CodeIgniter4 中文手册

版本 *4.6.0*

CodeIgniter 基金会

CodeIgniter 中国开发者社区

2025 年 02 月 19 日

目录

1 欢迎使用 CodeIgniter4 **1**

1.1 CodeIgniter 适合你吗? 1

1.1.1 服务器要求 2

1.1.2 致谢 4

1.1.3 PSR 兼容性 5

1.1.4 MIT 许可证 (MIT) 6

2 开始使用 **7**

2.1 安装 7

2.1.1 Composer 安装 8

2.1.2 手动安装 15

2.1.3 运行你的应用程序 17

2.1.4 故障排除 30

2.1.5 部署 34

2.1.6 变更记录 40

2.1.7 从前一版本升级 245

2.1.8 CodeIgniter 仓库 415

3 创建第一个应用 **417**

3.1 构建你的第一个应用程序 417

3.1.1 概述 418

3.1.2 启动并运行 444

3.1.3 欢迎页面 445

3.1.4 调试 446

4 概览和常规主题 **449**
4.1 CodeIgniter4 概览 449
4.1.1 应用程序结构 449
4.1.2 模型、视图和控制器 452
4.1.3 自动加载文件 454
4.1.4 服务 460
4.1.5 工厂 468
4.1.6 处理 HTTP 请求 480
4.1.7 安全指南 483
4.1.8 设计与架构目标 505
4.2 常规主题 506
4.2.1 配置 506
4.2.2 CodeIgniter URL 519
4.2.3 辅助函数 523
4.2.4 全局函数和常量 529
4.2.5 记录信息 547
4.2.6 错误处理 551
4.2.7 网页缓存 564
4.2.8 AJAX 请求 566
4.2.9 代码模块 568
4.2.10 管理应用程序 579
4.2.11 处理多个环境 582

5 请求处理 **587**
5.1 控制器和路由 587
5.1.1 URI 路由 587
5.1.2 控制器 623
5.1.3 控制器过滤器 640
5.1.4 自动路由（改进版） 655
5.1.5 HTTP 消息 669
5.1.6 请求类 675
5.1.7 IncomingRequest 类 680
5.1.8 内容协商 701
5.1.9 HTTP 方法欺骗 707
5.1.10 RESTful 资源处理 707
5.2 构建响应 716
5.2.1 视图 716

5.2.2 视图渲染器 723
5.2.3 视图布局 728
5.2.4 视图单元 732
5.2.5 视图解析器 743
5.2.6 视图装饰器 766
5.2.7 HTML 表格类 768
5.2.8 HTTP 响应 781
5.2.9 API 响应特性 798
5.2.10 内容安全策略 808
5.2.11 本地化 813
5.2.12 在视图文件中使用 PHP 替代语法 828

6 数据库 831

6.1 使用数据库 831
6.1.1 快速入门：使用示例 831
6.1.2 数据库配置 835
6.1.3 连接数据库 846
6.1.4 查询 849
6.1.5 生成查询结果 861
6.1.6 查询辅助方法 878
6.1.7 查询构建器类 880
6.1.8 事务 969
6.1.9 获取元数据 974
6.1.10 自定义函数调用 980
6.1.11 数据库事件 981
6.1.12 数据库实用工具 982
6.2 数据建模 987
6.2.1 使用 CodeIgniter 的模型 987
6.2.2 使用实体类 1031
6.3 管理数据库 1047
6.3.1 数据库 Forge 类 1047
6.3.2 数据库迁移 1069
6.3.3 数据库填充 1080
6.3.4 数据库命令 1083

7 类库和辅助函数 1087

7.1 类库参考 1087
7.1.1 缓存驱动 1087

7.1.2 Cookie . . . 1099
7.1.3 跨域资源共享 (CORS) . . . 1122
7.1.4 CURLRequest 类 . . . 1128
7.1.5 Email 类 . . . 1143
7.1.6 加密服务 . . . 1157
7.1.7 文件处理 . . . 1168
7.1.8 文件集合 . . . 1172
7.1.9 蜜罐类 . . . 1177
7.1.10 图像处理类 . . . 1179
7.1.11 分页 . . . 1189
7.1.12 Publisher . . . 1206
7.1.13 安全 . . . 1221
7.1.14 Session 库 . . . 1230
7.1.15 限速器 . . . 1253
7.1.16 时间与日期 . . . 1259
7.1.17 排版 . . . 1279
7.1.18 处理上传的文件 . . . 1281
7.1.19 使用 URI . . . 1296
7.1.20 User Agent 类 . . . 1306
7.1.21 数据验证 . . . 1311
7.2 辅助函数 . . . 1351
7.2.1 数组辅助函数 . . . 1351
7.2.2 Cookie 辅助函数 . . . 1365
7.2.3 日期辅助函数 . . . 1369
7.2.4 文件系统辅助函数 . . . 1371
7.2.5 表单辅助函数 . . . 1379
7.2.6 HTML 辅助函数 . . . 1403
7.2.7 Inflector 辅助函数 . . . 1421
7.2.8 数字辅助函数 . . . 1426
7.2.9 安全辅助函数 . . . 1430
7.2.10 测试辅助函数 . . . 1432
7.2.11 文本辅助函数 . . . 1433
7.2.12 URL 辅助函数 . . . 1447
7.2.13 XML 辅助函数 . . . 1462

8 高级主题 1465
8.1 测试 . . . 1465

8.1.1 测试 1465
8.1.2 测试数据库 1479
8.1.3 生成测试数据 1485
8.1.4 测试控制器 1497
8.1.5 HTTP 功能测试 1508
8.1.6 测试响应 1513
8.1.7 测试 CLI 命令 1525
8.1.8 模拟系统类 1532
8.1.9 基准测试 1533
8.1.10 调试应用程序 1539
8.2 命令行使用 1546
8.2.1 CLI 概览 1546
8.2.2 通过 CLI 运行控制器 1547
8.2.3 Spark 命令 1550
8.2.4 创建 Spark 命令 1552
8.2.5 CLI 生成器 1558
8.2.6 CLI 库 1568
8.2.7 CLIRequest 类 1581
8.3 扩展 CodeIgniter 1583
8.3.1 创建核心系统类 1583
8.3.2 替换通用函数 1587
8.3.3 事件 1587
8.3.4 扩展控制器 1592
8.3.5 身份验证 1595
8.3.6 创建 Composer 包 1595
8.3.7 为 CodeIgniter 做贡献 1600

9 官方包 1601
9.1 官方包 1601
9.1.1 Shield 1602
9.1.2 Settings 1602
9.1.3 任务 (BETA) 1602
9.1.4 队列 (BETA) 1602
9.1.5 Cache 1603
9.1.6 DevKit 1603
9.1.7 编码标准 1603

PHP Namespace Index 1605

章节 1

欢迎使用 CodeIgniter4

CodeIgniter 是一套轻量、快速、灵活且安全的 PHP 全栈 Web 框架。

CodeIgniter 也是一套给 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 适合你吗?

如果你:

- 你想要一个小巧的框架。
- 你需要出色的性能。
- 你想要一个包含多种特性和技术的框架，以便你轻松地执行良好的安全实践。

- 你想要一个没有魔法的框架，能够轻松找到并阅读源代码。
- 你想要一个几乎 0 配置的框架。
- 你想要一个不想被编码规则的条条框框限制住的框架。
- 你想要一个易于创建测试代码的框架。
- 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、全面的文档。

那么 CodeIgniter 就适合你。

1.1.1 服务器要求

- *PHP 与必需扩展*
- *可选 PHP 扩展*
- *支持的数据库*

PHP 与必需扩展

需要 PHP 8.1 或更高版本，并启用以下 PHP 扩展：

- intl
- mbstring
- json

警告:

- PHP 7.4 的生命周期结束日期是 2022 年 11 月 28 日。
- PHP 8.0 的生命周期结束日期是 2023 年 11 月 26 日。
- **如果你仍在使用 PHP 7.4 或 8.0，应该立即升级。**
- PHP 8.1 的生命周期结束日期将是 2025 年 12 月 31 日。

备注:

- PHP 8.4 需要 CodeIgniter 4.6.0 或更高版本。
- PHP 8.3 需要 CodeIgniter 4.4.4 或更高版本。
- PHP 8.2 需要 CodeIgniter 4.2.11 或更高版本。
- PHP 8.1 需要 CodeIgniter 4.1.6 或更高版本。
- **请注意我们只维护最新版本**。

可选 PHP 扩展

建议在服务器上启用以下 PHP 扩展:

- mysqlnd (如果使用 MySQL)
- curl (如果使用*CURLRequest*)
- imagick (如果使用*Image* 类的 ImageMagickHandler)
- gd (如果使用*Image* 类的 GDHandler)
- simplexml (如果处理 XML 格式)

使用缓存服务器时需要以下 PHP 扩展:

- memcache (如果使用*Cache* 类的 MemcachedHandler 配合 Memcache)
- memcached (如果使用*Cache* 类的 MemcachedHandler 配合 Memcached)
- redis (如果使用*Cache* 类的 RedisHandler)

使用 PHPUnit 时需要以下 PHP 扩展:

- dom (如果使用*TestResponse* 类)
- libxml (如果使用*TestResponse* 类)
- xdebug (如果使用 `CIUnitTestCase::assertHeaderEmitted()`)

支持的数据库

大多数 Web 应用程序需要数据库支持。当前支持的数据库包括:

- MySQL（通过 `MySQLi` 驱动，仅支持 5.1 及以上版本）
- PostgreSQL（通过 `Postgre` 驱动，仅支持 7.4 及以上版本）
- SQLite3（通过 `SQLite3` 驱动）
- Microsoft SQL Server（通过 `SQLSRV` 驱动，仅支持 2012 及以上版本）
- Oracle Database（通过 `OCI8` 驱动，仅支持 12.1 及以上版本）

并非所有驱动都已为 CodeIgniter4 完成转换/重写。以下是尚未完成的驱动列表:

- MySQL (5.1+) 通过 *pdo* 驱动
- Oracle 通过 *pdo* 驱动
- PostgreSQL 通过 *pdo* 驱动
- MSSQL 通过 *pdo* 驱动
- SQLite 通过 *sqlite* (版本 2) 和 *pdo* 驱动
- CUBRID 通过 *cubrid* 和 *pdo* 驱动
- Interbase/Firebird 通过 *ibase* 和 *pdo* 驱动
- ODBC 通过 *odbc* 和 *pdo* 驱动（需注意 ODBC 实际上是一个抽象层）

1.1.2 致谢

CodeIgniter 最初是由 EllisLab 开发。该框架是为了在真实世界中提供高性能而编写的, 许多原始的类库、辅助函数和子系统都是从 ExpressionEngine 的代码库中借鉴过来的。多年来, 它由 EllisLab、ExpressionEngine 开发团队和一个名为 Reactor Team 的社区成员组维护和开发。

2014 年,CodeIgniter 被 英属哥伦比亚理工学院 收购, 之后正式宣布成为一个社区维护的项目。

2019 年,CodeIgniter 基金会成立, 以提供一个永久的管理团体, 独立于任何其他实体, 以确保框架的未来发展。

1.1.3 PSR 兼容性

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

此建议涵盖了基本的类、方法和文件名命名规范。我们的 代码规范 符合 PSR-1 且在此基础上添加了自己的要求。

PSR-12: 扩展代码风格

我们的 代码规范 遵循该建议, 并添加了一组我们自己的代码风格约定。

PSR-3: 日志接口规范

CodeIgniter 的日志 实现了此 PSR 定义的所有接口。

PSR-4: 自动加载规范

此 PSR 提供了一种组织文件和命名空间的方法, 以允许标准化的自动加载类的方式。我们的自动加载 符合 PSR-4 建议。

PSR-6: 缓存接口 PSR-16: 简单缓存接口

虽然框架的缓存组件不遵循 PSR-6 或 PSR-16, 但 CodeIgniter4 组织提供了一组独立的适配器作为补充模块。建议项目直接使用原生的缓存驱动, 因为适配器仅用于与第三方库的兼容性。详情参见 CodeIgniter4 缓存仓库。

PSR-7: HTTP 消息接口规范

此 PSR 标准化了表示 HTTP 交互的方式。尽管其许多概念成为了我们的 HTTP 层的一部分, 但 CodeIgniter 不追求与此建议兼容。

如果你发现我们声称遵循某个 PSR 但实际执行不正确的地方, 请告知我们, 我们将修复它, 或者你可以通过提交 PR 来提供所需的更改。

1.1.4 MIT 许可证 (MIT)

版权所有 (c) 2014-2019 不列颠哥伦比亚理工学院

版权所有 (c) 2019-至今 CodeIgniter 基金会

特此免费授予任何获得本软件和相关文档文件 (以下简称“软件”) 副本的个人权利, 可以不受限制地处理本“软件”, 包括但不限于使用、复制、修改、合并、发布、分发、再许可和/或出售“软件”的副本的权利, 并允许软件所提供给的个人行使此权利, 须符合以下条件:

上述版权声明和本许可声明必须包含在“软件”的所有副本或主要部分中。

本“软件”是“按原样”提供的, 不提供任何形式的明示或暗示保证, 包括但不限于对适销性、特定用途适用性和非侵权性的保证。在任何情况下, 作者或版权持有人不对任何索赔、损害或其他责任负责, 无论是因使用、无法使用或与使用或其他交易“软件”相关的合同、侵权或其他行为而引起。

章节 2

开始使用

2.1 安装

CodeIgniter 支持两种安装方法: 手动下载和使用 Composer 。对你来说哪个是正确的?

- 我们推荐使用 Composer 安装, 因为它可以轻松地保持 CodeIgniter 的更新。
- 如果你希望使用 CodeIgniter 3 所知的简单 “下载并使用” 安装, 请选择手动安装。

无论你选择以何种方式安装和运行 CodeIgniter4, 最新的 用户指南 都可以在线访问。如果你想查看之前的版本, 可以从 codeigniter4/userguide 存储库下载。

备注: 在使用 CodeIgniter 4 之前, 请确保你的服务器满足要求 , 特别是 PHP 版本和所需的 PHP 扩展。例如, 你可能需要取消注释 `php.ini` 文件的 “extension” 行以启用 “curl” 和 “intl”。

2.1.1 Composer 安装

- *App Starter*
  - 安装
  - 初始配置
  - 升级
  - 优点
  - 缺点
  - 结构
  - 最新开发版本
- 将 *CodeIgniter4* 添加到现有项目中
  - 安装
  - 设置
  - 初始配置
  - 升级
  - 优点
  - 缺点
  - 结构
- 翻译安装

Composer 可以通过两种方式在你的系统上安装 CodeIgniter4。

重要: CodeIgniter4 需要 Composer 2.0.14 或更高版本。

备注: 如果你不熟悉 Composer, 我们建议你先阅读 基本用法。

第一个方法描述了如何使用 CodeIgniter4 创建一个项目骨架（App Starter），你可以将其作为新 Web 应用的基础。下面描述的第二个方法允许你将 CodeIgniter4 添加到已有的

Web 应用中。

备注: 如果你使用 Git 仓库来存储代码或与他人协作，那么 **vendor** 文件夹通常会被 "git 忽略"。在这种情况下，当你将仓库克隆到一个新系统时，你需要执行 `composer install` （如果你想更新所有 Composer 依赖项，则执行 `composer update`）。

App Starter

CodeIgniter 4 app starter 仓库包含一个骨架应用程序, 其中包含对最新版本框架的 Composer 依赖项。

此安装技术适用于希望启动基于 CodeIgniter4 的新项目的开发人员。

安装

在项目根目录上层文件夹中:

```
composer create-project codeigniter4/appstarter 项目根目录
```

上述命令将创建一个 **项目根目录**文件夹。

如果省略 "项目根目录" 参数, 该命令将创建一个 "appstarter" 文件夹, 可以根据需要重命名。

备注: 在 v4.4.0 之前，CodeIgniter 的自动加载器不允许在某些操作系统上的文件名中使用非法的特殊字符。可以使用的符号包括 /、_、.、:、\ 和空格。因此，如果你将 CodeIgniter 安装在包含特殊字符（如（、）等）的文件夹中，CodeIgniter 将无法正常工作。从 v4.4.0 开始，这个限制已经被移除。

重要: 当你将应用部署到生产服务器时, 不要忘记运行以下命令:

```
composer install --no-dev
```

上述命令将只移除开发环境下的 Composer 软件包, 这些软件包在生产环境中不需要。这将大大减少 vendor 文件夹的大小。

安装先前版本

例如，你可能希望在 v4.5.0 发布后安装 v4.4.8。

在这种情况下，在命令中指定版本：

```
composer create-project codeigniter4/appstarter:4.4.8 project-root
```

然后，在项目根文件夹中打开 **composer.json**，并指定框架版本：

```
"require": {
    ...
    "codeigniter4/framework": "4.4.8"
},
```

然后，运行 `composer update` 命令。

备注: 当你在 **composer.json** 中使用固定版本号如 `"codeigniter4/framework": "4.4.8"` 时，`composer update` 命令将不会更新框架到最新版本。请参见 Writing Version Constraints 了解如何指定版本。

初始配置

安装后, 需要进行一些初始配置。有关详细信息, 请参阅初始配置。

升级

每当有新版本发布时, 在项目根目录的命令行中运行:

```
composer update
```

阅读升级说明 和变更日志，并检查重大变更和增强功能。

升级到指定版本

例如，你可能希望在 v4.5.0 发布后从 v4.4.7 升级到 v4.4.8。

在这种情况下，在项目根文件夹中打开 **composer.json**，并指定框架版本：

```
"require": {
    ...
    "codeigniter4/framework": "4.4.8"
},
```

然后，运行 `composer update` 命令。

备注： 当你在 **composer.json** 中使用固定版本号如 `"codeigniter4/framework": "4.4.8"` 时，`composer update` 命令将不会更新框架到最新版本。请参见 Writing Version Constraints 了解如何指定版本。

优点

安装简单; 易于更新。

缺点

更新后, 你仍然需要检查 **项目空间**中的文件更改 (根目录、app、public、writable), 并合并它们。

备注： 有一些第三方 CodeIgniter 模块可用于协助合并项目空间的更改: 在 Packagist 上探索。

结构

设置后项目中的文件夹:

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

最新开发版本

App Starter 仓库带有 `builds` 脚本, 可在当前稳定版本和框架的最新开发分支之间切换 Composer 源。此脚本适用于愿意使用最新的未发布更改 (可能不稳定) 的开发者。

开发用户指南 可以在线访问。请注意, 这与已发布的用户指南不同, 并将明确适用于 develop 分支。

最新开发版更新

在你的项目根目录中执行以下命令:

```
php builds development
```

上述命令将更新 **composer.json**，将其指向工作仓库的 `develop` 分支，并更新配置文件和 XML 文件中的相应路径。

使用 `builds` 命令后，请确保运行 `composer update`，以使你的 vendor 文件夹与最新的目标构建同步。然后，根据需要检查从前一版本升级 并更新项目文件。

下一个次要版本

如果你想使用下一个次要版本的分支，在使用 `builds` 命令后手动编辑 **composer.json**。

如果你尝试使用 `4.6` 分支，请将版本更改为 `4.6.x-dev`:

```
"require": {
    "php": "^8.1",
    "codeigniter4/codeigniter4": "4.6.x-dev"
},
```

然后运行 `composer update`，以使你的 vendor 文件夹与最新的目标构建同步。然后，根据需要检查升级指南（**user_guide_src/source/installation/upgrade_{version}.rst**）并更新项目文件。

恢复到稳定版本

要恢复更改，请运行：

```
php builds release
```

将 CodeIgniter4 添加到现有项目中

"手动安装"中描述的相同 CodeIgniter 4 框架 仓库也可以使用 Composer 添加到现有项目中。

安装

在 `app` 文件夹中开发你的应用程序, `public` 文件夹将是你的文档根目录。

在项目根目录中:

```
composer require codeigniter4/framework
```

重要: 将应用程序部署到生产服务器时, 不要忘记运行以下命令:

```
composer install --no-dev
```

上述命令将只移除开发环境下的 Composer 软件包, 这些软件包在生产环境中不需要。这将大大减少 vendor 文件夹的大小。

设置

1. 从 **vendor/codeigniter4/framework** 复制 **app**、**public**、**tests** 和 **writable** 文件夹到项目根目录
2. 从 **vendor/codeigniter4/framework** 复制 **env**、**phpunit.xml.dist** 和 **spark** 文件到项目根目录
3. 你将必须调整 **app/Config/Paths.php** 中的 `$systemDirectory` 属性, 以引用 vendor 目录, 例如 `__DIR__ . '/../../vendor/codeigniter4/framework/system'`。

初始配置

需要进行一些初始配置。有关详细信息, 请参阅*初始配置*。

升级

每当有新版本发布时, 在项目根目录的命令行中运行:

```
composer update
```

阅读*升级说明* 和*变更日志*，并检查重大变更和增强功能。

升级到指定版本

例如，你可能希望在 v4.5.0 发布后从 v4.4.7 升级到 v4.4.8。

在这种情况下，在项目根文件夹中打开 **composer.json**，并指定框架版本：

```
"require": {
    ...
    "codeigniter4/framework": "4.4.8"
},
```

然后，运行 `composer update` 命令。

优点

相对简单的安装; 易于更新。

缺点

更新后, 你仍需检查 **项目空间**中的文件更改 (根目录、app、public、writable)。

备注: 有一些第三方 CodeIgniter 模块可用于协助合并项目空间的更改: 在 Packagist 上探索。

结构

设置后项目中的文件夹:

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

翻译安装

如果你想利用系统消息翻译, 可以以类似的方式将它们添加到项目中。

在项目根目录的命令行中:

```
composer require codeigniter4/translations
```

每次执行 `composer update` 时, 这些都会与框架一起更新。

2.1.2 手动安装

- 安装
- 初始配置
- 升级

- 优点
- 缺点
- 结构
- 翻译安装

CodeIgniter 4 框架 仓库包含框架的已发布版本。它适用于不希望使用 Composer 的开发人员。

在 **app** 文件夹中开发你的应用程序, **public** 文件夹将是你面向公众的文档根目录。不要更改 **system** 文件夹中的任何内容!

备注: 这是最接近为 CodeIgniter 3 描述的安装技术。

安装

下载 最新版本, 并将其提取到成为项目根目录。

备注: 在 v4.4.0 之前，CodeIgniter 的自动加载器不允许在某些操作系统上的文件名中使用非法的特殊字符。可以使用的符号包括 /、_、.、:、\ 和空格。因此，如果你将 CodeIgniter 安装在包含特殊字符（如（、）等）的文件夹中，CodeIgniter 将无法正常工作。从 v4.4.0 开始，这个限制已经被移除。

初始配置

安装后, 需要进行一些初始配置。请参阅初始配置 以获取详细信息。

升级

下载框架的新副本, 然后替换 **system** 文件夹。

阅读升级说明 和变更日志，并检查重大变更和增强功能。

优点

下载并运行。

缺点

你需要自行检查 **项目空间**中的文件更改 (根目录、app、public、tests、writable) 并合并它们。

结构

设置后项目中的文件夹:

- app、public、tests、writable、system

翻译安装

如果你想利用系统消息翻译, 可以以类似的方式将它们添加到项目中。

下载 最新版本。提取下载的 zip 文件, 并将其中的 **Language** 文件夹内容复制到你的 **app/Languages** 文件夹中。

这需要重复执行以合并翻译的任何更新。

2.1.3 运行你的应用程序

- *初始配置*
  - *为你的站点 URI 进行配置*
  - *配置数据库连接设置*
  - *设置为开发模式*
  - *设置可写文件夹权限*
  - *检查 PHP ini 设置*
- *本地开发服务器*
- *使用 Apache 托管*

- 配置主配置文件
  - 启用 *mod_rewrite*
  - 设置文档根目录
- 使用虚拟主机托管
  - 启用 *vhost_alias_module*
  - 添加主机别名
  - 设置虚拟主机
  - 测试
- 使用子文件夹进行托管
  - 创建符号链接
  - 使用别名
  - 添加.*htaccess*
- 使用 *mod_userdir* 进行托管（共享主机）
- 删除 *index.php*
- 设置环境

- 使用 *Nginx* 托管
  - *default.conf*
  - 设置环境
- 部署到共享主机服务
- 引导应用程序

CodeIgniter 4 应用程序可以以多种不同的方式运行：托管在 Web 服务器上、使用虚拟化技术，或者使用 CodeIgniter 的命令行工具进行测试。本节介绍如何使用每种技术，并解释其中的一些优缺点。

重要： 在文件名的大小写方面应始终小心。许多开发人员在 Windows 或 macOS 上使用不区分大小写的文件系统进行开发。然而，大多数服务器环境使用区分大小写的文件系统。如果文件名大小写不正确，本地上正常工作的代码在服务器上将无法正常工作。

如果你是 CodeIgniter 的新手，请阅读用户指南的入门 部分，开始学习如何构建动态的 PHP 应用程序。祝你使用愉快！

初始配置

为你的站点 URI 进行配置

使用文本编辑器打开 **app/Config/App.php** 文件。

1. **$baseURL**

   将你的基本 URL 设置为 `$baseURL`。如果你需要更大的灵活性，可以在*.env* 文件中设置 baseURL，例如 `app.baseURL = 'http://example.com/'`。**始终在基本 URL 的末尾使用斜杠！**

   备注： 如果你没有正确设置 `baseURL`，在开发模式下，调试工具栏可能无法正确加载，网页可能需要更长的时间才能显示。

2. **$indexPage**

   如果你不想在站点 URI 中包含 **index.php**，请将 `$indexPage` 设置为 `''`。当框架生成你的站点 URI 时，将使用此设置。

   备注： 你可能需要配置你的 Web 服务器以访问不包含 **index.php** 的 URL。请参阅*CodeIgniter URL*。

配置数据库连接设置

如果你打算使用数据库，使用文本编辑器打开 **app/Config/Database.php** 文件并设置数据库配置。或者，你也可以在 **.env** 文件中设置这些配置。详细信息请参阅数据库配置。

设置为开发模式

如果不是在生产服务器上，请在 **.env** 文件中将 `CI_ENVIRONMENT` 设置为 `development`，以利用提供的调试工具。有关详细信息，请参阅设置开发模式。

重要: 在生产环境中，应禁用错误显示和任何其他仅用于开发的功能。在 CodeIgniter 中，可以通过将环境设置为 "production" 来实现。默认情况下，应用程序将在 "production" 环境下运行。另请参阅*ENVIRONMENT 常量*。

设置可写文件夹权限

如果你将使用 Web 服务器（例如 Apache 或 nginx）运行你的站点，你需要修改项目中的 **writable** 文件夹的权限，以便它可以被你的 Web 服务器使用的用户或帐户写入。

检查 PHP ini 设置

在 4.5.0 版本加入.

PHP ini 设置 更改 PHP 的行为。CodeIgniter 提供了一个命令来检查重要的 PHP 设置。

```
php spark phpini:check
```

推荐列显示了生产环境的推荐值。它们在开发环境中可能会有所不同。

备注: 如果你不能使用 spark 命令，可以在你的控制器中使用 `CheckPhpIni::run(false)`。

例如，

```
<?php
```

(续下页)

(接上页)

```
namespace App\Controllers;

use CodeIgniter\Security\CheckPhpIni;

class Home extends BaseController
{
    public function index(): string
    {
        return CheckPhpIni::run(false);
    }
}
```

本地开发服务器

CodeIgniter 4 自带一个本地开发服务器，利用 PHP 的内置 Web 服务器和 CodeIgniter 的路由功能。你可以使用以下命令在主目录中启动它：

```
php spark serve
```

这将启动服务器，你现在可以在浏览器中通过 http://localhost:8080 查看你的应用程序。

备注： 内置的开发服务器只应在本地开发机器上使用。它绝不能在生产服务器上使用。

如果你需要在除 localhost 之外的主机上运行站点，你首先需要将主机添加到你的 **hosts** 文件中。文件的确切位置因每个主要操作系统而异，但所有的类 Unix 类型的系统（包括 macOS）通常将文件保存在 **/etc/hosts** 中。

本地开发服务器可以使用三个命令行选项进行自定义：

- 你可以使用 `--host` CLI 选项指定要运行应用程序的不同主机：

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

- 默认情况下，服务器在端口 8080 上运行，但你可能有多个站点正在运行，或者已经有其他应用程序使用该端口。你可以使用 `--port` CLI 选项指定不同的端口：

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

- 你还可以使用 `--php` CLI 选项指定要使用的特定版本的 PHP，将其值设置为你要使用的 PHP 可执行文件的路径：

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

使用 Apache 托管

CodeIgniter 4 网站通常托管在 Web 服务器上。Apache HTTP Server 是“标准”平台，在我们的文档中假定使用它。

Apache 与许多平台捆绑在一起，但也可以从 Bitnami 下载捆绑了数据库引擎和 PHP 的版本。

配置主配置文件

启用 mod_rewrite

“mod_rewrite”模块允许在 URL 中不包含“index.php”，我们在用户指南中假定了这一点。

确保在主配置文件中启用（取消注释）重写模块，例如 **apache2/conf/httpd.conf**：

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

设置文档根目录

还要确保默认文档根目录的 `<Directory>` 元素也启用了这一点，在 `AllowOverride` 设置中：

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

使用虚拟主机托管

我们建议使用“虚拟主机”来运行你的应用程序。你可以为你工作的每个应用程序设置不同的别名，

启用 vhost_alias_module

确保在主配置文件中启用（取消注释）虚拟主机模块，例如 **apache2/conf/httpd.conf**：

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

添加主机别名

在你的“hosts”文件中添加主机别名，通常在 Unix 类型平台上为 **/etc/hosts**，在 Windows 上为 **c:\Windows\System32\drivers\etc\hosts**。

在文件中添加一行。例如，可以是 `myproject.local` 或 `myproject.test`:

```
127.0.0.1 myproject.local
```

设置虚拟主机

在虚拟主机配置中添加一个 `<VirtualHost>` 元素，用于你的 Web 应用程序，例如 **apache2/conf/extra/httpd-vhost.conf**：

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

    <Directory "/opt/lamp/apache2/myproject/public">
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

上述配置假设项目文件夹位于以下位置：

```
apache2/
   ├── myproject/       (项目文件夹)
   │       └── public/  (myproject.local 的 DocumentRoot)
   └── htdocs/
```

重启 Apache。

测试

使用上述配置，在浏览器中使用 URL **http://myproject.local/** 访问你的 Web 应用程序。

每当更改 Apache 配置时，都需要重新启动 Apache。

使用子文件夹进行托管

如果你希望使用类似 **http://localhost/myproject/** 的子文件夹 baseURL，有三种方法可以实现。

创建符号链接

将你的项目文件夹放置在以下位置，其中 **htdocs** 是 Apache 的文档根目录:

```
├── myproject/ (项目文件夹)
│       └── public/
└── htdocs/
```

导航到 **htdocs** 文件夹并创建符号链接，如下所示：

```
cd htdocs/
ln -s ../myproject/public/ myproject
```

使用别名

将你的项目文件夹放置在以下位置，其中 **htdocs** 是 Apache 的文档根目录:

```
├── myproject/ (项目文件夹)
│       └── public/
└── htdocs/
```

在主配置文件中添加以下内容，例如 **apache2/conf/httpd.conf**：

```
Alias /myproject /opt/lamp/apache2/myproject/public
<Directory "/opt/lamp/apache2/myproject/public">
    AllowOverride All
    Require all granted
</Directory>
```

重启 Apache。

添加.htaccess

最后的选择是在项目根目录中添加 **.htaccess** 文件。

不建议将项目文件夹放置在文档根目录中。但是，如果你没有其他选择，例如在共享服务器上，你可以使用此方法。

将你的项目文件夹放置在以下位置，其中 **htdocs** 是 Apache 的文档根目录，并创建 **.htaccess** 文件:

```
└── htdocs/
    └── myproject/ (项目文件夹)
        ├── .htaccess
        └── public/
```

并将 **.htaccess** 编辑如下：

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteRule ^(.*)$ public/$1 [L]
</IfModule>
```

(续下页)

(接上页)

```
<FilesMatch "^\.">
    Require all denied
    Satisfy All
</FilesMatch>
```

并且移除 **public/.htaccess** 中的重定向设置：

```
--- a/public/.htaccess
+++ b/public/.htaccess
@@ -16,16 +16,6 @@ Options -Indexes
    # http://httpd.apache.org/docs/current/mod/mod_rewrite.html
↪#rewritebase
    # RewriteBase /

-   # Redirect Trailing Slashes...
-   RewriteCond %{REQUEST_FILENAME} !-d
-   RewriteCond %{REQUEST_URI} (.+)/$
-   RewriteRule ^ %1 [L,R=301]
-
-   # Rewrite "www.example.com -> example.com"
-   RewriteCond %{HTTPS} !=on
-   RewriteCond %{HTTP_HOST} ^www\.(.+)$ [NC]
-   RewriteRule ^ http://%1%{REQUEST_URI} [R=301,L]
-
    # 检查用户是否尝试访问有效文件，
    # 例如图像或 css 文档，如果不是真的则将请求发送到前端控制器␣
↪index.php
```

使用 mod_userdir 进行托管（共享主机）

在共享托管环境中，常见做法是使用 Apache 模块 “mod_userdir” 自动启用每个用户的虚拟主机。需要额外的配置才能允许 CodeIgniter4 从这些每个用户目录中运行。

以下假设服务器已经配置为 mod_userdir。有关启用此模块的指南，请参阅 Apache 文档中的 相关部分。

由于 CodeIgniter4 默认情况下期望服务器在 **public/index.php** 中找到框架前端控制器，因此你必须指定此位置作为替代位置以搜索请求（即使 CodeIgniter4 安装在每个用户的

Web 目录中)。

默认的用户 Web 目录 **~/public_html** 由 `UserDir` 指令指定，通常位于 **apache2/mods-available/userdir.conf** 或 **apache2/conf/extra/httpd-userdir.conf** 中：

```
UserDir public_html
```

因此，你需要配置 Apache 在尝试提供默认服务之前首先查找 CodeIgniter 的 public 目录：

```
UserDir "public_html/public" "public_html"
```

确保还为 CodeIgniter 的 public 目录指定选项和权限。一个 **userdir.conf** 可能如下所示：

```
<IfModule mod_userdir.c>
    UserDir "public_html/public" "public_html"
    UserDir disabled root

    <Directory /home/*/public_html>
        AllowOverride All
        Options MultiViews Indexes FollowSymLinks
        <Limit GET POST OPTIONS>
            # Apache <= 2.2:
            # Order allow,deny
            # Allow from all

            # Apache >= 2.4:
            Require all granted
        </Limit>
        <LimitExcept GET POST OPTIONS>
            # Apache <= 2.2:
            # Order deny,allow
            # Deny from all

            # Apache >= 2.4:
            Require all denied
        </LimitExcept>
    </Directory>

    <Directory /home/*/public_html/public>
        AllowOverride All
```

(续下页)

(接上页)

```
        Options MultiViews Indexes FollowSymLinks
        <Limit GET POST OPTIONS>
            # Apache <= 2.2:
            # Order allow,deny
            # Allow from all

            # Apache >= 2.4:
            Require all granted
        </Limit>
        <LimitExcept GET POST OPTIONS>
            # Apache <= 2.2:
            # Order deny,allow
            # Deny from all

            # Apache >= 2.4:
            Require all denied
        </LimitExcept>
    </Directory>
</IfModule>
```

删除 index.php

请参阅*CodeIgniter URL*。

设置环境

请参阅处理多个环境。

使用 Nginx 托管

Nginx 是第二常用的用于 Web 托管的 HTTP 服务器。以下是一个在 Ubuntu Server 上使用 PHP 8.1 FPM（Unix 套接字）的示例配置。

default.conf

此配置使 URL 中不包含 “index.php”，并对以 “.php” 结尾的 URL 使用 CodeIgniter 的 “404 - File Not Found”。

```
server {
    listen 80;
    listen [::]:80;

    server_name example.com;

    root  /var/www/example.com/public;
    index index.php index.html index.htm;

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

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;

        # 使用 php-fpm:
        fastcgi_pass unix:/run/php/php8.1-fpm.sock;
        # 使用 php-cgi:
        # fastcgi_pass 127.0.0.1:9000;
    }

    error_page 404 /index.php;

    # 禁止访问隐藏文件，如 .htaccess
    location ~ /\. {
        deny all;
    }
}
```

设置环境

请参阅处理多个环境。

部署到共享主机服务

参见*Deployment*。

引导应用程序

在某些情况下，你可能希望加载框架而不实际运行整个应用程序。这对于对项目进行单元测试非常有用，但也可能对使用第三方工具分析和修改代码很有用。框架提供了一个专门用于此场景的独立引导脚本：**system/Test/bootstrap.php**。

在引导过程中，大部分项目路径都会被定义。你可以使用预定义的常量来覆盖这些路径，但是当使用默认值时，请确保你的路径与安装方法的预期目录结构对齐。

2.1.4 故障排除

以下是一些常见的安装问题及建议的解决方法。

- *如何知道我的安装是否正常工作?*
- *我的 URL 中必须包含 index.php*
- *只有默认页面加载*
- *未指定输入文件*
- *我的应用在本地正常工作, 但在生产服务器上不正常*
- *教程中的所有页面都显示 404 错误:(*
- *“Whoops!”页面是什么?*
- *CodeIgniter 错误日志*

如何知道我的安装是否正常工作?

在项目根目录的命令行中:

```
php spark serve
```

然后在浏览器中打开 `http://localhost:8080` 应该可以看到默认的欢迎页面:

我的 URL 中必须包含 index.php

如果像 `/mypage/find/apple` 这样的 URL 不起作用, 但类似的 `/index.php/mypage/find/apple` 可以正常访问, 这可能是因为你的 **.htaccess** 规则 (用于 Apache) 没有正确设置, 或者 Apache 的 **httpd.conf** 中的 `mod_rewrite` 扩展被注释掉了。请参阅*删除 index.php 文件*。

只有默认页面加载

如果你发现无论在 URL 中输入什么, 只有默认页面加载, 这可能是因为你的服务器不支持提供搜索引擎友好 URL 所需的 REQUEST_URI 变量。作为第一步, 打开 **app/Config/App.php** 文件, 查看 URI 协议信息。它会建议你尝试一些替代设置。如果在你尝试后仍然不起作用, 则需要强制 CodeIgniter 在 URL 中添加问号 (?)。要做到这一点, 请打开 **app/Config/App.php** 文件, 并将此内容:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class App extends BaseConfig
{
    // ...

    public string $indexPage = 'index.php';

    // ...
}
```

改为:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
```

(续下页)

(接上页)

```
class App extends BaseConfig
{
    // ...

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

    // ...
}
```

未指定输入文件

如果看到 “No input file specified” , 请尝试如下更改重写规则 (在 `index.php` 后添加 ?):

```
RewriteRule ^([\s\S]*)$ index.php?/$1 [L,NC,QSA]
```

我的应用在本地正常工作, 但在生产服务器上不正常

确保文件夹和文件名的大小写与代码匹配。

许多开发者在 Windows 或 macOS 的大小写不敏感的文件系统上开发。然而, 大多数服务器环境使用大小写敏感的文件系统。

例如, 当你有 **app/Controllers/Product.php** 时, 必须使用 `Product` 作为短类名, 而不是 `product`。

如果文件名大小写不正确, 服务器上就找不到该文件。

教程中的所有页面都显示 404 错误:(

你不能使用 PHP 内置的 Web 服务器来按照教程进行操作。它无法处理所需的 **.htaccess** 文件以正确路由请求。

解决方案：使用 Apache 来提供你的网站，或者使用内置的 CodeIgniter 等效命令 `php spark serve` 从你的项目根目录运行。

"Whoops!" 页面是什么?

你会发现你的应用程序显示一个 "Whoops!" 页面, 然后是文本行 "We seem to have hit a snag. Please try again later…"。

这表示你处于生产模式并遇到了不可恢复的错误, 我们不想让 webapp 的查看者看到, 以实现更好的安全性。

你可以在日志文件中看到错误。请参阅下面的 "CodeIgniter 错误日志"。

如果在开发过程中达到此页面, 应将环境设置为 "development" (在 **.env** 中)。有关更多详细信息, 请参阅设置开发模式。之后, 重新加载页面。你将看到错误和回溯。

CodeIgniter 错误日志

参阅*CodeIgniter 错误日志*。

2.1.5 部署

- 优化
  - *spark optimize*
  - *Composer* 优化
    - 移除开发包
    - 指定发现的包
  - 配置缓存
  - *FileLocator* 缓存
  - *PHP* 预加载
    - 需求
    - 配置
- 部署到共享主机服务
  - 指定文档根目录
  - 使用两个目录

- *添加.htaccess*

优化

在将你的 CodeIgniter 应用程序部署到生产环境之前，有若干事情可以做以提高应用程序的运行效率。

本节将描述 CodeIgniter 提供的优化功能。

spark optimize

在 4.5.0 版本加入.

`spark optimize` 命令执行以下优化操作：

- *移除开发包*
- 启用*配置缓存*
- 启用*FileLocator 缓存*

Composer 优化

移除开发包

当你部署时，不要忘记运行以下命令：

```
composer install --no-dev
```

上述命令将移除仅在开发环境中需要的 Composer 包，这些包在生产环境中并不需要。这将极大地减少 vendor 目录的大小。

指定发现的包

如果启用了 Composer 包自动发现功能，则在需要时所有的 Composer 包都会被扫描。但没有必要扫描那些不是 CodeIgniter 包的包，所以指定需要扫描的包可以防止不必要的扫描。

参见*指定 Composer 包*。

配置缓存

重要: 一旦被缓存，配置值在缓存被删除之前将不会被改变，即使配置文件或 **.env** 被修改。

缓存配置对象可以提高性能。然而，在更改配置值时必须手动删除缓存。

参见配置缓存。

FileLocator 缓存

缓存 FileLocator 找到的文件路径可以提高性能。然而，在添加/删除/更改文件路径时必须手动删除缓存。

参见*FileLocator 缓存*。

PHP 预加载

使用 PHP 预加载，你可以指示服务器在启动时将核心文件如函数和类加载到内存中。这意味着这些元素在所有请求中都可立即使用，跳过了通常的加载过程，从而提升了应用程序的性能。然而，这会带来增加内存使用的代价，并且需要重启 PHP 引擎才能生效。

备注: 如果你想使用 预加载，我们提供了 预加载脚本。

需求

使用预加载需要一个专门的 PHP 处理器。通常情况下，Web 服务器配置为使用一个 PHP 处理器，所以一个应用程序需要一个专用的 Web 服务器。如果你想在一个 Web 服务器上为多个应用使用预加载，请配置你的服务器以使用带有多个 PHP 处理器的虚拟主机，例如多个 PHP-FPM，每个虚拟主机使用一个 PHP 处理器。预加载通过读取在 `opcache.preload` 中指定的文件将相关定义保留在内存中。

备注: 参见*使用一个 CodeIgniter 安装运行多个应用程序* 以使用一个核心的 CodeIgniter4

处理多个应用。

配置

打开 `php.ini` 或 `xx-opcache.ini` （如果你将 INI 配置分离开来），建议设置 `opcache.preload=/path/to/preload.php` 和 `opcache.preload_user=myuser`。

备注： `myuser` 是在你的 Web 服务器中运行的用户。如果你想找到分离的 INI 配置的位置，只需运行 `php --ini` 或打开 `phpinfo()` 文件并搜索 *Additional .ini files parsed*。

确保你使用的是 appstarter 安装。如果使用手动安装，你必须更改 `include` 路径中的目录。

```
<?php

// ...

class preload
{
    /**
     * @var array Paths to preload.
     */
    private array $paths = [
        [
            'include' => __DIR__ . '/system', // <== change this␣
↪line to where CI is installed
            // ...
        ],
    ];

    // ...
}

// ...
```

部署到共享主机服务

重要：index.php 不再位于项目的根目录中！它已移动到 **public** 目录中，这样更安全，并能更好地分离组件。

这意味着你应该配置你的 Web 服务器指向项目的 **public** 目录，而不是项目根目录。

指定文档根目录

最好的方式是在服务器配置中将文档根目录设置为 **public** 目录：

```
└── example.com/ (项目目录)
    └── public/ (文档根目录)
```

请与你的主机服务提供商确认是否可以更改文档根目录。如果不能更改文档根目录，请参考下一种方法。

使用两个目录

第二种方式是使用两个目录，并调整路径。一个用于应用程序，另一个是默认的文档根目录。

将 **public** 目录的内容上传到 **public_html** （默认的文档根目录），其他文件上传到用于应用程序的目录:

```
├── example.com/ (用于应用程序)
│       ├── app/
│       ├── vendor/ (或 system/)
│       └── writable/
└── public_html/ (默认的文档根目录)
        ├── .htaccess
        ├── favicon.ico
        ├── index.php
        └── robots.txt
```

参见 Install CodeIgniter 4 on Shared Hosting (cPanel) 获取详细信息。

添加.htaccess

最后一招是在项目根目录中添加 **.htaccess** 文件。

不建议将项目文件夹放在文档根目录中。然而，如果没有其他选择，你可以使用这种方法。

按照以下方式放置你的项目文件夹，其中 **public_html** 是文档根目录，并创建 **.htaccess** 文件：

```
└── public_html/ (默认的文档根目录)
    └── example.com/ (项目文件夹)
        ├── .htaccess  └── public/
```

并按如下内容编辑 **.htaccess**：

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteRule ^(.*)$ public/$1 [L]
</IfModule>

<FilesMatch "^\.">
    Require all denied
    Satisfy All
</FilesMatch>
```

然后，删除 **public/.htaccess** 中的重定向设置：

```
--- a/public/.htaccess
+++ b/public/.htaccess
@@ -16,16 +16,6 @@ Options -Indexes
    # http://httpd.apache.org/docs/current/mod/mod_rewrite.html
↪#rewritebase
    # RewriteBase /

-   # Redirect Trailing Slashes...
-   RewriteCond %{REQUEST_FILENAME} !-d
-   RewriteCond %{REQUEST_URI} (.+)/$
-   RewriteRule ^ %1 [L,R=301]
-
```

(续下页)

(接上页)

```
-   # Rewrite "www.example.com -> example.com"
-   RewriteCond %{HTTPS} !=on
-   RewriteCond %{HTTP_HOST} ^www\.(.+)$ [NC]
-   RewriteRule ^ http://%1%{REQUEST_URI} [R=301,L]
-
    # 检查用户是否试图访问有效文件，如图片或 css␣
↪文档，如果不是真的，则将
    # 请求发送到前端控制器 index.php
```

2.1.6 变更记录

版本 4.6

CodeIgniter4 所有版本的列表

查看所有更改。

版本 4.6.0

发布日期：2025 年 1 月 19 日

CodeIgniter4 的 4.6.0 版本发布

- 重大变更
  - 行为变更
    - 异常处理
    - 过滤器变更
    - 注册器
    - *Time::createFromTimestamp()*
    - 支持微秒的时间处理
    - *Time::setTimestamp()*
    - 非 *HTML* 请求的错误报告

        - 会话 *ID (SID)*
        - 响应头处理
    - 接口变更
    - 方法签名变更
        - 移除类型定义
    - 移除已弃用项
- 功能增强
    - 发布器
    - 异常处理
    - 命令行工具
    - 路由
    - 协商器
    - 分页
    - 数据库
        - 其他改进
    - 类库
    - 其他
- 消息变更
- 变更列表
    - 异常处理
- 已弃用项
- 已修复的 *Bug*

重大变更

行为变更

异常处理

异常类已重新设计。详细信息请参阅异常设计。以下是相应的破坏性变更:

- `Validation::setRule()` 现在抛出 `CodeIgniter\Exceptions\InvalidArgumentException` 而非 `TypeError`
- `CriticalError` 现在继承自 `CodeIgniter\Exceptions\RuntimeException` 而非 `Error`
- `DatabaseException` 现在继承自 `CodeIgniter\Exceptions\RuntimeException` 而非 `Error`
- `ConfigException` 现在继承自 `CodeIgniter\Exceptions\RuntimeException` 而非 `CodeIgniter\Exceptions\CriticalError`
- `TestException` 现在继承自 `CodeIgniter\Exceptions\LogicException` 而非 `CodeIgniter\Exceptions\CriticalError`

过滤器变更

`Filters` 类已修改，允许在 before 或 after 阶段使用不同参数多次运行相同过滤器。详细信息请参阅*Upgrading Guide*。

注册器

新增检查以防止注册器自动发现机制重复运行。若重复执行将抛出异常。详细信息请参阅注册器的脏数据修复。

Time::createFromTimestamp()

`Time::createFromTimestamp()` 处理时区的方式已变更。若未显式传递 `$timezone` 参数，实例时区将设为 UTC（此前使用系统默认时区）。详细信息请参阅*Upgrading Guide*。

支持微秒的时间处理

修复了 `Time` 类中部分方法丢失微秒的缺陷。详细信息请参阅*Upgrade Guide*。

Time::setTimestamp()

`Time::setTimestamp()` 的行为已修正。详细信息请参阅*Upgrade Guide*。

非 HTML 请求的错误报告

旧版本中，当请求不接受 HTML 时，CodeIgniter 仅在 `development` 和 `testing` 环境下显示错误详情。

由于在自定义环境中无法显示错误详情，现修正此行为：只要 PHP 配置中的 `display_errors` 启用即显示错误详情。

此修正后，HTML 请求与非 HTML 请求的错误详情显示条件现已统一。

会话 ID (SID)

现在 `Session` 类库强制使用 PHP 默认的 32 字符 SID（每字符 4 位熵值）。详细信息请参阅*Upgrade Guide*。

响应头处理

通过 `Response` 类设置的响应头将覆盖通过 PHP `header()` 函数设置的响应头。

旧版本中，`Response` 类设置的响应头会追加到现有响应头（无法修改），当存在互斥指令的同名响应头时可能导致意外行为。

例如，session 会自动通过 `header()` 函数设置响应头：

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
```

若再次设置 **Expires** 响应头将导致重复：

```
$response->removeHeader('Expires'); // 无效操作
return $response->setHeader('Expires', 'Sun, 17 Nov 2024 14:17:37␣
↪GMT');
```

响应头结果：

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
// ...
Expires: Sun, 17 Nov 2024 14:17:37 GMT
```

此时浏览器可能无法正确识别有效响应头。本版本变更后，旧响应头将被覆盖：

```
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Expires: Sun, 17 Nov 2024 14:17:37 GMT
```

接口变更

备注: 只要你未扩展相关 CodeIgniter 核心类或实现这些接口，所有变更均向后兼容且无需调整。

- **Router:** 以下方法已加入 `RouteCollectionInterface` 接口：
  - `getDefaultNamespace()`
  - `getRoutesOptions()`
  - `setHTTPVerb()`
  - `isFiltered()`
  - `getFiltersForRoute()`

方法签名变更

- **Router:** DefinedRouteCollector 的构造函数参数类型由 RouteCollection 改为 RouteCollectionInterface
- **View:** renderSection() 方法的返回类型改为 string，且不再调用 echo
- **Time:** createFromTimestamp() 的首个参数类型由 int 改为 int|float，并新增返回类型 static
- **Helpers:** 更新了 character_limiter() 的参数命名。若使用命名参数需更新函数调用

移除类型定义

- **Database:**
  - 移除 BaseConnection::escapeIdentifier() 首个参数的 string 类型
  - 移除 BaseConnection::getFieldNames() 和 SQLite3\Connection::getFieldNames() 首个参数的 string 类型
  - 移除 BaseConnection::_listColumns()、MySQLi\Connection::_listColumns()、OCI8\Connection::_listColumns()、Postgre\Connection::_listColumns()、SQLSRV\Connection::_listColumns()、SQLite3\Connection::_listColumns() 首个参数的 string 类型

移除已弃用项

- **API:** 移除 CodeIgniter\API\ResponseTrait 中已弃用的 failValidationError() 方法，改用 failValidationErrors()
- **HTTP:** 移除 CodeIgniter\HTTP\Response 和 ResponseInterface 中已弃用的 getReason() 方法，改用 getReasonPhrase()
- **Logger:** 移除已弃用的 CodeIgniter\Log\Logger::cleanFilenames() 和 CodeIgniter\Test\TestLogger::cleanup() 方法，改用 clean_path() 函数

- **Router:** 移除已弃用的 `CodeIgniter\Router\Exceptions\RedirectException` 异常类，改用 `CodeIgniter\HTTP\Exceptions\RedirectException`
- **Constants:** 移除已弃用的 `EVENT_PRIORITY_*` 常量，改用类常量 `CodeIgniter\Events\Events::PRIORITY_LOW`、`CodeIgniter\Events\Events::PRIORITY_NORMAL` 和 `CodeIgniter\Events\Events::PRIORITY_HIGH`
- **View:** 移除已弃用的 `CodeIgniter\View\View::$currentSection` 属性
- **Config:** 移除已弃用的 `Config\Cache::$storePath` 属性，改用 `Config\Cache::$file['storePath']`
- **Formatter:** 移除已弃用的 `Config\Format::getFormatter()` 方法，改用 `CodeIgniter\Format\Format::getFormatter()`
- **Security:** 移除 `Config\Security::$samesite` 配置项，改用 `Config\Cookie::$samesite`
- **Cookie:** 移除 `CodeIgniter\Cookie\CookieStore` 中的 `dispatch()`、`setRawCookie()`、`setCookie()` 方法，这些方法已移至 `CodeIgniter\HTTP\ResponseTrait`

功能增强

发布器

- `Publisher::discover()` 新增第二个参数（namespace）用于指定搜索发布器的命名空间。详细信息请参阅在指定命名空间中发现。

异常处理

异常类已重新设计。详细信息请参阅异常设计。新增以下异常类：

- `CodeIgniter\Exceptions\LogicException`
- `CodeIgniter\Exceptions\RuntimeException`
- `CodeIgniter\Exceptions\BadFunctionCallException`
- `CodeIgniter\Exceptions\BadMethodCallException`

- `CodeIgniter\Exceptions\InvalidArgumentException`

新增以下异常接口：

- `CodeIgniter\Files\Exceptions\ExceptionInterface`
- `CodeIgniter\HTTP\Exceptions\ExceptionInterface`
- `CodeIgniter\Router\Exceptions\ExceptionInterface`

非 HTML 响应的异常显示现基于 PHP 的 `display_errors` 设置，而非硬编码环境判断。

命令行工具

- `spark routes` 和 `spark filter:check` 命令现显示过滤器参数
- `spark filter:check` 命令现显示过滤器类名
- 新增 `spark lang:sync` 命令用于同步翻译文件。详细信息请参阅通过命令同步翻译文件
- `spark phpini:check` 命令新增可选 `opcache` 参数，用于显示 opcache 设置信息

路由

- 现可在限制路由时指定多个主机名

协商器

- 新增特性开关 `Feature::$strictLocaleNegotiation` 用于启用严格区域匹配。旧版本中，对于 `Accept-language: en-US,en-GB;q=0.9` 的请求会返回首个允许的语言 `en`（而非精确匹配 `en-US` 或 `en-GB`）。设为 `true` 后启用基于语言代码（'en' - ISO 639-1）和区域代码（'en-US' - ISO 639-1 + ISO 3166-1 alpha）的精确匹配。

分页

- 新增获取当前页总条目数和范围的功能。详细信息请参阅显示条目数。

数据库

其他改进

- 为 MySQLi 新增 `foundRows` 配置项以使用 `MYSQLI_CLIENT_FOUND_ROWS`
- 新增 `BaseConnection::resetTransStatus()` 方法用于重置事务状态。详细信息请参阅重置事务状态
- SQLite3 新增 `synchronous` 配置项，用于调整事务期间磁盘刷写策略。结合 `WAL` 日志模式时可有效优化性能

类库

- **File:** `File` 类新增 `getSizeByBinaryUnit()` 和 `getSizeByMetricUnit()` 方法。详细信息请参阅*File::getSizeByBinaryUnit()* 和*File::getSizeByMetricUnit()*。
- **FileCollection:** `FileCollection` 类新增 `retainMultiplePatterns()` 方法。详细信息请参阅*FileCollection::retainMultiplePatterns()*。
- **Validation:** `FileRules` 类新增 `min_dims` 验证规则。详细信息请参阅*Validation*。
- **Validation:** `is_unique` 和 `is_not_unique` 规则现允许在首个参数中指定 `dbGroup`。详细信息请参阅*Validation*。

其他

- **Filters:** 现可在 before 或 after 阶段使用不同参数多次执行同一过滤器
- **Services:** 新增 `BaseService::resetServicesCache()` 方法用于重置服务缓存。详细信息请参阅重置服务缓存
- **Errors:** 新增默认的“400 Bad Request”错误页面

消息变更

- 新增 `Validation.min_dims` 消息
- 新增 `Errors.badRequest` 和 `Errors.sorryBadRequest` 消息

变更列表

异常处理

异常类已重新设计。详细信息请参阅异常设计。以下是相应变更:

- 缓存处理器类的 `deleteMatching()` 方法现抛出 `CodeIgniter\Exceptions\BadMethodCallException` 而非 `Exception`
- `Cache\ResponseCache::get()` 现抛出 `CodeIgniter\Exceptions\RuntimeException` 而非 `Exception`
- 原抛出 `RuntimeException` 的类现改为抛出 `CodeIgniter\Exceptions\RuntimeException`
- 原抛出 `InvalidArgumentException` 的类现改为抛出 `CodeIgniter\Exceptions\InvalidArgumentException`
- 原抛出 `LogicException` 的类现改为抛出 `CodeIgniter\Exceptions\LogicException`
- 原抛出 `BadMethodCallException` 的类现改为抛出 `CodeIgniter\Exceptions\BadMethodCallException`
- 原抛出 `BadFunctionCallException` 的类现改为抛出 `CodeIgniter\Exceptions\BadFunctionCallException`
- `RedirectException` 现继承自 `CodeIgniter\Exceptions\RuntimeException` 而非 `Exception`
- `PageNotFoundException` 现继承自 `CodeIgniter\Exceptions\RuntimeException` 而非 `OutOfBoundsException`

已弃用项

- **Filters:**
  - 弃用 `Filters` 的 `$arguments` 和 `$argumentsClass` 属性（不再使用）
  - 弃用 `Filters::getArguments()` 方法（不再使用）
- **File:**
  - 弃用 `File` 的 `getSizeByUnit()` 方法，改用 `getSizeByBinaryUnit()` 或 `getSizeByMetricUnit()`

已修复的 Bug

- **Response:**
  - 现在通过 `Response` 类设置的响应头将优先覆盖通过 PHP `header()` 函数手动设置的响应头
- **View:** 为 `View::excerpt()` 新增多字节字符串支持
- **Helpers:** 为 `excerpt()` 新增多字节字符串支持

完整缺陷修复列表请查阅仓库的 CHANGELOG.md。

版本 4.5.8

发布日期：2025 年 1 月 19 日

CodeIgniter4 的 4.5.8 版本发布

- *安全更新*
- *已修复的 Bug*

安全更新

- **Header:** 修复了标头名称和值的验证问题。详细信息请查阅安全通告 GHSA-x5mq-jjr3-vmx6。

已修复的 Bug

- **Database:** 修复了当 `Postgre` 和 `SQLSRV` 驱动中前次查询调用失败时，`Builder::affectedRows()` 抛出错误的缺陷。
- **Security:** 修复了因错误格式输入导致 CSRF 令牌验证失败时直接返回通用 HTTP 500 状态码（而非优雅处理输入）的缺陷。

完整缺陷修复列表请查阅仓库的 CHANGELOG.md。

版本 4.5.7

发布日期：2024 年 12 月 31 日

CodeIgniter4 的 4.5.7 版本发布

- *已修复的 Bug*

已修复的 Bug

- **Common:** 修复了 `helper()` 方法在有效的命名空间 helper 上可能抛出 *FileNotFoundException* 的 bug。
- **Forge:** 修复了 *SQLite3* 的 Forge 在调用 `dropColumn()` 时总是返回 *false* 的问题。

查看仓库的 CHANGELOG.md 以获取完整的已修复 bug 列表。

版本 4.5.6

发布日期：2024 年 12 月 28 日

CodeIgniter4 的 4.5.6 版本发布

- *已修复的 Bug*

已修复的 Bug

- **RequestTrait:** 修复了 `fetchGlobal()` 方法在处理以列表形式存储的数据时，无法通过数字键处理数据的 bug。
- **Session 类库:** 会话初始化的调试消息现在使用正确的日志类型“debug”而不是“info”。
- **验证:** 修复了 `getValidated()` 方法在使用多个星号的验证规则时未返回有效数据的 bug。
- **数据库:** 修复了 `like()` 方法在处理带重音字符时的大小写不敏感选项。
- **解析器:** 修复了导致相同键名被第一个定义的键名替换的 bug。
- **DownloadResponse:** 修复了无法设置自定义缓存头的 bug。现在也可以使用 `setCache()` 方法。
- **DownloadResponse:** 修复了涉及发送自定义“Expires-Disposition”头的 bug。
- **路由:** 修复了当 *Routing::$translateURIDashes* 设置为 *true* 并且使用闭包定义路由时，*str_replace()* 中的 TypeError。
- **验证:** 修复了复杂语言字符串未正确处理的问题。
- **CURLRequest:** 添加了对使用非 1.1 版本的 HTTP 处理代理响应的支持。
- **数据库:** 修复了 `Postgre\Connection::reconnect()` 方法在连接尚未建立时抛出错误的 bug。
- **模型:** 修复了 `Model::getIdValue()` 方法在使用主键的数据映射时，无法正确识别 `Entity` 对象中主键的 bug。
- **数据库:** 修复了预处理语句中的一个错误，以正确处理二进制数据。

查看仓库的 CHANGELOG.md 以获取完整的已修复 bug 列表。

版本 4.5.5

发布日期：2024 年 9 月 7 日

CodeIgniter4 的 4.5.5 版本发布

- 修复的错误

修复的错误

- **URL 辅助函数:** 修复了 *auto_link()* 中正则表达式过时的错误。通过此修复，现在使用与 CodeIgniter 3 相同的正则表达式。

请参阅仓库的 CHANGELOG.md 以获取修复错误的完整列表。

版本 4.5.4

发布日期：2024 年 7 月 27 日

CodeIgniter4 的 4.5.4 版本发布

- 修复的错误

修复的错误

- **路由：** 修复了一个错误，即传递给 `$routes->group()` 的过滤器没有合并到传递给内部路由的过滤器中。
- **CURLRequest：** 修复了一个错误，即在进行请求时，配置数组中的 `version` 字符串无法使用。

请参阅仓库的 CHANGELOG.md 以获取修复错误的完整列表。

版本 4.5.3

发布日期：2024 年 6 月 25 日

CodeIgniter4 的 4.5.3 版本发布

- 修复的错误

修复的错误

请查看仓库的 CHANGELOG.md 以获取修复错误的完整列表。

版本 4.5.2

发布日期：2024 年 6 月 10 日

CodeIgniter4 的 4.5.2 版本发布

- 消息变更
- 修复的错误

消息变更

- 添加了 `Security.insecureCookie` 消息。

修复的错误

请查看仓库的 CHANGELOG.md 以获取修复错误的完整列表。

版本 4.5.1

发布日期：2024 年 4 月 14 日

CodeIgniter4 的 4.5.1 版本发布

- 修复的错误

修复的错误

请参阅仓库中的 CHANGELOG.md 获取完整的错误修复列表。

版本 4.5.0

发布日期：2024 年 4 月 7 日

CodeIgniter4 的 4.5.0 版本发布

- 亮点
- 增强功能
  - 必需过滤器
  - 路由
  - 命令
  - 测试
  - 数据库
    - 查询生成器
    - 其他
  - 模型
    - 模型字段转换
    - *findAll(0)* 行为
    - *$updateOnlyChanged*

    * 保存日期
  - 库
  - 其他
- 重大变更
  - 行为更改
    * 小写 *HTTP* 方法名
    * 过滤器执行顺序
    * 嵌套路由组和选项
    * *API\ResponseTrait*
    * *Factories* 类
    * 其他
  - 接口更改
  - 方法签名更改
    * 设置 *Cookie*
    * *FileLocatorInterface*
    * 返回类型更改
    * 传统验证规则
    * 其他
  - 移除的弃用项
    * *Request*
    * *Filters*
    * 数据库
    * 模型
    * *Response*
    * *Security*
    * *CodeIgniter*

    * 测试
    * *Spark* 命令
    * 其他
- 消息更改
- 更改
- 弃用项
- 修复的错误

亮点

- 更新最低 PHP 要求至 8.1。
- 更新最低 PHPUnit 要求至 10.5。
- **CORS 过滤器**（由 kenjis 贡献）详见*跨域资源共享 (CORS)*。
- 用于在生产环境中提升性能的 **spark optimize** 命令（由 kenjis 贡献）。见*spark optimize*。

增强功能

必需过滤器

引入了新的必需过滤器。这些是特殊的过滤器，它们在其他种类的过滤器之前和之后应用，并且即使路由不存在也总会应用。

以下现有功能已重新实现为必需过滤器。

- 强制全局安全请求
- 网页缓存
- 性能指标
- 调试工具栏

调试工具栏使用的 Benchmark **Timers** 现在收集 必需前过滤器和 必需后过滤器数据。

基准测试点已经更改：

- 之前:
  - `bootstrap`: 创建 Request 和 Response 对象，事件 `pre_system`，实例化 RouteCollection 对象，加载 Routes 文件，实例化 Router 对象，
  - `routing`: 路由，
- 之后:
  - `bootstrap`: 创建 Request 和 Response 对象，事件 `pre_system`。
  - `required_before_filters`: 实例化 Filters 对象，运行 必需前过滤器。
  - `routing`: 实例化 RouteCollection 对象，加载 Routes 文件，实例化 Router 对象，路由，

路由

- **AutoRouting 改进:** 添加了 `$translateUriToCamelCase` 选项，允许使用驼峰式（CamelCase）控制器和方法名称。详见*URI 转驼峰命名*。
- **其他改进:**
  - 添加了 `$multipleSegmentsOneParam` 选项。启用该选项时，匹配多个段的占位符（如 `(:any)`）将直接作为一个参数传递，即使它包含多个段。详见*多 URI 段作为单一参数*。
  - 现在你在 `$override404` 中设置的 404 控制器方法也会接收到 `PageNotFoundException` 消息作为第一个参数。
  - 现在你可以使用 `__invoke()` 方法作为默认方法。详见*默认方法*。

命令

- 添加了 `spark optimize` 命令来优化生产环境的配置。详见*spark optimize*。
- 添加了 `spark make:test` 命令来生成测试文件的骨架。详见*make:test*。
- 添加了 `spark config:check` 命令来检查配置值。详见*确认配置值*。
- 添加了 `spark phpini:check` 命令来检查重要的 PHP ini 设置。详见*检查 PHP ini 设置*。
- 添加了 `spark lang:find` 命令来更新翻译键。详见*通过命令生成翻译文件*。
- `spark db:table` 命令中已添加 `--dbgroup` 选项。详见*Database Commands*。

测试

- **DomParser:** 添加了新方法 `seeXPath()` 和 `dontSeeXPath()`，允许用户使用复杂表达式直接与 DOMXPath 对象交互。
- **CLI:** 添加了新类 `InputOutput`，现在如果你使用 `MockInputOutput`，可以更轻松地为命令编写测试。详见*使用 MockInputOutput*。
- **Fabricator:** Fabricator 类现在有 `setUnique()`、`setOptional()` 和 `setValid()` 方法，以允许在生成值之前对每个字段调用 Faker 的修饰符。
- **TestResponse:** TestResponse 不再继承 `PHPUnit\Framework\TestCase`，因为它不是一个测试。断言的返回类型现在本地化为 `void`。

数据库

查询生成器

limit(0) 行为

- 添加了一个功能标志 `Feature::$limitZeroAsAll` 来修正 `limit(0)` 的错误行为。
- 如果在 SQL 语句中指定了 `LIMIT 0`，则返回 0 条记录。然而，查询生成器中存在一个错误，如果指定了 `limit(0)`，生成的 SQL 语句将没有 `LIMIT` 子句，并返回所有记录。
- 建议在 **app/Config/Feature.php** 中将 `$limitZeroAsAll` 设置为 `false`，因为这个错误行为将在未来版本中修复。详见*findAll(0) 行为*。

其他

- 支持包含点（`.`）的数据库名称。

模型

模型字段转换

添加了一个功能来将从数据库检索到的数据转换为合适的 PHP 类型。详见*模型字段类型转换*。

findAll(0) 行为

- 添加了一个功能标志 `Feature::$limitZeroAsAll` 来修正 Query Builder 的 `limit(0)` 的错误行为。详见*limit(0) 行为*。
- 如果你禁用此标志，你需要将 `findAll(0, $offset)` 更改为 `findAll(null, $offset)`。

$updateOnlyChanged

添加了一个属性 `$updateOnlyChanged`，用于决定是否仅更新*Entity* 的更改字段。如果你将此属性设置为 `false`，当你更新一个 Entity 时，即使 Entity 中的值没有变化，也不会抛出 `DataException` "There is no data to update"。

详见*Using CodeIgniter's Model*。

保存日期

现在你可以配置保存*Time* 实例时的日期/时间格式。详见*保存日期*。

库

- **CORS:** 添加了*跨域资源共享 (CORS)* 过滤器和类。
- **Validation:**
  - 新增规则 `field_exists`，用于检查字段是否存在于要验证的数据中。
  - `Validation::run()` 的 `$dbGroup` 参数现在不仅接受数据库组名，还接受数据库连接实例或数据库设置数组。
- **Session:**

  - `RedisHandler` 现在可以配置获取锁的时间间隔（`$lockRetryInterval`）和重试次数（`$lockMaxRetries`）。
  - 现在你可以在 `RedisHandler` 中使用 Redis ACL（用户名和密码）。详见*RedisHandler 驱动程序*。

- **Security:** `Config\Security::$redirect` 现在是特定于环境的。在生产环境中默认改为 `true`，但在其他环境中仍然是 `false`。

其他

- **Bootstrap:** 引入了 `CodeIgniter\Boot` 类，取代了 **system/bootstrap.php**。
- **Autoloader:**
  - 使用 Composer 时的自动加载性能有所提升。在 **composer.json** 中在 `autoload.psr4` 设置中添加 `App` 命名空间也可能会提升应用的性能。详见应用程序命名空间。
  - 实现了 FileLocator 缓存。详见*FileLocator 缓存*。
  - 添加了 `FileLocatorInterface`。
- **CodeIgniter:** 新增伪变量 `{memory_usage}`，在视图文件中显示内存使用情况，这是 CodeIgniter 3 支持的功能。
- **Events:** 为 Spark 命令添加了事件点 `pre_command` 和 `post_command`。详见*Event Points*。
- **HTTP:** 添加了 `Message::addHeader()` 方法来添加另一个具有相同名称的头。详见*`CodeIgniter\HTTP\Message::addHeader()`*。
- **Web 页面缓存:** `ResponseCache` 已改进，包含在缓存键中的请求 HTTP 方法。意味着如果 HTTP 方法不同，相同的 URI 将分别缓存。
- **CSP:** 添加了 `ContentSecurityPolicy::clearDirective()` 方法来清除现有的 CSP 指令。详见清除指令。

重大变更

行为更改

小写 HTTP 方法名

由于历史原因，框架使用小写的 HTTP 方法名，如 “get”、” post”。但方法令牌是区分大小写的，因为它可能用于具有区分大小写方法名的基于对象的系统。按照惯例，标准化方法用全大写字母 US-ASCII 字母定义。详见 https://www.rfc-editor.org/rfc/rfc9110#name-overview。

现在框架使用正确的 HTTP 方法名，如 “GET”、” POST”。

- `Request::getMethod()` 返回大写的 HTTP 方法。
- `CURLRequest::request()` 不会将接受的 HTTP 方法更改为大写。

详情见*小写 HTTP 方法名*。

过滤器执行顺序

控制器过滤器的执行顺序已更改。详见升级指南。

嵌套路由组和选项

由于错误修复，行为已更改，使得传递给外部 `group()` 的选项与内部 `group()` 的选项合并。详见升级指南。

API\ResponseTrait

现在当响应格式为 JSON 时，如果你传递字符串数据，框架将返回 JSON 响应。在以前的版本中，它返回 HTML 响应。详见升级指南。

Factories 类

工厂 已更改为最终类（final class）。它是一个静态类，即使它被扩展，也没有替换它的方式。

其他

- **AutoRouting Legacy:** 如果请求 URI 对应的控制器不存在，则改为抛出 `PageNotFoundException`。
- **Logger:** *`log_message()`* 函数和 `CodeIgniter\Log\Logger` 中的 logger 方法现在不再返回 `bool` 值。返回类型已固定为 `void`，以遵循 PSR-3 接口。
- **Autoloader:** 已删除 `FileLocator::findQualifiedNameFromPath()` 返回的完全限定类名中的前缀 `\`。
- **BaseModel:** `getIdValue()` 方法已更改为 `abstract`。
- **Routing:** *404 重写* 功能默认改变 Response 状态代码为 404。详见*升级指南*。
- **system/bootstrap.php:** 此文件不能再使用。代码已移动到新类 `CodeIgniter\Boot`。

接口更改

备注: 只要你没有扩展相关的 CodeIgniter 核心类或实现这些接口，所有这些更改都是向后兼容的，不需要任何干预。

- **ResponseInterface:** `ResponseInterface::setCookie()` 的第三个参数 `$expire` 的默认值已从 `''` 修正为 `0`。
- **Logger:** psr/log 包已升级到 v3.0.0。
- **Validation:** `ValidationInterface::run()` 的方法签名已更改。删除了 `$dbGroup` 参数上的 `?string` 类型提示。

方法签名更改

设置 Cookie

set_cookie() 和*CodeIgniter\HTTP\Response::setCookie()* 的第三个参数 $expire 已修正。

类型已从 string 更改为 int，默认值已从 '' 更改为 0。

FileLocatorInterface

- **Router:** RouteCollection 构造函数的第一个参数已从 FileLocator 更改为 FileLocatorInterface。
- **View:** View 构造函数的第三个参数已从 FileLocator 更改为 FileLocatorInterface。

返回类型更改

- **Model:** Model 和 BaseModel 类中 objectToRawArray() 方法的返回类型已从 ?array 更改为 array。

传统验证规则

为了在框架代码库中添加 declare(strict_types=1)，所有传统验证规则类 CodeIgniter\Validation\FormatRules 和 CodeIgniter\Validation\Rules 中用于验证值的方法参数类型 ?string 已移除。

例如，方法签名更改如下:

```
之前：public function integer(?string $str = null): bool
之后：public function integer($str = null): bool
```

其他

- **Logger:** 实现 PSR-3 接口的 `CodeIgniter\Log\Logger` 中方法的签名已修正。`bool` 返回类型已更改为 `void`。`$message` 参数现在具有 `string|Stringable` 类型。
- **Validation:** `Validation::run()` 的方法签名已更改。去掉了 `?string` 类型提示。

移除的弃用项

Request

- `RequestInterface` 和 `Request` 中 `getMethod()` 的 `$upper` 参数已移除。详见*小写 HTTP 方法名*。
- `RequestInterface` 和 `Request` 中弃用的 `isValidIP()` 方法已移除。
- `IncomingRequest` 中弃用的 `$uri` 和 `$config` 属性的可见性已更改为 protected。
- `IncomingRequest` 中的 `$enableCSRF` 属性已移除。
- `IncomingRequest` 中的 `removeRelativeDirectory()` 方法已移除。
- `Request` 中的 `$proxyIPs` 属性已移除。

Filters

- 已移除以下弃用项，因为现在始终启用多重过滤器。
  - `Filters::enableFilter()`
  - `RouteCollection::getFilterForRoute()`
  - `Router::$filterInfo`
  - `Router::getFilter()`

数据库

- `ModelFactory`

模型

- `BaseModel::idValue()`
- `BaseModel::fillPlaceholders()`
- `Model::idValue()`
- `Model::classToArray()`

Response

- `ResponseTrait::$CSP` 属性的可见性已更改为 protected。
- 以下弃用的属性已移除：
  - `ResponseTrait::$CSPEnabled`
  - `ResponseTrait::$cookiePrefix`
  - `ResponseTrait::$cookieDomain`
  - `ResponseTrait::$cookiePath`
  - `ResponseTrait::$cookieSecure`
  - `ResponseTrait::$cookieHTTPOnly`
  - `ResponseTrait::$cookieSameSite`
  - `ResponseTrait::$cookies`

Security

- `SecurityInterface::isExpired()`
- `Security::isExpired()`
- `Security::CSRFVerify()`
- `Security::getCSRFHash()`

- `Security::getCSRFTokenName()`
- `Security::sendCookie()`
- `Security::doSendCookie()`

CodeIgniter

- `$path`
- `$useSafeOutput`
- `useSafeOutput()`
- `setPath()`

测试

- `CIDatabaseTestCase`
- `ControllerResponse`
- `ControllerTester`
- `FeatureResponse`
- `FeatureTestCase`
- `Mock\MockSecurityConfig`

Spark 命令

- `migrate:create`
- `session:migration`

其他

- **Cache:** 已移除 `CodeIgniter\Cache\Exceptions\ExceptionInterface`。
- **Config:**
  - 已移除 `CodeIgniter\Config\Config` 类。

  - 已移除 `CodeIgniter\Config\BaseService::discoverServices()` 方法。

- **Controller:** 已移除 `Controller::loadHelpers()` 方法。
- **Exceptions:** 已移除 `CodeIgniter\Exceptions\CastException` 类。
- **Entity:** 已移除 `CodeIgniter\Entity` 类。请使用 `CodeIgniter\Entity\Entity`。
- **spark:** 已移除 `SPARKED` 常量。

消息更改

- 添加了 `CLI.generator.className.test` 消息。
- 添加了 `Validation.field_exists` 错误消息。

更改

- **Bootstrap: .env** 的加载和 `ENVIRONMENT` 的定义已移至 **bootstrap.php** 之前加载。
- **Config:**
  - `Config\Feature::$multipleFilters` 已移除，因为现在始终启用多重过滤器。
  - 生产环境中的默认错误级别（**app/Config/Boot/production.php**）已更改为 `E_ALL & ~E_DEPRECATED`，以匹配生产环境的默认 **php.ini**。
- **RouteCollection:** 受保护属性 `$routes` 中的 HTTP 方法键已从小写修正为大写。
- **Exceptions:** 未使用的 `CodeIgniter\Exceptions\AlertError` 和 `CodeIgniter\Exceptions\EmergencyError` 已移除。
- **Forge:** `SQLSRV` Forge 现在在添加表列时将 `ENUM` 数据类型转换为 `VARCHAR(n)`。在以前的版本中，它被转换为 SQL Server 中弃用的 `TEXT`。
- `declare(strict_types=1)` 已添加到大多数框架代码库。

弃用项

- **Services:** `BaseService::$services` 属性已弃用，不再使用。
- **CodeIgniter:**
  - `determinePath()` 方法已弃用，不再使用。
  - `resolvePlatformExtensions()` 方法已弃用，不再使用。它已被移到 `CodeIgniter\Boot::checkMissingExtensions()` 方法。
  - `bootstrapEnvironment()` 方法已弃用，不再使用。它已被移到 `CodeIgniter\Boot::loadEnvironmentBootstrap()` 方法。
  - `initializeKint()` 方法已弃用，不再使用。它已移到 `Autoloader`。
  - `autoloadKint()` 方法已弃用，不再使用。它已移到 `Autoloader`。
  - `configureKint()` 方法已弃用，不再使用。它已移到 `Autoloader`。
- **Response:** 构造函数参数 `$config` 已弃用，不再使用。
- **Filters:**
  - `Filters` 接受 `Config\Filters::$methods` 的小写 HTTP 方法键的功能已弃用。请改用正确的大写 HTTP 方法键。
  - `spark filter:check` 命令接受小写 HTTP 方法的功能已弃用。请改用正确的大写 HTTP 方法。
- **RouteCollection:** `match()` 和 `setHTTPVerb()` 方法接受小写 HTTP 方法的功能已弃用。请改用正确的大写 HTTP 方法。
- **FeatureTestTrait:** `call()` 和 `withRoutes()` 方法接受小写 HTTP 方法的功能已弃用。请改用正确的大写 HTTP 方法。
- **Database:** `BaseConnection::$strictOn` 已弃用，未来将迁移到 `MySQLi\Connection`。

修复的错误

请参阅仓库中的 CHANGELOG.md 获取完整的错误修复列表。

版本 4.4.8

发布日期：2024 年 4 月 7 日

CodeIgniter4 的 4.4.8 版本发布

- 重大变更
- 修复的错误

重大变更

- 修复了一个导致*Exception handler* 显示与异常代码不对应的错误视图文件的错误。为此，`CodeIgniter\Debug\ExceptionHandler::determineView()` 添加了第三个参数 `int $statusCode = 500`。

修复的错误

请参阅仓库中的 CHANGELOG.md 获取完整的错误修复列表。

版本 4.4.7

发布日期：2024 年 3 月 29 日

CodeIgniter4 的 4.4.7 版本发布

- 安全更新
- 重大变更
- 修复的错误

安全更新

- **Language:** 修复了 *Language* 类 *DoS* 漏洞。详见 Security advisory GHSA-39fp-mqmm-gxj6。
- **URI Security:** 添加了检查 URI 中是否包含不允许字符串的功能。此检查相当于 CodeIgniter 3 中的 URI 安全性。此功能默认为启用，但升级用户需要添加设置以启用它。详情见*URI 安全性*。
- **Filters:** 修复了 Filters 处理的 URI 路径未进行 URL 解码的错误。详情见控制器过滤器中的路径。

重大变更

- 在以前的版本中，当使用 `Time::difference()` 比较日期时，如果日期包含不同于 24 小时的天数（由于夏令时导致），会返回意外结果。该错误已被修复。详情见*Times and Dates 中的说明*。

修复的错误

请参阅仓库中的 CHANGELOG.md 获取完整的错误修复列表。

版本 4.4.6

发布日期：2024 年 2 月 24 日

CodeIgniter4 的 4.4.6 版本发布

- 重大变更
  - *Time::createFromTimestamp()*
- 修复的错误

重大变更

Time::createFromTimestamp()

修复了一个导致*Time::createFromTimestamp()* 返回 UTC 时区的 Time 实例的错误。

从这个版本开始，当你不指定时区时，默认返回应用程序时区的 Time 实例。

修复的错误

- **Session:** 修复了 Redis session 处理程序中的一个错误，该错误导致锁定失败并清除了 session 数据。
- **DB Forge:** 修复了 SQLite3 Forge 中的一个错误，该错误导致 `Forge::modifyColumn()` 错误地修改了表定义。
- **CSP:** 修复了一个导致 CSP 阻止调试工具栏中某些元素的错误。

请参阅仓库中的 CHANGELOG.md 获取完整的错误修复列表。

版本 4.4.5

发布日期：2024 年 1 月 27 日

CodeIgniter4 的 4.4.5 版本发布

- 已修复的错误

已修复的错误

- **QueryBuilder:** 修复了因为 PostgreSQL 上的类型错误，导致 `updateBatch()` 方法无法工作的问题。

要查看完整的错误修复列表，可以查看仓库的 CHANGELOG.md。

版本 4.4.4

发布日期：2023 年 12 月 28 日

CodeIgniter4 的 4.4.4 版本发布

- 重大变化
  - 采用 *Dot* 数组语法进行验证
  - 验证规则匹配和差异
  - 在 *CURLRequest* 中移除了 ssl_key 选项的使用
  - 文件系统助手
- 改进
- 消息变更
- 弃用
- 已修复的错误

重大变化

采用 Dot 数组语法进行验证

使用通配符 * 的验证规则现在只验证符合”Dot 数组语法”的正确维度数据。详见升级获取详情。

验证规则匹配和差异

在严格和传统规则中，`matches` 和 `differs` 验证非字符串类型数据的情况已修复。

在 CURLRequest 中移除了 *ssl_key* 选项的使用

由于一个错误，我们在 CURLRequest 中使用了未记录的 *ssl_key* 配置选项来定义 CA bundle。现在已经修复，并且按照文档要求工作。你可以通过 *verify* 选项来定义你的 CA bundle。

文件系统助手

`get_filenames()` 现在会跟踪符号连接文件夹，而之前只是返回而不跟踪。

改进

- 完全支持 PHP 8.3。

消息变更

- 添加 `HTTP.invalidJSON` 错误消息。
- 添加 `HTTP.unsupportedJSONFormat` 错误消息。

弃用

- **请求：** `CodeIgniter\HTTP\Request::getEnv()` 方法已被弃用。此方法自从一开始就没有工作，请使用`env()`。

已修复的错误

- **CURLRequest：**修复了即使配置项‘verify’设置为 *false* 时，也会检查主机名的错误。

要查看完整的错误修复列表，请去看仓库的 CHANGELOG.md。

版本 4.4.3

发布日期：2023 年 10 月 26 日

CodeIgniter4 4.4.3 版本发布

- 安全性
- 已修复的错误

安全性

- 修复了 在生产环境中显示详细错误报告的问题。有关更多信息，请参阅 安全公告 GHSA-hwxf-qxj7-7rfj。

已修复的错误

- **UserGuide:** 修复了事件点 中 `pre_system` 和 `post_system` 的描述。

请参阅仓库的 CHANGELOG.md 获取完整的修复 bug 列表。

版本 4.4.2

发布日期：2023 年 10 月 19 日

CodeIgniter4 4.4.2 版本发布

- 消息变更
- 变更
- 弃用
- 修复的问题

消息变更

- 添加了 `Language.invalidMessageFormat` 错误消息。

变更

- **数据库迁移：** 移除了 `spark migrate:rollback` 命令的 `-g` 选项。该选项从一开始就无效。此外，回滚命令将数据库状态恢复到指定的批次号，并且无法仅指定特定的数据库组。
- **安全性：** 现在还会检查原始请求体（非 JSON 格式）中的 CSRF 令牌，用于 PUT、PATCH 和 DELETE 类型的请求。

弃用

- **过滤器：** 过滤器的自动发现和 `Filters::discoverFilters()` 方法已弃用。请改用注册器。有关详细信息，请参阅过滤器。
- **CLI：** 弃用了公共属性 `CLI::$readline_support` 和 `CLI::$wait_msg`。这些方法将被保护。
- **CodeIgniter：** `displayCache()` 方法的 `$config` 参数已弃用。未使用该参数。

修复的问题

- **CodeIgniter：** 修复了在页面未找到时返回“200 OK”响应状态码的 bug。
- **Spark：** 修复了在生产模式下 spark 无法显示异常或在发生异常时以 JSON 格式显示回溯的 bug。
- **Forge：** 修复了在给现有表添加主键时，如果没有添加其他键，则会被忽略的 bug。
- **路由：** 修复了 `spark routes` 可能显示不正确的路由名称的 bug。

请参阅仓库的 CHANGELOG.md 获取完整的修复 bug 列表。

版本 4.4.1

发布日期：2023 年 9 月 5 日

CodeIgniter4 4.4.1 版本发布

- 修复的问题

修复的问题

- **AutoRouting Legacy:** 修复了 Legacy 自动路由无法正常工作的 bug。
- **FeatureTest：**
  - 修复了 FeatureTest 可能导致风险测试的 bug。
  - 修复了 FeatureTest 在 forceGlobalSecureRequests 为 true 时失败的 bug。

请参阅仓库的 CHANGELOG.md 获取完整的修复 bug 列表。

版本 4.4.0

发布日期：2023 年 8 月 25 日

CodeIgniter4 4.4.0 版本发布

- 亮点
- 重大变更
  - 行为变更
    - *URI::setSegment()* 和不存在的段
    - 工厂
    - 自动加载器
    - *CodeIgniter* 和 *exit()*
    - 站点 *URI* 更改
  - 接口更改

  - 方法签名更改
    * 参数类型更改
    * 添加的参数
    * 删除的参数
    * 返回类型更改
- 增强功能
  - 命令
  - 测试
  - 数据库
  - 模型
  - 库
  - 辅助函数和方法
  - 其他
- 消息更改
- 变更
- 弃用
- 已修复的错误

亮点

- 调试工具栏现在具有新的“热重载”功能 (由 lonnieezell 贡献)。请参阅测试。

重大变更

行为变更

URI::setSegment() 和不存在的段

当你设置最后一个 +2 段时，现在会抛出异常。在之前的版本中，只有当指定了最后一个段的 +3 或更多时才会抛出异常。请参阅*URI::setSegment() 更改*。

当前最后一个段的下一个段（+1）可以像以前一样设置。

工厂

使用命名空间传递类名

现在，只有在请求不带命名空间的类名时，preferApp 才起作用。

例如，当你调用 model(\Myth\Auth\Models\UserModel::class) 或 model('Myth\Auth\Models\UserModel') 时：

- 之前：
  - 如果存在 App\Models\UserModel 并且 preferApp 为 true（默认值），则返回该类
  - 如果存在 Myth\Auth\Models\UserModel 并且 preferApp 为 false，则返回该类
- 现在：
  - 无论 preferApp 是否为 true（默认值），都返回 Myth\Auth\Models\UserModel
  - 如果在调用 model() 之前定义了 Factories::define('models', 'Myth\Auth\Models\UserModel', 'App\Models\UserModel'），则返回 App\Models\UserModel

如果你错误地传递了一个不存在的类名，之前的版本会返回 App 或 Config 命名空间中的类实例，因为存在 preferApp 功能。

例如，在控制器（namespace App\Controllers）中，如果你错误地调用了 config(Config\App::class)（注意类名缺少前导的 \），实际上传递的是 App\Controllers\Config\App。但是该类不存在，因此 Factories 现在将返回 null。

属性名称

属性 Factories::$basenames 已更名为 $aliases。

自动加载器

以前，CodeIgniter 的自动加载器允许加载以 *.php* 扩展名结尾的类名。这意味着可以实例化类似 *new Foo.php()* 的对象，并将其实例化为 *new Foo()*。由于 *Foo.php* 是无效的类名，自动加载器的行为已更改。现在，实例化这样的类将失败。

CodeIgniter 和 exit()

`CodeIgniter::run()` 方法不再调用 `exit(EXIT_SUCCESS)`。退出调用已移至 **public/index.php**。

站点 URI 更改

添加了一个扩展了 `URI` 类并表示站点 URI 的新 `SiteURI` 类，并且现在在许多需要当前 URI 的地方使用它。

控制器中的 `$this->request->getUri()` 返回 `SiteURI` 实例。此外，*site_url()*、*base_url()* 和 *current_url()* 在内部使用 SiteURI。

getPath()

`getPath()` 方法现在始终返回带有前导 `/` 的完整 URI 路径。因此，当你的 baseURL 具有子目录并且你想获取相对于 baseURL 的路径时，必须使用新的 `getRoutePath()` 方法。

例如:

```
       baseURL: http://localhost:8888/CodeIgniter4/
      当前 URI: http://localhost:8888/CodeIgniter4/foo/bar
     getPath(): /CodeIgniter4/foo/bar
getRoutePath(): foo/bar
```

站点 URI 值

SiteURI 类现在比以前更严格地规范化站点 URI，并修复了一些错误。

因此，与之前的版本相比，框架可能会以稍微不同的方式返回站点 URI 或 URI 路径。例如，在 `index.php` 之后会添加 `/`:

```
http://example.com/test/index.php?page=1
↓
http://example.com/test/index.php/?page=1
```

接口更改

备注: 只要你没有扩展相关的 CodeIgniter 核心类或实现这些接口，所有这些更改都是向后兼容的，无需干预。

- **Validation:** 在 `ValidationInterface` 中添加了 `getValidated()` 方法。

方法签名更改

参数类型更改

- **Services:**
  - `Services::security()` 的第一个参数已从 `Config\App` 更改为 `Config\Security`。
  - `Services::session()` 的第一个参数已从 `Config\App` 更改为 `Config\Session`。
- **Session:**
  - `Session::__construct()` 的第二个参数已从 `Config\App` 更改为 `Config\Session`。
  - 数据库的 `BaseHandler`、`DatabaseHandler`、`FileHandler`、`MemcachedHandler` 和 `RedisHandler` 中的 `__construct()` 的第一个参数已从 `Config\App` 更改为 `Config\Session`。

- **Security:** `Security::__construct()` 的第一个参数已从 `Config\App` 更改为 `Config\Security`。
- **Validation:** `Validation::check()` 的方法签名已更改。`$rule` 参数上的 `string` 类型提示已被删除。
- **CodeIgniter:** `CodeIgniter::setRequest()` 的方法签名已更改。`$request` 参数上的 `Request` 类型提示已被删除。
- **FeatureTestCase:**
  - `FeatureTestCase::populateGlobals()` 的方法签名已更改。`$request` 参数上的 `Request` 类型提示已被删除。
  - `FeatureTestCase::setRequestBody()` 的方法签名已更改。`$request` 参数上的 `Request` 类型提示和返回类型 `Request` 已被删除。

添加的参数

- **Routing:** 在 `RouteCollection::__construct()` 中添加了第三个参数 `Routing $routing`。

删除的参数

- **Services:** 在 `Services::exceptions()` 中删除了第二个参数 `$request` 和第三个参数 `$response`。
- **错误处理:** 在 `CodeIgniter\Debug\Exceptions::__construct()` 中删除了第二个参数 `$request` 和第三个参数 `$response`。

返回类型更改

- **自动加载器:** `loadClass` 和 `loadClassmap` 方法的返回签名都改为 `void`，以便与 `spl_autoload_register` 和 `spl_autoload_unregister` 函数中的回调兼容。

增强功能

命令

- **spark routes:**
  - 现在你可以在请求 URL 中指定主机。请参阅指定主机。
  - 它在 *Handler* 中显示视图路由 的视图文件，如下所示：

Method	Route	Name	Handler	Before Filters	After Filters
GET	about	»	(View) pages/about		toolbar

测试

- **调试工具栏:**
  - 调试工具栏现在具有新的“热重载”功能，可以在文件更改时自动重新加载页面。请参阅热重载。
  - 现在，在 *Routes* 选项卡的 *DEFINED ROUTES* 中显示视图路由。

数据库

- **MySQLi:** 在数据库配置中添加了 `numberNative` 属性，以保持 SQL 查询后获取的变量类型与数据库中设置的类型一致。请参阅*Database Configuration*。
- **SQLSRV:** 字段元数据现在包括 `nullable`。请参阅*$db->getFieldData()*。

模型

- 为实体添加了特殊的 getter/setter，以避免方法名称冲突。请参阅特殊的 *Getter/Setter*。

库

- **Validation:** 添加了 `Validation::getValidated()` 方法，用于获取实际验证的数据。请参阅*获取已验证数据* 了解详细信息。
- **Images:** 现在可以使用选项 `$quality` 压缩 WebP 图像。
- **Uploaded Files:** 添加了 `UploadedFiles::getClientPath()` 方法，如果通过目录上传方式上传文件，则返回文件的 *full_path* 索引的值。
- **CURLRequest:** 添加了请求选项 `proxy`。请参阅*CURLRequest Class*。
- **URI:** 添加了一个扩展了 `URI` 并表示站点 URI 的新 `SiteURI` 类。

辅助函数和方法

- **Array:** 添加了*`array_group_by()`* 辅助函数，用于将数据值分组在一起。支持点符号语法。
- **Common:** *`force_https()`* 不再终止应用程序，而是抛出 `RedirectException`。

其他

- **DownloadResponse:** 添加了 `DownloadResponse::inline()` 方法，将 `Content-Disposition: inline` 标头设置为在浏览器中显示文件。请参阅*在浏览器中打开文件* 了解详细信息。
- **View:** 在 `renderSection()` 上添加了可选的第二个参数 `$saveData`，以防止在显示后自动清除数据。请参阅*View Layouts* 了解详细信息。
- **自动路由（改进）:**
  - 现在你可以路由到模块。请参阅*模块路由* 了解详细信息。
  - 如果找到与 URI 段对应的控制器，并且该控制器没有为该 URI 段定义的方法，则将执行默认方法。这样可以更灵活地处理自动路由中的 URI。请参阅*默认方法回退* 了解详细信息。
- **过滤器：** 现在可以在*$filters 属性* 中使用过滤器参数。
- **请求：** 添加了 `IncomingRequest::setValidLocales()` 方法，用于设置有效的区域设置。

- **Table:** 添加了 `Table::setSyncRowsWithHeading()` 方法，用于将行列与标题同步。请参阅同步行与标题 了解详细信息。
- **错误处理：** 现在可以使用自定义异常处理器。
- **RedirectException:**
  - 它还可以接受实现 `ResponseInterface` 的对象作为第一个参数。
  - 它实现了 `ResponsableInterface`。
- **Factories:**
  - 现在可以定义实际加载的类名。请参阅定义要加载的类名。
  - 实现了配置缓存。请参阅配置缓存 了解详细信息。

消息更改

- 添加了 `Core.invalidDirectory` 错误消息。
- 改进了 `HTTP.invalidHTTPProtocol` 错误消息。

变更

- **Images:** 在 `GDHandler` 中，WebP 的默认质量从 80 改为 90。
- **Config:**
  - 删除了 **app/Config/App.php** 中已弃用的 Cookie 项。
  - 删除了 **app/Config/App.php** 中已弃用的 Session 项。
  - 删除了 **app/Config/App.php** 中已弃用的 CSRF 项。
  - 将路由设置移至 **app/Config/Routing.php** 配置文件。请参阅升级指南。
- **DownloadResponse:** 在生成响应标头时，如果之前已指定了 `Content-Disposition` 标头，则不替换它。
- **自动加载器：**
  - 在 v4.4.0 之前，CodeIgniter 的自动加载器不允许在某些操作系统上的文件名中使用特殊字符。可以使用的符号是 /、_、.、:、\ 和空格。因此，如果你将 CodeIgniter 安装在包含特殊字符（如（、）等）的文件夹中，CodeIgniter 将无法工作。从 v4.4.0 开始，此限制已被移除。

  - `Autoloader::loadClass()` 和 `Autoloader::loadClassmap()` 方法现在都标记为 `@internal`。

- **RouteCollection:** 受保护属性 `$routes` 的数组结构已进行了修改以提高性能。
- **HSTS:** 现在，无论是通过*force_https()* 还是 `Config\App::$forceGlobalSecureRequests = true`，都会设置 HTTP 状态码 307，允许在重定向后保留 HTTP 请求方法。在之前的版本中，它是 302。

弃用

- **Entity:** 弃用了 `Entity::setAttributes()` 方法。请改用 `Entity::injectRawData()`。
- **错误处理：**弃用了 `CodeIgniter\Debug\Exceptions` 中的许多方法和属性。因为这些方法已移至 `BaseExceptionHandler` 或 `ExceptionHandler`。
- **自动加载器：**弃用了 `Autoloader::sanitizeFilename()`。
- **CodeIgniter:**
  - 弃用了 `CodeIgniter::$returnResponse` 属性。不再使用。
  - 弃用了 `CodeIgniter::$cacheTTL` 属性。不再使用。请改用 `ResponseCache`。
  - 弃用了 `CodeIgniter::cache()` 方法。不再使用。请改用 `ResponseCache`。
  - 弃用了 `CodeIgniter::cachePage()` 方法。不再使用。请改用 `ResponseCache`。
  - 弃用了 `CodeIgniter::generateCacheName()` 方法。不再使用。请改用 `ResponseCache`。
  - 弃用了 `CodeIgniter::callExit()` 方法。不再使用。
- **RedirectException:** 弃用了 `\CodeIgniter\Router\Exceptions\RedirectException`。请改用 `\CodeIgniter\HTTP\Exceptions\RedirectException`。
- **Session:** 弃用了 `Session` 中的属性 `$sessionDriverName`、`$sessionCookieName`、`$sessionExpiration`、`$sessionSavePath`、`$sessionMatchIP`、`$sessionTimeToUpdate` 和 `$sessionRegenerateDestroy`，不再使用。请改用 `$config`。

- **Security:** 弃用了 Security 中的属性 $csrfProtection、$tokenRandomize、$tokenName、$headerName、$expires、$regenerate 和 $redirect，不再使用。请改用 $config。
- **URI:**
  - 弃用了 URI::$uriString。
  - 弃用了 URI::$baseURL。请改用 SiteURI。
  - 弃用了 URI::setSilent()。
  - 弃用了 URI::setScheme()。请改用 withScheme()。
  - 弃用了 URI::setURI()。
- **IncomingRequest:**
  - 弃用了 IncomingRequest::detectURI()，不再使用。
  - 弃用了 IncomingRequest::detectPath()，不再使用。已移至 SiteURIFactory。
  - 弃用了 IncomingRequest::parseRequestURI()，不再使用。已移至 SiteURIFactory。
  - 弃用了 IncomingRequest::parseQueryString()，不再使用。已移至 SiteURIFactory。
  - 弃用了 IncomingRequest::setPath()。

已修复的错误

- **自动路由（改进）：**在之前的版本中，当 $translateURIDashes 为 true 时，两个 URI 对应于单个控制器方法，一个 URI 用于破折号（例如 **foo-bar**），另一个 URI 用于下划线（例如 **foo_bar**）。修复了此错误。现在，下划线的 URI（**foo_bar**）无法访问。
- **输出缓冲：**修复了输出缓冲的错误。
- **ControllerTestTrait：** ControllerTestTrait::withUri() 使用 URI 创建一个新的 Request 实例。因为 Request 实例应该具有 URI 实例。此外，如果 URI 字符串中的主机名与 Config\App 中的有效主机名不匹配，则将设置有效的主机名。

有关修复的所有错误的完整列表，请参阅存储库的 CHANGELOG.md。

版本 4.3.8

发布日期：2023 年 8 月 25 日

CodeIgniter4 4.3.8 版本发布

- 已修复的错误

已修复的错误

- **控制器过滤器 (Controller Filters):** 在以前的版本中，`['except' => []]` 或 `['except' => '']` 意味着 "排除所有"。已修复此错误，现在
  - `['except' => []]` 意味着不排除任何内容。
  - `['except' => '']` 意味着只排除基础 URL。

请查看仓库的 CHANGELOG.md 以获取已修复错误的完整列表。

版本 4.3.7

发布日期：2023 年 7 月 30 日

CodeIgniter4 4.3.7 版本发布

- 重大变更
- 变更
- 已修复的错误

重大变更

- **路由集合 (RouteCollection):** 在 `RouteCollection::getRoutes()` 方法中添加了第二个参数 `bool $includeWildcard = true`。
- **AutoRouting Legacy:** `AutoRouter::__construct()` 的第一个参数从 `$protectedControllers` 更改为 `$cliRoutes`。

- **FeatureTestTrait:** 当使用*withBodyFormat()* 时，请求正文的优先级已更改。详情请参考升级指南。
- **验证 (Validation):** `Validation::loadRuleGroup()` 的返回值从 "**规则数组**" 更改为 "**规则数组**和 **自定义错误数组**的 **数组**" (`[rules, customErrors]`)。

变更

- 数字辅助函数*`number_to_amount()`*，以前返回 "1000"，现在在数字恰好为 1000 时已更正为返回 "1 thousand"。

已修复的错误

- **AutoRouting Legacy:** 修复了一个问题，当你使用 `$routes->add()` 添加路由时，控制器的其他方法在 Web 浏览器中无法访问。

请查看仓库的 CHANGELOG.md 以获取已修复错误的完整列表。

版本 4.3.6

发布日期:2023 年 6 月 18 日

CodeIgniter 4.3.6 版发布

- 不兼容变更
  - 接口变更
    - *AutoRouterInterface*
    - *ValidationInterface::check()*
  - 方法签名变更
    - *Validation::check()*
- 弃用功能
- 错误修复

不兼容变更

接口变更

备注: 只要你没有扩展相关的 CodeIgniter 核心类或实现这些接口, 所有这些变更都是向后兼容的, 不需要任何干预。

AutoRouterInterface

现在 `AutoRouterInterface::getRoute()` 有了新的第二个参数 `string $httpVerb`。

ValidationInterface::check()

- 第二个参数已从 `string $rule` 更改为 `$rules`。
- 添加了可选的第四个参数 `$dbGroup = null`。

方法签名变更

Validation::check()

- 第二个参数已从 `string $rule` 更改为 `$rules`。
- 添加了可选的第四个参数 `$dbGroup = null`。

弃用功能

- **AutoRouterImproved:** 构造函数参数 `$httpVerb` 已弃用。不再使用。

错误修复

- **验证:** 修复了检查占位符值时会忽略 `$DBGroup` 的错误。
- **验证:** 修复了 `check()` 无法指定非默认数据库组的错误。
- **数据库:** 修复了 Postgre 连接参数中的分号字符 (`;`) 会中断 DSN 字符串的错误。
- **AutoRouting Improved:** 修复了功能测试可能找不到控制器/方法的错误。

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG.md。

版本 4.3.5

发布日期:2023 年 5 月 21 日

CodeIgniter 4.3.5 版发布

- 安全性
- 变更
- 弃用功能
- 错误修复

安全性

- 修复了 验证占位符中的远程代码执行漏洞。更多信息请参阅 安全公告 GHSA-m6m8-6gq8-c9fj。
- 修复了 `Session::stop()` 未能销毁会话的问题。详情请参阅*Session Library*。

变更

- **make:cell 命令:** 创建新 cell 时, 控制器的类名总是 SUFFIX 为 `Cell`。对于视图文件, 最终的 `_cell` 总是被删除。
- **视图 Cell:** 为了与以前的版本兼容, 只要启用了自动检测视图文件 (通过将 `$view` 属性设置为空字符串), 以 `_cell` 结尾的视图文件名仍然可以被 `Cell` 定位。

弃用功能

- **Session:** *Session::stop()* 方法已弃用。请使用*Session::destroy()*。

错误修复

- **验证:** 修复了一个错误, 其中与 `permit_empty` 或 `if_exist` 规则组合使用的闭包会导致错误。
- **make:cell 命令:** 修复了生成类视图文件的问题。
- **make:cell 命令:** 修复了对大小写不敏感操作系统的单字类输入的处理。

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG.md。

版本 4.3.4

发布日期:2023 年 4 月 27 日

CodeIgniter 4.3.4 版发布

- 不兼容变更
  - 行为变化
    - 重定向状态码
    - *Forge::modifyColumn()*
- 弃用功能
- 错误修复

不兼容变更

行为变化

重定向状态码

- 由于一个错误, 在以前的版本中, 当使用 HTTP/1.1 或更高版本时, 即使指定了状态码, 实际重定向响应的状态码也可能会改变。例如, 对于 GET 请求,302 会更改为

307; 对于 POST 请求,307 和 302 会更改为 303。

- 从这个版本开始, 如果在*redirect* 中指定了状态码, 该代码将始终在响应中使用。
- 当使用 HTTP/1.1 或更高版本时,GET 请求的默认代码已更正为 302。
- 当使用 HTTP/1.1 或更高版本时,HEAD 和 OPTIONS 请求的默认代码已更正为 307。
- 在 `$routes->addRedirect()` 中, 默认指定 302。因此, 当你不指定状态码时, 总是会使用 302。在以前的版本中,302 可能会更改。

Forge::modifyColumn()

- *$forge->modifyColumn()* 已修复。由于一个错误, 在以前的版本中,SQLite3/Postgres/SQLSRV 可能会不可预测地更改 `NULL/NOT NULL`。
- 在以前的版本中, 当你不指定 `null` 键时,OCI8 驱动程序不会更改 `NULL/NOT NULL`。
- 现在在所有数据库驱动程序中, 如果你不指定 `null` 键, `$forge->modifyColumn()` 始终设置为 `NULL`。
- `NULL/NOT NULL` 的更改可能仍然出人意料, 建议总是指定 `null` 键。

弃用功能

- **文本辅助函数:** *`random_string()`* 的类型 `basic`、`md5` 和 `sha1` 已弃用。它们不是加密安全的。

错误修复

- **CURLRequest:** 修复了一个错误, 其中响应类在请求之间共享。

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG.md。

版本 4.3.3

发布日期:2023 年 3 月 26 日

CodeIgniter 4.3.3 版发布

- 安全性
- 错误修复

安全性

- **电子邮件:** 添加了缺失的 TLS 1.3 支持。
- **文本辅助函数:** `random_string()` 类型 **alpha**、**alnum**、**numeric** 和 **nozero** 现在是加密安全的。

错误修复

- **配置:** 添加了缺失的 `Config\Encryption::$cipher`。
- **用户指南:** 修复了用于与 *CI3* 保持兼容性的配置 的示例代码。
- **用户指南:** 在*ChangeLog* 和*Upgrading Guide* v4.3.2 中添加了 `uri_string()` 中缺失的不兼容变更。

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG.md。

版本 4.3.2

发布日期:2023 年 2 月 18 日

CodeIgniter 4.3.2 版发布

- 不兼容变更
  - 行为变化
    - *base_url()*
    - *uri_string()*

- 错误修复

不兼容变更

行为变化

base_url()

- 由于一个错误, 在以前的版本中, 不带参数的*base_url()* 返回没有尾随斜杠 (/) 的 baseURL, 如 `http://localhost:8080`。
- 现在它返回带有尾随斜杠的 baseURL。这与 CodeIgniter 3 中的 `base_url()` 的行为相同。

uri_string()

- 从*uri_string()* 中删除了参数 `$relative`。由于一个错误, 此函数总是返回相对于 baseURL 的路径。
- 当访问 baseURL 时, 它现在将返回一个空字符串 (`''`)。这与 CodeIgniter 3 中的 `uri_string()` 的行为相同。在以前的版本中它返回 `/`。

错误修复

- **查询构建器:** 使用 `RawSql` 时 `where()` 生成错误的 SQL
- **查询构建器:** 传递给 `set()` 的 `RawSql` 会无错误地消失
- **Session:** 无法通过 `RedisHandler` 使用 TLS 连接 Redis
- **自动加载:** 可能不会添加 Composer 包的命名空间
- **解析器:** 如果分隔符更改, ! 不起作用
- **用户指南:** 在 ChangeLog 和 Upgrading Guide v4.3.0 中添加了缺失的项目

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG.md。

版本 4.3.1

发布日期:2023 年 1 月 14 日

CodeIgniter 4.3.1 版发布

- 错误修复

错误修复

- 修复: 某些环境变量 (**.env**) 的值没有反映在电子邮件配置中
- 修复: 验证 `is_unique` 和 `is_not_unique` 中的 TypeError
- 修复: 意外更改了方法名 `BaseBuilder::resetQuery()`
- 修复: 验证严格规则中处理浮点数 (`greater_than`、`greater_than_equal_to`、`less_than`、`less_than_equal_to`)
- 修复: 用户指南中缺少 PHP 8.2 中 `Config\Exceptions` 的说明

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG.md。

版本 4.3.0

发布日期:2023 年 1 月 10 日

CodeIgniter 4.3.0 版发布

- 亮点
- 不兼容变更
  - 行为变化
    - 数据库错误时抛出的异常
    - 异常发生时的 *HTTP* 状态码和退出代码
    - 时间
    - 其他
  - 接口变更

        * *OutgoingRequestInterface*
        * 其他
    - 方法签名变化
        * 验证变更
        * 数据库
        * 数据库 *Forge*
        * 其他
- 增强功能
    - 命令
    - 测试
    - 数据库
        * 查询构建器
        * *Forge*
        * 其他
    - 模型
    - 库
    - 辅助函数和函数
    - *HTML5* 兼容性
    - 错误处理
    - 多域名支持
    - 其他
- 消息变更
- 变更
- 弃用功能
- 错误修复

亮点

- 查询构建器支持 **upsert()**、**upsertBatch()**、**deleteBatch()**, 现在 ***batch()** 方法可以从查询中设置数据 (由 sclubricants 贡献)。详情请参阅查询构建器。
- 数据库 Forge 支持在 **现有表中添加索引**和 **命名索引** (由 sclubricants 贡献)。详情请参阅*Forge*。
- 为了使默认配置更安全, 默认验证规则已更改为 **严格规则**。
- 当数据库错误发生时, 会抛出异常的条件和可以抛出的异常类已发生变化。请参阅数据库错误时抛出的异常。

不兼容变更

行为变化

数据库错误时抛出的异常

- 数据库连接类抛出的异常已更改为 `CodeIgniter\Database\Exceptions\DatabaseException`。以前, 不同的数据库驱动程序会抛出不同的异常类, 但这些已经统一为 `DatabaseException`。
- 准备好的查询的 `execute()` 方法抛出的异常已更改为 `DatabaseException`。以前, 不同的数据库驱动程序可能会抛出不同的异常类, 或者不抛出异常, 但这些已统一为 `DatabaseException`。
- 在事务期间, 即使 `DBDebug` 为 true, 默认情况下也不会抛出异常。
- `DBDebug` 和 `CI_DEBUG` 变化
  - 为了在不同环境下保持一致的行为, `Config\Database::$default['DBDebug']` 和 `Config\Database::$tests['DBDebug']` 默认更改为 `true`。通过这些设置, 当数据库错误发生时, 总是会抛出异常。以前, 在生产环境中默认为 `false`。
  - 现在, 如果 `$DBDebug` 为 true, 则在 `BaseBuilder` 中抛出的 `DatabaseException` 会被抛出。以前, 如果 `CI_DEBUG` 为 true 则会抛出。
  - `BaseConnection::$DBDebug` 的默认值已更改为 `true`。

  - 通过这些更改, DBDebug **现在表示数据库错误发生时是否抛出异常**。虽然与调试无关, 但名称未更改。
  - 当 DBDebug 为 true 运行事务时, 即使查询错误也不会抛出异常。以前, 如果发生查询错误, 所有查询都会回滚, 并抛出异常, 因此错误处理 或手动运行事务 不起作用。
  - 现在, 在 Model 中删除没有 WHERE 子句时, 即使 CI_DEBUG 为 false 也会抛出 DatabaseException。以前, 如果 CI_DEBUG 为 true 则会抛出。

异常发生时的 HTTP 状态码和退出代码

以前,CodeIgniter 的异常处理程序在某些情况下会使用 *异常代码*作为 *HTTP 状态码*, 并根据异常代码计算 *退出代码*。但是异常代码与 HTTP 状态码或退出代码之间不应该有逻辑联系。

- 现在, 异常处理程序默认将 HTTP 状态码设置为 500, 并将退出代码设置为常量 EXIT_ERROR (= 1)。
- 你可以在异常类中实现 HTTPExceptionInterface 或 HasExitCodeInterface 来更改 HTTP 状态码或退出代码。请参阅在异常中指定 *HTTP* 状态码 和在异常中指定退出码。

例如, 退出代码已发生如下更改:

- 如果发生未捕获的 ConfigException, 退出代码将是 EXIT_CONFIG (= 3), 而不是 12。
- 如果发生未捕获的 CastException, 退出代码将是 EXIT_CONFIG (= 3), 而不是 9。
- 如果发生未捕获的 DatabaseException, 退出代码将是 EXIT_DATABASE (= 8), 而不是 17。

时间

时间 类的以下方法存在会更改当前对象状态的错误。为了修复这些错误, 时间类已被修复:

- add()
- modify()

- setDate()
- setISODate()
- setTime()
- sub()
- 现在 Time 类扩展 DateTimeImmutable 并完全是不可变的。
- 添加了 TimeLegacy 类用于向后兼容性, 它的行为与未修改的 Time 类相同。

其他

- **辅助函数:** *script_tag()* 和*safe_mailto()* 不再在 <script> 标签中输出 type="text/javascript"。
- **CLI:** 由于 Spark 命令处理的更改, spark 文件已更改。
- **CLI:** CITestStreamFilter::$buffer = '' 不再导致过滤器注册为侦听流。现在有一个 CITestStreamFilter::registration() 方法用于此目的。详情请参阅*在测试中捕获 STDERR 和 STDOUT 流*。
- **数据库:** BaseBuilder::_whereIn() 中的 InvalidArgumentException 是 LogicException 的一种, 不会被配置禁止。以前如果 CI_DEBUG 为 false, 异常会被禁止。
- **数据库:** *BaseConnection::getForeignKeyData()* 返回的数据结构已更改。
- **数据库:** CodeIgniter\Database\BasePreparedQuery 类现在对写入类型的查询返回布尔值, 而不是 Result 类对象。
- **模型:** 如果 Model::update() 方法生成没有 WHERE 子句的 SQL 语句, 现在会引发 DatabaseException; 模型不支持更新所有记录的操作。
- **路由:** RouteCollection::resetRoutes() 会重置自动发现路由。以前一旦发现, 即使调用 RouteCollection::resetRoutes(),RouteCollection 也不会再发现 Routes 文件。

接口变更

备注: 只要你没有扩展相关的 CodeIgniter 核心类或实现这些接口, 所有这些变化都是向后兼容的, 不需要任何干预。

OutgoingRequestInterface

- 添加新的 `OutgoingRequestInterface`, 表示传出请求。
- 添加新的 `OutgoingRequest` 类, 实现 `OutgoingRequestInterface`。
- 现在 `RequestInterface` 扩展 `OutgoingRequestInterface`。
- 现在 `CURLRequest` 扩展 `OutgoingRequest`。
- 现在 `Request` 扩展 `OutgoingRequest`。

其他

- **HTTP:** 在 `MessageInterface` 中添加了缺失的 `getProtocolVersion()`、`getBody()`、`hasHeader()` 和 `getHeaderLine()` 方法。
- **HTTP:** 现在 `ResponseInterface` 扩展 `MessageInterface`。
- **HTTP:** 添加了缺失的 `ResponseInterface::getCSP()` (和 `Response::getCSP()`), `ResponseInterface::getReasonPhrase()` 和 `ResponseInterface::getCookieStore()` 方法。
- **数据库:** 添加了缺失的 `CodeIgniter\Database\ResultInterface::getNumRows()` 方法。
- 参阅验证变更。

方法签名变化

验证变更

ValidationInterface

`ValidationInterface` 已更改, 以消除 `ValidationInterface` 与 `Validation` 类之间的不匹配。

- 为 `ValidationInterface::run()` 添加了第三个参数 `$dbGroup`。
- 接口中添加了以下方法:
  - `ValidationInterface::setRule()`
  - `ValidationInterface::getRules()`
  - `ValidationInterface::getRuleGroup()`
  - `ValidationInterface::setRuleGroup()`
  - `ValidationInterface::loadRuleGroup()`
  - `ValidationInterface::hasError()`
  - `ValidationInterface::listErrors()`
  - `ValidationInterface::showError()`

Validation

当 `$group` 为空时, `Validation::loadRuleGroup()` 的返回值已从 `null` 更改为 `[]`。

数据库

- `CodeIgniter\Database\BasePreparedQuery::close()` 和 `CodeIgniter\Database\PreparedQueryInterface` 的返回类型已更改为 `bool` (之前未定义)。
- `CodeIgniter\Database\Database::loadForge()` 的返回类型已更改为 `Forge`。

- `CodeIgniter\Database\Database::loadUtils()` 的返回类型已更改为 `BaseUtils`。
- `Table::dropForeignKey()` 中的参数名 `$column` 已更改为 `$foreignName`。
- `BaseBuilder::updateBatch()` 的第二个参数 `$index` 已更改为 `$constraints`。它现在接受 array、string 或 `RawSql` 类型。扩展类也应相应更改类型。
- `BaseBuilder::insertBatch()` 和 `BaseBuilder::updateBatch()` 的 `$set` 参数现在接受单行数据的对象。
- **`BaseBuilder::_updateBatch()`**
  - 第二个参数 `$values` 已更改为 `$keys`。
  - 第三个参数 `$index` 已更改为 `$values`。参数类型也已更改为 `array`。

数据库 Forge

- `Forge::dropKey()` 的方法签名已更改。添加了一个可选参数 `$prefixKeyName`。
- `Forge::addKey()` 的方法签名已更改。添加了一个可选参数 `$keyName`。
- `Forge::addPrimaryKey()` 的方法签名已更改。添加了一个可选参数 `$keyName`。
- `Forge::addUniqueKey()` 的方法签名已更改。添加了一个可选参数 `$keyName`。
- 以下方法添加了一个额外的 `$asQuery` 参数。当设置为 `true` 时, 该方法返回一个独立的 SQL 查询。
  - `CodeIgniter\Database\Forge::_processPrimaryKeys()`
- 除了上面添加的 `$asQuery` 参数外, 以下方法现在也返回一个数组。
  - `CodeIgniter\Database\Forge::_processIndexes()`
  - `CodeIgniter\Database\Forge::_processForeignKeys()`

其他

- **API:** `API\ResponseTrait::failServerError()` 的返回类型已更改为 `ResponseInterface`。
- 以下方法已更改为接受 `ResponseInterface` 作为参数, 而不是 `Response`。
  - `Debug\Exceptions::__construct()`
  - `Services::exceptions()`
- **Request:** `IncomingRequest::getJsonVar()` 的 `$index` 参数现在接受 `array`、`string` 或 `null` 值。

增强功能

命令

- 从 `CodeIgniter\CodeIgniter` 类中提取了 Spark 命令的调用处理程序。这将减少控制台调用的成本。
- 添加了 `spark filter:check` 命令来检查路由的过滤器。详情请参阅*Controller Filters*。
- 添加了 `spark make:cell` 命令来创建新的 Cell 文件及其视图。详情请参阅通过命令生成单元。
- 现在 `spark routes` 命令显示路由名称。请参阅*URI 路由*。
- 现在 `spark routes` 命令可以按处理程序排序输出。请参阅按处理程序排序。
- 现在可以使用 `--help` 选项访问 spark 命令的帮助信息 (例如 `php spark serve --help`)
- 添加了 `CLI::promptByMultipleKeys()` 方法以支持多值输入, 与 `promptByKey()` 不同。详情请参阅*promptByMultipleKeys()*。
- HTTP/3 现在被视为有效协议。

测试

- 添加了 `StreamFilterTrait` 以更轻松地使用从 STDOUT 和 STDERR 流中捕获数据。请参阅*测试 CLI 输出*。
- CITestStreamFilter 过滤器类现在实现了向流添加过滤器的方法。请参阅*测试 CLI 输出*。
- 添加了 `PhpStreamWrapper` 以更轻松地使用 `php://stdin` 设置数据。请参阅*测试 CLI 输入*。
- 添加了*Timer::record()* 方法来测量可调用的性能。还增强了通用函数 `timer()` 以接受可选的可调用。
- 将布尔第三参数 `$useExactComparison` 添加到 `TestLogger::didLog()`，它设置是否逐字检查日志消息。默认为 `true`。
- 添加了 `CIUnitTestCase::assertLogContains()` 方法，它通过消息的一部分而不是整个消息来比较日志消息。

数据库

查询构建器

- 向 QueryBuilder 添加了 `upsert()` 和 `upsertBatch()` 方法。参见*更新插入数据*。
- 向 QueryBuilder 添加了 `deleteBatch()` 方法。参见*DeleteBatch*。
- 添加了 `when()` 和 `whenNot()` 方法以有条件地向查询添加子句。详情请参阅*BaseBuilder::when()*。
- 改进了 `Builder::updateBatch()` 的 SQL 结构。详情请参阅*UpdateBatch*。
- 添加了 `BaseBuilder::setQueryAsData()`，它允许从查询中使用 `insertBatch()`、`updateBatch()`、`upsertBatch()`、`deleteBatch()`。参见*insertBatch*。

Forge

- 添加了 `Forge::processIndexes()`，允许在现有表上创建索引。详情请参阅向表添加键。
- 现在可以手动设置索引名称。这些方法包括:`Forge::addKey()`、`Forge::addPrimaryKey()` 和 `Forge::addUniqueKey()`
- 新的 `Forge::dropPrimaryKey()` 方法允许删除表上的主键。参见删除主键。
- *`CodeIgniter\Database\Forge::addForeignKey()`* 现在包括一个名称参数来手动设置外键名称。SQLite3 不支持此功能。
- SQLSRV 现在在使用*Forge::dropColumn()* 时会自动删除 `DEFAULT` 约束。

其他

- SQLite3 有一个新的配置项 `busyTimeout` 来设置表锁定时的超时。
- `BaseConnection::escape()` 现在排除 `RawSql` 数据类型。这允许将 SQL 字符串传递到数据中。
- 改进了*BaseConnection::getForeignKeyData()* 返回的数据。所有 DBMS 返回相同的结构。
- SQLite *BaseConnection::getIndexData()* 现在可以为 *AUTOINCREMENT* 列返回伪索引名为 `PRIMARY`，并且每个返回的索引数据都有 `type` 属性。
- `BasePreparedQuery::close()` 现在在所有 DBMS 中都会释放准备好的语句。以前，它们在 Postgre、SQLSRV 和 OCI8 中没有被释放。参见*close()*。
- 添加了 `BaseConnection::transException()` 用于在事务过程中抛出异常。参见：抛出异常。

模型

- 向 `BaseModel::insertBatch()` 和 `BaseModel::updateBatch()` 方法添加了 before 和 after 事件。请参阅*使用查询构建器*。
- 添加了 `Model::allowEmptyInserts()` 方法以插入空数据。请参阅*Using CodeIgniter's Model*
- 为 Entity 添加了新的*属性转换* 类 `IntBoolCast`。

库

- **Publisher:** 在 Publisher 中添加了 `replace()`、`addLineAfter()` 和 `addLineBefore()` 方法以修改文件。详情请参阅*Publisher*。
- **Encryption:** 现在 Encryption 可以解密使用 CI3 Encryption 加密的数据。请参阅*用于与 CI3 保持兼容性的配置*。
- **CURLRequest:** 在*CURLRequest* 中添加了 HTTP2 版本选项。

辅助函数和函数

- 现在可以通过 **app/Config/Autoload.php** 自动加载辅助函数。
- 添加了新的表单辅助函数*validation_errors()*、*validation_list_errors()* 和*validation_show_error()* 来显示验证错误。
- 如果你将 locale 值作为最后一个参数传入, 可以为*route_to()* 设置 locale。
- 添加了*request()* 和*response()* 函数。
- 添加了*decamelize()* 函数将 camelCase 转换为 snake_case。
- 添加了*is_windows()* 全局函数来检测 Windows 平台。

HTML5 兼容性

通过在 **app/Config/DocTypes.php** 中设置 `$html5` 属性, 可以配置创建诸如 `<input>` 之类的空 HTML 元素时是否排除或包含 solidus 字符 (/) 和右尖括号 (>) 之间的字符。如果将其设置为 `true`, 则会输出不带 / 的 HTML5 兼容标签, 如 `<br>`。

以下项目会受到影响:

- 排版类: 创建 `br` 标签
- 视图解析器: `nl2br` 过滤器
- 诱饵模式:`input` 标签
- 表单辅助函数
- HTML 辅助函数
- 常用函数

错误处理

- 现在可以记录弃用警告而不是抛出异常。详情请参阅记录弃用警告。
- 弃用的记录默认启用。
- 要 临时启用弃用抛出, 请将环境变量 `CODEIGNITER_SCREAM_DEPRECATIONS` 设置为真值。
- `Config\Logger::$threshold` 现在默认为特定于环境。对于生产环境, 默认阈值仍为 `4`, 但对于其他环境已更改为 `9`。

多域名支持

- 添加了 `Config\App::$allowedHostnames` 以设置基准 URL 中主机名以外的主机名。
- 如果设置了 `Config\App::$allowedHostnames`, 当当前 URL 匹配时, 诸如*base_url()*、*current_url()*、*site_url()* 之类的与 URL 相关的函数将返回使用 `Config\App::$allowedHostnames` 中设置的主机名的 URL。

其他

- **路由:** 添加了 `$routes->useSupportedLocalesOnly(true)`, 以便当 URL 中的 locale 不在 `Config\App::$supportedLocales` 中受支持时, 路由器返回 404 Not Found。请参阅*Localization*
- **路由:** 添加了新的 `$routes->view()` 方法以直接返回视图。请参阅*View Routes*。
- **视图:** 视图 Cell 现在是一等公民, 可以位于 **app/Cells** 目录中。请参阅*View Cells*。
- **视图:** 添加了 "受控 Cell", 为你的视图 Cell 提供了更多结构和灵活性。详情请参阅*View Cells*。
- **验证:** 添加了闭包验证规则。详情请参阅*使用闭包规则*。
- **配置:** 现在可以指定要手动发现的 Composer 包。请参阅*Code Modules*。
- **配置:** 添加了 `Config\Session` 类来处理会话配置。
- **调试:** 将 Kint 更新到 5.0.2。
- **请求:** 添加了新的 `$request->getRawInputVar()` 方法从原始流中返回指定变量。请参阅*Retrieving Raw data*。
- **请求:** 添加了新的 `$request->is()` 方法来查询请求类型。请参阅*Determining Request Type*。

消息变更

- 更新英文语言字符串以保持更一致。
- 添加了 `CLI.generator.className.cell` 和 `CLI.generator.viewName.cell`。
- 添加了 **en/Errors.php** 文件。

变更

- **配置**
  - `Config` 类中的所有原子类型属性现已加类型。
  - 有关更改默认值的信息, 请参阅*Upgrading*。
- **更改了 Spark 命令的处理:**

  - CodeIgniter\CodeIgniter 不再处理 Spark 命令。
  - 已删除 CodeIgniter::isSparked() 方法。
  - 已删除 CodeIgniter\CLI\CommandRunner 类, 因为 Spark 命令处理发生了变化。
  - 已删除系统路由配置文件 system/Config/Routes.php。
  - 应用路由配置文件 app/Config/Routes.php 已更改。删除系统路由配置文件的包含。

弃用功能

- 弃用 RouteCollection::localizeRoute()。
- 弃用 RouteCollection::fillRouteParams()。请使用 RouteCollection::buildReverseRoute()。
- 弃用 BaseBuilder::setUpdateBatch() 和 BaseBuilder::setInsertBatch()。请使用 BaseBuilder::setData()。
- 弃用公共属性 Response::$CSP。它将变为 protected。请使用 Response::getCSP()。
- 弃用 CodeIgniter::$path 和 CodeIgniter::setPath()。不再使用。
- 弃用公共属性 IncomingRequest::$uri。它将变为 protected。请使用 IncomingRequest::getUri()。
- 弃用公共属性 IncomingRequest::$config。它将变为 protected。
- 弃用 CLI::isWindows() 方法。请使用 is_windows()。
- 弃用 Config\App 中的会话属性, 改用新的会话配置类 Config\Session。

错误修复

- 修复了所有类型的“准备查询”在写入类型查询中返回 Result 对象而不是 bool 值的错误。
- 修复了在使用 IncomingRequest::getVar() 或 IncomingRequest::getJsonVar() 方法时 JSON 请求中的变量过滤的错误。

- 修复了在使用指定索引调用 `IncomingRequest::getVar()` 或 `IncomingRequest::getJsonVar()` 方法时可能更改变量类型的错误。
- 修复了启用 CSP 时 Honeypot 字段出现的错误。另请参阅*Honeypot 和 CSP*。

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG.md。

版本 4.2.12

发布日期:2023 年 1 月 9 日

CodeIgniter4 4.2.12 版发布

- 错误修复

错误修复

- 修复了 `Request::getIPaddress()` 在 CLI 上会导致错误的问题。
- 修复了 `link_tag()` 缺失 `type="application/rss+xml"`。
- 修复了当格式为 `Y_m_d_His_` 时, `spark migrate:status` 显示了不正确的文件名。
- 修复了如果 `$useAutoIncrement` 为 false, `Model::save()` 在保存对象时会导致错误的问题。

详见仓库的 CHANGELOG_4.2.md 了解已修复错误的完整列表。

版本 4.2.11

发布日期:2022 年 12 月 21 日

CodeIgniter4 4.2.11 版发布

- 安全
- 破坏性变更
- 增强功能

- 错误修复

安全

- 当使用代理时, 攻击者可能会伪造 *IP* 地址的问题已修复。详见 安全公告 GHSA-ghw3-5qvm-3mqc。
- 潜在的会话处理程序漏洞已修复。详见 安全公告 GHSA-6cq5-8cj7-g558。

破坏性变更

- `Config\App::$proxyIPs` 值格式已更改。详见升级指南。
- *DatabaseHandler 驱动程序*、*MemcachedHandler 驱动程序* 和*RedisHandler 驱动程序*的会话数据记录的键已更改。详见升级指南。

增强功能

- 全面支持 PHP 8.2。

错误修复

- 修复了 `FileLocator::locateFile()` 的一个错误, 其中类似的命名空间名称可能被另一个替换, 导致无法找到已存在的文件。
- 修复了 `RedisHandler` 会话类在与 socket 连接一起使用时未使用正确配置的问题。

详见仓库的 CHANGELOG_4.2.md 了解已修复错误的完整列表。

版本 4.2.10

发布日期:2022 年 11 月 5 日

CodeIgniter4 4.2.10 版发布

- 错误修复

错误修复

- 修正了 Session 中错误的 PHPDoc 类型。

详见仓库的 CHANGELOG_4.2.md 了解已修复错误的完整列表。

版本 4.2.9

发布日期:2022 年 10 月 30 日

CodeIgniter4 4.2.9 版发布

- 错误修复

该版本是一个热修复版本, 用于修复 4.2.8 中的 PHPUnit 错误, 没有其他内容变更。

错误修复

- 修复了一个错误, 运行 PHPUnit 时会导致 *PHP Fatal error: Trait "NexusPHPUnitExtensionExpeditable" not found*。

详见仓库的 CHANGELOG_4.2.md 了解已修复错误的完整列表。

版本 4.2.8

发布日期:2022 年 10 月 30 日

CodeIgniter4 4.2.8 版发布

- 弃用
- 错误修复

弃用

- `CodeIgniter::handleRequest()` 的第三个参数 `$returnResponse` 已弃用。

错误修复

- 修复了一个错误, 当 `CodeIgniter\HTTP\IncomingRequest::getPostGet()` 和 `CodeIgniter\HTTP\IncomingRequest::getGetPost()` 方法在 `index` 设置为 `null` 时, 没能返回来自另一个流的值。
- 修复了一个错误, 当在上下文中多次调用 `CodeIgniter\Database\Postgre::replace()` 时, `binds` 没有被正确清理。
- 修复了一个错误, `CodeIgniter\Database\SQLSRV\PreparedQuery::_getResult()` 返回 bool 值而不是资源。
- 修复了错误处理程序中的一个错误, 在回调无法处理错误级别的情况下, 它不会将错误传递给 PHP 的标准错误处理程序。

详见仓库的 CHANGELOG_4.2.md 了解已修复错误的完整列表。

版本 4.2.7

发布日期:2022 年 10 月 6 日

CodeIgniter4 4.2.7 版发布

- 安全
- 破坏性变更
- 消息更改
- 错误修复

安全

- 固定在 *ConfigCookie* 中设置的 *Secure* 或 *HttpOnly* 标志无法反映在发出的 *Cookies* 中的问题。详见 安全公告 GHSA-745p-r637-7vvp。
- 修复了一个错误, 该错误会在 `Config\ContentSecurityPolicy::$autoNonce` 为 false 时导致 CSP 头无法发送。

破坏性变更

- *`set_cookie()`* 和 *`CodeIgniter\HTTP\Response::setCookie()`* 中参数的默认值已修正。现在 `$secure` 和 `$httponly` 的默认值为 `null`, 这些值将被 `Config\Cookie` 中的值替换。
- `Time::__toString()` 现在与区域设置无关。它会以数据库兼容的格式返回字符串, 如 '2022-09-07 12:00:00', 不受区域设置影响。
- Validation 规则 `Validation\Rule::required_without()` 和 `Validation\StrictRules\Rule::required_without()` 的参数已更改, 这些规则的逻辑也已修正。

消息更改

- 修正了 `Language/en/Email.php` 中一些项的拼写错误。
- 在 `Language/en/Validation.php` 中补充了缺失的 `valid_json` 项。

错误修复

详见仓库的 CHANGELOG_4.2.md 了解已修复错误的完整列表。

版本 4.2.6

发布日期:2022 年 9 月 4 日

CodeIgniter 4.2.6 版发布

- 弃用功能
- 错误修复

弃用功能

- 弃用`CodeIgniter\Cookie\Cookie::withNeverExpiring()`。

错误修复

修复了许多错误, 特别是:

- 在 CLI 中使用验证时会发生 AssertionError #6452

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.2.md。

版本 4.2.5

发布日期:2022 年 8 月 28 日

CodeIgniter 4.2.5 版发布

- 不兼容变更
- 增强功能
- 错误修复

不兼容变更

- `BaseConnection::tableExists()` 的方法签名已更改。添加了第二个可选参数 `$cached`。这指示是否使用缓存数据。默认为 `true`, 使用缓存数据。
- `BaseBuilder::_listTables()` 的抽象方法签名已更改。添加了第二个可选参数 `$tableName`。提供表名将只生成该表的 SQL 列表。
- `Validation::processRules()` 和 `Validation::getErrorMessage()` 的方法签名已更改。这两个方法都添加了新的 `$originalField` 参数。

增强功能

- 将 Kint 更新到 4.2.0。

错误修复

- 在主查询中使用子查询时, 会向表别名添加前缀。

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.2.md。

版本 4.2.4

发布日期:2022 年 8 月 13 日

CodeIgniter 4.2.4 版发布

热修复版本, 修复下载错误; 没有其他内容变化。

版本 4.2.3

发布日期:2022 年 8 月 6 日

CodeIgniter 4.2.3 版发布

- 增强功能
- 错误修复

增强功能

- 现在 `Security::generateHash()` 是公开的, 当 `Config\Security::$regenerate` 为 false 时, 可以用来手动重新生成 CSRF token。

错误修复

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.2.md。

版本 4.2.2

发布日期:2022 年 8 月 6 日

CodeIgniter 4.2.2 版发布

- 不兼容变更
- 消息变更
- 变更
- 弃用功能
- 错误修复

不兼容变更

- 现在 `Services::request()` 返回 `IncomingRequest` 或 `CLIRequest`。
- `CodeIgniter\Debug\Exceptions::__construct()` 的方法签名已更改。删除了 `$request` 参数上的 `IncomingRequest` 类型提示。扩展类也应该删除该参数, 以免破坏 LSP。
- `BaseBuilder.php::insert()` 和 `BaseBuilder.php::update()` 的方法签名已更改。删除了 `$set` 参数上的 `?array` 类型提示。
- 修复了一个错误, 该错误会在使用页面缓存时, 在 after 过滤器执行之前缓存页面。现在在 after 过滤器中添加 response headers 或更改 response body 会正确地缓存它们。
- 由于一个错误修复, 现在如果第二个参数 `$len` 是奇数, 带有第一个参数 `'crypto'` 的*random_string()* 会抛出 `InvalidArgumentException`。

消息变更

- 修复了 `Language/en/HTTP.php` 中的 `invalidRoute` 消息。

变更

- 修复:`BaseBuilder::increment()` 和 `BaseBuilder::decrement()` 在查询后不重置 `BaseBuilder` 状态。
- 修复: 带有前导星号 (通配符) 的字段的验证。
- 现在 `CLIRequest::isCLI()` 总是返回 true。
- 现在 `IncommingRequest::isCLI()` 总是返回 false。
- **Vagrantfile.dist** 已移动到 CodeIgniter DevKit。

弃用功能

- 弃用 `Services::request()` 的参数。
- 弃用 `CodeIgniter::gatherOutput()` 的第一个参数 `$cacheConfig`。
- 弃用 `Forge::_createTable()` 的第二个参数 `$ifNotExists`。

错误修复

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.2.md。

版本 4.2.1

发布日期:2022 年 6 月 16 日

CodeIgniter 4.2.1 版发布

- 不兼容变更
  - 行为变化
- 错误修复

不兼容变更

行为变化

- 如果提议的扩展名无效, 则从 MIME 类型猜测文件扩展名的行为已更改。以前, 如果给出的建议扩展名无效, 猜测会提前终止并返回 `null`。现在, 如果给定的建议扩展名无效,MIME 猜测将继续检查扩展名到 MIME 类型的映射。
- 如果存在带前缀的 cookie 和同名但没有前缀的 cookie, 以前的 `get_cookie()` 有一种棘手的行为, 它会返回没有前缀的 cookie。现在这个行为已被修复为一个错误, 并已更改。详细信息请参阅升级。

错误修复

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.2.md。

版本 4.2.0

发布日期:2022 年 6 月 3 日

CodeIgniter 4.2.0 版发布

- 亮点
  - 新贡献者
- 不兼容变更
  - 方法签名更改
  - 行为变化
- 增强功能
  - 新的改进的自动路由
  - 数据库
  - 帮助器和函数
  - 命令
  - 其它

- 变更
- 弃用功能
- 错误修复

亮点

- 将最低 PHP 要求更新为 7.4。
- 为了使默认配置更安全, 默认情况下关闭了自动路由。
- **OCI8 驱动程序用于 Oracle 数据库** (由 ytetsuro 贡献)。参见数据库。
- **改进的自动路由** (可选)(由 kenjis 贡献)。参见新的改进的自动路由。
- 查询构建器对 **子查询**和 **UNION** 的支持 (由 Andrey Pyzhikov 贡献)。参见数据库。

新贡献者

- @fdomgjoni99 在 #5608 中做出了他们的首次贡献
- @Nudasoft 在 #5715 中做出了他们的首次贡献
- @beganovich 在 #5823 中做出了他们的首次贡献
- @tcgumus 在 #5851 中做出了他们的首次贡献
- @michaelrk02 在 #5878 中做出了他们的首次贡献
- @datamweb 在 #5894 中做出了他们的首次贡献
- @xlii-chl 在 #5884 中做出了他们的首次贡献
- @valmorflores 在 #6051 中做出了他们的首次贡献
- @tearoom6 在 #6012 中做出了他们的首次贡献
- @lonnie-vault 在 #6060 中做出了他们的首次贡献

不兼容变更

方法签名更改

- `CodeIgniter\Database\BaseBuilder::join()` 和 `CodeIgniter\Database\*\Builder::join()` 的方法签名已更改。
- `Validation::setRule()` 的方法签名已更改。删除了 `$rules` 参数上的 `string` 类型提示。扩展类应相应地删除参数, 以免破坏 LSP。
- `CodeIgniter\CLI\CommandRunner::_remap()` 的方法签名已更改以修复一个错误。
- `Service::reset()` 和 `CIUnitTestCase::resetServices()` 的默认参数值已从 `false` 更改为 `true`。这是为了在测试期间消除意外问题, 例如 `lang()` 获取不到翻译消息。

行为变化

- **`CodeIgniter\CodeIgniter` 类有了一个新属性 `$context`, 它在运行时必须有正确的上下文。因此, 以下文件已更改:**
  - `public/index.php`
  - `spark`
- `system/bootstrap.php` 文件已修改, 可以轻松实现 预加载。返回 `CodeIgniter` 实例和加载 `.env` 文件已移动到 `index.php` 和 `spark` 中。
- `CodeIgniter\Autoloader\Autoloader::initialize()` 已更改行为以修复一个错误。以前, 当 `$modules->discoverInComposer` 为 true 时, 它只使用 Composer 的类映射。现在, 如果可用, 它总是使用 Composer 的类映射。
- 修复了一个错误, 更改了通过*CLI::color()* 输出的颜色代码。
- 为防止来自网页浏览器的意外访问, 如果控制器添加到 cli 路由中 (`$routes->cli()`), 则该控制器的所有方法都不再通过自动路由访问。
- 对于那些扩展历史记录收集器的用户, 他们可能需要更新 `History::setFiles()` 方法, 这可能会破坏向后兼容性。
- 修复了*dot_array_search()* 的意外行为。现在 `dot_array_search('foo.bar.baz', ['foo' => ['bar' => 23]])` 返回 `null`。以前的版本返回 `23`。

- `CodeIgniter::storePreviousURL()` 已更改为仅存储 Content-Type 为 `text/html` 的 URL。它还会影响*previous_url()* 和*redirect()->back()* 的行为。

增强功能

新的改进的自动路由

添加了一个可选的新的更安全的自动路由器。这些是与传统自动路由的更改:

- **控制器方法需要 HTTP 动词前缀, 如 `getIndex()`、`postCreate()`。**
  - 开发人员总是知道 HTTP 方法, 因此不期望的 HTTP 方法的请求不会通过。
- **默认控制器 (默认为 `Home`) 和默认方法 (默认为 `index`) 必须在 URI 中省略。**
  - 它限制控制器方法和 URI 之间的一对一对应关系。
  - 例如, 默认情况下, 你可以访问 `/`, 但 `/home` 和 `/home/index` 会返回 404。
- **它检查方法的参数数。**
  - 如果 URI 中的参数多于方法的参数, 则结果为 404。
- **它不支持 `_remap()` 方法。**
  - 它限制控制器方法和 URI 之间的一对一对应关系。
- **无法访问定义路由中的控制器。**
  - 它完全区分通过 **自动路由**可访问的控制器和通过 **定义路由**可访问的控制器。

详情请参阅自动路由（改进版）。

数据库

- **添加了新的 OCI8 数据库驱动。**
  - 它可以访问 Oracle 数据库并支持 SQL 和 PL/SQL 语句。
- **查询构建器**
  - 在 FROM 部分添加了子查询。请参阅子查询。

  - 在 SELECT 部分添加了子查询。请参阅*Select*。
  - `BaseBuilder::buildSubquery()` 方法现在可以接受可选的第三个参数 `string $alias`。
  - 并集查询。请参阅*联合查询*。
- **原生 SQL 字符串支持**
  - 添加了表示原生 SQL 字符串的类 `CodeIgniter\Database\RawSql`。
  - *select()*、*where()*、*like()*、*join()* 接受 `CodeIgniter\Database\RawSql` 实例。
  - `DBForge::addField()` 支持原生 SQL 字符串作为默认值。请参阅*作为默认值的原始 SQL 字符串*。
- SQLite3 有一个新的配置项 `foreignKeys`, 用于启用外键约束。

帮助器和函数

- HTML 辅助函数 `script_tag()` 现在使用 `null` 值以最小化形式编写布尔属性:`<script src="..." defer />`。请参阅*script_tag()* 的示例代码。
- 在 `get_filenames()` 中添加第 4 个参数 `$includeDir`。请参阅*get_filenames()*。
- **通过 `log_message()` 记录的异常信息现在得到了改进。它现在包括异常发生的文件和行号。它也不再截断消息。**
  - 日志格式也已更改。如果用户依赖应用中的日志格式, 新的日志格式为 "<1-based count> <cleaned filepath>(<line>): <class><function><args>"

命令

- **添加了 `spark db:table` 命令。详情请参见数据库命令。**
  - **你现在可以在终端中查看当前连接数据库中的所有表名。**
    - `spark db:table --show`
  - **或者你可以查看一个表的字段名和记录。**
    - `spark db:table my_table`

    * `spark db:table my_table --limit-rows 50 --limit-field-value 20 --desc`

  - **或者你可以查看元数据, 如列类型、表的最大长度。**

    * `spark db:table my_table --metadata`

- `spark routes` 命令现在显示闭包路由、自动路由和过滤器。请参阅*URI 路由*。

其它

- 在控制器中添加了 `$this->validateData()`。请参阅*$this->validateData()*。
- **内容安全策略 (CSP) 增强**
  - 在 `Config\ContentSecurityPolicy` 中添加了配置 `$scriptNonceTag` 和 `$styleNonceTag` 以自定义 CSP 占位符 (`{csp-script-nonce}` 和 `{csp-style-nonce}`)
  - 在 `Config\ContentSecurityPolicy` 中添加了配置 `$autoNonce` 以禁用 CSP 占位符替换
  - 添加了函数 `csp_script_nonce()` 和 `csp_style_nonce()` 来获取 nonce 属性
  - 详情请参阅内容安全策略。
- 新的视图装饰器 允许在缓存之前修改生成的 HTML。
- 添加了验证严格规则。请参阅传统规则与严格规则。
- 在 **app/Config/Mimes.php** 中添加了对 webp 文件的支持。
- `RouteCollection::addRedirect()` 现在可以使用占位符。详细信息请参阅重定向路由。
- **调试栏增强**
  - 调试工具栏现在使用 `microtime()` 而不是 `time()`。
- 添加了 预加载 的示例文件。请参阅 **preload.php**。

变更

- 将最低 PHP 要求更新为 7.4。
- 为了使默认配置更安全, 默认情况下关闭了自动路由。
- 验证。当使用带通配符 (*) 的字段时, 更改了错误生成方式。现在错误键包含完整路径。请参阅获取所有错误。
- 当使用通配符时, `Validation::getError()` 将返回匹配掩码的所有找到的错误字符串。
- 当前版本的内容安全策略 (CSP) 为脚本和样式标签输出一个 nonce。以前的版本为每个标签输出一个 nonce。
- 发送 cookie 的过程已移动到 `Response` 类中。现在 `Session` 类不再发送 cookie, 而是将它们设置到 Response 中。

弃用功能

- 弃用 `CodeIgniter\Database\SQLSRV\Connection::getError()`。请使用 `CodeIgniter\Database\SQLSRV\Connection::error()`。
- 弃用 `CodeIgniter\Debug\Exceptions::cleanPath()` 和 `CodeIgniter\Debug\Toolbar\Collectors\BaseCollector::cleanPath()`。请使用 `clean_path()` 函数。
- 弃用 `CodeIgniter\Log\Logger::cleanFilenames()` 和 `CodeIgniter\Test\TestLogger::cleanup()`。请使用 `clean_path()` 函数。
- 弃用 `CodeIgniter\Router\Router::setDefaultController()`。
- 在 **spark** 中弃用常量 `SPARKED`。请使用 `CodeIgniter\CodeIgniter` 中的 `$context` 属性。
- 弃用 `CodeIgniter\Autoloader\Autoloader::discoverComposerNamespaces()`, 不再使用。
- 弃用常量 `EVENT_PRIORITY_LOW`、`EVENT_PRIORITY_NORMAL` 和 `EVENT_PRIORITY_HIGH`。请使用类常量 `CodeIgniter\Events\Events::PRIORITY_LOW`、`CodeIgniter\Events\Events::PRIORITY_NORMAL` 和 `CodeIgniter\Events\Events::PRIORITY_HIGH`。

错误修复

- SQLSRV 驱动程序忽略配置中的端口值。

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.2.md。

版本 4.1.9

发布日期:2022 年 2 月 25 日

CodeIgniter 4.1.9 版发布

- 安全性

安全性

- 修复了 *远程 CLI 命令执行漏洞*。更多信息请参阅 安全公告 GHSA-xjp4-6w75-qrj7。
- 修复了 *跨站请求伪造 (CSRF) 保护绕过漏洞*。更多信息请参阅 安全公告 GHSA-4v37-24gm-h554。

版本 4.1.8

发布日期:2022 年 1 月 24 日

CodeIgniter 4.1.8 版发布

- 安全性

安全性

- 修复了 `API\ResponseTrait` 中的 *XSS 漏洞*。更多信息请参阅 安全公告。

版本 4.1.7

发布日期:2022 年 1 月 9 日

CodeIgniter 4.1.7 版发布

- 不兼容变更
- 增强功能
- 变更
- 弃用功能
- 错误修复

不兼容变更

- 因为 `FILTER_SANITIZE_STRING` 从 PHP 8.1 开始已被废弃, 当 `$xssClean` 为 true 时使用它的 `get_cookie()` 改变了输出。现在它使用 `FILTER_SANITIZE_FULL_SPECIAL_CHARS`。请注意, 使用 XSS 过滤是一种不好的做法。它不能完全防止 XSS 攻击。建议在视图中使用正确的 `$context` 调用 `esc()`。

增强功能

无

变更

无

弃用功能

无

错误修复

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.1.md。

版本 4.1.6

发布日期:2022 年 1 月 3 日

CodeIgniter 4.1.6 版发布

- 安全性
- 不兼容变更
  - 验证变更
- 增强功能
- 变更
- 弃用功能
  - 发送 *Cookie*
- 错误修复

安全性

- 修复了 `old()` 函数中存在的“不受信任数据的反序列化”问题。更多信息请参阅安全公告。

不兼容变更

- `BaseBuilder::$tableName` 中不再存储多个表名 - 会改用空字符串。

验证变更

- 验证的前一个版本无法处理数组项。由于此 bug 修复, 验证结果可能不同, 或引发 `TypeError`。但前一个版本的结果可能是不正确的。
- 验证将多个字段的验证过程分开, 如 `contacts.*.name` 和单个字段。当单个字段具有数组数据时, 前一个版本会验证数组的每个元素。验证规则将数组的一个元素作为参数。另一方面, 当前版本将数组作为一个整体传递给验证规则。

增强功能

- 全面支持 PHP 8.1。
- 调试工具栏上的数据库面板现在会显示查询调用的位置。还显示完整的回溯。
- QueryBuilder 中的子查询 现在可以是 BaseBuilder 类的实例。
- 将 Kint 从 ^3.3 升级到 ^4.0。

变更

- 发送 cookie 的过程已移动到 `Response` 类中。现在 `Security` 和 `CookieStore` 类不再发送 cookie, 而是将它们设置到 Response 中。

弃用功能

- 弃用 `Seeder::faker()` 和 `Seeder::$faker`。
- 弃用 `BaseBuilder::cleanClone()`。

发送 Cookie

发送 cookie 的过程已移至 `Response` 类中。

以下方法已被弃用:

- `CookieStore::dispatch()`
- `CookieStore::setRawCookie()`
- `CookieStore::setCookie()`
- `Security::sendCookie()`
- `Security::doSendCookie()`

错误修复

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.1.md。

版本 4.1.5

发布日期:2021 年 11 月 8 日

CodeIgniter 4.1.5 版发布

- 不兼容变更
- 增强功能
- 变更
- 弃用功能
- 错误修复

不兼容变更

- 修复了 CSRF 保护上的一个错误。现在, 当应用 CSRF 过滤器时,CSRF 保护也适用于 PUT/PATCH/DELETE 请求。如果使用这些请求, 则需要发送 CSRF token。
- 在以前的版本中, 如果你没有提供自己的 headers, `CURLRequest` 会由于一个错误而发送来自浏览器的请求头。从本版本开始, 它不再发送它们。
- 修复了 `BaseBuilder::insertBatch()` 在 `testMode` 下的返回值。现在它返回 SQL 字符串数组而不是受影响的行数。此更改是为了与批处理方法的返回类型保持兼容。现在 `BaseBuilder::insertBatch()` 的返回数据类型与 *updateBatch()* 方法相同。
- 对 `BaseBuilder::insertBatch()` 和 `BaseBuilder::updateBatch()` 方法中数据处理方式进行了重大优化。这减少了内存使用量和更快的查询处理速度。作为权衡, `$query->getOriginalQuery()` 方法生成的结果发生了变化。它不再返回带有绑定参数的查询, 而是实际运行的查询。

