '#title');
```

**assertDontSee(string $search = null, string $element = null)**

Asserts the exact opposite of the **assertSee()** method:

```
// Checks that "Hello World" does NOT exist on the page
$results->dontSee('Hello World');
```

(下页继续)

(续上页)

```
// Checks that "Hello World" does NOT exist within any h1 tag
$results->dontSee('Hello World', 'h1');
```

#### assertSeeElement(string $search)

Similar to **assertSee()**, however this only checks for an existing element. It does not check for specific text:

```
// Check that an element with class 'notice' exists
$results->seeElement('.notice');
// Check that an element with id 'title' exists
$results->seeElement('#title')
```

#### assertDontSeeElement(string $search)

Similar to **assertSee()**, however this only checks for an existing element that is missing. It does not check for specific text:

```
// Verify that an element with id 'title' does NOT exist
$results->dontSeeElement('#title');
```

#### assertSeeLink(string $text, string $details=null)

Asserts that an anchor tag is found with matching **$text** as the body of the tag:

```
// Check that a link exists with 'Upgrade Account' as the text::
$results->seeLink('Upgrade Account');
// Check that a link exists with 'Upgrade Account' as the text, AND a␣
↪class of 'upsell'
$results->seeLink('Upgrade Account', '.upsell');
```

#### assertSeeInField(string $field, string $value=null)

Asserts that an input tag exists with the name and value:

```
// Check that an input exists named 'user' with the value 'John Snow'
$results->seeInField('user', 'John Snow');
// Check a multi-dimensional input
$results->seeInField('user[name]', 'John Snow');
```

### Working With JSON

Responses will frequently contain JSON responses, especially when working with API methods. The following methods can help to test the responses.

#### getJSON()

This method will return the body of the response as a JSON string:

```
// Response body is this:
['foo' => 'bar']

$json = $result->getJSON();

// $json is this:
{
    "foo": "bar"
}
```

**注解：** Be aware that the JSON string will be pretty-printed in the result.

#### assertJSONFragment(array $fragment)

Asserts that $fragment is found within the JSON response. It does not need to match the entire JSON value.

```
// Response body is this:
[
    'config' => ['key-a', 'key-b']
]

// Is true
$this->assertJSONFragment(['config' => ['key-a']);
```

**注解：** This simply uses phpUnit's own assertArraySubset() method to do the comparison.

#### assertJSONExact($test)

Similar to **assertJSONFragment()**, but checks the entire JSON response to ensure exact matches.

### Working With XML

#### getXML()

If your application returns XML, you can retrieve it through this method.

## 8.1.5 基准测试类

CodeIgniter 提供了两个独立的工具来帮助你对代码进行基准测试，并测试不同的选项：Timer 和 Iterator。Timer 允许你轻松计算脚本执行中两点之间的时间。迭代器允许你

设置多个变量并运行这些测试，记录性能和内存统计信息，以帮助你确定哪个版本是最佳的。

Timer 类始终处于活动状态，从框架被调用的那一刻开始，直到发送输出到用户之前，才能使整个系统执行的时间非常准确。



## 使用定时器

使用 Timer，你可以测量执行应用程序的两个时刻之间的时间。这样可以轻松测量应用程序的不同方面的性能。所有测量都是使用 `start()` 和 `stop()` 方法完成的。

该 `start()` 方法采用单个参数：此定时器的名称。你可以使用任何字符串作为计时器的名称。它仅用于你以后参考以了解哪个测量是:

```
$benchmark = \Config\Services::timer();
$benchmark->start('render view');
```

**该 `stop()` 方法将要停止的计时器的名称作为唯一的参数，也是::** $benchmark->stop( ‘render view’ );

该名称不区分大小写，但除此之外必须与你在启动计时器时给出的名称相匹配。

或者，你可以使用全局函数 `timer()` 来启动和停止定时器:

```
// Start the timer
timer('render view');
// Stop a running timer,
// if one of this name has been started
timer('render view');
```

## 查看你的基准点

当你的应用程序运行时，你设置的所有定时器都将由 Timer 类收集。它不会自动显示它们。你可以通过调用 `getTimers()` 方法检索所有的计时器。该方法返回一组基准信息，包括开始，结束和持续时间:

```
$timers = $benchmark->getTimers();

// Timers =
array(
        'render view' => array(
                'start' => 1234567890,
                'end' => 1345678920,
                'duration' => 15.4315      // number of seconds
        )
)
```

你可以通过传递要显示的小数位数作为唯一参数来更改计算持续时间的精度。默认值为小数点后面的 4 个数字:

```
$timers = $benchmark->getTimers(6);
```

计时器会自动显示在 Debub 工具栏中。

### 显示执行时间

该 getTimers() 方法将为你的项目中的所有计时器提供原始数据，你可以使用‘getElapsedTime()‘方法检索单个计时器的持续时间（以秒为单位）。第一个参数是要显示的定时器的名称。第二个是要显示的小数位数。默认为 4:

```
echo timer()->getElapsedTime('render view');
// Displays: 0.0234
```

### 使用迭代器

Iterator 是一个简单的工具，旨在让你尝试解决方案中的多个变体，以查看速度差异和不同内存使用模式。你可以添加任何数量的“任务”，以便运行，该类将运行任务数百或数千次以获得更清晰的性能。然后，你的脚本可以检索和使用结果，或显示为 HTML 表格。

### 创建任务运行

任务在 Closures 内定义。任务创建的任何输出将被自动丢弃。它们通过 *add()* 方法添加到 Iterator 类中。第一个参数是您想要引用这个测试的名称; 第二个参数是 Closure，它自己本身:

```
$iterator = new \CodeIgniter\Benchmark\Iterator();

// Add a new task
$iterator->add('single_concat', function()
```

(下页继续)

(续上页)

```
        {
                $str = 'Some basic'.'little'.'string concatenation test.
↪';
        }
);

// Add another task
$iterator->add('double', function($a='little')
        {
                $str = "Some basic {$little} string test.";
        }
);
```

### 运行任务

你一旦添加了要运行的任务，你可以使用 `run()` 方法多次循环任务。默认情况下，它将循环运行 1000 次。这对大多数简单的测试来说可能就足够了，如果你需要运行测试多次，你可以将你希望运行数字作为第一个参数传递值:

```
// Run the tests 3000 times.
$iterator->run(3000);
```

一旦运行，它将返回带有测试结果的 HTML 表格。如果你不希望显示结果，可以通过传递第二个参数为 false:

```
// Don't display the results.
$iterator->run(1000, false);
```

## 8.1.6 调试你的应用

### 取代 var_dump

尽管使用在调试你的应用程序时，XDebug 以及一个优秀的 IDE 是不可或缺的，有时候一个简单的 `var_dump()` 就是你所需要的。CodeIgniter 通过集成了优秀的 Kint 调试工具来将这一过程更为优化。该功能比你常用的工具更为方便，可以提供多种类型的可选数据，类似于时间戳格式化，以颜色的方式展示十六进制码，以便于阅读的方式输出数组数据等等等。

### 启用 Kint

默认情况下，Kint 仅在 **development** and **testing** 环境中启用（开发和测试中）。该操作可以通过环境配置这节中所述的，修改主 **index.php** 文件中的 `$useKint` 值来实现:

```
$useKint = true;
```

### 使用 Kint

#### d()

`d()` 方法用于输出所有其所接受的所有参数，并将其输出到屏幕上，并允许脚本继续执行:

```
d($_SERVER);
```

#### dd()

与 `d()` 等同，除了该方法同时会执行 `dies()` ，导致该请求的后续代码无法执行。

#### trace()

该方法会对于当前执行点提供一个调用栈。以 Kint 独有的方式:

```
trace();
```

更多信息请参阅 Kint 主页.

### 调试工具条

调试工具条提供了对于当前页面请求的快照信息，包括性能测试结果，运行的语句，请求和响应数据等。而这些都在开发实践中证明了其在调试和优化过程中的实用性。

---

**注解:** 调试工具条仍处于构建中，并遗留着几个日后计划实现的特性功能

---

### 启用工具条

工具条在 **除了**生产环境之外的其他环境中默认启用。该功能会在 CI_DEBUG 这个常量被定义且值为正数时显示。这一常量在启用文件（例如 `app/Config/Boot/development.php` 中）定义，并可被修改并决定该功能用于哪个环境。

工具条本身作为一个后置过滤器 所展示。你可以通过将其从 **app/Config/Filters.php** 文件的 `$globals` 属性中移除的方式来将其停用。

### 选择显示内容

CodeIgniter 中装载了多个收集器，正如其名所示，用于收集数据并显示于工具条中。你可以创建自己的收集器来定制化工具条。为了决定哪些收集器显示，我们又回到 **app/Config/Toolbar.php** 这一配置文件:

```
public $collectors = [
        \CodeIgniter\Debug\Toolbar\Collectors\Timers::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Database::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Logs::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Views::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Cache::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Files::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Routes::class,
        \CodeIgniter\Debug\Toolbar\Collectors\Events::class,
];
```

将你不期望显示的收集器注释掉。并通过增加完全命名空间化的类名来增加自定义收集器。这里给定的收集器将影响哪些区块将会显示，以及哪些信息将会在时间线上呈现

---

**注解:** 某些区块，例如数据库和日志，将会仅在含有内容时展示。否则将会被移除以节省显示空间。

---

CodeIgniter 装载的控制器为:

- **Timers** 收集性能测试数据，包括系统和应用的
- **Database** 展示所有数据库连接所执行的查询语句与其运行时间
- **Logs** 所有日志信息将会在这里展示。在持久运行的系统或者是有许多日志项目的系统中，该功能可能会导致内存问题并需要被禁用。
- **Views** 以时间线的方式显示视图加载时间，并在独立区块中显示传递给该视图的所有数据。
- **Cache** 将会显示缓存命中和未命中情况以及执行时间
- **Files** 显示在本次请求中加载的所有文件列表
- **Routes** 显示对于当前路由以及系统中定义的所有路由的信息

- **Events** 显示本次请求中所有加载的事件的列表

### 设置性能测试目标

为了使性能测试器可以收集并展示性能测试数据，你必须使用特定的语法来标记测试点。

请阅读以下信息以设置性能测试基点基准测试类

### 创建自定义收集器

创建自定义收集器是一件简单直接的事情。你可以创建一个完全命名空间标识的类，并继承 CodeIgniter\Debug\Toolbar\Collectors\BaseCollector ，从而自动加载器可以将其定位。该类提供了许多你可以用于重载的方法，并含有四个需要设置的属性，来帮助你决定如何使用收集器:

```
<?php namespace MyNamespace;

use CodeIgniter\Debug\Toolbar\Collectors\BaseCollector;

class MyCollector extends BaseCollector
{
        protected $hasTimeline   = false;

        protected $hasTabContent = false;

        protected $hasVarData    = false;

        protected $title         = '';
}
```

**$hasTimeline** 对于任何想要在工具条的时间线上显示信息的收集器来说，该属性应该被设置为 true 。如果该属性为 true 的话，你需要实现 formatTimelineData() 方法以格式化并返回需要显示的数据。

**$hasTabContent** 对于任何想要拥有自定义标签的收集器来说，该属性应该被设置为 true 。如果该属性为 true 的话，你需要提供 $title 值，并实现 display() 方法以渲染标签页内容。如果你需要在标签标题右侧显示额外的信息的话，需要实现 getTitleDetails() 方法。

**$hasVarData** 如果该收集器需要为 变量标签页增加额外数据的话，该值应被设为 true 。如果该值为 true ，你需要实现 getVarData() 方法。

**$title** 在展开的标签页上显示

### 显示工具条标签

为了显示一个工具条标签，你必须:

1. 将需要同时显示在工具条标题和标签头部的文本赋值给 $title .
2. 将 $hasTabContent 属性设置为 true.
3. 实现 display() 方法.
4. 也可以选择性地实现 getTitleDetails() 方法.

display() 方法创建了标签内部显示的 HTML 内容。由于标签的标题会自动交由工具条来处理，因此该方法不会影响它。这一方法会返回一个 HTML 字符串。

getTitleDetails() 方法会返回一个用于显示在标签页标题右侧的字符串，该方法可用于更多额外的概览信息。例如，在数据库标签页上显示所有连接所执行的查询数，以及在文件标签页上显示打开的文件总个数等。

### 提供时间线数据

为了提供在时间线上展示的数据，你必须:

1. 将 $hasTimeline 变量设为 true.
2. 实现 formatTimelineData() 方法.

formatTimelineData() 方法必须返回一个以时间线可用的格式的数组，其中以正确的方式排序并返回正确的信息。内层数据必须包含以下信息:

```
$data[] = [
        'name'      => '',      // 在时间线左侧显示的名字
        'component' => '',      // 在时间线中间列出的部件名
        'start'     => 0.00,    // 开始时间，例如 microtime(true)
        'duration'  => 0.00     // 持续时间，例如 mircrotime(true) -␣
↪microtime(true)
];
```

### 提供变量

为了将数据加入到变量标签页中，你必须:

1. 将 $hasVarData 变量设为 true
2. 实现 getVarData() 方法。

getVarData() 方法应当返回一个需要显示的以键值对格式的数组外层数组的键为变量标签页的标签名:

```
$data = [
        'section 1' => [
            'foo' => 'bar',
            'bar' => 'baz'
        ],
        'section 2' => [
```

(下页继续)

(续上页)

```
            'foo' => 'bar',
            'bar' => 'baz'
        ]
];
```

## 8.2 命令行用法

CodeIgniter 4 也可以用于命令行程序。

### 8.2.1 通过 CLI 方式运行

除了通过在浏览器中输入 URL 的方式访问我们的应用程序 Controllers 我们还可以通过命令行（CLI）的方式调用程序。



#### 什么是 CLI ？

命令行是一个基于文本的与计算机交互方式。更多的内容，可以通过 维基百科的文章 了解。

#### 为什么要通过命令行的方式运行？

对于 CodeIgniter 而言，有很多理由需要你使用命令行。但他们并非显而易见。

- 在使用 *wget* 或者 *curl* 的方式执行你的定时脚本。
- 通过获取`is_cli()` 的返回值，使你的定制脚本无法通过 URL 访问。
- 编写交互式的“任务”，比如一些需要设置权限，修改缓存文件夹，执行备份等操作。
- 和其他语言编写的其他应用程度交互，比如：一个随机的 C++ 脚本可以通过调用一个命令的方式在你编写的模块中执行。

### 让我们尝试一下: Hello World!

首先我们来新建一个简单的控制器，这样你就可以看到他的行为。使用你的编辑器，新建一个名为 Tools.php 的文件，并在文件中写入如下代码:

```
<?php
class Tools extends \CodeIgniter\Controller {

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

然后将这个文件保存在 **application/Controllers/** 目录下。

通常你会使用如下的 URL 访问你的网站:

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

然而，我们现在要打开 Mac/Linux 下的 Terminal 或者在 Windows 下点击运行并输入 “cmd” 之后进入我们 CodeIgniter 项目的 web 根目录，并执行以下命令:

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

如果你的操作正确，你将会看到这个输出 *Hello World!*

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

我们可以在这里像传入 URL 参数一样，传入一个参数。“John Smith” 这个参数作为输入得到的的输出如下:

```
Hello John Smith!
```

### 这里是基础!

简而言之，就是我们要知道命令行上的控制器。需要记住的是，这是一个正常的控制器，所以路由和 `_remap()` 都是正常运作的。

但是，CodeIgniter 提供了额外的工具，可以是更加轻松地创建 CLI 可访问的脚本：包括 CLI-only 路由和一个帮助你使用 CLI-only 工具的库。

### CLI-Only 路由

在 **Routes.php** 文件中你可以像创建其他路由的方式轻松新建只能通过 CLI 方式访问的路由，这些路由并不是使用类似 `get()` 、`post()` ，或者其他类似的方法，在这里你需要使用 `cli()` 方法:

```
$routes->cli('tools/message/(:segment)', 'Tools::message/$1');
```