增强功能

- 为保留字符添加了缓存配置
- `Forge` 类的*addForeignKey()* 函数现在可以在数组中定义复合外键
- `Forge` 类的 `dropKey` 函数可以移除键
- 现在环境变量中可以使用 _ 作为分隔符。参见命名空间分隔符。
- 为路由添加了多个过滤器和类名过滤器
- 减少了 `insertBatch()` 和 `updateBatch()` 的内存使用量
- 添加了基于会话的 *CSRF* 保护
- 为 `Validation` 添加了 `valid_url_strict` 规则
- **调试工具栏**
  - 在时间线中添加了格式化的查询字符串
  - 改进了查询字符串的关键字高亮和转义

变更

- 始终在 `BaseBuilder` 的 `set`、`setUpdateBatch` 和 `insertBatch` 函数中转义标识符。

弃用功能

- 弃用 `CodeIgniter\\Cache\\Handlers\\BaseHandler::RESERVED_CHARACTERS`, 改用新的配置属性

错误修复

有关完整的错误修复列表, 请参阅仓库的 CHANGELOG_4.1.md。

版本 4.1.4

发布日期:2021 年 9 月 6 日

CodeIgniter 4.1.4 版发布

- 不兼容变更

本次发布侧重于代码样式。除下面提到的变更外, 所有变更都是为了使代码符合新的 CodeIgniter 编码标准 (基于 PSR-12) 而进行的视觉调整。

不兼容变更

- 以下方法已从 “public” 改为 “protected” , 以匹配其父类方法并更好地与其用法对齐:
  - `CodeIgniter\Database\MySQLi\Connection::execute()`
  - `CodeIgniter\Database\MySQLi\Connection::_fieldData()`
  - `CodeIgniter\Database\MySQLi\Connection::_indexData()`
  - `CodeIgniter\Database\MySQLi\Connection::_foreignKeyData()`
  - `CodeIgniter\Database\Postgre\Builder::_like_statement()`

  - `CodeIgniter\Database\Postgre\Connection::execute()`
  - `CodeIgniter\Database\Postgre\Connection::_fieldData()`
  - `CodeIgniter\Database\Postgre\Connection::_indexData()`
  - `CodeIgniter\Database\Postgre\Connection::_foreignKeyData()`
  - `CodeIgniter\Database\SQLSRV\Connection::execute()`
  - `CodeIgniter\Database\SQLSRV\Connection::_fieldData()`
  - `CodeIgniter\Database\SQLSRV\Connection::_indexData()`
  - `CodeIgniter\Database\SQLSRV\Connection::_foreignKeyData()`
  - `CodeIgniter\Database\SQLite3\Connection::execute()`
  - `CodeIgniter\Database\SQLite3\Connection::_fieldData()`
  - `CodeIgniter\Database\SQLite3\Connection::_indexData()`
  - `CodeIgniter\Database\SQLite3\Connection::_foreignKeyData()`
  - `CodeIgniter\Images\Handlers\GDHandler::_flatten()`
  - `CodeIgniter\Images\Handlers\GDHandler::_flip()`
  - `CodeIgniter\Images\Handlers\ImageMagickHandler::_flatten()`
  - `CodeIgniter\Images\Handlers\ImageMagickHandler::_flip()`
  - `CodeIgniter\Test\Mock\MockIncomingRequest::detectURI()`
  - `CodeIgniter\Test\Mock\MockSecurity.php::sendCookie()`
- 为了与 PHP 8.1 的严格继承检查兼容, 在可能的情况下, 以下方法签名添加了返回类型以匹配其父类签名:
  - `CodeIgniter\Cookie\Cookie::offsetExists()`
  - `CodeIgniter\Cookie\Cookie::offsetSet()`
  - `CodeIgniter\Cookie\Cookie::offsetUnset()`
  - `CodeIgniter\Cookie\CookieStore::getIterator()`
  - `CodeIgniter\I18n\Time::__wakeup()`
  - `CodeIgniter\Test\Filters\CITestStreamFilter::filter()`

- 与 PHP 8.1 的严格继承检查相关, 实现 `SessionHandlerInterface` 的以下会话处理程序的公共方法已修改为与接口匹配:
  - `CodeIgniter\Session\Handlers\ArrayHandler`
  - `CodeIgniter\Session\Handlers\DatabaseHandler`
  - `CodeIgniter\Session\Handlers\FileHandler`
  - `CodeIgniter\Session\Handlers\MemcachedHandler`
  - `CodeIgniter\Session\Handlers\RedisHandler`

有关完整的更改列表, 请参阅仓库的 CHANGELOG_4.1.md。

版本 4.1.3

发布日期:2021 年 6 月 6 日

CodeIgniter 4.1.3 版发布

- 增强功能
- 变更
- 错误修复

增强功能

- 文件辅助函数中新增功能:`directory_mirror()` 和 `same_file()`
- 实现了 NexusPHP 的 `Tachycardia` 用于识别缓慢的测试
- 为 `Cache` 配置新增了一个 `$ttl` 选项以供未来使用

变更

- 在测试矩阵中添加了 MySQL 8.0
- 从 `$_SERVER` 改进了环境检测
- 通过 Rector 和分析进行了大范围的代码改进

错误修复

- 修复了 `CURLRequest` 会尝试使用项目 URI 而不是其基本 URI 的问题
- 修复了在 `cgi-fcgi` 下没有检测到 CLI 模式的问题
- 修复了 Cookie 构造中的逻辑错误
- 修复了 SQLite3 的 `Forge` 类中与不正确的属性名相关的许多问题

有关已修复的错误列表, 请参见仓库的 CHANGELOG_4.1.md。

版本 4.1.2

发布日期:2021 年 5 月 16 日

CodeIgniter 4.1.2 版发布

- 不兼容变更
- 增强功能
- 变更
- 弃用功能
- 错误修复

不兼容变更

修复了 `current_url()` 中的一个错误, 该错误阻止了配置 `indexPage` 的返回值中包含该值。使用 `App::$indexPage` 的任何安装都应该期望 `current_url()` 及其所有依赖项 (包括响应测试、分页器、表单辅助函数、分页器和视图解析器) 返回的值发生变化。

增强功能

- 新增 HTTP 类 `Cookie` 和 `CookieStore`, 用于抽象化 Web cookie。
- 新增 HTTP 测试的 `assertRedirectTo()` 断言。
- 新增日志处理程序 `ErrorlogHandler`, 可写入 `error_log()`。
- 实体类。新增自定义类型转换功能。
- 路由中新增选项。`priority` 选项可降低特定路由处理的优先级。
- `Autoloader` 类现在可以加载不包含 PHP 类的文件。`Config\Autoload` 类的 `$files` 属性中将列出 非类文件。

变更

- 视图中的布局现在支持嵌套区段。
- `Response::getCookie` 现在返回 `Cookie` 实例, 而不是 cookie 属性数组。
- `Response::getCookies` 现在返回 `Cookie` 实例数组, 而不是属性数组的数组。
- 为了消除现代浏览器控制台的警告, 空 samesite 值在派发 cookie 时将默认为 `Lax`。
- *Model::errors()* 和 *BaseModel::errors()* 现在总是返回 *array*; 没有定义变化, 但 docblock 已更新。
- `Entity::castAs` 的行为发生了变化。新增参数 `$method`。类型转换已移动到单独的处理程序中。
- 实体类。将无效值传递给时间戳转换现在会引发异常。
- `Entity::castAsJson` 使用外部转换处理程序 `JsonCast::get`。
- `Entity::mutateDate` 使用外部转换处理程序 `DatetimeCast::get`。
- 为了使 `Config\**` 类能够从 `.env` 获取各自的属性值, 现在有必要用类的名称为属性命名空间。以前, 属性名称就足够了, 但现在不允许, 因为它可以获取系统环境变量, 如 `PATH`。
- 数组辅助函数 `_array_search_dot` 现在标记为仅 `@internal` 使用。由于 `dot_array_search` 使用了 `_array_search_dot`, 所以用户不应在代码中直接使用 `_array_search_dot`。
- `CacheInterface::getMetaData()` 对未命中返回 `null`, 对命中返回包含至少 `expires` 键的数组 (设置为绝对时期过期时间) 或 `null` (表示 “永不过期”)。

File、Memcached 和 Wincache 处理程序仍返回 `false`, 这在未来的版本中将变为 `null`。

弃用功能

- 弃用 `CodeIgniter\View\View::$currentSection` 属性。
- 弃用用于 `CookieException` 自身异常消息的无效 cookie samesite 的语言字符串和异常。
- 弃用 *CodeIgniterEntity*, 改用 *CodeIgniterEntityEntity*。
- 弃用 `Response` 的与 cookie 相关的属性, 改为使用 `Cookie` 类。
- 弃用 `Security` 的与 cookie 相关的属性, 改为使用 `Cookie` 类。
- 弃用 `Session` 的与 cookie 相关的属性, 改为使用 `Cookie` 类。
- 弃用 `Security::isExpired()`, 改为使用 `Cookie` 内部的过期状态。
- 弃用 `CIDatabaseTestCase`, 改为使用 `DatabaseTestTrait`。
- 弃用 `FeatureTestCase`, 改为使用 `FeatureTestTrait`。
- 弃用 `ControllerTester`, 改为使用 `ControllerTestTrait`。
- 统一并弃用 `ControllerResponse` 和 `FeatureResponse`, 改用 `TestResponse`。
- 弃用 `Time::instance()`, 改用 `Time::createFromInstance()` (现在接受 `DateTimeInterface`)。
- 弃用 `IncomingRequest::removeRelativeDirectory()`, 改用 `URI::removeDotSegments()`。
- 弃用 `\API\ResponseTrait::failValidationError()`, 改用 `\API\ResponseTrait::failValidationErrors()`。

错误修复

- `BaseConnection::query()` 现在对失败的查询返回 `false` (除非 `DBDebug==true`, 在这种情况下会抛出异常), 并根据文档为写入类型的查询返回布尔值。

有关已修复的错误列表, 请参见仓库的 CHANGELOG_4.1.md。

版本 4.1.1

发布日期:2021 年 2 月 1 日

CodeIgniter 4.1.1 版发布

- 错误修复

错误修复

- 修复了 `.gitattributes` 阻止框架下载的问题。

请注意, 此修复也被回溯应用于 **framework** 仓库中的 4.0.5 版。

有关已修复的错误列表, 请参见仓库的 CHANGELOG_4.1.md。

版本 4.1.0

发布日期:2021 年 1 月 31 日

CodeIgniter 4.1.0 版发布

- 不兼容变更
- 变更
- 弃用功能
- 错误修复

不兼容变更

- `Autoloader::loadLegacy()` 方法此前用于迁移到 CodeIgniter v4 时非命名空间类的迁移。从 v4.1.0 开始, 此支持已被删除。

变更

- 不再支持 PHP 7.2

弃用功能

- 弃用 `Model::fillPlaceholders()` 方法, 请使用 `Validation::fillPlaceholders()`。

错误修复

有关已修复的错误列表, 请参见仓库的 CHANGELOG_4.1.md。

版本 4.0.5

发布日期:2021 年 1 月 31 日

CodeIgniter 4.0.5 版发布

- 增强功能
- 变更
- 弃用功能
- 错误修复

增强功能

- 新增 URL 辅助函数*url_to()*, 可基于控制器创建绝对 URLs。
- 新增 Model 选项:`$useAutoIncrement`, 设置为 `false` 时, 允许你为表中的每条记录提供自己的主键。这在实现 1:1 关系或在模型中使用 UUID 时很方便。
- 新增 URL 辅助函数*url_is()*, 可检查当前 URL 是否匹配给定字符串。
- 服务类的参数类型现在进行了严格定义。这将确保不会传入其他配置实例。如果需要传递带有额外属性的新配置, 则需要扩展特定的配置类。
- 支持为 Session 和 CSRF cookie 设置 SameSite 属性。出于安全和与最新浏览器版本的兼容性考虑, 默认设置为 `Lax`。
- 在 `Config\Mimes::guessExtensionFromType()` 中, 现在只有在没有提议扩展名时 (即通常不用于上传的文件), 才会反向搜索 `$mimes` 数组。
- 上传文件的文件扩展名的 getter 函数现在具有不同的回退值 (`UploadedFile::getClientExtension()` 用于 `UploadedFile::getExtension()`, `''` 用于 `UploadedFile::guessExtension()`)。这是一个安全修复, 使该过程不太依赖客户端。
- Cache `FileHandler` 现在允许通过 `Config\Cache` 设置文件权限模式。

变更

- 定义在 **system/Language/en/** 中的系统消息现在严格用于内部框架使用, 不再涵盖向后兼容性 (BC) 保证。用户可以在应用中使用这些消息, 但有自己的风险。

弃用功能

- 弃用 `BaseCommand::getPad()` , 改用 `BaseCommand::setPad()`。
- 弃用 `CodeIgniter\Controller::loadHelpers()`, 改用 `helper()` 函数。
- 弃用 `Config\Format::getFormatter()`, 改用 `CodeIgniter\Format\Format::getFormatter()`。
- 弃用 `CodeIgniter\Security\Security::CSRFVerify()`, 改用 `CodeIgniter\Security\Security::verify()`。

- 弃用 CodeIgniter\Security\Security::getCSRFHash()，改用 CodeIgniter\Security\Security::getHash()。
- 弃用 CodeIgniter\Security\Security::getCSRTokenName()，改用 CodeIgniter\Security\Security::getTokenName()。
- 弃用 Config\App::$CSRFTokenName，改用 Config\Security::$tokenName。
- 弃用 Config\App::$CSRFHeaderName，改用 Config\Security::$headerName。
- 弃用 Config\App::$CSRFCookieName，改用 Config\Security::$cookieName。
- 弃用 Config\App::$CSRFExpire，改用 Config\Security::$expire。
- 弃用 Config\App::$CSRFRegenerate，改用 Config\Security::$regenerate。
- 弃用 Config\App::$CSRFRedirect，改用 Config\Security::$redirect。
- 弃用 Config\App::$CSRFSameSite，改用 Config\Security::$samesite。
- 弃用 migrate:create 命令，改用 make:migration 命令。
- 弃用 CodeIgniter\Database\ModelFactory，改用 CodeIgniter\Config\Factories::models()。
- 弃用 CodeIgniter\Config\Config，改用 CodeIgniter\Config\Factories::config()。
- 弃用 CodeIgniter\HTTP\Message::getHeader()，改用 header() 以准备迁移到 PSR-7。
- 弃用 CodeIgniter\HTTP\Message::getHeaders()，改用 headers() 以准备迁移到 PSR-7。
- 弃用 Config\Cache::$storePath，改用 Config\Cache::$file['storePath']。

错误修复

- 修复了 `Entity` 类中的一个错误, 声明类参数阻止了数据传播到 `attributes` 数组。
- 对环境变量 `encryption.key` 的处理发生了变化。以前, 显式函数调用, 如 `getenv('encryption.key')` 或 `env('encryption.key')`, 其中值具有特殊前缀 `hex2bin:`, 会自动转换为二进制字符串并返回。这现在已更改为只返回具有前缀的字符字符串。此更改是由于在 Windows 平台上处理环境变量中的二进制字符串不兼容导致的。但是, 通过 `Encryption` 类配置访问 `$key` 保持不变, 仍然返回二进制字符串。
- `Config\Services` (在 **app/Config/Services.php** 中) 现在扩展 `CodeIgniter\Config\BaseService`, 以允许正确发现第三方服务。

有关已修复的错误列表, 请参见仓库的 CHANGELOG_4.0.md。

版本 4.0.4

发布日期:2020 年 7 月 15 日

CodeIgniter4 4.0.4 版发布

- *不兼容变更*
- *增强功能*
- *修复的 Bug*

不兼容变更

- 在 `FilterInterface` 中为 `after()` 和 `before()` 添加了 `$arguments` 参数。这是一个不兼容的变更, 所以所有实现 `FilterInterface` 的代码都必须更新。

增强功能

- SQLite3 数据库的位置已更改, 默认现在将位于 `writable` 文件夹中, 而不是 `public` 文件夹中。
- 新增 CLI 命令:`cache:clear` 将销毁当前缓存引擎中的所有数据。
- 感谢 samsonasik, 我们有几个库被提升到了 100% 的测试覆盖率。
- 一些小的性能提升。每一点提升都很重要!
- `getFieldData()` 在数据库结果类中现在会返回列类型和长度。
- 默认.htaccess 文件已更新, 可对扩展字符工作得更好。
- 新增测试功能:*Fabricator* 可以简化在测试中创建模拟类的过程。
- Model 类现在可以在运行时覆盖回调函数。对测试很有用。
- *功能测试* 在整体上有许多改进。
- 新增*command() 辅助函数* 以通过编程方式运行 CLI 命令。对测试和计划任务很有用。
- 新增命令 `make:seeder` 以生成*数据库 Seed 类* 骨架文件。
- Windows 中现在可以在 CLI 使用颜色, 以及其他与 Windows 相关的 CLI 改进。
- 新增辅助函数*mb_url_title()*, 功能与 `url_title()` 相同, 但会自动转义扩展的 URL 字符。
- *图像类库* 现在支持 `webp` 图像。
- 在路由器中为正则表达式添加了 Unicode 支持。
- 为*delete_files()* 辅助函数添加了删除隐藏文件夹的支持。
- `fetchGlobal()` 在请求类中现在支持对数组数据应用过滤器, 而不仅仅是第一个项目。
- `file` 验证现在支持文件数组。
- URI 类现在支持 `setSilent()` 方法, 可禁用抛出异常。
- 新增参数到 `URI::getSegment()`, 允许我们改变如果什么都不存在时的默认返回值。
- 在图像中实现 `withResource()`, 以便可以使用方法链调用代替提供的 getResource()(在测试中使用)。只需压缩图像。

- 重定向时, 现在可以从全局响应对象复制 Cookie 和 header, 使用新的 `withCookies()` 和 `withHeaders()` 方法。
- 支持在加密类 中使用 `$key` 参数或.env 文件中使用特殊前缀 "hex2bin:"。

修复的 Bug

- 修复了 SQLite3 数据库的位置, 默认现在将位于 `writable` 文件夹中, 而不是 `public` 文件夹中。
- 修复了 `force_https()` 可能会添加第二次 `https://` 的 bug。
- 修复了 CurlRequest 中可能导致不正确的 "100 Continue" 头的 bug。
- 修复了当 `$target` 参数为 `null` 时 Image::save() 的 bug。
- 修复了当 `$default` 参数设置为 `true` 时 `set_checkbox()` 和 `set_radio()` 的问题。
- 修复了 Model 类中结果对象处理的问题。
- 修复了 SQLite 数据库中的转义字符。
- 修复了在主键为 null 时在 Postgres 和实体上的插入问题。
- CLI 脚本现在可以正确识别参数中的破折号。
- CURLRequest 现在使用多部分数据正确设置内容长度。
- ImageMagick 处理程序的稳定性各种改进。
- 在配置文件中设置验证错误现在应该可以工作。
- 从实体保存 JSON 时不再转义 Unicode 字符。
- 重定向现在应该可以使用自定义 HTTP 代码正常工作。
- `Time::setTimezone()` 现在可以正确工作。
- 为 Postgres 添加了完全外连接支持。
- 填充过程中, 实体中的一些转换项 (如数组、json) 没有被正确设置。
- 修复了在某些情况下,GD 处理程序会尝试两次压缩图像的 bug。
- 确保翻译输出逻辑可用于所选语言环境、破折号语言环境和回退的 "en"。
- 修复通过 API 在 PostgreSQL 上通过 POST/PUT 调用的 `is_unique/is_not_unique` 验证。

- 修复了 `after()` 中没有传递过滤器参数的 bug。

有关完整的已修复 bug 列表, 请参阅代码库的 CHANGELOG_4.0.md。

版本 4.0.3

发布日期:2020 年 5 月 1 日

CodeIgniter4 4.0.3 版发布

- *增强功能*
- *修复的 Bug*

感谢社区的贡献, 总共关闭了 74 个 bug, 关闭了 21 个 issue, 合并了 88 个 pull 请求。详细信息请参阅 CHANGELOG.md。这里简要介绍一些较大的变动。

增强功能

- API 响应特性现在只在控制器的 `$format` 变量为 `null` 时才通过内容协商来确定最终格式 (json/xml)。如果它包含 `json` 或 `xml`, 那么将总是返回那种格式。
- 分页现在可以真正创建下一页和上一页的链接, 而不仅仅是下一组/上一组的链接。
- 想在本地构建用户指南副本的 Windows 用户现在可以使用包含的 `make.bat` 文件。
- IncomingRequest 类中的 Locale 匹配现在可以匹配更宽泛的组, 例如即使浏览器只提供更具体的 locale 代码 (如 `fr-FR`), 也可以匹配到 `fr`。
- 添加了嵌套语言定义的功能。
- 在 form_open(‘action’) 中可以将 {locale} 替换为 request->getLocale()。
- CLI 命令生成的表格现在可以着色。

修复的 Bug

- 在具有字符串主键的表上通过 Model 删除现在可以工作。
- 默认分页模板修复为使用正确的 locale。
- 用户指南中进行了大量调整和更正。
- 修复了自定义命名空间中偶尔找不到文件的问题。主要影响控制台命令。

有关完整的已修复 bug 列表, 请参阅代码库的 CHANGELOG_4.0.md。

版本 4.0.0

发布日期:2020 年 2 月 24 日

CodeIgniter4 4.0 版发布

- 增强功能
- 仓库变更

增强功能

- 更新了欢迎视图文件
- 调试工具栏现在支持深色模式
- 新增了 alpha_numeric_punct 验证规则
- 将 Kint 从 2.x 分支更新到了最新的 3.x 分支, 并新增了配置文件以指定选项
- 新增了更多入门指南
- 修复了 CLI 处理复杂参数的问题
- 改进了 File 类
- 新增了`model()` 辅助方法, 用于轻松实现单例模式的 Model
- 完全重组了测试代码, 以使应用层测试开箱即用更简单。

仓库变更

devstarter 仓库已被废弃。

我们现在在主仓库的 CHANGELOG.md 文件中自动生成了变更日志。详情请见该文件。

版本 4.0.0-rc.4

发布日期:2020 年 2 月 6 日

CodeIgniter4 RC.4 版发布

- 增强功能
- 合并的 *PR*

增强功能

- 修复了 URL 系统, 以便将系统提供为子文件夹时仍可工作。
- 为 sqlite3 和 mysql 添加了所需的插入忽略支持。
- 添加验证函数 *is_not_unique*
- 对 Email 类进行了各种改进和清理

合并的 PR

- #2527 更新 manual.rst
- #2454 官方文档中有关使用 iSAJAX() 进行 ajax 请求的页面修复
- #2525 删除不正确的内联文档类型
- #2524 回归修复命名空间。
- #2523 替换遗留的 CI3 常量。
- #2522 在“从 3.x 升级到 4.x”部分添加 Events 信息
- #2518 修复分页器 URI 以在子文件夹中工作。
- #2516 HTML 辅助函数 - 修复列表的属性类型

- #2515 布局渲染器修复
- #2513 用户指南“实体类 - 业务逻辑”中的拼写错误
- #2511 数据库添加高亮
- #2509 还原渲染器部分重置
- #2507 更新搜索位置的顺序, 以进行更好的优先级排序。
- #2506 HTTP 响应 - 修复当 CSP 被禁用时 CSP 方法崩溃
- #2504 BaseConnection - 在 getConnectStart() 中为返回类型添加 Nullable
- #2502 视图渲染器 - 生成输出后重置部分
- #2501 在 initController 方法上调用控制器的 view_cell。
- #2499 视图解析器 - 使用过滤器修复 ParsePair()
- #2497 修复 splitQueryPart()
- #2496 对 RedirectResponse 使用 site_url。
- #2495 更新工具栏用户指南
- #2494 调试工具栏 - 修复 Debugbar-Time 标头, 在 <head> 中渲染
- #2493 修复 sphinx 版本。
- #2490 修复。工具栏初始化视图错误
- #2489 修复分页器
- #2486 在视图解析器文档中更新 current_url 和 previous_url。
- #2485 用户指南“通过命令行运行”中的拼写错误
- #2482 服务请求添加 URI 核心系统扩展支持
- #2481 优先重定向。
- #2472 ControllerTest 应该在不指定 URI 的情况下工作。修复 #2470
- #2471 从 Zend Escaper 过渡到 Laminas Escaper
- #2462 修复迁移表 id 的不可能长度。
- #2458 将 *composer install* 替换为 *composer require*
- #2450 当 $_SESSION 为 null 时关键 / 传递给 dot_array_search() 的参数 2 必须是 []
- #2449 用户指南: 查询生成器 selectCount - 示例中的错误更正

- #2447 现有文件检查 (Nowackipawel/patch-69)
- #2446 DB 插入忽略 (Tada5hi/database-feature)
- #2438 调试工具栏中的漂亮数组视图
- #2436 修复 Message 方法引用
- #2433 通过模型插入应遵守所有验证规则。修复 #2384
- #2432 在 php 7.4 中修复大括号弃用警告
- #2429 修复。safe_mailto 多字节安全
- #2427 向 ConfigEmail 添加 $recipients 属性
- #2426 添加十六进制验证规则、测试、指南
- #2425 修复:Router setDefaultNameSpace 无法工作
- #2422 在 PHP 服务器下运行时不显示重复的 Date 标头。
- #2420 将 current_url() 更改为使用克隆的 URI
- #2417 修订加密服务文档
- #2416 添加条件‘hasError()’缺失的关闭大括号
- #2415 向 MySQL 字段数据添加‘nullable’
- #2413 修复。工具栏文件 301
- #2411 修复插件的参数解析
- #2408 确保 previous_url() 获取准确的 URI。
- #2407 修复 url 辅助函数以在子文件夹中托管站点时工作。
- #2406 修复问题 #2391 CodeIgniter::display404errors()
- #2402 删除无意义的 isset() 检查
- #2401 从条件语句中删除无意义的检查
- #2400 删除条件语句中的冗余检查
- #2399 修订控制器文档
- #2398 编辑.htaccess
- #2392 添加验证函数 *is_not_unique*
- #2389 为嵌套的种子赋予沉默状态

- #2388 修复复制粘贴的命令注释
- #2387 仅使用数字进行迁移顺序
- #2382 快速修复 postgresql 插入 id
- #2381 修复: 使用 CodeIgniterConfigServices 会阻止服务覆盖
- #2379 替换 null 日志文件扩展名检查
- #2377 文档修订: 替换核心类
- #2369 从 Email 类中删除 LoggerAwareTrait
- #2368 从 Email::__construct 中删除 log_message
- #2364 Email 配置不包含.env 项目
- #2362 修复 SMTP 协议问题
- #2359 Bug 修复 Model after 事件数据
- #2358 修复 Logger 配置
- #2356 修复 Services.php 注释中的拼写错误
- #2352 在日期和时间用户指南中将方法名修复为‘toDateString()’

版本 4.0.0-rc.3

发布日期:2019 年 10 月 19 日

CodeIgniter4 RC.3 版发布

- 增强功能
- 应用变更
- 消息变更
- 变更的文件
- 合并的 *PR*

增强功能

- 加强了数据库、会话和路由处理。
- 修复了许多错误和用户指南勘误。

应用变更

- App/Config/App 中的新 $CSRFHeaderName 属性

消息变更

变更的文件

变更文件的列表如下, 带有 PR 编号:

- 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

合并的 PR

- #2348 CodeIgniter 基金会获得版权
- #2346 修复 FilerHandlerTest.php 奇怪的地方
- #2345 测试自述文件优化
- #2344 设置 vs 建立
- #2343 用户指南小修复。修复类名和代码区域
- #2341 简化 Validation::getErrors()
- #2339 修复当值为 (int) 0 时 Session::get(‘key’) 返回 null
- #2338 还原 RedirectException 更改
- #2337 指南: 小的语法校正
- #2336 正确清理 Windows 中的命名空间
- #2333 指南:RESTful 表格式
- #2332 在实际数据后更改 after 方法
- #2328 更新应用程序结构
- #2327 纠正旅游 UG 页面
- #2325 修复 url_title() 函数在含变音符号时的错误
- #2324 渲染工具栏调试切换
- #2321 更新 RESTful 用户指南
- #2316 将 getValidationRules() 添加到模型 UG 页面
- #2315 增强 Toolbar::renderTimeline
- #2313 RESTful 用户指南清理
- #2312 BaseBuilder 变量类型修复
- #2311 将所有语言返回转换为单引号
- #2308 修复额外的自动路由斜杠错误
- #2307 解决会话保存处理程序问题
- #2305 修复 curl 调试错误

- #2303 如果定义则使用迁移类中的 DBGroup 变量
- #2300 在通过 _remap 计算方法名称时, 工具栏的 Routes 收集器不应该死亡
- #2298 修复 session_regenerate 问题
- #2296 向 Inflector 辅助函数添加 counted()
- #2295 更全面地测试 Builder 类中的 set() 方法
- #2290 修复代码模块文档中关于 psr4 命名空间配置的部分
- #2289 不要以只读方式限制模型对属性的访问
- #2288 修复 Debug/Exceptions 类中的行编号
- #2285 修复 CURLRequest 类中 Host 标头的错误
- #2284 修复验证时 getErrors() 的错误
- #2283 热修复: 将收集器 _*.tpl.php 重命名为 _*.tpl
- #2282 修复 Message 类的用户指南
- #2279 CSRF 参数清理中的错误
- #2272 处理 X-CSRF-TOKEN - CSRF
- #2252 批量更新 Where 重置
- #2123 WIP 修复 store() 默认值错误
- #2100 在为 MySQLi 创建之前验证数据库是否存在

版本 4.0.0-rc.2

发布日期:2019 年 9 月 27 日

CodeIgniter4 RC.2 版发布

- 增强功能
- 应用变更
- 消息变更
- 变更的文件
- 合并的 *PR*

增强功能

- 新的属性简化了查询构建器的可测试性, 但删除了方法参数 (破坏性更改)
- 数据库、迁移和会话得到加强
- 大量更小的错误得到纠正

应用变更

- Config/Constants、Paths 和一些配置设置发生了变化

消息变更

- 无

变更的文件

变更文件的列表如下, 带有 PR 编号:

- 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

合并的 PR

- #2271 修复 ResourcePresenter::setModel()
- #2270 groupStart() 重构
- #2269 BaseBuilder 的 testMode() 方法
- #2268 仅在存在时使用会话验证
- #2267 测试 setUp 和 tearDown:void
- #2265 修复多文件上传的验证问题
- #2264 修复。解析器允许其他扩展
- #2262 在 Debug/Exceptions 中修复参数类型
- #2261 修复 lang() 签名
- #2260 解释 whoops 页面
- #2259 添加 URI 和 url_helper 测试
- #2257 对 HAVING 子句进行了几项更新
- #2253 修复无效参数

- #2246 GIF 不支持 EXIF
- #2245 修复类引用参数类型
- #2241 修复 ini_set 参数类型
- #2240 在 CSRF 中处理 JSON POST
- #2232 修复 BaseBuilder getWhere() 错误
- #2231 为具有 __get 的类添加魔术 __isset
- #2230 为 SQLite _listTables() 添加转义
- #2229 MySQLi escapeLikeStringDirect()
- #2228 从 listTables() 中排除 *sqlite_%*
- #2224 将 new ConfigDatabase() 改为 config(‘Database’)
- #2221 文档修复
- #2218 纠正拼写错误
- #2216 更新 uri.rst
- #2213 在 constrainPrefix 上过滤 listTables 缓存响应
- #2211 添加 listTable() 测试
- #2209 添加 trace()
- #2208 添加 $db->getPrefix()
- #2205 修复 DBPrefix 上的 empty() 错误
- #2201 外键列
- #2200 通知 Kint 别名 dd
- #2199 向用户指南添加 getForeignKeyData
- #2187 更新 Session.php
- #2191 迁移回滚反转
- #2190 修复 ForeignKeyChecks 的名称
- #2189 缺失返回
- #2184 修复“Seeds/”目录的大小写
- #2183 检查常量的 *defined*

- #2181 删除复制粘贴的额外文本
- #2168 修复 CURL 的‘debug’选项

版本 4.0.0-rc.1

发布日期:2019 年 9 月 3 日

CodeIgniter4 RC.1 版发布

- 增强功能
- 应用变更
- 消息变更
- 变更的文件
- 合并的 *PR*

增强功能

- CI3 电子邮件移植到 CI4
- 添加加密 (基本)
- 迁移重构和优化为更全面的功能 (BC)
- 向 ImageHandlerInterface 添加 convert()
- 为下载禁用调试工具栏
- CLI 命令现在返回错误码 (“spark”已更改)
- 添加了 RESTful 控制器, 以缩短 RESTful API 的开发时间
- 作为 RESTful 支持的一部分, 添加了 RouteCollection::presenter()

应用变更

- 添加了 app/Common 以更轻松地重写常用函数
- 添加了 Config/Email 和 Encryption
- 修改了 Config/Migration, 并具有不同的设置
- 修复了 Controllers/Home, 删除了不必要的模型引用

消息变更

- 迁移具有新的和修改后的消息
- 消息现在具有 RESTful 集

变更的文件

变更文件的列表如下, 带有 PR 编号:

- **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

合并的 PR

- #2178 添加缺失 finfo_open 的回退方法
- #2177 修复缺失的表单关闭标签
- #2171 Setheader 重复
- #2169 为图像库添加 $quality 用法
- #2166 Cookie 错误
- #2165 RESTful 帮助
- #2164 CLI 命令失败时退出错误码
- #2162 用户指南针对 Common.php 的更新
- #2160 为 BaseBuilder 添加 SelectCount
- #2155 在启动器中包含.gitignore
- #2153 使用 LIMIT 时修复 countAllResults 的错误
- #2154 修复电子邮件和迁移文档; 更新变更日志
- #2151 ImageMagick->save() 的返回值
- #2150 针对 Image->fit() 的新逻辑
- #2149 listNamespaceFiles: 确保尾随斜杠
- #2145 从 Home 控制器中删除 UserModel 引用
- #2144 更新 Redis 遗留函数
- #2142 修复获取 SQL 时 BaseBuilder 重置
- #2137 新的迁移逻辑
- #2136 迁移用户指南修复
- #2135 加密
- #2134 修复本地化说明
- #2132 更新迁移用户指南
- #2131 向 APIResponseTrait 添加 No Content 响应
- #2129 向 DownloadResponse 添加 setFileName()

- #2128 回退到 clientExtension 进行扩展名猜测
- #2127 更新 limit 函数, 因为 $offset 是可以为空的
- #2126 将 storePreviousURL 限制到某些请求
- #2125 更新 redis 会话处理程序以支持 redis 5.0.x
- #2118 在下载上禁用工具栏
- #2113 添加 Image->convert()
- #2112 更新 *Entity.php* 中的 *__isset* 方法
- #2110 添加了 app/Common.php
- #2109 修复检查 db_connect() 是否存在的拼写错误
- #2092 原始电子邮件移植
- #2090 修复在未设置条件的情况下防止软删除所有
- #2082 更新 BaseConfig.php
- #2065 更新的迁移逻辑以获得更全面的功能
- #2046 清理基本控制器代码
- #2027 修复 CSRF 散列重新生成
- #2023 $value 不必为数组
- #2010 修复 CSRF 哈希再生器词改动
- #2001 BaseBuilder 中的子查询

版本 4.0.0-beta.4

发布日期:2019 年 7 月 25 日

- 亮点
- 新消息
- 应用变更
- 测试变更
- 变更的文件

- *合并的 PR*

亮点

有一些破坏性变更…

- Entity 类已被重构;
- Model 类的变更已更新, 以更好地处理软删除
- 路由已经加强

新消息

- 新的翻译键:Database/noDateFormat

应用变更

测试变更

- 在 tests/_support 中增强了数据库和迁移测试

变更的文件

变更文件的列表如下, 带有 PR 编号:

- 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

合并的 PR

- #2104 文件和上传文件修复
- #2091 时区选择
- #2081 改进 JSON 格式检查
- #2079 更新 config() 以检查所有命名空间
- #2074 Throttler 可以访问桶的生命周期
- #2071 修复 autoloader.rst 格式
- #2069 验证规则:then -> than(拼写)
- #2064 修复文件定位器斜杠错误的 Bug
- #2063 确保查询变量是 request->uri 的一部分。修复 #2062
- #2060 缓存驱动备份
- #2059 为 *locateFile()* 添加多路径支持
- #2054 为缺失/无效的 dateFormat 添加模型异常
- #2053 将 Model 的 deleted 标志更改为 deleted_at 日期时间/时间戳。修复 #2041
- #2052 为 (not) null 添加各种测试
- #2051 软删除使用 deleted_at
- #2050 在触发事件之前保存插入 ID
- #2043 路由时应传入零参数。修复 #2032
- #2042 SQLite3 现在支持删除外键。修复 #1982
- #2040 更新 CURLRequest.php

- #2039 限制 URI 的过滤器匹配, 使其需要精确匹配。修复 #2038
- #2036 使 *force_https()* 在退出之前发送标头
- #2035 各种拼写错误和指南更正
- #2031 回退到服务器请求以获取默认方法
- #2030 在调试工具栏中支持新的 *router* 服务
- #2026 扩展 Pager::makeLinks(可选组名)
- #2024 重构 router 和 route collection 确定当前 HTTP 动词的方式
- #2019 SQLite 和 Mysql 驱动的附加测试以及迁移运行程序测试修复
- #2015 安装后直接用户遵循升级步骤
- #2014 添加了一个新的 Session/ArrayHandler, 可在测试期间使用
- #2012 对 HTTP 动词使用 request->method
- #2011 为实体设置没有任何变异的原始数据数组
- #2008 为命令“routes”添加 *patch* 方法
- #2005 插件闭包文档更新和测试
- #2004 允许不带参数调用 hasChanged()
- #2002 实体重构
- #1999 使用 CodeIgniterController; 不需要因为 Home 控制器扩展…
- #1996 尝试修复 CURLRequest 调试问题。#1994
- #e21823 修正了解析器插件的文档。关闭 #1995
- #1989 参数 set() 必须是字符串类型 - 无法同意
- #1959 防止 reverseRoute 搜索闭包

版本 4.0.0-beta.3

发布日期:2019 年 5 月 7 日

- 亮点
- 新消息

- 应用变更
- 变更的文件
- 合并的 *PR*

亮点

- 在整个项目中添加了类型提示并纠正了拼写错误 (参见 API 文档)
- 修复了许多 model、database、validation 和 debug toolbar 问题

新消息

- Database.FieldNotExists
- Validation.equals、not_equals

应用变更

- 在 app/Config/App 中删除了 $salt 配置项
- 在 app/Config/Migrations 中默认启用了迁移
- 简化了 public/.htaccess

变更的文件

变更文件的列表如下, 带有 PR 编号:

- **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

合并的 PR

- #1987 纠正 API 文档块中的问题以生成 phpdocs
- #1986 将文档块版本更新为 4.0.0
- #1985 修复过滤器处理。修复 #1907
- #cbe4b1d 修复 SQLite 测试
- #1984 为 HTML 表添加页脚

- #1981 使用软删除不应在连接表时返回模糊字段消息
- #1980 修正了 Session/RedisHandler::read 的返回值
- #1978 为 CI4 实现 HTML Table(遗失的功能)
- #1977 Test/featuretestcase
- #1974 从自动加载器的 classmap 中删除框架类
- #1973 默认值修复
- #1972 针对自定义收集器的工具栏修复
- #1970 再次添加过滤器参数
- #1968 修复 pathinfo 模式下的 404 错误
- #1963 在数据库更新时, 字符串类型的主键也应包装成数组
- #1962 修复边缘问题
- #1961 修复 Debugbar url 尾部斜杠问题
- #1957 新的通用字符串验证规则
- #1956 使用空合并运算符
- #1955 修复 Travis-CI 构建失败
- #1954 修复验证表格式
- #1952 添加 *equals()* 和 *not_equals()* 的验证
- #1951 系统拼写更改和代码清理
- #1950 修复一些边缘问题
- #1949 Toobar/Routes 更正
- #1947 修复 BaseConfig 没有正确加载 Registrar 文件
- #1945 从 debugbar 文件中提取日期时间
- #1943 Model、Entity、Exception 和 Migration 测试用例
- #1939 移除阻止盗链的部分
- #1938 数据库拼写更改
- #1936 文档: 改进应用测试说明
- #1935 更新 phpunit.xml 脚本。修复 #1932

- #1933 having (删除 Is NULL)
- #1931 Toolbar IE11 修复
- #1930 根据 #1773 对 Model 进行更改
- #1927 针对不存在属性抛出 Entity 异常
- #1926 文档: 更新安装指南
- #1925 删除了 $_SERVER[‘CI_ENVIRONMENT’]
- #1923 缺失返回
- #1918 JSONFormatter
- #1917 数据库测试用例
- #1916 检查值是否为字符串
- #1915 修复 POST + JSON(添加 Content-Length)
- #1911 JSON 强制转换异常测试用例

版本 4.0.0-beta.2

发布日期:2019 年 4 月 4 日

- 亮点
- 新消息
- 应用变更
- 变更的文件
- 合并的 *PR*

亮点

- 大量修复和改进, 尤其是针对 Model 和测试类
- Model 现在需要主键
- 生成的 API 文档可在 https://codeigniter4.github.io/api/ 访问
- 验证规则得到增强

- .htaccess 加强

新消息

- Database.noPrimaryKey、forFindColumnHaveMultipleColumns、Database.forEmptyInputGiven

应用变更

- 更新了 app/Config/Events
- 添加了 app/Controllers/BaseController
- 添加了测试文件夹用于单元测试
- 添加了 phpunit.xml.dist 用于单元测试配置

变更的文件

变更文件的列表如下, 带有 PR 编号:

- .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

合并的 PR

- #1913 更多覆盖重写的 RouteCollection 测试。关闭 #1692
- #1912 额外的 RouteCollectionTests
- #1910 为 CLI 库添加了 print 方法, 以便在同一行上多次打印
- #1908 在用户指南中添加过滤器参数
- #1906 与 #1775 相关的子查询测试用例
- #1902 BaseBuilder 修正
- #1900 为更好的安全性和缓存更新.htaccess
- #1899 数据库 Forge 修正
- #1897 针对 #1779 的工具栏修复
- #1896 Mysql 连接使用 SSL 证书的问题 (#1219)
- #1894 修正拼写错误
- #1893 用 remove escapeshellarg() 修复 spark serve
- #1892 在用户指南中添加 CLI 背景颜色列表
- #1891 允许 CLI::strlen 为空参数
- #1886 修复问题 #1880, 修复了一些拼写错误和更新了代码风格
- #1882 与 #1541 相关的路由器更改
- #1873-1889 文档: 移动命名空间声明和添加缺失的类 docblocks

- #1872 文档: 修复 phpdoc 配置
- #1871 不匹配的缓存库 *get()* 返回 null
- #1869 History::SetFiles 检查 #1778
- #1863 按模块进行拼写更改
- #1861 新的 Find Column 方法与 #1619 相关
- #1860 Migrationrunner 使用自动加载程序
- #1867 事件现在应该可以与任何可调用项一起使用了。修复 #1835
- #1865 解决定义中的 MigrationRunner 问题
- #1862 required_with 和 required_without 定义更改
- #1859 在路由列表中忽略回调
- #1858 DB 模块中的拼写更正
- #1856 在 pre_system 事件上确保 ob_get_level() > 0 时 ob_end_flush()
- #1855 修复:ControllerTester::execute。修复 #1834
- #1854 File::move 现在会为重定位的文件返回新的文件实例。修复 #1782
- #1851 用根 CI4 版本替换旧的 CI3 .gitignore
- #1850 安全可路由的控制器方法
- #1848 测试: 修复和测试 Test/ControllerTest, 已测试
- #1847 默认将 Controller 扩展到 BaseController
- #1846 修复贡献链接
- #1844 Model 修复
- #1843 替换 CI3 $this->input 引用
- #1842 异常‘forPageNotFound’缺少默认值
- #1839 不要在 to 路由中将斜杠替换为反斜杠
- #1836 测试: 改进 ViewView 覆盖率
- #1831 修复一些 PHPDoc 注释错误
- #1829 改进:Model 现在需要主键。这部分是为了保持代码…
- #1828 修复: 从验证视图中删除引导样式。

- #1827 修复: 向视图库添加 include 方法来渲染视图片段…视图。
- #1823 文档: 在 Pagination 类中删除遗留的 Bootstrap 引用
- #1822 测试: 增强 RouteCollection 覆盖率
- #1820 修复: 在 model.rst 中正确的 sphinx 错误
- #1819 改进: 使用 phpDocumentor 添加 apibot 用于 API 文档
- #1818 改进: 在 exact_length 规则中改进代码
- #1817 改进: 引入 Model setValidationMessage 函数
- #895ae0 修复: 每当使用旧命令时都启动会话
- #1814 增强:extended exact_length[1,3,5]
- #1813 修复:Model::save 对于早期 PR 的修复
- #1812 测试: 改进 Filters 覆盖率
- #1811 测试: 改进 Config 模块覆盖率
- #1809 修复文件移动失败。修复 #1785
- #1808 修复: 修复 save 方法的返回值
- #1805 文档:Query 类更改
- #1804 文档: 一些基本功能更改
- #1803 文档: 一些辅助函数更改
- #1802 文档: 通用函数更正
- #1796 测试: 改进 Cache 覆盖率
- #1794 替换不存在的“getAffectedRows”
- #1793 设置 Model->chunk 返回类型
- #1791 修复: 在 ValidationRules 中删除 is_numeric 测试
- #d08b68 在缺少 UserAgent 的 ControllerTester 中修复
- #1790 根据问题 #1781 中提到的修正文档中的拼写错误
- #1777 向示例添加 CLI 命名空间
- #1776 修复: 仅替换字段名称中的最后一个操作符
- #1771 修复:matchSimpleBinds 中的拼写错误

- #1769 方法和拼写更正
- #1762 修复:decimal 规则。它不应该接受整数吗?
- #1746 改进: 更新 Model, 选择性更新 created_at/updated_at 字段。
- #1590 改进: 增强 404Override

版本 4.0.0-beta.1

发布日期:2019 年 3 月 1 日

- 亮点
- 新消息
- 应用变更
- 变更的文件
- 合并的 *PR*

亮点

- 新的视图布局提供了创建站点视图模板的简单方法。
- 修复了用户指南的 CSS, 以适当显示宽表格
- 将 UploadedFile 转换为使用系统消息
- 修复了大量数据库、迁移和模型错误
- 为应用启动器和框架分发重构了单元测试

新消息

- Database.tableNotFound
- HTTP.uploadErr…

应用变更

- app/Config/Cache 有新的设置:database
- app/Views/welcome_message 的徽标已着色
- composer.json 有一个大小写更正
- env 添加了 CI_ENVIRONMENT 建议

变更的文件

变更文件的列表如下, 带有 PR 编号:

- **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

合并的 PR

- #1774 beta.1 的杂务
- #1768 辅助函数更改 - 签名和拼写错误
- #1764 修复未指定默认路由时的路由问题。解决 #1758
- #1763 确保验证在带有规则错误的 Model 中工作。解决 #1574
- #1757 纠正不必要的双引号 (拼写错误)
- #1755 在 composer 文件中小写‘vfsStream’
- #1752 修复阻止链接格式的拼写错误
- #1751 指南: 将误放文本移到正确的标题下
- #1750 从用户指南中删除加密密钥引用
- #1749 在.env 中添加环境
- #1745 为 SQLite3 支持更新了复合键测试。解决 #1478
- #1744 根据当前框架状态更新实体文档。修复 #1727
- #1743 手动排序找到的迁移, 而不依赖操作系统。解决 #1666
- #1742 修复 required_without 规则错误。
- #1741 现在可以加载具有特定命名空间的辅助函数了。解决 #1726
- #1740 重构应用启动器的测试支持
- #1739 修复拼写错误
- #1738 修复 required_with 规则错误。解决 #1728
- #1737 为 SQLite 驱动添加了对 dropTable 和 modifyTable 的支持
- #1736 适应 travis 执行时间过长

- #1733 修复 Postgres 的自增和自减错误
- #1732 不要从 CLI 检查路由。解决 #1724
- #1729 新的视图布局功能用于简单模板
- #1725 更新 Request.php
- #1723 如果认证失败, 记录 redis 错误
- #1722 Seeder 为种子添加默认命名空间
- #1719 更新缓存 RedisHandler 以支持选择数据库
- #4ff1f5 插入和必填验证失败的附加测试 (#1717)
- #549d7d 关于在模型内外正确转义的另一次尝试
- #1712 可读性细微更改
- #37dbc1 确保 Model 验证规则可以是组名
- #1709 修复资源路由 websafe 方法顺序检查
- #1708 UploadedFile 的语言
- #36fbb8 BaseBuilder 只应在运行查询时关闭 Connection 的设置转义标志…
- #6e549a 提供与开发服务器一起使用的默认 baseURL, 以便初次设置更容易 (修复 #1646)
- #1704 修复 viewsDirectory 错误 (#1701)
- #4f4a37 从 Model 中删除调试。
- #1699 修复用户指南中的安装链接
- #1696 修复页面结构等
- #1695 整理用户指南中的代码块

版本 4.0.0-alpha.5

发布日期:2019 年 1 月 30 日

CodeIgniter4 的下一个内测版本

- 亮点
- 变更的文件
- 合并的 *PR*

亮点

- 在 app/Config/Toolbar.php 中添加了 $maxQueries 设置
- 将 PHP 依赖更新到 7.2
- 为电子邮件和队列模块创建了新的功能分支, 以便它们不影响 4.0.0 的发布
- 删除了几条未使用的语言消息 (如 Migrations.missingTable), 并添加了一些新消息 (如 Migrations.invalidType)
- 修复了大量 Bug, 尤其是数据库支持
- 提供的过滤器 (CSRF、Honeypot、DebugToolbar) 已从 app/Filters/ 移动到 system/Filters/
- 重新审视了用户指南的安装和教程部分
- 代码覆盖率达到 77% …我们的目标 80% 越来越近了:)

我们希望这将是最后一个内测版本, 下一个预发布版本将是我们的第一个测试版…祝我们好运!

变更的文件

变更文件的列表如下, 带有 PR 编号:

- **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

合并的 PR

- #1693 文档/教程
- #5951c3 允许域/子域重写现有路由
- #1691 更新运行文档
- #1690 重写安装文档
- #bea1dd AliasTests 的补充, 用于潜在的 LeftJoin 问题
- #1689 Model 验证修复
- #1687 为过滤器添加版权声明
- #1686 重构/过滤器
- #1685 修复管理员 - 应用启动器创建
- #1684 为 filehandler 更新会话 id 清理
- #1683 修复 migrate:refresh 错误
- #d2b377 修复 Postgres replace 命令以适应新的绑定存储方式
- #4a1886 文档方法欺骗
- #2e698a 也 urldecode URI 键和值。
- #1679 save_path - 用于 memcached
- #1678 修复路由未替换正斜杠
- #1677 为数据库引擎实现不转义功能
- #1675 添加缺失的测试组指令
- #1674 更新变更日志
- #1673 更新下载和安装文档
- #1672 更新 Autoloader.php
- #1670 将 PHP 依赖更新到 7.2
- #1671 更新文档
- #1669 增强 Parser 和插件测试

- #1665 Composer PSR4 命名空间现在是模块自动发现的一部分
- #6dab8f 过滤器不区分大小写匹配
- #1663 修复 whereIn 使用时的绑定问题
- #1660 迁移测试和数据库调整
- #1656 __get() 中的 DBGroup, 允许在模型外验证“数据库”数据
- #1654 工具栏 - 返回 Logger::$logCache 项
- #1649 在 travis 配置中将 php 7.3 从“allow_failures”中删除
- #1648 更新“管理应用”文档
- #1645 修复启用事务时令人困惑的地方 (文档)
- #1643 移除电子邮件模块
- #1642 CSP nonce 属性值为“”
- #81d371 自动加载和迁移期间对配置文件进行安全检查
- #1641 更多单元测试调整
- #1640 在 BaseBuilder 中更新 getCompiledX 方法
- #1637 修复启动器自述文件
- #1636 重构 Files 模块
- #5f305a UG - 管理应用中的拼写错误
- #1635 单元测试增强
- #1633 使用 csrf_field 和 form_hidden
- #1632 应该将 DBGroup 传递给 ->run 而不是 ->setRules
- #1631 在 UploadedFile 类中许可证文档之后移动 use 语句
- #1630 版权更新到 2019
- #1629 将“application”目录文档和注释改为“app”
- #3a4ade view() 现在可以再次适当读取应用配置
- #7993a7 使 translateURIDashes 正常工作的最终部分
- #9e435c 修复 translateURIDashes
- #1626 清理 Paths::$viewDirectory 属性

- #1625 匹配后不是设置为空
- #1623 如果定义为可空, 则不转换属性
- #1622 __set 的可空支持
- #1617 countAllResults() 应该遵守软删除
- #1616 修复 View 配置合并顺序
- #614216 将 honeypot 服务从应用 Services 文件移到它所属的系统 Services
- #6b8b8b 允许 db forge 和工具接受连接信息数组而不是组名
- #1613 文档中的拼写错误
- #1538 img 修复 (?)- html_helper

版本 4.0.0-alpha.4

发布日期:2018 年 12 月 15 日

CodeIgniter4 的下一个内测版本

- *亮点*
- *变更的文件*
- *合并的 PR*

亮点

- **重构以保持一致:application 文件夹重命名为 app;**
  BASEPATH 常量重命名为 SYSTEMPATH
- 调试工具栏获得自己的配置、历史收集器
- 大量纠正和增强

变更的文件

变更文件的列表如下, 带有 PR 编号:

- **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

合并的 PR

- #1610 测试、修复和增强语言类
- #a0fc68 在插入、更新和查询后清除绑定
- #1608 在用户指南中注明环境配置
- #1606 发布框架脚本清理
- #1603 充实 I18n 测试
- #8f305a 捕获 mysql 连接错误并消毒用户名和密码
- #1602 Model 的 first 和 update 在没有主键的表中不工作

- #1601 在 Common.php 中清理 ConfigServices
- #1600 清理 admin/starter/composer.json
- #1598 将数据库会话的默认 DBGroup 设置为 $defaultGroup
- #1595 通过 pre_system 处理致命错误
- #1594 修复工具栏无效的 css
- #1593 充实 Test 包测试
- #1589 修复工具栏文件加载抛出异常
- #1588 修复 site_url 生成无效 url
- #1587 添加语言回退
- #1586 修复教程中的模型命名空间
- #1585 为 MigrationRunner 方法添加类型提示
- #4c7bfe Entity 的 fill() 现在尊重映射的属性
- #547792 为 Entity 类添加 _get 和 _set 说明
- #1582 修复变更日志索引和通用函数的 UG 缩进
- #1581 ContentSecurityPolicy 测试和增强
- #1579 使用绝对路径
- #1576 Testing13/http
- #1575 添加?integer、?double、?string 等转换类型
- #ab8b5b 在测试中默认将 baseURL 设置为 example.com
- #d7dfc5 关于重定向的文档调整
- #1573 吸取的教训
- #1571 工具栏更新
- #1569 用不同编码测试 esc(), 忽略仅应用的辅助函数
- #1563 为 csrf_field 添加 id 属性支持
- #1562 集成 Autoloader 和 FileLocator
- #1561 更新 Connection.php
- #1557 移除 use 语句中的前缀

- #1556 在测试中为 setUp() 函数使用 protected 修饰符而不是 public
- #1555 自动加载清理: 从 composer.json 中删除 PsrLog 命名空间
- #1551 在 ComposerScripts 中删除手动定义的 “system/” 目录前缀
- #1548 允许设置空 html 属性
- #1459 添加 Vagrantfile

版本 4.0.0-alpha.3

发布日期:2018 年 11 月 30 日

CodeIgniter4 的下一个内测版本

- *变更的文件*
- *合并的 PR*

变更的文件

变更文件的列表如下, 带有对应的 PR 编号:

- **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 已移除 #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
    - bootstrap #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

合并的 PR

- #1554 Service 实例
- #1553 Admin/脚本
- #1550 在 FileLocator 中删除注释的 CLI::newLine($tempFiles)
- #1549 在 Database/Seeds 目录中使用.gitkeep 代替.gitignore
- #1547 将文件存在更改为是文件
- #1546 ImageMagickHandler::__construct…
- #1540 更新验证类用户指南
- #1530 数据库性能改进: 尽可能使用 foreach()
- 2047b5a 使用 spark 时不运行过滤器。
- #1539 在 CreditCardRules 中删除 mb_* (mb 字符串使用)
- #1536 composer.json 中的 ext-json
- #1525 删除不需要的 try {} catch {}
- #1524 用‘websafe’选项测试路由资源
- #1523 检查匹配的路由正则是否被过滤
- #1522 在 BaseBuilder 中添加 property_exists 检查
- #1521 .gitignore 清理
- #1518 小错误修正: 将 setCreatedOn 改为 setCreatedAt
- #1517 将每个目录中的.htaccess 从 writable/{directory} 移动到 writable/
- #1513 更安全的重定向
- #1509 删除未使用的 use 语句
- #1508 在 URI::setScheme() 调用中删除重复的 strtolower() 调用
- #1507 修复用“,”分隔的多个“empty”字符串被标记为有效电子邮件
- #1506 充实 HTTP/File 单元测试
- #1505 直到所有 Response 完成之前不退出
- 27868b 为 {field} 和 {param} 占位符添加缺失的文档

- #1504 还原 RedirectResponse 更改
- #1500 忽略用 @ 禁止的错误
- #1499 修复 form_helper 的 set_value 说明
- #1498 为 CURLRequest 添加辅助方法
- #1497 移除未使用的 RedirectException
- #1496 修复 Common::old()
- #1495 添加 URI 段测试
- #1494 用户指南中的方法命名
- #1491 错误日志记录
- #1490 变更日志重构
- #1489 为 CLI 添加::strlen()
- #1488 从其他位置加载语言字符串
- #1486 测试 RedirectResponse 问题报告
- #1484 缺少斜杠
- #1483 SessionHandlersBaseHandler.php 中的小拼写错误
- #1482 修复 Seeds 文档中的查询绑定问题
- #1481 RedisHandler 测试清理
- #1480 修复语言关键文件混淆
- #1479 修复另一个时间测试
- #1477 添加 Response 发送测试
- #1475 纠正 Forge::addField() 的 phpdocs
- #1473 将另一个时间测试模糊化
- #1472 HTTPResponse cookie 测试和缺失功能
- #1471 在 XMLFormatter::format() 中删除未使用的局部变量 $result
- #1470 允许使用数组字段约束创建表
- #1469 对保护的/公共函数使用 static:: 而不是 self::
- #1468 修复 FeatureTestCase 测试输出缓冲

- #1467 提供容差时间测试
- #1466 修正 BaseBuilder 的 phpdocs
- #1465 对保护和公共属性使用 static:: 而不是 self::
- #1464 删除未使用的 use 语句
- #1463 修复其余的 bcit-ci 引用
- #1461 拼写错误修正:donload -> download
- #1460 在 HoneyPot 中删除不需要的三元检查
- #1457 在 public/index.php 中使用 $paths->systemDirectory
- #1456 加强 HTTP URI 和 Response 测试
- #1455 取消忽略 app/Database/Migrations 目录
- #1454 在 Email::getEncoding() 的循环中添加缺失的 break;
- #1453 BugFix 如果扩展只有一个 mime 类型
- #1451 在 RedirectResponse 中删除不需要的 $session->start() 检查
- #1450 phpcbf: 一次修复所有
- #1449 简化从 mysql/mariadb 获取 indexData 的方式
- #1448 文档: 添加缺失的应用结构
- #1447 在循环卡片以获取卡信息的 CreditCardRules 中添加缺失的 break;
- #1445 在 HTTPIncomingRequest 中使用现有的 is_cli() 函数
- #1444 关于重组库管理的文档 (4/4)
- #1443 修复未捕获的单元测试输出
- #1442 从 app/View/ 中移除表单视图以及在创建新项目教程中使用表单辅助函数
- #1440 访问模型最后插入的 ID
- #1438 尾部库组织名称 (3/4)
- #1437 在大多数 php 文档中替换库组织名称 (2/4)
- #1436 在文档中更改 github 组织名称 (1/4)
- #1432 使用 mb_strlen 获取列长度
- #1431 从命令迁移中无法调用 run() 方法并传参

- #1427 修复 FeatureTestCase 中的“选项”请求调用参数
- #1416 DatabaseBaseResult 中的性能改进
- #1425 确保 FileHandlerTest 使用 MockFileHandler
- #1424 修复 FileMovingTest 遗留问题
- #1423 修复 Controller 验证使用错误
- #1422 修复 Migrations.classNotFound
- #1418 规范化 composer.json
- #1417 修复 Parser::parsePairs 总是转义
- #1416 在 applicationConfigAutoload 中删除 $psr4[‘TestsSupport’] 定义
- #1415 删除不需要的“defined(‘BASEPATH’)…”
- #1413 在所有 uniqid() 用法中设置 more_entropy = true
- #1412 修复 array_helper 中的 function_exists() 拼写错误
- #1411 在 View::render() 的循环中添加缺失的 break;
- #1410 修复 2d0b325 提交导致 spark serve 不工作
- #1407 在 BaseConnection->prepare() 中添加缺失的 initialize() 调用检查
- #1406 为 RouteCollectionInterface 添加缺失的参数
- #1405 修复 DownloadException 中使用的语言字符串
- #1402 纠正用户指南中的类命名空间
- #1399 允许在 guessExtension 中使用类型提示
- #1398 调整节流测试
- #1397 纠正测试中 _support/_bootstrap.php 的 FCPATH 设置
- #1396 仅在“testing”环境中为“TestsSupport”命名空间注册 PSR4
- #1395 文档中使用短数组语法
- #1394 将 php 7.3 添加到 travis 配置
- #1390 修复测试执行时输出“Hello”
- #1389 使电子邮件文件名大写
- #1388 提交时自动 phpcs 修复

- #1387 重定向到命名路由
- #1385 修复迁移页面; 更新变更日志
- #1384 为 ResponseInterface 添加缺失常量
- #1383 修复 MemcachedHandler::__construct() 中的 TypeError
- #1381 删除未使用的 use 语句
- #1380 改进 count(), 使用真值检查
- #1378 更新 Filters 语言文件
- #1377 修复 monolog 会导致错误
- #1376 修复无法在 AppFiltersHoneypot 中使用 Honeypot 类, 因为已经在使用
- #1375 根据 RFC 6266 给下载一个标头
- #1374 缺失的功能迁移。
- #1373 关闭数据库插入/保存的类型转换
- #1371 更新编码样式中的方法名称
- #1370 工具栏需要日志记录。修复 #1258
- #1369 移除不可见字符
- #1368 UploadedFile->guessExtenstion()…
- #1360 删除–cached php_errors.log 文件
- #1357 更新模板文件与.php 不兼容
- #1345 is_unique 尝试连接默认数据库而不是在 DBGroup 中定义的数据库
- #1344 不对不必要的表选项加引号
- #1343 避免在约束中添加两个单引号
- #1335 审查和改进 MySQLi、Postgre 和 SQLite 数据库驱动
- #1213 在分页中使用 URI 段作为页码
- #1180 在 HTTPRequest 实例中获取 IP 地址
- #622 为分页添加头部链接

版本 4.0.0-alpha.2

发布日期:2018 年 10 月 26 日

CodeIgniter4 的第二个内测版本

- 变更的文件
- 合并的 *PR*

变更的文件

变更文件的列表如下, 带有 PR 编号:

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

合并的 PR

- #1361 给 CIUnitTestCase 增加定时断言
- #1312 给 CIUnitTestCase 增加 headerEmitted 断言
- #1356 Testing/commands
- #1355 适当处理重复的 HTTP 动词和通用规则
- #1350 检查类是否可实例化并且是一个命令
- #1348 修复 sessions 中的 sphinx 格式问题
- #1347 修复 sessions 中的 sphinx 格式问题
- #1342 工具栏样式
- #1341 在 Paths.php 中使 viewpath 可配置。解决 #1296
- #1340 更新下载文档以反映需要返回它。解决 #1331
- #1339 修复 Forge 类可能未返回的错误。解决 #1225
- #1337 路由器中的过滤器解决 #1315
- #1336 还原 alpha.2
- #1334 为 alpha.2 提议的变更日志
- #1333 用户指南中 session 配置的错误。解决 #1330
- #1329 调整
- #1327 修复 form_hidden 和 form_open - 如同 form_input 中那样转义值。
- #1323 修复文档错误:show_404() 不再存在
- #1321 添加缺失的 xml_helper 用户指南页面

- #1319 Testing/entity
- #1316 重构 TimeTest
- #1314 修复与扩展 Honeypot 及其测试
- #1313 清理异常
- #1311 实体存储一组原始值以与之比较以便我们进行确定。。。
- #1306 Testing3/http
- #1305 将 chdir(‘public’) 改为 chdir($public)
- #1304 在 parseRequestURI() 中重构脚本名称剥离
- #1303 Testing/http
- #1302 异常: 没有为 mime 类型 “ 定义 Formatter
- #1300 允许使用当前请求的查询变量重定向。
- #1295 修正前端控制器注释中的语法。
- #1294 更新最后一个教程页面。解决 #1292
- #1291 允许扩展辅助函数。解决 #1264
- #1286 Cookies
- #1285 确保在任何 * 匹配规则之前匹配当前 HTTP 动词路由…
- #1283 实体
- #1282 system/Test/FeatureTestCase::setupRequest(), 略微修正 phpdoc 块…
- #1281 教程
- #1280 在用户指南中添加参与指引
- #1273 修复/计时
- #1272 修复 cli 404 中未定义的变量“heading”
- #1271 移除在 AutoloadConfig::classmap 中不存在的“CodeIgniterLoader”
- #1269 发布说明与流程
- #1266 调整发布构建脚本
- #1265 WIP 修复文档中关于 PHP 服务器的部分
- #1245 修复 #1244(form_hidden 声明)

- #1239 [不请自来的 PR] 我将下载方法改成可测试的。
- #1217 在 Builder 的 countAll() 调用中为 resetSelect() 调用添加可选参数;
- #1209 修复 DatabaseBaseUtils 中未定义的函数 xml_convert

版本 4.0.0-alpha.1

发布日期:2018 年 9 月 28 日

CodeIgniter 框架的重写

- 新软件包列表

新软件包列表

- **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

2.1.7 从前一版本升级

请阅读你需要升级的版本所对应的升级指南。

另请参阅后向兼容性说明。

备注: 如果你不知道正在运行的 CodeIgniter 版本, 可以从调试工具栏 获取, 或者简单地输出常量 `\CodeIgniter\CodeIgniter::CI_VERSION`。

后向兼容性说明

我们尽力开发具有尽可能好的后向兼容性 (BC) 的产品。

只有主版本发布 (如 4.0、5.0 等) 允许破坏后向兼容性。次版本发布 (如 4.2、4.3 等) 可以引入新特性, 但必须在不破坏现有 API 的情况下实现。

然而, 代码还不够成熟, bug 修复可能会在次版本发布甚至补丁版本发布 (如 4.2.5) 中破坏兼容性。在这种情况下, 所有破坏兼容性的更改都会在变更记录 中描述。

什么不是破坏兼容性的更改

- 已弃用的项目不受后向兼容性 (BC) 承诺约束。它可能会在下一个 **次要**版本或更高版本中被移除。例如，如果一个项目从 4.3.x 版本开始被弃用，那么它可能会在 4.5.0 版本中被移除。
- 定义在 **system/Language/en/** 中的系统消息严格用于内部框架使用, 不受后向兼容性 (BC) 承诺约束。如果开发者依赖语言字符串输出, 应该检查函数调用 (`lang('...')`), 而不是内容。
- 命名参数 不受后向兼容性 (BC) 承诺约束。当必要时, 我们可能会重命名方法/函数的参数名以改进代码库。

从 4.5.8 升级到 4.6.0

请根据你的安装方式参考对应的升级指南:

- *Composer 安装 App Starter 升级指南*
- *Composer 安装将 CodeIgniter4 添加到现有项目升级指南*
- *手动安装升级指南*

- 重大变更
  - 异常类变更
  - *Time::createFromTimestamp()* 时区变更
  - *Time* 保留微秒
  - *Time::setTimestamp()* 行为修正
  - 注册器的脏数据修复
  - *Session ID (SID)* 变更
  - 接口变更
  - 方法签名变更
  - 移除已弃用项

- 重大增强
  - 过滤器变更
- 项目文件
  - 内容变更
  - 所有变更

重大变更

异常类变更

部分类抛出的异常类已变更，部分异常类的父类已调整。详见更新日志。

如果你的代码捕获了这些异常，请修改对应的异常类。

Time::createFromTimestamp() 时区变更

当未显式传递时区参数时，*Time::createFromTimestamp()* 现在返回 **UTC** 时区的 Time 实例。在 v4.4.6 至 v4.6.0 之前的版本中，返回的是当前设置的默认时区的 Time 实例。

此行为变更是为了与 PHP 8.4 新增的 `DateTimeInterface::createFromTimestamp()` 方法行为保持一致。

如需保持默认时区，需显式传递时区作为第二个参数:

```
use CodeIgniter\I18n\Time;

$time = Time::createFromTimestamp(1501821586, date_default_timezone_
↪get());
```

Time 保留微秒

在先前版本中，*Time* 在某些情况下会丢失微秒。这些问题已修复。

由于修复，`Time` 的比较结果可能发生变化:

```
use CodeIgniter\I18n\Time;

$time1 = new Time('2024-01-01 12:00:00.654321');
$time2 = new Time('2024-01-01 12:00:00');

$time1->equals($time2);
// Before: true
//  After: false
```

在此情况下，你需要手动移除微秒:

```
use CodeIgniter\I18n\Time;

$time1 = new Time('2024-01-01 12:00:00.654321');
$time2 = new Time('2024-01-01 12:00:00');

// Removes the microseconds.
$time1 = Time::createFromFormat(
    'Y-m-d H:i:s',
    $time1->format('Y-m-d H:i:s'),
    $time1->getTimezone(),
);

$time1->equals($time2);
// Before: true
//  After: true
```

以下情况现在会保留微秒:

```
use CodeIgniter\I18n\Time;

$time = Time::createFromFormat('Y-m-d H:i:s.u', '2024-07-09␣
↪09:13:34.654321');
echo $time->format('Y-m-d H:i:s.u');
// Before: 2024-07-09 09:13:34.000000
//  After: 2024-07-09 09:13:34.654321
```

```
use CodeIgniter\I18n\Time;
```

(续下页)

(接上页)

```
$time = new Time('1 hour ago');
echo $time->format('Y-m-d H:i:s.u');
// Before: 2024-07-26 21:05:57.000000
//  After: 2024-07-26 21:05:57.857235
```

注意：表示当前时间的 `Time` 实例在此前版本中已保留微秒。

```
use CodeIgniter\I18n\Time;

$time = Time::now();
echo $time->format('Y-m-d H:i:s.u');
// Before: 2024-07-26 21:39:32.249072
//  After: 2024-07-26 21:39:32.249072
```

返回 `int` 类型的方法仍会丢失微秒:

```
use CodeIgniter\I18n\Time;

$time1 = new Time('2024-01-01 12:00:00');
echo $time1->getTimestamp(); // 1704110400

$time2 = new Time('2024-01-01 12:00:00.654321');
echo $time2->getTimestamp(); // 1704110400
```

Time::setTimestamp() 行为修正

在先前版本中，对非默认时区的 Time 实例调用 `Time::setTimestamp()` 可能返回错误日期/时间的实例。

此问题已修复，现在行为与 `DateTimeImmutable` 一致:

```
use CodeIgniter\I18n\Time;

// The Application Timezone is "UTC".

// Set $time1 timezone to "America/Chicago".
$time1 = Time::parse('2024-08-20', 'America/Chicago');
```

(续下页)

(接上页)

```
// The timestamp is "2024-08-20 00:00" in "UTC".
$stamp = strtotime('2024-08-20'); // 1724112000

// But $time2 timezone is "America/Chicago".
$time2 = $time1->setTimestamp($stamp);

echo $time2->format('Y-m-d H:i:s P');
// Before: 2024-08-20 00:00:00 -05:00
//  After: 2024-08-19 19:00:00 -05:00
```

注意：使用默认时区时行为未改变:

```
use CodeIgniter\I18n\Time;

// The Application Timezone is "America/Chicago".

// $time1 timezone is "America/Chicago".
$time1 = Time::parse('2024-08-20');

// The timestamp is "2024-08-20 00:00" in "America/Chicago".
$stamp = strtotime('2024-08-20'); // 1724130000

// $time2 timezone is also "America/Chicago".
$time2 = $time1->setTimestamp($stamp);

echo $time2->format('Y-m-d H:i:s P');
// Before: 2024-08-20 00:00:00 -05:00
//  After: 2024-08-20 00:00:00 -05:00
```

注册器的脏数据修复

为防止注册器 的自动发现机制重复执行，当 Registrar 类被加载或实例化时，如果实例化了 Config 类（继承自 `CodeIgniter\Config\BaseConfig`），将会抛出 `ConfigException`。

这是因为注册器的自动发现机制若重复执行，可能导致 Config 类属性被重复赋值。

所有 Registrar 类（所有命名空间中的 **Config/Registrar.php**）必须修改为在加载或实例化时不实例化任何 Config 类。

如果你使用的包/模块包含此类 Registrar 类，需要修复这些包/模块中的 Registrar 类。

以下是不再适用的代码示例:

```
<?php

namespace CodeIgniter\Shield\Config;

use Config\App;

class Registrar
{
    public function __construct()
    {
        $config = new App(); // Bad. When this class is␣
↪instantiated, Config\App will be instantiated.

        // Does something.
    }

    public static function Pager(): array
    {
        return [
            'templates' => [
                'module_pager' => 'MyModule\Views\Pager',
            ],
        ];
    }

    public static function hack(): void
    {
        $config = config('Cache');

        // Does something.
    }
}
```

(续下页)

(接上页)

```
Registrar::hack(); // Bad. When this class is loaded, Config\Cache␣
↪will be instantiated.
```

Session ID (SID) 变更

现在*Session 库* 强制使用 PHP 默认的 32 字符 SID（每字符 4 位熵）。此变更是为了匹配 PHP 9 的行为。

即始终使用以下设置:

```
session.sid_bits_per_character = 4
session.sid_length = 32
```

先前版本遵循 PHP ini 设置。因此此变更可能改变你的 SID 长度。

如无法接受此变更，请自定义 Session 类库。

接口变更

部分接口已变更。实现这些接口的类应更新其 API 以反映变更。详见更新日志。

方法签名变更

部分方法签名已变更。继承这些方法的类应更新其 API 以反映变更。详见更新日志。

移除已弃用项

部分已弃用项已被移除。如仍在使用这些项或继承这些类，请更新代码。详见更新日志。

重大增强

过滤器变更

`Filters` 类已变更，允许在 before 或 after 阶段多次运行相同过滤器（使用不同参数）。

如继承 `Filters` 类，需根据以下变更进行调整:

- 数组属性 `$filters` 和 `$filtersClasses` 的结构已变更
- 属性 `$arguments` 和 `$argumentsClass` 已停用
- `Filters` 已调整为不重复实例化相同过滤器类。如过滤器类在 before 和 after 阶段均使用，将复用同一实例

项目文件

项目空间（根目录、app、public、writable）中的部分文件已更新。由于这些文件位于 **system** 范围之外，需手动干预才能更新。

可通过第三方 CodeIgniter 模块辅助合并项目空间变更：在 Packagist 上探索。

内容变更

以下文件有重大变更（包含弃用或视觉调整），建议将更新版本合并至你的应用：

配置

- **app/Config/Feature.php**
  - `Config\Feature::$autoRoutesImproved` 已变更为 `true`
  - 新增 `Config\Feature::$strictLocaleNegotiation`
- **app/Config/Routing.php**
  - `Config\Routing::$translateUriToCamelCase` 已变更为 `true`

所有变更

以下是 **项目空间**中所有发生变更的文件列表（多数为不影响运行的注释或格式调整）：

- app/Config/Cache.php
- app/Config/Constants.php
- app/Config/Database.php
- app/Config/Feature.php
- app/Config/Format.php

- app/Config/Kint.php
- app/Config/Routing.php
- app/Config/Security.php
- app/Views/errors/html/debug.css
- app/Views/errors/html/error_400.php
- preload.php
- public/index.php
- spark

从 4.5.7 升级到 4.5.8

请参考与你的安装方法对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到现有项目升级*
- *手动安装升级*

- 项目文件
  - 内容更改
  - 所有更改

项目文件

项目空间（root、app、public、writable）中的一些文件已更新。由于这些文件位于 **system** 范围之外，它们不会在没有你干预的情况下被更改。

备注： 有一些第三方 CodeIgniter 模块可用于帮助合并项目空间的更改：在 Packagist 上探索。

内容更改

以下文件收到了重大更改（包括弃用或视觉调整），建议你将更新后的版本与你的应用程序合并：

所有更改

这是 **项目空间**中所有已更改文件的列表；许多更改只是注释或格式调整，不会影响运行时：

- preload.php
- public/index.php
- spark

从 4.5.6 升级到 4.5.7

请参考与你的安装方法对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到现有项目升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

项目空间（root、app、public、writable）中的一些文件已更新。由于这些文件位于 **system** 范围之外，它们不会在没有你干预的情况下被更改。

备注： 有一些第三方 CodeIgniter 模块可用于帮助合并项目空间的更改：在 Packagist 上探索。

所有更改

这是 **项目空间**中所有已更改文件的列表；许多更改只是注释或格式调整，不会影响运行时：

- app/Views/errors/cli/error_exception.php

从 4.5.5 升级到 4.5.6

请参考与你的安装方法对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到现有项目升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

项目空间（root、app、public、writable）中的一些文件已更新。由于这些文件位于 **system** 范围之外，它们不会在没有你干预的情况下被更改。

备注： 有一些第三方 CodeIgniter 模块可用于帮助合并项目空间的更改：在 Packagist 上探索。

所有更改

这是 **项目空间**中所有已更改文件的列表；许多更改只是注释或格式调整，不会影响运行时：

- app/Config/Events.php
- app/Config/Format.php
- app/Controllers/BaseController.php

- app/Views/errors/cli/error_exception.php
- app/Views/errors/html/error_exception.php

从 4.5.4 升级到 4.5.5

请参考与你的安装方法相对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到现有项目升级*
- *手动安装升级*

- 项目文件
  - 内容更改
  - 所有更改

项目文件

项目空间（根目录、app、public、writable）中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，因此不会在没有你干预的情况下进行更改。

备注： 有一些第三方 CodeIgniter 模块可以帮助合并项目空间的更改：在 Packagist 上探索。

内容更改

以下文件进行了重大更改（包括弃用或视觉调整），建议你将更新版本与应用程序合并：

配置

- preload.php

所有更改

这是 **项目空间**中所有收到更改的文件列表；许多只是简单的注释或格式调整，对运行时没有影响：

- preload.php

从 4.5.3 升级到 4.5.4

请参考与你的安装方法相对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到现有项目升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

项目空间（root、app、public、writable）中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，因此不会在没有你干预的情况下更改。

备注： 有一些第三方的 CodeIgniter 模块可以帮助合并项目空间的更改：在 Packagist 上探索。

所有更改

这是 **项目空间**中所有收到更改的文件列表；许多只是简单的注释或格式调整，对运行时没有影响：

- app/Config/Events.php
- composer.json

从 4.5.2 升级到 4.5.3

请参考与你的安装方式对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到一个现有项目升级*
- *手动安装升级*

- 项目文件
  - 内容更改
  - 所有更改

项目文件

项目空间（root, app, public, writable）中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，没有你的干预它们不会被更改。

备注： 有一些第三方的 CodeIgniter 模块可以帮助合并项目空间的变化：在 Packagist 上探索。

内容更改

以下文件收到了显著更改（包括弃用或视觉调整），建议你将更新版本与你的应用程序合并：

配置

- composer.json

所有更改

以下是 **项目空间** 中所有收到更改的文件列表；许多更改将是简单的注释或格式调整，不会影响运行时：

- composer.json

从 4.5.1 升级到 4.5.2

请参考与你的安装方式对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到一个现有项目升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

项目空间（root, app, public, writable）中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，没有你的干预它们不会被更改。

备注： 有一些第三方的 CodeIgniter 模块可以帮助合并项目空间的变化：在 Packagist 上探索。

所有更改

以下是 **项目空间**中所有收到更改的文件列表；许多更改将是简单的注释或格式调整，不会影响运行时：

- app/Config/DocTypes.php
- app/Config/Exceptions.php
- preload.php
- spark
- writable/.htaccess
- writable/cache/index.html
- writable/debugbar/index.html
- writable/index.html
- writable/logs/index.html
- writable/session/index.html

从 4.5.0 升级到 4.5.1

请参阅与你的安装方法对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到一个现有项目升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

项目空间（root, app, public, writable）中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，没有你的干预它们不会被更改。

有一些第三方的 CodeIgniter 模块可以帮助合并对项目空间的更改：在 Packagist 上探索。

所有更改

这是一个 **项目空间**内所有收到更改的文件列表；许多将只是简单的注释或格式更改，对运行时没有影响：

- .gitignore
- composer.json
- phpunit.xml.dist
- tests/.htaccess
- tests/index.html
- writable/debugbar/.gitkeep (已移除)
- writable/debugbar/index.html
- writable/index.html

从 4.4.8 升级到 4.5.0

请参阅与你的安装方法对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到一个现有项目升级*
- *手动安装升级*

- *强制文件更改*
  - *index.php 和 spark*
- *破坏性变更*
  - *小写 HTTP 方法名*

  - 嵌套路由组和选项
  - 过滤器执行顺序
  - *API\ResponseTrait* 和字符串数据
  - *FileLocator::findQualifiedNameFromPath()*
  - *BaseModel::getIdValue()*
  - *Factories*
  - 自动路由（传统）
  - 方法签名更改
  - 移除的弃用项
- 破坏性改进
  - *404* 覆盖状态码
  - *Validation::run()* 签名
- 项目文件
  - 内容变更
  - 所有更改

强制文件更改

index.php 和 spark

以下文件进行了破坏性变更，**你必须将更新的版本**合并到你的应用程序中：

- `public/index.php`
- `spark`

重要： 如果不更新上述文件，在运行 `composer update` 之后 CodeIgniter 将无法正常工作。

升级步骤例如如下：

```
composer update
cp vendor/codeigniter4/framework/public/index.php public/index.php
cp vendor/codeigniter4/framework/spark spark
```

破坏性变更

小写 HTTP 方法名

Request::getMethod()

由于历史原因，`Request::getMethod()` 默认返回小写的 HTTP 方法名。

但是方法标记是区分大小写的，因为它可能用作区分大小写的方法名的对象系统的网关。按照惯例，标准化的方法都定义为全大写的 US-ASCII 字母。详情见 https://www.rfc-editor.org/rfc/rfc9110#name-overview。

现在，`Request::getMethod()` 中已弃用的 `$upper` 参数已被移除，而 `getMethod()` 返回的是原样的 HTTP 方法名。即，大写形式如 "GET"，"POST" 等等。

如果你需要小写的 HTTP 方法名，请使用 PHP 的 `strtolower()` 函数:

```
strtolower($request->getMethod())
```

并且在你的应用代码中应使用大写的 HTTP 方法名。

app/Config/Filters.php

你应将 **app/Config/Filters.php** 中 `$methods` 的键更新为大写:

```
public array $methods = [
    'POST' => ['invalidchars', 'csrf'],
    'GET'  => ['csrf'],
];
```

CURLRequest::request()

在之前的版本中，你可以将小写的 HTTP 方法传递给 `request()` 方法。这个错误已被修复。

现在，你必须传递正确的 HTTP 方法名，如 `GET`, `POST`。否则你会得到错误响应:

```
$client   = \Config\Services::curlrequest();
$response = $client->request('get', 'https://www.google.com/', [
    'http_errors' => false,
]);
$response->getStatusCode(); // 之前版本: 200
                            // 现在版本: 405
```

嵌套路由组和选项

阻止传递给外部 `group()` 的选项与内部 `group()` 的选项合并的错误已被修复。

请检查并更正你的路由配置，因为这可能会更改应用的选项值。

例如，

```
$routes->group('admin', ['filter' => 'csrf'], static function (
↪$routes) {
    $routes->get('/', static function () {
        // ...
    });

    $routes->group('users', ['namespace' => 'Users'], static␣
↪function ($routes) {
        $routes->get('/', static function () {
            // ...
        });
    });
});
```

现在，`csrf` 过滤器将为 `admin` 和 `admin/users` 路由执行。在之前的版本中，它仅为 `admin` 路由执行。另请参阅嵌套分组。

过滤器执行顺序

控制器过滤器执行顺序已更改。如果你希望维持之前版本的执行顺序，请在 `Config\Feature::$oldFilterOrder` 中设置 `true`。另请参阅过滤器执行顺序。

1. 过滤器组的执行顺序已更改。

   前置过滤器:

   ```
   之前: route → globals → methods → filters
   现在: globals → methods → filters → route
   ```

   后置过滤器:

   ```
   之前: route → globals → filters
   现在: route → filters → globals
   ```

2. 在 *Route* 过滤器和 *Filters* 过滤器中的后置过滤器执行顺序现在是反向的。

   与以下配置有关:

   ```
   // 在 app/Config/Routes.php 中
   $routes->get('/', 'Home::index', ['filter' => ['route1',
   ↪ 'route2']]);

   // 在 app/Config/Filters.php 中
   public array $filters = [
       'filter1' => ['before' => '*', 'after' => '*'],
       'filter2' => ['before' => '*', 'after' => '*'],
   ];
   ```

   前置过滤器:

   ```
   之前: route1 → route2 → filter1 → filter2
   现在: filter1 → filter2 → route1 → route2
   ```

   后置过滤器:

   ```
   之前: route1 → route2 → filter1 → filter2
   现在: route2 → route1 → filter2 → filter1
   ```

API\ResponseTrait 和字符串数据

在以前的版本中，如果你将字符串数据传递给 trait 方法，即使响应格式被确定为 JSON，框架还是会返回 HTML 响应。

现在，如果你传递字符串数据，它将正确返回 JSON 响应。另请参阅处理响应类型。

如果你希望保持之前版本的行为，请在控制器中将 `$stringAsHtml` 属性设置为 `true`。

FileLocator::findQualifiedNameFromPath()

在以前的版本中，`FileLocator::findQualifiedNameFromPath()` 返回带有前导 \ 的完全限定类名。现在，前导 \ 已被移除。

如果你的代码依赖于带前导 \ 的结果，请修正。

BaseModel::getIdValue()

`BaseModel::getIdValue()` 已更改为 `abstract`，实现已被移除。

如果你扩展了 `BaseModel`，请在子类中实现 `getIdValue()` 方法。

Factories

工厂 已更改为最终类。在极不可能的情况下，如果你继承了 Factories，请停止继承并将代码复制到你的 Factories 类中。

自动路由（传统）

在以前的版本中，即使未找到相应的控制器，也可能会执行控制器过滤器。

此错误已被修复，现在如果未找到控制器，将抛出 `PageNotFoundException` 且不会执行过滤器。

如果你的代码依赖于此错误，例如你期望即使在不存在的页面上也会执行全局过滤器，请使用新的必需过滤器。

方法签名更改

一些方法签名已更改。扩展它们的类应更新其 API 以反映这些更改。详情请参阅*ChangeLog*。

移除的弃用项

一些弃用项已被移除。如果你仍在使用这些项或扩展这些类，请升级你的代码。详情请参阅*ChangeLog*。

破坏性改进

404 覆盖状态码

在以前的版本中，*404 重写* 默认返回状态码为 `200` 的响应。现在它默认返回 `404`。

如果你需要 `200`，请在控制器中设置:

```
$routes->set404Override(static function () {
    response()->setStatusCode(200);

    echo view('my_errors/not_found.html');
});
```

Validation::run() 签名

`Validation::run()` 和 `ValidationInterface::run()` 的方法签名已更改。`$dbGroup` 参数的 `?string` 类型提示已被移除。扩展的类也应移除该参数以不破坏 LSP。

项目文件

项目空间（root, app, public, writable）中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，没有你的干预它们不会被更改。

有一些第三方的 CodeIgniter 模块可以帮助合并对项目空间的更改：在 Packagist 上探索。

内容变更

以下文件进行了重要更改（包括弃用或视觉调整），建议将更新的版本与应用程序合并：

配置

app/Config/Filters.php

已添加必需过滤器，因此做了以下更改。另请参阅*ChangeLog*。

基类已更改:

```
class Filters extends \CodeIgniter\Config\Filters
```

在 `$aliases` 属性中添加了以下项目:

```
public array $aliases = [
    // ...
    'forcehttps'    => \CodeIgniter\Filters\ForceHTTPS::class,
    'pagecache'     => \CodeIgniter\Filters\PageCache::class,
    'performance'   => \CodeIgniter\Filters\
↪PerformanceMetrics::class,
];
```

添加了一个新属性 `$required`，并设置如下:

```
public array $required = [
    'before' => [
        'forcehttps', // 强制全局安全请求
        'pagecache',  // 网页缓存
    ],
    'after' => [
        'pagecache',   // 网页缓存
        'performance', // 性能指标
        'toolbar',     // 调试工具栏
    ],
];
```

`$global['after']` 中的 `'toolbar'` 被移除。

其他

- **app/Config/Boot/production.php**
  - error_reporting() 的默认错误级别已更改为 E_ALL & ~E_DEPRECATED。
- **app/Config/Cors.php**
  - 添加了处理 CORS 配置。
- **app/Config/Database.php**
  - $default 中 charset 的默认值已更改为 utf8mb4。
  - $default 中 DBCollat 的默认值已更改为 utf8mb4_general_ci。
  - $tests 中 DBCollat 的默认值已更改为 ''。
- **app/Config/Feature.php**
  - 添加了 Config\Feature::$oldFilterOrder。另请参阅过滤器执行顺序。
  - 添加了 Config\Feature::$limitZeroAsAll。另请参阅*limit(0) 行为*。
  - 移除了 Config\Feature::$multipleFilters，因为现在多重过滤器 已经默认启用。
- **app/Config/Kint.php**
  - 不再继承 BaseConfig，因为启用配置缓存 可能会导致错误。
- **app/Config/Optimize.php**
  - 添加了处理优化配置。
- **app/Config/Security.php**
  - 在 production 环境中将 $redirect 属性更改为 true。

所有更改

这是一个 **项目空间**内所有收到更改的文件列表；许多将只是简单的注释或格式更改，对运行时没有影响：

- app/Config/Autoload.php
- app/Config/Boot/production.php
- app/Config/Cache.php
- app/Config/Cors.php
- app/Config/Database.php
- app/Config/Feature.php
- app/Config/Filters.php
- app/Config/Generators.php
- app/Config/Kint.php
- app/Config/Optimize.php
- app/Config/Routing.php
- app/Config/Security.php
- app/Config/Session.php
- app/Views/errors/cli/error_exception.php
- app/Views/errors/html/error_exception.php
- app/Views/welcome_message.php
- composer.json
- env
- phpunit.xml.dist
- preload.php
- public/index.php
- spark

从 4.4.7 升级到 4.4.8

请参阅与你的安装方法对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到一个现有项目升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

项目空间(root, app, public, writable)中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，没有你的干预它们不会被更改。

有一些第三方的 CodeIgniter 模块可以帮助合并对项目空间的更改：在 Packagist 上探索。

所有更改

这是一个 **项目空间**内所有收到更改的文件列表；许多将只是简单的注释或格式更改，对运行时没有影响：

- app/.htaccess
- public/.htaccess
- writable/.htaccess

从 4.4.6 升级到 4.4.7

请参阅与你的安装方法对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到一个现有项目升级*
- *手动安装升级*

- 强制文件更改
  - *URI* 安全性
  - 错误文件
- 重大变更
  - 控制器过滤器中的路径
  - *Time::difference()* 和夏令时
- 项目文件
  - 内容变更
  - 所有更改

强制文件更改

URI 安全性

添加了检查 URI 中不包含不允许的字符串的功能。此检查等同于 CodeIgniter 3 中的 URI 安全性。

我们建议你启用此功能。在 **app/Config/App.php** 文件中添加以下内容:

```
public string $permittedURIChars = 'a-z 0-9~%.:_\-';.
```

详情请参阅*URI 安全性*。

错误文件

错误页面已更新。请更新以下文件:

- app/Views/errors/html/debug.css
- app/Views/errors/html/error_exception.php

重大变更

控制器过滤器中的路径

已修复*控制器过滤器* 处理的 URI 路径未进行 URL 解码的错误。

备注: 请注意*Router* 处理 URL 解码后的 URI 路径。

`Config\Filters` 中有一些地方可以指定 URI 路径。如果路径在 URL 解码后有不同的值，请将它们更改为 URL 解码后的值。

例如:

```
public array $globals = [
    'before' => [
        'csrf' => ['except' => '%E6%97%A5%E6%9C%AC%E8%AA%9E/*'],
    ],
    // ...
];
```

↓

```
public array $globals = [
    'before' => [
        'csrf' => ['except' => '日本語/*'],
    ],
    // ...
];
```

Time::difference() 和夏令时

在以前的版本中，当使用 `Time::difference()` 比较日期时，如果由于夏令时 (DST) 导致日期包含不同于 24 小时的一天，则会返回意外结果。详情请参阅*Times and Dates 中的备注*。

此错误已修复，因此在这种情况下，日期比较将被移后一日。

在某些不太可能的情况下，如果你希望保持以前版本的行为，请在将要比较的两个日期传递给 `Time::difference()` 之前，将它们的时区更改为 UTC。

项目文件

项目空间（root, app, public, writable）中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，没有你的干预它们不会被更改。

有一些第三方的 CodeIgniter 模块可以帮助合并对项目空间的更改：在 Packagist 上探索。

内容变更

以下文件进行了重要更改（包括弃用或视觉调整），建议你将更新的版本与应用程序合并：

配置

- **app/Config/App.php**
  - 添加了属性 `$permittedURIChars`。详情请参阅*URI 安全性*。

所有更改

这是一个 **项目空间**内所有收到更改的文件列表；许多将只是简单的注释或格式更改，对运行时没有影响：

- app/Config/App.php
- app/Config/Cache.php
- app/Config/ContentSecurityPolicy.php
- app/Config/Database.php
- app/Config/Exceptions.php
- app/Config/Filters.php
- app/Config/Format.php
- app/Config/Logger.php
- app/Config/Mimes.php
- app/Config/Routing.php
- app/Config/Toolbar.php

- app/Config/Validation.php
- app/Config/View.php
- app/Controllers/BaseController.php
- app/Views/errors/html/debug.css
- app/Views/errors/html/error_exception.php
- composer.json

从 4.4.5 升级到 4.4.6

请参阅与你的安装方法对应的升级说明。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到一个现有项目升级*
- *手动安装升级*

- 重大变更
  - *Time::createFromTimestamp()* 时区变更
- 项目文件
  - 所有更改

重大变更

Time::createFromTimestamp() 时区变更

当你没有指定时区时，现在*Time::createFromTimestamp()* 返回一个具有应用程序时区的 Time 实例。

如果你想保持时区为 UTC，你需要调用 `setTimezone('UTC')`:

```
use CodeIgniter\I18n\Time;

$time = Time::createFromTimestamp(1501821586)->setTimezone('UTC');
```

项目文件

项目空间（root, app, public, writable）中的一些文件收到了更新。由于这些文件位于 **system** 范围之外，没有你的干预它们不会被更改。

有一些第三方的 CodeIgniter 模块可以帮助合并对项目空间的更改：在 Packagist 上探索。

所有更改

这是一个 **项目空间**内所有收到更改的文件列表；许多将只是简单的注释或格式更改，对运行时没有影响：

- app/Config/App.php
- app/Config/Routing.php
- app/Views/welcome_message.php
- composer.json

从 4.4.4 升级到 4.4.5

请参考与你的安装方法相对应的升级指南。

- *Composer 安装 App Starter 升级*
- *Composer 安装将 CodeIgniter4 添加到现有项目的升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

项目空间（根目录，app，public，writable）中的一些文件接收到更新。由于这些文件在 **system** 范围之外，它们不会在没有你的干预的情况下被更改。

有一些第三方 CodeIgniter 模块可用于帮助合并对项目空间的更改：在 Packagist 上探索。

所有更改

这是在 **项目空间**中接收到更改的所有文件的列表；许多文件只是简单的注释或格式更改，对运行时没有影响：

- composer.json

从 4.4.3 升级到 4.4.4

请参考对应于你的安装方法的升级指南。

- *Composer 安装 App Starter 升级*
- *Composer 安装在已有项目中添加 CodeIgniter4 升级*
- *手动安装升级*

- 强制性文件更改
  - 错误文件
- 重大更改
  - 使用 *Dot* 数组语法验证
  - 验证规则匹配和差异
  - 在 *CURLRequest* 中使用 ssl_key 选项已被移除
- 项目文件
  - 所有更改

强制性文件更改

错误文件

更新以下文件以显示正确的错误信息：

- app/Views/errors/cli/error_exception.php
- app/Views/errors/html/error_exception.php

重大更改

使用 Dot 数组语法验证

如果你在验证规则中使用*Dot 数组语法*，已修复了一个 * 在错误下标验证数据的错误。

在以前的版本中，规则 key `contacts.*.name` 错误地捕获了任何级别的数据，如 `contacts.*.name`，`contacts.*.*.name`，`contacts.*.*.*.name` 等。

以下代码解释了详细信息：

```
use Config\Services;

$validation = Services::validation();

$data = [
    'contacts' => [
        'name' => 'Joe Smith',
        'just' => [
            'friends' => [
                ['name' => 'SATO Taro'],
                ['name' => 'Li Ming'],
                ['name' => 'Heinz Müller'],
            ],
        ],
    ],
];

$validation->setRules(
    ['contacts.*.name' => 'required|max_length[8]'],
);

$validation->run($data); // false

d($validation->getErrors());
/*
 Before: Captured `contacts.*.*.*.name` incorrectly.
 [
   contacts.just.friends.0.name => "The contacts.*.name field␣
↪cannot exceed 8 characters in length.",
```

(续下页)

(接上页)

```
  contacts.just.friends.2.name => "The contacts.*.name field␣
↪cannot exceed 8 characters in length.",
 ]

 After: Captures no data for `contacts.*.name`.
 [
  contacts.*.name => string (38) "The contacts.*.name field is␣
↪required.",
 ]
*/
```

如果你有依赖于这个错误的代码，修复规则 key。

验证规则匹配和差异

由于在严格和传统规则中使用 `matches` 和 `differs` 验证非字符串类型数据的情况下已经修复了错误，如果你正在使用这些规则并验证非字符串数据，验证结果可能会被更改（修复）。

注意，传统规则不应该用于验证非字符串的数据。

在 CURLRequest 中使用 *ssl_key* 选项已被移除

CURLRequest 选项 *ssl_key* 不再被识别。如果在使用，选项 *ssl_key* 必须被选项 *verify* 替代，以定义 CURLRequest 的 CA 包路径。

CURLRequest 选项 *verify* 也可以像往常一样接受 布尔值。

项目文件

项目空间（root，app，public，writable）中的一些文件已经更新。由于这些文件在 **system** 范围之外，没有你的干预不会发生改变。

有一些第三方 CodeIgniter 模块可用于帮助合并对项目空间的更改：在 Packagist 上探索。

所有更改

这是 **项目空间**中所有更改的文件列表；许多文件只是简单的注释或格式更改，对运行时没有影响：

- app/Config/App.php
- app/Config/Autoload.php
- app/Config/Boot/development.php
- app/Config/Boot/testing.php
- app/Config/Cache.php
- app/Config/Email.php
- app/Config/Filters.php
- app/Config/Kint.php
- app/Config/Modules.php
- app/Config/Publisher.php
- app/Config/Session.php
- app/Views/errors/cli/error_exception.php
- app/Views/errors/html/error_exception.php
- composer.json
- env
- spark

从 4.4.2 升级到 4.4.3

请参考与你的安装方法相对应的升级说明。

- *使用 Composer 安装 App Starter 升级*
- *使用 Composer 将 CodeIgniter4 添加到现有项目中升级*
- *手动安装升级*

- 必要的文件更改
  - *error_exception.php*
- 项目文件
  - 所有更改

必要的文件更改

error_exception.php

以下文件已经发生了重大更改，**你必须将更新的版本与你的应用程序合并**：

- app/Views/errors/html/error_exception.php

项目文件

项目空间 (根目录、app、public、writable) 中的一些文件已经更新。由于这些文件位于 **system** 范围之外，因此在没有你的干预下不会更改。

有一些第三方 CodeIgniter 模块可用于帮助合并对项目空间的更改：在 Packagist 上查看。

所有更改

这是 **项目空间**中所有已经更改的文件的列表；其中许多只是注释或格式化的简单更改，对运行时没有影响：

- app/Config/Boot/development.php
- app/Config/Boot/production.php
- app/Config/Boot/testing.php
- app/Config/Filters.php
- app/Views/errors/html/error_404.php
- app/Views/errors/html/error_exception.php

从 4.4.1 升级到 4.4.2

请参考与你的安装方法相对应的升级说明。

- *使用 Composer 安装 App Starter 升级*
- *使用 Composer 安装将 CodeIgniter4 添加到现有项目并进行升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

项目空间 (根目录、app、public、writable) 中的一些文件已经更新。由于这些文件位于 **system** 范围之外，如果没有你的干预，它们将不会更改。

有一些第三方的 CodeIgniter 模块可用于帮助合并项目空间的更改：在 Packagist 上查看。

所有更改

这是 **项目空间**中所有已更改的文件列表；其中许多只是注释或格式化的简单更改，对运行时没有影响：

- app/Config/Migrations.php
- app/Config/View.php
- composer.json

从 4.4.0 升级到 4.4.1

请参考与你的安装方法相对应的升级说明。

- *使用 Composer 安装 App Starter 升级*
- *使用 Composer 安装将 CodeIgniter4 添加到现有项目并进行升级*
- *手动安装升级*

- 项目文件
  - 内容更改
  - 所有更改

项目文件

项目空间 (根目录、app、public、writable) 中的一些文件已经更新。由于这些文件位于 **system** 范围之外，如果没有你的干预，它们将不会更改。

有一些第三方的 CodeIgniter 模块可用于帮助合并项目空间的更改：在 Packagist 上查看。

内容更改

版本 4.4.1 没有更改项目文件中的任何可执行代码。

所有更改

这是 **项目空间**中所有已更改的文件列表；其中许多只是注释或格式化的简单更改，对运行时没有影响：

- app/Config/Autoload.php
- app/Config/DocTypes.php
- app/Config/Email.php
- app/Config/ForeignCharacters.php
- app/Config/Mimes.php
- app/Config/Modules.php
- composer.json

从 4.3.8 升级到 4.4.0

请参考与你的安装方法对应的升级说明。

- 使用 *Composer* 安装 *App Starter* 升级
- 使用 *Composer* 安装将 *CodeIgniter4* 添加到现有项目并进行升级
- 手动安装升级

- 安全性
  - 使用 *$this->validate()* 时
- 破坏性变更
  - *URI::setSegment()* 更改
  - 站点 *URI* 更改
  - 当你扩展异常时
  - 自动路由（改进版）和 *translateURIDashes*
  - 传递带有命名空间的类名到工厂时
  - 接口更改
  - 方法签名更改
  - *RouteCollection::$routes*
- 必要的文件更改
  - *index.php* 和 *spark*
  - 配置文件
- 重大改进
- 项目文件
  - 内容更改
  - 所有更改

安全性

使用 $this->validate() 时

在控制器的*$this->validate()* 中存在已知的潜在漏洞，可绕过验证。攻击可以使开发人员误解未经验证的空数据为已验证数据并继续处理。

已添加*Validation::getValidated()* 方法，以确保获取已验证数据。

因此，在你的控制器中使用 `$this->validate()` 时，应使用新的 `Validation::getValidated()` 方法获取已验证的数据。

```
// In Controller.

if (! $this->validateData($data, [
    'username' => 'required',
    'password' => 'required|min_length[10]',
])) {
    // The validation failed.
    return view('login', [
        'errors' => $this->validator->getErrors(),
    ]);
}

// The validation was successful.

// Get the validated data.
$validData = $this->validator->getValidated();
```

破坏性变更

URI::setSegment() 更改

由于一个错误，在之前的版本中，如果指定了最后一个段 `+2`，将不会抛出异常。这个错误已经修复。

如果你的代码依赖于这个错误，请修复段编号。

```
// URI: http://example.com/one/two

// Before:
$uri->setSegment(4, 'three');
// The URI will be http://example.com/one/two/three

// After:
$uri->setSegment(4, 'three'); // Will throw Exception
$uri->setSegment(3, 'three');
// The URI will be http://example.com/one/two/three
```

站点 URI 更改

- 由于对当前 URI 确定进行了重新制定，框架可能以与以前版本不同的方式返回站点 URI 或 URI 路径。这可能会破坏你的测试代码。如果现有测试失败，请更新断言。
- 如果你的 baseURL 具有子目录，并且通过 `URI::getPath()` 方法获取当前 URI 的相对路径到 baseURL，你必须改用新的 `SiteURI::getRoutePath()` 方法。

有关详细信息，请参见*站点 URI 更改*。

当你扩展异常时

如果你扩展了 `CodeIgniter\Debug\Exceptions` 并且未覆盖 `exceptionHandler()` 方法，那么在 **app/Config/Exceptions.php** 中定义新的 `Config\Exceptions::handler()` 方法将导致执行指定的异常处理程序。

你的覆盖代码将不再执行，因此请通过定义自己的异常处理程序进行必要的更改。

请参阅*自定义异常处理器* 了解详细信息。

自动路由（改进版）和 translateURIDashes

在使用自动路由（改进版）和 `$translateURIDashes` 为 true 时（`$routes->setTranslateURIDashes(true)`），在以前版本中由于错误，两个 URI 对应一个控制器方法，一个 URI 用于破折号（例如 **foo-bar**），另一个 URI 用于下划线（例如 **foo_bar**）。

此错误已经修复，现在不再支持下划线 URI（**foo_bar**）。

如果你有指向下划线 URI（**foo_bar**）的链接，请将其更新为破折号 URI（**foo-bar**）。

传递带有命名空间的类名到工厂时

传递带有命名空间的类名到工厂的行为已更改。有关详细信息，请参见*ChangeLog*。

如果你有类似于 `model(\Myth\Auth\Models\UserModel::class)` 或 `model('Myth\Auth\Models\UserModel')` 的代码（代码可能在第三方包中），并且希望加载你的 `App\Models\UserModel`，你需要在加载该类之前定义要加载的类名:

```
Factories::define('models', 'Myth\Auth\Models\UserModel', 'App\
↪Models\UserModel');
```

有关详细信息，请参见*定义要加载的类名*。

接口更改

已进行了一些接口更改。实现它们的类应该更新其 API 以反映更改。有关详细信息，请参见*接口更改*。

方法签名更改

已进行了一些方法签名更改。扩展它们的类应该更新其 API 以反映更改。有关详细信息，请参见*方法签名更改*。

此外，某些构造函数和 `Services::security()` 的参数类型已更改。如果你使用这些参数调用它们，请更改参数值。有关详细信息，请参见*参数类型更改*。

RouteCollection::$routes

受保护属性 `$routes` 的数组结构已进行了修改以提高性能。

如果你扩展了 `RouteCollection` 并使用了 `$routes`，请更新你的代码以匹配新的数组结构。

必要的文件更改

index.php 和 spark

以下文件已经接收到重大更改，**你必须将更新后的版本与你的应用程序合并**：

- `public/index.php` (还请参阅*CodeIgniter 和 exit()*)
- `spark`

重要: 如果你不更新上述文件，运行 `composer update` 后 CodeIgniter 将无法正常工作。

升级过程，例如如下：

```
composer update
cp vendor/codeigniter4/framework/public/index.php public/index.php
cp vendor/codeigniter4/framework/spark spark
```

配置文件

app/Config/App.php

属性 `$proxyIPs` 必须是数组。如果你不使用代理服务器，则它必须为 `public array $proxyIPs = [];`。

app/Config/Routing.php

为了清理路由系统，进行了以下更改：

- 新的 **app/Config/Routing.php** 文件保存了以前在 Routes 文件中的设置。
- **app/Config/Routes.php** 文件经过简化，仅包含路由，没有设置和冗余的内容。
- 不再自动加载特定于环境的路由文件。

因此，你需要执行以下操作：

1. 从新框架中复制 **app/Config/Routing.php** 到你的 **app/Config** 目录，并进行配置。
2. 删除不再需要的 **app/Config/Routes.php** 中的所有设置。
3. 如果使用特定于环境的路由文件，请将它们添加到 **app/Config/Routing.php** 中的 `$routeFiles` 属性中。

app/Config/Toolbar.php

你需要添加新属性 `$watchedDirectories` 和 `$watchedExtensions` 以进行热重载：

```
--- a/app/Config/Toolbar.php
+++ b/app/Config/Toolbar.php
@@ -88,4 +88,31 @@ class Toolbar extends BaseConfig
      * `$maxQueries` defines the maximum amount of queries that␣
↪will be stored.
      */
     public int $maxQueries = 100;
+
+    /**
+     * ------------------------------------------------------------------
↪--------------
+     * Watched Directories
+     * ------------------------------------------------------------------
↪--------------
+     *
+     * Contains an array of directories that will be watched for␣
↪changes and
+     * used to determine if the hot-reload feature should reload␣
```

(续下页)

(接上页)

```
↪the page or not.
+     * We restrict the values to keep performance as high as␣
↪possible.
+     *
+     * NOTE: The ROOTPATH will be prepended to all values.
+     */
+    public array $watchedDirectories = [
+        'app',
+    ];
+
+    /**
+     * ------------------------------------------------------------
↪--------------
+     * Watched File Extensions
+     * ------------------------------------------------------------
↪--------------
+     *
+     * Contains an array of file extensions that will be watched␣
↪for changes and
+     * used to determine if the hot-reload feature should reload␣
↪the page or not.
+     */
+    public array $watchedExtensions = [
+        'php', 'css', 'js', 'html', 'svg', 'json', 'env',
+    ];
 }
```

app/Config/Events.php

你需要添加代码以为热重载 添加一个路由:

```
--- a/app/Config/Events.php
+++ b/app/Config/Events.php
@@ -4,6 +4,7 @@ namespace Config;

 use CodeIgniter\Events\Events;
 use CodeIgniter\Exceptions\FrameworkException;
```

(续下页)

(接上页)

```
+use CodeIgniter\HotReloader\HotReloader;

 /*
  * ----------------------------------------------------------------
 ↪----
@@ -44,5 +45,11 @@ Events::on('pre_system', static function () {
     if (CI_DEBUG && ! is_cli()) {
         Events::on('DBQuery', 'CodeIgniter\Debug\Toolbar\
 ↪Collectors\Database::collect');
         Services::toolbar()->respond();
+        // Hot Reload route - for framework use on the hot␣
 ↪reloader.
+        if (ENVIRONMENT === 'development') {
+            Services::routes()->get('__hot-reload', static␣
 ↪function () {
+                (new HotReloader())->run();
+            });
+        }
     }
 });
```

app/Config/Cookie.php

app/Config/App.php 中的 Cookie 配置项不再使用。

1. 从新框架中复制 **app/Config/Cookie.php** 到你的 **app/Config** 目录，并进行配置。
2. 删除 **app/Config/App.php** 中的属性（从 `$cookiePrefix` 到 `$cookieSameSite`）。

app/Config/Security.php

app/Config/App.php 中的 CSRF 配置项不再使用。

1. 从新框架中复制 **app/Config/Security.php** 到你的 **app/Config** 目录，并进行配置。
2. 删除 **app/Config/App.php** 中的属性（从 `$CSRFTokenName` 到 `$CSRFSameSite`）。

app/Config/Session.php

app/Config/App.php 中的 Session 配置项不再使用。

1. 从新框架中复制 **app/Config/Session.php** 到你的 **app/Config** 目录，并进行配置。
2. 删除 **app/Config/App.php** 中的属性（从 `$sessionDriver` 到 `$sessionDBGroup`）。

重大改进

- **路由：** `RouteCollection::__construct()` 的方法签名已更改。添加了第三个参数 `Routing $routing`。扩展类应该同样添加参数以不违反 LSP。
- **验证：** `Validation::check()` 的方法签名已更改。`$rule` 参数上的 `string` 类型提示已被删除。扩展类应该同样删除类型提示以不违反 LSP。

项目文件

项目空间中的一些文件（根目录、app、public、writable）已接收到更新。由于这些文件位于 **system** 范围之外，它们将不会在没有你干预的情况下更改。

有一些第三方 CodeIgniter 模块可帮助你合并对项目空间的更改：在 Packagist 上查看。

内容更改

以下文件已接收到重大更改（包括弃用或视觉调整），建议你将更新后的版本与你的应用程序合并：

配置

- **app/Config/CURLRequest.php**
  - *$shareOptions* 的默认值已更改为 `false`。
- **app/Config/Exceptions.php**
  - 添加了新方法 `handler()`，定义自定义异常处理程序。请参阅自定义异常处理器。

所有更改

这是 **项目空间**中所有文件的更改列表；其中许多将是对运行时没有影响的注释或格式化：

- app/Config/App.php
- app/Config/CURLRequest.php
- app/Config/Cookie.php
- app/Config/Database.php
- app/Config/Events.php
- app/Config/Exceptions.php
- app/Config/Filters.php
- app/Config/Routes.php
- app/Config/Routing.php
- app/Config/Toolbar.php
- public/index.php
- spark

从 4.3.7 升级到 4.3.8

请参考与你的安装方法相对应的升级说明。

- *使用 Composer 安装的应用程序启动器升级*
- *使用 Composer 安装的将 CodeIgniter4 添加到现有项目中升级*
- *手动安装升级*

- *项目文件*
  - *内容更改*
  - *所有更改*

项目文件

项目空间 (根目录、app、public、writable) 中的一些文件已经更新。由于这些文件位于 **system** 范围之外，因此不会在没有你干预的情况下进行更改。

有一些第三方 CodeIgniter 模块可用于帮助合并对项目空间的更改：在 Packagist 上查看。

内容更改

以下文件已经进行了重大更改（包括弃用或视觉调整），建议你将更新后的版本与你的应用程序合并：

配置

- composer.json

所有更改

这是 **项目空间**中所有已更改的文件的列表；其中许多只是注释或格式变化，对运行时没有影响：

- composer.json

从 4.3.6 升级到 4.3.7

请参考与你的安装方法相对应的升级说明。

- *使用 Composer 安装的应用程序启动器升级*
- *使用 Composer 安装的将 CodeIgniter4 添加到现有项目中升级*
- *手动安装升级*

- *重大变更*
  - *功能测试请求体*
  - *Validation::loadRuleGroup()* 的返回值
- *项目文件*

- 内容更改
- 所有更改

重大变更

功能测试请求体

如果你调用了以下方法:

1. *withBody()*
2. 并且*withBodyFormat()*
3. 并将 `$params` 传递给*call()* (或简写方法)

则请求体的优先级已更改。如果你的测试代码受到此更改的影响，请进行修改。

例如，现在使用 `$params` 来构建请求体，而不使用 `$body`:

```
$this->withBody($body)->withBodyFormat('json')->call('post',
↪$params)
```

以前，`$body` 用于请求体。

Validation::loadRuleGroup() 的返回值

`Validation::loadRuleGroup()` 的返回值已从 "**rules 数组**" 更改为 "**rules 数组**和 **customErrors 数组**" 的 "**数组**" (`[rules, customErrors]`)。

如果你使用了该方法，请将代码更新如下:

```
$rules = $this->validation->loadRuleGroup($rules);
    ↓
[$rules, $customErrors] = $this->validation->loadRuleGroup($rules);
```

项目文件

项目空间 (根目录、app、public、writable) 中的一些文件已经更新。由于这些文件位于 **system** 范围之外，因此不会在没有你干预的情况下进行更改。

有一些第三方 CodeIgniter 模块可用于帮助合并对项目空间的更改：在 Packagist 上查看。

内容更改

以下文件已经进行了重大更改（包括弃用或视觉调整），建议你将更新后的版本与你的应用程序合并：

配置

- app/Config/Kint.php

所有更改

这是 **项目空间**中所有已更改的文件的列表；其中许多只是注释或格式变化，对运行时没有影响：

- app/Config/App.php
- app/Config/Autoload.php
- app/Config/Cache.php
- app/Config/ContentSecurityPolicy.php
- app/Config/Filters.php
- app/Config/Kint.php
- app/Config/Logger.php
- app/Config/Migrations.php
- app/Config/Modules.php
- app/Config/Paths.php
- app/Controllers/BaseController.php
- app/Controllers/Home.php

- composer.json

从 4.3.5 升级到 4.3.6

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *重大变更*
- *重大增强*
- *项目文件*
  - *所有更改*

重大变更

- `AutoRouterInterface::getRoute()` 新增了第二个参数 `string $httpVerb`。如果你实现了它, 请添加该参数。

重大增强

- `ValidationInterface::check()` 和 `Validation::check()` 的方法签名已更改。如果你扩展或实现了它们, 请更新签名。

项目文件

4.3.6 版本没有更改项目文件中的任何可执行代码。

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- composer.json

从 4.3.4 升级到 4.3.5

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- 重大变更
  - 验证占位符
  - *Session::stop()*
- 项目文件
  - 内容更改
  - 所有更改

重大变更

验证占位符

为了安全地使用验证占位符, 请记得为你将用作占位符的字段创建一个验证规则。

例如，如果你有以下代码:

```
$validation->setRules([
    'email' => 'required|max_length[254]|valid_email|is_
↪unique[users.email,id,{id}]',
]);
```

你需要为 `{id}` 添加规则:

```
$validation->setRules([
    'id'    => 'max_length[19]|is_natural_no_zero', // Add this
    'email' => 'required|max_length[254]|valid_email|is_
↪unique[users.email,id,{id}]',
]);
```

Session::stop()

在 v4.3.5 之前, 由于一个错误, `Session::stop()` 方法并没有销毁 session。这个方法已被修改为销毁 session, 并已不建议使用, 因为它与 `Session::destroy()` 方法完全相同。所以请使用*Session::destroy()* 方法替代。

如果你的代码依赖这个错误, 请用 `session_regenerate_id(true)` 替换它。

参见*Session 库*。

项目文件

项目空间中的一些文件 (根目录、app、public、writable) 已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。

有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改:在 Packagist 上探索。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

配置

- app/Config/Generators.php

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/App.php
- app/Config/Generators.php
- composer.json

从 4.3.3 升级到 4.3.4

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *重大变更*
  - *重定向状态码*
  - *Forge::modifyColumn() 和 NULL*
- *项目文件*
  - *内容更改*
  - *所有更改*

重大变更

重定向状态码

- 由于一个错误修复, 重定向的状态码可能会改变。参见*更新日志 v4.3.4*, 如果状态码不是你想要的, 可以*指定状态码*。

Forge::modifyColumn() 和 NULL

一个错误修复可能改变了*$forge->modifyColumn()* 的结果中的 NULL 约束。参见更新日志。要设置所需的 NULL 约束, 请更改 `Forge::modifyColumn()` 以始终指定 `null` 键。

请注意, 该错误可能在以前的版本中改变了意外的 NULL 约束。

项目文件

项目空间中的一些文件 (根目录、app、public、writable) 已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。

有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改:在 Packagist 上探索。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

配置

- app/Config/Generators.php

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/App.php
- app/Config/Generators.php
- composer.json
- public/index.php

从 4.3.2 升级到 4.3.3

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *项目文件*
  - *内容更改*
  - *所有更改*

项目文件

项目空间中的一些文件 (根目录、app、public、writable) 已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。

有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改:在 Packagist 上探索。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

配置

- **app/Config/Encryption.php**
  - 添加了缺失的属性 `$cipher` 以实现 CI3 加密兼容性。参见*用于与 CI3 保持兼容性的配置*。

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Common.php
- app/Config/Encryption.php
- composer.json

从 4.3.1 升级到 4.3.2

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- 重大变更
  - *base_url()*
  - *uri_string()*
- 必备文件变更
  - *composer.json*
- 项目文件
  - 内容更改
  - 所有更改

重大变更

base_url()

`base_url()` 的行为已修复。在以前的版本中, 当你调用 `base_url()` **不带参数时**, 它会返回不带尾部斜杠 (/) 的 baseURL。现在它会返回带有尾部斜杠的 baseURL。例如:

- 之前:`http://example.com`
- 之后:`http://example.com/`

如果你有调用不带参数的 `base_url()` 的代码, 可能需要调整 URL。

uri_string()

uri_string() 的行为已修复。在以前的版本中, 当你导航到 baseURL 时, 它会返回 `/`。现在它返回一个空字符串 (`''`)。

如果你有调用 `uri_string()` 的代码, 可能需要调整它。

备注: *uri_string()* 返回相对于 baseURL 的 URI 路径。如果 baseURL 包含子文件夹, 它不是完整的 URI 路径。如果要用于 HTML 链接, 最好与*site_url()* 一起使用, 如 `site_url(uri_string())`。

必备文件变更

composer.json

如果你手动安装了 CodeIgnter 并且正在使用或计划使用 Composer, 请删除以下行:

```
{
    ...
    "scripts": {
        "post-update-cmd": [
            "CodeIgniter\\ComposerScripts::postUpdate"  <-- 移除此行
        ],
        "test": "phpunit"
    },
    ...
}
```

项目文件

项目空间中的一些文件 (根目录、app、public、writable) 已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。

有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改:在 Packagist 上探索。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

- app/Config/Mimes.php
- app/Views/errors/html/error_exception.php
- composer.json
- public/.htaccess

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/App.php
- app/Config/Mimes.php
- app/Views/errors/html/error_exception.php
- composer.json
- public/.htaccess

从 4.3.0 升级到 4.3.1

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *Composer* 版本
- 必备文件变更
  - 配置文件
- 项目文件
  - 内容更改
  - 所有更改

Composer 版本

重要: 如果你使用 Composer,CodeIgniter v4.3 需要 Composer 2.0.14 或更高版本。

如果你使用的是更早版本的 Composer, 请升级你的 `composer` 工具, 删除 **vendor/** 目录, 并再次运行 `composer update`。

例如, 过程如下:

```
composer self-update
rm -rf vendor/
composer update
```

必备文件变更

配置文件

app/Config/Email.php

- 如果你在升级到 v4.3.0 时更新了 **app/Config/Email.php**, 你必须为 `$fromEmail`、`$fromName`、`$recipients`、`$SMTPHost`、`$SMTPUser` 和 `$SMTPPass` 设置默认值, 以应用环境变量 (**.env**) 值。
- 如果没有设置默认值, 设置这些环境变量将不会反映在配置对象中。

app/Config/Exceptions.php

- 如果你使用 PHP 8.2, 需要添加新的属性 `$logDeprecations` 和 `$deprecationLogLevel`。

项目文件

项目空间中的一些文件 (根目录、app、public、writable) 已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。

有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改:在 Packagist 上探索。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

配置

- **app/Config/Email.php**
  - 为 `$fromEmail`、`$fromName`、`$recipients`、`$SMTPHost`、`$SMTPUser` 和 `$SMTPPass` 设置默认值为 `''`, 以应用环境变量 (**.env**) 值。

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/Email.php
- composer.json

从 4.2.12 升级到 4.3.0

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- *Composer* 版本
- 必备文件变更
  - *spark*
  - 配置文件
  - *composer.json*
- 重大变更
  - 数据库异常变化
  - 未捕获异常的 *HTTP* 状态码和退出码
  - *redirect()->withInput()* 和验证错误
  - 验证更改
  - *Time* 修复
  - 在测试中捕获 *STDERR* 和 *STDOUT* 流
  - 接口变化
  - 外键数据
- 重大增强
  - 支持多个域名
  - 数据库
  - *Honeypot* 和 *CSP*
  - 其它
- 项目文件
  - 内容更改

– 所有更改

Composer 版本

重要: 如果你使用 Composer,CodeIgniter v4.3.0 需要 Composer 2.0.14 或更高版本。

如果你使用的是更早版本的 Composer, 请升级你的 `composer` 工具, 删除 **vendor/** 目录, 并再次运行 `composer update`。

例如, 过程如下:

```
composer self-update
rm -rf vendor/
composer update
```

必备文件变更

spark

以下文件进行了重大更改, **你必须将更新后的版本**与应用程序合并:

- `spark`

重要: 如果不更新此文件, 在运行 `composer update` 后 Spark 命令将完全无法工作。

升级过程例如如下:

```
composer update
cp vendor/codeigniter4/framework/spark .
```

配置文件

app/Config/Kint.php

- **app/Config/Kint.php** 已更新为兼容 Kint 5.0。
- 你需要替换:
  - `Kint\Renderer\Renderer` 为 `Kint\Renderer\AbstractRenderer`
  - `Renderer::SORT_FULL` 为 `AbstractRenderer::SORT_FULL`

app/Config/Exceptions.php

- 如果你使用 PHP 8.2, 需要添加新的属性 `$logDeprecations` 和 `$deprecationLogLevel`。

模拟配置类

- 如果你在测试中使用以下模拟配置类, 需要更新 **app/Config** 中对应的配置文件:
  - `MockAppConfig` (`Config\App`)
  - `MockCLIConfig` (`Config\App`)
  - `MockSecurityConfig` (`Config\Security`)
- 在这些配置类中为属性添加 **类型**。你可能需要调整属性值以匹配属性类型。

composer.json

如果你手动安装了 CodeIgnter, 并使用 Composer, 你需要删除以下行, 并运行 `composer update`。

```
{
    ...
    "require": {
        ...
        "kint-php/kint": "^4.2",  <-- 移除此行
        ...
```

(续下页)

(接上页)

```
    },
    ...
    "scripts": {
        "post-update-cmd": [
            "CodeIgniter\\ComposerScripts::postUpdate"  <-- 移除此行
        ],
        "test": "phpunit"
    },
    ...
}
```

重大变更

数据库异常变化

- 当发生数据库错误时, 可能会改变异常类。如果你捕获了异常, 必须确认你的代码可以捕获这些异常。
- 现在即使 `CI_DEBUG` 为 false, 也会抛出一些异常。
- 在事务期间, 即使 `DBDebug` 为 true, 默认情况下也不会抛出异常。如果要抛出异常, 需要调用 `transException(true)`。参见抛出异常。
- 有关详细信息, 请参阅数据库错误时抛出的异常。

未捕获异常的 HTTP 状态码和退出码

- 如果你希望 **异常代码**作为 **HTTP 状态码**, 则 HTTP 状态码将会改变。在这种情况下, 需要在异常中实现 `HTTPExceptionInterface`。参见在异常中指定 *HTTP* 状态码。
- 如果你根据 **异常代码**期望 **退出码**, 则退出码将会改变。在这种情况下, 需要在异常中实现 `HasExitCodeInterface`。参见在异常中指定退出码。

redirect()->withInput() 和验证错误

redirect()->withInput() 和验证错误之前有一个未记录的行为。如果你使用 withInput() 重定向,CodeIgniter 会将验证错误存储在会话中, 并且你可以在重定向页面的验证对象中获取错误, 在执行新的验证之前:

```
// 在控制器中
if (! $this->validate($rules)) {
    return redirect()->back()->withInput();
}

// 在重定向页面的视图中
<?= service('Validation')->listErrors() ?>
```

这种行为是一个错误, 在 v4.3.0 中已修复。

如果你的代码依赖于此错误, 则需要更改代码。使用新的 Form 辅助函数, *validation_errors()*、*validation_list_errors()* 和*validation_show_error()* 来显示验证错误, 而不是 Validation 对象。

验证更改

- ValidationInterface 已更改。实现的类也应该添加方法和参数, 以免违反 LSP。有关详细信息, 请参阅*验证变更*。
- Validation::loadRuleGroup() 的返回值在 $group 为空时已从 null 改为 []。如果依赖于该行为, 请更新代码。

Time 修复

- 由于错误修复, *Time* 中的一些方法已从可变行为更改为不可变; Time 现在扩展 DateTimeImmutable。详细信息请参阅*ChangeLog*。
- 如果需要修改前 Time 的行为, 已添加了一个兼容的 TimeLegacy 类。请在应用程序代码中全部替换 Time 为 TimeLegacy。
- 但是 TimeLegacy 已被废弃。因此我们建议你更新代码。

例如:

```
// 之前
$time = Time::now();
// ...
if ($time instanceof DateTime) {
    // ...
}

// 之后
$time = Time::now();
// ...
if ($time instanceof DateTimeInterface) {
    // ...
}
```

```
// 之前
$time1 = new Time('2022-10-31 12:00');
$time2 = $time1->modify('+1 day');
echo $time1; // 2022-11-01 12:00:00
echo $time2; // 2022-11-01 12:00:00

// 之后
$time1 = new Time('2022-10-31 12:00');
$time2 = $time1->modify('+1 day');
echo $time1; // 2022-10-31 12:00:00
echo $time2; // 2022-11-01 12:00:00
```

在测试中捕获 STDERR 和 STDOUT 流

捕获错误和输出流的方式已更改。现在需要这样使用:

```
use CodeIgniter\Test\Filters\CITestStreamFilter;

protected function setUp(): void
{
    CITestStreamFilter::registration();
    CITestStreamFilter::addOutputFilter();
    CITestStreamFilter::addErrorFilter();
```

(续下页)

(接上页)

```
}

protected function tearDown(): void
{
    CITestStreamFilter::removeOutputFilter();
    CITestStreamFilter::removeErrorFilter();
}
```

而不是:

```
use CodeIgniter\Test\Filters\CITestStreamFilter;

protected function setUp(): void
{
    CITestStreamFilter::$buffer = '';
    $this->streamFilter         = stream_filter_append(STDOUT,
↪'CITestStreamFilter');
    $this->streamFilter         = stream_filter_append(STDERR,
↪'CITestStreamFilter');
}

protected function tearDown(): void
{
    stream_filter_remove($this->streamFilter);
}
```

或者使用 trait `CodeIgniter\Test\StreamFilterTrait`。参见*测试 CLI 输出*。

接口变化

一些接口已修复。详细信息请参阅*接口变更*。

外键数据

- `BaseConnection::getForeignKeyData()` 返回的数据结构已更改。你需要相应调整依赖此方法的任何代码, 以使用新的结构。

示例:`tableprefix_table_column1_column2_foreign`

返回的数据具有以下结构:

```
/**
 * @return array[
 *     {constraint_name} =>
 *         stdClass[
 *             'constraint_name'     => string,
 *             'table_name'          => string,
 *             'column_name'         => string[],
 *             'foreign_table_name'  => string,
 *             'foreign_column_name' => string[],
 *             'on_delete'           => string,
 *             'on_update'           => string,
 *             'match'               => string
 *         ]
 * ]
 */
```

重大增强

支持多个域名

- 如果设置了 `Config\App::$allowedHostnames`, 则当当前 URL 与其中一个匹配时, 像*base_url()*、*current_url()*、*site_url()* 这样的与 URL 相关的函数会返回带有 `Config\App::$allowedHostnames` 中设置的主机名的 URL。

数据库

- `CodeIgniter\Database\Database::loadForge()` 的返回类型已更改为 `Forge`。扩展类也应相应更改类型。
- `CodeIgniter\Database\Database::loadUtils()` 的返回类型已更改为 `BaseUtils`。扩展类也应相应更改类型。
- `BaseBuilder::updateBatch()` 的第二个参数 `$index` 已更改为 `$constraints`。它现在接受 array、string 或 `RawSql` 类型。扩展类也应相应更改类型。
- `BaseBuilder::insertBatch()` 和 `BaseBuilder::updateBatch()` 的 `$set` 参数现在接受单行数据的对象。扩展类也应相应更改类型。
- `BaseBuilder::_updateBatch()` 的第三个参数 `$index` 已更改为 `$values`, 参数类型已更改为 `array`。扩展类也应相应更改类型。
- 如果 `Model::update()` 方法生成不带 WHERE 子句的 SQL 语句, 现在会引发 `DatabaseException`。如果需要更新表中的所有记录, 请使用 Query Builder, 例如 `$model->builder()->update($data)`。

Honeypot 和 CSP

当启用 CSP 时, 会向 Honeypot 字段的容器标签中注入 id 属性 `id="hpc"`, 以隐藏该字段。如果视图中已经使用了该 id, 则需要用 `Config\Honeypot::$containerId` 更改它。并且可以在 `Config\Honeypot::$container` 中删除 `style="display:none"`。

其它

- **辅助函数:** 由于 `html_helper`、`form_helper` 或常用函数中的空 HTML 元素 (例如 `<input>`) 已默认更改为 HTML5 兼容, 如果你需要与 XHTML 兼容, 必须在 **app/Config/DocTypes.php** 中将 `$html5` 属性设置为 `false`。
- **CLI:** 由于从 `CodeIgniter\CodeIgniter` 中提取了 Spark 命令的启动, 如果 `Services::codeigniter()` 服务被覆盖, 运行这些命令时可能会出现问题。

项目文件

项目空间中的许多文件 (根目录、app、public、writable) 都已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改:在 Packagist 上探索。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

配置

- **app/Config/App.php**
  - 添加了新属性 `$allowedHostnames`, 用于在站点 URL 中设置主机名, 除了 `$baseURL` 中的主机名之外。参见多域名支持。
  - 属性 `$appTimezone` 已更改为 `UTC`, 以避免受夏令时的影响。
- **app/Config/Autoload.php**
  - 添加了新属性 `$helpers` 以自动加载辅助函数。
- **app/Config/Database.php**
  - `$default['DBDebug']` 和 `$test['DBDebug']` 默认更改为 `true`。参见数据库错误时抛出的异常。
- **app/Config/DocTypes.php**
  - 添加了属性 `$html5` 以确定是否移除空 HTML 元素 (如 `<input>`) 中的 solidus (/) 字符, 默认为 `true` 以实现 HTML5 兼容性。
- **app/Config/Encryption.php**
  - 添加了新属性 `$rawData`、`$encryptKeyInfo` 和 `$authKeyInfo` 以实现 CI3 加密兼容性。参见用于与 *CI3* 保持兼容性的配置。
- **app/Config/Exceptions.php**
  - 添加了两个新的公共属性:`$logDeprecations` 和 `$deprecationLogLevel`。详细信息请参阅记录弃用警告。
- **app/Config/Honeypot.php**

  - 添加了新属性 `$containerId` 以在启用 CSP 时设置容器标签的 id 属性值。
  - 属性 `$template` 中的值的 `input` 标签已更改为 HTML5 兼容。
- **app/Config/Logger.php**
  - 属性 `$threshold` 在非 `production` 环境中默认更改为 `9`。
- **app/Config/Modules.php**
  - 添加了新属性 `$composerPackages` 以限制 Composer 包自动发现, 提高性能。
- **app/Config/Routes.php**
  - 由于启动 Spark 命令的方式已更改, 不再需要加载框架的内部路由 (`SYSTEMPATH . 'Config/Routes.php'`)。
- **app/Config/Security.php**
  - 将属性 `$redirect` 的值更改为 `false`, 以防止 CSRF 检查失败时发生重定向。这可以更轻松地识别它是 CSRF 错误。
- **app/Config/Session.php**
  - 添加以处理 session 配置。
- **app/Config/Validation.php**
  - 默认验证规则已更改为严格规则, 以提高安全性。请参阅传统规则与严格规则。

视图文件

以下视图文件已更改为 HTML5 兼容标签。此外, 错误消息现在在 **Errors** 语言文件中定义。

- app/Views/errors/html/error_404.php
- app/Views/errors/html/error_exception.php
- app/Views/errors/html/production.php
- app/Views/welcome_message.php

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时。`Config` 类中的所有原子类型属性已加上类型:

- app/Config/App.php
- app/Config/Autoload.php
- app/Config/CURLRequest.php
- app/Config/Cache.php
- app/Config/ContentSecurityPolicy.php
- app/Config/Cookie.php
- app/Config/Database.php
- app/Config/DocTypes.php
- app/Config/Email.php
- app/Config/Encryption.php
- app/Config/Exceptions.php
- app/Config/Feature.php
- app/Config/Filters.php
- app/Config/Format.php
- app/Config/Generators.php
- app/Config/Honeypot.php
- app/Config/Images.php
- app/Config/Kint.php
- app/Config/Logger.php
- app/Config/Migrations.php
- app/Config/Mimes.php
- app/Config/Modules.php
- app/Config/Pager.php
- app/Config/Paths.php

- app/Config/Routes.php
- app/Config/Security.php
- app/Config/Session.php
- app/Config/Toolbar.php
- app/Config/UserAgents.php
- app/Config/Validation.php
- app/Views/errors/html/error_404.php
- app/Views/errors/html/error_exception.php
- app/Views/errors/html/production.php
- app/Views/welcome_message.php
- composer.json
- env
- phpunit.xml.dist
- spark

从 4.2.11 升级到 4.2.12

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *项目文件*
  - *所有更改*

项目文件

4.2.12 版本没有更改项目文件中的任何可执行代码。

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/Cache.php
- app/Config/Migrations.php
- app/Controllers/BaseController.php
- composer.json

从 4.2.10 升级到 4.2.11

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *重大变更*
  - *Config\App::$proxyIPs*
  - *Session 处理程序密钥更改*
- *项目文件*
  - *所有更改*

重大变更

Config\App::$proxyIPs

配置值格式已更改。现在你必须将代理 IP 地址和客户端 IP 地址的 HTTP 头名称设置为数组:

```
public $proxyIPs = [
        '10.0.1.200'     => 'X-Forwarded-For',
        '192.168.5.0/24' => 'X-Forwarded-For',
];
```

旧格式的配置值会抛出 `ConfigException`。

Session 处理程序密钥更改

DatabaseHandler 驱动程序、*MemcachedHandler 驱动程序* 和*RedisHandler 驱动程序* 的 session 数据记录的密钥已更改。因此, 如果使用这些 session 处理程序, 在升级后现有的 session 数据将失效。

- 使用 `DatabaseHandler` 时,session 表中的 `id` 列值现在包含 session cookie 名称 (`Config\App::$sessionCookieName`)。
- 使用 `MemcachedHandler` 或 `RedisHandler` 时, 密钥值包含 session cookie 名称 (`Config\App::$sessionCookieName`)。

`id` 列和 Memcached 密钥都有最大长度 (250 字节)。如果以下值超过那些最大长度,session 将无法正常工作。

- 使用 `DatabaseHandler` 时,session cookie 名称、分隔符和 session id(默认为 32 个字符) 的组合
- 使用 `MemcachedHandler` 时, 前缀 (`ci_session`)、session cookie 名称、分隔符和 session id 的组合

项目文件

4.2.11 版本没有更改项目文件中的任何可执行代码。

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/App.php
- app/Config/Autoload.php
- app/Config/Logger.php
- app/Config/Toolbar.php
- app/Views/welcome_message.php
- composer.json
- phpunit.xml.dist

从 4.2.9 升级到 4.2.10

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- 项目文件
  - 所有更改

项目文件

4.2.10 版本没有更改项目文件中的任何可执行代码。

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- composer.json

从 4.2.7 升级到 4.2.8

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *项目文件*
  - *所有更改*

项目文件

以下文件对代码或行为进行了改变, 建议在项目中更新:

- app/Views/errors/html/error_exception.php

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/Logger.php
- app/Views/errors/html/error_exception.php

从 4.2.6 升级到 4.2.7

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 重大变更
  - *set_cookie()*
  - 其他
- 项目文件
  - 所有更改

重大变更

set_cookie()

由于一个错误，之前版本的*set_cookie()* 和*CodeIgniter\HTTP\Response::setCookie()* 没有使用 `Config\Cookie` 中的 `$secure` 和 `$httponly` 值。即使在 `Config\Cookie` 中设置了 `$secure = true`，以下代码也不会发出带有安全标志的 Cookie:

```
helper('cookie');

$cookie = [
    'name'  => $name,
    'value' => $value,
];
set_cookie($cookie);
// 或者
$this->response->setCookie($cookie);
```

但是现在对于未指定的选项，会使用 `Config\Cookie` 中的值。如果在 `Config\Cookie` 中设置了 `$secure = true`，上面的代码现在会发出带有安全标志的 Cookie。

如果你的代码依赖于此错误, 请更改为显式指定必要的选项:

```
$cookie = [
    'name'     => $name,
    'value'    => $value,
    'secure'   => false, // 显式设置
    'httponly' => false, // 显式设置
];
set_cookie($cookie);
// 或者
$this->response->setCookie($cookie);
```

其他

- `Time::__toString()` 现在与本地设置无关。它在所有本地设置中都返回类似 '2022-09-07 12:00:00' 的与数据库兼容的字符串。大多数本地设置不受此更改的影响。但在一些如 *ar*、*fa* 的本地设置中, `Time::__toString()` (或 `(string) $time` 或隐式转换为字符串) 不再返回本地化的日期时间字符串。如果你想获取本地化的日期时间字符串, 请使用 *Time::toDateTimeString()*。
- 当验证带星号 (*) 的字段时, 验证规则 `required_without` 的逻辑已更改为单独验证每个数组项, 并且规则方法的方法签名也已更改。扩展类应相应地更新参数, 以免违反 LSP。

项目文件

4.2.7 版本没有更改项目文件中的任何可执行代码。

所有更改

这是 **项目空间** 中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Common.php

从 4.2.5 升级到 4.2.6

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 项目文件
  - 所有更改

项目文件

项目空间中的一些文件 (根目录、app、public、writable) 收到了视觉优化的更新。你完全不需要碰这些文件。有一些第三方 CodeIgniter 模块可以帮助合并项目空间的更改: 在 Packagist 上探索。

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/App.php
- app/Config/ContentSecurityPolicy.php
- app/Config/Routes.php
- app/Config/Validation.php

从 4.2.3 升级到 4.2.5

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 项目文件

项目文件

4.2.5 版本没有更改任何项目文件。

从 4.2.2 升级到 4.2.3

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 项目文件

项目文件

4.2.3 版本是出于安全考虑的内部变更, 项目中不需要任何干预。

从 4.2.1 升级到 4.2.2

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 重大变更
  - 网页缓存错误修复
  - 其它
- 重大增强

- 项目文件
  - 内容更改
  - 所有更改

重大变更

网页缓存错误修复

- 网页缓存 现在会在后置过滤器 执行后缓存 Response 数据。
- 例如, 如果你启用*SecureHeaders*, 那么从缓存中获取页面时现在也会发送 Response 头。

重要: 如果你编写了 **基于此错误的代码**, 假定 "after" 过滤器中的 Response 更改不会被缓存, 那么 **敏感信息可能会被缓存并泄露**。如果是这种情况, 请更改代码以禁用对页面的缓存。

其它

- `Forge::createTable()` 方法不再执行 `CREATE TABLE IF NOT EXISTS`。当 `$ifNotExists` 为 true 时, 如果在 `$db->tableExists($table)` 中未找到表, 则执行 `CREATE TABLE`。
- `Forge::_createTable()` 的第二个参数 `$ifNotExists` 已被废弃。它不再被使用, 将在未来版本中移除。
- 当使用*random_string()* 的第一个参数为 `'crypto'` 时, 现在如果把第二个参数 `$len` 设置为奇数, 会抛出 `InvalidArgumentException`。请将参数改为偶数。

重大增强

项目文件

项目空间中的许多文件 (根目录、app、public、writable) 都已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改: 在 Packagist 上探索。

备注: 除非极少数情况进行错误修复, 否则对项目空间文件的任何更改都不会破坏你的应用程序。在下一个主要版本之前, 这里注明的所有更改都是可选的, 强制性更改将在上面部分介绍。

内容更改

- app/Views/errors/html/error_404.php
- app/Views/welcome_message.php
- public/index.php
- spark

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/App.php
- app/Config/Constants.php
- app/Config/Logger.php
- app/Config/Paths.php
- app/Views/errors/html/error_404.php
- app/Views/welcome_message.php

从 4.2.0 升级到 4.2.1

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *必备文件变更*
  - *app/Config/Mimes.php*
- *重大变更*
  - *get_cookie()*
- *重大增强*
- *项目文件*
  - *内容更改*
  - *所有更改*

必备文件变更

app/Config/Mimes.php

- **app/Config/Mimes.php** 中文件扩展名与 MIME 类型的映射已更新以修复一个错误。同时, `Mimes::getExtensionFromType()` 的逻辑也已改变。

重大变更

get_cookie()

如果存在一个带前缀的 cookie 和一个同名但不带前缀的 cookie, 之前的 `get_cookie()` 有一种棘手的行为, 它会返回不带前缀的 cookie。

例如, 当 `Config\Cookie::$prefix` 为 `prefix_` 时, 存在两个 cookie:`test` 和 `prefix_test`:

```
$_COOKIES = [
    'test'        => '非 CI Cookie',
    'prefix_test' => 'CI Cookie',
];
```

以前, `get_cookie()` 返回如下:

```
get_cookie('test');        // 返回 "非 CI Cookie"
get_cookie('prefix_test'); // 返回 "CI Cookie"
```

现在该行为已被修复为一个错误, 并进行了如下改变:

```
get_cookie('test');              // 返回 "CI Cookie"
get_cookie('prefix_test');       // 返回 null
get_cookie('test', false, null); // 返回 "非 CI Cookie"
```

如果你依赖之前的行为, 则需要更改代码。

备注: 在上面的例子中, 如果只有一个 cookie `prefix_test`, 之前的 `get_cookie('test')` 也会返回 `"CI Cookie"`。

重大增强

项目文件

项目空间中的许多文件 (根目录、app、public、writable) 都已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改: 在 Packagist 上探索。

备注: 除非极少数情况进行错误修复, 否则对项目空间文件的任何更改都不会破坏你的应用程序。在下一个主要版本之前, 这里注明的所有更改都是可选的, 强制性更改将在上面部分介绍。

内容更改

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/Mimes.php

从 4.1.9 升级到 4.2.0

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *必备文件变更*
  - *index.php 和 spark*
  - *Config/Constants.php*
  - *composer.json*
- *重大变更*
- *重大增强*
- *项目文件*
  - *内容更改*
  - *所有更改*

必备文件变更

index.php 和 spark

以下文件进行了重大更改, **你必须将更新后的版本**与应用程序合并:

- `public/index.php`
- `spark`

重要: 如果你不更新以上两个文件, 在运行 `composer update` 后 CodeIgniter 将完全无法工作。

升级过程例如如下:

```
composer update
cp vendor/codeigniter4/framework/public/index.php public/index.php
cp vendor/codeigniter4/framework/spark .
```

Config/Constants.php

常量 `EVENT_PRIORITY_LOW`、`EVENT_PRIORITY_NORMAL` 和 `EVENT_PRIORITY_HIGH` 已被废弃, 定义移至 `app/Config/Constants.php`。如果你使用这些常量, 请在 `app/Config/Constants.php` 中定义它们。或者使用新的类常量 `CodeIgniter\Events\Events::PRIORITY_LOW`、`CodeIgniter\Events\Events::PRIORITY_NORMAL` 和 `CodeIgniter\Events\Events::PRIORITY_HIGH`。

composer.json

备注: 此步骤在 v4.5.0 或更高版本中不再需要。

如果你使用 Composer, 在安装 CodeIgniter v4.1.9 或更早版本时, 如果 `/composer.json` 的 `autoload.psr-4` 中存在类似下面的 `App\\` 和 `Config\\` 命名空间, 你需要删除这些行并运行 `composer dump-autoload`。

```
{
    ...
    "autoload": {
        "psr-4": {
            "App\\": "app",              <-- 移除此行
            "Config\\": "app/Config"     <-- 移除此行
        }
    },
    ...
}
```

重大变更

- `system/bootstrap.php` 文件不再返回 `CodeIgniter` 实例, 也不再加载 `.env` 文件 (现在在 `index.php` 和 `spark` 中处理)。如果你的代码依赖这些行为则不再起作用, 必须进行修改。这已更改是为了更易实现 预加载。

重大增强

- `Validation::setRule()` 的方法签名已更改。`$rules` 参数上的 `string` 类型提示已移除。扩展类同样应移除参数类型声明, 以避免违反 LSP。
- `CodeIgniter\Database\BaseBuilder::join()` 和 `CodeIgniter\Database\*\Builder::join()` 的方法签名已更改。`$cond` 参数上的 `string` 类型提示已移除。扩展类同样应移除参数类型声明, 以避免违反 LSP。

项目文件

项目空间中的许多文件 (根目录、app、public、writable) 都已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改: 在 Packagist 上探索。

备注: 除非极少数情况进行错误修复, 否则对项目空间文件的任何更改都不会破坏你的应用程序。在下一个主要版本之前, 这里注明的所有更改都是可选的, 强制性更改将在上面部分介绍。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

- **`app/Config/Routes.php`**
  - 为了使默认配置更安全, 默认情况下自动路由已更改为禁用。

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- app/Config/App.php
- app/Config/Constants.php
- app/Config/ContentSecurityPolicy.php
- app/Config/Database.php
- app/Config/Events.php
- app/Config/Feature.php
- app/Config/Filters.php
- app/Config/Format.php
- app/Config/Logger.php
- app/Config/Mimes.php
- app/Config/Publisher.php
- app/Config/Routes.php
- app/Config/Security.php
- app/Config/Validation.php
- app/Config/View.php
- app/Controllers/BaseController.php
- app/Views/errors/html/debug.css
- app/Views/errors/html/debug.js
- app/Views/errors/html/error_404.php

- app/Views/errors/html/error_exception.php
- app/Views/errors/html/production.php
- app/Views/welcome_message.php
- app/index.html
- preload.php
- public/index.php
- spark

从 4.1.7 升级到 4.1.8

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *重大变更*

重大变更

- 由于 `API\ResponseTrait` 中的一个安全问题, 所有 trait 方法的作用域现在都被限定为 `protected`。更多信息请参阅 安全公告 GHSA-7528-7jg5-6g62。

从 4.1.6 升级到 4.1.7

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- 重大变更

重大变更

- 当 `$xssClean` 为 true 时, `get_cookie()` 改变了输出。现在使用 `FILTER_SANITIZE_FULL_SPECIAL_CHARS`, 而不是 `FILTER_SANITIZE_STRING`。确保更改可以接受。请注意, 使用 XSS 过滤是一种不好的做法。它不能完美地防止 XSS 攻击。建议在视图中使用正确的 `$context` 来 `esc()`。

从 4.1.5 升级到 4.1.6

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 重大变更
  - 验证结果变化
- 重大增强
- 项目文件
  - 内容更改
  - 所有更改

重大变更

验证结果变化

由于一个错误修复, 现在验证数组项时, 验证可能会改变验证结果 (参见更新日志)。所以请检查所有验证数组的验证结果代码。像 `contacts.*.name` 这样验证多个字段不受影响。

如果你有以下表单:

```
<input type='text' name='invoice_rule[1]'>
<input type='text' name='invoice_rule[2]'>
```

并且你有以下验证规则:

```
'invoice_rule' => ['rules' => 'numeric', 'errors' => ['numeric' =>
↪'Not numeric']]
```

将规则键改为 `invoice_rule.*` 这样验证就可以工作了。

重大增强

无。

项目文件

项目空间中的许多文件 (根目录、app、public、writable) 都已更新。由于这些文件超出 **系统**范围, 如果不进行干预, 它们将不会更改。有一些第三方 CodeIgniter 模块可以协助合并项目空间的更改: 在 Packagist 上探索。

备注: 除非极少数情况进行错误修复, 否则对项目空间文件的任何更改都不会破坏你的应用程序。在下一个主要版本之前, 这里注明的所有更改都是可选的, 强制性更改将在上面部分介绍。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

- `app/Config/Filters.php`
- `app/Config/Mimes.php`
- `app/Config/Security.php`
- `app/Config/Toolbar.php`

所有更改

这是 **项目空间**中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- `app/Config/Filters.php`
- `app/Config/Mimes.php`
- `app/Config/Security.php`
- `app/Config/Toolbar.php`
- `app/Views/errors/html/error_exception.php`

从 4.1.4 升级到 4.1.5

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *重大变更*
  - *BaseBuilder 和 Model 类中 set() 方法的变化*
  - *Session DatabaseHandler 的数据库表更改*
  - *CSRF 保护*
  - *CURLRequest 头更改*
  - *Query Builder 更改*
- *重大增强*
  - *为路由设置多个过滤器*
- *项目文件*
  - *内容更改*
  - *所有更改*

重大变更

BaseBuilder 和 Model 类中 set() 方法的变化

已移除 `set()` 方法中 `$value` 参数的强制转换, 以修复一个将数组和字符串参数传递给 `set()` 方法时会有不同处理的 bug。如果你扩展了 `BaseBuilder` 类或 `Model` 类并修改了 `set()` 方法, 则需要将其定义从 `public function set($key, ?string $value = '', ?bool $escape = null)` 改为 `public function set($key, $value = '', ?bool $escape = null)`。

Session DatabaseHandler 的数据库表更改

为了优化,session 表中的以下列的类型发生了改变:

- **MySQL**
  - `timestamp`
- **PostgreSQL**
  - `ip_address`
  - `timestamp`
  - `data`

请更新 session 表的定义。参见*Session 库* 查看新的定义。

此更改在 v4.1.2 中引入。但由于 一个错误, DatabaseHandler 驱动程序无法正常工作。

CSRF 保护

由于一个错误修复, 当应用 CSRF 过滤器时,CSRF 保护现在不仅适用于 **POST** 请求, 也适用于 **PUT/PATCH/DELETE** 请求。

当你使用 **PUT/PATCH/DELETE** 请求时, 你需要发送 CSRF token。或者如果你不需要为它们提供 CSRF 保护, 可以为这些请求移除 CSRF 过滤器。

如果你想要与先前版本相同的行为, 可以在 **app/Config/Filters.php** 中像下面这样设置 CSRF 过滤器:

```php
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public $methods = [
        'GET'  => ['csrf'],
        'POST' => ['csrf'],
    ];
    // ...
}
```

只有在你使用 `form_open()` 自动生成 CSRF 字段时才需要保护 **GET** 方法。

> **警告:** 一般来说, 如果你使用 `$methods` 过滤器, 你应该*禁用自动路由 (传统)*, 因为*自动路由（传统版）* 允许任意 HTTP 方法访问一个控制器。用你不期望的方法访问控制器可能会绕过过滤器。

CURLRequest 头更改

在以前的版本中, 如果你没有提供自己的 header, `CURLRequest` 会发送来自浏览器的请求 header。这个错误已被修复。如果你的请求依赖 header, 升级后你的请求可能会失败。在这种情况下, 需要手动添加必要的 header。参见*CURLRequest 类* 了解如何添加。

Query Builder 更改

为了优化和修复一个错误, 主要用于测试的以下行为已经改变。

- 当你使用 `insertBatch()` 和 `updateBatch()` 时, `$query->getOriginalQuery()` 的返回值已改变。它不再返回带有绑定参数的查询, 而是返回实际运行的查询。

- 如果 `testMode` 为 `true`, `insertBatch()` 将返回 SQL 字符串数组, 而不是受影响的行数。此更改是为了使返回的数据类型与 `updateBatch()` 方法相同。

重大增强

为路由设置多个过滤器

一个为路由设置多个过滤器的新功能。

重要: 默认情况下, 此功能是禁用的。因为它破坏了向后兼容性。

如果要使用它, 需要在 `app/Config/Feature.php` 中将 `$multipleFilters` 属性设置为 `true`。如果启用它:

- **`CodeIgniter\CodeIgniter::handleRequest()` 使用**
  - `CodeIgniter\Filters\Filters::enableFilters()`, 而不是 `enableFilter()`
- **`CodeIgniter\CodeIgniter::tryToRouteIt()` 使用**
  - `CodeIgniter\Router\Router::getFilters()`, 而不是 `getFilter()`
- **`CodeIgniter\Router\Router::handle()` 使用**
  - 属性 `$filtersInfo`, 而不是 `$filterInfo`
  - `CodeIgniter\Router\RouteCollection::getFiltersForRoute()`, 而不是 `getFilterForRoute()`

如果你扩展了上述类, 则需要更改它们。

以下方法和属性已被废弃:

- `CodeIgniter\Filters\Filters::enableFilter()`
- `CodeIgniter\Router\Router::getFilter()`
- `CodeIgniter\Router\RouteCollection::getFilterForRoute()`
- `CodeIgniter\Router\RouteCollection` 的属性 `$filterInfo`

有关功能的信息, 请参阅应用过滤器。

项目文件

项目空间 (根目录、app、public、writable) 中的许多文件都已更新。由于这些文件超出系统范围, 如果不进行干预, 它们将不会更改。有一些第三方 CodeIgniter 模块可用于帮助合并项目空间中的更改: 在 Packagist 上探索。

备注: 除非极少数情况进行错误修复, 否则对项目空间文件的任何更改都不会破坏你的应用程序。在下一个主要版本之前, 这里注明的所有更改都是可选的, 强制性更改将在上面部分介绍。

内容更改

以下文件已作出重大更改 (包括弃用或视觉调整), 建议你将更新版本与应用程序合并:

- `app/Config/CURLRequest.php`
- `app/Config/Cache.php`
- `app/Config/Feature.php`
- `app/Config/Generators.php`
- `app/Config/Publisher.php`
- `app/Config/Security.php`
- `app/Views/welcome_message.php`

所有更改

这是项目空间中已更改的所有文件的列表; 其中许多仅为注释或格式更改, 不会影响运行时:

- `app/Config/CURLRequest.php`
- `app/Config/Cache.php`
- `app/Config/Feature.php`
- `app/Config/Generators.php`
- `app/Config/Kint.php`

- app/Config/Publisher.php
- app/Config/Security.php
- app/Views/welcome_message.php

从 4.1.3 升级到 4.1.4

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 重大变更
  - 方法作用域
- 项目文件

此版本专注于代码风格。所有更改 (除下面注明的外) 都是为了使代码符合新的 CodeIgniter 编码标准 (基于 PSR-12)。

重大变更

方法作用域

以下方法的作用域从 public 改为 protected, 以匹配其父类方法并更好地与其用法保持一致。如果你依赖任何这些方法是 public 的 (极少可能), 请相应调整你的代码:

- CodeIgniter\Database\MySQLi\Connection::execute()
- CodeIgniter\Database\MySQLi\Connection::_fieldData()
- CodeIgniter\Database\MySQLi\Connection::_indexData()
- CodeIgniter\Database\MySQLi\Connection::_foreignKeyData()
- CodeIgniter\Database\Postgre\Builder::_like_statement()
- CodeIgniter\Database\Postgre\Connection::execute()
- CodeIgniter\Database\Postgre\Connection::_fieldData()

- `CodeIgniter\Database\Postgre\Connection::_indexData()`
- `CodeIgniter\Database\Postgre\Connection::_foreignKeyData()`
- `CodeIgniter\Database\SQLSRV\Connection::execute()`
- `CodeIgniter\Database\SQLSRV\Connection::_fieldData()`
- `CodeIgniter\Database\SQLSRV\Connection::_indexData()`
- `CodeIgniter\Database\SQLSRV\Connection::_foreignKeyData()`
- `CodeIgniter\Database\SQLite3\Connection::execute()`
- `CodeIgniter\Database\SQLite3\Connection::_fieldData()`
- `CodeIgniter\Database\SQLite3\Connection::_indexData()`
- `CodeIgniter\Database\SQLite3\Connection::_foreignKeyData()`
- `CodeIgniter\Images\Handlers\GDHandler::_flatten()`
- `CodeIgniter\Images\Handlers\GDHandler::_flip()`
- `CodeIgniter\Images\Handlers\ImageMagickHandler::_flatten()`
- `CodeIgniter\Images\Handlers\ImageMagickHandler::_flip()`
- `CodeIgniter\Test\Mock\MockIncomingRequest::detectURI()`
- `CodeIgniter\Test\Mock\MockSecurity.php::sendCookie()`

项目文件

项目空间中的所有文件都使用新的编码风格进行了重新格式化。这不会影响现有代码，但是你可能希望将更新的编码风格应用于自己的项目，以使它们与这些文件的框架版本保持一致。

从 4.1.2 升级到 4.1.3

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- 重大增强
  - 缓存 TTL
- 项目文件

重大增强

缓存 TTL

在 **app/Config/Cache.php** 中有一个新的值:`$ttl`。这不会被框架中的处理程序使用, 其中硬编码为 60 秒, 但对项目和模块可能很有用。在未来的版本中, 此值将替换硬编码的版本, 因此请将其保留为 `60`, 或者停止依赖硬编码的版本。

项目文件

项目空间 (root、app、public、writable) 中的只有少数文件收到了更新。由于这些文件超出系统范围, 如果不进行干预, 它们将不会更改。以下文件已被更改, 建议你将更新后的版本与应用程序合并:

- `app/Config/Cache.php`
- `spark`

从 4.1.1 升级到 4.1.2

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 重大变更
  - *current_url()* 和 *indexPage*
  - 缓存键

- *BaseConnection::query()* 的返回值
- 重大增强
  - 添加了 *ConnectionInterface::isWriteType()* 声明
  - 测试 *Traits*
  - 测试响应
- 项目文件
  - 内容更改
  - 所有更改

重大变更

current_url() 和 indexPage

由于 一个 bug 导致 `current_url()` 的结果可能与项目配置不匹配, 最重要的是:`indexPage` 不会被包含。使用 `App::$indexPage` 的项目应该期望 `current_url()` 及所有依赖它的内容 (包括响应测试、分页器、表单辅助函数、分页器和视图解析器) 的值发生改变。请相应更新你的项目。

缓存键

缓存处理程序在键的兼容性方面差异很大。更新的缓存驱动现在会通过验证传递所有键, 大致匹配 PSR-6 的建议:

> 一个至少由一个字符组成的字符串, uniquely identifies a cached item. 实现库 MUST 支持由字符 A-Z、a-z、0-9、_ 和. 组成的键, 可以任意顺序, 使用 UTF-8 编码, 长度上限为 64 个字符。实现库 MAY 支持额外的字符和编码或者更长的长度, 但必须至少支持那个最小要求。库自己负责根据需要对键字符串进行转义, 但必须能够返回原始未修改的键字符串。以下保留字符是为未来扩展而预留的, 实现库 MUST NOT 支持: `{}()/\@:`

请更新项目以删除任何无效的缓存键。

BaseConnection::query() 的返回值

之前版本中的 `BaseConnection::query()` 方法错误地总是返回 BaseResult 对象, 即使查询失败。该方法现在对失败的查询会返回 `false` (如果 `DBDebug` 为 `true` 则会抛出异常), 对写类型的查询会返回布尔值。请检查 `query()` 方法的任何使用, 评估返回值是否可能是布尔类型而不是 Result 对象。要更好地了解什么查询是写类型的查询, 请查看 `BaseConnection::isWriteType()` 和相关 Connection 类中任何特定于 DBMS 的 `isWriteType()` 覆盖。

重大增强

添加了 ConnectionInterface::isWriteType() 声明

如果你编写了任何实现 ConnectionInterface 的类, 现在必须实现 `isWriteType()` 方法, 声明为 `public function isWriteType($sql): bool`。如果你的类扩展了 BaseConnection, 那么该类将提供一个基本的 `isWriteType()` 方法, 你可能想覆盖它。

测试 Traits

`CodeIgniter\Test` 命名空间进行了大量改进, 以帮助开发人员自己的测试用例。最显著的是测试扩展移至 Traits 以使它们更易于在各种测试用例需求之间进行选择。`CIDatabaseTestCase` 和 `FeatureTestCase` 类已被废弃, 它们的方法分别移至 `DatabaseTestTrait` 和 `FeatureTestTrait`。请更新测试用例以扩展主要测试用例并使用任何所需的 traits。例如:

```
<?php

use CodeIgniter\Test\DatabaseTestCase;

class MyDatabaseTest extends DatabaseTestCase
{
    public function testBadRow()
    {
        // ...
    }
}
```

…变为:

```
<?php

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\DatabaseTestTrait;

class MyDatabaseTest extends CIUnitTestCase
{
    use DatabaseTestTrait;

    public function testBadRow()
    {
        // ...
    }
}
```

最后, `ControllerTester` 已被 `ControllerTestTrait` 取代, 以统一方法并利用更新的响应测试 (见下文)。

测试响应

用于测试响应的工具已经进行了合并和改进。新的 `TestResponse` 替换了 `ControllerResponse` 和 `FeatureResponse`, 提供了两个类的完整方法和属性集。在大多数情况下, 这些更改将由 `ControllerTestTrait` 和 `FeatureTestCase` "幕后" 处理, 但要注意两点:

- `TestResponse` 的 `$request` 和 `$response` 属性是受保护的, 只能通过它们的 getter 方法 `request()` 和 `response()` 访问
- `TestResponse` 没有 `getBody()` 和 `setBody()` 方法, 而是直接使用 Response 方法, 例如:`$body = $result->response()->getBody();`

项目文件

项目空间 (root、app、public、writable) 中的许多文件都已更新。由于这些文件超出系统范围, 如果不进行干预, 它们将不会更改。有一些第三方 CodeIgniter 模块可用于帮助合并项目空间中的更改: 在 Packagist 上探索。

备注: 除了极少数的错误修复情况外, 对项目空间文件的任何更改都不会破坏你的应用

程序。直到下一个主版本之前,这里注明的所有更改都是可选的,任何强制性更改都将在上面的部分中介绍。

内容更改

以下文件收到了显着更改 (包括不推荐使用或视觉调整), 建议你将更新版本与应用程序合并:

- `app/Config/App.php`
- `app/Config/Autoload.php`
- `app/Config/Cookie.php`
- `app/Config/Events.php`
- `app/Config/Exceptions.php`
- `app/Config/Security.php`
- `app/Views/errors/html/*`
- `env`
- `spark`

所有更改

这是项目空间中收到更改的所有文件的列表; 其中许多只是注释或格式更改, 不会对运行时产生影响:

- `app/Config/App.php`
- `app/Config/Autoload.php`
- `app/Config/ContentSecurityPolicy.php`
- `app/Config/Cookie.php`
- `app/Config/Events.php`
- `app/Config/Exceptions.php`
- `app/Config/Logger.php`
- `app/Config/Mimes.php`

- app/Config/Modules.php
- app/Config/Security.php
- app/Controllers/BaseController.php
- app/Views/errors/html/debug.css
- app/Views/errors/html/error_404.php
- app/Views/errors/html/error_exception.php
- app/Views/welcome_message.php
- composer.json
- contributing/guidelines.rst
- env
- phpstan.neon.dist
- phpunit.xml.dist
- public/.htaccess
- public/index.php
- rector.php
- spark

从 4.0.5 升级到 4.1.0 或 4.1.1

请参考与你的安装方法相对应的升级说明。

- 通过 *Composer* 安装应用启动器升级
- 通过 *Composer* 安装到现有项目升级
- 手动安装升级

- 重大变更
  - 传统自动加载

重大变更

传统自动加载

`Autoloader::loadLegacy()` 方法原本是为过渡到 CodeIgniter v4 而设计的。自 v4.1.0 开始, 此支持已被移除。所有类必须采用命名空间。

从 4.0.4 升级到 4.0.5

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- 重大增强
  - *Cookie SameSite 支持*
  - *Message::getHeader(s)*
  - *ResponseInterface*
  - *Config\Services*
- 项目文件
  - 内容更改
  - 所有更改

重大增强

Cookie SameSite 支持

CodeIgniter 4.0.5 引入了 cookie 的 SameSite 属性设置。先前版本没有设置此属性。cookie 的默认设置现在是 'Lax'。这将影响 cookie 在跨域环境中的处理, 你可能需要在项目中调整此设置。在 **app/Config/App.php** 中为响应 cookie 和 CSRF cookie 分别存在独立设置。

有关详细信息, 请参阅 MDN Web 文档。SameSite 规范描述在 RFC 6265 和 RFC 6265bis 修订版 中。

Message::getHeader(s)

HTTP 层正在向 PSR-7 兼容迈进。为此, `Message::getHeader()` 和 `Message::getHeaders()` 已被废弃, 应分别替换为 `Message::header()` 和 `Message::headers()`。请注意, 这也涉及到所有扩展 `Message` 的类:`Request`、`Response` 及其子类。

来自 HTTP 层的其他相关废弃:

- `Message::isJSON()`: 直接检查 "Content-Type" 头
- `Request[Interface]::isValidIP()`: 使用 Validation 类及 `valid_ip`
- `Request[Interface]::getMethod()`: 将删除 `$upper` 参数, 使用 strtoupper()
- `Request[Trait]::$ipAddress`: 该属性将变为私有
- `Request::$proxyIPs`: 该属性将被删除; 直接访问 `config('App')->proxyIPs`
- `Request::__construct()`: 构造函数不再接收 `Config\App` , 已变为可空以方便过渡
- `Response[Interface]::getReason()`: 请使用 `getReasonPhrase()`
- `Response[Interface]::getStatusCode()`: 将删除显式的 `int` 返回类型 (无需操作)

ResponseInterface

该接口旨在包括任何框架兼容的响应类所需的方法。缺少许多框架所需的方法, 现已添加。如果你使用任何直接实现 `ResponseInterface` 的类, 它们需要与更新后的要求兼容。这些方法如下:

- `setLastModified($date)`
- `setLink(PagerInterface $pager)`
- `setJSON($body, bool $unencoded = false)`
- `getJSON()`
- `setXML($body)`
- `getXML()`

- `send()`
- `sendHeaders()`
- `sendBody()`
- `setCookie($name, $value = '', $expire = '', $domain = '', $path = '/', $prefix = '', $secure = false, $httponly = false, $samesite = null)`
- `hasCookie(string $name, string $value = null, string $prefix = ''): bool`
- `getCookie(string $name = null, string $prefix = '')`
- `deleteCookie(string $name = '', string $domain = '', string $path = '/', string $prefix = '')`
- `getCookies()`
- `redirect(string $uri, string $method = 'auto', int $code = null)`
- `download(string $filename = '', $data = '', bool $setMime = false)`

为方便使用此接口, 这些方法已从框架的 `Response` 移至 `ResponseTrait` 中, 你可以使用它, `DownloadResponse` 现在直接扩展 `Response` 以确保最大兼容性。

Config\Services

服务发现已更新, 允许第三方服务 (在通过 Modules 启用时) 优先于核心服务。请更新 **app/Config/Services.php**, 使类扩展 `CodeIgniter\Config\BaseService` 以允许正确发现第三方服务。

项目文件

项目空间 (根目录、app、public、writable) 中的许多文件都已更新。由于这些文件超出系统范围, 如果不进行干预, 它们将不会更改。有一些第三方 CodeIgniter 模块可用于帮助合并项目空间中的更改: 在 Packagist 上探索。

备注: 除了极少数的错误修复情况外, 对项目空间文件的任何更改都不会破坏你的应用

程序。直到下一个主版本之前,这里注明的所有更改都是可选的,任何强制性更改都将在上面的部分中介绍。

内容更改

建议你将更新版本与应用程序合并,因为以下文件收到显着更改 (包括不推荐使用或视觉调整):

- `app/Views/*`
- `public/index.php`
- `public/.htaccess`
- `spark`
- `phpunit.xml.dist`
- `composer.json`

所有更改

这是项目空间中已更改的所有文件的列表;其中许多只是注释或格式更改,不会对运行时产生影响:

- `LICENSE`
- `README.md`
- `app/Config/App.php`
- `app/Config/Autoload.php`
- `app/Config/Boot/development.php`
- `app/Config/Boot/production.php`
- `app/Config/Boot/testing.php`
- `app/Config/Cache.php`
- `app/Config/Constants.php`
- `app/Config/ContentSecurityPolicy.php`
- `app/Config/Database.php`

- app/Config/DocTypes.php
- app/Config/Email.php
- app/Config/Encryption.php
- app/Config/Events.php
- app/Config/Exceptions.php
- app/Config/Filters.php
- app/Config/ForeignCharacters.php
- app/Config/Format.php
- app/Config/Generators.php
- app/Config/Honeypot.php
- app/Config/Images.php
- app/Config/Kint.php
- app/Config/Logger.php
- app/Config/Migrations.php
- app/Config/Mimes.php
- app/Config/Modules.php
- app/Config/Pager.php
- app/Config/Paths.php
- app/Config/Routes.php
- app/Config/Security.php
- app/Config/Services.php
- app/Config/Toolbar.php
- app/Config/UserAgents.php
- app/Config/Validation.php
- app/Config/View.php
- app/Controllers/BaseController.php
- app/Controllers/Home.php

- app/Views/errors/cli/error_404.php
- app/Views/errors/cli/error_exception.php
- app/Views/errors/html/debug.css
- app/Views/errors/html/debug.js
- app/Views/errors/html/error_exception.php
- composer.json
- env
- license.txt
- phpunit.xml.dist
- public/.htaccess
- public/index.php
- spark

从 4.0.x 升级到 4.0.4

请参考与你的安装方法相对应的升级说明。

- *通过 Composer 安装应用启动器升级*
- *通过 Composer 安装到现有项目升级*
- *手动安装升级*

- *重大变更*
  - *更新 FilterInterface 声明*

CodeIgniter 4.0.4 修复了控制器过滤器实现中的一个 bug, 破坏了遵循 FilterInterface 的代码。

重大变更

更新 FilterInterface 声明

`after()` 和 `before()` 方法签名必须更新为包含 `$arguments`。函数定义应从:

```
public function before(RequestInterface $request)
public function after(RequestInterface $request, ResponseInterface
↪$response)
```

更改为:

```
public function before(RequestInterface $request, $arguments = null)
public function after(RequestInterface $request, ResponseInterface
↪$response, $arguments = null)
```

从 3.x 升级到 4.x

CodeIgniter 4 是框架的重写, 并且不向后兼容。将你的应用程序转换更合适, 而不是升级它。一旦你完成了转换, 从 CodeIgniter 4 的一个版本升级到下一个版本将很简单。

"精简、敏捷、简单" 的理念仍然保留, 但实现与 CodeIgniter 3 有很多不同。

升级没有 12 步检查表。相反, 请在一个新的项目文件夹中使用 CodeIgniter 4 的副本开始, 选择你希望的安装和使用方式, 然后转换和集成你的应用组件。我们将尽量指出这里最重要的注意事项。

为了升级你的项目, 我们总结出两项主要工作。首先, 有一些对每个项目都很重要的一般调整, 必须处理。其次是 CodeIgniter 构建的库, 包含一些最重要的函数。这些库可以互相独立工作, 所以你必须一一查看它们。

在启动项目转换之前, 请阅读用户指南!

- 一般调整
  - 下载
  - 命名空间
  - 应用程序结构
  - 路由

  - 模型、视图和控制器
  - 核心类更改
  - 类加载
  - 库
  - 辅助函数
  - 钩子
  - 错误处理
  - 扩展框架
- 升级库

一般调整

下载

- CI4 仍以*ready-to-run 压缩包或 tarball* 形式提供。
- 它也可以使用*Composer* 安装。

命名空间

- CI4 是为 PHP 8.1+ 构建的, 框架中的所有内容都使用了命名空间, 除了 helper 和 lang 文件。

应用程序结构

重要: index.php 不再位于项目的根目录！为了更好的安全性和组件分离，它已被移到 **public** 文件夹内。

这意味着你需要配置你的 Web 服务器指向你项目的 **public** 文件夹，而不是项目根目录。

如果你使用共享主机，参见部署到共享主机服务。

- **application** 文件夹重命名为 **app**, 框架仍然有 **system** 文件夹, 与以前的解释相同。

- 框架现在提供了 **public** 文件夹, 旨在作为你的应用程序的文档根目录。
- `defined('BASEPATH') OR exit('No direct script access allowed');` 这一行不是必需的, 因为在默认配置下, **public** 文件夹之外的文件不可访问。并且 CI4 不再定义常量 `BASEPATH`, 所以在所有文件中删除该行。
- 还有一个 **writable** 文件夹, 用于保存缓存数据、日志和 session 数据。
- **app** 文件夹与 CI3 的 **application** 非常相似, 只是一些名称更改, 一些子文件夹移到了 **writable** 文件夹。
- 不再有嵌套的 **application/core** 文件夹, 因为我们有一个不同的机制来扩展框架组件 (见下文)。

路由

- 默认情况下自动路由被禁用。你需要定义所有路由。
- 如果你希望以与 CI3 相同的方式使用自动路由, 则需要启用自动路由（传统版）。
- CI4 还具有可选的新的更安全的自动路由（改进版）。

模型、视图和控制器

- CodeIgniter 基于 MVC 概念。因此, 模型、视图和控制器的更改是你必须处理的最重要的事项之一。
- 在 CodeIgniter 4 中, 模型现在位于 **app/Models** 中, 在打开的 php 标记之后, 你必须添加 `namespace App\Models;` 以及 `use CodeIgniter\Model;`。最后一步是将 `extends CI_Model` 替换为 `extends Model`。
- CodeIgniter 4 的视图已移至 **app/Views**。此外, 你必须将加载视图的语法从 `$this->load->view('directory_name/file_name')` 更改为 `echo view('directory_name/file_name');`。
- CodeIgniter 4 的控制器必须移至 **app/Controllers**。之后, 在打开的 php 标记后添加 `namespace App\Controllers;`。最后, 将 `extends CI_Controller` 替换为 `extends BaseController`。
- 有关更多信息, 我们推荐你参考以下升级指南, 这些指南将为你提供一些分步说明, 以在 CodeIgniter4 中转换 MVC 类:

升级模型

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x 模型文档
- *CodeIgniter 4.x* 模型文档

变更点

- CI4 模型具有更多功能, 包括自动数据库连接、基本 CRUD、模型内验证和自动分页。
- 由于 CodeIgniter 4 添加了命名空间, 模型必须进行更改以支持命名空间。

升级指南

1. 首先，将所有模型文件移动到文件夹 **app/Models**。
2. 在 PHP 标签的开头之后添加这一行：`namespace App\Models;`。
3. 在 `namespace App\Models;` 行的下面添加这一行：`use CodeIgniter\Model;`。
4. 将 `extends CI_Model` 替换为 `extends Model`。
5. 添加 `protected $table` 属性并设置表名。
6. 添加 `protected $allowedFields` 属性并设置允许插入/更新的字段名称数组。

7. 代替 CI3 的 `$this->load->model('x');`，你现在应该使用 `$this->x = new X();`，遵循组件的命名空间约定。或者，你可以使用*model()* 函数: `$this->x = model('X');`。

如果在模型结构中使用子目录, 则必须根据情况更改命名空间。例如: 你有一个版本 3 模型位于 **application/models/users/user_contact.php**, 命名空间必须是 `namespace App\Models\Users;`, 版本 4 中的模型路径应如下所示:**app/Models/Users/UserContact.php**

CI4 中的新 Model 有很多内置方法。例如 `find($id)` 方法。使用它可以找到主键等于 `$id` 的数据。插入数据现在也比以前更简单。在 CI4 中有一个 `insert($data)` 方法。你可以选择使用所有这些内置方法, 并将代码迁移到新方法。

可以在*使用 CodeIgniter 的模型* 中找到有关这些方法的更多信息。

代码示例

CodeIgniter 3.x 版本

路径:**application/models**:

```
<?php

class News_model extends CI_Model
{
    public function set_news($title, $slug, $text)
    {
        $data = array(
            'title' => $title,
            'slug'  => $slug,
            'text'  => $text,
        );

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

CodeIgniter 4.x 版本

路径:**app/Models**:

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class NewsModel extends Model
{
    // Sets the table name.
    protected $table = 'news';

    public function setNews($title, $slug, $text)
    {
        $data = [
            'title' => $title,
            'slug'  => $slug,
            'text'  => $text,
        ];

        // Gets the Query Builder for the table, and calls␣
↪`insert()`.
        return $this->builder()->insert($data);
    }
}
```

上述代码是从 CI3 到 CI4 的直接翻译。它在模型中直接使用了查询构建器。请注意，当你直接使用查询构建器时，你将无法使用 CodeIgniter 模型中的功能。

如果你想使用 CodeIgniter 模型的功能，代码将是：

```
<?php

namespace App\Models;

use CodeIgniter\Model;
```

(续下页)

(接上页)

```
class NewsModel extends Model
{
    // Sets the table name.
    protected $table = 'news';

    // Sets the field names to allow to insert/update.
    protected $allowedFields = ['title', 'slug', 'text'];

    public function setNews($title, $slug, $text)
    {
        $data = [
            'title' => $title,
            'slug'  => $slug,
            'text'  => $text,
        ];

        // Uses Model's`insert()` method.
        return $this->insert($data);
    }
}
```

升级视图

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x 视图文档
- *CodeIgniter 4.x 视图文档*

变更点

- 你的视图看起来与以前基本相似, 但是调用它们的方式不同……不是 CI3 的 `$this->load->view('x');`, 可以使用 `return view('x');`。
- CI4 支持视图单元 来分段构建响应, 和视图布局 用于页面布局。
- 模板解析器 仍然存在, 并得到实质性增强。

升级指南

1. 首先, 将所有视图移动到 **app/Views** 文件夹
2. **在每个加载视图的脚本中更改视图加载语法:**
    - 从 `$this->load->view('directory_name/file_name')` 到 `return view('directory_name/file_name');`
    - 从 `$content = $this->load->view('file', $data, TRUE);` 到 `$content = view('file', $data);`
3. (可选) 可以将视图中的 echo 语法从 `<?php echo $title; ?>` 更改为 `<?= $title ?>`
4. 如果存在, 请删除 `defined('BASEPATH') OR exit('No direct script access allowed');` 这一行。

代码示例

CodeIgniter 3.x 版本

路径:**application/views**:

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

    <h3>My Todo List</h3>

    <ul>
    <?php foreach ($todo_list as $item): ?>
        <li><?php echo html_escape($item); ?></li>
    <?php endforeach; ?>
    </ul>

</body>
</html>
```

CodeIgniter 4.x 版本

路径:**app/Views**:

```
<html>
<head>
    <title><?= esc($title) ?></title>
</head>
<body>
    <h1><?= esc($heading) ?></h1>

    <h3>My Todo List</h3>

    <ul>
    <?php foreach ($todo_list as $item): ?>
        <li><?= esc($item) ?></li>
    <?php endforeach ?>
    </ul>
```

(续下页)

(接上页)

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

升级控制器

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x 版本*
  - *CodeIgniter 4.x 版本*

文档

- CodeIgniter 3.x 控制器文档
- *CodeIgniter 4.x 控制器文档*

变更点

- 由于 CodeIgniter 4 添加了命名空间, 必须对控制器进行更改以支持命名空间。
- CI4 控制器的构造函数不会自动将核心类加载到属性中。
- CI4 的控制器有一个特殊的构造函数*initController()*。
- CI4 为你提供了*Request* 和*Responses* 对象来使用 - 比 CI3 的方式更强大。
- 如果你需要一个基类控制器 (CI3 中的 `MY_Controller`), 请使用 **app/Controllers/BaseController.php**。
- 在控制器中调用 `echo`，如同在 CI3 中一样，仍然被支持，但建议从控制器返回字符串或 Response 对象。

升级指南

1. 首先, 将所有控制器文件移动到 **app/Controllers** 文件夹中。
2. 在打开的 php 标签之后添加此行:`namespace App\Controllers;`
3. 将 `extends CI_Controller` 替换为 `extends BaseController`。
4. 如果存在, 请删除 `defined('BASEPATH') OR exit('No direct script access allowed');` 这一行。

如果你在控制器结构中使用子目录, 则必须根据情况更改命名空间。例如, 你有一个版本 3 控制器位于 **application/controllers/users/auth/Register.php**, 则命名空间必须是 `namespace App\Controllers\Users\Auth;`, 版本 4 中的控制器路径应如下所示:**app/Controllers/Users/Auth/Register.php**。请确保子目录的首字母大写。
之后, 你必须在命名空间定义下面插入一个 `use` 语句, 以扩展 `BaseController`:: `use App\Controllers\BaseController;`

代码示例

CodeIgniter 3.x 版本

路径:**application/controllers**:

```
<?php

class Helloworld extends CI_Controller
{
    public function index($name)
    {
        echo 'Hello ' . html_escape($name) . '!';
    }
}
```

CodeIgniter 4.x 版本

路径:**app/Controllers**:

```
<?php

namespace App\Controllers;

class Helloworld extends BaseController
{
    public function index($name)
    {
        return 'Hello ' . esc($name) . '!';
    }
}
```

核心类更改

- **Input**
  - CI3 的 Input 对应于 CI4 的*IncomingRequest*。
  - 因为历史原因，CI3 和 CI4 使用了不正确的 HTTP 方法名称，比如 “get”，“post”。从 v4.5.0 开始，CI4 使用了正确的 HTTP 方法名称，比如 “GET”，“POST”。
- **Output**
  - CI3 的 Output 对应于 CI4 的*Responses*。

类加载

- 不再有 CodeIgniter “超级对象”，其中框架组件引用以属性的形式神奇地注入到你的控制器中。
- 类根据需要进行实例化，框架组件通过服务 进行管理。
- 自动加载程序 自动使用 PSR-4 风格定位类，在 `App` (**app** 文件夹) 和 `CodeIgniter` (即 **system** 文件夹) 顶级命名空间内；具有 Composer 自动加载支持。
- 你可以配置类加载以支持你最习惯的任何应用程序结构，包括 “HMVC” 风格。

- CI4 提供可以像 CI3 中的 `$this->load` 一样加载类和共享实例的工厂。

库

- 你的应用类仍然可以放在 **app/Libraries** 中, 但不必这样做。
- 不再使用 CI3 的 `$this->load->library('x');`，现在可以使用 `$this->x = new \App\Libraries\X();`，遵循你组件的命名空间约定。或者，你可以使用工厂:`$this->x = \CodeIgniter\Config\Factories::libraries('X');`。

辅助函数

- 辅助函数 与以前基本相同, 尽管有些进行了简化。
- 从 v4.3.0 开始, 你可以通过 **app/Config/Autoload.php** 自动加载辅助函数, 就像 CI3 一样。
- CodeIgniter 3 中的一些辅助函数在版本 4 中不再存在。对于所有这些辅助函数, 你必须找到一种新的方法来实现你的函数。这些辅助函数是 CAPTCHA Helper, Email Helper, Path Helper 和 Smiley Helper。
- CI3 的 Download Helper 已移除。你需要在使用 `force_download()` 的地方使用 Response 对象。请参阅强制文件下载。
- CI3 的 Language Helper 已移除。但在 CI4 中 `lang()` 始终可用。请参阅*lang()*。
- CI3 的 Typography Helper 在 CI4 中将是排版库。
- CI3 的 Directory Helper 和 File Helper 在 CI4 中将是文件系统辅助函数。
- CI3 的 String Helper 函数在 CI4 的文本辅助函数 中。
- **在 CI4 中, `redirect()` 与 CI3 中的完全不同。**
  - redirect() 文档 CodeIgniter 3.x
  - redirect() 文档 CodeIgniter 4.x
  - 在 CI4 中，*redirect()* 返回一个 `RedirectResponse` 实例，而不是重定向并终止脚本执行。你必须从控制器或控制器过滤器中返回它。
  - 在调用 `redirect()` 之前设置的 Cookie 和 Header 不会自动携带到 `RedirectResponse`。如果你想发送它们，你需要手动调用

withCookies() 或 withHeaders()。

- 你需要将 CI3 的 redirect('login/form') 改为 return redirect()->to('login/form')。

钩子

- 钩子已被事件替换。
- 不再使用 CI3 的 $hook['post_controller_constructor']，现在使用 Events::on('post_controller_constructor', ['MyClass', 'MyFunction']);，命名空间为 CodeIgniter\Events\Events;。
- 事件始终启用，并全局可用。
- 挂钩点 pre_controller 和 post_controller 已被移除。使用控制器过滤器代替。
- 挂钩点 display_override 和 cache_override 已被移除。因为基础方法已被移除。
- 挂钩点 post_system 已经移动到在发送最终渲染页面之前。

错误处理

- CI4 中的行为已经稍有更改。
  - 在 CI3 中，行为在 **index.php** 文件中设置：
    - 错误级别由 error_reporting() 设置的错误将被记录（但根据 log_threshold 设置，它们可能不会被写入日志文件）。
    - 错误级别为 E_ERROR | E_PARSE | E_COMPILE_ERROR | E_CORE_ERROR | E_USER_ERROR 的错误将停止框架处理，无论在 error_reporting() 中设置的错误级别如何。
  - 在 CI4 中，行为在 **app/Config/Boot/{environment}.php** 文件中设置：
    - 错误级别由 error_reporting() 设置的错误将被记录（但根据 Config\Logger::$threshold 设置，它们可能不会被写入日志文件）。
    - 所有不被 error_reporting() 忽略的错误都将停止框架处理。

扩展框架

- 你不需要 **core** 文件夹来保存 `MY_...` 框架组件扩展或替换文件。
- 你不需要在 libraries 文件夹中使用 `MY_X` 类来扩展或替换 CI4 组件。
- 将这些类放在任何地方, 并在 **app/Config/Services.php** 中添加适当的服务方法来加载你的组件, 而不是默认组件。
- 详细信息请参见创建核心系统类。

升级库

- 你的应用类仍然可以放在 **app/Libraries** 中, 但不必这样做。
- 不再使用 CI3 的 `$this->load->library('x');`, 现在可以使用 `$this->x = new \App\Libraries\X();`, 遵循你组件的命名空间约定。或者, 你可以使用工厂:`$this->x = \CodeIgniter\Config\Factories::libraries('X');`。
- CodeIgniter 3 中的一些库在版本 4 中不再存在。对于所有这些库, 你必须找到一种新的方法来实现你的函数。这些库是 日历, FTP, Javascript, 购物车, 引用通告, XML-RPC /服务器 和 Zip 编码。
- 存在于两个 CodeIgniter 版本中的所有其他库都可以通过一些调整来升级。最重要和使用最广泛的库都有一个升级指南, 它将通过简单的步骤和示例帮助你调整代码。

升级配置

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x 版本*
  - *CodeIgniter 4.x 版本*

文档

- CodeIgniter 3.x 配置文档
- *CodeIgniter 4.x 配置文档*

变更点

- 在 CI4 中, 配置现在存储在扩展 `CodeIgniter\Config\BaseConfig` 的类中。
- CI3 中的 **application/config/config.php** 将变为 **app/Config/App.php** 以及一些像 **app/Config/Security.php** 这样的特定类文件。
- 在配置类中, 配置值存储为公共类属性。
- 获取配置值的方法也进行了更改。

升级指南

1. 你需要根据 CI3 文件中的更改来更改 CI4 默认配置文件的值。配置名称与 CI3 中的名称大体相同。
2. 如果你在 CI3 项目中使用自定义配置文件, 则需要在 CI4 项目中的 **app/Config** 内将这些文件创建为新的 PHP 类。这些类应该在 `Config` 命名空间内, 并继承 `CodeIgniter\Config\BaseConfig`。
3. 创建所有自定义配置类后, 你需要将 CI3 配置中的变量复制为新的 CI4 配置类中的公共类属性。
4. 现在, 你需要在所有获取配置值的地方更改配置获取语法。CI3 语法类似于 `$this->config->item('item_name');`。你需要将其更改为 `config('MyConfig')->item_name;`。

代码示例

CodeIgniter 3.x 版本

路径:**application/config/site.php**:

```
<?php

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

$siteName  = 'My Great Site';
$siteEmail = 'webmaster@example.com';
```

CodeIgniter 4.x 版本

路径:**app/Config/Site.php**:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

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

升级数据库

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x 版本*
  - *CodeIgniter 4.x 版本*

文档

- CodeIgniter 3.x 数据库参考文档
- *CodeIgniter 4.x 使用数据库文档*

变更点

- CI4 的功能基本与 CI3 相同。
- CI3 中已知的 '数据库缓存 <https://www.codeigniter.com/userguide3/database/caching.html>'_ 功能已被删除。
- 方法名已更改为 camelCase 样式, 并且在运行查询之前, 现在需要初始化查询构建器。

升级指南

1. 将数据库凭据添加到 **app/Config/Database.php**。选项与 CI3 基本相同, 只是一些名称略有变化。
2. 在使用数据库的任何地方, 都必须用 `$db = db_connect();` 替换 `$this->load->database();`。
3. 如果使用多个数据库, 请使用以下代码加载其他数据库 `$db = db_connect('group_name');`。
4. 现在必须更改所有数据库查询。这里最重要的变化是用 `$db` 替换 `$this->db`, 并调整方法名和 `$db`。这里有一些例子:
   - `$this->db->query('YOUR QUERY HERE');` 改为 `$db->query('YOUR QUERY HERE');`
   - `$this->db->simple_query('YOUR QUERY')` 改为 `$db->simpleQuery('YOUR QUERY')`
   - `$this->db->escape("something")` 改为 `$db->escape("something");`
   - `$this->db->affected_rows();` 改为 `$db->affectedRows();`
   - `$query->result();` 改为 `$query->getResult();`
   - `$query->result_array();` 改为 `$query->getResultArray();`

- `echo $this->db->count_all('my_table');` 改为 `echo $db->table('my_table')->countAll();`

5. 要使用新的查询构建器类, 必须在 `$builder = $db->table('mytable');` 之后初始化构建器, 之后可以在 `$builder` 上运行查询。这里有一些例子:
   - `$this->db->get()` 改为 `$builder->get();`
   - `$this->db->get_where('mytable', array('id' => $id), $limit, $offset);` 改为 `$builder->getWhere(['id' => $id], $limit, $offset);`
   - `$this->db->select('title, content, date');` 改为 `$builder->select('title, content, date');`
   - `$this->db->select_max('age');` 改为 `$builder->selectMax('age');`
   - `$this->db->join('comments', 'comments.id = blogs.id');` 改为 `$builder->join('comments', 'comments.id = blogs.id');`
   - `$this->db->having('user_id', 45);` 改为 `$builder->having('user_id', 45);`
6. CI4 不提供 CI3 中已知的 '数据库缓存 <https://www.codeigniter.com/userguide3/database/caching.html>'_ 层, 所以如果需要缓存结果, 请改用*缓存驱动*。
7. 如果你在 Query Builder 中使用 `limit(0)`，由于一个 bug，CI4 会返回所有记录而不是没有记录。但从 v4.5.0 开始，你可以通过一个设置来改变这种不正确的行为。所以请更改该设置。详细信息参见*limit(0) 行为*。

代码示例

CodeIgniter 3.x 版本

```
<?php

$query = $this->db->select('title')
              ->where('id', $id)
              ->limit(10, 20)
              ->get('mytable');
```

CodeIgniter 4.x 版本

```
<?php

$builder = $db->table('mytable');

$query = $builder->select('title')
    ->where('id', $id)
    ->limit(10, 20)
    ->get();
```

升级邮件

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x 邮件文档
- *CodeIgniter 4.x* 邮件文档

变更点

- 只是一些小变化, 如方法名称和库的加载。
- 使用 SMTP 协议时的行为已经稍有更改。如果你使用 CI3 的设置，可能无法与你的 SMTP 服务器正确通信。请参见*SMTP 协议的 SSL 与 TLS* 和电子邮件首选项。

升级指南

1. 在类中，将 `$this->load->library('email');` 改为 `$email = service('email');`。
2. 从那时起, 需要将以 `$this->email` 开头的每一行改为 `$email`。
3. Email 类中的方法命名略有不同。除 `send()`、`attach()`、`printDebugger()` 和 `clear()` 之外的所有方法都有一个 `set` 前缀, 后跟之前的方法名。`bcc()` 现在变为 `setBcc()`, 等等。
4. **app/Config/Email.php** 中的配置属性已更改。你应该查看设置电子邮件首选项 以获取新的属性列表。

代码示例

CodeIgniter 3.x 版本

```
<?php

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

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

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

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

CodeIgniter 4.x 版本

```
<?php

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

升级加密

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x 加密库文档
- *CodeIgniter 4.x* 加密服务文档

变更点

- 不再支持 `MCrypt`, 它在 PHP 7.2 中已被弃用。

升级指南

1. 在配置中, `$config['encryption_key'] = 'abc123';` 从 **application/config/config.php** 移到了 **app/Config/Encryption.php** 中的 `public $key = 'abc123';`。
2. 如果需要解密用 CI3 加密的数据, 请配置设置以保持兼容性。参见*用于与 CI3 保持兼容性的配置*。
3. 在使用加密库的任何地方,都必须将 `$this->load->library('encryption');` 替换为 `$encrypter = service('encrypter');`, 并如下例代码中更改加密和解密的方法。

代码示例

CodeIgniter 3.x 版本

```
<?php

$this->load->library('encryption');

$plain_text = 'This is a plain-text message!';
$ciphertext = $this->encryption->encrypt($plain_text);

// Outputs: This is a plain-text message!
echo $this->encryption->decrypt($ciphertext);
```

CodeIgniter 4.x 版本

```
<?php

$encrypter = service('encrypter');

$plainText  = 'This is a plain-text message!';
$ciphertext = $encrypter->encrypt($plainText);

// Outputs: This is a plain-text message!
echo $encrypter->decrypt($ciphertext);
```

升级上传文件处理

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x 文件上传类文档
- *CodeIgniter 4.x* 上传文件处理文档

变更点

- 文件上传的功能发生了很大变化。你现在可以检查文件是否成功上传, 移动/存储文件也更简单了。

升级指南

在 CI4 中, 通过 `$file = $this->request->getFile('userfile')` 访问上传的文件。然后可以使用 `$file->isValid()` 检验文件是否成功上传。要存储文件, 可以使用 `$path = $this->request->getFile('userfile')->store('head_img/', 'user_name.jpg');`。这将文件存储在 **writable/uploads/head_img/user_name.jpg**。

你必须根据新方法更改文件上传代码。

代码示例

CodeIgniter 3.x 版本

```
<?php

class Upload extends CI_Controller {

    public function __construct()
    {
        parent::__construct();
        $this->load->helper(array('form', 'url'));
    }

    public function index()
    {
        $this->load->view('upload_form', array('error' => ' ' ));
    }

    public function do_upload()
    {
        $config['upload_path']   = './uploads/';
```

(续下页)

(接上页)

```
        $config['allowed_types'] = 'png|jpg|gif';
        $config['max_size']      = 100;
        $config['max_width']     = 1024;
        $config['max_height']    = 768;

        $this->load->library('upload', $config);

        if (! $this->upload->do_upload('userfile')) {
            $error = array('error' => $this->upload->display_
↪errors());

            $this->load->view('upload_form', $error);
        } else {
            $data = array('upload_data' => $this->upload->data());

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

CodeIgniter 4.x 版本

```
<?php

namespace App\Controllers;

class Upload extends BaseController
{
    public function index()
    {
        return view('upload_form', ['error' => ' ']);
    }

    public function do_upload()
    {
        $this->validateData([], [
```

(续下页)

(接上页)

```
            'userfile' => [
                'uploaded[userfile]',
                'max_size[userfile,100]',
                'mime_in[userfile,image/png,image/jpg,image/gif]',
                'ext_in[userfile,png,jpg,gif]',
                'max_dims[userfile,1024,768]',
            ],
        ]);

        $file = $this->request->getFile('userfile');

        if (! $path = $file->store()) {
            return view('upload_form', ['error' => 'upload failed
↪']);
        }
        $data = ['upload_file_path' => $path];

        return view('upload_success', $data);
    }
}
```

升级 HTML 表格

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x HTML 表格文档
- *CodeIgniter 4.x HTML 表格文档*

变更点

- 只是一些小变化, 如方法名称和库的加载。

升级指南

1. 在类中, 将 `$this->load->library('table');` 改为 `$table = new \CodeIgniter\View\Table();`。
2. 从那时起, 需要将以 `$this->table` 开头的每一行改为 `$table`。例如:`echo $this->table->generate($query);` 会变成 `echo $table->generate($query);`
3. HTML 表格类中的方法可能命名稍有不同。最重要的命名变化是从下划线方法名切换到 camelCase。版本 3 中的方法 `set_heading()` 现在命名为 `setHeading()`, 等等。

代码示例

CodeIgniter 3.x 版本

```
<?php

$this->load->library('table');

$this->table->set_heading('Name', 'Color', 'Size');

$this->table->add_row('Fred', 'Blue', 'Small');
$this->table->add_row('Mary', 'Red', 'Large');
$this->table->add_row('John', 'Green', 'Medium');

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

CodeIgniter 4.x 版本

```
<?php

$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();
```

升级图像处理类

- 文档
- 变更内容
- 升级指南
- 代码示例
  - *CodeIgniter 版本 3.x*
  - *CodeIgniter 版本 4.x*

文档

- CodeIgniter 3.x 图像处理类文档
- *CodeIgniter 4.x 图像处理类文档*

变更内容

- 在 CI3 中传递给构造函数或 `initialize()` 方法的首选项已更改为在 CI4 中的新方法中指定。
- 一些首选项如 `create_thumb` 被移除了。
- 在 CI4 中，必须调用 `save()` 方法来保存处理后的图像。
- `display_errors()` 已被移除，如果发生错误，将抛出异常。

升级指南

1. 在你的类中，将 `$this->load->library('image_lib');` 更改为 `$image = \Config\Services::image();`。
2. 更改传递给构造函数或 `initialize()` 方法的首选项为在相应方法中指定。
3. 调用 `save()` 方法保存文件。

代码示例

CodeIgniter 版本 3.x

```
<?php

$config['image_library']   = 'gd2';
$config['source_image']    = '/path/to/image/mypic.jpg';
$config['create_thumb']    = TRUE;
$config['maintain_ratio'] = TRUE;
$config['width']           = 75;
$config['height']          = 50;
```

(续下页)

(接上页)

```
$this->load->library('image_lib', $config);

$this->image_lib->resize();
```

CodeIgniter 版本 4.x

```
<?php

$image = \Config\Services::image();

$image
    ->withFile('/path/to/image/mypic.jpg')
    ->resize(75, 50, true)
    ->save('/path/to/image/mypic_thumb.jpg');
```

升级本地化

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x 语言文档
- *CodeIgniter 4.x* 本地化文档

变更点

- 在 CI4 中, 语言文件以数组形式返回语言线。

升级指南

1. 在 **Config/App.php** 中指定默认语言:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class App extends BaseConfig
{
    // ...

    public string $defaultLocale = 'en';

    // ...
}
```

2. 现在将语言文件移到 **app/Language/<locale>**。
3. 之后需要更改语言文件中的语法。下面的代码示例中可以看到文件中的语言数组应该如何表示。
4. 从每个文件中删除语言加载器 `$this->lang->load($file, $lang);`。
5. 用 `echo lang('Errors.errorEmailMissing');` 替换加载语言行的方法 `$this->lang->line('error_email_missing')`。

代码示例

CodeIgniter 3.x 版本

```
<?php

// error.php
$lang['error_email_missing']     = 'You must submit an email address
↪';
$lang['error_url_missing']       = 'You must submit a URL';
$lang['error_username_missing'] = 'You must submit a username';

// ...

$this->lang->load('error', $lang);
echo $this->lang->line('error_email_missing');
```

CodeIgniter 4.x 版本

```
<?php

// Errors.php
return [
    'errorEmailMissing'    => 'You must submit an email address',
    'errorURLMissing'      => 'You must submit a URL',
    'errorUsernameMissing' => 'You must submit a username',
    'nested'               => [
        'error' => [
            'message' => 'A specific error message',
        ],
    ],
];

// ...

echo lang('Errors.errorEmailMissing');
```

升级迁移

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x 版本*
  - *CodeIgniter 4.x 版本*
- 搜索和替换

文档

- CodeIgniter 3.x 数据库迁移文档
- *CodeIgniter 4.x 数据库迁移文档*

变更点

- 首先, 迁移文件的顺序命名 (`001_create_users`、`002_create_posts`) 不再被支持。CodeIgniter 4 版本仅支持时间戳方案 (`20121031100537_create_users`、`20121031500638_create_posts`)。如果使用了顺序命名, 则需要重命名每个迁移文件。
- 迁移表定义已更改。如果从 CI3 升级到 CI4 并使用相同的数据库, 则需要升级迁移表定义及其数据。
- 迁移过程也已更改。你现在可以使用简单的 CLI 命令迁移数据库:

```
php spark migrate
```

升级指南

1. 如果 v3 项目使用顺序迁移名, 则需要将其更改为时间戳名称。
2. 必须将所有迁移文件移至新的文件夹 **app/Database/Migrations**。
3. 如果存在, 请删除 `defined('BASEPATH') OR exit('No direct script access allowed');` 这一行。
4. 在打开的 php 标记之后添加此行:`namespace App\Database\Migrations;`。
5. 在 `namespace App\Database\Migrations;` 行下面添加此行:`use CodeIgniter\Database\Migration;`
6. 将 `extends CI_Migration` 替换为 `extends Migration`。
7. `Forge` 类中的方法名已更改为使用 camelCase。例如:
   - `$this->dbforge->add_field` 改为 `$this->forge->addField`
   - `$this->dbforge->add_key` 改为 `$this->forge->addKey`
   - `$this->dbforge->create_table` 改为 `$this->forge->addTable`
   - `$this->dbforge->drop_table` 改为 `$this->forge->addTable`
8. (可选) 可以将数组语法从 `array(...)` 更改为 `[...]`
9. 如果使用相同的数据库, 请升级迁移表。
   - **(开发环境)** 在完全新的数据库中运行 CI4 迁移, 以创建新的迁移表。
   - **(开发环境)** 导出迁移表。
   - **(生产环境)** 删除 (或重命名) 现有的 CI3 迁移表。
   - **(生产环境)** 导入新的迁移表和数据。

代码示例

CodeIgniter 3.x 版本

路径:**application/migrations**:

```
<?php
```

(续下页)

(接上页)

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

class Migration_Add_blog extends CI_Migration
{
    public function up()
    {
        $this->dbforge->add_field(array(
            'blog_id' => array(
                'type'           => 'INT',
                'constraint'     => 5,
                'unsigned'       => true,
                'auto_increment' => true,
            ),
            'blog_title' => array(
                'type'       => 'VARCHAR',
                'constraint' => '100',
            ),
            'blog_description' => array(
                'type' => 'TEXT',
                'null' => true,
            ),
        ));
        $this->dbforge->add_key('blog_id', true);
        $this->dbforge->create_table('blog');
    }

    public function down()
    {
        $this->dbforge->drop_table('blog');
    }
}
```

CodeIgniter 4.x 版本

路径:**app/Database/Migrations**:

```
<?php

namespace App\Database\Migrations;

use CodeIgniter\Database\Migration;

class AddBlog extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'blog_id' => [
                'type'           => 'INT',
                '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');
    }
}
```

搜索和替换

你可以使用以下表格搜索和替换旧的 CI3 文件。

搜索	替换
extends CI_Migration	extends Migration
$this->dbforge->add_field	$this->forge->addField
$this->dbforge->add_key	$this->forge->addKey
$this->dbforge->create_table	$this->forge->createTable
$this->dbforge->drop_table	$this->forge->dropTable

升级输出类

- 文档
- 有哪些变化
- 升级指南
- 代码示例
  - *CodeIgniter 3.x 版本*
  - *CodeIgniter 4.x 版本*

文档

- CodeIgniter 3.x 输出类文档
- *CodeIgniter 4.x HTTP 响应文档*

有哪些变化

- 输出类已更改为响应类。
- 方法已被重命名。

升级指南

1. HTTP 响应类中的方法命名略有不同。最重要的命名变化是从下划线方法名切换到 camelCase。版本 3 中的方法 `set_content_type()` 现在命名为 `setContentType()`, 等等。
2. 在大多数情况下, 你需要将 `$this->output` 改为 `$this->response` 后跟方法。可以在*HTTP 响应* 中找到所有方法。

代码示例

CodeIgniter 3.x 版本

```
<?php

$this->output->set_status_header(404);

// ...

$this->output
    ->set_content_type('application/json')
    ->set_output(json_encode(array('foo' => 'bar')));
```

CodeIgniter 4.x 版本

```
<?php

$this->response->setStatusCode(404);

// ...

return $this->response->setJSON(['foo' => 'bar']);
```

升级分页

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x 分页类文档
- *CodeIgniter 4.x* 分页文档

变更点

- 你需要更改视图和控制器以使用新的分页库。
- 如果要自定义分页链接, 需要创建视图模板。
- 在 CI4 中, 分页只使用实际的页码。你无法使用 CI3 默认的项目起始索引 (偏移量)。
- 如果使用*CodeIgnite\Model*, 可以使用 Model 类中的内置方法。

升级指南

1. 在视图中进行以下更改:
   - `<?php echo $this->pagination->create_links(); ?>` 改为 `<?= $pager->links() ?>`
2. 在控制器中需要做以下更改:
   - 你可以在每个 Model 上使用内置的 `paginate()` 方法。请参阅下面的代码示例, 看看如何在特定模型上设置分页。

代码示例

CodeIgniter 3.x 版本

```
<?php

$this->load->library('pagination');
$config['base_url']     = base_url().'users/index/';
$config['total_rows']   = $this->db->count_all('users');
$config['per_page']     = 10;
$config['uri_segment'] = 3;
$config['attributes']   = array('class' => 'pagination-link');
$this->pagination->initialize($config);

$data['users'] = $this->user_model->get_users(FALSE, $config['per_
↪page'], $offset);

$this->load->view('posts/index', $data);
```

CodeIgniter 4.x 版本

```
<?php

$model = new \App\Models\UserModel();

$data = [
    'users' => $model->paginate(10),
    'pager' => $model->pager,
];

return view('users/index', $data);
```

升级路由

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x 版本*
  - *CodeIgniter 4.x 版本*

文档

- CodeIgniter 3.x URI 路由文档
- *CodeIgniter 4.x URI 路由文档*

变更点

- 在 CI4 中, 默认关闭自动路由。
- 在 CI4 中引入了新的更安全的自动路由（改进版）。
- 在 CI4 中, 路由配置不再通过设置路由数组来完成。
- CI3 中的通配符 `(:any)` 在 CI4 中将会是占位符 `(:segment)`。在 CI4 中的 `(:any)` 匹配多个段。请参见*URI 路由*。

升级指南

1. 如果你以与 CI3 相同的方式使用自动路由, 则需要启用自动路由（传统版）。
2. 你必须更改每个路由行的语法, 并将其附加到 **app/Config/Routes.php** 中。例如:
   - `$route['journals'] = 'blogs';` 改为 `$routes->add('journals', 'Blogs::index');`。这将映射到 `Blogs` 控制器中的 `index()` 方法。

- `$route['product/(:any)'] = 'catalog/product_lookup';` 改为 `$routes->add('product/(:segment)', 'Catalog::productLookup');`。别忘了将 `(:any)` 替换为 `(:segment)`。
- `$route['login/(.+)'] = 'auth/login/$1';` 改为 `$routes->add('login/(.+)', 'Auth::login/$1');`

备注: 为了向后兼容，这里使用了 `$routes->add()`。但我们强烈建议使用*HTTP 方法路由* 中的 `$routes->get()` 替代 `$routes->add()`，以增强安全性。

代码示例

CodeIgniter 3.x 版本

路径:**application/config/routes.php**:

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

// ...

$route['posts/index']  = 'posts/index';
$route['teams/create'] = 'teams/create';
$route['teams/update'] = 'teams/update';

$route['posts/create']   = 'posts/create';
$route['posts/update']   = 'posts/update';
$route['drivers/create'] = 'drivers/create';
$route['drivers/update'] = 'drivers/update';
$route['posts/(:any)']   = 'posts/view/$1';
```

CodeIgniter 4.x 版本

路径:**app/Config/Routes.php**:

```
<?php

use CodeIgniter\Router\RouteCollection;

/**
 * @var RouteCollection $routes
 */
$routes->get('/', 'Home::index');

$routes->add('posts/index', 'Posts::index');
$routes->add('teams/create', 'Teams::create');
$routes->add('teams/update', 'Teams::update');

$routes->add('posts/create', 'Posts::create');
$routes->add('posts/update', 'Posts::update');
$routes->add('drivers/create', 'Drivers::create');
$routes->add('drivers/update', 'Drivers::update');
$routes->add('posts/(:segment)', 'Posts::view/$1');
```

备注: 为了向后兼容，这里使用了 `$routes->add()`。但我们强烈建议使用*HTTP 方法路由* 中的 `$routes->get()` 替代 `$routes->add()`，以增强安全性。

升级安全性

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x 版本*

– *CodeIgniter 4.x 版本*

文档

- CodeIgniter 3.x 安全类文档
- *CodeIgniter 4.x 安全性文档*

备注: 如果使用表单辅助函数 并全局启用 CSRF 过滤器, 那么`form_open()` 将自动在表单中插入隐藏的 CSRF 字段。所以你不需要自行升级这个。

变更点

- 实现 CSRF 令牌到 HTML 表单的方法已经更改。

升级指南

1. 要在 CI4 中启用 CSRF 保护, 必须在 **app/Config/Filters.php** 中启用它:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public $globals = [
        'before' => [
            // 'honeypot',
            'csrf',
        ],
    ];
```

(续下页)

(接上页)

```
    // ...
}
```

2. 在 HTML 表单中，必须删除类似 `<input type="hidden" name="<?= $csrf['name'] ?>" value="<?= $csrf['hash'] ?>" />` 的 CSRF 输入字段。
3. 现在, 在 HTML 表单中, 必须在表单主体的某处添加 `<?= csrf_field() ?>`, 除非使用 `form_open()`。

代码示例

CodeIgniter 3.x 版本

```
<?php

$csrf = array(
    'name' => $this->security->get_csrf_token_name(),
    'hash' => $this->security->get_csrf_hash()
);

?>

<form>
    <input name="name" type="text">
    <input name="email" type="text">
    <input name="password" type="password">

    <input type="hidden" name="<?= $csrf['name'] ?>" value="<?=
↪$csrf['hash'] ?>">
    <input type="submit" value="Save">
</form>
```

CodeIgniter 4.x 版本

```
<form>
    <input name="name" type="text">
    <input name="email" type="text">
    <input name="password" type="password">

    <?= csrf_field() ?>
    <input type="submit" value="Save">
</form>
```

升级 Session

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x 版本*
  - *CodeIgniter 4.x 版本*

文档

- CodeIgniter 3.x Session 库文档
- *CodeIgniter 4.x Session 库文档*

变更点

- 只是一些小变化, 如方法名称和库的加载。
- 在数据库驱动中，Session 表的定义已经发生了变化。

升级指南

1. 在使用 Session 库的任何地方，用 `$session = session();` 替换 `$this->load->library('session');`。
2. 从那时起，必须用 `$session` 后跟新方法名替换以 `$this->session` 开头的每一行。
   - 要访问 Session 数据，请使用 `$session->item` 或 `$session->get('item')` 语法，而不是 CI3 语法 `$this->session->name`。
   - 要设置数据，请使用 `$session->set($array);` 代替 `$this->session->set_userdata($array);`。
   - 要删除数据，请使用 `unset($_SESSION['some_name']);` 或 `$session->remove('some_name');` 代替 `$this->session->unset_userdata('some_name');`。
   - 要将 Session 数据标记为只在下一个请求中可用的闪存数据，请使用 `$session->markAsFlashdata('item');` 代替 `$this->session->mark_as_flash('item');`
3. 如果你使用数据库驱动，你需要重新创建 Session 表。参见*DatabaseHandler 驱动程序*。

代码示例

CodeIgniter 3.x 版本

```
<?php

$this->load->library('session');

$_SESSION['item'];
$this->session->item;
$this->session->userdata('item');
```

CodeIgniter 4.x 版本

```
<?php

$session = session();

$_SESSION['item'];  // But we do not recommend to use superglobal␣
↪directly.
$session->get('item');
$session->item;
session('item');
```

升级验证

- 库文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

库文档

- CodeIgniter 3.x 表单验证文档
- *CodeIgniter 4.x* 验证文档

变更点

- 如果你想更改验证错误的显示方式，你需要设置 CI4 的验证视图模板。
- CI4 验证中没有 CI3 中的 回调 *<https://codeigniter.org.cn/userguide3/libraries/form_validation.html#id1.* 请使用*Callable Rules* （从 v4.5.0 开始）或*Closure Rules* （从 v4.3.0 开始）或*Rule Classes* 代替。
- 在 CI3 中，Callbacks/Callable 规则有优先级，但在 CI4 中，Closure/Callable 规则没有优先级，按列出的顺序进行检查。
- 从 v4.5.0 开始引入了*Callable Rules*，但它与 CI3 的 Callable 有些不同。
- CI4 验证格式规则不允许空字符串。
- CI4 验证永远不会更改你的数据。
- 从 v4.3.0 开始，引入了`validation_errors()`，但其 API 与 CI3 的不同。

升级指南

1. 在包含表单的视图中进行更改:
   - `<?php echo validation_errors(); ?>` 改为 `<?= validation_list_errors() ?>`
2. 在控制器中进行更改:
   - `$this->load->helper(array('form', 'url'));` 改为 `helper('form');`
   - 移除 `$this->load->library('form_validation');`
   - `if ($this->form_validation->run() == FALSE)` 改为 `if (!$this->validateData($data, $rules))` 其中 `$data` 是要验证的数据，通常是 POST 数据 `$this->request->getPost()`。
   - `$this->load->view('myform');` 改为 `return view('myform', ['validation' => $this->validator,]);`
3. 必须更改验证规则。新语法是在控制器中将规则设置为数组:

```
<?php
```

(续下页)

(接上页)

```
$isValid = $this->validateData($data, [
    'name'  => 'required|min_length[3]',
    'email' => 'required|valid_email',
    'phone' => 'required|numeric|max_length[10]',
]);
```

代码示例

CodeIgniter 3.x 版本

路径:**application/views**:

```
<html>
<head>
    <title>My Form</title>
</head>
<body>

    <?php echo validation_errors(); ?>

    <?php echo form_open('form'); ?>

    <h5>Username</h5>
    <input type="text" name="username" value="" size="50" />

    <h5>Password</h5>
    <input type="text" name="password" value="" size="50" />

    <h5>Password Confirm</h5>
    <input type="text" name="passconf" value="" size="50" />

    <h5>Email Address</h5>
    <input type="text" name="email" value="" size="50" />

    <div><input type="submit" value="Submit" /></div>

    </form>
```

(续下页)

(接上页)

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

路径:**application/controllers**:

```
<?php

class Form extends CI_Controller {

    public function index()
    {
        $this->load->helper(array('form', 'url'));

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

        // Set validation rules

        if ($this->form_validation->run() == FALSE) {
                $this->load->view('myform');
        } else {
                $this->load->view('formsuccess');
        }
    }
}
```

CodeIgniter 4.x 版本

路径:**app/Views**:

```
<html>
<head>
    <title>My Form</title>
</head>
<body>

    <?= validation_list_errors() ?>
```

(续下页)

(接上页)

```
    <?= 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>
```

路径:**app/Controllers**:

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class Form extends Controller
{
    public function index()
    {
        helper('form');

        $data = $this->request->getPost();
```

(续下页)

(接上页)

```
        if (! $this->validateData($data, [
            // Validation rules
        ])) {
            return view('myform');
        }

        return view('formsuccess');
    }
}
```

升级视图解析器

- 文档
- 变更点
- 升级指南
- 代码示例
  - *CodeIgniter 3.x* 版本
  - *CodeIgniter 4.x* 版本

文档

- CodeIgniter 3.x 模板解析器文档
- *CodeIgniter 4.x* 视图解析器文档

变更点

- 你必须更改解析器库的实现和加载方式。
- 视图可以从 CI3 复制。通常不需要对其进行任何更改。

升级指南

1. 在使用视图解析器库的任何地方, 用 `$parser = service('parser');` 替换 `$this->load->library('parser');`。
2. 你必须将控制器中的渲染部分从 `$this->parser->parse('blog_template', $data);` 改为 `return $parser->setData($data)->render('blog_template');`。

代码示例

CodeIgniter 3.x 版本

```
<?php

$this->load->library('parser');

$data = array(
    'blog_title'   => 'My Blog Title',
    'blog_heading' => 'My Blog Heading'
);

$this->parser
    ->parse('blog_template', $data);
```

CodeIgniter 4.x 版本

```
<?php

$parser = service('parser');

$data = [
    'blog_title'   => 'My Blog Title',
    'blog_heading' => 'My Blog Heading',
];

return $parser->setData($data)->render('blog_template');
```

2.1.8 CodeIgniter 仓库

- *codeigniter4 组织*
- *Composer 包*
- *CodeIgniter 4 项目*

codeigniter4 组织

CodeIgniter 4 开源项目有自己的 GitHub 组织。

有几个开发仓库, 对潜在贡献者感兴趣:

仓库	受众	描述
CodeIgniter4	贡献者	项目代码库, 包括测试和用户指南源码
translations	开发者	系统消息翻译
coding-standard	贡献者	编码风格约定和规则
devkit	开发者	CodeIgniter 库和项目的开发工具包
settings	开发者	CodeIgniter 4 的设置库
shield	开发者	CodeIgniter 4 的身份验证和授权库
tasks	开发者	CodeIgniter 4 的任务调度程序
cache	开发者	CodeIgniter 4 的 PSR-6 和 PSR-16 缓存适配器
queue	开发者	CodeIgniter 4 的队列

还有几个部署仓库, 在安装说明中引用。部署仓库在发布新版本时会自动构建, 不直接贡献。

仓库	受众	描述
framework	开发者	框架的已发布版本
appstarter	开发者	启动项目 (app/public/writable)。依赖于 “framework”
userguide	任何人	预构建用户指南

在上述所有仓库中, 可以通过在其 GitHub 仓库页面的 “Code” 选项卡中的二级导航栏中选择 “releases” 链接来下载仓库的最新版本。可以通过选择仓库主页右侧的 “克隆或下载” 下拉按钮来克隆或下载每个仓库的当前 (开发中) 版本。

Composer 包

我们在 packagist.org 上也维护 composer 可安装包。这些与上面提到的仓库对应:

- codeigniter4/framework
- codeigniter4/appstarter
- codeigniter4/translations
- codeigniter/coding-standard
- codeigniter4/devkit
- codeigniter4/settings
- codeigniter4/shield
- codeigniter4/cache

有关更多信息, 请参阅安装 页面。

CodeIgniter 4 项目

我们在 GitHub 上也维护一个 codeigniter4projects 组织, 其中包含不属于框架本身的项目, 但展示了它或使它更易于使用!

仓库	受众	描述
website	开发者	使用 CodeIgniter 4 编写的 codeigniter.com 网站
playground	开发者	以项目形式的基本代码示例。仍在增长。

这些不是 composer 可安装的仓库。

章节 3

创建第一个应用

3.1 构建你的第一个应用程序

- 概述
- 启动并运行
  - 安装 *CodeIgniter*
  - 设置开发模式
  - 运行开发服务器
- 欢迎页面
- 调试
  - 调试工具栏
  - 错误页面

3.1.1 概述

本教程旨在向你介绍 CodeIgniter4 框架以及 MVC 架构的基本原则。它将以一步一步的方式向你展示一个基本的 CodeIgniter 应用程序是如何构建的。

如果你不熟悉 PHP, 我们建议你在继续之前先查看 W3Schools PHP 教程。

在本教程中, 你将创建一个 **基本的新闻应用程序**。你将从编写能够加载静态页面的代码开始。接下来, 你将创建一个新闻部分, 它从数据库中读取新闻条目。最后, 你将添加一个表单以在数据库中创建新闻条目。

本教程将主要关注:

- Model-View-Controller 基础知识
- 路由基础知识
- 表单验证
- 使用 CodeIgniter 的模型执行基本数据库查询

整个教程分成几个页面, 每个页面解释 CodeIgniter 框架的一小部分功能。你将按照以下页面进行操作:

- 简介, 本页面, 概述你可以期待的内容, 并获取默认应用程序下载并运行。
- 静态页面, 将教你控制器、视图和路由的基础知识。
- 新闻部分, 你将开始使用模型并执行一些基本的数据库操作。
- 创建新闻条目, 将介绍更高级的数据库操作和表单验证。
- 结束语, 将为你提供一些进一步阅读和其他资源的指导。

享受你对 CodeIgniter 框架的探索。

静态页面

- 设置路由规则
- 让我们制作第一个控制器
  - 创建 *Pages* 控制器
  - 创建视图
- 向控制器添加逻辑

- 创建 *home.php* 和 *about.php*
- 完成 *Pages::view()* 方法

- 运行应用程序

备注: 本教程假设你已经下载了 CodeIgniter 并在开发环境中安装了框架。

首先，你需要设置路由规则来处理静态页面。

设置路由规则

路由将 URI 关联到控制器的方法。控制器只是一个帮助委派工作的类。我们稍后将创建一个控制器。

让我们设置路由规则。打开位于 **app/Config/Routes.php** 的路由文件。

开始时，唯一的路由指令应该是:

```
<?php

use CodeIgniter\Router\RouteCollection;

/**
 * @var RouteCollection $routes
 */
$routes->get('/', 'Home::index');
```

该指令表示任何没有指定内容的传入请求应由 `Home` 控制器内的 `index()` 方法处理。

在 `'/'` 的路由指令之后，添加以下行。

```
use App\Controllers\Pages;

$routes->get('pages', [Pages::class, 'index']);
$routes->get('(:segment)', [Pages::class, 'view']);
```

CodeIgniter 从上到下读取其路由规则，并将请求路由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名称（右侧）。当请求到达时，CodeIgniter 查找第一个匹配项，并调用适当的控制器和方法，可能带有参数。

有关路由的更多信息，请参阅*URI 路由*。

在这里，`$routes` 对象中的第二个规则匹配到一个 GET 请求，URI 路径为 **/pages**，并映射到 `Pages` 类的 `index()` 方法。

`$routes` 对象中的第三个规则匹配到一个 GET 请求，使用占位符 `(:segment)`，并将参数传递给 `Pages` 类的 `view()` 方法。

让我们制作第一个控制器

接下来，你需要设置一个 **控制器**来处理静态页面。控制器只是一个帮助委派工作的类，它是你的 Web 应用程序的粘合剂。

创建 Pages 控制器

在 **app/Controllers/Pages.php** 中创建一个带以下代码的文件。

重要: 你应该始终注意文件名的大小写。许多开发人员在 Windows 或 macOS 上的大小写不敏感的文件系统上开发。然而，大多数服务器环境使用大小写敏感的文件系统。如果文件名大小写不正确，本地工作的代码将无法在服务器上工作。

```php
<?php

namespace App\Controllers;

class Pages extends BaseController
{
    public function index()
    {
        return view('welcome_message');
    }

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

你创建了一个名为 `Pages` 的类, 它有一个名为 `view()` 的方法, 该方法接受一个名为 `$page` 的参数。它还有一个 `index()` 方法, 与 **app/Controllers/Home.php** 中的默认控制器相同; 该方法显示 CodeIgniter 欢迎页面。

备注: 本教程中提到了两个 `view()` 函数。一个是使用 `public function view($page = 'home')` 和 `return view('welcome_message')` 显示视图而创建的类方法。从技术上讲, 两者都是一个函数。但是当你在一个类中创建一个函数时, 它被称为方法。

`Pages` 类正在扩展 `BaseController` 类, 后者扩展了 `CodeIgniter\Controller` 类。这意味着新的 Pages 类可以访问在 `CodeIgniter\Controller` 类中定义的方法和属性 (**system/Controller.php**)。

控制器将成为你的 Web 应用程序的每个请求的中心。与任何 PHP 类一样, 你可以在控制器中通过 `$this` 来引用它。

创建视图

既然你已经创建了第一个方法, 是时候制作一些基本的页面模板了。我们将创建两个"视图" (页面模板) 作为我们的页面页脚和页眉。

在 **app/Views/templates/header.php** 中创建页眉, 并添加以下代码:

```
<!doctype html>
<html>
<head>
    <title>CodeIgniter 教程</title>
</head>
<body>

    <h1><?= esc($title) ?></h1>
```

页眉包含在加载主视图之前要显示的基本 HTML 代码, 以及一个标题。它还将输出 `$title` 变量, 我们将在控制器中定义它。现在, 在 **app/Views/templates/footer.php** 中创建一个页脚, 其中包含以下代码:

```
    <em>&copy; 2022</em>
</body>
```

(续下页)

(接上页)

```
</html>
```

备注: 如果仔细查看 **header.php** 模板, 我们正在使用*esc()* 函数。这是 CodeIgniter 提供的全局函数, 可帮助防止 XSS 攻击。你可以在全局函数和常量 中了解更多信息。

向控制器添加逻辑

创建 home.php 和 about.php

早些时候, 你设置了一个带有 `view()` 方法的控制器。该方法接受一个参数, 即要加载的页面的名称。

静态页面正文将位于 **app/Views/pages** 目录中。

在该目录中, 创建两个名为 **home.php** 和 **about.php** 的文件。在这些文件中输入一些文本 (任何你想要的), 然后保存它们。如果你想特别原创, 可以试试 "Hello World!"。

完成 Pages::view() 方法

为了加载这些页面, 你将不得不检查请求的页面是否确实存在。这将是在上面创建的 `Pages` 控制器中的 `view()` 方法的主体:

```
<?php

namespace App\Controllers;

// Add this line to import the class.
use CodeIgniter\Exceptions\PageNotFoundException;

class Pages extends BaseController
{
    // ...

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

(续下页)

(接上页)

```
        if (! is_file(APPPATH . 'Views/pages/' . $page . '.php')) {
            // Whoops, we don't have a page for that!
            throw new PageNotFoundException($page);
        }

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

        return view('templates/header', $data)
            . view('pages/' . $page)
            . view('templates/footer');
    }
}
```

并在 namespace 行后添加 use CodeIgniter\Exceptions\PageNotFoundException; 来导入 PageNotFoundException 类。

现在, 当请求的页面确实存在时, 它将被加载, 包括页眉和页脚, 并返回给用户。如果控制器返回一个字符串, 它将显示给用户。

备注: 控制器必须返回一个字符串或*Response* 对象。

如果请求的页面不存在, 将显示 "404 页面未找到" 错误。

此方法的第一行检查页面是否实际存在。PHP 原生的 `is_file()` 函数用于检查文件是否在预期的位置。`PageNotFoundException` 是一个 CodeIgniter 异常, 它会导致显示 404 页面未找到错误页面。

在页眉模板中, 使用 `$title` 变量来自定义页面标题。此方法中定义了 title 的值, 但不是将值赋给变量, 而是将其赋给 `$data` 数组中的 title 元素。

最后要做的就是以它们应显示的顺序加载视图。将使用 CodeIgniter 中内置的*view()* 函数来完成此操作。`view()` 函数中的第二个参数用于向视图传递值。`$data` 数组中的每个值都分配给一个其键的名称的变量。所以控制器中的 `$data['title']` 的值在视图中等效于 `$title`。

备注: 传递给*view()* 函数的任何文件和目录名称必须匹配实际目录和文件本身的情

况, 否则系统将在区分大小写的平台上抛出错误。你可以在视图 中了解更多信息。

运行应用程序

准备测试了吗? 你不能使用 PHP 的内置服务器运行应用程序, 因为它不会正确处理 **public** 中提供的 **.htaccess** 规则, 这些规则消除了在 URL 中指定 "**index.php/**" 的需要。不过 CodeIgniter 有自己的命令可以使用。

在项目的根目录下, 在命令行中:

```
php spark serve
```

将启动一个网页服务器, 可以在 8080 端口上访问。如果你将浏览器的 location 字段设置为 **localhost:8080**, 则应该会看到 CodeIgniter 欢迎页面。

现在访问 **localhost:8080/home**。是否正确路由到 `Pages` 控制器中的 `view()` 方法? 太棒了!

你应该看到类似以下内容:

你现在可以在浏览器的地址栏中尝试多个 URL, 以查看上面制作的 `Pages` 控制器生成的内容…

URL	将显示
localhost:8080/	CodeIgniter "欢迎" 页面。来自 `Home` 控制器中的 `index()` 方法的结果。
local-host:8080/pages	来自我们的 `Pages` 控制器中的 `index()` 方法的结果, 它显示 CodeIgniter "欢迎" 页面。
local-host:8080/home	因为我们明确请求了上面创建的 "home" 页面, 所以结果来自我们 `Pages` 控制器中的 `view()` 方法。
local-host:8080/about	显示上面制作的 "关于" 页面, 因为我们明确要求它。
local-host:8080/shop	一个 "404 - 文件未找到" 错误页面, 因为没有 **app/Views/pages/shop.php**。

新闻部分

- *建立教程所需的数据库*
- *连接数据库*
- *设置模型*
  - *创建 NewsModel*
  - *添加 NewsModel::getNews() 方法*
- *显示新闻*
  - *添加路由规则*
  - *创建 News 控制器*
  - *完成 News::index() 方法*
  - *创建 news/index 视图文件*
  - *完成 News::show() 方法*
  - *创建 news/view 视图文件*

在最后一节中, 我们通过编写引用静态页面的类来概述了框架的一些基本概念。我们通过添加自定义路由规则来清理 URI。现在是时候引入动态内容并开始使用数据库了。

建立教程所需的数据库

CodeIgniter 安装假定你已经按要求 设置了合适的数据库。在本教程中, 我们为 MySQL 数据库提供了 SQL 代码, 并且我们还假设你有合适的客户端来发出数据库命令 (mysql、MySQL Workbench 或 phpMyAdmin)。

你需要为本教程创建一个数据库 `ci4tutorial`, 然后配置 CodeIgniter 来使用它。

使用数据库客户端, 连接到你的数据库并运行以下 SQL 命令 (MySQL):

```
CREATE TABLE news (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    title VARCHAR(128) NOT NULL,
    slug VARCHAR(128) NOT NULL,
    body TEXT NOT NULL,
    PRIMARY KEY (id),
    UNIQUE slug (slug)
);
```

此外, 添加一些种子记录。现在, 我们只向你展示创建表所需的 SQL 语句, 但是你应该意识到, 一旦对 CodeIgniter 更熟悉, 就可以以编程方式完成此操作; 稍后, 你可以阅读有关迁移 和种子 的信息, 以创建更有用的数据库设置。

一个有趣的注释：在 Web 发布的上下文中，“slug” 是一个用户友好且符合 SEO 的短文本，用于在 URL 中标识和描述资源。

种子记录可能如下:

```
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.');
```

连接数据库

当你安装 CodeIgniter 时创建的本地配置文件 **.env** 应该已经对要使用的数据库的数据库属性设置进行了取消注释和适当设置。确保你已按*数据库配置* 中所述正确配置了数据库:

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

设置模型

不要在控制器中直接编写数据库操作, 查询应放在模型中, 以便以后可以轻松重用。模型是检索、插入和更新数据库或其他数据存储中的信息的地方。它们为数据提供访问。你可以在*使用 CodeIgniter 的模型* 中阅读更多相关信息。

创建 NewsModel

打开 **app/Models** 目录并创建一个新文件 **NewsModel.php**, 添加以下代码。

```
<?php

namespace App\Models;

use CodeIgniter\Model;

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

此代码看起来类似于早先使用的控制器代码。它通过扩展 `CodeIgniter\Model` 并加载数据库库来创建一个新模型。这将通过 `$this->db` 对象使数据库类可用。

添加 NewsModel::getNews() 方法

现在数据库和模型已经设置好了, 你需要一个从数据库中获取所有帖子的方法。为此,CodeIgniter 包含的数据库抽象层查询构建器 在 `CodeIgniter\Model` 中使用。这使你可以编写一次’查询’, 并在所有支持的数据库系统 上使用。Model 类也允许你轻松使用 Query Builder 并提供一些额外的工具, 以简化使用数据。向模型添加以下代码。

```
/**
 * @param false|string $slug
 *
 * @return array|null
 */
public function getNews($slug = false)
{
    if ($slug === false) {
        return $this->findAll();
    }

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

使用此代码, 你可以执行两种不同的查询。你可以获取所有新闻记录, 也可以通过其 slug 获取新闻项。你可能已经注意到, 在运行查询之前没有转义 `$slug` 变量; 查询构建器 会为你完成这一步。

这里使用的两个方法 `findAll()` 和 `first()` 由 `CodeIgniter\Model` 类提供。它们已经知道基于我们早先在 `NewsModel` 类中设置的 `$table` 属性要使用的表。它们是使用 Query Builder 在当前表上运行命令的辅助方法, 并以你选择的格式返回结果数组。在本示例中, `findAll()` 返回数组的数组。

显示新闻

现在查询已经编写好了, 应该将模型与要显示新闻项的视图绑定。这可以在我们早先创建的 Pages 控制器中完成, 但是为了清晰起见, 定义了一个新的 News 控制器。

添加路由规则

修改你的 **app/Config/Routes.php** 文件，使其如下所示：

```
<?php

// ...

use App\Controllers\News; // Add this line
use App\Controllers\Pages;

$routes->get('news', [News::class, 'index']);           // Add this␣
↪line
$routes->get('news/(:segment)', [News::class, 'show']); // Add this␣
↪line

$routes->get('pages', [Pages::class, 'index']);
$routes->get('(:segment)', [Pages::class, 'view']);
```

这样可以确保请求到达 News 控制器，而不是直接到达 Pages 控制器。第二个 $routes->get() 行将带有 slug 的 URI 路由到 News 控制器中的 show() 方法。

创建 News 控制器

在 **app/Controllers/News.php** 中创建新的控制器。

```
<?php

namespace App\Controllers;

use App\Models\NewsModel;

class News extends BaseController
```

(续下页)

(接上页)

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

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

    public function show(?string $slug = null)
    {
        $model = model(NewsModel::class);

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

查看代码, 你可能会发现与我们之前创建的文件有些相似之处。首先, 它扩展了 `BaseController`, 后者扩展了核心 CodeIgniter 类 `Controller`, 它提供了一些辅助方法, 并确保你可以访问当前的 `Request` 和 `Response` 对象以及 `Logger` 类, 用于将信息保存到磁盘。

接下来, 有两个方法, 一个用于查看所有新闻项, 一个用于特定新闻项。

接下来, 使用函数*model()* 创建 `NewsModel` 实例。这是一个辅助函数。你可以在*全局函数和常量* 中阅读更多相关信息。如果不使用它, 也可以写 `$model = new NewsModel();`。

你可以看到 `$slug` 变量被传递到第二个方法中的模型方法。模型使用这个 slug 来标识要返回的新闻项。

完成 News::index() 方法

现在数据通过我们的模型被控制器检索, 但还没有显示任何内容。下一步要做的就是将这些数据传递给视图。将 `index()` 方法修改为如下所示:

```
<?php

namespace App\Controllers;
```

(续下页)

(接上页)

```
use App\Models\NewsModel;

class News extends BaseController
{
    public function index()
    {
        $model = model(NewsModel::class);

        $data = [
            'news_list' => $model->getNews(),
            'title'     => 'News archive',
        ];

        return view('templates/header', $data)
            . view('news/index')
            . view('templates/footer');
    }

    // ...
}
```

上面的代码从模型中获取所有新闻记录, 并赋值给一个变量。`$data['title']` 元素的值也被赋值, 所有数据被传递给视图。你现在需要创建一个视图来渲染新闻项。

创建 news/index 视图文件

在 **app/Views/news/index.php** 中创建并添加下一段代码。

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

<?php if ($news_list !== []): ?>

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

        <h3><?= esc($news_item['title']) ?></h3>
```

(续下页)

(接上页)

```
        <div class="main">
            <?= esc($news_item['body']) ?>
        </div>
        <p><a href="/news/<?= esc($news_item['slug'], 'url') ?>">
↪View article</a></p>

    <?php endforeach ?>

<?php else: ?>

    <h3>No News</h3>

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

<?php endif ?>
```

备注: 我们再次使用*esc()* 来帮助防止 XSS 攻击。但这次我们还传递了 "url" 作为第二个参数。这是因为攻击模式会根据输出使用的上下文而有所不同。

在这里, 每个新闻条目被循环并显示给用户。你可以看到我们使用 PHP 混合 HTML 编写了模板。如果你更喜欢使用模板语言, 可以使用 CodeIgniter 的视图解析器 或第三方解析器。

完成 News::show() 方法

新闻概述页面现已完成, 但是仍缺少页面来显示单个新闻条目。早期创建的模型以这样的方式制作, 可以轻松地用于此功能。你只需要添加一些控制器代码并创建一个新视图。返回 `News` 控制器并使用以下内容更新 `show()` 方法:

```
<?php

namespace App\Controllers;

use App\Models\NewsModel;
use CodeIgniter\Exceptions\PageNotFoundException;
```

(续下页)

(接上页)

```
class News extends BaseController
{
    // ...

    public function show(?string $slug = null)
    {
        $model = model(NewsModel::class);

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

        if ($data['news'] === null) {
            throw new PageNotFoundException('Cannot find the news␣
↪item: ' . $slug);
        }

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

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

不要忘记添加 `use CodeIgniter\Exceptions\PageNotFoundException;` 来导入 `PageNotFoundException` 类。

与不带参数调用 `getNews()` 方法不同, 传递了 `$slug` 变量, 因此它将返回特定的新闻条目。

创建 news/view 视图文件

唯一剩下的就是在 **app/Views/news/view.php** 中创建相应的视图。在此文件中放入以下代码。

```
<h2><?= esc($news['title']) ?></h2>
<p><?= esc($news['body']) ?></p>
```

将浏览器指向你的“news”页面，即 **localhost:8080/news**，你应该会看到新闻项目的列表，每个项目都有一个链接，可以显示单独的文章。

创建新闻

- *启用 CSRF 过滤器*
- *添加路由规则*
- *创建表单*
  - *创建 news/create 视图文件*
  - *News 控制器*
    - *添加 News::new() 方法以显示表单*
    - *添加 News::create() 以创建新闻项目*
- *更新 NewsModel*

- 创建新闻项目
- 恭喜

你现在已经知道如何使用 CodeIgniter 从数据库中读取数据, 但是你还没有将任何信息写入数据库。在本节中, 你将扩展之前创建的新闻控制器和模型, 以包括此功能。

启用 CSRF 过滤器

在创建表单之前, 让我们启用 CSRF 保护。

打开 **app/Config/Filters.php** 文件, 并如下更新 `$methods` 属性:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public array $methods = [
        'POST' => ['csrf'],
    ];

    // ...
}
```

它将配置 CSRF 过滤器以对所有 **POST** 请求启用。你可以在*Security* 库中阅读更多关于 CSRF 保护的信息。

> **警告:** 一般来说, 如果你使用 `$methods` 过滤器, 你应该禁用自动路由 (传统), 因为自动路由 (传统版) 允许任何 HTTP 方法访问控制器。使用你不期望的方法访问控制器可能会绕过过滤器。

添加路由规则

在你开始向 CodeIgniter 应用程序中添加新闻项目之前，你需要在 **app/Config/Routes.php** 文件中添加一个额外的规则。确保你的文件包含以下内容：

```
<?php

// ...

use App\Controllers\News;
use App\Controllers\Pages;

$routes->get('news', [News::class, 'index']);
$routes->get('news/new', [News::class, 'new']); // Add this line
$routes->post('news', [News::class, 'create']); // Add this line
$routes->get('news/(:segment)', [News::class, 'show']);

$routes->get('pages', [Pages::class, 'index']);
$routes->get('(:segment)', [Pages::class, 'view']);
```

`'news/new'` 的路由指令放置在 `'news/(:segment)'` 的指令之前，以确保显示创建新闻项目的表单。

`$routes->post()` 行定义了一个 POST 请求的路由器。它仅匹配 URI 路径 **/news** 的 POST 请求，并映射到 `News` 类的 `create()` 方法。

你可以在设置路由规则 中了解更多关于不同路由类型的信息。

创建表单

创建 news/create 视图文件

为了将数据输入数据库，你需要创建一个表单，在表单中你可以输入要存储的信息。这意味着你需要一个带有两个字段的表单，一个用于标题，一个用于文本。我们会在模型中从标题中派生 slug。

在 **app/Views/news/create.php** 中创建一个新的视图：

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

(续下页)

(接上页)

```
<?= session()->getFlashdata('error') ?>
<?= validation_list_errors() ?>

<form action="/news" method="post">
    <?= csrf_field() ?>

    <label for="title">Title</label>
    <input type="input" name="title" value="<?= set_value('title') ?
↪>">
    <br>

    <label for="body">Text</label>
    <textarea name="body" cols="45" rows="4"><?= set_value('body') ?
↪></textarea>
    <br>

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

这里可能只有四件事看起来不太熟悉。

session() 函数用于获取 Session 对象, `session()->getFlashdata('error')` 用于向用户显示与 CSRF 保护相关的错误。但是, 默认情况下, 如果 CSRF 验证检查失败, 将抛出异常, 所以它现在还不起作用。有关更多信息, 请参阅*失败时重定向*。

validation_list_errors() 函数由*表单辅助函数* 提供, 用于报告与表单验证相关的错误。

csrf_field() 函数创建一个带有 CSRF 令牌的隐藏输入, 有助于防止一些常见攻击。

set_value() 函数由*表单辅助函数* 提供, 用于在发生错误时显示旧输入数据。

News 控制器

返回到你的 News 控制器。

添加 News::new() 方法以显示表单

首先，创建一个方法来显示你创建的 HTML 表单。

```
<?php

namespace App\Controllers;

use App\Models\NewsModel;
use CodeIgniter\Exceptions\PageNotFoundException;

class News extends BaseController
{
    // ...

    public function new()
    {
        helper('form');

        return view('templates/header', ['title' => 'Create a news␣
↪item'])
            . view('news/create')
            . view('templates/footer');
    }
}
```

我们使用*helper()* 函数加载*Form 辅助函数*。大多数辅助函数在使用之前都需要加载辅助函数。

然后返回创建的表单视图。

添加 News::create() 以创建新闻项目

接下来，创建一个方法来根据提交的数据创建新闻项目。

在这里，你将完成三件事：

1. 检查提交的数据是否通过了验证规则。
2. 将新闻项目保存到数据库中。
3. 返回一个成功页面。

```
<?php

namespace App\Controllers;

use App\Models\NewsModel;
use CodeIgniter\Exceptions\PageNotFoundException;

class News extends BaseController
{
    // ...

    public function create()
    {
        helper('form');

        $data = $this->request->getPost(['title', 'body']);

        // Checks whether the submitted data passed the validation␣
↪rules.
        if (! $this->validateData($data, [
            'title' => 'required|max_length[255]|min_length[3]',
            'body'  => 'required|max_length[5000]|min_length[10]',
        ])) {
            // The validation fails, so returns the form.
            return $this->new();
        }

        // Gets the validated data.
        $post = $this->validator->getValidated();
```

(续下页)

(接上页)

```
        $model = model(NewsModel::class);

        $model->save([
            'title' => $post['title'],
            'slug'  => url_title($post['title'], '-', true),
            'body'  => $post['body'],
        ]);

        return view('templates/header', ['title' => 'Create a news␣
↪item'])
            . view('news/success')
            . view('templates/footer');
    }
}
```

上面的代码添加了很多功能。

获取数据

首先，我们使用由框架在控制器中设置的*IncomingRequest* 对象 `$this->request`。

我们从用户的 **POST** 数据中获取必要的项目，并将它们设置在 `$data` 变量中。

验证数据

接下来，你将使用由 Controller 提供的辅助函数*validateData()* 来验证提交的数据。在这种情况下，标题和正文字段是必需的，并且具有特定的长度。

如上所示，CodeIgniter 拥有一个强大的验证库。你可以阅读更多关于验证库 的信息。

如果验证失败，我们调用刚刚创建的 `new()` 方法并返回 HTML 表单。

保存新闻项目

如果验证通过了所有规则，我们通过*$this->validator->getValidated()* 获取验证后的数据，并将其设置在 `$post` 变量中。

加载并调用 `NewsModel`。这将负责将新闻项目传递给模型。根据是否找到与主键匹配的数组键，*save()* 方法会自动处理插入或更新记录。

这包含一个新函数*`url_title()`*。这个函数由*URL 辅助函数* 提供 - 它会剥离你传递给它的字符串, 用破折号 (-) 替换所有空格, 并确保所有内容都是小写。这会给你一个不错的 slug, 非常适合创建 URI。

返回成功页面

之后, 视图文件被加载并返回以显示成功消息。在 **app/Views/news/success.php** 中创建一个视图, 并编写成功消息。

这可以简单地写成:

```
<p>新闻项目创建成功。</p>
```

更新 NewsModel

唯一剩下的就是确保你的模型设置为允许数据被正确保存。在使用的 `save()` 方法将确定信息应插入还是如果行已经存在则应更新, 这取决于主键的存在。在这种情况下, 没有传递 `id` 字段给它, 所以它会在它的表格 `news` 中插入新行。

但是, 默认情况下, 模型中的 insert 和 update 方法实际上不会保存任何数据, 因为它不知道哪些字段是安全更新的。编辑 `NewsModel` 以在 `$allowedFields` 属性中为其提供可更新字段的列表。

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class NewsModel extends Model
{
```

(续下页)

(接上页)

```
    protected $table = 'news';

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

这个新属性现在包含我们允许保存到数据库的字段。请注意, 我们排除了 `id` 字段? 这是因为你几乎永远不需要这样做, 因为它是一个数据库中的自动递增字段。这有助于防止批量分配漏洞。如果你的模型处理了时间戳, 你也会排除它们。

创建新闻项目

现在指向你安装了 CodeIgniter 的本地开发环境的浏览器, 并在 URL 中添加 **/news/new**。添加一些新闻并查看你创建的不同页面。

恭喜

你刚刚完成了你的第一个 CodeIgniter4 应用程序!

下面的图表显示了你的项目的 **app** 文件夹，其中包含你创建或修改的所有文件。

```
app/
├── Config
│   ├── Filters.php (已修改)
│   └── Routes.php  (已修改)
├── Controllers
│   ├── News.php
│   └── Pages.php
├── Models
│   └── NewsModel.php
└── Views
    ├── news
    │   ├── create.php
    │   ├── index.php
    │   ├── success.php
```

(续下页)

(接上页)

```
    │   └── view.php
    ├── pages
    │   ├── about.php
    │   └── home.php
    └── templates
        ├── footer.php
        └── header.php
```

结束语

本教程没有覆盖你可能期待的完整内容管理系统的所有内容, 但是它向你介绍了路由、编写控制器和模型等更重要的主题。我们希望本教程能够让你对 CodeIgniter 一些基本的设计模式有所了解, 你可以在此基础上进行扩展。

现在你已经完成了本教程, 我们建议你查看文档的其余部分。CodeIgniter 经常因其全面的文档而受到赞扬。请你充分利用这一优势, 仔细阅读概述 和常规主题 部分。在需要时, 你应该参考类库 和辅助函数 参考。

任何有中级 PHP 编程经验的程序员都应该能在几天内掌握 CodeIgniter。

如果你对框架或自己的 CodeIgniter 代码还有疑问, 你可以:

- 查看我们的 论坛
- 加入我们的 Slack

3.1.2 启动并运行

安装 CodeIgniter

你可以从网站手动下载版本, 但是对于本教程, 我们将使用推荐的方法, 通过 Composer 安装 AppStarter 包。在命令行输入以下命令:

```
composer create-project codeigniter4/appstarter ci-news
```

这将创建一个新文件夹 **ci-news**, 其中包含你的应用程序代码,CodeIgniter 安装在 vendor 文件夹中。

设置开发模式

默认情况下,CodeIgniter 以生产模式启动。这是一个安全特性, 以防止在站点上线后设置被搞乱。所以首先让我们解决这个问题。复制或重命名 **env** 文件为 **.env**。打开它。

此文件包含服务器特定的设置。这意味着你永远不需要将任何敏感信息提交到版本控制系统。它已经包含一些你想要输入的常见设置, 不过都是注释掉的。因此, 取消注释带有 `CI_ENVIRONMENT` 的那行, 并将 `production` 更改为 `development`:

```
CI_ENVIRONMENT = development
```

运行开发服务器

搞定这些后, 是时候在浏览器中查看你的应用程序了。你可以通过任何你选择的服务器提供服务, 比如 Apache、Nginx 等, 但是 CodeIgniter 提供了一个简单的命令, 利用 PHP 的内置服务器快速在你的开发机器上启动并运行。在项目的根目录下在命令行输入以下命令:

```
php spark serve
```

3.1.3 欢迎页面

现在将浏览器指向正确的 URL, 你将看到欢迎屏幕。现在通过访问以下 URL 尝试一下:

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

你应该会看到以下页面:

这意味着你的应用程序可以工作了, 你可以开始对其进行更改了。

3.1.4 调试

调试工具栏

现在你处于开发模式, 你将在应用程序的右下角看到 CodeIgniter 火焰图标。点击它, 你将看到调试工具栏。

此工具栏包含许多在开发期间可以参考的有用项目。这在生产环境中永远不会显示。点击底部任意标签将弹出额外信息。点击工具栏右侧的 X 会将其最小化为一个小正方形, 带有 CodeIgniter 火焰图标。如果单击该图标, 工具栏将再次显示。

错误页面

除此之外, 如果在程序中遇到异常或其他错误,CodeIgniter 还具有一些有用的错误页面。打开 **app/Controllers/Home.php** 并更改某些行以生成错误 (删除分号或大括号应该可以起作用!)。你将看到一个类似下面的页面:

这里有几件事需要注意:

1. 将鼠标悬停在顶部的红色标题上会显示一个 **搜索**链接, 它会在新标签页中打开 DuckDuckGo.com 并搜索异常。
2. 在 Backtrace 中的任意一行上单击 **arguments** 链接会展开一个传入该函数调用的参数列表。

当你看到它时, 其他所有内容都应该很清楚。

现在我们知道如何开始和如何进行简单的调试, 让我们开始构建这个小新闻应用程序吧。

章节 4

概览和常规主题

4.1 CodeIgniter4 概览

以下页面描述了 CodeIgniter4 背后的架构概念:

4.1.1 应用程序结构

为了充分利用 CodeIgniter, 你需要了解默认情况下应用程序的结构, 以及可以更改什么来满足应用程序的需要。

- 默认目录
  - *app*
  - *system*
  - *public*
  - *writable*
  - *tests*
- 修改目录位置

默认目录

一个全新安装有 5 个目录:

- **app**
- **public**
- **writable**
- **tests**
- **vendor** 或 **system**

每个目录都有非常具体的作用。

app

app 目录是所有应用程序代码的存放位置。它具有默认的目录结构，适用于许多应用程序。

以下文件夹组成基本内容:

```
app/
    Config/          存储配置文件
    Controllers/     控制器确定程序流程
    Database/        存储数据库迁移和种子文件
    Filters/         存储可以在控制器之前和之后运行的过滤器类
    Helpers/         辅助函数存储独立函数的集合
    Language/        支持多语言会从这里读取语言字符串
    Libraries/       不适合其他类别的有用类
    Models/          模型与数据库一起工作来表示业务实体
    ThirdParty/      应用程序中可以使用的第三方库
    Views/           视图组成向客户端显示的 HTML
```

由于 **app** 目录已经有了命名空间，你可以随意修改此目录的结构以适应应用程序的需要。

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**，并添加一个新的 **Entities** 目录。

此目录下的所有文件都位于 `App` 命名空间下，尽管你可以在 **app/Config/Constants.php** 中自由更改命名空间。

system

备注: 如果使用 Composer 安装 CodeIgniter, **system** 位于 **vendor/codeigniter4/framework/system**。

此目录存储构成框架本身的文件。虽然你在如何使用应用程序目录方面有很大的灵活性, 但是不应修改 system 目录中的文件。相反, 你应该扩展类或创建新类以提供所需的功能。

此目录下的所有文件位于 `CodeIgniter` 命名空间下。

public

public 文件夹包含 web 应用程序的面向浏览器的部分, 防止直接访问源代码。

它包含主要的 **.htaccess** 文件、**index.php** 以及你添加的任何应用程序资源, 如 CSS、JavaScript 或图片。

此文件夹旨在成为站点的“网页根目录”, 你的 web 服务器会配置为指向它。

writable

此目录包含在应用程序生命周期中可能需要写入的任何目录。这包括用于缓存文件、日志和用户上传的任何目录。

你应该在这里添加应用程序需要写入的任何其他目录。这使你可以保持其他主目录不可写, 作为额外的安全措施。

tests

此目录设置为保存测试文件。**_support** 目录包含可在编写测试时使用的各种模拟类和其他实用程序。

此目录不需要传输到生产服务器。

修改目录位置

如果你已重新定位任何主目录, 可以在 **app/Config/Paths.php** 内更改配置设置。

请阅读管理你的应用程序。

4.1.2 模型、视图和控制器

- 什么是 *MVC?*
- 组件
  - 视图
  - 模型
  - 控制器

什么是 MVC?

无论何时创建应用程序, 都必须找到一种组织代码的方式, 以便轻松定位正确的文件并简化维护。与大多数 Web 框架一样,CodeIgniter 使用模型、视图、控制器 (MVC) 模式来组织文件。这将数据、表示和应用程序流保持为单独的部分。

应该指出, 对于每个元素的确切角色有许多不同看法, 但本文档描述了我们的看法。如果你有不同的看法, 你可以根据需要自由修改如何使用每个部分。

模型管理应用程序的数据, 并帮助执行应用程序可能需要的任何特殊业务规则。

视图是简单的文件, 几乎没有逻辑, 用于向用户显示信息。

控制器充当胶水代码, 在视图 (或看到它的用户) 和数据存储之间来回组织数据。

最基本意义上, 控制器和模型只是有特定任务的类。显然, 它们不是你可以使用的唯一类型的类, 但它们组成了此框架设计如何使用的核心。它们甚至在 **app** 目录中有指定的目录用于存储, 尽管只要命名空间正确, 你可以根据需要将它们存储在任何地方。我们将在下面更详细地讨论这一点。

让我们更仔细地看看这三个主要组件中的每个组件。

组件

视图

视图是最简单的文件, 通常是包含很少 PHP 的 HTML。PHP 应该非常简单, 通常只是显示变量的内容, 或者循环一些项目并在表中显示它们的信息。

视图从控制器获取要显示的数据, 控制器将其作为变量传递给视图, 然后可以用简单的 `echo` 调用显示。你也可以在视图中显示其他视图, 这样就可以很容易地在每个页面显示通用标头或页脚。

视图通常存储在 **app/Views** 中, 但是如果不以某种方式组织, 很快就会变得难以管理。CodeIgniter 不强制任何类型的组织, 但是一个好的经验法则是在 **Views** 目录中为每个控制器创建一个新目录。然后, 用方法名称命名视图。这样以后就很容易找到它们。例如, 用户个人资料可能在名为 `User` 的控制器以及名为 `profile` 的方法中显示。你可以将此方法的视图文件存储在 **app/Views/user/profile.php** 中。

这种组织方式作为一种基本习惯非常有效。有时候你可能需要以不同的方式组织。这没关系。只要 CodeIgniter 可以找到文件, 就可以显示它。

了解更多关于视图的信息

模型

模型的任务是为应用程序维护一种数据类型。这可能是用户、博客文章、事务等。在这种情况下, 模型的工作有两部分: 在从数据库中获取或放入数据库时, 对数据执行业务规则; 以及从数据库保存和检索数据。

对许多开发人员来说, 确定执行了哪些业务规则会造成困惑。这简单意味着任何对数据的限制或要求都由模型处理。这可能包括在保存以满足公司标准之前规范化原始数据, 或者以某种方式格式化列, 然后才将其交给控制器。通过在模型中保留这些业务要求, 你不会在多个控制器中重复代码, 并且不会意外地遗漏更新某个区域。

模型通常存储在 **app/Models** 中, 尽管它们可以使用命名空间按需要进行分组。

了解更多关于模型的信息

控制器

控制器有几个不同的角色。最明显的一点是它们从用户那里接收输入, 然后确定对其进行何种处理。这通常涉及将数据传递给模型以保存它, 或者从模型请求数据, 然后将其传递给视图以显示。如果需要, 这还包括加载其他实用程序类来处理模型范围之外的专门任务。

控制器的另一个职责是处理与 HTTP 请求相关的所有内容 - 重定向、身份验证、Web 安全、编码等。简而言之, 控制器是确保人们被允许访问那里, 并以他们可以使用的格式获取所需数据的地方。

控制器通常存储在 **app/Controllers** 中, 尽管它们可以使用命名空间按需要分组。

了解更多关于控制器的信息

4.1.3 自动加载文件

- *CodeIgniter4 自动加载器*
- 配置
  - 命名空间
  - 确认命名空间
  - 应用程序命名空间
  - 类映射
- *Composer 支持*
  - 加载器的优先级
- *FileLocator 缓存*
  - 工作原理
  - 如何删除缓存数据
  - 如何启用 *FileLocator* 缓存

每个应用程序由许多不同位置的大量类组成。框架为核心功能提供类。你的应用程序将具有许多库、模型和其他实体才能正常工作。你可能会使用第三方类。跟踪每个文件的位置, 并在一系列 `require()` 中硬编码该位置是一个巨大的头疼且极易出错。这就是自动加载器的用武之地。

CodeIgniter4 自动加载器

CodeIgniter 提供了一个非常灵活的自动加载器，只需进行很少的配置即可使用。它可以定位符合 PSR-4 自动加载目录结构的各个命名空间类。

自动加载器本身工作良好, 但也可以与其他自动加载器 (如 Composer) 一起使用, 如果需要的话, 甚至可以与你自己的自定义自动加载器一起使用。因为它们都通过 spl_autoload_register 注册, 所以它们顺序工作, 不会相互干扰。

自动加载器始终处于活动状态, 在框架执行开始时通过 `spl_autoload_register()` 注册。

重要: 你应该始终小心文件名的大小写。许多开发人员在 Windows 或 macOS 上使用不区分大小写的文件系统开发。然而, 大多数服务器环境使用区分大小写的文件系统。如果文件名大小写不正确, 自动加载程序无法在服务器上找到该文件。

配置

初始配置在 **app/Config/Autoload.php** 中完成。该文件包含两个主要数组: 一个用于类映射, 一个用于 PSR-4 兼容命名空间。

命名空间

组织类的推荐方法是为应用程序文件创建一个或多个命名空间。

配置文件中的 `$psr4` 数组允许你将命名空间映射到可以找到这些类的目录:

```
<?php

namespace Config;

use CodeIgniter\Config\AutoloadConfig;

class Autoload extends AutoloadConfig
{
    // ...
    public $psr4 = [
```

(续下页)

(接上页)

```
        APP_NAMESPACE => APPPATH,
    ];

    // ...
}
```

每行的键是命名空间本身。这个不需要尾部反斜杠。值是可以找到类的目录位置。

默认情况下，命名空间 `App` 位于 **app** 目录中，命名空间 `Config` 位于 **app/Config** 目录中。

如果你在这些位置根据 PSR-4 创建类文件，自动加载器将自动加载它们。

确认命名空间

你可以使用 `spark namespaces` 命令检查命名空间配置:

```
php spark namespaces
```

应用程序命名空间

默认情况下, 应用程序目录被映射到 `App` 命名空间。你必须为应用程序目录中的控制器、库或模型添加命名空间, 它们将在 `App` 命名空间下被找到。

Config 命名空间

配置文件位于 `Config` 命名空间中，而不是你可能预期的 `App\Config` 中。这使得核心系统文件即使在应用命名空间发生变化时也能始终找到它们。

备注: 自 v4.5.3 appstarter 版本起，`Config\\` 命名空间已被添加到 **composer.json** 的 `autoload.psr-4` 中。

更改应用命名空间

你可以通过编辑 **app/Config/Constants.php** 文件并在 `APP_NAMESPACE` 设置下设置新的命名空间值来更改此命名空间:

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

如果你使用 Composer 自动加载器，你还需要在 **composer.json** 中更改 `App` 命名空间，然后运行 `composer dump-autoload`。

```
{
    ...
    "autoload": {
        "psr-4": {
            "App\\": "app/"     <-- Change
        },
        ...
    },
    ...
}
```

备注: 自 v4.5.0 appstarter 起，`App\\` 命名空间已被添加到 **composer.json** 的 `autoload.psr-4` 中。如果你的 **composer.json** 中没有此项，添加它可能会提升你应用的自动加载性能。

你需要修改引用当前命名空间的所有现有文件。

类映射

CodeIgniter 通过不通过文件系统进行额外的 `is_file()` 调用来获取系统最后的性能，广泛使用类映射。你可以使用类映射链接到未使用命名空间的第三方库:

如果你使用的第三方库不是 Composer 包且没有命名空间，你可以使用类映射（classmap）来加载这些类:

```
<?php
```

(续下页)

(接上页)

```
namespace Config;

use CodeIgniter\Config\AutoloadConfig;

class Autoload extends AutoloadConfig
{
    // ...
    public $classmap = [
        'Markdown' => APPPATH . 'ThirdParty/markdown.php',
    ];

    // ...
}
```

每行的键是你要定位的类的名称。值是定位它的路径。

Composer 支持

默认情况下会自动初始化 Composer 支持。

默认情况下，它会在 `ROOTPATH . 'vendor/autoload.php'` 查找 Composer 的自动加载文件。如果由于任何原因需要更改该文件的位置，可以修改 **app/Config/Constants.php** 中定义的值。

加载器的优先级

如果同一个命名空间在 CodeIgniter 和 Composer 中同时定义，Composer 的自动加载器将优先尝试定位文件。

备注： 在 v4.5.0 之前，如果同一个命名空间在 CodeIgniter 和 Composer 中同时定义，CodeIgniter 的自动加载器会优先尝试定位文件。

FileLocator 缓存

在 4.5.0 版本加入.

FileLocator 负责查找文件或从文件中获取类名，这无法通过 PHP 自动加载来实现。

为了提高其性能，FileLocator 缓存已经被实现。

工作原理

- 在析构时，如果缓存数据已更新，则将 FileLocator 找到的所有数据保存到缓存文件中。
- 如果有缓存数据可用，则在实例化时恢复缓存数据。

缓存数据会永久使用。

如何删除缓存数据

一旦存储，缓存数据将永不过期。

因此，如果你添加或删除文件，或者更改现有文件路径或命名空间，旧的缓存数据将被返回，你的应用可能无法正常工作。

在这种情况下，你必须手动删除缓存文件。如果你通过 Composer 添加了 CodeIgniter 包，你也需要删除缓存文件。

你可以使用 `spark cache:clear` 命令：

```
php spark cache:clear
```

或者直接删除 **writable/cache/FileLocatorCache** 文件。

备注： `spark optimize` 命令会清除缓存。

如何启用 FileLocator 缓存

在 **app/Config/Optimize.php** 中将以下属性设置为 `true`:

```
public bool $locatorCacheEnabled = true;
```

或者你可以使用 `spark optimize` 命令来启用它。

备注: 此属性无法通过环境变量 重写。

4.1.4 服务

- 简介
  - 什么是服务?
  - 为什么使用服务？
- 如何获取服务
  - 获取新实例
  - 便利函数
- 定义服务
  - 允许参数
  - 共享类
- 服务发现
  - 重置服务缓存

简介

什么是服务?

CodeIgniter 4 中的 **服务**提供了创建和共享新类实例的功能。它由 `Config\Services` 类实现。

CodeIgniter 的所有核心类都以“服务”提供。这仅仅意味着, 不是硬编码一个类名进行加载, 而是在一个非常简单的配置文件中定义要调用的类。这个文件充当工厂的角色, 用来创建所需类的新实例。

为什么使用服务？

一个快速的例子可能会让事情更清晰，所以想象一下你需要引入一个 Timer 类的实例。最简单的方法可能就是直接创建这个类的一个新的实例：

```
<?php

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

这工作得很好。直到你决定用一个不同的计时器类来替代它。也许这个类有一些默认计时器不提供的高级报告功能。为了做到这一点，现在你必须找到你应用程序中使用计时器类的所有位置。由于你可能已经将它们保留下来，以持续运行应用程序的性能日志，这可能是一个耗时且容易出错的处理方式。这就是服务派上用场的地方。

我们不是自己创建实例，而是让一个中心类为我们创建类的实例。这个类保持非常简单。它只包含我们想要作为服务使用的每个类的方法。该方法通常返回该类的 **共享实例**，并将任何它可能具有的依赖项传递给它。然后，我们将替换我们的计时器创建代码，使用调用这个全局函数或服务类的代码：

```
<?php

$timer = service('timer');

// The code above is the same as the code below.
$timer = \Config\Services::timer();
```

当你需要更改使用的实现时, 可以修改服务配置文件, 更改会自动传播到整个应用程序, 而你不需要做任何事情。现在你只需要利用任何新的功能, 就大功告成了。非常简单且不易出错。

备注: 建议只在控制器中创建服务。其他文件, 如模型和库应该通过构造函数或设置器方法传入依赖项。

如何获取服务

由于许多 CodeIgniter 类作为服务提供, 你可以像如下获取它们:

```
<?php

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

`$timer` 是一个 Timer 类的实例，如果你再次调用 `service('timer')`，你将会得到完全相同的实例。

服务通常返回类的共享实例。下面的代码在首次调用时创建一个 `CURLRequest` 实例。第二次调用返回完全相同的实例。

```
<?php

$options1 = [
    'baseURI' => 'http://example.com/api/v1/',
    'timeout' => 3,
];
$client1 = service('curlrequest', $options1);

$options2 = [
    'baseURI' => 'http://another.example.com/api/v2/',
    'timeout' => 10,
];
$client2 = service('curlrequest', $options2);
// $options2 does not work.
// $client2 is the exactly same instance as $client1.
```

因此, `$client2` 的参数 `$options2` 不起作用。它被忽略了。

获取新实例

如果你想获取 Timer 类的新实例, 需要向参数 `$getShared` 传递 `false`:

```
<?php

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

便利函数

提供了两个获取服务的函数。这些函数始终可用。

service()

第一个是 `service()`, 它返回所请求服务的新实例。唯一必需的参数是服务名称。这与 Services 文件中的方法名称相同, 总是返回类的一个共享实例, 所以多次调用函数应该始终返回相同的实例:

```
<?php

$logger = service('logger');

// The code above is the same as the code below.
$logger = \Config\Services::logger();
```

备注: 自 v4.5.0 起，当你没有向服务传递参数时，推荐使用全局函数 `service()`，因为性能有所提升。

如果创建方法需要其他参数, 可以在服务名称后传入:

```
<?php

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

// The code above is the same as the code below.
$renderer = \Config\Services::renderer(APPPATH . 'views/');
```

single_service()

第二个函数 `single_service()` 的工作方式与 `service()` 相同, 但返回类的新实例:

```
<?php

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

(续下页)

(接上页)

```
// The code above is the same as the code below.
$logger = \Config\Services::logger(false);
```

定义服务

为了使服务能够良好地工作, 你必须能够依赖于每个类具有一个恒定的 API 或 接口 来使用它。CodeIgniter 的几乎所有类都提供了它们要遵守的接口。当你想扩展或替换核心类时, 你只需要确保满足接口的要求, 你就会知道这些类是兼容的。

例如, `RouteCollection` 类实现了 `RouteCollectionInterface`。当你想要创建一个提供不同路由创建方式的替代类时, 你只需要创建一个实现 `RouteCollectionInterface` 的新类:

```
<?php

namespace App\Router;

use CodeIgniter\Router\RouteCollectionInterface;

class MyRouteCollection implements RouteCollectionInterface
{
    // Implement required methods here.
}
```

最后, 在 **app/Config/Services.php** 中添加 `routes()` 方法, 以创建 `MyRouteCollection` 的新实例, 而不是 `CodeIgniter\Router\RouteCollection`:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseService;

class Services extends BaseService
{
```

(续下页)

(接上页)

```
    // ...

    public static function routes()
    {
        return new \App\Router\MyRouteCollection(static::locator(),␣
↪config('Modules'));
    }
}
```

允许参数

在某些情况下, 你会希望在实例化时有选择地向类传递设置。由于 services 文件是一个非常简单的类, 因此很容易实现这一点。

一个很好的例子是 `renderer` 服务。默认情况下, 我们希望这个类能够在 `APPPATH . 'views/'` 中找到视图。我们希望开发人员有可能更改该路径, 但如果他们的需求需要的话。所以该类接受 `$viewPath` 作为构造函数参数。服务方法如下所示:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseService;

class Services extends BaseService
{
    // ...

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

这会在构造函数中设置默认路径, 但允许轻松更改它使用的路径:

```
<?php
```

(续下页)

(接上页)

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

共享类

有时候你需要要求只创建服务的单个实例。这可以通过在工厂方法中调用的 `getSharedInstance()` 方法轻松处理。这会处理检查实例是否已在类中创建和保存, 如果没有, 则创建一个新实例。所有工厂方法都提供 `$getShared = true` 作为最后一个参数。你也应该坚持使用该方法:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseService;

class Services extends BaseService
{
    // ...

    public static function routes($getShared = true)
    {
        if ($getShared) {
            return static::getSharedInstance('routes');
        }

        return new \App\Router\MyRouteCollection(static::locator(),␣
↪config('Modules'));
    }
}
```

服务发现

CodeIgniter 可以自动发现你可能在任何定义的命名空间中创建的任何 **Config/Services.php** 文件。这允许简单使用任何模块的 Services 文件。为了发现自定义 Services 文件, 它们必须满足以下要求:

- 其命名空间必须在 **app/Config/Autoload.php** 中定义
- 在命名空间内, 该文件必须位于 **Config/Services.php**
- 它必须扩展 `CodeIgniter\Config\BaseService`

一个小例子应该澄清这一点。

假设在项目的根目录中创建了一个新目录 **Blog**。这将保存带有控制器、模型等的 **Blog 模块**, 你想将其中一些类作为服务提供。第一步是创建一个新文件:**Blog/Config/Services.php**。该文件框架应该是:

```
<?php

namespace Blog\Config;

use CodeIgniter\Config\BaseService;

class Services extends BaseService
{
    public static function postManager()
    {
        // ...
    }
}
```

现在你可以像上面描述的那样使用此文件。当你想从任何控制器获取文章服务时, 只需使用框架的 `Config\Services` 类获取你的服务:

```
<?php

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

备注: 如果多个 Services 文件具有相同的方法名, 则返回找到的第一个实例。

重置服务缓存

在 4.6.0 版本加入.

当 Services 类在框架初始化过程的早期首次被调用时，通过自动发现找到的 Services 类会被缓存到一个属性中，并且后续不会更新。

如果后续动态加载了模块，且这些模块中包含 Services 类，则必须更新缓存。

可以通过运行 `Config\Services::resetServicesCache()` 来实现。这将清除缓存，并在需要时强制重新进行服务发现。

4.1.5 工厂

- 简介
  - 什么是工厂？
  - 与服务的区别
- 加载类
  - 加载一个类
- 便利函数
  - *config()*
  - *model()*
- 定义要加载的类名
- 工厂参数
- 工厂选项
- 工厂行为
  - 配置
  - *setOptions* 方法
  - 参数选项
- 配置缓存
  - 先决条件

– 工作原理
– 如何更新配置值
– 如何启用配置缓存

简介

什么是工厂?

与服务 一样, **工厂**是自动加载的扩展, 可以帮助保持代码简洁且高效, 而不需要在类之间传递对象实例。

工厂在以下几点上类似于 CodeIgniter 3 的 `$this->load`:

- 加载一个类
- 共享加载的类实例

简单来说, 工厂提供了一种常见的方式来创建类实例并从任何地方访问它。这是一种很好的方法来重用对象状态并减少在整个应用程序中保留多个实例加载的内存负载。

任何类都可以通过工厂加载, 但最好的例子是那些用于处理或传输公共数据的类。框架本身在内部使用工厂, 例如, 使用 `Config` 类时确保加载正确的配置。

与服务的区别

工厂需要一个具体的类名来实例化, 并且没有创建实例的代码。

因此, 工厂不适合创建一个需要许多依赖项的复杂实例, 并且你无法更改要返回的实例的类。

另一方面, 服务具有创建实例的代码, 所以它可以创建一个需要其他服务或类实例的复杂实例。获取服务时, 服务需要一个服务名称, 而不是一个类名, 所以可以在不更改客户端代码的情况下更改返回的实例。

加载类

加载一个类

以 **模型**为例。你可以通过使用 Factories 类的魔术静态方法 `Factories::models()` 访问特定于模型的工厂。

静态方法名为 *component*。

不带命名空间的类名

如果你传递一个不带命名空间的类名，Factories 首先会在 `App` 命名空间中搜索与魔术静态方法名对应的路径。`Factories::models()` 会搜索 **app/Models** 目录。

传递短类名

在下面的代码中，如果你有 `App\Models\UserModel`，将返回该实例：

```
<?php

use CodeIgniter\Config\Factories;

$users = Factories::models('UserModel');
```

如果没有 `App\Models\UserModel`，它会在所有命名空间中搜索 `Models\UserModel`。

下次你在代码中的任何地方请求相同的类时，Factories 将确保你获得之前的实例：

```
<?php

use CodeIgniter\Config\Factories;

class SomeOtherClass
{
    public function someFunction()
    {
        $users = Factories::models('UserModel');
```

(续下页)

(接上页)

```
        // ...
    }
}
```

传递带有子目录的短类名

如果要加载子目录中的类，可以使用 / 作为分隔符。如果存在，以下代码将加载 **app/Libraries/Sub/SubLib.php**：

```
use CodeIgniter\Config\Factories;

$lib = Factories::libraries('Sub/SubLib');
```

传递完全限定类名

你还可以请求一个完全限定的类名：

```
use CodeIgniter\Config\Factories;

$users = Factories::models('Blog\Models\UserModel');
// Or
$users = Factories::models(\Blog\Models\UserModel::class);
```

如果存在，它将返回 `Blog\Models\UserModel` 的实例。

备注: 在 v4.4.0 之前，当你请求一个完全限定的类名时，如果只有 `Blog\Models\UserModel`，将返回该实例。但是，如果同时存在 `App\Models\UserModel` 和 `Blog\Models\UserModel`，将返回 `App\Models\UserModel` 的实例。

如果你想获取 `Blog\Models\UserModel`，你需要禁用选项 `preferApp`：

```
use CodeIgniter\Config\Factories;

$users = Factories::models('Blog\Models\UserModel', ['preferApp' =>␣
↪false]);
```

便利函数

为工厂提供了两个快捷函数。这些函数始终可用。

config()

第一个是*config()*, 它返回一个新的 Config 类实例。唯一必需的参数是类名称:

```
<?php

$appConfig = config('App');

// The code above is the same as the code below.
$appConfig = \CodeIgniter\Config\Factories::config('App');
```

model()

第二个函数*model()* 返回一个新的模型类实例。唯一必需的参数是类名称:

```
<?php

$user = model('UserModel');

// The code above is the same as the code below.
$user = \CodeIgniter\Config\Factories::models('UserModel');
```

定义要加载的类名

在 4.4.0 版本加入.

你可以使用 `Factories::define()` 方法定义在加载类之前要加载的类名:

```
use CodeIgniter\Config\Factories;

Factories::define('models', 'Myth\Auth\Models\UserModel', 'App\
↪Models\UserModel');
```

第一个参数是组件。第二个参数是类别名（Factories 魔术静态方法的第一个参数），第三个参数是要加载的真实完全限定类名。

之后，如果使用 Factories 加载 `Myth\Auth\Models\UserModel`，将返回 `App\Models\UserModel` 的实例：

```
$users = model('Myth\Auth\Models\UserModel');
```

工厂参数

`Factories` 的第二个参数是一个选项值数组 (如下所述)。这些指令将覆盖为每个组件配置的默认选项。

同时传递的任何更多参数将转发到类构造函数, 使你可以即时配置类实例。例如, 假设你的应用使用单独的数据库进行身份验证, 并且你希望确保尝试访问用户记录的任何尝试都通过该连接:

```
<?php

use CodeIgniter\Config\Factories;

$conn  = db_connect('auth');
$users = Factories::models('UserModel', [], $conn);
```

现在从 `Factories` 加载的 `UserModel` 每次实际上都会返回使用备用数据库连接的类实例。

工厂选项

默认行为可能不适用于每个组件。例如, 假设你的组件名称及其路径不匹配, 或者你需要将实例限制为某种类型的类。每个组件都接受一组选项来指导发现和实例化。

键	类型	描述	默认值
component	string 或 null	组件名称 (如果与静态方法不同)。这可以用于将一个组件别名到另一个。	`null` (默认为组件名称)
path	string 或 null	命名空间/文件夹内要查找类的相对路径。	`null` (默认为组件名称, 但将首字母大写)
instanceOf	string 或 null	要匹配返回实例上的必需类名称。	`null` (无过滤)
getShared	boolean	是否返回类的共享实例或者加载一个新实例。	`true`
preferApp	boolean	是否优先使用 App 命名空间中具有相同基本名称的类而不是其他明确的类请求。	`true`

备注: 自 v4.4.0 起, `preferApp` 仅在你请求不带命名空间的类名 时起作用。

工厂行为

可以通过三种方式 (按优先级降序排列) 应用选项:

- 配置类 `Config\Factory`, 其中包含与组件名称匹配的属性。
- 静态方法 `Factories::setOptions()`。
- 在调用时直接传递参数。

配置

要设置默认组件选项, 请在 **app/Config/Factory.php** 中创建一个新的 Config 文件, 以数组属性的形式提供与组件名称匹配的选项。

示例: 过滤器工厂

例如, 如果你要通过工厂创建 **过滤器**, 组件名称将是 `filters`。如果你想确保每个过滤器都是实现了 CodeIgniter 的 `FilterInterface` 的类的实例, 你的 **app/Config/Factory.php** 文件可能如下所示:

```
<?php

namespace Config;

use CodeIgniter\Config\Factory as BaseFactory;
use CodeIgniter\Filters\FilterInterface;

class Factory extends BaseFactory
{
    public $filters = [
        'instanceOf' => FilterInterface::class,
    ];
}
```

现在你可以用类似 `Factories::filters('SomeFilter')` 的代码创建过滤器, 并且返回的实例一定是 CodeIgniter 的过滤器。

这将防止第三方模块意外地在其命名空间中具有不相关的 `Filters` 路径而发生冲突。

示例: 库工厂

如果你想用 `Factories::library('SomeLib')` 在 **app/Libraries** 目录中加载库类, 路径 *Libraries* 与默认路径 *Library* 不同。

在这种情况下, 你的 **app/Config/Factory.php** 文件如下所示:

```
<?php
```

(续下页)

(接上页)

```
namespace Config;

use CodeIgniter\Config\Factory as BaseFactory;

class Factory extends BaseFactory
{
    public $library = [
        'path' => 'Libraries',
    ];
}
```

现在你可以使用 `Factories::library()` 方法加载你的库:

```
use CodeIgniter\Config\Factories;

$someLib = Factories::library('SomeLib');
```

setOptions 方法

`Factories` 类有一个静态方法允许运行时选项配置: 只需使用 `setOptions()` 方法提供所需的选项数组, 它们将与默认值合并并存储以备下次调用:

```
<?php

use CodeIgniter\Config\Factories;
use CodeIgniter\Filters\FilterInterface;

Factories::setOptions('filters', [
    'instanceOf' => FilterInterface::class,
    'prefersApp' => false,
]);
```

参数选项

Factories 的魔术静态调用以选项值数组作为第二个参数。这些指令将覆盖为每个组件配置的存储选项, 并可在调用时用于获得你所需的内容。输入应为以每个覆盖值为键的选项名称数组。

例如, 默认情况下 Factories 假设你希望定位组件的共享实例。通过向魔术静态调用添加第二个参数, 你可以控制该单个调用是否返回新实例还是共享实例:

```
use CodeIgniter\Config\Factories;

$users = Factories::models('UserModel', ['getShared' => true]);  //␣
↪Default; will always be the same instance
$other = Factories::models('UserModel', ['getShared' => false]); //␣
↪Will always create a new instance
```

配置缓存

在 4.4.0 版本加入.

重要: 除非你已经仔细阅读了这一节并理解了这个功能是如何工作的，否则不要使用这个功能。否则，你的应用程序将无法正常工作。

为了提高性能，实现了配置缓存。

先决条件

重要: 当不满足先决条件时使用此功能将阻止 CodeIgniter 正常运行。在这种情况下不要使用此功能。

- 要使用此功能，Factories 中实例化的所有 Config 对象的属性在实例化后不能被修改。换句话说，Config 类必须是不可变或只读的类。
- 默认情况下，每个被缓存的 Config 类必须实现 `__set_state()` 方法。

工作原理

重要: 一旦缓存，配置值在缓存被删除之前永远不会改变，即使配置文件或 **.env** 发生了变化。

- 如果 Factories 中的 Config 实例的状态发生变化，则在关闭之前将所有 Config 实例保存到缓存文件中。
- 如果有缓存可用，则在 CodeIgniter 初始化之前恢复缓存的 Config 实例。

简而言之，Factories 持有的所有 Config 实例在关闭之前都会被缓存，并且缓存的实例将永久使用。

如何更新配置值

一旦存储，缓存的版本将永不过期。更改现有的 Config 文件（或更改其环境变量）不会更新缓存或 Config 值。

因此，如果要更新 Config 值，请更新 Config 文件或其环境变量，并且必须手动删除缓存文件。

你可以使用 `spark cache:clear` 命令:

```
php spark cache:clear
```

或者直接删除 **writable/cache/FactoriesCache_config** 文件。

备注: 自 v4.5.0 起，`spark optimize` 命令会清除缓存。

如何启用配置缓存

在 4.5.0 版本加入.

在 **app/Config/Optimize.php** 中将以下属性设置为 `true`:

```
public bool $configCacheEnabled = true;
```

或者你可以使用 `spark optimize` 命令来启用此功能。

备注: 此属性无法通过环境变量 重写。

备注: 在 v4.4.x 中，请在 **public/index.php** 中取消以下代码的注释:

```
--- a/public/index.php
+++ b/public/index.php
@@ -49,8 +49,8 @@ if (! defined('ENVIRONMENT')) {
 }

 // Load Config Cache
-// $factoriesCache = new \CodeIgniter\Cache\FactoriesCache();
-// $factoriesCache->load('config');
+$factoriesCache = new \CodeIgniter\Cache\FactoriesCache();
+$factoriesCache->load('config');
 // ^^^ Uncomment these lines if you want to use Config Caching.

 /*
@@ -79,7 +79,7 @@ $app->setContext($context);
 $app->run();

 // Save Config Cache
-// $factoriesCache->save('config');
+$factoriesCache->save('config');
 // ^^^ Uncomment this line if you want to use Config Caching.