更多信息，可以查看这里 Routes 。

#### CLI 库

CLI 库让我们的 CLI 工作变得简单。它提供了简单的方法然我们将多种颜色的文本输出在终端上。它还可以让你给用户输出提示信息，构建出一个更加智能的工具。

更多信息，可以查看这里 CLI Library 。

### 8.2.2 自定义 CLI 命令

使用 CLI 命令和使用路由等其他模块一样的轻松愉快，也有一些不同的地方，这就是 CLI 命令的由来。CLI 命令是不需要为其定义路由的简单类，可以让开发人员更简单的创建一些工具。例如，数据库迁移、数据库填充、检查计划任务（cronjob）状态，甚至为公司做一个自定义代码生成器。



#### 运行命令

命令是在项目根目录中通过命令行来执行的。在 **/app** 和 **/system** 同级目录中，提供了一个自定义脚本 **spark** 用于运行任何 CLI 命令:

```
> php spark
```

在不指定命令名称的情况下调用命令，会显示帮助页面并展示出所有可用的命令。你应该将命令名称作为运行该命令的第一个参数传递:

```
> php spark migrate
```

一些命令需要使用到更多的参数，你需要直接写在后面并以空格分隔开:

```
> php spark db:seed DevUserSeeder
```

对于 CodeIgniter 提供的命令，如果不提供必需的参数，则将提示你输入正确运行所需的参数:

```
> php spark migrate:version
> Version?
```

### 使用帮助命令

你可以使用 help 命令来获得 CLI 帮助:

```
> php spark help db:seed
```

### 创建新的命令

你可以非常容易的创建新的命令，并在开发中使用。每个类都必须独立成一个文件，且必须继承自 `CodeIgniter\CLI\BaseCommand`，同时需要实现 `run()` 方法。

为了在 CLI 命令中列出你的命令并添加相应帮助功能，需要使用到以下属性:

- ($group): 一个字符串，用于描述列出命令时命令所属的组。例如（数据库）
- ($name): 表示命令名称的字符串。例如（migrate:create）
- ($description): 描述命令的字符串。例如（创建一个新的迁移文件）
- ($usage): 描述命令用法的字符串。例如（migrate:create [migration_name] [选项]）
- ($arguments): 描述每个命令参数的字符串数组。例如（' migration_name' => '迁移文件名'）
- ($options): 描述每个命令选项的字符串数组。例如（' -n' => '设置迁移命名空间'）

**帮助描述将根据以上参数自动生成**。

### 文件位置

命令必须放在名为 **Commands** 的目录中。但是，该目录可以位于自动加载器 可以找到的任何位置。该目录可以在 **/app/Commands** 中，或者一个你用来放置所有命令的目录，例如 **Acme/Commands** 。

---

**注解:** 执行命令时，将加载完整的 CodeIgniter CLI 环境，从而可以获取环境信息，路径信息以及编写控制器时所用到的任何工具。

---

### 命令示例

让我们逐步通过一个示例命令来演示，这个命令的功能是报告关于应用程序自身的一些基本信息。首先创建一个 **/app/Commands/AppInfo.php** 文件，代码如下:

```
<?php namespace App\Commands;

use CodeIgniter\CLI\BaseCommand;
use CodeIgniter\CLI\CLI;

class AppInfo extends BaseCommand
{
    protected $group       = 'demo';
    protected $name        = 'app:info';
    protected $description = ' 显示基本应用信息';

    public function run(array $params)
    {

    }
}
```

如果运行 **list** 命令，你将在其自己的 demo 组下看到新命令。如果你仔细观察，就会很容易的发现它们是如何工作的。$group 属性说明他所在的组，及存在的所有命令，并会指出他在哪个列表之下。

$name 属性是命令用来被调用的名字，仅有的要求是不能包含空格，并且所有字符必须是命令行有效字符。按照惯例命令行都是小写，通过使用命名空间的冒号，可以规避掉重名的风险。

最后一个属性 $description 是在 **list** 命令中显示的一条描述信息，用来描述命令的用途。

### run()

run() 方法在命令运行时会被调用，$params 数组是紧跟在命令名之后的 CLI 参数列表，例如下面的 CLI 内容:

```
> php spark foo bar baz
```

**foo** 是命令名称，$params 数组将是:

```
$params = ['bar', 'baz'];
```

This can also be accessed through the *CLI* library, but this already has your command removed from the string. These parameters can be used to customize how your scripts behave.

在我们的 demo 命令中，run 方法类似如下写法:

```
public function run(array $params)
{
    CLI::write('PHP Version: '. CLI::color(phpversion(), 'yellow'));
    CLI::write('CI Version: '. CLI::color(CodeIgniter::CI_VERSION,
↪'yellow'));
    CLI::write('APPPATH: '. CLI::color(APPPATH, 'yellow'));
    CLI::write('SYSTEMPATH: '. CLI::color(SYSTEMPATH, 'yellow'));
    CLI::write('ROOTPATH: '. CLI::color(ROOTPATH, 'yellow'));
    CLI::write('Included files: '. CLI::color(count(get_included_
↪files()), 'yellow'));
}
```

### BaseCommand

所有命令必须继承自 BaseCommand 类，它拥有一些很有用的方法，在创建自己的命令时应熟悉这些方法。在 **$this->logger** 上也有一个*Logger* 。

**CodeIgniter\CLI\BaseCommand**

**call**(*string $command*[, *array $params=[]*])

**参数**

- **$command** (*string*) – 要调用的另一个命令的名称。
- **$params** (*array*) – 使该命令可用的附加 CLI 参数。

此方法使你可以在当前命令执行期间运行其他命令:

```
$this->call('command_one');
$this->call('command_two', $params);
```

**showError**(*Exception $e*)

**参数**

- **$e** (*Exception*) – 用于错误报告的 Exception。

一种方便的方法，用于向 CLI 保持一致且清晰的错误输出:

```
try
{
    . . .
}
catch (\Exception $e)
{
    $this->showError($e);
}
```

**showHelp()**
显示命令帮助的方法：（用法，参数，描述，选项）

**getPad**(*$array*, *$pad*)

**参数**

- **$array** (*array*) – $key => $value 数组
- **$pad** (*integer*) – 填充空格数

一个为 $key => $value 数组输出计算填充的方法。填充可用于在 CLI 中输出格式化的表格：

```
$pad = $this->getPad($this->options, 6);
foreach ($this->options as $option => $description)
{
        CLI::write($tab . CLI::color(str_pad($option, $pad),
↪'green') . $description, 'yellow');
}

// 输出应该会像这样
-n                  设置迁移命名空间
-r                  覆盖文件
```

### 8.2.3 CLI Library

CodeIgniter 的 CLI 库，让创建命令行交互脚本变得简单。其中包括：

- 为用户提供更多信息
- 在终端上打印彩色文本
- Beeping (be nice!)
- 在长任务中显示进度条
- 让长文本行适应窗口



#### 初始化类

你不需要创建 CLI 库的实例，因为它的所有方法都是静态方法。你只需要确保你的控制器可以通过 use 声明找到它：

```
<?php namespace App\Controllers;

use CodeIgniter\CLI\CLI;

class MyController extends \CodeIgniter\Controller
{
        . . .
}
```

首次加载该文件时，这个类会自动初始化。

### 获取用户输入

有时你需要询问用户更多的信息。他们可能没有提供可选的命令行参数，或者脚本遇到了存在的文件，在覆写前需要进行确认。这时使用 prompt() 方法处理

你可以提供一个问题作为方法的第一个参数：

```
$color = CLI::prompt('What is your favorite color?');
```

你可以提供一个默认的答案作为方法的第二个参数。如果用户没有任何输入，只是按下 Enter 键，则将使用该默认答案：

```
$color = CLI::prompt('What is your favorite color?', 'blue');
```

你可以向第二个参数传入允许答案的数组，以限制可以接受的答案：

```
$overwrite = CLI::prompt('File exists. Overwrite?', ['y','n']);
```

最后你可以将验证规则作为第三个参数，以限制输入的答案：

```
$email = CLI::prompt('What is your email?', null, 'required|valid_email
↪');
```

### 提供反馈

#### write()

多个方法可以用来向用户提供反馈。它可以像更新单个状态一样简单，也可以包装复杂的信息表到用户终端。其核心是 write() 方法，该方法将要输出的字符串作为第一个参数：

```
CLI::write('The rain in Spain falls mainly on the plains.');
```

你可以输入颜色名称作为第二个参数来更改文本的颜色：

```
CLI::write('File created.', 'green');
```

你可以向第三个参数输入颜色名称来设置背景颜色。这可以用于按状态区分消息，或使用其他颜色创建“标题”：

```
CLI::write('File overwritten.', 'light_red', 'dark_gray');
```

可以使用以下前景色：

- black
- dark_gray
- blue
- dark_blue
- light_blue
- green
- light_green
- cyan
- light_cyan
- red
- light_red
- purple
- light_purple
- light_yellow
- yellow
- light_gray
- white

少数可以用作背景色：

- black
- blue
- green
- cyan
- red
- yellow
- light_gray
- magenta

**print()**

Print 方法的功能和 write 相同，不同的是它不会在行前或者行尾强制换行。它将信息打印到当前屏幕光标所在的位置上，这让你可以在不同的代码位置，打印信息到屏幕的同一行上。当你显示状态，执行某些操作后在同一行打印 “Done” 时，这十分有用：

```
for ($i = 0; $i <= 10; $i++)
{
    CLI::print($i);
}
```

### color()

write() 方法将单行文本打印到终端，并打印 EOL 标识符结尾。color() 方法以相同的方式处理文本，不同的是它不会在结尾打印 EOL 标识符。这可以让你在同一行创建多个输出。更常见的用法是在 write() 方法内部使用，以在同一行中显示不同颜色的文本：

```
CLI::write("fileA \t". CLI::color('/path/to/file', 'white'), 'yellow');
```

该示例将打印一行文本到终端。首先用黄色打印 fileA ，接着打印一个制表符，最后用白色打印 /path/to/file。

### error()

如果你需要输出错误信息，则应该使用 error 方法。它会将浅红色的文本写入到 STDERR，而不是像 write() 和 color() 一样写入到 STDOUT。如果你使用脚本监视错误信息，这样就可以只捕获到实际的错误信息，不必从所有信息中进行筛选：

```
CLI::error('Cannot write to file: ' . $file);
```

### wrap()

该方法将获取一个字符串，并开始在当前行开始打印。它将会根据设置的长度对字符串进行包装，每行只显示设置的长度的内容。他可以用来显示带有说明的显示列表，避免过多内容显示在一行内，影响阅读：

```
CLI::color("task1\t", 'yellow');
CLI::wrap("Some long description goes here that might be longer than the␣
↪current window.");
```

默认情况下，字符串将用终端宽度进行包装。Windows 目前无法提供确定的窗口大小，所以默认使用 80 个字符。如果你希望将宽度设置的更短一些，可以将最大行长度作为第二个参数传递。这将在最接近该长度的单词处断开字符，以避免破坏单词：

```
// Wrap the text at max 20 characters wide
CLI::wrap($description, 20);
```

你会发现需要左边需要一列显示标题、文件或任务，而右边需要一列显示文本和他的说明。默认情况下，这将换行到窗口的左边缘，即不允许信息按列排列。这时你可以用空格来填充第一行之后的每一行，以便让左侧具有清晰的列边缘：

```
// 确定所有标题的最大长度
// 确定左列的宽度
$maxlen = max(array_map('strlen', $titles));

for ($i=0; $i < count($titles); $i++)
{
        CLI::write(
                // 在行的左侧列显示标题
                $titles[$i] . '    ' .
                // 在行的右侧列包装信息
                // 与左侧最宽的内容间隔三个宽度
                CLI::wrap($descriptions[$i], 40, $maxlen + 3)
        );
}
```

将创建以下内容：

```
task1a   Lorem Ipsum is simply dummy
           text of the printing and typesetting
           industry.
task1abc   Lorem Ipsum has been the industry's
           standard dummy text ever since the
```

### newLine()

newLine() 方法向用户显示一个空行，它不需要任何参数：

```
CLI::newLine();
```

### clearScreen()

你可以使用 clearScreen() 方法清除当前窗口。在多数的 Windows 系统中，它将插入 40 行空白行，因为 Windows 不支持该功能。Windows 10 bash 因该能改变这点：

```
CLI::clearScreen();
```

### showProgress()

如果你有一个长时间运行的任务，你希望让用户了解当前的执行进度，则可以使用 showProgress() 方法，它显示以下内容：

```
[####......] 40% Complete
```

该行将设置为动态显示，以获得良好的展示效果。

将当前的步骤作为第一个参数传入，并将总步骤作为第二个参数传入。完成的百分比和显示长度将根据该数字确认。当任务完成后，传递 false 作为第一个参数，进度条将被删除：

```
$totalSteps = count($tasks);
$currStep   = 1;

foreach ($tasks as $task)
{
        CLI::showProgress($currStep++, $totalSteps);
        $task->run();
}

// Done, so erase it...
CLI::showProgress(false);
```

**table()**

```
$thead = ['ID', 'Title', 'Updated At', 'Active'];
$tbody = [
        [7, 'A great item title', '2017-11-15 10:35:02', 1],
        [8, 'Another great item title', '2017-11-16 13:46:54', 0]
];

CLI::table($tbody, $thead);
```

```
+----+--------------------------+---------------------+--------+
| ID | Title                    | Updated At          | Active |
+----+--------------------------+---------------------+--------+
| 7  | A great item title       | 2017-11-16 10:35:02 | 1      |
| 8  | Another great item title | 2017-11-16 13:46:54 | 0      |
+----+--------------------------+---------------------+--------+
```

**wait()**

等待一定的秒数。可以选择显示等待消息，或等待按键：

```
// 等待指定的时间间隔，并显示倒计时信息
CLI::wait($seconds, true);

// 显示等待输入的信息，并等待输入
CLI::wait(0, false);

// 等待指定的时间间隔，不显示任何信息
CLI::wait($seconds, false);
```

### 8.2.4 CLIRequest Class

If a request comes from a command line invocation, the request object is actually a `CLIRequest`. It behaves the same as a *conventional request* but adds some accessor methods for convenience.

#### Additional Accessors

**getSegments()**

Returns an array of the command line arguments deemed to be part of a path:

```
// command line: php index.php users 21 profile -foo bar
echo $request->getSegments();  // ['users', '21', 'profile']
```

**getPath()**

Returns the reconstructed path as a string:

```
// command line: php index.php users 21 profile -foo bar
echo $request->getPath();  // users/21/profile
```

**getOptions()**

Returns an array of the command line arguments deemed to be options:

```
// command line: php index.php users 21 profile -foo bar
echo $request->getOptions();  // ['foo' => 'bar']
```

**getOption($which)**

Returns the value of a specific command line argument deemed to be an option:

```
// command line: php index.php users 21 profile -foo bar
echo $request->getOption('foo');  // bar
echo $request->getOption('notthere'); // NULL
```

**getOptionString()**

Returns the reconstructed command line string for the options:

```
// command line: php index.php users 21 profile -foo bar
echo $request->getOptionPath();  // -foo bar
```

## 8.3 扩展 CodeIgniter

扩展或基于 CodeIgniter 4 构建非常容易。

### 8.3.1 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。但你也可以使用你自己的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 核心的人来说，我们依然提供了替换和扩展核心类的选择。**

**注解：** 变动核心系统类意味着一系列的挑战，所以，请三思后行。

### 系统类列表

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

- ConfigServices
- CodeIgniterAutoloaderAutoloader
- CodeIgniterConfigDotEnv
- CodeIgniterController
- CodeIgniterDebugExceptions
- CodeIgniterDebugTimer
- CodeIgniterEventsEvents
- CodeIgniterHTTPCLIRequest (if launched from command line only)
- CodeIgniterHTTPIncomingRequest (if launched over HTTP)
- CodeIgniterHTTPRequest
- CodeIgniterHTTPResponse
- CodeIgniterHTTPMessage
- CodeIgniterLogLogger
- CodeIgniterLogHandlersBaseHandler
- CodeIgniterLogHandlersFileHandler
- CodeIgniterRouterRouteCollection
- CodeIgniterRouterRouter
- CodeIgniterSecuritySecurity
- CodeIgniterViewView
- CodeIgniterViewEscaper

### 替换核心类

要使用你的系统类替换 CodeIgniter 默认的系统类时，首先确保*Autoloader* 能找到你的类；其次你的新类继承了正确的接口，同时修改*Service* 以保证加载的是你自己的类。