 // Exits the application, setting the exit code for CLI-based␣
↪applications
```

4.1.6 处理 HTTP 请求

为了充分利用 CodeIgniter, 你需要对 HTTP 请求和响应的工作原理有基本的了解。由于这是你在开发 Web 应用程序时使用的内容, 所以所有想要成功的开发人员都必须了解 HTTP 背后的概念。

本章的第一部分概述了概念。在概念说明清楚之后, 我们将讨论在 CodeIgniter 中如何处理请求和响应。

- *什么是 HTTP?*
  - *请求*
  - *响应*
- *使用请求和响应*

什么是 HTTP?

HTTP 简单来说就是一种基于文本的约定, 允许两台机器互相通信。当浏览器请求一个页面时, 它会问服务器是否可以获取该页面。然后服务器准备好页面, 并向请求它的浏览器发送响应。基本上就这么简单。显然, 你可以使用一些复杂的功能, 但基础真的非常简单。

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)。你开发 Web 应用程序的目标是始终了解浏览器正在请求什么, 并能够做出适当的响应。

请求

每当客户端 (Web 浏览器、智能手机应用等) 发出请求时, 它都会向服务器发送一小段文本消息并等待响应。

请求看起来类似这样:

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

此消息显示了客户端请求所需的所有信息。它告诉请求的方法 (GET、POST、DELETE 等) 和它支持的 HTTP 版本。

请求还包括可以包含广泛信息的可选请求头, 例如客户端希望以什么语言显示内容, 客户端接受的格式类型等等。如果你想查看一下,Wikipedia 有一篇文章列出了 所有标题字段。

响应

一旦服务器接收到请求, 你的应用程序将获取该信息并生成一些输出。服务器将你的输出作为其对客户端的响应的一部分进行打包。这也表示为类似这样的简单文本消息:

```
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>
```

响应告诉客户端它使用的 HTTP 规范版本, 可能最重要的是状态码 (200)。状态码是对客户端具有非常特定含义的标准化代码之一。这可以告诉它请求成功 (200), 或者页面未找到 (404)。如果你想查看完整的 HTTP 状态码列表, 请访问 IANA 的 完整 HTTP 状态码列表。

使用请求和响应

虽然 PHP 提供了与请求和响应标头交互的方式, 但 CodeIgniter 和大多数框架一样, 将它们抽象化, 以便为它们提供一致、简单的接口。*IncomingRequest 类* 是 HTTP 请求的面向对象表示。它提供了你需要的一切:

```
<?php

use CodeIgniter\HTTP\IncomingRequest;

$request = request();
```

(续下页)

(接上页)

```
// the URI path being requested (i.e., /about)
$request->getUri()->getPath();

// Retrieve $_GET and $_POST variables
$request->getGet('foo');
$request->getPost('foo');

// Retrieve from $_REQUEST which should include
// both $_GET and $_POST contents
$request->getVar('foo');

// Retrieve JSON from AJAX calls
$request->getJSON();

// Retrieve server variables
$request->getServer('Host');

// Retrieve an HTTP Request header, with case-insensitive names
$request->header('host');
$request->header('Content-Type');

// Checks the HTTP method
$request->is('get');
$request->is('post');
```

请求类会默默地为你完成很多工作, 你永远不需要担心。`isAJAX()` 和 `isSecure()` 方法检查了几种不同的方法来确定正确的答案。

备注: `isAJAX()` 方法依赖于 `X-Requested-With` 标头, 在某些情况下, 这在通过 JavaScript 发出的 XHR 请求中默认不会发送 (即 fetch)。请参阅*AJAX 请求* 部分了解如何避免此问题。

CodeIgniter 还提供了一个*Response 类*, 它是 HTTP 响应的面向对象表示。这为你构建对客户端的响应提供了一个简单而强大的方式:

```
<?php
```

(续下页)

(接上页)

```
use CodeIgniter\HTTP\Response;

$response = response();

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

// Sends the output to the browser
// This is typically handled by the framework
$response->send();
```

此外,Response 类允许你使用 HTTP 缓存层进行工作以获得最佳性能。

4.1.7 安全指南

我们非常重视安全。CodeIgniter 包含许多功能和技术，以便强制执行良好的安全实践，或者让你轻松地做到这一点。

我们尊重 开放 Web 应用安全项目 (OWASP) 并尽可能地遵循他们的建议。

以下内容来自于 OWASP Top Ten 和 OWASP API Security Top 10 识别出网络应用和 API 的主要漏洞。我们为每个漏洞提供一个简短的描述、OWASP 的建议，然后介绍 CodeIgniter 解决这些问题的措施。

- *OWASP Top 10 2021*
  - *A01:2021 访问控制失效*
  - *A02:2021 密码学失败*
  - *A03:2021 注入攻击*
  - *A04:2021 不安全设计*
  - *A05:2021 安全配置错误*
  - *A06:2021 漏洞和过时的组件*
  - *A07:2021 身份识别和认证失败*

  - *A08:2021 软件和数据完整性失败*
  - *A09:2021 安全日志记录和监控失败*
  - *A10:2021 服务器端请求伪造（SSRF）*
- *OWASP API Security Top 10 2023*
  - *API1:2023 对象级授权失效*
  - *API2:2023 认证失效*
  - *API3:2023 对象属性级授权失效*
  - *API4:2023 不受限制的资源消耗*
  - *API5:2023 功能级授权失效*
  - *API6:2023 不受限制访问敏感业务流程*
  - *API7:2023 服务端请求伪造*
  - *API8:2023 安全配置错误*
  - *API9:2023 不当的库存管理*
  - *API10:2023 不安全的 API 消费*

OWASP Top 10 2021

A01:2021 访问控制失效

访问控制执行策略，以确保用户不能在他们预期的权限之外进行操作。失败通常会导致未经授权的信息泄露、修改或销毁所有数据，或执行超出用户限制的业务功能。

常见的访问控制漏洞包括：

- 违反最小特权原则或默认拒绝原则，访问应该只授予特定功能、角色或用户，但对任何人开放。
- 通过修改 URL（参数篡改或强制浏览）、内部应用状态或 HTML 页面，或使用攻击工具修改 API 请求来绕过访问控制检查。
- 允许通过提供其唯一标识符来查看或编辑他人的帐户（不安全的直接对象引用）。
- 访问缺少 POST、PUT 和 DELETE 访问控制的 API。
- 权限提升。作为未登录的用户或作为已登录的用户时作为管理员操作。

- 元数据操作，例如重放或篡改 JSON Web 令牌 (JWT) 访问控制令牌、或操纵 cookie 或隐藏字段以提升权限或滥用 JWT 失效。
- CORS 配置错误允许从未经授权/未受信任的起源访问 API。
- 强制浏览到身份验证页面的未认证用户或标准用户的特权页面。

OWASP 建议

访问控制仅在受信任的服务器端代码或无服务器 API 中有效，在这些情况下，攻击者无法修改访问控制检查或元数据。

- 除了公共资源外，一律默认拒绝。
- 实现一次访问控制机制，并在整个应用中重用，包括最小化跨域资源共享 (CORS) 的使用。
- 模型访问控制应执行记录所有权，而不是接受用户可以创建、读取、更新或删除任何记录的情况。
- 应通过域模型强制执行独特的应用业务限制要求。
- 禁用 Web 服务器目录列表，并确保文件元数据（例如.git）和备用文件不在 Web 根目录中。
- 记录访问控制失败，在适当时向管理员发出警报（例如，重复失败）。
- 限制 API 和控制器访问，以最小化自动化攻击工具的危害。
- 在注销后在服务器上使状态会话标识符失效。无状态 JWT 令牌应该是短命的，以使攻击者的机会之窗最小化。对于更长寿命的 JWT，强烈建议遵循 OAuth 标准撤销访问。

CodeIgniter 的保护措施

- *Public* 文件夹，包括外部的应用和系统文件
- *数据验证* 库
- *Security* 库提供的*CSRF 保护*
- *Session 库* 库
- *限速器* 库用于限速
- *跨域资源共享 (CORS)* 过滤器

- 用于记录的`log_message()`函数
- 官方认证与授权框架*CodeIgniter Shield*
- 易于添加第三方认证

A02:2021 密码学失败

首先需要确定传输和静态数据的保护需要。例如，密码、信用卡号、健康记录、个人信息和商业机密需要额外保护，尤其是如果这些数据受到隐私法律（例如欧盟一般数据保护条例 (GDPR)）或法规（例如金融数据保护，如 PCI 数据安全标准 (PCI DSS)）的约束。在所有此类数据中：

- 有任何数据以明文传输吗？这涉及到诸如 HTTP, SMTP, FTP 等协议同样使用 TLS 升级如 STARTTLS。外部互联网流量是危险的。验证所有内部流量，例如负载均衡器、Web 服务器或后端系统之间的流量。
- 是否存在默认使用、弱加密算法或协议，或者旧代码中使用？
- 是否使用默认加密密钥，生成或重复使用弱加密密钥，或者缺乏适当的密钥管理或轮换？密钥是否已被提交到源代码库？
- 是否没有强制加密，例如，是否缺少任何 HTTP 头（浏览器）安全指令或头？
- 服务器证书和信任链是否正确验证？
- 初始化向量是否被忽略、重复使用，还是未为加密模式生成足够安全的初始化向量？是否在使用不安全的操作模式，例如 ECB 模式？当使用认证加密更为合适时，是否仅使用加密？
- 在缺乏基于密码的密钥派生函数的情况下，是否正在使用密码作为加密密钥？
- 随机性是否用于密码学目的，但未设计为满足密码学要求？即使选择了正确的函数，是否需要开发者播种，如果没有，开发者是否覆盖了其内建的强播种功能，使用了缺乏足够熵/不可预测性的种子？
- 是否使用了如 MD5 或 SHA1 等已废弃的哈希函数，或在需要密码学哈希函数时使用了非密码学哈希函数？
- 是否正在使用如 PKCS 编号 1 v1.5 等已废弃的密码学填充方法？
- 是否存在可被利用的密码学错误消息或侧信道信息，例如填充 Oracle 攻击中的形式？

OWASP 建议

至少执行以下操作，并参考相应文档：

- 对由应用程序处理、存储或传输的数据进行分类。根据隐私法、法规要求或业务需求，确定哪些数据是敏感的。
- 不要不必要地存储敏感数据。尽快丢弃它，或使用符合 PCI DSS 的令牌化或甚至截断。无法保留的数据不能被窃取。
- 确保所有敏感数据在静态时都加密。
- 确保使用最新的强标准算法、协议和密钥；使用适当的密钥管理。
- 用安全协议（如具有前向安全性 (FS) 密码的 TLS、服务器优先的密码序列和安全参数）加密所有传输中的数据。使用诸如 HTTP 严格传输安全 (HSTS) 等指令强制加密。
- 禁用包含敏感数据的响应缓存。
- 根据数据分类应用所需的安全控制。
- 不使用旧协议如 FTP 和 SMTP 传输敏感数据。
- 使用具有工作因子（延迟因子）强自适应和加盐哈希函数来存储密码，例如 Argon2、scrypt、bcrypt 或 PBKDF2。
- 初始化向量必须为操作模式选择合适的值。对于许多模式，这意味着使用 CSPRNG（密码学安全伪随机数生成器）。对于需要 nonce 的模式，初始化向量 (IV) 不需要 CSPRNG。在所有情况下，IV 不得在固定密钥的情况下重复使用。
- 始终使用认证加密而不仅仅是加密。
- 应密码学随机生成密钥，并以字节数组的形式存储在内存中。如果使用密码，则必须通过适当的密码基于密钥派生函数将其转换为密钥。
- 确保在适当的情况下使用密码学随机性，且没有以可预测的方式或低熵例播种。大多数现代 API 不需要开发者播种 CSPRNG 以获得安全性。
- 避免已弃用的密码学函数和填充方案，如 MD5、SHA1、PKCS 编号 1 v1.5。
- 独立验证配置和设置的有效性。

CodeIgniter 的保护措施

- 全局安全访问的配置 (`Config\App::$forceGlobalSecureRequests`)
- *force_https()* 函数
- *加密服务*
- 数据库配置中的*数据库配置* (`encrypt`)
- 官方认证与授权框架*CodeIgniter Shield*

A03:2021 注入攻击

当应用程序存在以下情况时，容易遭受攻击：

- 用户提供的数据没有经过应用程序验证、过滤或清理。
- 直接在解释器中使用动态查询或没有上下文感知转义的非参数化调用。
- 使用对象关系映射（ORM）搜索参数中的恶意数据来提取额外的敏感记录。
- 在动态查询、命令或存储过程中的结构和恶意数据直接使用或拼接恶意数据。

一些常见的注入类型包括 SQL、NoSQL、操作系统命令、对象关系映射（ORM）、LDAP 和表达式语言（EL）或对象图导航库（OGNL）注入。所有解释器中的概念都是相同的。源代码审查是检测应用程序是否易受注入攻击的最佳方法。强烈建议自动化测试所有参数、头信息、URL、Cookies、JSON、SOAP 和 XML 数据输入。组织可以将静态（SAST）、动态（DAST）和交互式（IAST）应用程序安全测试工具纳入 CI/CD 管道，以在生产部署前识别引入的注入漏洞。

OWASP 建议

防止注入攻击需要将数据与命令和查询分开：

- 首选选项是使用安全的 API，它完全避免使用解释器、提供参数化接口或迁移到对象关系映射工具（ORM）。
  - 注意：即使是参数化的存储过程，如果 PL/SQL 或 T-SQL 拼接查询和数据或使用 EXECUTE IMMEDIATE 或 exec() 执行恶意数据，也可能引入 SQL 注入。
- 使用正向服务器端输入验证。这不是一个完全的防御，因为许多应用程序需要使用特殊字符，如文本区域或移动应用程序的 API。

- 对于任何剩余的动态查询，使用特定解释器的转义语法来转义特殊字符。
  - 注意：无法转义 SQL 结构（如表名、列名等），因此用户提供的结构名是危险的。这在报告生成软件中是一个常见问题。
- 在查询中使用 LIMIT 和其他 SQL 控制，以防止在 SQL 注入的情况下大规模披露记录。

CodeIgniter 的保护措施

- *URI 安全性*
- *InvalidChars* 过滤器
- *数据验证* 库
- `esc()` 函数
- *HTTP library* 提供*input field filtering*
- 支持*内容安全策略*
- *查询构建器类*
- *Database escape methods*
- *查询绑定*

A04:2021 不安全设计

不安全设计是代表不同弱点的一个广泛类别，表述为“缺失或无效的控制设计”。不安全设计不是所有其他前 10 名风险类别的来源。我们需要区分不安全设计和不安全实现，它们的根本原因和补救措施不同。

一个安全的设计在实现上存在缺陷，可能会导致易于被利用的漏洞。而不安全的设计，即使有完美的实现，也无法弥补其缺陷，因为从定义上说，所需的安全控制从未被创建用于防御特定攻击。不安全设计的一个因素是缺乏在开发软件或系统时内在的业务风险评估，从而未能确定需要什么级别的安全设计。

OWASP 建议

- 建立并使用一个安全开发生命周期，与 AppSec 专业人员合作，帮助评估和设计与安全和隐私相关的控制
- 建立并使用一个安全设计模式库或预先准备好使用的组件
- 使用威胁建模来针对关键的认证、访问控制、业务逻辑和关键流程
- 将安全语言和控制集成到用户故事中
- 在应用程序的每一层（从前端到后端）集成合理性检查
- 编写单元和集成测试，验证所有关键流程是否能抵御威胁模型。编写每一层的用例和错误用例。
- 根据暴露和保护需求在系统和网络层分隔层级
- 在所有层级中通过设计来稳固地分隔租户
- 限制用户或服务的资源消耗

CodeIgniter 的保护措施

- *PHPUnit testing*
- 使用限速器 进行速率限制
- 一个官方的身份验证和授权框架*CodeIgniter Shield*

A05:2021 安全配置错误

如果应用程序存在以下情况，可能会暴露于安全风险中：

- 在应用程序堆栈的任何部分缺乏适当的安全加固，或云服务上配置不当的权限。
- 启用了或安装了不必要的功能（例如，不必要的端口、服务、页面、账户或权限）。
- 默认账户及其密码仍然启用且未更改。
- 错误处理向用户透露了堆栈跟踪或其他过度详细的错误信息。
- 对于升级系统，最新的安全功能被禁用或未安全配置。
- 应用服务器、应用框架（如 Struts、Spring、ASP.NET）、库、数据库等的安全设置未设定为安全值。

- 服务器没有发送安全头或指令，或它们未设定为安全值。
- 软件过时或存在漏洞（参见 A06:2021-漏洞和过时的组件）。

如果没有一个集中的、可重复的应用程序安全配置过程，系统将面临更高的风险。

OWASP 建议

应实施安全的安装流程，包括：

- 一个可重复的加固过程，使部署另一个适当锁定的环境变得快速且简单。开发、QA 和生产环境应全部按相同的方式配置，并在每个环境中使用不同的凭证。此过程应自动化，以尽量减少设置新安全环境所需的精力。
- 一个最小的平台，没有任何不必要的功能、组件、文档和示例。删除或不安装未使用的功能和框架。
- 将审查和更新配置作为补丁管理过程的一部分，适用于所有安全说明、更新和补丁（参见 A06:2021-漏洞和过时的组件）。审查云存储权限（例如 S3 存储桶权限）。
- 一个分段的应用程序架构提供了组件或租户之间的有效且安全的分离，使用分段、容器化或云安全组（ACL）。
- 向客户端发送安全指令，例如安全头信息。
- 一个自动化过程，在所有环境中验证配置和设置的有效性。

CodeIgniter 的保护措施

- *spark config:check* 命令
- *spark phpini:check* 命令
- 默认情况下使用生产模式
- *SecureHeaders* 过滤器

A06:2021 漏洞和过时的组件

当存在以下情况时，你可能会受到漏洞的影响：

- 如果你不知道所使用的所有组件（包括客户端和服务端）的版本。这包括你直接使用的组件以及嵌套依赖项。
- 如果软件存在漏洞、不再受支持或过期。这包括操作系统、Web/应用服务器、数据库管理系统（DBMS）、应用程序、API 以及所有组件、运行环境和库。
- 如果你没有定期扫描漏洞并订阅与所用组件相关的安全公告。
- 如果你没有基于风险在及时的基础上修复或升级底层平台、框架和依赖项。这在补丁管理是月度或季度任务的环境中很常见，会使组织在数天或数月内不必要地暴露于已修复的漏洞中。
- 如果软件开发人员没有测试更新、升级或打补丁的库的兼容性。
- 如果你没有保障组件的配置安全（参见 A05:2021-安全配置错误）。

OWASP 建议

应有一个补丁管理流程，以：

- 删除未使用的依赖项、不必要的功能、组件、文件和文档。
- 持续清点客户端和服务端组件（如框架、库）及其依赖项的版本，使用 tools like versions、OWASP Dependency Check、retire.js 等工具。持续监控如公共漏洞和暴露（CVE）和国家漏洞数据库（NVD）等来源，以查找组件中的漏洞。使用软件组成分析工具来自动化此过程。订阅电子邮件警报，以获取与所用组件相关的安全漏洞。
- 仅从官方来源通过安全链接获取组件。优先选择签名包以减少包含已修改的恶意组件的可能（参见 A08:2021-软件和数据完整性失败）。
- 监控未维护或不为旧版本创建安全补丁的库和组件。如果打补丁不可能，考虑部署虚拟补丁来监控、检测或防护发现的问题。

每个组织必须确保有一个持续的计划，用于在应用程序或组合的生命周期内监控、分类和应用更新或配置更改。

CodeIgniter 的保护措施

- 使用 Composer 升级

A07:2021 身份识别和认证失败

确认用户身份、认证和 Session 管理对防范与认证相关的攻击至关重要。如果应用程序存在以下情况，可能存在认证弱点：

- 允许自动攻击，例如凭证填充攻击，攻击者拥有一份有效的用户名和密码列表。
- 允许暴力破解或其他自动化攻击。
- 允许默认、弱或众所周知的密码，例如“Password1”或“admin/admin”。
- 使用弱或无效的凭证恢复和忘记密码流程，例如不能保障安全的“基于知识的答案”。
- 使用明文、加密或弱哈希的密码数据存储（参见 A02:2021-加密失败）。
- 缺失或无效的多因素认证。
- 在 URL 中暴露 Session 标识符。
- 成功登录后重用 Session 标识符。
- 未能正确使 Session ID 无效。用户 Session 或认证令牌（主要是单点登录（SSO）令牌）在注销或一段时间不活动期间未被正确使无效。

OWASP 建议

- 在可能的情况下，实现多因素认证，以防范自动化凭证填充、暴力破解和被盗凭证重复使用攻击。
- 不要使用任何默认凭证进行运输或部署，特别是对于管理员用户。
- 实施弱密码检查，例如针对最差的 10,000 个密码列表测试新或更改的密码。
- 根据国家标准与技术研究所（NIST）800-63b 第 5.1.1 节的记忆密码或其他现代、基于证据的密码策略，调整密码长度、复杂性和轮换策略。
- 确保注册、凭证恢复和 API 路径针对账户枚举攻击进行了加强，即对所有结果使用相同的消息。

- 限制或逐渐延迟失败的登录尝试，但要小心不要创建拒绝服务情景。记录所有失败并在检测到凭证填充、暴力破解或其他攻击时警告管理员。
- 使用服务器端的安全内置 Session 管理器，在登录后生成高熵的新随机 Session ID。Session 标识符不应在 URL 中出现，应安全存储，并在注销、空闲和绝对超时后使其失效。

CodeIgniter 的保护措施

- *Session* 库
- 官方的认证和授权框架*CodeIgniter Shield*

A08:2021 软件和数据完整性失败

软件和数据完整性失败涉及未能保护代码和基础设施免受完整性违反的影响。例如，当应用程序依赖来自不受信任来源、仓库和内容分发网络（CDNs）的插件、库或模块时，就可能存在问题。不安全的 CI/CD 管道可能引入未经授权的访问、恶意代码或系统泄露的潜在风险。

最后，许多应用程序现在包括自动更新功能，而这些更新在没有充分的完整性验证的情况下下载并应用于先前受信任的应用程序。攻击者可能会上传他们自己的更新，并分发和运行在所有安装中。

另一个例子是，当对象或数据被编码或序列化为攻击者可以看到和修改的结构时，存在不安全的反序列化风险。

OWASP 建议

- 使用数字签名或类似机制来验证软件或数据是否来自预期来源，并且未被修改。
- 确保库和依赖项（如 npm 或 Maven）使用受信任的仓库。如果你有更高的风险配置文件，请考虑托管一个经过验证的内部已知良好仓库。
- 确保使用软件供应链安全工具（如 OWASP Dependency Check 或 OWASP CycloneDX）来验证组件不包含已知漏洞。
- 确保对代码和配置更改进行审查过程，以尽量减少引入恶意代码或配置到软件管道中的可能性。

- 确保你的 CI/CD 管道具有适当的隔离、配置和访问控制，以确保代码在构建和部署过程中的完整性。
- 确保未签名或未加密的序列化数据未发送到不受信任的客户端，而不进行某种形式的完整性检查或数字签名，以检测序列化数据的篡改或重放。

CodeIgniter 的保护措施

- 不适用

A09:2021 安全日志记录和监控失败

此类别旨在帮助检测、升级和响应活动中的入侵。如果没有日志记录和监控，入侵将无法被检测到。以下情况下发生日志记录、检测、监控和主动响应不足的情况：

- 可审计事件（如登录、登录失败和高价值交易）未被记录。
- 警告和错误没有生成、生成不充分或不清晰的日志消息。
- 未监控应用程序和 API 的日志以发现可疑活动。
- 日志仅本地存储。
- 适当的警报阈值和响应升级过程没有到位或无效。
- 动态应用安全测试（DAST）工具（如 OWASP ZAP）的渗透测试和扫描未触发警报。
- 应用程序无法实时或接近实时检测、升级或警报活跃攻击。

如果将日志记录和警报事件暴露给用户或攻击者（参见 A01:2021-访问控制失效），你就可能会受到信息泄露的影响。

OWASP 建议

开发人员应根据应用程序的风险实施以下一些或所有控制：

- 确保所有登录、访问控制和服务器端输入验证失败事件都能够记录下来，并具有足够的用户上下文以识别可疑或恶意账户，并保存足够长的时间以允许延迟的法证分析。
- 确保生成的日志格式易于日志管理解决方案消费。

- 确保日志数据正确编码，以防止对日志记录或监控系统的注入或攻击。
- 确保高价值交易有完整性控制的审计追踪，以防止篡改或删除，如只追加数据库表或类似机制。
- DevSecOps 团队应建立有效的监控和警报系统，以便快速检测和响应可疑活动。
- 建立或采用事故响应和恢复计划，如国家标准与技术研究所（NIST）800-61r2 或更新版本。

存在商业和开源的应用保护框架，如 OWASP ModSecurity Core Rule Set，以及开源日志关联软件，如 Elasticsearch、Logstash、Kibana（ELK）堆栈，具有自定义仪表板和警报功能。

CodeIgniter 的保护措施

- *Logging* 库
- 官方的认证和授权框架*CodeIgniter Shield*

A10:2021 服务器端请求伪造（SSRF）

当一个 Web 应用程序在获取远程资源时未验证用户提供的 URL，就会发生 SSRF 漏洞。它允许攻击者强迫应用程序将构造的请求发送到意料之外的目的地，即使受到防火墙、VPN 或其他类型的网络访问控制列表（ACL）的保护。

由于现代 Web 应用程序为最终用户提供了便利的功能，获取 URL 成为了常见的情况。因此，SSRF 的发生率在增加。与此同时，由于云服务和架构的复杂性，SSRF 的严重性也在增加。

OWASP 建议

开发人员可以通过实施以下一些或所有的深度防御控制来预防 SSRF：

从网络层：

- 将远程资源访问功能划分到不同的网络中，以减少 SSRF 的影响
- 强制执行“默认拒绝”的防火墙策略或网络访问控制规则，以阻止所有不重要的内部网络流量。
  - 提示：

    * 基于应用程序建立防火墙规则的所有权和生命周期。
    * 记录防火墙上所有接受和阻止的网络流（参见 A09:2021-安全日志记录和监控失败）。

从应用层：

- 清理和验证所有客户端提供的输入数据
- 通过允许列表强制 URL 模式、端口和目的地
- 不要将原始响应发送给客户端
- 禁用 HTTP 重定向
- 注意 URL 一致性，以避免 DNS 重新绑定和“检查时与使用时”(TOCTOU) 竞争条件等攻击

不要通过使用拒绝列表或正则表达式来缓解 SSRF。攻击者拥有绕过拒绝列表的有效负载列表、工具和技能。

CodeIgniter 的保护措施

- *数据验证* 库
- *HTTP 库* 提供了*输入字段过滤*

OWASP API Security Top 10 2023

API1:2023 对象级授权失效

API 往往会暴露处理对象标识符的端点，这样会产生广泛的对象级访问控制问题。在每个使用来自用户的 ID 访问数据源的函数中，都应考虑对象级授权检查。

OWASP 建议

- 实施依赖于用户策略和层级的适当授权机制。
- 使用授权机制检查登录用户是否有权在每个使用客户端输入访问数据库记录的函数中执行请求的操作。
- 优先使用随机且不可预测的值作为记录 ID 的 GUID。
- 编写测试以评估授权机制的漏洞。不要部署使测试失败的更改。

CodeIgniter 的保护措施

- 官方的认证和授权框架*CodeIgniter Shield*
- *PHPUnit 测试*

API2:2023 认证失效

认证机制常常被错误地实现，使得攻击者能够破解认证令牌或利用实现缺陷临时或永久地假冒其他用户的身份。破坏系统识别客户端/用户的能力就会破坏 API 的整体安全性。

OWASP 建议

- 确保了解所有可能的 API 认证流程（移动应用/Web/实现单击认证的深层链接等）。询问你的工程师是否遗漏了某些流程。
- 了解你的认证机制。确保理解它们的用途和使用方法。OAuth 不是认证，API 密钥也不是。
- 不要在认证、令牌生成或密码存储方面重新发明轮子。使用标准。
- 凭据恢复/忘记密码端点应在防御暴力破解、速率限制和锁定保护方面与登录端点同等对待。
- 对于敏感操作（例如更改账户所有者的电子邮件地址/两因素认证电话号码），需要重新认证。
- 使用 OWASP 认证备忘清单。
- 尽可能实施多因素认证。
- 实施防暴力破解机制，以减轻凭据填充、字典攻击和针对认证端点的暴力破解攻击。此机制应比 API 上的常规速率限制机制更严格。
- 实施账号锁定/验证码机制，以防止针对特定用户的暴力破解攻击。实施弱密码检查。
- API 密钥不应用于用户认证。它们仅应用于 API 客户端认证。

CodeIgniter 的保护措施

- 控制器过滤器
- *spark* 路由 命令
- 官方的认证和授权框架*CodeIgniter Shield*
- 用于速率限制的限速器

API3:2023 对象属性级授权失效

这一类别结合了 API3:2019 的过度数据暴露和 API6:2019 的大量赋值问题，集中在根本原因：缺乏或不当的对象属性级授权验证。这导致了未经授权方的信息暴露或篡改。

OWASP 建议

- 当通过 API 端点暴露一个对象时，始终确保用户应该访问你暴露的对象属性。
- 避免使用诸如 to_json() 和 to_string() 之类的通用方法。相反，应选择具体的对象属性进行返回。
- 如果可能，避免使用自动将客户端输入绑定到代码变量、内部对象或对象属性的功能（“大量赋值”）。
- 仅允许客户端更新对象的特定属性。
- 实施基于模式的响应验证机制作为额外的安全层。作为该机制的一部分，定义并强制执行所有 API 方法返回的数据。
- 根据端点的业务/功能需求，将返回的数据结构保持在最低限度。

CodeIgniter 的保护措施

- 模型的*$allowedFields*
- 官方的认证和授权框架*CodeIgniter Shield*

API4:2023 不受限制的资源消耗

满足 API 请求需要诸如网络带宽、CPU、内存和存储等资源。其他诸如电子邮件/SMS/电话或生物信息验证等资源由服务提供商通过 API 集成提供，并按请求付费。成功的攻击可能导致拒绝服务或运营成本增加。

OWASP 建议

- 使用一种可以轻松限制内存、CPU、重启次数、文件描述符和进程（如容器/无服务器代码，例如 Lambdas）的方法。
- 定义并强制执行所有传入参数和负载数据的最大尺寸，例如字符串的最大长度、数组中的最大元素数和最大上传文件大小（无论存储在本地还是云存储中）。
- 实施限制客户端在定义时间范围内与 API 交互频率的机制（速率限制）。
- 速率限制应根据业务需求进行微调。某些 API 端点可能需要更严格的策略。
- 限制/节流单个 API 客户端/用户执行单个操作的次数或频率（例如验证一次性密码，或在不访问一次性 URL 的情况下请求密码恢复）。
- 添加适当的服务器端验证以控制查询字符串和请求体参数，尤其是那些控制响应中返回记录数量的参数。
- 为所有服务提供商/API 集成配置消费限制。如果无法设置消费限制，应配置帐单警报。

CodeIgniter 的保护措施

- 数据验证 库
- 用于速率限制的限速器

API5:2023 功能级授权失效

复杂的访问控制策略，包括不同的层级、组和角色，以及行政和常规功能之间不明确的分离，往往导致授权缺陷。通过利用这些问题，攻击者可以访问其他用户的资源和/或管理功能。

OWASP 建议

你的应用程序应有一个一致且易于分析的授权模块，该模块应从所有业务功能中调用。通常，这种保护是由应用代码外部的一个或多个组件提供的。

- 执行机制应该默认拒绝所有访问，需要对每个功能的访问显式授予特定角色。
- 根据功能级授权缺陷审查你的 API 端点，同时牢记应用程序的业务逻辑和组层次结构。
- 确保所有管理控制器都继承自实现基于用户组/角色的授权检查的管理抽象控制器。
- 确保常规控制器内的管理功能实现基于用户组和角色的授权检查。

CodeIgniter 的保护措施

- 控制器过滤器
- 官方的认证和授权框架*CodeIgniter Shield*

API6:2023 不受限制访问敏感业务流程

存在这种风险的 API 暴露了某些业务流程——例如购票或发布评论——而没有考虑到这些功能如果以自动化的方式被过度使用会对业务造成怎样的损害。这并不一定来源于实现上的漏洞。

OWASP 建议

缓解计划应该分两层进行：

- 业务层：识别出如果过度使用可能对业务造成损害的业务流程。
- 工程层：选择合适的保护机制来减轻业务风险。

一些保护机制相对简单，而另一些则更复杂。以下方法通常用于减缓自动化威胁：

- 设备指纹识别：拒绝意外客户端设备的服务（例如无头浏览器）往往会使威胁行为者使用更复杂的解决方案，从而增加其成本。
- 人类检测：使用验证码或更先进的生物识别解决方案（例如输入模式）。

- 非人类模式检测：分析用户流程以检测非人类模式（例如用户在不到一秒钟内访问了“添加到购物车”和“完成购买”功能）。
- 考虑阻止 Tor 出站节点和知名代理的 IP 地址。

保护并限制直接由机器使用的 API（如开发者和 B2B API）的访问。这些 API 往往是攻击者的容易目标，因为它们通常没有实现所有必要的保护机制。

CodeIgniter 的保护措施

- 不适用

API7:2023 服务端请求伪造

服务端请求伪造（SSRF）漏洞可能会在 API 获取远程资源时未对用户提供的 URI 进行验证时出现。这使得攻击者能够迫使应用程序向意外的目标发送伪造的请求，即使这些目标受防火墙或 VPN 保护。

OWASP 建议

- 在你的网络中隔离资源获取机制：通常这些功能是为了获取远程资源而不是内部资源。
- 在可能的情况下，使用白名单：
  - 用户预计下载资源的远程来源（例如 Google Drive, Gravatar 等）
  - URL 方案和端口
  - 给定功能的可接受媒体类型
- 禁用 HTTP 重定向。
- 使用经过良好测试和维护的 URL 解析器，以避免由于 URL 解析不一致引起的问题。
- 验证并清理所有客户端提供的输入数据。
- 不要向客户端发送原始响应。

CodeIgniter 的保护措施

- *数据验证* 库
- *HTTP library* 提供*输入字段过滤*
- *CURLRequest* 类
- *URI* 类

API8:2023 安全配置错误

API 及其支持系统通常包含复杂的配置，旨在使 API 更加可定制。软件和 DevOps 工程师可能会忽视这些配置，或在配置时未遵循安全最佳实践，从而为各种类型的攻击打开了大门。

OWASP 建议

API 生命周期应包括：

- 一个可重复的强化过程，以快速轻松地部署一个适当锁定的环境。
- 在整个 API 栈中审查和更新配置的任务。审查应包含：编排文件、API 组件和云服务（例如 S3 存储桶权限）。
- 一个自动化过程，持续评估所有环境中配置和设置的有效性。

此外：

- 确保所有从客户端到 API 服务器及任何上下游组件的 API 通信都在加密通信通道（TLS）上进行，无论它是内部 API 还是公开 API。
- 明确每个 API 可以访问的 HTTP 动词：应禁用所有其他 HTTP 动词（例如 HEAD）。
- 预计从基于浏览器的客户端（例如，Web 应用前端）访问的 API 应至少：
  - 实施适当的跨域资源共享（CORS）策略
  - 包含适用的安全头
- 将传入内容类型/数据格式限制为符合业务/功能要求的那些。
- 确保 HTTP 服务器链中的所有服务器（例如，负载均衡器、反向和正向代理、后端服务器）以一致的方式处理传入请求，以避免反序列化问题。

- 在适用的情况下，定义并强制执行所有 API 响应负载模式，包括错误响应，以防止异常跟踪和其他有价值的信息被返回给攻击者。

CodeIgniter 的保护措施

- 全局安全访问配置（`Config\App::$forceGlobalSecureRequests`）
- *force_https()* 函数
- *已定义路由*
- *自动路由（改进版）*
- *跨域资源共享 (CORS)* 过滤器

API9:2023 不当的库存管理

相比传统的 web 应用程序，API 往往暴露更多的端点，因此适当且更新的文档变得尤为重要。适当的主机和已部署 API 版本的库存管理也很重要，可以减轻诸如弃用 API 版本和暴露调试端点等问题。

OWASP 建议

- 清点所有 API 主机并记录每个主机的重要方面，关注 API 环境（例如生产、暂存、测试、开发），谁应具有对主机的网络访问权限（例如公共、内部、合作伙伴）以及 API 版本。
- 清点集成的服务并记录其重要方面，例如它们在系统中的角色、交换的数据（数据流）及其敏感性。
- 记录 API 的所有方面，例如认证、错误、重定向、速率限制、跨域资源共享（CORS）策略和端点，包括它们的参数、请求和响应。
- 通过采用开放标准自动生成文档。在你的 CI/CD 流水线中包含文档构建。
- 仅向授权使用 API 的人提供 API 文档。
- 对所有暴露的 API 版本使用外部保护措施（例如专门的 API 安全解决方案），而不仅限于当前的生产版本。
- 避免在非生产 API 部署中使用生产数据。如果无法避免，这些端点应得到与生产端点相同的安全处理。

- 当新版本的 API 包含安全改进时，进行风险分析，以通知旧版本所需的缓解措施。例如，是否可以在不破坏 API 兼容性的情况下向后移植改进，或需要迅速移除旧版本并强迫所有客户端迁移到最新版本。

CodeIgniter 的保护措施

- *spark 路由* 命令

API10:2023 不安全的 API 消费

开发者往往比起用户输入更信任从第三方 API 接收的数据，因此更倾向于采用较弱的安全标准。为了攻击 API，攻击者更可能是针对集成的第三方服务，而不是直接尝试攻击目标 API。

OWASP 建议

- 在评估服务提供商时，评估其 API 的安全状态。
- 确保所有 API 交互都通过安全通信通道（TLS）进行。
- 在使用从集成 API 接收到的数据之前，始终验证并适当清理这些数据。
- 保持一个集成 API 可能重定向到的已知位置白名单：不要盲目跟随重定向。

CodeIgniter 的保护措施

- *CURLRequest* 类
- *数据验证* 库

4.1.8 设计与架构目标

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

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

从技术和架构角度看，CodeIgniter 按照下列目标创建：

- **动态实例化**。在 CodeIgniter 中，组件的导入和函数的执行都是在被请求的时候才执行，而不是全局的。除核心资源外，系统不需要任何其他资源，因此系统默认是非常轻量级的。HTTP 请求所触发的事件以及你设计的控制器和视图将决定哪些资源是需要加载的。
- **低耦合**。耦合是指一个系统中组件之间的依赖程度。组件之间的依赖程度越低，系统的重用性和灵活性就越好。我们的目标就是打造一个低耦合的系统。
- **组件专一性**。专一性指的是组件对某个目标的专注程度。在 CodeIgniter 中，每一个类和方法都是高度独立的，从而可以最大程度地被复用。

CodeIgniter 是一个动态实例化，高度组件专一性的低耦合系统。它在小巧的基础上力求做到简单、灵活和高性能。

4.2 常规主题

4.2.1 配置

每个框架都使用配置文件来定义许多参数和初始设置。CodeIgniter 配置文件定义了简单的类, 其中所需的设置是公共属性。

与许多其他框架不同,CodeIgniter 的可配置项不包含在单个文件中。相反, 每个需要可配置项的类都有一个与使用它的类同名的配置文件。你可以在 **app/Config** 文件夹中找到应用程序配置文件。

- 什么是配置类？
- 使用配置文件
  - 获取配置对象
  - 获取配置属性
- 创建配置文件
- 环境变量
  - *Dotenv* 文件
- 配置类和环境变量
  - 作为数据的环境变量

- 将环境变量视为数组
- 处理不同环境
- 注册器
  - 隐式注册器
  - 显式注册器
- 确认配置值
  - *config:check*

什么是配置类？

配置类用于定义系统默认配置值。系统配置值通常是 * 静态 * 的。配置类旨在保留配置应用程序操作方式的设置，而不是响应每个用户的个别设置。

不建议在配置类实例化后，在执行期间修改值。换句话说，建议将配置类视为不可变或只读的类。这尤其重要，如果你使用配置缓存。

配置值可以在类文件中硬编码，也可以在实例化时从环境变量中获取。

使用配置文件

获取配置对象

你可以通过几种不同的方式访问类的配置文件。

使用 new 关键字

使用 `new` 关键字创建一个实例:

```
<?php

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

config()

使用*config()* 函数:

```
<?php

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

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

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

如果未提供命名空间，它将首先在 **app/Config** 文件夹中查找文件，如果找不到，则在所有定义的命名空间的 **Config** 文件夹中查找。

CodeIgniter 提供的所有配置文件都使用 `Config` 命名空间。在你的应用程序中使用这个命名空间将提供最佳性能, 因为它确切知道在哪里可以找到这些文件。

你可以通过使用不同的命名空间将配置文件放在任何你想要的文件夹中。这允许你在生产服务器上将配置文件放在一个不可公开访问的文件夹中, 同时在开发期间保持其位于 **/app** 下方便访问。

备注: 在 v4.4.0 之前，`config()` 会在有与 shortname 相同的类时，在 **app/Config/** 中查找文件，即使你指定了完全限定的类名，如 `config(\Acme\Blog\Config\Blog::class)`。在 v4.4.0 中修复了此行为，并返回指定的实例。

获取配置属性

所有配置对象属性都是公共的, 所以你可以像访问任何其他属性一样访问设置:

```
<?php

$config = config('Pager');
```

(续下页)

(接上页)

```
// Access settings as object properties
$pageSize = $config->perPage;
```

创建配置文件

当你需要一个新的配置时, 首先在所需位置创建一个新文件。默认文件位置 (大多数情况下推荐) 是 **app/Config**。该类应使用适当的命名空间, 并且它应扩展 `CodeIgniter\Config\BaseConfig` 以确保它可以接收特定环境的设置。

你可以通过使用不同的命名空间将配置文件放置在任何 **Config** 文件夹中。

该类应使用适当的命名空间，并应扩展 `CodeIgniter\Config\BaseConfig` 以确保它可以接收特定于环境的设置。

定义类并用代表你的设置的公共属性填充它:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

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

环境变量

今天应用程序设置的最佳实践之一是使用环境变量。原因之一是环境变量可以在不更改任何代码的情况下在部署之间轻松更改。配置在部署之间可能会有很大变化, 但代码不会。例如, 多个环境 (如开发者的本地机器和生产服务器) 通常需要针对每个特定设置配置不同的值。

环境变量也应该用于任何私人信息, 如密码、API 密钥或其他敏感数据。

Dotenv 文件

CodeIgniter 通过使用 “dotenv” 文件使设置环境变量变得简单轻松。该术语来源于文件名, 文件名以点号开头, 然后是 “env” 文本。

创建 Dotenv 文件

CodeIgniter 期望 **.env** 文件与 **app** 目录一起位于项目的根目录中。CodeIgniter 中分发了一个位于项目根目录 named 的模板文件 **env** (注意开头没有点号 (`.`)?)。

它包含了项目可能会使用的大量变量, 并分配了空、虚拟或默认值。你可以通过重命名模板为 **.env** 或复制为名为 **.env** 的副本, 将此文件用作应用程序的起点。

> **警告:** 确保 **.env** 文件 NOT 被你的版本控制系统跟踪。对于 *git* 来说, 这意味着将其添加到 **.gitignore**。否则可能会导致敏感证书被公开。

设置变量

设置以简单的名称/值对的集合存储在 **.env** 文件中, 用等号分隔。

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

当你的应用程序运行时, **.env** 将自动加载, 并将变量放入环境中。如果环境中已经存在一个变量, 它将不会被覆盖。

获取变量

加载的环境变量可以使用下列任意一种访问: `getenv()`、`$_SERVER` 或 `$_ENV`。

```
<?php

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

> **警告:** 请注意, 来自 **.env** 文件的设置会添加到 `$_SERVER` and `$_ENV` 中。由此带来的一个副作用是, 如果你的 CodeIgniter 应用程序生成一个 `var_dump($_ENV)` 或 `phpinfo()` (用于调试或其他有效原因), 或者在 `development` 环境中显示了详细的错误报告, **你的安全凭据可能会公开暴露**。

嵌套变量

为了省去输入, 你可以通过在 `${...}` 内包装变量名来重用已经在文件中指定的变量:

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

命名空间变量

有时你会有多个同名变量。系统需要一种方法来确定应使用的正确设置。这通过为变量 "命名空间" 来解决这个问题。

命名空间变量使用点表示法来限定变量名, 以便在合并到环境时它们是唯一的。这是通过在变量名称前面包含区别前缀和点号 (.) 来完成的。

```
// 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
```

命名空间分隔符

某些环境, 例如 Docker、CloudFormation 不允许带点号 (.) 的变量名。在这种情况下, 从 v4.1.5 开始, 你也可以使用下划线 (_) 作为分隔符。

```
// 使用下划线的命名空间变量
app_forceGlobalSecureRequests = true
app_CSPEnabled = true
```

配置类和环境变量

当你实例化一个配置类时, 任何 命名空间环境变量都会被考虑合并到配置对象的属性中。

重要: 你无法通过设置环境变量来添加新属性, 也不能将标量值改变为数组。请参见作为数据的环境变量。

备注: 此功能是在 `CodeIgniter\Config\BaseConfig` 类中实现的。因此，它不适用于 **app/Config** 文件夹中的一些文件，这些文件不扩展该类。

如果命名空间变量的前缀正好匹配配置类的命名空间, 那么设置的尾部 (点之后) 将被视为配置属性。如果它与现有的配置属性匹配, 环境变量的值将替换配置文件中相应的值。如果没有匹配, 配置类属性保持不变。在此用法中, 前缀必须是类的完整 (区分大小写) 命名空间。

```
Config\App.forceGlobalSecureRequests = true
Config\App.CSPEnabled = true
```

备注: 命名空间前缀和属性名均区分大小写。它们必须完全匹配配置类文件中定义的完整命名空间和属性名称。

使用仅包含配置类名称的小写版本的 短前缀相同。如果短前缀匹配类名, 则 **.env** 中的值将替换配置文件中的值。

```
app.forceGlobalSecureRequests = true
app.CSPEnabled = true
```

从 v4.1.5 开始, 你也可以使用下划线:

```
app_forceGlobalSecureRequests = true
app_CSPEnabled = true
```

备注: 使用 短前缀时, 属性名称仍必须完全匹配类中定义的名称。

作为数据的环境变量

务必要始终记住，你的 **.env** 文件中的环境变量 **只是现有标量值的替代**。

简单来说，你只能通过在 **.env** 文件中设置来更改 Config 类中存在的属性的标量值。

1. 你不能添加 Config 类中未定义的属性。
2. 你不能将属性中的标量值更改为数组。
3. 你不能向现有数组中添加元素。

例如, 你不能只是在 **.env** 中放置 `app.myNewConfig = foo` 并期望你的 `Config\App` 在运行时神奇地拥有该属性和值。

当你在 `Config\Database` 中有属性 `$default = ['encrypt' => false]` 时, 即使你在 **.env** 中放置 `database.default.encrypt.ssl_verify = true`, 也不能将 `encrypt` 值更改为数组。如果你想这样做，请参阅*Database Configuration*。

将环境变量视为数组

可以进一步将命名空间环境变量视为数组。如果前缀与配置类匹配, 则环境变量名称的其余部分在也包含点时将被视为数组引用。

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

// 数组命名空间变量
```

(续下页)

(接上页)

```
Config\SimpleConfig.address.city = "Berlin"
Config\SimpleConfig.address.country = "Germany"
```

如果这是指向 SimpleConfig 配置对象, 那么上面的示例将被视为:

```
<?php

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

`$address` 属性的任何其他元素保持不变。

你也可以使用数组属性名称作为前缀。如果环境文件包含以下内容, 结果与上面相同。

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

处理不同环境

通过使用带有修改后的值来满足该环境需求的单独 **.env** 文件, 可以轻松配置多个环境。

该文件不应包含应用程序使用的每个可能的配置类的每一个可能设置。事实上, 它应该只包含特定于该环境的项目, 以及密码、API 密钥等不应公开暴露的敏感详细信息。但是任何在部署之间更改的都很合适。

在每个环境中, 将 **.env** 文件放在项目的根目录中。对于大多数设置来说, 这将与 `app` 目录处于同一级别。

不要使用版本控制系统跟踪 **.env** 文件。如果这样做, 并且存储库被公开, 你将在所有人都可以找到的地方放置敏感信息。

注册器

"注册器" 是可以在命名空间和文件之间在运行时提供其他配置属性的任何其他类。注册器提供了一种在运行时跨命名空间和文件更改配置的方法。

如果在模块 中启用了自动发现, 则注册器可以在命名空间和文件之间在运行时更改配置属性。

备注: 此功能在 `CodeIgniter\Config\BaseConfig` 类中实现。因此，它不适用于 **app/Config** 文件夹中未继承该类的某些文件。

有两种实现注册器的方法: **隐式**和 **显式**。

备注: 来自 **.env** 的值始终优先于注册器。

隐式注册器

隐式注册器可以更改任何配置类的属性。

任何命名空间都可以通过使用 **Config/Registrar.php** 文件定义隐式注册器。这些文件是类，其方法的名称与你希望扩展的每个配置类的名称相同。

例如，第三方模块或 Composer 包可能希望为 `Config\Pager` 提供额外的模板，而不会覆盖开发人员已经配置的内容。在 **src/Config/Registrar.php** 中，将有一个名为 `Registrar` 的类，其中只有一个 `Pager()` 方法（注意大小写敏感）:

```
<?php

namespace CodeIgniter\Shield\Config;

class Registrar
{
    public static function Pager(): array
    {
        return [
            'templates' => [
                'module_pager' => 'MyModule\Views\Pager',
            ],
        ];
    }
}
```

注册方法必须始终返回一个数组, 其中的键对应目标配置文件中的属性。存在的值被合并, 注册器属性具有覆盖优先级。

显式注册器

显式注册器只能更改其注册的配置类属性。

配置文件还可以显式指定任意数量的注册器。这是通过在配置文件中添加一个 $registrars 属性来完成的, 其中包含候选注册器的名称数组:

```
<?php

namespace Config;

// ...

class MyConfig extends BaseConfig
{
    public static $registrars = [
        SupportingPackageRegistrar::class,
    ];

    // ...
}
```

为了充当“注册器”, 标识的类必须具有一个与配置类同名的静态函数, 它应返回一个关联数组的属性设置。

在实例化配置对象时, 它将循环遍历 $registrars 中指定的类。对于这些类中的每个类, 它都会调用以配置类命名的方法, 并合并任何返回的属性。

针对此设置的配置类示例:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class MySalesConfig extends BaseConfig
{
    public int $target        = 100;
    public string $campaign   = 'Winter Wonderland';
```

(续下页)

(接上页)

```
    public static $registrars = [
        '\App\Models\RegionalSales',
    ];
}
```

…相关的区域销售模型可能如下所示:

```
<?php

namespace App\Models;

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

通过上面的示例, 在实例化 MySalesConfig 时, 它将最终具有声明的三个属性, 但 $target 属性的值将通过将 RegionalSales 视为 "注册器" 来覆盖。生成的配置属性:

```
<?php

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

确认配置值

实际的 Config 对象属性值在运行时由注册器、环境变量 和配置缓存 进行更改。

CodeIgniter 有以下命令 来检查实际的配置值。

config:check

在 4.5.0 版本加入.

例如，如果你想检查 `Config\App` 实例：

```
php spark config:check App
```

输出结果如下：

```
Config\App#6 (12) (
    public 'baseURL' -> string (22) "http://localhost:8080/"
    public 'allowedHostnames' -> array (0) []
    public 'indexPage' -> string (9) "index.php"
    public 'uriProtocol' -> string (11) "REQUEST_URI"
    public 'defaultLocale' -> string (2) "en"
    public 'negotiateLocale' -> boolean false
    public 'supportedLocales' -> array (1) [
        0 => string (2) "en"
    ]
    public 'appTimezone' -> string (3) "UTC"
    public 'charset' -> string (5) "UTF-8"
    public 'forceGlobalSecureRequests' -> boolean false
    public 'proxyIPs' -> array (0) []
    public 'CSPEnabled' -> boolean false
)

Config Caching: Disabled
```

你可以看到配置缓存是否已启用。

备注： 如果启用了配置缓存，则始终使用缓存的值。有关详情，请参阅配置缓存。

4.2.2 CodeIgniter URL

- *URL 结构*
  - *基本 URL 仅包含主机名*
  - *基本 URL 包含子文件夹*
- *URI 安全性*
  - *添加允许的字符*
- *删除 index.php 文件*
  - *Apache Web 服务器*
  - *Nginx*

默认情况下,CodeIgniter 的 URL 旨在对搜索引擎和人类友好。它使用基于 **段**的方法, 而不是与动态系统同义的标准“查询字符串”方法:

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

你可以使用*URI 路由* 功能灵活地定义 URL。

URI 库 和*URL 辅助函数* 包含可以轻松使用 URI 数据的函数。

URL 结构

基本 URL 仅包含主机名

当你有基本 URL **https://www.example.com/** 并想象以下 URL 时:

```
https://www.example.com/blog/news/2022/10?page=2
```

我们使用以下术语:

术语	例子	描述
基本 URL	**https://www.example.com/**	基本 URL 通常表示为 **baseURL**。
URI 路径	/blog/news/2022/10	
路由路径	/blog/news/2022/10	相对于基本 URL 的 URI 路径。它也称为 **URI 字符串**。
查询	page=2	

基本 URL 包含子文件夹

当你有基本 URL **https://www.example.com/ci-blog/** 并想象以下 URL 时:

```
https://www.example.com/ci-blog/blog/news/2022/10?page=2
```

我们使用以下术语:

术语	例子	描述
基本 URL	**https://www.example.com/ci-blog/**	基本 URL 通常表示为 **baseURL**。
URI 路径	/ci-blog/blog/news/2022/10	
路由路径	/blog/news/2022/10	相对于基本 URL 的 URI 路径。它也称为 **URI 字符串**。
查询	page=2	

URI 安全性

在 4.4.7 版本加入.

重要: 从 v4.4.7 版本之前升级的用户需要在 **app/Config/App.php** 中添加以下内容才能使用此功能:

```
public string $permittedURIChars = 'a-z 0-9~%.:_\-';
```

为了帮助尽量减少可能将恶意数据传递到你的应用程序的可能性，CodeIgniter 对 URI 字符串（路由路径）允许的字符相当严格。URI 只能包含以下内容：

- 字母数字文本（仅限拉丁字符）
- 波浪号：~
- 百分号：%
- 句点：.
- 冒号：:
- 下划线：_
- 减号：-
- 空格："

备注： 该检查由 `Router` 执行。Router 获取由 `SiteURI` 类保存的 URL 编码值，对其进行解码，然后检查它是否包含不允许的字符串。

添加允许的字符

可以通过 `Config\App::$permittedURIChars` 更改允许的字符。

如果你想在 URI 路径中使用 Unicode，请对其进行修改以允许使用这些字符。例如，如果你想使用孟加拉语字符，你需要在 **app/Config/App.php** 中设置以下值:

```
public string $permittedURIChars = 'a-z 0-9~%.:_\-\x{0980}-\x{09ff}
↪';
```

可以在维基百科的 Unicode block 中找到完整的 Unicode 范围列表。

删除 index.php 文件

当你使用 Apache Web 服务器时，默认情况下，在你的 URL 中需要 **index.php** 文件:

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

如果你的服务器支持重写 URL, 你可以使用 URL 重写轻松删除此文件。这由不同的服务器以不同的方式处理, 但我们将在这里展示两个最常见的 Web 服务器的示例。

Apache Web 服务器

Apache 必须启用 *mod_rewrite* 扩展。如果是, 你可以使用一些简单规则的 `.htaccess` 文件。这里是一个这样文件的示例, 使用 "否定" 方法, 其中重定向除指定项之外的所有内容:

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

在这个例子中, 除了现有目录和现有文件的任何 HTTP 请求都被视为对你的 index.php 文件的请求。

备注: 这些特定规则可能不适用于所有服务器配置。

备注: 也要从上述规则中排除你可能需要从外界访问的任何资源。

Nginx

在 Nginx 中, 你可以定义一个位置块并使用 `try_files` 指令来获取与我们在上面的 Apache 配置中相同的效果:

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

这将首先查找与 URI 匹配的文件或目录 (从 root 和 alias 指令的设置构造每个文件的完整路径), 然后将请求以及任何参数发送到 index.php 文件。

4.2.3 辅助函数

- 什么是辅助函数?
- 加载辅助函数
  - 加载单个辅助函数
  - 加载多个辅助函数
  - 在控制器中加载
  - 从指定命名空间加载
  - 自动加载辅助函数
- 使用辅助函数
- 创建辅助函数
  - 创建自定义辅助函数
  - "扩展"辅助函数
- 接下来呢?

什么是辅助函数?

顾名思义，辅助函数可以帮助你完成任务。每个 helper 文件只是某个特定类别的函数集合。有*URL 辅助函数*，可以帮助创建链接，有*表单辅助函数* 可以帮助创建表单元素，*文本辅助函数* 执行各种文本格式化，*Cookie 辅助函数* 设置和读取 Cookie，*文件系统辅助函数* 帮助处理文件等等。

与 CodeIgniter 中的大多数其他系统不同, 辅助函数不是面向对象的格式。它们是简单的程序性函数。每个辅助函数执行一个特定的任务, 不依赖于其他函数。

CodeIgniter 默认不加载辅助文件, 所以使用辅助函数的第一步是加载它。一旦加载, 它就可以在你的控制器 和视图 中全局使用。

辅助函数通常存储在 **system/Helpers** 或 **app/Helpers** 目录中。

加载辅助函数

备注: *URL 辅助函数* 总是加载的, 所以你不需要自己加载它。

加载单个辅助函数

使用以下方法加载辅助函数文件非常简单:

```
<?php

helper('name');
```

上述代码会加载 **name_helper.php** 文件。

重要: CodeIgniter 辅助函数文件名全部小写。因此, 在区分大小写的文件系统 (如 Linux) 上, `helper('Name')` 将无法工作。

例如, 要加载名为 **cookie_helper.php** 的*Cookie 辅助函数* 文件, 你会这样做:

```
<?php

helper('cookie');
```

备注: *`helper()`* 函数不返回值, 所以不要试图将其分配给变量。只按上面示例的方式使用它。

自动发现和 Composer 包

默认情况下，CodeIgniter 会通过*自动发现* 在所有定义的命名空间中搜索辅助函数文件。你可以使用 spark 命令来检查你定义的命名空间。请参阅*确认命名空间*。

如果你使用了许多 Composer 包，那么你将有许多已定义的命名空间。CodeIgniter 默认会扫描所有命名空间。

为了避免浪费时间扫描不相关的 Composer 包，你可以手动指定要进行自动发现的包。请参阅指定 *Composer* 包 了解详细信息。

或者，你可以为要加载的辅助函数指定一个命名空间。

加载顺序

`helper()` 函数会扫描通过所有定义的命名空间，并加载所有名称匹配的辅助函数。这样可以加载任何模块的辅助函数，以及你为此应用专门创建的任何辅助函数。

加载顺序如下：

1. app/Helpers - 这里的文件总是首先加载。
2. {namespace}/Helpers - 所有的命名空间都会按照它们定义的顺序依次循环。
3. system/Helpers - 基础文件最后加载。

加载多个辅助函数

如果你需要一次加载多个辅助函数, 可以传递一个文件名数组, 它们都会被加载:

```
<?php

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

在控制器中加载

可以在控制器方法中的任何位置加载辅助函数 (甚至在视图文件中, 尽管这不是一个好的实践), 只要在使用它之前加载它即可。

你可以在控制器构造函数中加载辅助函数, 以使它们自动在任何方法中可用, 或者你可以在需要它的特定方法中加载辅助函数。

但是, 如果你想在控制器构造函数中加载, 则可以改用 Controller 中的 `$helpers` 属性。参见控制器。

从指定命名空间加载

默认情况下，CodeIgniter 会在所有定义的命名空间中搜索辅助函数文件，并加载所有找到的文件。

如果你只想加载特定命名空间中的一个辅助函数，在辅助函数的名称前加上它所在的命名空间作为前缀。在该命名空间目录中，加载器预期它位于一个名为 **Helpers** 的子目录内。以示例来帮助理解这一点。

对于此示例, 假设我们已经将所有与博客相关的代码分组到自己的命名空间 `Example\Blog` 中。文件存在于我们的服务器上的 **Modules/Blog/** 中。因此, 我们会将博客模块的辅助函数文件放在 **Modules/Blog/Helpers/** 中。**blog_helper** 文件将位于 **Modules/Blog/Helpers/blog_helper.php**。在我们的控制器中, 我们可以使用以下命令加载辅助函数:

```
<?php

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

你也可以使用以下方式:

```
<?php

helper('Example\Blog\Helpers\blog');
```

备注: 以这种方式加载的文件中的函数并不是真正的命名空间化的。命名空间只是用作方便定位文件的一种方式。

自动加载辅助函数

在 4.3.0 版本加入.

如果你发现整个应用程序都需要一个特定的辅助函数, 你可以告诉 CodeIgniter 在系统初始化期间自动加载它。这是通过打开 **app/Config/Autoload.php** 文件, 并将辅助函数添加到 `$helpers` 属性来完成的。

使用辅助函数

一旦你加载了包含要使用的函数的辅助文件, 你就可以像调用标准 PHP 函数一样调用它。

例如, 要在视图文件中使用 `anchor()` 函数创建一个链接, 你会这样做:

```
<div>
<?= anchor('blog/comments', 'Click Here') ?>
</div>
```

其中 "Click Here" 是链接的名称, "blog/comments" 是你想要链接到的控制器/方法的 URI。

创建辅助函数

创建自定义辅助函数

辅助函数文件名是 **辅助函数名**和 **_helper.php**。

例如，要创建 info 辅助函数，你需要创建一个名为 **app/Helpers/info_helper.php** 的文件，并向文件中添加一个函数：

```
<?php

// app/Helpers/info_helper.php
use CodeIgniter\CodeIgniter;

/**
 * Returns CodeIgniter's version.
 */
function ci_version(): string
{
    return CodeIgniter::CI_VERSION;
}
```

你可以在一个辅助函数文件中添加尽可能多的函数。

“扩展”辅助函数

要“扩展”辅助函数, 请在 **app/Helpers** 文件夹中创建一个与现有辅助函数相同名称的文件。

如果你只需要为现有辅助函数添加一些功能 - 可能添加一个或两个函数, 或者更改某个特定辅助函数的工作方式 - 那么用你的版本完全替换整个辅助函数有点过度设计。在这种情况下, 最好只是“扩展”辅助函数。

备注: 这里“扩展”一词使用很宽松, 因为辅助函数是程序性的和离散的, 在传统意义上无法扩展。在底层, 这为你提供了添加或替换辅助函数提供的函数的能力。

例如, 要扩展原生的 **Array 辅助函数**, 你需要创建一个名为 **app/Helpers/array_helper.php** 的文件, 并添加或覆盖函数:

```
<?php

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

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

    return false;
}

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

(续下页)

(接上页)

```
    return array_pop($array);
}
```

重要: 不要指定命名空间 `App\Helpers`。

参见加载顺序 了解辅助函数文件的加载顺序。

接下来呢?

在目录中, 你会找到所有可用辅助函数 的列表。浏览每个函数以查看它们的作用。

4.2.4 全局函数和常量

CodeIgniter 提供了一些全局定义的函数和变量, 在任何时候都可以使用。这些不需要加载任何额外的库或辅助函数。

- 全局函数
  - 服务访问器
  - 杂项函数
- 全局常量
  - 核心常量
  - 时间常量

全局函数

服务访问器

cache([*$key*])

参数

- **$key** (string) –要从缓存中检索的缓存项名称 (可选)

返回

缓存对象实例, 或从缓存中检索的项目

返回类型

`mixed`

如果没有提供 $key, 将返回缓存引擎实例。如果提供了 $key, 将返回当前缓存中 $key 的值, 如果找不到值则返回 null。

例子:

```
<?php

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

config(*string $name*[, *bool $getShared = true*])

参数

- **$name** (`string`) –配置类名。
- **$getShared** (`bool`) –是否返回共享实例。

返回

配置实例。

返回类型

`object|null`

从工厂获取配置实例的更简单方式。

有关详细信息，请参阅*Configuration* 和*Factories*。

`config()` 在内部使用 `Factories::config()`。有关第一个参数 `$name` 的详细信息，请参阅加载类。

cookie(*string $name*[, *string $value = ''*[, *array $options = []*]])

参数

- **$name** (`string`) –Cookie 名称
- **$value** (`string`) –Cookie 值
- **$options** (`array`) –Cookie 选项

返回类型

`Cookie`

返回

`Cookie` 实例

Throws

`CookieException`

创建新的 Cookie 实例的更简单方法。

cookies(*[array $cookies = [][, bool $getGlobal = true]]*)

参数

- **$cookies** (`array`) –如果 `getGlobal` 为 `false`, 则传入 `CookieStore` 构造函数
- **$getGlobal** (`bool`) –如果为 `false`, 创建 `CookieStore` 的新实例

返回类型

`CookieStore`

返回

保存在当前 `Response` 中的 `CookieStore` 实例, 或新的 `CookieStore` 实例

获取 `Response` 中保存的全局 `CookieStore` 实例。

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

参数

- **$key** (`string`) –要检索的环境变量名称
- **$default** (`mixed`) –如果找不到值, 返回的默认值

返回

环境变量、默认值或 null

返回类型

`mixed`

用于检索之前设置到环境中的值, 如果找不到则返回默认值。会将布尔值格式化为实际的布尔值, 而不是字符串表示。

结合 **.env** 文件使用时特别有用, 可设置特定于环境本身的值, 如数据库设置、API 密钥等。

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

参数

- **$data** (`string|array`) –要转义的信息
- **$context** (`string`) –转义上下文。默认为 'html'
- **$encoding** (`string`) –字符串的字符编码

返回

转义后的数据

返回类型

`mixed`

为了帮助防止 XSS 攻击, 对要包含在网页中的数据进行转义。这使用 Laminas Escaper 库来实际过滤数据。

如果 $data 是字符串, 则简单转义并返回它。如果 $data 是数组, 则遍历它, 转义每个键/值对的 'value'。

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

helper(*$filename*)

参数

- **$filename** (`string|array`) –要加载的辅助器文件名, 或文件名数组

加载辅助器文件。

有关完整详细信息, 请参阅辅助器 页面。

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

参数

- **$line** (`string`) –需要检索的语言文件名和文本 key。
- **$args** (`array`) –用于替换占位符的数据数组。
- **$locale** (`string`) –指定使用当前区域设置之外的区域设置。

返回

语言文件中的文本

返回类型

`list<string>|string`

从语言文件中检索文本。

更多信息，请参见语言本地化。

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

参数

- **$name** (`string`) –模型类名
- **$getShared** (`boolean`) –是否返回共享实例
- **$conn** (`ConnectionInterface|null`) –数据库连接

返回

模型实例

返回类型

object

获取模型实例的更简单方法。

`model()` 在内部使用 `Factories::models()`。有关第一个参数 `$name` 的详细信息, 请参阅加载类。

另请参阅使用 *CodeIgniter* 的模型。

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

参数

- **$key** (`string`) –要检查的旧表单数据的名称
- **$default** (`string|null`) –如果 $key 不存在, 返回的默认值
- **$escape** (`false|string`) –转义 上下文或设置 false 禁用它

返回

定义键的值或默认值

返回类型

`array|string|null`

提供了一种简单的方式来访问提交表单后的 "旧输入数据"。

例子:

```
<?php

// in controller, checking form submittal
if (! $model->save($user)) {
    // 'withInput()' is what specifies "old data" should be␣
↪saved.
    return redirect()->back()->withInput();
}

?>

<!-- In your view file: -->
<input type="email" name="email" value="<?= old('email') ?>">

<!-- Or with arrays: -->
<input type="email" name="user[email]" value="<?= old('user.
↪email') ?>">
```

备注: 如果你在表单辅助函数 中使用了*set_value()*、*set_select()*、*set_checkbox()* 和*set_radio()* 函数，这个功能已经内置了。只有在不使用表单辅助函数时才需要使用此函数。

session([*$key*])

参数

- **$key** (string) –要检查的会话项目名称

返回

如果没有 $key, 则是 Session 对象的实例; 如果有 $key, 则是会话中为 $key 找到的值, 如果找不到则为 null

返回类型

mixed

提供了方便访问 session 类和检索存储值的方法。有关更多信息, 请参阅会话 页面。

timer([*$name*])

参数

- **$name** (string) –基准点的名称

返回

Timer 实例

返回类型

CodeIgniterDebugTimer

方便地快速访问 Timer 类的方法。你可以将基准点的名称作为唯一参数传递。这将从此点开始计时, 或如果已运行具有此名称的计时器, 则停止计时。

例子:

```
<?php

// Get an instance
$timer = timer();

// Set timer start and stop points
timer('controller_loading');     // Will start the timer
// ...
timer('controller_loading');     // Will stop the running timer
```

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

参数

- **$name** (string) –要加载的文件的名称
- **$data** (array) –要在视图中可用的键/值对数组
- **$options** (array) –将传递给渲染类的选项数组

返回

来自视图的输出

返回类型

string

获取当前的 RendererInterface 兼容类 (默认为视图 类), 并告诉它渲染指定的视图。这仅仅提供了一个方便的方法, 可以在控制器、库以及在路由闭包中使用。

当前, 这些选项可用于 $options 数组中:

- saveData 指定数据在同一请求内对 view() 的多次调用之间持久化。如果不想持久化数据, 请指定 false。

- cache 指定缓存视图的秒数。有关详细信息, 请参阅缓存视图。
- debug 可以设置为 false 以禁用为*Debug 工具栏* 添加调试代码。

$option 数组主要是为了方便与 Twig 等库的第三方集成。

例子:

```
<?php

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

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

有关更多详细信息, 请参阅视图 和视图渲染器 页面。

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

参数

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

返回

视图单元用于在视图中插入由其他类管理的 HTML 块。

返回类型

string

更多详情请参考视图单元 页面。

杂项函数

app_timezone()

返回

应用程序设置要显示日期的时区

返回类型

string

返回应用程序设置要显示日期的时区。

csp_script_nonce()

返回

脚本标签的 CSP 随机数属性

返回类型

`string`

返回脚本标签的随机数属性。例如:`nonce="Eskdikejidojdk978Ad8jf"`。请参阅Ⓕ容安全策略。

csp_style_nonce()

返回

样式标签的 CSP 随机数属性

返回类型

`string`

返回样式标签的随机数属性。例如:`nonce="Eskdikejidojdk978Ad8jf"`。请参阅Ⓕ容安全策略。

csrf_token()

返回

当前 CSRF 令牌的名称

返回类型

`string`

返回当前 CSRF 令牌的名称。

csrf_header()

返回

当前 CSRF 令牌的标头名称

返回类型

`string`

当前 CSRF 令牌的标头名称。

csrf_hash()

返回

当前 CSRF 哈希值

返回类型

`string`

返回当前 CSRF 哈希值。

csrf_field()

返回

包含所有必需 CSRF 信息的隐藏输入的 HTML 字符串

返回类型

`string`

返回包含所有必需 CSRF 信息的隐藏输入:

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

csrf_meta()

返回

包含所有必需 CSRF 信息的 meta 标签的 HTML 字符串

返回类型

`string`

返回包含所有必需 CSRF 信息的 meta 标签:

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

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

参数

- **$duration** (`int`) –浏览器应将此资源的链接转换为 HTTPS 的秒数
- **$request** (`RequestInterface`) –当前 Request 对象的实例
- **$response** (`ResponseInterface`) –当前 Response 对象的实例

检查当前页面是否通过 HTTPS 访问。如果是，则不执行任何操作。如果不是，则将用户重定向回当前 URI，但通过 HTTPS 进行访问。将设置 HTTP 严格传输安全（HTST）头，指示现代浏览器将任何 HTTP 请求自动修改为 HTTPS 请求，持续时间为 `$duration`。

备注: 当你将 `Config\App:$forceGlobalSecureRequests` 设置为 true 时，也会使用此函数。

---

function_usable(*$function_name*)

参数

- **$function_name** (`string`) –要检查的函数

返回

如果函数存在且可安全调用则为 true, 否则为 false

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`) –被检查的文件名

返回

如果可以写入文件则为 true, 否则为 false

返回类型

`bool`

is_windows([*$mock = null*])

参数

- **$mock** (`bool|null`) –如果给出且为布尔值, 则将其用作返回值

返回类型

`bool`

检测平台是否在 Windows 下运行。

备注: 提供给 $mock 的布尔值将在后续调用中持久化。要重置此模拟值, 用户必须为函数调用显式传递 `null`。这将刷新函数以使用自动检测。

```
<?php

is_windows(true);

// some code ...

if (is_windows()) {
    // do something ..
}

is_windows(null); // reset
```

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

参数

- **$level** (string) –严重级别
- **$message** (string) –要记录的消息
- **$context** (array) –应在 $message 中替换的标签及其值的关联数组

返回

void

返回类型

bool

备注: 自 v4.5.0 起，返回值被固定为兼容 PSR Log。在以前的版本中，如果日志记录成功则返回 `true`，如果有问题则返回 `false`。

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

日志级别可以是以下值之一:`emergency`、`alert`、`critical`、`error`、`warning`、

notice、info 或 debug。

上下文可以用来在消息字符串中替换值。有关完整详细信息, 请参阅日志记录信息页面。

redirect (*string $route*)

参数

- **$route** (string) – 要重定向用户的路由名称或 Controller::method

返回类型

RedirectResponse

返回 RedirectResponse 实例, 可轻松创建重定向。详情请参阅重定向。

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

参数

- **$str** (string) – 输入字符串
- **$urlEncoded** (bool) – 是否也删除 URL 编码字符

返回

经过清理的字符串

返回类型

string

此函数可防止在 ASCII 字符 (如 Java\0script) 之间插入空字符。

例子:

```
<?php

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

request ()

在 4.3.0 版本加入.

返回

共享的 Request 对象

返回类型

`IncomingRequest|CLIRequest`

此函数是 `Services::request()` 和 `service('request')` 的包装器。

response()

在 4.3.0 版本加入.

返回

共享的 Response 对象

返回类型

`Response`

此函数是 `Services::response()` 和 `service('response')` 的包装器。

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

参数

- **$method** (`string`) –路由名称或 Controller::method
- **...$params** (`int|string`) –要传递给路由的一个或多个参数。最后一个参数允许你设置区域设置。

返回

路由路径 (基于 baseURL 的 URI 相对路径)

返回类型

`string`

备注: 此函数要求控制器/方法必须在 **app/Config/Routes.php** 中定义路由。

重要: `route_to()` 返回一个 路由路径, 而不是站点的完整 URI 路径。如果你的 **baseURL** 包含子文件夹, 返回值与链接的 URI 并不相同。在这种情况下, 请改用*`url_to()`*。另请参阅*URL 结构*。

根据 controller::method 组合为你生成路由。将根据提供的参数生成路由。

```
<?php
```

(续下页)

(接上页)

```
// The route is defined as:
$routes->get('users/(:num)/gallery/(:num)',
↪'Galleries::showUserGallery/$1/$2');

?>

<?php

// Generate the route with user ID 15, gallery 12:
route_to('Galleries::showUserGallery', 15, 12);
// Result: '/users/15/gallery/12'
```

根据路由名称为你生成路由。

```
<?php

// The route is defined as:
$routes->get('users/(:num)/gallery/(:num)',
↪'Galleries::showUserGallery/$1/$2', ['as' => 'user_gallery']);

?>

<?php

// Generate the route with user ID 15, gallery 12:
route_to('user_gallery', 15, 12);
// Result: '/users/15/gallery/12'
```

从 v4.3.0 开始, 当你在路由中使用 `{locale}` 时, 可以可选地将区域设置值作为最后一个参数指定。

```
<?php

// The route is defined as:
$routes->add(
    '{locale}/users/(:num)/gallery/(:num)',
    'Galleries::showUserGallery/$1/$2',
    ['as' => 'user_gallery'],
```

(续下页)

(接上页)

```
);

?>

<?php

// Generate the route with user ID 15, gallery 12 and locale en:
route_to('user_gallery', 15, 12, 'en');
// Result: '/en/users/15/gallery/12'
```

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

参数

- **$name** (string) –要加载的服务名称
- **$params** (mixed) –要传递给服务方法的一个或多个参数

返回

指定的服务类的实例

返回类型

mixed

提供对系统中定义的任何服务 的简单访问。这将始终返回该类的共享实例, 因此无论在单次请求期间调用多少次, 都只会创建一个类实例。

例子:

```
<?php

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

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

参数

- **$name** (string) –要加载的服务名称
- **$params** (mixed) –要传递给服务方法的一个或多个参数

返回

指定的服务类的实例

返回类型

`mixed`

与上面描述的 **service()** 函数相同, 但此函数的所有调用都将返回一个新的类实例, 而 **service** 每次都返回相同的实例。

slash_item(*$item*)

参数

- **$item** (`string`) –配置项目名称

返回

配置项目或如果项目不存在则为 null

返回类型

`string|null`

获取附加斜杠的配置文件项目 (如果不为空)

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

参数

- **$attributes** (`mixed`) –字符串、键值对数组或对象
- **$js** (`boolean`) –如果值不需要引号 (Javascript 风格) 则为 true

返回

逗号分隔的包含属性键/值对的字符串

返回类型

`string`

将字符串、数组或属性对象转换为字符串的辅助函数。

全局常量

以下常量在应用程序中的任何位置始终可用。

核心常量

constant APPPATH

app 目录的路径。

constant ROOTPATH

项目根目录的路径。刚好在 `APPPATH` 之上。

constant SYSTEMPATH

system 目录的路径。

constant FCPATH

保存前端控制器的目录的路径。

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 记录信息

- 日志级别
- 配置
  - 使用多个日志处理程序
- 使用上下文修改消息
- 使用第三方日志器

日志级别

你可以通过使用`log_message()`方法将信息记录到本地日志文件中。你必须在第一个参数中提供错误的“级别”，指示消息的类型 (调试、错误等)。第二个参数是消息本身:

```
<?php

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

有八种不同的日志级别, 与 RFC 5424 级别匹配, 如下所示:

级别	描述
debug	详细的调试信息。
info	应用程序中的有趣事件, 如用户登录、记录 SQL 查询等。
notice	应用程序中的正常但重要事件。
warning	非错误的异常事件, 如使用废弃的 API、错误使用 API 或其他不一定错误的不可取之处。
error	不需要立即处理但通常应记录和监控的运行时错误。
critical	关键状态, 如应用程序组件不可用, 或意外异常。
alert	必须立即采取行动, 如整个网站关闭、数据库不可用等。
emergency	系统无法使用。

日志系统不提供通知系统管理员或网站管理员这些事件的方法, 它仅记录信息。对于许多更关键的事件级别, 日志由上述错误处理程序自动完成。

配置

你可以修改实际记录的级别，以及为不同级别分配不同的记录器，都在 **app/Config/Logger.php** 配置文件中完成。

配置文件中的 `threshold` 值确定跨应用程序记录的级别。如果应用程序请求记录任何级别，但阈值当前不允许它们记录，则它们将被忽略。使用的最简单方法是将此值设置为你希望记录的最低级别。例如，如果你想记录警告消息而不是信息消息，你应该将阈值设置为 `5`。级别为 5 或更低 (包括运行时错误、系统错误等) 的任何日志请求都会被记录，而 info、notice 和 debug 会被忽略:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Logger extends BaseConfig
{
    public $threshold = 5;

    // ...
}
```

配置文件中包含完整的级别列表及其对应的阈值以供参考。

你可以通过将日志级别编号的数组分配给阈值来选择要记录的特定级别:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Logger extends BaseConfig
{
    // Log only debug and info type messages
    public $threshold = [5, 8];

    // ...
```

(续下页)

(接上页)

```
}
```

使用多个日志处理程序

日志系统可以同时支持运行多个日志处理方法。每个处理程序可以设置为处理特定级别并忽略其余级别。默认安装中提供了三个处理程序:

- **File Handler** 是默认的处理程序, 将每天在本地创建一个文件。这是推荐的日志方法。
- **ChromeLogger Handler** 如果你在 Chrome 网页浏览器中安装了 ChromeLogger 扩展, 则可以使用此处理程序在 Chrome 的控制台窗口中显示日志信息。
- **Errorlog Handler** 此处理程序将利用 PHP 的原生 `error_log()` 函数并将日志写入其中。目前仅支持 `error_log()` 的 0 和 4 消息类型。

主配置文件中的 `$handlers` 属性配置了处理程序, 它简单地是一个处理程序数组及其配置。每个处理程序通过键指定, 即完全限定的类名。值将是特定于每个处理程序的各种属性的数组。每个处理程序部分将有一个共同点:`handles`, 这是一个日志级别 名称数组, 处理程序将为其记录信息。

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Logger extends BaseConfig
{
    public $handlers = [
        // File Handler
        'CodeIgniter\Log\Handlers\FileHandler' => [
            'handles' => ['critical', 'alert', 'emergency', 'debug',
↪ 'error', 'info', 'notice', 'warning'],
        ],
    ];

    // ...
}
```

使用上下文修改消息

你会经常想根据日志的事件上下文修改消息的详细信息。你可能需要记录用户 id、IP 地址、当前的 POST 变量等。你可以通过在消息中使用占位符来实现这一点。每个占位符必须用大括号包装。在第三个参数中, 你必须提供一个占位符名称 (不带括号) 和它们的值的数组。这些将插入到消息字符串中:

```
<?php

// Generates a message like: User 123 logged into the system from␣
↪127.0.0.1
$info = [
    'id'         => $user->id,
    'ip_address' => $this->request->getIPAddress(),
];

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

如果你想记录异常或错误, 可以使用 ‘exception’ 键, 值是异常或错误本身。将从该对象生成包含错误消息、文件名和行号的字符串。你仍必须在消息中提供异常占位符:

```
<?php

try {
    // Something throws error here
} catch (\Exception $e) {
    log_message('error', '[ERROR] {exception}', ['exception' =>
↪$e]);
}
```

根据当前页面请求, 存在几个核心占位符将自动为你展开:

占位符	插入的值
{post_vars}	$_POST 变量
{get_vars}	$_GET 变量
{session_vars}	$_SESSION 变量
{env}	当前环境名称, 即 development
{file}	调用 logger 的文件名
{line}	{file} 中调用 logger 的行
{env:foo}	$_ENV 中的 'foo' 值

使用第三方日志器

只要它扩展自 `Psr\Log\LoggerInterface` 并且与 PSR-3 兼容, 你就可以使用任何其他你喜欢的日志器。这意味着你可以轻松使用任何兼容 PSR-3 的日志器, 或创建你自己的日志器。

你必须确保系统能够找到第三方日志器, 方法是将其添加到 **app/Config/Autoload.php** 配置文件中, 或通过另一个自动加载器 (如 Composer)。接下来, 你应该修改 **app/Config/Services.php** 以将 `logger` 别名指向新的类名。

现在, 通过 *`log_message()`* 函数执行的任何调用都将使用你的库。

4.2.6 错误处理

CodeIgniter 通过异常机制构建了错误报告系统，既包含 SPL 集合 中的异常，也提供了框架专属的异常类型。

根据运行环境的配置，当发生错误或抛出异常时，默认行为是显示详细错误报告（除非应用处于 `production` 环境）。在 `production` 环境中，会显示更通用的信息以保持最佳用户体验。

- 使用异常
  - 什么是异常
  - 捕获异常
- 配置
  - 错误报告

  - 异常日志记录
  - 记录弃用警告
- 框架异常
  - 异常设计
  - *LogicException*
  - *RuntimeException*
  - *PageNotFoundException*
  - *ConfigException*
  - *DatabaseException*
  - *RedirectException*
- 在异常中指定 *HTTP* 状态码
- *HTTP* 状态码与错误视图
- 在异常中指定退出码
- 自定义异常处理器
  - 定义新处理器
  - 配置新处理器

使用异常

本节为新手程序员或不熟悉异常使用的开发者提供快速概览。

什么是异常

异常是当程序“抛出”异常时发生的事件。这会中断当前脚本流程，并将执行权转交给错误处理程序以显示相应的错误页面：

```
<?php

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

捕获异常

当调用可能抛出异常的方法时，可以使用 `try/catch` 代码块来捕获异常：

```
<?php

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

如果 `$userModel` 抛出异常，该异常会被捕获并执行 catch 块中的代码。在此示例中，脚本终止并输出 `UserModel` 定义的错误信息。

捕获特定异常

上例中我们捕获所有类型的 Exception。若只需捕获特定类型的异常（如 `DataException`），可在 catch 参数中指定。其他未被捕获的异常类型将传递给错误处理程序：

```
<?php

use CodeIgniter\Database\Exceptions\DataException;

try {
    $user = $userModel->find($id);
} catch (DataException $e) {
    // do something here...
}
```

这种方式便于自行处理错误或在脚本结束前执行清理操作。若希望错误处理程序按常规方式处理，可在 catch 块中重新抛出异常：

```
<?php

use CodeIgniter\Database\Exceptions\DataException;
```

(续下页)

(接上页)

```
try {
    $user = $userModel->find($id);
} catch (DataException $e) {
    // do something here...

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

配置

错误报告

当 PHP ini 设置中的 `display_errors` 启用时，CodeIgniter 将显示包含所有错误的详细报告

默认情况下，CodeIgniter 在 `development` 和 `testing` 环境中显示详细错误报告，在 `production` 环境中不显示任何错误。

可通过设置 `CI_ENVIRONMENT` 变量来更改环境配置，详见设置环境。

重要: 禁用错误报告不会阻止错误日志的写入。

警告: 注意 **.env** 文件中的设置会被添加到 `$_SERVER` 和 `$_ENV`。副作用是当显示详细错误报告时，**你的敏感凭证可能被公开暴露**。

异常日志记录

默认情况下，除 "404 - 页面未找到" 异常外，所有异常都会被记录。可通过修改 **app/Config/Exceptions.php** 中的 `$log` 值来开关此功能:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Exceptions extends BaseConfig
{
    // ...
    public bool $log = true;
    // ...
}
```

要忽略其他状态码的日志记录，可在同一文件中设置:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Exceptions extends BaseConfig
{
    // ...
    public array $ignoreCodes = [404];
```

(续下页)

(接上页)

```
    // ...
}
```

备注: 如果当前日志设置 未配置记录 critical 级别错误（所有异常均按此级别记录），异常可能仍不会被记录。

记录弃用警告

在 4.3.0 版本加入.

在 v4.3.0 之前，所有通过 error_reporting() 报告的错误都会被抛出为 ErrorException 对象。

随着 PHP 8.1+ 的普及，用户可能会遇到因 向内部函数的非空参数传递 null 值 导致的异常抛出。

为简化迁移到 PHP 8.1 的过程，从 v4.3.0 开始，CodeIgniter 新增了仅记录弃用错误（E_DEPRECATED 和 E_USER_DEPRECATED）而不将其作为异常抛出的功能。

默认情况下，CodeIgniter 在开发环境中仅记录弃用警告而不抛出异常。在生产环境中，既不记录也不抛出异常。

配置

该功能的配置步骤如下：首先确保 **ConfigExceptions** 已更新并包含以下两个新属性：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
use Psr\Log\LogLevel;

class Exceptions extends BaseConfig
{
    // ...
```

(续下页)

(接上页)

```
    public bool $logDeprecations = true; // If set to false, an␣
↪exception will be thrown.
    // ...
    public string $deprecationLogLevel = LogLevel::WARNING; // This␣
↪should be one of the log levels supported by PSR-3.
    // ...
}
```

其次，根据 Config\Exceptions::$deprecationLogLevel 设置的日志级别，检查 Config\Logger::$threshold 定义的日志阈值是否涵盖该级别。如未涵盖需相应调整：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Logger extends BaseConfig
{
    // ...
    // This must contain the log level (5 for LogLevel::WARNING)␣
↪corresponding to $deprecationLogLevel.
    public $threshold = (ENVIRONMENT === 'production') ? 4 : 9;
    // ...
}
```

配置完成后，后续弃用警告将按配置记录而不作为异常抛出。

该功能也支持用户自定义弃用警告：

```
<?php

@trigger_error('Do not use this class!', E_USER_DEPRECATED);
// Your logs should contain a record with a message like:
↪"[DEPRECATED] Do not use this class!"
```

测试应用时若需强制抛出弃用警告，可设置环境变量 CODEIGNITER_SCREAM_DEPRECATIONS 为真值。

框架异常

异常设计

自 v4.6.0 起，框架抛出的所有异常类：

- 实现 `CodeIgniter\Exceptions\ExceptionInterface`
- 继承 `CodeIgniter\Exceptions\LogicException` 或 `CodeIgniter\Exceptions\RuntimeException`

备注： 框架仅抛出上述类型异常，但 PHP 或其他使用的库可能抛出其他异常。

框架抛出的两种基础异常类：

LogicException

`CodeIgniter\Exceptions\LogicException` 继承自 `\LogicException`。该异常表示程序逻辑错误，应直接通过修改代码修复。

RuntimeException

`CodeIgniter\Exceptions\RuntimeException` 继承自 `\RuntimeException`。该异常在运行时发生错误时抛出。

其他可用框架异常：

PageNotFoundException

用于触发 404 页面未找到错误：

```
<?php

use CodeIgniter\Exceptions\PageNotFoundException;

$page = $pageModel->find($id);
```

(续下页)

(接上页)

```
if ($page === null) {
    throw PageNotFoundException::forPageNotFound();
}
```

可传递自定义消息替代默认的 404 页面信息。默认 404 视图文件位置参见*HTTP 状态码与错误视图*。

如果在 **app/Config/Routing.php** 或 **app/Config/Routes.php** 中配置了*404 重写*，将调用该覆盖配置而非标准 404 页面。

ConfigException

当配置类值无效或配置类类型不符时使用此异常：

```
<?php

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

该异常提供退出码 3。

DatabaseException

在数据库连接创建失败或临时丢失等数据库错误时抛出：

```
<?php

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

该异常提供退出码 8。

RedirectException

备注： 自 v4.4.0 起，`RedirectException` 的命名空间已变更。原为 `CodeIgniter\Router\Exceptions\RedirectException`，该旧类已在 v4.6.0 移除。

此特殊异常允许覆盖其他响应路由并强制重定向到指定 URI：

```
<?php

throw new \CodeIgniter\HTTP\Exceptions\RedirectException($uri);
```

$uri 是相对于 baseURL 的 URI 路径。可指定替代默认值（302,” 临时重定向”）的重定向代码：

```
<?php

throw new \CodeIgniter\HTTP\Exceptions\RedirectException($uri, 301);
```

自 v4.4.0 起，第一个参数可使用实现 ResponseInterface 的对象。此方案适用于需要添加额外头信息或 cookies 的场景：

```
<?php

$response = service('response')
    ->redirect('https://example.com/path')
    ->setHeader('Some', 'header')
    ->setCookie('and', 'cookie');

throw new \CodeIgniter\HTTP\Exceptions\RedirectException($response);
```

在异常中指定 HTTP 状态码

在 4.3.0 版本加入.

自 v4.3.0 起，可通过让异常类实现 CodeIgniter\Exceptions\HTTPExceptionInterface 来指定 HTTP 状态码。

当 CodeIgniter 异常处理器捕获到实现 HTTPExceptionInterface 的异常时，异常代码将作为 HTTP 状态码。

HTTP 状态码与错误视图

异常处理器会显示与 HTTP 状态码对应的错误视图（如果存在）。

例如，`PageNotFoundException` 实现了 `HTTPExceptionInterface`，其异常代码 `404` 将作为 HTTP 状态码。当该异常被抛出时：

- 网页请求会显示 **app/Views/errors/html** 目录下的 **error_404.php**
- CLI 请求会显示 **app/Views/errors/cli** 目录下的 **error_404.php**

若无对应 HTTP 状态码的视图文件，将显示 **production.php** 或 **error_exception.php**。

备注： 若 PHP ini 设置中 `display_errors` 开启，将选择 **error_exception.php** 并显示详细错误报告。

建议在 **app/Views/errors/html** 目录下自定义所有错误视图。

可为特定 HTTP 状态码创建错误视图。例如创建“400 Bad Request”错误视图需添加 **error_400.php**。

> **警告：** 若存在对应 HTTP 状态码的错误视图文件，异常处理器将始终显示该文件。必须确保视图文件在生产环境中不会自行显示详细错误信息。

在异常中指定退出码

在 4.3.0 版本加入.

自 v4.3.0 起，可通过让异常类实现 `CodeIgniter\Exceptions\HasExitCodeInterface` 来指定退出码。

当 CodeIgniter 异常处理器捕获到实现 `HasExitCodeInterface` 的异常时，将从 `getExitCode()` 方法获取退出码。

自定义异常处理器

在 4.4.0 版本加入.

若需更精细控制异常显示方式，可定义自定义处理器并指定其应用场景。

定义新处理器

首先创建实现 `CodeIgniter\Debug\ExceptionHandlerInterface` 的新类。也可继承 `CodeIgniter\Debug\BaseExceptionHandler`，该类包含默认异常处理器使用的实用方法。新处理器需实现 `handle()` 方法：

```
<?php

namespace App\Libraries;

use CodeIgniter\Debug\BaseExceptionHandler;
use CodeIgniter\Debug\ExceptionHandlerInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
use Throwable;

class MyExceptionHandler extends BaseExceptionHandler implements␣
↪ExceptionHandlerInterface
{
    // You can override the view path.
    protected ?string $viewPath = APPPATH . 'Views/exception/';

    public function handle(
        Throwable $exception,
        RequestInterface $request,
        ResponseInterface $response,
        int $statusCode,
        int $exitCode,
    ): void {
        $this->render($exception, $statusCode, $this->viewPath .
↪"error_{$statusCode}.php");

        exit($exitCode);
```

(续下页)

(接上页)

```
    }
}
```

此示例展示了典型的最小实现：显示视图并以正确退出码终止。BaseExceptionHandler 还提供其他辅助功能和对象。

配置新处理器

在 **app/Config/Exceptions.php** 配置文件的 handler() 方法中指定使用新异常处理器类：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
use CodeIgniter\Debug\ExceptionHandler;
use CodeIgniter\Debug\ExceptionHandlerInterface;
use Throwable;

class Exceptions extends BaseConfig
{
    // ...

    public function handler(int $statusCode, Throwable $exception):␣
↪ExceptionHandlerInterface
    {
        return new ExceptionHandler($this);
    }
}
```

可使用任意逻辑决定是否处理异常，最常见的是检查 HTTP 状态码或异常类型。若应由自定义类处理，则返回该类实例：

```
<?php

namespace Config;
```

(续下页)

(接上页)

```
use CodeIgniter\Config\BaseConfig;
use CodeIgniter\Debug\ExceptionHandlerInterface;
use CodeIgniter\Exceptions\PageNotFoundException;
use Throwable;

class Exceptions extends BaseConfig
{
    // ...

    public function handler(int $statusCode, Throwable $exception):␣
↪ExceptionHandlerInterface
    {
        if (in_array($statusCode, [400, 404, 500], true)) {
            return new \App\Libraries\MyExceptionHandler($this);
        }

        if ($exception instanceof PageNotFoundException) {
            return new \App\Libraries\MyExceptionHandler($this);
        }

        return new \CodeIgniter\Debug\ExceptionHandler($this);
    }
}
```

4.2.7 网页缓存

CodeIgniter 允许你缓存网页以达到最大性能。

虽然 CodeIgniter 很快, 但你在页面上显示的动态信息量将直接关联到服务器资源、内存和处理周期的利用, 这会影响你的页面加载速度。通过缓存页面, 由于它们以完全渲染的状态保存, 你可以达到更接近静态网页的性能。

- 缓存是如何工作的?
- 配置缓存
  - 设置缓存引擎

- 设置 *$cacheQueryString*
- 启用缓存
- 删除缓存

缓存是如何工作的?

缓存可以基于每个页面进行启用，你可以设置页面在刷新之前应保持缓存的时间长度。

备注: 基于每页意味着基于每个 URI。从 v4.5.0 开始，请求的 HTTP 方法也被考虑在内。这意味着如果 HTTP 方法不同，相同的 URI 将分别被缓存。

当页面第一次加载时，页面将使用当前配置的缓存引擎进行缓存。在后续的页面加载中，缓存将被检索并发送到请求用户的浏览器。

如果缓存已过期，它将被删除并在发送到浏览器之前刷新。

备注: Benchmark 标签不会被缓存, 所以启用缓存后你仍可以查看页面加载速度。

配置缓存

设置缓存引擎

在使用网页缓存之前，你必须通过编辑 **app/Config/Cache.php** 来设置缓存引擎。有关详细信息，请参阅配置缓存。

设置 $cacheQueryString

你可以使用 `Config\Cache::$cacheQueryString` 设置是否在生成缓存时包含查询字符串。

有效选项为:

- `false`: （默认）禁用。不考虑查询字符串；对于具有相同 URI 路径但不同查询字符串的请求，返回相同的缓存。

- `true`: 启用，考虑所有查询参数。请注意，这可能导致为同一页面反复生成大量缓存。
- **array**: 启用，但仅考虑指定的查询参数列表。例如: `['q', 'page']`。

启用缓存

要启用缓存, 请在任何控制器方法中放入以下标签:

```
<?php

$this->cachePage($n);
```

其中 `$n` 是页面在刷新之间保持缓存的 **秒数**。

上面的标签可以放在一个方法的任何位置。它的位置不会受到影响, 所以将其放在你认为最合适的位置。一旦标签就位, 你的页面就会开始被缓存。

重要: 如果你改变了可能影响输出的配置选项, 你必须手动删除缓存。

删除缓存

如果你不再希望缓存一个页面, 可以删除缓存标签, 它就不会在过期时刷新。

备注: 删除标签不会立即删除缓存。它必须正常过期。

4.2.8 AJAX 请求

IncomingRequest::isAJAX() 方法使用 X-Requested-With 头来定义请求是否是 XHR 还是普通的。然而, 最新的 JavaScript 实现 (即 fetch) 在发送请求时不再发送此头, 因此 IncomingRequest::isAJAX() 的使用变得不太可靠, 因为没有此头就无法定义请求是否是 XHR。

为了解决这个问题, 最有效的解决方案 (到目前为止) 是手动定义请求头, 强制向服务器发送信息, 然后服务器将能够识别请求是 XHR。

下面是如何在 Fetch API 和其他 JavaScript 库中强制发送 X-Requested-With 头。

- *Fetch API*
- *jQuery*
- *VueJS*
- *React*
- *htmx*

Fetch API

```
fetch(url, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "X-Requested-With": "XMLHttpRequest"
    }
});
```

jQuery

对于像 jQuery 这样的库, 不必明确发送此头, 因为根据官方文档, 所有 $.ajax() 请求都是标准头。但如果你仍要强制发送以防万一, 只需如下所示:

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

VueJS

在 VueJS 中, 只要你使用 Axios 进行这种请求, 就需要将以下代码添加到 created 函数中。

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

React

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

htmx

你可以使用 ajax-header 扩展。

```
<body hx-ext="ajax-header">
...
</body>
```

4.2.9 代码模块

CodeIgniter 支持一种代码模块化形式, 以帮助你创建可重用的代码。模块通常围绕特定主题展开, 可以认为是你更大的应用程序中的微型应用程序。

框架支持的任何标准文件类型都受支持, 如控制器、模型、视图、配置文件、辅助函数、语言文件等。模块可以包含尽可能少或多的这些文件。

如果你想将一个模块创建为 Composer 包，请参阅创建 *Composer* 包。

- 命名空间
- 自动加载非类文件
- 自动发现
  - 启用/禁用发现
  - 指定要发现的项
  - 发现和 *Composer*
- 使用文件
  - 路由
  - 过滤器
  - 控制器

- 配置文件
- 迁移
- 种子
- 辅助函数
- 语言文件
- 库
- 模型
- 视图

命名空间

模块功能的核心元素来自 CodeIgniter 使用的兼容 *PSR-4* 的自动加载。虽然任何代码都可以使用 PSR-4 自动加载器和命名空间, 但充分利用模块的主要方法是为你的代码添加命名空间并将其添加到 **app/Config/Autoload.php** 中的 `$psr4` 属性。

例如, 假设我们想保留一个简单的博客模块, 以便在应用程序之间重用。我们可以创建一个文件夹, 名为 Acme, 来存储所有的模块。我们将它放在主项目根目录中的 **app** 目录旁边:

```
acme/          // 新模块目录
app/
public/
system/
tests/
writable/
```

打开 **app/Config/Autoload.php** 并将 `Acme\Blog` 命名空间添加到 `$psr4` 数组属性:

```php
<?php

namespace Config;

use CodeIgniter\Config\AutoloadConfig;

class Autoload extends AutoloadConfig
```

(续下页)

(接上页)

```
{
    // ...
    public $psr4 = [
        APP_NAMESPACE => APPPATH,
        'Acme\Blog'   => ROOTPATH . 'acme/Blog',
    ];

    // ...
}
```

现在设置好后, 我们可以通过 `Acme\Blog` 命名空间访问 **acme/Blog** 文件夹中的任何文件。仅此一点就解决了模块工作所需的 80%, 所以你应该确保熟悉命名空间并熟练使用它们。通过所有定义的命名空间会自动扫描几种文件类型 - 使用模块的关键组成部分。

模块中的常见目录结构将模拟主应用程序文件夹:

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

当然, 没有什么能强制你使用这个确切的结构, 你应该以最适合模块的方式组织它, 省略不需要的目录, 为实体、接口或存储库等创建新目录。

自动加载非类文件

通常, 你的模块不仅包含 PHP 类, 还包含像程序函数、引导文件、模块常量文件等通常不会像加载类那样加载的文件。一种方法是在使用文件位置的开头 `require` 这些文件。

CodeIgniter 提供的另一种方法是像自动加载类一样自动加载这些 非类文件。我们需要做的就是提供这些文件路径的列表, 并将它们包含在 **app/Config/Autoload.php** 文件的 `$files` 属性中。

```
<?php

namespace Config;

use CodeIgniter\Config\AutoloadConfig;

class Autoload extends AutoloadConfig
{
    // ...

    public $files = [
        'path/to/my/functions.php',
        'path/to/my/constants.php',
        'path/to/my/bootstrap.php',
    ];

    // ...
}
```

自动发现

通常, 你需要指定要包含的文件的完全命名空间, 但是可以通过自动发现许多不同类型的文件来配置 CodeIgniter, 从而使将模块集成到应用程序中更简单, 包括:

- *Events*
- *Filters*
- *注册器*
- *Route files*
- *Services*

这在文件 **app/Config/Modules.php** 中配置。

自动发现系统通过扫描在 **Config/Autoload.php** 和 Composer 包中定义的 psr4 命名空间下的特定目录和文件来工作。

例如, 发现过程将在路径中查找可以发现的项, 并应该找到 **acme/Blog/Config/Routes.php** 中的 routes 文件。

启用/禁用发现

你可以通过系统中的 `$enabled` 类变量打开或关闭所有自动发现。False 将禁用所有发现, 优化性能, 但会消除模块和 Composer 包的特殊功能。

指定要发现的项

使用 `$aliases` 选项, 你可以指定要自动发现的项。如果不存在该项, 则不会为该项执行自动发现, 但数组中的其他项仍将被发现。

发现和 Composer

使用 PSR-4 命名空间通过 Composer 安装的包也将默认被发现。使用 PSR-0 命名空间的包将不会被检测到。

指定 Composer 包

在 4.3.0 版本加入.

为避免花时间扫描不相关的 Composer 包, 你可以通过编辑 **app/Config/Modules.php** 中的 `$composerPackages` 变量手动指定要发现的包:

```
<?php

namespace Config;

use CodeIgniter\Modules\Modules as BaseModules;

class Modules extends BaseModules
{
```

(续下页)

(接上页)

```
    // ...

    public $composerPackages = [
        'only' => [
            // List up all packages to auto-discover
            'codeigniter4/shield',
        ],
    ];

    // ...
}
```

或者, 你可以指定要从发现中排除的包。

```
<?php

namespace Config;

use CodeIgniter\Modules\Modules as BaseModules;

class Modules extends BaseModules
{
    // ...

    public $composerPackages = [
        'exclude' => [
            // List up packages to exclude.
            'pestphp/pest',
        ],
    ];

    // ...
}
```

禁用 Composer 包发现

如果你不希望在查找文件时扫描 Composer 的所有已知目录, 可以通过编辑 **app/Config/Modules.php** 中的 `$discoverInComposer` 变量将其关闭:

```
<?php

namespace Config;

use CodeIgniter\Modules\Modules as BaseModules;

class Modules extends BaseModules
{
    public $discoverInComposer = false;

    // ...
}
```

使用文件

本节将查看每种文件类型 (控制器、视图、语言文件等) 以及如何在模块中使用它们。用户指南的相关位置已对其中一些信息进行了更详细的描述, 但在此重复以更容易掌握所有部分的关系。

路由

默认情况下, 模块内会自动扫描路由。可以在上面描述的 **Modules** 配置文件中将其关闭。

备注: 由于文件被包含到当前作用域中, 因此 `$routes` 实例已为你定义。如果尝试重新定义该类, 则会导致错误。

使用模块时, 如果应用程序中的路由包含通配符, 这可能是一个问题。在这种情况下, 请参阅路由优先级。

过滤器

自 4.4.2 版本弃用.

备注: 此功能已被弃用。请改用注册器，如下所示:

```
<?php

namespace CodeIgniter\Shield\Config;

use CodeIgniter\Shield\Filters\SessionAuth;
use CodeIgniter\Shield\Filters\TokenAuth;

class Registrar
{
    /**
     * Registers the Shield filters.
     */
    public static function Filters(): array
    {
        return [
            'aliases' => [
                'session' => SessionAuth::class,
                'tokens'  => TokenAuth::class,
            ],
        ];
    }
}
```

默认情况下, 模块内会自动扫描过滤器。可以在上面描述的 **Modules** 配置文件中将其关闭。

备注: 由于文件被包含到当前作用域中, 因此 `$filters` 实例已为你定义。如果尝试重新定义该类, 则会导致错误。

在模块的 **Config/Filters.php** 文件中, 你需要定义使用的过滤器的别名:

```
<?php

$filters->aliases['menus'] = \App\Filters\MenusFilter::class;
```

控制器

app/Controllers 目录之外的控制器无法通过 URI 检测自动路由, 而必须在 Routes 文件本身中指定:

```
<?php

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

为了减少这里所需的输入量, **group** 路由功能很有用:

```
<?php

$routes->group('blog', ['namespace' => 'Acme\Blog\Controllers'],␣
↪static function ($routes) {
    $routes->get('/', 'Blog::index');
});
```

配置文件

使用配置文件时不需要特殊更改。这些仍然是命名空间类, 并使用 `new` 命令加载:

```
<?php

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

无论何时使用始终可用的*`config()`* 函数，并将一个短类名传递给它，配置文件都会被自动发现。

备注: 我们不建议在模块中使用相同的短类名。需要覆盖或添加 **app/Config/** 中已知配置的模块应使用*Implicit Registrars*。

备注: 在 v4.4.0 之前，即使你指定了一个完全限定的类名，如 `config(\Acme\Blog\Config\Blog::class)`，`config()` 仍会在 **app/Config/** 中查找文件，只要存在与短类名相同的类。在 v4.4.0 中修复了这个行为，并返回指定的实例。

迁移

定义命名空间中的迁移文件将被自动发现。跨所有命名空间找到的所有迁移将在每次运行时都执行。

种子

只要提供完全限定的命名空间, 就可以从 CLI 和其他种子文件中调用种子文件。如果在 CLI 上调用, 则需要提供双反斜杠:

For Unix:

```
php spark db:seed Acme\\Blog\\Database\\Seeds\\TestPostSeeder
```

For Windows:

```
php spark db:seed Acme\Blog\Database\Seeds\TestPostSeeder
```

辅助函数

在使用*helper()* 函数时, 定义的命名空间内的辅助函数将被自动发现, 只要它们在 **Helpers** 目录内:

```
<?php

helper('blog');
```

你可以指定命名空间。详情请参阅*从指定命名空间加载*。

语言文件

只要文件遵循与主应用程序目录相同的目录结构，在使用 *lang()* 函数时就会从定义的命名空间自动定位语言文件。

库

库总是通过它们的完全限定类名实例化的, 所以不提供特殊访问:

```
<?php

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

模型

如果你通过完全限定的类名用 `new` 关键字实例化模型, 则不提供特殊访问:

```
<?php

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

每当使用始终可用的 *model()* 函数时, 都会自动发现模型文件。

备注: 我们不建议在模块中使用相同的短类名。

备注: 在 v4.4.0 之前，即使你指定了一个完全限定的类名，如 `model(\Acme\Blog\Model\PostModel::class)`，`model()` 也会在 **app/Models/** 中查找具有相同短名称的类的文件。有关更多信息，请参阅 *传递完全限定类名* 中的注释。

视图

如视图 文档中所述，可以使用类命名空间加载视图：

```
<?php

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

4.2.10 管理应用程序

默认情况下, 假设你只打算在 **app** 目录中使用 CodeIgniter 来管理一个应用程序。但是, 可以拥有多个共享单个 CodeIgniter 安装的应用程序集, 或者重命名或重新定位你的应用程序目录。

- 重命名或重新定位应用程序目录
- 使用一个 *CodeIgniter* 安装运行多个应用程序

重命名或重新定位应用程序目录

如果你想要重命名应用程序目录或者甚至将其移动到服务器上的项目根目录之外的其他位置, 请打开主 **app/Config/Paths.php** 文件, 并在 `$appDirectory` 变量中设置一个完整的服务器路径 (约第 44 行):

```
<?php

namespace Config;

class Paths
{
    // ...

    public $appDirectory = '/path/to/your/app';

    // ...
}
```

你需要修改项目根目录中的另外两个文件, 以便它们可以找到 **Paths** 配置文件:

- **spark** 运行命令行应用程序。

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

- **public/index.php** 是你的 Web 应用程序的前端控制器。

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

使用一个 CodeIgniter 安装运行多个应用程序

如果你想共享一个公共的 CodeIgniter 框架安装来管理几个不同的应用程序, 只需将位于应用程序目录内的所有目录都放入自己的 (子) 目录即可。

例如, 假设你要创建两个名为 **foo** 和 **bar** 的应用程序。你可以像这样组织应用程序项目目录:

```
foo/
    app/
    public/
    tests/
    writable/
    env
    phpunit.xml.dist
    spark
bar/
    app/
    public/
    tests/
    writable/
    env
    phpunit.xml.dist
    spark
vendor/
    autoload.php
    codeigniter4/framework/
composer.json
composer.lock
```

备注: 如果你从 Zip 文件安装 CodeIgniter, 目录结构将是:

```
foo/
bar/
codeigniter4/system/
```

这将有两个应用程序 **foo** 和 **bar**, 都有标准的应用程序目录和 **public** 文件夹, 并共享一个公共的 **codeigniter4/framework**。

每个应用程序内部的 **app/Config/Paths.php** 中的 `$systemDirectory` 变量将被设置为指向共享的公共 **codeigniter4/framework** 文件夹:

```php
<?php

namespace Config;

class Paths
{
    // ...

    public $systemDirectory = __DIR__ . '/../../../vendor/
↪codeigniter4/framework/system';

    // ...
}
```

备注: 如果你从 Zip 文件安装 CodeIgniter, `$systemDirectory` 将是 `__DIR__ . '/../../../codeigniter4/system'`。

并修改每个应用程序内部的 **app/Config/Constants.php** 中的 `COMPOSER_PATH` 常量:

```php
<?php

defined('COMPOSER_PATH') || define('COMPOSER_PATH', ROOTPATH . '../
↪vendor/autoload.php');
```

只有在你更改应用程序目录时, 请参阅重命名或重新定位应用程序目录 并修改 **in-**

dex.php 和 **spark** 中的路径。

4.2.11 处理多个环境

开发人员通常希望根据应用程序是在开发还是生产环境中运行来实现不同的系统行为。例如, 在开发应用程序时, 详细的错误输出是有用的, 但在 "生产环境" 时可能也会带来安全问题。在开发环境中, 你可能需要加载在生产环境中不需要的其他工具等等。

- 定义的环境
- 设置环境
  - *ENVIRONMENT* 常量
    - *.env*
    - *Apache*
    - *Nginx*
- 添加环境
  - 引导文件
- 确认当前环境
- 对默认框架行为的影响
  - 错误报告

定义的环境

默认情况下,CodeIgniter 定义了三个环境。

- `production` 用于生产
- `development` 用于开发
- `testing` 用于 PHPUnit 测试

重要: 环境 `testing` 保留用于 PHPUnit 测试。它在框架的各处内置了一些特殊条件以协助测试。你不能在开发中使用它。

如果你想要另一个环境, 例如用于暂存, 你可以添加自定义环境。请参阅添加环境。

设置环境

ENVIRONMENT 常量

要设置环境,CodeIgniter 提供了 `ENVIRONMENT` 常量。如果设置了 `$_SERVER['CI_ENVIRONMENT']`, 将使用该值, 否则默认为 `production`。

根据你的服务器设置, 可以通过几种方式设置此值。

.env

在*.env 文件* 中设置该变量是最简单的方法。

```
CI_ENVIRONMENT = development
```

备注: 你可以通过 `spark env` 命令更改 **.env** 文件中的 `CI_ENVIRONMENT` 值:

```
php spark env production
```

Apache

可以在 **.htaccess** 文件或 Apache 配置中使用 SetEnv 设置此服务器变量。

```
SetEnv CI_ENVIRONMENT development
```

Nginx

在 Nginx 下, 必须通过 `fastcgi_params` 传递环境变量, 以便它在 `$_SERVER` 变量下显示。这允许它在虚拟主机级别工作, 而不是使用 *env* 为整个服务器设置它, 尽管这在专用服务器上也可以很好地工作。然后可以将服务器配置修改为类似以下内容:

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

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

Nginx 和其他服务器可用的替代方法, 或者你可以完全删除此逻辑, 并根据服务器的 IP 地址设置常量 (例如)。

除了影响一些基本框架行为 (参见下一节), 你还可以在自己的开发中使用此常量来区分正在运行的环境。

添加环境

要添加自定义环境, 你只需要为它们添加引导文件。

引导文件

CodeIgniter 要求与环境名称匹配的 PHP 脚本位于 **APPPATH/Config/Boot** 下。这些文件可以包含你希望针对环境进行的任何自定义, 无论是更新错误显示设置、加载其他开发人员工具还是其他任何内容。这些由系统自动加载。在初始安装中已经创建了以下文件:

- development.php
- production.php
- testing.php

例如, 如果你想添加 `staging` 环境用于暂存, 你只需要:

1. 将 **APPPATH/Config/Boot/production.php** 复制到 **staging.php**。
2. 如有必要, 在 **staging.php** 中自定义设置。

确认当前环境

要确认当前环境, 只需打印常量 `ENVIRONMENT`。

你也可以通过 `spark env` 命令检查当前环境:

```
php spark env
```

对默认框架行为的影响

CodeIgniter 系统中有一些地方使用了 `ENVIRONMENT` 常量。本节描述了默认框架行为如何受到影响。

错误报告

将 `ENVIRONMENT` 常量设置为 `development` 值将导致所有 PHP 错误在发生时渲染到浏览器。相反, 将常量设置为 `production` 将禁用所有错误输出。在生产中禁用错误报告是一项很好的安全实践。

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 URI 路由

- 什么是 *URI* 路由？
- 设置路由规则
  - 示例
  - *HTTP* 方法路由
  - 指定路由处理器
    - 控制器的命名空间
    - 数组可调用语法
    - 使用闭包
  - 指定路由路径

    * 占位符
    * 自定义占位符
    * 正则表达式
  - 视图路由
  - 重定向路由
  - 环境限制
  - 任意 *HTTP* 方法路由
  - 批量映射路由
  - 仅命令行路由
- 全局选项
  - 应用过滤器
    * 别名过滤器
    * 类名过滤器
    * 多重过滤器
  - 分配命名空间
  - 限制主机名
    * 多主机名限制
  - 限制子域
  - 偏移匹配参数
- 反向路由
- 命名路由
- 分组路由
  - 设置命名空间
  - 设置过滤器
  - 设置其他选项
  - 嵌套分组

- 路由优先级
  - 调整路由优先级
- 路由配置选项
  - 默认命名空间
  - 默认方法
  - 转换 *URI* 短横线
  - 仅使用定义路由
  - *404* 重写
  - 按优先级处理路由
  - 多 *URI* 段作为单一参数
- 自动路由（改进版）
- 自动路由（传统版）
  - 启用传统自动路由
  - *URI* 分段（传统版）
  - 配置选项（传统版）
    - 默认控制器 *(*传统版*)*
    - 默认方法（传统版）
- 验证路由
  - *spark* 路由
    - 自动路由 *(*改进版*)*
    - 自动路由 *(*传统*)*
    - 按处理程序排序
    - 指定主机
- 获取路由信息
  - 检索当前控制器/方法名称
  - 获取当前路由的活动过滤器

什么是 URI 路由？

URI 路由将 URI 与控制器的方法关联起来。

CodeIgniter 有两种路由方式。一种是 **定义式路由**，另一种是 **自动路由**。通过定义式路由，你可以手动定义路由规则，这种方式允许更灵活的 URL 结构。自动路由则基于约定自动映射 HTTP 请求到对应的控制器方法，无需手动定义路由。

首先我们来看定义式路由。如需使用自动路由，请参阅自动路由（改进版）。

设置路由规则

路由规则定义在 **app/Config/Routes.php** 文件中。在该文件中，你会看到创建了一个 RouteCollection 类的实例（`$routes`），用于指定自定义路由条件。可以使用占位符或正则表达式来定义路由。

当定义路由时，需选择与 HTTP 方法（请求方法）对应的路由方法。例如处理 GET 请求时使用 `get()` 方法：

```
<?php

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

路由左侧指定 **路由路径**（相对于 BaseURL 的 URI 路径，以 / 开头），右侧映射到 **路由处理器**（控制器和方法 `Home::index`），并可传递参数给控制器。

控制器和方法应按静态方法的形式列出，使用双冒号分隔类和方法，例如 `Users::list`。

若方法需要参数，可在方法名后使用斜杠分隔：

```
<?php

// Calls $Users->list()
$routes->get('users', 'Users::list');

// Calls $Users->list(1, 23)
$routes->get('users/1/23', 'Users::list/1/23');
```

示例

以下是几个基础路由示例：

当 URL 第一段包含 **journals** 时，将映射到 `\App\Controllers\Blogs` 类，并调用默认方法（通常为 `index()`）：

```
<?php

$routes->get('journals', 'Blogs');
```

当 URL 包含 **blog/joe** 时，映射到 `\App\Controllers\Blogs` 类的 `users()` 方法，ID 参数设为 `34`：

```
<?php

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

当 URL 第一段为 **product**，第二段为任意内容时，映射到 `\App\Controllers\Catalog` 类的 `productLookup()` 方法：

```
<?php

$routes->get('product/(:segment)', 'Catalog::productLookup');
```

当 URL 第一段为 **product**，第二段为数字时，映射到 `\App\Controllers\Catalog` 类的 `productLookupByID()` 方法，并将匹配值作为参数传递：

```
<?php

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

HTTP 方法路由

可以使用任意标准 HTTP 方法（GET、POST、PUT、DELETE、OPTIONS 等）：

```
<?php

$routes->post('products', 'Product::feature');
```

(续下页)

(接上页)

```
$routes->put('products/1', 'Product::feature');
$routes->delete('products/1', 'Product::feature');
```

通过 `match()` 方法传入方法数组，可匹配多个 HTTP 方法：

```
<?php

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

指定路由处理器

控制器的命名空间

当以字符串形式指定控制器和方法名时，若控制器名称未以 \ 开头，系统会自动添加默认命名空间：

```
<?php

// Routes to \App\Controllers\Api\Users::update()
$routes->post('api/users', 'Api\Users::update');
```

若以 \ 开头，则视为完全限定类名：

```
<?php

// Routes to \Acme\Blog\Controllers\Home::list()
$routes->get('blog', '\Acme\Blog\Controllers\Home::list');
```

也可通过 `namespace` 选项指定命名空间：

```
<?php

// Routes to \Admin\Users::index()
$routes->get('admin/users', 'Users::index', ['namespace' => 'Admin
↪']);
```

详见分配命名空间。

数组可调用语法

在 4.2.0 版本加入.

从 v4.2.0 开始，可使用数组可调用语法指定控制器：

```
$routes->get('/', [\App\Controllers\Home::class, 'index']);
```

或使用 use 关键字：

```
use App\Controllers\Home;

$routes->get('/', [Home::class, 'index']);
```

若忘记添加 `use App\Controllers\Home;`，控制器类名将被解析为 `\Home` 而非 `App\Controllers\Home`。

备注: 使用数组可调用语法时，类名始终视为完全限定类名，因此默认命名空间和*namespace 选项* 将失效。

数组可调用语法与占位符

若存在占位符，参数将按指定顺序自动设置：

```
use App\Controllers\Product;

$routes->get('product/(:num)/(:num)', [Product::class, 'index']);

// The above code is the same as the following:
$routes->get('product/(:num)/(:num)', 'Product::index/$1/$2');
```

但在路由中使用正则表达式时，自动配置的参数可能不正确。此时可手动指定参数：

```
use App\Controllers\Product;

$routes->get('product/(:num)/(:num)', [[Product::class, 'index'], '
↪$2/$1']);
```

(续下页)

(接上页)

```
// The above code is the same as the following:
$routes->get('product/(:num)/(:num)', 'Product::index/$2/$1');
```

使用闭包

可使用匿名函数（闭包）作为路由目标。当用户访问对应 URI 时，该函数将被执行，适用于快速执行小任务或显示简单视图：

```
<?php

use App\Libraries\RSSFeeder;

$routes->get('feed', static function () {
    $rss = new RSSFeeder();

    return $rss->feed('general');
});
```

指定路由路径

占位符

典型路由示例如下：

```
<?php

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

路由的第一个参数是待匹配的 URI，第二个参数是目标路由。当 URL 路径第一段为 “product” 且第二段为数字时，将使用 `Catalog` 类的 `productLookup` 方法。

占位符是代表正则表达式模式的字符串，在路由过程中会被替换为实际正则表达式，主要用于提升可读性。

可用占位符列表：

占位符	描述
(:any)	匹配从该位置到 URI 末尾的所有字符，可能包含多个段
(:segment)	匹配除斜杠（/）外的任意字符，限制为单个段
(:num)	匹配任意整数
(:alpha)	匹配任意字母字符串
(:alphanum)	匹配任意字母数字组合字符串
(:hash)	与（`:segment`）相同，便于识别使用哈希 ID 的路由

备注： `{locale}` 不能作为占位符或路由其他部分，保留用于本地化。

(:any) 的行为

注意单个（`:any`）会匹配 URL 中的多个段（如果存在）。

例如路由：

```
<?php

$routes->get('product/(:any)', 'Catalog::productLookup/$1');
```

将匹配 **product/123**、**product/123/456**、**product/123/456/789** 等。

默认情况下，上述示例中若 `$1` 占位符包含斜杠（/），传递给 `Catalog::productLookup()` 时仍会分割为多个参数。

备注： 自 v4.5.0 起，可通过配置选项修改此行为，详见*多 URI 段作为单一参数*。

控制器实现应考虑最大参数数量：

```
<?php

namespace App\Controllers;

class Catalog extends BaseController
{
```

(续下页)

(接上页)

```
    public function productLookup($seg1 = false, $seg2 = false,
↪$seg3 = false)
    {
        echo $seg1; // Will be 123 in all examples
        echo $seg2; // false in first, 456 in second and third␣
↪example
        echo $seg3; // false in first and second, 789 in third
    }
}
```

或使用 可变数量的参数值列表:

```
<?php

namespace App\Controllers;

class Catalog extends BaseController
{
    public function productLookup(...$params)
    {
        echo $params[0] ?? null; // Will be 123 in all examples
        echo $params[1] ?? null; // null in first, 456 in second␣
↪and third example
        echo $params[2] ?? null; // null in first and second, 789␣
↪in third
    }
}
```

重要: 请勿在 (:any) 后放置其他占位符，因为传递给控制器方法的参数数量可能变化。

若不需要匹配多段，应在定义路由时使用 (:segment):

```
<?php

$routes->get('product/(:segment)', 'Catalog::productLookup/$1');
```

该路由仅匹配 **product/123**，其他情况返回 404 错误。

自定义占位符

可创建自定义占位符来完全定制路由体验。

使用 `addPlaceholder()` 方法添加新占位符，第一个参数是占位符字符串，第二个参数是替换的正则表达式。需在添加路由前调用：

```
<?php

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

正则表达式

可使用正则表达式定义路由规则。允许任何有效正则表达式及反向引用。

重要： 注意：使用反向引用时需使用美元符号语法而非双反斜杠语法。典型正则路由示例：

```
<?php

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

上述示例中，类似 **products/shirts/123** 的 URI 将调用 `Products` 控制器的 `show()` 方法，原始第一、二段作为参数传递。

通过正则表达式可捕获包含斜杠的段（通常用于分隔多个段）。例如用户访问受密码保护区域后重定向回原页面：

```
<?php

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

默认情况下，若 `$1` 占位符包含斜杠，传递给 `Auth::login()` 时仍会分割为多个参数。

备注： 自 v4.5.0 起，可通过配置选项修改此行为，详见*多 URI 段作为单一参数*。

关于正则表达式学习，推荐访问 regular-expressions.info。

备注： 可混合使用占位符和正则表达式。

视图路由

在 4.3.0 版本加入.

若只需渲染无逻辑视图，可使用 `view()` 方法（始终视为 GET 请求）。第二个参数指定视图名称：

```
<?php

// Displays the view in /app/Views/pages/about.php
$routes->view('about', 'pages/about');
```

若路由中使用占位符，可通过 `$segments` 数组在视图中访问：

```
<?php

// Displays the view in /app/Views/map.php
$routes->view('map/(:segment)/(:segment)', 'map');

// Within the view, you can access the segments with
// $segments[0] and $segments[1] respectively.
```

重定向路由

网站改版常需页面重定向。使用 `addRedirect()` 方法指定旧路由重定向到新路由。第一个参数是旧路由 URI 模式，第二个参数是新 URI 或命名路由名称，第三个参数是 HTTP 状态码（默认 302 临时重定向）：

```
<?php

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

// Redirect to a named route
$routes->addRedirect('users/about', 'profile');
// Redirect to a URI
$routes->addRedirect('users/about', 'users/profile');

// Redirect with placeholder
$routes->get('post/(:num)/comment/(:num)', 'PostsComments::index', [
↪'as' => 'post.comment']);

// Redirect to a URI
$routes->addRedirect('article/(:num)/(:num)', 'post/$1/comment/$2');
// Redirect to a named route
$routes->addRedirect('article/(:num)/(:num)', 'post.comment');
```

备注: 自 v4.2.0 起，`addRedirect()` 支持占位符。

匹配重定向路由时，用户将在控制器加载前立即跳转。

环境限制

可创建仅特定环境可见的路由（如开发者本地工具）。使用 `environment()` 方法，参数为环境名称，闭包内定义的路由仅在该环境下可用：

```
<?php

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

任意 HTTP 方法路由

重要: 此方法仅为向后兼容保留，新项目请勿使用。建议使用更合适的 HTTP 方法路由。

警告: 使用*CSRF 保护* 时，不会保护 **GET** 请求。若 `add()` 方法指定的 URI 可通过 GET 访问，CSRF 保护将失效。

使用 `add()` 方法定义支持任意 HTTP 方法的路由：

```
<?php

$routes->add('products', 'Product::feature');
```

备注: 使用 HTTP 方法路由可提升性能，因为仅存储匹配当前请求方法的路由。

批量映射路由

重要: 此方法仅为向后兼容保留，新项目请勿使用。建议使用更合适的方法。

警告: 由于 `map()` 内部调用 `add()`，同样不推荐使用。

使用 `map()` 方法批量定义路由数组：

```
<?php

$multipleRoutes = [
    'product/(:num)'       => 'Catalog::productLookupById',
    'product/(:alphanum)' => 'Catalog::productLookupByName',
```

(续下页)

(接上页)

```
];

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

仅命令行路由

备注: 建议使用 Spark 命令处理 CLI 脚本，而非通过 CLI 调用控制器。详见*创建 Spark 命令*。

通过 HTTP 方法创建的路由 CLI 不可访问，但 `add()` 创建的路由仍可在命令行使用。使用 `cli()` 方法创建仅 CLI 可用的路由：

```
<?php

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

警告: 若启用*自动路由（传统版）*并将命令文件置于 **app/Controllers**，他人可能通过自动路由（传统）HTTP 访问该命令。

全局选项

所有路由创建方法（`get()`、`post()`、*resource()* 等）均可接受选项数组作为最后一个参数，用于修改或限制生成的路由：

```
<?php

$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);
```

(续下页)

(接上页)

```
$routes->patch('from', 'to', $options);
$routes->match(['GET', 'PUT'], 'from', 'to', $options);
$routes->resource('photos', $options);
$routes->map($array, $options);
$routes->group('name', $options, static function () {});
```

应用过滤器

可通过为路由添加过滤器来修改特定路由行为（如身份验证或 API 日志记录）。过滤器值可以是字符串或字符串数组：

- 匹配 **app/Config/Filters.php** 中定义的别名
- 过滤器类名

详见控制器过滤器。

> **警告：** 若在 **app/Config/Routes.php** 设置路由过滤器（非 **app/Config/Filters.php**），建议禁用自动路由（传统）。启用自动路由（传统版）时，控制器可能通过不同 URL 访问，导致路由过滤器未生效。详见仅使用定义路由。

别名过滤器

你可以为过滤器值指定一个*在 app/Config/Filters.php 中定义* 的别名。

```
<?php

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

可为别名过滤器的 `before()` 和 `after()` 方法传递参数：

```
<?php

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

类名过滤器

在 4.1.5 版本加入.

直接指定过滤器类名:

```
<?php

$routes->get('admin', ' AdminController::index', ['filter' => \App\
↪Filters\SomeFilter::class]);
```

多重过滤器

在 4.1.5 版本加入.

重要: 自 v4.5.0 起始终启用 多重过滤器, v4.5.0 之前默认禁用, 如需使用请参考*从 4.1.4 升级到 4.1.5*。

指定过滤器数组:

```
<?php

$routes->get('admin', ' AdminController::index', ['filter' => [
↪'admin-auth', \App\Filters\SomeFilter::class]]);
```

过滤器参数

可向过滤器传递额外参数:

```
<?php

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

此例中数组 `['dual', 'noreturn']` 将作为 `$arguments` 传递给过滤器的 `before()` 和 `after()` 方法。

分配命名空间

虽然系统会自动添加默认命名空间 到生成的控制器，但可通过 namespace 选项指定不同命名空间：

```
<?php

// Routes to \Admin\Users::index()
$routes->get('admin/users', 'Users::index', ['namespace' => 'Admin
↪']);
```

新命名空间仅作用于单路由创建方法（如 get、post 等）。对于创建多路由的方法（如 group()），新命名空间将附加到所有生成的路由。

限制主机名

通过 “hostname” 选项限制路由组仅在特定域名或子域生效：

```
<?php

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

此示例仅允许 **accounts.example.com** 域名访问，主域 **example.com** 不可用。

多主机名限制

在 4.6.0 版本加入.

支持多个主机名限制：

```
<?php

$routes->get('from', 'to', ['hostname' => ['s1.example.com', 's2.
↪example.com']]);
```

限制子域

通过 subdomain 选项限制路由仅在特定子域可用：

```
<?php

// Limit to media.example.com
$routes->get('from', 'to', ['subdomain' => 'media']);
```

设置值为星号（*）可匹配任意子域，但无子域的 URL 不匹配：

```
<?php

// Limit to any sub-domain
$routes->get('from', 'to', ['subdomain' => '*']);
```

重要： 此功能并非完美，生产环境前需充分测试。某些含点的域名可能导致误判。

偏移匹配参数

通过 offset 选项数值偏移匹配参数。适用于 API 版本号或语言字符串作为首段的情况：

```
<?php

$routes->get('users/(:num)', 'users/show/$1', ['offset' => 1]);

// Creates:
$routes['users/(:num)'] = 'users/show/$2';
```

反向路由

反向路由允许通过控制器、方法及参数定义链接，由路由器查找当前路由。这使得修改路由定义无需更新应用代码，常用于视图创建链接。

使用*url_to()* 辅助函数获取路由。第一个参数是完全限定的控制器和方法（用双冒号分隔），后续参数传递路由参数：

```
<?php

// The route is defined as:
$routes->get('users/(:num)/gallery/(:num)',
↪'Galleries::showUserGallery/$1/$2');

?>

<!-- Generate the URI to link to user ID 15, gallery 12: -->
<a href="<?= url_to('Galleries::showUserGallery', 15, 12) ?>">View␣
↪Gallery</a>
<!-- Result: 'http://example.com/users/15/gallery/12' -->
```

命名路由

你可以为路由命名以使你的应用更健壮。这为路由应用一个名称，之后可以被调用，即使路由定义发生变化，应用中所有使用*url_to()* 构建的链接仍能正常工作，而无需你做任何修改。通过传递 as 选项并指定路由名称来为路由命名：

```
<?php

// The route is defined as:
$routes->get('users/(:num)/gallery/(:num)',
↪'Galleries::showUserGallery/$1/$2', ['as' => 'user_gallery']);

?>

<!-- Generate the URI to link to user ID 15, gallery 12: -->
<a href="<?= url_to('user_gallery', 15, 12) ?>">View Gallery</a>
<!-- Result: 'http://example.com/users/15/gallery/12' -->
```

这样做还有一个额外的好处，就是让视图更具可读性。

分组路由

你可以使用 `group()` 方法将路由分组到一个共用名称下。分组名称会成为出现在组内定义路由之前的一个路径段。这允许你减少构建大量共享相同开头字符串的路由所需的输入量，例如在构建管理区域时：

```
<?php

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

这将会为 **users** 和 **blog** URI 添加 **admin** 前缀，处理如 **admin/users** 和 **admin/blog** 的 URL。

设置命名空间

如果你需要为分组分配选项，例如分配命名空间，请在回调函数之前进行设置：

```
<?php

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

这将处理指向 `App\API\v1\Users` 控制器的资源路由，对应的 URI 为 **api/users**。

设置过滤器

你也可以为一组路由使用特定的过滤器。这将在控制器之前或之后始终运行该过滤器。这在身份验证或 API 日志记录场景中特别有用：

```
<?php

$routes->group('api', ['filter' => 'api-auth'], static function (
```

(续下页)

(接上页)

```
↪$routes) {
    $routes->resource('users');
});
```

过滤器的值必须与 **app/Config/Filters.php** 中定义的别名之一匹配。

备注: 在 v4.5.4 之前的版本中，由于存在 bug，传递给 `group()` 的过滤器不会合并到传递给内部路由的过滤器中。

设置其他选项

有时可能需要为路由组应用过滤器或其他配置选项（如命名空间、子域等），而无需添加前缀。此时可将前缀设为空字符串：

```
<?php

$routes->group('', ['namespace' => 'Myth\Auth\Controllers'], static␣
↪function ($routes) {
    $routes->get('login', 'AuthController::login', ['as' => 'login
↪']);
    $routes->post('login', 'AuthController::attemptLogin');
    $routes->get('logout', 'AuthController::logout');
});
```

嵌套分组

支持多层级分组以实现更精细的组织结构：

```
<?php

$routes->group('admin', ['filter' => 'myfilter1:config'], static␣
↪function ($routes) {
    $routes->get('/', 'Admin\Admin::index');

    $routes->group('users', ['filter' => 'myfilter2:region'],␣
```

(续下页)

(接上页)

```
↪static function ($routes) {
        $routes->get('list', 'Admin\Users::list');
    });
});
```

此例将处理 **admin/users/list** URL。外层 `group()` 的 `filter` 选项会与内层 `group()` 的选项合并。上述代码中，`admin` 路由运行 `myfilter1:config` 过滤器，`admin/users/list` 路由运行 `myfilter1:config` 和 `myfilter2:region` 过滤器。

备注： v4.6.0 之前，同一过滤器无法使用不同参数多次运行。

内层 `group()` 的选项会覆盖外层同名选项。

备注： v4.5.0 之前存在 bug，外层 `group()` 的选项不会与内层合并。

路由优先级

路由按定义顺序注册到路由表中。当访问 URI 时，将执行首个匹配的路由。

警告： 若同一路由路径被多次定义且处理器不同，仅首个定义的路由生效。

可通过运行*spark routes* 命令查看路由表。

调整路由优先级

处理模块路由时，若应用路由包含通配符可能导致模块路由无法正确处理。通过 `priority` 选项可降低路由处理优先级（数值越大优先级越低）：

```
<?php

// First you need to enable processing of the routes queue by␣
↪priority.
$routes->setPrioritize();
```

(续下页)

(接上页)

```
// Config\Routes
$routes->get('(.*)', 'Posts::index', ['priority' => 1]);

// Modules\Acme\Config\Routes
$routes->get('admin', 'Admin::index');

// The "admin" route will now be processed before the wildcard␣
↪route.
```

要禁用此功能，传入 `false` 参数：

```
<?php

$routes->setPrioritize(false);
```

备注： 默认所有路由优先级为 0，负值将转为绝对值。

路由配置选项

RouteCollection 类提供多个全局配置选项（位于 **app/Config/Routing.php**），可根据需求调整。

备注： **app/Config/Routing.php** 配置文件自 v4.4.0 起新增，旧版本需在 **app/Config/Routes.php** 使用 setter 方法修改设置。

默认命名空间

匹配控制器时，系统会将默认命名空间值添加到控制器名称前（默认 `App\Controllers`）。设为空字符串（`''`）则需每个路由指定完全限定命名空间：

```
<?php
```

(续下页)

(接上页)

```
// In app/Config/Routing.php
use CodeIgniter\Config\Routing as BaseRouting;

// ...
class Routing extends BaseRouting
{
    // ...
    public string $defaultNamespace = '';
    // ...
}

// In app/Config/Routes.php
// Controller is \Users
$routes->get('users', 'Users::index');

// Controller is \Admin\Users
$routes->get('users', 'Admin\Users::index');
```

若控制器已命名空间化，可修改此值减少输入：

```
<?php

// This can be overridden in app/Config/Routes.php
$routes->setDefaultNamespace('App');

// Controller is \App\Users
$routes->get('users', 'Users::index');

// Controller is \App\Admin\Users
$routes->get('users', 'Admin\Users::index');
```

默认方法

当路由处理器仅指定控制器名时，使用此设置的方法（默认 index）:

```
// In app/Config/Routing.php
public string $defaultMethod = 'index';
```

备注: $defaultMethod 也常用于自动路由。请参见自动路由（改进版）或自动路由（传统版）。

如果你定义了以下路由:

```
$routes->get('/', 'Home');
```

当路由匹配时，将执行 App\Controllers\Home 控制器的 index() 方法。

备注: 方法名称以 _ 开头时不能用作默认方法。但是，从 v4.5.0 开始，允许使用 __invoke 方法。

转换 URI 短横线

此选项在自动路由中将短横线（-）自动转为下划线（因短横线非有效类/方法名字符）:

```
<?php

// In app/Config/Routing.php
use CodeIgniter\Config\Routing as BaseRouting;

// ...
class Routing extends BaseRouting
{
    // ...
    public bool $translateURIDashes = true;
    // ...
}
```

(续下页)

(接上页)

```
// This can be overridden in app/Config/Routes.php
$routes->setTranslateURIDashes(true);
```

备注： 在使用自动路由（改进版）时，在 v4.4.0 之前，如果 `$translateURIDashes` 为 true，两个 URI 对应一个控制器方法，一个 URI 用于破折号（例如 **foo-bar**），另一个 URI 用于下划线（例如 **foo_bar**）。这是错误的行为。从 v4.4.0 开始，下划线的 URI（**foo_bar**）不可访问。

仅使用定义路由

v4.2.0 起默认禁用自动路由。

当未找到与当前 URI 匹配的定义路由时，系统尝试通过自动路由匹配控制器方法。将 `$autoRoute` 设为 `false` 可完全禁用自动路由：

```
<?php

// In app/Config/Routing.php
use CodeIgniter\Config\Routing as BaseRouting;

// ...
class Routing extends BaseRouting
{
    // ...
    public bool $autoRoute = false;
    // ...
}

// This can be overridden in app/Config/Routes.php
$routes->setAutoRoute(false);
```

警告： 启用*CSRF 保护* 时，**GET** 请求不受保护。若 URI 可通过 GET 访问，CSRF 保护将失效。

404 重写

当找不到与当前 URI 匹配的页面时，系统将显示一个通用的 404 页面。通过在路由配置文件中使用 `$override404` 属性，你可以为 404 路由定义控制器类/方法。

```
<?php

// In app/Config/Routing.php
use CodeIgniter\Config\Routing as BaseRouting;

// ...
class Routing extends BaseRouting
{
    // ...
    public ?string $override404 = 'App\Errors::show404';
    // ...
}
```

你还可以在路由配置文件中使用 `set404Override()` 方法指定在发生 404 错误时执行的操作。该值可以是一个有效的类/方法对，或者是一个闭包：

```
<?php

// In app/Config/Routes.php
// Would execute the show404 method of the App\Errors class
$routes->set404Override('App\Errors::show404');

// Will display a custom view.
$routes->set404Override(static function () {
    // If you want to get the URI segments.
    $segments = request()->getUri()->getSegments();

    return view('my_errors/not_found.html');
});
```

备注： 从 v4.5.0 开始，404 覆盖功能默认将响应状态代码设置为 `404`。在之前的版本中，状态代码是 `200`。如果你想在控制器中更改状态代码，请参见*CodeIgniter\HTTP*

Response::setStatusCode() 获取有关如何设置状态代码的信息。

按优先级处理路由

启用或禁用按优先级处理路由队列。在路由选项中降低优先级。默认禁用。此功能影响所有路由。有关降低优先级的示例用法，请参阅路由优先级：

```
<?php

// In app/Config/Routing.php
use CodeIgniter\Config\Routing as BaseRouting;

// ...
class Routing extends BaseRouting
{
    // ...
    public bool $prioritize = true;
    // ...
}

// In app/Config/Routes.php
// to enable
$routes->setPrioritize();

// to disable
$routes->setPrioritize(false);
```

多 URI 段作为单一参数

在 4.5.0 版本加入.

启用此选项后，匹配多段的占位符（如 `(:any)`）将作为单一参数传递（即使包含斜杠）：

```
<?php

// In app/Config/Routing.php
use CodeIgniter\Config\Routing as BaseRouting;
```

(续下页)

(接上页)

```
class Routing extends BaseRouting
{
    // ...
    public bool $multipleSegmentsOneParam = true;
    // ...
}
```

例如路由：

```
<?php

$routes->get('product/(:any)', 'Catalog::productLookup/$1');
```

将匹配 **product/123**、**product/123/456**、**product/123/456/789** 等等。如果 URI 是 **product/123/456**，`123/456` 将被传递给 `Catalog::productLookup()` 方法的第一个参数。

自动路由（改进版）

在 4.2.0 版本加入.

这是更安全的新自动路由系统，详见自动路由（改进版）。

自动路由（传统版）

重要： 这个功能只为了向后兼容而存在。在新项目中不要使用它。即使你已经在使用它，我们也推荐你使用自动路由（改进版）替代。

自动路由 (传统) 是来自 CodeIgniter 3 的路由系统。它可以根据约定自动路由 HTTP 请求, 并执行相应的控制器方法。

推荐在 **app/Config/Routes.php** 文件中定义所有路由, 或者使用自动路由（改进版）。

> **警告：** 为了防止配置错误和编码错误, 我们建议你不要使用自动路由 (传统) 功能。很容易创建容易受攻击的应用程序, 其中控制器过滤器或 CSRF 保护被绕过。

重要: 自动路由 (传统) 会将任何 HTTP 方法的 HTTP 请求路由到控制器方法。

启用传统自动路由

自 v4.2.0 起，默认禁用自动路由。

要使用它, 你需要在 **app/Config/Routing.php** 中将 `$autoRoute` 选项设置为 `true`:

```
public bool $autoRoute = true;
```

并且在 **app/Config/Feature.php** 中，将属性 `$autoRoutesImproved` 设置为 `false`:

```
public bool $autoRoutesImproved = false;
```

URI 分段（传统版）

遵循模型-视图-控制器（MVC）模式，URL 中的各段通常表示:

```
example.com/class/method/ID
```

1. 第一段表示应被调用的控制器 **class**
2. 第二段表示应被调用的类 **method**
3. 第三段及后续各段表示将传递给控制器的 ID 和其他变量

考虑以下 URI:

```
example.com/index.php/helloworld/index/1
```

在上述示例中，CodeIgniter 将尝试查找名为 **Helloworld.php** 的控制器，并执行 `index()` 方法，同时传递 `'1'` 作为第一个参数。

更多信息请参阅*控制器中的自动路由 (传统模式)*。

配置选项（传统版）

这些选项在 **app/Config/Routing.php** 文件中可用。

默认控制器 (传统版)

针对网站根 URI(传统版)

当用户访问你网站的根（例如，**example.com**）时，除非存在明确的路由，否则将根据 `$defaultController` 属性设定的值来确定要使用的控制器。

对于这个属性，默认值是 `Home`，它匹配在 **app/Controllers/Home.php** 的控制器:

```
public string $defaultController = 'Home';
```

针对目录 URI(传统版)

默认控制器也在未找到匹配的路由且 URI 指向控制器目录中的目录时使用。例如, 如果用户访问 **example.com/admin**, 如果在 **app/Controllers/Admin/Home.php** 中找到了一个控制器, 则会使用它。

更多信息请参阅*控制器中的自动路由 (传统)*。

默认方法（传统版）

这与默认控制器设置类似, 但用于在找到与 URI 匹配的控制器但不存在方法段时确定使用的默认方法。默认值为 `index`。

在此示例中, 如果用户访问 **example.com/products**, 且存在 `Products` 控制器, 将执行 `Products::listAll()` 方法:

public string $defaultMethod = ‘listAll’;

验证路由

通过*spark 命令* 查看所有路由：

spark 路由

显示所有路由及过滤器：

```
php spark routes
```

输出示例：

```
+---------+---------+---------------+-------------------------------
↪+----------------+---------------+
| Method  | Route   | Name          | Handler                       ␣
↪| Before Filters | After Filters |
+---------+---------+---------------+-------------------------------
↪+----------------+---------------+
| GET     | /       | »             | \App\Controllers\Home::index ␣
↪|                | toolbar       |
| GET     | feed    | »             | (Closure)                     ␣
↪|                | toolbar       |
+---------+---------+---------------+-------------------------------
↪+----------------+---------------+
```

Method 列显示路由监听的 HTTP 方法。

Route 列显示要匹配的路由路径。定义路由的路由以正则表达式表示。

自 v4.3.0 起, *Name* 列显示路由名称。» 表示名称与路由路径相同。

重要： 系统并非完美。对于包含如 `([^/]+)` 或 `{locale}` 的正则表达式模式的路由，显示的 *Filters* 可能不正确（如果你在 **app/Config/Filters.php** 中为过滤器设置了复杂的 URI 模式），或者它显示为 `<unknown>`。

spark filter:check 命令可以用来检查 100% 准确的过滤器。

自动路由 (改进版)

当你使用自动路由 (改进版) 时, 输出类似以下内容:

```
+-----------+-----------------------+---------------+-------------
↪----------------------+----------------+---------------+
| Method    | Route                 | Name          | Handler    ␣
↪                      | Before Filters | After Filters |
+-----------+-----------------------+---------------+-------------
↪----------------------+----------------+---------------+
| GET(auto) | product/list/../..[/..] |               | \App\
↪Controllers\Product::getList |                | toolbar       |
+-----------+-----------------------+---------------+-------------
↪----------------------+----------------+---------------+
```

Method 将显示为 `GET(auto)`。

Route 列中的 `/..` 表示一个段。`[/..]` 表示可选。

备注: 当启用自动路由并且你有 `home` 路由时, 它也可以通过 `Home` 访问, 或者通过 `hOme`、`hoMe`、`HOME` 等访问, 但是该命令只会显示 `home`。

如果你看到以 `x` 开头的路由, 如下所示, 这表示一个无效路由, 不会路由, 但是控制器有公共方法进行路由。

```
+-----------+----------------+------+-----------------------------
↪------+----------------+---------------+
| Method    | Route          | Name | Handler                     ␣
↪      | Before Filters | After Filters |
+-----------+----------------+------+-----------------------------
↪------+----------------+---------------+
| GET(auto) | x home/foo     |      | \App\Controllers\Home::getFoo␣
↪      | <unknown>      | <unknown>     |
+-----------+----------------+------+-----------------------------
↪------+----------------+---------------+
```

上面的示例显示你有 `\App\Controllers\Home::getFoo()` 方法, 但是它没有路由, 因为它是默认控制器 (默认为 `Home`), 默认控制器名称必须在 URI 中省略。你应该删除 `getFoo()` 方法。

备注: 在 v4.3.4 之前, 由于一个错误, 无效路由会显示为正常路由。

自动路由 (传统)

当你使用自动路由 (传统) 时, 输出类似以下内容:

```
+--------+--------------------+---------------+---------------------
↪--------------+----------------+---------------+
| Method | Route              | Name          | Handler            ␣
↪              | Before Filters | After Filters |
+--------+--------------------+---------------+---------------------
↪--------------+----------------+---------------+
| auto   | product/list[/...] |               | \App\Controllers\
↪Product::getList |                | toolbar       |
+--------+--------------------+---------------+---------------------
↪--------------+----------------+---------------+
```

Method 将显示为 `auto`。

Route 列中的 `[/...]` 表示任意数量的段。

备注: 当启用自动路由并且你有 `home` 路由时, 它也可以通过 `Home` 访问, 或者通过 `hOme`、`hoMe`、`HOME` 等访问, 但是该命令只会显示 `home`。

按处理程序排序

在 4.3.0 版本加入.

你可以按 *Handler* 对路由进行排序:

```
php spark routes -h
```

指定主机

在 4.4.0 版本加入.

通过 `--host` 指定请求主机:

```
php spark routes --host accounts.example.com
```

获取路由信息

在 CodeIgniter 4 中，理解和管理路由信息对于有效处理 HTTP 请求至关重要。这涉及到检索有关活动控制器和方法的详细信息，以及应用于特定路由的过滤器。下面，我们探讨如何访问这些路由信息，以帮助完成诸如日志记录、调试或实现条件逻辑等任务。

检索当前控制器/方法名称

在某些情况下，你可能需要确定当前 HTTP 请求触发了哪个控制器和方法。这对于日志记录、调试或基于活动控制器方法的条件逻辑非常有用。

CodeIgniter 4 提供了一种简单的方法来使用 `Router` 类访问当前路由的控制器和方法名称。以下是一个示例:

```
<?php

// Get the router instance.
/** @var \CodeIgniter\Router\Router $router */
$router = service('router');

// Retrieve the fully qualified class name of the controller␣
↪handling the current request.
$controller = $router->controllerName();

// Retrieve the method name being executed in the controller for␣
↪the current request.
$method = $router->methodName();

echo 'Current Controller: ' . $controller . '<br>';
echo 'Current Method: ' . $method;
```

当你需要动态地与控制器交互或记录处理特定请求的方法时，这个功能特别有用。

获取当前路由的活动过滤器

过滤器 是一个强大的功能，使你能够在处理 HTTP 请求之前或之后执行诸如身份验证、日志记录和安全检查等操作。要访问特定路由的活动过滤器，你可以使用 `Router` 类中的 `CodeIgniter\Router\Router::getFilters()` 方法。

此方法返回当前正在处理的路由的活动过滤器列表：

```
<?php

// Get the router instance.
/** @var \CodeIgniter\Router\Router $router */
$router  = service('router');
$filters = $router->getFilters();

echo 'Active Filters for the Route: ' . implode(', ', $filters);
```

备注: `getFilters()` 方法仅返回为特定路由定义的过滤器。它不包括全局过滤器或在 **app/Config/Filters.php** 文件中指定的过滤器。

5.1.2 控制器

控制器是你应用程序的核心, 因为它们确定了如何处理 HTTP 请求。

- 什么是控制器?
- 构造函数
- 包含的属性
  - 请求对象
  - 响应对象
  - 日志对象
  - 辅助函数

- *forceHTTPS*
- 验证数据
  - *$this->validateData()*
  - *$this->validate()*
- 保护方法
- 自动路由（改进版）
- 自动路由 (传统)
  - 让我们试一试: *Hello World!*(传统)
  - 方法 (传统)
  - 将 *URI* 段传递给你的方法 (传统)
  - 默认控制器 (传统)
  - 将控制器组织到子目录中 (传统)
- 重映射方法调用
- 扩展控制器
- 就这样了！

什么是控制器?

一个控制器简单来说就是一个处理 HTTP 请求的类文件。*URI 路由* 将一个 URI 与一个控制器关联起来。控制器会返回一个视图字符串或 `Response` 对象。

你创建的每个控制器都应该扩展 `BaseController` 类。这个类为你所有的控制器提供了几个可用的功能。

构造函数

CodeIgniter 的控制器有一个特殊的构造函数 `initController()`。它会在 PHP 的构造函数 `__construct()` 执行后由框架调用。

如果你想重写 `initController()`，不要忘记在方法中添加 `parent::initController($request, $response, $logger);`:

```php
<?php

namespace App\Controllers;

use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
use Psr\Log\LoggerInterface;

class Product extends BaseController
{
    public function initController(
        RequestInterface $request,
        ResponseInterface $response,
        LoggerInterface $logger,
    ) {
        parent::initController($request, $response, $logger);

        // Add your code here.
    }

    // ...
}
```

重要: 你不能在构造函数中使用 `return`。所以 `return redirect()->to('route');` 不起作用。

`initController()` 方法设置了以下三个属性。

包含的属性

CodeIgniter 的控制器提供了这些属性。

请求对象

应用程序的主要请求实例 始终作为类属性 `$this->request` 可用。

响应对象

应用程序的主要响应实例 始终作为类属性 `$this->response` 可用。

日志对象

日志 类的一个实例作为类属性 `$this->logger` 可用。

辅助函数

你可以将辅助函数文件的数组定义为类属性。每当加载控制器时, 这些辅助函数文件都会自动加载到内存中, 以便你可以在控制器内的任何地方使用它们的方法:

```
<?php

namespace App\Controllers;

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

forceHTTPS

所有控制器中都可以使用强制通过 HTTPS 访问方法的便利方法:

```
<?php

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

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

```
<?php

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

备注: 你可以始终使用一些基于时间的常量, 包括 `YEAR`、`MONTH` 等。

验证数据

$this->validateData()

在 4.2.0 版本加入.

为了简化数据检查，控制器也提供了方便的方法 `validateData()`。

该方法接受 (1) 一个要验证的数据数组，(2) 规则数组，(3) 如果项目无效，显示自定义错误消息的可选数组，(4) 使用的可选数据库组。

在验证库文档 中有规则和消息数组格式的详细信息，以及可用的规则:

```
<?php

namespace App\Controllers;
```

(续下页)

(接上页)

```
class StoreController extends BaseController
{
    public function product(int $id)
    {
        $data = [
            'id'   => $id,
            'name' => $this->request->getPost('name'),
        ];

        $rule = [
            'id'   => 'integer',
            'name' => 'required|max_length[255]',
        ];

        if (! $this->validateData($data, $rule)) {
            return view('store/product', [
                'errors' => $this->validator->getErrors(),
            ]);
        }

        // ...
    }
}
```

$this->validate()

重要: 该方法仅存在于向后兼容中。在新项目中不要使用它。即使你已经在使用它，我们也建议你使用 `validateData()` 方法。

控制器也提供了便利的方法 `validate()`。

警告: 与其使用 `validate()`，不如使用 `validateData()` 仅验证 POST 数据。`validate()` 使用 `$request->getVar()`，它返回 `$_GET`，`$_POST` 或

> $_COOKIE 数据，按照该顺序（取决于 php.ini 的 request-order）。新的值会覆写旧的值。如果它们的名字相同，POST 值可能被 Cookie 覆写。

该方法在第一个参数中接受规则数组, 在可选的第二个参数中, 接受自定义错误消息的数组, 以在项目无效时显示。

在内部, 这使用控制器的 $this->request 实例来获取要验证的数据。

验证库文档 有关于规则和消息数组格式以及可用规则的详细信息:

```
<?php

namespace App\Controllers;

class UserController extends BaseController
{
    public function updateUser(int $userID)
    {
        if (! $this->validate([
            'email' => "required|is_unique[users.email,id,{$userID}]
↪",
            'name'  => 'required|alpha_numeric_spaces',
        ])) {
            // The validation failed.
            return view('users/update', [
                'errors' => $this->validator->getErrors(),
            ]);
        }

        // The validation was successful.

        // Get the validated data.
        $validData = $this->validator->getValidated();

        // ...
    }
}
```

警告: 当你使用 `validate()` 方法时，应该使用*getValidated()* 方法来获取经过验证的数据。因为 `validate()` 方法在内部使用了*Validation::withRequest()* 方法，并且它会验证来自*$request->getJSON()*、*$request->getRawInput()* 或*$request->getVar()* 的数据，而攻击者可能会更改要验证的数据。

备注: 自 v4.4.0 版本开始，可以使用*$this->validator->getValidated()* 方法。

如果你发现在配置文件中保持规则更简单, 你可以用 **app/Config/Validation.php** 中定义的组名替换 `$rules` 数组:

```
<?php

namespace App\Controllers;

class UserController extends BaseController
{
    public function updateUser(int $userID)
    {
        if (! $this->validate('userRules')) {
            // The validation failed.
            return view('users/update', [
                'errors' => $this->validator->getErrors(),
            ]);
        }

        // The validation was successful.

        // Get the validated data.
        $validData = $this->validator->getValidated();

        // ...
    }
}
```

备注: 验证也可以在模型中自动处理, 但有时在控制器中更容易。由你决定在哪里。

保护方法

在某些情况下, 你可能希望某些方法隐藏不对公众开放。为此, 只需将方法声明为 `private` 或 `protected`。这将阻止通过 URL 请求提供服务。

例如, 如果你为 `Helloworld` 控制器定义了一个这样的方法:

```
<?php

namespace App\Controllers;

class Helloworld extends BaseController
{
    protected function utility()
    {
        // some code
    }
}
```

并为该方法定义一个路由 (`helloworld/utitilty`)。然后尝试使用以下 URL 访问它不会起作用:

```
example.com/index.php/helloworld/utility
```

自动路由也不会起作用。

自动路由（改进版）

在 4.2.0 版本加入.

自动路由（改进版）是一个全新的、更安全的自动路由系统。

详见自动路由（改进版）。

自动路由 (传统)

重要: 该功能仅存在于向后兼容中。在新项目中不要使用它。即使你已经在使用它，我们也建议你使用自动路由（改进版） 替代。

本节描述自动路由 (传统) 的功能, 这是 CodeIgniter 3 的路由系统。它会自动路由 HTTP 请求, 并执行相应的控制器方法, 而无需路由定义。自动路由默认被禁用。

警告: 为了防止配置错误和编码错误, 我们建议你不要使用自动路由 (传统)。很容易创建漏洞应用, 其中控制器过滤器或 CSRF 保护被绕过。

重要: 自动路由 (传统) 可以使用 **任何** HTTP 方法将 HTTP 请求路由到控制器方法。

重要: 自 v4.5.0 版本起，如果自动路由 (传统) 找不到控制器，它会在控制器过滤器执行之前抛出 `PageNotFoundException` 异常。

考虑这个 URI:

```
example.com/index.php/helloworld/
```

在上面的例子中,CodeIgniter 会尝试查找一个名为 **Helloworld.php** 的控制器并加载它。

备注: 当控制器的短名称与 URI 的第一段匹配时, 它会被加载。

让我们试一试: Hello World!(传统)

让我们创建一个简单的控制器, 以便你看到它的实际效果。使用文本编辑器, 创建一个名为 **Helloworld.php** 的文件, 并将以下代码放入其中。你会注意到 `Helloworld` 控制器正在扩展 `BaseController`。你也可以扩展 `CodeIgniter\Controller`, 如果你不需要 BaseController 的功能的话。

BaseController 为加载组件和执行所有控制器需要的函数提供了方便的位置。你可以在任何新控制器中扩展此类。

出于安全考虑, 请确保将任何新实用程序方法声明为 `protected` 或 `private`:

```
<?php

namespace App\Controllers;

class Helloworld extends BaseController
{
    public function index()
    {
        return 'Hello World!';
    }
}
```

然后将该文件保存到你的 **app/Controllers** 目录中。

重要: 该文件必须命名为 **Helloworld.php**,H 字母大写。当你使用自动路由时, 控制器类名称必须以大写字母开头, 并且只有第一个字符可以大写。

现在使用类似以下的 URL 访问你的站点:

```
example.com/index.php/helloworld
```

如果你正确执行了, 应该会看到:

```
Hello World!
```

有效的写法:

```
<?php

namespace App\Controllers;

class Helloworld extends BaseController
{
    // ...
```

(续下页)

(接上页)

```
}
```

无效的写法:

```
<?php

namespace App\Controllers;

class helloworld extends BaseController
{
    // ...
}
```

无效的写法:

```
<?php

namespace App\Controllers;

class HelloWorld extends BaseController
{
    // ...
}
```

此外, 始终确保你的控制器扩展父控制器类, 以便它可以继承其所有方法。

备注: 如果没有与定义的路由匹配, 系统将尝试通过匹配每个段与 **app/Controllers** 中的目录/文件来匹配 URI 与控制器。这就是为什么你的目录/文件必须以大写字母开头, 其余必须是小写字母。

如果你想要另一种命名约定, 你需要使用定义路由 手动定义它。这里有一个基于 PSR-4 自动加载的例子:

```
<?php

/*
 * Folder and file structure:
 * \<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>
```

(续下页)

(接上页)

```
 */

$routes->get('helloworld', '\App\Controllers\HelloWorld::index');
```

方法 (传统)

在上面的示例中, 方法名称是 `index()`。`index()` 方法如果 URI 的 **第二段**为空, 总是被默认加载。另一种显示 "Hello World" 消息的方法是:

```
example.com/index.php/helloworld/index/
```

URI 的第二段决定了控制器中调用哪个方法。

让我们试一试。在你的控制器中添加一个新方法:

```
<?php

namespace App\Controllers;

class Helloworld extends BaseController
{
    public function index()
    {
        return 'Hello World!';
    }

    public function comment()
    {
        return 'I am not flat!';
    }
}
```

现在使用下面的 URL 来查看 comment 方法的效果:

```
example.com/index.php/helloworld/comment/
```

你应该可以看到新的消息。

将 URI 段传递给你的方法 (传统)

如果 URI 包含两个以上段, 则会作为参数传递给你的方法。

例如, 假设你有这样的一个 URI:

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

你的方法将获取传入的 URI 的第 3 和第 4 段 (`'sandals'` 和 `'123'`):

```
<?php

namespace App\Controllers;

class Products extends BaseController
{
    public function shoes($sandals, $id)
    {
        return $sandals . $id;
    }
}
```

默认控制器 (传统)

默认控制器是一个特殊的控制器, 在 URI 以目录名结束或者 URI 不存在的情况下使用, 这通常发生在仅请求站点根 URL 的情况。

定义默认控制器 (传统)

让我们以 `Helloworld` 控制器为例。

要指定默认控制器, 打开配置文件 **app/Config/Routing.php**, 设置如下属性:

```
public string $defaultController = 'Helloworld';
```

其中 `Helloworld` 是希望用作默认控制器的控制器类名称。

并注释掉 **app/Config/Routes.php** 文件中的这一行:

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

现在如果在不指定任何 URI 段的情况下浏览你的站点, 你将看到 "Hello World" 消息。

备注: `$routes->get('/', 'Home::index');` 这一行是优化, 在 "真实的" 应用中会使用。但是为了演示的目的, 我们不想使用这个功能。`$routes->get()` 在*URI 路由* 中有解释。

有关更多信息, 请参阅配置选项（传统版） 文档。

将控制器组织到子目录中 (传统)

如果你正在构建一个大型应用程序, 你可能需要分层组织控制器结构到子目录中。CodeIgniter 支持这种方式。

只需要在主目录 **app/Controllers** 下创建子目录, 并将控制器类放入其中即可。

重要: 目录名称必须以大写字母开头, 只有首字母可以大写。

使用此功能时,URI 的第一段必须指定目录。例如, 假设你有一个位于如下位置的控制器:

```
app/Controllers/Products/Shoes.php
```

要调用上述控制器, 你的 URI 将是这样:

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

备注: 在 **app/Controllers** 和 **public** 中不能有同名的目录。这是因为如果目录存在,web 服务器会进行查找, 路由不会转到 CodeIgniter。

每个子目录下可以包含一个默认控制器, 如果 URL 仅包含 **子目录**, 则会调用该默认控制器。只需在相应位置放置一个控制器, 使其与 **app/Config/Routing.php** 文件中指定的默认控制器名称匹配即可。

CodeIgniter 也支持通过定义路由 来映射 URI。

重映射方法调用

备注: 自动路由 (改进版) 有意不支持此功能。

如上所述,URI 的第二段通常决定控制器中调用哪个方法。CodeIgniter 允许你通过使用 `_remap()` 方法来覆盖此行为:

```
<?php

namespace App\Controllers;

class Products extends BaseController
{
    public function _remap()
    {
        // Some code here...
    }
}
```

重要: 如果你的控制器包含名为 `_remap()` 的方法, 无论 URI 包含什么, 它都将 **始终**被调用。它会覆盖 URI 决定调用哪个方法的正常行为, 允许你定义自己的方法路由规则。

被覆盖的方法调用 (通常是 URI 的第二段) 将作为参数传递给 `_remap()` 方法:

```
<?php

namespace App\Controllers;

class Products extends BaseController
{
    public function _remap($method)
    {
        if ($method === 'some_method') {
            return $this->{$method}();
        }
```

(续下页)

(接上页)

```
        return $this->default_method();
    }
}
```

在方法名之后任何多余的段也会作为参数传递给 `_remap()`。这些参数可以传递给该方法, 以模拟 CodeIgniter 的默认行为。

例子:

```
<?php

namespace App\Controllers;

class Products extends BaseController
{
    public function _remap($method, ...$params)
    {
        $method = 'process_' . $method;

        if (method_exists($this, $method)) {
            return $this->{$method}(...$params);
        }

        throw \CodeIgniter\Exceptions\
↪PageNotFoundException::forPageNotFound();
    }
}
```

扩展控制器

如果你想扩展控制器, 请查看扩展控制器。

就这样了!

这基本上就是关于控制器需要了解的所有内容。

5.1.3 控制器过滤器

- 创建过滤器
  - 前置过滤器
    - 替换请求对象
    - 终止后续过滤器执行
    - 返回响应对象
  - 后置过滤器
- 配置过滤器
  - *app/Config/Filters.php*
    - *$aliases*
    - *$required*
    - *$globals*
    - *$methods*
    - *$filters*
  - 过滤器执行顺序
- 验证过滤器配置
  - *filter:check*
- 内置过滤器
  - *ForceHTTPS*
  - *PerformanceMetrics*
  - *InvalidChars*
  - *SecureHeaders*

控制器过滤器允许你在控制器执行前后执行特定操作。与事件 不同，你可以选择将过滤器应用于特定的 URI 或路由。前置过滤器可以修改请求（Request），而后置过滤器可以操作甚至修改响应（Response），这为开发者提供了极大的灵活性和控制力。

以下是使用过滤器实现的常见场景示例：

- 对传入请求实施 CSRF 保护
- 根据用户角色限制网站区域访问
- 对特定端点进行速率限制
- 显示「网站维护中」页面
- 执行自动内容协商
- 以及其他更多应用场景…

创建过滤器

过滤器是实现了 `CodeIgniter\Filters\FilterInterface` 接口的简单类。它们包含两个方法：`before()` 和 `after()`，分别用于在控制器执行前后运行代码。你的类必须包含这两个方法，但如果不需要具体实现，可以保持方法体为空。一个基础的过滤器类结构如下：

```
<?php

namespace App\Filters;

use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;

class MyFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments =␣
↪null)
    {
        // Do something here
    }

    public function after(RequestInterface $request,␣
```

(续下页)

(接上页)

```
↪ResponseInterface $response, $arguments = null)
    {
        // Do something here
    }
}
```

前置过滤器

替换请求对象

在任何前置过滤器中，你可以返回修改后的 `$request` 对象，该对象将替换当前请求，确保控制器执行时使用的是修改后的请求实例。

终止后续过滤器执行

当存在多个过滤器时，你可能需要在某个过滤器之后终止后续过滤器的执行。只需返回**任意非空值**即可实现。若前置过滤器返回空值，控制器操作及后续过滤器仍会继续执行。

需要注意的是，返回 `Request` 实例属于例外情况。在前置过滤器中返回该实例不会终止执行流程，仅会替换当前的 `$request` 对象。

返回响应对象

由于前置过滤器在控制器执行前运行，有时你可能希望阻止控制器的后续操作。这通常用于执行重定向，如下例所示：

```
<?php

namespace App\Filters;

use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;

class MyFilter implements FilterInterface
```

(续下页)

(接上页)

```
{
    public function before(RequestInterface $request, $arguments =␣
↪null)
    {
        $auth = service('auth');

        if (! $auth->isLoggedIn()) {
            return redirect()->to(site_url('login'));
        }
    }
}
```

如果返回 `Response` 实例，该响应将直接发送至客户端并终止脚本执行。这在实现 API 速率限制时非常有用，具体示例可参考*Throttler* 库文档。

后置过滤器

后置过滤器与前置过滤器类似，但只能返回 `$response` 对象且无法终止脚本执行。这允许你修改最终输出内容或对其进行后续处理，例如确保安全头正确设置、缓存最终输出或实施敏感词过滤等。

配置过滤器

有两种方式配置过滤器的执行时机：一种在 **app/Config/Filters.php** 中配置，另一种在 **app/Config/Routes.php** 中配置。若需为定义的路由指定过滤器，请使用 **app/Config/Routes.php** 并参考*URI 路由* 章节。

备注： 最安全的过滤器应用方式是禁用自动路由 并为路由设置过滤器。

---

app/Config/Filters.php

app/Config/Filters.php 文件包含四个属性，用于精确控制过滤器的执行时机。

> **警告:** 建议在过滤器设置的 URI 末尾始终添加 * 通配符。因为控制器方法可能通过你未预料到的其他 URL 访问。例如，当启用传统自动路由 时，若存在 `Blog::index()` 方法，可通过 `blog`、`blog/index` 和 `blog/index/1` 等路径访问。

$aliases

`$aliases` 数组用于将一个简短的别名与一个或多个完整的过滤器类名关联：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public array $aliases = [
        'csrf' => \CodeIgniter\Filters\CSRF::class,
    ];

    // ...
}
```

别名是强制性的，若尝试直接使用完整类名，系统将抛出错误。

这种定义方式便于后续更换过滤器类（例如切换认证系统时），只需修改别名对应的类即可完成迁移。

你还可以将多个过滤器组合到一个别名下，简化复杂过滤器集的配置：

```
<?php

namespace Config;
```

(续下页)

(接上页)

```
use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public array $aliases = [
        'api-prep' => [
            \App\Filters\Negotiate::class,
            \App\Filters\ApiAuth::class,
        ],
    ];

    // ...
}
```

根据需求定义任意数量的别名。

$required

在 4.5.0 版本加入.

第二部分用于定义 **必选过滤器**。这些特殊过滤器会应用于框架处理的每个请求，且在其他类型过滤器之前和之后执行。

备注： 必选过滤器始终执行。但如果路由不存在，仅会执行前置过滤器。

需谨慎设置此处过滤器的数量，过多会影响请求处理性能。默认提供的必选过滤器支撑框架核心功能，移除可能导致相关功能失效。详见内置过滤器。

通过将别名添加至 `before` 或 `after` 数组来指定必选过滤器：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
```

(续下页)

(接上页)

```
class Filters extends BaseConfig
{
    // ...
    public array $required = [
        'before' => [
            'forcehttps', // Force Global Secure Requests
            'pagecache',  // Web Page Caching
        ],
        'after' => [
            'pagecache',   // Web Page Caching
            'performance', // Performance Metrics
            'toolbar',     // Debug Toolbar
        ],
    ];

    // ...
}
```

$globals

第三部分用于定义应用于所有有效请求的全局过滤器。需注意过滤器数量对性能的影响。

通过将别名添加至 `before` 或 `after` 数组来指定全局过滤器:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public array $globals = [
        'before' => [
```

(续下页)

(接上页)

```
            'csrf',
        ],
        'after' => [],
    ];

    // ...
}
```

排除特定 URI

有时需要对绝大多数请求应用过滤器，但排除少数例外。例如在 CSRF 保护中排除第三方网站可访问的特定 URI。

在别名旁添加包含 `except` 键和 URI 路径（相对于 BaseURL）的数组实现：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public array $globals = [
        'before' => [
            'csrf' => ['except' => 'api/*'],
        ],
        'after' => [],
    ];

    // ...
}
```

> **警告：** 在 v4.4.7 之前，由于漏洞，过滤器处理的 URI 路径未进行 URL 解码。即路由中的 URI 路径与过滤器配置的路径可能不一致。详见控制器过滤器中的路径。

在过滤器设置中，可使用正则表达式或通配符（*）匹配 URI 路径。上例中，所有以 `api/` 开头的路径将豁免 CSRF 保护。

如需指定多个路径，可使用路径模式数组：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public array $globals = [
        'before' => [
            'csrf' => ['except' => ['foo/*', 'bar/*']],
        ],
        'after' => [],
    ];

    // ...
}
```

$methods

> **警告：** 若使用 `$methods` 过滤器，应禁用传统自动路由，因为传统自动路由 允许通过任意 HTTP 方法访问控制器，可能导致过滤器被绕过。

可为特定 HTTP 方法（如 POST、GET、PUT 等）的所有请求应用过滤器。其值为要运行的过滤器数组：

```php
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public array $methods = [
        'POST' => ['invalidchars', 'csrf'],
        'GET'  => ['csrf'],
    ];

    // ...
}
```

备注： 与 `$globals` 或 `$filters` 不同，这些过滤器仅作为前置过滤器运行。

除标准 HTTP 方法外，还支持特殊值 `CLI`，该值应用于所有命令行请求。

备注： v4.5.0 之前版本需使用 **小写**指定 HTTP 方法名称。

$filters

该属性是过滤器别名数组。每个别名可指定 `before` 和 `after` 数组，包含过滤器应应用的 URI 路径模式（相对于 BaseURL）：

```php
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
```

(续下页)

(接上页)

```
class Filters extends BaseConfig
{
    // ...

    public array $filters = [
        'foo' => ['before' => ['admin/*'], 'after' => ['users/*']],
        'bar' => ['before' => ['api/*', 'admin/*']],
    ];

    // ...
}
```

> **警告:** 在 v4.4.7 之前，由于漏洞，过滤器处理的 URI 路径未进行 URL 解码。详见控制器过滤器中的路径。

过滤器参数

在 4.4.0 版本加入.

配置 `$filters` 时，可向过滤器传递额外参数:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public $filters = [
        'group:admin,superadmin'  => ['before' => ['admin/*']],
        'permission:users.manage' => ['before' => ['admin/users/*
↪']],
    ];
```

(续下页)

(接上页)

```
    // ...
}
```

本例中，当 URI 匹配 `admin/*` 时，数组 `['admin', 'superadmin']` 将作为 `$arguments` 传递给 `group` 过滤器的 `before()` 方法；当匹配 `admin/users/*` 时，数组 `['users.manage']` 将传递给 `permission` 过滤器的 `before()` 方法。

备注: v4.6.0 之前版本，同一过滤器无法使用不同参数多次运行。

过滤器执行顺序

重要: 自 v4.5.0 起，过滤器执行顺序已变更。如需保持旧版顺序，请将 `Config\Feature::$oldFilterOrder` 设为 `true`。

过滤器按以下顺序执行:

- **前置过滤器**: 必选 → 全局 → 方法 → 过滤器 → 路由
- **后置过滤器**: 路由 → 过滤器 → 全局 → 必选

备注: 必选过滤器自 v4.5.0 起可用。

备注: v4.5.0 之前版本，路由过滤器（在 **app/Config/Routes.php** 中定义）优先于 **app/Config/Filters.php** 中的过滤器执行，且后置过滤器的执行顺序未反转。详见升级指南。

验证过滤器配置

CodeIgniter 提供以下命令 用于检查路由的过滤器配置。

filter:check

在 4.3.0 版本加入.

示例：检查 / 路由的 GET 方法过滤器配置：

```
php spark filter:check get /
```

输出结果如下：

```
+--------+-------+----------------------+---------------------------
↪----+
| Method | Route | Before Filters       | After Filters             ␣
↪    |
+--------+-------+----------------------+---------------------------
↪----+
| GET    | /     | forcehttps pagecache | pagecache performance␣
↪toolbar |
+--------+-------+----------------------+---------------------------
↪----+

Before Filter Classes:
CodeIgniter\Filters\ForceHTTPS → CodeIgniter\Filters\PageCache
After Filter Classes:
CodeIgniter\Filters\PageCache → CodeIgniter\Filters\
↪PerformanceMetrics → CodeIgniter\Filters\DebugToolbar
```

备注： 自 v4.6.0 起，输出表格中会显示过滤器参数，并展示实际使用的过滤器类名。

也可通过 `spark routes` 命令查看路由和过滤器，但使用正则表达式定义路由时可能显示不准确。详见*URI 路由*。

内置过滤器

CodeIgniter4 内置以下过滤器:

- `cors` => *跨域资源共享 (CORS)*
- `csrf` => *CSRF*
- `toolbar` => *DebugToolbar*
- `honeypot` => *Honeypot*
- `invalidchars` => *InvalidChars*
- `secureheaders` => *SecureHeaders*
- `forcehttps` => *ForceHTTPS*
- `pagecache` => *页面缓存*
- `performance` => *PerformanceMetrics*

备注: 过滤器按配置文件中的顺序执行。但若启用，`DebugToolbar` 始终最后执行，以便捕获其他过滤器的所有操作。

ForceHTTPS

在 4.5.0 版本加入.

该过滤器提供「强制全局安全请求」功能。若设置 `Config\App:$forceGlobalSecureRequests` 为 true，所有请求必须通过 HTTPS 连接。若请求不安全，用户将被重定向至安全页面，并设置 HSTS 头。

PerformanceMetrics

在 4.5.0 版本加入.

该过滤器提供性能指标伪变量。若需在视图中显示从框架启动到输出生成的总耗时，使用:

```
{elapsed_time}
```

显示内存使用量则使用:

```
{memory_usage}
```

若不需要此功能，从 `$required['after']` 中移除 `'performance'`。

InvalidChars

该过滤器禁止用户输入数据（`$_GET`、`$_POST`、`$_COOKIE`、`php://input`）包含以下字符:

- 无效 UTF-8 字符
- 除换行符和制表符外的控制字符

SecureHeaders

该过滤器添加增强应用安全性的 HTTP 响应头。如需自定义头信息，可继承 `CodeIgniter\Filters\SecureHeaders` 并覆盖 `$headers` 属性，然后在 **app/Config/Filters.php** 中修改 `$aliases`:

```php
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public array $aliases = [
        // ...
        'secureheaders' => \App\Filters\SecureHeaders::class,
    ];

    // ...
}
```

有关安全头的详细信息，请参考 OWASP 安全头项目。

5.1.4 自动路由（改进版）

在 4.2.0 版本加入.

- 什么是自动路由（改进版）？
- 与自动路由（旧版）的差异
- 启用自动路由（改进版）
- *URI* 分段
- 实践演练：*Hello World!*
  - 创建控制器
  - 检查路由
  - 访问页面
- 控制器命名示例
- 控制器方法
  - 方法可见性
  - 默认方法
  - 常规方法
- 向方法传递 *URI* 分段
- 默认控制器
- 默认方法回退
  - 回退至默认控制器
- 控制器子目录组织
- 控制器/方法与 *URI* 对应示例
- 应用过滤器
- 配置选项
  - 默认控制器
    - 站点根 *URI*

    * 目录 URI
  - 默认方法
  - URI 转驼峰命名
    * 禁用 URI 转驼峰命名
- 模块路由

什么是自动路由（改进版）?

默认情况下，所有路由都必须在配置文件中通过*defined* 方式定义。

但通过 **自动路由（改进版）**，你可以根据约定定义控制器名称及其方法名称，系统将自动进行路由。换句话说，无需手动定义路由。

启用自动路由（改进版）后，当未找到与 URI 匹配的已定义路由时，系统将尝试根据控制器和方法进行匹配。

重要: 出于安全考虑，若控制器已在定义的路由中使用，则自动路由（改进版）不会路由到该控制器。

备注: 自动路由（改进版）默认禁用。要启用请参阅启用自动路由（改进版）。

与自动路由（旧版）的差异

自动路由（传统版）是 CodeIgniter 3 的路由系统。如果你不熟悉该功能，可直接阅读下一章节。

如果你已了解旧版路由，以下是 **自动路由（改进版）** 的主要变化：

- **控制器方法需要添加 HTTP 动词前缀如 `getIndex()`、`postCreate()`**
  - 由于开发者始终明确 HTTP 方法，使用非预期的 HTTP 方法请求将不会执行控制器
- **URI 中必须省略默认控制器（默认为 `Home`）和默认方法（默认为 `index`）**
  - 这保证了控制器方法与 URI 的一一对应关系

  - 例如默认情况下可访问 /，但 /home 和 /home/index 将返回 404 错误
- **检查方法参数数量**
  - 若 URI 中的参数数量超过方法参数数量，将返回 404 错误
- **不支持 _remap() 方法**
  - 这保证了控制器方法与 URI 的一一对应关系
  - 但提供了默认方法回退 功能作为替代
- **无法访问已定义路由中的控制器**
  - 通过 **自动路由**访问的控制器与通过 **定义路由**访问的控制器完全分离

启用自动路由（改进版）

要使用该功能，需在 **app/Config/Routing.php** 中将 $autoRoute 选项设为 true:

```
public bool $autoRoute = true;
```

同时需在 **app/Config/Feature.php** 中将 $autoRoutesImproved 属性设为 true:

```
public bool $autoRoutesImproved = true;
```

重要: 使用自动路由（改进版）时，必须移除 **app/Config/Routes.php** 中的 $routes->get('/', 'Home::index'); 行。因为定义路由优先级高于自动路由，且出于安全考虑，自动路由（改进版）会拒绝访问已定义路由中的控制器。

URI 分段

遵循 MVC 模式，URL 中的分段通常表示:

```
http://example.com/{class}/{method}/{param1}
```

1. 第一分段表示要调用的控制器 **类**
2. 第二分段表示要调用的类 **方法**
3. 第三及后续分段表示传递给控制器方法的 **参数**

示例 URI:

```
http://example.com/hello-world/hello/1
```

当使用 **GET** 方法发送 HTTP 请求时，自动路由（改进版）会尝试查找名为 `App\Controllers\HelloWorld` 的控制器，并执行 `getHello()` 方法，同时传递 `'1'` 作为第一个参数。

备注： 通过自动路由（改进版）执行的控制器方法需要添加 HTTP 动词（`get`、`post`、`put` 等）前缀，如 `getIndex()`、`postCreate()`。

备注： 当控制器的短名称与 URI 的第一分段匹配时，该控制器将被加载。

实践演练：Hello World!

让我们创建一个简单控制器来演示该功能。

创建控制器

在 **app/Controllers** 目录中创建名为 **HelloWorld.php** 的文件，并添加以下代码：

```
<?php

namespace App\Controllers;

class HelloWorld extends BaseController
{
    public function getIndex()
    {
        return 'Hello World!';
    }
}
```

重要： 文件名必须为 **HelloWorld.php**。使用自动路由（改进版）时，控制器类名必须采

用大驼峰式命名法。

注意 `HelloWorld` 控制器继承自 `BaseController`。若不需要基控制器功能，也可直接继承 `CodeIgniter\Controller`。

BaseController 提供了方便的组件加载方式，并包含所有控制器共用的功能。你可以在任何新控制器中继承此类。

重要: 通过自动路由（改进版）执行的控制器方法必须添加 HTTP 动词前缀，如 `getIndex()`、`postCreate()`。

检查路由

可通过 `spark routes` 命令查看路由信息：

```
php spark routes
```

成功操作后将显示：

```
+-----------+-------------+------+-----------------------------------
↪-----+----------------+---------------+
| Method    | Route       | Name | Handler                          ␣
↪     | Before Filters | After Filters |
+-----------+-------------+------+-----------------------------------
↪-----+----------------+---------------+
| GET(auto) | hello-world |      | \App\Controllers\
↪HelloWorld::getIndex |                |               |
+-----------+-------------+------+-----------------------------------
↪-----+----------------+---------------+
```

详情请参阅*spark 路由*。

访问页面

现在通过以下 URL 访问你的站点:

```
http://example.com/hello-world
```

系统会自动将 URI 中的短横线（-）转换为控制器和方法的大驼峰式命名。

例如 URI `sub-dir/hello-controller/some-method` 将执行 `SubDir\HelloController::getSomeMethod()` 方法。

成功操作后将显示:

```
Hello World!
```

控制器命名示例

以下为有效控制器名称，因为 `App\Controllers\HelloWorld` 采用大驼峰式命名:

```
<?php

namespace App\Controllers;

class HelloWorld extends BaseController
{
    // ...
}
```

以下为无效命名，因为首字母（h）未大写:

```
<?php

namespace App\Controllers;

class helloworld extends BaseController
{
    // ...
}
```

以下同样无效，因为首字母（h）未大写:

```
<?php

namespace App\Controllers;

class helloWorld extends BaseController
{
    // ...
}
```

控制器方法

方法可见性

通过 HTTP 请求执行的方法必须声明为 `public`。

> **警告:** 出于安全考虑，请将所有工具方法声明为 `protected` 或 `private`。

默认方法

上述示例中的 `getIndex()` 方法称为 **默认方法**，当 URI 的 **第二分段**为空时将被调用。

常规方法

URI 的第二分段决定调用控制器中的哪个方法。

添加新方法进行测试:

```
<?php

namespace App\Controllers;

class HelloWorld extends BaseController
{
    public function getIndex()
    {
```

(续下页)

(接上页)

```
        return 'Hello World!';
    }

    public function getComment()
    {
        return 'I am not flat!';
    }
}
```

现在访问以下 URL 查看 `getComment()` 方法:

```
http://example.com/hello-world/comment/
```

你将看到新消息。

向方法传递 URI 分段

若 URI 包含超过两个分段，后续分段将作为参数传递给方法。

示例 URI:

```
http://example.com/products/shoes/sandals/123
```

该方法将接收第 3、4 分段（`'sandals'` 和 `'123'`）:

```
<?php

namespace App\Controllers;

class Products extends BaseController
{
    public function getShoes($type, $id)
    {
        return $type . $id;
    }
}
```

备注: 若 URI 参数数量超过方法参数数量，自动路由（改进版）不会执行该方法，并返

回 404 错误。

默认控制器

默认控制器是在 URI 以目录名结尾或未指定 URI（如访问站点根 URL）时使用的特殊控制器。

默认控制器为 `Home`。

备注: 默认控制器中只应定义默认方法（GET 请求对应 `getIndex()`）。若定义其他公共方法，这些方法将无法执行。

更多信息请参考配置选项。

默认方法回退

在 4.4.0 版本加入.

当 URI 方法名分段对应的控制器方法不存在，但默认方法已定义时，剩余 URI 分段将传递给默认方法执行。

```
<?php

namespace App\Controllers;

class Product extends BaseController
{
    public function getIndex($id = null, $action = '')
    {
        // ...
    }
}
```

访问以下 URL:

```
http://example.com/product/15/edit
```

该方法将接收第 2、3 分段（`'15'` 和 `'edit'`）:

重要: 若 URI 参数数量超过方法参数数量，自动路由（改进版）不会执行该方法，并返回 404 错误。

回退至默认控制器

当 URI 控制器名分段对应的控制器不存在，但目录中存在默认控制器（默认为 `Home`）时，剩余 URI 分段将传递给默认控制器的默认方法。

示例在 **app/Controllers/News** 目录中创建默认控制器 `Home`：

```
<?php

namespace App\Controllers\News;

use App\Controllers\BaseController;

class Home extends BaseController
{
    public function getIndex($id = null)
    {
        // ...
    }
}
```

访问以下 URL:

```
http://example.com/news/101
```

系统将找到 `News\Home` 控制器及其默认 `getIndex()` 方法。默认方法将接收第二 URI 分段（`'101'`）：

备注: 若存在 `App\Controllers\News` 控制器，则优先使用。URI 分段将按顺序搜索，使用首个找到的控制器。

备注: 若 URI 参数数量超过方法参数数量，自动路由（改进版）不会执行该方法，并返

回 404 错误。

控制器子目录组织

在大型应用中，你可能希望将控制器组织到子目录中。CodeIgniter 支持此功能。

只需在 **app/Controllers** 下创建子目录（目录名必须以大写字母开头并采用大驼峰式命名），并将控制器类存放其中。

示例控制器路径:

```
app/Controllers/Products/Shoes.php
```

调用该控制器的 URI 形如:

```
http://example.com/products/shoes/show/123
```

备注: **app/Controllers** 和 **public** 目录不能存在同名目录，否则 Web 服务器将直接访问该目录而不会路由到 CodeIgniter。

每个子目录可包含默认控制器，当 URL 仅包含子目录时将调用该控制器。需确保默认控制器名称与 **app/Config/Routing.php** 中配置一致。

控制器/方法与 URI 对应示例

在默认配置下，GET 请求的控制器/方法与 URI 对应关系如下:

控制器/方法	URI	说明
`Home::getIndex()`	/	默认控制器和默认方法
`Blog::getIndex()`	/blog	默认方法
`UserProfile::getIndex()`	/user-profile	默认方法
`Blog::getTags()`	/blog/tags	
`Blog::getNews($id)`	/blog/news/123	
`Blog\Home::getIndex()`	/blog	子目录 `Blog` 中的默认控制器和默认方法。若存在 `Blog` 控制器则优先使用
`Blog\Tags::getIndex()`	/blog/tags	子目录 `Blog` 中的默认方法。若存在 `Blog` 控制器则优先使用
`Blog\News::getIndex($id)`	/blog/news/123	子目录 `Blog` 中的默认方法回退。若存在 `Blog` 控制器则优先使用

应用过滤器

应用控制器过滤器可在控制器方法执行前后添加处理逻辑，适用于身份验证或 API 日志记录等场景。

使用自动路由时，需在 **app/Config/Filters.php** 中设置要应用的过滤器。详情请参阅控制器过滤器 文档。

配置选项

以下选项位于 **app/Config/Routing.php** 文件。

默认控制器

站点根 URI

当用户访问站点根目录（如 **http://example.com**）时，除非存在显式路由，否则将使用 `$defaultController` 属性指定的控制器。

默认值为 `Home`，对应 **app/Controllers/Home.php** 控制器:

```
public string $defaultController = 'Home';
```

目录 URI

当未找到匹配路由且 URI 指向控制器目录时，也使用默认控制器。例如用户访问 **http://example.com/admin**，若存在 **app/Controllers/Admin/Home.php** 控制器，则使用该控制器。

重要: 无法通过控制器名称的 URI 访问默认控制器。当默认控制器为 `Home` 时，可访问 **http://example.com/**，但访问 **http://example.com/home** 将返回 404 错误。

默认方法

此设置与默认控制器类似，用于确定当找到匹配 URI 的控制器但未指定方法分段时使用的默认方法。默认值为 `index`。

示例配置:

```
public string $defaultMethod = 'listAll';
```

当用户访问 **example.com/products** 且存在 `Products` 控制器时，将执行 `Products::getListAll()` 方法。

重要: 无法通过默认方法名称的 URI 访问控制器。上述示例中可访问 **example.com/products**，但访问 **example.com/products/listall** 将返回 404 错误。

URI 转驼峰命名

在 4.5.0 版本加入.

备注: 自 v4.6.0 起，`$translateUriToCamelCase` 选项默认启用

自 v4.5.0 起实现的 `$translateUriToCamelCase` 选项完美适配当前 CodeIgniter 的编码规范。

此选项可自动将 URI 中的短横线（-）转换为控制器和方法的大驼峰式命名。

示例 URI `sub-dir/hello-controller/some-method` 将执行 `SubDir\HelloController::getSomeMethod()` 方法。

备注: 启用此选项时，`$translateURIDashes` 选项将被忽略。

禁用 URI 转驼峰命名

备注: 禁用“URI 转驼峰命名”的选项仅用于向后兼容，不建议禁用。

在 **app/Config/Routing.php** 中将 `$translateUriToCamelCase` 设为 `false` 即可禁用:

```
public bool $translateUriToCamelCase = false;
```

模块路由

在 4.4.0 版本加入.

即使使用代码模块 并将控制器放置在不同命名空间，仍可使用自动路由。

要路由到模块，需在 **app/Config/Routing.php** 中设置 `$moduleRoutes` 属性:

```
public array $moduleRoutes = [
    'blog' => 'Acme\Blog\Controllers',
];
```

键名为模块的第一个 URI 分段，值为控制器命名空间。上述配置中，**http://localhost:8080/blog/foo/bar** 将路由到 `Acme\Blog\Controllers\Foo::getBar()`。

备注: 若定义 `$moduleRoutes`，模块路由将优先处理。上述示例中即使存在 `App\Controllers\Blog` 控制器，**http://localhost:8080/blog** 仍会路由到默认控制器 `Acme\Blog\Controllers\Home`。

5.1.5 HTTP 消息

Message 类为 HTTP 消息中请求和响应共有的部分提供了一个接口, 包括消息体、协议版本、用于处理头的实用程序以及处理内容协商的方法。

这个类是请求类 和响应类 的父类，不能直接使用。

类参考

class CodeIgniter\HTTP**Message**

getBody()

返回
当前消息体

返回类型
mixed

如果已设置, 返回当前消息体。如果不存在正文, 则返回 null:

```
<?php

echo $message->getBody();
```

setBody(*$data*)

参数

- **$data** (mixed) –消息的正文。

返回
Message|Response 实例以允许链式调用方法。

返回类型
CodeIgniter\HTTP\Message|CodeIgniter\HTTP\Response

设置当前请求的消息体。

appendBody(*$data*)

参数

- **$data** (mixed) –消息的正文。

返回

Message|Response 实例以允许链式调用方法。

返回类型

`CodeIgniter\HTTP\Message|CodeIgniter\HTTP\Response`

向当前请求的消息体附加数据。

populateHeaders()

返回

void

扫描并解析 SERVER 数据中的标头，并存储以供以后访问。这由*IncomingRequest 类* 使用，以使当前请求的标头可用。

标头是以 `HTTP_` 开头的任何服务器数据，如 `HTTP_HOST`。每个消息都从其标准的大写和下划线格式转换为 ucwords 和破折号格式。从字符串中删除前导 `HTTP_`。因此 `HTTP_ACCEPT_LANGUAGE` 变成 `Accept-Language`。

headers()

返回

找到的所有标头的数组。

返回类型

`array`

返回找到或先前设置的所有标头的数组。

header(*$name*)

参数

- **$name** (`string`) –你要检索其值的标头的名称。

返回

返回单个标头对象。如果存在多个同名标头，则返回标头对象数组。

返回类型

`CodeIgniter\HTTP\Header|array`

允许你检索单个消息标头的当前值。`$name` 是不区分大小写的标头名称。尽管内部会按上述方式转换标头，但你可以使用任何情况访问标头:

```
<?php

// These are all the same:
$message->header('HOST');
$message->header('Host');
$message->header('host');
```

如果标头有多个值，`getValue()` 将返回值数组。你可以使用 `getValueLine()` 方法将值作为字符串检索:

```
<?php

echo $message->header('Accept-Language');
/*
 * Outputs something like:
 * 'Accept-Language: en,en-US'
 */

echo $message->header('Accept-Language')->getValue();
/*
 * Outputs something like:
 * [
 *     'en',
 *     'en-US',
 * ]
 */

echo $message->header('Accept-Language')->getValueLine();
/*
 * Outputs something like:
 * en,en-US'
 */
```

你可以通过第二个参数传递过滤值来过滤标头:

```
<?php

$message->header('Document-URI', FILTER_SANITIZE_URL);
```

hasHeader(*$name*)

参数

- **$name** (string) –你要查看其是否存在的标头的名称。

返回

如果存在则返回 true, 否则返回 false。

返回类型

bool

getHeaderLine(*$name*)

参数

- **$name** (string) –要检索的标头的名称。

返回

表示标头值的字符串。

返回类型

string

以字符串形式返回标头的值。当标头有多个值时, 此方法使你可以轻松获取标头值的字符串表示形式。值被适当连接:

```
<?php

echo $message->getHeaderLine('Accept-Language');
/*
 * Outputs:
 * 'en,en-US'
 */
```

setHeader(*$name*, *$value*)

参数

- **$name** (string) –要为其设置值的标头的名称。
- **$value** (mixed) –要设置标头的值。

返回

当前的 MessagelResponse 实例

返回类型

`CodeIgniter\HTTP\Message|CodeIgniter\HTTP\Response`

设置单个标头的值。`$name` 是标头的不区分大小写的名称。如果集合中还不存在该标头, 则会创建它。`$value` 可以是字符串或字符串数组:

```
<?php

$message->setHeader('Host', 'codeigniter.com');
```

removeHeader(*$name*)

参数

- **$name** (`string`) –要删除的标头的名称。

返回

当前消息实例

返回类型

`CodeIgniter\HTTP\Message`

从消息中删除标头。`$name` 是标头的不区分大小写的名称:

```
<?php

$message->removeHeader('Host');
```

appendHeader(*$name*, *$value*)

参数

- **$name** (`string`) –要修改的标头的名称
- **$value** (`string`) –要添加到标头的值。

返回

当前消息实例

返回类型

`CodeIgniter\HTTP\Message`

向现有标头添加一个值。标头必须已经是一个值数组, 而不是单个字符串。如果它是一个字符串, 则会抛出 LogicException。

```
<?php

$message->appendHeader('Accept-Language', 'en-US; q=0.8');
```

prependHeader(*$name*, *$value*)

参数

- **$name** (string) –要修改的标头的名称
- **$value** (string) –要在标头前面添加的值。

返回

当前消息实例

返回类型

CodeIgniter\HTTP\Message

在现有标头前面添加一个值。标头必须已经是一个值数组, 而不是单个字符串。如果它是一个字符串, 则会抛出 LogicException。

```
<?php

$message->prependHeader('Accept-Language', 'en,');
```

addHeader(*$name*, *$value*)

在 4.5.0 版本加入.

参数

- **$name** (string) –要添加的头名称。
- **$value** (string) –头的值。

返回

当前消息实例

返回类型

CodeIgniter\HTTP\Message

添加具有相同名称的头（不是头的值）。仅在设置多个具有相同名称的头时使用。

```
<?php

$message->addHeader('Set-Cookie', 'logged_in=no; Path=/');
$message->addHeader('Set-Cookie', 'sessid=123456; Path=/');
```

getProtocolVersion()

返回

当前的 HTTP 协议版本

返回类型

`string`

返回消息的当前 HTTP 协议。如果未设置, 将返回 `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`

设置此消息使用的 HTTP 协议版本。有效值为 `1.0`, `1.1`, `2.0` 和 `3.0`:

```
<?php

$message->setProtocolVersion('1.1');
```

5.1.6 请求类

请求类是 HTTP 请求的面向对象表示。这旨在适用于传入请求, 例如来自浏览器对应用程序的请求, 以及传出请求, 例如应用程序对第三方应用程序的请求所用的请求。

这个类提供了它们都需要的常见功能, 但这两种情况都有自定义类来扩展请求类以添加特定功能。在实践中, 你需要使用这些类。

有关更多使用详细信息, 请参阅*IncomingRequest 类* 和*CURLRequest 类* 的文档。

类参考

class CodeIgniter\HTTP**Request**

getIPAddress()

返回

如果可以检测到, 则为用户的 IP 地址。如果 IP 地址不是有效的 IP 地址, 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, 该方法将返回 `0.0.0.0`:

```
<?php

echo $request->getIPAddress();
```

重要: 该方法会考虑 `Config\App::$proxyIPs` 设置, 并将 HTTP 头中报告的允许 IP 地址的客户端 IP 地址返回。

---

isValidIP(*$ip*[, *$which* = ''])

自 4.0.5 版本弃用: 请改用数据验证。

重要: 此方法已弃用。它将在未来版本中删除。

参数

- **$ip** (`string`) –IP 地址
- **$which** (`string`) –IP 协议 (`ipv4` 或 `ipv6`)

返回

如果地址有效则为 true, 如果无效则为 false

返回类型

`bool`

将 IP 地址作为输入, 并根据它是否有效返回 true 或 false(布尔值)。

备注: 上面的 `$request->getIPAddress()` 方法会自动验证 IP 地址。

```
<?php

if (! $request->isValidIP($ip)) {
    echo 'Not Valid';
} else {
    echo 'Valid';
}
```

可选的第二个字符串参数为 "ipv4" 或 "ipv6" 来指定 IP 格式。默认检查这两种格式。

getMethod()

返回

HTTP 请求方法

返回类型

`string`

返回 `$_SERVER['REQUEST_METHOD']`。

```
<?php

echo $request->getMethod(); // Outputs: POST
```

setMethod(*$method*)

自 4.0.5 版本弃用: 请改用 *CodeIgniter\HTTP\Request::withMethod()*。

参数

- **$method** (`string`) –设置请求方法。在伪造请求时使用。

返回

这个请求

返回类型

Request

withMethod(*$method*)

在 4.0.5 版本加入.

参数

- **$method** (string) –设置请求方法。

返回

新的请求实例

返回类型

Request

getServer([*$index = null*[, *$filter = null*[, *$flags = null*]]])

参数

- **$index** (mixed) –值名称
- **$filter** (int) –要应用的过滤类型。过滤器列表可在 PHP 手册 中找到。
- **$flags** (int|array) –要应用的标志。标志列表可在 PHP 手册 中找到。

返回

如果找到, 则返回 $_SERVER 项目的值, 如果没有找到, 则为 null

返回类型

mixed

此方法与*IncomingRequest 类* 中的 getPost()、getGet() 和 getCookie() 方法相同, 只是它获取服务器数据 ($_SERVER):

```
<?php

$request->getServer('some_data');
```

要返回多个 $_SERVER 值的数组, 请传递所有所需键的数组。

```
<?php

$request->getServer(['SERVER_PROTOCOL', 'REQUEST_URI']);
```

getEnv([*$index = null*[, *$filter = null*[, *$flags = null*]]])

自 4.4.4 版本弃用: 从一开始，这个方法就不起作用。请改用*env()*。

参数

- **$index** (mixed) –值名称
- **$filter** (int) –要应用的过滤类型。过滤器列表可在 PHP 手册 中找到。
- **$flags** (int|array) –要应用的标志。标志列表可在 PHP 手册 中找到。

返回

如果找到, 则返回 $_ENV 项目的值, 如果没有找到, 则为 null

返回类型

mixed

此方法与*IncomingRequest 类* 中的 getPost()、getGet() 和 getCookie() 方法相同, 只是它获取环境数据 ($_ENV):

```
<?php

$request->getEnv('some_data');
```

要返回多个 $_ENV 值的数组, 请传递所有所需键的数组。

```
<?php

$request->getEnv(['CI_ENVIRONMENT', 'S3_BUCKET']);
```

setGlobal(*$method*, *$value*)

参数

- **$method** (string) –方法名称
- **$value** (mixed) –要添加的数据

返回

这个请求

返回类型

Request

允许手动设置 PHP 全局变量的值, 如 `$_GET`、`$_POST` 等。

fetchGlobal(*$method*[, *$index = null*[, *$filter = null*[, *$flags = null*]]])

参数

- **$method** (string) –输入过滤常量
- **$index** (mixed) –值名称
- **$filter** (int) –要应用的过滤类型。过滤器列表可在 PHP 手册 中找到。
- **$flags** (int|array) –要应用的标志。标志列表可在 PHP 手册 中找到。

返回类型

mixed

从 cookie、get、post 等全局变量中获取一个或多个项目。可以通过传递过滤器在检索时可选地过滤输入。

5.1.7 IncomingRequest 类

IncomingRequest 类为来自客户端 (如浏览器) 的 HTTP 请求提供了面向对象的表示。它扩展并可以访问*Request* 和*Message* 类的所有方法, 以及下面列出的方法。

访问请求

如果当前类是 `CodeIgniter\Controller` 的后代, 则已经为你填充了请求类的一个实例, 可以将其作为类属性访问:

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class UserController extends Controller
{
    public function index()
    {
```

(续下页)

(接上页)

```
        if ($this->request->isAJAX()) {
            // ...
        }
    }
}
```

如果你不在控制器中, 但仍然需要访问应用程序的 Request 对象, 你可以通过*Services 类*获取它的一个副本:

```
<?php

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

不过, 如果类是控制器之外的任何其他类, 最好是将请求作为依赖项传递, 以便将其保存为类属性:

```
<?php

namespace App\Libraries;

use CodeIgniter\HTTP\RequestInterface;

class SomeClass
{
    protected $request;

    public function __construct(RequestInterface $request)
    {
        $this->request = $request;
    }
}

$someClass = new SomeClass(service('request'));
```

确定请求类型

请求可以是几种类型, 包括 AJAX 请求或来自命令行的请求。这可以通过 `isAJAX()` 和 `isCLI()` 方法检查:

```
<?php

// Check for AJAX request.
if ($request->isAJAX()) {
    // ...
}

// Check for CLI Request
if ($request->isCLI()) {
    // ...
}
```

备注: `isAJAX()` 方法取决于 `X-Requested-With` 头, 但在通过 JavaScript 发出的 XHR 请求中 (即 fetch), 该头默认不会发送。请参阅*AJAX 请求* 部分了解如何避免此问题。

is()

在 4.3.0 版本加入.

自 v4.3.0 起，你可以使用 `is()` 方法。它接受一个 HTTP 方法、`'ajax'` 或 `'json'`，并返回布尔值。

备注: HTTP 方法应该区分大小写，但参数是不区分大小写的。

```
<?php

// Checks HTTP methods. Returns boolean.
$request->is('get');
$request->is('post');
$request->is('put');
```

(续下页)

(接上页)

```
$request->is('delete');
$request->is('head');
$request->is('patch');
$request->is('options');

// Checks if it is an AJAX request. The same as `$request->
↪isAJAX()`.
$request->is('ajax');

// Checks if it is a JSON request.
$request->is('json');
```

getMethod()

你可以使用 `getMethod()` 方法检查此请求所代表的 HTTP 方法:

```
<?php

// Returns 'POST'
$method = $request->getMethod();
```

HTTP 方法是区分大小写的，按照惯例，标准化方法是用全大写的 US-ASCII 字母定义的。

备注: 在 v4.5.0 之前，默认情况下，该方法会返回小写字符串（即 `'get'`、`'post'` 等）。但这是一个 bug。

你可以通过用 `strtolower()` 包装调用来获取小写版本的字符串:

```
// 返回 'get'
$method = strtolower($request->getMethod());
```

你还可以使用 `isSecure()` 方法检查请求是否通过 HTTPS 连接发出:

```
<?php
```

(续下页)

(接上页)

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

检索输入

你可以通过 Request 对象检索来自 $_GET、$_POST、$_COOKIE、$_SERVER 和 $_ENV 的输入。数据不会自动过滤, 并以请求中传递的原始输入数据形式返回。

备注: 使用全局变量是不好的做法。基本上, 应该避免使用它, 建议使用 Request 对象的方法。

与直接访问它们 ($_POST['something']) 的主要优点是, 如果项不存在, 这些方法将返回 null, 并且你可以对数据进行过滤。这使你可以方便地使用数据, 而无需先测试一个项是否存在。换句话说, 通常你可能会做这样的事情:

```
<?php

$something = $_POST['foo'] ?? null;
```

使用 CodeIgniter 内置的方法, 你可以简单地这样做:

```
<?php

$something = $request->getPost('foo');
```

获取数据

getGet()

getGet() 方法将从 $_GET 中获取。

- $request->getGet()

getPost()

`getPost()` 方法将从 `$_POST` 中获取。

- `$request->getPost()`

getCookie()

`getCookie()` 方法将从 `$_COOKIE` 中获取。

- `$request->getCookie()`

getServer()

`getServer()` 方法将从 `$_SERVER` 中获取。

- `$request->getServer()`

getEnv()

自 4.4.4 版本弃用: 该方法从一开始就不起作用。请使用*`env()`* 替代。

`getEnv()` 方法将从 `$_ENV` 中获取。

- `$request->getEnv()`

getPostGet()

此外，还有一些用于从 `$_GET` 或 `$_POST` 中检索信息的实用方法，同时保持控制查找的顺序：

- `$request->getPostGet()` - 先检查 `$_POST`，然后是 `$_GET`

getGetPost()

- `$request->getGetPost()` - 先检查 `$_GET`，然后是 `$_POST`

getVar()

重要: 该方法仅存在于向后兼容中。不要在新项目中使用它。即使你已经在使用，我们也建议你使用另一个更合适的方法。

`getVar()` 方法将从 `$_REQUEST` 中获取，因此会返回任何来自 `$_GET`, `$POST`, 或者 `$_COOKIE` 的数据（取决于 php.ini request-order）。

> **警告:** 如果你只想验证 POST 数据，不要使用 `getVar()`。新值会覆盖旧值。如果 POST 值和 Cookie 有相同的名字，并且你在 request-order 中先设置 "P" 之后设置 "C"，则 POST 的值可能会被 Cookie 覆盖。

备注: 如果传入的请求的 `Content-Type` 头设置为 `application/json`，则 `getVar()` 方法会返回 JSON 数据，而不是 `$_REQUEST` 数据。

获取 JSON 数据

你可以使用 `getJSON()` 将 `php://input` 的内容作为 JSON 流获取。

备注: 这无法检查传入的数据是否为有效的 JSON。你只应在知道正在期望 JSON 时使用此方法。

```
<?php

$json = $request->getJSON();
```

默认情况下, 这将返回 JSON 数据中的任何对象作为对象。如果你想要将其转换为关联数组, 请在第一个参数中传递 `true`。

第二和第三个参数与 json_decode() PHP 函数的 `$depth` 和 `$flags` 参数对应。

从 JSON 获取特定数据

你可以通过向 `getJsonVar()` 传入变量名来从 JSON 流中获取特定的数据片段, 用于获取所需的数据, 或者可以使用 "点" 表示法深入到 JSON 中, 以获取不在根级别的数据。

```
<?php

/*
 * With a request body of:
 * {
 *     "foo": "bar",
 *     "fizz": {
 *         "buzz": "baz"
 *     }
 * }
 */

$data = $request->getJsonVar('foo');
// $data = "bar"

$data = $request->getJsonVar('fizz.buzz');
// $data = "baz"
```

如果你希望结果是关联数组而不是对象，你可以在第二个参数中传入 true：

```
<?php

// With the same request as above
$data = $request->getJsonVar('fizz');
// $data->buzz = "baz"

$data = $request->getJsonVar('fizz', true);
// $data = ["buzz" => "baz"]
```

备注: 有关“点”表示法的更多信息，请参阅 Array 辅助函数中的*dot_array_search()* 文档。

检索原始数据 (PUT、PATCH、DELETE)

最后，你可以使用 getRawInput() 将 php://input 的内容作为原始流获取:

```
<?php

$data = $request->getRawInput();
```

这将检索数据并将其转换为数组。像这样:

```
<?php

var_dump($request->getRawInput());

/*
 * Outputs:
 * [
 *     'Param1' => 'Value1',
 *     'Param2' => 'Value2',
 * ]
 */
```

你还可以使用 getRawInputVar()，从原始流中获取指定的变量并对其进行过滤。

```
<?php

// When the request body is 'foo=one&bar=two&baz[]=10&baz[]=20'
var_dump($request->getRawInputVar('bar'));
// Outputs: two

// foo=one&bar=two&baz[]=10&baz[]=20
var_dump($request->getRawInputVar(['foo', 'bar']));
/*
 * Outputs:
```

(续下页)

(接上页)

```
 * [
 *       'foo' => 'one',
 *       'bar' => 'two'
 * ]
 */

// foo=one&bar=two&baz[]=10&baz[]=20
var_dump($request->getRawInputVar('baz'));
/*
 * Outputs:
 * [
 *       '10',
 *       '20'
 * ]
 */

// foo=one&bar=two&baz[]=10&baz[]=20
var_dump($request->getRawInputVar('baz.0'));
// Outputs: 10
```

过滤输入数据

为了保持应用程序的安全, 你会想要过滤所有输入。你可以将要使用的过滤器类型作为这些方法的第二个参数传递。使用内置的 `filter_var()` 函数进行过滤。前往 PHP 手册获取 有效过滤器类型列表。

过滤 POST 变量的代码如下:

```
<?php

$email = $request->getPost('email', FILTER_SANITIZE_EMAIL);
```

上面提到的所有方法都支持作为第二个参数传递过滤器类型, `getJSON()` 和 `getRawInput()` 除外。

检索标头

你可以通过 `headers()` 方法访问与请求一起发送的任何标头, 它返回一个数组, 其中键是标头的名称, 值是 `CodeIgniter\HTTP\Header` 的一个实例:

```
<?php

var_dump($request->headers());

/*
 * Outputs:
 * [
 *     'Host'          => CodeIgniter\HTTP\Header,
 *     'Cache-Control' => CodeIgniter\HTTP\Header,
 *     'Accept'        => CodeIgniter\HTTP\Header,
 * ]
 */
```

如果你只需要单个标头, 可以将名称传递给 `header()` 方法。这将以不区分大小写的方式获取指定的标头对象 (如果存在)。如果不存在, 则返回 `null`:

```
<?php

// these are all equivalent
$host = $request->header('host');
$host = $request->header('Host');
$host = $request->header('HOST');
```

你可以始终使用 `hasHeader()` 来查看该请求中是否存在标头:

```
<?php

if ($request->hasHeader('DNT')) {
    // Don't track something...
}
```

如果你需要将标头的值作为单行字符串, 其中所有值在一行中, 可以使用 `getHeaderLine()` 方法:

```
<?php

// Accept-Encoding: gzip, deflate, sdch
echo 'Accept-Encoding: ' . $request->getHeaderLine('accept-encoding
↪');
```

如果你需要将标头及其名称和值合并为单个字符串, 只需将标头转换为字符串:

```
<?php

echo (string) $header;
```

请求 URL

你可以通过 `$request->getUri()` 方法检索表示当前 URI 的*URI* 对象。你可以将此对象转换为字符串以获取当前请求的完整 URL:

```
<?php

$uri = (string) $request->getUri();
```

该对象使你能够自行获取请求的任何部分:

```
<?php

$uri = $request->getUri();

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->getRoutePath();       // path/to/page
echo $uri->getQuery();           // foo=bar&bar=baz
print_r($uri->getSegments());    // Array ( [0] => path [1] => to␣
↪[2] => page )
echo $uri->getSegment(1);        // path
```

(续下页)

(接上页)

```
echo $uri->getTotalSegments();  // 3
```

你可以使用 getRoutePath() 方法来处理当前 URI 字符串（相对于你的 baseURL 的路径）。

备注: 自 v4.4.0 版本开始，可以使用 getRoutePath() 方法。在 v4.4.0 之前，getPath() 方法返回相对于你的 baseURL 的路径。

上传的文件

可以通过 $request->getFiles() 获取有关所有上传文件信息，它返回 CodeIgniter\HTTP\Files\UploadedFile 实例的数组。这有助于减轻使用上传文件时的痛苦，并使用最佳实践来最大程度地减少任何安全风险。

```
<?php

$files = $request->getFiles();
```

参见使用上传的文件 以获取详细信息。

你可以根据 HTML 文件输入中给出的文件名检索单独上传的文件:

```
<?php

$file = $request->getFile('userfile');
```

你可以检索作为多文件上传一部分上传的同名文件数组，基于 HTML 文件输入中给出的文件名:

```
<?php

$files = $request->getFileMultiple('userfile');
```

备注: 这里的文件对应于 $_FILES。即使用户仅点击表单的提交按钮而不上传任何文件，文件也会存在。你可以通过 UploadedFile 中的 isValid() 方法检查文件是否实际

被上传。有关详细信息, 请参阅验证文件。

内容协商

你可以通过 `negotiate()` 方法轻松地与请求协商内容类型:

```
<?php

$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
↪']);
```

有关更多详细信息, 请参阅内容协商 页面。

类参考

备注: 除了这里列出的方法之外, 此类还继承了请求类 和消息类 的方法。

父类提供的可用方法有:

- `CodeIgniter\HTTP\Request::getIPAddress()`
- `CodeIgniter\HTTP\Request::isValidIP()`
- `CodeIgniter\HTTP\Request::getMethod()`
- `CodeIgniter\HTTP\Request::setMethod()`
- `CodeIgniter\HTTP\Request::getServer()`
- `CodeIgniter\HTTP\Request::getEnv()`
- `CodeIgniter\HTTP\Request::setGlobal()`
- `CodeIgniter\HTTP\Request::fetchGlobal()`

- `CodeIgniter\HTTP\Message::getBody()`
- `CodeIgniter\HTTP\Message::setBody()`
- `CodeIgniter\HTTP\Message::appendBody()`
- `CodeIgniter\HTTP\Message::populateHeaders()`
- `CodeIgniter\HTTP\Message::headers()`
- `CodeIgniter\HTTP\Message::header()`
- `CodeIgniter\HTTP\Message::hasHeader()`
- `CodeIgniter\HTTP\Message::getHeaderLine()`
- `CodeIgniter\HTTP\Message::setHeader()`
- `CodeIgniter\HTTP\Message::removeHeader()`
- `CodeIgniter\HTTP\Message::appendHeader()`
- `CodeIgniter\HTTP\Message::prependHeader()`
- `CodeIgniter\HTTP\Message::getProtocolVersion()`
- `CodeIgniter\HTTP\Message::setProtocolVersion()`

class `CodeIgniter\HTTP\`**`IncomingRequest`**

`isCLI``()`

返回

如果请求是从命令行发起的, 则为 True, 否则为 False

返回类型

`bool`

`isAJAX``()`

返回

如果请求是 AJAX 请求, 则为 True, 否则为 False

返回类型

`bool`

`isSecure``()`

返回

如果请求是 HTTPS 请求, 则为 True, 否则为 False

返回类型

bool

getVar([*$index = null*[, *$filter = null*[, *$flags = null*]]])

参数

- **$index** (string) –要查找的变量/键的名称。
- **$filter** (int) –要应用的过滤器类型。过滤器列表可在 过滤器类型 中找到。
- **$flags** (int) –要应用的标志。标志列表可在 过滤器 flag 中找到。

返回

如果没有提供参数, 则返回 $_REQUEST, 否则如果找到则返回 REQUEST 值, 如果没找到则为 null

返回类型

array|bool|float|int|object|string|null

重要: 该方法仅存在于向后兼容中。不要在新项目中使用它。即使你已经在使用，我们也建议你使用另一个更合适的方法。

这个方法与 getGet() 相同，只是它获取的是 REQUEST 数据。

getGet([*$index = null*[, *$filter = null*[, *$flags = null*]]])

参数

- **$index** (string) –要查找的变量/键的名称。
- **$filter** (int) –要应用的过滤类型。过滤器类型列表可以在 过滤器类型 中找到。
- **$flags** (int) –要应用的标记。标记列表可以在 过滤器 flag 中找到。

返回

如果没有提供参数，则为 $_GET，否则如果找到 GET 值则为 GET 值，如果未找到则为 null

返回类型

array|bool|float|int|object|string|null

第一个参数将包含你正在查找的 GET 项的名称:

```
<?php

$request->getGet('some_data');
```

如果尝试检索的项目不存在, 该方法将返回 null。

第二个可选参数允许你通过 PHP 的过滤器运行数据。将所需的过滤器类型作为第二个参数传递:

```
<?php

$request->getGet('some_data', FILTER_SANITIZE_FULL_SPECIAL_
↪CHARS);
```

若要返回所有 GET 项, 请不带任何参数调用。

要返回所有 GET 项并通过过滤器传递它们, 请将第一个参数设置为 null, 同时将第二个参数设置为要使用的过滤器:

```
<?php

$request->getGet(null, FILTER_SANITIZE_FULL_SPECIAL_CHARS);
// returns all GET items with string sanitation
```

要返回多个 GET 参数的数组, 请传递所有所需键的数组:

```
<?php

$request->getGet(['field1', 'field2']);
```

这里也应用了相同的规则, 要使用过滤检索参数, 请将第二个参数设置为要应用的过滤器类型:

```
<?php

$request->getGet(['field1', 'field2'], FILTER_SANITIZE_FULL_
↪SPECIAL_CHARS);
```

getPost([*$index* = *null*[, *$filter* = *null*[, *$flags* = *null*]]])

参数

- **$index** (string) –要查找的变量/键的名称。
- **$filter** (int) –要应用的过滤器类型。过滤器列表可在 这里 找到。
- **$flags** (int) –要应用的标志。标志列表可在 这里 找到。

返回

如果没有提供参数, 则返回 $_POST, 否则如果找到则返回 POST 值, 如果没找到则为 null

返回类型

array|bool|float|int|object|string|null

此方法与 getGet() 相同, 只是它获取 POST 数据。

getPostGet([*$index = null*[, *$filter = null*[, *$flags = null*]]])

参数

- **$index** (string) –要查找的变量/键的名称。
- **$filter** (int) –要应用的过滤器类型。过滤器列表可在 过滤器类型 中找到。
- **$flags** (int) –要应用的标志。标志列表可在 过滤器 flag 中找到。

返回

如果没有指定参数, 则返回 $_POST 和 $_GET 组合 (冲突时优先 POST 值), 否则首先查找 POST 值, 找不到则查找 GET 值, 如果没找到则返回 null

返回类型

array|bool|float|int|object|string|null

这个方法的工作原理与 getPost() 和 getGet() 基本相同, 只是结合了两者。它将在 POST 和 GET 流中搜索数据, 先在 POST 中查找, 然后在 GET 中查找:

```
<?php

$request->getPostGet('field1');
```

如果没有指定索引, 它将返回 POST 和 GET 流组合。如果名称冲突, 将优先 POST 数据。

getGetPost ([*$index = null*[, *$filter = null*[, *$flags = null*]]])

参数

- **$index** (string) –要查找的变量/键的名称。
- **$filter** (int) –要应用的过滤器类型。过滤器列表可在 过滤器类型 中找到。
- **$flags** (int) –要应用的标志。标志列表可在 过滤器 flag 中找到。

返回

如果没有指定参数, 则返回 $_GET 和 $_POST 组合 (冲突时优先 GET 值), 否则首先查找 GET 值, 找不到则查找 POST 值, 如果没找到则返回 null

返回类型

array|bool|float|int|object|string|null

这个方法的工作原理与 getPost() 和 getGet() 基本相同, 只是结合了两者。它将在 GET 和 POST 流中搜索数据, 先在 GET 中查找, 然后在 POST 中查找:

```
<?php

$request->getGetPost('field1');
```

如果没有指定索引, 它将返回 GET 和 POST 流组合。如果名称冲突, 将优先 GET 数据。

getCookie ([*$index = null*[, *$filter = null*[, *$flags = null*]]])

参数

- **$index** (array|string|null) –COOKIE 名称
- **$filter** (int) –要应用的过滤器类型。过滤器列表可在 过滤器类型 中找到。
- **$flags** (int) –要应用的标志。标志列表可在 过滤器 flag 中找到。

返回

如果没有提供参数, 则返回 `$_COOKIE`, 否则如果找到则返回 COOKIE 值, 如果没有找到则为 null

返回类型

`array|bool|float|int|object|string|null`

此方法与 `getPost()` 和 `getGet()` 相同, 只是它获取 cookie 数据:

```
<?php

$request->getCookie('some_cookie');
$request->getCookie('some_cookie', FILTER_SANITIZE_FULL_
↪SPECIAL_CHARS); // with filter
```

要返回多个 cookie 值的数组, 请传递所有所需键的数组:

```
<?php

$request->getCookie(['some_cookie', 'some_cookie2']);
```

备注: 与*Cookie 辅助函数* 函数*`get_cookie()`* 不同, 此方法不会在配置的 `Config\Cookie::$prefix` 值前加上前缀。

---

getServer([*$index* = *null*[, *$filter* = *null*[, *$flags* = *null*]]])

参数

- **$index** (`array|string|null`) –值名称
- **$filter** (`int`) –要应用的过滤器类型。过滤器列表可在 过滤器类型 中找到。
- **$flags** (`int`) –要应用的标志。标志列表可在 过滤器 flag 中找到。

返回

如果找到则返回 `$_SERVER` 项的值, 否则为 null

返回类型

`array|bool|float|int|object|string|null`

此方法与 getPost()、getGet() 和 getCookie() 方法相同, 只是它获取 Server 数据 ($_SERVER):

```
<?php

$request->getServer('some_data');
```

要返回多个 $_SERVER 值的数组, 请传递所有所需键的数组。

```
<?php

$request->getServer(['SERVER_PROTOCOL', 'REQUEST_URI']);
```

getUserAgent()

返回

在 SERVER 数据中找到的用户代理字符串, 如果没有找到则为 null。

返回类型

CodeIgniter\HTTP\UserAgent

此方法返回来自 SERVER 数据的用户代理实例:

```
<?php

$request->getUserAgent();
```

getPath()

返回

相对于 baseURL 的当前 URI 路径

返回类型

string

该方法返回相对于 baseURL 的当前 URI 路径。

备注: 在 v4.4.0 之前, 这是确定 "当前 URI" 的最安全的方法, 因为 IncomingRequest::$uri 可能不知道完整的 App 配置的 base URL。

---

setPath(*$path*)

自 4.4.0 版本弃用.

参数

- **$path** (string) –用作当前 URI 的相对路径

返回

此传入请求

返回类型

IncomingRequest

备注: 在 v4.4.0 之前，主要用于测试目的，这允许你设置当前请求的相对路径值，而不是依赖于 URI 检测。这也会更新底层的 URI 实例的新路径。

5.1.8 内容协商

- 什么是内容协商？
- 加载类
- 协商过程
  - 媒体类型
  - 语言协商
  - 编码协商
  - 字符集协商

什么是内容协商？

内容协商是一种根据客户端处理能力和服务器处理能力，决定返回何种内容类型给客户端的方式。这种方式可用于判断客户端需要 HTML 还是 JSON 响应，图像应以 JPEG 还是 PNG 格式返回，支持哪种压缩类型等。该机制通过分析四个不同的头部实现，每个头部可支持多个具有优先级的选项值。

手动实现这种匹配可能颇具挑战。CodeIgniter 提供了 `Negotiator` 类来为你处理这些工作。

本质上，内容协商是 HTTP 规范的一部分，允许单一资源提供多种内容类型，让客户端请求最适合其需求的数据格式。

一个经典例子是：无法显示 PNG 文件的浏览器可以请求仅限 GIF 或 JPEG 图像。服务器接收请求时，会查看客户端请求的可用文件类型，并从支持的图像格式中选择最佳匹配（本例中可能选择返回 JPEG 图像）。

这种协商可应用于四种数据类型：

- **媒体/文档类型** - 可以是图像格式，或 HTML、XML、JSON 等文档格式
- **字符集** - 返回文档应使用的字符集，通常为 UTF-8
- **文档编码** - 通常指对结果使用的压缩类型
- **文档语言** - 对于支持多语言的站点，帮助确定返回哪种语言版本

加载类

你可以通过 Service 类手动加载该类的实例：

```
<?php

$negotiate = service('negotiator');
```

这将获取当前请求实例并自动注入到 Negotiator 类中。

此类无需单独加载。你可以通过请求的 `IncomingRequest` 实例访问其方法。虽然不能直接访问，但可以通过 `negotiate()` 方法轻松调用所有功能：

```
<?php

$request->negotiate('media', ['foo', 'bar']);
```

通过此方式访问时，第一个参数是你尝试匹配的内容类型，第二个参数是支持的选项数组。

协商过程

本节将讨论可协商的四种内容类型，并展示使用上述两种方法访问协商器的示例。

媒体类型

首要处理的是「媒体类型」协商。这些信息由 `Accept` 头部提供，是最复杂的头部之一。常见用例是客户端告知服务器期望的数据格式，这在 API 中尤为常见。例如，客户端可能请求 API 端点返回 JSON 格式数据:

```
GET /foo HTTP/1.1
Accept: application/json
```

服务器现在需要提供其支持的内容类型列表。在此示例中，API 可能返回原始 HTML、JSON 或 XML 格式数据。该列表应按优先级顺序排列：

```php
<?php

$supported = [
    'application/json',
    'text/html',
    'application/xml',
];

$format = $request->negotiate('media', $supported);
// or
$format = $negotiate->media($supported);
```

本例中，客户端和服务器都同意使用 JSON 格式，因此 negotiate 方法返回 ‘json’。默认情况下，若未找到匹配项，将返回 `$supported` 数组的第一个元素。但有时需要严格匹配格式，若将最后一个参数设为 `true`，则无匹配时返回空字符串：

```php
<?php

$format = $request->negotiate('media', $supported, true);
// or
$format = $negotiate->media($supported, true);
```

语言协商

另一个常见用途是确定返回内容的语言。对于单语言站点影响不大，但任何支持多语言翻译的站点都会发现其用处，因为浏览器通常会在 `Accept-Language` 头部发送首选语言:

```
GET /foo HTTP/1.1
Accept-Language: fr; q=1.0, en; q=0.5
```

本例中，浏览器首选法语，次选英语。若你的网站支持英语和德语，可进行如下操作:

```
<?php

$supported = [
    'en',
    'de',
];

$lang = $request->negotiate('language', $supported);
// or
$lang = $negotiate->language($supported);
```

此例将返回 ‘en’ 作为当前语言。若无匹配项，则返回 `$supported` 数组的第一个元素，因此该元素应始终设为首选语言。

严格区域设置协商

在 4.6.0 版本加入.

默认情况下，区域设置基于近似匹配（仅考虑 locale 字符串的首部分即语言）。这通常已足够。但有时我们需要区分诸如 `en-US` 和 `en-GB` 等区域版本以提供不同内容。

针对此类情况,我们新增了可通过 `Config\Feature::$strictLocaleNegotiation` 启用的设置。这将确保首先进行严格比较。

备注: CodeIgniter 仅为主语言标签（’en’，‘fr’ 等）提供翻译。若启用此功能且 `Config\App::$supportedLocales` 包含区域语言标签（’en-US’，‘fr-FR’ 等），请注意：若你拥有自定义翻译文件，**必须同时修改** CodeIgniter 翻译文件的文件夹名称

以匹配 $supportedLocales 数组中的设置。

现在考虑以下示例，浏览器首选语言设置为:

```
GET /foo HTTP/1.1
Accept-Language: fr; q=1.0, en-GB; q=0.5
```

本例中，浏览器首选法语，次选英语（英国）。而你的网站支持德语和英语（美国）:

```
<?php

$supported = [
    'de',
    'en-US',
];

$lang = $request->negotiate('language', $supported);
// or
$lang = $negotiate->language($supported);
```

此例将返回‘en-US’作为当前语言。若无匹配项，则返回 $supported 数组的首元素。以下是区域选择过程的具体工作原理。

尽管浏览器首选‘fr’，但其不在我们的 $supported 数组中。’en-GB’同样存在匹配问题，但我们可以搜索变体。首先回退到最通用的区域设置（本例为‘en’），其仍不在数组中。接着搜索区域设置‘en- ‘，此时将匹配 $supported 数组中的‘en-US’并返回。

区域选择流程如下:

1. 严格匹配（’en-GB’）- ISO 639-1 加 ISO 3166-1 alpha-2
2. 通用区域匹配（’en’）- ISO 639-1
3. 区域通配符匹配（’en-’）- ISO 639-1 加 ISO 3166-1 alpha-2 通配符

编码协商

Accept-Encoding 头部包含客户端偏好的字符集，用于指定支持的压缩类型:

```
GET /foo HTTP/1.1
Accept-Encoding: compress, gzip
```

你的 Web 服务器将定义可使用的压缩类型。某些服务器（如 Apache）仅支持 **gzip**:

```
<?php

$type = $request->negotiate('encoding', ['gzip']);
// or
$type = $negotiate->encoding(['gzip']);
```

更多信息参见 维基百科。

字符集协商

期望的字符集通过 Accept-Charset 头部传递:

```
GET /foo HTTP/1.1
Accept-Charset: utf-16, utf-8
```

默认情况下若无匹配项，将返回 **utf-8**:

```
<?php

$charset = $request->negotiate('charset', ['utf-8']);
// or
$charset = $negotiate->charset(['utf-8']);
```

5.1.9 HTTP 方法欺骗

当使用 HTML 表单时, 你只能使用 GET 或 POST HTTP 动词。在大多数情况下, 这已经足够了。然而, 为了支持 RESTful 路由, 你需要支持其他更正确的动词, 比如 DELETE 或 PUT。由于浏览器不支持这些,CodeIgniter 为你提供了一种办法来欺骗所使用的方法。这使得你可以发出一个 POST 请求, 但是告诉应用程序它应该被视为不同的请求类型。

要欺骗方法, 需要在表单中添加一个隐藏的输入, 名称为 `_method`。它的值是你希望请求采用的 HTTP 动词:

```
<form action="" method="post">
    <input type="hidden" name="_method" value="PUT">
</form>
```

这个表单会被转换成一个 PUT 请求, 对于路由和 IncomingRequest 类来说, 它是一个真正的 PUT 请求。

你使用的表单必须是一个 POST 请求。GET 请求无法被欺骗。

备注: 请确保检查你的 Web 服务器配置, 因为一些服务器默认配置不支持所有 HTTP 动词, 必须启用其他软件包才能正常工作。

5.1.10 RESTful 资源处理

- 资源路由
  - 更改使用的控制器
  - 更改使用的占位符
  - 限制生成的路由
- *ResourceController*
- *Presenter* 路由
  - 更改使用的控制器
  - 更改使用的占位符
  - 限制生成的路由

- *ResourcePresenter*
- *Presenter/Controller* 对比

表述性状态转移 (REST) 是一种用于分布式应用程序的架构风格, 首先由 Roy Fielding 在他的 2000 年博士论文 《Architectural Styles and the Design of Network-based Software Architectures》 中描述。这可能有点枯燥, 你可能会发现 Martin Fowler 的 《Richardson 成熟度模型》 是一个更温和的介绍。

REST 的解释和误解的方式比大多数软件体系结构都要多, 可以说你在体系结构中采用的 Roy Fielding 原则越多, 你的应用程序就越被认为是 "RESTful"。

CodeIgniter 通过其资源路由和 *ResourceController* 可以轻松创建资源的 RESTful API。

资源路由

你可以使用 `resource()` 方法快速为单个资源创建一组 RESTful 路由。这将创建用于完整 CRUD 操作的 5 个最常用的路由：创建新资源、更新现有资源、列出所有资源、显示单个资源和删除单个资源。第一个参数是资源名称：

```
<?php

$routes->resource('photos');

// Equivalent to the following:
$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');
```

备注： 上面的顺序是为了清晰起见, 而实际创建路由的顺序在 RouteCollection 中确保了正确的路由解析

重要： 路由按指定顺序匹配, 因此如果你在上方有一个资源 photos, 然后有一个 get 'pho-

tos/poll’ , 资源线的 show 操作的路由将在 get 线之前匹配。要解决此问题, 请将 get 行移动到资源行之上, 以便先匹配它。

第二个参数接受可以用于修改生成的路由的选项数组。虽然这些路由面向 API 使用, 其中允许更多方法, 但你可以传入 `websafe` 选项, 使其生成适用于 HTML 表单的 update 和 delete 方法:

```
<?php

$routes->resource('photos', ['websafe' => 1]);

// The following equivalent routes are created:
$routes->post('photos/(:segment)/delete', 'Photos::delete/$1');
$routes->post('photos/(:segment)', 'Photos::update/$1');
```

更改使用的控制器

你可以通过使用应该使用的控制器的名称传递 `controller` 选项来指定应该使用的控制器:

```
<?php

$routes->resource('photos', ['controller' => 'Gallery']);
// Would create routes like:
$routes->get('photos', '\App\Controllers\Gallery::index');
```

```
<?php

$routes->resource('photos', ['controller' => '\App\Gallery']);
// Would create routes like:
$routes->get('photos', '\App\Gallery::index');
```

```
<?php

use App\Controllers\Gallery;

$routes->resource('photos', ['namespace' => '', 'controller' =>␣
```

(续下页)

(接上页)

```
↪Gallery::class]);
// Would create routes like:
$routes->get('photos', '\App\Controllers\Gallery::index');
```

也可参考控制器的命名空间。

更改使用的占位符

默认情况下, 当需要资源 ID 时, 会使用 (:segment) 占位符。你可以通过传递 placeholder 选项及要使用的新字符串来更改此占位符:

```
<?php

$routes->resource('photos', ['placeholder' => '(:num)']);

// Generates routes like:
$routes->get('photos/(:num)', 'Photos::show/$1');
```

限制生成的路由

你可以使用 only 选项限制生成的路由。这应该是 **数组**或 **以逗号分隔的方法名列表**, 应该创建这些方法。仅将创建与这些方法之一匹配的路由。其余的会被忽略:

```
<?php

$routes->resource('photos', ['only' => ['index', 'show']]);
```

否则, 你可以使用 except 选项删除未使用的路由。这也应该是 **数组**或 **以逗号分隔的方法名列表**。此选项在 only 之后运行:

```
<?php

$routes->resource('photos', ['except' => 'new,edit']);
```

有效的方法是:index、show、create、update、new、edit 和 delete。

ResourceController

ResourceController 为你的 RESTful API 提供了一个方便的起点, 其方法对应于上面的资源路由。

扩展它, 覆盖 modelName 和 format 属性, 然后实现你想要处理的那些方法:

```
<?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());
    }

    // ...
}
```

路由如下:

```
<?php

$routes->resource('photos');
```

Presenter 路由

你可以使用 presenter() 方法快速创建与资源控制器对齐的表示控制器。这将创建对应于上面的资源控制器方法的路由, 这些方法会为你的资源返回视图, 或处理来自这些视图的表单提交。

这不是必需的, 因为表示可以通过常规控制器处理 - 这只是为了方便。其用法与资源路由类似:

```
<?php

$routes->presenter('photos');

// Equivalent to the following:
$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::delete/$1');
```

备注: 上面的顺序是为了清晰起见, 而实际创建路由的顺序在 RouteCollection 中确保了正确的路由解析

你不会为资源和表示控制器都有 *photos* 路由。你需要加以区分, 例如:

```
<?php

$routes->resource('api/photo');
$routes->presenter('admin/photos');
```

第二个参数接受可以用于修改生成的路由的选项数组。

更改使用的控制器

你可以通过使用应该使用的控制器的名称传递 `controller` 选项来指定应该使用的控制器:

```
<?php

$routes->presenter('photos', ['controller' => 'Gallery']);
```

(续下页)

(接上页)

```
// Would create routes like:
$routes->get('photos', '\App\Controllers\Gallery::index');
```

```
<?php

$routes->presenter('photos', ['controller' => '\App\Gallery']);
// Would create routes like:
$routes->get('photos', '\App\Gallery::index');
```

```
<?php

use App\Controllers\Gallery;

$routes->presenter('photos', ['namespace' => '', 'controller' =>␣
↪Gallery::class]);
// Would create routes like:
$routes->get('photos', '\App\Controllers\Gallery::index');
```

也可参考控制器的命名空间。

更改使用的占位符

默认情况下，当需要资源 ID 时，会使用 `(:segment)` 占位符。你可以通过传递 `placeholder` 选项及要使用的新字符串来更改此占位符:

```
<?php

$routes->presenter('photos', ['placeholder' => '(:num)']);

// Generates routes like:
$routes->get('photos/(:num)', 'Photos::show/$1');
```

限制生成的路由

你可以使用 only 选项限制生成的路由。这应该是 **数组**或 **以逗号分隔的方法名列表**, 应该创建这些方法。仅将创建与这些方法之一匹配的路由。其余的会被忽略:

```
<?php

$routes->presenter('photos', ['only' => ['index', 'show']]);
```

否则, 你可以使用 except 选项删除未使用的路由。这也应该是 **数组**或 **以逗号分隔的方法名列表**。此选项在 only 之后运行:

```
<?php

$routes->presenter('photos', ['except' => 'new,edit']);
```

有效的方法是:index、show、new、create、edit、update、remove 和 delete。

ResourcePresenter

ResourcePresenter 为呈现资源视图以及处理这些视图中的表单数据提供了一个方便的起点, 其方法与上面的资源路由对齐。

扩展它, 重写 modelName 属性, 然后实现你想要处理的方法:

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

(续下页)

(接上页)

```
    // ...
}
```

路由如下:

```
<?php

$routes->presenter('photos');
```

Presenter/Controller 对比

此表比较了 *resource()* 和 *presenter()* 创建的默认路由及其相应的 Controller 函数。

操作	方法	控制器路由	表示器路由	控制器函数	表示器函数
New	GET	photos/new	photos/new	`new()`	`new()`
Create	POST	photos	photos	`create()`	`create()`
Create(别名)	POST		photos/create		`create()`
List	GET	photos	photos	`index()`	`index()`
Show	GET	photos/(:segment)	photos/(:segment)	`show($id = null)`	`show($id = null)`
Show(别名)	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(网页安全)	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(网页安全)	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

- 创建视图
- 显示视图
- 加载多个视图
- 在子目录中存储视图
- 命名空间视图
- 缓存视图
- 向视图添加动态数据
  - *saveData* 选项
- 创建循环

视图只是一个网页或页面片段, 例如头部、尾部、侧边栏等。事实上, 如果需要这种层次结构, 可以将视图灵活地嵌入其他视图 (嵌入其他视图等)。

视图从不直接调用, 它们必须由控制器或视图路由 加载。

请记住, 在 MVC 框架中, 控制器充当交通警察角色, 因此它负责获取特定视图。如果你还没有阅读控制器 页面, 在继续之前应该阅读它。

使用你在控制器页面中创建的示例控制器, 让我们为它添加一个视图。

创建视图

使用你的文本编辑器, 创建一个名为 **blog_view.php** 的文件, 并将此内容放入其中:

```
<html>
    <head>
        <title>我的博客</title>
    </head>
```

(续下页)

(接上页)

```
    <body>
        <h1>欢迎访问我的博客!</h1>
    </body>
</html>
```

然后将该文件保存到你的 **app/Views** 目录中。

显示视图

要加载和显示特定的视图文件，你可以在控制器中使用 `view()` 函数，如以下代码所示：

```
return view('name');
```

其中 *name* 是你的视图文件的名称。

重要： 如果省略了文件扩展名，则视图预期以 **.php** 扩展名结尾。

备注： `view()` 函数内部使用了视图渲染器。

现在，在 **app/Controllers** 目录中创建一个名为 **Blog.php** 的文件，并将此内容放入其中：

```
<?php

namespace App\Controllers;

class Blog extends BaseController
{
    public function index()
    {
        return view('blog_view');
    }
}
```

打开位于 **app/Config/Routes.php** 的路由文件，并查找 “路由定义”。添加以下代码：

```
use App\Controllers\Blog;

$routes->get('blog', [Blog::class, 'index']);
```

如果你访问你的网站, 应该可以看到你的新视图。URL 类似于这样:

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

加载多个视图

CodeIgniter 会智能地处理控制器内对 `view()` 的多次调用。如果发生多次调用，它们将被拼接在一起。

例如，你可能希望有一个头部视图、一个菜单视图、一个内容视图和一个底部视图。代码可能如下所示:

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class Page extends Controller
{
    public function index()
    {
        $data = [
            'page_title' => 'Your title',
        ];

        return view('header')
            . view('menu')
            . view('content', $data)
            . view('footer');
    }
}
```

在上面的示例中, 我们使用了“动态添加的数据”，稍后你会看到。

在子目录中存储视图

如果你更喜欢那种组织方式, 也可以将视图文件存储在子目录中。在这种情况下, 在加载视图时需要包括目录名称。例如:

```
return view('directory_name/file_name');
```

命名空间视图

你可以在命名空间下的 **View** 目录中存储视图, 并加载那个视图, 就像它带有命名空间一样。尽管 PHP 不支持从命名空间加载非类文件, 但 CodeIgniter 提供了这个功能, 以便以模块化的方式打包视图以进行轻松重用或分发。

如果你在*自动加载器* 中有一个映射了 PSR-4 命名空间 `Example\Blog` 的 **example/blog** 目录, 你可以像命名空间一样检索视图文件。

按照此示例, 你可以通过在视图名称前加上命名空间来加载 **example/blog/Views** 中的 **blog_view.php** 文件:

```
<?php

return view('Example\Blog\Views\blog_view');
```

缓存视图

你可以通过在*view()* 函数的第三个参数中传递一个 `cache` 选项，并指定缓存视图的秒数来缓存视图:

```
// Cache the view for 60 seconds
return view('file_name', $data, ['cache' => 60]);
```

默认情况下, 视图将使用视图文件本身的相同名称进行缓存。你可以通过传递 `cache_name` 和希望使用的缓存 ID 来自定义此名称:

```
// Cache the view for 60 seconds
return view('file_name', $data, ['cache' => 60, 'cache_name' => 'my_
↪cached_view']);
```

向视图添加动态数据

数据通过*view()* 函数的第二个参数以数组的形式从控制器传递到视图。

这里有个例子:

```
$data = [
    'title'   => 'My title',
    'heading' => 'My Heading',
    'message' => 'My Message',
];

return view('blog_view', $data);
```

让我们在控制器文件中试一试。打开它并添加这段代码:

```
<?php

namespace App\Controllers;

class Blog extends BaseController
{
    public function index()
    {
        $data['title']   = 'My Real Title';
        $data['heading'] = 'My Real Heading';

        return view('blog_view', $data);
    }
}
```

现在打开你的视图文件, 并将文本更改为与数据数组中的数组键对应的参数:

```
<html>
<head>
    <title><?= esc($title) ?></title>
</head>
<body>
    <h1><?= esc($heading) ?></h1>
```

(续下页)

(接上页)

```
    <h2>My Todo List</h2>

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

        <li><?= esc($item) ?></li>

    <?php endforeach ?>
    </ul>

</body>
</html>
```

然后在你一直使用的 URL 加载页面, 你应该可以看到变量被替换了。

saveData 选项

传递的数据在后续对*view()* 的调用中会被保留。如果你在单个请求中多次调用该函数，你不需要每次都将所需数据传递给每个 `view()`。

但这可能无法防止任何数据 "渗透" 到其他视图中, 从而潜在地造成问题。如果你更喜欢在一次调用后清除数据, 可以将 `saveData` 选项传递到第三个参数中的 `$option` 数组中。

```
$data = [
    'title'   => 'My title',
    'heading' => 'My Heading',
    'message' => 'My Message',
];

return view('blog_view', $data, ['saveData' => false]);
```

另外，如果你希望 `view()` 函数的默认功能是在调用之间清除数据，可以在 **app/Config/Views.php** 中将 `$saveData` 设置为 `false`。

创建循环

你传递给视图文件的数组不限于简单变量。你可以传递多维数组, 它可以循环以生成多行。例如, 如果从数据库中拉取数据, 它通常以多维数组的形式出现。

这是一个简单的例子。将此内容添加到你的控制器中:

```
<?php

namespace App\Controllers;

class Blog extends BaseController
{
    public function index()
    {
        $data = [
            'todo_list' => ['Clean House', 'Call Mom', 'Run Errands
↪'],
            'title'     => 'My Real Title',
            'heading'   => 'My Real Heading',
        ];

        return view('blog_view', $data);
    }
}
```

现在打开你的视图文件并创建一个循环:

```
<html>
<head>
    <title><?= esc($title) ?></title>
</head>
<body>
    <h1><?= esc($heading) ?></h1>

    <h2>My Todo List</h2>

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

(续下页)

(接上页)

```
        <li><?= esc($item) ?></li>

    <?php endforeach ?>
    </ul>

</body>
</html>
```

5.2.2 视图渲染器

- 使用视图渲染器
  - 它的工作原理
  - 设置视图参数
  - 方法链式调用
  - 转义数据
  - 视图渲染器选项
- 类参考

使用视图渲染器

view() 函数是一个方便的函数, 它获取 `renderer` 服务的一个实例, 设置数据并渲染视图。尽管这通常正是你想要的, 但你可能会发现有时你更希望直接与它一起工作。在这种情况下, 你可以直接访问视图服务:

```
$view = service('renderer');
```

或者, 如果你没有使用 `View` 类作为默认渲染器, 你可以直接实例化它:

```
$view = new \CodeIgniter\View\View();
```

重要: 你应该仅在控制器中创建服务。如果你需要从库中访问 View 类, 应该在库的构

造函数中将其设置为依赖项。

然后, 你可以使用它提供的三种标准方法中的任何一种: *render()*、*setVar()* 和 `setData()` <CodeIgniter\View\View::setData()。

它的工作原理

`View` 类在提取视图参数为 PHP 变量后处理应用程序视图路径中的常规 HTML/PHP 脚本, 使脚本可以访问它们。这意味着视图参数名称需要是合法的 PHP 变量名称。

View 类在内部使用关联数组来累积视图参数, 直到你调用它的 `render()`。这意味着你的参数 (或变量) 名称需要是唯一的, 否则后面的变量设置将覆盖早期的设置。

这也会影响根据脚本中的不同上下文对参数值进行转义。你将必须为每个转义值提供一个唯一的参数名称。

值为数组的参数没有特殊含义。需要你在 PHP 代码中适当处理数组。

设置视图参数

setVar() 方法设置一个视图参数。

```
$view->setVar('name', 'Joe', 'html');
```

setData() 方法一次设置多个视图参数。

```
$view->setData(['name' => 'George', 'position' => 'Boss']);
```

方法链式调用

`setVar()` 和 `setData()` 方法是可链式调用的, 允许你将许多不同的调用组合在一起:

```
$view->setVar('one', $one)
    ->setVar('two', $two)
    ->render('myView');
```

转义数据

当你将数据传递给 `setVar()` 和 `setData()` 函数时, 你可以选择对数据进行转义以防止跨站脚本攻击。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来转义数据。请参见下文了解上下文描述。

如果你不想转义数据, 可以将 `'raw'` 或 `null` 作为每个函数的最后一个参数传递:

```
$view->setVar('one', $one, 'raw');
```

如果选择不转义数据, 或者正在传递对象实例, 则可以在视图中使用*esc()* 函数手动转义数据。第一个参数是要转义的字符串。第二个参数是转义数据的上下文 (见下文):

```
<?= esc($object->getStat()) ?>
```

转义上下文

默认情况下, `esc()` 以及转而 `setVar()` 和 `setData()` 函数假设你要转义的数据打算在标准 HTML 中使用。然而, 如果数据打算用于 JavaScript、CSS 或 href 属性中, 你需要不同的转义规则才能有效。你可以将上下文的名称作为第二个参数传递。有效的上下文是 `'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

可以将多个选项传递给 `render()` 或 `renderString()` 方法:

- `$options`
  - `cache` - 缓存视图结果的时间（以秒为单位）；对于 `renderString()` 无效。
  - `cache_name` - 用于保存/检索缓存视图结果的 ID；默认为 `$viewPath`；对于 `renderString()` 无效。
  - `debug` - 可以设置为 false 以禁用调试工具栏 的调试代码添加。
- `$saveData` - 如果视图数据参数应保留以供后续调用，则为 true。

备注: 接口定义的 `$saveData` 必须是布尔值，但实现类（如下面的 `View`）可以扩展为包括 `null` 值。

类参考

class `CodeIgniter\View\View`

render(*$view*[, *$options*[, *$saveData = false*]])

参数

- **$view** (`string`) –视图源文件的名称
- **$options** (`array`) –选项的键/值对数组
- **$saveData** (`boolean|null`) –如果为 true, 将保存数据供任何其他调用使用。如果为 false, 渲染视图后将清除数据。如果为 null, 使用配置设置。

返回

所选视图的渲染文本

返回类型

`string`

根据文件名和已设置的数据构建输出:

```
echo $view->render('myview');
```

renderString(*$view*[, *$options*[, *$saveData = false*]])

参数

- **$view** (string) –要渲染的视图内容, 例如从数据库检索的内容
- **$options** (array) –选项的键/值对数组
- **$saveData** (boolean|null) –如果为 true, 将保存数据供任何其他调用使用。如果为 false, 渲染视图后将清除数据。如果为 null, 使用配置设置。

返回

所选视图的渲染文本

返回类型

string

根据视图片段和已设置的数据构建输出:

```
echo $view->renderString('<div>My Sharona</div>');
```

> **警告:** 这可以用来显示可能存储在数据库中的内容, 但你需要注意这是一个潜在的安全漏洞, 并且你 **必须**验证任何此类数据, 可能适当地对其进行转义!

setData([*$data*[, *$context = null*]])

参数

- **$data** (array) –视图数据字符串的关联数组, 作为键/值对
- **$context** (string) –用于数据转义的上下文

返回

渲染器, 用于方法链

返回类型

CodeIgniter\View\RendererInterface

一次设置多个视图数据:

```
$view->setData(['name' => 'George', 'position' => 'Boss']);
```

支持的转义上下文:html、css、js、url 或 attr 或 raw。如果是 'raw', 将不进行转义。

每次调用都会向对象累积的数据数组添加数据, 直到渲染视图为止。

setVar(*$name*[, *$value* = *null*[, *$context* = *null*]])

参数

- **$name** (string) –视图数据变量的名称
- **$value** (mixed) –此视图数据的值
- **$context** (string) –用于数据转义的上下文

返回

渲染器, 用于方法链

返回类型

CodeIgniter\View\RendererInterface

设置单个视图数据:

```
$view->setVar('name', 'Joe', 'html');
```

支持的转义上下文: html、css、js、url、attr 或 raw。如果是 'raw', 将不进行转义。

如果你使用先前已对此对象使用过的视图数据变量, 新值将替换现有值。

5.2.3 视图布局

- 创建布局
- 在视图中使用布局
- 渲染视图
- 包含视图局部

CodeIgniter 支持一个简单且非常灵活的布局系统, 可以轻松地在整个应用程序中使用一个或多个基本页面布局。布局支持可以从任何被渲染的视图中插入的内容部分。你可以

创建不同的布局以支持单列、双列、博客存档页面等。布局从不直接渲染。相反, 你渲染一个视图, 它指定了它想要扩展的布局。

创建布局

布局与任何其他视图一样。唯一的区别在于它们的预期用途。布局是唯一会使用 `renderSection()` 方法的视图文件。此方法充当内容的占位符。

例如 **app/Views/default.php**:

```
<!doctype html>
<html>
<head>
    <title>我的布局</title>
</head>
<body>
    <?= $this->renderSection('content') ?>
</body>
</html>
```

`renderSection()` 方法有两个参数: `$sectionName` 和 `$saveData`。`$sectionName` 是任何子视图用来命名内容部分的部分名称。如果布尔参数 `$saveData` 设置为 true，该方法会保存数据以供后续调用使用。否则，该方法在显示内容后会清除数据。

例如 **app/Views/welcome_message.php**:

```
<!doctype html>
<html>
<head>
    <title><?= $this->renderSection('page_title', true) ?></title>
</head>
<body>
    <h1><?= $this->renderSection('page_title') ?><h1>
    <p><?= $this->renderSection('content') ?></p>
</body>
</html>
```

备注: `$saveData` 可以在 v4.4.0 版本之后使用。

在视图中使用布局

当一个视图想要插入布局时, 它必须在文件顶部使用 `extend()` 方法:

```
<?= $this->extend('default') ?>
```

`extend()` 方法获取希望使用的任何视图文件的名称。由于它们是标准视图, 定位它们的方式与定位视图相同。默认情况下, 它将在应用程序的视图目录中查找, 但也会扫描其他 PSR-4 定义的命名空间。你可以包含命名空间以在特定命名空间的视图目录中查找视图:

```
<?= $this->extend('Blog\Views\default') ?>
```

扩展布局的视图中的所有内容必须包含在 `section()` 和 `endSection()` 方法调用中。这些调用之间的任何内容都将被插入布局中 `renderSection($name)` 调用与部分名称匹配的位置。

例如 **app/Views/some_view.php**:

```
<?= $this->extend('default') ?>

<?= $this->section('content') ?>
    <h1>Hello World!</h1>
<?= $this->endSection() ?>
```

`endSection()` 不需要部分名称。它会自动知道要关闭哪一个。

部分可以包含嵌套部分:

```
<?= $this->extend('default') ?>

<?= $this->section('content') ?>
    <h1>Hello World!</h1>
    <?= $this->section('javascript') ?>
       let a = 'a';
```

(续下页)

(接上页)

```
    <?= $this->endSection() ?>
<?= $this->endSection() ?>
```

渲染视图

渲染视图及其布局的方式与控制器中显示任何其他视图完全相同:

```
<?php

namespace App\Controllers;

class MyController extends BaseController
{
    public function index()
    {
        return view('some_view');
    }
}
```

它渲染了视图 **app/Views/some_view.php**, 如果它扩展了 `default`, 布局 **app/Views/default.php** 也会自动使用。渲染器智能到足以检测视图是否应该自行渲染, 还是需要布局。

包含视图局部

视图局部是不扩展任何布局的视图文件。它们通常包含可以在视图之间重用的内容。当使用视图布局时, 你必须使用 `$this->include()` 来包含任何视图局部。

```
<?= $this->extend('default') ?>

<?= $this->section('content') ?>
    <h1>Hello World!</h1>

    <?= $this->include('sidebar') ?>
<?= $this->endSection() ?>
```

调用 `include()` 方法时, 你可以传递渲染普通视图时可以传递的所有相同选项, 包括缓

存指令等。

5.2.4 视图单元

许多应用程序都有一些小的视图片段，可以在页面之间重复使用，或者在页面的不同位置使用。这些通常是帮助框、导航控件、广告、登录表单等。CodeIgniter 允许你将这些呈现块的逻辑封装在视图单元中。它们基本上是可以包含在其他视图中的小视图。它们可以内置逻辑来处理任何特定于单元的显示逻辑。它们可以通过将每个单元的逻辑分离到自己的类中，使你的视图更易读和可维护。

- 简单和受控制的单元
- 调用视图单元
  - 省略命名空间
  - 将参数作为键/值字符串传递
- 简单单元
- 受控单元
  - 创建受控单元
  - 通过命令生成单元
  - 使用不同的视图
  - 自定义渲染
  - 计算属性
  - 演示方法
  - 执行设置逻辑
- 单元缓存

简单和受控制的单元

CodeIgniter 支持两种类型的视图单元：简单的和受控制的。

简单的视图单元可以从你选择的任何类和方法生成，不必遵循任何规则，只需返回一个字符串。

受控制的视图单元必须从扩展了 `Codeigniter\View\Cells\Cell` 类的类生成，这提供了额外的功能，使你的视图单元更加灵活和快速使用。

调用视图单元

无论你使用哪种类型的视图单元，都可以使用 `view_cell()` 辅助函数从任何视图中调用它。

第一个参数是（1）*类和方法的名称*（简单单元）或（2）*类的名称和可选方法*（受控制的单元），第二个参数是要传递给该方法的参数数组或字符串：

```
// In a View.

// Simple Cell
<?= view_cell('App\Cells\MyClass::myMethod', ['param1' => 'value1',
↪'param2' => 'value2']) ?>

// Controlled Cell
<?= view_cell('App\Cells\MyCell', ['param1' => 'value1', 'param2' =>
↪ 'value2']) ?>
```

单元返回的字符串将被插入到调用 `view_cell()` 函数的视图中。

省略命名空间

在 4.3.0 版本加入.

如果你没有包含类的完整命名空间，它将假定可以在 `App\Cells` 命名空间中找到。因此，以下示例将尝试在 **app/Cells/MyClass.php** 中查找 `MyClass` 类。如果在那里找不到，将扫描所有命名空间，直到找到为止，在每个命名空间的 **Cells** 子目录中搜索：

```
// In a View.
<?= view_cell('MyClass::myMethod', ['param1' => 'value1', 'param2'␣
↪=> 'value2']) ?>
```

将参数作为键/值字符串传递

你还可以将参数作为键/值字符串传递：

```
// In a View.
<?= view_cell('MyClass::myMethod', 'param1=value1, param2=value2') ?
↪>
```

简单单元

简单单元是从所选方法返回字符串的类。一个简单的警告消息单元的示例可能如下所示：

```
<?php

namespace App\Cells;

class AlertMessage
{
    public function show(array $params): string
    {
        return "<div class=\"alert alert-{$params['type']}\">{
↪$params['message']}</div>";
    }
}
```

你可以在视图中这样调用它：

```
// In a View.
<?= view_cell('AlertMessage::show', ['type' => 'success', 'message'␣
↪=> 'The user has been updated.']) ?>
```

此外，你可以使用与方法中的参数变量匹配的参数名称以提高可读性。当你以这种方式使用时，视图单元调用中必须始终指定所有参数：

```
// In a View.
<?= view_cell('Blog::recentPosts', 'category=codeigniter, limit=5')␣
↪?>
```

```
<?php

// In a Cell.

namespace App\Cells;

class Blog
{
    // ...

    public function recentPosts(string $category, int $limit):␣
↪string
    {
        $posts = $this->blogModel->where('category', $category)
            ->orderBy('published_on', 'desc')
            ->limit($limit)
            ->get();

        return view('recentPosts', ['posts' => $posts]);
    }
}
```

受控单元

在 4.3.0 版本加入.

受控单元有两个主要目标：(1) 尽可能快地构建单元，(2) 并为视图提供额外的逻辑和灵活性（如果需要）。

该类必须扩展 `CodeIgniter\View\Cells\Cell`。它们应该在同一文件夹中有一个视图文件。按照惯例，类名应为 PascalCase，后缀为 `Cell`，视图应为类名的 snake_case 版本，不包括后缀。例如，如果你有一个 `MyCell` 类，视图文件应为 `my.php`。

备注： 在 v4.3.5 之前，生成的视图文件以 `_cell.php` 结尾。尽管 v4.3.5 及更高版本将

生成不带 _cell 后缀的视图文件，但现有的视图文件仍将被定位和加载。

创建受控单元

在类中实现的最基本的级别上，你只需要实现公共属性。这些属性将自动提供给视图文件。

将上面的 AlertMessage 实现为受控单元将如下所示：

```
<?php

// app/Cells/AlertMessageCell.php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class AlertMessageCell extends Cell
{
    public $type;
    public $message;
}
```

```
// app/Cells/alert_message.php
<div class="alert alert-<?= esc($type, 'attr') ?>">
    <?= esc($message) ?>
</div>
```

```
// Called in main View:
<?= view_cell('AlertMessageCell', 'type=warning, message=Failed.') ?
↪>
```

备注： 如果你使用类型化属性，你必须设置初始值：

```
<?php
```

(续下页)

(接上页)

```
// app/Cells/AlertMessageCell.php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class AlertMessageCell extends Cell
{
    public string $type    = '';
    public string $message = '';
}
```

通过命令生成单元

你还可以通过 CLI 中的内置命令创建受控单元。该命令是 `php spark make:cell`。它接受一个参数，要创建的单元的名称。名称应为 PascalCase，类将在 **app/Cells** 目录中创建。视图文件也将在 **app/Cells** 目录中创建。

```
php spark make:cell AlertMessageCell
```

使用不同的视图

你可以通过在类中设置 `view` 属性来指定自定义视图名称。视图将像正常情况下一样被定位：

```
<?php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class AlertMessageCell extends Cell
{
    public $type;
    public $message;
```

(续下页)

(接上页)

```
    protected string $view = 'my/custom/view';
}
```

自定义渲染

如果你需要更多控制 HTML 的渲染过程，可以实现一个 `render()` 方法。该方法允许你执行其他逻辑并向视图传递额外的数据（如果需要）。`render()` 方法必须返回一个字符串。

为了充分利用受控单元的全部功能，你应该使用 `$this->view()` 而不是普通的 `view()` 辅助函数：

```
<?php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class AlertMessageCell extends Cell
{
    public $type;
    public $message;

    public function render(): string
    {
        return $this->view('my/custom/view', ['extra' => 'data']);
    }
}
```

计算属性

如果你需要为一个或多个属性执行其他逻辑，可以使用计算属性。这需要将属性设置为 protected 或 private，并实现一个公共方法，该方法的名称由属性名称包围 get 和 Property 组成：

```
// In a View. Initialize the protected properties.
<?= view_cell('AlertMessageCell', ['type' => 'note', 'message' =>
↪'test']) ?>
```

```
<?php

// app/Cells/AlertMessageCell.php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class AlertMessageCell extends Cell
{
    protected $type;
    protected $message;
    private $computed;

    public function mount(): void
    {
        $this->computed = sprintf('%s - %s', $this->type, $this->
↪message);
    }

    public function getComputedProperty(): string
    {
        return $this->computed;
    }

    public function getTypeProperty(): string
    {
        return $this->type;
```

(续下页)

(接上页)

```
    }

    public function getMessageProperty(): string
    {
        return $this->message;
    }
}
```

```
// app/Cells/alert_message.php
<div>
    <p>type - <?= esc($type) ?></p>
    <p>message - <?= esc($message) ?></p>
    <p>computed: <?= esc($computed) ?></p>
</div>
```

重要: 无法设置在单元初始化期间声明为私有的属性。

演示方法

有时你需要为视图执行其他逻辑，但不想将其作为参数传递。你可以实现一个在单元的视图内部调用的方法。这可以提高视图的可读性：

```
<?php

// app/Cells/RecentPostsCell.php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class RecentPostsCell extends Cell
{
    protected $posts;

    public function linkPost($post): string
```

(续下页)

(接上页)

```
    {
        return anchor('posts/' . $post->id, $post->title);
    }
}
```

```
// app/Cells/recent_posts.php
<ul>
    <?php foreach ($posts as $post): ?>
        <li><?= $this->linkPost($post) ?></li>
    <?php endforeach ?>
</ul>
```

执行设置逻辑

如果你需要在渲染视图之前执行其他逻辑，可以实现一个 mount() 方法。该方法将在类实例化后立即调用，并可用于设置其他属性或执行其他逻辑：

```
<?php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class RecentPostsCell extends Cell
{
    protected $posts;

    public function mount(): void
    {
        $this->posts = model('PostModel')->orderBy('created_at',
↪'DESC')->findAll(10);
    }
}
```

你可以通过将它们作为数组传递给 view_cell() 辅助函数来将其他参数传递给 mount() 方法。任何与 mount() 方法的参数名称匹配的参数将被传递进去：

```php
<?php

// app/Cells/RecentPostsCell.php

namespace App\Cells;

use CodeIgniter\View\Cells\Cell;

class RecentPostsCell extends Cell
{
    protected $posts;

    public function mount(?int $categoryId): void
    {
        $this->posts = model('PostModel')
            ->when(
                $categoryId,
                static fn ($query, $categoryId) => $query->where(
↪'category_id', $categoryId),
            )
            ->orderBy('created_at', 'DESC')
            ->findAll(10);
    }
}
```

```php
// Called in main View:
<?= view_cell('RecentPostsCell', ['categoryId' => 5]) ?>
```

单元缓存

你可以通过将要缓存数据的秒数作为第三个参数传递来缓存视图单元调用的结果。这将使用当前配置的缓存引擎：

```php
// Cache the view for 5 minutes
<?= view_cell('App\Cells\Blog::recentPosts', 'limit=5', 300) ?>
```

如果需要，你可以提供一个自定义名称，而不是自动生成的名称，通过将新名称作为第四个参数传递：

```
// Cache the view for 5 minutes
<?= view_cell('App\Cells\Blog::recentPosts', 'limit=5', 300,
↪'newcacheid') ?>
```

5.2.5 视图解析器

- 使用视图解析器类
  - 它的工作原理
  - 解析器模板
  - 解析器配置选项
- 替换变体
  - 循环替换
  - 嵌套替换
  - 注释
  - 数据级联
  - 阻止解析
  - 条件逻辑
  - 转义数据
  - 过滤器
  - 解析器插件
- 使用说明
  - 视图片段
- 类参考

视图解析器可以对视图文件中的伪变量进行简单的文本替换。它可以解析简单变量或变量标签对。

伪变量名称或控制结构用大括号括起来, 像这样:

```
<html>
<head>
    <title>{blog_title}</title>
</head>
<body>
    <h3>{blog_heading}</h3>

    {blog_entries}
        <h5>{title}</h5>
        <p>{body}</p>
    {/blog_entries}

</body>
</html>
```

这些变量不是实际的 PHP 变量, 而是普通文本表示, 允许你从模板 (视图文件) 中消除 PHP。

备注: 由于在视图页面中使用纯 PHP(例如使用视图渲染器) 可以让它们运行得稍快一点,CodeIgniter **不要求**你使用这个类。然而, 一些开发人员更喜欢在与设计师合作时使用某种模板引擎, 因为他们觉得设计师在使用 PHP 时会感到困惑。

使用视图解析器类

通过其服务加载解析器类的最简单方法是:

```
<?php

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

另外, 如果你没有使用 `Parser` 类作为默认渲染器, 你可以直接实例化它:

```
<?php

$parser = new \CodeIgniter\View\Parser();
```

然后你可以使用它提供的三种标准渲染方法中的任何一种:`render()`、`setVar()` 和

setData()。你还可以通过 setDelimiters() 方法直接指定分隔符。

重要: 使用 Parser, 你的视图模板只由 Parser 本身处理, 而不是作为常规视图 PHP 脚本。这样的脚本中的 PHP 代码会被解析器忽略, 只执行替换。

这是有意为之的: 不包含 PHP 的视图文件。

它的工作原理

Parser 类处理存储在应用程序的视图路径中的 "PHP/HTML 脚本"。这些脚本不能包含任何 PHP。

每个视图参数 (我们称之为伪变量) 都会触发一次替换, 基于你为它提供的值的类型。伪变量不会提取到 PHP 变量中; 相反, 它们的值是通过伪变量语法访问的, 其名称引用在大括号内。

Parser 类在内部使用关联数组来累积伪变量设置, 直到你调用 render()。这意味着你的伪变量名称需要是唯一的, 否则后面的参数设置将覆盖较早的设置。

这也会影响根据脚本中的不同上下文对参数值进行转义。你将必须为每个转义值提供一个唯一的参数名称。

解析器模板

你可以使用 render() 方法来解析 (或渲染) 简单模板, 像这样:

```
<?php

$data = [
    'blog_title'   => 'My Blog Title',
    'blog_heading' => 'My Blog Heading',
];

return $parser->setData($data)->render('blog_template');
```

视图参数通过关联数组的形式传递给 setData()，用于在模板中替换数据。

在上面的例子中，模板将包含两个变量：{blog_title} 和 {blog_heading}

render() 的第一个参数包含模板的名称，其中 blog_template 是你的视图模板文件的名称。

重要: 如果省略了文件扩展名, 则视图预计以.php 扩展名结束。

解析器配置选项

可以将几个选项传递给 render() 或 renderString() 方法。

- cache - 以秒为单位, 保存视图结果的时间; 对 renderString() 忽略
- cache_name - 用于保存/检索缓存视图结果的 ID; 默认为视图路径; 对 renderString() 忽略
- saveData - 如果为 true, 视图数据参数应保留以供随后的调用; 默认为 **true**
- cascadeData - 如果嵌套或循环替换发生时, 数据对是否应该传播给内部替换; 默认为 **true**

```
<?php

return $parser->render('blog_template', [
    'cache'      => HOUR,
    'cache_name' => 'something_unique',
]);
```

替换变体

支持三种替换类型: 简单、循环和嵌套。替换的执行顺序与添加伪变量的顺序相同。

解析器执行的 **简单替换**是一对一地替换伪变量, 其中相应的数据参数具有标量或字符串值, 如本例所示:

```
<?php

$template = '<head><title>{blog_title}</title></head>';
$data     = ['blog_title' => 'My ramblings'];
```

(续下页)

(接上页)

```
return $parser->setData($data)->renderString($template);
// Result: <head><title>My ramblings</title></head>
```

解析器通过“变量对”大大扩展了替换, 用于嵌套替换或循环, 以及一些用于条件替换的高级结构。

当解析器执行时, 它通常会

- 处理任何条件替换
- 处理任何嵌套/循环替换
- 处理其余的单个替换

循环替换

当伪变量的值是数组的序列化数组时, 就会发生循环替换, 例如数据库记录的数组。

上面的示例代码允许简单变量被替换。如果你想让整个变量块重复, 每个迭代都包含新值呢? 考虑我们在页面顶部显示的模板示例:

```
<html>
<head>
    <title>{blog_title}</title>
</head>
<body>
    <h3>{blog_heading}</h3>

    {blog_entries}
        <h5>{title}</h5>
        <p>{body}</p>
    {/blog_entries}

</body>
</html>
```

在上面的代码中, 你会注意到一对变量:`{blog_entries}` 数据… `{/blog_entries}`。在这种情况下, 这对变量之间的数据的整个块将重复多次, 对应于参数数组中的“blog_entries”元素的行数。

解析变量对使用与解析单个变量完全相同的代码, 不同之处在于, 你需要添加一个与变量对数据对应的多维数组。考虑这个例子:

```
<?php

$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'],
    ],
];

return $parser->setData($data)->render('blog_template');
```

伪变量 `blog_entries` 的值是一个关联数组的顺序数组。外部级别与嵌套的“行”没有关联的键。

如果你的“pair”数据来自数据库结果, 它已经是一个多维数组, 你可以简单地使用数据库的 `getResultArray()` 方法:

```
<?php

$query = $db->query('SELECT * FROM blog');

$data = [
    'blog_title'   => 'My Blog Title',
    'blog_heading' => 'My Blog Heading',
    'blog_entries' => $query->getResultArray(),
];

return $parser->setData($data)->render('blog_template');
```

如果要循环的数组包含对象而不是数组, 解析器将首先在对象上查找 `asArray()` 方法。如果存在, 则调用该方法并像上面描述的那样循环结果数组。如果没有 `asArray()` 方法, 对象将转换为数组, 它的公共属性将可用于解析器。

当与实体类一起使用时, 这尤其有用, 因为它有一个 `asArray()` 方法, 该方法返回所有公共和受保护的属性 (减去 _options 属性), 并使它们可用于解析器。

嵌套替换

当伪变量的值是关联数组时, 就会发生嵌套替换, 例如来自数据库的记录:

```
<?php

$data = [
    'blog_title'   => 'My Blog Title',
    'blog_heading' => 'My Blog Heading',
    'blog_entries' => [
        [
            'title' => 'Title 1',
            'body'  => 'Body 1',
        ],
    ],
];

return $parser->setData($data)->render('blog_template');
```

伪变量 `blog_entries` 的值是一个关联数组。在它内部定义的键/值对将在该变量对循环内为该变量公开。

可能适用于上述内容的 **blog_template.php**

```
<h1>{blog_title} - {blog_heading}</h1>
{blog_entries}
    <div>
        <h2>{title}</h2>
        <p>{body}</p>
    </div>
{/blog_entries}
```

如果希望 `blog_entries` 作用域内的其他伪变量可用, 请确保 `cascadeData` 选项设置为 true。

注释

你可以在模板中用 {# #} 符号将注释括起来, 它们在解析期间将被忽略并删除。

```
{# 这个注释在解析过程中会被删除。#}
{blog_entry}
    <div>
        <h2>{title}</h2>
        <p>{body}</p>
    </div>
{/blog_entry}
```

数据级联

对于嵌套替换和循环替换, 你可以选择将数据对级联到内部替换。

以下示例不受级联的影响:

```
<?php

$template = '{name} lives in {locations}{city} on {planet}{/
↪locations}.';

$data = [
    'name'      => 'George',
    'locations' => [
        ['city' => 'Red City', 'planet' => 'Mars'],
    ],
];

return $parser->setData($data)->renderString($template);
// Result: George lives in Red City on Mars.
```

这个例子的结果与级联的不同:

```
<?php

$template = '{locations}{name} lives in {city} on {planet}{/
↪locations}.';
```

(续下页)

(接上页)

```
$data = [
    'name'      => 'George',
    'locations' => [
        ['city' => 'Red City', 'planet' => 'Mars'],
    ],
];

return $parser->setData($data)->renderString($template, [
↪'cascadeData' => false]);
// Result: {name} lives in Red City on Mars.

// or

return $parser->setData($data)->renderString($template, [
↪'cascadeData' => true]);
// Result: George lives in Red City on Mars.
```

阻止解析

你可以使用 `{noparse}` `{/noparse}` 标签对指定不要解析的页面部分。这对括号之间的任何内容都将完全保持原样, 不会发生变量替换、循环等。

```
{noparse}
    <h1>Untouched Code</h1>
{/noparse}
```

条件逻辑

解析器类支持一些基本条件来处理 `if`、`else` 和 `elseif` 语法。所有 `if` 块必须用 `endif` 标签关闭:

```
{if $role=='admin'}
    <h1>Welcome, Admin!</h1>
{endif}
```

这简单的块在解析期间转换为以下内容:

```
<?php if ($role === 'admin'): ?>
    <h1>Welcome, Admin!</h1>
<?php endif ?>
```

if 语句中使用的所有变量必须先以相同的名称设置过。除此之外, 它的处理方式与标准 PHP 条件完全相同, 这里也将应用所有标准 PHP 规则。你可以使用通常会用到的任何比较运算符, 如 ==、===、!==、<、> 等。

```
{if $role=='admin'}
    <h1>Welcome, Admin</h1>
{elseif $role=='moderator'}
    <h1>Welcome, Moderator</h1>
{else}
    <h1>Welcome, User</h1>
{endif}
```

> **警告:** 在后台, 条件语句使用 `eval()` 进行解析, 所以你必须确保在条件语句中使用的数据来自可信来源, 否则可能会面临安全风险。

更改条件分隔符

如果你的模板中有像下面的 JavaScript 代码, 解析器会抛出语法错误, 因为存在可以解释为条件的字符串:

```
<script type="text/javascript">
    var f = function() {
        if (hasAlert) {
            alert('{message}');
        }
    }
</script>
```

在这种情况下, 你可以使用 `setConditionalDelimiters()` 方法更改条件分隔符, 以避免误解:

```
<?php

$parser->setConditionalDelimiters('{%', '%}');
```

在这种情况下, 你可以在模板中编写代码:

```
{% if $role=='admin' %}
    <h1>Welcome, Admin</h1>
{% else %}
    <h1>Welcome, User</h1>
{% endif %}
```

转义数据

默认情况下, 所有变量替换都被转义, 以帮助防止页面上的 XSS 攻击。CodeIgniter 的 `esc()` 方法支持几种不同的上下文, 如常规的 `html`、HTML `attr` 中的、`css` 中的、`js` 中的等。如果没有指定其他内容, 数据将假定在 HTML 上下文中。你可以使用 `esc()` 过滤器指定所使用的上下文:

```
{ user_styles | esc(css) }
<a href="{ user_link | esc(attr) }">{ title }</a>
```

有时你绝对需要使用而不转义的数据。你可以在打开和关闭括号中添加感叹号来实现这一点:

```
{! unescaped_var !}
```

过滤器

可以对单个变量替换应用一个或多个过滤器以修改其呈现方式。这些旨在修改输出, 而不是大幅改变它。上面讨论的 `esc` 过滤器就是一个例子。日期是另一个常见的用例, 其中你可能需要以页面上的几个部分不同的方式格式化同一数据。

过滤器是在伪变量名称之后、用竖线符号 `|` 分隔的命令:

```
// 显示 -55 为 55
{ value|abs }
```

如果参数需要任何参数, 必须用逗号分隔并用括号括起来:

```
{ created_at|date(Y-m-d) }
```

可以通过管道连接多个过滤器来应用值。它们从左到右处理:

```
{ created_at|date_modify(+5 days)|date(Y-m-d) }
```

提供的过滤器

使用解析器时, 可用以下过滤器:

过滤器	参数	描述	示例
abs		显示数字的绝对值。	{ v|abs }
capitalize		以句子大小写显示字符串: 全部小写, 第一个字母大写。	{ v|capitalize}
date	格式 (Y-m-d)	与 PHP **date** 兼容的格式化字符串。	{ v|date(Y-m-d) }
date_modify	要添加/减去的值	与 **strtotime** 兼容的字符串, 用于修改日期, 如 `+5 day` 或 `-1 week`。	{ v|date_modify(+1 day) }
default	默认值	如果变量为 empty(), 显示默认值。	{ v|default(just in case) }
esc	html、attr、css、js	指定转义数据的上下文。	{ v|esc(attr) }
excerpt	短语、半径词数	返回给定短语半径词数内的文本。与 **excerpt** 辅助函数相同。	{ v|excerpt(green giant, 20) }
highlight	短语	使用‘<mark></mark>’标记在文本中突出显示给定短语。	{ v|highlight(view parser) }
highlight_code		使用 HTML/CSS 突出显示代码示例。	{ v|highlight_code }
limit_chars	限制个数	将字符数限制为 $limit。	{ v|limit_chars(100) }
limit_words	限制个数	将词数限制为 $limit。	{ v|limit_words(20) }
local_currency	货币、区域设置、小数位数	显示货币的本地化版本。“货币”值是任何 3 字节 ISO 4217 货币代码。	{ v|local_currency(EUR,en_US) }
local_number	类型、精度、区域设置	显示数字的本地化版本。“类型”可以是:decimal、currency、percent、scientific、spellout、ordinal、duration 之一。	{ v|local_number(decimal,2,en_US) }
lower		转换字符串为小写。	{ v|lower }
nl2br		用 HTML 标签替换所有换行符 (n)。	{ v|nl2br }
number_format	小数位数	封装 PHP **number_format** 函数以在解析器中使用。	{ v|number_format(3) }

有关与“local_number”过滤器相关的详细信息，请参阅 PHP 的 NumberFormatter。

自定义过滤器

你可以通过编辑 **app/Config/View.php** 并向 `$filters` 数组中添加新条目来轻松创建自己的过滤器。每个键是视图中调用过滤器的名称，其值是任何有效的 PHP 可调用对象：

```
<?php

namespace Config;

use CodeIgniter\Config\View as BaseView;

class View extends BaseView
{
    public $filters = [
        'foo'        => '\Some\Class::methodName',
        'str_repeat' => 'str_repeat', // native php function
    ];

    // ...
}
```

解析器插件

插件允许你扩展解析器, 为每个项目添加自定义功能。它们可以是任何 PHP 可调用的, 因此实现起来非常简单。在模板中, 插件由 `{+ +}` 标记指定:

```
{+ foo +} inner content {+ /foo +}
```

这个示例显示了一个名为 **foo** 的插件。它可以操作在其打开和关闭标记之间的任何内容。在这个例子中, 它可以使用文本 “inner content”。插件在任何伪变量替换发生之前进行处理。

虽然插件通常由标签对组成, 如上所示, 但它们也可以是一个单标签, 没有闭合标签:

```
{+ foo +}
```

打开标签也可以包含可自定义插件工作方式的参数。参数表示为键/值对:

```
{+ foo bar=2 baz="x y" +}
```

参数也可以是单个值:

```
{+ include somefile.php +}
```

提供的插件

使用解析器时, 可用以下插件:

插件	参数	描述	示例
current_url		current_url 辅助函数的别名。	{+ current_url +}
previous_url		previous_url 辅助函数的别名。	{+ previous_url +}
siteURL	"login"	site_url 辅助函数的别名。	{+ siteURL "login" +}
mailto	email、标题、属性	mailto 辅助函数的别名。	{+ mailto email=foo@example.com title="Stranger Things" +}
safe_mailto	email、标题、属性	safe_mailto 辅助函数的别名。	{+ safe_mailto email=foo@example.com title="Stranger Things" +}
lang	语言字符串	lang 辅助函数的别名。	{+ lang number.terabyteAbbr +}
validation_errors	字段名 (可选)	返回字段的错误字符串 (如果指定),	{+ validation_errors +} , {+ validation_errors field="email" +}
route	route 名称	route_to 辅助函数的别名。	{+ route "login" +}
csp_script_nonce		csp_script_nonce 辅助函数的别名。	{+ csp_script_nonce +}
csp_style_nonce		csp_style_nonce 辅助函数的别名。	{+ csp_style_nonce +}

注册插件

最简单的方法是将新插件添加到 **app/Config/View.php** 中的 `$plugins` 数组, 即可注册并准备使用新插件。键是模板文件中使用的插件名称。值是任何有效的 PHP 可调用项, 包括静态类方法:

```
<?php

namespace Config;

use CodeIgniter\Config\View as BaseView;

class View extends BaseView
{
    public $plugins = [
        'foo' => '\Some\Class::methodName',
    ];

    // ...
}
```

你也可以使用闭包, 但只能在配置文件的构造函数中定义它们:

```
<?php

namespace Config;

use CodeIgniter\Config\View as BaseView;

class View extends BaseView
{
    public $plugins = [];

    public function __construct()
    {
        $this->plugins['bar'] = static fn (array $params = []) =>
↪$params[0] ?? '';

        parent::__construct();
```

(续下页)

(接上页)

```
    }

    // ...
}
```

如果可调用的独立存在, 则将其视为单标签, 而不是打开/关闭标签。它将被插件的返回值替换:

```
<?php

namespace Config;

use CodeIgniter\Config\View as BaseView;

class View extends BaseView
{
    public $plugins = [
        'foo' => '\Some\Class::methodName',
    ];

    // ...
}

/*
 * Tag is replaced by the return value of Some\Class::methodName()␣
↪static function.
 * {+ foo +}
 */
```

如果可调用项包含在数组中, 则将其视为打开/关闭标签对, 可以操作其标记之间的任何内容:

```
<?php

namespace Config;

use CodeIgniter\Config\View as BaseView;
```

(续下页)

(接上页)

```
class View extends BaseView
{
    public $plugins = [
        'foo' => ['\Some\Class::methodName'],
    ];

    // ...
}

// {+ foo +} inner content {+ /foo +}
```

使用说明

如果包含了模板中未引用的替换参数, 会被忽略:

```
<?php

$template = 'Hello, {firstname} {lastname}';
$data     = [
    'title'     => 'Mr',
    'firstname' => 'John',
    'lastname'  => 'Doe',
];

return $parser->setData($data)->renderString($template);
// Result: Hello, John Doe
```

如果不包含模板中引用的替换参数, 将显示原始伪变量:

```
<?php

$template = 'Hello, {firstname} {initials} {lastname}';
$data     = [
    'title'     => 'Mr',
    'firstname' => 'John',
    'lastname'  => 'Doe',
];
```

(续下页)

(接上页)

```
return $parser->setData($data)->renderString($template);
// Result: Hello, John {initials} Doe
```

如果为应该是数组的变量对提供字符串替换参数, 即用于变量对, 则仅为开始变量对标记执行替换, 但不正确渲染结束变量对标记:

```
<?php

$template = 'Hello, {firstname} {lastname} ({degrees}{degree} {/
↪degrees})';
$data     = [
    'degrees'   => 'Mr',
    'firstname' => 'John',
    'lastname'  => 'Doe',
    'titles'    => [
        ['degree' => 'BSc'],
        ['degree' => 'PhD'],
    ],
];

return $parser->setData($data)->renderString($template);
// Result: Hello, John Doe (Mr{degree} {/degrees})
```

视图片段

你不必使用变量对在视图中实现迭代。可以使用视图片段代替变量对内部的内容, 并在控制器中控制迭代。

在视图中控制迭代的示例:

```
$template = '<ul>{menuitems}
    <li><a href="{link}">{title}</a></li>
{/menuitems}</ul>';

$data = [
    'menuitems' => [
```

(续下页)

(接上页)

```
        ['title' => 'First Link', 'link' => '/first'],
        ['title' => 'Second Link', 'link' => '/second'],
    ]
];

return $parser->setData($data)->renderString($template);
```

结果:

```
<ul>
    <li><a href="/first">First Link</a></li>
    <li><a href="/second">Second Link</a></li>
</ul>
```

在控制器中控制迭代、使用视图片段的示例:

```
<?php

$temp      = '';
$template1 = '<li><a href="{link}">{title}</a></li>';
$data1     = [
    ['title' => 'First Link', 'link' => '/first'],
    ['title' => 'Second Link', 'link' => '/second'],
];

foreach ($data1 as $menuItem) {
    $temp .= $parser->setData($menuItem)->renderString($template1);
}

$template2 = '<ul>{menuitems}</ul>';
$data      = [
    'menuitems' => $temp,
];

return $parser->setData($data)->renderString($template2);
```

结果:

```
<ul>
    <li><a href="/first">First Link</a></li>
    <li><a href="/second">Second Link</a></li>
</ul>
```

类参考

class CodeIgniter\View**Parser**

render(*$view*[, *$options*[, *$saveData*]])

参数

- **$view** (string) –视图源文件的名称
- **$options** (array) –选项的键/值对数组
- **$saveData** (boolean) –如果为 true, 将保存数据供随后调用, 如果为 false, 在渲染视图后清除数据

返回

所选视图的渲染文本

返回类型

string

根据文件名和已设置的任何数据构建输出:

```
<?php

return $parser->render('myview');
```

支持的选项:

- cache - 以秒为单位, 保存视图结果的时间
- cache_name - 用于保存/检索缓存视图结果的 ID; 默认为视图路径
- cascadeData - 嵌套或循环替换发生时, 当前生效的数据对是否应传播
- saveData - 视图数据参数是否应保留以供后续调用

首先执行任何条件替换, 然后对每个数据对执行其余替换。

renderString(*$template*[, *$options*[, *$saveData*]])

参数

- **$template** (string) –作为字符串提供的视图源
- **$options** (array) –选项的键/值对数组
- **$saveData** (boolean) –如果为 true, 将保存数据供随后调用, 如果为 false, 在渲染视图后清除数据

返回

所选视图的渲染文本

返回类型

string

根据提供的模板源和已设置的任何数据构建输出:

```
<?php

return $parser->renderString('<ul><li>Item 1</li><li>Item 2
↪</li></ul>');
```

支持的选项和行为与上述相同。

setData([*$data*[, *$context = null*]])

参数

- **$data** (array) –视图数据字符串的关联数组, 作为键/值对
- **$context** (string) –用于数据转义的上下文

返回

渲染器, 用于方法链

返回类型

CodeIgniter\View\RendererInterface

一次设置多个视图数据:

```
<?php

$parser->setData(['name' => 'George', 'position' => 'Boss
↪']);
```

支持的转义上下文:html、css、js、url 或 attr 或 raw。如果是‘raw’,将不进行转义。

setVar(*$name*[, *$value* = *null*[, *$context* = *null*]])

参数

- **$name** (string) –视图数据变量的名称
- **$value** (mixed) –此视图数据的值
- **$context** (string) –用于数据转义的上下文

返回

渲染器, 用于方法链

返回类型

CodeIgniter\View\RendererInterface

设置单个视图数据:

```
<?php

$parser->setVar('name', 'Joe', 'html');
```

支持的转义上下文:html、css、js、url、attr 或 raw。如果是‘raw’,将不进行转义。

setDelimiters(*$leftDelimiter* = *'{'*, *$rightDelimiter* = *'}'*)

参数

- **$leftDelimiter** (string) –替换字段的左分隔符
- **$rightDelimiter** (string) –替换字段的右分隔符

返回

渲染器, 用于方法链

返回类型

CodeIgniter\View\RendererInterface

覆盖替换字段分隔符:

```
<?php

$parser->setDelimiters('[', ']');
```

setConditionalDelimiters(*$leftDelimiter = '{', $rightDelimiter = '}'*)

参数

- **$leftDelimiter** (string) –条件的左分隔符
- **$rightDelimiter** (string) –条件的右分隔符

返回

渲染器, 用于方法链

返回类型

CodeIgniter\View\RendererInterface

覆盖条件分隔符:

```
<?php

$parser->setConditionalDelimiters('{%', '%}');
```

5.2.6 视图装饰器

- 什么是视图装饰器？
- 创建装饰器
  - 创建装饰器类
  - 注册装饰器类

什么是视图装饰器？

视图装饰器允许你的应用程序在渲染过程中修改 HTML 输出。这发生在缓存之前, 允许你将自定义功能应用于视图。

创建装饰器

创建装饰器类

创建自己的视图装饰器需要创建一个新的类，该类实现 `CodeIgniter\View\ViewDecoratorInterface`。这需要一个单一的方法，它获取生成的 HTML 字符串，对其执行任何修改，并返回结果 HTML。

```
<?php

namespace App\Views\Decorators;

use CodeIgniter\View\ViewDecoratorInterface;

class MyDecorator implements ViewDecoratorInterface
{
    public static function decorate(string $html): string
    {
        // Modify the output here

        return $html;
    }
}
```

注册装饰器类

创建后，必须在 **app/Config/View.php** 中注册该类：

```
<?php

namespace Config;

use CodeIgniter\Config\View as BaseView;

class View extends BaseView
{
    // ...
```

(续下页)

(接上页)

```
    public array $decorators = [
        'App\Views\Decorators\MyDecorator',
    ];
}
```

现在它已注册, 每渲染或解析的视图都将调用装饰器。装饰器的调用顺序与此配置设置中指定的顺序相同。

5.2.7 HTML 表格类

Table 类提供了使你能够从数组或数据库结果集自动生成 HTML 表格的方法。

- 使用 *Table* 类
  - 初始化类
  - 例子
  - 更改表格外观
  - 同步行与标题
- 类参考

使用 Table 类

初始化类

Table 类没有作为服务提供, 应该进行“正常”实例化, 例如:

```
<?php

$table = new \CodeIgniter\View\Table();
```

例子

下面是一个示例, 展示了如何从多维数组创建表格。请注意, 第一个数组索引将成为表格标题 (或者你可以使用下面函数参考中描述的 `setHeading()` 方法设置自己的标题)。

```
<?php

$table = new \CodeIgniter\View\Table();

$data = [
    ['Name', 'Color', 'Size'],
    ['Fred', 'Blue', 'Small'],
    ['Mary', 'Red', 'Large'],
    ['John', 'Green', 'Medium'],
];

echo $table->generate($data);
```

下面是从数据库查询结果创建的表格示例。表类将自动基于表名生成标题 (或者你可以使用下面类参考中描述的 `setHeading()` 方法设置自己的标题)。

```
<?php

$table = new \CodeIgniter\View\Table();

$query = $db->query('SELECT * FROM my_table');

echo $table->generate($query);
```

下面是一个使用离散参数创建表格的示例:

```
<?php

$table = new \CodeIgniter\View\Table();

$table->setHeading('Name', 'Color', 'Size');

$table->addRow('Fred', 'Blue', 'Small');
$table->addRow('Mary', 'Red', 'Large');
```

(续下页)

(接上页)

```
$table->addRow('John', 'Green', 'Medium');

echo $table->generate();
```

下面是相同的示例, 只是使用数组代替各个参数:

```
<?php

$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();
```

更改表格外观

Table 类允许你设置一个表格模板来指定布局设计。下面是模板原型:

```
<?php

$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>',
```

(续下页)

(接上页)

```
    '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>',
    'cell_alt_start' => '<td>',
    'cell_alt_end'   => '</td>',

    'table_close' => '</table>',
];

$table->setTemplate($template);
```

备注: 你会注意到模板中有两组“row”块。这允许你创建交替的行颜色或与每次迭代行数据交替的设计元素。

你不需要提交完整的模板。如果你只需要更改布局的一部分, 只需提交这些元素即可。在此示例中, 仅更改表格打开标签:

```
<?php

$template = [
    'table_open' => '<table border="1" cellpadding="2" cellspacing=
↪"1" class="mytable">',
```

(续下页)

(接上页)

```
];

$table->setTemplate($template);
```

你还可以通过向 Table 构造函数传递模板设置数组来为这些设置默认值:

```
<?php

$customSettings = [
    'table_open' => '<table border="1" cellpadding="2" cellspacing=
↪"1" class="mytable">',
];

$table = new \CodeIgniter\View\Table($customSettings);
```

同步行与标题

在 4.4.0 版本加入.

`setSyncRowsWithHeading(true)` 方法使得每个数据值都放置在与 `setHeading()` 中定义的相同列中，如果参数使用了关联数组。这在处理通过 REST API 加载的数据时特别有用，因为其顺序可能不符合你的要求，或者如果 API 返回了过多的数据。

如果数据行包含一个在标题中不存在的键，则其值将被过滤。相反，如果数据行中没有列在标题中列出的键，则会在其位置放置一个空单元格。

```
<?php

$table = new \CodeIgniter\View\Table();

$table->setHeading(['name' => 'Name', 'color' => 'Color', 'size' =>
↪'Size'])
    ->setSyncRowsWithHeading(true)
    ->addRow(['color' => 'Blue', 'name' => 'Fred', 'size' => 'Small
↪'])
    ->addRow(['size' => 'Large', 'age' => '24', 'name' => 'Mary'])
    ->addRow(['color' => 'Green']);
```

(续下页)

(接上页)

```
echo $table->generate();
?>

<!-- Generates a table with this prototype: -->
<table border="0" cellpadding="4" cellspacing="0">
    <thead>
        <tr>
            <th>Name</th>
            <th>Color</th>
            <th>Size</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Fred</td>
            <td>Blue</td>
            <td>Small</td>
        </tr>
        <tr>
            <td>Mary</td>
            <td></td>
            <td>Large</td>
        </tr>
        <tr>
            <td></td>
            <td>Green</td>
            <td></td>
        </tr>
    </tbody>
</table>
```

重要: 你必须在通过 `addRow([...])` 添加任何行之前调用 `setSyncRowsWithHeading(true)` 和 `setHeading([...])`，以进行列的重新排列。

使用数组作为 `generate()` 的输入会产生相同的结果:

```
<?php

$data = [
    [
        'color' => 'Blue',
        'name'  => 'Fred',
        'size'  => 'Small',
    ],
    [
        'size' => 'Large',
        'age'  => '24',
        'name' => 'Mary',
    ],
    [
        'color' => 'Green',
    ],
];

$table = new \CodeIgniter\View\Table();

$table->setHeading(['name' => 'Name', 'color' => 'Color', 'size' =>
↪'Size'])
    ->setSyncRowsWithHeading(true);

echo $table->generate($data);
```

类参考

class CodeIgniter\View**Table**

$function = null

允许你指定 native PHP 函数或一个有效的函数数组对象应用于所有单元格数据。

```
<?php

$table = new \CodeIgniter\View\Table();
```

(续下页)

(接上页)

```
$table->setHeading('Name', 'Color', 'Size');
$table->addRow('Fred', '<strong>Blue</strong>', 'Small');

$table->function = 'htmlspecialchars';
echo $table->generate();
```

在上面的例子中, 所有单元格数据都将通过 PHP 的 `htmlspecialchars()` 函数运行, 结果是:

```
<td>Fred</td><td>&lt;strong&gt;Blue&lt;/strong&gt;</td><td>
↪Small</td>
```

generate([*$tableData = null*])

参数

- **$tableData** (mixed) –用来填充表格行的数据

返回

HTML 表格

返回类型

string

返回包含生成表格的字符串。接受一个可选参数, 可以是数组或数据库结果对象。

setCaption(*$caption*)

参数

- **$caption** (string) –表格标题

返回

Table 实例 (方法链式调用)

返回类型

Table

允许你为表格添加标题。

```
<?php
```

(续下页)

(接上页)

```
$table->setCaption('Colors');
```

setHeading([*$args = []*[, ...]])

参数

- **$args** (mixed) –包含表格列标题的数组或多个字符串

返回

Table 实例 (方法链式调用)

返回类型

Table

允许你设置表格标题。你可以提交数组或离散参数:

```
<?php

$table->setHeading('Name', 'Color', 'Size'); // or

$table->setHeading(['Name', 'Color', 'Size']);
```

setFooting([*$args = []*[, ...]])

参数

- **$args** (mixed) –包含表格页脚值的数组或多个字符串

返回

Table 实例 (方法链式调用)

返回类型

Table

允许你设置表格页脚。你可以提交数组或离散参数:

```
<?php

$table->setFooting('Subtotal', $subtotal, $notes); // or

$table->setFooting(['Subtotal', $subtotal, $notes]);
```

addRow([*$args = []*[, ...]])

参数

- **$args** (mixed) –包含行值的数组或多个字符串

返回

Table 实例 (方法链式调用)

返回类型

Table

允许你向表格添加行。你可以提交数组或离散参数:

```
<?php

$table->addRow('Blue', 'Red', 'Green'); // or

$table->addRow(['Blue', 'Red', 'Green']);
```

如果你想为单个单元格的标签属性设置值, 可以为该单元格使用关联数组。关联键 **data** 定义单元格的数据。任何其他的 key => val 对会作为 key=' val' 属性添加到标签中:

```
<?php

$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

- **$array** (array) –包含多个行数据的数组
- **$columnLimit** (int) –表格中的列数

返回

HTML 表格列的多维数组

返回类型

array

此方法接受一维数组作为输入, 并创建深度等于所需列数的多维数组。这允许具有大量元素的单个数组以固定列数显示在表格中。考虑这个例子:

```
<?php

$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) –包含模板值的关联数组

返回

成功为 true, 失败为 false

返回类型

bool

允许你设置模板。你可以提交完整或部分模板。

```
<?php

$template = [
    'table_open' => '<table border="1" cellpadding="2"␣
↪cellspacing="1" class="mytable">',
];

$table->setTemplate($template);
```

setEmpty(*$value*)

参数

- **$value** (mixed) –放入空单元格中的值

返回

Table 实例 (方法链式调用)

返回类型

Table

允许你为使用在任何空表格单元格中的默认值设置值。例如, 你可以设置一个不间断的空格:

```
<?php

$table->setEmpty(' ');
```

clear()

返回

Table 实例 (方法链式调用)

返回类型

Table

允许你清除表格标题、行数据和标题。如果你需要显示包含不同数据的多个表格，你应该在每个表格生成后调用此方法，以清除之前的表格信息。

例子

```
<?php

$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();
```

setSyncRowsWithHeading(*bool $orderByKey*)

返回

Table 实例（方法链式调用）

返回类型

Table

启用每个行数据键按照标题键进行排序。这样可以更好地控制数据在正确列

中的显示位置。确保在调用第一个 `addRow()` 方法之前设置此值。

5.2.8 HTTP 响应

Response 类继承自*HTTP 消息类*，并添加了仅适用于服务器响应客户端请求的方法。

- 使用响应类
  - 设置输出内容
  - 设置头部
- 重定向
  - 重定向到 *URI* 路径
  - 重定向到定义的路由
  - 返回重定向
  - 带 *Cookie* 的重定向
  - 带头部的重定向
  - 重定向状态码
- 强制文件下载
  - 在浏览器中打开文件
- *HTTP* 缓存
- 类参考

使用响应类

系统会自动为你实例化一个 Response 类，并传递给你的控制器。你可以通过 `$this->response` 访问它。它与 `Services::response()` 返回的是同一个实例，我们称之为全局响应实例。

很多时候你不需要直接操作该类，因为 CodeIgniter 会自动处理头部和内容的发送。这在页面成功生成所需内容时非常方便。当出现错误、需要发送特定状态码或利用强大的 HTTP 缓存功能时，你可以直接使用该类。

设置输出内容

当需要直接设置脚本输出内容，而不依赖 CodeIgniter 自动获取时，可以使用 `setBody` 方法手动设置。通常与设置响应状态码配合使用：

```
<?php

$this->response->setStatusCode(404)->setBody($body);
```

原因短语（'OK'、'Created'、'Moved Permanently'）会自动添加，但你也可以通过 `setStatusCode()` 方法的第二个参数自定义原因短语：

```
<?php

$this->response->setStatusCode(404, 'Nope. Not here.');
```

你可以使用 `setJSON()` 和 `setXML()` 方法将数组格式化为 JSON 或 XML，并设置相应的内容类型头部。通常你会发送一个待转换的数据数组：

```
<?php

$data = [
    'success' => true,
    'id'      => 123,
];

return $this->response->setJSON($data);

// or
return $this->response->setXML($data);
```

设置头部

setHeader()

通常需要为响应设置头部。Response 类通过 `setHeader()` 方法简化了这个操作。

第一个参数是头部名称。第二个参数是值，可以是字符串或将被正确组合的值数组：

```
<?php

$this->response->setHeader('Location', 'http://example.com')
    ->setHeader('WWW-Authenticate', 'Negotiate');
```

使用这些方法代替原生 PHP 函数可以确保头部不会过早发送导致错误，并支持测试。

重要： 自 v4.6.0 起，如果使用 PHP 原生 `header()` 函数设置头部后，再通过 `Response` 类设置相同头部，前者将被覆盖。

备注： 此方法仅将头部设置到响应实例。因此，如果创建并返回另一个响应实例（例如调用 *redirect()*），此处设置的头部不会自动发送。

---

appendHeader()

如果头部已存在且允许多个值，可以使用 `appendHeader()` 和 `prependHeader()` 方法分别在值列表末尾或开头添加值。第一个参数是头部名称，第二个是要追加或前置的值：

```
<?php

$this->response->setHeader('Cache-Control', 'no-cache')
    ->appendHeader('Cache-Control', 'must-revalidate');
```

removeHeader()

可以通过 `removeHeader()` 方法移除响应中的头部，参数为不区分大小写的头部名称：

```
<?php

$this->response->removeHeader('Location');
```

重定向

如需创建重定向，请使用*redirect()* 函数。

该函数返回一个 RedirectResponse 实例。这与 Services::response() 返回的全局响应实例不同。

> **警告:** 如果在调用 redirect() 前设置了 Cookie 或响应头部，它们会被设置到全局响应实例，而不会自动复制到 RedirectResponse 实例。要发送它们，需手动调用 withCookies() 或 withHeaders() 方法。

重要: 若要进行重定向，必须在控制器 或控制器过滤器 的方法中返回 RedirectResponse 实例。注意 __construct() 或 initController() 方法不能返回任何值。如果忘记返回 RedirectResponse，将不会发生重定向。

重定向到 URI 路径

当需要传递相对于 baseURL 的 URI 路径时，使用 redirect()->to()：

```
// Go to specific URI path. "admin/home" is the URI path relative␣
↪to baseURL.
return redirect()->to('admin/home');
```

备注: 如果 URL 中包含需要移除的片段，可以在方法中使用 refresh 参数。例如 return redirect()->to('admin/home', null, 'refresh');。

重定向到定义的路由

当需要传递路由名称 或用于反向路由 的 Controller::method 时，使用 redirect()->route()：

```
// Go to a named route. "user_gallery" is the route name, not a URI␣
↪path.
return redirect()->route('user_gallery');
```

当向函数传递参数时，会被视为反向路由的路由名称或 Controller::method，而非相对/完整 URI，效果等同于使用 `redirect()->route()`：

```
// Go to a named/reverse-routed URI.
return redirect('named_route');
```

返回重定向

当需要返回上一页时，使用 `redirect()->back()`：

```
// Go back to the previous page.
return redirect()->back();

// Keep the old input values upon redirect so they can be used by␣
↪the `old()` function.
return redirect()->back()->withInput();

// Set a flash message.
return redirect()->back()->with('foo', 'message');
```

备注： `redirect()->back()` 与浏览器“返回”按钮不同。当 Session 可用时，它会将访问者带到“Session 期间最后查看的页面”。如果 Session 未加载或不可用，则会使用经过处理的 HTTP_REFERER。

带 Cookie 的重定向

如果在调用 `redirect()` 前设置了 Cookie，它们会被设置到全局响应实例，而不会自动复制到 `RedirectResponse` 实例。

要发送这些 Cookie，需手动调用 `withCookies()` 方法：

```
// Copies all cookies from global response instance.
return redirect()->back()->withCookies();
```

带头部的重定向

如果在调用 `redirect()` 前设置了响应头部，它们会被设置到全局响应实例，而不会自动复制到 `RedirectResponse` 实例。

要发送这些头部，需手动调用 `withHeaders()` 方法：

```
// Copies all headers from the global response instance.
return redirect()->back()->withHeaders();
```

重定向状态码

GET 请求的默认 HTTP 状态码是 302。但在使用 HTTP/1.1 或更高版本时，POST/PUT/DELETE 请求使用 303，其他请求使用 307。

可以指定状态码：

```
// Redirect to a URI path relative to baseURL with status code 301.
return redirect()->to('admin/home', 301);

// Redirect to a route with status code 308.
return redirect()->route('user_gallery', [], 308);

// Redirect back with status code 302.
return redirect()->back(302);
```

备注： 由于漏洞，在 v4.3.3 或更早版本中，即使指定了状态码，实际重定向响应的状态码也可能被更改。详见*更新日志 v4.3.4*。

如果不了解 HTTP 重定向状态码，建议阅读 HTTP 重定向。

强制文件下载

Response 类提供了向客户端发送文件并提示浏览器下载的简便方法。它会设置适当的头部来实现此功能。

第一个参数是 **下载文件的名称**，第二个参数是文件数据。

如果第二个参数设为 null 且 `$filename` 是存在的可读文件路径，则会读取该文件内容。

如果第三个参数设为 true，则会发送实际的文件 MIME 类型（基于文件扩展名），以便浏览器使用对应的处理器。

示例：

```
<?php

$data = 'Here is some text!';
$name = 'mytext.txt';

return $this->response->download($name, $data);
```

如果要下载服务器上的现有文件，需显式将第二个参数设为 `null`：

```
<?php

// Contents of photo.jpg will be automatically read
return $this->response->download('/path/to/photo.jpg', null);
```

使用可选的 `setFileName()` 方法修改发送到客户端浏览器的文件名：

```
<?php

return $this->response->download('awkwardEncryptedFileName.fakeExt',
↪ null)->setFileName('expenses.csv');
```

备注： 必须返回响应对象才能将下载内容发送到客户端。这允许响应在发送前通过所有 **after** 过滤器。

在浏览器中打开文件

某些浏览器可以显示 PDF 等文件。要告知浏览器显示而非保存文件，可调用 `DownloadResponse::inline()` 方法：

```
<?php

$data = 'Here is some text!';
$name = 'mytext.txt';

return $this->response->download($name, $data)->inline();
```

HTTP 缓存

HTTP 规范内置了帮助客户端（通常是浏览器）缓存结果的工具。正确使用可以极大提升应用性能，因为它会告知客户端无需联系服务器（当内容未变化时）。

这通过 `Cache-Control` 和 `ETag` 头部实现。本指南不深入讲解所有缓存头部，但你可以通过 Google 开发者文档 获得详细理解。

默认情况下，CodeIgniter 发送的所有响应对象都关闭了 HTTP 缓存。由于场景差异过大，我们选择关闭作为默认设置。你可以通过 `setCache()` 方法轻松设置所需缓存值：

```
<?php

$options = [
    'max-age'  => 300,
    's-maxage' => 900,
    'etag'     => 'abcde',
];
$this->response->setCache($options);
```

`$options` 数组接收键值对，除个别例外，这些参数会被分配到 `Cache-Control` 头部。你可以根据具体需求自由设置所有选项。虽然大多数选项应用于 `Cache-Control` 头部，但该方法会智能处理 `etag` 和 `last-modified` 选项到对应头部。

类参考

备注: 除了列出的方法外，此类继承自消息类 的方法。

继承自消息类的方法包括:

- `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()`

class `CodeIgniter\HTTP\`**Response**

getStatusCode()

返回

当前响应的 HTTP 状态码

返回类型

`int`

返回当前响应的状态码。如果未设置状态码，将抛出 BadMethodCallException：

```
<?php

echo $response->getStatusCode();
```

setStatusCode(*$code*[, *$reason=''*])

参数

- **$code** (int) –HTTP 状态码
- **$reason** (string) –可选原因短语

返回

当前 Response 实例

返回类型

CodeIgniter\HTTP\Response

设置响应应发送的 HTTP 状态码：

```
<?php

$response->setStatusCode(404);
```

原因短语会根据官方列表自动生成。如需为自定义状态码设置短语，可通过第二个参数传递：

```
<?php

$response->setStatusCode(230, 'Tardis initiated');
```

getReasonPhrase()

返回

当前原因短语

返回类型

string

返回当前响应的原因短语。如果未设置状态码，则返回空字符串：

```
<?php

echo $response->getReasonPhrase();
```

setDate(*$date*)

参数

- **$date** (DateTime) –包含响应时间的 DateTime 实例

返回

当前响应实例

返回类型

CodeIgniter\HTTP\Response

设置响应日期。$date 参数必须是 DateTime 实例。

setContentType(*$mime*[, *$charset='UTF-8'*])

参数

- **$mime** (string) –响应内容类型
- **$charset** (string) –响应字符集

返回

当前响应实例

返回类型

CodeIgniter\HTTP\Response

设置响应内容类型:

```
<?php

$response->setContentType('text/plain');
$response->setContentType('text/html');
$response->setContentType('application/json');
```

默认字符集为 UTF-8。如需修改，可通过第二个参数传递:

```
<?php

$response->setContentType('text/plain', 'x-pig-latin');
```

noCache()

返回

当前响应实例

返回类型

CodeIgniter\HTTP\Response

设置 Cache-Control 头部关闭所有 HTTP 缓存。这是所有响应消息的默认设置:

```
<?php

$response->noCache();
/*
 * Sets the following header:
 * Cache-Control: no-store, max-age=0, no-cache
 */
```

setCache(*$options*)

参数

- **$options** (array) –缓存控制键值对数组

返回

当前响应实例

返回类型

CodeIgniter\HTTP\Response

设置 Cache-Control 头部，包括 ETags 和 Last-Modified。常用键包括:

- etag
- last-modified
- max-age
- s-maxage
- private
- public
- must-revalidate

- proxy-revalidate
- no-transform

传递 last-modified 选项时，可以是日期字符串或 DateTime 对象。

setLastModified(*$date*)

参数

- **$date** (`string|DateTime`) –设置 Last-Modified 头部的日期

返回

当前响应实例

返回类型

`CodeIgniter\HTTP\Response`

设置 `Last-Modified` 头部。`$date` 可以是字符串或 `DateTime` 实例：

```
<?php

$response->setLastModified(date('D, d M Y H:i:s'));
$response->setLastModified(\DateTime::createFromFormat('!U',
↪ $timestamp));
```

send() → Response

返回

当前响应实例

返回类型

`CodeIgniter\HTTP\Response`

指示响应将所有内容发送回客户端。这会先发送头部，再发送响应体。主应用响应无需手动调用此方法，CodeIgniter 会自动处理。

setCookie(*$name = ''*[, *$value = ''*[, *$expire = 0*[, *$domain = ''*[, *$path = '/'*[, *$prefix = ''*[, *$secure = false*[, *$httponly = false*[, *$samesite = null*]]]]]]]]])

参数

- **$name** (`array|Cookie|string`) –Cookie 名称 或包含本方法所有参数的关联数组 或 `CodeIgniter\Cookie\Cookie` 实例
- **$value** (`string`) –Cookie 值

- **$expire** (int) –Cookie 过期时间（秒）。设为 0 时 Cookie 仅在浏览器打开期间有效
- **$domain** (string) –Cookie 域名
- **$path** (string) –Cookie 路径
- **$prefix** (string) –Cookie 名称前缀。设为 '' 时使用 **app/Config/Cookie.php** 的默认值
- **$secure** (bool) –是否仅通过 HTTPS 传输。设为 null 时使用 **app/Config/Cookie.php** 的默认值
- **$httponly** (bool) –是否仅允许 HTTP 请求访问（禁止 JavaScript）。设为 null 时使用 **app/Config/Cookie.php** 的默认值
- **$samesite** (string) –SameSite 参数值。设为 '' 时不设置该属性。设为 null 时使用 **app/Config/Cookie.php** 的默认值

返回类型

void

备注: 在 v4.2.7 之前，由于漏洞，$secure 和 $httponly 的默认值为 false，且 **app/Config/Cookie.php** 中的值未被使用。

向响应实例设置包含指定值的 Cookie。

有两种传递信息设置 Cookie 的方式：数组法和离散参数法。

数组法

使用此方法时，第一个参数传递关联数组：

```
<?php

$cookie = [
    'name'     => 'The Cookie Name',
    'value'    => 'The Value',
    'expire'   => 86500,
    'domain'   => '.some-domain.com',
    'path'     => '/',
    'prefix'   => 'myprefix_',
```

(续下页)

(接上页)

```
    'secure'   => true,
    'httponly' => false,
    'samesite' => 'Lax',
];

$response->setCookie($cookie);
```

仅需 name 和 value。要删除 Cookie 可将 value 设为空。

expire 以秒为单位，会添加到当前时间。请勿包含具体时间，只需设置从现在起有效的秒数。设为 0 时 Cookie 仅在浏览器打开期间有效。

备注： 但如果 value 设为空字符串且 expire 为 0，Cookie 将被删除。

要为整个站点设置 Cookie（无论请求方式），域名前加句点，如：.your-domain.com

通常无需设置 path，因为方法默认使用根路径。

仅在需要避免服务器上同名 Cookie 冲突时需设置 prefix。

仅在需要安全 Cookie 时设置 secure 为 true。

samesite 控制 Cookie 在域和子域间的共享方式。允许值为 'None'、'Lax'、'Strict' 或空字符串 ''。设为空字符串时使用默认 SameSite 属性。

独立参数

也可通过独立参数设置 Cookie：

```
<?php

$response->setCookie($name, $value, $expire, $domain, $path,
↪ $prefix, $secure, $httponly, $samesite);
```

deleteCookie(*$name* = ''[, *$domain* = ''[, *$path* = '/'[, *$prefix* = '']]])

参数

- **$name** (mixed) –Cookie 名称或参数数组

- **$domain** (string) –Cookie 域名
- **$path** (string) –Cookie 路径
- **$prefix** (string) –Cookie 名称前缀

返回类型

void

删除现有 Cookie。

备注： 此方法只是设置浏览器 Cookie 来删除 Cookie。

仅需 name 参数。

仅在需要避免服务器上同名 Cookie 冲突时需设置 prefix。

设置 prefix 可限定仅删除该前缀的 Cookie。设置 domain 可限定仅删除该域名的 Cookie。设置 path 可限定仅删除该路径的 Cookie。

如果任意可选参数为空，则删除所有符合条件的同名 Cookie。

示例：

```
<?php

$response->deleteCookie($name);
```

hasCookie(*$name = ''*[, *$value = null*[, *$prefix = ''*]])

参数

- **$name** (mixed) –Cookie 名称或参数数组
- **$value** (string) –Cookie 值
- **$prefix** (string) –Cookie 名称前缀

返回类型

bool

检查响应是否包含指定 Cookie。

注意

仅需 name 参数。若指定 prefix，会将其预置到 Cookie 名称前。

若未提供 `value`，仅检查是否存在该名称的 Cookie。若提供 `value`，则同时检查 Cookie 是否存在且值匹配。

示例：

```
<?php

if ($response->hasCookie($name)) {
    // ...
}
```

getCookie(*$name* = ''[, *$prefix* = ''])

参数

- **$name** (`string`) –Cookie 名称
- **$prefix** (`string`) –Cookie 名称前缀

返回类型

`Cookie|Cookie[]|null`

返回找到的指定 Cookie，未找到则返回 `null`。若未提供 `name`，返回所有 `Cookie` 对象数组。

示例：

```
<?php

$cookie = $response->getCookie($name);
```

getCookies()

返回类型

`Cookie[]`

返回响应实例中当前设置的所有 Cookie。这些是你在本次请求中明确指定要设置的 Cookie。

5.2.9 API 响应特性

大多数现代 PHP 开发需要构建 API, 无论是简单地为 javascript 密集的单页应用提供数据, 还是作为独立产品。CodeIgniter 提供了一个 API 响应特性, 可以与任何控制器一起使用, 使常见的响应类型简单化, 而无需记住哪个 HTTP 状态码应该用于哪种响应类型。

- 示例用法
- 处理响应类型
- 类参考

示例用法

下面的示例显示了控制器中常见的使用模式。

```
<?php

namespace App\Controllers;

use CodeIgniter\API\ResponseTrait;
use CodeIgniter\Controller;

class Users extends Controller
{
    use ResponseTrait;

    public function createUser()
    {
        $model = new UserModel();
        $user  = $model->save($this->request->getPost());

        // Respond with 201 status code
        return $this->respondCreated();
    }
}
```

在此示例中, 返回 HTTP 状态码 201, 以及通用的状态消息 “Created”。方法存在最常见的用例:

```php
<?php

// Generic response method
$this->respond($data, 200);

// Generic failure response
$this->fail($errors, 400);

// Item created response
$this->respondCreated($data);

// Item successfully deleted
$this->respondDeleted($data);

// Command executed by no response required
$this->respondNoContent($message);

// Client isn't authorized
$this->failUnauthorized($description);

// Forbidden action
$this->failForbidden($description);

// Resource Not Found
$this->failNotFound($description);

// Data was not validated
$this->failValidationErrors($errors);

// Resource already exists
$this->failResourceExists($description);

// Resource previously deleted
$this->failResourceGone($description);

// Client made too many requests
$this->failTooManyRequests($description);
```

处理响应类型

当你在任何这些方法中传递数据时, 它们将根据以下标准确定数据类型以格式化结果:

- 格式是根据控制器的 `$this->format` 值确定的。如果该值为 `null`，它将尝试与客户端请求的内容类型进行协商，默认为 **app/Config/Format.php** 中 `$supportedResponseFormats` 属性的第一个元素（默认是 JSON）。
- 数据将根据格式进行格式化。如果格式不是 JSON 且数据是字符串，它将被视为 HTML 发送回客户端。

备注: 在 v4.5.0 之前，由于一个错误，如果数据是字符串，即使格式是 JSON，它也会被视为 HTML。

要定义使用的格式化程序，请编辑 **app/Config/Format.php**。`$supportedResponseFormats` 包含你的应用程序可以自动格式化响应的 mime 类型列表。默认情况下，系统知道如何格式化 XML 和 JSON 响应:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Format extends BaseConfig
{
    public $supportedResponseFormats = [
        'application/json',
        'application/xml',
    ];

    // ...
}
```

这是在内容协商 期间确定要返回哪种类型响应时使用的数组。如果客户端请求和你支持的之间没有匹配, 则返回此数组中的第一种格式。

接下来, 你需要定义用于格式化数据数组的类。这必须是一个完全限定的类名, 并且该类必须实现 `CodeIgniter\Format\FormatterInterface`。开箱即用地支持 JSON 和 XML 的格式化程序:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Format extends BaseConfig
{
    public $formatters = [
        'application/json' => \CodeIgniter\Format\
↪JSONFormatter::class,
        'application/xml'  => \CodeIgniter\Format\
↪XMLFormatter::class,
    ];

    // ...
}
```

因此, 如果请求在 **Accept** 头中请求 JSON 格式的数据, 传递给任何 `respond*` 或 `fail*` 方法的数据数组将由 `CodeIgniter\Format\JSONFormatter` 类格式化。生成的 JSON 数据将发送回客户端。

类参考

setResponseFormat(*$format*)

参数

- **$format** (string) –要返回的响应类型, `json` 或 `xml`

这定义了在响应中格式化数组时使用的格式。如果为 `$format` 提供 null 值, 它将通过内容协商自动确定。

```
<?php

return $this->setResponseFormat('json')->respond(['error' =>␣
↪false]);
```

respond(*$data*[, *$statusCode* = *200*[, *$message* = ”]])

参数

- **$data** (mixed) –要返回给客户端的数据。字符串或数组。
- **$statusCode** (int) –要返回的 HTTP 状态码。默认为 200
- **$message** (string) –要返回的自定义“原因”消息。

这是特性中所有其他方法用于向客户端返回响应的方法。

$data 元素可以是字符串或数组。默认情况下, 字符串将作为 HTML 返回, 而数组将通过 json_encode 运行并返回为 JSON, 除非内容协商 确定应以不同格式返回。

如果传递了 $message 字符串, 它将替代标准 IANA 原因代码用于响应状态。但是, 并非每个客户端都会遵守自定义代码, 它们会使用与状态码匹配的 IANA 标准。

备注: 由于它在活动的 Response 实例上设置状态码和主体, 所以这应该始终是脚本执行中的最后一个方法。

---

fail(*$messages*[, *int $status = 400*[, *string $code = null*[, *string $message = ”*]]])

参数

- **$messages** (mixed) –遇到的错误消息的字符串或字符串数组。
- **$status** (int) –要返回的 HTTP 状态码。默认为 400。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

客户端首选格式的多部分响应。

这是表示失败响应的通用方法, 所有其他“fail”方法都使用它。

$messages 元素可以是字符串或字符串数组。

$status 参数是应返回的 HTTP 状态码。

由于许多 API 更适合使用自定义错误码, 所以第三个参数可以传入自定义错误码。如果没有值, 它将与 $status 相同。

如果传递了 $message 字符串, 它将替代标准 IANA 原因代码用于响应状态。但是, 并非每个客户端都会遵守自定义代码, 它们会使用与状态码匹配的 IANA 标准。

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码。“messages”元素包含错误消息数组。它看起来像:

```
<?php

$response = [
    'status'   => 400,
    'code'     => '321a',
    'messages' => [
        'Error message 1',
        'Error message 2',
    ],
];
```

respondCreated(*$data = null*[, *string $message = ''*])

参数

- **$data** (mixed) –要返回给客户端的数据。字符串或数组。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置在创建新资源时通常使用的适当状态码, 通常为 201:

```
<?php

$user = $userModel->insert($data);

return $this->respondCreated($user);
```

respondDeleted(*$data = null*[, *string $message = ''*])

参数

- **$data** (mixed) –要返回给客户端的数据。字符串或数组。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置由于此 API 调用删除新资源而通常使用的适当状态码, 通常为 200。

```
<?php

$user = $userModel->delete($id);

return $this->respondDeleted(['id' => $id]);
```

respondNoContent(*string $message = 'No Content'*)

参数

- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置在服务器成功执行命令但没有可发送回客户端的有意义响应时通常使用的适当状态码, 通常为 204。

```
<?php

sleep(1);

return $this->respondNoContent();
```

failUnauthorized(*string $description = 'Unauthorized'*[, *string $code = null*[, *string $message = ''*]])

参数

- **$description** (string) –要显示给用户的错误消息。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置用户未经授权或授权不正确时使用的适当状态码。状态码为 401。

```
<?php

return $this->failUnauthorized('Invalid Auth token');
```

failForbidden(*string $description = 'Forbidden'*[, *string $code=null*[, *string $message = ''*]])

参数

- **$description** (string) –要显示给用户的错误消息。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

与 `failUnauthorized()` 不同, 当请求的 API 端点从不允许时, 应使用此方法。未授权意味着鼓励客户端使用不同的凭据重试。禁止意味着客户端不应重试, 因为它不会有帮助。状态码通常为 403。

```
<?php

return $this->failForbidden('Invalid API endpoint.');
```

failNotFound(*string $description = 'Not Found'*[, *string $code=null*[, *string $message = ''*]])

参数

- **$description** (string) –要显示给用户的错误消息。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置在找不到请求的资源时使用的适当状态码。状态码通常为 404。

```
<?php

return $this->failNotFound('User 13 cannot be found.');
```

failValidationErrors(*$errors*[, *string $code=null*[, *string $message = ''*]])

参数

- **$errors** (mixed) –要显示给用户的错误消息或消息数组。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置在客户端发送的数据未通过验证规则时使用的适当状态码。状态码通常为 400。

```
<?php

return $this->failValidationErrors($validation->getErrors());
```

failResourceExists(*string $description = 'Conflict'*[, *string $code=null*[, *string $message = ''*]])

参数

- **$description** (string) –要显示给用户的错误消息。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置在客户端试图创建的资源已经存在时使用的适当状态码。状态码通常为 409。

```
<?php

return $this->failResourceExists('A user already exists with␣
↪that email.');
```

failResourceGone(*string $description = 'Gone'*[, *string $code=null*[, *string $message = ''*]])

参数

- **$description** (string) –要显示给用户的错误消息。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置在先前删除的请求资源不再可用时使用的适当状态码。状态码通常为 410。

```
<?php

return $this->failResourceGone('That user has been previously␣
↪deleted.');
```

failTooManyRequests(*string $description = 'Too Many Requests'*[, *string $code=null*[, *string $message = ''*]])

参数

- **$description** (string) –要显示给用户的错误消息。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置当客户端调用 API 端点次数过多时使用的适当状态码。这可能是由于某种形式的限流或速率限制。状态码通常为 400。

```
<?php

return $this->failTooManyRequests('You must wait 15 seconds␣
↪before making another request.');
```

failServerError(*string $description = 'Internal Server Error'*[, *string $code = null*[, *string $message = ''*]])

参数

- **$description** (string) –要显示给用户的错误消息。
- **$code** (string) –自定义的 API 特定错误码。
- **$message** (string) –要返回的自定义“原因”消息。

返回

Response 对象的 send() 方法的值。

设置服务器错误时使用的适当状态码。

```
<?php

return $this->failServerError('Server error.');
```

5.2.10 内容安全策略

- 什么是内容安全策略？
- 开启 *CSP*
- 运行时配置
  - 仅报告
  - 清除指令
- 内联内容
  - 使用占位符
  - 使用函数

什么是内容安全策略？

你可以采取的最好的防范 XSS 攻击的措施之一就是在网站上实施内容安全策略（CSP）。这要求你指定并授权在你的网站 HTML 中包含的每一个内容源，包括图片、样式表、JavaScript 文件等等。浏览器会拒绝来自未明确授权的源的内容。这个授权在响应的 `Content-Security-Policy` 头部中定义，并提供了各种配置选项。

这听起来很复杂，在某些网站上，确实可以是一项挑战。但是，对于许多简单的网站，其中所有的内容都由同一个域名（比如，**http://example.com**）提供，集成起来非常简单。

由于这是一个复杂的主题，这个用户指南不会详细介绍所有的细节。更多信息，你应该访问以下网站：

- 内容安全策略主站
- W3C 规范
- HTML5Rocks 的介绍

- SitePoint 的文章

开启 CSP

重要: *Debug Toolbar* 可能会使用 Kint，它会输出内联脚本。因此，当 CSP 开启时，CSP nonce 会自动为 Debug Toolbar 输出。然而，如果你没有使用 CSP nonce，这将会改变你并未打算的 CSP 头部，它的行为将会与生产环境中的不同；如果你想验证 CSP 的行为，关闭 Debug Toolbar。

默认情况下，这项支持是关闭的。要在你的应用程序中启用支持，编辑 **app/Config/App.php** 中的 `CSPEnabled` 值：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class App extends BaseConfig
{
    // ...

    public bool $CSPEnabled = true;
}
```

当启用时，响应对象会包含一个 `CodeIgniter\HTTP\ContentSecurityPolicy` 的实例。**app/Config/ContentSecurityPolicy.php** 中设定的值会被应用到这个实例，如果在运行时不需要任何改变，那么正确格式化的头部就会被发送，你就完成了所有的工作。

在启用 CSP 后，两行头部行会被添加到 HTTP 响应：一个 **Content-Security-Policy** 头部，它的策略明确允许不同上下文的内容类型或源，和一个 **Content-Security-Policy-Report-Only** 头部，它识别将被允许但也会被报告给你选择的目标的内容类型或源。

我们的实现提供了一个默认的处理方法，可以通过 `reportOnly()` 方法进行更改。当一个额外的条目被添加到 CSP 指令，如下所示，它将被添加到适当的 CSP 头部以进行阻止或预防。这可以在每次调用的基础上被覆盖，通过向添加方法调用提供一个可选的第二个参数。

运行时配置

如果你的应用程序需要在运行时进行更改，你可以在你的控制器中通过 `$this->response->getCSP()` 访问实例。

这个类包含了一些方法，这些方法与你需要设置的适当的头部值相当明显的映射。下面展示了一些例子，它们有不同的参数组合，尽管所有的都接受一个指令名或者它们的数组：

```
<?php

// get the CSP instance
$csp = $this->response->getCSP();

// specify the default directive treatment
$csp->reportOnly(false);

// specify the origin to use if none provided for a directive
$csp->setDefaultSrc('cdn.example.com');

// specify the URL that "report-only" reports get sent to
$csp->setReportURI('http://example.com/csp/reports');

// specify that HTTP requests be upgraded to HTTPS
$csp->upgradeInsecureRequests(true);

// add types or origins to CSP directives
// assuming that the default treatment is to block rather than just␣
↪report
$csp->addBaseURI('example.com', true); // report only
$csp->addChildSrc('https://youtube.com'); // blocked
$csp->addConnectSrc('https://*.facebook.com', false); // blocked
$csp->addFontSrc('fonts.example.com');
$csp->addFormAction('self');
$csp->addFrameAncestor('none', true); // report this one
$csp->addImageSrc('cdn.example.com');
$csp->addMediaSrc('cdn.example.com');
$csp->addManifestSrc('cdn.example.com');
$csp->addObjectSrc('cdn.example.com', false); // reject from here
```

(续下页)

(接上页)

```
$csp->addPluginType('application/pdf', false); // reject this media␣
↪type
$csp->addScriptSrc('scripts.example.com', true); // allow but␣
↪report requests from here
$csp->addStyleSrc('css.example.com');
$csp->addSandbox(['allow-forms', 'allow-scripts']);
```

每个 “add” 方法的第一个参数是一个适当的字符串值，或者是它们的数组。

仅报告

`reportOnly()` 方法允许你为后续的资源指定默认的报告处理方式，除非被覆盖。

例如，你可以指定 youtube.com 被允许，然后提供几个允许但需要报告的资源：

```
<?php

// get the CSP instance
$csp = $this->response->getCSP();

$csp->addChildSrc('https://youtube.com'); // allowed
$csp->reportOnly(true);
$csp->addChildSrc('https://metube.com'); // allowed but reported
$csp->addChildSrc('https://ourtube.com', false); // allowed
```

清除指令

如果你想清除现有的 CSP 指令，可以使用 `clearDirective()` 方法：

```
<?php

// get the CSP instance
$csp = $this->response->getCSP();

$csp->clearDirective('style-src');
```

内联内容

有可能设置一个网站不保护其自身页面上的内联脚本和样式，因为这可能是用户生成内容的结果。为了防止这种情况，CSP 允许你在 `<style>` 和 `<script>` 标签中指定一个 nonce，并将这些值添加到响应的头部。

使用占位符

这在现实生活中是很痛苦的，当它在飞行中生成时最安全。为了简化这个过程，你可以在标签中包含一个 `{csp-style-nonce}` 或 `{csp-script-nonce}` 占位符，它将自动为你处理:

```
// 原始
<script {csp-script-nonce}>
    console.log("脚本不会运行，因为它不包含 nonce 属性");
</script>

// 变为
<script nonce="Eskdikejidojdk978Ad8jf">
    console.log("脚本不会运行，因为它不包含 nonce 属性");
</script>

// 或者
<style {csp-style-nonce}>
    . . .
</style>
```

> **警告:** 如果攻击者注入像 `<script {csp-script-nonce}>` 这样的字符串，它可能会因为这个功能而成为真实的 nonce 属性。你可以在 **app/Config/ContentSecurityPolicy.php** 中通过 `$scriptNonceTag` 和 `$styleNonceTag` 属性自定义占位符字符串。

使用函数

如果你不喜欢上面的自动替换功能，你可以通过在 **app/Config/ContentSecurityPolicy.php** 中设置 `$autoNonce = false` 来关闭它。

在这种情况下，你可以使用函数，*csp_script_nonce()* 和*csp_style_nonce()*:

```
// 原始
<script <?= csp_script_nonce() ?>>
    console.log("脚本不会运行，因为它不包含 nonce 属性");
</script>

// 变为
<script nonce="Eskdikejidojdk978Ad8jf">
    console.log("脚本不会运行，因为它不包含 nonce 属性");
</script>

// 或者
<style <?= csp_style_nonce() ?>>
    . . .
</style>
```

5.2.11 本地化

- 处理区域设置
  - 配置区域设置
    - 设置默认区域设置
  - 区域检测
    - 内容协商
    - 在路由中
  - 设置当前区域设置
    - *IncomingRequest* 区域设置
    - 语言区域设置

- 获取当前区域设置
- 语言本地化
  - 创建语言文件
  - 基本用法
    - 替换参数
    - 指定区域设置
    - 嵌套数组
  - 语言回退
  - 系统消息翻译
  - 覆盖系统消息翻译
  - 通过命令生成翻译文件
    - 通过命令同步翻译文件

处理区域设置

CodeIgniter 提供了多种工具来帮助你对应用程序进行不同语言的本地化。虽然完整的应用程序本地化是一个复杂的主题，但替换应用程序中不同支持语言的字符串非常简单。

配置区域设置

设置默认区域设置

每个站点都有一个默认的操作语言/区域设置。这可以在 **app/Config/App.php** 中设置：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class App extends BaseConfig
{
```

(续下页)

(接上页)

```
    // ...

    public string $defaultLocale = 'en';

    // ...
}
```

该值可以是应用程序用于管理文本字符串和其他格式的任何字符串。建议使用 BCP 47 语言代码。这会生成像 en-US（美式英语）或 fr-FR（法语/法国）这样的语言代码。在 W3C 的网站 上可以找到更易读的介绍。

如果找不到完全匹配项，系统会智能回退到更通用的语言代码。如果区域设置代码设置为 `en-US`，而我们只设置了 `en` 的语言文件，则将使用这些文件，因为更具体的 `en-US` 不存在。但是，如果 **app/Language/en-US** 目录中存在语言目录，则将优先使用该目录。

区域检测

重要: 区域检测仅适用于使用 IncomingRequest 类的基于 Web 的请求。命令行请求将不具备这些功能。

有两种支持的方法可以在请求期间检测正确的区域设置。

1. 内容协商：第一种是“设置即忘记”方法，将自动执行内容协商 以确定要使用的正确区域设置。
2. 在路由中：第二种方法允许你在路由中指定一个段来设置区域设置。

如果需要直接设置区域设置，请参阅设置当前区域设置。

自 v4.4.0 起，添加了 `IncomingRequest::setValidLocales()` 来设置（和重置）从 `Config\App::$supportedLocales` 设置的有效区域。

内容协商

你可以通过 **app/Config/App.php** 中的两个附加设置来配置自动内容协商。第一个值告诉 Request 类我们确实希望协商区域设置，因此只需将其设置为 true：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class App extends BaseConfig
{
    // ...

    public bool $negotiateLocale = true;

    // ...
}
```

启用此功能后，系统将根据你在 `$supportLocales` 中定义的区域设置数组自动协商正确的语言。如果在你支持的语言与请求的语言之间找不到匹配项，则将使用 `$supportedLocales` 中的第一个项。在以下示例中，如果找不到匹配项，将使用 `en` 区域设置：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class App extends BaseConfig
{
    // ...

    public array $supportedLocales = ['en', 'es', 'fr-FR'];

    // ...
}
```

在路由中

第二种方法使用自定义占位符来检测所需区域设置并在请求中设置。占位符 `{locale}` 可以作为段放置在路由中。如果存在，匹配段的内容将是你的区域设置：

```
$routes->get('{locale}/books', 'App\Books::index');
```

在此示例中，如果用户尝试访问 **http://example.com/fr/books**，则区域设置将设置为 `fr`，前提是它被配置为有效区域设置。

如果该值与 **app/Config/App.php** 中 `$supportedLocales` 定义的有效区域设置不匹配，则将使用默认区域设置，除非你设置为仅使用 App 配置文件中定义的受支持区域设置：

```
$routes->useSupportedLocalesOnly(true);
```

备注： 自 v4.3.0 起可以使用 `useSupportedLocalesOnly()` 方法。

设置当前区域设置

IncomingRequest 区域设置

如果要直接设置区域设置，可以使用*IncomingRequest 类* 中的 `setLocale()` 方法：

```
/** @var \CodeIgniter\HTTP\IncomingRequest $request */
$request->setLocale('ja');
```

在设置区域设置之前，必须设置有效区域。因为任何尝试设置无效区域设置的操作都将导致设置默认区域设置。

默认情况下，有效区域在 **app/Config/App.php** 的 `Config\App::$supportedLocales` 中定义：

```
<?php

namespace Config;
```

(续下页)

(接上页)

```
use CodeIgniter\Config\BaseConfig;

class App extends BaseConfig
{
    // ...

    public array $supportedLocales = ['en', 'es', 'fr-FR'];

    // ...
}
```

备注: 自 v4.4.0 起，添加了 `IncomingRequest::setValidLocales()` 来设置（和重置）有效区域。如果要动态更改有效区域，请使用此方法。

语言区域设置

lang() 函数中使用的 `Language` 类也具有当前区域设置。该设置在实例化期间设置为 `IncommingRequest` 区域设置。

如果要在实例化语言类后更改区域设置，请使用 `Language::setLocale()` 方法。

```
/** @var \CodeIgniter\Language\Language $lang */
$lang = service('language');
$lang->setLocale('ja');
```

获取当前区域设置

始终可以通过 `getLocale()` 方法从 IncomingRequest 对象获取当前区域设置。如果你的控制器继承自 `CodeIgniter\Controller`，则可以通过 `$this->request` 获取:

```
<?php

namespace App\Controllers;

class UserController extends BaseController
```

(续下页)

(接上页)

```
{
    public function index()
    {
        $locale = $this->request->getLocale();
    }
}
```

或者，你可以使用*Services 类* 来获取当前请求：

```
$locale = service('request')->getLocale();
```

语言本地化

创建语言文件

语言字符串存储在 **app/Language** 目录中，每个支持的语言（区域设置）都有一个子目录:

```
app/
    Language/
        en/
            App.php
        fr/
            App.php
```

备注: 语言文件没有命名空间。

语言没有必须遵循的特定命名约定。文件应逻辑命名以描述其包含的内容类型。例如，假设你要创建一个包含错误消息的文件。你可以简单地将其命名为：**Errors.php**。

在文件中，你将返回一个数组，其中数组中的每个元素都有一个语言键，并可以返回字符串：

```
<?php

return [
```

(续下页)

(接上页)

```
    'languageKey' => 'The actual message to be shown.',
];
```

备注: 不能在语言键的开头和结尾使用点 (.)。

它还支持嵌套定义:

```
<?php

return [
    'languageKey' => [
        'nested' => [
            'key' => 'The actual message to be shown.',
        ],
    ],
];
```

```
<?php

return [
    'errorEmailMissing'    => 'You must submit an email address',
    'errorURLMissing'      => 'You must submit a URL',
    'errorUsernameMissing' => 'You must submit a username',
    'nested'               => [
        'error' => [
            'message' => 'A specific error message',
        ],
    ],
];
```

基本用法

你可以使用 *lang()* 辅助函数通过传递文件名和语言键作为第一个参数（用句点 . 分隔）从任何语言文件中检索文本。

例如，要从 **Errors.php** 语言文件加载 `errorEmailMissing` 字符串，可以执行以下操作：

```
echo lang('Errors.errorEmailMissing');
```

对于嵌套定义，可以执行以下操作：

```
echo lang('Errors.nested.error.message');
```

如果请求的语言键在当前区域设置的文件中不存在（在 *语言回退* 之后），将原样返回字符串。在此示例中，如果不存在，它将返回 `Errors.errorEmailMissing` 或 `Errors.nested.error.message`。

替换参数

备注： 以下函数都需要在系统上加载 intl 扩展才能工作。如果未加载扩展，将不会尝试替换。在 Sitepoint 上可以找到一个很好的概述。

你可以将值的数组作为第二个参数传递给 `lang()` 函数，以替换语言字符串中的占位符。这允许进行非常简单的数字翻译和格式化：

```
<?php

// The language file, Tests.php:
return [
    'apples'      => 'I have {0, number} apples.',
    'men'         => 'The top {1, number} men out-performed the␣
↪remaining {0, number}',
    'namedApples' => 'I have {number_apples, number, integer}␣
↪apples.',
];
```

(续下页)

(接上页)

```
// Displays "I have 3 apples."
echo lang('Tests.apples', [3]);
```

占位符中的第一个项对应于数组中的项索引（如果是数字）：

```
// Displays "The top 23 men out-performed the remaining 20"
echo lang('Tests.men', [20, 23]);
```

你也可以使用命名键以便更清晰：

```
// Displays "I have 3 apples."
echo lang('Tests.namedApples', ['number_apples' => 3]);
```

显然，你可以做的不仅仅是数字替换。根据底层库的 官方 ICU 文档，可以替换以下类型的数据：

- 数字 - 整数、货币、百分比
- 日期 - 短、中、长、完整
- 时间 - 短、中、长、完整
- 拼写 - 拼出数字（例如，34 变为三十四）
- 序数
- 持续时间

以下是一些示例：

```
<?php

// 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}.',
```

(续下页)

(接上页)

```
    '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()]);
```

你应该阅读 MessageFormatter 类和底层 ICU 格式化的文档，以更好地了解其功能，例如执行条件替换、复数化等。前面提供的两个链接将让你很好地了解可用选项。

指定区域设置

要指定用于替换参数的不同区域设置，可以将区域设置作为第三个参数传递给*lang()*函数。

```
<?php

// 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');
```

如果要更改当前区域设置，请参阅语言区域设置。

嵌套数组

语言文件还允许使用嵌套数组以便更轻松地处理列表等。

```
<?php

// Language/en/Fruit.php

return [
    'list' => [
        'Apples',
        'Bananas',
        'Grapes',
        'Lemons',
        'Oranges',
        'Strawberries',
    ],
];

// Displays "Apples, Bananas, Grapes, Lemons, Oranges, Strawberries"
echo implode(', ', lang('Fruit.list'));
```

语言回退

如果你为某个区域设置（例如 **Language/en/App.php**）提供了一组消息，则可以为该区域设置添加语言变体，每个变体位于自己的文件夹中，例如 **Language/en-US/App.php**。

你只需为该区域变体本地化不同的消息提供值。任何缺失的消息定义将自动从主区域设置中提取。

更好的是，本地化可以一直回退到英语（**en**），以防框架添加了新消息而你尚未有机会为你的区域设置翻译它们。

因此，如果你使用区域设置 `fr-CA`，则将首先在 **Language/fr-CA** 目录中查找本地化消息，然后在 **Language/fr** 目录中查找，最后在 **Language/en** 目录中查找。

系统消息翻译

我们在 自己的仓库 中提供了一套“官方”系统消息翻译。

你可以下载该仓库，并将其 **Language** 文件夹复制到你的 **app** 文件夹中。由于 `App` 命名空间映射到你的 **app** 文件夹，因此合并的翻译将自动被识别。

或者，更好的做法是在项目中运行以下命令：

```
composer require codeigniter4/translations
```

由于翻译文件夹被正确映射，翻译后的消息将自动被识别。

覆盖系统消息翻译

框架提供系统消息翻译，你安装的包也可能提供消息翻译。

如果要覆盖某些语言消息，请在 **app/Language** 目录中创建语言文件。然后，在文件中仅返回要覆盖的数组。

通过命令生成翻译文件

在 4.5.0 版本加入.

你可以自动生成和更新 **app** 文件夹中的翻译文件。该命令将搜索 `lang()` 函数的使用，通过定义 `Config\App` 中的 `defaultLocale` 区域设置来合并 **app/Language** 中的当前翻译键。操作完成后，你需要自行翻译语言键。该命令通常能够识别嵌套键 `File.array.nested.text`。先前保存的键不会更改。

```
php spark lang:find
```

```
<?php

// Controllers/Translation/Lang.php
$message  = lang('Text.info.success');
$message2 = lang('Text.paragraph');

// The following will be saved in Language/en/Text.php
return [
    'info' => [
        'success' => 'Text.info.success',
    ],
    'paragraph' => 'Text.paragraph',
];
```

备注: 扫描文件夹时，将跳过 **app/Language**。

生成的翻译文件很可能不符合你的编码标准。建议进行格式化。例如，如果安装了 `php-cs-fixer`，则运行 `vendor/bin/php-cs-fixer fix ./app/Language`。

在更新之前，可以预览命令找到的翻译:

```
php spark lang:find --verbose --show-new
```

`--verbose` 的详细输出还会显示无效键的列表。例如:

```
...
```

(续下页)

(接上页)

```
Files found: 10
New translates found: 30
Bad translates found: 5
+------------------------+---------------------------------+
| Bad Key                | Filepath                        |
+------------------------+---------------------------------+
| ..invalid_nested_key.. | app/Controllers/Translation.php |
| .invalid_key           | app/Controllers/Translation.php |
| TranslationBad         | app/Controllers/Translation.php |
| TranslationBad.        | app/Controllers/Translation.php |
| TranslationBad...      | app/Controllers/Translation.php |
+------------------------+---------------------------------+

All operations done!
```

为了更精确地搜索，请指定要扫描的区域设置或目录。

```
php spark lang:find --dir Controllers/Translation --locale en --
↪show-new
```

可以通过运行命令获取详细信息：

```
php spark lang:find --help
```

通过命令同步翻译文件

在 4.6.0 版本加入.

当你完成当前语言的翻译后，可能需要为另一种语言创建文件。你可以使用 spark `lang:find` 命令来帮助完成此操作。但是，它可能无法检测到所有翻译，特别是那些具有动态设置参数的翻译，例如 `lang('App.status.' . $key, ['payload' => 'John'], 'en')`。

为了确保不遗漏任何翻译，最好复制已完成的语言文件并手动翻译它们。这种方法可以保留命令可能遗漏的任何唯一键。

只需执行：

```
// 指定新/更新翻译的区域设置
php spark lang:sync --target ru

// 或设置原始区域设置
php spark lang:sync --locale en --target ru
```

结果你将获得包含翻译键的文件。如果目标区域设置中存在重复键，则会保存这些键。

> **警告:** 新翻译中不匹配的键将被删除!

5.2.12 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, 你将在视图文件中使用纯 PHP 。为了最大限度地减少这些文件中的 PHP 代码, 并更易于识别代码块, 建议使用 PHP 的替代语法来控制结构和短标签 echo 语句。如果你不熟悉这种语法, 它可以消除代码中的大括号, 并消除 “echo” 语句。

替代 Echo

通常要输出或打印一个变量, 你会这样做:

```
<?php echo esc($variable); ?>
```

使用替代语法, 你可以这样做:

```
<?= esc($variable) ?>
```

替代控制结构

控制结构, 像 if、for、foreach 和 while 也可以使用简化格式。这里有一个 `foreach` 的例子:

```
<ul>

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

(续下页)

(接上页)

```
    <li><?= esc($item) ?></li>

<?php endforeach ?>

</ul>
```

注意这里没有大括号。相反, 结束的大括号被 endforeach 替换。上面列出的每个控制结构都有类似的结束语法: endif、endfor、endforeach 和 endwhile

同样要注意的是, 除了最后一个结构外, 每个结构后面都使用冒号, 而不是分号。这很重要!