例如，你有一个名为 “ AppLibrariesRouteCollection “ 的新类想要替换系统的核心类，你应该像这样创建你的类:

```
class RouteCollection implements␣
↪\CodeIgniter\Router\RouteCollectionInterface
{

}
```

然后，你应该修改路由文件来加载你自己的类:

```
public static function routes($getShared = false)
{
        if (! $getShared)
        {
                return new \App\Libraries\RouteCollection();
        }

        return self::getSharedInstance('routes');
}
```

### 扩展核心类

如果你需要往一个现有的库里添加一些功能-或许只是添加一两个方法，重写这整个库显然是没必要的。这时更好的通常是对其中的类进行扩展。对类进行扩展与替换掉类几乎相同，除了一点:

- 类的声明必须继承父类。

比如，继承 RouteCollection 这个原生类，你应该这样声明:

```
class RouteCollection extends \CodeIgniter\Router\RouteCollection
{

}
```

如果你需要在类中使用构造器来确保子类继承了父类的构造器:

```
class RouteCollection implements \CodeIgniter\Router\RouteCollection
{
    public function __construct()
    {
        parent::__construct();
    }
}
```

**Tip:** 在你自己的类中，所有与父类方法名相同的函数将会覆盖父类方法，此为”方法覆盖”．这样你就可以充分地改动 CodeIgniter 的核心类。

你若扩展了控制器核心类，则需确保你的新类继承了应用下的控制器类的构造器:

```
class Home extends App\BaseController {

}
```

### 8.3.2 替换通用函数

对于 CodeIgniter 来说，需要在核心类中提前加载许多必需的函数，因此这些函数不应该放入辅助函数中。尽管大多数用户可能永远不会面对这种情况，我们依旧为哪些想要手动修改 CodeIgniter 内核的用户提供了一个选项来修改这些函数。在 App 目录下有一个 Common.php 文件，其中定义的所有函数都会在 system/Common.php 文件所定义的函数版本前生效。同样这也是一个创建在框架里全局有效的函数的好机会。

---

**注解:** 在核心系统类中加入很多复杂成分会带来很多影响，在尝试做这件事之前，请确保你知道自己正在做什么。

---

### 8.3.3 事件

CodeIgniter 事件特性提供了一种方法来修改框架的内部运作流程或功能，而无需修改核心文件的能力。CodeIgniter 遵循着一个特定的流程来运行。但是，在某些情况下，你可能想在执行特定流程时执行某些特定的操作。例如在加载控制器之前或之后立即运行一个特定的脚本。或者在其他的某些位置触发你的脚本。

事件已发布/订阅模式工作，可以在脚本执行过程中的某个时刻触发事件。其他脚本可以通过向 Events 类来注册订阅事件，使它知道在脚本触发事件时该执行什么操作。

#### 启用事件

事件始终处于启用状态，并且全局可用。

#### 定义事件

大多数的事件都定义在 **app/Config/Events.php** 文件中。不过你也可以通过 Events 类的 on() 方法定义事件。第一个参数是事件名称，第二个参数是当触发该事件时执行的操作:

```
use CodeIgniter\Events\Events;

Events::on('pre_system', ['MyClass', 'MyFunction']);
```

在这个例子中，任何时候触发 **pre_controller** 事件，都会创建 MyClass 实例并运行 MyFunction 方法。

第二个参数可以是 PHP 能识别的任何 可调用结构:

```
// 调用 some_function 方法
Events::on('pre_system', 'some_function');

// 调用实例方法
$user = new User();
Events::on('pre_system', [$user, 'some_method']);

// 调用静态方法
Events::on('pre_system', 'SomeClass::someMethod');

// 使用闭包形式
Events::on('pre_system', function(...$params)
{
        . . .
});
```

### 设置执行优先顺序

由于可以将多个方法订阅到一个事件中，因此需要一种方式来定义这些方法的调用顺序。你可以通过传递优先级作为 on() 方法的第三个参数来实现。事件系统将优先执行优先级较低的值，优先级最高的值为 1:

```
Events::on('post_controller_constructor', 'some_function', 25);
```

如果出现相同优先级的情况，那么事件系统将按定义的顺序执行。

---

**注解:** 可以理解为事件系统会根据事件名称分组排序，按第三个参数升序排列，然后依次执行。

---

Codeigniter 内置了三个常量供您使用，仅供参考。你也可以不使用它，但你会发现他们有助于提高可读性:

```
define('EVENT_PRIORITY_LOW', 200);
define('EVENT_PRIORITY_NORMAL', 100);
define('EVENT_PRIORITY_HIGH', 10);
```

排序后，将按顺序执行所有订阅者。如果任意订阅者返回了布尔类型 false，订阅者将停止执行。

### 发布自定义的事件

使用事件系统，你可以轻松创建自己的事件。要使用此功能，只需要调用 **Events** 类的 trigger() 方法即可:

```
\CodeIgniter\Events\Events::trigger('some_event');
```

当然，你也可以为订阅者传递任意数量的参数，订阅者将会按相同的顺序接收参数:

```
\CodeIgniter\Events\Events::trigger('some_events', $foo, $bar, $baz);

Events::on('some_event', function($foo, $bar, $baz) {
        ...
});
```

#### 模拟事件

在测试期间，你可能不希望事件被真正的触发，因为每天发送数百封电子邮件记缓慢又适得其反。你可以告诉 Events 类使用 `simulate()` 方法模拟运行事件。如果为 **true**，那么将跳过所有事件，不过其他的内容都会正常运行:

```
Events::simulate(true);
```

你也可以传递 **false** 停止模拟:

```
Events::simulate(false);
```

#### 事件触发点

以下是 Codeigniter 核心代码中可用的事件触发点列表:

- **pre_system** 系统执行过程中最早被调用。此时，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的流程。
- **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未调用。
- **post_system** 最终数据发送到浏览器之后，系统执行结束时调用。

### 8.3.4 扩展 Controller

CodeIgniter 的核心类 Controller 不应该被修改，但是在 **app/Controllers/BaseController.php** 提供了一个默认的 Controller 扩展。你创建的任何新控制器，都应该继承 `BaseController` 以利用组件预加载和你添加的任何其他功能:

```
<?php namespace App\Controllers;

use CodeIgniter\Controller;

class Home extends BaseController {
```

(下页继续)

(续上页)

```
}
```

### 组件预加载

基础控制器是每次运行项目时，加载你希望使用的任何 helpers, models, libraries, services 等的好位置。Helpers 应该添加到预先提供的 `$helpers` 数组。例如，如果你想要在所有控制器中使用 HTML 和 Text 辅助函数：

```
protected $helpers = ['html', 'text'];
```

其他任何要加载的组件或者要处理的数据，都应该添加到 `initController()` 中。例如，如果你的项目要大量使用 Session 类，那你可以在这里启动它：

```
public function initController(...)
{
        // Do Not Edit This Line
        parent::initController($request, $response, $logger);

        $this->session = \Config\Services::session();
}
```

### 附加方法

基础控制器不可以被路由（系统配置会将它路由到 404 Page Not Found）。作为一项附加的安全措施，你应该将创建的 **所有**新方法声明为 `protected` 或者 `private`，并且只允许继承 `BaseController` 的控制器进行访问。

### 其他配置

你可能会需要多个基础控制器。你可以创建新的基础控制器，只要确保你创建的任何控制器正确继承了基础控制器。例如，你的项目同时有面向普通用户的公共控制器和面向管理员的后台控制器。则你可以让所有公共控制器继承 `BaseController` ，创建一个 `AdminController` 让所有后台控制器来继承。

如果你不想使用基础控制器，则可以通过继承系统控制器来代替：

```
class Home extends \CodeIgniter\Controller
{

}
```

### 8.3.5 鉴权

CodeIgniter 本身不提供一个内部鉴权或认证的类。不过有许多优秀的第三方模块可以提供类似的服务，而且在社区里也有许多资源以帮助你自己实现一个类似的功能。我们鼓励开发者们基于以下原则来共享模块，项目以及框架本身。

#### Recommendations

- 处理登入和登出操作的模块需要在操作成功时触发 `login` 和 `logout` 事件
- 定义了“当前用户”的模块应该定义一个 `user_id()` 方法来返回当前用户的唯一识别符，或者是在不存在当前用户时返回 `null`

### 8.3.6 贡献给 CodeIgniter

CodeIgniter 是一个大众驱动项目并且它接受自大众提供的编码和文档编制贡献。这些贡献将在 Github 的 CodeIgniter4 repository 上以讨论的形式或者以 Pull Requests 形式产生。

讨论是指出一个程序错误最快捷的方式。如果你在 Codeigniter 中找到了程序错误或者文档编制错误，请首先检查一些要事：

- 是否存在一个已经开放的讨论。
- 讨论已经被解决了。(检查开发分支，或者查看关闭的讨论。)
- 你明确的确实要独自解决问题吗?

发布讨论是有帮助而且发出 Pull Request 是一个更好的方式，PR 是基于“Forking”主要的内容并提交到你自己拷贝版本里。

请查看代码库的 贡献给 CodeIgniter4 章节。

#### 支持

请记住 GitHub 决不支持一般使用性的问题！如果将来你在使用 Codeigniter 中有了困难，请去网络论坛寻求帮助代替发表在 forums 上。

如果你不能保证你使用中出现的事情是否正确或者你又发现了一处程序错误，请首先在网络论坛中询问。

#### 安全性

你已经在 CodeIgniter 中找到一个安全问题了吗?

请不要公开揭露你发现的安全问题，但是你要发送邮件给 security@codeigniter.com，或者经由我们 HackerOne 的页面发布它。如果你已经找到了一个濒临崩溃的安全危险，我们很高兴把你的发现放在我们的 ChangeLog 里。

#### 优良的讨论报告贴士

使用有描述的主题原则（例如 parser library chokes on commas）好于含糊不清的主题（例如 your code broke ）。

在报告里计算机物理地址是单独说明的问题。

识别清楚 codeigniter 的版本（例如 3.0 - develop）和你知道的组件（例如 parser library）

阐述你预期将要发生的事或者已经发生的事。包括任何错误的信息和堆栈轨迹。

如果代码程序段能够帮助说明要把短代码程序段考虑在内。使用 pastebin 或者 dropbox 很容易提取更长的代码程序段或者截图————截图并不包含讨论报告自身。本段文字的主旨是设定问题解决的合理终结，直到问题解决或者关闭。

如果你知道如何解决讨论，你要在你自己的 fork & branch 做好解决方案，并且提交堆栈请求（pull request）。上文中的问题报告信息应当是整个报告的一部分。

如果你的讨论报告描述能分步骤的再现问题，那是极好的。如果你在再现问题时能把单元测试考虑在内，那将更好，讨论报告要给任何正在解决问题的人一个更加清楚的目标!

## 8.4 The MIT License (MIT)



## 8.5 Change Logs

### 8.5.1 Version 4.0

Release Date: Not Released

**Next release of CodeIgniter4**

*See all the changes.*

### Version 4.0

Release Date: Not released

**Next alpha release of CodeIgniter4**

The list of changed files follows, with PR numbers shown.

### PRs merged:

### Version 4.0.0

Release Date: February 24, 2020

**4.0 release of CodeIgniter4**

Enhancements:

- Updated welcome view file
- Debug Toolbar now supports dark mode
- New alpha_numeric_punct validation rule
- Kint was updated from the 2.x branch to the latest 3.x branch, with new config file to specify options
- Additional docs for getting started
- CLI fixed to better handle complex arguments
- Improvements to File class
- New model() helper method for easy singleton Models.
- Tests completely reorganized to make app-level tests simpler out of the box.

Repo changes:

The devstarter repo has been deprecated.

We now have an automated changelog generated in the CHANGELOG.md file in the main repo. See it for more details.

### Version 4.0.0-rc.4

Release Date: February 6, 2020

**RC.4 release of CodeIgniter4**

Enhancements:

- Fixed url systems so that it would work when system is served out of subfolders.
- Added required insert ignore support for sqlite3 and mysql.
- Add validation function *is_not_unique*
- Various improvements and cleanup to the Email class

### PRs merged:

- #2527 Update manual.rst
- #2454 Page in the official documentation on ajax requests with iSAJAX() fixes
- #2525 Remove incorrect inline doc type
- #2524 Restore namespace after regress.
- #2523 Replace legacy CI3 constant.
- #2522 Adding Events information in the ‘Upgrading from 3.x to 4.x’ section
- #2518 Fix pager URI to work in subfolders.
- #2516 HTML Helper - Fix attribute type for lists
- #2515 Layout Renderer Fix
- #2513 Typo in userguide “Entity Classes - Business Logic”
- #2511 Database add highlight
- #2509 Revert Renderer section reset
- #2507 Update ordering of search locations for better prioritization.
- #2506 HTTP Response - Fix crash on CSP methods CSP is disabled
- #2504 BaseConnection - Nullable return type in getConnectStart()
- #2502 View Renderer - Reset sections after generating the ouput
- #2501 view_cell call controller on initController method.
- #2499 View Parser - Fix ParsePair() with filter
- #2497 Fix splitQueryPart()
- #2496 Use site_url for RedirectResponse.
- #2495 update toolbar userguide
- #2494 Debug Toolbar - Fix Debugbar-Time header, Render in <head>
- #2493 fix sphinx version.
- #2490 fix. Toolbar init view Error
- #2489 Fix pager
- #2486 Update current_url and previous_url in the docs for View Parser.

- #2485 Typo in user guide “Running via the Command Line”
- #2482 Services request add URI Core System extend support
- #2481 Priority Redirection.
- #2472 ControllerTest should work without URI specified. Fixes #2470
- #2471 Transition from Zend Escaper to Laminas Escaper
- #2462 Fix impossible length for migration table id.
- #2458 Replace *composer install* by *composer require*
- #2450 CRITICAL when $_SESSION is null / Argument 2 passed to dot_array_search() must be []
- #2449 User Guide: Query Builder selectCount - error correction in example
- #2447 Existing File checks (Nowackipawel/patch-69)
- #2446 DB Insert Ignore (Tada5hi/database-feature)
- #2438 Nice array view in debug toolbar
- #2436 Fix Message method reference
- #2433 Inserting through a model should respect all validation rules. Fixes #2384
- #2432 Fix curly brace deprecation in php 7.4
- #2429 fix. safe_mailto multi-byte safe
- #2427 Add $recipients property to ConfigEmail
- #2426 Add hex validation rule, test, Guide
- #2425 fix: Router setDefaultNameSpace can’t worker
- #2422 Don’t show duplicate Date headers when running under PHPs server.
- #2420 Change current_url() to use cloned URI
- #2417 Revise Encryption Service Documentation
- #2416 Add missing closing braces of condition ‘hasError()’
- #2415 Add ‘nullable’ to MySQL field data
- #2413 fix. toolbar file 301
- #2411 fix parse params of plugin
- #2408 Ensure previous_url() gets accurate URI.
- #2407 Fix url helper functions to work when site hosted in subfolders.
- #2406 Fix issue #2391 CodeIgniter::display404errors()
- #2402 Removed pointless isset() check
- #2401 Remove pointless check from conditional

- #2400 Remove redundant check in conditionals
- #2399 Revise Controllers Documentation
- #2398 Edit .htaccess
- #2392 Add validation function *is_not_unique*
- #2389 Confer silent status to nested seeders
- #2388 Fix copypaste command comment
- #2387 Use only digits for migrations order
- #2382 quick fix postgresql insert id
- #2381 Fix: Use of CodeIgniterConfigServices prevents Service overriding
- #2379 Replace null log file extension check
- #2377 Docs Rev: Replacing Core Classes
- #2369 Remove LoggerAwareTrait from Email class
- #2368 Remove log_message from Email::__construct
- #2364 Email config doesn't incorporate .env items
- #2362 Fix SMTP protocol problem
- #2359 Bugfix Model after event data
- #2358 Fix Logger config
- #2356 Fix typo in comments of Services.php
- #2352 Fix method name to 'toDateString()' in Date and Times user guide

### Version 4.0.0-rc.3

Release Date: Oct 19, 2019

**RC.3 release of CodeIgniter4**

Enhancements:

- Beefed up database, session & routing handling.
- Fixed numerous bugs & user guide errata.

App changes:

- New $CSRFHeaderName property in App/Config/App

Message changes:

The list of changed files follows, with PR numbers shown.