这里是一个使用 if/elseif/else 的例子。注意冒号:

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

    <h3>Hi Sally</h3>

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

    <h3>Hi Joe</h3>

<?php else: ?>

    <h3>Hi unknown user</h3>

<?php endif ?>
```

章节 6

数据库

6.1 使用数据库

CodeIgniter 带有一个功能齐全且非常快速的抽象数据库类, 它支持传统结构和查询构建器模式。数据库函数提供清晰、简单的语法。

6.1.1 快速入门：使用示例

以下页面包含显示数据库类用法的示例代码。有关完整详细信息, 请阅读描述每个函数的单独页面。

备注： CodeIgniter 不支持在表名和列名中使用点（.）。自 v4.5.0 起，支持带点的数据库名称。

- 初始化数据库类
- 具有多个结果的标准查询 (对象版本)
- 具有多个结果的标准查询 (数组版本)

- 具有单个结果的标准查询
- 具有单个结果的标准查询 (数组版本)
- 标准插入
- 查询构建器查询
- 查询构建器插入

初始化数据库类

以下代码基于你的配置 设置加载并初始化数据库类:

```
<?php

$db = \Config\Database::connect();
```

一旦加载, 该类即可按照下述方式使用。

备注: 如果所有页面都需要数据库访问, 则可以自动连接。有关详细信息, 请参阅连接数据库 页面。

具有多个结果的标准查询 (对象版本)

```
<?php

$query   = $db->query('SELECT name, title, email FROM my_table');
$results = $query->getResult();

foreach ($results as $row) {
    echo $row->title;
    echo $row->name;
    echo $row->email;
}

echo 'Total Results: ' . count($results);
```

上述 `getResult()` 函数返回 **对象**数组。

示例:`$row->title`

具有多个结果的标准查询 (数组版本)

```
<?php

$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'];
}
```

上述 `getResultArray()` 函数返回标准数组索引的数组。

示例:`$row['title']`

具有单个结果的标准查询

```
<?php

$query = $db->query('SELECT name FROM my_table LIMIT 1');
$row   = $query->getRow();
echo $row->name;
```

上述 `getRow()` 函数返回一个 **对象**。示例:`$row->name`

具有单个结果的标准查询 (数组版本)

```
<?php

$query = $db->query('SELECT name FROM my_table LIMIT 1');
$row   = $query->getRowArray();
echo $row['name'];
```

上述 `getRowArray()` 函数返回一个 **数组**。示例: `$row['name']`。

标准插入

```
<?php

$sql = 'INSERT INTO mytable (title, name) VALUES (' . $db->escape(
↪$title) . ', ' . $db->escape($name) . ')';
$db->query($sql);
echo $db->affectedRows();
```

查询构建器查询

查询构建器 为你提供了一种简化的检索数据方式:

```
<?php

$query = $db->table('table_name')->get();

foreach ($query->getResult() as $row) {
    echo $row->title;
}
```

上述 `get()` 函数检索从提供的表中获取的所有结果。查询构建器 类包含用于处理数据的完整功能。

查询构建器插入

```
<?php

$data = [
    'title' => $title,
    'name'  => $name,
    'date'  => $date,
];

$db->table('mytable')->insert($data);
// Produces: INSERT INTO mytable (title, name, date) VALUES ('{
↪$title}', '{$name}', '{$date}')
```

6.1.2 数据库配置

- 配置文件
  - 设置默认数据库
    - *DSN*
    - 故障转移
  - 设置多个数据库
  - 自动更改数据库
- 使用.*env* 文件配置
- 值的描述
  - *MySQLi*
    - *hostname*
    - *encrypt*

备注: 请参阅支持的数据库 以获取当前支持的数据库驱动。

配置文件

CodeIgniter 有一个配置文件, 可让你存储数据库连接值 (用户名、密码、数据库名称等)。配置文件位于 **app/Config/Database.php**。你也可以在 **.env** 文件中设置数据库连接值。下面详细介绍。

设置默认数据库

配置设置存储在一个类属性中, 该属性是一个数组, 原型如下:

```
<?php

namespace Config;

use CodeIgniter\Database\Config;

class Database extends Config
{
    // ...

    public array $default = [
        'DSN'      => '',
        'hostname' => 'localhost',
        'username' => 'root',
        'password' => '',
        'database' => 'database_name',
        'DBDriver' => 'MySQLi',
        'DBPrefix' => '',
        'pConnect' => false,
        'DBDebug'  => true,
        'charset'  => 'utf8mb4',
        'DBCollat' => 'utf8mb4_general_ci',
        'swapPre'  => '',
        'encrypt'  => false,
        'compress' => false,
        'strictOn' => false,
        'failover' => [],
        'port'     => 3306,
```

(续下页)

(接上页)

```
    ];

    // ...
}
```

类属性的名称是连接名称, 在连接时可以用作指定组名。

备注: SQLite3 数据库的默认位置是 **writable** 文件夹。如果要更改位置，则必须设置新文件夹（例如，'database' => WRITEPATH . 'db/database_name.db'）的完整路径。

DSN

某些数据库驱动程序（如 Postgre、OCI8）需要完整的 DSN（Data Source Name）字符串才能连接。但是，如果你没有为需要 DSN 字符串的驱动程序指定 DSN 字符串，CodeIgniter 将尝试使用其余提供的设置来构建它。

如果指定了 DSN, 你应该使用 `'DSN'` 配置设置, 就像你正在使用驱动程序的底层原生 PHP 扩展一样, 如下所示:

```
    // OCI8
    public array $default = [
        'DSN' => '//localhost/XE',
        // ...
    ];
```

通用方式的 DSN

你还可以以通用方式 (URL 格式) 设置 DSN。在这种情况下,DSN 必须具有以下原型:

```
    public array $default = [
        'DSN' => 'DBDriver://username:password@hostname:port/
↪database',
        // ...
    ];
```

要使用 DSN 字符串的通用版本覆盖默认配置值, 请将配置变量作为查询字符串添加:

```
    // MySQLi
    public array $default = [
        'DSN' => 'MySQLi://username:password@hostname:3306/database?
↪charset=utf8mb4&DBCollat=utf8mb4_general_ci',
        // ...
    ];
```

```
    // Postgre
    public array $default = [
        'DSN' => 'Postgre://username:password@hostname:5432/
↪database?charset=utf8&connect_timeout=5&sslmode=require',
        // ...
    ];
```

备注: 如果你提供了一个 DSN 字符串, 但缺少配置字段中存在的一些有效设置 (例如数据库字符集),CodeIgniter 将会追加它们。

故障转移

你还可以针对主连接由于某些原因无法连接的情况指定故障转移。可以通过像这样为连接设置故障转移来指定故障转移:

```
<?php

namespace Config;

use CodeIgniter\Database\Config;

class Database extends Config
{
    // ...

    public array $default = [
        // ...
        'failover' => [
```

(续下页)

(接上页)

```
            [
                'hostname' => 'localhost1',
                'username' => '',
                'password' => '',
                'database' => '',
                'DBDriver' => 'MySQLi',
                'DBPrefix' => '',
                'pConnect' => true,
                'DBDebug'  => true,
                'charset'  => 'utf8mb4',
                'DBCollat' => 'utf8mb4_general_ci',
                'swapPre'  => '',
                'encrypt'  => false,
                'compress' => false,
                'strictOn' => false,
            ],
            [
                'hostname' => 'localhost2',
                'username' => '',
                'password' => '',
                'database' => '',
                'DBDriver' => 'MySQLi',
                'DBPrefix' => '',
                'pConnect' => true,
                'DBDebug'  => true,
                'charset'  => 'utf8mb4',
                'DBCollat' => 'utf8mb4_general_ci',
                'swapPre'  => '',
                'encrypt'  => false,
                'compress' => false,
                'strictOn' => false,
            ],
        ],
        // ...
    ];

    // ...
}
```

你可以指定任意多个故障转移。

设置多个数据库

你可以可选地存储多个连接值集。例如, 如果你在单个安装下运行多个环境 (开发、生产、测试等), 则可以为每个环境设置一个连接组, 然后根据需要在组之间切换。例如, 要设置 “测试” 环境, 你可以这样做:

```
<?php

namespace Config;

use CodeIgniter\Database\Config;

class Database extends Config
{
    // ...

    public array $test = [
        'DSN'      => '',
        'hostname' => 'localhost',
        'username' => 'root',
        'password' => '',
        'database' => 'database_name',
        'DBDriver' => 'MySQLi',
        'DBPrefix' => '',
        'pConnect' => true,
        'DBDebug'  => true,
        'charset'  => 'utf8mb4',
        'DBCollat' => 'utf8mb4_general_ci',
        'swapPre'  => '',
        'compress' => false,
        'encrypt'  => false,
        'strictOn' => false,
        'failover' => [],
    ];

    // ...
```

(续下页)

(接上页)

```
}
```

然后, 要在全局范围内告诉系统使用该组, 请设置配置文件中的此变量:

```
<?php

namespace Config;

use CodeIgniter\Database\Config;

class Database extends Config
{
    // ...

    public string $defaultGroup = 'test';

    // ...
}
```

备注: 名称 `test` 任意的。它可以是你想要的任何内容。默认情况下, 我们已经将主连接的名称设置为 `default`, 但它也可以重命名为与项目更相关的名称。

自动更改数据库

你可以修改配置文件以检测环境并自动更新 `defaultGroup` 值为正确的值, 方法是在类构造函数中添加所需的逻辑:

```
<?php

namespace Config;

use CodeIgniter\Database\Config;

/**
 * Database Configuration
```

(续下页)

(接上页)

```
 */
class Database extends Config
{
    // ...
    public $development = [/* ... */];
    public $test        = [/* ... */];
    public $production  = [/* ... */];

    public function __construct()
    {
        // ...

        $this->defaultGroup = ENVIRONMENT;
    }
}
```

使用.env 文件配置

你还可以在 **.env** 文件中保存你的配置值，其中包含当前服务器的数据库设置。你只需要输入与默认配置组中的设置不同的值。这些值应该遵循以下格式，其中 `default` 是组名:

```
database.default.username = 'root';
database.default.password = '';
database.default.database = 'ci4';
```

但是你不能通过设置环境变量来添加新属性, 也不能将标量值更改为数组。有关详细信息, 请参阅*作为数据的环境变量*。

因此, 如果要对 MySQL 使用 SSL, 你需要一个 hack。例如, 在你的 **.env** 文件中将数组值设置为 JSON 字符串:

```
database.default.encrypt = {"ssl_verify":true,"ssl_ca":"/var/www/
↪html/BaltimoreCyberTrustRoot.crt.pem"}
```

并在 Config 类的构造函数中解码它:

```php
<?php

namespace Config;

use CodeIgniter\Database\Config;

/**
 * Database Configuration
 */
class Database extends Config
{
    // ...

    public function __construct()
    {
        // ...

        $array = json_decode($this->default['encrypt'], true);
        if (is_array($array)) {
            $this->default['encrypt'] = $array;
        }
    }
}
```

值的描述

名称	描述
DSN	DSN 连接字符串 (一体化配置序列)。
hostname	数据库服务器的主机名。通常是‘localhost’。
username	用于连接数据库的用户名。(`SQLite3` 不使用此用户名)
password	用于连接数据库的密码。(`SQLite3` 不使用此密码)
database	要连接的数据库名称。 **备注：** CodeIgniter 不支持在表名和列名中使用点 (`.`)。从 v4.5.0 版本开始，支持带点的数据库名。
DBDriver	数据库驱动名称。驱动名称区分大小写。你可以设置完全限定的类名以使用自定义驱动。支持的驱动:`MySQLi`、`Postgre`、`SQLite3`、`SQLSRV` 和 `OCI8`。
DBPrefix	可选的表前缀，在运行时会添加到表名中查询构建器 查询。这允许多个 CodeIgniter 安装共享一个数据库。
pConnect	true/false (布尔值)- 是否使用持久连接。
DBDebug	true/false (布尔值)- 当发生数据库错误时是否抛出异常。
charset	与数据库通信使用的字符集。
DBCollat	（仅限 `MySQLi`）与数据库通信时使用的字符集。
swapPre	一个默认的表前缀，应该与 `DBPrefix` 互换。这对于分布式应用程序很有用，在那里你可能运行手动编写的查询，并需要最终用户仍可自定义前缀。
schema	（仅限 `Postgre` 和 `SQLSRV`）数据库模式，默认值因驱动而异。
encrypt	（仅限 `MySQLi` 和 `SQLSRV`）是否使用加密连接。有关 `MySQLi` 设置，请参见*MySQLi encrypt*。`SQLSRV` 驱动程序接受 true/false。
compress	（仅限 `MySQLi`）是否使用客户端压缩。
strictOn	（仅限 `MySQLi`）true/false（布尔值）- 是否强制使用“严格模式”连接，有助于确保在开发应用程序时使用严格的 SQL。
port	数据库端口号 - 默认端口为空字符串 `''` (或使用 `SQLSRV` 动态端口)。

备注: 根据你使用的数据库驱动程序 (`MySQLi`、`Postgre` 等), 并非所有的值都是必需的。例如, 在使用 `SQLite3` 时, 你不需要提供用户名或密码, 数据库名称将是数据库文件的路径。

MySQLi

hostname

配置一个 Socket 连接

要通过文件系统套接字连接到 MySQL 服务器，应在 `'hostname'` 设置中指定套接字的路径。CodeIgniter 的 MySQLi 驱动程序会注意到这一点并正确配置连接。

```
    // MySQLi over a socket
    public array $default = [
        // ...
        'hostname' => '/cloudsql/toolbox-tests:europe-
↪north1:toolbox-db',
        // ...
        'DBDriver' => 'MySQLi',
        // ...
    ];
```

encrypt

MySQLi 驱动程序接受包含以下选项的数组：

- `ssl_key` - 私钥文件的路径
- `ssl_cert` - 公钥证书文件的路径
- `ssl_ca` - 证书颁发机构文件的路径
- `ssl_capath` - 包含以 PEM 格式存储的可信 CA 证书的目录路径
- `ssl_cipher` - 允许用于加密的密码列表，用冒号（`:`）分隔
- `ssl_verify` - true/false (布尔值) - 是否验证服务器证书

6.1.3 连接数据库

- 连接数据库
  - 连接默认组
  - 可用参数
  - 连接特定组
  - 连接到同一数据库的多个连接
- 连接多个数据库
- 使用自定义设置连接
- 重新连接/保持连接活动
- 手动关闭连接

连接数据库

连接默认组

你可以通过在任何需要的函数中添加此代码行来连接数据库, 或者在类构造函数中添加此行以在该类中全局提供数据库。

```
$db = \Config\Database::connect();
```

如果上述函数的第一个参数不包含任何信息, 则它将连接数据库配置文件中指定的默认组。对于大多数人来说, 这是首选的使用方法。

为方便起见, 还提供了一个纯包装器方法, 代码如下:

```
$db = db_connect();
```

可用参数

\Config\Database::connect($group = null, bool $getShared = true): BaseConnection

1. `$group`: 数据库组名称, 必须与配置类属性名称匹配的字符串。默认值为 `Config\Database::$defaultGroup`。
2. `$getShared`: true/false(布尔值)。是否返回共享连接 (参见下面的连接多个数据库)。

连接特定组

此函数的第一个参数可以用来指定配置文件中的特定数据库组。示例:

要从配置文件中选择一个特定的组, 可以这样做:

```
$db = \Config\Database::connect('group_name');
```

其中 `group_name` 是配置文件中连接组的名称。

连接到同一数据库的多个连接

默认情况下, `connect()` 方法每次都会返回数据库连接的同一实例。如果你需要与同一数据库建立一个单独的连接, 请将 `false` 作为第二个参数发送:

```
$db = \Config\Database::connect('group_name', false);
```

连接多个数据库

如果你需要同时连接多个数据库, 可以这样做:

```
$db1 = \Config\Database::connect('group_one');
$db2 = \Config\Database::connect('group_two');
```

注意: 请将 "group_one" 和 "group_two" 更改为你正在连接的特定组名称。

备注: 如果你只需要在同一连接上使用不同的数据库, 则不需要创建单独的数据库配置。当需要时, 你可以切换到不同的数据库, 像这样:

```
$db->setDatabase($database2_name);
```

使用自定义设置连接

你可以传递一个数据库设置数组而不是组名称来获取使用自定义设置的连接。传递的数组必须与配置文件中定义组的格式相同:

```
$custom = [
    'DSN'      => '',
    'hostname' => 'localhost',
    'username' => '',
    'password' => '',
    'database' => '',
    'DBDriver' => 'MySQLi',
    'DBPrefix' => '',
    'pConnect' => false,
    'DBDebug'  => true,
    'charset'  => 'utf8mb4',
    'DBCollat' => 'utf8mb4_general_ci',
    'swapPre'  => '',
    'encrypt'  => false,
    'compress' => false,
    'strictOn' => false,
    'failover' => [],
    'port'     => 3306,
];
$db = \Config\Database::connect($custom);
```

重新连接/保持连接活动

如果在执行一些繁重的 PHP 操作 (处理图像等) 时超过数据库服务器的空闲超时, 则在发送进一步查询之前, 应考虑通过使用 `reconnect()` 方法向服务器发出 ping, 这可以优雅地保持连接活动或重新建立连接。

重要: 如果使用 MySQLi 数据库驱动程序, `reconnect()` 方法不会向服务器发出 ping, 而是关闭连接然后再次连接。

```
$db->reconnect();
```

手动关闭连接

虽然 CodeIgniter 会智能地关闭数据库连接, 但你可以显式关闭连接。

```
$db->close();
```

6.1.4 查询

- 查询基础知识
  - 常规查询
  - 简化的查询
- 手动使用数据库前缀
  - *$db->prefixTable()*
  - *$db->setPrefix()*
  - *$db->getPrefix()*
- 保护标识符
  - *$db->protectIdentifiers()*
- 转义值
  - *1. $db->escape()*
  - *2. $db->escapeString()*
  - *3. $db->escapeLikeString()*
- 查询绑定
  - 命名绑定
- 处理错误
  - *$db->error()*
- 预处理查询

- 准备查询
- 执行查询
- 其他方法
- 使用查询对象
  - *$db->getLastQuery()*
  - 查询类

查询基础知识

备注: CodeIgniter 不支持在表名和列名中使用点（.）。自 v4.5.0 起，支持带点的数据库名称。

常规查询

$db->query()

要提交查询, 请使用 `query()` 方法:

```
<?php

$db = db_connect();
$db->query('YOUR QUERY HERE');
```

当运行“读取”类型查询时, `query()` 方法会返回一个可以用来显示结果 的数据库结果 **对象**。当运行“写入”类型查询时, 它根据成功或失败简单地返回 true 或 false。检索数据时, 你通常会将查询分配给自己的变量, 如下所示:

```
<?php

$query = $db->query('YOUR QUERY HERE');
```

备注: 如果使用 OCI8 驱动程序,SQL 语句不应以分号 (`;`) 结尾。PL/SQL 语句应以分号

(;) 结尾。

简化的查询

$db->simpleQuery()

`simpleQuery()` 方法是 `$db->query()` 方法的简化版本。它不会返回数据库结果集,也不会设置查询计时器或编译绑定数据或存储调试查询。它只是让你提交一个查询。大多数用户很少使用此功能。

它返回数据库驱动程序的 "execute" 函数返回的任何内容。对于写入类型的查询 (如 INSERT、DELETE 或 UPDATE 语句), 这通常在成功或失败时返回 true/false(这确实是应该使用它的地方), 并在具有可获取结果的查询成功时返回资源/对象。

```
<?php

if ($db->simpleQuery('YOUR QUERY')) {
    echo 'Success!';
} else {
    echo 'Query failed!';
}
```

备注: 例如,PostgreSQL 的 `pg_exec()` 函数总是在成功时返回资源, 即使对于写入类型的查询也是如此。所以如果你正在查找布尔值, 请记住这一点。

手动使用数据库前缀

$db->prefixTable()

如果你已经配置了数据库前缀, 并希望在本机 SQL 查询 (例如) 中将其添加到表名前, 那么可以使用以下代码:

```
<?php

$db->prefixTable('tablename'); // outputs prefix_tablename
```

$db->setPrefix()

如果由于任何原因你想以编程方式更改前缀, 而不需要创建新的连接, 则可以使用此方法:

```
<?php

$db->setPrefix('newprefix_');
$db->prefixTable('tablename'); // outputs newprefix_tablename
```

$db->getPrefix()

你可以使用此方法随时获取当前前缀:

```
<?php

$DBPrefix = $db->getPrefix();
```

保护标识符

$db->protectIdentifiers()

在许多数据库中, 保护表格和字段名称 (例如在 MySQL 中使用反引号) 是可取的。**查询构建器查询会自动受保护**, 但是如果你需要手动保护一个标识符, 可以使用:

```
<?php

$db->protectIdentifiers('table_name');
```

重要: 尽管查询构建器会尽力正确引用你提供的任何字段和表名，但请注意它并不适用于任意用户输入。请勿将未经过处理的用户数据提供给它。

如果在数据库配置文件中指定了前缀, 此函数也会将 **表前缀**添加到表格名, 以启用前缀, 请通过第二个参数设置 `true` (布尔值):

```
<?php

$db->protectIdentifiers('table_name', true);
```

转义值

在将数据提交到数据库之前对其进行转义是非常好的安全实践。CodeIgniter 提供了三种帮助你实现这一点的方法:

1. $db->escape()

此函数确定数据类型, 以便它只能转义字符串数据。它还会自动在数据周围添加单引号, 所以你不必这样做:

```
<?php

$sql = 'INSERT INTO table (title) VALUES(' . $db->escape($title) .
↪')';
```

2. $db->escapeString()

此函数转义传入的数据, 而不考虑类型。大多数时间你会使用上面的函数而不是这个。像这样使用该函数:

```
<?php

$sql = "INSERT INTO table (title) VALUES('" . $db->escapeString(
↪$title) . "')";
```

3. $db->escapeLikeString()

当字符串将在 LIKE 条件中使用时, 应使用此方法, 以便字符串中的 LIKE 通配符 (%、_) 也适当转义。

```
<?php

$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) 来转义 `LIKE` 条件的特殊字符。因为此方法转义了你自己要用引号括起来的部分字符串, 所以它无法自动为你添加 `ESCAPE '!'` 条件, 因此你必须手动完成这一操作。

查询绑定

绑定使你可以通过让系统为你组装查询来简化查询语法。考虑以下示例:

```
<?php

$sql = 'SELECT * FROM some_table WHERE id = ? AND status = ? AND␣
↪author = ?';
$db->query($sql, [3, 'live', 'Rick']);
```

查询中的问号自动替换为查询函数第二个参数数组中的值。

绑定也适用于数组, 它将转换为 IN 集:

```
<?php

$sql = 'SELECT * FROM some_table WHERE id IN ? AND status = ? AND␣
↪author = ?';
$db->query($sql, [[3, 6], 'live', 'Rick']);
```

生成的查询将是:

```
SELECT * FROM some_table WHERE id IN (3,6) AND status = 'live' AND␣
↪author = 'Rick'
```

使用绑定的次要好处是值会自动转义, 从而产生更安全的查询。你不必记住手动转义数据 - 引擎会自动为你完成这一操作。

命名绑定

除了使用问号标记绑定值的位置外, 你还可以命名绑定, 允许传入值的键与查询中的占位符匹配:

```
<?php

$sql = 'SELECT * FROM some_table WHERE id = :id: AND status =␣
↪:status: AND author = :name:';
$db->query($sql, [
    'id'     => 3,
    'status' => 'live',
    'name'   => 'Rick',
]);
```

备注: 查询中的每个名称必须用冒号括起来。

处理错误

$db->error()

如果你需要获取最后发生的错误, `error()` 方法将返回包含代码和消息的数组。这是一个快速示例:

```
<?php

if (! $db->simpleQuery('SELECT `example_field` FROM `example_table`
↪')) {
    $error = $db->error(); // Has keys 'code' and 'message'
}
```

预处理查询

大多数数据库引擎都支持某种形式的预编译语句, 允许你预先准备一次查询, 然后使用新的数据集多次运行该查询。这消除了 SQL 注入的可能性, 因为数据采用不同于查询本身的格式传递给数据库。当需要多次运行相同的查询时, 它也可以快得多。但是, 对每个查询都这样做可能会大大降低性能, 因为调用数据库的频率加倍。由于查询生成器和数据库连接已经为你处理了数据的转义, 所以安全方面已经为你照顾好了。但是, 有时候你需要通过运行预编译语句或预处理查询来优化查询。

准备查询

这可以通过 `prepare()` 方法轻松完成。它接受一个参数, 该参数是一个返回查询对象的闭包。查询对象由任何 "final" 类型的查询自动生成, 包括 **insert**、**update**、**delete**、**replace** 和 **get**。通过使用查询构建器运行查询来实现这一点最简单。查询实际上不会运行, 值也不重要, 因为它们从未应用, 而是充当占位符。这将返回一个 PreparedQuery 对象:

```
<?php

$pQuery = $db->prepare(static fn ($db) => $db->table('user')->
↪insert([
    'name'    => 'x',
    'email'   => 'y',
    'country' => 'US',
]));
```

如果不想使用查询构建器, 可以使用问号作为值占位符手动创建查询对象:

```
<?php

use CodeIgniter\Database\Query;

$pQuery = $db->prepare(static function ($db) {
    $sql = 'INSERT INTO user (name, email, country) VALUES (?, ?, ?)
↪';

    return (new Query($db))->setQuery($sql);
});
```

如果数据库在预编译语句阶段需要一个选项数组传递给它, 则可以在第二个参数中传递

该数组:

```
<?php

use CodeIgniter\Database\Query;

$pQuery = $db->prepare(static function ($db) {
    $sql = 'INSERT INTO user (name, email, country) VALUES (?, ?, ?)
↪';

    return (new Query($db))->setQuery($sql);
}, $options);
```

备注: 目前，唯一实际使用选项数组的数据库是 SQLSRV。

执行查询

一旦你有了预处理的查询, 就可以使用 `execute()` 方法实际运行查询。你可以根据查询中的占位符数量传递任意多个变量。必须传入的参数数目必须与查询中的占位符数目匹配。它们还必须以在原始查询中出现的占位符的顺序传递:

```
<?php

// Prepare the Query
$pQuery = $db->prepare(static fn ($db) => $db->table('user')->
↪insert([
    'name'    => 'x',
    'email'   => 'y',
    'country' => 'US',
]));

// Collect the Data
$name    = 'John Doe';
$email   = 'j.doe@example.com';
$country = 'US';
```

(续下页)

(接上页)

```
// Run the Query
$results = $pQuery->execute($name, $email, $country);
```

对于“写入”类型的查询, 它返回 true 或 false, 指示查询的成功或失败。对于“读取”类型的查询, 它返回一个标准的结果集。

其他方法

除了这两个主要方法之外, 预处理查询对象还有以下方法:

close()

虽然 PHP 在数据库关闭所有打开的语句方面做得很好, 但是当不需要预处理语句时关闭它总是一个好主意:

```
<?php

if ($pQuery->close()) {
    echo 'Success!';
} else {
    echo 'Deallocation of prepared statements failed!';
}
```

备注: 从 v4.3.0 开始, `close()` 方法在所有 DBMS 中释放预编译语句。以前, 它们在 Postgre、SQLSRV 和 OCI8 中没有被释放。

getQueryString()

这将返回预处理查询的字符串。

hasError()

如果最后一个 `execute()` 调用产生任何错误, 则返回布尔值 true/false。

getErrorCode()

getErrorMessage()

如果遇到任何错误, 可以使用这些方法检索错误代码和字符串。

使用查询对象

在内部, 所有查询都作为 `CodeIgniter\Database\Query` 的实例进行处理和存储。此类负责绑定参数, 否则准备查询, 并存储有关其查询的性能数据。

$db->getLastQuery()

当你只需要检索最后一个查询对象时, 使用 `getLastQuery()` 方法:

```
<?php

$query = $db->getLastQuery();
echo (string) $query;
```

查询类

每个查询对象都存储与查询本身相关的几个信息片段。这在一定程度上由时间线功能使用, 但也可供你使用。

getQuery()

在所有处理完成后返回最终查询。这是发送到数据库的确切查询:

```
<?php

$sql = $query->getQuery();
```

通过将查询对象转换为字符串, 也可以检索相同的值:

```
<?php

$sql = (string) $query;
```

getOriginalQuery()

返回传入对象的原始 SQL。其中不会有任何绑定, 也不会替换前缀等:

```
<?php

$sql = $query->getOriginalQuery();
```

hasError()

如果在执行此查询期间遇到错误, 则此方法将返回 true:

```
<?php

if ($query->hasError()) {
    echo 'Code: ' . $query->getErrorCode();
    echo 'Error: ' . $query->getErrorMessage();
}
```

isWriteType()

如果查询被确定为写入类型查询 (即 INSERT、UPDATE、DELETE 等), 则返回 true:

```
<?php

if ($query->isWriteType()) {
    // ... do something
}
```

swapPrefix()

用另一个值替换 SQL 中的一个表前缀。第一个参数是要替换的原始前缀, 第二个参数是要替换的值:

```
<?php

$sql = $query->swapPrefix('ci3_', 'ci4_');
```

getStartTime()

以秒 (含微秒) 为单位获取查询执行的时间:

```
<?php

$microtime = $query->getStartTime();
```

getDuration()

以秒 (含微秒) 为单位返回查询持续时间的浮点数:

```
<?php

$microtime = $query->getDuration();
```

6.1.5 生成查询结果

有几种生成查询结果的方法:

- 结果数组
  - *getResult()*
    - 获取 *stdClass* 的数组
    - 获取数组的数组
    - 获取自定义对象的数组

- *getResultArray()*
- 结果集
  - *getRow()*
  - *getRowArray()*
  - *getUnbufferedRow()*
- 自定义结果对象
  - *getCustomResultObject()*
  - *getCustomRowObject()*
- 结果辅助方法
  - *getFieldCount()*
  - *getFieldNames()*
  - *getNumRows()*
  - *freeResult()*
  - *dataSeek()*
- 类参考

结果数组

getResult()

此方法将查询结果作为 **对象**的数组返回, 如果失败则返回 **空数组**。

获取 stdClass 的数组

通常你会在 foreach 循环中使用它, 如下所示:

```
<?php

$query = $db->query('YOUR QUERY');
```

(续下页)

(接上页)

```
foreach ($query->getResult() as $row) {
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

上面的方法是*CodeIgniter\Database\BaseResult::getResultObject()* 的别名。

获取数组的数组

如果希望以数组的数组形式获取结果, 可以在第一个参数中传递 'array' 字符串:

```
<?php

$query = $db->query('YOUR QUERY');

foreach ($query->getResult('array') as $row) {
    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

上面的用法是*getResultArray()* 的别名。

获取自定义对象的数组

你也可以将表示要为每个结果对象实例化的类的字符串传递给 `getResult()`

```
<?php

$query = $db->query('SELECT * FROM users;');

foreach ($query->getResult(\App\Entities\User::class) as $user) {
    echo $user->name;           // access attributes
    echo $user->reverseName(); // or methods defined on the 'User'␣
```

(续下页)

(接上页)

```
↪class
}
```

上面的方法是*getCustomResultObject()* 的别名。

getResultArray()

此方法将查询结果作为纯数组返回, 如果没有生成结果, 则返回空数组。通常你会在 foreach 循环中使用它, 如下所示:

```
<?php

$query = $db->query('YOUR QUERY');

foreach ($query->getResultArray() as $row) {
    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

结果集

getRow()

此方法返回单个结果集。如果查询有多个行, 则只返回第一行。结果作为 **对象**返回。这里有个使用示例:

```
<?php

$query = $db->query('YOUR QUERY');

$row = $query->getRow();

if (isset($row)) {
    echo $row->title;
    echo $row->name;
```

(续下页)

(接上页)

```
    echo $row->body;
}
```

如果要返回特定的行, 可以在第一个参数中提交行号作为数字:

```
<?php

$row = $query->getRow(5);
```

你还可以添加第二个字符串参数, 该参数是要使用的类的名称:

```
<?php

$query = $db->query('SELECT * FROM users LIMIT 1;');
$row   = $query->getRow(0, \App\Entities\User::class);

echo $row->name;            // access attributes
echo $row->reverse_name(); // or methods defined on the 'User' class
```

getRowArray()

与上面的 `getRow()` 方法相同, 只是它返回数组。例如:

```
<?php

$query = $db->query('YOUR QUERY');

$row = $query->getRowArray();

if (isset($row)) {
    echo $row['title'];
    echo $row['name'];
    echo $row['body'];
}
```

如果要返回特定的行, 可以在第一个参数中提交行号作为数字:

```
<?php

$row = $query->getRowArray(5);
```

此外, 你可以通过这些变体在结果中向前/向后/第一行/最后一行移动:

```
$row = $query->getFirstRow()
$row = $query->getLastRow()
$row = $query->getNextRow()
$row = $query->getPreviousRow()
```

默认情况下, 除非在参数中放入 "array" 字样, 否则它们返回对象:

```
$row = $query->getFirstRow('array')
$row = $query->getLastRow('array')
$row = $query->getNextRow('array')
$row = $query->getPreviousRow('array')
```

备注: 上面的所有方法都会将整个结果集加载到内存中 (预取)。对于处理大型结果集, 请使用 `getUnbufferedRow()`。

---

getUnbufferedRow()

此方法返回单个结果集, 而不像 `getRow()` 那样在内存中预取全部结果。如果查询有多个行, 它会返回当前行并将内部数据指针向前移动。

```
<?php

$query = $db->query('YOUR QUERY');

while ($row = $query->getUnbufferedRow()) {
    echo $row->title;
    echo $row->name;
    echo $row->body;
}
```

对于使用 MySQLi, 你可以将 MySQLi 的结果模式设置为 `MYSQLI_USE_RESULT`, 以节

省最大内存。使用这种方式通常不推荐, 但在某些情况下可能是有益的, 例如将大型查询写入 csv。如果更改结果模式, 请注意与之相关的权衡。

```
<?php

$db->resultMode = MYSQLI_USE_RESULT; // for unbuffered results

$query = $db->query('YOUR QUERY');

$file = new \CodeIgniter\Files\File(WRITEPATH . 'data.csv');

$csv = $file->openFile('w');

while ($row = $query->getUnbufferedRow('array')) {
    $csv->fputcsv($row);
}

$db->resultMode = MYSQLI_STORE_RESULT; // return to default mode
```

备注: 在使用 `MYSQLI_USE_RESULT` 时, 在所有记录被提取或进行 `freeResult()` 调用之前, 对同一连接的后续所有调用都将导致错误。`getNumRows()` 方法将仅基于数据指针的当前位置返回行数。MyISAM 表将保持锁定, 直到提取了所有记录或进行了 `freeResult()` 调用。

你可以选择传递 'object' (默认) 或 'array' 以指定返回值的类型:

```
<?php

$query->getUnbufferedRow();          // object
$query->getUnbufferedRow('object'); // object
$query->getUnbufferedRow('array');  // associative array
```

自定义结果对象

你可以让结果作为 stdClass 或数组的自定义类的实例返回, 正如 getResult() 和 getResultArray() 方法允许的那样。如果类还未加载到内存中, 则会自动加载。该对象将具有从数据库设置为属性的所有返回值。如果这些已声明且非公共, 则应提供 __set() 方法以允许设置它们。

例子:

```
<?php

namespace App\Entities;

class User
{
    public $id;
    public $email;
    public $username;

    protected $lastLogin;

    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};
        }
```

(续下页)

(接上页)

```
    }
}
```

除了下面列出的两种方法外，以下方法也可以使用类名称将结果返回为:getFirstRow()、getLastRow()、getNextRow() 和 getPreviousRow()。

getCustomResultObject()

将整个结果集作为请求类的实例数组返回。唯一的参数是要实例化的类的名称。

例子:

```
<?php

$query = $db->query('YOUR QUERY');

$rows = $query->getCustomResultObject(\App\Entities\User::class);

foreach ($rows as $row) {
    echo $row->id;
    echo $row->email;
    echo $row->lastLogin('Y-m-d');
}
```

getCustomRowObject()

从查询结果中返回单行。第一个参数是结果的行号。第二个参数是要实例化的类名称。

例子:

```
<?php

$query = $db->query('YOUR QUERY');

$row = $query->getCustomRowObject(0, \App\Entities\User::class);

if (isset($row)) {
```

(续下页)

(接上页)

```
    echo $row->email;              // access attributes
    echo $row->lastLogin('Y-m-d'); // access class methods
}
```

你也可以以完全相同的方式使用 `getRow()` 方法。

例子:

```
<?php

$row = $query->getRow(0, \App\Entities\User::class);
```

结果辅助方法

getFieldCount()

查询返回的字段 (列) 数。请确保使用查询结果对象调用该方法:

```
<?php

$query = $db->query('SELECT * FROM my_table');

echo $query->getFieldCount();
```

getFieldNames()

以数组形式返回查询返回的字段 (列) 的名称。请确保使用查询结果对象调用该方法:

```
<?php

$query = $db->query('SELECT * FROM my_table');

echo $query->getFieldNames();
```

getNumRows()

查询返回的记录数。请确保使用查询结果对象调用该方法:

```
<?php

$query = $db->query('SELECT * FROM my_table');

echo $query->getNumRows();
```

备注: 因为 SQLite3 缺乏有效的返回记录数的方法, CodeIgniter 将在内部提取和缓冲查询结果记录, 并返回生成的记录数组的计数, 这可能效率低下。

freeResult()

它释放与结果相关的内存并删除结果资源 ID。通常 PHP 会在脚本执行结束时自动释放其内存。但是, 如果在特定脚本中运行了大量查询, 你可能希望在生成每个查询结果后释放结果, 以减少内存消耗。

例子:

```
<?php

$query = $thisdb->query('SELECT title FROM my_table');

foreach ($query->getResult() as $row) {
    echo $row->title;
}

$query->freeResult(); // The $query result object will no longer be␣
↪available

$query2 = $db->query('SELECT name FROM some_table');

$row = $query2->getRow();
echo $row->name;
```

(续下页)

(接上页)

```
$query2->freeResult(); // The $query2 result object will no longer␣
↪be available
```

dataSeek()

此方法将下一个要提取的结果集的内部指针设置为。它仅与 `getUnbufferedRow()` 结合使用时才有用。

它接受一个正整数值, 默认为 0 并在成功时返回 true, 失败时返回 false。

```
<?php

$query = $db->query('SELECT `field_name` FROM `table_name`');
$query->dataSeek(5); // Skip the first 5 rows
$row = $query->getUnbufferedRow();
```

备注: 并非所有数据库驱动程序都支持此功能并会返回 false。最明显的是 - 你将无法与 PDO 一起使用它。

类参考

class CodeIgniter\Database**BaseResult**

getResult([*$type = 'object'*])

参数

- **$type** (string) –请求结果的类型 - array、object 或类名

返回

包含提取行的数组

返回类型

array

`getResultArray()`、`getResultObject()` 和 `getCustomResultObject()` 方法的包装器。

用法: 参见结果数组 。

getResultArray()

返回
包含提取行的数组

返回类型
array

将查询结果作为行数组返回, 其中每行本身是一个关联数组。

用法: 参见结果数组 。

getResultObject()

返回
包含提取行的数组

返回类型
array

将查询结果作为行数组返回, 其中每行是一个 stdClass 类型的对象。

用法: 参见获取 *stdClass* 的数组 。

getCustomResultObject(*$class_name*)

参数

- **$class_name** (string) –结果集的类名

返回
包含提取行的数组

返回类型
array

将查询结果作为行数组返回, 其中每行是指定类的实例。

getRow([*$n = 0*[, *$type = 'object'*]])

参数

- **$n** (int) –要返回的查询结果集的索引
- **$type** (string) –请求结果的类型 - array、object 或类名

返回
请求的行或不存在则为 null

返回类型

`mixed`

`getRowArray()`、`getRowObject()` 和 `getCustomRowObject()` 方法的包装器。

用法: 参见结果集 。

getUnbufferedRow([*$type = 'object'*])

参数

- **$type** (`string`) –请求结果的类型 - array、object 或类名

返回

结果集的下一行或不存在则为 null

返回类型

`mixed`

获取下一行结果并以请求的形式返回。

用法: 参见结果集 。

getRowArray([*$n = 0*])

参数

- **$n** (`int`) –要返回的查询结果集的索引

返回

请求的行或不存在则为 null

返回类型

`array`

将请求的结果集作为关联数组返回。

用法: 参见结果集 。

getRowObject([*$n = 0*])

参数

- **$n** (`int`) –要返回的查询结果集的索引

返回

请求的行或不存在则为 null

返回类型

stdClass

将请求的结果集作为 stdClass 类型的对象返回。

用法: 参见结果集 。

getCustomRowObject(*$n*, *$type*)

参数

- **$n** (int) –要返回的结果集的索引
- **$class_name** (string) –结果集的类名

返回

请求的行或不存在则为 null

返回类型

$type

将请求的结果集作为请求类的实例返回。

dataSeek([*$n = 0*])

参数

- **$n** (int) –下一个要返回的结果集的索引

返回

成功则为 true, 失败则为 false

返回类型

bool

将内部结果集指针移动到所需的偏移量。

用法: 参见结果辅助方法。

setRow(*$key*[, *$value = null*])

参数

- **$key** (mixed) –列名称或键/值对的数组
- **$value** (mixed) –如果 $key 是单个字段名, 则分配给该列的值

返回类型

void

为特定列赋值。

getNextRow([*$type = 'object'*])

参数

- **$type** (string) –请求结果的类型 - array、object 或类名

返回

结果集的下一行, 不存在则为 null

返回类型

mixed

从结果集返回下一行。

getPreviousRow([*$type = 'object'*])

参数

- **$type** (string) –请求结果的类型 - array、object 或类名

返回

结果集的上一行, 不存在则为 null

返回类型

mixed

从结果集返回上一行。

getFirstRow([*$type = 'object'*])

参数

- **$type** (string) –请求结果的类型 - array、object 或类名

返回

结果集的第一行, 不存在则为 null

返回类型

mixed

从结果集返回第一行。

getLastRow([*$type = 'object'*])

参数

- **$type** (string) –请求结果的类型 - array、object 或类名

返回

结果集的最后一行, 不存在则为 null

返回类型

`mixed`

从结果集返回最后一行。

getFieldCount()

返回

结果集中的字段数

返回类型

`int`

返回结果集中的字段数。

用法: 参见结果辅助方法。

getFieldNames()

返回

列名称数组

返回类型

`array`

返回结果集中包含的字段名称数组。

getFieldData()

返回

包含字段元数据的数组

返回类型

`array`

生成包含字段元数据的 `stdClass` 对象数组。

getNumRows()

返回

结果集中的行数

返回类型

`int`

返回查询返回的行数

`freeResult()`

返回类型

`void`

释放结果集。

用法: 参见结果辅助方法。

6.1.6 查询辅助方法

- 执行查询的信息
  - *$db->insertID()*
  - *$db->affectedRows()*
  - *$db->getLastQuery()*
- 有关数据库的信息
  - *$db->countAll()*
  - *$db->countAllResults()*
  - *$db->getPlatform()*
  - *$db->getVersion()*

执行查询的信息

$db->insertID()

执行数据库插入时的插入 ID 号。

备注: 如果使用 PDO 驱动程序与 PostgreSQL 一起使用, 或者使用 Interbase 驱动程序, 此函数需要一个 $name 参数, 该参数指定要检查插入 ID 的适当序列。

$db->affectedRows()

显示受影响的行数, 当执行 “写入” 类型的查询时 (插入、更新等)。

备注: 在 MySQL 中,” DELETE FROM TABLE” 返回 0 受影响的行。数据库类对此进行了一个小 Hack, 允许它返回正确的受影响行数。默认情况下, 此 Hack 已启用, 但可以在数据库驱动程序文件中将其关闭。

$db->getLastQuery()

返回代表最后执行的查询的 Query 对象 (查询字符串, 而不是结果)。

有关数据库的信息

$db->countAll()

允许你确定特定表中的行数。在第一个参数中提交表名。这是查询构建器的一部分。

```
<?php

echo $db->table('my_table')->countAll();
// Produces an integer, like 25
```

$db->countAllResults()

允许你确定特定结果集中的行数。在第一个参数中提交表名。这是查询构建器的一部分。

```
<?php

echo $db->table('my_table')->like('title', 'match')->
↪countAllResults();
// Produces an integer, like 5
```

$db->getPlatform()

输出你正在运行的数据库平台 (DBDriver)(MySQLi、SQLSRV、Postgre 等):

```
<?php

echo $db->getPlatform();
```

$db->getVersion()

输出你正在运行的数据库版本:

```
<?php

echo $db->getVersion();
```

6.1.7 查询构建器类

CodeIgniter 为你提供了访问查询构建器类的方式。该模式允许你通过最少的脚本执行数据库的检索、插入和更新操作。在某些情况下，只需一两行代码即可执行数据库操作。CodeIgniter 不要求每个数据库表对应一个独立的类文件，而是提供了一个更为简化的接口。

除了简单性，使用查询构建器功能的一个主要优势是它允许你创建与数据库无关的应用程序，因为查询语法由每个数据库适配器生成。它还允许更安全的查询，因为系统会自动对值进行转义。

备注： CodeIgniter 不支持表名和列名中包含点号（.）。自 v4.5.0 起，支持包含点号的数据库名。

- *SQL 注入防护*
- *加载查询构建器*
- *选择数据*
  - *Get*

  - *Select*
  - *From*
  - 子查询
  - *Join*
- 查找特定数据
  - *Where*
- 查找相似数据
  - *Like*
- 排序结果
  - *OrderBy*
- 限制或计数结果
  - *Limit*
- 联合查询
  - *Union*
- 查询分组
  - *Group*
- 插入数据
  - *Insert*
  - *insertBatch*
- 更新插入数据
  - *Upsert*
  - *upsertBatch*
- 更新数据
  - *Update*
  - *UpdateBatch*
- 删除数据

  - *Delete*
  - *DeleteBatch*
- 条件语句
  - *When*
  - *WhenNot*
- 方法链
- 重置查询构建器
  - *ResetQuery*
- 类参考

SQL 注入防护

你可以借助查询构建器相当安全地生成 SQL 语句。然而，它并非被设计为无论你传递什么数据都能防止 SQL 注入。

传递给查询构建器的参数可以是：

1. **标识符**，例如字段（或表）名
2. 它们的 **值**
3. **SQL 字符串**的一部分

查询构建器默认会对所有 **值**进行转义。

默认情况下，它也会尝试正确保护 **标识符**和 **SQL 字符串**中的标识符。然而，该实现是为了在多数用例中良好工作，并非设计用于防御所有攻击。因此，在未经适当验证的情况下，你绝不应将用户输入传递给它们。

此外，许多方法具有 `$escape` 参数，可以设置为禁用转义。如果 `$escape` 设置为 false，查询构建器将不提供任何保护，因此在传递给查询构建器之前，你必须自行确保它们已正确转义或保护。使用指定原始 SQL 语句的 `RawSql` 时也是如此。

加载查询构建器

查询构建器通过数据库连接上的 `table()` 方法加载。这会为你设置查询的 **FROM** 部分，并返回查询构建器类的新实例：

```
<?php

$db      = \Config\Database::connect();
$builder = $db->table('users');
```

只有在你明确请求该类时，查询构建器才会加载到内存中，因此默认情况下不会占用资源。

选择数据

以下方法允许你构建 SQL **SELECT** 语句。

Get

$builder->get()

运行选择查询并返回结果。可单独使用以从表中检索所有记录：

```
<?php

$builder = $db->table('mytable');
$query   = $builder->get();  // Produces: SELECT * FROM mytable
```

第一个和第二个参数允许你设置 limit 和 offset 子句：

```
<?php

$query = $builder->get(10, 20);
/*
 * Executes: SELECT * FROM mytable LIMIT 20, 10
 * (in MySQL. Other databases have slightly different syntax)
 */
```

你会注意到上述方法被赋值给名为 $query 的变量，该变量可用于显示结果：

```
<?php

$query = $builder->get();

foreach ($query->getResult() as $row) {
    echo $row->title;
}
```

关于结果生成的完整讨论，请访问*getResult()* 方法。

$builder->getCompiledSelect()

像 `$builder->get()` 一样编译选择查询，但不会 运行查询。此方法仅以字符串形式返回 SQL 查询。

示例：

```
<?php

$sql = $builder->getCompiledSelect();
echo $sql;
// Prints string: SELECT * FROM mytable
```

第一个参数（false）允许你设置查询构建器是否会被重置（因为参数的默认值为 true，即 `getCompiledSelect(bool $reset = true)`，默认情况下会像使用 `$builder->get()` 一样被重置）：

```
<?php

echo $builder->limit(10, 20)->getCompiledSelect(false);
/*
 * Prints string: SELECT * FROM mytable LIMIT 20, 10
 * (in MySQL. Other databases have slightly different syntax)
 */

echo $builder->select('title, content, date')->getCompiledSelect();
// Prints string: SELECT title, content, date FROM mytable LIMIT 20,
↪ 10
```

需要注意的关键点是，第二个查询未使用 `limit(10, 20)`，但生成的 SQL 查询包含 `LIMIT 20, 10`。造成此结果的原因是第一个查询的参数设置为 `false`，导致 `limit(10, 20)` 保留在第二个查询中。

$builder->getWhere()

与 `get()` 方法相同，区别在于它允许你在第一个参数中添加 "where" 子句，而无需使用 `$builder->where()` 方法：

```
<?php

$query = $builder->getWhere(['id' => $id], $limit, $offset);
```

有关 `where()` 方法的更多信息，请阅读下文。

Select

$builder->select()

允许你编写查询的 **SELECT** 部分：

```
<?php

$builder->select('title, content, date');
$query = $builder->get();
// Executes: SELECT title, content, date FROM mytable
```

备注： 如果从表中选择所有（`*`），则无需使用此方法。当省略时，CodeIgniter 假定你希望选择所有字段并自动添加 `SELECT *`。

`$builder->select()` 接受可选的第二个参数。如果将其设置为 `false`，CodeIgniter 将不会尝试保护你的字段或表名。这在需要复合 select 语句且自动转义字段可能破坏它们时非常有用。

```
<?php
```

(续下页)

(接上页)

```
$builder->select('(SELECT SUM(payments.amount) FROM payments WHERE␣
↪payments.invoice_id=4) AS amount_paid', false);
$query = $builder->get();
```

RawSql

在 4.2.0 版本加入.

自 v4.2.0 起，`$builder->select()` 接受表示原始 SQL 字符串的 `CodeIgniter\Database\RawSql` 实例。

```
<?php

use CodeIgniter\Database\RawSql;

$sql = 'REGEXP_SUBSTR(ral_anno,"[0-9]{1,2}([,.][0-9]{1,3})([,.][0-9]
↪{1,3})") AS ral';
$builder->select(new RawSql($sql));
$query = $builder->get();
```

> **警告：** 使用 `RawSql` 时，你必须手动转义值并保护标识符。否则可能导致 SQL 注入。

$builder->selectMax()

为你的查询编写 **SELECT MAX(field)** 部分。你可以选择包含第二个参数以重命名结果字段。

```
<?php

$builder->selectMax('age');
$query = $builder->get();
// Produces: SELECT MAX(age) as age FROM mytable

$builder->selectMax('age', 'member_age');
```

(续下页)

(接上页)

```
$query = $builder->get();
// Produces: SELECT MAX(age) as member_age FROM mytable
```

$builder->selectMin()

为你的查询编写 **SELECT MIN(field)** 部分。与 `selectMax()` 类似，你可以选择包含第二个参数以重命名结果字段。

```
<?php

$builder->selectMin('age');
$query = $builder->get();
// Produces: SELECT MIN(age) as age FROM mytable
```

$builder->selectAvg()

为你的查询编写 **SELECT AVG(field)** 部分。与 `selectMax()` 类似，你可以选择包含第二个参数以重命名结果字段。

```
<?php

$builder->selectAvg('age');
$query = $builder->get();
// Produces: SELECT AVG(age) as age FROM mytable
```

$builder->selectSum()

为你的查询编写 **SELECT SUM(field)** 部分。与 `selectMax()` 类似，你可以选择包含第二个参数以重命名结果字段。

```
<?php

$builder->selectSum('age');
$query = $builder->get();
// Produces: SELECT SUM(age) as age FROM mytable
```

$builder->selectCount()

为你的查询编写 **SELECT COUNT(field)** 部分。与 `selectMax()` 类似，你可以选择包含第二个参数以重命名结果字段。

备注: 此方法在与 `groupBy()` 结合使用时特别有用。对于一般计数，请参见 `countAll()` 或 `countAllResults()`。

```
<?php

$builder->selectCount('age');
$query = $builder->get();
// Produces: SELECT COUNT(age) as age FROM mytable
```

$builder->selectSubquery()

向 SELECT 部分添加子查询。

```
$subquery = $db->table('countries')->select('name')->where('id', 1);
$builder  = $db->table('users')->select('name')->selectSubquery(
↪$subquery, 'country');
$query    = $builder->get();
// Produces: SELECT `name`, (SELECT `name` FROM `countries` WHERE␣
↪`id` = 1) `country` FROM `users`
```

From

$builder->from()

允许你编写查询的 **FROM** 部分：

```
<?php

$builder = $db->table('users');
$builder->select('title, content, date');
```

(续下页)

(接上页)

```
$builder->from('mytable');
$query = $builder->get();
// Produces: SELECT title, content, date FROM users, mytable
```

备注: 如前所示，查询的 **FROM** 部分可以在 `$db->table()` 方法中指定。对 `from()` 的额外调用将向查询的 FROM 部分添加更多表。

子查询

$builder->fromSubquery()

允许你将 **FROM** 查询的一部分作为子查询编写。

这是将子查询添加到现有表的位置:

```
<?php

$subquery = $db->table('users');
$builder  = $db->table('jobs')->fromSubquery($subquery, 'alias');
$query    = $builder->get();
// Produces: SELECT * FROM `jobs`, (SELECT * FROM `users`) `alias`
```

使用 `$db->newQuery()` 方法将子查询作为主表:

```
<?php

$subquery = $db->table('users')->select('id, name');
$builder  = $db->newQuery()->fromSubquery($subquery, 't');
$query    = $builder->get();
// Produces: SELECT * FROM (SELECT `id`, `name` FROM users) `t`
```

Join

$builder->join()

允许你编写查询的 **JOIN** 部分：

```
<?php

$builder = $db->table('blogs');
$builder->select('*');
$builder->join('comments', 'comments.id = blogs.id');
$query = $builder->get();
/*
 * Produces:
 * SELECT * FROM blogs JOIN comments ON comments.id = blogs.id
 */
```

如果在一个查询中需要多个连接，可以进行多次方法调用。

如果需要特定类型的 **JOIN**，可以通过该方法的第三个参数指定。选项包括：`left`、`right`、`outer`、`inner`、`left outer` 和 `right outer`。

```
<?php

$builder->join('comments', 'comments.id = blogs.id', 'left');
// Produces: LEFT JOIN comments ON comments.id = blogs.id
```

RawSql

在 4.2.0 版本加入.

自 v4.2.0 起，`$builder->join()` 接受表示原始 SQL 字符串的 `CodeIgniter\Database\RawSql` 实例作为 JOIN ON 条件。

```
<?php

use CodeIgniter\Database\RawSql;

$sql = 'user.id = device.user_id AND ((1=1 OR 1=1) OR (1=1 OR 1=1))
```

(续下页)

(接上页)

```
↪';
$builder->join('user', new RawSql($sql), 'LEFT');
// Produces: LEFT JOIN "user" ON user.id = device.user_id AND ((1=1␣
↪OR 1=1) OR (1=1 OR 1=1))
```

> **警告:** 使用 `RawSql` 时，你必须手动转义值并保护标识符。否则可能导致 SQL 注入。

查找特定数据

Where

$builder->where()

此方法允许你使用五种方法之一设置 **WHERE** 子句:

备注: 传递给此方法的所有值都会自动转义，从而生成更安全的查询，除非使用自定义字符串。

备注: `$builder->where()` 接受可选的第三个参数。如果将其设置为 `false`，CodeIgniter 将不会尝试保护你的字段或表名。

---

1. 简单键/值方法

```
<?php

$builder->where('name', $name);
// Produces: WHERE name = 'Joe'
```

注意等号已为你添加。

如果多次调用该方法，它们将通过 **AND** 链接:

```
<?php

$builder->where('name', $name);
$builder->where('title', $title);
$builder->where('status', $status);
// WHERE name = 'Joe' AND title = 'boss' AND status =
↪'active'
```

2. 自定义键/值方法

你可以在第一个参数中包含运算符以控制比较:

```
<?php

$builder->where('name !=', $name);
$builder->where('id <', $id);
// Produces: WHERE name != 'Joe' AND id < 45
```

3. 关联数组方法

```
<?php

$array = ['name' => $name, 'title' => $title, 'status' =>
↪$status];
$builder->where($array);
// Produces: WHERE name = 'Joe' AND title = 'boss' AND␣
↪status = 'active'
```

你也可以使用此方法包含自己的运算符:

```
<?php

$array = ['name !=' => $name, 'id <' => $id, 'date >' =>
↪$date];
$builder->where($array);
```

4. 自定义字符串

你可以手动编写自己的子句:

```
<?php

$where = "name='Joe' AND status='boss' OR status='active'";
$builder->where($where);
```

> **警告:** 如果在字符串中使用用户提供的数据，你必须手动转义值并保护标识符。否则可能导致 SQL 注入。
>
> ```
> <?php
>
> $name  = $builder->db->escape('Joe');
> $where = "name={$name} AND status='boss' OR status=
> ↪'active'";
> $builder->where($where);
> ```

5. RawSql

在 4.2.0 版本加入.

自 v4.2.0 起，`$builder->where()` 接受表示原始 SQL 字符串的 `CodeIgniter\Database\RawSql` 实例。

```
<?php

use CodeIgniter\Database\RawSql;

$sql = "id > 2 AND name != 'Accountant'";
$builder->where(new RawSql($sql));
```

> **警告:** 使用 `RawSql` 时，你必须手动转义值并保护标识符。否则可能导致 SQL 注入。

6. 子查询

```
<?php

// With closure
use CodeIgniter\Database\BaseBuilder;

$builder->where('advance_amount <', static function␣
↪(BaseBuilder $builder) {
    $builder->select('MAX(advance_amount)', false)->from(
↪'orders')->where('id >', 2);
});
// Produces: WHERE "advance_amount" < (SELECT MAX(advance_
↪amount) FROM "orders" WHERE "id" > 2)

// With builder directly
$subQuery = $db->table('orders')->select('MAX(advance_
↪amount)', false)->where('id >', 2);
$builder->where('advance_amount <', $subQuery);
```

$builder->orWhere()

此方法与上述方法相同，区别在于多个实例通过 **OR** 连接：

```
<?php

$builder->where('name !=', $name);
$builder->orWhere('id >', $id);
// Produces: WHERE name != 'Joe' OR id > 50
```

$builder->whereIn()

生成 **WHERE field IN（'item'，'item'）** SQL 查询，并在适当时通过 **AND** 连接：

```
<?php

$names = ['Frank', 'Todd', 'James'];
```

(续下页)

(接上页)

```
$builder->whereIn('username', $names);
// Produces: WHERE username IN ('Frank', 'Todd', 'James')
```

你可以使用子查询代替值数组：

```
<?php

// With closure
use CodeIgniter\Database\BaseBuilder;

$builder->whereIn('id', static function (BaseBuilder $builder) {
    $builder->select('job_id')->from('users_jobs')->where('user_id',
↪ 3);
});
// Produces: WHERE "id" IN (SELECT "job_id" FROM "users_jobs" WHERE
↪"user_id" = 3)

// With builder directly
$subQuery = $db->table('users_jobs')->select('job_id')->where('user_
↪id', 3);
$builder->whereIn('id', $subQuery);
```

$builder->orWhereIn()

生成 **WHERE field IN（'item'，'item'）** SQL 查询，并在适当时通过 **OR** 连接：

```
<?php

$names = ['Frank', 'Todd', 'James'];
$builder->orWhereIn('username', $names);
// Produces: OR username IN ('Frank', 'Todd', 'James')
```

你可以使用子查询代替值数组：

```
<?php

// With closure
```

(续下页)

(接上页)

```
use CodeIgniter\Database\BaseBuilder;

$builder->orWhereIn('id', static function (BaseBuilder $builder) {
    $builder->select('job_id')->from('users_jobs')->where('user_id',
↪ 3);
});
// Produces: OR "id" IN (SELECT "job_id" FROM "users_jobs" WHERE
↪"user_id" = 3)

// With builder directly
$subQuery = $db->table('users_jobs')->select('job_id')->where('user_
↪id', 3);
$builder->orWhereIn('id', $subQuery);
```

$builder->whereNotIn()

生成 **WHERE field NOT IN ('item' , 'item')** SQL 查询，并在适当时通过 **AND** 连接：

```
<?php

$names = ['Frank', 'Todd', 'James'];
$builder->whereNotIn('username', $names);
// Produces: WHERE username NOT IN ('Frank', 'Todd', 'James')
```

你可以使用子查询代替值数组：

```
<?php

// With closure
use CodeIgniter\Database\BaseBuilder;

$builder->whereNotIn('id', static function (BaseBuilder $builder) {
    $builder->select('job_id')->from('users_jobs')->where('user_id',
↪ 3);
});
// Produces: WHERE "id" NOT IN (SELECT "job_id" FROM "users_jobs"␣
↪WHERE "user_id" = 3)
```

(续下页)

(接上页)

```
// With builder directly
$subQuery = $db->table('users_jobs')->select('job_id')->where('user_
↪id', 3);
$builder->whereNotIn('id', $subQuery);
```

$builder->orWhereNotIn()

生成 **WHERE field NOT IN ('item' , 'item')** SQL 查询，并在适当时通过 **OR** 连接：

```
<?php

$names = ['Frank', 'Todd', 'James'];
$builder->orWhereNotIn('username', $names);
// Produces: OR username NOT IN ('Frank', 'Todd', 'James')
```

你可以使用子查询代替值数组：

```
<?php

// With closure
use CodeIgniter\Database\BaseBuilder;

$builder->orWhereNotIn('id', static function (BaseBuilder $builder)
↪{
    $builder->select('job_id')->from('users_jobs')->where('user_id',
↪ 3);
});
// Produces: OR "id" NOT IN (SELECT "job_id" FROM "users_jobs"␣
↪WHERE "user_id" = 3)

// With builder directly
$subQuery = $db->table('users_jobs')->select('job_id')->where('user_
↪id', 3);
$builder->orWhereNotIn('id', $subQuery);
```

查找相似数据

Like

$builder->like()

此方法允许你生成 **LIKE** 子句，适用于执行搜索。

备注： 传递给此方法的所有值都会自动转义。

备注： 所有 `like*` 方法变体可以通过将第五个参数设置为 `true` 来强制执行不区分大小写的搜索。这将使用平台特定的功能（如果可用），否则将强制将值转换为小写，即 `WHERE LOWER(column) LIKE '%search%'`。这可能需要在 `LOWER(column)` 而非 `column` 上创建索引才能生效。

---

1. 简单键/值方法

```
<?php

$builder->like('title', 'match');
// Produces: WHERE `title` LIKE '%match%' ESCAPE '!'
```

如果多次调用该方法，它们将通过 **AND** 链接：

```
<?php

$builder->like('title', 'match');
$builder->like('body', 'match');
// WHERE `title` LIKE '%match%' ESCAPE '!' AND  `body`␣
↪LIKE '%match%' ESCAPE '!'
```

如果要控制通配符（% ）的位置，可以使用可选的第三个参数。选项为 `before`、`after` 和 `both` （默认）。

```
<?php

$builder->like('title', 'match', 'before'); // Produces:␣
↪WHERE `title` LIKE '%match' ESCAPE '!'
$builder->like('title', 'match', 'after');  // Produces:␣
↪WHERE `title` LIKE 'match%' ESCAPE '!'
$builder->like('title', 'match', 'both');   // Produces:␣
↪WHERE `title` LIKE '%match%' ESCAPE '!'
```

2. 关联数组方法

```
<?php

$array = ['title' => $match, 'page1' => $match, 'page2' =>
↪$match];
$builder->like($array);
/*
 * WHERE `title` LIKE '%match%' ESCAPE '!'
 *     AND  `page1` LIKE '%match%' ESCAPE '!'
 *     AND  `page2` LIKE '%match%' ESCAPE '!'
 */
```

3. RawSql

在 4.2.0 版本加入.

自 v4.2.0 起，`$builder->like()` 接受表示原始 SQL 字符串的 `CodeIgniter\Database\RawSql` 实例。

```
<?php

use CodeIgniter\Database\RawSql;

$sql    = "CONCAT(users.name, ' ', IF(users.surname IS␣
↪NULL OR users.surname = '', '', users.surname))";
$rawSql = new RawSql($sql);
$builder->like($rawSql, 'value', 'both');
```

> **警告:** 使用 `RawSql` 时，你必须手动转义值并保护标识符。否则可能导致 SQL 注入。

$builder->orLike()

此方法与上述方法相同，区别在于多个实例通过 **OR** 连接:

```
<?php

$builder->like('title', 'match');
$builder->orLike('body', $match);
// WHERE `title` LIKE '%match%' ESCAPE '!' OR  `body` LIKE '%match%
↪' ESCAPE '!'
```

$builder->notLike()

此方法与 `like()` 相同，但生成 **NOT LIKE** 语句:

```
<?php

$builder->notLike('title', 'match'); // WHERE `title` NOT LIKE '
↪%match% ESCAPE '!'
```

$builder->orNotLike()

此方法与 `notLike()` 相同，但多个实例通过 **OR** 连接:

```
<?php

$builder->like('title', 'match');
$builder->orNotLike('body', 'match');
// WHERE `title` LIKE '%match% OR  `body` NOT LIKE '%match%' ESCAPE
↪'!'
```

$builder->groupBy()

允许你编写查询的 **GROUP BY** 部分：

```
<?php

$builder->groupBy('title');
// Produces: GROUP BY title
```

你也可以传递多个值的数组：

```
<?php

$builder->groupBy(['title', 'date']);
// Produces: GROUP BY title, date
```

$builder->distinct()

向查询添加 **DISTINCT** 关键字：

```
<?php

$builder->distinct();
$builder->get();
// Produces: SELECT DISTINCT * FROM mytable
```

$builder->having()

允许你编写查询的 **HAVING** 部分。有两种可能的语法，1 个参数或 2 个：

```
<?php

$builder->having('user_id = 45'); // Produces: HAVING user_id = 45
$builder->having('user_id', 45); // Produces: HAVING user_id = 45
```

你也可以传递多个值的数组：

```
<?php

$builder->having(['title =' => 'My Title', 'id <' => $id]);
// Produces: HAVING title = 'My Title', id < 45
```

如果你使用的数据库由 CodeIgniter 转义值，可以通过传递可选的第三个参数并设置为 `false` 来防止转义内容。

```
<?php

$builder->having('user_id', 45); // Produces: HAVING `user_id` = 45␣
↪in some databases such as MySQL
$builder->having('user_id', 45, false); // Produces: HAVING user_id␣
↪= 45
```

$builder->orHaving()

与 `having()` 相同，但多个子句通过 **OR** 分隔。

$builder->havingIn()

生成 **HAVING field IN（'item'，'item'）** SQL 查询，并在适当时通过 **AND** 连接：

```
<?php

$groups = [1, 2, 3];
$builder->havingIn('group_id', $groups);
// Produces: HAVING group_id IN (1, 2, 3)
```

你可以使用子查询代替值数组：

```
<?php

// With closure
use CodeIgniter\Database\BaseBuilder;

$builder->havingIn('id', static function (BaseBuilder $builder) {
```

(续下页)

(接上页)

```
    $builder->select('user_id')->from('users_jobs')->where('group_id
↪', 3);
});
// Produces: HAVING "id" IN (SELECT "user_id" FROM "users_jobs"␣
↪WHERE "group_id" = 3)

// With builder directly
$subQuery = $db->table('users_jobs')->select('user_id')->where(
↪'group_id', 3);
$builder->havingIn('id', $subQuery);
```

$builder->orHavingIn()

生成 **HAVING field IN (‘item’, ‘item’)** SQL 查询，并在适当时通过 **OR** 连接：

```
<?php

$groups = [1, 2, 3];
$builder->orHavingIn('group_id', $groups);
// Produces: OR group_id IN (1, 2, 3)
```

你可以使用子查询代替值数组：

```
<?php

// With closure
use CodeIgniter\Database\BaseBuilder;

$builder->orHavingIn('id', static function (BaseBuilder $builder) {
    $builder->select('user_id')->from('users_jobs')->where('group_id
↪', 3);
});
// Produces: OR "id" IN (SELECT "user_id" FROM "users_jobs" WHERE
↪"group_id" = 3)

// With builder directly
$subQuery = $db->table('users_jobs')->select('user_id')->where(
```

(续下页)

(接上页)

```
↪'group_id', 3);
$builder->orHavingIn('id', $subQuery);
```

$builder->havingNotIn()

生成 **HAVING field NOT IN ('item' , 'item')** SQL 查询，并在适当时通过 **AND** 连接：

```
<?php

$groups = [1, 2, 3];
$builder->havingNotIn('group_id', $groups);
// Produces: HAVING group_id NOT IN (1, 2, 3)
```

你可以使用子查询代替值数组：

```
<?php

// With closure
use CodeIgniter\Database\BaseBuilder;

$builder->havingNotIn('id', static function (BaseBuilder $builder) {
    $builder->select('user_id')->from('users_jobs')->where('group_id
↪', 3);
});
// Produces: HAVING "id" NOT IN (SELECT "user_id" FROM "users_jobs"␣
↪WHERE "group_id" = 3)

// With builder directly
$subQuery = $db->table('users_jobs')->select('user_id')->where(
↪'group_id', 3);
$builder->havingNotIn('id', $subQuery);
```

$builder->orHavingNotIn()

生成 **HAVING field NOT IN ('item' , 'item')** SQL 查询，并在适当时通过 **OR** 连接：

```
<?php

$groups = [1, 2, 3];
$builder->havingNotIn('group_id', $groups);
// Produces: OR group_id NOT IN (1, 2, 3)
```

你可以使用子查询代替值数组：

```
<?php

// With closure
use CodeIgniter\Database\BaseBuilder;

$builder->orHavingNotIn('id', static function (BaseBuilder
↪$builder) {
    $builder->select('user_id')->from('users_jobs')->where('group_id
↪', 3);
});
// Produces: OR "id" NOT IN (SELECT "user_id" FROM "users_jobs"␣
↪WHERE "group_id" = 3)

// With builder directly
$subQuery = $db->table('users_jobs')->select('user_id')->where(
↪'group_id', 3);
$builder->orHavingNotIn('id', $subQuery);
```

$builder->havingLike()

此方法允许你为查询的 **HAVING** 部分生成 **LIKE** 子句，适用于执行搜索。

备注： 传递给此方法的所有值都会自动转义。

备注： 所有 `havingLike*()` 方法变体可以通过将第五个参数设置为 `true` 来强制

执行不区分大小写的搜索。这将使用平台特定的功能（如果可用），否则将强制将值转换为小写，即 `HAVING LOWER(column) LIKE '%search%'`。这可能需要在 `LOWER(column)` 而非 `column` 上创建索引才能生效。

1. 简单键/值方法

```
<?php

$builder->havingLike('title', 'match');
// Produces: HAVING `title` LIKE '%match%' ESCAPE '!'
```

如果多次调用该方法，它们将通过 **AND** 链接：

```
<?php

$builder->havingLike('title', 'match');
$builder->havingLike('body', 'match');
// HAVING `title` LIKE '%match%' ESCAPE '!' AND  `body`␣
↪LIKE '%match% ESCAPE '!'
```

如果要控制通配符（% ）的位置，可以使用可选的第三个参数。选项为 `before`、`after` 和 `both` （默认）。

```
<?php

$builder->havingLike('title', 'match', 'before'); //␣
↪Produces: HAVING `title` LIKE '%match' ESCAPE '!'
$builder->havingLike('title', 'match', 'after');  //␣
↪Produces: HAVING `title` LIKE 'match%' ESCAPE '!'
$builder->havingLike('title', 'match', 'both');   //␣
↪Produces: HAVING `title` LIKE '%match%' ESCAPE '!'
```

2. 关联数组方法

```
<?php

$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()

此方法与上述方法相同，区别在于多个实例通过 **OR** 连接：

```
<?php

$builder->havingLike('title', 'match');
$builder->orHavingLike('body', $match);
// HAVING `title` LIKE '%match%' ESCAPE '!' OR  `body` LIKE '%match%
↪' ESCAPE '!'
```

$builder->notHavingLike()

此方法与 `havingLike()` 相同，但生成 **NOT LIKE** 语句：

```
<?php

$builder->notHavingLike('title', 'match');
// HAVING `title` NOT LIKE '%match% ESCAPE '!'
```

$builder->orNotHavingLike()

此方法与 `notHavingLike()` 相同，但多个实例通过 **OR** 连接：

```
<?php

$builder->havingLike('title', 'match');
$builder->orNotHavingLike('body', 'match');
// HAVING `title` LIKE '%match% OR  `body` NOT LIKE '%match%'␣
↪ESCAPE '!'
```

排序结果

OrderBy

$builder->orderBy()

允许你设置 **ORDER BY** 子句。

第一个参数包含要排序的列名。

第二个参数允许你设置结果的方向。选项为 `ASC`、`DESC` 和 `RANDOM`。

```
<?php

$builder->orderBy('title', 'DESC');
// Produces: ORDER BY `title` DESC
```

你也可以在第一个参数中传递自己的字符串：

```
<?php

$builder->orderBy('title DESC, name ASC');
// Produces: ORDER BY `title` DESC, `name` ASC
```

如果需要多个字段，可以进行多次方法调用。

```
<?php
```

(续下页)

(接上页)

```
$builder->orderBy('title', 'DESC');
$builder->orderBy('name', 'ASC');
// Produces: ORDER BY `title` DESC, `name` ASC
```

如果选择 RANDOM 方向选项，除非指定数字种子值，否则将忽略第一个参数。

```
<?php

$builder->orderBy('title', 'RANDOM');
// Produces: ORDER BY RAND()

$builder->orderBy(42, 'RANDOM');
// Produces: ORDER BY RAND(42)
```

限制或计数结果

Limit

$builder->limit()

允许你限制查询返回的行数：

```
<?php

$builder->limit(10);
// Produces: LIMIT 10
```

备注： 如果在 SQL 语句中指定了 `LIMIT 0`，将返回 0 条记录。然而，在查询构建器中存在一个错误：如果指定 `limit(0)`，生成的 SQL 语句将没有 `LIMIT` 子句，并返回所有记录。在 v4.5.0 中添加了设置以修复此错误行为。详细信息请参阅*limit(0) 行为*。此错误行为将在未来版本中修复，因此建议你更改默认设置。

第二个参数允许你设置结果偏移量。

```
<?php

$builder->limit(10, 20);
// Produces: LIMIT 20, 10 (in MySQL. Other databases have slightly␣
↪different syntax)
```

$builder->countAllResults()

允许你确定特定查询构建器查询中的行数。查询支持查询构建器的限制条件，如 `where()`、`orWhere()`、`like()`、`orLike()` 等。示例：

```
<?php

echo $builder->countAllResults(); // Produces an integer, like 25
$builder->like('title', 'match');
$builder->from('my_table');
echo $builder->countAllResults(); // Produces an integer, like 17
```

但是，此方法还会重置你可能传递给 `select()` 的任何字段值。如果需要保留它们，可以将第一个参数传递为 `false`。

```
<?php

echo $builder->countAllResults(false); // Produces an integer, like␣
↪17
```

$builder->countAll()

允许你确定特定表中的行数。示例：

```
<?php

echo $builder->countAll(); // Produces an integer, like 25
```

与 `countAllResult()` 方法类似，此方法也会重置你可能传递给 `select()` 的任何字段值。如果需要保留它们，可以将第一个参数传递为 `false`。

联合查询

Union

$builder->union()

用于组合两个或多个 SELECT 语句的结果集。它将仅返回唯一结果。

```
<?php

$builder = $db->table('users')->select('id, name')->limit(10);
$union   = $db->table('groups')->select('id, name');
$builder->union($union)->get();
/*
 * Produces:
 * SELECT * FROM (SELECT `id`, `name` FROM `users` LIMIT 10) uwrp0
 * UNION SELECT * FROM (SELECT `id`, `name` FROM `groups`) uwrp1
 */
```

备注: 为了与 DBMS（如 MSSQL 和 Oracle）正确配合，查询会被包装在 `SELECT * FROM ( ... ) alias` 中。主查询将始终具有别名 `uwrp0`。通过 `union()` 添加的每个后续查询将具有别名 `uwrpN+1`。

所有联合查询将添加在主查询之后，无论 `union()` 方法的调用顺序如何。也就是说，即使在调用 `union()` 之后调用 `limit()` 或 `orderBy()` 方法，这些方法也将相对于主查询。

在某些情况下，可能需要对查询结果进行排序或限制记录数。解决方案是使用通过 `$db->newQuery()` 创建的包装器。在下面的示例中，我们获取前 5 个用户 + 后 5 个用户并按 id 排序结果：

```
<?php

$union   = $db->table('users')->select('id, name')->orderBy('id',
↪'DESC')->limit(5);
$builder = $db->table('users')->select('id, name')->orderBy('id',
↪'ASC')->limit(5)->union($union);
```

(续下页)

(接上页)

```
$db->newQuery()->fromSubquery($builder, 'q')->orderBy('id', 'DESC')-
↪>get();
/*
 * Produces:
 * SELECT * FROM (
 *      SELECT * FROM (SELECT `id`, `name` FROM `users` ORDER BY␣
↪`id` ASC LIMIT 5) uwrp0
 *      UNION
 *      SELECT * FROM (SELECT `id`, `name` FROM `users` ORDER BY␣
↪`id` DESC LIMIT 5) uwrp1
 * ) q ORDER BY `id` DESC
 */
```

$builder->unionAll()

行为与 `union()` 方法相同。但是，将返回所有结果，而不仅仅是唯一结果。

查询分组

Group

查询分组允许你通过将 **WHERE** 子句括在括号中来创建分组。这将允许你创建具有复杂 **WHERE** 子句的查询。支持嵌套分组。示例：

```
<?php

$builder->select('*')->from('my_table')
    ->groupStart()
        ->where('a', 'a')
        ->orGroupStart()
            ->where('b', 'b')
            ->where('c', 'c')
        ->groupEnd()
    ->groupEnd()
    ->where('d', 'd')
```

(续下页)

(接上页)

```
    ->get();
/*
 * Generates:
 * SELECT * FROM (`my_table`) WHERE ( `a` = 'a' OR ( `b` = 'b' AND␣
↪`c` = 'c' ) ) AND `d` = 'd'
 */
```

备注: 分组需要平衡，确保每个 `groupStart()` 都有对应的 `groupEnd()`。

$builder->groupStart()

通过向查询的 **WHERE** 子句添加左括号开始新分组。

$builder->orGroupStart()

通过向查询的 **WHERE** 子句添加左括号开始新分组，并添加前缀 **OR**。

$builder->notGroupStart()

通过向查询的 **WHERE** 子句添加左括号开始新分组，并添加前缀 **NOT**。

$builder->orNotGroupStart()

通过向查询的 **WHERE** 子句添加左括号开始新分组，并添加前缀 **OR NOT**。

$builder->groupEnd()

通过向查询的 **WHERE** 子句添加右括号结束当前分组。

$builder->havingGroupStart()

通过向查询的 **HAVING** 子句添加左括号开始新分组。

$builder->orHavingGroupStart()

通过向查询的 **HAVING** 子句添加左括号开始新分组，并添加前缀 **OR**。

$builder->notHavingGroupStart()

通过向查询的 **HAVING** 子句添加左括号开始新分组，并添加前缀 **NOT**。

$builder->orNotHavingGroupStart()

通过向查询的 **HAVING** 子句添加左括号开始新分组，并添加前缀 **OR NOT**。

$builder->havingGroupEnd()

通过向查询的 **HAVING** 子句添加右括号结束当前分组。

插入数据

Insert

$builder->insert()

根据你提供的数据生成 insert 字符串并运行查询。你可以向该方法传递 **数组**或 **对象**。以下是使用数组的示例：

```
<?php

use CodeIgniter\Database\RawSql;

$data = [
    'id'          => new RawSql('DEFAULT'),
    'title'       => 'My title',
```

(续下页)

(接上页)

```
    'name'        => 'My Name',
    'date'        => '2022-01-01',
    'last_update' => new RawSql('CURRENT_TIMESTAMP()'),
];

$builder->insert($data);
/* Produces:
    INSERT INTO mytable (id, title, name, date, last_update)
    VALUES (DEFAULT, 'My title', 'My name', '2022-01-01', CURRENT_
↪TIMESTAMP())
*/
```

第一个参数是值的关联数组。

备注: 除 RawSql 外，所有值都会自动转义，生成更安全的查询。

警告: 使用 RawSql 时，你必须手动转义数据。否则可能导致 SQL 注入。

以下是使用对象的示例：

```
<?php

namespace App\Libraries;

class MyClass
{
    public $title   = 'My Title';
    public $content = 'My Content';
    public $date    = 'My Date';
}
```

```
<?php

use App\Libraries\MyClass;
```

(续下页)

(接上页)

```
$object = new MyClass();
$builder->insert($object);
// Produces: INSERT INTO mytable (title, content, date) VALUES ('My␣
↪Title', 'My Content', 'My Date')
```

第一个参数是一个对象。

$builder->ignore()

根据你提供的数据生成 insert ignore 字符串并运行查询。因此，如果具有相同主键的条目已存在，则不会插入该查询。你可以选择向该方法传递 **布尔值**。也可用于 **insertBatch**、**update** 和 **delete** （在支持时）。以下是使用上述示例数组的示例：

```
<?php

$data = [
    'title' => 'My title',
    'name'  => 'My Name',
    'date'  => 'My date',
];

$builder->ignore(true)->insert($data);
// Produces: INSERT OR IGNORE INTO mytable (title, name, date)␣
↪VALUES ('My title', 'My name', 'My date')
```

$builder->getCompiledInsert()

像 `$builder->insert()` 一样编译插入查询，但不会 运行查询。此方法仅以字符串形式返回 SQL 查询。

示例：

```
<?php

$data = [
    'title' => 'My title',
    'name'  => 'My Name',
```

(续下页)

(接上页)

```
    'date'  => 'My date',
];

$sql = $builder->set($data)->getCompiledInsert();
echo $sql;
// Produces string: INSERT INTO mytable (`title`, `name`, `date`)␣
↪VALUES ('My title', 'My name', 'My date')
```

第一个参数允许你设置是否重置查询构建器查询（默认情况下会重置，就像使用 `$builder->insert()` 一样）：

```
<?php

echo $builder->set('title', 'My Title')->getCompiledInsert(false);
// Produces string: INSERT INTO mytable (`title`) VALUES ('My Title
↪')

echo $builder->set('content', 'My Content')->getCompiledInsert();
// Produces string: INSERT INTO mytable (`title`, `content`) VALUES␣
↪('My Title', 'My Content')
```

第二个查询起作用的原因是第一个参数设置为 `false`。

备注： 此方法不适用于批量插入。

---

insertBatch

$builder->insertBatch()

从数据插入

根据你提供的数据生成 insert 字符串并运行查询。你可以向该方法传递 **数组**或 **对象**。以下是使用数组的示例：

```
<?php
```

(续下页)

(接上页)

```
$data = [
    [
        'title' => 'My title',
        'name'  => 'My Name',
        'date'  => 'My date',
    ],
    [
        'title' => 'Another title',
        'name'  => 'Another Name',
        'date'  => 'Another date',
    ],
];

$builder->insertBatch($data);
/*
 * Produces:
 * INSERT INTO mytable (title, name, date)
 *      VALUES ('My title', 'My name', 'My date'),
 *      ('Another title', 'Another name', 'Another date')
 */
```

第一个参数是值的关联数组。

备注: 除 `RawSql` 外，所有值都会自动转义，生成更安全的查询。

警告: 使用 `RawSql` 时，你必须手动转义数据。否则可能导致 SQL 注入。

从查询插入

你也可以从查询插入:

```
<?php

use CodeIgniter\Database\RawSql;
```

(续下页)

(接上页)

```
$query = 'SELECT user2.name, user2.email, user2.country
          FROM user2
          LEFT JOIN user ON user.email = user2.email
          WHERE user.email IS NULL';

$sql = $builder
    ->ignore(true)
    ->setQueryAsData(new RawSql($query), null, 'name, country, email
↪')
    ->insertBatch();
/* MySQLi produces:
    INSERT IGNORE INTO `db_user` (`name`, `country`, `email`)
    SELECT user2.name, user2.email, user2.country
    FROM user2
    LEFT JOIN user ON user.email = user2.email
    WHERE user.email IS NULL
*/
```

备注: 自 v4.3.0 起，可以使用 `setQueryAsData()`。

备注: 必须将 select 查询的列别名与目标表的列匹配。

更新插入数据

Upsert

$builder->upsert()

在 4.3.0 版本加入.

根据你提供的数据生成更新插入字符串并运行查询。你可以向该方法传递 **数组**或 **对象**。默认情况下，将按顺序定义约束。首先选择主键，然后是唯一键。MySQL 将默认使用任何约束。以下是使用数组的示例：

```
<?php

$data = [
    'email'   => 'ahmadinejad@example.com',
    'name'    => 'Ahmadinejad',
    'country' => 'Iran',
];

$builder->upsert($data);
// MySQLi  produces: INSERT INTO.. ON DUPLICATE KEY UPDATE..
// Postgre produces: INSERT INTO.. ON CONFLICT.. DO UPDATE..
// SQLite3 produces: INSERT INTO.. ON CONFLICT.. DO UPDATE..
// SQLSRV  produces: MERGE INTO.. WHEN MATCHED THEN UPDATE.. WHEN␣
↪NOT MATCHED THEN INSERT..
// OCI8    produces: MERGE INTO.. WHEN MATCHED THEN UPDATE.. WHEN␣
↪NOT MATCHED THEN INSERT..
```

第一个参数是值的关联数组。

以下是使用对象的示例:

```
<?php

namespace App\Libraries;

class MyClass
{
    public $email   = 'ahmadinejad@example.com';
    public $name    = 'Ahmadinejad';
    public $country = 'Iran';
}
```

```
<?php

use App\Libraries\MyClass;

$object = new MyClass();
$builder->upsert($object);
```

第一个参数是一个对象。

备注： 所有值都会自动转义，生成更安全的查询。

$builder->getCompiledUpsert()

在 4.3.0 版本加入.

像 `$builder->upsert()` 一样编译更新插入查询，但不会 运行查询。此方法仅以字符串形式返回 SQL 查询。

示例：

```
<?php

$data = [
    'email'   => 'ahmadinejad@example.com',
    'name'    => 'Ahmadinejad',
    'country' => 'Iran',
];

$sql = $builder->setData($data)->getCompiledUpsert();
echo $sql;
/* MySQLi produces:
    INSERT INTO `db_user` (`country`, `email`, `name`)
    VALUES ('Iran','ahmadinejad@example.com','Ahmadinejad')
    ON DUPLICATE KEY UPDATE
    `country` = VALUES(`country`),
    `email` = VALUES(`email`),
    `name` = VALUES(`name`)
*/
```

备注： 此方法不适用于批量更新插入。

upsertBatch

$builder->upsertBatch()

在 4.3.0 版本加入.

从数据更新插入

根据你提供的数据生成更新插入字符串并运行查询。你可以向该方法传递 **数组**或 **对象**。默认情况下，将按顺序定义约束。首先选择主键，然后是唯一键。MySQL 将默认使用任何约束。

以下是使用数组的示例：

```
<?php

$data = [
    [
        'id'      => 2,
        'email'   => 'ahmadinejad@example.com',
        'name'    => 'Ahmadinejad',
        'country' => 'Iran',
    ],
    [
        'id'      => null,
        'email'   => 'pedro@example.com',
        'name'    => 'Pedro',
        'country' => 'El Salvador',
    ],
];

$builder->upsertBatch($data);
// MySQLi  produces: INSERT INTO.. ON DUPLICATE KEY UPDATE..
// Postgre produces: INSERT INTO.. ON CONFLICT.. DO UPDATE..
// SQLite3 produces: INSERT INTO.. ON CONFLICT.. DO UPDATE..
// SQLSRV  produces: MERGE INTO.. WHEN MATCHED THEN UPDATE.. WHEN␣
↪NOT MATCHED THEN INSERT..
// OCI8    produces: MERGE INTO.. WHEN MATCHED THEN UPDATE.. WHEN␣
↪NOT MATCHED THEN INSERT..
```

第一个参数是值的关联数组。

备注: 所有值都会自动转义，生成更安全的查询。

从查询更新插入

你也可以从查询更新插入:

```
<?php

use CodeIgniter\Database\RawSql;

$query = $this->db->table('user2')
    ->select('user2.name, user2.email, user2.country')
    ->join('user', 'user.email = user2.email', 'left')
    ->where('user.email IS NULL');

$additionalUpdateField = ['updated_at' => new RawSql('CURRENT_
↪TIMESTAMP')];

$sql = $builder->setQueryAsData($query)
    ->onConstraint('email')
    ->updateFields($additionalUpdateField, true)
    ->upsertBatch();
/* MySQLi produces:
    INSERT INTO `db_user` (`country`, `email`, `name`)
    SELECT user2.name, user2.email, user2.country
    FROM user2
    LEFT JOIN user ON user.email = user2.email
    WHERE user.email IS NULL
    ON DUPLICATE KEY UPDATE
    `country` = VALUES(`country`),
    `email` = VALUES(`email`),
    `name` = VALUES(`name`),
    `updated_at` = CURRENT_TIMESTAMP
*/
```

备注: 自 v4.3.0 起，可以使用 `setQueryAsData()`、`onConstraint()` 和 `updateFields()` 方法。

备注: 必须将 select 查询的列别名与目标表的列匹配。

$builder->onConstraint()

在 4.3.0 版本加入.

允许手动设置用于更新插入的约束。这不适用于 MySQL，因为 MySQL 默认检查所有约束。

```
<?php

$data = [
    'id'      => 2,
    'email'   => 'ahmadinejad@example.com',
    'name'    => 'Ahmadinejad',
    'country' => 'Iran',
];

$builder->onConstraint('email')->upsert($data);
/* Postgre produces:
    INSERT INTO "db_user"("country", "email", "id", "name")
    VALUES ('Iran','ahmadinejad@example.com',2,'Ahmadinejad')
    ON CONFLICT("email")
    DO UPDATE SET
    "country" = "excluded"."country",
    "id" = "excluded"."id",
    "name" = "excluded"."name"
*/
```

此方法接受字符串或列数组。

$builder->updateFields()

在 4.3.0 版本加入.

允许手动设置执行更新插入时要更新的字段。

```
<?php

$data = [
    'id'      => 2,
    'email'   => 'ahmadinejad@example.com',
    'name'    => 'Ahmadinejad Zaghari',
    'country' => 'Afghanistan',
];

$builder->updateFields('name, country')->setData($data, null, '_
↪upsert')->upsert();
/* SQLSRV produces:
    MERGE INTO "test"."dbo"."db_user"
    USING (
     VALUES ('Iran','ahmadinejad@example.com',2,'Ahmadinejad')
    ) "_upsert" ("country", "email", "id", "name")
    ON ("test"."dbo"."db_user"."id" = "_upsert"."id")
    WHEN MATCHED THEN UPDATE SET
    "country" = "_upsert"."country",
    "name" = "_upsert"."name"
    WHEN NOT MATCHED THEN INSERT ("country", "email", "id", "name")
    VALUES ("_upsert"."country", "_upsert"."email", "_upsert"."id",
↪"_upsert"."name");
*/
```

此方法接受字符串、列数组或 RawSql。你还可以指定不包含在数据集中的额外列进行更新。这可以通过将第二个参数设置为 `true` 来实现。

```
<?php

use CodeIgniter\Database\RawSql;

$data = [
```

(续下页)

(接上页)

```
    [
        'id'      => 2,
        'email'   => 'ahmadinejad@example.com',
        'name'    => 'Ahmadinejad',
        'country' => 'Iran',
    ],
    [
        'id'      => null,
        'email'   => 'pedro@example.com',
        'name'    => 'Pedro',
        'country' => 'El Salvador',
    ],
];

$additionalUpdateField = ['updated_at' => new RawSql('CURRENT_
↪TIMESTAMP')];

$sql = $builder->setData($data)->updateFields(
↪$additionalUpdateField, true)->upsertBatch();
/* MySQLi produces:
    INSERT INTO `db_user` (`country`, `email`, `name`)
    VALUES ('Iran','ahmadinejad@example.com','Ahmadinejad'),('El␣
↪Salvador','pedro@example.com','Pedro')
    ON DUPLICATE KEY UPDATE
    `country` = VALUES(`country`),
    `email` = VALUES(`email`),
    `name` = VALUES(`name`),
    `updated_at` = CURRENT_TIMESTAMP
*/
```

注意 updated_at 字段未插入，但用于更新。

更新数据

Update

$builder->replace()

此方法执行 **REPLACE** 语句，本质上是 SQL 标准的（可选）**DELETE** + **INSERT**，使用 *PRIMARY* 和 *UNIQUE* 作为决定因素。在我们的案例中，它将使你无需通过组合不同的 `select()`、`update()`、`delete()` 和 `insert()` 调用来实现复杂逻辑。

示例：

```
<?php

$data = [
    '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')
```

在上述示例中，如果我们假设 `title` 字段是主键，则包含 `My title` 作为 `title` 值的行将被删除，并用我们的新行数据替换。

也允许使用 `set()` 方法，并且所有值都会自动转义，就像 `insert()` 一样。

$builder->set()

此方法允许你为插入或更新设置值。

它可以代替直接向 insert() 或 update() 方法传递数据数组：

```
<?php

$builder->set('name', $name);
$builder->insert();
// Produces: INSERT INTO mytable (`name`) VALUES ('{$name}')
```

如果多次调用该方法，它们将根据你执行的是插入还是更新正确组装：

```
<?php

$builder->set('name', $name);
$builder->set('title', $title);
$builder->set('status', $status);
$builder->insert();
```

set() 也接受可选的第三个参数（$escape），如果设置为 false，将防止值被转义。为了说明差异，以下是使用和不使用转义参数的 set()。

```
<?php

$builder->set('field', 'field+1', false);
$builder->where('id', 2);
$builder->update();
// gives UPDATE mytable SET field = field+1 WHERE `id` = 2

$builder->set('field', 'field+1');
$builder->where('id', 2);
$builder->update();
// gives UPDATE `mytable` SET `field` = 'field+1' WHERE `id` = 2
```

你也可以向此方法传递关联数组：

```
<?php

$array = [
    'name'   => $name,
    'title'  => $title,
    'status' => $status,
];

$builder->set($array);
$builder->insert();
```

或对象：

```php
<?php

namespace App\Libraries;

class MyClass
{
    public $title   = 'My Title';
    public $content = 'My Content';
    public $date    = 'My Date';
}
```

```php
<?php

use App\Libraries\MyClass;

$object = new MyClass();
$builder->set($object);
$builder->insert();
```

$builder->update()

根据你提供的数据生成 update 字符串并运行查询。你可以传递 **数组**或 **对象**。以下是使用数组的示例：

```php
<?php

$data = [
    'title' => $title,
    'name'  => $name,
    'date'  => $date,
];

$builder->where('id', $id);
$builder->update($data);
/*
 * Produces:
 * UPDATE mytable
```

(续下页)

(接上页)

```
 * SET title = '{$title}', name = '{$name}', date = '{$date}'
 * WHERE id = $id
 */
```

或者你可以提供对象:

```
<?php

namespace App\Libraries;

class MyClass
{
    public $title   = 'My Title';
    public $content = 'My Content';
    public $date    = 'My Date';
}
```

```
<?php

use App\Libraries\MyClass;

$object = new MyClass();
$builder->where('id', $id);
$builder->update($object);
/*
 * Produces:
 * UPDATE `mytable`
 * SET `title` = '{$title}', `content` = '{$content}', `date` = '{
↪$date}'
 * WHERE id = `$id`
 */
```

备注: 除 RawSql 外，所有值都会自动转义，生成更安全的查询。

警告: 使用 RawSql 时，你必须手动转义数据。否则可能导致 SQL 注入。

你会注意到使用 $builder->where() 方法，允许你设置 **WHERE** 子句。你可以直接将此信息作为字符串传递到 update() 方法中：

```
<?php

$builder->update($data, 'id = 4');
```

或作为数组：

```
<?php

$builder->update($data, ['id' => $id]);
```

执行更新时，也可以使用上述的 $builder->set() 方法。

$builder->getCompiledUpdate()

此方法与 $builder->getCompiledInsert() 的工作方式完全相同，区别在于生成的是 **UPDATE** SQL 字符串而非 **INSERT** SQL 字符串。

有关详细信息，请查看*$builder->getCompiledInsert()* 的文档。

备注: 此方法不适用于批量更新。

UpdateBatch

$builder->updateBatch()

备注: 自 v4.3.0 起，updateBatch() 的第二个参数 $index 已更改为 $constraints。现在接受数组、字符串或 RawSql 类型。

从数据更新

根据你提供的数据生成 update 字符串并运行查询。你可以向该方法传递 **数组**或 **对象**。以下是使用数组的示例：

```
<?php

$data = [
    [
        'title'  => 'Title 1',
        'author' => 'Author 1',
        'name'   => 'Name 1',
        'date'   => 'Date 1',
    ],
    [
        'title'  => 'Title 2',
        'author' => 'Author 2',
        'name'   => 'Name 2',
        'date'   => 'Date 2',
    ],
];
$builder->updateBatch($data, ['title', 'author']);
/*
 * Produces:
 * UPDATE `mytable`
 * INNER JOIN (
 * SELECT 'Title 1' `title`, 'Author 1' `author`, 'Name 1' `name`,
↪'Date 1' `date` UNION ALL
 * SELECT 'Title 2' `title`, 'Author 2' `author`, 'Name 2' `name`,
↪'Date 2' `date`
 * ) `u`
 * ON `mytable`.`title` = `u`.`title` AND `mytable`.`author` = `u`.
↪`author`
 * SET
 * `mytable`.`title` = `u`.`title`,
 * `mytable`.`name` = `u`.`name`,
 * `mytable`.`date` = `u`.`date`
 */
```

第一个参数是值的关联数组，第二个参数是 where 键。

备注: 自 v4.3.0 起，生成的 SQL 结构已改进。

自 v4.3.0 起，你也可以使用 `onConstraint()` 和 `updateFields()` 方法：

```
<?php

use CodeIgniter\Database\RawSql;

$builder->setData($data)->onConstraint('title, author')->
↪updateBatch();

// OR
$builder->setData($data, null, 'u')
    ->onConstraint(['`mytable`.`title`' => '`u`.`title`', 'author'␣
↪=> new RawSql('`u`.`author`')])
    ->updateBatch();

// OR
foreach ($data as $row) {
    $builder->setData($row);
}
$builder->onConstraint('title, author')->updateBatch();

// OR
$builder->setData($data, true, 'u')
    ->onConstraint(new RawSql('`mytable`.`title` = `u`.`title` AND␣
↪`mytable`.`author` = `u`.`author`'))
    ->updateFields(['last_update' => new RawSql('CURRENT_TIMESTAMP()
↪')], true)
    ->updateBatch();
/*
 * Produces:
 * UPDATE `mytable`
 * INNER JOIN (
 * SELECT 'Title 1' `title`, 'Author 1' `author`, 'Name 1' `name`,
↪'Date 1' `date` UNION ALL
```

(续下页)

(接上页)

```
 * SELECT 'Title 2' `title`, 'Author 2' `author`, 'Name 2' `name`,
↪'Date 2' `date`
 * ) `u`
 * ON `mytable`.`title` = `u`.`title` AND `mytable`.`author` = `u`.
↪`author`
 * SET
 * `mytable`.`title` = `u`.`title`,
 * `mytable`.`name` = `u`.`name`,
 * `mytable`.`date` = `u`.`date`,
 * `mytable`.`last_update` = CURRENT_TIMESTAMP() // this only␣
↪applies to the last scenario
 */
```

备注: 除 `RawSql` 外，所有值都会自动转义，生成更安全的查询。

警告: 使用 `RawSql` 时，你必须手动转义数据。否则可能导致 SQL 注入。

备注: 因为工作原理的原因，若使用此方法则 `affectedRows()` 无法提供正确的结果。相反，`updateBatch()` 返回受影响的行数。

从查询更新

自 v4.3.0 起，你也可以使用 `setQueryAsData()` 方法从查询更新：

```php
<?php

use CodeIgniter\Database\RawSql;

$query = $this->db->table('user2')
    ->select('user2.name, user2.email, user2.country')
    ->join('user', 'user.email = user2.email', 'inner')
    ->where('user2.country', 'US');
```

(续下页)

(接上页)

```
$additionalUpdateField = ['updated_at' => new RawSql('CURRENT_
↪TIMESTAMP')];

$sql = $builder->table('user')
    ->setQueryAsData($query, null, 'u')
    ->onConstraint('email')
    ->updateFields($additionalUpdateField, true)
    ->updateBatch();
/*
 * Produces:
 * UPDATE `user`
 * INNER JOIN (
 * SELECT user2.name, user2.email, user2.country
 * FROM user2
 * INNER JOIN user ON user.email = user2.email
 * WHERE user2.country = 'US'
 * ) `u`
 * ON `user`.`email` = `u`.`email`
 * SET
 * `mytable`.`name` = `u`.`name`,
 * `mytable`.`email` = `u`.`email`,
 * `mytable`.`country` = `u`.`country`,
 * `mytable`.`updated_at` = CURRENT_TIMESTAMP()
 */
```

备注： 必须将 select 查询的列别名与目标表的列匹配。

删除数据

Delete

$builder->delete()

生成 **DELETE** SQL 字符串并运行查询。

```
<?php

$builder->delete(['id' => $id]);
// Produces: DELETE FROM mytable WHERE id = $id
```

第一个参数是 where 子句。你也可以使用 `where()` 或 `orWhere()` 方法，而不是将数据传递到方法的第一个参数：

```
<?php

$builder->where('id', $id);
$builder->delete();
/*
 * Produces:
 * DELETE FROM mytable
 * WHERE id = $id
 */
```

如果要删除表中的所有数据，可以使用 `truncate()` 方法或 `emptyTable()`。

$builder->getCompiledDelete()

此方法与 `$builder->getCompiledInsert()` 的工作方式完全相同，区别在于生成的是 **DELETE** SQL 字符串而非 **INSERT** SQL 字符串。

有关详细信息，请查看*$builder->getCompiledInsert()* 的文档。

DeleteBatch

$builder->deleteBatch()

在 4.3.0 版本加入.

从数据删除

根据一组数据生成批量 **DELETE** 语句。

```
<?php

$data = [
    [
        'order'   => 48372,
        'line'    => 3,
        'product' => 'Keyboard',
        'qty'     => 1,
    ],
    [
        'order'   => 48372,
        'line'    => 4,
        'product' => 'Mouse',
        'qty'     => 1,
    ],
    [
        'order'   => 48372,
        'line'    => 5,
        'product' => 'Monitor',
        'qty'     => 2,
    ],
];

$builder->setData($data, true, 'del')
    ->onConstraint('order, line')
    ->where('del.qty >', 1)
    ->deleteBatch();

/*
 * MySQL Produces:
 * DELETE `order_line` FROM `order_line`
 * INNER JOIN (
 * SELECT 3 `line`, 48372 `order`, 'Keyboard' `product`, 1 `qty`␣
↪UNION ALL
 * SELECT 4 `line`, 48372 `order`, 'Mouse'    `product`, 1 `qty`␣
```

(续下页)

(接上页)

```
↪UNION ALL
* SELECT 5 `line`, 48372 `order`, 'Monitor'  `product`, 2 `qty`
* ) `del`
* ON `order_line`.`order` = `del`.`order` AND `order_line`.`line`␣
↪= `del`.`line`
* WHERE `del`.`qty` > 1
*/
```

此方法在删除具有复合主键的表中的数据时可能特别有用。

备注: SQLite3 不支持使用 `where()`。

从查询删除

你也可以从查询删除:

```
<?php

use CodeIgniter\Database\RawSql;

$query = $this->db->table('user2')->select('email, name, country')->
↪where('country', 'Greece');

$this->db->table('user')
    ->setQueryAsData($query, 'alias')
    ->onConstraint('email')
    ->where('alias.name = user.name')
    ->deleteBatch();

/* MySQLi produces:
    DELETE `user` FROM `user`
    INNER JOIN (
    SELECT `email`, `name`, `country`
    FROM `user2`
    WHERE `country` = 'Greece') `alias`
    ON `user`.`email` = `alias`.`email`
```

(续下页)

(接上页)

```
    WHERE `alias`.`name` = `user`.`name`
*/
```

$builder->emptyTable()

生成 **DELETE** SQL 字符串并运行查询:

```
<?php

$builder->emptyTable('mytable');
// Produces: DELETE FROM mytable
```

$builder->truncate()

生成 **TRUNCATE** SQL 字符串并运行查询。

```
<?php

$builder->truncate();
/*
 * Produce:
 * TRUNCATE mytable
 */
```

备注: 如果 TRUNCATE 命令不可用，`truncate()` 将执行 "DELETE FROM table"。

条件语句

When

$builder->when()

在 4.3.0 版本加入.

这允许根据条件修改查询，而不会中断查询构建器链。第一个参数是条件，应评估为布尔值。第二个参数是在条件为 true 时运行的回调函数。

例如，你可能希望仅根据 HTTP 请求中发送的值应用给定的 WHERE 语句：

```
<?php

$status = service('request')->getPost('status');

$users = $this->db->table('users')
    ->when($status, static function ($query, $status) {
        $query->where('status', $status);
    })
    ->get();
```

由于条件评估为 `true`，将调用回调函数。条件中设置的值将作为第二个参数传递给回调函数，以便在查询中使用。

有时，你可能希望在条件评估为 false 时应用不同的语句。这可以通过提供第二个闭包实现：

```
<?php

$onlyInactive = service('request')->getPost('return_inactive');

$users = $this->db->table('users')
    ->when($onlyInactive, static function ($query, $onlyInactive) {
        $query->where('status', 'inactive');
    }, static function ($query) {
        $query->where('status', 'active');
    })
    ->get();
```

WhenNot

$builder->whenNot()

在 4.3.0 版本加入.

此方法与 `$builder->when()` 的工作方式完全相同，区别在于仅在条件评估为 `false`

时运行回调函数，而 `when()` 在 `true` 时运行。

```
<?php

$status = service('request')->getPost('status');

$users = $this->db->table('users')
    ->whenNot($status, static function ($query, $status) {
        $query->where('active', 0);
    })
    ->get();
```

方法链

方法链允许你通过连接多个方法来简化语法。考虑以下示例：

```
<?php

$query = $builder->select('title')
    ->where('id', $id)
    ->limit(10, 20)
    ->get();
```

重置查询构建器

ResetQuery

$builder->resetQuery()

重置查询构建器允许你在不首先使用 `$builder->get()` 或 `$builder->insert()` 等方法执行查询的情况下重新开始查询。

这在以下情况下非常有用：你使用查询构建器生成 SQL（例如 `$builder->getCompiledSelect()`），但随后选择运行查询：

```
<?php

// Note that the parameter of the `getCompiledSelect()` method is␣
```

(续下页)

(接上页)

```
↪false
$sql = $builder->select(['field1', 'field2'])
    ->where('field3', 5)
    ->getCompiledSelect(false);

// ...
// Do something crazy with the SQL code... like add it to a cron␣
↪script for
// later execution or something...
// ...

$data = $builder->get()->getResultArray();
/*
 * Would execute and return an array of results of the following␣
↪query:
 * SELECT field1, field2 FROM mytable WHERE field3 = 5;
 */
```

类参考

class CodeIgniter\Database**BaseBuilder**

db()

返回

当前使用的数据库连接

返回类型

ConnectionInterface

从 $db 返回当前数据库连接。用于访问不直接对查询构建器可用的 ConnectionInterface 方法，如 insertID() 或 errors()。

resetQuery()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

重置当前查询构建器状态。在希望构建可在某些条件下取消的查询时非常有用。

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字符串，统计查询构建器查询返回的所有记录。

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字符串，统计特定表中的所有记录。

get(*[$limit = null[, $offset = null[, $reset = true]]]*)

参数

- **$limit** (int) –LIMIT 子句
- **$offset** (int) –OFFSET 子句
- **$reset** (bool) –是否清除查询构建器的值?

返回

\CodeIgniter\Database\ResultInterface 实例（方法链）

返回类型

\CodeIgniter\Database\ResultInterface

根据已调用的查询构建器方法编译并运行 SELECT 语句。

getWhere([*$where = null[, $limit = null[, $offset = null[, $reset = true]]]]*])

参数

- **$where** (`string`) –WHERE 子句
- **$limit** (`int`) –LIMIT 子句
- **$offset** (`int`) –OFFSET 子句
- **$reset** (`bool`) –是否清除查询构建器的值?

返回

`\CodeIgniter\Database\ResultInterface` 实例（方法链）

返回类型

`\CodeIgniter\Database\ResultInterface`

与 `get()` 相同，但允许直接添加 WHERE 子句。

select([*$select = '*'[, $escape = null]*])

参数

- **$select** (`array|RawSql|string`) –查询的 SELECT 部分
- **$escape** (`bool`) –是否转义值和标识符

返回

`BaseBuilder` 实例（方法链）

返回类型

`BaseBuilder`

向查询添加 `SELECT` 子句。

selectAvg([*$select = ''[, $alias = '']*])

参数

- **$select** (`string`) –计算平均值的字段
- **$alias** (`string`) –结果值名称的别名

返回

`BaseBuilder` 实例（方法链）

返回类型

`BaseBuilder`

向查询添加 `SELECT AVG(field)` 子句。

selectMax([*$select* = ''[, *$alias* = '']])

参数

- **$select** (string) –计算最大值的字段
- **$alias** (string) –结果值名称的别名

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 `SELECT MAX(field)` 子句。

selectMin([*$select* = ''[, *$alias* = '']])

参数

- **$select** (string) –计算最小值的字段
- **$alias** (string) –结果值名称的别名

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 `SELECT MIN(field)` 子句。

selectSum([*$select* = ''[, *$alias* = '']])

参数

- **$select** (string) –计算总和的字段
- **$alias** (string) –结果值名称的别名

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 `SELECT SUM(field)` 子句。

selectCount([*$select* = ''[, *$alias* = '']])

参数

- **$select** (string) –计算计数的字段
- **$alias** (string) –结果值名称的别名

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 SELECT COUNT(field) 子句。

selectSubquery(*BaseBuilder $subquery*, *string $as*)

参数

- **$subquery** (string) –BaseBuilder 实例
- **$as** (string) –结果值名称的别名

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向选择部分添加子查询。

distinct([*$val* = *true*])

参数

- **$val** (bool) –"distinct" 标志的期望值

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

设置一个标志，告诉查询构建器向查询的 SELECT 部分添加 DISTINCT 子句。

from(*$from*[, *$overwrite* = *false*])

参数

- **$from** (mixed) –表名；字符串或数组
- **$overwrite** (bool) –是否覆盖第一个现有表?

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

指定查询的 FROM 子句。

fromSubquery(*$from*, *$alias*)

参数

- **$from** (*BaseBuilder*) –BaseBuilder 类的实例
- **$alias** (string) –子查询别名

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

使用子查询指定查询的 FROM 子句。

setQueryAsData(*$query*[, *$alias*[, *$columns = null*]])

在 4.3.0 版本加入.

参数

- **$query** (BaseBuilder|RawSql) –BaseBuilder 或 RawSql 实例
- **$alias** (string|null) –查询的别名
- **$columns** (array|string|null) –查询中的列数组或逗号分隔的字符串

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

设置查询作为 insertBatch()、updateBatch()、upsertBatch() 的数据源。如果 $columns 为 null，将运行查询以生成列名。

join(*$table*, *$cond*[, *$type* = ''[, *$escape* = *null*]])

参数

- **$table** (string) –要连接的表名
- **$cond** (string|RawSql) –JOIN ON 条件
- **$type** (string) –JOIN 类型
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 JOIN 子句。自 v4.2.0 起，可以使用 RawSql 作为 JOIN ON 条件。另请参阅*$builder->join()*。

where(*$key*[, *$value* = *null*[, *$escape* = *null*]])

参数

- **$key** (array|RawSql|string) –要比较的字段名，或关联数组
- **$value** (mixed) –如果是单个键，则与此值比较
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成查询的 WHERE 部分。多个调用之间用 AND 分隔。

orWhere(*$key*[, *$value* = *null*[, *$escape* = *null*]])

参数

- **$key** (mixed) –要比较的字段名，或关联数组
- **$value** (mixed) –如果是单个键，则与此值比较

- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成查询的 WHERE 部分。多个调用之间用 OR 分隔。

orWhereIn([*$key* = *null*[, *$values* = *null*[, *$escape* = *null*]]])

参数

- **$key** (string) –要搜索的字段
- **$values** (array|BaseBulder|Closure) –目标值数组，或用于子查询的匿名函数
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成 WHERE field IN('item', 'item') SQL 查询，并在适当时用 OR 连接。

orWhereNotIn([*$key* = *null*[, *$values* = *null*[, *$escape* = *null*]]])

参数

- **$key** (string) –要搜索的字段
- **$values** (array|BaseBulder|Closure) –目标值数组，或用于子查询的匿名函数
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成 WHERE field NOT IN('item', 'item') SQL 查询，并在适当时用 OR 连接。

whereIn([*$key* = *null*[, *$values* = *null*[, *$escape* = *null*]]])

参数

- **$key** (string) –要检查的字段名
- **$values** (array|BaseBulder|Closure) –目标值数组,或用于子查询的匿名函数
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成 WHERE field IN('item', 'item') SQL 查询,并在适当时用 AND 连接。

whereNotIn([*$key* = *null*[, *$values* = *null*[, *$escape* = *null*]]])

参数

- **$key** (string) –要检查的字段名
- **$values** (array|BaseBulder|Closure) –目标值数组,或用于子查询的匿名函数
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成 WHERE field NOT IN('item', 'item') SQL 查询，并在适当时用 AND 连接。

groupStart()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

开始一个组表达式，使用 AND 连接内部条件。

orGroupStart()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

开始一个组表达式，使用 OR 连接内部条件。

notGroupStart()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

开始一个组表达式，使用 AND NOT 连接内部条件。

orNotGroupStart()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

开始一个组表达式，使用 OR NOT 连接内部条件。

groupEnd()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

结束一个组表达式。

like(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *null*[, *$insensitiveSearch* = *false*]]]])

参数

- **$field**(array|RawSql|string)–字段名

- **$match** (string) –要匹配的文本部分
- **$side** (string) –在表达式的哪一侧放置‘%’通配符
- **$escape** (bool) –是否转义值和标识符
- **$insensitiveSearch** (bool) –是否强制不区分大小写的搜索

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 LIKE 子句，多个调用之间用 AND 分隔。

orLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *null*[, *$insensitiveSearch* = *false*]]]])

参数

- **$field** (string) –字段名
- **$match** (string) –要匹配的文本部分
- **$side** (string) –在表达式的哪一侧放置‘%’通配符
- **$escape** (bool) –是否转义值和标识符
- **$insensitiveSearch** (bool) –是否强制不区分大小写的搜索

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 LIKE 子句，多个调用之间用 OR 分隔。

notLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *null*[, *$insensitiveSearch* = *false*]]]])

参数

- **$field** (string) –字段名
- **$match** (string) –要匹配的文本部分

- **$side** (string) –在表达式的哪一侧放置‘%’通配符
- **$escape** (bool) –是否转义值和标识符
- **$insensitiveSearch** (bool) –是否强制不区分大小写的搜索

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 NOT LIKE 子句，多个调用之间用 AND 分隔。

orNotLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *null*[, *$insensitiveSearch* = *false*]]]])

参数

- **$field** (string) –字段名
- **$match** (string) –要匹配的文本部分
- **$side** (string) –在表达式的哪一侧放置‘%’通配符
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 NOT LIKE 子句，多个调用之间用 OR 分隔。

having(*$key*[, *$value* = *null*[, *$escape* = *null*]])

参数

- **$key** (mixed) –标识符（字符串）或字段/值对的关联数组
- **$value** (string) –如果 $key 是标识符，则查找的值
- **$escape** (string) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 HAVING 子句，多个调用之间用 AND 分隔。

orHaving(*$key*[, *$value* = *null*[, *$escape* = *null*]])

参数

- **$key** (mixed) –标识符（字符串）或字段/值对的关联数组
- **$value** (string) –如果 $key 是标识符，则查找的值
- **$escape** (string) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 HAVING 子句，多个调用之间用 OR 分隔。

orHavingIn([*$key* = *null*[, *$values* = *null*[, *$escape* = *null*]]])

参数

- **$key** (string) –要搜索的字段
- **$values** (array|BaseBulder|Closure) –目标值数组，或用于子查询的匿名函数
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成 HAVING field IN('item', 'item') SQL 查询，并在适当时用 OR 连接。

orHavingNotIn([*$key* = *null*[, *$values* = *null*[, *$escape* = *null*]]])

参数

- **$key** (string) –要搜索的字段

- **$values** (array|BaseBulder|Closure) –目标值数组,或用于子查询的匿名函数
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成 HAVING field NOT IN('item', 'item') SQL 查询，并在适当时用 OR 连接。

havingIn([*$key* = *null*[, *$values* = *null*[, *$escape* = *null*]]])

参数

- **$key** (string) –要检查的字段名
- **$values** (array|BaseBulder|Closure) –目标值数组,或用于子查询的匿名函数
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成 HAVING field IN('item', 'item') SQL 查询，并在适当时用 AND 连接。

havingNotIn([*$key* = *null*[, *$values* = *null*[, *$escape* = *null*]]])

参数

- **$key** (string) –要检查的字段名
- **$values** (array|BaseBulder|Closure) –目标值数组,或用于子查询的匿名函数
- **$escape** (bool) –是否转义值和标识符
- **$insensitiveSearch** (bool) –是否强制不区分大小写的搜索

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

生成 HAVING field NOT IN('item', 'item') SQL 查询，并在适当时用 AND 连接。

havingLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *null*[, *$insensitiveSearch* = *false*]]]])

参数

- **$field** (string) –字段名
- **$match** (string) –要匹配的文本部分
- **$side** (string) –在表达式的哪一侧放置‘%’通配符
- **$escape** (bool) –是否转义值和标识符
- **$insensitiveSearch** (bool) –是否强制不区分大小写的搜索

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询的 HAVING 部分添加 LIKE 子句，多个调用之间用 AND 分隔。

orHavingLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *null*[, *$insensitiveSearch* = *false*]]]])

参数

- **$field** (string) –字段名
- **$match** (string) –要匹配的文本部分
- **$side** (string) –在表达式的哪一侧放置‘%’通配符
- **$escape** (bool) –是否转义值和标识符
- **$insensitiveSearch** (bool) –是否强制不区分大小写的搜索

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询的 HAVING 部分添加 LIKE 子句，多个调用之间用 OR 分隔。

notHavingLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *null*[, *$insensitiveSearch* = *false*]]]])

参数

- **$field** (string) –字段名
- **$match** (string) –要匹配的文本部分
- **$side** (string) –在表达式的哪一侧放置‘%’通配符
- **$escape** (bool) –是否转义值和标识符
- **$insensitiveSearch** (bool) –是否强制不区分大小写的搜索

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询的 HAVING 部分添加 NOT LIKE 子句，多个调用之间用 AND 分隔。

orNotHavingLike(*$field*[, *$match* = ''[, *$side* = 'both'[, *$escape* = *null*[, *$insensitiveSearch* = *false*]]]])

参数

- **$field** (string) –字段名
- **$match** (string) –要匹配的文本部分
- **$side** (string) –在表达式的哪一侧放置‘%’通配符
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询的 HAVING 部分添加 NOT LIKE 子句，多个调用之间用 OR 分隔。

havingGroupStart()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

开始 HAVING 子句的组表达式，使用 AND 连接内部条件。

orHavingGroupStart()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

开始 HAVING 子句的组表达式，使用 OR 连接内部条件。

notHavingGroupStart()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

开始 HAVING 子句的组表达式，使用 AND NOT 连接内部条件。

orNotHavingGroupStart()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

开始 HAVING 子句的组表达式，使用 OR NOT 连接内部条件。

havingGroupEnd()

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

结束 HAVING 子句的组表达式。

groupBy(*$by*[, *$escape = null*])

参数

- **$by** (mixed) –分组的字段；字符串或数组

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 GROUP BY 子句。

orderBy(*$orderby*[, *$direction = ''*[, *$escape = null*]])

参数

- **$orderby** (string) –排序字段
- **$direction** (string) –排序方向 - ASC、DESC 或 random
- **$escape** (bool) –是否转义值和标识符

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 ORDER BY 子句。

limit(*$value*[, *$offset = 0*])

参数

- **$value** (int) –限制结果的行数
- **$offset** (int) –跳过的行数

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 LIMIT 和 OFFSET 子句。

offset(*$offset*)

参数

- **$offset** (int) –跳过的行数

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

向查询添加 OFFSET 子句。

union(*$union*)

参数

- **$union** (BaseBulder|Closure) –联合查询

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

添加 UNION 子句。

unionAll(*$union*)

参数

- **$union** (BaseBulder|Closure) –联合查询

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

添加 UNION ALL 子句。

set(*$key*[, *$value* = ''[, *$escape* = *null*]])

参数

- **$key** (mixed) –字段名，或字段/值对的数组
- **$value** (mixed) –字段值，如果 $key 是单个字段
- **$escape** (bool) –是否转义值

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

添加稍后传递给 insert()、update() 或 replace() 的字段/值对。

insert([*$set = null*[, *$escape = null*]])

参数

- **$set** (array) –字段/值对的关联数组
- **$escape** (bool) –是否转义值

返回

成功时返回 true，失败时返回 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
- **$batch_size** (int) –单次插入的行数

返回

插入的行数，或在无数据执行插入操作时返回 false

返回类型

int|false

编译并执行批量 INSERT 语句。

备注: 当提供的行数超过 `$batch_size` 时，将执行多个 `INSERT` 查询，每个查询尝试插入最多 `$batch_size` 行。

---

setInsertBatch(*$key*[, *$value* = ''[, *$escape* = *null*]])

自 4.3.0 版本弃用: 请改用*CodeIgniter\Database\BaseBuilder::setData()*。

参数

- **$key** (`mixed`) –字段名或字段/值对的数组
- **$value** (`string`) –字段值，如果 $key 是单个字段
- **$escape** (`bool`) –是否转义值

返回

`BaseBuilder` 实例（方法链）

返回类型

`BaseBuilder`

添加稍后通过 `insertBatch()` 插入到表中的字段/值对。

重要: 此方法已弃用，将在未来版本中移除。

---

upsert([*$set* = *null*[, *$escape* = *null*]])

参数

- **$set** (`array`) –字段/值对的关联数组
- **$escape** (`bool`) –是否转义值

返回

成功时返回 `true`，失败时返回 `false`

返回类型

`bool`

编译并执行 `UPSERT` 语句。

upsertBatch([*$set* = *null*[, *$escape* = *null*[, *$batch_size* = *100*]]])

参数

- **$set** (array) –要更新插入的数据
- **$escape** (bool) –是否转义值
- **$batch_size** (int) –单次更新插入的行数

返回

更新插入的行数，或在失败时返回 false

返回类型

int|false

编译并执行批量 UPSERT 语句。

备注： MySQL 使用 ON DUPLICATE KEY UPDATE，每行的受影响行数为 1（如果作为新行插入）、2（如果更新现有行）和 0（如果现有行设置为当前值）。

备注： 当提供的行数超过 $batch_size 时，将执行多个 UPSERT 查询，每个查询尝试更新插入最多 $batch_size 行。

---

update([*$set = null*[, *$where = null*[, *$limit = null*]]])

参数

- **$set** (array) –字段/值对的关联数组
- **$where** (string) –WHERE 子句
- **$limit** (int) –LIMIT 子句

返回

成功时返回 true，失败时返回 false

返回类型

bool

编译并执行 UPDATE 语句。

updateBatch([*$set = null*[, *$constraints = null*[, *$batchSize = 100*]]])

参数

- **$set** (array|object|null) –字段名，或字段/值对的关联数组

- **$constraints** (array|RawSql|string|null) –用作更新键的字段或字段集
- **$batchSize** (int) –单次查询中分组条件的数量

返回

更新的行数，或在失败时返回 false

返回类型

int|false

备注: 自 v4.3.0 起，参数 $set 和 $constraints 的类型已更改。

编译并执行批量 UPDATE 语句。$constraints 参数接受逗号分隔的字段字符串、数组、关联数组或 RawSql。

备注: 当提供的字段/值对超过 $batchSize 时，将执行多个查询，每个查询处理最多 $batchSize 字段/值对。如果我们将 $batchSize 设置为 0，则所有字段/值对将在单个查询中执行。

---

updateFields(*$set*[, *$addToDefault = false*[, *$ignore = null*]])

在 4.3.0 版本加入.

参数

- **$set** (mixed) –行或行数组，行是数组或对象
- **$addToDefault** (bool) –添加数据集中不存在的额外列
- **$ignore** (bool) –要忽略的列数组

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

与 updateBatch() 和 upsertBatch() 方法一起使用。定义将更新的字段。

onConstraint(*$set*)

在 4.3.0 版本加入.

参数

- **$set** (mixed) –用作键或约束的字段或字段集

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

与 updateBatch() 和 upsertBatch() 方法一起使用。接受逗号分隔的字段字符串、数组、关联数组或 RawSql。

setData(*$set*[, *$escape* = *null*[, *$alias* = '']])

在 4.3.0 版本加入.

参数

- **$set** (mixed) –行或行数组，行是数组或对象
- **$escape** (bool) –是否转义值
- **$alias** (bool) –数据集的表别名

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

用于 *Batch() 方法设置插入、更新、更新插入的数据。

setUpdateBatch(*$key*[, *$value* = ''[, *$escape* = *null*]])

自 4.3.0 版本弃用: 请改用*CodeIgniter\Database\BaseBuilder::setData()*。

参数

- **$key** (mixed) –字段名或字段/值对的数组
- **$value** (string) –字段值，如果 $key 是单个字段
- **$escape** (bool) –是否转义值

返回

BaseBuilder 实例（方法链）

返回类型

BaseBuilder

添加稍后通过 updateBatch() 更新表中的字段/值对。

重要: 此方法已弃用，将在未来版本中移除。

---

replace([*$set = null*])

参数

- **$set** (array) –字段/值对的关联数组

返回

成功时返回 true，失败时返回 false

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) –是否重置查询的“write”子句

返回

BaseBuilder 实例（方法链）或失败时返回 false

返回类型

BaseBuilder|false

编译并执行 DELETE 查询。

deleteBatch([*$set = null*[, *$constraints = null*[, *$batchSize = 100*]]])

参数

- **$set** (array|object|null) –字段名，或字段/值对的关联数组

- **$constraints** (array|RawSql|string|null) –用作删除键的字段或字段集
- **$batchSize** (int) –单次查询中分组条件的数量

返回

删除的行数，或在失败时返回 false

返回类型

int|false

编译并执行批量 DELETE 查询。

increment(*$column*[, *$value = 1*])

参数

- **$column** (string) –要递增的列名
- **$value** (int) –递增的量

将字段的值递增指定量。如果字段不是数字字段（如 VARCHAR），可能会被替换为 $value。

decrement(*$column*[, *$value = 1*])

参数

- **$column** (string) –要递减的列名
- **$value** (int) –递减的量

将字段的值递减指定量。如果字段不是数字字段（如 VARCHAR），可能会被替换为 $value。

truncate()

返回

成功时返回 true，失败时返回 false，测试模式下返回字符串

返回类型

bool|string

在表上执行 TRUNCATE 语句。

备注： 如果使用的数据库平台不支持 TRUNCATE，将改用 DELETE 语句。

---

emptyTable()

返回

成功时返回 `true`，失败时返回 `false`

返回类型

`bool`

通过 `DELETE` 语句删除表中的所有记录。

getCompiledSelect([*$reset = true*])

参数

- **$reset** (`bool`) –是否重置当前 QB 值

返回

编译后的 SQL 语句字符串

返回类型

`string`

编译 `SELECT` 语句并以字符串形式返回。

getCompiledInsert([*$reset = true*])

参数

- **$reset** (`bool`) –是否重置当前 QB 值

返回

编译后的 SQL 语句字符串

返回类型

`string`

编译 `INSERT` 语句并以字符串形式返回。

getCompiledUpdate([*$reset = true*])

参数

- **$reset** (`bool`) –是否重置当前 QB 值

返回

编译后的 SQL 语句字符串

返回类型

`string`

编译 UPDATE 语句并以字符串形式返回。

getCompiledDelete([*$reset = true*])

参数

- **$reset** (bool) –是否重置当前 QB 值

返回

编译后的 SQL 语句字符串

返回类型

string

编译 DELETE 语句并以字符串形式返回。

6.1.8 事务

CodeIgniter 的数据库抽象层允许你在支持事务安全表类型的数据库中使用事务。在 MySQL 中，你需要使用 InnoDB 或 BDB 表类型，而不是更常见的 MyISAM。大多数其他数据库平台原生支持事务。

如果你不熟悉事务，我们建议你寻找适合你所用数据库的优质在线资源进行学习。以下内容假定你已经具备事务的基本理解。

- *CodeIgniter 的事务处理方式*
- 运行事务
- 严格模式
  - 重置事务状态
- 错误处理
- 抛出异常
- 禁用事务
- 测试模式
- 手动运行事务
- 嵌套事务

CodeIgniter 的事务处理方式

CodeIgniter 采用的事务处理方法与流行的数据库类库 ADODB 的处理过程非常相似。我们选择这种方法是因为它能极大简化事务的执行流程。在大多数情况下，你只需要两行代码即可完成操作。

传统的事务实现需要大量额外工作，因为需要跟踪所有查询并根据查询的成功与否决定提交或回滚。这在处理嵌套查询时尤为繁琐。相比之下，我们实现了一个智能事务系统，可以自动为你完成所有这些工作（你也可以选择手动管理事务，但实际上这样做没有任何优势）。

备注： 自 v4.3.0 起，在事务过程中，即使 `DBDebug` 设为 true，默认也不会抛出异常。

运行事务

要通过事务执行查询，需使用 `$this->db->transStart()` 和 `$this->db->transComplete()` 方法，如下所示：

```
<?php

$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 `transStart()`/`transComplete()` 方法之间运行任意数量的查询，系统将根据这些查询的整体成功或失败情况决定提交或回滚。

严格模式

默认情况下，CodeIgniter 在严格模式下运行所有事务。

当启用严格模式时，如果运行多个事务组，其中一个组失败会导致所有后续组被回滚。

如果禁用严格模式，每个组将被独立处理，意味着一个组的失败不会影响其他组。

可以通过以下方式禁用严格模式：

```
<?php

$this->db->transStrict(false);
```

重置事务状态

在 4.6.0 版本加入.

当启用严格模式时，如果某个事务失败，所有后续事务将被回滚。

若要在失败后重新启动事务，可以重置事务状态：

```
<?php

$this->db->resetTransStatus();
```

错误处理

备注： 自 v4.3.0 起，在事务过程中，即使 `DBDebug` 设为 true，默认也不会抛出异常。

你可以通过以下方式自行处理错误：

```
<?php

$this->db->transStart();
$this->db->query('AN SQL QUERY...');
$this->db->query('ANOTHER QUERY...');
$this->db->transComplete();

if ($this->db->transStatus() === false) {
    // generate an error... or use the log_message() function to␣
↪log your error
}
```

抛出异常

在 4.3.0 版本加入.

备注: 自 v4.3.0 起，在事务过程中，即使 `DBDebug` 设为 true，默认也不会抛出异常。

若要在查询出错时抛出异常，可以使用 `$this->db->transException(true)`：

```
<?php

// When DBDebug in the Database Config must be true.

use CodeIgniter\Database\Exceptions\DatabaseException;

try {
    $this->db->transException(true)->transStart();
    $this->db->query('AN SQL QUERY...');
    $this->db->query('ANOTHER QUERY...');
    $this->db->query('AND YET ANOTHER QUERY...');
    $this->db->transComplete();
} catch (DatabaseException $e) {
    // Automatically rolled back already.
}
```

如果发生查询错误，所有查询将被回滚，并抛出 `DatabaseException` 异常。

禁用事务

事务功能默认启用。可以通过 `$this->db->transOff()` 禁用事务：

```
<?php

$this->db->transOff();
$this->db->transStart();
$this->db->query('AN SQL QUERY...');
$this->db->transComplete();
```

禁用事务后，查询将自动提交，就像在没有使用事务的情况下运行查询一样。

测试模式

你可以选择将事务系统置于「测试模式」，此模式下即使查询有效也会被回滚。要启用测试模式，只需将 `$this->db->transStart()` 方法的第一个参数设为 true：

```
<?php

$this->db->transStart(true); // Query will be rolled back
$this->db->query('AN SQL QUERY...');
$this->db->transComplete();
```

手动运行事务

当在 **app/Config/Database.php** 文件中将 `DBDebug` 设为 false 时，可以通过以下方式手动运行事务：

```
<?php

$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();
}
```

备注: 手动运行事务时请确保使用 `$this->db->transBegin()`，**不要**使用 `$this->db->transStart()`。

嵌套事务

在 CodeIgniter 中，事务可以嵌套执行，只有最外层（顶级）的事务命令会被真正执行。你可以在事务块中包含任意数量的 `transStart()/transComplete()` 或 `transBegin()/transCommit()/transRollback()` 组合。CodeIgniter 会跟踪事务的「深度」，并仅在最外层（零深度）执行实际操作。

```
<?php

$this->db->transStart(); // actually starts a transaction
$this->db->query('SOME QUERY 1 ...');
$this->db->transStart(); // doesn't necessarily start another␣
↪transaction
$this->db->query('SOME QUERY 2 ...');
$this->db->transComplete(); // doesn't necessarily end the␣
↪transaction, but required to finish the inner transaction
$this->db->query('SOME QUERY 3 ...');
$this->db->transComplete(); // actually ends the transaction
```

备注: 如果事务结构非常复杂，你需要确保内部事务能够再次到达最外层，以便数据库完整执行这些操作，从而避免意外的提交/回滚。

6.1.9 获取元数据

- 表格元数据
  - 列出数据库中的表格
  - 确定表格是否存在
- 字段元数据
  - 列出表中的字段
  - 确定表中是否存在字段
  - 检索字段元数据
  - 列出表中的索引

表格元数据

这些函数让你获取表格信息。

列出数据库中的表格

$db->listTables()

返回一个数组, 其中包含当前连接数据库中的所有表格名称。例如:

```
<?php

$db = db_connect();

$tables = $db->listTables();

foreach ($tables as $table) {
    echo $table;
}
```

备注: 一些驱动程序有其他系统表被排除在此返回之外。

确定表格是否存在

$db->tableExists()

在对表运行操作之前, 知道特定表格是否存在有时很有帮助。返回布尔值 true/false。用法示例:

```
<?php

$db = db_connect();

if ($db->tableExists('table_name')) {
    // some code...
}
```

备注: 用你要查找的表格名称替换 *table_name*。

字段元数据

列出表中的字段

$db->getFieldNames()

返回包含字段名称的数组。可以通过两种方式调用此查询:

1. 你可以提供表格名称并从 $db 对象调用它:

```
<?php

$db = db_connect();

$fields = $db->getFieldNames('table_name');

foreach ($fields as $field) {
    echo $field;
}
```

2. 你可以通过从查询结果对象调用函数来收集与任何查询关联的字段名称:

```
<?php

$db = db_connect();

$query = $db->query('SELECT * FROM some_table');

foreach ($query->getFieldNames() as $field) {
    echo $field;
}
```

确定表中是否存在字段

$db->fieldExists()

在执行操作之前, 有时知道某个特定字段是否存在很有帮助。返回布尔值 true/false。用法示例:

```
<?php

$db = db_connect();

if ($db->fieldExists('field_name', 'table_name')) {
    // some code...
}
```

备注: 将 *field_name* 和 *table_name* 替换为你要查找的列名和表名。

检索字段元数据

$db->getFieldData()

返回包含字段信息的对象数组。

有时收集字段名称或其他元数据 (如列类型、最大长度等) 很有帮助。

备注: 并非所有数据库都提供元数据。

用法示例:

```
<?php

$db = db_connect();

$fields = $db->getFieldData('table_name');
```

(续下页)

(接上页)

```
foreach ($fields as $field) {
    echo $field->name;
    echo $field->type;
    echo $field->max_length;
    echo $field->primary_key;
}
```

如果你的数据库支持，下列数据可以通过此函数获取：

- name - 列名称
- type - 列的类型
- max_length - 列的最大长度
- nullable - 如果列允许为空，则为布尔值 true ，否则为布尔值 false
- default - 默认值
- primary_key - 如果列是主键，则为整数 1 （即使有多个主键，所有主键值都是整数 1），否则为整数 0 （此字段目前仅对 MySQLi 和 SQLite3 可用）

备注： 自 v4.4.0 起，SQLSRV 支持 nullable。

$query->getFieldData()

如果你已经运行了一个查询，可以使用结果对象而不是提供表名：

```
<?php

$db = db_connect();

$query  = $db->query('YOUR QUERY');
$fields = $query->getFieldData();
```

备注： 返回的数据与 $db->getFieldData() 返回的数据不同。如果你无法获取所需的数据，请使用 $db->getFieldData()。

列出表中的索引

$db->getIndexData()

返回包含索引信息的对象数组。

用法示例:

```
<?php

$db = db_connect();

$keys = $db->getIndexData('table_name');

foreach ($keys as $key) {
    echo $key->name;
    echo $key->type;
    echo $key->fields; // array of field names
}
```

关键字类型可能是你使用的数据库所独有的。例如,MySQL 将为与表关联的每个键返回 primary、fulltext、spatial、index 或 unique 中的一个。

SQLite3 返回一个名为 `PRIMARY` 的伪索引。但它是一个特殊的索引, 你不能在 SQL 命令中使用它。

$db->getForeignKeyData()

返回包含外键信息的对象数组。

用法示例:

```
<?php

$db = db_connect();

$keys = $db->getForeignKeyData('table_name');

foreach ($keys as $key => $object) {
    echo $key === $object->constraint_name;
```

(续下页)

(接上页)

```
    echo $object->constraint_name;
    echo $object->table_name;
    echo $object->column_name[0]; // array
    echo $object->foreign_table_name;
    echo $object->foreign_column_name[0]; // array
    echo $object->on_delete;
    echo $object->on_update;
    echo $object->match;
}
```

外键使用命名约定 `tableprefix_table_column1_column2_foreign`。Oracle 使用稍微不同的后缀 `_fk`。

6.1.10 自定义函数调用

- *$db->callFunction()*

$db->callFunction()

此函数使你可以以与平台无关的方式调用 PHP 数据库函数, 这些函数不是 CodeIgniter 原生支持的。例如, 假设你想调用 `mysql_get_client_info()` 函数,CodeIgniter 原生不支持此函数。你可以这样做:

```
<?php

$db->callFunction('get_client_info');
```

你必须在第一个参数中提供函数名称, 不带 `mysql_` 前缀。前缀会根据当前使用的数据库驱动自动添加。这允许你在不同的数据库平台上运行相同的函数。显然, 并非所有函数调用在所有平台上都是相同的, 所以就可移植性而言, 此函数的用途有限。

被调用函数需要的任何参数都将添加到第二个参数中。

```
<?php

$db->callFunction('some_function', $param1, $param2 /* , ... */);
```

你通常需要提供数据库连接 ID 或数据库结果 ID。可以使用以下方式访问连接 ID:

```
<?php

$db->connID;
```

可以从结果对象内部访问结果 ID, 如下所示:

```
<?php

$query = $db->query('SOME QUERY');

$query->resultID;
```

6.1.11 数据库事件

数据库类包含几个可以利用的事件, 以了解数据库执行期间发生的更多信息。这些事件可用于收集数据以进行分析和报告。调试工具栏 使用此操作来收集在工具栏中显示的查询。

- 事件
  - *DBQuery*

事件

DBQuery

此事件在每次运行新查询时触发，无论成功与否。唯一的参数是当前查询的*Query* 实例。

你可以使用此事件将所有查询显示在 STDOUT，或记录到文件，甚至创建工具进行自动查询分析，以帮助你发现潜在的缺失索引、慢查询等。

记录所有查询的示例：

```
<?php

// In app/Config/Events.php
```

(续下页)

(接上页)

```
namespace Config;

use CodeIgniter\Events\Events;
use CodeIgniter\Exceptions\FrameworkException;
use CodeIgniter\HotReloader\HotReloader;

// ...

Events::on(
    'DBQuery',
    static function (\CodeIgniter\Database\Query $query) {
        log_message('info', (string) $query);
    },
);
```

6.1.12 数据库实用工具

数据库实用工具类包含了帮助你管理数据库的方法。

- 初始化实用工具类
- 使用数据库实用工具
  - 检索数据库名称列表
  - 判断数据库是否存在
  - 优化数据表
  - 优化数据库
  - 将查询结果导出为 *CSV* 文件
  - 将查询结果导出为 *XML* 文档

初始化实用工具类

按照以下方式加载实用工具类：

```
$dbutil = \Config\Database::utils();
```

你也可以将另一数据库组传递给 DB Utility 加载器，以防你要管理的数据库不是默认的：

```
$dbutil = \Config\Database::utils('group_name');
```

在上述示例中，我们将数据库组名称作为第一参数传递了进去。

使用数据库实用工具

检索数据库名称列表

返回一个包含数据库名称的数组：

```
$dbutil = \Config\Database::utils();

$dbs = $dbutil->listDatabases();

foreach ($dbs as $db) {
    echo $db;
}
```

判断数据库是否存在

有时候，我们需要知道特定的数据库是否存在。返回一个布尔值 `true/false`。使用示例：

```
$dbutil = \Config\Database::utils();

if ($dbutil->databaseExists('database_name')) {
    // some code...
}
```

备注: 将 database_name 替换为你正在查找的数据库名称。此方法区分大小写。

优化数据表

允许你使用第一参数中特定的表名来优化一个表。根据成功或失败返回 true/false:

```
$dbutil = \Config\Database::utils();

if ($dbutil->optimizeTable('table_name')) {
    echo 'Success!';
}
```

备注: 并非所有的数据库平台都支持表优化。它主要用于 MySQL。

优化数据库

允许你优化 DB 类当前连接的数据库。成功返回包含数据库状态消息的数组，失败返回 false:

```
$dbutil = \Config\Database::utils();

$result = $dbutil->optimizeDatabase();

if ($result !== false) {
    print_r($result);
}
```

备注: 并非所有的数据库平台都支持数据库优化。它主要用于 MySQL。

将查询结果导出为 CSV 文件

允许你生成一个来自查询结果的 CSV 文件。方法的第一参数必须包含你的查询结果对象。示例：

```
$db     = db_connect();
$dbutil = \Config\Database::utils();

$query = $db->query('SELECT * FROM mytable');

echo $dbutil->getCSVFromResult($query);
```

第二、第三和第四参数分别允许你设置分隔符、换行和封闭字符。默认的分隔符是逗号，`"\n"` 用作新行，双引号用作封闭符。示例：

```
$db     = db_connect();
$dbutil = \Config\Database::utils();

$query = $db->query('SELECT * FROM mytable');

$delimiter = ',';
$newline   = "\r\n";
$enclosure = '"';

echo $dbutil->getCSVFromResult($query, $delimiter, $newline,
↪$enclosure);
```

重要： 这个方法不会为你写入 CSV 文件。它仅创建 CSV 布局。如果你需要写入文件，使用 *write_file()* 辅助函数。

将查询结果导出为 XML 文档

通过此方法，你可以生成一个来自查询结果的 XML 文件。第一参数需要是一个查询结果对象，第二参数可能包含一个可选的配置参数数组。示例：

```
<?php

$db     = db_connect();
$dbutil = \Config\Database::utils();

$query = $db->query('SELECT * FROM mytable');

$config = [
    'root'    => 'root',
    'element' => 'element',
    'newline' => "\n",
    'tab'     => "\t",
];

echo $dbutil->getXMLFromResult($query, $config);
```

当 `mytable` 有 `id` 和 `name` 列时，将得到以下 xml 结果:

```
<root>
    <element>
        <id>1</id>
        <name>bar</name>
    </element>
</root>
```

重要: 这个方法不会为你写入 XML 文件。它仅仅创建 XML 布局。如果你需要写入文件，使用*write_file()* 辅助函数。

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 使用 CodeIgniter 的模型

- 模型
- 访问模型
- *CodeIgniter* 的模型
- 创建模型
  - *initialize()*
  - 连接数据库
  - 配置模型
    - *$table*
    - *$primaryKey*
    - *$useAutoIncrement*
    - *$returnType*
    - *$useSoftDeletes*
    - *$allowedFields*
    - *$allowEmptyInserts*
    - *$updateOnlyChanged*
    - *$casts*
    - 日期配置
    - 验证
    - 回调
- 模型字段类型转换
  - 定义数据类型

- 数据类型
  - *csv*
  - *datetime*
  - *timestamp*
- 自定义类型转换
  - 创建自定义处理器
  - 注册自定义处理器
  - 参数

- 处理数据
  - 查找数据
    - *find()*
    - *findColumn()*
    - *findAll()*
    - *first()*
    - *withDeleted()*
    - *onlyDeleted()*
  - 保存数据
    - *insert()*
    - *allowEmptyInserts()*
    - *update()*
    - *save()*
    - 保存日期
  - 删除数据
    - *delete()*
    - *purgeDeleted()*
  - 模型内验证

    * 验证数据
    * 设置验证规则
    * 获取验证结果
    * 获取验证错误
    * 检索验证规则
    * 验证占位符
  - 保护字段
  - 运行时返回类型变更
    * *asArray()*
    * *asObject()*
  - 处理大量数据
- 使用查询构建器
  - 获取模型的查询构建器
  - 获取其他表的查询构建器
  - 混合使用查询构建器和模型方法
- 模型事件
  - 定义回调
  - 指定运行的回调
  - 事件参数
  - 修改 *Find** 数据
- 手动创建模型

模型

CodeIgniter 的模型提供了便捷功能和额外特性，使得在数据库中操作 **单张表**更加方便。

它内置了常用数据库表交互的辅助函数，包括查找记录、更新记录、删除记录等标准操作。

访问模型

模型通常存储在 **app/Models** 目录中，其命名空间应与目录位置匹配，例如 `namespace App\Models`。

可以通过创建新实例或使用*model()* 辅助函数在类中访问模型：

```
<?php

// Create a new class manually.
$userModel = new \App\Models\UserModel();

// Create a shared instance of the model.
$userModel = model('UserModel');
// or
$userModel = model('App\Models\UserModel');
// or
$userModel = model(\App\Models\UserModel::class);

// Create a new class with the model() function.
$userModel = model('UserModel', false);

// Create shared instance with a supplied database connection.
$db        = db_connect('custom');
$userModel = model('UserModel', true, $db);
```

`model()` 内部使用 `Factories::models()`。有关第一个参数的详细信息，请参阅加载类。

CodeIgniter 的模型

CodeIgniter 提供的模型类包含以下特性：

- 自动数据库连接
- 基础 CRUD 方法
- 模型内验证
- 自动分页
- 及其他功能

该类为构建自定义模型提供了坚实基础，可快速构建应用程序的模型层。

创建模型

要使用 CodeIgniter 的模型，只需新建继承自 `CodeIgniter\Model` 的模型类：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    // ...
}
```

这个空类提供了对数据库连接、查询构建器和多个便捷方法的访问。

initialize()

如需在模型中进行额外设置，可扩展 `initialize()` 方法。该方法会在模型构造函数之后立即执行，避免重复构造函数参数，例如扩展其他模型：

```
<?php

namespace App\Models;
```

(续下页)

(接上页)

```
use Modules\Authentication\Models\UserAuthModel;

class UserModel extends UserAuthModel
{
    // ...

    /**
     * Called during initialization. Appends
     * our custom field to the module's model.
     */
    protected function initialize()
    {
        $this->allowedFields[] = 'middlename';
    }
}
```

连接数据库

当类首次实例化时，如果没有数据库连接实例传递给构造函数，且未在模型类中设置 `$DBGroup` 属性，模型会自动连接到数据库配置中设置的默认组。

通过添加 `$DBGroup` 属性可修改每个模型使用的数据库组，确保模型中所有 `$this->db` 引用都通过正确的连接进行：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    protected $DBGroup = 'group_name';

    // ...
}
```

将“group_name”替换为数据库配置文件中定义的数据库组名称。

配置模型

模型类提供以下配置选项，使类方法能无缝工作：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    protected $table      = 'users';
    protected $primaryKey = 'id';

    protected $useAutoIncrement = true;

    protected $returnType     = 'array';
    protected $useSoftDeletes = true;

    protected $allowedFields = ['name', 'email'];

    protected bool $allowEmptyInserts = false;
    protected bool $updateOnlyChanged = true;

    // Dates
    protected $useTimestamps = false;
    protected $dateFormat    = 'datetime';
    protected $createdField  = 'created_at';
    protected $updatedField  = 'updated_at';
    protected $deletedField  = 'deleted_at';

    // Validation
    protected $validationRules      = [];
    protected $validationMessages   = [];
    protected $skipValidation       = false;
```

(续下页)

(接上页)

```
    protected $cleanValidationRules = true;

    // Callbacks
    protected $allowCallbacks = true;
    protected $beforeInsert   = [];
    protected $afterInsert    = [];
    protected $beforeUpdate   = [];
    protected $afterUpdate    = [];
    protected $beforeFind     = [];
    protected $afterFind      = [];
    protected $beforeDelete   = [];
    protected $afterDelete    = [];
}
```

$table

指定模型主要操作的数据表。仅适用于内置 CRUD 方法，自定义查询不受限制。

$primaryKey

指定表中唯一标识记录的列名。不必与数据库主键完全匹配，但需与 `find()` 等方法使用的匹配列一致。

备注： 所有模型必须指定 primaryKey 以确保功能正常。

$useAutoIncrement

指定表是否对*$primaryKey* 使用自增特性。设为 `false` 时需手动提供主键值，适用于 1:1 关系或 UUID 场景。默认值为 `true`。

备注： 当*$useAutoIncrement* 设为 `false` 时，请确保数据库主键设为 `unique`，以保证模型功能正常。

$returnType

模型的 **find*()** 方法将自动返回结果数据，而非 Result 对象。

此设置允许你定义返回的数据类型，有效值为‘**array**’（默认）、’**object**’或可与 Result 对象的 `getCustomResultObject()` 方法配合使用的 **类的完全限定名称**。

使用类的特殊 `::class` 常量，可使大多数 IDE 实现自动补全，并支持重构等功能以更好地理解你的代码。

$useSoftDeletes

若设为 `true`，任何 `delete()` 方法调用都会在数据库中设置 `deleted_at` 字段值，而非实际删除行。这可以保留可能被其他位置引用的数据，维护可恢复对象的「回收站」，或简单地将其作为安全审计轨迹的一部分。当启用时，**find*()** 方法默认仅返回未删除行，除非在调用 **find*()** 方法前先调用 `withDeleted()` 方法。

根据模型的*$dateFormat* 设置，数据库需要包含 DATETIME 或 INTEGER 类型的字段。默认字段名为 `deleted_at`，但可通过*$deletedField* 属性配置为任意名称。

重要： 数据库中的 `deleted_at` 字段必须可为空。

$allowedFields

定义可通过 `save()`、`insert()` 或 `update()` 方法设置的字段名列表，防止大规模赋值漏洞。

备注： *$primaryKey* 字段不应包含在允许字段中。

$allowEmptyInserts

在 4.3.0 版本加入.

是否允许插入空数据。默认值是 `false`，这意味着如果你尝试插入空数据，将会抛出带有 "There is no data to insert." 信息的 `DataException`。

你也可以通过*allowEmptyInserts()* 方法来改变这个设置。

$updateOnlyChanged

在 4.5.0 版本加入.

是否仅更新*Entity* 的已更改字段。默认值是 `true`，这意味着在更新到数据库时仅使用已更改的字段数据。因此，如果你尝试更新一个没有更改的 Entity，将会抛出带有 "There is no data to update." 信息的 `DataException`。

将此属性设置为 `false` 将确保 Entity 的所有允许字段在任何时候都提交到数据库并进行更新。

$casts

在 4.5.0 版本加入.

该功能允许你将从数据库检索的数据转换为适当的 PHP 类型。此选项应为数组格式，其中键名对应字段名称，键值对应数据类型。详细信息请参阅模型字段类型转换。

日期配置

$useTimestamps

这个布尔值决定了是否自动将当前日期添加到所有插入和更新操作中。如果为 `true`，将按照*$dateFormat* 指定的格式设置当前时间。这要求表中存在适当数据类型的 **created_at**、**updated_at** 和 **deleted_at** 列。另请参阅*$createdField*、*$updatedField* 和*$deletedField*。

$dateFormat

该值配合*$useTimestamps* 和*$useSoftDeletes* 使用，确保正确类型的日期值被插入数据库。默认情况下会生成 DATETIME 值，但有效选项包括：`'datetime'`、`'date'` 或 `'int'` （UNIX 时间戳）。如果与无效或缺失的*$dateFormat* 一起使用*$useSoftDeletes* 或*$useTimestamps* 将会引发异常。

$createdField

指定用于记录数据创建时间戳的数据库字段。设置为空字符串 (`''`) 可避免更新该字段（即使启用了*$useTimestamps*）。

$updatedField

指定用于记录数据更新时间戳的数据库字段。设置为空字符串 (`''`) 可避免更新该字段（即使启用了*$useTimestamps*）。

$deletedField

指定用于软删除操作的数据库字段。详见*$useSoftDeletes*。

验证

$validationRules

包含一个验证规则数组（如如何保存规则 所述）或一个验证组名称的字符串（如相同章节所述）。另请参阅设置验证规则。

$validationMessages

包含一个自定义错误消息数组，用于验证过程中（如设置自定义错误信息 所述）。另请参阅设置验证规则。

$skipValidation

是否在所有 **插入**和 **更新**操作中跳过验证。默认值为 `false`，表示始终尝试验证数据。这主要由 `skipValidation()` 方法使用，但可更改为 `true` 以使模型永不验证。

$cleanValidationRules

是否移除传入数据中不存在的验证规则。这用于 **更新**操作。默认值为 `true`，表示在验证前会（临时）移除传入数据中不存在字段的验证规则，以避免在仅更新部分字段时出现验证错误。

也可以通过 `cleanRules()` 方法更改此值。

备注： 在 v4.2.7 之前，由于存在 bug，`$cleanValidationRules` 无法正常工作。

回调

$allowCallbacks

是否使用下面定义的回调。详见模型事件。

$beforeInsert

$afterInsert

$beforeUpdate

$afterUpdate

$beforeFind

$afterFind

$beforeDelete

$afterDelete

$beforeInsertBatch

$afterInsertBatch

$beforeUpdateBatch

$afterUpdateBatch

这些数组允许你指定在属性名指定时间点运行的回调方法。详见模型事件。

模型字段类型转换

在 4.5.0 版本加入.

从数据库检索数据时，整数类型的数据可能在 PHP 中被转换为字符串类型。你可能希望将日期/时间数据转换为 PHP 的 Time 对象。

模型字段类型转换允许你将从数据库检索的数据转换为适当的 PHP 类型。

重要: 如果将此功能与实体 一起使用，请勿同时使用实体属性类型转换。同时使用两种类型转换将无法正常工作。

实体属性类型转换作用于 (1)(4)，而此类型转换作用于 (2)(3):

```
[应用代码] --- (1) --> [实体] --- (2) --> [数据库]
[应用代码] <-- (4) --- [实体] <-- (3) --- [数据库]
```

使用此类型转换时，实体将在属性中持有正确类型的 PHP 值。此行为与之前的行为完全不同。不要期望属性持有数据库的原始数据。

定义数据类型

$casts 属性设置其定义。此选项应为数组，其中键是字段名称，值是数据类型：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    // ...
    protected array $casts = [
        'id'        => 'int',
        'birthdate' => '?datetime',
        'hobbies'   => 'json-array',
        'active'    => 'int-bool',
    ];
    // ...
}
```

数据类型

默认提供以下类型。在类型前添加问号可将字段标记为可空，例如 ?int、?datetime。

类型	PHP 类型	数据库字段类型
int	int	int 类型
float	float	float（数值）类型
bool	bool	bool/int/string 类型
int-bool	bool	int 类型（1 或 0）
array	array	string 类型（序列化）
csv	array	string 类型（CSV）
json	stdClass	json/string 类型
json-array	array	json/string 类型
datetime	Time	datetime 类型
timestamp	Time	int 类型（UNIX 时间戳）
uri	URI	string 类型

csv

使用 csv 类型转换时，使用 PHP 内置的 implode() 和 explode() 函数，并假定所有值都是字符串安全且不含逗号。对于更复杂的数据转换，请尝试 array 或 json。

datetime

你可以传递类似 datetime[ms] 的参数表示带毫秒的日期/时间，或 datetime[us] 表示带微秒的日期/时间。

日期时间格式在数据库配置 的 dateFormat 数组中设置，位于 **app/Config/Database.php** 文件。

备注： 当使用 ms 或 us 作为参数时，**模型**会处理 Time 的秒的小数部分。但 **查询构建器**不会。因此在将 Time 传递给查询构建器的方法（如 where()）时，仍需使用 format() 方法：

```
$model = model('SomeModel');

$now = \CodeIgniter\I18n\Time::now();

// The following code passes the microseconds to Query Builder.
```

(续下页)

(接上页)

```
$model->where('my_dt_field', $now->format('Y-m-d H:i:s.u'))->
↪findAll();
// Generates: SELECT * FROM `my_table` WHERE `my_dt_field` = '2024-
↪07-28 18:57:58.900326'

// But the following code loses the microseconds.
$model->where('my_dt_field', $now)->findAll();
// Generates: SELECT * FROM `my_table` WHERE `my_dt_field` = '2024-
↪07-28 18:57:58'
```

备注: 在 v4.6.0 之前，由于存在 bug，无法使用 `ms` 或 `us` 作为参数，因为 Time 的秒的小数部分会丢失。

timestamp

创建的 `Time` 实例的时区将是默认时区（应用的时区），而非 UTC。

自定义类型转换

你可以定义自己的转换类型。

创建自定义处理器

首先需要为你的类型创建一个处理器类。假设类位于 **app/Models/Cast** 目录：

```
<?php

namespace App\Models\Cast;

use CodeIgniter\DataCaster\Cast\BaseCast;
use InvalidArgumentException;

// The class must inherit the CodeIgniter\DataCaster\Cast\BaseCast␣
```

(续下页)

(接上页)

```
↪class
class CastBase64 extends BaseCast
{
    public static function get(
        mixed $value,
        array $params = [],
        ?object $helper = null,
    ): string {
        if (! is_string($value)) {
            self::invalidTypeValueError($value);
        }

        $decoded = base64_decode($value, true);

        if ($decoded === false) {
            throw new InvalidArgumentException('Cannot decode: ' .
↪$value);
        }

        return $decoded;
    }

    public static function set(
        mixed $value,
        array $params = [],
        ?object $helper = null,
    ): string {
        if (! is_string($value)) {
            self::invalidTypeValueError($value);
        }

        return base64_encode($value);
    }
}
```

如果不需要在获取或设置值时更改值，只需不实现相应方法：

```
<?php

namespace App\Models\Cast;

use CodeIgniter\DataCaster\Cast\BaseCast;
use InvalidArgumentException;

class CastBase64 extends BaseCast
{
    public static function get(
        mixed $value,
        array $params = [],
        ?object $helper = null,
    ): string {
        if (! is_string($value)) {
            self::invalidTypeValueError($value);
        }

        $decoded = base64_decode($value, true);

        if ($decoded === false) {
            throw new InvalidArgumentException('Cannot decode: ' .
↪$value);
        }

        return $decoded;
    }
}
```

注册自定义处理器

现在需要注册它:

```
<?php

namespace App\Models;
```

(续下页)

(接上页)

```
use App\Models\Cast\CastBase64;
use CodeIgniter\Model;

class MyModel extends Model
{
    // ...

    // Specify the type for the field
    protected array $casts = [
        'column1' => 'base64',
    ];

    // Bind the type to the handler
    protected array $castHandlers = [
        'base64' => CastBase64::class,
    ];

    // ...
}
```

参数

在某些情况下，单一类型可能不够。此时可以使用附加参数。附加参数用方括号表示，并用逗号分隔，例如 `type[param1, param2]`。

```
<?php

namespace App\Models;

use App\Models\Cast\SomeHandler;
use CodeIgniter\Model;

class MyModel extends Model
{
    // ...

    // Define a type with parameters
```

(续下页)

(接上页)

```
    protected array $casts = [
        'column1' => 'class[App\SomeClass, param2, param3]',
    ];

    // Bind the type to the handler
    protected array $castHandlers = [
        'class' => SomeHandler::class,
    ];

    // ...
}
```

```
<?php

namespace App\Models\Cast;

use CodeIgniter\DataCaster\Cast\BaseCast;

class SomeHandler extends BaseCast
{
    public static function get(
        mixed $value,
        array $params = [],
        ?object $helper = null,
    ): mixed {
        var_dump($params);
        /*
         * Output:
         * array(3) {
         *   [0]=>
         *   string(13) "App\SomeClass"
         *   [1]=>
         *   string(6) "param2"
         *   [2]=>
         *   string(6) "param3"
         * }
         */
```

(续下页)

(接上页)

```
    }
}
```

备注: 如果类型标记为可空（如 `?bool`）且传递的值不为 null，则会将带有 `nullable` 值的参数传递给类型转换处理器。如果类型转换已有预定义参数，则 `nullable` 将添加到列表末尾。

处理数据

查找数据

提供了多个函数用于对表执行基本的 CRUD 操作，包括 `find()`、`insert()`、`update()`、`delete()` 等。

find()

返回主键与第一个参数匹配的单行数据：

```
<?php

$user = $userModel->find($userId);
```

返回值格式由*$returnType* 指定。

通过传递主键值数组（而非单个值）可返回多行数据：

```
<?php

$users = $userModel->find([1, 2, 3]);
```

备注: 如果不传递参数，`find()` 将返回模型表中的所有行，实际上等同于 `findAll()`，但不够明确。

findColumn()

返回 null 或列值的索引数组：

```
<?php

$user = $userModel->findColumn($columnName);
```

`$column_name` 应为单个字段名，否则将抛出 `DataException`。

findAll()

返回所有结果：

```
<?php

$users = $userModel->findAll();
```

可在调用此方法前插入查询构建器命令来修改查询：

```
<?php

$users = $userModel->where('active', 1)->findAll();
```

可分别传递限制和偏移值作为第一和第二个参数：

```
<?php

$users = $userModel->findAll($limit, $offset);
```

first()

返回结果集中的第一行。最好与查询构建器结合使用。

```
<?php

$user = $userModel->where('deleted', 0)->first();
```

withDeleted()

如果*$useSoftDeletes* 为 true，则 **find*()** 方法不会返回 `deleted_at IS NOT NULL` 的行。要临时覆盖此行为，可在调用 **find*()** 方法前使用 `withDeleted()` 方法。

```
<?php

// Only gets non-deleted rows (deleted = 0)
$activeUsers = $userModel->findAll();

// Gets all rows
$allUsers = $userModel->withDeleted()->findAll();
```

onlyDeleted()

`withDeleted()` 会返回已删除和未删除的行，而此方法会修改后续的 **find*()** 方法仅返回软删除的行：

```
<?php

$deletedUsers = $userModel->onlyDeleted()->findAll();
```

保存数据

insert()

第一个参数是关联数组，用于在数据库中创建新行数据。如果传递对象而非数组，将尝试将其转换为数组。

数组的键必须与*$table* 中的列名匹配，数组的值是要保存的值。

可选的第二个参数为布尔类型，若设为 false，方法将返回布尔值表示查询成功与否。

可使用 `getInsertID()` 方法获取最后插入行的主键。

```
<?php

$data = [
```

(续下页)

(接上页)

```
    'username' => 'darth',
    'email'    => 'd.vader@theempire.com',
];

// Inserts data and returns inserted row's primary key
$userModel->insert($data);

// Inserts data and returns true on success and false on failure
$userModel->insert($data, false);

// Returns inserted row's primary key
$userModel->getInsertID();
```

allowEmptyInserts()

在 4.3.0 版本加入.

可使用 `allowEmptyInserts()` 方法插入空数据。默认情况下，模型在尝试插入空数据时会抛出异常。但调用此方法后，将不再执行检查。

```
<?php

$userModel->allowEmptyInserts()->insert([]);
```

也可通过*$allowEmptyInserts* 属性更改此设置。

通过调用 `allowEmptyInserts(false)` 可重新启用检查。

update()

更新数据库中的现有记录。第一个参数是要更新记录的*$primaryKey*。第二个参数是包含数据的关联数组。数组的键必须与*$table* 中的列名匹配，数组的值则是要保存的对应值：

```
<?php

$data = [
    'username' => 'darth',
```

(续下页)

(接上页)

```
    'email'     => 'd.vader@theempire.com',
];

$userModel->update($id, $data);
```

重要: 自 v4.3.0 起，如果生成的 SQL 语句没有 WHERE 子句，此方法会抛出 `DatabaseException`。在早期版本中，如果调用时未指定*$primaryKey* 且生成的 SQL 语句没有 WHERE 子句，查询仍会执行并更新表中的所有记录。

通过将主键数组作为第一个参数传递，可以在一次调用中更新多条记录：

```
<?php

$data = [
    'active' => 1,
];

$userModel->update([1, 2, 3], $data);
```

当需要更灵活的解决方案时，可以留空参数，此时其功能类似于查询构建器的 update 命令，并额外具备验证、事件等优势：

```
<?php

$userModel
    ->whereIn('id', [1, 2, 3])
    ->set(['active' => 1])
    ->update();
```

save()

这是对 `insert()` 和 `update()` 方法的封装，根据是否找到匹配 **主键**值的数组键来自动处理记录的插入或更新：

```
<?php

// 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);
```

save 方法还能通过识别非简单对象并将其公共和受保护值提取到数组，简化与自定义类结果对象的交互。这使得你可以非常简洁地使用实体类。实体类是表示单个对象类型实例的简单类（如用户、博客文章、任务等），负责维护围绕对象本身的业务逻辑（如特定格式的元素处理等），不应了解如何保存到数据库。最简单的实体类可能如下所示：

```
<?php

namespace App\Entities;

class Job
{
    protected $id;
    protected $name;
    protected $description;

    public function __get($key)
    {
```

(续下页)

(接上页)

```
        if (property_exists($this, $key)) {
            return $this->{$key};
        }
    }

    public function __set($key, $value)
    {
        if (property_exists($this, $key)) {
            $this->{$key} = $value;
        }
    }
}
```

与之配合的简单模型可能如下：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class JobModel extends Model
{
    protected $table          = 'jobs';
    protected $returnType     = \App\Entities\Job::class;
    protected $allowedFields = [
        'name', 'description',
    ];
}
```

此模型处理来自 jobs 表的数据，并将所有结果作为 App\Entities\Job 实例返回。当需要将记录持久化到数据库时，你可以编写自定义方法，或使用模型的 save() 方法来检查类、提取公共和私有属性并保存到数据库：

```
<?php

// Retrieve a Job instance
$job = $model->find(15);
```

(续下页)

(接上页)

```
// Make some changes
$job->name = 'Foobar';

// Save the changes
$model->save($job);
```

备注: 如果你需要频繁使用实体类，CodeIgniter 提供了内置的实体类，其中包含多个便捷功能可简化实体开发。

保存日期

在 4.5.0 版本加入.

保存数据时，如果传递*Time* 实例，它们会被转换为字符串格式。转换使用的格式定义在数据库配置 的 `dateFormat['datetime']` 和 `dateFormat['date']` 中。

备注: 在 v4.5.0 之前，Model 类中日期/时间格式硬编码为 `Y-m-d H:i:s` 和 `Y-m-d`。

删除数据

delete()

以主键值作为第一个参数，从模型表中删除匹配记录:

```
<?php

$userModel->delete(12);
```

如果模型的*$useSoftDeletes* 值为 true，此操作会将行的 `deleted_at` 设为当前日期时间。通过将第二个参数设为 true 可强制永久删除。

传递主键数组作为第一个参数可批量删除多条记录:

```
<?php

$userModel->delete([1, 2, 3]);
```

不传递参数时，其行为类似于查询构建器的 delete 方法，需要预先调用 where 条件：

```
<?php

$userModel->where('id', 12)->delete();
```

purgeDeleted()

通过永久删除所有‘deleted_at IS NOT NULL’的行来清理数据库表：

```
<?php

$userModel->purgeDeleted();
```

模型内验证

> **警告：** 模型内验证在数据存储到数据库之前执行。在此之前数据尚未验证。在验证前处理用户输入数据可能引入安全漏洞。

验证数据

Model 类提供在通过 `insert()`、`update()` 或 `save()` 方法保存到数据库前自动验证数据的功能。

重要： 更新数据时，默认情况下模型类中的验证仅验证提供的字段，以避免在更新部分字段时出现验证错误。

这意味着并非所有设置的验证规则都会在更新时检查。因此不完整数据可能通过验证。

例如，需要其他字段值的 `required*` 规则或 `is_unique` 规则可能无法按预期工作。

为避免此类问题，可通过配置更改此行为。详见*$cleanValidationRules*。

设置验证规则

第一步是在*$validationRules* 类属性中填写要应用的字段和规则。

备注： 内置验证规则列表参见可用规则。

如果有自定义错误信息，可将其放入*$validationMessages* 数组：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    // ...

    protected $validationRules = [
        'username'     => 'required|max_length[30]|alpha_numeric_
↪space|min_length[3]',
        'email'        => 'required|max_length[254]|valid_email|is_
↪unique[users.email]',
        'password'     => 'required|max_length[255]|min_length[8]',
        'pass_confirm' => 'required_with[password]|max_
↪length[255]|matches[password]',
    ];
    protected $validationMessages = [
        'email' => [
            'is_unique' => 'Sorry. That email has already been␣
↪taken. Please choose another.',
        ],
    ];
}
```

如果更愿意在验证配置文件 中组织规则和错误信息，可创建验证规则组并将*$validationRules* 设为组名：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    // ...

    protected $validationRules = 'users';
}
```

也可以通过函数设置字段验证规则：

class CodeIgniter**Model**

CodeIgniter\Model::**setValidationRule**(*$field*, *$fieldRules*)

参数

- **$field** (string) –
- **$fieldRules** (array) –

此函数设置字段验证规则。

使用示例：

```
<?php

$fieldName  = 'username';
$fieldRules = 'required|max_length[30]|alpha_numeric_space|min_
↪length[3]';

$model->setValidationRule($fieldName, $fieldRules);
```

CodeIgniter\Model::**setValidationRules**(*$validationRules*)

参数

- **$validationRules** (array) –

此函数设置验证规则。

使用示例:

```
<?php

$validationRules = [
    'username' => 'required|max_length[30]|alpha_numeric_
↪space|min_length[3]',
    'email'    => [
        'rules'  => 'required|max_length[254]|valid_email|is_
↪unique[users.email]',
        'errors' => [
            'required' => 'We really need your email.',
        ],
    ],
];
$model->setValidationRules($validationRules);
```

通过函数设置字段验证信息:

CodeIgniter\Model::**setValidationMessage**(*$field*, *$fieldMessages*)

参数

- **$field** (string) –
- **$fieldMessages** (array) –

此函数设置字段错误信息。

使用示例:

```
<?php

$fieldName              = 'name';
$fieldValidationMessage = [
    'required' => 'Your name is required here',
];
$model->setValidationMessage($fieldName,
↪$fieldValidationMessage);
```

CodeIgniter\Model::**setValidationMessages**(*$fieldMessages*)

参数

- **$fieldMessages** (array) –

此函数设置字段信息。

使用示例:

```
<?php

$fieldValidationMessage = [
    'name' => [
        'required'   => 'Your baby name is missing.',
        'min_length' => 'Too short, man!',
    ],
];
$model->setValidationMessages($fieldValidationMessage);
```

获取验证结果

当调用 insert()、update() 或 save() 方法时，数据会被验证。如果验证失败，模型返回布尔值 **false**。

获取验证错误

使用 errors() 方法获取验证错误:

```
<?php

if ($model->save($data) === false) {
    return view('updateUser', ['errors' => $model->errors()]);
}
```

返回包含字段名及其关联错误的数组，可用于在表单顶部显示所有错误或单独显示:

```
<?php if (! empty($errors)): ?>
    <div class="alert alert-danger">
    <?php foreach ($errors as $field => $error): ?>
```

(续下页)

(接上页)

```
        <p><?= esc($error) ?></p>
    <?php endforeach ?>
    </div>
<?php endif ?>
```

检索验证规则

可通过访问 validationRules 属性检索模型的验证规则:

```
<?php

$rules = $model->validationRules;
```

也可通过调用访问方法直接检索规则子集（带选项):

```
<?php

$rules = $model->getValidationRules($options);
```

$options 参数是包含一个元素的关联数组，其键为 'except' 或 'only'，值为相关字段名数组:

```
<?php

// 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']]);
```

验证占位符

模型提供简单方法来替换规则中基于传入数据的部分。这在 is_unique 验证规则中特别有用。占位符是由花括号包围的字段名（或数组键），会被匹配传入字段的 **值**替换。示例:

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class MyModel extends Model
{
    // ...

    protected $validationRules = [
        'id'    => 'max_length[19]|is_natural_no_zero',
        'email' => 'required|max_length[254]|valid_email|is_
↪unique[users.email,id,{id}]',
    ];
}
```

备注： 自 v4.3.5 起，必须为占位符字段（id）设置验证规则。

在此规则集中，声明电子邮件地址在数据库中应唯一，除了 id 匹配占位符值的行。假设表单 POST 数据如下：

```
<?php

$_POST = [
    'id'    => 4,
    'email' => 'foo@example.com',
];
```

则 {id} 占位符会被替换为数字 **4**，生成修订后的规则：

```
<?php

namespace App\Models;

use CodeIgniter\Model;
```

(续下页)

(接上页)

```
class MyModel extends Model
{
    // ...

    protected $validationRules = [
        'id'    => 'max_length[19]|is_natural_no_zero',
        'email' => 'required|max_length[254]|valid_email|is_
↪unique[users.email,id,4]',
    ];
}
```

因此在校验电子邮件唯一性时，会忽略数据库中 `id=4` 的行。

备注： 自 v4.3.5 起，如果占位符（`id`）值未通过验证，占位符不会被替换。

只要注意动态键不与表单数据冲突，这也可用于在运行时创建更动态的规则。

保护字段

为防止大规模赋值攻击，Model 类 **要求**在*$allowedFields* 类属性中列出所有可在插入和更新时修改的字段名。超出这些字段的数据会在触及数据库前被移除。这能有效防止时间戳或主键被修改。

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class MyModel extends Model
{
    // ...

    protected $allowedFields = ['name', 'email', 'address'];
}
```

有时需要在测试、迁移或种子数据时修改这些元素。此时可开关保护：

```
<?php

$model->protect(false)
    ->insert($data)
    ->protect(true);
```

运行时返回类型变更

可通过类属性*$returnType* 指定使用 **find*()** 方法时数据的返回格式。有时可能需要不同格式的数据。模型提供方法实现这一点。

备注： 这些方法仅改变下一次 **find*()** 方法调用的返回类型，之后会重置为默认值。

---

asArray()

将下一次 **find*()** 方法的数据作为关联数组返回：

```
<?php

$users = $userModel->asArray()->where('status', 'active')->
↪findAll();
```

asObject()

将下一次 **find*()** 方法的数据作为标准对象或自定义类实例返回：

```
<?php

// Return as standard objects
$users = $userModel->asObject()->where('status', 'active')->
↪findAll();

// Return as custom class instances
$users = $userModel->asObject('User')->where('status', 'active')->
↪findAll();
```

处理大量数据

处理大量数据时可能存在内存不足风险。可使用 chunk() 方法获取小块数据进行处理。第一个参数是单块检索的行数，第二个参数是处理每行数据的闭包。

此方法适用于定时任务、数据导出等大型任务。

```
<?php

$userModel->chunk(100, static function ($data) {
    // do something.
    // $data is a single row of data.
});
```

使用查询构建器

获取模型的查询构建器

CodeIgniter 模型有一个针对模型数据库连接的查询构建器实例。可随时访问此 **共享**实例：

```
<?php

$builder = $userModel->builder();
```

此构建器已配置模型的*$table*。

备注： 获取查询构建器实例后，可调用查询构建器 的方法。但由于查询构建器不是模型，不能调用模型的方法。

获取其他表的查询构建器

如需访问其他表，可获取另一个查询构建器实例。传递表名作为参数，但注意这会返回**非共享**实例：

```
<?php

$groupBuilder = $userModel->builder('groups');
```

混合使用查询构建器和模型方法

可在同一链式调用中混合使用查询构建器方法和模型的 CRUD 方法，实现优雅操作：

```
<?php

$users = $userModel->where('status', 'active')
    ->orderBy('last_login', 'asc')
    ->findAll();
```

此例中，操作的是模型持有的查询构建器共享实例。

重要： 模型并非查询构建器的完美接口。模型和查询构建器是不同目的的独立类，不应期望返回相同数据。

如果查询构建器返回结果，则原样返回。此时结果可能与模型方法返回的不同，且可能不符合预期。不会触发模型事件。

为避免意外行为，请勿在方法链末尾使用返回结果的查询构建器方法并指定模型方法。

备注： 也可无缝访问模型的数据库连接：

```
<?php

$userName = $userModel->escape($name);
```

模型事件

在模型执行的多个节点可指定多个回调方法。这些方法可用于规范化数据、哈希密码、保存关联实体等。

以下执行节点可通过类属性设置回调:

- *$beforeInsert*, *$afterInsert*
- *$beforeUpdate*, *$afterUpdate*
- *$beforeFind*, *$afterFind*
- *$beforeDelete*, *$afterDelete*
- *$beforeInsertBatch*, *$afterInsertBatch*
- *$beforeUpdateBatch*, *$afterUpdateBatch*

备注: `$beforeInsertBatch`、`$afterInsertBatch`、`$beforeUpdateBatch` 和 `$afterUpdateBatch` 自 v4.3.0 起可用。

定义回调

首先在模型中创建新类方法作为回调。

此方法始终接收 `$data` 数组作为唯一参数。

`$data` 数组的具体内容因事件而异,但始终包含键名为 `data` 的主要数据。对于 **insert*()** 或 **update*()** 方法，这是要插入/更新到数据库的键值对。主 `$data` 数组还包含传递给方法的其他值，详见事件参数。

回调方法必须返回原始 `$data` 数组以便其他回调使用完整信息。

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class MyModel extends Model
{
```

(续下页)

(接上页)

```
    // ...

    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;
    }
}
```

指定运行的回调

通过将方法名添加到相应的类属性（*$beforeInsert*、*$afterUpdate* 等）来指定回调运行时机。单个事件可添加多个回调并按序处理。同一回调可用于多个事件：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class MyModel extends Model
{
    // ...

    protected $beforeInsert = ['hashPassword'];
    protected $beforeUpdate = ['hashPassword'];

    // ...
}
```

此外，每个模型可通过设置*$allowCallbacks* 属性全局允许（默认）或禁止回调：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class MyModel extends Model
{
    // ...

    protected $allowCallbacks = false;

    // ...
}
```

也可使用 `allowCallbacks()` 方法临时更改单个模型调用的设置：

```
<?php

$model->allowCallbacks(false)->find(1); // No callbacks triggered
$model->find(1); // Callbacks subject to original property value
```

事件参数

各事件传递给回调的 `$data` 参数内容如下：

事件	$data 内容
beforeInsert	**data** = 要插入的键值对。如果向 `insert()` 传递对象或 Entity 类，会先转换为数组。
afterInsert	**id** = 新行的主键，失败时为 0。**data** = 要插入的键值对。**result** = 通过查询构建器使用的 `insert()` 方法结果。
beforeUpdate	**id** = 传递给 `update()` 方法的主键数组。**data** = 要更新的键值对。如果向 `update()` 传递对象或 Entity 类，会先转换为数组。
afterUpdate	**id** = 传递给 `update()` 方法的主键数组。**data** = 要更新的键值对。**result** = 通过查询构建器使用的 `update()` 方法结果。
beforeFind	调用 **方法**的名称，是否请求 **单例**，以及以下附加字段：
• `first()`	无附加字段
• `find()`	**id** = 要搜索行的主键。
• `findAll()`	**limit** = 要查找的行数。**offset** = 搜索期间跳过的行数。
afterFind	同 **beforeFind**，但包含结果数据行（无结果时为 null）。
beforeDelete	**id** = 传递给 `delete()` 方法的主键。**purge** = 是否硬删除软删除行的布尔值。
afterDelete	**id** = 传递给 `delete()` 方法的主键。**purge** = 是否硬删除软删除行的布尔值。**result** = 查询构建器上 `delete()` 调用的结果。**data** = 未使用。
beforeInsertBatch	**data** = 要插入的值的关联数组。如果向 `insertBatch()` 传递对象或 Entity 类，会先转换为数组。
afterInsertBatch	**data** = 要插入的值的关联数组。**result** = 通过查询构建器使用的 `insertbatch()` 方法结果。
beforeUpdateBatch	**data** = 要更新的值的关联数组。如果向 `updateBatch()` 传递对象或 Entity 类，会先转换为数组。

修改 Find* 数据

beforeFind 和 afterFind 方法都可以返回修改后的数据集来覆盖模型的正常响应。对于 afterFind，返回数组中 data 的任何修改都会自动传递回调用上下文。为了让 beforeFind 拦截查找工作流，它还必须返回一个额外的布尔值 returnData：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class MyModel extends Model
{
    // ...

    protected $beforeFind = ['checkCache'];

    // ...

    protected function checkCache(array $data)
    {
        // Check if the requested item is already in our cache
        if (isset($data['id']) && $item = $this->getCachedItem(
↪$data['id'])) {
            $data['data']       = $item;
            $data['returnData'] = true;

            return $data;
        }

        // ...
    }
}
```

手动创建模型

你不需要继承任何特殊类来为应用程序创建模型。你只需要获取数据库连接的实例即可开始使用。这允许你绕过 CodeIgniter 模型开箱即用的功能，创建完全自定义的体验。

```
<?php

namespace App\Models;

use CodeIgniter\Database\ConnectionInterface;

class UserModel
{
    protected $db;

    public function __construct(ConnectionInterface $db)
    {
        $this->db = $db;
    }
}
```

6.2.2 使用实体类

CodeIgniter 全面支持实体类, 同时保持它们的完全可选。它们通常用作存储库模式的一部分, 但如果更符合你的需求, 也可以直接与*Model* 一起使用。

- 实体用法
  - 创建实体类
  - 创建模型
  - 使用实体类
  - 快速填充属性
  - 批量访问属性
- 处理业务逻辑
  - 特殊的 *Getter/Setter*

- 数据映射
- 变更器
  - 日期变更器
  - 属性转换
- 检查更改的属性

实体用法

本质上, 实体类仅仅是一个代表单个数据库行的类。它具有表示数据库列的类属性, 并提供任何其他方法来实现该行的业务逻辑。

备注: 为了便于理解，这里的解释是基于使用数据库的情况。然而，实体也可以用于不来自数据库的数据。

然而, 关键是它不知道如何持久化自己。这是模型或存储库类的责任。这样, 如果你需要保存对象的方式发生了任何更改, 你不需要更改整个应用程序中该对象的使用方式。

这使得在快速原型阶段使用 JSON 或 XML 文件存储对象成为可能, 然后在概念证明有效时轻松切换到数据库。

让我们来看一个非常简单的用户实体示例, 并介绍如何使用它以使事情变得清楚。

假设你有一个名为 `users` 的数据库表, 具有以下模式:

```
id          - integer
username    - string
email       - string
password    - string
created_at  - datetime
```

重要: `attributes` 是一个保留字, 供内部使用。如果你将其用作列名, 实体将无法正确工作。

创建实体类

现在创建一个新的实体类。由于没有默认的位置来存储这些类, 也不符合现有的目录结构, 所以在 **app/Entities** 中创建一个新目录。在 **app/Entities/User.php** 中创建实体本身。

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class User extends Entity
{
    // ...
}
```

就这么简单, 尽管我们马上会让它更有用。

创建模型

首先在 **app/Models/UserModel.php** 创建模型, 以便我们可以与其交互:

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    protected $table          = 'users';
    protected $allowedFields = [
        'username', 'email', 'password',
    ];
    protected $returnType     = \App\Entities\User::class;
    protected $useTimestamps = true;
}
```

模型在数据库中使用 `users` 表进行所有活动。

我们设置了 `$allowedFields` 属性来包含所有我们希望外部类可以更改的字段。`id`、`created_at` 和 `updated_at` 字段由类或数据库自动处理，所以我们不希望更改这些字段。

最后，我们将我们的实体类设置为 `$returnType`。这确保了模型中所有内置的方法在从数据库返回行时，将返回我们的 User 实体类的实例，而不是像通常那样返回对象或数组。

备注: 当然，如果你向模型添加自定义方法，你必须实现它，以便返回 `$returnType` 的实例。

使用实体类

现在各部分就绪, 你将像使用任何其他类一样使用实体类:

```
<?php

$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);
```

你可能已经注意到, User 类还没有为列设置任何属性, 但你仍然可以像它们是公共属性一样访问它们。基类 CodeIgniter\Entity\Entity 会替你处理这些, 以及提供使用 isset() 检查属性, 或 unset() 属性的能力, 并跟踪对象创建或从数据库中提取后哪些列发生了更改。

备注: 实体类在内部的类属性 $attributes 中存储数据。

当 User 传递给模型的 save() 方法时, 它会自动读取属性并保存 $allowedFields 属性中列出的任何更改。它还知道是创建新行还是更新现有行。

备注: 当我们调用 insert() 时, 实体的所有值都传递给该方法, 但当我们调用 update() 时, 只传递更改的值。

快速填充属性

实体类还提供了一个方法 fill(), 允许你将键/值对数组推入类中并填充类属性。数组中的任何属性都将在实体上设置。但是, 通过模型保存时, 实际上只会将 $allowedFields 中的字段保存到数据库, 所以你可以在实体上存储其他数据, 而不必担心错误地保存多余的字段。

```
<?php

$data = $this->request->getPost();

$user = new \App\Entities\User();
$user->fill($data);
$userModel->save($user);
```

你也可以在构造函数中传递数据, 数据将在实例化过程中通过 fill() 方法传递。

```
<?php

$data = $this->request->getPost();

$user = new \App\Entities\User($data);
$userModel->save($user);
```

批量访问属性

实体类有两个方法可以将所有可用属性提取到一个数组中:toArray() 和 toRawArray()。使用原始版本将绕过魔术 "getter" 方法和转换。两个方法都可以接受一个布尔第一个参数, 指定返回的值是否应该由更改的那些过滤, 以及一个布尔最后一个参数, 以使方法递归, 以防出现嵌套的实体。

处理业务逻辑

虽然上面的例子很方便, 但它们没有帮助执行任何业务逻辑。基本实体类实现了一些智能的 __get() 和 __set() 方法, 这些方法将检查特殊方法并使用那些方法, 而不是直接使用属性, 从而允许你执行任何需要的业务逻辑或数据转换。

这是一个更新的用户实体, 提供了一些如何使用它的示例:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;
use CodeIgniter\I18n\Time;

class User extends Entity
{
    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;
    }
```

(续下页)

(接上页)

```
    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);
    }
}
```

首先要注意我们添加的方法的名称。对于每一个, 类都期望 snake_case 列名转换为 PascalCase, 并分别以 `set` 或 `get` 为前缀。然后, 每当你使用直接语法 (即 `$user->email`) 设置或检索类属性时, 就会自动调用这些方法。除非你希望它们从其他类访问, 否则这些方法不需要是公共的。例如, `created_at` 类属性将通过 `setCreatedAt()` 和 `getCreatedAt()` 方法访问。

备注: 这只适用于试图从类外部访问属性时。类内部的任何方法必须直接调用 `setX()` 和 `getX()` 方法。

在 `setPassword()` 方法中, 我们确保密码始终被散列。

在 `setCreatedAt()` 中, 我们将从模型接收的字符串转换为 DateTime 对象, 确保我们的时区为 UTC, 以便轻松转换查看者当前的时区。在 `getCreatedAt()` 中, 它将时间转换为应用程序当前时区的格式化字符串。

虽然相当简单, 但这些例子表明, 使用实体类可以以非常灵活的方式执行业务逻辑和创建愉快使用的对象。

```
<?php

// Auto-hash the password - both do the same thing
$user->password = 'my great password';
$user->setPassword('my great password');
```

特殊的 Getter/Setter

在 4.4.0 版本加入.

例如，如果你的实体的父类已经定义了一个名为 `getParent()` 的方法，并且你的实体还有一个名为 `parent` 的列，当你尝试在实体类中为 `getParent()` 方法添加业务逻辑时，该方法已经被定义了。

在这种情况下，你可以使用特殊的 getter/setter。而不是使用 `getX()/setX()`，使用 `_getX()/_setX()`。

在上面的示例中，如果你的实体有一个名为 `_getParent()` 的方法，当你获取 `$entity->parent` 时将使用该方法，当你设置 `$entity->parent` 时将使用 `_setParent()` 方法。

数据映射

在你的职业生涯的多个时间点上, 你会遇到应用程序使用的情况发生变化, 数据库中的原始列名不再有意义的情况。或者你发现你的编程风格更喜欢驼峰式类属性, 但你的数据库模式要求使用蛇形名称。这些情况可以通过实体类的数据映射功能轻松处理。

举个例子, 假设你有在整个应用程序中使用的简化的 User 实体:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class User extends Entity
{
    protected $attributes = [
        'id'         => null,
        'name'       => null, // Represents a username
        'email'      => null,
        'password'   => null,
        'created_at' => null,
        'updated_at' => null,
    ];
}
```

你的老板过来告诉你现在没有人再使用用户名了, 所以你们要切换到只使用电子邮件登录。但他们确实希望对应用程序进行一些个性化, 所以他们现在想要你把 name 字段改为代表用户的全名, 而不仅仅是目前的用户名。为了保持数据库的整洁和保证继续有意义, 你制作了一个迁移来将 name 字段重命名为 full_name, 以获得清晰度。

无视这个例子有多牵强, 我们现在对 User 类有两个选择。我们可以将类属性从 $name 修改为 $full_name, 但这将需要整个应用程序的更改。或者, 我们可以简单地将数据库中的 full_name 列映射到 $name 属性, 就完成了实体的更改:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class User extends Entity
{
    protected $attributes = [
        'id'         => null,
        'full_name'  => null, // In the $attributes, the key is the␣
↪db column name
        'email'      => null,
        'password'   => null,
        'created_at' => null,
        'updated_at' => null,
    ];

    protected $datamap = [
        // property_name => db_column_name
        'name' => 'full_name',
    ];
}
```

通过将新的数据库名称添加到 $datamap 数组中, 我们可以告诉类数据库列应该通过哪个类属性访问。数组的键是要映射到的类属性, 数组中的值是数据库中的列名称。

在这个例子中, 当模型在 User 类上设置 full_name 字段时, 它实际上会将该值分配给类的 $name 属性, 所以它可以通过 $user->name 设置和检索。该值仍然可以通过原始的 $user->full_name 访问, 这也是模型将数据取回并保存到数据库所需的。但是, unset() 和 isset() 只适用于映射的属性 $user->name, 而不适用于数据库列

名 `$user->full_name`。

备注: 当你使用数据映射时, 你必须为数据库列名定义 `set*()` 和 `get*()` 方法。在这个例子中, 你必须定义 `setFullName()` 和 `getFullName()`。

变更器

日期变更器

默认情况下, 当设置或检索名称为 *created_at*、*updated_at* 或 *deleted_at* 的字段时, 实体类会将其转换为时间 实例。Time 类以不可变的本地化方式提供了大量有用的方法。

你可以通过将名称添加到 `$dates` 属性来定义哪些属性会自动转换:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class User extends Entity
{
    protected $dates = ['created_at', 'updated_at', 'deleted_at'];
}
```

现在, 每当设置这些属性中的任何一个时, 它都会使用应用程序的当前时区 (在 **app/Config/App.php** 中设置) 转换为 Time 实例:

```
<?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();
```

(续下页)

(接上页)

```
echo $user->created_at->setTimezone('Europe/London')->
↪toDateString();
```

属性转换

你可以使用 `$casts` 属性指定实体中的属性应该转换为常见的数据类型。该选项应该是一个数组，其中键是类属性的名称，值是应该转换为的数据类型。

属性转换影响读取（获取）和写入（设置），但某些类型只影响读取（获取）。

标量类型转换

属性可以转换为以下任何数据类型：**integer**、**float**、**double**、**string**、**boolean**、**object**、**array**、**datetime**、**timestamp**、**uri** 和 **int-bool**。在类型的开头加上问号，将属性标记为可为空，例如 **?string**、**?integer**。

备注： **int-bool** 可以从 v4.3.0 开始使用。

例如, 如果你有一个带有 `is_banned` 属性的 User 实体, 可以将其转换为布尔值:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class User extends Entity
{
    protected $casts = [
        'is_banned'          => 'boolean',
        'is_banned_nullable' => '?boolean',
    ];
}
```

数组/JSON 转换

当转换为以下类型时, 数组/JSON 转换特别适用于存储序列化数组或 JSON 的字段:

- **array** 时, 它们将自动反序列化,
- **json** 时, 它们将自动设置为 `json_decode($value, false)` 的值,
- **json-array** 时, 它们将自动设置为 `json_decode($value, true)` 的值,

当你设置属性的值时。与可以将属性转换成的其他数据类型不同,

- **array** 转换类型在设置属性时将序列化,
- **json** 和 **json-array** 转换在设置属性时将使用 json_encode 函数

在该值上:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class User extends Entity
{
    protected $casts = [
        'options'        => 'array',
        'options_object' => 'json',
        'options_array'  => 'json-array',
    ];
}
```

```
<?php

$user    = $userModel->find(15);
$options = $user->options;

$options['foo'] = 'bar';

$user->options = $options;
$userModel->save($user);
```

CSV 转换

如果你知道有一个简单值的平面数组, 将它们编码为序列化或 JSON 字符串可能比原始结构更复杂。转换为逗号分隔值 (CSV) 是一个更简单的替代方法, 结果是一个使用的空间更少、更容易被人类读取的字符串:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class Widget extends Entity
{
    protected $casts = [
        'colors' => 'csv',
    ];
}
```

在数据库中存储为 “red,yellow,green” :

```
<?php

$widget->colors = ['red', 'yellow', 'green'];
```

备注: CSV 转换使用 PHP 的内部 `implode` 和 `explode` 方法, 并假设所有值都是安全的不包含逗号的字符串。对于更复杂的数据转换, 请尝试 `array` 或 `json`。

自定义转换

你可以为获取和设置数据定义自己的转换类型。

首先你需要为你的类型创建一个处理程序类。假设这个类将位于 **app/Entities/Cast** 目录中:

```
<?php
```

(续下页)

(接上页)

```
namespace App\Entities\Cast;

use CodeIgniter\Entity\Cast\BaseCast;

// The class must inherit the CodeIgniter\Entity\Cast\BaseCast class
class CastBase64 extends BaseCast
{
    public static function get($value, array $params = [])
    {
        return base64_decode($value, true);
    }

    public static function set($value, array $params = [])
    {
        return base64_encode($value);
    }
}
```

现在你需要注册它:

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class MyEntity extends Entity
{
    // Specify the type for the field
    protected $casts = [
        'key' => 'base64',
    ];

    // Bind the type to the handler
    protected $castHandlers = [
        'base64' => Cast\CastBase64::class,
    ];
}
```

(续下页)

(接上页)

```
// ...

$entity->key = 'test'; // dGVzdA==
echo $entity->key;     // test
```

如果在获取或设置值时不需要更改值。那么就不要实现相应的方法:

```
<?php

namespace App\Entities\Cast;

use CodeIgniter\Entity\Cast\BaseCast;

class CastBase64 extends BaseCast
{
    public static function get($value, array $params = [])
    {
        return base64_decode($value, true);
    }
}
```

参数

在某些情况下, 一种类型是不够的。在这种情况下, 你可以使用额外的参数。额外的参数用方括号表示, 并以逗号分隔列表, 例如 `type[param1, param2]`。

```
<?php

namespace App\Entities;

use CodeIgniter\Entity\Entity;

class MyEntity extends Entity
{
    // Define a type with parameters
    protected $casts = [
```

(续下页)

(接上页)

```
        'some_attribute' => 'class[App\SomeClass, param2, param3]',
    ];

    // Bind the type to the handler
    protected $castHandlers = [
        'class' => 'SomeHandler',
    ];
}
```

```
<?php

namespace App\Entities\Cast;

use CodeIgniter\Entity\Cast\BaseCast;

class SomeHandler extends BaseCast
{
    public static function get($value, array $params = [])
    {
        var_dump($params);
        /*
         * Output:
         * array(3) {
         *   [0]=>
         *   string(13) "App\SomeClass"
         *   [1]=>
         *   string(6) "param2"
         *   [2]=>
         *   string(6) "param3"
         * }
         */
    }
}
```

备注： 如果强制转换类型标记为 nullable 类型，如 `?bool`，并且传递的值不为 null，那么带有值 `nullable` 的参数将传递给强制转换类型处理器。如果强制转换类型有预定

义的参数，那么 `nullable` 将被添加到参数列表的末尾。

检查更改的属性

你可以检查自创建以来实体属性是否发生了更改。唯一的参数是要检查的属性名称:

```
<?php

$user = new \App\Entities\User();
$user->hasChanged('name'); // false

$user->name = 'Fred';
$user->hasChanged('name'); // true
```

或者省略参数检查整个实体的更改值:

```
<?php

$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 Forge 类

数据库 Forge 类包含帮助你管理数据库的方法。

- *初始化 Forge 类*
- *创建和删除数据库*
  - *$forge->createDatabase('db_name')*
  - *$forge->dropDatabase('db_name')*
  - *在命令行中创建数据库*

- 创建表
  - 添加字段
  - 添加键
  - 添加外键
  - 创建表格
- 删除表
  - 删除一张表
- 修改表
  - 向表中添加字段
  - 从表中删除字段
  - 修改表中的字段
  - 向表添加键
  - 删除主键
  - 删除键
  - 删除外键
  - 重命名表
- 类参考

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运行, 因为 Forge 类依赖于它。

如下加载 Forge 类:

```
<?php

$forge = \Config\Database::forge();
```

你也可以向 DB Forge 加载器传递另一个数据库组名称, 以防要管理的数据库不是默认数

据库:

```
<?php

$this->myforge = \Config\Database::forge('other_db');
```

在上面的示例中, 我们正在作为第一个参数传递一个不同的数据库组名称进行连接。

创建和删除数据库

$forge->createDatabase('db_name')

允许你创建第一个参数中指定的数据库。基于成功或失败返回 true/false:

```
<?php

if ($forge->createDatabase('my_db')) {
    echo 'Database created!';
}
```

可选的第二个参数设置为 true 将添加 `IF EXISTS` 语句，或者在创建数据库之前检查数据库是否存在（具体取决于 DBMS）。

```
<?php

$forge->createDatabase('my_db', true);
/*
 * gives CREATE DATABASE IF NOT EXISTS `my_db`
 * or will check if a database exists
 */
```

$forge->dropDatabase('db_name')

允许你删除第一个参数中指定的数据库。基于成功或失败返回 true/false:

```
<?php

if ($forge->dropDatabase('my_db')) {
```

(续下页)

(接上页)

```
    echo 'Database deleted!';
}
```

在命令行中创建数据库

CodeIgniter 支持直接从喜欢的终端使用专用的 `db:create` 命令创建数据库。通过使用此命令, 假定数据库还不存在。否则,CodeIgniter 将抱怨数据库创建失败。

首先, 只需键入命令和数据库名称 (例如 `foo`):

```
php spark db:create foo
```

如果一切顺利, 你应该会看到显示的 `Database "foo" successfully created.` 消息。

如果你在测试环境中或正在使用 SQLite3 驱动程序, 可以使用 `--ext` 选项为将创建数据库的文件传递文件扩展名。有效值为 `db` 和 `sqlite`, 默认为 `db`。请记住, 这些前面不应有句点。:

```
php spark db:create foo --ext sqlite
```

上述命令将创建名为 **WRITEPATH/foo.sqlite** 的数据库文件。

备注: 当使用特殊的 SQLite3 数据库名称 `:memory:` 时, 请注意命令仍会生成成功消息, 但不会创建数据库文件。这是因为 SQLite3 将只使用内存中的数据库。

创建表

在创建表时, 你可能希望执行几件事。添加字段、向表添加键、更改列。CodeIgniter 为此提供了一种机制。

添加字段

$forge->addField()

字段通常通过关联数组创建。在数组中, 你必须包含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) 还需要一个 constraint 键。

```
<?php

$fields = [
    'users' => [
        'type'       => 'VARCHAR',
        'constraint' => 100,
    ],
];
// will translate to "users VARCHAR(100)" when the field is added.
```

另外, 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 DEFAULT 约束。
- null/true : 在字段定义中生成 null。如果不指定, 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这一点的类型, 如 INTEGER。
- unique/true : 为字段定义生成唯一键。

```
<?php

$fields = [
    'id' => [
        'type'           => 'INT',
        'constraint'     => 5,
        'unsigned'       => true,
        'auto_increment' => true,
    ],
    'title' => [
```

(续下页)

(接上页)

```
        'type'       => 'VARCHAR',
        'constraint' => '100',
        'unique'     => true,
    ],
    'author' => [
        'type'       => 'VARCHAR',
        'constraint' => 100,
        'default'    => 'King of Town',
    ],
    'description' => [
        'type' => 'TEXT',
        'null' => true,
    ],
    'status' => [
        'type'       => 'ENUM',
        'constraint' => ['publish', 'pending', 'draft'],
        'default'    => 'pending',
    ],
];
$forge->addField($fields);
```

在定义了字段后, 可以使用 `$forge->addField($fields)` 后跟对*createTable()* 方法的调用来添加它们。

关于数据类型的注解

浮点类型

浮点类型，如 `FLOAT` 和 `DOUBLE`，表示的是近似值。因此，当需要精确值时，不应使用它们。

```
mysql> CREATE TABLE t (f FLOAT, d DOUBLE);
mysql> INSERT INTO t VALUES(99.9, 99.9);

mysql> SELECT * FROM t WHERE f=99.9;
Empty set (0.00 sec)
```

(续下页)

(接上页)

```
mysql> SELECT * FROM t WHERE f > 99.89 AND f < 99.91;
+------+------+
| f    | d    |
+------+------+
| 99.9 | 99.9 |
+------+------+
1 row in set (0.01 sec)
```

当需要保存精确的精度时，例如在处理货币数据，应使用 DECIMAL 或 NUMERIC。

TEXT

SQLSRV 上不应使用 TEXT，它已被弃用。欲知详情，请参见 ntext, text, 和 image (Transact-SQL) - SQL Server | Microsoft Learn。

ENUM

并非所有数据库都支持 ENUM。

从 v4.5.0 开始，SQLSRV Forge 会将 ENUM 数据类型转换为 VARCHAR(n)。之前的版本转换为 TEXT。

作为默认值的原始 SQL 字符串

在 4.2.0 版本加入.

从 v4.2.0 开始, $forge->addField() 接受一个 CodeIgniter\Database\RawSql 实例, 它表示原始 SQL 字符串。

```
<?php

use CodeIgniter\Database\RawSql;

$fields = [
    'id' => [
        'type'           => 'INT',
        'constraint'     => 5,
```

(续下页)

(接上页)

```
        'unsigned'       => true,
        'auto_increment' => true,
    ],
    'created_at' => [
        'type'    => 'TIMESTAMP',
        'default' => new RawSql('CURRENT_TIMESTAMP'),
    ],
];
$forge->addField($fields);
/*
gives:
    "id" INT(5) UNSIGNED NOT NULL AUTO_INCREMENT,
    "created_at" TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL
*/
```

警告: 当你使用 `RawSql` 时, 必须手动对数据进行转义。否则可能会导致 SQL 注入。

作为字段传递字符串

如果确切知道如何创建字段, 可以将字符串传递到 `addField()` 中的字段定义中:

```
<?php

$forge->addField("label varchar(100) NOT NULL DEFAULT 'default label
↪'");
```

备注: 不能在传递原始字符串作为字段后对这些字段调用 `addKey()`。

备注: 对 `addField()` 的多次调用是累积的。

创建 id 字段

创建 id 字段有一个特殊的例外。类型为 id 的字段将自动被赋值为 INT(9) 自增主键。

```
<?php

$forge->addField('id');
// gives `id` INT(9) NOT NULL AUTO_INCREMENT
```

添加键

$forge->addKey()

通常, 你会希望表具有键。这是通过 `$forge->addKey('field')` 完成的。可选的第二个参数设置为 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。你可以使用第四个参数指定名称。请注意, `addKey()` 必须在表已存在的情况下后跟对 `createTable()` 或 `processIndexes()` 的调用。

多个非主键列必须作为数组发送。以下为 MySQL 的示例输出。

```
<?php

$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(['blog_name', 'blog_label'], false, false, 'my_key_
↪name');
// gives KEY `my_key_name` (`blog_name`, `blog_label`)

$forge->addKey(['blog_id', 'uri'], false, true, 'my_key_name');
// gives UNIQUE KEY `my_key_name` (`blog_id`, `uri`)
```

$forge->addPrimaryKey()

$forge->addUniqueKey()

为了使代码更易读, 也可以使用特定方法添加主键和唯一键:

```
<?php

$forge->addPrimaryKey('blog_id', 'pd_name');
// gives PRIMARY KEY `pd_name` (`blog_id`)

$forge->addUniqueKey(['blog_id', 'uri'], 'key_name');
// gives UNIQUE KEY `key_name` (`blog_id`, `uri`)
```

备注: 当你添加主键时, 即使提供了名称,MySQL 和 SQLite 也会假定名称为 `PRIMARY`。

添加外键

外键有助于在表之间强制关系和操作。对于支持外键的表, 可以直接在 forge 中添加它们:

```
<?php

$forge->addForeignKey('users_id', 'users', 'id');
// gives CONSTRAINT `TABLENAME_users_id_foreign` FOREIGN KEY(`users_
↪id`) REFERENCES `users`(`id`)

$forge->addForeignKey(['users_id', 'users_name'], 'users', ['id',
↪'name']);
// gives CONSTRAINT `TABLENAME_users_id_foreign` FOREIGN KEY(`users_
↪id`, `users_name`) REFERENCES `users`(`id`, `name`)
```

你还可以指定约束的“更新时”和“删除时”属性的所需操作以及名称:

```
<?php
```

(续下页)

(接上页)

```
$forge->addForeignKey('users_id', 'users', 'id', 'CASCADE', 'CASCADE
↪', 'my_fk_name');
// gives CONSTRAINT `my_fk_name` FOREIGN KEY(`users_id`) REFERENCES␣
↪`users`(`id`) ON DELETE CASCADE ON UPDATE CASCADE

$forge->addForeignKey('users_id', 'users', 'id', '', 'CASCADE');
// gives CONSTRAINT `TABLENAME_users_foreign` FOREIGN KEY(`users_
↪id`) REFERENCES `users`(`id`) ON DELETE CASCADE

$forge->addForeignKey(['users_id', 'users_name'], 'users', ['id',
↪'name'], 'CASCADE', 'CASCADE', 'my_fk_name');
// gives CONSTRAINT `my_fk_name` FOREIGN KEY(`users_id`, `users_
↪name`) REFERENCES `users`(`id`, `name`) ON DELETE CASCADE ON␣
↪UPDATE CASCADE
```

备 注: SQLite3 不支持命名外键。CodeIgniter 将引用它们的 `prefix_table_column_foreign`。

创建表格

在声明字段和键之后, 可以使用以下方法创建新表格

```
<?php

$forge->createTable('table_name');
// gives CREATE TABLE table_name
```

可选的第二个参数设置为 true 将只在表不存在时创建该表。

```
<?php

$forge->createTable('table_name', true);
// creates table only if table does not exist
```

你也可以传递可选的表属性, 例如 MySQL 的 `ENGINE`:

```
<?php

$attributes = ['ENGINE' => 'InnoDB'];
$forge->createTable('table_name', false, $attributes);
// produces: CREATE TABLE `table_name` (...) ENGINE = InnoDB␣
↪DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci
```

备注: 除非指定了 CHARACTER SET 和/或 COLLATE 属性, 否则 createTable() 将始终使用配置的 *charset* 和 *DBCollat* 值添加它们, 只要它们不为空 (仅限 MySQL)。

删除表

删除一张表

执行 DROP TABLE 语句, 并可选地添加 IF EXISTS 子句。

```
<?php

// Produces: DROP TABLE `table_name`
$forge->dropTable('table_name');

// Produces: DROP TABLE IF EXISTS `table_name`
$forge->dropTable('table_name', true);
```

可以传递第三个参数以添加 CASCADE 选项, 某些驱动程序可能需要它来处理具有外键的表的删除。

```
<?php

// Produces: DROP TABLE `table_name` CASCADE
$forge->dropTable('table_name', false, true);
```

修改表

向表中添加字段

$forge->addColumn()

addColumn() 方法用于修改现有表。它接受与创建表 相同的字段数组, 可用于添加其他字段。

备注: 与创建表不同, 如果未指定 null, 列将为 NULL, 而不是 NOT NULL。

```
<?php

$fields = [
    'preferences' => ['type' => 'TEXT'],
];
$forge->addColumn('table_name', $fields);
// Executes: ALTER TABLE `table_name` ADD `preferences` TEXT
```

如果使用 MySQL 或 CUBIRD, 则可以利用它们的 AFTER 和 FIRST 子句来定位新列。

例子:

```
<?php

// Will place the new column after the `another_field` column:
$fields = [
    'preferences' => ['type' => 'TEXT', 'after' => 'another_field'],
];

// Will place the new column at the start of the table definition:
$fields = [
    'preferences' => ['type' => 'TEXT', 'first' => true],
];
```

从表中删除字段

$forge->dropColumn()

用于从表中删除列。

```
<?php

$forge->dropColumn('table_name', 'column_to_drop'); // to drop one␣
↪single column
```

用于从表中删除多个列。

```
<?php

$forge->dropColumn('table_name', 'column_1,column_2');      // by␣
↪proving comma separated column names
$forge->dropColumn('table_name', ['column_1', 'column_2']); // by␣
↪proving array of column names
```

修改表中的字段

$forge->modifyColumn()

此方法的使用与 `addColumn()` 相同, 只是它更改现有列而不是添加新列。为了更改名称, 可以将 "name" 键添加到定义字段的数组中。

```
<?php

$fields = [
    'old_name' => [
        'name' => 'new_name',
        'type' => 'TEXT',
        'null' => false,
    ],
];
$forge->modifyColumn('table_name', $fields);
```

(续下页)

(接上页)

```
// gives ALTER TABLE `table_name` CHANGE `old_name` `new_name` TEXT␣
↪NOT NULL
```

备注: `modifyColumn()` 可能会意外地更改 `NULL/NOT NULL`。因此, 建议始终为 `null` 键指定值。与创建表不同, 如果未指定 `null`, 列将为 `NULL`, 而不是 `NOT NULL`。

备注: 由于一个错误, 在 v4.3.4 之前, 即使指定 `'null' => false`,SQLite3 也可能不设置 `NOT NULL`。

备注: 由于一个错误, 在 v4.3.4 之前,Postgres 和 SQLSRV 即使指定 `'null' => true` 也会设置 `NOT NULL`。

向表添加键

在 4.3.0 版本加入.

你可以通过使用 `addKey()`、`addPrimaryKey()`、`addUniqueKey()` 或 `addForeignKey()` 和 `processIndexes()` 向现有表添加键:

```
<?php

$this->forge->addKey(['category', 'name'], false, false, 'category_
↪name');
$this->forge->addPrimaryKey('id', 'pk_actions');
$this->forge->addForeignKey('userid', 'user', 'id', '', '', 'userid_
↪fk');
$this->forge->processIndexes('actions');
/* gives:
ALTER TABLE `actions` ADD KEY `category_name` (`category`, `name`);
ALTER TABLE `actions` ADD CONSTRAINT `pk_actions` PRIMARY KEY(`id`);
ALTER TABLE `actions` ADD CONSTRAINT `userid_fk` FOREIGN KEY␣
↪(`userid`) REFERENCES `user`(`id`);
```

(续下页)

(接上页)

```
*/
```

删除主键

在 4.3.0 版本加入.

执行 DROP PRIMARY KEY。

```
<?php

// MySQLi Produces: ALTER TABLE `tablename` DROP PRIMARY KEY
// Others Produces: ALTER TABLE `tablename` DROP CONSTRAINT `pk_
↪tablename`
$forge->dropPrimaryKey('tablename');
```

删除键

执行 DROP KEY。

```
<?php

// For Indexes Produces:         DROP INDEX `users_index` ON␣
↪`tablename`
// For Unique Indexes Produces: ALTER TABLE `tablename` DROP␣
↪CONSTRAINT `users_index`
$forge->dropKey('tablename', 'users_index', false);
```

删除外键

执行 DROP FOREIGN KEY。

```
<?php

// Produces: ALTER TABLE `tablename` DROP FOREIGN KEY `users_
↪foreign`
$forge->dropForeignKey('tablename', 'users_foreign');
```

重命名表

执行 TABLE RENAME

```
<?php

$forge->renameTable('old_table_name', 'new_table_name');
// gives ALTER TABLE `old_table_name` RENAME TO `new_table_name`
```

类参考

class CodeIgniter\Database**Forge**

addColumn(*$table*[, *$field = []*])

参数

- **$table** (string) –要向其中添加列的表名
- **$field** (array) –列定义

返回

成功则为 true, 失败则为 false

返回类型

bool

向现有表添加列。用法: 参见向表中添加字段.

addField(*$field*)

参数

- **$field** (array) –要添加的字段定义

返回

\CodeIgniter\Database\Forge 实例 (方法链)

返回类型

\CodeIgniter\Database\Forge

将用于创建表的字段添加到集合中。用法: 参见添加字段。

addForeignKey(*$fieldName*, *$tableName*, *$tableField*[, *$onUpdate = ''*, *$onDelete = ''*, *$fkName = ''*])

参数

- **$fieldName** (string|string[]) –键字段的名称或字段数组
- **$tableName** (string) –父表的名称
- **$tableField** (string|string[]) –父表字段的名称或字段数组
- **$onUpdate** (string) –“更新时” 的所需操作
- **$onDelete** (string) –“删除时” 的所需操作
- **$fkName** (string) –外键的名称。这与 SQLite3 不兼容

返回

\CodeIgniter\Database\Forge 实例 (方法链)

返回类型

\CodeIgniter\Database\Forge

将用于创建表的外键添加到集合中。用法: 参见添加外键。

备注: 从 v4.3.0 开始可以使用 $fkName。

---

addKey(*$key*[, *$primary = false*[, *$unique = false*[, *$keyName = ”*]]])

参数

- **$key** (mixed) –键字段的名称或字段数组
- **$primary** (bool) –设置为 true 将其设置为主键, 否则设置为常规键
- **$unique** (bool) –设置为 true 将其设置为唯一键, 否则设置为常规键
- **$keyName** (string) –要添加的键的名称

返回

\CodeIgniter\Database\Forge 实例 (方法链)

返回类型

\CodeIgniter\Database\Forge

将用于创建表的键添加到集合中。用法: 参见添加键。

备注: 从 v4.3.0 开始可以使用 `$keyName`。

---

addPrimaryKey(*$key*[, *$keyName* = ''])

参数

- **$key** (mixed) –键字段的名称或字段数组
- **$keyName** (string) –要添加的键的名称

返回

`\CodeIgniter\Database\Forge` 实例 (方法链)

返回类型

`\CodeIgniter\Database\Forge`

将用于创建表的主键添加到集合中。用法: 参见添加键。

备注: 从 v4.3.0 开始可以使用 `$keyName`。

---

addUniqueKey(*$key*[, *$keyName* = ''])

参数

- **$key** (mixed) –键字段的名称或字段数组
- **$keyName** (string) –要添加的键的名称

返回

`\CodeIgniter\Database\Forge` 实例 (方法链)

返回类型

`\CodeIgniter\Database\Forge`

将用于创建表的唯一键添加到集合中。用法: 参见添加键。

备注: 从 v4.3.0 开始可以使用 `$keyName`。

---

createDatabase(*$dbName*[, *$ifNotExists* = *false*])

参数

- **$db_name** (string) –要创建的数据库名称
- **$ifNotExists** (string) –设置为 true 将添加 IF NOT EXISTS 子句或检查数据库是否存在

返回

成功则为 true, 失败则为 false

返回类型

bool

创建新数据库。用法: 参见创建和删除数据库。

createTable(*$table*[, *$if_not_exists = false*[, *array $attributes = []*]])

参数

- **$table** (string) –要创建的表的名称
- **$if_not_exists** (string) –设置为 true 将添加 IF NOT EXISTS 子句
- **$attributes** (string) –表属性的关联数组

返回

成功则为查询对象, 失败则为 false

返回类型

mixed

创建新表。用法: 参见创建表格。

dropColumn(*$table*, *$column_name*)

参数

- **$table** (string) –表名
- **$column_names** (mixed) –逗号分隔的字符串或列名称数组

返回

成功则为 true, 失败则为 false

返回类型

bool

从表中删除单个或多个列。用法: 参见从表中删除字段。

dropDatabase(*$dbName*)

参数

- **$dbName** (string) –要删除的数据库名称

返回

成功则为 true, 失败则为 false

返回类型

bool

删除数据库。用法: 参见创建和删除数据库。

dropKey(*$table*, *$keyName*[, *$prefixKeyName = true*])

参数

- **$table** (string) –具有键的表的名称
- **$keyName** (string) –要删除的键的名称
- **$prefixKeyName** (string) –是否要添加数据库前缀到 $keyName

返回

成功则为 true, 失败则为 false

返回类型

bool

删除索引或唯一索引。

备注: 从 v4.3.0 开始可以使用 $keyName 和 $prefixKeyName。

---

dropPrimaryKey(*$table*[, *$keyName = ''*])

参数

- **$table** (string) –要删除主键的表的名称
- **$keyName** (string) –要删除的主键的名称

返回

成功则为 true, 失败则为 false

返回类型

bool

从表中删除主键。

备注: 从 v4.3.0 开始可以使用 `$keyName`。

---

dropTable(*$table_name*[, *$if_exists = false*])

参数

- **$table** (string) –要删除的表的名称
- **$if_exists** (string) –设置为 true 将添加 `IF EXISTS` 子句

返回

成功则为 true, 失败则为 false

返回类型

bool

删除表。用法: 参见删除一张表。

processIndexes(*$table*)

在 4.3.0 版本加入.

参数

- **$table** (string) –要向其中添加索引的表的名称

返回

成功则为 true, 失败则为 false

返回类型

bool

跟在 `addKey()`、`addPrimaryKey()`、`addUniqueKey()` 和 `addForeignKey()` 之后, 向已有表添加索引。参见向表添加键。

modifyColumn(*$table*, *$field*)

参数

- **$table** (string) –表名

- **$field** (array) –列定义

返回

成功则为 true, 失败则为 false

返回类型

bool

修改表列。用法: 参见修改表中的字段。

renameTable (*$table_name*, *$new_table_name*)

参数

- **$table** (string) –表的当前名称
- **$new_table_name** (string) –表的新名称

返回

成功则为查询对象, 失败则为 false

返回类型

mixed

重命名表。用法: 参见重命名表。

6.3.2 数据库迁移

迁移是一种以结构化和有序的方式修改数据库的便捷方法。你可以手工编辑 SQL 片段, 但这样你就需要告知其他开发者他们需要运行这些片段。你也需要在下次部署到生产环境时跟踪哪些更改需要运行。

数据库表 **迁移**用于跟踪已经运行的迁移, 因此你只需确保你的迁移文件已经准备好, 并运行 `spark migrate` 命令将数据库更新到最新状态。你还可以使用 `spark migrate --all` 命令来包括所有命名空间的迁移。

- 迁移文件命名
- 创建迁移
  - 外键
  - 数据库组
  - 命名空间

- 命令行工具
  - *migrate*
  - *migrate:rollback*
  - *migrate:refresh*
  - *migrate:status*
  - *make:migration*
- 迁移配置
- 类参考

迁移文件命名

迁移文件名由时间戳前缀、下划线（_）和描述性名称（类名）组成。

- 2024-09-08-013653_AddBlogTable.php

每个迁移都使用创建迁移时的时间戳（**2024-09-08-013653**）进行编号，格式为 **YYYY-MM-DD-HHIISS**。

迁移的描述性名称（**AddBlogTable**）是 PHP 中的类名，因此你必须为其命名一个有效的类名。

前缀中的年、月、日和时间的分隔符可以是短横线（-）、下划线（_）或者不使用任何分隔符。例如：

- 2012-10-31-100538_AlterBlogTrackViews.php
- 2012_10_31_100539_AlterBlogAddTranslations.php
- 20121031100537_AddBlog.php

每个迁移根据所采取的方法按数字顺序向前或向后运行。这有助于在团队协作环境中避免编号冲突。

创建迁移

这将是为一个带博客的新网站创建的第一个迁移。所有迁移都在 **app/Database/Migrations/** 目录下, 文件名类似 **2022-01-31-013057_AddBlog.php**。

```
<?php

namespace App\Database\Migrations;

use CodeIgniter\Database\Migration;

class AddBlog extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'blog_id' => [
                'type'           => 'INT',
                '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');
    }
```

(续下页)

(接上页)

```
}
```

数据库连接和数据库 Forge 类都可以通过 `$this->db` 和 `$this->forge` 获取。

或者，你可以使用命令行调用来生成一个骨架迁移文件。更多详情请参阅命令行工具中的*make:migration*。

备注： 由于迁移类是一个 PHP 类, 每个迁移文件中的类名必须是唯一的。

外键

当你的表包含外键时, 迁移经常在尝试删除表和列时会遇到问题。要在运行迁移时暂时绕过外键检查, 可以在数据库连接上使用 `disableForeignKeyChecks()` 和 `enableForeignKeyChecks()` 方法。

```php
<?php

namespace App\Database\Migrations;

use CodeIgniter\Database\Migration;

class AddBlog extends Migration
{
    public function up()
    {
        $this->db->disableForeignKeyChecks();

        // Migration rules would go here..

        $this->db->enableForeignKeyChecks();
    }
}
```

数据库组

一个迁移只会对单个数据库组执行。如果你在 **app/Config/Database.php** 中定义了多个组，那么它将按照该配置文件中指定的 `$defaultGroup` 运行。

有时你可能需要为不同的数据库组使用不同的模式。也许你有一个数据库用于所有常规站点信息，而另一个数据库用于业务关键的数据。

你可以通过在迁移上设置 `$DBGroup` 属性来确保迁移只针对适当的组运行。此名称必须与数据库组的名称完全匹配:

```
<?php

namespace App\Database\Migrations;

use CodeIgniter\Database\Migration;

class AddBlog extends Migration
{
    protected $DBGroup = 'alternate_db_group';

    public function up()
    {
        // ...
    }

    public function down()
    {
        // ...
    }
}
```

备注: 跟踪已经运行过的迁移的 **migrations** 表将始终在默认数据库组中创建。

命名空间

迁移库可以自动扫描你在 **app/Config/Autoload.php** 中定义的所有命名空间, 或者从 Composer 等外部源加载的命名空间, 使用 `$psr4` 属性匹配目录名称。它将包含在 **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, 这将帮助你升级和降级每个模块 (命名空间) 而不影响其他命名空间。

例如, 假设我们在 Autoload 配置文件中定义了以下命名空间:

```
$psr4 = [
    APP_NAMESPACE => APPPATH,
    'MyCompany'   => ROOTPATH . 'MyCompany',
];
```

这将查找 **APPPATH/Database/Migrations** 和 **ROOTPATH/MyCompany/Database/Migrations** 中的任何迁移。这使得在你的可重用、模块化代码套件中包含迁移变得很简单。

命令行工具

CodeIgniter 自带了几个*commands*, 可通过命令行访问, 以帮助你使用迁移。这些工具使得使用迁移更加方便。这些工具主要提供了 MigrationRunner 类中可用的相同方法的访问。

migrate

使用所有可用的迁移迁移一个数据库组:

```
php spark migrate
```

你可以对 `migrate` 使用以下选项:

- `-g` - 用于指定数据库组。如果指定了该选项，只会运行指定数据库组的迁移。如果未指定，则会运行所有迁移。
- `-n` - 用于选择命名空间，否则将使用 `App` 命名空间。
- `--all` - 迁移所有命名空间到最新的迁移。

这个例子将在 test 数据库组上使用任何新的迁移迁移 `Acme\Blog` 命名空间:

For Unix:

```
php spark migrate -g test -n Acme\\Blog
```

For Windows:

```
php spark migrate -g test -n Acme\Blog
```

当使用 `--all` 选项时, 它将扫描所有命名空间, 尝试找到任何未运行的迁移。这些迁移将一起收集, 然后按创建日期排序为一组。这应该有助于最大限度地减少主应用程序和任何模块之间的潜在冲突。

migrate:rollback

回滚所有迁移到空白状态, 有效迁移到 0:

```
php spark migrate:rollback
```

你可以对 `migrate:rollback` 使用以下选项:

- `-b` - 选择批次: 自然数指定批次。
- `-f` - 强制绕过确认问题, 它仅在生产环境中询问。

migrate:refresh

首先回滚所有迁移, 然后迁移所有来刷新数据库状态:

```
php spark migrate:refresh
```

你可以对 `migrate:refresh` 使用以下选项:

- `-g` - 用于指定数据库组。如果指定了该选项，只会运行指定数据库组的迁移。如果未指定，则会运行所有迁移。
- `-n` - 用于选择命名空间，否则将使用 `App` 命名空间。
- `--all` - 刷新所有命名空间。
- `-f` - 强制绕过确认问题, 它仅在生产环境中询问。

migrate:status

显示所有迁移的列表以及它们运行的日期和时间, 如果未运行则显示 '–':

```
php spark migrate:status

...

+---------------------+-------------------+-----------------------
↪+---------+---------------------+-------+
| Namespace           | Version           | Filename              ␣
↪| Group   | Migrated On         | Batch |
+---------------------+-------------------+-----------------------
↪+---------+---------------------+-------+
| App                 | 2022-04-06-234508 | CreateCiSessionsTable␣
↪| default | 2022-04-06 18:45:14 | 2     |
| CodeIgniter\Settings | 2021-07-04-041948 | CreateSettingsTable  ␣
↪| default | 2022-04-06 01:23:08 | 1     |
| CodeIgniter\Settings | 2021-11-14-143905 | AddContextColumn     ␣
↪| default | 2022-04-06 01:23:08 | 1     |
+---------------------+-------------------+-----------------------
↪+---------+---------------------+-------+
```

你可以对 `migrate:status` 使用以下选项:

- `-g` - 用于指定数据库组。如果指定了该选项，只会检查指定数据库组的迁移。如果未指定，则会检查所有迁移。

make:migration

在 **app/Database/Migrations** 中创建一个骨架迁移文件。它会自动在文件名前加上当前时间戳。它创建的类名是文件名的大驼峰版本。

```
php spark make:migration <class> [options]
```

你可以对 `make:migration` 使用以下选项:

- `--namespace` - 设置根命名空间。默认: `APP_NAMESPACE`。
- `--suffix` - 在类名后追加组件标题。

以下选项也可用于为数据库 Sessions 生成迁移文件:

- `--session` - 为数据库 sessions 生成迁移文件。
- `--table` - 数据库 sessions 使用的表名。默认: `ci_sessions`。
- `--dbgroup` - 数据库 sessions 使用的数据库组。默认: `default`。

迁移配置

下表列出了所有迁移的配置选项, 在 **app/Config/Migrations.php** 中可用。

首选项	默认值	可选值	描述
enabled	true	true / false	启用或禁用迁移。
table	migrations	None	用于存储 schema 版本号的表名。该表始终在默认数据库组（`$defaultGroup`）中创建。
timestampFormat	Y-m-d-His_		创建迁移时使用的时间戳格式。

类参考

class `CodeIgniter\Database\`**`MigrationRunner`**

findMigrations()

返回
迁移文件数组

返回类型
`array`

返回在 `path` 属性中找到的迁移文件名数组。

latest(*$group*)

参数

- **$group** (`mixed`) –数据库组名称, 如果为 null 则使用默认数据库组。

返回

成功则为 `true`, 失败则为 `false`

返回类型

`bool`

该方法定位命名空间 (或所有命名空间) 的迁移, 确定哪些迁移尚未运行, 并按版本顺序运行它们 (命名空间交错)。

regress (*$targetBatch*, *$group*)

参数

- **$targetBatch** (`int`) –要迁移到的前一批次; 1+ 指定批次,0 还原全部, 负数指相对批次 (例如 -3 表示“往前三批”)
- **$group** (`?string`) –数据库组名称, 如果为 null 则使用默认数据库组。

返回

成功则为 `true`, 失败或找不到迁移则为 `false`

返回类型

`bool`

回滚可用于将更改回滚到以前的状态, 逐批进行。

```
<?php

$migration->regress(5);
$migration->regress(-1);
```

force (*$path*, *$namespace*, *$group*)

参数

- **$path** (`mixed`) –有效迁移文件的路径。
- **$namespace** (`mixed`) –所提供迁移的命名空间。
- **$group** (`mixed`) –数据库组名称, 如果为 null 则使用默认数据库组。

返回

成功则为 `true`, 失败则为 `false`

返回类型

bool

该方法强制单文件迁移, 不考虑顺序或批次。基于它是否已经迁移来检测 up() 或 down() 方法。

备注: 该方法仅建议用于测试, 可能会导致数据一致性问题。

---

setNamespace(*$namespace*)

参数

- **$namespace** (string|null) –应用程序命名空间。null 为所有命名空间。

返回

当前的 MigrationRunner 实例

返回类型

CodeIgniter\Database\MigrationRunner

设置库应查找迁移文件的命名空间:

```
<?php

$migration->setNamespace($namespace)->latest();
```

备注: 如果设置为 null, 则它将查找所有命名空间中的迁移文件。

---

setGroup(*$group*)

参数

- **$group** (string) –数据库组名称。

返回

当前的 MigrationRunner 实例

返回类型

CodeIgniter\Database\MigrationRunner

设置库应查找迁移文件的组:

```
<?php

$migration->setGroup($group)->latest();
```

6.3.3 数据库填充

数据库填充是向数据库中添加数据的一个简单方法。它在开发过程中特别有用, 你需要用一些样本数据来填充数据库以进行开发, 但它的用途不仅限于此。填充器可以包含一些不想放入迁移文件的静态数据, 像国家信息、地理编码表、事件或设置信息等等。

- 数据库填充器
- 嵌套填充器
- 使用填充器
  - 命令行填充
- 创建填充器文件

数据库填充器

数据库填充器是简单的类, 必须有一个 `run()` 方法, 并扩展 `CodeIgniter\Database\Seeder`。在 `run()` 内, 该类可以创建任何它需要的形式的数据。它可以通过 `$this->db` 和 `$this->forge` 访问数据库连接和伪造器。填充文件必须存储在 **app/Database/Seeds** 目录中。文件名必须与类名匹配。

```
<?php

namespace App\Database\Seeds;

use CodeIgniter\Database\Seeder;

class SimpleSeeder extends Seeder
{
    public function run()
    {
        $data = [
```

(续下页)

(接上页)

```
            '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);
    }
}
```

嵌套填充器

填充器可以通过 `call()` 方法调用其他填充器。这使你可以轻松组织一个中心填充器，但将任务组织到单独的填充器文件中:

```
<?php

namespace App\Database\Seeds;

use CodeIgniter\Database\Seeder;

class TestSeeder extends Seeder
{
    public function run()
    {
        $this->call('UserSeeder');
        $this->call('CountrySeeder');
        $this->call('JobSeeder');
    }
}
```

在 `call()` 方法中，你也可以使用完全限定的类名，这使你可以将填充器保存在自动加载程序可以找到的任何地方。这对于更模块化的代码库很有帮助:

```php
<?php

namespace App\Database\Seeds;

use CodeIgniter\Database\Seeder;

class SimpleSeeder extends Seeder
{
    public function run()
    {
        $this->call('UserSeeder');
        $this->call('My\Database\Seeds\CountrySeeder');
    }
}
```

使用填充器

你可以通过数据库配置类获取主填充器的副本:

```php
<?php

$seeder = \Config\Database::seeder();
$seeder->call('TestSeeder');
```

命令行填充

你也可以通过命令行作为迁移 CLI 工具的一部分从命令行填充数据, 如果你不想创建一个专用的控制器:

```
php spark db:seed TestSeeder
```

创建填充器文件

使用命令行, 你可以轻松生成填充器文件。

```
php spark make:seeder user --suffix
```

上述命令将输出位于 **app/Database/Seeds** 目录下的 **UserSeeder.php** 文件。

你可以通过提供 `--namespace` 选项来指定填充器文件要存储的 `root` 命名空间:

For Unix:

```
php spark make:seeder MySeeder --namespace Acme\\Blog
```

For Windows:

```
php spark make:seeder MySeeder --namespace Acme\Blog
```

如果 `Acme\Blog` 映射到 **app/Blog** 目录, 那么此命令将在 **app/Blog/Database/Seeds** 目录中生成 **MySeeder.php**。

提供 `--force` 选项将覆盖目标位置中的现有文件。

6.3.4 数据库命令

CodeIgniter 提供了一些简单的数据库管理命令。

- 显示表信息
  - 列出数据库中的表
  - 指定数据库组
  - 检索一些记录
  - 检索字段元数据

显示表信息

列出数据库中的表

db:table –show

要直接从喜欢的终端列出数据库中的所有表, 可以使用 `db:table --show` 命令:

```
php spark db:table --show
```

使用此命令时，假设数据库中已存在表。否则，CodeIgniter 会提示数据库中没有表。

指定数据库组

db:table –dbgroup

在 4.5.0 版本加入.

你可以使用 `--dbgroup` 选项来指定使用的数据库组:

```
php spark db:table --show --dbgroup tests
```

检索一些记录

db:table

当你有一个名为 `my_table` 的表时, 你可以看到表的字段名和记录:

```
php spark db:table my_table
```

如果数据库中没有表 `my_table`,CodeIgniter 会显示可用表列表以供选择。

你也可以不使用表名使用以下命令:

```
php spark db:table
```

在这种情况下, 将询问表名。

你还可以传递一些选项:

```
php spark db:table my_table --limit-rows 50 --limit-field-value 20 -
↪-desc
```

选项 `--limit-rows 50` 将行数限制为 50 行。

选项 `--limit-field-value 20` 将字段值的长度限制为 20 个字符, 以防止表输出在终端中混淆。

选项 `--desc` 将排序方向设置为 "DESC"。

检索字段元数据

db:table –metadata

当你有一个名为 `my_table` 的表时, 你可以使用 `--metadata` 选项查看元数据, 如列类型、表的最大长度:

```
php spark db:table my_table --metadata
```

使用此命令时, 假定表存在。否则,CodeIgniter 将显示表列表以供选择。此外, 你可以将此命令用作 `db:table --metadata`。

章节 7

类库和辅助函数

7.1 类库参考

7.1.1 缓存驱动

CodeIgniter 提供了一些最常用的快速动态缓存的封装。除基于文件的缓存外, 所有缓存都需要特定的服务器要求, 如果服务器要求不符, 会抛出一个致命异常。

- 使用示例
- 配置缓存
  - *$handler*
  - *$backupHandler*
  - *$prefix*
  - *$ttl*
  - *$file*
  - *$memcached*
  - *$redis*

- 命令行工具
  - *cache:clear*
  - *cache:info*
- 类参考
- 驱动程序
  - 基于文件的缓存
  - *Memcached* 缓存
  - *WinCache* 缓存
  - *Redis* 缓存
  - *Predis* 缓存
  - *Dummy* 缓存

使用示例

下面的示例展示了在控制器中的一种常见用法:

```
<?php

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;
```

你可以通过全局函数 `service()` 直接获取缓存引擎的实例:

```
<?php

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

(续下页)

(接上页)

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

配置缓存

缓存引擎的所有配置都在 **app/Config/Cache.php** 中。该文件中可用的选项如下:

$handler

这是启动缓存引擎时使用的主处理程序的名称。可用名称有:dummy、file、memcached、redis、predis、wincache。

$backupHandler

如果首选的 `$handler` 不可用, 则此处是要加载的下一个缓存处理程序。由于文件系统总是可用的, 因此这通常是 `File` 处理程序, 但可能不适合更复杂的多服务器设置。

$prefix

如果你有多个应用程序使用相同的缓存存储, 你可以在这里添加一个自定义的前缀字符串, 该字符串会添加到所有键名的前面。

$ttl

没有指定时保存项目的默认秒数。

> **警告:** 这在框架处理程序中没有使用, 因为有 60 秒的硬编码值, 但对项目和模块可能有用。这会在未来的版本中替换硬编码值。

$file

这是一组针对 **File** 处理程序的设置数组, 用来确定其应如何保存缓存文件。

$memcached

这是在使用 **Memcached** 处理程序时要使用的一组服务器数组。

$redis

使用 **Redis** 和 **Predis** 处理程序时，希望使用的 Redis 服务器的设置。

命令行工具

CodeIgniter 提供了几个可以从命令行使用的*commands*, 以帮助你使用缓存。这些工具不是使用缓存驱动所必需的, 但可能对你有帮助。

cache:clear

清除当前系统缓存:

```
php spark cache:clear
```

cache:info

显示当前系统中的文件缓存信息:

```
php spark cache:info
```

备注: 这个命令只支持 File 缓存处理程序。

类参考

class CodeIgniter\Cache**CacheInterface**

isSupported()

返回

如果支持则为 `true`, 不支持则为 `false`

返回类型

`bool`

get(*$key*) → mixed

参数

- **$key** (`string`) –缓存项名称

返回

项的值, 如果没找到则为 `null`

返回类型

`mixed`

这个方法将尝试从缓存中获取一个项。如果该项不存在, 该方法将返回 null。

例如:

```
<?php

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

remember(*string $key*, *int $ttl*, *Closure $callback*)

参数

- **$key** (`string`) –缓存项名称
- **$ttl** (`int`) –生存时间, 以秒为单位
- **$callback** (`Closure`) –当缓存项返回 null 时要调用的回调

返回

缓存项的值

返回类型

`mixed`

从缓存中获取一个项。如果返回 `null`, 则调用回调并保存结果。无论哪种方式, 都会返回该值。

save(*string $key*, *$data*[, *int $ttl = 60*])

参数

- **$key** (`string`) –缓存项名称
- **$data** (`mixed`) –要保存的数据
- **$ttl** (`int`) –生存时间, 以秒为单位, 默认 60

返回

保存成功则为 `true`, 失败则为 `false`

返回类型

`bool`

这个方法将一个项保存到缓存存储中。如果保存失败, 该方法将返回 `false`。

例如:

```
<?php

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

delete(*$key*) → bool

参数

- **$key** (`string`) –缓存项名称

返回

删除成功则为 `true`, 失败则为 `false`

返回类型

`bool`

这个方法将从缓存中删除一个特定的项。如果删除失败, 该方法将返回 false。

例如:

```
<?php

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

deleteMatching(*$pattern*) → integer

参数

- **$pattern** (string) –要匹配缓存项键的 glob 样式模式

返回

已删除项的数量

返回类型

integer

这个方法将一次性从缓存中删除多个项, 方法是通过 glob 样式模式匹配它们的键。它将返回已删除项的总数。

重要: 这个方法只在 File、Redis 和 Predis 处理程序中实现。由于局限, 在 Memcached 和 Wincache 处理程序中无法实现。

例如:

```
<?php

$cache->deleteMatching('prefix_*'); // deletes all items of
↪which keys start with "prefix_"
$cache->deleteMatching('*_suffix'); // deletes all items of
↪which keys end with "_suffix"
```

关于 glob 样式语法的更多信息, 请查看 Glob (programming)。

increment(*$key*[, *$offset = 1*]) → mixed

参数

- **$key** (string) –缓存 ID
- **$offset** (int) –要添加的步长/值

返回

成功则返回新值, 失败则返回 false

返回类型

mixed

对一个原始存储的值执行原子增量操作。

例如:

```
<?php

// '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*]) → mixed

参数

- **$key** (string) –缓存 ID
- **$offset** (int) –要减少的步长/值

返回

成功则返回新值, 失败则返回 false

返回类型

mixed

对一个原始存储的值执行原子减量操作。

例如:

```
<?php

// 'iterator' has a value of 6
$cache->decrement('iterator'); // 'iterator' is now 5
$cache->decrement('iterator', 2); // 'iterator' is now 3
```

clean()

返回

清除成功则为 true, 失败则为 false

返回类型

bool

这个方法将 '清空' 整个缓存。如果缓存文件的删除失败, 该方法将返回 false。

例如:

```
<?php

$cache->clean();
```

getCacheInfo()

返回

整个缓存数据库的信息

返回类型

mixed

这个方法将返回整个缓存的信息。

例如:

```
<?php

var_dump($cache->getCacheInfo());
```

备注: 返回的信息及数据结构取决于所使用的适配器。

---

getMetadata(*string $key*)

参数

- **$key** (string) –缓存项名称

返回

缓存项的元数据。缺少项时为 null, 如果绝对到期时间是永不过期, 则至少应包含 “expire” 键的数组。

返回类型

array|null

这个方法将返回缓存中特定项的详细信息。

例如:

```
<?php

var_dump($cache->getMetadata('my_cached_item'));
```

备注: 返回的信息和数据结构取决于所使用的适配器。一些适配器 (File、Memcached、Wincache) 对缺失的项仍然返回 `false`。

static validateKey(*string $key*, *string $prefix*)

参数

- **$key** (`string`) –潜在的缓存键
- **$prefix** (`string`) –可选的前缀

返回

验证和加前缀后的键。如果键超过了驱动的最大键长度, 它将被哈希。

返回类型

`string`

这个方法由处理程序方法用来检查键是否有效。它会对非字符串、无效字符和空长度抛出 `InvalidArgumentException`。

例如:

```
<?php

use CodeIgniter\Cache\Handlers\BaseHandler;

$prefixedKey = BaseHandler::validateKey($key, $prefix);
```

驱动程序

基于文件的缓存

这需要缓存目录确实可被应用程序写入。

使用时请小心，并确保对你的应用程序进行基准测试，因为磁盘 I/O 可能会在某个点抵消缓存的正向收益。

Memcached 缓存

可以在缓存配置文件中指定 Memcached 服务器。可用选项如下:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Cache extends BaseConfig
{
    // ...

    public $memcached = [
        'host'   => '127.0.0.1',
        'port'   => 11211,
        'weight' => 1,
        'raw'    => false,
    ];

    // ...
}
```

有关 Memcached 的更多信息, 请查看 https://www.php.net/memcached。

WinCache 缓存

在 Windows 下, 你也可以使用 WinCache 驱动程序。

有关 WinCache 的更多信息, 请查看 https://www.php.net/wincache。

Redis 缓存

Redis 是一个内存中的键值存储, 可以以 LRU 缓存模式运行。要使用它, 你需要 Redis server 和 phpredis PHP 扩展。

连接到 redis 服务器的配置选项存储在缓存配置文件中。可用选项如下:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Cache extends BaseConfig
{
    // ...

    public $redis = [
        'host'     => '127.0.0.1',
        'password' => null,
        'port'     => 6379,
        'timeout'  => 0,
        'database' => 0,
    ];

    // ...
}
```

有关 Redis 的更多信息, 请查看 https://redis.io。

Predis 缓存

Predis 是一个用于 Redis 键值存储的灵活且功能完善的 PHP 客户端库。要使用它, 从项目根目录的命令行中运行:

```
composer require predis/predis
```

有关 Redis 的更多信息, 请查看 https://github.com/nrk/predis。

Dummy 缓存

这是一个缓存后端, 将始终“未命中”。它不存储任何数据, 但允许你在不支持你选择的缓存的环境中保持缓存代码。

7.1.2 Cookie

HTTP Cookie （网页 Cookie、浏览器 Cookie）是服务器发送到用户浏览器的一小段数据。浏览器会存储该数据并在后续向同一服务器发起请求时将其带回。通常用于判断两个请求是否来自同一浏览器——例如保持用户登录状态。它为无状态的 HTTP 协议提供了状态信息记忆能力。

Cookies 主要用于以下三个场景：

- **会话管理**：登录状态、购物车、游戏分数等需要服务器记忆的信息
- **个性化设置**：用户偏好、主题及其他配置
- **行为追踪**：记录和分析用户行为

为帮助你高效地向浏览器发送 Cookies，CodeIgniter 提供了 `CodeIgniter\Cookie\Cookie` 类来抽象化 Cookie 交互。

- 创建 *Cookie*
  - 覆盖默认值
- 访问 *Cookie* 属性
- 不可变 *Cookie*
- 验证 *Cookie* 属性
  - 验证名称属性
  - 验证前缀属性
  - 验证 *SameSite* 属性
- 发送 *Cookie*
- 使用 *Cookie* 存储库
  - 从 *Response* 获取存储库
  - 创建 *CookieStore*

- 检查存储库中的 Cookie
- 获取存储库中的 Cookie
- 添加/删除存储库中的 Cookie
- 分发存储库中的 Cookie

- Cookie 个性化配置
- 类参考

创建 Cookie

目前有四种方式创建新的 `Cookie` 值对象:

```
<?php

use CodeIgniter\Cookie\Cookie;
use DateTime;

// Using the constructor
$cookie = new Cookie(
    'remember_token',

↪'f699c7fd18a8e082d0228932f3acd40e1ef5ef92efcedda32842a211d62f0aa6
↪',
    [
        'expires'  => new DateTime('+2 hours'),
        'prefix'   => '__Secure-',
        'path'     => '/',
        'domain'   => '',
        'secure'   => true,
        'httponly' => true,
        'raw'      => false,
        'samesite' => Cookie::SAMESITE_LAX,
    ],
);

// Supplying a Set-Cookie header string
```

(续下页)

(接上页)

```
$cookie = Cookie::fromHeaderString(
    'remember_
↪token=f699c7fd18a8e082d0228932f3acd40e1ef5ef92efcedda32842a211d62f0aa6;
↪ Path=/; Secure; HttpOnly; SameSite=Lax',
    false, // raw
);

// Using the fluent builder interface
$cookie = (new Cookie('remember_token'))
    ->withValue(
↪'f699c7fd18a8e082d0228932f3acd40e1ef5ef92efcedda32842a211d62f0aa6
↪')
    ->withPrefix('__Secure-')
    ->withExpires(new DateTime('+2 hours'))
    ->withPath('/')
    ->withDomain('')
    ->withSecure(true)
    ->withHTTPOnly(true)
    ->withSameSite(Cookie::SAMESITE_LAX);

// Using the global function `cookie` which implicitly calls `new␣
↪Cookie()`
$cookie = cookie('remember_token',
↪'f699c7fd18a8e082d0228932f3acd40e1ef5ef92efcedda32842a211d62f0aa6
↪');
```

构建 Cookie 对象时，仅 name 属性为必填项，其他属性均为可选。若未修改可选属性，它们的值将由 Cookie 类中保存的默认值填充。

覆盖默认值

要覆盖类中存储的默认值，可以通过静态方法 Cookie::setDefaults() 传入 Config\Cookie 实例或包含默认值的数组：

```
<?php

use CodeIgniter\Cookie\Cookie;
```

(续下页)

(接上页)

```
use Config\Cookie as CookieConfig;

// pass in a Config\Cookie instance before constructing a Cookie␣
↪class
Cookie::setDefaults(config(CookieConfig::class));
$cookie = new Cookie('login_token');

// pass in an array of defaults
$myDefaults = [
    'expires'  => 0,
    'samesite' => Cookie::SAMESITE_STRICT,
];
Cookie::setDefaults($myDefaults);
$cookie = new Cookie('login_token');
```

向 `Cookie::setDefaults()` 传递 `Config\Cookie` 实例或数组将永久覆盖默认值，直到传入新的默认值为止。

临时修改默认值

若希望临时修改默认值而非永久覆盖，可利用 `Cookie::setDefaults()` 返回旧默认值数组的特性：

```
<?php

use CodeIgniter\Cookie\Cookie;
use Config\Cookie as CookieConfig;

$oldDefaults = Cookie::setDefaults(config(CookieConfig::class));
$cookie      = new Cookie('my_token', 'muffins');

// return the old defaults
Cookie::setDefaults($oldDefaults);
```

访问 Cookie 属性

实例化后，可通过 getter 方法轻松访问 `Cookie` 属性：

```
<?php

use CodeIgniter\Cookie\Cookie;
use DateTime;
use DateTimeZone;

$cookie = new Cookie(
    'remember_token',

↪'f699c7fd18a8e082d0228932f3acd40e1ef5ef92efcedda32842a211d62f0aa6
↪',
    [
        'expires'  => new DateTime('2025-02-14 00:00:00', new␣
↪DateTimeZone('UTC')),
        'prefix'   => '__Secure-',
        'path'     => '/',
        'domain'   => '',
        'secure'   => true,
        'httponly' => true,
        'raw'      => false,
        'samesite' => Cookie::SAMESITE_LAX,
    ],
);

$cookie->getName();              // 'remember_token'
$cookie->getPrefix();            // '__Secure-'
$cookie->getPrefixedName();      // '__Secure-remember_token'
$cookie->getExpiresTimestamp(); // UNIX timestamp
$cookie->getExpiresString();     // 'Fri, 14-Feb-2025 00:00:00 GMT'
$cookie->isExpired();            // false
$cookie->getMaxAge();            // the difference from time() to␣
↪expires
$cookie->isRaw();                // false
$cookie->isSecure();             // true
$cookie->getPath();              // '/'
```

(续下页)

(接上页)

```
$cookie->getDomain();           // ''
$cookie->isHTTPOnly();          // true
$cookie->getSameSite();         // 'Lax'

// additional getter
$cookie->getId(); // '__Secure-remember_token;;/'

// when using `setcookie()`'s alternative signature on PHP 7.3+
// you can easily use the `getOptions()` method to supply the
// $options parameter
$cookie->getOptions();
```

不可变 Cookie

`Cookie` 实例是 HTTP Cookie 的不可变值对象。由于其不可变性，修改实例属性不会影响原实例，修改操作 **始终**返回新实例。需保留新实例方可使用：

```
<?php

use CodeIgniter\Cookie\Cookie;

$cookie = new Cookie('login_token', 'admin');
$cookie->getName(); // 'login_token'

$cookie->withName('remember_token');
$cookie->getName(); // 'login_token'

$new = $cookie->withName('remember_token');
$new->getName(); // 'remember_token'
```

验证 Cookie 属性

HTTP Cookie 受多个规范约束以确保浏览器接受。因此在创建或修改 `Cookie` 的某些属性时，会进行规范符合性验证。若验证失败将抛出 `CookieException`。

验证名称属性

Cookie 名称可为任意 US-ASCII 字符，但以下字符除外：

- 控制字符
- 空格或制表符
- 分隔字符：`( ) < > @ , ; : \ " / [ ] ? = { }`

若设置 `$raw` 参数为 `true` 将执行严格验证，因为 PHP 的 setcookie() 和 setrawcookie() 会拒绝无效名称。此外，名称不能为空字符串。

验证前缀属性

使用 `__Secure-` 前缀时，必须设置 `$secure` 标志为 `true`。使用 `__Host-` 前缀时需满足：

- `$secure` 标志设为 `true`
- `$domain` 为空
- `$path` 必须为 `/`

验证 SameSite 属性

SameSite 属性接受三个值：

- **Lax** （宽松模式）：跨站子请求（如图片加载）不发送 Cookie，但导航到源站时（如点击链接）会发送
- **Strict** （严格模式）：仅在第一方上下文中发送 Cookie
- **None** （无限制）：所有上下文中均发送 Cookie

CodeIgniter 允许将 SameSite 设为空字符串，此时使用 `Cookie` 类保存的默认值。可通过上文所述的 `Cookie::setDefaults()` 修改默认值。

现代浏览器规范要求未指定 SameSite 时默认使用 `Lax`。若 SameSite 设为空字符串且默认值也为空，则 Cookie 将被赋予 `Lax` 值。

当设置 SameSite 为 `None` 时，必须同时设置 `Secure` 为 `true`。

`Cookie` 类接受不区分大小写的 SameSite 值，也可使用类常量简化操作：

```
<?php

use CodeIgniter\Cookie\Cookie;

Cookie::SAMESITE_LAX;        // 'lax'
Cookie::SAMESITE_STRICT;     // 'strict'
Cookie::SAMESITE_NONE;       // 'none'
```

发送 Cookie

将 `Cookie` 对象存入 Response 对象的 `CookieStore` 中，框架会自动发送 Cookie。

使用*`CodeIgniter\HTTP\Response::setCookie()`* 设置：

```
<?php

use CodeIgniter\Cookie\Cookie;

$response = service('response');

$cookie = new Cookie(
    'remember_token',

↪'f699c7fd18a8e082d0228932f3acd40e1ef5ef92efcedda32842a211d62f0aa6
↪',
    [
        'max-age' => 3600 * 2, // Expires in 2 hours
    ],
);

$response->setCookie($cookie);
```

也可使用*`set_cookie()`* 辅助函数：

```
<?php

use CodeIgniter\Cookie\Cookie;

helper('cookie');

$cookie = new Cookie(
    'remember_token',
 ↪'f699c7fd18a8e082d0228932f3acd40e1ef5ef92efcedda32842a211d62f0aa6
 ↪',
    [
        'max-age' => 3600 * 2, // Expires in 2 hours
    ],
);

set_cookie($cookie);
```

使用 Cookie 存储库

备注： 通常无需直接操作 CookieStore。

`CookieStore` 类表示 `Cookie` 对象的不可变集合。

从 Response 获取存储库

可通过当前 `Response` 对象访问 `CookieStore` 实例：

```
<?php

$cookieStore = service('response')->getCookieStore();
```

创建 CookieStore

CodeIgniter 提供三种方式创建新 `CookieStore` 实例：

```
<?php

use CodeIgniter\Cookie\Cookie;
use CodeIgniter\Cookie\CookieStore;

// Passing an array of `Cookie` objects in the constructor
$store = new CookieStore([
    new Cookie('login_token'),
    new Cookie('remember_token'),
]);

// Passing an array of `Set-Cookie` header strings
$store = CookieStore::fromCookieHeaders([
    'remember_token=me; Path=/; SameSite=Lax',
    'login_token=admin; Path=/; SameSite=Lax',
]);

// using the global `cookies` function
$store = cookies([new Cookie('login_token')], false);

// retrieving the `CookieStore` instance saved in our current␣
↪`Response` object
$store = cookies();
```

备注： 使用全局函数*cookies()* 时，仅当第二个参数 `$getGlobal` 设为 `false` 时才会考虑传入的 `Cookie` 数组。

检查存储库中的 Cookie

可通过多种方式检查 CookieStore 实例中是否存在某 Cookie 对象:

```
<?php

use CodeIgniter\Cookie\Cookie;
use CodeIgniter\Cookie\CookieStore;

// check if cookie is in the current cookie collection
$store = new CookieStore([
    new Cookie('login_token'),
    new Cookie('remember_token'),
]);
$store->has('login_token');

// check if cookie is in the current Response's cookie collection
cookies()->has('login_token');
service('response')->hasCookie('remember_token');

// using the cookie helper to check the current Response
// not available to v4.1.1 and lower
helper('cookie');
has_cookie('login_token');
```

获取存储库中的 Cookie

从 Cookie 集合中检索 Cookie 实例非常简单:

```
<?php

use CodeIgniter\Cookie\Cookie;
use CodeIgniter\Cookie\CookieStore;
use Config\Services;

// getting cookie in the current cookie collection
$store = new CookieStore([
    new Cookie('login_token'),
```

(续下页)

(接上页)

```
    new Cookie('remember_token'),
]);
$store->get('login_token');

// getting cookie in the current Response's cookie collection
cookies()->get('login_token');
service('response')->getCookie('remember_token');

// using the cookie helper to get cookie from the Response's cookie␣
↪collection
helper('cookie');
get_cookie('remember_token');
```

直接从 CookieStore 获取实例时，无效名称会抛出 CookieException：

```
<?php

// throws CookieException
$store->get('unknown_cookie');
```

从当前 Response 的 Cookie 集合获取时，无效名称返回 null：

```
<?php

cookies()->get('unknown_cookie'); // null
```

若从 Response 获取时不带参数，将显示存储库中所有 Cookie 对象：

```
<?php

cookies()->get(); // array of Cookie objects

// alternatively, you can use the display method
cookies()->display();

// or even from the Response
service('response')->getCookies();
```

备注: 辅助函数*get_cookie()* 从当前 Request 对象而非 Response 获取 Cookie。该函数检查 $_COOKIE 数组并直接获取。

添加/删除存储库中的 Cookie

如前所述，CookieStore 对象不可变。需保存修改后的实例才能生效，原实例保持不变：

```
<?php

use CodeIgniter\Cookie\Cookie;
use CodeIgniter\Cookie\CookieStore;

$store = new CookieStore([
    new Cookie('login_token'),
    new Cookie('remember_token'),
]);

// adding a new Cookie instance
$new = $store->put(new Cookie('admin_token', 'yes'));

// removing a Cookie instance
$new = $store->remove('login_token');
```

备注: 从存储库删除 Cookie **不会**从浏览器删除。若要从浏览器删除 Cookie，必须向存储库添加同名空值 Cookie。

与当前 Response 对象中的 Cookies 交互时，可安全添加/删除 Cookies，无需担心集合的不可变性。Response 对象会自动替换为修改后的实例：

```
<?php

service('response')->setCookie('admin_token', 'yes');
service('response')->deleteCookie('login_token');
```

(续下页)

(接上页)

```
// using the cookie helper
helper('cookie');
set_cookie('admin_token', 'yes');
delete_cookie('login_token');
```

分发存储库中的 Cookie

重要: 该方法在 4.1.6 版本弃用，4.6.0 版本移除。

通常无需手动发送 Cookie，CodeIgniter 会自动处理。如需手动发送，可使用 `dispatch` 方法。需通过 `headers_sent()` 检查确保标头未发送：

```
<?php

use CodeIgniter\Cookie\Cookie;
use CodeIgniter\Cookie\CookieStore;

$store = new CookieStore([
    new Cookie('login_token'),
    new Cookie('remember_token'),
]);

$store->dispatch(); // After dispatch, the collection is now empty.
```

Cookie 个性化配置

Cookie 类已预设合理默认值以确保正常创建。可通过修改 **app/Config/Cookie.php** 中的 Config\Cookie 类配置项来自定义：

设置项	选项/类型	默认值	描述
$prefix	string	''	Cookie 名前缀
$expires	DateTimeInterface|string|int	0	过期时间戳
$path	string	/	Cookie 路径属性
$domain	string	''	Cookie 域名属性（带尾部斜线）
$secure	true/false	false	是否仅通过 HTTPS 发送
$httponly	true/false	true	是否禁止 JavaScript 访问
$samesite	Lax/None/Strict	Lax	SameSite 属性
$raw	true/false	false	是否使用 setrawcookie() 发送

运行时可通过 Cookie::setDefaults() 方法手动设置新默认值。

类参考

class CodeIgniter\Cookie**Cookie**

static setDefaults([*$config = []*])

参数

- **$config** (\Config\Cookie|array) –配置数组或实例

返回类型

array

返回

旧默认值数组

通过注入 Config\Cookie 配置或数组来设置 Cookie 实例的默认属性。

static fromHeaderString(*string $header*[, *bool $raw = false*])

参数

- **$header** (string) –Set-Cookie 标头字符串
- **$raw** (bool) –是否使用原始 Cookie

返回类型

Cookie

返回

Cookie 实例

Throws

CookieException

从 Set-Cookie 标头创建新 Cookie 实例。

__construct(*string $name*[, *string $value* = ''[, *array $options* = *[]*]])

参数

- **$name** (string) –Cookie 名称
- **$value** (string) –Cookie 值
- **$options** (array) –Cookie 选项

返回类型

Cookie

返回

Cookie 实例

Throws

CookieException

构造新 Cookie 实例。

getId()

返回类型

string

返回

用于 Cookie 集合索引的 ID

getPrefix() → string

getName() → string

getPrefixedName() → string

getValue() → string

getExpiresTimestamp() → int

getExpiresString() → string

isExpired() → bool

getMaxAge() → int

getDomain() → string

getPath() → string

isSecure() → bool

isHTTPOnly() → bool

getSameSite() → string

isRaw() → bool

getOptions() → array

withRaw([*bool $raw = true*])

参数

- **$raw** (bool) –

返回类型

Cookie

返回

新 Cookie 实例

创建带更新 URL 编码选项的新 Cookie。

withPrefix([*string $prefix = ''*])

参数

- **$prefix** (string) –

返回类型

`Cookie`

返回

新 `Cookie` 实例

创建带新前缀的 Cookie。

withName(*string $name*)

参数

- **$name** (`string`) –

返回类型

`Cookie`

返回

新 `Cookie` 实例

创建带新名称的 Cookie。

withValue(*string $value*)

参数

- **$value** (`string`) –

返回类型

`Cookie`

返回

新 `Cookie` 实例

创建带新值的 Cookie。

withExpires(*$expires*)

参数

- **$expires** (`DateTimeInterface|string|int`) –

返回类型

`Cookie`

返回

新 `Cookie` 实例

创建带新过期时间的 Cookie。

withExpired()

返回类型

Cookie

返回

新 Cookie 实例

创建即将过期的 Cookie。

withNeverExpiring()

重要: 该方法在 4.2.6 版本弃用，4.6.0 版本移除。

参数

- **$name** (string) –

返回类型

Cookie

返回

新 Cookie 实例

创建永不过期的 Cookie（已移除）。

withDomain(*?string $domain*)

参数

- **$domain** (string|null) –

返回类型

Cookie

返回

新 Cookie 实例

创建带新域名的 Cookie。

withPath(*?string $path*)

参数

- **$path** (string|null) –

返回类型

`Cookie`

返回

新 `Cookie` 实例

创建带新路径的 Cookie。

withSecure([*bool $secure = true*])

参数

- **$secure** (`bool`) –

返回类型

`Cookie`

返回

新 `Cookie` 实例

创建带新“Secure”属性的 Cookie。

withHTTPOnly([*bool $httponly = true*])

参数

- **$httponly** (`bool`) –

返回类型

`Cookie`

返回

新 `Cookie` 实例

创建带新“HttpOnly”属性的 Cookie。

withSameSite(*string $samesite*)

参数

- **$samesite** (`string`) –

返回类型

`Cookie`

返回

新 `Cookie` 实例

创建带新“SameSite”属性的 Cookie。

toHeaderString()

返回类型

string

返回

可作为标头字符串传递的字符串表示

toArray()

返回类型

array

返回

Cookie 实例的数组表示

class CodeIgniter\Cookie**CookieStore**

static fromCookieHeaders(*array $headers*[, *bool $raw = false*])

参数

- **$header** (array) –Set-Cookie 标头数组
- **$raw** (bool) –是否使用原始 Cookie

返回类型

CookieStore

返回

CookieStore 实例

Throws

CookieException

从 Set-Cookie 标头数组创建 CookieStore。

__construct(*array $cookies*)

参数

- **$cookies** (array) –Cookie 对象数组

返回类型

CookieStore

返回

CookieStore 实例

Throws

`CookieException`

has(*string $name*[, *string $prefix* = ''[, *?string $value* = *null*]]) → bool

参数

- **$name** (`string`) –Cookie 名称
- **$prefix** (`string`) –Cookie 前缀
- **$value** (`string|null`) –Cookie 值

返回类型

`bool`

返回

检查指定名称和前缀的 Cookie 是否存在

get(*string $name*[, *string $prefix* = '']) → Cookie

参数

- **$name** (`string`) –Cookie 名称
- **$prefix** (`string`) –Cookie 前缀

返回类型

`Cookie`

返回

获取指定名称和前缀的 Cookie 实例

Throws

`CookieException`

put(*Cookie $cookie*) → CookieStore

参数

- **$cookie** (*`Cookie`*) –Cookie 对象

返回类型

`CookieStore`

返回

新 `CookieStore` 实例

存储新 Cookie 并返回新集合，原集合保持不变。

remove(*string $name*[, *string $prefix* = ''])→ CookieStore

参数

- **$name** (string) –Cookie 名称
- **$prefix** (string) –Cookie 前缀

返回类型

CookieStore

返回

新 CookieStore 实例

移除 Cookie 并返回新集合，原集合保持不变。

dispatch()→ void

重要: 该方法在 4.1.6 版本弃用，4.6.0 版本移除。

返回类型

void

分发存储库中所有 Cookies（已移除）。

display()→ array

返回类型

array

返回

返回存储中的所有 Ccookie

clear()→ void

返回类型

void

清除 Cookie 集合。

7.1.3 跨域资源共享 (CORS)

在 4.5.0 版本加入.

跨域资源共享 (CORS) 是一种基于 HTTP 头的安全机制，允许服务器指示浏览器应允许从其自身以外的任何来源（域名、协议或端口）加载资源。

CORS 通过在 HTTP 请求和响应中添加头来工作，以指示请求的资源是否可以跨不同来源共享，从而帮助防止恶意攻击，如跨站请求伪造 (CSRF) 和数据盗窃。

如果你不熟悉 CORS 和 CORS 头，请阅读 MDN 上的 CORS 文档。

CodeIgniter 提供了 CORS 过滤器和 helper 类。

- 配置 *CORS*
  - 设置默认配置
  - 启用 *CORS*
  - 检查路由和过滤器
  - 设置其他配置
- 类参考

配置 CORS

设置默认配置

可以通过 **app/Config/Cors.php** 配置 CORS。

至少需要设置 `$default` 属性中的以下项目：

- `allowedOrigins`: 明确列出你想要允许的来源。
- `allowedHeaders`: 明确列出你想要允许的 HTTP 头。
- `allowedMethods`: 明确列出你想要允许的 HTTP 方法。

> **警告:** 基于最小特权原则，只应允许必要的最小来源、方法和头。

如果你在跨域请求中发送凭证（例如，cookies），请将 `supportsCredentials` 设置为 `true`。

启用 CORS

要启用 CORS，你需要做两件事：

1. 为允许 CORS 的路由指定 `cors` 过滤器。
2. 为 CORS 预检请求添加 **OPTIONS** 路由。

设置路由

你可以在 **app/Config/Routes.php** 中为路由设置 `cors` 过滤器。

例如，

```
<?php

use CodeIgniter\Router\RouteCollection;

$routes->group('', ['filter' => 'cors'], static function␣
↪(RouteCollection $routes): void {
    $routes->resource('product');

    $routes->options('product', static function () {
        // Implement processing for normal non-preflight OPTIONS␣
↪requests,
        // if necessary.
        $response = response();
        $response->setStatusCode(204);
        $response->setHeader('Allow:', 'OPTIONS, GET, POST, PUT,␣
↪PATCH, DELETE');

        return $response;
    });
    $routes->options('product/(:any)', static function () {});
});
```

不要忘记为预检请求添加 OPTIONS 路由。因为如果路由不存在，控制器过滤器（必需过滤器除外）将不起作用。

CORS 过滤器处理所有预检请求，因此通常不会调用 OPTIONS 路由的闭包控制器。

在 Config\Filters 中设置

或者，你可以在 **app/Config/Filters.php** 中为 URI 路径设置 `cors` 过滤器。

例如，

```
<?php

namespace Config;

use CodeIgniter\Config\Filters as BaseFilters;

// ...

class Filters extends BaseFilters
{
    // ...
    public array $filters = [
        // ...
        'cors' => [
            'before' => ['api/*'],
            'after'  => ['api/*'],
        ],
    ];
}
```

不要忘记为预检请求添加 OPTIONS 路由。因为如果路由不存在，控制器过滤器（必需过滤器除外）将不起作用。

例如，

```
<?php

use CodeIgniter\Router\RouteCollection;
```

(续下页)

(接上页)

```
$routes->group('', ['filter' => 'cors'], static function␣
↪(RouteCollection $routes): void {
    $routes->options('api/(:any)', static function () {});
});
```

CORS 过滤器处理所有预检请求，因此通常不会调用 OPTIONS 路由的闭包控制器。

检查路由和过滤器

配置完成后，你可以使用*spark 路由* 命令检查路由和过滤器。

设置其他配置

如果你想使用不同于默认配置的配置，请在 **app/Config/Cors.php** 中添加一个属性。

例如，添加 `$api` 属性。

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

/**
 * Cross-Origin Resource Sharing (CORS) Configuration
 *
 * @see https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
 */
class Cors extends BaseConfig
{
    // ...

    public array $api = [
        'allowedOrigins'          => ['https://app.example.com'],
        'allowedOriginsPatterns' => [],
        'supportsCredentials'     => true,
        'allowedHeaders'          => ['Authorization', 'Content-Type
```

(续下页)

(接上页)

```
↪'],
        'exposedHeaders'          => [],
        'allowedMethods'          => ['GET', 'POST', 'PUT', 'DELETE
↪'],
        'maxAge'                  => 7200,
    ];
}
```

属性名称（在上述示例中为 `api`）将成为配置名称。

然后，像 `cors:api` 一样将属性名称指定为过滤器参数：

```
<?php

use CodeIgniter\Router\RouteCollection;

$routes->group('api', ['filter' => 'cors:api'], static function␣
↪(RouteCollection $routes): void {
    $routes->resource('user');

    $routes->options('user', static function () {});
    $routes->options('user/(:any)', static function () {});
});
```

你也可以使用过滤器参数。

类参考

class `CodeIgniter\HTTP\`**Cors**

`CodeIgniter\HTTP\Cors::`**addResponseHeaders**(*RequestInterface $request*, *ResponseInterface $response*) → ResponseInterface

参数

- **$request** (`RequestInterface`) –请求实例
- **$response** (`ResponseInterface`) –响应实例

返回

响应实例

返回类型

`ResponseInterface`

添加 CORS 的响应头。

`CodeIgniter\HTTP\Cors::`**`handlePreflightRequest`**(*RequestInterface $request*, *ResponseInterface $response*) → ResponseInterface

参数

- **`$request`** (`RequestInterface`) –请求实例
- **`$response`** (`ResponseInterface`) –响应实例

返回

响应实例

返回类型

`ResponseInterface`

处理预检请求。

`CodeIgniter\HTTP\Cors::`**`isPreflightRequest`**(*IncomingRequest $request*) → bool

参数

- **`$request`** (`IncomingRequest`) –请求实例

返回

如果是预检请求则返回 True。

返回类型

`bool`

检查请求是否为预检请求。

7.1.4 CURLRequest 类

`CURLRequest` 类是一个基于 CURL 的轻量级 HTTP 客户端，允许你与其他网站和服务器进行通信。它可以用于获取 Google 搜索结果、检索网页或图像，以及与 API 交互等多种用途。

- *CURLRequest* 配置
  - 共享选项
- 加载类库
- 使用类库
  - 发起请求
  - 使用响应
- 请求选项
  - *allow_redirects*
  - *auth*
  - *body*
  - *cert*
  - *connect_timeout*
  - *cookie*
  - *debug*
  - *delay*
  - *form_params*
  - *headers*
  - *http_errors*
  - *json*
  - *multipart*
  - *proxy*
  - *query*

- *timeout*
- *user_agent*
- *verify*
- *force_ip_resolve*
- *version*

该类的设计灵感来源于 Guzzle HTTP Client 库，因为它是使用最广泛的 HTTP 客户端库之一。在可能的情况下，我们保持了语法的一致性，以便当你的应用需要比本库更强大的功能时，只需极少修改即可迁移到 Guzzle。

备注： 本类需要 PHP 安装 cURL 库。这是非常常见的库，通常默认可用，但部分主机可能未提供，若遇到问题请与主机服务商确认。

CURLRequest 配置

共享选项

重要： 此设置仅用于向后兼容。新项目请勿使用。即使你正在使用，我们也建议禁用它。

备注： 自 v4.4.0 起，默认值已改为 `false`。

若要在请求间共享所有选项，请在 **app/Config/CURLRequest.php** 中将 `$shareOptions` 设为 `true`：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class CURLRequest extends BaseConfig
```

(续下页)

(接上页)

```
{
    // ...
    public bool $shareOptions = true;
}
```

如果使用类的实例发送多个请求，此行为可能导致带有不必要标头和正文的错误请求。

备注： 在 v4.2.0 之前，由于一个错误，即使 `$shareOptions` 设为 false 请求体也不会重置。

加载类库

可以通过手动加载或通过*Services 类* 加载本库。

使用 Services 类加载时，调用 `curlrequest()` 方法或全局函数 `service()`：

```
<?php

$client = service('curlrequest'); // Since v4.5.0, this code is␣
↪recommended due to performance improvements

// The code above is the same as the code below.
$client = \Config\Services::curlrequest();
```

你可以在第一个参数中传递默认选项数组来修改 cURL 处理请求的方式。选项将在本文档后续部分说明：

```
<?php

$options = [
    'baseURI' => 'http://example.com/api/v1/',
    'timeout' => 3,
];
$client = service('curlrequest', $options);
```

备注： 当 `$shareOptions` 为 false 时，传递给类构造函数的默认选项将用于所有请求。

其他选项将在发送请求后重置。

手动创建类时，需要传入几个依赖项。第一个参数是 `Config\App` 类的实例，第二个是 URI 实例，第三个是 Response 对象，第四个是可选的默认 `$options` 数组：

```
<?php

use Config\App;

$client = new \CodeIgniter\HTTP\CURLRequest(
    config(App::class),
    new \CodeIgniter\HTTP\URI(),
    new \CodeIgniter\HTTP\Response(config(App::class)),
    $options,
);
```

使用类库

使用 CURL 请求只需创建 Request 并获取*Response 对象*。它负责处理通信，之后你可以完全控制信息处理方式。

发起请求

主要通过 `request()` 方法进行通信，该方法触发请求并返回 Response 实例。它接受 HTTP 方法、URL 和选项数组作为参数：

```
<?php

$client = service('curlrequest');

$response = $client->request('GET', 'https://api.github.com/user', [
    'auth' => ['user', 'pass'],
]);
```

重要： 默认情况下，如果返回的 HTTP 状态码大于等于 400，CURLRequest 将抛出 `HTTPException`。若需获取响应，请参阅*http_errors* 选项。

备注: 当 $shareOptions 为 false 时，传递给方法的选项将用于该请求。发送请求后这些选项会被清除。若要将选项应用于所有请求，请在构造函数中传递选项。

由于响应是 CodeIgniter\HTTP\Response 的实例，你可以访问所有常规信息:

```
<?php

echo $response->getStatusCode();
echo $response->getBody();
echo $response->header('Content-Type');
$language = $response->negotiateLanguage(['en', 'fr']);
```

虽然 request() 方法最灵活，但也可以使用以下快捷方法。它们都将 URL 作为第一个参数，选项数组作为第二个:

```
<?php

$client->get('http://example.com');
$client->delete('http://example.com');
$client->head('http://example.com');
$client->options('http://example.com');
$client->patch('http://example.com');
$client->put('http://example.com');
$client->post('http://example.com');
```

基础 URI

可以在类实例化时通过选项设置 baseURI。这允许你设置基础 URI，之后使用该客户端的所有请求都使用相对 URL。这在处理 API 时特别方便:

```
<?php

$client = service('curlrequest', [
    'baseURI' => 'https://example.com/api/v1/',
]);

// GET http:example.com/api/v1/photos
```

(续下页)

(接上页)

```
$client->get('photos');

// GET http:example.com/api/v1/photos/13
$client->delete('photos/13');
```

当向 `request()` 方法或任何快捷方法提供相对 URI 时，它将根据 RFC 2986 第 2 节 描述的规则与 baseURI 组合。以下是组合解析的示例：

baseURI	URI	结果
http://foo.com	/bar	http://foo.com/bar
http://foo.com/foo	/bar	http://foo.com/bar
http://foo.com/foo	bar	http://foo.com/bar
http://foo.com/foo/	bar	http://foo.com/foo/bar
http://foo.com	http://baz.com	http://baz.com
http://foo.com/?bar	bar	http://foo.com/bar

使用响应

每个 `request()` 调用返回的 Response 对象包含大量有用信息和实用方法。最常用的方法用于确定响应内容。

获取状态码和原因短语：

```
<?php

$code   = $response->getStatusCode();   // 200
$reason = $response->getReasonPhrase(); // OK
```

从响应中检索标头：

```
<?php

// Get a header line
echo $response->getHeaderLine('Content-Type');

// Get all headers
foreach ($response->headers() as $name => $value) {
```

(续下页)

(接上页)

```
    echo $name . ': ' . $response->getHeaderLine($name) . "\n";
}
```

使用 getBody() 方法获取正文:

```
<?php

$body = $response->getBody();
```

正文是远程服务器返回的原始内容。如果内容类型需要格式化，你需要确保脚本能处理:

```
<?php

if (str_contains($response->header('content-type'), 'application/
↪json')) {
    $body = json_decode($body);
}
```

请求选项

本节描述可传递给构造函数、request() 方法或任何快捷方法的所有可用选项。

allow_redirects

默认情况下，cURL 不会跟踪远程服务器返回的任何 “Location:” 标头。allow_redirects 选项允许你修改此行为。

设为 true 时跟踪重定向:

```
<?php

$client->request('GET', 'http://example.com', ['allow_redirects' =>␣
↪true]);
/*
 * Sets the following defaults:
 *    'max'        => 5,                  // Maximum number of␣
```

(续下页)

(接上页)

```
↪redirects to follow before stopping
 *    'strict'     => true,              // Ensure POST requests stay␣
↪POST requests through redirects
 *    'protocols' => ['http', 'https'] // Restrict redirects to one␣
↪or more protocols
 */
```

警告: 请注意启用重定向可能会跳转到意外 URL，并可能导致 SSRF 攻击。

设为 `false` 将应用请求的默认设置:

```
<?php

$client->request('GET', 'http://example.com', ['allow_redirects' =>␣
↪false]);
```

可以传递数组作为 `allow_redirects` 的值来指定新设置:

```
<?php

$client->request('GET', 'http://example.com', ['allow_redirects' =>␣
↪[
    'max'        => 10,
    'protocols' => ['https'], // Force HTTPS domains only.
]]);
```

备注: 当 PHP 处于安全模式或启用 open_basedir 时，跟踪重定向无效。

auth

允许为 HTTP 基本认证 和 摘要认证 提供凭据。值必须是数组，第一个元素是用户名，第二个是密码。第三个参数是认证类型（basic 或 digest）：

```
<?php

$client->request('GET', 'http://example.com', ['auth' => ['username
↪', 'password', 'digest']]);
```

body

对于支持正文的请求类型（如 PUT 或 POST），有两种设置正文的方式。第一种是使用 setBody() 方法：

```
<?php

$client->setBody($body)->request('PUT', 'http://example.com');
```

第二种是通过传递 body 选项。此方法为保持与 Guzzle API 兼容，功能与上例相同。值必须是字符串：

```
<?php

$client->request('PUT', 'http://example.com', ['body' => $body]);
```

cert

要指定 PEM 格式客户端证书的位置，可将完整文件路径作为字符串传递给 cert 选项。若需密码，可将值设为数组，第一个元素是证书路径，第二个是密码：

```
<?php

$client->request('GET', '/', ['cert' => ['/path/server.pem',
↪'password']]);
```

connect_timeout

默认情况下，CodeIgniter 不限制 cURL 连接网站的时间。可通过 `connect_timeout` 选项修改（单位：秒），0 表示无限等待：

```
<?php

$client->request('GET', 'http://example.com', ['connect_timeout' =>␣
↪0]);
```

cookie

指定 cURL 用于读取和保存 cookie 值的文件名。通过 `CURL_COOKIEJAR` 和 `CURL_COOKIEFILE` 选项实现：

```
<?php

$client->request('GET', 'http://example.com', ['cookie' =>␣
↪WRITEPATH . 'CookieSaver.txt']);
```

debug

当 `debug` 设为 `true` 时，将启用额外调试信息并输出到 STDERR。

这是通过设置 `CURLOPT_VERBOSE` 并回显输出来实现的。使用 `spark serve` 运行内置服务器时，输出将显示在控制台；否则写入服务器错误日志：

```
<?php

$client->request('GET', 'http://example.com', ['debug' => true]);
```

可将文件名作为 debug 的值来将输出写入文件：

```
<?php

$client->request('GET', 'http://example.com', ['debug' => '/usr/
↪local/curl_log.txt']);
```

delay

允许在发送请求前暂停指定毫秒数:

```
<?php

// Delay for 2 seconds
$client->request('GET', 'http://example.com', ['delay' => 2000]);
```

form_params

通过 form_params 选项发送 application/x-www-form-urlencoded POST 请求的关联数组。如果未设置，会将 Content-Type 标头设为 application/x-www-form-urlencoded:

```
<?php

$client->request('POST', '/post', [
    'form_params' => [
        'foo' => 'bar',
        'baz' => ['hi', 'there'],
    ],
]);
```

备注: form_params 不能与*multipart* 选项共用。对于 application/x-www-form-urlencoded 请求使用 form_params，对于 multipart/form-data 请求使用 multipart。

headers

虽然可以使用 setHeader() 方法设置请求需要的标头，也可以通过选项传递关联数组。每个键是标头名称，值是该标头字段值的字符串或字符串数组:

```
<?php
```

(续下页)

(接上页)

```
$client->request('GET', '/', [
    'headers' => [
        'User-Agent' => 'testing/1.0',
        'Accept'     => 'application/json',
        'X-Foo'      => ['Bar', 'Baz'],
    ],
]);
```

如果标头在构造函数中传递，它们将被视为默认值，会被后续标头数组或 `setHeader()` 调用覆盖。

http_errors

默认情况下，当返回的 HTTP 状态码大于等于 400 时，CURLRequest 会抛出 `HTTPException`。

若需查看响应正文，可将 `http_errors` 设为 `false` 以返回内容：

```
<?php

$client->request('GET', '/status/500');
// If the response code is 500, an HTTPException is thrown,
// and a detailed error report is displayed if in development mode.

$response = $client->request('GET', '/status/500', ['http_errors' =>
↪ false]);
echo $response->getStatusCode(); // 500
echo $response->getBody();       // You can see the response body.
```

json

`json` 选项用于轻松上传 JSON 编码数据作为请求正文。会添加 `application/json` 的 Content-Type 标头，覆盖已设置的任何 Content-Type。数据可以是 `json_encode()` 接受的任何值：

```
<?php
```

(续下页)

(接上页)

```
$response = $client->request('PUT', '/put', ['json' => ['foo' =>
↪'bar']]);
```

备注: 此选项不允许自定义 `json_encode()` 函数或 Content-Type 标头。如需此功能，需手动编码数据并通过 CURLRequest 的 `setBody()` 方法传递，同时使用 `setHeader()` 方法设置 Content-Type。

multipart

需要通过 POST 请求发送文件和其他数据时，可使用 `multipart` 选项和 CURLFile 类。

值应为要发送的 POST 数据关联数组。为安全起见，已禁用通过前缀 `@` 上传文件的旧方法。要发送的文件必须作为 CURLFile 实例传递：

```
<?php

$client->request('POST', '/post', [
    'multipart' => [
        'foo'      => 'bar',
        'userfile' => new \CURLFile('/path/to/file.txt'),
    ],
]);
```

备注: `multipart` 不能与*form_params* 选项共用。对于 `application/x-www-form-urlencoded` 请求使用 `form_params`，对于 `multipart/form-data` 请求使用 `multipart`。

proxy

在 4.4.0 版本加入.