- admin/
- **app/**

    - **Config/**
        - App #2272
- public/
- **system/**
    - **Autoloader/**
        - FileLocator #2336
    - **Database/**
        - MySQLi/Forge #2100
        - Postgre/Forge #2100
        - SQLite3/Forge #2100
        - BaseBuilder #2252, 2312
        - Forge $2100
        - Migration #2303
        - MigrationRunner #2303
    - **Debug/**
        - Exceptions #2288
        - **Toolbar/Collectors/**
            - Route #2300
        - Toolbar #2315
        - Views/ #2283
    - **Helpers/**
        - inflector_helper #2296
        - url_helper #2325
    - **HTTP/**
        - CURLRequest #2285, 2305
        - Files/UploadedFile #2123
    - **Language/en/**
        - Encryption #2311
        - RESTful #2311
        - Session #2311
    - **Router/**
        - Exceptions/RedirectException #2338

        * Router #2308, 2338
    - **Security/**
        * Security #2272, 2279
    - **Session/**
        * **Handlers/**
            · DatabaseHandler #2298
            · FileHandler #2298, 2307
            · MemcachedHandler #2298
            · RedisHandler #2298
        * Session #2339
    - **Validation/**
        * Validation #2284, 2341
    - **View/**
        * View #2324
    - CodeIgniter #2338
    - Common #2279
    - Model #2289, 2332
- tests/README.md #2345
- **tests/_support/**
    - **Config/**
        * MockAppConfig #2272
- **tests/system/**
    - **Database/**
        * **Builder/**
            · UpdateTest #2295
        * **Live/**
            · ForgeTest #2100
    - **Helpers/**
        * InflectorHelperTest #2296
        * URLHelperTest #2325
    - **HTTP/**
        * CURLRequestTest #2285

    - **Log/**
        - FileHandlerTest #2346
    - **Security/**
        - SecurityTest #2279
    - **Session/**
        - SessionTest #2339
    - CommonFunctionsTest #2279
- **user_guide_src/**
    - **dbmgmt/**
        - forge #2100
        - migration #2337
    - **general/**
        - common_functions #2279
        - errors #2338
        - modules #2290
    - **helpers/**
        - inflector_helper #2296
    - **incoming/**
        - message #2282
        - restful #2313, 2321, 2333
        - routing #2327
    - **libraries/**
        - curlrequest #2305
        - security #2279
    - **models/**
        - model #2316, 2332
    - **outgoing/**
        - table #2337

### PRs merged:

- #2348 CodeIgniter Foundation gets copyright
- #2346 Fix FilerHandlerTest.php wierdness

- #2345 Tests readme polish
- #2344 Setup vs Set Up
- #2343 User guide minor fixes. Fix class names and code area
- #2341 Simplify Validation::getErrors()
- #2339 Fix Session::get( 'key' ) returns null when value is (int) 0
- #2338 Revert RedirectException change
- #2337 Guide: Minor grammar corrections
- #2336 Correct cleaning of namespaces in Windows
- #2333 Guide: RESTful table formatting
- #2332 Change after methods to use actual data
- #2328 Update Application structure
- #2327 Correct the tourint UG page
- #2325 Fix bug in url_title() function with diacritics
- #2324 Renderer Toolbar Debug Toggle
- #2321 Update RESTful User Guide
- #2316 Add getValidationRules() to model UG page
- #2315 Enhance Toolbar::renderTimeline
- #2313 RESTful User Guide cleanup
- #2312 BaseBuilder variable type fix
- #2311 Convert all language returns to single quote
- #2308 Bugfix extra autoroute slashes
- #2307 Resolve session save handler issue
- #2305 Fix curl debug bug
- #2303 Use DBGroup variable from migration class if defined
- #2300 Routes collector for toolbar should not die when a method name is calculated through _remap
- #2298 fix issue on session_regenerate
- #2296 Add counted() to Inflector Helper
- #2295 Test set() method in Builder class more
- #2290 Fix Code Modules documentation for psr4 namespace configuration
- #2289 Don't restrict model's access to properties in a read-only manner
- #2288 Fix line numbering in Debug/Exceptions class

- #2285 Fix error with Host header for CURLRequest class
- #2284 Fix getErrors() for validation with redirect
- #2283 Hotfix: Rename collectors _*.tpl.php to _*.tpl
- #2282 Fix user guide for Message class
- #2279 Bug in CSRF parameter cleanup
- #2272 Handle X-CSRF-TOKEN - CSRF
- #2252 Batch Update Where Reset
- #2123 WIP fix store() default value bug
- #2100 Added validation on exists database before created for MySQLi

### Version 4.0.0-rc.2

Release Date: Sept 27, 2019

**RC.2 release of CodeIgniter4**

Enhancements:

- **query builder testability simplified with new property, but dropped** method parameters (breaking change)
- database, migrations and sessions beefed up
- numerous smaller bugs corrected

App changes:

- Config/Constants, Paths & some config settings have had changes

Message changes:

- N/A

The list of changed files follows, with PR numbers shown.

- admin/
- **app/**
  - **Config/**
    - Boot/* #2241
    - Constants #2183
    - Paths #2181
- public/
- **system/**
  - **CLI/**

    * BaseCommand #2231
  - **Database/**
    * MySQLi/Connection #2201, 2229
    * **Postgre/**
      · BaseBuilder #2269
      · Connection #2201
    * SQLite3/Connection #2201, 2228, 2230
    * BaseBuilder #2257, 2232, 2269, 2270
    * BaseConnection #2208, 2213, 2231
    * Config #2224
    * Forge #2205
    * MigrationRunner #2191
  - **Debug/**
    * Exceptions #2262
  - **Encryption/**
    * Encryption #2231
    * Handlers/BaseHandler #2231
  - **Files/**
    * FileCollection #2265
  - **HTTP/**
    * CURLRequest #2168
    * IncomingRequest #2265
    * Request #2253
    * Response #2253
  - **I18n/**
    * Time #2231
    * TimeDifference #2231
  - **Images/**
    * Handlers/BaseHandler #2246
  - **RESTful/**
    * ResourcePresenter #2271
  - **Security/**

        * Security #2240
    - **Session/**
        * Session #2197, 2231
    - **Test/**
        * CIDatabaseTestCase #2205
        * CIDatabaseUnitTestCase #2184
    - **Validation/**
        * FileRules #2265
        * Validation #2268
    - **View/**
        * Parser #2264
    - Common #2200, 2209, 2261
    - Model #2231
- tests/_support/
- **tests/system/**
    - **Commands/**
        * CommandClassTest #2231
    - **Database/**
        * **Builder/**
            · GetTest #2232
            · CountTest #2269
            · DeleteTest #2269
            · EmptyTest #2269
            · GetTest #2269
            · GroupTest #2257
            · InsertTest #2269
            · ReplaceTest #2269
            · TruncateTest #2269
            · UpdateTest #2269
        * **Live/**
            · EscapeTest #2229
            · ForgeTest #2201, 2211

            - GroupTest #2257
            - MetadataTest #2211
            - ModelTest #2231
        * BaseConnectionTest #2229, 2231
    - **Encryption/**
        * EncryptionTest #2231
    - **Helpers/**
        * URLHelperTest #2259
    - **HTTP/**
        * CURLRequestTest #2168
        * FileCOllectionTest #2265
        * URITest #2259
    - **I18n/**
        * TimeDifferenceTest #2231
        * TimeTest #2231
    - **Pager/**
        * pagerTest #2259
    - **RESTful/**
        * ResourcePresenterTest #2271
    - **Session/**
        * SessionTest #2231
    - **View/**
        * ParserTest #2264
- **user_guide_src/**
    - **concepts/**
        * structure #2221
    - **database/**
        * metadata #2199, 2201, 2208
        * queries #2208
        * query_builder #2257, 2232, 2269
    - **dbmgmt/**
        * migration #2190, 2191

- **extending/**
  - contributing #2221
- **general/**
  - errors #2221
- **helpera/**
  - url_helper #2259
- **incoming/**
  - restful #2189
  - routing #2221
- **installation/**
  - troubleshooting #2260
- **libraries/**
  - encryption #2221
  - pagination #2216
  - time #2221
  - uti #2216
- **outgoing/**
  - api_responses #2245
  - view_layouts #2218
  - view_parser #2218, 2264
- **testing/**
  - controllers #2221
  - debugging #2221, 2209
  - feature #2218, 2221
  - overview #2221
- **tutorial/**
  - news_section #2221
  - static_pages #2221

### PRs merged:

- #2271 fix ResourcePresenter::setModel()
- #2270 groupStart() refactorization

- #2269 testMode() method for BaseBuilder
- #2268 Validation session use only if exists
- #2267 Tests setUp and tearDown: void
- #2265 Fix a validation issue on multiple file upload
- #2264 fix. Parser allow other extension
- #2262 Fix parameter type in Debug/Exceptions
- #2261 Fix lang() signature
- #2260 Explain the whoops page
- #2259 Add URI & url_helper tests
- #2257 Several updates to the HAVING clauses
- #2253 Fix invalid parameters
- #2246 EXIF not supported for GIF
- #2245 Fix class ref parameter types
- #2241 Fix ini_set parameter type
- #2240 Handle JSON POSTs in CSRF
- #2232 Fixes BaseBuilder getWhere() bug
- #2231 Add magic ___isset to classes with ___get
- #2230 Add escape to SQLite _listTables()
- #2229 MySQLi escapeLikeStringDirect()
- #2228 Exclude *sqlite_%* from listTables()
- #2224 change new ConfigDatabase() to config( 'Database' )
- #2221 Documentation fixes
- #2218 Typo corrected
- #2216 Update uri.rst
- #2213 Filter listTables cache response on constrainPrefix
- #2211 Add listTable() tests
- #2209 Add trace()
- #2208 Add $db->getPrefix()
- #2205 Fix empty() bug on DBPrefix
- #2201 Foreign key columns
- #2200 Notify Kint of dd alias
- #2199 Add getForeignKeyData to User Guide

- #2187 Update Session.php
- #2191 Migration rollback reverse
- #2190 Fix name of ForeignKeyChecks
- #2189 missing return
- #2184 Fix case on "Seeds/" directory
- #2183 Check *defined* for constants
- #2181 Remove copy-paste extraneous text
- #2168 Fix for CURL for 'debug' option

### Version 4.0.0-rc.1

Release Date: Not released

**RC.1 release of CodeIgniter4**

Enhancements:

- CI3 Email ported to CI4
- Encryption (basic) added
- Migrations refactored and streamlined for more wholistic functionality (BC)
- added convert() to ImageHandlerInterface
- disabled debug toolbar for downloads
- CLI commands returns an error code now ( "spark" changed)
- RESTful controllers added to shorten dev time for RESTful APIs
- added RouteCollection::presenter() as part of the RESTful support

App changes:

- added app/Common to make it easier to override common functions
- Config/Email and Encryption added
- Config/Migration modified, and has different settings
- Controllers/Home fixed, removing unnecessary model reference

Message changes:

- Migration has new & modified messages
- Messages now has RESTful set

The list of changed files follows, with PR numbers shown.

- **admin/**
    - release-appstarter #2155

    - release-framework #2155
- **app/**
    - **Config/**
        * Email #2092
        * Encryption #2135
        * Migrations #2065
    - **Controllers/**
        * BaseController #2046
        * Home #2145
    - Common #2110
- public/
- **system/**
    - **API/**
        * ResponseTrait #2131
    - **Autoloader/**
        * Autoloader #2149
        * FileLocator #2149
    - **Cache/Handlders/**
        * RedisHandler #2144
    - **CLI/**
        * CommandRunner #2164
    - **Commands/Database/**
        * CreateMigration #2065
        * Migrate #2065, 2137
        * MigrateRefresh #2065, 2137
        * MigrateRollback #2065, 2137
        * MigrateStatus #2137
        * MigrateVersion #2137
    - **Config/**
        * BaseConfig #2082
        * Services #2135, 2092
    - **Database/**

    * BaseBuilder #2127, 2090, 2142, 2153, 2160, 2023, 2001
    * MigrationRunner #2065, 2137
- **Debug/**
    * Toolbar #2118
- **Email/**
    * Email #2092
- **Encryption/**
    * EncrypterInterface #2135
    * Encryption #2135
    * Exceptions/EncryptionException #2135
    * **Handlers/**
        · BaseHandler #2135
        · OpenSSLHandler #2135
- **Exceptions/**
    * ConfigException #2065
- **Files/**
    * File #2178
- **Filters/**
    * DebugToolbar #2118
- **Helpers/**
    * inflector_helper #2065
- **Honeypot/**
    * Honeypot #2177
- **HTTP/**
    * DownloadResponse #2129
    * Files/UploadedFile #2128
    * Message @2171
    * Response #2166
- **Images/**
    * **Handlers/** -BaseHandler #2113, 2150 - ImageMagickHandler #2151
    * BImageHandlerInterface #2113

  - **Language/en/**
    - Email #2092
    - Encryption #2135
    - Migrations #2065, 2137
    - RESTful #2165
  - **RESTful/**
    - ResourceController #2165
    - ResourcePresenter #2165
  - **Router/**
    - RouteCollection #2165
  - **Security/**
    - Security #2027
  - **Session/Handlers/**
    - RedisHandler #2125
  - **Test/**
    - CIDatabaseTestCase #2137
  - bootstrap #2110
  - CodeIgniter #2126, 2164
  - Common #2109
  - Entity #2112
  - Model #2090
- **tests/_support/**
  - RESTful/…#2165
- **tests/system/**
  - **API/**
    - ResponseTraitTest #2131
  - **Database/**
    - **Builder/**
      - GetTest #2142
      - SelectTest #2160
      - WhereTest #2001
    - **Live/**

            - GroupTest #2160
            - ModelTest #2090
            - SelectTest #2160
        * Migrations/MigrationRunnerTest #2065, 2137
    - **Encryption/**
        * EncryptionTest #2135
        * OpenSSLHandlerTest #2135
    - **Helpers/**
        * InflectorHelperTest #2065
    - **HTTP/**
        * DownloadResponseTest #2129
        * MessageTest #2171
    - **Images/**
        * GDHandlerTest #2113
    - **RESTful/**
        * ResourceControllerTest #2165
        * ResourcePresenterTest #2165
    - **Router/**
        * RouteCollectionTest #2165
    - ControllerTest #2165
    - EntityTest #2112
- **user_guide_src/**
    - **changelogs/**
        * next #2154
    - **database/**
        * query_builder #2160, 2001
    - **dbmgmt/**
        * migrations #2065, 2132, 2136, 2154, 2137
    - **extending/**
        * common #2162
    - **helpers/**
        * inflector_helper #2065

    - **incoming/**
        - restful #2165
        - routing #2165
    - **libraries/**
        - email #2092, 2154
        - encryption #2135
        - images #2113, 2169
    - **outgoing/**
        - api_responses #2131
        - localization #2134
        - response #2129
    - **testing/**
        - database #2137
- CONTRIBUTING.md #2010
- README.md #2010
- spark

### PRs merged:

- #2178 Add fallback for missing finfo_open
- #2177 Fix missing form close tag
- #2171 Setheader dupes
- #2169 Add $quality usage for Image Library
- #2166 Cookie error
- #2165 RESTful help
- #2164 Exit error code on CLI Command failure
- #2162 User Guide updates for Common.php
- #2160 Add BaseBuilder SelectCount
- #2155 Include .gitignore in starters
- #2153 Bug fix countAllResults with LIMIT
- #2154 Fix email & migrations docs; update changelog
- #2151 ImageMagick->save() return value
- #2150 New logic for Image->fit()

- #2149 listNamespaceFiles: Ensure trailing slash
- #2145 Remove UserModel reference from Home controller
- #2144 Update Redis legacy function
- #2142 Fixing BuilderBase resetting when getting the SQL
- #2137 New Migration Logic
- #2136 Migrations user guide fixes
- #2135 Encryption
- #2134 Fix localization writeup
- #2132 Update migration User Guide
- #2131 Added No Content response to APIResponseTrait
- #2129 Add setFileName() to DownloadResponse
- #2128 guessExtension fallback to clientExtension
- #2127 Update limit function since $offset is nullable
- #2126 Limit storePreviousURL to certain requests
- #2125 Updated redis session handler to support redis 5.0.x
- #2118 Disabled Toolbar on downloads
- #2113 Add Image->convert()
- #2112 Update *Entity.php* ___*isset* method
- #2110 Added app/Common.php
- #2109 Fix typo in checking if exists db_connect()
- #2092 Original email port
- #2090 Fix prevent soft delete all without conditions set
- #2082 Update BaseConfig.php
- #2065 Migration updates for more wholistic functionality
- #2046 clean base controller code
- #2027 Fix CSRF hash regeneration
- #2023 whereIn $value do not have to be an array
- #2010 Fix CSRF hash regenerationerbiage revisions
- #2001 Subqueries in BaseBuilder

### Version 4.0.0-beta.4

Release Date: Not released

Highlights:

There are some breaking changes…

- The Entity class has been refactored;
- The Model class changing has been updated to better handle soft deletes
- The routing has been beefed up

New messages:

- new translation key: Database/noDateFormat

App changes:

Testing changes:

- enhanced database & migration testing in tests/_support

The list of changed files follows, with PR numbers shown.

- admin/
- **app/**
    - **Controllers/**
        * Home #1999
- public/
- **system/**
    - **Autoloader/**
        * FileLocator #2059, #2064
    - **Cache/**
        * CacheFactory #2060
        * **Handlers/**
            · MemcachedHandler #2060
            · PredisHandler #2060
            · RedisHandler #2060
    - **Commands/**
        * Utilities/Routes #2008
    - **Config/**
        * Config #2079
        * Services #2024

- **Database/**
  - **MySQLi/**
    - Connection #2042
    - Result #2011
  - **Postgre/**
    - Connection #2042
    - Result #2011
  - **SQLite3/**
    - Connection #2042
    - Forge #2042
    - Result #2011
    - Table #2042
  - BaseBuilder #1989
  - BaseConnection #2042
  - BaseResult #2002
  - Forge #2042
  - MigratiopnRollback #2035
  - MigrationRunner #2019
- **Debug/**
  - Toolbar/Collectors/Routes #2030
- **Exceptions.**
  - ModelException #2054
- **Files/**
  - File #2104
- **Filters/**
  - Filters #2039 - helpers/
  - date_helper #2091
- **HTTP/**
  - CLIRequest #2024
  - CURLRequest #1996, #2050
  - IncomingRequest #2063
  - Request #2024

  - **Language/en/**
    - Database #2054
  - **Pager/**
    - Pager #2026
  - **Router/**
    - RouteCollection #1959, #2012, #2024
    - Router #2024, #2031, #2043
    - RouterInterface #2024
  - **Session/**
    - Handlers/ArrayHandler #2014
  - **Test/**
    - CIUnitTestCase #2002
    - FeatureTestCase #2043
  - **Throttle/**
    - Throttler #2074
  - CodeIgniter #2012, #2024
  - Common #2036
  - Entity #2002, #2004, #2011, #2081
  - Model #2050, #2051, #2053, #2054
- **tests/system/**
  - **CLI/**
    - ConsoleTest #2024
  - **Database/**
    - **Live/**
      - DbUtilsTest #2051, #2053
      - ForgeTest #2019, #2042
      - ModelTest #2002, #2051, #2053, #2054
      - SQLite/AlterTablesTest #2042
      - WhereTest #2052
    - Migrations/MigrationRunnerTest #2019
  - **HTTP/**
    - CLIRequest #2024

        * CURLRequestTest #1996
    - **Router/**
        * RouteCollectionTest #1959, #2012, #2024
        * RouterTest #2024, #2043
    - **Test/**
        * FeatureTestCaseTest #2043
    - **Throttle/**
        * ThrottleTest #2074
    - **View/**
        * ParserTest #2005
    - CodeIgniterTest #2024
    - EntityTest #2002, #2004
- **user_guide_src/**
    - **concepts/**
        * autoloader #2035, #2071
    - **database/**
        * query_builder #2035
    - **dbmgmt/**
        * forge #2042
        * migration #2042
    - **helpers/**
        * date_helper #2091
    - **incoming/**
        * routing #2035
    - **installation/**
        * installing_composer #2015, #2035
    - **libraries/**
        * pagination #2026
        * sessions #2014, #2035
        * validaiton #2069
        * uploaded_files #2104
    - **models/**

        * entitites #2002, #2004, #2035
        * model #2051, #2053, #2054
    - **outgoing/**
        * view_parser #e21823, 32005
    - **testing/**
        * database #2051, #2053

### PRs merged:

- #2104 File & UploadFile Fixes
- #2091 Timezone select
- #2081 JSON format checking improved
- #2079 Update config() to check all namespaces
- #2074 Throttler can access bucket for bucket life time
- #2071 Fix autoloader.rst formatting
- #2069 validation rule: then -> than (spelling)
- #2064 Bugfix file locator slash error
- #2063 Ensure query vars are part of request->uri. Fixes #2062
- #2060 Cache Drive Backups
- #2059 Add multi-path support to *locateFile()*
- #2054 Add model exceptions for missing/invalid dateFormat
- #2053 Change Model' s deleted flag to a deleted_at datetime/timestamp. Fixes #2041
- #2052 Add various tests for (not) null
- #2051 Soft deletes use deleted_at
- #2050 Stash insert ID before event trigger
- #2043 Zero params should be passed through when routing. Fixes #2032
- #2042 SQLite3 now supports dropping foreign keys. Fixes #1982
- #2040 Update CURLRequest.php
- #2039 Restrict filter matching of uris so they require an exact match. Fixes #2038
- #2036 Make *force_https()* send headers before exit
- #2035 Various typos and Guide corrections
- #2031 Fallback to server request for default method

- #2030 Support the new *router* service in Debug Toolbar
- #2026 Extension Pager::makeLinks (optional grup name)
- #2024 Refactor the way the router and route collection determine the current HTTP verb
- #2019 SQLite and Mysql driver additional tests and migration runner test fixes
- #2015 Direct user to follow the upgrade steps after installation
- #2014 Added a new Session/ArrayHandler that can be used during testing
- #2012 Use request->method for HTTP verb
- #2011 Set the raw data array without any mutations for the Entity
- #2008 Add *patch* method to command "routes"
- #2005 Plugin closures docs update and test
- #2004 Allow hasChanged() without parameter
- #2002 Entity Refactor
- #1999 use CodeIgniterController; not needed since Home Controller extends ···
- #1996 Attempting to fix CURLRequest debug issue. #1994
- #e21823 Corrected docs for parser plugins. Closes #1995
- #1989 argument set() must by type of string - cannot agree
- #1959 Prevent reverseRoute from searching closures

### Version 4.0.0-beta.3

Release Date: Not released

Highlights:

- Type hinting added throughout & typos corrected (see API docs)
- Fixed a number of model, database, validation & debug toolbar issues

New messages:

- Database.FieldNotExists
- Validation.equals, not_equals

App changes:

- Removed $salt config item in app/Config/App
- Enabled migrations by default in app/Config/Migrations
- Simplified public/.htaccess

The list of changed files follows, with PR numbers shown.

- **admin/**
  - framework/composer.json #1935
  - starter/composer.json #1935
- **app/**
  - **Config/**
    - App #1973
    - Migrations #1973
- **public/**
  - .htaccess #1973
- **system/**
  - **API/**
    - ResponseTrait #1962
  - **Commands/**
    - Server/rewrite #1925
  - **Config/**
    - AutoloadConfig #1974
    - BaseConfig #1947
  - **Database/ #1938**
    - BaseBuilder #1923, #1933, #1950
    - BaseConnection #1950
    - BaseResult #1917
    - BaseUtils #1917
    - Forge #1917
    - **SQLite3/**
      - Connection #1917
      - Result #1917
  - **Debug/**
    - Toolbar #1916
    - **Toolbar/Collectors/**
      - BaseCollector #1972
      - Config #1973
      - History #1945

            - Routes #1949
        * **Toolbar/Views/**
            - _config.tpl.php #1973
            - toolbar.tpl.php #1972
            - toolbarloader.js #1931, #1961
- **Exceptions/**
    * EntityException #1927
- **Filters/** Filters #1970, #1985
- **Format/**
    * FormatterInterface #1918
    * JSONFormatter #1918
    * XMLFormatter #1918
- **HTTP/**
    * CLIRequest #1956
    * CURLRequest #1915
- **Images/Handlers/**
    * BaseHandler #1956
- **Language/en/**
    * Database #1917
    * Validation #1952
- **Router/**
    * Router #1968
    * RouteCollection #1977
- **Session/Handlers/**
    * RedisHandler #1980
- **Test/**
    * FeatureResponse #1977
    * FeatureTestCase #1977
- **Validation/**
    * FormatRules #1957
    * Rules #1952
- **View/**

        * Table #1984
    - Entity #1911, #1927, #1943, #1950, #1955
    - Model #1930, #1943, #1963, #1981
- **tests/system/**
    - **Config/**
        * BaseConfigTest #1947
    - **Database/**
        * BaseQueryTest #1917
        * **Live/**
            · DbUtilsTest #1917, #1943
            · ForgeTest #1917
            · GetTest #1917, #1943
            · ModelTest #1930, #1943, #1981
        * **Migrations/**
            · MigrationRunnerTest #1917
            · MigrationTest #1943
    - **Filters/**
        * FilterTest #1985
    - **Test/**
        * FeatureTestCaseTest #1977
    - **Validation/**
        * FormatRulesTest #1957
        * RulesTest #1952, #cbe4b1d
    - **View/**
        * TableTest #1978, #1984
    - EntityTest #1911
- **user_guide_src/**
    - **dbmgmt/**
        * migrations #1973
    - **installation/**
        * installing_composer #1926
        * running #1935

    - **libraries/**
        - validation #1952, #1954, #1957
    - **outgoing/**
        - index #1978
        - table #1978, #1984
    - **testing/**
        - feature #1977
        - overview #1936
- .htaccess #1939
- composer.json #1935
- phpdoc.dist.xml #1987

### PRs merged:

- #1987 Correct API docblock problems for phpdocs
- #1986 Update docblock version to 4.0.0
- #1985 Fix filter processing. Fixes #1907
- #cbe4b1d Fix SQLite tests
- #1984 Add footing to HTML Table
- #1981 Using soft deletes should not return an ambiguous field message when joining tables
- #1980 Corrected return value for Session/RedisHandler::read
- #1978 Implement HTML Table for CI4 (missed feature)
- #1977 Test/featuretestcase
- #1974 Remove framework classes from the autoloader classmap
- #1973 Defaultfixes
- #1972 Toolbar fix for custom collectors
- #1970 Add back filter arguments
- #1968 Fixed pathinfo mode 404 error
- #1963 String type primary key should also wrap into an array during db update
- #1962 Fix side issue
- #1961 Fix Debugbar url tail slash issue
- #1957 New generic string validation rule

- #1956 Use Null Coalesce Operator
- #1955 Travis-CI build failed fix
- #1954 Fix validation table format
- #1952 Add Validations for *equals()* and *not_equals()*
- #1951 System typos changes & code cleanup
- #1950 Fix some side issue
- #1949 Toobar/Routes correction
- #1947 Fix BaseConfig didn't load Registrar files properly
- #1945 Fix datetime extraction from debugbar file
- #1943 Model, Entity, Exception & Migration test cases
- #1939 Remove section that prevents hotlinking
- #1938 Database typos changes
- #1936 Docs: improce app testing writeup
- #1935 Update phpunit.xml scripts. Fixes #1932
- #1933 having (Is NULL deletion)
- #1931 Toolbar IE11 fix
- #1930 Model Changes w.r.t. #1773
- #1927 Entity exception for non existed props
- #1926 Docs: update installation guide
- #1925 removed $_SERVER['CI_ENVIRONMENT']
- #1923 missing return
- #1918 JSONFormatter
- #1917 Database Test Cases
- #1916 Check if the value is string
- #1915 Fix for POST + JSON (Content-Length added)
- #1911 JSON Cast exception test cases

### Version 4.0.0-beta.2

Release Date: April 4, 2019

Highlights:

- A number of fixes & improvements, importantly for the Model and testing classes
- Models now require a primary key

- Generated API docs accessible at https://codeigniter4.github.io/api/
- Validation rules have been enhanced
- .htaccess beefed up

New messages:

- Database.noPrimaryKey, forFindColumnHaveMultipleColumns, Database.forEmptyInputGiven

App changes:

- updated app/Config/Events
- added app/Controllers/BaseController
- added tests/ folder for unit testing
- added phpunit.xml.dist for unit testing configuration

The list of changed files follows, with PR numbers shown.

- .htaccess #1900
- **app/**
    - **Config/**
        - Events #1856
    - **Controllers/**
        - BaseController #1847
        - Home #1847
- **contributing/**
    - README.rst #1846
    - styleguide #1872
- contributing.md #1846
- phpdoc.dist.xml #1872
- **system/**
    - **Autoloader/**
        - FileLocator #1860
    - **Cache/Handlers/**
        - FileHandler #1895
        - MemcachedHandler #1895
        - PredisHandler #1895
        - RedisHandler #1863, #1895

    * WincacheHandler #1895

- **CLI/**
    * CLI #1891, #1910
- **Commands/**
    * Server/Serve #1893
    * Utilities/Routes #1859
- **Config/**
    * BaseConfig #1811
    * Routes #1847, #1850
- **Database/**
    * BaseBuilder $1776, #1902
    * BaseConnection #1899
    * Forge #1844, #1899
    * MigrationRunner #1860, #1865
    * MySQLi/Connection #1896
    * MySQLi/Forge #1899
    * Postgre/Builder #1902
    * Postgre/Forge #1899
    * Query #1805, #1771
    * SQLite3/Builder #1902
    * SQLite3/Forge #1899
- **Debug/**
    * Toolbar/Collectors/History #1869
    * Toolbar #1897
- **Events/**
    * Events #1867
- **Exceptions/**
    * ModelException #1829
    * PageNotFoundException #1844
- **Files/**
    * File #1809, #1854
- **Helpers/**

    * date_helper #d08b68
    * form_helper #1803
    * html_helper #1803
    * number_helper #d08b68, #1803
    * security_helper #d08b68
    * text_helper #d08b68, #1803
    * url_helper #d08b68, #1803
    * xml_helper #1803
- **Honeypot/**
    * Honeypot #1894
- **HTTP/**
    * Header #1769
    * IncomingRequest #1831
- **Language/en/**
    * Database #1829, #1861, #1902
- **Router/**
    * RouteCollection #1769
    * Router #1839, #1882
- **Session/**
    * Session #1769
- **Test/**
    * ControllerTester #1769, #1848, #1855
    * DOMParser #1848
- **Validation/**
    * FormatRules #1762, #1863
    * Rules #1791, #1814, #1818, #1862
    * Validation #1769
    * Views/list #1828
- **View/**
    * Filters #1769
    * Parser #1769
    * View #1769, #1827

  - CodeIgniter #1769, #1804, #1590
  - Common #1802, #895ae0
  - ComposerScripts #1804
  - Controller #1769, #1850
  - Entity #1769, #1804
  - Model #1793, #1769, #1804, #1808, #1812, #1813, #1817, #1829, #1746, #1861

- **tests/system/**
  - **Cache/**
    - **Handlers/**
      - FileHandlerTest #1796, #1895
      - MemcachedHandlerTest #1895
      - RedisHandlerTest #1895
    - CacheFactoryTest #1796
  - **CLI/**
    - CLITest #1910
  - **Config/**
    - BaseConfigTest #1811
    - ConfigTest #1811
  - **Database/**
    - Builder/EmptyTest #1902
    - Builder/SelectTest #1902
    - Live/ModelTest #1817, #1829, #1861
    - Live/WhereTest #1906
  - **Events/**
    - EventsTest #1867
  - **HTTP/**
    - ContentSecurityPolicyTest #1848
  - **Router/**
    - RouteCollectionTest #1822, #1912, #1913
  - **Test/**
    - ControllerTesterTest #1848, #1855

        * DOMParserTest #1848
    - **Validation/**
        * FormatRulesTest #1762
        * RulesTest #1791
    - **View/**
        * ViewTest #1827, #1836
    - ControllerTest #1850
- **user_guide_src/**
    - **cli/**
        * cli_commands #1777
        * cli_library #1892, #1910
    - **concepts/**
        * services #1811
    - **database/**
        * examples #1794
    - **dbmgmt/**
        * forge #1844, #1899
        * migration #1860, #1865
    - **extending/**
        * basecontroller #1847
        * core_classes #1847
    - **general/**
        * common_functions #1802, #1895
    - **helpers/**
        * number_helper #d08b68
        * url_helper #1803
    - **incoming/**
        * routing #1908
    - **libraries/**
        * caching #1895
        * files #1790, #1854
        * pagination #1823

        * sessions #1843
        * validation #1814, #1828, #1862
    - **models/**
        * models #1817, #1820, #1829, #1746, #1861
    - **outgoing/**
        * view_layouts #1827
    - **testing/**
        * controllers #1848

### PRs merged:

- #1913 More RouteCollection tests for overwriting. Closes #1692
- #1912 Additional RouteCollectionTests
- #1910 Added print method to CLI library so you can print multiple times on same line
- #1908 Add filter parameters to User Guide
- #1906 SubQuery related test cases w.r.t #1775
- #1902 BaseBuilder corrections
- #1900 Update .htaccess for better security and caching
- #1899 Database Forge correction
- #1897 Toolbar fix w.r.t #1779
- #1896 Mysql connection issue with SSL cert (#1219)
- #1894 Typos fixings
- #1893 Fix spark serve with remove escapeshellarg()
- #1892 Add CLI background color list to the user guide
- #1891 Allow CLI::strlen null parameter
- #1886 Fixed issue #1880, fixed a few typos and updated code style
- #1882 Router Changes w.r.t #1541
- #1873-1889 Docs: move namespace declarations & add missing class docblocks
- #1872 Docs: fix phpdoc config
- #1871 Unmatched Cache Library *get()* return null
- #1869 History::SetFiles check #1778
- #1863 Module wise Typos changes

- #1861 New method Find Column w.r.t. #1619
- #1860 Migrationrunner use autoloader
- #1867 Events should actually work with any callable now. Fixes #1835
- #1865 MigrationRunner issue with definition resolved
- #1862 required_with and required_without definition changes
- #1859 Ignore callbacks in routes list
- #1858 Typos correction in DB module
- #1856 ensure ob_end_flush() when ob_get_level() > 0 on pre_system event
- #1855 Fix: ControllerTester::execute. Fixes #1834
- #1854 File::move now returns new file instance for relocated file. Fixes #1782
- #1851 Replace old CI3 .gitignore with root CI4 version
- #1850 Secure routable controller methods
- #1848 Test: fix & test Test/ControllerTest, tested
- #1847 Extend Controller to BaseController by default
- #1846 Fix contributing links
- #1844 Model Fix
- #1843 Replace CI3 $this->input reference
- #1842 Exception ‘forPageNotFound’ missing default value
- #1839 Dont replace slashes with backslashes in the to route
- #1836 Test: Improve ViewView coverage
- #1831 Fix some PHPDoc comments error
- #1829 Improve: Models now require a primary key. This is partially to keep the code ···
- #1828 Fix: Remove bootstrap styles from validation views.
- #1827 Fix: Adding include method to View library to render view partials···views.
- #1823 Docs: Remove legacy Bootstrap references in Pagination class
- #1822 Test: enhance RouteCollection coverage
- #1820 Fix: Correct sphinx errors in model.rst
- #1819 Improve: Add apibot for API docs using phpDocumentor
- #1818 Improve: Code improvement in exact_length Rule
- #1817 Improve: Model setValidationMessage functions introduced
- #895ae0 Fix: Start session whenver using the old command

- #1814 Enhance: extended exact_length[1,3,5]
- #1813 Fix: Model::save fix for earlier PRs
- #1812 Test: Improve Filters coverage
- #1811 Test: Config module coverage improved
- #1809 Fix file move failed. Fixex #1785
- #1808 Fix: Fix save method return value
- #1805 Docs: Query Class Changes
- #1804 Docs: Some Base Functional Changes
- #1803 Docs: Some Helper Changes
- #1802 Docs: Common function correction
- #1796 Test: Improve Cache coverage
- #1794 Replace nonexistent "getAffectedRows"
- #1793 Set Model->chunk return type
- #1791 Fix: Remove is_numeric tests in ValidationRules
- #d08b68 Fix in ControllerTester for missing UserAgent
- #1790 Correction of typos in documentation as mentioned in issue #1781
- #1777 Add CLI namespace to example
- #1776 Fix: replace only last operator in field name
- #1771 Fix: fix typo in matchSimpleBinds
- #1769 Correction in Methods and Spellings
- #1762 Fix: decimal rule. shouldn't it accept integers?
- #1746 Improve: Update Model, to selective update created_at / updated_at field.
- #1590 Improve: Enhance 404Override

### Version 4.0.0-beta.1

Release Date: Not released

Highlights:

- New View Layouts provide simple way to create site site view templates.
- Fixed user guide CSS for proper wide table display
- Converted UploadedFile to use system messages
- Numerous database, migration & model bugs fixed
- Refactored unit testing for appstarter & framework distributions

New messages:

- Database.tableNotFound
- HTTP.uploadErr…

App changes:

- app/Config/Cache has new setting: database
- app/Views/welcome_message has logo tinted
- composer.json has a case correction
- env adds CI_ENVIRONMENT suggestion

The list of changed files follows, with PR numbers shown.

- **app/**
  - **Config/**
    - Cache #1719
  - **Views/**
    - welome_message #1774
- **system/**
  - **Cache/Handlers/**
    - RedisHandler #1719, #1723
  - **Config/**
    - Config #37dbc1
    - Services #1704, #37dbc1
  - **Database/**
    - Exceptions/DatabaseException #1739
    - **Postgre/**
      - Builder #1733
    - **SQLite3/**
      - Connection #1739
      - Forge #1739
      - Table #1739
    - BaseBuilder #36fbb8, #549d7d
    - BaseConnection #549d7d, #1739
    - Forge #1739
    - MigrationRunner #1743

    * Query #36fbb8
    * Seeder #1722
- **Debug/**
    * Exceptions #1704
- **Files/**
    * UploadedFile #1708
- **Helpers/**
    * date_helper #1768
    * number_helper #1768
    * security_helper #1768
    * text_helper #1768
    * url_helper #1768
- **HTTP/**
    * Request #1725
- **Language/en/**
    * Database #1739
    * HTTP #1708
    * View #1757
- **Router/**
    * RouteCollection #1709, #1732
    * Router #1764
- **Test/**
    * ControllerResponse #1740
    * ControllerTester #1740
    * DOMParser #1740
    * FeatureResponse #1740
- **Validation/**
    * Rules #1738, #1743
    * Validation #37dbc1, #1763
- **View/**
    * View #1729
- Common #1741

    - Entity #6e549a, #1739
    - Model #4f4a37, #6e549a, #37dbc1, #1712, #1763
- **tests/system/**
    - **Database/**
        * BaseQueryTest #36fbb8
        * **Live/**
            · SQLite3/AlterTableTest #1739, #1740
            · ForgeTest #1739, #1745
            · ModelTest #37dbc1, #4ff1f5, #1763
        * Migrations/MigrationRunnerTest #1743
    - **Helpers/**
        * FilesystemHelperTest #1740
    - **I18n/**
        * TimeTest # 1736
    - **Test/**
        * DOMParserTest #1740
    - **Validation/**
        * ValidationTest #1763
    - **View/**
        * ViewTest #1729
    - EntityTest #6e549a, #1736
- **user_guide_src/**
    - **_themes/…/**
        * citheme.css #1696
    - **changelogs/**
        * v4.0.0-alpha.5 #1699
    - **database/**
        * migrate #1696
    - **dbmgmt/**
        * forge #1751
    - **installation/**
        * install_manual #1699

        * running #1750
    - **intro/**
        * psr #1752
    - **libraries/**
        * caching #1719
        * validation #1742
    - **models/**
        * entities #1744
    - **outgoing/**
        * index #1729
        * view_layouts #1729
    - **testing/**
        * controllers #1740
    - **tutorial/**
        * static_pages #1763
- composer.json #1755
- .env #1749

### PRs merged:

- #1774 Housekeeping for beta.1
- #1768 Helper changes - signatures & typos
- #1764 Fix routing when no default route has been specified. Fixes #1758
- #1763 Ensure validation works in Model with errors as part of rules. Fixes #1574
- #1757 Correct the unneeded double-quote (typo)
- #1755 lowercase ‘vfsStream’ in composer files
- #1752 Fixed typo preventing link format
- #1751 Guide: Moving misplaced text under correct heading
- #1750 Remove reference to Encryption Key in User Guide
- #1749 Adding environment to .env
- #1745 Updated composite key tests for SQLite3 support. Fixes #1478
- #1744 Update entity docs for current framework state. Fixes #1727
- #1743 Manually sort migrations found instead of relying on the OS. Fixes #1666

- #1742 Fix required_without rule bug.
- #1741 Helpers with a specific namespace can be loaded now. Fixes #1726
- #1740 Refactor test support for app starter
- #1739 Fix typo
- #1738 Fix required_with rule bug. Fixes #1728
- #1737 Added support for dropTable and modifyTable with SQLite driver
- #1736 Accommodate long travis execution times
- #1733 Fix increment and decrement errors with Postgres
- #1732 Don't check from CLI in Routes. Fixes #1724
- #1729 New View Layout functionality for simple template
- #1725 Update Request.php
- #1723 Log an error if redis authentication is failed
- #1722 Seeder adds default namespace to seeds
- #1719 Update Cache RedisHandler to support select database
- #4ff1f5 Additional tests for inserts and required validation failing (#1717)
- #549d7d Another try at getting escaping working correctly both when in and out of models
- #1712 Minor readability changes
- #37dbc1 Ensure Model validation rules can be a group name
- #1709 Fix resource routing websafe method order checking
- #1708 Language for UploadedFile
- #36fbb8 BaseBuilder should only turn off Connection's setEscapeFlags when running a query…
- #6e549a Provide default baseURL that works with the development server for easier first time setup (Fixes #1646)
- #1704 Fix viewsDirectory bug (#1701)
- #4f4a37 remove debugging from Model.
- #1699 Fix install link in user guide
- #1696 Fix page structure etc
- #1695 Tidy up code blocks in the user guide

### Version 4.0.0-alpha.5

Release Date: Jan 30, 2019

**Next alpha release of CodeIgniter4**

Highlights:

- added $maxQueries setting to app/Config/Toolbar.php
- updated PHP dependency to 7.2
- new feature branches have been created for the email and queue modules, so they don't impact the release of 4.0.0
- dropped several language messages that were unused (eg Migrations.missingTable) and added some new (eg Migrations.invalidType)
- lots of bug fixes, especially for the database support
- provided filters (CSRF, Honeypot, DebugToolbar) have been moved from app/Filters/ to system/Filters/
- revisited the installation and tutorial sections of the user guide
- code coverage is at 77% ···getting ever closer to our target of 80% :)

We hope this will be the last alpha, and that the next pre-release will be our first beta ···fingers crossed!

The list of changed files follows, with PR numbers shown.

- **admin/**
  - **starter/**
    - README.md #1637
    - app/Config/Paths.php #1685
  - release-appstarter #1685
- **app/**
  - **Config/**
    - Filters #1686
    - Modules #1665
    - Services #614216
    - Toolbar
- **contributing/**
  - guidelines.rst #1671, #1673
  - internals.rst #1671
- **public/**
  - index.php #1648, #1670
- **system/**

- **Autoloader/**
  - Autoloader #1665, #1672
  - FileLocator #1665
- **Commands/**
  - Database/MigrationRollback #1683
- **Config/**
  - BaseConfig #1635
  - BaseService #1635, #1665
  - Paths #1626
  - Services #614216, #3a4ade, #1643
  - View #1616
- **Database/**
  - BaseBuilder #1640, #1663, #1677
  - BaseConnection #1677
  - Config #6b8b8b, #1660
  - MigrationRunner #81d371, #1660
  - Query #1677
- **Database/Postgre/**
  - Builder #d2b377
- **Debug/Toolbar/Collectors/**
  - Logs #1654
  - Views #3a4ade
- **Events/**
  - Events #1635
- **Exceptions/**
  - ConfigException #1660
- **Files/**
  - Exceptions/FileException #1636
  - File #1636
- **Filters/**
  - Filters #1635, #1625, #6dab8f
  - CSRF #1686

        * DebugToolbar #1686
        * Honeypot #1686
    - **Helpers/**
        * form_helper #1633
        * html_helper #1538
        * xml_helper #1641
    - **HTTP/**
        * ContentSecurityPolicy #1641, #1642
        * URI #2e698a
    - **Language/**
        * /en/Files #1636
        * Language #1641
    - **Log/**
        * Handlers/FileHandler #1641
    - **Router/**
        * RouteCollection #1665, #5951c3
        * Router #9e435c, #7993a7, #1678
    - **Session/**
        * Handlers/BaseHandler #1684
        * Handlers/FileHandler #1684
        * Handlers/MemcachedHandler #1679
        * Session #1679
    - bootstrap #81d371, #1665
    - Common #1660
    - Entity #1623, #1622
    - Model #1617, #1632, #1656, #1689
- **tests/**
    - README.md #1671
- **tests/system/**
    - **API/**
        * ResponseTraitTest #1635
    - **Autoloader/**

    * AutoloaderTest #1665
    * FileLocatorTest #1665, #1686
- **CLI/**
    * CommandRunnerTest #1635
    * CommandsTest #1635
- **Config/**
    * BaseConfigTest #1635
    * ConfigTest #1643
    * ServicesTest #1635, #1643
- **Database/Builder/**
    * AliasTest #bea1dd
    * DeleteTest #1677
    * GroupTest #1640
    * InsertTest #1640, #1677
    * LikeTest #1640, #1677
    * SelectTest #1663
    * UpdateTest #1640, #1677
    * WhereTest #1640, #1677
- **Database/Live/**
    * AliasTest #1675
    * ConnectTest #1660, #1675
    * ForgeTest #6b8b8b
    * InsertTest #1677
    * Migrations/MigrationRunnerTest #1660, #1675
    * ModelTest #1617, #1689
- **Events/**
    * EventTest #1635
- **Filters/**
    * CSRFTest #1686
    * DebugToolbarTest #1686
    * FiltersTest #1635, #6dab8f, #1686
    * HoneypotTest #1686

  - **Helpers/**
    - FormHelperTest #1633
    - XMLHelperTest #1641
  - **Honeypot/**
    - HoneypotTest #1686
  - **HTTP/**
    - ContentSecurityPolicyTest #1641
    - IncomingRequestTest #1641
  - **Language/**
    - LanguageTest #1643
  - **Router/**
    - RouteCollectionTest #5951c3
    - RouterTest #9e435c
  - **Validation/**
    - RulesTest #1689
  - **View/**
    - ParserPluginTest #1669
    - ParserTest #1669
- user_guide_src/
  - **concepts/**
    - autoloader #1665
    - structure #1648
  - **database/**
    - connecting #1660
    - transactions #1645
  - **general/**
    - configuration #1643
    - managing_apps #5f305a, #1648
    - modules #1613, #1665
  - **helpers/**
    - form_helper #1633
  - **incoming/**

        * filters #1686
        * index #4a1886
        * methodspoofing #4a1886
    - **installation/**
        * index #1690, #1693
        * installing_composer #1673, #1690
        * installing_git #1673, #1690
        * installing_manual #1673, #1690
        * repositories #1673, #1690
        * running #1690, #1691
        * troubleshooting #1690, #1693
    - **libraries/**
        * honeypot #1686
        * index #1643, #1690
        * throttler #1686
    - **tutorial/**
        * create_news_item #1693
        * index #1693
        * news_section #1693
        * static_pages #1693
- composer.json #1670
- contributing.md #1670
- README.md #1670
- spark #1648
- .travis.yml #1649, #1670

### PRs merged:

- #1693 Docs/tutorial
- #5951c3 Allow domain/sub-domain routes to overwrite existing routes
- #1691 Update the running docs
- #1690 Rework install docs
- #bea1dd Additional AliasTests for potential LeftJoin issue

- #1689 Model Validation Fix
- #1687 Add copyright blocks to filters
- #1686 Refactor/filters
- #1685 Fix admin - app starter creation
- #1684 Updating session id cleanup for filehandler
- #1683 Fix migrate:refresh bug
- #d2b377 Fix Postgres replace command to work new way of storing binds
- #4a1886 Document method spoofing
- #2e698a urldecode URI keys as well as values.
- #1679 save_path - for memcached
- #1678 fix route not replacing forward slashes
- #1677 Implement Don't Escape feature for db engine
- #1675 Add missing test group directives
- #1674 Update changelog
- #1673 Updated download & installation docs
- #1672 Update Autoloader.php
- #1670 Update PHP dependency to 7.2
- #1671 Update docs
- #1669 Enhance Parser & Plugin testing
- #1665 Composer PSR4 namespaces are now part of the modules auto-discovery
- #6dab8f Filters match case-insensitively
- #1663 Fix bind issue that occurred when using whereIn
- #1660 Migrations Tests and database tweaks
- #1656 DBGroup in ___get(), allows to validate "database" data outside the model
- #1654 Toolbar - Return Logger::$logCache items
- #1649 remove php 7.3 from "allow_failures" in travis config
- #1648 Update "managing apps" docs
- #1645 Fix transaction enabling confusing (docu)
- #1643 Remove email module
- #1642 CSP nonce attribute value in ""
- #81d371 Safety checks for config files during autoload and migrations
- #1641 More unit testing tweaks

- #1640 Update getCompiledX methods in BaseBuilder
- #1637 Fix starter README
- #1636 Refactor Files module
- #5f305a UG - Typo in managing apps
- #1635 Unit testing enhancements
- #1633 Uses csrf_field and form_hidden
- #1632 DBGroup should be passed to ->run instead of ->setRules
- #1631 move use statement after License doc at UploadedFile class
- #1630 Update copyright to 2019
- #1629 "application" to "app" directory doc and comments
- #3a4ade view() now properly reads the app config again
- #7993a7 Final piece to get translateURIDashes working appropriately
- #9e435c TranslateURIDashes fix
- #1626 clean up Paths::$viewDirectory property
- #1625 After matches is not set empty
- #1623 Property was not cast if was defined as nullable
- #1622 Nullable support for ___set
- #1617 countAllResults() should respect soft deletes
- #1616 Fix View config merge order
- #614216 Moved honeypot service out of the app Services file to the system Services where it belongs
- #6b8b8b Allow db forge and utils to take an array of connection info instead of a group name
- #1613 Typo in documentation
- #1538 img fix(?) - html_helper

### Version 4.0.0-alpha.4

Release Date: Dec 15, 2018

**Next alpha release of CodeIgniter4**

Highlights:

- **Refactor for consistency: folder application renamed to app;** constant BASEPATH renamed to SYSTEMPATH
- Debug toolbar gets its own config, history collector

- Numerous corrections and enhancements

The list of changed files follows, with PR numbers shown.

- **admin/**
  - docbot #1573
  - framework/composer.json #1555
  - release #1573
  - release-deploy #1573
  - starter/composer.json #1573, #1600
- **app/**
  - **Config/**
    - App #1571
    - Autoload #1579
    - ContentSecurityPolicy #1581
    - Events #1571, #1595
    - Paths #1579
    - Routes #1579
    - Services #1579
    - Toolbar #1571, #1579
  - **Filters/**
    - Toolbar #1571
  - **Views/**
    - errors/* #1579
- **public/**
  - index #1579
- **system/**
  - **Autoloader/**
    - Autoloader #1562
    - FileLocator #1562, #1579
  - **CLI/**
    - CommandRunner #1562
  - **Config/**
    - AutoloadConfig #1555, #1579

    * BaseConfig #1562
    * Services #1571, #1562
- **Database/**
    * BaseBuilder #a0fc68
    * MigrationRunner #1585
    * MySQLi/Connection #1561, #8f205a
- **Debug/**
    * Collectors/* #1571, #1589, #1579
    * Exceptions #1579
    * Toolbar #1571
    * Views/toolbar.tpl #1571
    * Views/toolbarloader.js #1594
- **Helpers/**
    * form_helper #1548
    * url_helper #1588
- **HTTP/**
    * ContentSecurityPolicy #1581
    * DownloadResponse
- **I18n/**
    * Time #1603
- **Language/**
    * Language #1587, #1562, #1610
    * **en/**
        · CLI #1562
        · HTTP #d7dfc5
- **Log/**
    * Handlers/FileHandler #1579
    * Logger #1562, #1579
- **Session/**
    * Handlers/DatabaseHandler #1598
- **Test/**
    * CIUnitTest #1581, #1593, #1579

        * FeatureResponse #1593
        * FeatureTestCase #1593
    - **View/**
        * View #1571, #1579
    - bootstrap #1579
    - CodeIgniter #ab8b5b, #1579
    - Common #1569, #1563, #1562, #1601, #1579
    - Entity #4c7bfe, #1575
    - Model #1602, #a0fc68
- **tests/**
    - **Autoloader/**
        * AutolaoderTest #1562, #1579
        * FileLocatorTest #1562, #1579
    - **Config/**
        * ServicesTest #1562
    - **Database/**
        * Live/ModelTest #1602, #a0fc68
    - **Files/**
        * FileTest #1579
    - **Helpers/**
        * FormHelperTest #1548
        * URLHelperTest #1588
    - **HTTP/**
        * ContentSecurityPolicyTest #1581
        * DownloadResponseTest #1576, #1579
        * IncomingRequestDetectingTest #1576
        * IncomingRequestTest #1576
        * RedirectResponseTest #1562
        * ResponseTest #1576
    - **I18n/**
        * TimeDifferenceTest #1603
        * TimeTest #1603

    - **Language/** -LanguageTest #1587, #1610
    - **Log/**
        - FileHandlerTest #1579
    - **Router/**
        - RouterCollectionTest #1562
        - RouterTest #1562
    - **Test/**
        - FeatureResponseTest #1593
        - FeatureTestCaseTest #1593
        - TestCaseTest #1593
    - **Validation/**
        - ValidationTest #1562
    - **View/**
        - ParserPluginTest #1562
        - ParserTest #1562
        - ViewTest #1562
    - CodeIgniterTest #1562
    - CommonFunctionsTest #1569, #1562
    - EntityTest #4c7bfe, #1575
- **user_guide_src/source/**
    - **cli/**
        - cli #1579
        - cli_commands #1579
    - **concepts/**
        - autoloader #1579
        - mvc #1579
        - services #1579
        - structure #1579
    - **database/**
        - configuration #1579
    - **dbmgt/**
        - migration #1579

        * seeds #1579
    - **general/**
        * common_functions #d7dfc5, #1579
        * configuration #1608
        * errors #1579
    - **installation/**
        * downloads #1579
    - **models/**
        * entities #547792, #1575
    - **outgoing/**
        * localization #1610
        * response #1581, #1579
        * view_parser #1579
    - **testing/**
        * debugging #1579
        * overview #1593, #1579
    - **tutorial/**
        * news_section #1586
        * static_pages #1579
- composer.json #1555
- ComposerScripts #1551
- spark #1579
- Vagrantfile.dist #1459

### PRs merged:

- #1610 Test, fix & enhance Language
- #a0fc68 Clear binds after inserts, updates, and find queries
- #1608 Note about environment configuration in UG
- #1606 release framework script clean up
- #1603 Flesh out I18n testing
- #8f305a Catch mysql connection errors and sanitize username and password
- #1602 Model’ s first and update didn’ t work primary key-less tables

- #1601 clean up ConfigServices in Common.php
- #1600 admin/starter/composer.json clean up
- #1598 use $defaultGroup as default value for database session DBGroup
- #1595 handle fatal error via pre_system
- #1594 Fix Toolbar invalid css
- #1593 Flesh out the Test package testing
- #1589 Fix Toolbar file loading throw exception
- #1588 Fix site_url generate invalid url
- #1587 Add Language fallback
- #1586 Fix model namespace in tutorial
- #1585 Type hint MigrationRunner methods
- #4c7bfe Entity fill() now respects mapped properties
- #547792 Add _get and _set notes for Entity class
- #1582 Fix changelog index & common functions UG indent
- #1581 ContentSecurityPolicy testing & enhancement
- #1579 Use Absolute Paths
- #1576 Testing13/http
- #1575 Adds ?integer, ?double, ?string, etc. cast types
- #ab8b5b Set baseURL to example.com during testing by default.
- #d7dfc5 Doc tweaks for redirects
- #1573 Lessons learned
- #1571 Toolbar updates
- #1569 Test esc() with different encodings and ignore app-only helpers
- #1563 id attribute support added for csrf_field
- #1562 Integrates Autoloader and FileLocator
- #1561 Update Connection.php
- #1557 remove prefix on use statements
- #1556 using protected intead of public modifier for setUp() function in tests
- #1555 autoload clean up: remove PsrLog namespace from composer.json
- #1551 remove manual define "system/" directory prefix at ComposerScripts
- #1548 allows to set empty html attr
- #1459 Add Vagrantfile

### Version 4.0.0-alpha.3

Release Date: November 30, 2018

**Next alpha release of CodeIgniter4**

The list of changed files follows, with PR numbers shown.

- **admin/**
  - framework/* #1553
  - starter/* #1553
  - docbot #1553
  - release* #1484,
  - pre-commit #1388
  - README.md #1553
  - setup.sh #1388
- **application /**
  - **Config/**
    - Autoload #1396, #1416
    - Mimes #1368, #1465
    - Pager #622
    - Services #1469
  - Filters/Honeypot #1376
  - **Views/**
    - errors/* #1415, #1413, #1469
    - form.php removed #1442
- **public /**
  - index.php #1388, #1457
- **system /**
  - **Autoloader/**
    - Autoloader #1547
    - FileLocator #1547, #1550
  - **Cache/**
    - Exceptions/CacheException #1525
    - Handlers/FileHandler #1547, #1525
    - Handlers/MemcachedHandler #1383

- **CLI/**
  - CLI #1432, #1489
- **Commands/**
  - **Database/**
    - CreateMigration #1374, #1422, #1431
    - MigrateCurrent #1431
    - MigrateLatest #1431
    - MigrateRollback #1431
    - MigrateStatus #1431
    - MigrateVersion #1431
  - Sessions/CrateMigration #1357
- **Config/**
  - AutoloadConfig #1416
  - BaseService #1469
  - Mimes #1453
  - Services #1180, #1469
- **Database/**
  - BaseBuilder #1335, #1491, #1522
  - BaseConnection #1335, #1407, #1491, #1522
  - BaseResult #1426
  - Config #1465, #1469, #1554
  - Forge #1343, #1449, #1470, #1530
  - MigrationRunner #1371
  - MySQLi/Connection #1335, #1449
  - MySQLi/Forge #1343, #1344, #1530
  - MySQLi/Result #1530
  - Postgre/Connection #1335, #1449
  - Postgre/Forge #1530
  - SQLite3/Connection #1335, #1449
  - SQLite3/Forge #1470, #1547
- **Debug**
  - Exceptions #1500

    * Toolbar #1370, #1465, #1469, #1547
    * Toolbar/Views/toolbar.tpl #1469
- **Email/**
    * Email #1389, #1413, #1438, #1454, #1465, #1469, #1547
- **Events/**
    * Events #1465, #1469, #1547
- **Files/**
    * File #1399, #1547
- **Format/**
    * XMLFormatter #1471
- **Helpers/**
    * array_helper #1412
    * filesystem_helper #1547
- **Honeypot/**
    * Honeypot #1460
- **HTTP/**
    * CURLRequest #1547, #1498
    * DownloadResponse #1375
    * Exceptions/DownloadException #1405
    * Files/FileCollection #1506
    * Files/UploadedFile #1335, #1399, #1500, #1506, #1547
    * IncomingRequest #1445, #1469, #1496
    * Message #1497
    * RedirectResponse #1387, #1451, #1464
    * Response #1456, #1472, #1477, #1486, #1504, #1505, #1497, #622
    * ResponseInterface #1384
    * UploadedFile #1368, #1456
    * URI #1213, #1469, #1508
- **Images/Handlers/**
    * ImageMagickHandler #1546
- **Language/**

    * en/Cache #1525
    * en/Database #1335
    * en/Filters #1378
    * en/Migrations #1374
    * Language #1480, #1489
- **Log/**
    * Handlers/FileHandler #1547
- **Pager/**
    * Pager #1213, #622
    * PagerInterface #622
    * PagerRenderer #1213, #622
    * Views/default_full #622
    * Views/default_head #622
    * Views/default_simple #622
- **Router/**
    * RouteCollection #1464, #1524
    * RouteCollectionInterface #1406, #1410
    * Router #1523, #1547
- **Session/Handlers/**
    * BaseHandler #1180, #1483
    * DatabaseHandler #1180
    * FileHandler #1180, #1547
    * MemcachedHandler #1180
    * RedisHandler #1180
- **Test/**
    * CIUnitTestCase #1467
    * FeatureTestCase #1427, #1468
    * Filters/CITestStreamFilter #1465
- **Validation /**
    * CreditCardRules #1447, #1529
    * FormatRules #1507
    * Rules #1345

        * Validation #1345
    - **View/**
        * Filters #1469
        * Parser #1417, #1547
        * View #1357, #1377, #1410, #1547
    - bootstrao #1547
    - CodeIgniter #1465, #1505, #1523, 2047b5a, #1547
    - Common #1486, #1496, #1504, #1513
    - ComposerScripts #1469, #1547
    - Controller #1423
    - Entity #1369, #1373
    - Model #1345, #1380, #1373, #1440
- **tests /**
    - **_support/**
        * HTTP/MockResponse #1456
        * _bootstrap.php #1397, #1443
    - **Cache/Handlers/**
        * FileHandlerTest #1547, #1525
        * MemcachedHandlerTest #1180, #1383
        * RedisHandlerTest #1180, #1481
    - **CLI/**
        * CLITest #1467, #1489
    - **Commands/**
        * SessionCommandsTest #1455
    - **Database/Live/**
        * ConnectTest #1554
        * ForgeTest #1449, #1470
    - **HTTP/**
        * CURLRequestTest#1498
        * Files/FileCOllectionTest #1506
        * Files/FileMovingTest #1424
        * DownloadResponseTest #1375

    * IncomingRequestTest #1496
    * RedirectResponseTest #1387, #1456
    * ResponseCookieTest #1472, #1509
    * ResponseSendTest #1477, #1486, #1509
    * ResponseTest #1375, #1456, #1472, #1486, #622
    * URITest #1456, #1495
- **Helpers/**
    * DateHelperTest #1479
- **I18n/**
    * TimeTest #1467, #1473
- **Language/**
    * LanguageTest #1480
- **Log/**
    * FileHandlerTest #1425
- **Pager/**
    * PagerRendererTest #1213, #622
    * PagerTest #622
- **Router/**
    * RouteCollectionTest #1438, #1524
    * RouterTest #1438, #1523
- **Session/**
    * SessionTest #1180
- **Test/**
    * BootstrapFCPATHTest #1397
    * FeatureTestCase #1468
    * TestCaseEmissionsTest #1477
    * TestCaseTest #1390
- **Throttle/**
    * ThrottleTest #1398
- **Validation/**
    * FormatRulesTest #1507
- **View/**

        * ParserTest #1335
    - CodeIgniterTest #1500
    - CommonFunctionsSendTest #1486, #1509
    - CommonFunctionsTest #1180, #1486, #1496
- **user_guide_src /source/**
    - changelogs/ #1385, #1490, #1553
    - **concepts/**
        * autoloader #1547
        * security #1540
        * services #1469
        * structure #1448
    - **database/**
        * queries #1407
    - **dbmgmt/**
        * forge #1470
        * migration #1374, #1385, #1431
        * seeds #1482
    - **extending/**
        * core_classes #1469
    - **helpers/**
        * form_helper #1499
    - **installation/**
        * index #1388
    - **libraries/**
        * caching #1525
        * pagination #1213
        * validation #27868b, #1540
    - **models/**
        * entities #1518, #1540
    - **outgoing/**
        * response #1472, #1494
    - **testing/**

        * overview #1467
    - **tutorial/**
        * create_news_item #1442
        * static_pages #1547
- /
    - composer.json #1388, #1418, #1536, #1553
    - README.md #1553
    - spark 2047b5a
    - .travis.yml #1394

### PRs merged:

- #1554 Serviceinstances
- #1553 Admin/scripts
- #1550 remove commented CLI::newLine($tempFiles) at FileLocator
- #1549 use .gitkeep instead of .gitignore in Database/Seeds directory
- #1547 Change file exists to is file
- #1546 ImageMagickHandler::__construct …
- #1540 Update validation class User Guide
- #1530 database performance improvement : use foreach() when possible
- 2047b5a Don’t run filters when using spark.
- #1539 remove mb_* (mb string usage) in CreditCardRules
- #1536 ext-json in composer.json
- #1525 remove unneeded try {} catch {}
- #1524 Test routes resource with ‘websafe’ option
- #1523 Check if the matched route regex is filtered
- #1522 add property_exists check on BaseBuilder
- #1521 .gitignore clean up
- #1518 Small typo: changed setCreatedOn to setCreatedAt
- #1517 move .htaccess from per-directory in writable/{directory} to writable/
- #1513 More secure redirection
- #1509 remove unused use statements
- #1508 remove duplicate strtolower() call in URI::setScheme() call

- #1507 Fix multi “empty” string separated by “,” marked as valid emails
- #1506 Flesh out HTTP/File unit testing
- #1505 Do not exit until all Response is completed
- 27868b Add missing docs for {field} and {param} placeholders
- #1504 Revert RedirectResponse changes
- #1500 Ignoring errors suppressed by @
- #1499 Fix form_helper’ s set_value writeup
- #1498 Add CURLRequest helper methods
- #1497 Remove unused RedirectException
- #1496 Fix Common::old()
- #1495 Add URI segment test
- #1494 Method naming in user guide
- #1491 Error logging
- #1490 Changelog(s) restructure
- #1489 Add CLI::strlen()
- #1488 Load Language strings from other locations
- #1486 Test RedirectResponse problem report
- #1484 missing slash
- #1483 Small typo in SessionHandlersBaseHandler.php
- #1482 doc fix: query binding fix in Seeds documentation
- #1481 RedisHandler test clean up
- #1480 Fix Language Key-File confusion
- #1479 Yet another time test to fix
- #1477 Add Response send testing
- #1475 Correct phpdocs for Forge::addField()
- #1473 Fuzzify another time test
- #1472 HTTPResponse cookie testing & missing functionality
- #1471 remove unused local variable $result in XMLFormatter::format()
- #1470 Allow create table with array field constraints
- #1469 use static:: instead of self:: for call protected/public functions as well
- #1468 Fix FeatureTestCaseTest output buffer
- #1467 Provide time testing within tolerance

- #1466 Fix phpdocs for BaseBuilder
- #1465 use static:: instead of self:: for protected and public properties
- #1464 remove unused use statements
- #1463 Fix the remaining bcit-ci references
- #1461 Typo fix: donload -> download
- #1460 remove unneeded ternary check at HoneyPot
- #1457 use $paths->systemDirectory in public/index.php
- #1456 Beef up HTTP URI & Response testing
- #1455 un-ignore app/Database/Migrations directory
- #1454 add missing break; in loop at Email::getEncoding()
- #1453 BugFix if there extension has only one mime type
- #1451 remove unneeded $session->start(); check on RedirectResponse
- #1450 phpcbf: fix all at once
- #1449 Simplify how to get indexData from mysql/mariadb
- #1448 documentation: add missing application structures
- #1447 add missing break; on loop cards to get card info at CreditCardRules
- #1445 using existing is_cli() function in HTTPIncomingRequest
- #1444 Dox for reorganized repo admin (4 of 4)
- #1443 Fixes unit test output not captured
- #1442 remove form view in app/View/ and form helper usage in create new items tutorial
- #1440 Access to model’s last inserted ID
- #1438 Tailor the last few repo org names (3 of 4)
- #1437 Replace repo org name in MOST php docs (2 of 4)
- #1436 Change github organization name in docs (1 of 4)
- #1432 Use mb_strlen to get length of columns
- #1431 can’t call run() method with params from commands migrations
- #1427 Fixes “options” request call parameter in FeatureTestCase
- #1416 performance improvement in DatabaseBaseResult
- #1425 Ensure FileHandlerTest uses MockFileHandler
- #1424 Fix FileMovingTest leaving cruft
- #1423 Fix Controller use validate bug

- #1422 fix Migrations.classNotFound
- #1418 normalize composer.json
- #1417 fix Parser::parsePairs always escapes
- #1416 remove $psr4[ 'TestsSupport' ] definition in applicationConfigAutoload
- #1415 remove unneded "defined( 'BASEPATH' ) ···
- #1413 set more_entropy = true in all uniqid() usage
- #1412 function_exists() typo fixes on array_helper
- #1411 add missing break; in loop in View::render()
- #1410 Fix spark serve not working from commit 2d0b325
- #1407 Database: add missing call initialize() check on BaseConnection->prepare()
- #1406 Add missing parameter to RouteCollectionInterface
- #1405 Fix language string used in DownloadException
- #1402 Correct class namespacing in the user guide
- #1399 optional type hinting in guessExtension
- #1398 Tweak throttle testing
- #1397 Correcting FCPATH setting in tests/_support/_bootstrap.php
- #1396 only register PSR4 "TestsSupport" namespace in "testing" environment
- #1395 short array syntax in docs
- #1394 add php 7.3 to travis config
- #1390 Fixed not to output "Hello" at test execution
- #1389 Capitalize email filename
- #1388 Phpcs Auto-fix on commit
- #1387 Redirect to named route
- #1385 Fix migration page; udpate changelog
- #1384 add missing ResponseInterface contants
- #1383 fix TypeError in MemcachedHandler::__construct()
- #1381 Remove unused use statements
- #1380 count() improvement, use truthy check
- #1378 Update Filters language file
- #1377 fix monolog will cause an error
- #1376 Fix cannot use class Honeypot because already in use in AppFiltersHoneypot
- #1375 Give download a header conforming to RFC 6266

- #1374 Missing feature migration.
- #1373 Turning off casting for db insert/save
- #1371 update method name in coding style
- #1370 Toolbar needs logging. Fixes #1258
- #1369 Remove invisible character
- #1368 UploadedFile->guessExtenstion()…
- #1360 rm –cached php_errors.log file
- #1357 Update template file is not .php compatibility
- #1345 is_unique tried to connect to default database instead of defined in DBGroup
- #1344 Not to quote unecessary table options
- #1343 Avoid add two single quote to constraint
- #1335 Review and improvements in databases drivers MySQLi, Postgre and SQLite
- #1213 URI segment as page number in Pagination
- #1180 using HTTPRequest instance to pull ip address
- #622 Add Header Link Pagination

### Version 4.0.0-alpha.2

Release Date: Oct 26, 2018

**Second alpha release of CodeIgniter4**

The list of changed files follows, with PR numbers shown.

**application /**

- composer.json #1312
- Config/Boot/development, production, testing #1312
- Config/Paths #1341
- Config/Routes #1281
- Filters/Honeypot #1314
- Views/errors/cli/error_404 #1272
- Views/welcome_message #1342

**public /**

- .htaccess #1281
- index #1295, #1313

**system /**

- **CLI/**
  - CommandRunner #1350, #1356
- **Commands/**
  - Server/Serve #1313
- **Config/**
  - AutoloadConfig #1271
  - Services #1341
- **Database/**
  - BaseBuilder #1217
  - BaseUtils #1209, #1329
  - Database #1339
  - MySQLi/Utils #1209
- **Debug/Toolbar/**
  - Views/toolbar.css #1342
- **Exceptions/**
  - CastException #1283
  - DownloadException #1239
  - FrameworkException #1313
- **Filters/**
  - Filters #1239
- **Helpers/**
  - cookie_helper #1286
  - form_helper #1244, #1327
  - url_helper #1321
  - xml_helper #1209
- **Honeypot/**
  - Honeypot #1314
- **HTTP/**
  - CliRequest #1303
  - CURLRequest #1303
  - DownloadResponse #1239
  - Exceptions/HTTPException #1303

        - IncomingRequest #1304, #1313
        - Negotiate #1306
        - RedirectResponse #1300, #1306, #1329
        - Response #1239, #1286
        - ResponseInterface #1239
        - URI #1300
    - **Language/en/**
        - Cast #1283
        - HTTP #1239
    - **Router/**
        - RouteCollection #1285, #1355
    - **Test/**
        - CIUnitTestCase #1312, #1361
        - FeatureTestCase #1282
    - CodeIgniter #1239 #1337
    - Common #1291
    - Entity #1283, #1311
    - Model #1311

**tests /**

    - **API/**
        - ResponseTraitTest #1302
    - **Commands/**
        - CommandsTest #1356
    - **Database/**
        - BaseBuilderTest #1217
        - Live/ModelTest #1311
    - **Debug/**
        - TimerTest #1273
    - **Helpers/**
        - CookieHelperTest #1286
    - **Honeypot/**
        - HoneypotTest #1314

- **HTTP/**
  - **Files/**
    - FileMovingTest #1302
    - UploadedFileTest #1302
  - CLIRequestTest #1303
  - CURLRequestTest #1303
  - DownloadResponseTest #1239
  - NegotiateTest #1306
  - RedirectResponseTest #1300, #1306, #1329
  - ResponseTest #1239
- **I18n/**
  - TimeTest #1273, #1316
- **Router/**
  - RouteTest #1285, #1355
- **Test/**
  - TestCaseEmissionsTest #1312
  - TestCaseTest #1312
- **View/**
  - ParserTest #1311
- EntityTest #1319

**user_guide_src /source/**

- **cli/**
  - cli_request #1303
- **database/**
  - query_builder #1217
  - utilities #1209
- **extending/**
  - contributing #1280
- **general/**
  - common_functions #1300, #1329
  - helpers #1291
  - managing_apps #1341

- **helpers/**
  - xml_helper #1321
- **incoming/**
  - controllers #1323
  - routing #1337
- **intro/**
  - requirements #1280, #1303
- **installation/ #1280, #1303**
  - troubleshooting #1265
- **libraries/**
  - curlrequest #1303
  - honeypot #1314
  - sessions #1333
  - uploaded_files #1302
- **models/**
  - entities #1283
- **outgoing/**
  - response #1340
- **testing/**
  - overview #1312
- tutorial···#1265, #1281, #1294

/

- spark #1305

### PRs merged:

- #1361 Add timing assertion to CIUnitTestCase
- #1312 Add headerEmitted assertions to CIUnitTestCase
- #1356 Testing/commands
- #1355 Handle duplicate HTTP verb and generic rules properly
- #1350 Checks if class is instantiable and is a command
- #1348 Fix sphinx formatting in sessions
- #1347 Fix sphinx formatting in sessions

- #1342 Toolbar Styles
- #1341 Make viewpath configurable in Paths.php. Fixes #1296
- #1340 Update docs for downloads to reflect the need to return it. Fixes #1331
- #1339 Fix error where Forge class might not be returned. Fixes #1225
- #1337 Filter in the router Fixes #1315
- #1336 Revert alpha.2
- #1334 Proposed changelog for alpha.2
- #1333 Error in user guide for session config. Fixes #1330
- #1329 Tweaks
- #1327 FIX form_hidden and form_open - value escaping as is in form_input.
- #1323 Fix doc error : show_404() doesn’t exist any more
- #1321 Added missing xml_helper UG page
- #1319 Testing/entity
- #1316 Refactor TimeTest
- #1314 Fix & expand Honeypot & its tests
- #1313 Clean exception
- #1311 Entities store an original stack of values to compare against so we d···
- #1306 Testing3/http
- #1305 Change chdir(‘public’) to chdir($public)
- #1304 Refactor script name stripping in parseRequestURI()
- #1303 Testing/http
- #1302 Exception：No Formatter defined for mime type °
- #1300 Allow redirect with Query Vars from the current request.
- #1295 Fix grammar in front controller comment.
- #1294 Updated final tutorial page. Fixes #1292
- #1291 Allows extending of helpers. Fixes #1264
- #1286 Cookies
- #1285 Ensure current HTTP verb routes are matched prior to any * matched ro···
- #1283 Entities
- #1282 system/Test/FeatureTestCase::setupRequest(), minor fixes phpdoc block···
- #1281 Tut
- #1280 Add contributing reference to user guide

- #1273 Fix/timing
- #1272 Fix undefined variable “heading” in cli 404
- #1271 remove inexistent “CodeIgniterLoader” from AutoloadConfig::classmap
- #1269 Release notes & process
- #1266 Adjusting the release build scripts
- #1265 WIP Fix docs re PHP server
- #1245 Fix #1244 (form_hidden declaration)
- #1239 【Unsolicited PR】I changed the download method to testable.
- #1217 Optional parameter for resetSelect() call in Builder’s countAll();
- #1209 Fix undefined function xml_convert at DatabaseBaseUtils

### Version 4.0.0-alpha.1

Release Date: September 28, 2018

**Rewrite of the CodeIgniter framework**

**New packages list:**

- **API**
  - \ ResponseTrait
- **Autoloader**
  - \ AutoLoader, FileLocator
- **CLI**
  - \ BaseCommand, CLI, CommandRunner, Console
- **Cache**
  - \ CacheFactory, CacheInterface
  - \ Handlers ···Dummy, File, Memcached, Predis, Redis, Wincache
- **Commands**
  - \ Help, ListCommands
  - \ Database \ CreateMigration, MigrateCurrent, MigrateLatest, MigrateRefresh, MigrateRollback, MigrateStatus, MigrateVersion, Seed
  - \ Server \ Serve
  - \ Sessions \ CreateMigration
  - \ Utilities \ Namespaces, Routes
- **Config**

  - \ AutoloadConfig, BaseConfig, BaseService, Config, DotEnv, ForeignCharacters, Routes, Services, View

- **Database**
  - \ BaseBuilder, BaseConnection, BasePreparedQuery, BaseResult, BaseUtils, Config, ConnectionInterface, Database, Forge, Migration, MigrationRunner, PreparedQueryInterface, Query, QueryInterface, ResultInterface, Seeder
  - \ MySQLi \ Builder, Connection, Forge, PreparedQuery, Result
  - \ Postgre \ Builder, Connection, Forge, PreparedQuery, Result, Utils
  - \ SQLite3 \ Builder, Connection, Forge, PreparedQuery, Result, Utils
- **Debug**
  - \ Exceptions, Iterator, Timer, Toolbar
  - \ Toolbar \ Collectors···
- **Email**
  - \ Email
- **Events**
  - \ Events
- **Files**
  - \ File
- **Filters**
  - \ FilterInterface, Filters
- **Format**
  - \ FormatterInterface, JSONFormatter, XMLFormatter
- **HTTP**
  - \ CLIRequest, CURLRequest, ContentSecurityPolicy, Header, IncomingRequest, Message, Negotiate, Request, RequestInterface, Response, ResponseInterface, URI, UserAgent
  - \ Files \ FileCollection, UploadedFile, UploadedFileInterface
- **Helpers**
  - ···array, cookie, date, filesystem, form, html, inflector, number, security, text, url
- **Honeypot**
  - \ Honeypot
- **I18n**

  - \ Time, TimeDifference

- **Images**
  - \ Image, ImageHandlerInterface
  - \ Handlers ···Base, GD, ImageMagick
- **Language**
  - \ Language
- **Log**
  - Logger, LoggerAwareTrait
  - \ Handlers ···Base, ChromeLogger, File, HandlerInterface
- **Pager**
  - \ Pager, PagerInterface, PagerRenderer
- **Router**
  - \ RouteCollection, RouteCollectionInterface, Router, RouterInterface
- **Security**
  - \ Security
- **Session**
  - \ Session, SessionInterface
  - \ Handlers ···Base, File, Memcached, Redis
- **Test**
  - \ CIDatabaseTestCase, CIUnitTestCase, FeatureResponse, FeatureTestCase, ReflectionHelper
  - \ Filters \ CITestStreamFilter
- **ThirdParty (bundled)**
  - \ Kint (for \Debug)
  - \ PSR \ Log (for \Log)
  - \ ZendEscaper \ Escaper (for \View)
- **Throttle**
  - \ Throttler, ThrottlerInterface
- **Typography**
  - \ Typography
- **Validation**
  - \ CreditCardRules, FileRules, FormatRules, Rules, Validation, ValidationInterface

- **View**
  - \ Cell, Filters, Parser, Plugins, RendererInterface, View

# 索引

## Non-alphabetical



## A



## B



## C



## D

## E



## F



## G



## H

## T



## U



## V



## W



## X



## Y