可通过传递关联数组作为 `proxy` 选项来设置代理:

```
<?php

$client->request(
    'GET',
    'http://example.com',
    ['proxy' => 'http://localhost:3128'],
);
```

query

通过传递关联数组作为 `query` 选项来发送查询字符串参数:

```
<?php

// Send a GET request to /get?foo=bar
$client->request('GET', '/get', ['query' => ['foo' => 'bar']]);
```

timeout

默认情况下，cURL 函数执行没有时间限制。可通过 `timeout` 选项修改（单位：秒），0 表示无限等待:

```
<?php

$client->request('GET', 'http://example.com', ['timeout' => 5]);
```

user_agent

指定请求的 User Agent：

```
<?php

$client->request('GET', 'http://example.com', ['user_agent' =>
↪'CodeIgniter Framework v4']);
```

verify

此选项描述 SSL 证书验证行为。若 `verify` 为 `true`，启用 SSL 证书验证并使用操作系统提供的默认 CA 包。设为 `false` 将禁用验证（不安全，允许中间人攻击）。设为自定义证书路径可启用验证。默认值为 true：

```
<?php

// Use the system's CA bundle (this is the default setting)
$client->request('GET', '/', ['verify' => true]);

// Use a custom SSL certificate on disk.
$client->request('GET', '/', ['verify' => '/path/to/cert.pem']);

// Disable validation entirely. (Insecure!)
$client->request('GET', '/', ['verify' => false]);
```

force_ip_resolve

在 4.6.0 版本加入.

设置 HTTP 处理器使用 `v4` （仅 IPv4）或 `v6` （IPv6）协议：

```
<?php

// Force ipv4 resolve
$client->request('GET', '/', ['force_ip_resolve' => 'v4']); // v4␣
↪or v6
```

version

要设置使用的 HTTP 协议版本，可传递版本号字符串或浮点数（通常为 `1.0`、`1.1`，v4.3.0 起支持 `2.0`）：

```
<?php

// Force HTTP/1.0
$client->request('GET', '/', ['version' => 1.0]);
```

7.1.5 Email 类

CodeIgniter 强大的 Email 类支持以下功能:

- 多种协议:Mail、Sendmail 和 SMTP
- SMTP 的 TLS 和 SSL 加密
- 多个收件人
- 抄送和密送
- HTML 或纯文本电子邮件
- 附件
- 文字换行
- 优先级
- BCC 批量模式，可将大型邮件列表拆分为多个小的 BCC 批次。
- 电子邮件调试工具

- *使用 Email 库*
  - *发送电子邮件*
  - *设置电子邮件首选项*
    - *通过传递数组设置电子邮件首选项*
    - *在配置文件中设置电子邮件首选项*
    - *SMTP 协议的 SSL 与 TLS*

    * 检查首选项
  - 电子邮件首选项
  - 覆盖文字换行
- 类参考

使用 Email 库

发送电子邮件

发送电子邮件不仅很简单, 而且你可以即时配置或在 **app/Config/Email.php** 文件中设置首选项。

下面是一个基本示例, 演示了如何发送电子邮件:

```
<?php

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

设置电子邮件首选项

有 21 个不同的首选项可用于定制电子邮件消息的发送方式。你可以像这里描述的那样手动设置它们, 也可以通过存储在配置文件中的首选项自动设置, 如电子邮件首选项 中所述:

通过传递数组设置电子邮件首选项

首选项是通过向电子邮件初始化方法传递首选项值数组来设置的。下面是一个如何设置一些首选项的示例:

```
<?php

$config['protocol'] = 'sendmail';
$config['mailPath'] = '/usr/sbin/sendmail';
$config['charset']  = 'iso-8859-1';
$config['wordWrap'] = true;

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

备注: 如果你不设置它们, 大多数首选项都有默认值。

在配置文件中设置电子邮件首选项

如果你不喜欢使用上述方法设置首选项, 你可以将它们放入配置文件中。只需打开 **app/Config/Email.php** 文件, 并在电子邮件属性中设置你的配置。然后保存文件, 它将被自动使用。如果你在配置文件中设置了首选项, 将 **不需要**使用 `$email->initialize()` 方法。

SMTP 协议的 SSL 与 TLS

为了在与 SMTP 服务器通信时保护用户名、密码和电子邮件内容, 应该对通道使用加密。已经广泛部署了两种不同的标准, 在尝试排除电子邮件发送问题时, 了解这些差异很重要。

当提交电子邮件时, 大多数 SMTP 服务器允许在端口 465 或 587 上连接。(原始端口 25 很少使用, 因为许多 ISP 有屏蔽规则, 而且通信完全是明文的)。

关键差异在于端口 465 要求从一开始就使用 TLS 按照 RFC 8314 来保护通信通道。而端口 587 上的连接允许明文连接, 之后会使用 `STARTTLS` SMTP 命令升级通道以使用加密。

端口 465 上的连接是否支持升级可能由服务器决定, 所以如果服务器不允许, `STARTTLS`

SMTP 命令可能会失败。如果你将端口设置为 465, 你应该尝试设置 `SMTPCrypto` 为空字符串（`''`）, 因为通信从一开始就是用 TLS 保护的, 不需要 `STARTTLS`。

如果你的配置要求你连接到端口 587, 你最好将 `SMTPCrypto` 设置为 `tls`, 因为这将在与 SMTP 服务器通信时实现 `STARTTLS` 命令, 将明文通道切换为加密通道。初始通信将是明文的, 并使用 `STARTTLS` 命令将通道升级为 TLS。

检查首选项

成功发送的最后使用的设置可以从实例属性 `$archive` 获取。这对于测试和调试很有帮助, 以确定在 `send()` 调用时的实际值。

电子邮件首选项

以下是发送电子邮件时可以设置的所有首选项列表。

首选项	默认值	选项	描述
fromEmail			在“from”标头中设置的电子邮件地址。
fromName			在“from”标头中设置的名称。
userAgent	CodeIgniter		“user agent”。
protocol	mail	`mail`，`sendmail`，或 `smtp`	邮件发送协议。
mailPath	/usr/sbin/sendmail		Sendmail 的服务器路径。
SMTPHost			SMTP 服务器主机名。
SMTPUser			SMTP 用户名。
SMTPPass			SMTP 密码。
SMTPPort	25		SMTP 端口。（如果设置为 `465`，则无论 `SMTPCrypto` 设置如何，都会使用 TLS 进行连接。）
SMTPTimeout	5		SMTP 超时时间（以秒为单位）。
SMTPKeepAlive	false	`true/false`	启用持久的 SMTP 连接。
SMTPCrypto	tls	`tls`，`ssl`，或空字符串（`''`）	SMTP 加密。将此设置为 `ssl` 将使用 SSL 创建到服务器的安全通道，而 `tls` 将向服务器发出 `STARTTLS` 命令。在端口 `465` 上连接应将此设置为空字符串 (`''`)。另请参见*SMTP 协议的 SSL 与 TLS*。
wordWrap	true	`true/false`	启用自动换行。
wrapChars	76		换行的字符数。
mailType	text	`text` 或 `html`	邮件类型。如果发送 HTML 邮件，必须将其作为完整的网页发送。确保没有任何相对链接或相对图像路径，否则它们将无法工作。
charset	UTF-8		字符集 (`utf-8`，`iso-8859-1` 等)。
validate	true	`true/false`	是否验证电子邮件地址。

覆盖文字换行

如果你启用了文字换行 (遵循 RFC 822 的推荐) 并且电子邮件中有一个非常长的链接, 该链接也可能被换行, 导致收件人无法点击。CodeIgniter 允许你在消息的一部分手动覆盖文字换行, 如下所示:

```
换行显示正常的电子邮件文本。

{unwrap}http://example.com/a_long_link_that_should_not_be_wrapped.
↪html{/unwrap}

更多正常显示换行的文本。
```

将你不想换行的项放在: {unwrap} {/unwrap} 之间。

类参考

class CodeIgniter\Email**Email**

setFrom(*$from*[, *$name* = ''[, *$returnPath* = *null*]])

参数

- **$from** (string) – "From" 电子邮件地址
- **$name** (string) – "From" 显示名称
- **$returnPath** (string) – 可选的电子邮件地址，用于重定向未送达的邮件

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

设置发送电子邮件的人的电子邮件地址和名称:

```
<?php

$email->setFrom('you@example.com', 'Your Name');
```

你还可以设置 Return-Path，以帮助重定向未送达的邮件:

```
<?php

$email->setFrom('you@example.com', 'Your Name', 'returned_
↪emails@example.com');
```

备注： 如果你将协议配置为‘smtp’，则不能使用 Return-Path。

setReplyTo(*$replyto*[, *$name* = ''])

参数

- **$replyto** (string) –回复的电子邮件地址
- **$name** (string) –回复电子邮件地址的显示名称

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

设置回复地址。如果未提供信息，则使用 *setFrom* 方法中的信息。示例：

```
<?php

$email->setReplyTo('you@example.com', 'Your Name');
```

setTo(*$to*)

参数

- **$to** (mixed) –逗号分隔的字符串或电子邮件地址数组

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

设置收件人的电子邮件地址。可以是单个电子邮件、逗号分隔的列表或数组：

```
<?php

$email->setTo('someone@example.com');
```

```
<?php

$email->setTo('one@example.com, two@example.com,␣
↪three@example.com');
```

```
<?php

$email->setTo(['one@example.com', 'two@example.com',
↪'three@example.com']);
```

setCC(*$cc*)

参数

- **$cc** (mixed) –逗号分隔的字符串或电子邮件地址数组

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

设置 CC 电子邮件地址。就像 “to” 一样，可以是单个电子邮件、逗号分隔的列表或数组。

setBCC(*$bcc*[, *$limit* = ''])

参数

- **$bcc** (mixed) –逗号分隔的字符串或电子邮件地址数组
- **$limit** (int) –每批发送的最大电子邮件数量

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

设置 BCC 电子邮件地址。就像 `setTo()` 方法一样，可以是单个电子邮件、逗号分隔的列表或数组。

如果设置了 `$limit`，将启用 "批处理模式"，这将按批次发送电子邮件，每批次不超过指定的 `$limit`。

setSubject(*$subject*)

参数

- **$subject** (string) –电子邮件主题

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

设置电子邮件主题：

```
<?php

$email->setSubject('This is my subject');
```

setMessage(*$body*)

参数

- **$body** (string) –电子邮件正文

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

设置电子邮件正文：

```
<?php

$email->setMessage('This is my message');
```

setAltMessage(*$str*)

参数

- **$str** (string) –替代电子邮件正文

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

设置替代电子邮件正文：

```
<?php

$email->setAltMessage('This is the alternative message');
```

这是一个可选的消息字符串，可以在你发送 HTML 格式的电子邮件时使用。它允许你指定一个没有 HTML 格式的替代消息，该消息将添加到头字符串中，以便那些不接受 HTML 电子邮件的人使用。如果你没有设置自己的消息，CodeIgniter 将从你的 HTML 电子邮件中提取消息并去除标签。

setHeader(*$header*, *$value*)

参数

- **$header** (string) –头名称
- **$value** (string) –头值

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

向电子邮件添加附加头：

```
<?php

$email->setHeader('Header1', 'Value1');
$email->setHeader('Header2', 'Value2');
```

clear(*$clearAttachments = false*)

参数

- **$clearAttachments** (bool) –是否清除附件

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

将所有电子邮件变量初始化为空状态。此方法旨在用于在循环中运行电子邮件发送方法，允许在循环之间重置数据。

```
<?php

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();
}
```

如果将参数设置为 true，任何附件也将被清除：

```
<?php

$email->clear(true);
```

send(*$autoClear = true*)

参数

- **$autoClear** (bool) –是否自动清除消息数据

返回

成功时返回 true，失败时返回 false

返回类型

bool

电子邮件发送方法。根据成功或失败返回布尔值 true 或 false，使其可以有条件地使用：

```
<?php
```

(续下页)

(接上页)

```
if (! $email->send()) {
    // Generate error
}
```

如果请求成功，此方法将自动清除所有参数。要停止此行为，请传递 false：

```
<?php

if ($email->send(false)) {
    // Parameters won't be cleared
}
```

备注： 为了使用 `printDebugger()` 方法，你需要避免清除电子邮件参数。

备注： 如果启用了 `BCCBatchMode`，并且收件人超过 `BCCBatchSize`，此方法将始终返回布尔值 `true`。

---

attach(*$filename*[, *$disposition* = ''[, *$newname* = *null*[, *$mime* = '']]])

参数

- **$filename** (string) –文件名
- **$disposition** (string) –附件的 'disposition'。大多数电子邮件客户端会根据此处使用的 MIME 规范自行决定。https://www.iana.org/assignments/cont-disp/cont-disp.xhtml
- **$newname** (string) –在电子邮件中使用的自定义文件名
- **$mime** (string) –要使用的 MIME 类型（对缓冲数据有用）

返回

CodeIgniter\Email\Email 实例（方法链）

返回类型

CodeIgniter\Email\Email

允许你发送附件。将文件路径/名称放在第一个参数中。对于多个附件，请多次使用该方法。例如：

```
<?php

$email->attach('/path/to/photo1.jpg');
$email->attach('/path/to/photo2.jpg');
$email->attach('/path/to/photo3.jpg');
```

要使用默认的 disposition（附件），请将第二个参数留空，否则使用自定义 disposition：

```
<?php

$email->attach('image.jpg', 'inline');
```

你还可以使用 URL：

```
<?php

$email->attach('http://example.com/filename.pdf');
```

如果你想使用自定义文件名，可以使用第三个参数：

```
<?php

$email->attach('filename.pdf', 'attachment', 'report.pdf');
```

如果你需要使用缓冲字符串而不是实际的物理文件，可以将第一个参数用作缓冲区，第三个参数用作文件名，第四个参数用作 MIME 类型：

```
<?php

$email->attach($buffer, 'attachment', 'report.pdf',
↪'application/pdf');
```

setAttachmentCID(*$filename*)

参数

- **$filename** (string) –已存在的附件文件名

返回

附件内容 ID 或未找到时返回 false

返回类型

string

设置并返回附件的内容 ID，这使你能够将内嵌（图片）附件嵌入 HTML 中。第一个参数必须是已附加的文件名。

```
<?php

$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();
}
```

备注： 每封电子邮件的内容 ID 必须重新创建以确保其唯一性。

printDebugger(*$include = ['headers', 'subject', 'body']*)

参数

- **$include** (array) –要打印的消息部分

返回

格式化的调试数据

返回类型

string

返回包含任何服务器消息、电子邮件头和电子邮件消息的字符串。对调试很有用。

你可以选择性地指定应打印消息的哪些部分。有效选项是：**headers**、**subject**、**body**。

示例：

```
<?php

// 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']);
```

备注: 默认情况下，将打印所有原始数据。

7.1.6 加密服务

重要: 不要使用这个或任何其他 *加密库来存储密码!* 密码必须是 *哈希过的*, 你应该通过 PHP 的 Password Hashing 扩展 来完成。

Encryption 服务提供对称 (密钥) 数据的双向加密。正如下面所解释的, 该服务将实例化和/或初始化一个加密 **处理程序**以适应你的参数。

Encryption 服务处理程序必须实现 CodeIgniter 的简单 `EncrypterInterface`。使用适当的 PHP 加密扩展或第三方库可能需要在你的服务器上安装额外的软件和/或需要在你的 PHP 实例中明确启用。

目前支持以下 PHP 扩展:

- OpenSSL
- Sodium

这不是一个完整的加密解决方案。例如, 如果你需要更多功能, 比如公钥加密, 我们建议你考虑直接使用 OpenSSL 或其他 Cryptography Extensions。像 Halite 这样更全面的包 (在 libsodium 上构建的面向对象包) 也是一个选择。

备注: 对 `MCrypt` 扩展的支持已经放弃, 因为从 PHP 7.2 开始它已被废弃。

- 使用 *Encryption* 库
  - 配置库
    - 用于与 *CI3* 保持兼容性的配置
    - 支持的 *HMAC* 认证算法
  - 默认行为
  - 设置加密密钥
    - 编码密钥或结果
    - 使用前缀存储密钥
  - 填充
  - 加密处理程序注意事项
    - *OpenSSL* 说明
    - *Sodium* 说明
  - 消息长度
  - 直接使用加密服务
- 类参考

使用 Encryption 库

与 CodeIgniter 中的所有服务一样, 它可以通过 `Config\Services` 加载:

```
<?php

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

假设你已经设置了起始密钥 (参见配置库), 加密和解密数据很简单 - 只需将适当的字符串传递给 `encrypt()` 和/或 `decrypt()` 方法:

```
<?php

$plainText  = 'This is a plain-text message!';
$ciphertext = $encrypter->encrypt($plainText);

// Outputs: This is a plain-text message!
echo $encrypter->decrypt($ciphertext);
```

就是这样!Encryption 库将自动完成整个过程中所有必要的加密安全性。你不需要担心它。

配置库

上面的示例使用了在 **app/Config/Encryption.php** 中找到的配置设置。

选项	可能的值 (默认值在括号中)
key	加密密钥起始值
driver	首选处理程序, 例如 OpenSSL 或 Sodium (`OpenSSL`)
digest	消息摘要算法 (`SHA512`)
blockSize	[仅 **SodiumHandler**] 填充长度, 以字节为单位 (`16`)
cipher	[仅 **OpenSSLHandler**] 要使用的密码 (`AES-256-CTR`)
encryptKeyInfo	[仅 **OpenSSLHandler**] 加密密钥信息 (`''`)
authKeyInfo	[仅 **OpenSSLHandler**] 认证密钥信息 (`''`)
rawData	[仅 **OpenSSLHandler**] 密文是否应为原始数据 (`true`)

你可以通过向 `Services` 调用传入自己的配置对象来替换配置文件中的设置。`$config` 变量必须是 `Config\Encryption` 类的一个实例。

```
<?php

use Config\Encryption;

$config          = config(Encryption::class);
$config->key     = 'aBigsecret_ofAtleast32Characters';
$config->driver = 'OpenSSL';

$encrypter = service('encrypter', $config);
```

用于与 CI3 保持兼容性的配置

在 4.3.0 版本加入.

从 v4.3.0 开始, 你可以解密用 CI3 加密的数据。如果你需要解密这样的数据, 请使用以下设置来保持兼容性。

```
<?php

use Config\Encryption;

$config          = new Encryption();
$config->driver = 'OpenSSL';

// Your CI3's 'encryption_key'
$config->key = hex2bin(
↪'64c70b0b8d45b80b9eba60b8b3c8a34d0193223d20fea46f8644b848bf7ce67f
↪');
// Your CI3's 'cipher' and 'mode'
$config->cipher = 'AES-128-CBC';

$config->rawData         = false;
$config->encryptKeyInfo = 'encryption';
$config->authKeyInfo     = 'authentication';

$encrypter = service('encrypter', $config);
```

支持的 HMAC 认证算法

对于 HMAC 消息认证,Encryption 库支持使用 SHA-2 系列算法:

算法	原始长度 (字节)	十六进制编码长度 (字节)
SHA512	64	128
SHA384	48	96
SHA256	32	64
SHA224	28	56

不包括其他流行算法, 例如 MD5 或 SHA1 的原因是它们不再被认为足够安全, 因此

我们不想鼓励它们的使用。如果你绝对需要使用它们, 可以轻松地通过 PHP 的原生 hash_hmac() 函数来实现。

当然, 更强大的算法在未来随着它们出现和广泛可用时也会加入。

默认行为

默认情况下,Encryption 库使用 OpenSSL 处理程序。该处理程序使用 AES-256-CTR 算法、你配置的 *key* 和 SHA512 HMAC 认证来加密。

设置加密密钥

你的加密密钥的长度 **必须**与正在使用的加密算法所允许的一样长。对于 AES-256 来说, 那是 32 字节 (字符)。

该密钥应该尽可能随机, 而且它 **不能**是一个常规的文本字符串, 也不能是散列函数的输出等。要创建一个适当的密钥, 你可以使用 Encryption 库的 `createKey()` 方法。

```
<?php

// $key will be assigned a 32-byte (256-bit) random key
$key = \CodeIgniter\Encryption\Encryption::createKey();

// for the SodiumHandler, you can use either:
$key = sodium_crypto_secretbox_keygen();
$key = \CodeIgniter\Encryption\Encryption::createKey(SODIUM_CRYPTO_
↪SECRETBOX_KEYBYTES);
```

可以将密钥存储在 **app/Config/Encryption.php** 中, 或者你可以设计自己的存储机制, 并在加密/解密时动态传递密钥。

要将你的密钥保存到 **app/Config/Encryption.php** 中, 打开该文件并设置:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
```

(续下页)

(接上页)

```
class Encryption extends BaseConfig
{
    public $key = 'YOUR KEY';

    // ...
}
```

编码密钥或结果

你会注意到 `createKey()` 方法输出二进制数据, 这很难处理 (即, 复制粘贴可能会损坏它), 所以你可以使用 `bin2hex()` 或 `base64_encode` 以更友好的方式处理密钥。例如:

```
<?php

// Get a hex-encoded representation of the key:
$encoded = bin2hex(\CodeIgniter\Encryption\
↪Encryption::createKey(32));

// Put the same value with hex2bin(),
// so that it is still passed as binary to the library:
$key = hex2bin('your-hex-encoded-key');
```

对加密结果, 你可能也会发现同样的技术很有用:

```
<?php

// Encrypt some text & make the results text
$encoded = base64_encode($encrypter->encrypt($plaintext));
```

使用前缀存储密钥

在存储加密密钥时, 你可以利用两个特殊前缀:`hex2bin:` 和 `base64:`。当这些前缀紧接在密钥值之前时, `Encryption` 将智能解析密钥, 并仍然将二进制字符串传递给库。

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Encryption extends BaseConfig
{
    // In Encryption, you may use
    public $key = 'hex2bin:<your-hex-encoded-key>';
    // or
    public $key = 'base64:<your-base64-encoded-key>';
    // ...
}
```

类似地, 你也可以在 **.env** 文件中使用这些前缀!

```
// 对于 hex2bin
encryption.key = hex2bin:<your-hex-encoded-key>

// 或者
encryption.key = base64:<your-base64-encoded-key>
```

填充

有时, 消息的长度可能会提供大量关于其性质的信息。如果消息之一是 “yes”、“no” 和 “maybe” , 则加密消息无济于事: 知道长度就足够知道消息是什么。

填充是一种通过使长度成为给定块大小的倍数来缓解这种情况的技术。

填充是在 `SodiumHandler` 中使用 libsodium 的原生 `sodium_pad` 和 `sodium_unpad` 函数实现的。这需要在加密之前向纯文本消息添加填充长度 (以字节为单位), 并在解密后删除它。填充通过 `Config\Encryption` 的 `$blockSize` 属性进行配置。该值应大于零。

重要: 建议你不要设计自己的填充实现。你必须始终使用库的更安全实现。另外, 密码不应进行填充。使用填充来隐藏密码的长度是不推荐的。愿意向服务器发送密码的客户端应该对其进行散列 (即使只对散列函数进行一次迭代)。这可以确保传输数据的长度是常量的, 并且服务器不会轻易获得密码的副本。

加密处理程序注意事项

OpenSSL 说明

OpenSSL 扩展在 PHP 中已经存在很长时间了。

CodeIgniter 的 OpenSSL 处理程序使用 AES-256-CTR 密码。

配置中提供的 *key* 用于生成另外两个密钥, 一个用于加密, 一个用于认证。这是通过称为 HMAC 键导出函数 (HKDF) 的技术实现的。

Sodium 说明

Sodium 扩展默认打包在 PHP 7.2.0 及更高版本中。

Sodium 在端到端方案中发送秘密消息时使用 XSalsa20 加密、Poly1305 进行 MAC 和 XS25519 进行密钥交换。要使用共享密钥 (例如对称加密) 对字符串进行加密和/或认证,Sodium 使用 XSalsa20 算法进行加密和 HMAC-SHA512 进行认证。

备注: CodeIgniter 的 `SodiumHandler` 在每次加密或解密会话中都使用 `sodium_memzero`。在每个会话之后, 无论是纯文本还是密文, 启动密钥都会从缓冲区中清除。在启动新会话之前, 你可能需要再次提供密钥。

消息长度

加密后的字符串通常比原始的纯文本字符串长 (取决于密码)。

这受密码算法本身的影响, 前缀为密文的初始化向量 (IV), 以及也前缀的 HMAC 认证消息。此外, 加密消息也进行了 Base64 编码, 以便无论使用的字符集如何, 都可以安全地存储和传输。

在选择数据存储机制时, 请记住这一信息。例如,Cookie 只能容纳 4K 的信息。

直接使用加密服务

除了如*使用 Encryption 库* 所述通过 `Services` 使用之外, 你还可以直接创建一个 "Encrypter" , 或更改现有实例的设置。

```
<?php

// create an Encryption instance
$encryption = new \CodeIgniter\Encryption\Encryption();

// reconfigure an instance with different settings
$encrypter = $encryption->initialize($config);
```

记住, `$config` 必须是 `Config\Encryption` 类的一个实例。

类参考

class `CodeIgniter\Encryption\`**`Encryption`**

static **`createKey`**([*$length = 32*])

参数

- **`$length`** (`int`) –输出长度

返回

具有指定长度的伪随机加密密钥, 如果失败则为 `false`

返回类型

`string`

通过从操作系统的来源 (即 `/dev/urandom`) 获取随机数据来创建加密密钥。

`initialize`([*Encryption $config = null*])

参数

- **`$config`** (`Config\Encryption`) –配置参数

返回

`CodeIgniter\Encryption\EncrypterInterface` 实例

返回类型

`CodeIgniter\Encryption\EncrypterInterface`

Throws

`CodeIgniter\Encryption\Exceptions\EncryptionException`

使用不同的设置初始化 (配置) 库。

示例:

```
<?php

$encrypter = $encryption->initialize(['cipher' => 'AES-256-
↪CTR']);
```

详细信息请参阅配置库 部分。

interface `CodeIgniter\Encryption\`**`CodeIgniter\Encryption\EncrypterInterface`**

encrypt(*$data*[, *$params = null*])

参数

- **$data** (`string`) –要加密的数据
- **$params** (`array|string|null`) –配置参数 (密钥)

返回

加密数据

返回类型

`string`

Throws

`CodeIgniter\Encryption\Exceptions\EncryptionException`

加密输入数据并返回其密文。

如果你在第二个参数中传入参数, 则如果 `$params` 为数组, `key` 元素将用作此操作的起始密钥; 或者可以作为字符串传入起始密钥。

如果你使用 SodiumHandler 并希望在运行时传递不同的 `blockSize`, 请在 `$params` 数组中传递 `blockSize` 键。

示例:

```
<?php

$ciphertext = $encrypter->encrypt('My secret message');
$ciphertext = $encrypter->encrypt('My secret message', ['key
↪' => 'New secret key']);
$ciphertext = $encrypter->encrypt('My secret message', ['key
↪' => 'New secret key', 'blockSize' => 32]);
$ciphertext = $encrypter->encrypt('My secret message', 'New␣
↪secret key');
$ciphertext = $encrypter->encrypt('My secret message', [
↪'blockSize' => 32]);
```

decrypt (*$data*[, *$params = null*])

参数

- **$data** (string) –要解密的数据
- **$params** (array|string|null) –配置参数 (密钥)

返回

解密数据

返回类型

string

Throws

CodeIgniter\Encryption\Exceptions\EncryptionException

解密输入数据并返回纯文本。

如果你在第二个参数中传入参数, 则如果 $params 为数组, key 元素将用作此操作的起始密钥; 或者可以作为字符串传入起始密钥。

如果你使用 SodiumHandler 并希望在运行时传递不同的 blockSize, 请在 $params 数组中传递 blockSize 键。

示例:

```
<?php
```

(续下页)

(接上页)

```
echo $encrypter->decrypt($ciphertext);
echo $encrypter->decrypt($ciphertext, ['key' => 'New secret␣
↪key']);
echo $encrypter->decrypt($ciphertext, ['key' => 'New secret␣
↪key', 'blockSize' => 32]);
echo $encrypter->decrypt($ciphertext, 'New secret key');
echo $encrypter->decrypt($ciphertext, ['blockSize' => 32]);
```

7.1.7 文件处理

CodeIgniter 提供了一个 File 类，它封装了 SplFileInfo 类并添加了一些便捷方法。该类是上传文件 和图像 的基类。

- *获取 File 实例*
- *利用 Spl 特性*
- *新增功能*
  - *getRandomName()*
  - *getSize()*
  - *getSizeByUnit()*
  - *getSizeByBinaryUnit()*
  - *getSizeByMetricUnit()*
  - *getMimeType()*
  - *guessExtension()*
  - *移动文件*

获取 File 实例

你可以通过将文件路径传入构造函数来创建新的 File 实例。

```
$file = new \CodeIgniter\Files\File($path);
```

默认情况下，文件不需要存在。但是你可以传递额外的 `true` 参数来检查文件是否存在，如果不存在则会抛出 `FileNotFoundException()`。

利用 Spl 特性

一旦获得实例，你就可以使用 SplFileInfo 类的全部功能，包括：

```
// Get the file's basename
echo $file->getBasename();
// Get last modified time
echo $file->getMTime();
// Get the true real path
echo $file->getRealPath();
// Get the file permissions
echo $file->getPerms();

// Write CSV rows to it.
if ($file->isWritable()) {
    $csv = $file->openFile('w');

    foreach ($rows as $row) {
        $csv->fputcsv($row);
    }
}
```

新增功能

除了 SplFileInfo 类的所有方法外，还提供了一些新工具。

getRandomName()

你可以使用 `getRandomName()` 方法生成带有当前时间戳前缀的加密安全随机文件名。这在移动文件时重命名文件特别有用，可以使文件名不可猜测：

```
// Generates something like: 1465965676_385e33f741.jpg
$newName = $file->getRandomName();
```

getSize()

返回文件的大小（以字节为单位）：

```
$size = $file->getSize(); // 256901
```

如果文件不存在或发生错误，将抛出 `RuntimeException`。

getSizeByUnit()

自 4.6.0 版本弃用.

默认返回文件大小（以字节为单位）。你可以传入 `'kb'` 或 `'mb'` 作为第一个参数，分别以千字节或兆字节为单位获取结果：

```
$bytes     = $file->getSizeByUnit(); // 256901
$kilobytes = $file->getSizeByUnit('kb'); // 250.880
$megabytes = $file->getSizeByUnit('mb'); // 0.245
```

如果文件不存在或发生错误，将抛出 `RuntimeException`。

getSizeByBinaryUnit()

在 4.6.0 版本加入.

默认返回文件大小（以字节为单位）。你可以传入不同的 FileSizeUnit 值作为第一个参数，分别以 KiB、MiB 等单位获取结果。可以通过第二个参数传入精度值来定义小数位数。

```
$bytes     = $file->getSizeByBinaryUnit(); // 256901
$kibibytes = $file->getSizeByBinaryUnit(FileSizeUnit::KB); // 250.
```

(续下页)

(接上页)

```
↪880
$mebibytes = $file->getSizeByBinaryUnit(FileSizeUnit::MB); // 0.245
```

如果文件不存在或发生错误，将抛出 `RuntimeException`。

getSizeByMetricUnit()

在 4.6.0 版本加入.

默认返回文件大小（以字节为单位）。你可以传入不同的 FileSizeUnit 值作为第一个参数，分别以 KB、MB 等单位获取结果。可以通过第二个参数传入精度值来定义小数位数。

```
$bytes     = $file->getSizeByMetricUnit(); // 256901
$kilobytes = $file->getSizeByMetricUnit(FileSizeUnit::KB); // 256.
↪901
$megabytes = $file->getSizeByMetricUnit(FileSizeUnit::MB); // 0.256
```

如果文件不存在或发生错误，将抛出 `RuntimeException`。

getMimeType()

检索文件的媒体类型（MIME 类型）。使用尽可能安全的方法来确定文件类型：

```
$type = $file->getMimeType();

echo $type; // image/png
```

guessExtension()

尝试基于可信的 `getMimeType()` 方法确定文件扩展名。如果 MIME 类型未知，则返回 null。这通常比仅使用文件名提供的扩展名更可靠。使用 **app/Config/Mimes.php** 中的值来确定扩展名：

```
// Returns 'jpg' (WITHOUT the period)
$ext = $file->guessExtension();
```

移动文件

每个文件都可以使用 `move()` 方法移动到新位置。第一个参数指定目标目录:

```
$file->move(WRITEPATH . 'uploads');
```

默认使用原始文件名。你可以通过第二个参数指定新文件名:

```
$newName = $file->getRandomName();
$file->move(WRITEPATH . 'uploads', $newName);
```

`move()` 方法会返回移动后文件的新 File 实例，因此如果需要使用新位置，必须捕获结果:

```
$file = $file->move(WRITEPATH . 'uploads');
```

7.1.8 文件集合

处理文件组可能很繁琐，因此框架提供了 `FileCollection` 类来简化跨文件系统的文件组定位和操作。

- 基本用法
- 创建集合
  - *__construct(string[] $files = [])*
  - *define()*
  - *set(array $files)*
- 输入文件
  - *add(string[]\string $paths, bool $recursive = true)*
  - *addFile(string $file) / addFiles(array $files)*
  - *removeFile(string $file) / removeFiles(array $files)*
  - *addDirectory(string $directory, bool $recursive = false)*
  - *addDirectories(array $directories, bool $recursive = false)*
- 过滤文件

  - *removePattern(string $pattern, string $scope = null)*
  - *retainPattern(string $pattern, string $scope = null)*
  - *retainMultiplePatterns(array $pattern, string $scope = null)*
- 检索文件
  - *get(): string[]*

基本用法

在最基础的用法中，`FileCollection` 是一个由你设置或构建的文件数组：

```
<?php

use CodeIgniter\Files\FileCollection;

$files = new FileCollection([
    FCPATH . 'index.php',
    ROOTPATH . 'spark',
]);
$files->addDirectory(APPPATH . 'Filters');
```

在你输入要处理的文件后，可以移除文件或使用过滤命令来删除/保留匹配特定正则表达式或 glob 风格模式的文件：

```
<?php

$files->removeFile(APPPATH . 'Filters/DevelopToolbar.php');

$files->removePattern('#\.gitkeep#');
$files->retainPattern('*.php');
```

当集合构建完成后，可以使用 `get()` 获取最终文件路径列表，或利用 `FileCollection` 的可计数和可迭代特性直接操作每个 `File`：

```
<?php

echo 'My files: ' . implode(PHP_EOL, $files->get());
```

(续下页)

(接上页)

```
echo 'I have ' . count($files) . ' files!';

foreach ($files as $file) {
    echo 'Moving ' . $file->getBasename() . ', ' . $file->
↪getSizeByUnit('mb');
    $file->move(WRITABLE . $file->getRandomName());
}
```

以下是操作 `FileCollection` 的具体方法。

创建集合

__construct(string[] $files = [])

构造函数接受一个可选的文件路径数组作为初始集合。这些路径会传递给 `add()` 方法，因此子类通过 `$files` 提供的文件会保留。

define()

允许子类定义自己的初始文件。此方法由构造函数调用，支持预定义集合而无需重复调用其方法。

示例：

```
<?php

namespace App;

use CodeIgniter\Files\FileCollection;

class ConfigCollection extends FileCollection
{
    protected function define(): void
    {
        $this->add(APPPATH . 'Config', true)->retainPattern('*.php
↪');
```

(续下页)

(接上页)

```
    }
}
```

现在你可以在项目任意位置使用 `ConfigCollection` 来访问 **app/Config/** 下的所有 PHP 文件，无需每次重新调用集合方法。

set(array $files)

将输入文件列表设置为提供的文件路径字符串数组。这会移除集合中所有现有文件，因此 `$collection->set([])` 本质上是硬重置。

输入文件

add(string[]|string $paths, bool $recursive = true)

添加路径或路径数组指示的所有文件。如果路径解析为目录，则 `$recursive` 会包含子目录。

addFile(string $file) / addFiles(array $files)

将单个文件或多个文件添加到当前输入文件列表。文件必须是实际文件的绝对路径。

removeFile(string $file) / removeFiles(array $files)

从当前输入文件列表中移除单个文件或多个文件。

addDirectory(string $directory, bool $recursive = false)

addDirectories(array $directories, bool $recursive = false)

添加目录或多个目录中的所有文件，可选是否递归子目录。目录必须是实际目录的绝对路径。

过滤文件

removePattern(string $pattern, string $scope = null)

retainPattern(string $pattern, string $scope = null)

通过模式（和可选作用域）过滤当前文件列表，移除或保留匹配文件。

`$pattern` 可以是完整正则表达式（如 `'#\A[A-Za-z]+\.php\z#'`）或类似 `glob()` 的伪正则表达式（如 `'*.css'`）。

如果提供 `$scope`，则只有该目录下或其子目录中的文件会被考虑（即 `$scope` 外的文件始终保留）。未提供作用域时，所有文件都会参与过滤。

示例：

```
<?php

use CodeIgniter\Files\FileCollection;

$files = new FileCollection();
$files->add(APPPATH . 'Config', true); // Adds all Config files and␣
↪directories

$files->removePattern('*tion.php'); // Would remove Encryption.php,␣
↪Validation.php, and boot/production.php
$files->removePattern('*tion.php', APPPATH . 'Config/boot'); //␣
↪Would only remove boot/production.php

$files->retainPattern('#A.+php$#'); // Would keep only Autoload.php
$files->retainPattern('#d.+php$#', APPPATH . 'Config/boot'); //␣
↪Would keep everything but boot/production.php and boot/testing.php
```

retainMultiplePatterns(array $pattern, string $scope = null)

提供与 `retainPattern()` 相同的功能，但接受模式数组来保留匹配所有模式的文件。

示例：

```
<?php

$files->retainMultiplePatterns(['*.css', '*.js']); // Would keep␣
↪only *.css and *.js files
```

检索文件

get(): string[]

返回所有已加载输入文件的数组。

备注： `FileCollection` 实现了 `IteratorAggregate` 接口，因此可以直接操作（例如 `foreach ($collection as $file)`）。

7.1.9 蜜罐类

如果在 **app/Config/Filters.php** 文件中启用蜜罐，蜜罐类可以确定何时机器人向 CodeIgniter4 应用程序发出请求。这是通过将表单字段附加到任何表单上完成的，这个表单字段对人类隐藏但对机器人可访问。当数据输入字段时，假定请求来自机器人，你可以抛出一个 `HoneypotException`。

- 启用蜜罐
- 自定义蜜罐

启用蜜罐

要启用蜜罐, 需要对 **app/Config/Filters.php** 进行更改。只需从 `$globals` 数组中取消注释 honeypot, 如:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    // ...

    public $globals = [
        'before' => [
            'honeypot',
            // 'csrf',
            // 'invalidchars',
        ],
        'after' => [
            'toolbar',
            'honeypot',
            // 'secureheaders',
        ],
    ];

    // ...
}
```

附带了一个示例蜜罐过滤器，位于 **system/Filters/Honeypot.php**。如果它不合适，你可以在 **app/Filters/Honeypot.php** 创建自己的过滤器，并相应地修改 **app/Config/Filters.php** 中的 `$aliases`。

自定义蜜罐

可以自定义蜜罐。以下字段可以在 **app/Config/Honeypot.php** 或 **.env** 中设置。

- `$hidden` - `true` 或 `false` 来控制蜜罐字段的可见性; 默认为 `true`
- `$label` - 蜜罐字段的 HTML 标签, 默认为 `'Fill This Field'`
- `$name` - 用于模板的 HTML 表单字段的名称; 默认为 `'honeypot'`
- `$template` - 用于蜜罐的表单字段模板; 默认为 `'<label>{label}</label><input type="text" name="{name}" value="">'`
- `$container` - 模板的容器标签; 默认为 `'<div style="display:none">{template}</div>'`。如果你启用了 CSP, 可以删除 `style="display:none"`。
- `$containerId` - [v4.3.0 新增] 此设置仅在启用 CSP 时使用。你可以更改容器标签的 id 属性; 默认为 `'hpc'`

7.1.10 图像处理类

CodeIgniter 的图像处理类允许你执行以下操作:

- 图像缩放
- 缩略图创建
- 图像裁剪
- 图像旋转
- 图像添加水印

支持以下图像库:GD/GD2 和 ImageMagick。

- 初始化类
- 处理图像
  - 图像质量
- 处理方法
  - 裁剪图像
  - 转换图像

- 调整图像大小
- 扁平化图像
- 翻转图像
- 调整图像大小
- 旋转图像
- 添加文本水印

初始化类

与 CodeIgniter 中的其他大多数类一样, 在控制器中通过调用全局函数 `service()` 来初始化图像类:

```
<?php

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

你可以将你希望使用的图像库的别名传递给全局函数 `service()`:

```
<?php

$image = service('image', 'imagick');
```

可用的处理程序如下:

- `gd` GD/GD2 图像库
- `imagick` ImageMagick 库。

如果使用 ImageMagick 库, 必须在 **app/Config/Images.php** 中设置库的路径。

备注: ImageMagick 处理程序需要 imagick 扩展。

处理图像

无论你想执行的处理类型 (调整大小、裁剪、旋转或添加水印), 过程都是相同的。你将设置与你计划执行的操作相对应的一些首选项, 然后调用可用的处理函数之一。

例如, 要创建图像缩略图, 你将执行以下操作:

```
<?php

$image->withFile('/path/to/image/mypic.jpg')
    ->fit(100, 100, 'center')
    ->save('/path/to/image/mypic_thumb.jpg');
```

上面的代码告诉库去寻找一个名为 **mypic.jpg** 的图像，该图像位于 **/path/to/image** 文件夹中，然后从中创建一个 100 x 100 像素的新图像，并将其保存到一个新文件 **mypic_thumb.jpg** 中。由于它使用了 `fit()` 方法，它将尝试根据所需的纵横比找到图像的最佳部分进行裁剪，然后裁剪并调整结果的大小。

在保存之前, 可以通过尽可能多的可用方法对图像进行处理。原始图像保持不变, 使用新的图像并通过每个方法传递, 在之前的结果上叠加结果:

```
<?php

$image->withFile('/path/to/image/mypic.jpg')
    ->reorient()
    ->rotate(90)
    ->crop(100, 100, 0, 0)
    ->save('/path/to/image/mypic_thumb.jpg');
```

这个示例首先会修复任何移动手机方向问题, 然后将图像旋转 90 度, 然后从左上角开始裁剪结果为 100 x 100 像素的图像。结果将保存为缩略图。

备注: 为了允许图像类执行任何处理, 包含图像文件的文件夹必须具有写入权限。

备注: 对于某些操作, 图像处理可能需要相当多的服务器内存。如果在处理图像时遇到内存溢出错误, 则可能需要限制它们的最大大小和/或调整 PHP 内存限制。

图像质量

`save()` 可以接受额外的参数 `$quality` 来更改结果图像的质量。值的范围从 0 到 100, 默认值为 90。此参数仅适用于 JPEG 和 WebP 图像, 否则将被忽略:

备注: 自 v4.4.0 起，WebP 格式可以使用 `$quality` 参数。

```
<?php

$image->withFile('/path/to/image/mypic.jpg')
    // processing methods
    ->save('/path/to/image/my_low_quality_pic.jpg', 10);
```

备注: 更高的质量会导致文件大小更大。另请参阅 https://www.php.net/manual/en/function.imagejpeg.php

如果你只对更改图像质量而不做任何处理感兴趣。你需要包含图像资源, 否则最终会得到一个完全相同的副本:

```
<?php

$image->withFile('/path/to/image/mypic.jpg')
    ->withResource()
    ->save('/path/to/image/my_low_quality_pic.jpg', 10);
```

处理方法

有 8 种可用的处理方法:

- `$image->crop()`
- `$image->convert()`
- `$image->fit()`
- `$image->flatten()`
- `$image->flip()`

- `$image->resize()`
- `$image->rotate()`
- `$image->text()`

这些方法返回类实例, 所以它们可以链式调用, 如上所示。如果它们失败, 它们将抛出一个包含错误消息的 `CodeIgniter\Images\ImageException`。一个好的做法是捕获异常, 在失败时显示错误, 像这样:

```
<?php

$image = service('image');

try {
    $image->withFile('/path/to/image/mypic.jpg')
        ->fit(100, 100, 'center')
        ->save('/path/to/image/mypic_thumb.jpg');
} catch (\CodeIgniter\Images\Exceptions\ImageException $e) {
    echo $e->getMessage();
}
```

裁剪图像

可以裁剪图像, 使只保留原始图像的一部分。这通常用于在必须与某个大小/纵横比匹配时创建缩略图图像。这是通过 `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')
```

- `$width` 是所需的结果图像的宽度, 以像素为单位。
- `$height` 是所需的结果图像的高度, 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。
- `$maintainRatio` 如果为 true, 将根据需要调整最终尺寸以维持图像的原始纵横比。
- `$masterDim` 指定在 `$maintainRatio` 为 true 时应保持不变的维度。值可以是: `'width'`、`'height'` 或 `'auto'`。

要从图像中心取一个 50 x 50 像素的正方形, 你需要首先计算适当的 x 和 y 偏移值:

```
<?php

$info = service('image', 'imagick')
    ->withFile('/path/to/image/mypic.jpg')
    ->getFile()
    ->getProperties(true);

$xOffset = ($info['width'] / 2) - 25;
$yOffset = ($info['height'] / 2) - 25;

service('image', 'imagick')
    ->withFile('/path/to/image/mypic.jpg')
    ->crop(50, 50, $xOffset, $yOffset)
    ->save('/path/to/new/image.jpg');
```

转换图像

`convert()` 方法更改库的内部指示器, 以获得所需的文件格式。这不会触及实际的图像资源, 而是向 `save()` 指示要使用的格式:

```
convert(int $imageType)
```

- `$imageType` 是 PHP 的图像类型常量之一 (参见例如 https://www.php.net/manual/en/function.image-type-to-mime-type.php):

  ```
  <?php

  service('image')
      ->withFile('/path/to/image/mypic.jpg')
      ->convert(IMAGETYPE_PNG)
      ->save('/path/to/new/image.png');
  ```

备注: ImageMagick 已经以扩展名指示的类型保存文件, 忽略 `$imageType`。

调整图像大小

fit() 方法旨在帮助以“智能”的方式裁剪图像的一部分, 执行以下步骤:

- 确定应裁剪原始图像的正确部分, 以维持所需的纵横比。
- 裁剪原始图像。
- 调整到最终尺寸。

```
fit(int $width, int $height = null, string $position = 'center')
```

- $width 是图像的所需最终宽度。
- $height 是图像的所需最终高度。
- $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'。

这提供了一种更简单的裁剪方式, 将始终保持纵横比:

```
<?php

service('image', 'imagick')
    ->withFile('/path/to/image/mypic.jpg')
    ->fit(100, 150, 'left')
    ->save('/path/to/new/image.jpg');
```

扁平化图像

flatten() 方法旨在为透明图像 (PNG) 添加背景色, 并将 RGBA 像素转换为 RGB 像素

- 将透明图像转换为 jpg 时指定背景色。

```
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。

```
<?php

service('image', 'imagick')
    ->withFile('/path/to/image/mypic.png')
    ->flatten()
    ->save('/path/to/new/image.jpg');

service('image', 'imagick')
    ->withFile('/path/to/image/mypic.png')
    ->flatten(25, 25, 112)
    ->save('/path/to/new/image.jpg');
```

翻转图像

可以沿着水平或垂直轴翻转图像:

```
flip(string $dir)
```

- `$dir` 指定要翻转的轴。可以是 `'vertical'` 或 `'horizontal'`。

```
<?php

service('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()` 方法将图像调整到任何所需尺寸:

```
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` 确定图像是拉伸以适应新尺寸, 还是保持原始纵横比。

- $masterDim 指定在保持比例时应遵守哪个轴的尺寸。可以是 'width'、'height'。

在调整图像大小时, 你可以选择是保持原始图像的比例, 还是拉伸/挤压新图像以适应所需尺寸。如果 $maintainRatio 为 true, $masterDim 指定的维度将保持不变, 而另一维度将改变以匹配原始图像的纵横比。

```
<?php

service('image', 'imagick')
    ->withFile('/path/to/image/mypic.jpg')
    ->resize(200, 100, true, 'height')
    ->save('/path/to/new/image.jpg');
```

旋转图像

rotate() 方法允许你以 90 度为增量旋转图像:

```
rotate(float $angle)
```

- $angle 是要旋转的角度数。90、180、270 之一。

备注: 尽管 $angle 参数接受浮点数, 但在处理过程中它会将其转换为整数。如果值与上面列出的三个值之一不同, 它将抛出一个 CodeIgniterImagesImageException。

添加文本水印

你可以使用 text() 方法非常简单地在图像上覆盖文本水印。这对于放置版权声明、摄影师姓名或简单地将图像标记为预览很有用, 这样它们就不会在其他人的最终产品中使用。

```
text(string $text, array $options = [])
```

第一个参数是你希望显示的文本字符串。第二个参数是一个选项数组, 允许你指定文本的显示方式:

```
<?php

service('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');
```

识别的可能选项如下:

- `color` 文本颜色 (十六进制编号), 例如 `#ff0000`
- `opacity` 介于 `0` 和 `1` 之间表示文本不透明度的数字。
- `withShadow` 布尔值, 决定是否显示阴影。
- `shadowColor` 阴影的颜色 (十六进制编号)
- `shadowOffset` 阴影偏移多少像素。同时适用于垂直和水平值。
- `hAlign` 水平对齐: `'left'`, `'center'`, `'right'`
- `vAlign` 垂直对齐: `'top'`, `'middle'`, `'bottom'`
- `hOffset` x 轴上的额外偏移, 以像素为单位
- `vOffset` y 轴上的额外偏移, 以像素为单位
- `fontPath` 你要使用的 TTF 字体的完整服务器路径。如果没有给出, 则使用系统字体。
- `fontSize` 要使用的字体大小。对于使用系统字体的 GD 处理程序, 有效值为 `1` 到 `5` 之间。

备注: ImageMagick 驱动程序不识别字体路径的完整服务器路径。相反, 只提供你希望使用的已安装系统字体的名称, 例如 Calibri。

7.1.11 分页

CodeIgniter 提供了一个非常简洁但灵活的分页库，易于主题化，可与模型配合使用，并支持在单个页面上使用多个分页器。

- 加载库文件
- 使用模型进行分页
  - 自定义分页查询
  - 显示分页链接
  - 分页多个结果
  - 手动设置页码
  - 指定页码的 *URI* 段
- 手动分页
- 仅使用预期查询进行分页
- 自定义链接
  - 视图配置
  - 创建视图
  - 显示条目数

加载库文件

与 CodeIgniter 中的所有服务一样，可以通过 `Config\Services` 加载分页库，不过通常你不需要手动加载它：

```
<?php

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

使用模型进行分页

在大多数情况下，你会使用 Pager 库来对从数据库检索的结果进行分页。当使用模型 类时，可以使用其内置的 `paginate()` 方法自动检索当前批次的查询结果，并设置 Pager 库以便在控制器中使用。该方法还会通过 `page=X` 查询变量从当前 URL 中读取应显示的当前页码。

要在应用程序中提供分页的用户列表，控制器的方法应如下所示：

```
<?php

namespace App\Controllers;

use App\Models\UserModel;

class UserController extends BaseController
{
    public function index()
    {
        $model = model(UserModel::class);

        $data = [
            'users' => $model->paginate(10),
            'pager' => $model->pager,
        ];

        return view('users/index', $data);
    }
}
```

在此示例中，我们首先创建 `UserModel` 的新实例。然后将数据填充到要发送到视图的数据中。第一个元素是从数据库检索的 **users** 结果，该结果会针对正确的页面返回，每页返回 10 个用户。必须发送到视图的第二个项是 Pager 实例本身。为了方便起见，模型会保留其使用的实例并将其存储在公共属性 `$pager` 中。因此，我们获取该实例并将其分配给视图中的 `$pager` 变量。

自定义分页查询

要自定义模型中的分页查询，可以在 `paginate()` 方法之前添加查询构建器 方法。

添加 WHERE 条件

如果要添加 WHERE 条件，可以直接指定条件：

```
use App\Models\UserModel;

// In your Controller.
$model = model(UserModel::class);

$data = [
    'users' => $model->where('ban', 1)->paginate(10),
    'pager' => $model->pager,
];
```

也可以将条件移动到单独的方法中：

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    // ...

    public function banned()
    {
        $this->builder()->where('ban', 1);

        return $this; // This will allow the call chain to be used.
    }
}
```

```
use App\Models\UserModel;

// In your Controller.
$model = model(UserModel::class);

$data = [
    'users' => $model->banned()->paginate(10),
    'pager' => $model->pager,
];
```

添加 JOIN

可以连接其他表:

```
<?php

namespace App\Models;

use CodeIgniter\Model;

class NewsModel extends Model
{
    protected $table = 'news';

    // ...

    public function getPagination(?int $perPage = null): array
    {
        $this->builder()
            ->select('news.*, category.name')
            ->join('category', 'news.category_id = category.id');

        return [
            'news'  => $this->paginate($perPage),
            'pager' => $this->pager,
        ];
    }
}
```

重要: 需要理解的是，`Model::paginate()` 方法使用 **模型**和模型中的 **查询构建器**实例。因此，尝试将 `Model::paginate()` 与*$db->query()* 结合使用 **将无法工作**，因为 `$db->query()` 会立即执行查询且不与查询构建器关联。

如果需要编写无法通过查询构建器实现的复杂 SQL 查询，请尝试使用*$db->query()* 和*手动分页*。

显示分页链接

在视图中，我们需要指定显示分页链接的位置:

```
<?= $pager->links() ?>
```

这样就完成了所有设置。Pager 类将渲染首页和末页链接，以及当前页两侧超过两页的任何页面的下一页和上一页链接。

需要注意的是，该库的下一页和上一页模式与传统分页结果的方式不同。

此处的下一页和上一页链接到分页结构中要显示的链接组，而不是记录的下一页或上一页。

如果更喜欢简单的输出，可以使用 `simpleLinks()` 方法，该方法仅使用 “较旧” 和 “较新” 链接，而不是详细的分页链接:

```
<?= $pager->simpleLinks() ?>
```

在底层，该库加载一个视图文件来确定链接的格式化方式，这使得根据需求进行修改变得简单。有关如何完全自定义输出的详细信息，请参阅下文。

分页多个结果

如果需要从两个不同的结果集提供链接，可以将组名传递给大多数分页方法以保持数据分离:

```
<?php

namespace App\Controllers;
```

(续下页)

(接上页)

```
use App\Models\PageModel;
use App\Models\UserModel;

class UserController extends BaseController
{
    public function index()
    {
        $userModel = model(UserModel::class);
        $pageModel = model(PageModel::class);

        $data = [
            'users' => $userModel->paginate(10, 'group1'),
            'pages' => $pageModel->paginate(15, 'group2'),
            'pager' => $userModel->pager,
        ];

        echo view('users/index', $data);
    }
}
?>

<!-- In your view file: -->
<?= $pager->links('group1') ?>
<?= $pager->simpleLinks('group2') ?>
```

手动设置页码

如果需要指定要返回的结果页码，可以将页码作为第三个参数。当使用不同于默认 $_GET 变量的方式控制显示页面时，这会非常方便。

```
<?php

use App\Models\UserModel;

$userModel = model(UserModel::class);
$page      = 3;
```

(续下页)

(接上页)

```
$users = $userModel->paginate(10, 'group1', $page);
```

指定页码的 URI 段

也可以使用 URI 段作为页码，而不是页面查询参数。只需将段编号指定为第四个参数。由分页器生成的 URI 将类似于 **https://domain.tld/foo/bar/[pageNumber]** 而不是 **https://domain.tld/foo/bar?page=[pageNumber]**。

```
<?php

$users = $userModel->paginate(10, 'group1', null, $segment);
```

注意：`$segment` 值不能大于 URI 段数加 1。

手动分页

有时可能需要基于已知数据创建分页。可以使用 `makeLinks()` 方法手动创建链接，该方法分别将当前页码、每页结果数和总项数作为第一、第二和第三个参数：

```
<?php

namespace App\Controllers;

class UserController extends BaseController
{
    public function index()
    {
        // ...

        $pager = service('pager');

        $page    = (int) ($this->request->getGet('page') ?? 1);
        $perPage = 20;
        $total   = 200;

        // Call makeLinks() to make pagination links.
```

(续下页)

(接上页)

```
        $pager_links = $pager->makeLinks($page, $perPage, $total);

        $data = [
            // ...
            'pager_links' => $pager_links,
        ];

        return view('users/index', $data);
    }
}
?>

<!-- In your view file: -->
<?= $pager_links ?>
```

默认情况下，这将以正常方式显示链接，作为一系列链接，但可以通过将模板名称作为第四个参数传递来更改使用的显示模板。更多细节可以在以下部分找到:

```
$pager->makeLinks($page, $perPage, $total, 'template_name');
```

如前一节所述，也可以使用 URI 段作为页码，而不是页面查询参数。将段编号指定为 `makeLinks()` 的第五个参数:

```
$pager->makeLinks($page, $perPage, $total, 'template_name',
↪$segment);
```

注意：`$segment` 值不能大于 URI 段数加 1。

如果需要在单个页面上显示多个分页器，则定义组的附加参数会很有帮助:

```
<?php

$pager = service('pager');
$pager->setPath('path/for/my-group', 'my-group'); // Additionally␣
↪you could define path for every group.
$pager->makeLinks($page, $perPage, $total, 'template_name',
↪$segment, 'my-group');
```

分页库默认使用 **page** 查询参数进行 HTTP 查询（如果未指定组或给定 `default` 组名），或使用 `page_[groupName]` 作为自定义组名。

仅使用预期查询进行分页

默认情况下，所有 GET 查询都会显示在分页链接中。

例如，当访问 URL **https://domain.tld?search=foo&order=asc&hello=i+am+here&page=2** 时，可以生成第 3 页链接以及其他链接，如下所示：

```
<?php

echo $pager->links();
// Page 3 link: https://domain.tld?search=foo&order=asc&
↪hello=i+am+here&page=3
```

`only()` 方法允许将此限制为已预期的查询：

```
<?php

echo $pager->only(['search', 'order'])->links();
// Page 3 link: https://domain.tld?search=foo&order=asc&page=3
```

page 查询默认启用。并且 `only()` 作用于所有分页链接。

自定义链接

视图配置

当链接渲染到页面时，它们使用视图文件来描述 HTML。可以通过编辑 **app/Config/Pager.php** 轻松更改使用的视图：

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Pager extends BaseConfig
{
    public $templates = [
        'default_full'    => 'CodeIgniter\Pager\Views\default_full',
```

(续下页)

(接上页)

```
        'default_simple' => 'CodeIgniter\Pager\Views\default_simple
↪',
    ];

    // ...
}
```

此设置存储应使用的视图的别名和命名空间视图路径。default_full 和 default_simple 视图分别用于 links() 和 simpleLinks() 方法。要全局更改这些显示方式，可以在此处分配新视图。

例如，假设你创建了一个与 Foundation CSS 框架配合使用的新视图文件，并将其放置在 **app/Views/Pagers/foundation_full.php**。由于 **application** 目录的命名空间为 App，并且其下的所有目录直接映射到命名空间的段，因此可以通过其命名空间定位视图文件:

```
'default_full' => 'App\Views\Pagers\foundation_full'
```

由于它位于标准的 **app/Views** 目录下，因此不需要使用命名空间，因为 view() 方法可以通过文件名定位它。在这种情况下，只需提供子目录和文件名:

```
'default_full' => 'Pagers/foundation_full'
```

创建视图并在配置中设置后，它将自动被使用。你无需替换现有模板。你可以在配置文件中创建任意数量的附加模板。常见情况是在应用程序的前端和后端需要不同的样式。

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Pager extends BaseConfig
{
    public $templates = [
        'default_full'   => 'CodeIgniter\Pager\Views\default_full',
        'default_simple' => 'CodeIgniter\Pager\Views\default_simple
↪',
        'front_full'     => 'App\Views\Pagers\foundation_full',
    ];
```

(续下页)

(接上页)

```
    // ...
}
```

配置完成后，可以将其指定为 `links()`、`simpleLinks()` 和 `makeLinks()` 方法的最后一个参数:

```
<?= $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，只需编写用于创建分页链接本身的代码。不应创建不必要的包装 div，因为它可能在多个位置使用，这只会限制其可用性。通过展示现有的 `default_full` 模板，可以最容易地演示如何创建新视图:

```
<?php $pager->setSurroundCount(2) ?>

<nav aria-label="Page navigation">
    <ul class="pagination">
    <?php if ($pager->hasPrevious()) : ?>
        <li>
            <a href="<?= $pager->getFirst() ?>" aria-label="<?=␣
↪lang('Pager.first') ?>">
                <span aria-hidden="true"><?= lang('Pager.first') ?>
↪</span>
            </a>
        </li>
        <li>
            <a href="<?= $pager->getPrevious() ?>" aria-label="<?=␣
↪lang('Pager.previous') ?>">
                <span aria-hidden="true"><?= lang('Pager.previous')␣
↪?></span>
            </a>
        </li>
    <?php endif ?>
```

(续下页)

(接上页)

```
    <?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="<?= lang(
↪'Pager.next') ?>">
                <span aria-hidden="true"><?= lang('Pager.next') ?></
↪span>
            </a>
        </li>
        <li>
            <a href="<?= $pager->getLast() ?>" aria-label="<?= lang(
↪'Pager.last') ?>">
                <span aria-hidden="true"><?= lang('Pager.last') ?></
↪span>
            </a>
        </li>
    <?php endif ?>
    </ul>
</nav>
```

setSurroundCount()

在第一行中，`setSurroundCount()` 方法指定我们希望在当前页面链接两侧显示两个链接。它接受的唯一参数是要显示的链接数。

备注： 必须首先调用此方法以生成正确的分页链接。

---

hasPrevious() 和 hasNext()

如果基于传递给*setSurroundCount()* 的值，在当前页面任一侧有更多链接可以显示，则这些方法返回布尔值 `true`。

例如，假设我们有 20 页数据。当前页是第 3 页。如果周围计数为 2，则显示的链接如下所示:

```
1  |  2  |  3  |  4  |  5
```

由于显示的第一个链接是首页，因此 `hasPrevious()` 将返回 `false`，因为不存在第 0 页。但是，`hasNext()` 将返回 `true`，因为在第 5 页之后还有 15 页结果。

getPrevious() 和 getNext()

这些方法返回当前页面编号链接任一侧的上一组或下一组结果的 **URL**。

例如，当前页设置为 5，你希望前后链接（surroundCount）各为 2 个，这将给出如下内容:

```
3  |  4  |  5  |  6  |  7
```

`getPrevious()` 返回第 2 页的 URL。`getNext()` 返回第 8 页的 URL。

如果要获取第 4 页和第 6 页，请改用*getPreviousPage() 和 getNextPage()*。

getFirst() 和 getLast()

与*getPrevious() 和 getNext()* 类似，这些方法返回结果集中首页和末页的 **URL**。

links()

返回有关所有编号链接的数据数组。每个链接的数组包含链接的 uri、标题（即数字）以及一个布尔值，指示该链接是否是当前/活动链接:

```
<?php

$link = [
    'active' => false,
```

(续下页)

(接上页)

```
    'uri'     => 'https://example.com/foo?page=2',
    'title'   => 1,
];
```

在标准分页结构的代码中，*getPrevious()* 和 *getNext()* 方法用于分别获取上一组和下一组分页链接的 URL。

如果希望使用基于当前页的上一页和下一页链接的分页结构，只需将*getPrevious()* 和 *getNext()* 方法替换为*getPreviousPage()* 和 *getNextPage()*，并将*hasPrevious()* 和 *hasNext()* 方法分别替换为*hasPreviousPage()* 和 *hasNextPage()*。

请参阅以下示例：

```
<nav aria-label="<?= lang('Pager.pageNavigation') ?>">
    <ul class="pagination">
        <?php if ($pager->hasPreviousPage()) : ?>
            <li>
                <a href="<?= $pager->getFirst() ?>" aria-label="<?=␣
↪lang('Pager.first') ?>">
                    <span aria-hidden="true"><?= lang('Pager.first
↪') ?></span>
                </a>
            </li>
            <li>
                <a href="<?= $pager->getPreviousPage() ?>" aria-
↪label="<?= lang('Pager.previous') ?>">
                    <span aria-hidden="true"><?= lang('Pager.
↪previous') ?></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 ?>
```

(续下页)

(接上页)

```
        <?php if ($pager->hasNextPage()) : ?>
            <li>
                <a href="<?= $pager->getNextPage() ?>" aria-label="
↪<?= lang('Pager.next') ?>">
                    <span aria-hidden="true"><?= lang('Pager.next')␣
↪?></span>
                </a>
            </li>
            <li>
                <a href="<?= $pager->getLast() ?>" aria-label="<?=␣
↪lang('Pager.last') ?>">
                    <span aria-hidden="true"><?= lang('Pager.last')␣
↪?></span>
                </a>
            </li>
        <?php endif ?>
    </ul>
</nav>
```

hasPreviousPage() 和 hasNextPage()

如果当前显示页面的前后存在页面链接，则这些方法返回布尔值 `true`。

例如，假设我们有 20 页数据。当前页是第 3 页。如果周围计数为 2，则显示的链接如下所示:

```
1 | 2 | 3 | 4 | 5
```

`hasPreviousPage()` 将返回 `true`，因为存在第 2 页。而 `hasNextPage()` 将返回 `true`，因为存在第 4 页。

备注: 与*hasPrevious()* 和 *hasNext()* 的区别在于，这些方法基于当前页，而*hasPrevious()* 和 *hasNext()* 基于根据*setSurroundCount()* 传递的值在当前页前后显示的链接组。

---

getPreviousPage() 和 getNextPage()

这些方法返回与当前显示页面前后页面对应的 **URL**。

例如，当前页设置为 5，你希望前后链接（surroundCount）各为 2 个，这将给出如下内容:

```
3 | 4 | 5 | 6 | 7
```

`getPreviousPage()` 返回第 4 页的 URL。`getNextPage()` 返回第 6 页的 URL。

备注: *getPrevious() 和 getNext()* 返回编号链接任一侧的上一组或下一组结果的 URL。

如果需要页码而不是 URL，可以使用以下方法:

getPreviousPageNumber() 和 getNextPageNumber()

这些方法返回与当前显示页面前后页面对应的页码。

getFirstPageNumber() 和 getLastPageNumber()

这些方法返回要显示的链接组中首页和末页的页码。例如，如果要显示的链接组如下所示:

```
3 | 4 | 5 | 6 | 7
```

`getFirstPageNumber()` 将返回 3，而 `getLastPageNumber()` 将返回 7。

备注: 要获取整个结果集的首页和末页页码，可以使用以下方法：首页页码始终为 1，*getPageCount()* 可用于检索末页页码。

---

getCurrentPageNumber()

此方法返回当前页的页码。

getPageCount()

此方法返回总页数。

显示条目数

在 4.6.0 版本加入.

在对内容进行分页时，通常需要显示总条目数以及当前页显示的条目范围。为了简化这一任务，我们新增了一些方法。这些方法能够更轻松地管理和显示分页详细信息。以下是一个示例：

```
<?php $pager->setSurroundCount(1) ?>

<p>
    Showing <span class="font-medium"><?= $pager->getPerPageStart()␣
↪?></span>
    to <span class="font-medium"><?= $pager->getPerPageEnd() ?></
↪span>
    of <span class="font-medium"><?= $pager->getTotal() ?></span>␣
↪results
</p>
```

getTotal()

返回页面的总条目数。

getPerPage()

返回页面上显示的条目数。

getPerPageStart()

返回页面起始条目的编号。

getPerPageEnd()

返回页面结束条目的编号。

7.1.12 Publisher

Publisher 类库提供了一种在项目内复制文件的方法，具备强大的检测和错误检查功能。

- 加载类库
- 概念与用法
  - 按需使用
  - 自动化与发现
  - 安全性
- 示例
  - 文件同步示例
  - 资源依赖示例
  - 模块部署示例
- 类库参考
  - 支持方法
  - 文件输出方法
  - 文件修改方法

加载类库

由于 Publisher 实例与其源路径和目标路径相关联，因此该类库不通过 Services 提供，而应直接实例化或扩展。例如：

```
<?php

$publisher = new \CodeIgniter\Publisher\Publisher();
```

概念与用法

Publisher 解决了在后端框架中工作时的一些常见问题：

- 如何维护具有版本依赖关系的项目资源？
- 如何管理需要 Web 访问的上传文件和其他”动态”文件？
- 当框架或模块变更时如何更新项目？
- 组件如何将新内容注入现有项目？

本质上，发布操作等同于将文件复制到项目中。Publisher 扩展了 FileCollection，通过流式方法链来读取、过滤和处理输入文件，然后将它们复制或合并到目标路径。你可以在控制器或其他组件中按需使用 Publisher，也可以通过扩展类并利用 spark publish 的发现机制来分阶段执行发布操作。

按需使用

通过直接实例化类来使用 Publisher：

```
<?php

$publisher = new \CodeIgniter\Publisher\Publisher();
```

默认情况下，源路径和目标路径分别设置为 ROOTPATH 和 FCPATH，这使得 Publisher 可以轻松访问项目中的任何文件并使其可通过 Web 访问。你也可以在构造函数中传入新的源路径或同时传入源路径和目标路径：

```
<?php
```

(续下页)

(接上页)

```
use CodeIgniter\Publisher\Publisher;

$vendorPublisher = new Publisher(ROOTPATH . 'vendor');
$filterPublisher = new Publisher('/path/to/module/Filters', APPPATH␣
↪. 'Filters');

// Once the source and destination are set you may start adding␣
↪relative input files
$frameworkPublisher = new Publisher(ROOTPATH . 'vendor/codeigniter4/
↪codeigniter4');

// All "path" commands are relative to $source
$frameworkPublisher->addPath('app/Config/Cookie.php');

// You may also add from outside the source, but the files will not␣
↪be merged into subdirectories
$frameworkPublisher->addFiles([
    '/opt/mail/susan',
    '/opt/mail/ubuntu',
]);
$frameworkPublisher->addDirectory(SUPPORTPATH . 'Images');
```

当所有文件准备就绪后，使用输出命令（**copy()** 或 **merge()**）将暂存文件处理到目标路径：

```
<?php

// Place all files into $destination
$frameworkPublisher->copy();

// Place all files into $destination, overwriting existing files
$frameworkPublisher->copy(true);

// Place files into their relative $destination directories,␣
↪overwriting and saving the boolean result
$result = $frameworkPublisher->merge(true);
```

完整方法描述请参阅类库参考。

自动化与发现

你可能需要在应用部署或维护时执行定期发布任务。Publisher 利用强大的 Autoloader 来定位所有准备发布的子类：

```
<?php

use CodeIgniter\CLI\CLI;
use CodeIgniter\Publisher\Publisher;

foreach (Publisher::discover() as $publisher) {
    $result = $publisher->publish();

    if ($result === false) {
        CLI::error($publisher::class . ' failed to publish!', 'red
↪');
    }
}
```

默认情况下 discover() 会在所有命名空间中搜索“Publishers”目录，但你可以指定其他目录来返回找到的子类：

```
<?php

use CodeIgniter\Publisher\Publisher;

$memePublishers = Publisher::discover('CatGIFs');
```

大多数情况下你无需自行处理发现机制，直接使用提供的“publish”命令即可：

```
php spark publish
```

默认情况下，类扩展中的 publish() 会从 $source 添加所有文件并将其合并到目标路径，遇到冲突时覆盖。

在指定命名空间中发现

在 4.6.0 版本加入.

自 v4.6.0 起，你还可以扫描特定命名空间。这不仅减少了需要扫描的文件数量，也避免了重新运行 Publisher 的需求。只需在 `discover()` 方法的第二个参数中指定目标根命名空间：

```
<?php

use CodeIgniter\Publisher\Publisher;

$memePublishers = Publisher::discover('Publishers', 'Namespace\
↪Vendor\Package');
```

指定的命名空间必须已注册到 CodeIgniter。你可以使用 "spark namespaces" 命令查看所有命名空间列表：

```
php spark namespaces
```

"publish" 命令也提供 `--namespace` 选项来定义搜索 Publisher 的命名空间，适用于来自库的情况：

```
php spark publish --namespace Namespace\Vendor\Package
```

安全性

为防止模块向项目注入恶意代码，`Publisher` 包含一个配置文件来定义允许的目标目录和文件模式。默认情况下，文件只能发布到项目内（防止访问文件系统其他部分），且 **public/** 文件夹（`FCPATH`）仅接收以下扩展名的文件：

- Web 资源：css, scss, js, map
- 非可执行 Web 文件：htm, html, xml, json, webmanifest
- 字体：ttf, eot, woff, woff2
- 图像：gif, jpg, jpeg, tif, tiff, png, webp, bmp, ico, svg

如需调整项目安全设置，请修改 **app/Config/Publisher.php** 中 `Config\Publisher` 的 `$restrictions` 属性。

示例

以下是几个典型用例及其实现，帮助你快速上手发布操作。

文件同步示例

你希望在首页展示”每日图片”。虽然已有每日图片源，但需要将实际文件同步到项目的可浏览位置 **public/images/daily_photo.jpg**。可以设置每日运行的自定义命令 来处理：

```
<?php

namespace App\Commands;

use CodeIgniter\CLI\BaseCommand;
use CodeIgniter\Publisher\Publisher;
use Throwable;

class DailyPhoto extends BaseCommand
{
    protected $group       = 'Publication';
    protected $name        = 'publish:daily';
    protected $description = 'Publishes the latest daily photo to␣
↪the homepage.';

    public function run(array $params)
    {
        $publisher = new Publisher('/path/to/photos/', FCPATH .
↪'assets/images');

        try {
            $publisher->addPath('daily_photo.jpg')->copy(true); //␣
↪`true` to enable overwrites
        } catch (Throwable $e) {
            $this->showError($e);
        }
    }
}
```

现在运行 `spark publish:daily` 即可保持首页图片更新。如果图片来自外部 API 怎

么办？可以使用 addUri() 替代 addPath() 来下载远程资源并发布：

```
<?php

$publisher->addUri('https://example.com/feeds/daily_photo.jpg')->
↪copy(true);
```

资源依赖示例

你想在项目中集成前端库"Bootstrap"，但频繁更新带来维护难题。可以通过扩展 Publisher 创建发布定义来同步前端资源。例如 **app/Publishers/BootstrapPublisher.php** 可能如下：

```
<?php

namespace App\Publishers;

use CodeIgniter\Publisher\Publisher;

class BootstrapPublisher extends Publisher
{
    /**
     * Tell Publisher where to get the files.
     * Since we will use Composer to download
     * them we point to the "vendor" directory.
     *
     * @var string
     */
    protected $source = VENDORPATH . 'twbs/bootstrap/';

    /**
     * FCPATH is always the default destination,
     * but we may want them to go in a sub-folder
     * to keep things organized.
     *
     * @var string
     */
    protected $destination = FCPATH . 'bootstrap';
```

(续下页)

(接上页)

```
    /**
     * Use the "publish" method to indicate that this
     * class is ready to be discovered and automated.
     */
    public function publish(): bool
    {
        return $this
            // Add all the files relative to $source
            ->addPath('dist')

            // Indicate we only want the minimized versions
            ->retainPattern('*.min.*')

            // Merge-and-replace to retain the original directory␣
↪structure
            ->merge(true);
    }
}
```

备注: 目录 `$destination` 必须在执行命令前创建。

现在通过 Composer 添加依赖并调用 `spark publish` 执行发布:

```
composer require twbs/bootstrap
php spark publish
```

最终会生成如下结构:

```
public/.htaccess
public/favicon.ico
public/index.php
public/robots.txt
public/
    bootstrap/
        css/
            bootstrap.min.css
```

(续下页)

(接上页)

```
            bootstrap-utilities.min.css.map
            bootstrap-grid.min.css
            bootstrap.rtl.min.css
            bootstrap.min.css.map
            bootstrap-reboot.min.css
            bootstrap-utilities.min.css
            bootstrap-reboot.rtl.min.css
            bootstrap-grid.min.css.map
        js/
            bootstrap.esm.min.js
            bootstrap.bundle.min.js.map
            bootstrap.bundle.min.js
            bootstrap.min.js
            bootstrap.esm.min.js.map
            bootstrap.min.js.map
```

模块部署示例

你希望让使用流行认证模块的开发者能够扩展默认的 Migration、Controller 和 Model 行为。可以创建模块专属的”publish”命令来向应用中注入这些组件：

```
<?php

namespace Math\Auth\Commands;

use CodeIgniter\CLI\BaseCommand;
use CodeIgniter\Publisher\Publisher;
use Throwable;

class AuthPublish extends BaseCommand
{
    protected $group       = 'Auth';
    protected $name        = 'auth:publish';
    protected $description = 'Publish Auth components into the␣
↪current application.';

    public function run(array $params)
```

(续下页)

(接上页)

```
    {
        // Use the Autoloader to figure out the module path
        $source = service('autoloader')->getNamespace('Math\\Auth
↪')[0];

        $publisher = new Publisher($source, APPPATH);

        try {
            // Add only the desired components
            $publisher->addPaths([
                'Controllers',
                'Database/Migrations',
                'Models',
            ])->merge(false); // Be careful not to overwrite␣
↪anything
        } catch (Throwable $e) {
            $this->showError($e);

            return;
        }

        // If publication succeeded then update namespaces
        foreach ($publisher->getPublished() as $file) {
            // Replace the namespace
            $contents = file_get_contents($file);
            $contents = str_replace('namespace Math\\Auth',
↪'namespace ' . APP_NAMESPACE, $contents);
            file_put_contents($file, $contents);
        }
    }
}
```

现在当模块用户运行 `php spark auth:publish` 时，项目中会添加以下文件:

```
app/Controllers/AuthController.php
app/Database/Migrations/2017-11-20-223112_create_auth_tables.php.php
app/Models/LoginModel.php
app/Models/UserModel.php
```

类库参考

备注: `Publisher` 继承自*FileCollection*，因此可以使用该类的所有文件读取和过滤方法。

支持方法

[static] discover(string $directory = 'Publishers'): Publisher[]

发现并返回指定命名空间目录中的所有 Publisher。例如，如果同时存在 **app/Publishers/FrameworkPublisher.php** 和 **myModule/src/Publishers/AssetPublisher.php** 且都是 `Publisher` 的扩展，则 `Publisher::discover()` 会返回每个实例。

publish(): bool

处理完整的输入-处理-输出链。默认情况下等同于调用 `addPath($source)` 和 `merge(true)`，但子类通常提供自己的实现。运行 `spark publish` 时会对所有发现的 Publisher 调用 `publish()`。返回成功或失败状态。

getScratch(): string

返回临时工作区路径（必要时创建）。某些操作使用中间存储来暂存文件和变更，此方法提供可写入的临时目录路径。

getErrors(): array<string, Throwable>

返回上次写入操作的错误信息。数组键是引发错误的文件路径，值是对应的 Throwable 对象。使用 Throwable 的 `getMessage()` 获取错误信息。

addPath(string $path, bool $recursive = true)

添加相对路径指示的所有文件。路径是相对于 `$source` 的实际文件或目录引用。如果相对路径解析为目录，则 `$recursive` 会包含子目录。

addPaths(array $paths, bool $recursive = true)

添加多个相对路径指示的所有文件。路径是相对于 `$source` 的实际文件或目录引用。如果相对路径解析为目录，则 `$recursive` 会包含子目录。

addUri(string $uri)

使用 `CURLRequest` 将 URI 内容下载到临时工作区，然后将结果文件添加到列表。

addUris(array $uris)

使用 `CURLRequest` 将多个 URI 内容下载到临时工作区，然后将结果文件添加到列表。

备注： CURL 请求是简单的 `GET` 请求并使用响应体作为文件内容。某些远程文件可能需要自定义请求处理。

文件输出方法

wipe()

删除 `$destination` 中的所有文件、目录和子目录。

重要： 谨慎使用。

---

copy(bool $replace = true): bool

将所有文件复制到 `$destination`。不重建目录结构，所有文件将置于同一目标目录。`$replace` 为 true 时会覆盖现有文件。返回成功或失败状态，使用 `getPublished()` 和 `getErrors()` 排查故障。注意同名文件冲突，例如：

```
<?php

use CodeIgniter\Publisher\Publisher;

$publisher = new Publisher('/home/source', '/home/destination');
$publisher->addPaths([
    'pencil/lead.png',
    'metal/lead.png',
]);

// This is bad! Only one file will remain at /home/destination/lead.
↪png
$publisher->copy(true);
```

merge(bool $replace = true): bool

将所有文件按相对子目录结构复制到 `$destination`。匹配 `$source` 的文件会置于 `$destination` 的对应目录，实现”镜像”或”rsync”操作。`$replace` 为 true 时覆盖现有文件（不影响目标目录其他文件）。返回成功或失败状态，使用 `getPublished()` 和 `getErrors()` 排查故障。

示例：

```
<?php

use CodeIgniter\Publisher\Publisher;

$publisher = new Publisher('/home/source', '/home/destination');
$publisher->addPaths([
    'pencil/lead.png',
    'metal/lead.png',
]);
```

(续下页)

(接上页)

```
// Results in "/home/destination/pencil/lead.png" and "/home/
↪destination/metal/lead.png"
$publisher->merge();
```

文件修改方法

replace(string $file, array $replaces): bool

在 4.3.0 版本加入.

替换 `$file` 文件内容。第二个参数 `$replaces` 数组以搜索字符串为键，替换内容为值。

```
<?php

use CodeIgniter\Publisher\Publisher;

$source    = service('autoloader')->getNamespace('CodeIgniter\\
↪Shield')[0];
$publisher = new Publisher($source, APPPATH);

$file = APPPATH . 'Config/Auth.php';

$publisher->replace(
    $file,
    [
        'use CodeIgniter\Config\BaseConfig;' . "\n" => '',
        'class App extends BaseConfig'              => 'class App␣
↪extends \Some\Package\SomeConfig',
    ],
);
```

addLineAfter(string $file, string $line, string $after): bool

在 4.3.0 版本加入.

在包含特定字符串 `$after` 的行之后添加 `$line`。

```
<?php

use CodeIgniter\Publisher\Publisher;

$source    = service('autoloader')->getNamespace('CodeIgniter\\
↪Shield')[0];
$publisher = new Publisher($source, APPPATH);

$file = APPPATH . 'Config/App.php';

$publisher->addLineAfter(
    $file,
    '    public int $myOwnConfig = 1000;', // Adds this line
    'public bool $CSPEnabled = false;',     // After this line
);
```

addLineBefore(string $file, string $line, string $after): bool

在 4.3.0 版本加入.

在包含特定字符串 `$after` 的行之前添加 `$line`。

```
<?php

use CodeIgniter\Publisher\Publisher;

$source    = service('autoloader')->getNamespace('CodeIgniter\\
↪Shield')[0];
$publisher = new Publisher($source, APPPATH);

$file = APPPATH . 'Config/App.php';

$publisher->addLineBefore(
```

(续下页)

(接上页)

```
    $file,
    '    public int $myOwnConfig = 1000;', // Add this line
    'public bool $CSPEnabled = false;',     // Before this line
);
```

7.1.13 安全

安全类包含帮助保护你的网站免受跨站请求伪造 (CSRF) 攻击的方法。

- 加载库
- 跨站请求伪造 (*CSRF*)
  - 先决条件
    - 当自动路由被禁用时
    - 当自动路由被启用时
  - *CSRF* 配置
    - *CSRF* 保护方法
    - 令牌随机化
    - 令牌再生成
    - 失败时重定向
  - 启用 *CSRF* 保护
  - *HTML* 表单
  - 用户发送令牌的顺序
- 其他有用的方法
  - *sanitizeFilename()*

加载库

如果你加载库的唯一兴趣是处理 CSRF 保护, 那么你永远不需要加载它, 因为它作为过滤器运行且没有手动交互。

但是, 如果你发现确实需要直接访问, 你可以通过 Services 文件加载它:

```
<?php

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

跨站请求伪造 (CSRF)

> **警告:** CSRF 保护仅适用于 **POST/PUT/PATCH/DELETE** 请求。不保护其他方法的请求。

先决条件

当你使用 CodeIgniter 的 CSRF 保护时, 仍需要按如下方式编写代码。否则,CSRF 保护可能会被绕过。

当自动路由被禁用时

执行以下操作之一:

1. 不要使用 `$routes->add()`, 并在路由中使用 HTTP 动词。
2. 在处理之前, 在控制器方法中检查请求方法。

例如:

```
if (! $this->request->is('post')) {
    return $this->response->setStatusCode(405)->setBody('Method Not␣
↪Allowed');
}
```

备注: 自 v4.3.0 起, 可以使用*$this->request->is()* 方法。在以前的版本中, 你需要使用 `if`

`(strtolower($this->request->getMethod()) !== 'post')`。

当自动路由被启用时

1. 在处理之前, 在控制器方法中检查请求方法。

例如:

```
if (! $this->request->is('post')) {
    return $this->response->setStatusCode(405)->setBody('Method Not␣
↪Allowed');
}
```

CSRF 配置

CSRF 保护方法

> **警告:** 如果你使用*Session*, 一定要使用基于 Session 的 CSRF 保护。基于 Cookie 的 CSRF 保护无法防止同站攻击。详情请参见 GHSA-5hm8-vh6r-2cjq 。

默认情况下, 使用基于 Cookie 的 CSRF 保护。它是 OWASP 跨站请求伪造预防备忘单上的 双重提交 Cookie。

你也可以使用基于会话的 CSRF 保护。它是 同步令牌模式。

你可以通过编辑以下配置参数的值在 **app/Config/Security.php** 来设置使用基于会话的 CSRF 保护:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Security extends BaseConfig
{
```

(续下页)

(接上页)

```
    public $csrfProtection = 'session';

    // ...
}
```

令牌随机化

为了缓解像 BREACH 这样的压缩旁道攻击, 并阻止攻击者猜测 CSRF 令牌, 你可以配置令牌随机化 (默认关闭)。

如果你启用它, 随机掩码会添加到令牌中并用于扰乱它。

你可以通过编辑以下配置参数的值在 **app/Config/Security.php** 来启用它:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Security extends BaseConfig
{
    public $tokenRandomize = true;

    // ...
}
```

令牌再生成

令牌可以在每次提交时重新生成 (默认), 或者在 Session 或 CSRF Cookie 的整个生命周期内保持不变。

令牌的默认再生成提供了更严格的安全性, 但可能导致可用性问题, 因为其他令牌变得无效 (后退/前进导航、多个选项卡/窗口、异步操作等)。你可以通过编辑以下配置参数的值在 **app/Config/Security.php** 来更改此行为:

```php
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Security extends BaseConfig
{
    public $regenerate = true;

    // ...
}
```

警告: 如果你使用基于 Cookie 的 CSRF 保护，并在提交后使用 *redirect()*，你必须调用 `withCookie()` 来发送重新生成的 CSRF Cookie。详情请参见重定向。

备注: 自 v4.2.3 起, 你可以用 `Security::generateHash()` 方法手动重新生成 CSRF 令牌。

失败时重定向

从 v4.5.0 开始，当请求未通过 CSRF 验证检查时，在生产环境中，默认情况下用户会被重定向到前一页面；在其他环境中，则会抛出一个 SecurityException。

备注: 在生产环境中, 当你使用 HTML 表单时, 建议启用此重定向以获得更好的用户体验。

升级用户应检查他们的配置文件。

如果你希望将其重定向到前一页面，请在 **app/Config/Security.php** 中将以下配置参数值设置为 `true`:

```php
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Security extends BaseConfig
{
    // ...

    public bool $redirect = true;

    // ...
}
```

重定向后, 会设置一个 `error` 闪存消息, 可以使用以下视图代码显示给最终用户:

```php
<?= session()->getFlashdata('error') ?>
```

这比简单地崩溃提供了更好的体验。

即使重定向值为 `true`,AJAX 调用也不会重定向, 而会抛出 SecurityException。

启用 CSRF 保护

你可以通过更改 **app/Config/Filters.php** 并全局启用 *csrf* 过滤器来启用 CSRF 保护:

```php
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public $globals = [
        'before' => [
            // 'honeypot',
            'csrf',
```

(续下页)

(接上页)

```
        ],
    ];

    // ...
}
```

可以将某些 URI 从 CSRF 保护中排除 (例如期望外部 POST 内容的 API 端点)。你可以通过在过滤器中将它们添加为例外来添加这些 URI:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public $globals = [
        'before' => [
            'csrf' => ['except' => ['api/record/save']],
        ],
    ];

    // ...
}
```

正则表达式也支持 (不区分大小写):

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public $globals = [
        'before' => [
```

(续下页)

(接上页)

```
            'csrf' => ['except' => ['api/record/[0-9]+']],
        ],
    ];

    // ...
}
```

也可以仅针对特定方法启用 CSRF 过滤器:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public $methods = [
        'GET'  => ['csrf'],
        'POST' => ['csrf'],
    ];

    // ...
}
```

警告: 如果使用 `$methods` 过滤器, 你应该禁用自动路由 (传统), 因为自动路由（传统版）允许任何 HTTP 方法访问控制器。使用你不期望的方法访问控制器可能会绕过过滤器。

HTML 表单

如果使用表单辅助函数, 那么 `form_open()` 将自动在你的表单中插入一个隐藏的 csrf 字段。

备注: 要使用 CSRF 字段的自动生成, 你需要对表单页面打开 CSRF 过滤器。在大多数

情况下, 它使用 GET 方法请求。

如果没有, 你可以使用始终可用的 csrf_token() 和 csrf_hash() 函数:

```
<input type="hidden" name="<?= csrf_token() ?>" value="<?= csrf_
↪hash() ?>" />
```

另外, 你可以使用 csrf_field() 方法为你生成这个隐藏的输入字段:

```
// 生成:<input type="hidden" name="{csrf_token}" value="{csrf_hash}
↪" />
<?= csrf_field() ?>
```

在发送 JSON 请求时,CSRF 令牌也可以作为一个参数传递。通过 csrf_header() 函数可用的下一个传递 CSRF 令牌的方式是特殊的 Http 头。

另外, 你可以使用 csrf_meta() 方法为你生成这个方便的 meta 标签:

```
// 生成:<meta name="{csrf_header}" content="{csrf_hash}" />
<?= csrf_meta() ?>
```

用户发送令牌的顺序

检查 CSRF 令牌可用性的顺序如下:

1. $_POST 数组
2. HTTP 头
3. php://input (JSON 请求) - 请记住, 这种方法是最慢的, 因为我们必须解码 JSON 然后重新编码它
4. php://input (原始 body) - 适用于 PUT、PATCH 和 DELETE 类型的请求

备注: 自 v4.4.2 起，会检查 php://input (原始 body)。

其他有用的方法

你很少需要直接使用 Security 类中的大多数方法。以下是与 CSRF 保护无关的可能对你有帮助的方法。

sanitizeFilename()

试图清理文件名以防止目录遍历攻击和其他安全威胁, 这对于通过用户输入提供的文件特别有用。第一个参数是要清理的路径。

如果用户输入可以包含相对路径, 例如 **file/in/some/approved/folder.txt**, 你可以将第二个可选参数 `$relativePath` 设置为 `true`。

```
<?php

$path = $security->sanitizeFilename($request->getVar('filepath'));
```

7.1.14 Session 库

Session 类允许你在用户浏览你的站点时维护用户的 "状态" 并跟踪他们的活动。

CodeIgniter 带有几个 session 存储驱动器, 你可以在目录内容的最后一节中看到:

- *使用 Session 类*
  - *初始化 Session*
  - *Session 如何工作?*
  - *什么是 Session 数据?*
  - *检索 Session 数据*
  - *添加 Session 数据*
  - *向 Session 数据中推送新值*
  - *删除 Session 数据*
  - *Flashdata*
  - *Tempdata*

- 关闭一个 Session
- 销毁一个 Session
- 访问 Session 元数据

- Session 首选项
- Session 驱动程序
  - FileHandler 驱动程序 (默认)
  - DatabaseHandler 驱动程序
  - RedisHandler 驱动程序
  - MemcachedHandler 驱动程序

使用 Session 类

初始化 Session

Session 通常会与每个页面加载一起全局运行, 所以 Session 类应该自动初始化。

要访问和初始化 Session:

```
<?php

$session = service('session', $config);
```

`$config` 参数是可选的 - 你的应用配置。如果没有提供, 服务注册器将实例化你的默认配置。

加载后，Session 库对象将可通过以下方式访问:

```
$session
```

另外, 你可以使用辅助函数, 它将使用默认配置选项。这个版本的可读性更好一些, 但不接受任何配置选项。

```
<?php

$session = session();
```

Session 如何工作?

当页面加载时,session 类将检查用户的浏览器是否发送了有效的 session cookie。如果 session cookie **不存在** (或与服务器上存储的不匹配或已过期), 则将创建一个新 session 并保存。

如果存在有效的 session, 则会更新其信息。使用每次更新, 如果配置了 session ID 可能会被重新生成。

Initialized 后,Session 类会自动运行这一点非常重要。你不需要做任何事情就可以引起上述行为发生。如下所示, 你可以使用 session 数据, 但读取、写入和更新 session 的过程是自动的。

备注: 在 CLI 下,Session 库将自动停止自己, 因为这是一个完全基于 HTTP 协议的概念。

关于并发的注意事项

除非你正在开发一个使用大量 AJAX 的网站, 否则可以跳过这个部分。但是, 如果是这样, 并且如果遇到性能问题, 那么这条注意事项正是你所需要的。

CodeIgniter 2.x 中的 session 并没有实现锁定, 这意味着可以完全同时运行两个使用相同 session 的 HTTP 请求。使用一个更合适的技术术语来说——请求是非阻塞的。

但是, 在 session 背景下的非阻塞请求也意味着不安全, 因为一个请求中的 session 数据修改 (或 session ID 重新生成) 可能会干扰第二个并发请求的执行。这一细节是许多问题的根源, 也是 CodeIgniter 3 完全重写 Session 库的主要原因。

为什么要告诉你这些? 因为在试图找到性能问题的原因后, 你可能会得出锁定是问题的结论, 因此会研究如何删除锁……

不要这样做! 删除锁将是 **错误**的, 并且会给你带来更多问题!

锁定不是问题, 它是解决方案。你的问题在于, 你仍然打开了 session, 而你已经处理了它, 因此不再需要它。所以, 你需要的是在不再需要它时关闭当前请求的 session。

```
<?php

$session->close();
```

什么是 Session 数据?

Session 数据只是一个与特定 Session ID 相关联的数组 (Cookie)。

如果你之前使用过 PHP 的 session, 你应该熟悉 PHP 的 $_SESSION 超全局变量 (如果不熟悉, 请阅读该链接的内容)。

CodeIgniter 通过相同的方式提供对其 Session 数据的访问, 因为它使用 PHP 提供的 Session 处理程序机制。使用 Session 数据就像操作 (读取、设置和取消设置值) `$_SESSION` 数组一样简单。

备注: 一般来说, 使用全局变量是不好的实践。所以直接使用超全局 `$_SESSION` 不是推荐的做法。

此外,CodeIgniter 还提供了 2 种特殊类型的 Session 数据, 下面将进一步解释: *Flashdata* 和*Tempdata*。

备注: 出于历史原因, 我们将不包括 Flashdata 和 Tempdata 的 Session 数据称为 "userdata"。

检索 Session 数据

可以通过 `$_SESSION` 超全局变量访问 Session 数组中的任何信息:

```
<?php

$item = $_SESSION['item'];
```

或者通过常规的访问器方法:

```
<?php

$item = $session->get('item');
```

或者通过魔术 getter:

```
<?php

$item = $session->item;
```

甚至可以通过 session 辅助函数方法:

```
<?php

$item = session('item');
```

其中 `item` 是与你希望获取的项对应的数组键。例如, 要将先前存储的 `name` 项赋值给 `$name` 变量, 你将执行:

```
<?php

$name = $_SESSION['name'];

// or:

$name = $session->name;

// or:

$name = $session->get('name');
```

备注: 如果尝试访问的项不存在, `get()` 方法将返回 null。

如果你想检索所有现有的 session 数据, 只需省略项键 (魔术 getter 仅适用于单个属性值):

```
<?php

$userData = $_SESSION;
// or:
$userData = $session->get();
```

重要: `get()` 方法在通过键检索单个项时, 将返回 flashdata 或 tempdata 项。但是在从

session 中获取所有数据时不会返回 flashdata 或 tempdata。

添加 Session 数据

假设特定用户登录你的站点。一旦认证, 你可以将用户名和电子邮件地址添加到 session 中, 这使得当你需要时可以全局访问它们而无需运行数据库查询。

你可以简单地像对任何其他变量一样将数据分配给 `$_SESSION` 数组。或者作为 `$session` 的属性。

你可以传递一个包含新 Session 数据的数组到 `set()` 方法:

```
<?php

$session->set($array);
```

其中 `$array` 是一个关联数组, 包含你的新数据。这里是一个例子:

```
<?php

$newdata = [
    'username'  => 'johndoe',
    'email'     => 'johndoe@some-site.com',
    'logged_in' => true,
];

$session->set($newdata);
```

如果你想一次添加一个 Session 数据, `set()` 也支持这种语法:

```
<?php

$session->set('some_name', 'some_value');
```

如果你想验证一个 Session 值是否存在, 只需使用 `isset()` 检查:

```
<?php

// returns false if the 'some_name' item doesn't exist or is null,
```

(续下页)

(接上页)

```
// true otherwise:
if (isset($_SESSION['some_name'])) {
    // ...
}
```

或者你可以调用 has():

```
<?php

$session->has('some_name');
```

向 Session 数据中推送新值

push() 方法用于将新值推送到一个是数组的 Session 值上。例如, 如果 hobbies 键包含爱好数组, 你可以像这样向数组添加一个新值:

```
<?php

$session->push('hobbies', ['sport' => 'tennis']);
```

删除 Session 数据

与任何其他变量一样, 可以通过 unset() 取消设置 $_SESSION 中的值:

```
<?php

unset($_SESSION['some_name']);
// or multiple values:
unset(
    $_SESSION['some_name'],
    $_SESSION['another_name']
);
```

同样, 正如 set() 可用于向 Session 添加信息一样, remove() 可用于通过传递 session 键来删除它。例如, 如果你要从 Session 数据数组中删除 some_name:

```
<?php

$session->remove('some_name');
```

该方法还接受要取消设置的项键数组:

```
<?php

$array_items = ['username', 'email'];
$session->remove($array_items);
```

Flashdata

CodeIgniter 支持“flashdata”, 也就是只在下一次请求中可用, 然后自动清除的 session 数据。

这在需要一次性信息、错误或状态消息时非常有用 (例如:“记录 2 已删除”)。

需要注意的是,flashdata 变量是由 CodeIgniter session 处理程序管理的普通 session 变量。

要将现有项标记为“flashdata”:

```
<?php

$session->markAsFlashdata('item');
```

如果要将多个项标记为 flashdata, 只需将键作为数组传递即可:

```
<?php

$session->markAsFlashdata(['item', 'item2']);
```

要添加 flashdata:

```
<?php

$_SESSION['item'] = 'value';
$session->markAsFlashdata('item');
```

或者可以使用 `setFlashdata()` 方法:

```
<?php

$session->setFlashdata('item', 'value');
```

与 `set()` 一样, 你也可以向 `setFlashdata()` 传递数组。

通过 `$_SESSION` 读取 flashdata 变量, 就像读取常规 session 数据一样:

```
<?php

$item = $_SESSION['item'];
```

重要: `get()` 方法在通过键检索单个项时, 将返回 flashdata 项。但是在从 session 中获取所有数据时不会返回 flashdata。

但是, 如果你想确定正在读取 "flashdata" (而不是任何其他数据), 也可以使用 `getFlashdata()` 方法:

```
<?php

$session->getFlashdata('item');
```

备注: 如果找不到该项, `getFlashdata()` 方法将返回 null。

当然, 如果你想检索所有现有的 flashdata:

```
<?php

$session->getFlashdata();
```

如果你发现需要通过其他请求保留 flashdata 变量, 可以使用 `keepFlashdata()` 方法。你可以保留单个项或 flashdata 项数组。

```
<?php

$session->keepFlashdata('item');
$session->keepFlashdata(['item1', 'item2', 'item3']);
```

Tempdata

CodeIgniter 还支持 “tempdata” , 也就是在特定过期时间后自动删除的 session 数据。在值过期或 session 过期或删除后, 该值将自动删除。

与 flashdata 类似,tempdata 变量由 CodeIgniter session 处理程序内部管理。

要将现有项标记为 “tempdata” , 只需传递其键和过期时间 (以秒为单位!) 给 `markAsTempdata()` 方法:

```
<?php

// 'item' will be erased after 300 seconds
$session->markAsTempdata('item', 300);
```

你可以通过两种方式标记多个项为 tempdata, 这取决于是否希望它们都具有相同的过期时间:

```
<?php

// Both 'item' and 'item2' will expire after 300 seconds
$session->markAsTempdata(['item', 'item2'], 300);

// 'item' will be erased after 300 seconds, while 'item2'
// will do so after only 240 seconds
$session->markAsTempdata([
    'item'  => 300,
    'item2' => 240,
]);
```

添加 tempdata:

```
<?php

$_SESSION['item'] = 'value';
$session->markAsTempdata('item', 300); // Expire in 5 minutes
```

或者也可以使用 `setTempdata()` 方法:

```
<?php
```

(续下页)

(接上页)

```
$session->setTempdata('item', 'value', 300);
```

你也可以向 `setTempdata()` 传递数组:

```
<?php

$tempdata = ['newuser' => true, 'message' => 'Thanks for joining!'];
$session->setTempdata($tempdata, null, $expire);
```

备注: 如果省略过期时间或设置为 0, 将使用默认的 300 秒 (5 分钟) 的生存时间。

要读取 tempdata 变量, 再次只需通过 `$_SESSION` 超全局数组访问它:

```
<?php

$item = $_SESSION['item'];
```

重要: `get()` 方法在通过键检索单个项时, 将返回 tempdata 项。但是在从 session 中获取所有数据时不会返回 tempdata。

或者如果你想确定正在读取 "tempdata" (而不是任何其他数据), 也可以使用 `getTempdata()` 方法:

```
<?php

$session->getTempdata('item');
```

备注: 如果找不到该项, `getTempdata()` 方法将返回 null。

当然, 如果你想检索所有现有的 tempdata:

```
<?php
```

(续下页)

(接上页)

```
$session->getTempdata();
```

如果你需要在过期之前删除 tempdata 值, 可以直接从 `$_SESSION` 数组中取消设置它:

```
<?php

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 请求上失效), 所以如果你打算在同一请求中重用相同的键, 你会想使用 `removeTempdata()`:

```
<?php

$session->removeTempdata('item');
```

关闭一个 Session

close()

在 4.4.0 版本加入.

在不再需要当前 Session 时，可以使用 `close()` 方法手动关闭 Session：

```
<?php

$session->close();
```

你不必手动关闭 Session，PHP 会在脚本终止后自动关闭它。但是，由于 Session 数据被锁定以防止并发写入，因此一次只能有一个请求操作 Session。通过在所有对 Session 数据的更改完成后立即关闭 Session，可以提高网站性能。

此方法的工作方式与 PHP 的 session_write_close() 函数完全相同。

销毁一个 Session

destroy()

要清除当前 session(例如在退出登录时), 可以使用类库的 `destroy()` 方法:

```
<?php

$session->destroy();
```

此方法的工作方式与 PHP 的 session_destroy() 函数完全相同。

这必须是在同一请求中进行的最后一个与 Session 相关的操作。所有 Session 数据（包括 flashdata 和 tempdata）将被永久销毁。

备注: 你不必在常规代码中调用此方法。清理 Session 数据而不是销毁会话。

stop()

自 4.3.5 版本弃用.

Session 类还有 `stop()` 方法。

警告: 在 v4.3.5 之前, 由于一个错误, 此方法不会销毁 session。

从 v4.3.5 开始, 此方法已被修改为销毁 session。但是, 由于它与 `destroy()` 方法完全相同, 已被弃用。请使用 `destroy()` 方法。

访问 Session 元数据

在 CodeIgniter 2 中, 默认情况下 session 数据数组包含 4 个项: ‘session_id’、’ip_address’、’user_agent’、’ last_activity’。

这是由于 session 工作方式的特殊性, 但现在在我们的新实现中不再是必需的。但是, 你的应用程序可能依赖于这些值, 所以这里提供了访问它们的替代方法:

- session_id: `$session->session_id` 或 `session_id()` (PHP 的内置函数)

- ip_address: `$_SERVER['REMOTE_ADDR']`
- user_agent: `$_SERVER['HTTP_USER_AGENT']` (session 不使用它)
- last_activity: 取决于存储, 没有直接的方法。抱歉!

Session 首选项

CodeIgniter 通常会使一切正常工作。但是,Session 是任何应用程序中一个非常敏感的组件, 因此必须谨慎配置。请花时间考虑所有选项及其影响。

备注: 自 v4.3.0 起, 添加了新的 **app/Config/Session.php** 文件。之前,Session 首选项在你的 **app/Config/App.php** 文件中。

你会在 **app/Config/Session.php** 文件中找到以下与 Session 相关的首选项:

首选项	默认值	选项	描述
driver	CodeIgniter\Session\Handlers\FileHandler	CodeIgniter\Session\Handlers\FileHandler CodeIgniter\Session\Handlers\DatabaseHandler CodeIgniter\Session\Handlers\MemcachedHandler CodeIgniter\Session\Handlers\RedisHandler CodeIgniter\Session\Handlers\ArrayHandler	要使用的 session 存储驱动程序。
cookieName	ci_session	[A-Za-z_-] 字符	用于 session cookie 的名称。
expiration	7200 (2 小时)	秒数 (整数)	你希望 session 持续的秒数。如果你希望一个不过期的 session(直到浏览器关闭), 请将值设置为零:0
savePath	nil	无	根据所使用的驱动程序指定存储位置。
matchIP	false	true/false(布尔值)	从 session cookie 读取时是否验证用户的 IP 地址。请注意, 某些 ISP 会动态更改 IP, 因此如果你需要一个不过期的 session, 你可能会将此设置为 false。
timeToUpdate	300	秒数 (整数)	此选项控制 session 类重新生成自身和创建新的 session ID 的频率。将其设置为 0 将禁用 session ID 重新生成。
regenerateDestroy	false	true/false(布尔值)	是否在自动重新生成 session ID 时销毁与旧 session ID 关联的数据。将其设置为 false 时, 稍后数据将由垃圾收集器删除。

备注: 作为最后的手段, 如果上述任何内容都未配置,Session 库将尝试获取 PHP 的与 session 相关的 INI 设置, 以及 CodeIgniter 3 设置, 如 'sess_expire_on_close'。但是, 你永远不应该依赖这种行为, 因为它可能会导致意外结果或在未来更改。请正确配置一切。

备注: 如果 `expiration` 设置为 `0`, 则将原封不动地使用 PHP 在会话管理中设置的 `session.gc_maxlifetime` 设置 (通常默认值为 `1440`)。根据需要，这需要在 `php.ini` 或通过 `ini_set()` 进行更改。

此外, 在你的 **app/Config/Cookie.php** 文件中使用了以下配置值用于 Session cookie:

Preference	Default	Description
domain	''	Session 适用的域
path	/	Session 适用的路径
secure	false	是否仅在加密连接 (HTTPS) 上创建 session cookie
sameSite	Lax	Session cookie 的 SameSite 设置

备注: `httponly` 设置（在 **app/Config/Cookie.php** 中）不会对 session 产生影响。出于安全原因，HttpOnly 参数始终启用。另外，完全忽略了 `Config\Cookie::$prefix` 设置。

Session 驱动程序

如前所述，Session 库提供了 5 个处理程序或存储引擎可以使用:

- CodeIgniter\Session\Handlers\FileHandler
- CodeIgniter\Session\Handlers\DatabaseHandler
- CodeIgniter\Session\Handlers\MemcachedHandler
- CodeIgniter\Session\Handlers\RedisHandler
- CodeIgniter\Session\Handlers\ArrayHandler

初始化 session 时，如果不指定，将使用 `FileHandler`，因为这是最安全的选择，并且预期它可以在任何环境中使用 (几乎每种环境都有文件系统)。

然而，如果你愿意，可以通过 **app/Config/Session.php** 文件中的 `$driver` 设置来选择任何其他驱动。但请记住，每个驱动都有不同的注意事项，所以在你做出选择之前，确保你熟悉它们（如下所示）。

备注: ArrayHandler 在测试时使用, 并将所有数据存储在 PHP 数组中, 同时防止数据持久化。

FileHandler 驱动程序 (默认)

‘FileHandler’驱动程序使用你的文件系统来存储 session 数据。

可以肯定地说，它的工作方式与 PHP 自带的默认 Session 实现完全一样。但如果你认为这是一个重要的细节，实际上，它并不是相同的代码，并且有一些限制（以及优势）。

更具体地说，它不支持 PHP 的 session.save_path 中使用的目录级别和模式格式。并且，出于安全考虑，大部分选项都是硬编码的。相反，只支持使用 `$savePath` 设置的绝对路径。

另一件你需要知道的重要事情是，确保你不要使用公开可读或共享的目录来存储你的 session 文件。只有你可以访问你选择的 *savePath* 目录的内容。否则，任何人都可以查看并窃取 Session 数据（这也被称为“会话固定”攻击）。

在类 UNIX 操作系统上, 这通常通过使用 *chmod* 命令对该目录设置 0700 模式权限来实现, 它仅允许目录所有者在其上执行读写操作。但是要小心, 因为 运行脚本的系统用户通常不是你自己, 而是类似‘www-data’的用户, 所以只设置这些权限可能会中断你的应用程序。

相反, 你应该执行类似以下操作, 这取决于你的环境:

```
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/
```

奖励提示

你们中一些人可能会选择另一个 session 驱动程序, 因为文件存储通常较慢。这只有一半是真的。

一个非常基本的测试可能会让你误以为 SQL 数据库更快，但在 99% 的情况下，这种观点只有在你的当前 Session 数量很少时才成立。随着 Session 数量的增加和服务器负载的提升——这才是真正重要的时刻——文件系统将始终优于几乎所有的关系型数据库设置。

另外，如果性能是你唯一的关注点，你可能需要研究使用 tmpfs，它可以使你的 session 飞快。

DatabaseHandler 驱动程序

重要: 由于其他平台缺乏咨询锁机制，因此官方仅支持 MySQL 和 PostgreSQL 数据库。在不使用锁的情况下使用 Session 可能会导致各种问题，特别是在大量使用 AJAX 的情况下。如果你遇到性能问题，在处理完 Session 数据后，请使用*close()* 方法。

‘DatabaseHandler’驱动程序使用 MySQL 或 PostgreSQL 等关系数据库来存储会话。这对许多用户来说是一个流行的选择, 因为它允许开发人员轻松访问应用程序中的 session 数据——它只是数据库中的另一个表。

然而，有一个限制：你不能使用持久连接。

配置 DatabaseHandler

设置表名

为了使用‘DatabaseHandler’session 驱动程序, 还必须创建我们已经提到的表, 然后将其设置为你的 `$savePath` 值。例如, 如果你想使用‘ci_sessions’作为表名, 你将执行以下操作:

```
<?php

namespace Config;
```

(续下页)

(接上页)

```
use CodeIgniter\Config\BaseConfig;
use CodeIgniter\Session\Handlers\FileHandler;

class Session extends BaseConfig
{
    // ...
    public string $driver = 'CodeIgniter\Session\Handlers\
↪DatabaseHandler';

    // ...
    public string $savePath = 'ci_sessions';

    // ...
}
```

创建数据库表

然后当然，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" inet NOT NULL,
    "timestamp" bigint DEFAULT 0 NOT NULL,
    "data" text DEFAULT '' NOT NULL
);
```

(续下页)

(接上页)

```
CREATE INDEX "ci_sessions_timestamp" ON "ci_sessions" ("timestamp");
```

备注: `id` 值包含 session cookie 名称 (`Config\Session::$cookieName`) 和 session ID 以及一个分隔符。根据需要应增加它, 例如在使用长 session ID 时。

添加主键

根据你的 $matchIP 设置，你还需要添加一个主键。以下示例适用于 MySQL 和 PostgreSQL:

```
// 当 $matchIP = true 时
ALTER TABLE ci_sessions ADD PRIMARY KEY (id, ip_address);

// 当 $matchIP = false 时
ALTER TABLE ci_sessions ADD PRIMARY KEY (id);

// 删除先前创建的主键(更改设置时使用)
ALTER TABLE ci_sessions DROP PRIMARY KEY;
```

重要: 如果你没有添加正确的主键，可能会出现以下错误:

```
Uncaught mysqli_sql_exception: Duplicate entry 'ci_session:***' for␣
↪key 'ci_sessions.PRIMARY'
```

更改数据库组

默认情况下使用默认数据库组。你可以通过更改 **app/Config/Session.php** 文件中的 `$DBGroup` 属性为要使用的组的名称来更改数据库组:

```
<?php
```

(续下页)

(接上页)

```
namespace Config;

use CodeIgniter\Config\BaseConfig;
use CodeIgniter\Session\Handlers\FileHandler;

class Session extends BaseConfig
{
    // ...
    public ?string $DBGroup = 'groupName';
}
```

使用命令设置数据库表

当然，如果你不想手动执行所有这些操作，可以使用 cli 中的 `php spark make:migration --session` 命令为你生成迁移文件:

```
php spark make:migration --session
php spark migrate
```

此命令将考虑 `$savePath` 和 `$matchIP` 设置并生成代码。

RedisHandler 驱动程序

备注: 由于 Redis 没有暴露锁机制，因此该驱动的锁是通过一个单独的值来模拟的，该值最多保留 300 秒。

备注: 从 v4.3.2 开始，你可以使用 **TLS** 协议连接 Redis。

Redis 是一个通常用于缓存且以高性能而著称的存储引擎, 这也可能是你使用 'RedisHandler' session 驱动程序的原因。

缺点是它不像关系型数据库那样普遍，并且需要安装 phpredis PHP 扩展在你的系统上，而这个扩展并不随 PHP 一起捆绑提供。很可能，你只有在已经熟悉 Redis 并且还出于其他目的使用它的情况下，才会使用 RedisHandler 驱动。

配置 RedisHandler

就像‘FileHandler’和‘DatabaseHandler’驱动一样，你也必须通过 `$savePath` 设置来配置你的 Session 存储位置。这里的格式有点不同且复杂。最好是通过 *phpredis* 扩展的 README 文件来解释，因此我们将直接提供一个链接：

https://github.com/phpredis/phpredis

重要： CodeIgniter 的 Session 库不使用实际的‘redis’`session.save_handler`。在上面的链接中 **仅**注意路径格式。

但是, 对于最常见的情况, 一个简单的 `host:port` 对应关系应该就足够了:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
use CodeIgniter\Session\Handlers\FileHandler;

class Session extends BaseConfig
{
    // ...
    public string $driver = 'CodeIgniter\Session\Handlers\
↪RedisHandler';

    // ...
    public string $savePath = 'tcp://localhost:6379';

    // ...
}
```

从 v4.5.0 开始，你可以使用 Redis ACL（用户名和密码）：

```
public string $savePath = 'tcp://localhost:6379?auth[user]=username&
↪auth[pass]=password';
```

备注： 从 v4.5.0 开始，获取锁的间隔时间（`$lockRetryInterval`）和重试次数

($lockMaxRetries) 是可配置的。

MemcachedHandler 驱动程序

备注: 由于 Memcached 没有公开锁定机制, 因此通过单独保留 300 秒的额外值来模拟此驱动程序的锁。

'MemcachedHandler' 驱动程序几乎与 'RedisHandler' 驱动程序的所有属性相同, 可能仅在可用性方面有所不同, 因为 PHP 的 Memcached 扩展通过 PECL 分发, 一些 Linux 发行版将其作为易于安装的包。

除此之外, 如果没有任何故意的偏见针对 Redis, 关于 Memcached 就没有太多不同的可说的——它也是一个流行的产品, 以其速度而闻名。

但是, 值得注意的是,Memcached 所做的唯一保证是, 设置值 X 在 Y 秒后过期将导致在 Y 秒过去后删除该值 (但不一定保证不会比该时间过期更早)。这种情况非常罕见, 但应该考虑到它可能导致 session 丢失。

配置 MemcachedHandler

这里的 $savePath 格式相当简单, 只是一个 host:port 对:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
use CodeIgniter\Session\Handlers\FileHandler;

class Session extends BaseConfig
{
    // ...
    public string $driver = 'CodeIgniter\Session\Handlers\
↪MemcachedHandler';

    // ...
```

(续下页)

(接上页)

```
    public string $savePath = 'localhost:11211';

    // ...
}
```

奖励提示

也支持多服务器配置, 以可选的 *weight* 参数作为第三个冒号分隔 (`:weight`) 值, 但我们必须注意, 我们还没有测试这一特性的可靠性。

如果你想要实验此功能 (自负风险), 只需用逗号分隔多个服务器路径:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
use CodeIgniter\Session\Handlers\FileHandler;

class Session extends BaseConfig
{
    // ...

    // localhost will be given higher priority (5) here,
    // compared to 192.0.2.1 with a weight of 1.
    public string $savePath = 'localhost:11211:5,192.0.2.1:11211:1';

    // ...
}
```

7.1.15 限速器

- 概览
- 速率限制
  - 代码

- 应用过滤器
- 类参考

限速器类提供了一种非常简单的方法来限制在一定时间内执行的活动次数。这最常用于对 API 进行速率限制, 或者限制用户尝试表单的次数以帮助防止暴力攻击。该类本身可以用于你需要根据一定时间间隔内的动作进行限速的任何事物。

概览

限速器实现了 令牌桶 算法的简化版本。这基本上将你想要的每个操作视为一个桶。当你调用 `check()` 方法时, 你告诉它桶的大小, 它可以容纳的令牌数和时间间隔。默认情况下, 每个 `check()` 调用会使用可用令牌中的 1 个。让我们通过一个示例来说明这一点。

假设我们想要每秒钟发生一次操作。首次调用限速器将如下所示。第一个参数是桶的名称, 第二个参数是桶容纳的令牌数, 第三个是桶重新填充所需的时间量:

```
<?php

$throttler = service('throttler');
$throttler->check($name, 60, MINUTE);
```

这里我们使用全局常量 之一来表示时间, 使其更具可读性。这表示该桶允许每分钟 60 次操作, 或每秒 1 次操作。

假设第三方脚本尝试重复点击一个 URL。起初, 它将能够在不到 1 秒钟内使用这 60 个令牌。然而, 在那之后, 限速器只允许每秒执行一个操作, 可能会减慢请求的速度, 以至于攻击不再值得。

备注: 要使限速器类起作用, 必须将缓存库设置为使用 dummy 之外的处理程序。为获得最佳性能, 建议使用内存缓存, 如 Redis 或 Memcached。

速率限制

限速器类本身不执行任何速率限制或请求限制, 但它是使速率限制起作用的关键。提供了一个示例过滤器, 它实现了每个 IP 地址每秒一个请求的非常简单的速率限制。在这里, 我们将介绍它的工作原理, 以及如何在你的应用程序中设置和开始使用它。

代码

你可以在 **app/Filters/Throttle.php** 中创建自己的限速器过滤器, 如下所示:

```
<?php

namespace App\Filters;

use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;

class Throttle implements FilterInterface
{
    /**
     * This is a demo implementation of using the Throttler class
     * to implement rate limiting for your application.
     *
     * @param list<string>|null $arguments
     *
     * @return ResponseInterface|void
     */
    public function before(RequestInterface $request, $arguments =␣
↪null)
    {
        $throttler = service('throttler');

        // Restrict an IP address to no more than 1 request
        // per second across the entire site.
        if ($throttler->check(md5($request->getIPAddress()), 60,␣
↪MINUTE) === false) {
            return service('response')->setStatusCode(429);
```

(续下页)

(接上页)

```
        }
    }

    /**
     * We don't have anything to do here.
     *
     * @param list<string>|null $arguments
     *
     * @return void
     */
    public function after(RequestInterface $request,␣
↪ResponseInterface $response, $arguments = null)
    {
        // ...
    }
}
```

运行时, 此方法首先获取限速器的一个实例。接下来, 它使用 IP 地址作为桶名称, 并将其设置为限制每秒一个请求。如果限速器拒绝检查, 返回 false, 那么我们返回一个响应状态代码设置为 429 - 太多尝试的响应, 并且在它到达控制器之前脚本执行就结束了。这个例子将基于针对站点的所有请求对单个 IP 地址进行限制, 而不是针对每个页面。

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, 这最适合仅应用于 POST 请求, 尽管 API 可能希望限制用户发出的每个请求。为了将其应用于传入请求, 你需要编辑 **app/Config/Filters.php** 并首先为过滤器添加一个别名:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public $aliases = [
```

(续下页)

(接上页)

```
        // ...
        'throttle' => \App\Filters\Throttle::class,
    ];

    // ...
}
```

接下来, 我们将其分配给站点上发出的所有 POST 请求:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Filters extends BaseConfig
{
    public $methods = [
        'POST' => ['throttle'],
    ];

    // ...
}
```

> **警告:** 如果使用 `$methods` 过滤器, 则应禁用自动路由 (遗留), 因为自动路由（传统版）允许使用任何 HTTP 方法访问控制器。使用你不期望的方法访问控制器可能会绕过过滤器。

就是这样。现在站点上的所有 POST 请求都必须受到速率限制。

类参考

check(*string $key*, *int $capacity*, *int $seconds*[, *int $cost = 1*])

参数

- **$key** (string) –桶的名称
- **$capacity** (int) –桶容纳的令牌数
- **$seconds** (int) –桶完全填充所需的秒数
- **$cost** (int) –此操作消耗的令牌数

返回

如果可以执行操作则为 true, 否则为 false

返回类型

bool

检查桶中是否还有任何令牌, 或者在分配的时间限制内是否使用了太多。如果成功, 每次检查将按 $cost 扣除可用令牌。

getTokentime()

返回

直到另一个令牌可用的秒数。

返回类型

整数

在 check() 运行并返回 false 后, 可以使用此方法来确定新的令牌应该可用并可以再次尝试该操作的时间。在这种情况下, 强制等待时间最小为 1 秒。

remove(*string $key*) → self

参数

- **$key** (string) –桶的名称

返回

$this

返回类型

self

移除并重置桶。如果桶不存在也不会失败。

7.1.16 时间与日期

CodeIgniter 提供了一个完全本地化、不可变的日期/时间类，该类基于 PHP 的 DateTimeImmutable 类构建，但利用 Intl 扩展的功能来转换时区时间，并针对不同区域设置正确显示输出。这个类就是 `Time` 类，位于 `CodeIgniter\I18n` 命名空间。

备注： 由于 Time 类继承自 `DateTimeImmutable`，如果你需要本类未提供的功能，可以在 DateTimeImmutable 类本身中找到这些功能。

备注： 在 v4.3.0 之前，Time 类继承自 `DateTime`，某些继承的方法会改变当前对象状态。此问题已在 v4.3.0 中修复。如果需要向后兼容旧的 Time 类，可以暂时使用已弃用的 `TimeLegacy` 类。

- 实例化
  - *now()*
  - *parse()*
  - *today()*
  - *yesterday()*
  - *tomorrow()*
  - *createFromDate()*
  - *createFromTime()*
  - *create()*
  - *createFromFormat()*
  - *createFromTimestamp()*
  - *createFromInstance()*
  - *toDateTime()*
- 显示值
  - *toLocalizedString()*

- *toDateTimeString()*
- *toDateString()*
- *toTimeString()*
- *humanize()*

• 处理独立值
- *Getter 方法*
- *Setter 方法*
- 修改值
- 比较两个时间
- 查看差异

实例化

创建新的 Time 实例有几种方式。第一种是像普通类一样创建新实例。

这种方式下，你可以传入表示预期时间的字符串。这可以是 PHP 的 DateTimeImmutable 构造函数能解析的任何字符串。详情参见 支持的日期和时间格式。

```
<?php

use CodeIgniter\I18n\Time;

$myTime = new Time('2024-01-01');
$myTime = new Time('2024-01-01 12:00:00');
$myTime = new Time('now');
$myTime = new Time('+3 week');
```

你可以在第二个和第三个参数中分别传入表示时区和区域设置的字符串。时区可以是 PHP 的 DateTimeZone 类支持的任何时区。区域设置可以是 PHP 的 Locale 类支持的任何区域。如果未提供区域或时区，将使用应用默认设置。

```
<?php

use CodeIgniter\I18n\Time;
```

(续下页)

(接上页)

```
$myTime = new Time('now', 'America/Chicago', 'en_US');
```

now()

Time 类有几个辅助方法来实例化该类。第一个是 `now()` 方法，返回设置为当前时间的新实例。你可以在第二个和第三个参数中分别传入时区和区域设置的字符串。如果未提供区域或时区，将使用应用默认设置。

```
<?php

use CodeIgniter\I18n\Time;

$myTime = Time::now('America/Chicago', 'en_US');
```

parse()

这个辅助方法是默认构造函数的静态版本。它接受 DateTimeImmutable 构造函数可接受的字符串作为第一个参数，时区作为第二个参数，区域设置作为第三个参数：

```
<?php

use CodeIgniter\I18n\Time;

$myTime = Time::parse('next Tuesday', 'America/Chicago', 'en_US');
```

today()

返回日期设置为当前日期、时间设置为午夜的新实例。第一个和第二个参数接受时区和区域设置的字符串：

```
<?php

use CodeIgniter\I18n\Time;
```

(续下页)

(接上页)

```
$myTime = Time::today('America/Chicago', 'en_US');
```

yesterday()

返回日期设置为昨天日期、时间设置为午夜的新实例。第一个和第二个参数接受时区和区域设置的字符串：

```
<?php

use CodeIgniter\I18n\Time;

$myTime = Time::yesterday('America/Chicago', 'en_US');
```

tomorrow()

返回日期设置为明天日期、时间设置为午夜的新实例。第一个和第二个参数接受时区和区域设置的字符串：

```
<?php

use CodeIgniter\I18n\Time;

$myTime = Time::tomorrow('America/Chicago', 'en_US');
```

createFromDate()

给定 **年**、**月**、**日**的独立输入，将返回新实例。如果未提供任何参数，将使用当前年、月、日。第四个和第五个参数接受时区和区域设置的字符串：

```
<?php

use CodeIgniter\I18n\Time;

$today       = Time::createFromDate();      // Uses current year,␣
```

(续下页)

(接上页)

```
↪month, and day
$anniversary = Time::createFromDate(2018); // Uses current month␣
↪and day
$date        = Time::createFromDate(2018, 3, 15, 'America/Chicago',
↪'en_US');
```

createFromTime()

类似于 `createFromDate()`，但仅关注 **小时**、**分钟**和 **秒**。使用当前日期作为 Time 实例的日期部分。第四个和第五个参数接受时区和区域设置的字符串：

```
<?php

use CodeIgniter\I18n\Time;

$lunch  = Time::createFromTime(11, 30);      // 11:30 am today
$dinner = Time::createFromTime(18, 00, 00); // 6:00 pm today
$time   = Time::createFromTime($hour, $minutes, $seconds, $timezone,
↪ $locale);
```

create()

前两个方法的组合，将 **年**、**月**、**日**、**小时**、**分钟**和 **秒**作为独立参数。任何未提供的值将使用当前日期和时间。第四个和第五个参数接受时区和区域设置的字符串：

```
<?php

use CodeIgniter\I18n\Time;

$time = Time::create($year, $month, $day, $hour, $minutes, $seconds,
↪ $timezone, $locale);
```

createFromFormat()

这是 DateTimeImmutable 同名方法的替代方法。允许同时设置时区，并返回 `Time` 实例而非 DateTimeImmutable：

```
<?php

use CodeIgniter\I18n\Time;

$time = Time::createFromFormat('j-M-Y', '15-Feb-2009', 'America/
↪Chicago');
```

createFromTimestamp()

此方法接受 UNIX 时间戳，并可选择时区和区域设置来创建新的 Time 实例：

```
<?php

use CodeIgniter\I18n\Time;

$time = Time::createFromTimestamp(1501821586, 'America/Chicago',
↪'en_US');
```

如果未显式传递时区，则返回具有 **UTC** 的 Time 实例。

备注： 我们建议始终使用 2 个参数调用 `createFromTimestamp()` （即显式传递时区），除非使用 UTC 作为默认时区。

备注： 在 v4.4.6 到 v4.6.0 之前，当未指定时区时，此方法返回具有默认时区的 Time 实例。

---

createFromInstance()

当处理其他提供 DateTime 实例的库时，可以使用此方法将其转换为 Time 实例，并可选择设置区域设置。时区将自动从传入的 DateTime 实例中确定：

```
<?php

use CodeIgniter\I18n\Time;

$dt   = new \DateTime('now');
$time = Time::createFromInstance($dt, 'en_US');
```

toDateTime()

虽然不是实例化方法，但此方法是 **instance** 方法的反向操作，允许将 Time 实例转换为 DateTime 实例。这会保留时区设置，但丢失区域设置，因为 DateTime 不感知区域：

```
<?php

use CodeIgniter\I18n\Time;

$datetime = Time::toDateTime();
```

显示值

由于 Time 类继承自 DateTimeImmutable，你可以使用其提供的所有输出方法，包括 `format()` 方法。但是 DateTimeImmutable 方法不提供本地化结果。Time 类提供了许多辅助方法来显示值的本地化版本。

toLocalizedString()

这是 DateTimeImmutable 的 `format()` 方法的本地化版本。不过，你必须使用 IntlDateFormatter 类可接受的值，而不是你可能熟悉的格式值。完整值列表可在此处找到。

```
<?php
```

(续下页)

(接上页)

```
use CodeIgniter\I18n\Time;

// Locale: en
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toLocalizedString('MMM d, yyyy'); // March 9, 2016

// Locale: fa
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toLocalizedString('MMM d, yyyy'); // ???? ?, ????
```

toDateTimeString()

这是三个辅助方法中的第一个，用于使用 IntlDateFormatter 而无需记住其值。这将返回格式为 (`Y-m-d H:i:s`) 的本地化字符串：

```
<?php

use CodeIgniter\I18n\Time;

// Locale: en
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toDateTimeString(); // 2016-03-09 12:00:00

// Locale: fa
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toDateTimeString(); // ????-??-?? ??:??:??
```

toDateString()

仅显示值的本地化日期部分：

```
<?php

use CodeIgniter\I18n\Time;

// Locale: en
```

(续下页)

(接上页)

```
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toDateString(); // 2016-03-09

// Locale: fa
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toDateString(); // ????-??-??
```

toTimeString()

仅显示值的本地化时间部分：

```
<?php

use CodeIgniter\I18n\Time;

// Locale: en
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toTimeString(); // 12:00:00

// Locale: fa
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');
echo $time->toTimeString(); // ??:??:??
```

humanize()

此方法返回一个字符串，以易于理解的人类可读格式显示当前日期/时间与实例之间的差异。可以生成如“3 hours ago”、” in 1 month” 等字符串：

```
<?php

use CodeIgniter\I18n\Time;

// Assume current time is: March 10, 2017 (America/Chicago)
$time = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');

echo $time->humanize(); // 1 year ago
```

显示的确切时间由以下方式确定：

时间差	结果
1 年 < $time < 2 年	in 1 year / 1 year ago
1 个月 < $time < 1 年	in 6 months / 6 months ago
7 天 < $time < 1 个月	in 3 weeks / 3 weeks ago
今天 < $time < 7 天	in 4 days / 4 days ago
$time == 昨天 / 明天	Tomorrow / Yesterday
59 分钟 < $time < 1 天	in 2 hours / 2 hours ago
现在 < $time < 1 小时	in 35 minutes / 35 minutes ago
$time == 现在	Now

结果字符串来自语言文件 **system/Language/en/Time.php**。如需覆盖，请创建 **app/Language/{locale}/Time.php**。

处理独立值

Time 对象提供多个方法用于获取和设置现有实例的独立项（如年、月、小时等）。通过以下方法获取的所有值都将完全本地化，并尊重创建 Time 实例时使用的区域设置。

所有以下 `getX()` 和 `setX()` 方法也可以像类属性一样使用。因此，像 `getYear()` 这样的方法调用也可以通过 `$time->year` 等方式访问。

Getter 方法

存在以下基本 getter 方法：

```
<?php

use CodeIgniter\I18n\Time;

$time = Time::parse('August 12, 2016 4:15:23pm');

// The output may vary based on locale.
echo $time->getYear();    // '2016'
echo $time->getMonth();   // '8'
echo $time->getDay();     // '12'
```

(续下页)

(接上页)

```
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'
```

此外，还有一些方法可提供有关日期的额外信息：

```
<?php

use CodeIgniter\I18n\Time;

$time = Time::parse('August 12, 2016 4:15:23pm');

// The output may vary based on locale.
echo $time->getDayOfWeek();   // '6'
echo $time->getDayOfYear();   // '225'
echo $time->getWeekOfMonth(); // '2'
echo $time->getWeekOfYear();  // '33'
echo $time->getTimestamp();   // 1471018523 - UNIX timestamp␣
↪(locale independent)
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年龄（以年为单位）。非常适合根据生日检查年龄：

```
<?php

use CodeIgniter\I18n\Time;

$time = Time::parse('5 years ago');

echo $time->getAge(); // 5
echo $time->age;      // 5
```

getDST()

根据 Time 实例当前是否处于夏令时返回布尔值 true/false：

```
<?php

use CodeIgniter\I18n\Time;

echo Time::createFromDate(2012, 1, 1)->getDst(); // false
echo Time::createFromDate(2012, 9, 1)->dst;      // true
```

getLocal()

如果 Time 实例与时区与应用程序当前运行时区相同，则返回布尔值 true：

```
<?php

use CodeIgniter\I18n\Time;

echo Time::now()->getLocal();            // true
echo Time::now('Europe/London')->local; // false
```

getUtc()

如果 Time 实例处于 UTC 时间，则返回布尔值 true：

```
<?php

use CodeIgniter\I18n\Time;

echo Time::now('America/Chicago')->getUtc(); // false
echo Time::now('UTC')->utc;                  // true
```

getTimezone()

返回一个新的 DateTimeZone 对象，该对象设置为 Time 实例的时区：

```
<?php

use CodeIgniter\I18n\Time;

$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：

```
<?php

use CodeIgniter\I18n\Time;

echo Time::now('America/Chicago')->getTimezoneName(); // America/
↪Chicago
echo Time::now('Europe/London')->timezoneName;        // Europe/
↪London
```

Setter 方法

存在以下基本 setter 方法。如果设置的任何值超出范围，将抛出 `InvalidArgumentExeption`。

备注： 所有 setter 方法将返回新的 Time 实例，原始实例保持不变。

备注： 所有 setter 方法在值超出范围时将抛出 InvalidArgumentException。

```
<?php

$time = $time->setYear(2017);
$time = $time->setMonth(4);        // April
$time = $time->setMonth('April');
$time = $time->setMonth('Feb');    // February
$time = $time->setDay(25);
$time = $time->setHour(14);        // 2:00 pm
$time = $time->setMinute(30);
$time = $time->setSecond(54);
```

setTimezone()

将时间从当前时区转换到新时区：

```
<?php

use CodeIgniter\I18n\Time;

$time  = Time::parse('13 May 2020 10:00', 'America/Chicago');
$time2 = $time->setTimezone('Europe/London'); // Returns new␣
↪instance converted to new timezone

echo $time->getTimezoneName();  // American/Chicago
echo $time2->getTimezoneName(); // Europe/London
```

(续下页)

(接上页)

```
echo $time->toDateTimeString();  // 2020-05-13 10:00:00
echo $time2->toDateTimeString(); // 2020-05-13 18:00:00
```

setTimestamp()

返回日期设置为新时间戳的新实例：

```
<?php

use CodeIgniter\I18n\Time;

// The Application Timezone is "America/Chicago".

$time  = Time::parse('May 10, 2017');
$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
```

备注： 在 v4.6.0 之前，由于存在 bug，此方法可能返回不正确的日期/时间。详情参见升级指南。

修改值

以下方法允许你通过向当前 Time 添加或减去值来修改日期。这不会修改现有 Time 实例，而是返回新实例。

```
<?php

$time = $time->addSeconds(23);
$time = $time->addMinutes(15);
$time = $time->addHours(12);
$time = $time->addDays(21);
$time = $time->addMonths(14);
```

(续下页)

(接上页)

```
$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);
```

比较两个时间

以下方法允许你将一个 Time 实例与另一个进行比较。所有比较在转换到 UTC 后进行，以确保不同时区能正确响应。

equals()

确定传入的日期时间是否等于当前实例。此处的”相等”意味着它们代表同一时刻，不要求处于相同时区，因为两个时间都会转换为 UTC 进行比较：

```
<?php

use CodeIgniter\I18n\Time;

$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
```

被测试值可以是 Time 实例、DateTime 实例，或包含完整日期时间的字符串（需能被新 DateTime 实例理解）。当第一个参数传递字符串时，可以在第二个参数中传递时区字符串。如果未提供时区，将使用系统默认值：

```
<?php

$time1->equals('January 11, 2017 03:50:00', 'Europe/London'); //␣
```

(续下页)

(接上页)

```
↪true
```

sameAs()

此方法与 `equals()` 相同，但仅当日期、时间和时区都完全相同时才返回 true：

```
<?php

use CodeIgniter\I18n\Time;

$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()

检查传入时间是否早于当前实例。比较基于两个时间的 UTC 版本：

```
<?php

use CodeIgniter\I18n\Time;

$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
```

被测试值可以是 Time 实例、DateTime 实例，或包含完整日期时间的字符串（需能被新 DateTime 实例理解）。当第一个参数传递字符串时，可以在第二个参数中传递时区字符串。如果未提供时区，将使用系统默认值：

```
<?php

$time1->isBefore('March 15, 2013', 'America/Chicago'); // false
```

isAfter()

工作方式与 isBefore() 完全相同，但检查时间是否晚于传入时间：

```
<?php

use CodeIgniter\I18n\Time;

$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 Time 实例，可以使用 difference() 方法，该方法返回一个 CodeIgniter\I18n\TimeDifference 实例。

第一个参数可以是 Time 实例、DateTime 实例或日期/时间字符串。如果第一个参数传递字符串，第二个参数可以是时区字符串：

```
<?php

use CodeIgniter\I18n\Time;

$time = Time::parse('March 10, 2017', 'America/Chicago');

$diff = $time->difference(Time::now());
$diff = $time->difference(new \DateTime('July 4, 1975', 'America/
↪Chicago'));
```

(续下页)

(接上页)

```
$diff = $time->difference('July 4, 1975 13:32:05', 'America/Chicago
↪');
```

获得 TimeDifference 实例后，可以使用多个方法获取两个时间差异的信息。如果差异时间在过去，返回值为负；如果在未来则为正：

```
<?php

use CodeIgniter\I18n\Time;

$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
```

备注： 在 v4.4.7 之前，Time 始终在比较前将时区转换为 UTC。当包含因夏令时 (DST) 导致天数不同于 24 小时的情况时，可能导致意外结果。

从 v4.4.7 开始，当比较处于相同时区的日期/时间时，直接进行比较而不转换为 UTC：

```
<?php

use CodeIgniter\I18n\Time;

// 31 Mar 2024 - Daylight Saving Time Starts
$current = Time::parse('2024-03-31', 'Europe/Madrid');
$test    = Time::parse('2024-04-01', 'Europe/Madrid');

$diff = $current->difference($test);
```

(续下页)

(接上页)

```
echo $diff->getDays();
// 0 in v4.4.6 or before
// 1 in v4.4.7 or later
```

你可以使用 `getX()` 方法，或像访问属性一样访问计算值：

```
<?php

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()

类似于 Time 的 `humanize()` 方法，此方法返回一个字符串，以人类可读格式显示时间差异，便于理解。可以生成如 “3 hours ago”、” in 1 month” 等字符串。主要区别在于处理最近日期的方式：

```
<?php

use CodeIgniter\I18n\Time;

$current = Time::parse('March 10, 2017', 'America/Chicago');
$test    = Time::parse('March 9, 2016 12:00:00', 'America/Chicago');

$diff = $current->difference($test);

echo $diff->humanize(); // 1 year ago
```

显示的确切时间由以下方式确定：

时间差	结果
1 年 < $time < 2 年	in 1 year / 1 year ago
1 个月 < $time < 1 年	in 6 months / 6 months ago
7 天 < $time < 1 个月	in 3 weeks / 3 weeks ago
今天 < $time < 7 天	in 4 days / 4 days ago
1 小时 < $time < 1 天	in 8 hours / 8 hours ago
1 分钟 < $time < 1 小时	in 35 minutes / 35 minutes ago
$time < 1 分钟	Now

结果字符串来自语言文件 **system/Language/en/Time.php**。如需覆盖，请创建 **app/Language/{locale}/Time.php**。

7.1.17 排版

排版库包含了一些帮助你以语义化方式格式化文本的方法。

- 加载库
- 可用的静态方法

加载库

和 CodeIgniter 中的其他服务一样, 可以通过 `Config\Services` 来加载, 不过通常你不需要手动加载:

```
<?php

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

可用的静态方法

以下方法可用:

autoTypography(*$str*[, *$reduce_linebreaks = false*])

参数

- **$str** (string) –输入字符串
- **$reduce_linebreaks** (bool) –是否把多个连续的空行减少到两个

返回

HTML 格式的适合排版的字符串

返回类型

string

格式化文本, 使其在语义和排版上是正确的 HTML。

使用示例:

```
<?php

$string = $typography->autoTypography($string);
```

备注: 排版格式化可能需要大量处理, 特别是你有很多需要格式化的内容。如果你选择使用这个方法, 你可能需要考虑*caching* 你的页面。

---

formatCharacters(*$str*)

参数

- **$str** (string) –输入字符串

返回

格式化后的字符串

返回类型

string

这个方法主要将双引号和单引号转换为花括号实体, 也会转换破折号、双空格和和号。

使用示例:

```
<?php

$string = $typography->formatCharacters($string);
```

nl2brExceptPre(*$str*)

参数

- **$str**(string) –输入字符串

返回

包含 HTML 格式换行的字符串

返回类型

string

在 <pre> 标签外把换行转换为
 标签。这个方法和原生 PHP 的 nl2br() 函数相同, 只是忽略了 <pre> 标签。

使用示例:

```
<?php

$string = $typography->nl2brExceptPre($string);
```

7.1.18 处理上传的文件

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。这是文件类 的扩展, 因此获得了该类所有的特性。

备注: 这和 CodeIgniter 3 中的文件上传类不太一样。这里提供了一个访问上传文件的原始接口和一些小特性。

- 文件上传表单教程
  - 创建上传表单
  - 成功页面

  - 控制器
  - 路由
  - 上传目录
  - 试一试!
- 访问文件
  - 所有文件
  - 单个文件
  - 多个文件
- 处理文件
  - 验证文件
  - 文件名
  - 其他文件信息
  - 移动文件
  - 存储文件

文件上传表单教程

上传一个文件涉及以下一般过程:

- 显示一个上传表单, 允许用户选择一个文件并上传。
- 当表单提交时, 文件被上传到你指定的目的地。
- 在上传过程中, 会验证文件是否被允许上传, 基于你设置的首选项。
- 一旦上传完成, 用户将看到一个成功的消息。

为了演示这个过程, 这里是一个简单的教程。之后你会找到参考信息。

创建上传表单

使用文本编辑器, 创建一个名为 **upload_form.php** 的表单。在其中放入下面的代码, 并保存到你的 **app/Views** 目录:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Upload Form</title>
</head>
<body>

<?php foreach ($errors as $error): ?>
    <li><?= esc($error) ?></li>
<?php endforeach ?>

<?= form_open_multipart('upload/upload') ?>
    <input type="file" name="userfile" size="20">
    <br><br>
    <input type="submit" value="upload">
</form>

</body>
</html>
```

你会注意到我们使用了一个表单辅助函数来创建表单开标签。文件上传需要一个多部分表单, 所以辅助函数帮我们创建了正确的语法。

你也会注意到我们有一个 `$errors` 变量。这是为了在用户做错事时显示错误信息。

成功页面

使用文本编辑器, 创建一个名为 **upload_success.php** 的页面。在其中放入下面的代码, 并保存到你的 **app/Views** 目录:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>上传表单</title>
```

(续下页)

(接上页)

```
</head>
<body>

<h3>你的文件上传成功!</h3>

<ul>
    <li>名称:<?= esc($uploaded_fileinfo->getBasename()) ?></li>
    <li>大小:<?= esc($uploaded_fileinfo->getSizeByUnit('kb')) ?> KB
↪</li>
    <li>扩展名:<?= esc($uploaded_fileinfo->guessExtension()) ?></li>
</ul>

<p><?= anchor('upload', '上传另一个文件!') ?></p>

</body>
</html>
```

控制器

使用文本编辑器, 创建一个名为 **Upload.php** 的控制器。在其中放入下面的代码, 并保存到你的 **app/Controllers** 目录:

```
<?php

namespace App\Controllers;

use CodeIgniter\Files\File;

class Upload extends BaseController
{
    protected $helpers = ['form'];

    public function index()
    {
        return view('upload_form', ['errors' => []]);
    }
```

(续下页)

(接上页)

```
    public function upload()
    {
        $validationRule = [
            'userfile' => [
                'label' => 'Image File',
                'rules' => [
                    'uploaded[userfile]',
                    'is_image[userfile]',
                    'mime_in[userfile,image/jpg,image/jpeg,image/
↪gif,image/png,image/webp]',
                    'max_size[userfile,100]',
                    'max_dims[userfile,1024,768]',
                ],
            ],
        ];
        if (! $this->validateData([], $validationRule)) {
            $data = ['errors' => $this->validator->getErrors()];

            return view('upload_form', $data);
        }

        $img = $this->request->getFile('userfile');

        if (! $img->hasMoved()) {
            $filepath = WRITEPATH . 'uploads/' . $img->store();

            $data = ['uploaded_fileinfo' => new File($filepath)];

            return view('upload_success', $data);
        }

        $data = ['errors' => 'The file has already been moved.'];

        return view('upload_form', $data);
    }
}
```

备注: 由于 HTML 文件上传字段的值不存在, 它存储在 `$_FILES` 全局变量中, 所以只能使用文件上传规则 来验证上传的文件, 不能使用验证器。`required` 规则也不能使用, 请使用 `uploaded` 代替。

只有文件上传规则 可以用于验证上传的文件。

因此，规则 `required` 也不能使用，所以如果文件是必需的，请使用规则 `uploaded`。

注意，一个空数组（`[]`）作为第一个参数传递给 `$this->validateData()`。这是因为文件验证规则直接从 Request 对象获取上传文件的数据。

如果表单中有除了文件上传之外的字段，请将字段数据作为第一个参数传递。

路由

使用文本编辑器, 打开 **app/Config/Routes.php**。在其中添加以下两个路由:

```
<?php

// ...

/*
 * --------------------------------------------------------------------
 ↪---
 * Route Definitions
 * --------------------------------------------------------------------
 ↪---
 */

// We get a performance increase by specifying the default
// route since we don't have to scan directories.
$routes->get('/', 'Home::index');

$routes->get('upload', 'Upload::index');          // Add this line.
$routes->post('upload/upload', 'Upload::upload'); // Add this line.

// ...
```

上传目录

上传的文件存储在 **writable/uploads/** 目录下。

试一试!

要测试你的表单, 使用类似这样的 URL 访问你的网站:

```
example.com/index.php/upload/
```

你应该可以看到一个上传表单。尝试上传一个图像文件 (可以是 **jpg**、**gif**、**png** 或 **webp**)。如果控制器中的路径正确, 它应该可以工作。

访问文件

所有文件

当你上传文件时, 可以通过 PHP 的 `$_FILES` 超全局变量以原生方式访问它们。当处理一次上传的多个文件时, 这个数组有一些重大缺陷, 也存在许多开发者可能不知道的潜在安全问题。CodeIgniter 通过把文件操作标准化到一个通用接口后面, 可以帮助解决这两个问题。

文件是通过当前的 `IncomingRequest` 实例访问的。要检索与这个请求一起上传的所有文件, 使用 `getFiles()`。它将返回一个由 `CodeIgniter\HTTP\Files\UploadedFile` 实例表示的文件数组:

```
<?php

$files = $this->request->getFiles();
```

当然, 文件输入有多种命名方式, 任何不简单的都会产生奇怪的结果。数组的返回方式和你期望的一样。使用最简单的方式, 单个文件可能像这样提交:

```
<input type="file" name="avatar">
```

它将返回一个简单的像这样的数组:

```
[
    'avatar' => // 上传的文件实例
];
```

备注: UploadedFile 实例对应 `$_FILES` 。即使用户只是点击提交按钮而没有上传任何文件, 该实例也仍然存在。你可以通过 UploadedFile 的 `isValid()` 方法检查文件是否真的被上传。参见验证文件。

如果你为名称使用了数组表示法, 输入看起来像这样:

```
<input type="file" name="my-form[details][avatar]">
```

`getFiles()` 返回的数组看起来更像这样:

```
[
     'my-form' => [
        'details' => [
            'avatar' => // 上传的文件实例
        ],
    ],
]
```

在某些情况下, 你可以指定一个文件数组来上传:

```
上传头像: <input type="file" name="my-form[details][avatars][]">
上传头像: <input type="file" name="my-form[details][avatars][]">
```

在这种情况下, 返回的文件数组更像是:

```
[
    'my-form' => [
        'details' => [
            'avatar' => [
                0 => // 上传的文件实例,
                1 => // 上传的文件实例,
            ],
        ],
```

(续下页)

(接上页)

```
    ],
]
```

单个文件

如果你只需要访问单个文件, 可以使用 `getFile()` 直接获取文件实例。它将返回一个 `CodeIgniter\HTTP\Files\UploadedFile` 实例:

最简单的用法

使用最简单的方式, 单个文件可能这样提交:

```
<input type="file" name="userfile">
```

它将返回一个简单的文件实例, 像这样:

```
<?php

$file = $this->request->getFile('userfile');
```

数组表示法

如果你为名称使用数组表示法, 输入看起来像这样:

```
<input type="file" name="my-form[details][avatar]">
```

获取文件实例:

```
<?php

$file = $this->request->getFile('my-form.details.avatar');
```

多个文件

```
<input type="file" name="images[]" multiple>
```

在控制器中:

```
<?php

if ($imagefile = $this->request->getFiles()) {
    foreach ($imagefile['images'] as $img) {
        if ($img->isValid() && ! $img->hasMoved()) {
            $newName = $img->getRandomName();
            $img->move(WRITEPATH . 'uploads', $newName);
        }
    }
}
```

其中 `images` 是表单字段名称的循环。

如果有多个相同名称的文件, 你可以使用 `getFile()` 来单独获取每个文件。

在控制器中:

```
<?php

$file1 = $this->request->getFile('images.0');
$file2 = $this->request->getFile('images.1');
```

你可能会发现使用 `getFileMultiple()` 更方便, 它可以获取一个具有相同名称的上传文件数组:

```
<?php

$files = $this->request->getFileMultiple('images');
```

另一个例子:

```
上传头像: <input type="file" name="my-form[details][avatars][]">
上传头像: <input type="file" name="my-form[details][avatars][]">
```

在控制器中:

```
<?php

$file1 = $this->request->getFile('my-form.details.avatars.0');
$file2 = $this->request->getFile('my-form.details.avatars.1');
```

备注: 使用 `getFiles()` 更合适。

处理文件

一旦你获取了 UploadedFile 实例，你可以以安全的方式获取有关文件的信息，并将文件移动到新位置。

验证文件

你可以通过调用 `isValid()` 方法来检查文件是否真的通过 HTTP 上传且没有错误:

```
<?php

if (! $file->isValid()) {
    throw new \RuntimeException($file->getErrorString() . '(' .
↪$file->getError() . ')');
}
```

如这个例子所示，如果文件有上传错误，你可以通过 `getError()` 和 `getErrorString()` 方法获取错误码 (整数) 和错误消息。可以通过这个方法发现以下错误:

- 文件超过了你的 `upload_max_filesize` ini 设置。
- 文件超过了表单中定义的上传限制。
- 文件只被部分上传。
- 没有文件被上传。
- 文件无法写入磁盘。
- 文件上传失败: 缺少临时目录。
- 文件上传被 PHP 扩展停止。

文件名

getName()

你可以使用 `getName()` 方法获取客户提供的原始文件名。这通常是客户端发送的文件名, 不应该相信它。如果文件已经移动, 这将返回移动后的文件的最终名称:

```
<?php

$name = $file->getName();
```

getClientName()

即使文件已经移动, 也总是返回上传文件的原始名称, 就是客户端发送的名称:

```
<?php

$originalName = $file->getClientName();
```

getTempName()

要获取上传过程中创建的临时文件的完整路径, 你可以使用 `getTempName()` 方法:

```
<?php

$tempfile = $file->getTempName();
```

其他文件信息

getClientExtension()

根据上传的文件名返回原始文件扩展名:

```
<?php

$ext = $file->getClientExtension();
```

> **警告:** 这不是可信的来源。要获取可信的版本, 请改用 `guessExtension()`。

getClientMimeType()

返回客户端提供的文件的 MIME 类型。这不是一个可信的值。要获取可信的版本，请改用 `getMimeType()`：

```
<?php

$type = $file->getClientMimeType();

echo $type; // image/png
```

getClientPath()

在 4.4.0 版本加入.

当客户端通过目录上传方式上传文件时，返回上传文件的 webkit 相对路径。在 PHP 8.1 以下的版本中，返回 `null`。

```
<?php

$clientPath = $file->getClientPath();
echo $clientPath; // dir/file.txt, or dir/sub_dir/file.txt
```

移动文件

使用原始文件名

每个文件都可以使用贴切的 `move()` 方法移动到新位置。第一个参数是要移动文件的目录:

```
<?php

$file->move(WRITEPATH . 'uploads');
```

默认情况下, 使用原始文件名。

指定新文件名

你可以通过第二个参数指定一个新文件名:

```
<?php

$newName = $file->getRandomName();
$file->move(WRITEPATH . 'uploads', $newName);
```

覆盖现有文件

默认情况下, 如果目标文件已经存在, 会使用一个新的文件名。例如, 如果 **image_name.jpg** 已经存在于目录中, 那么文件名会自动设置为 **image_name_1.jpg**。

你可以传入 `true` 作为第三个参数来覆盖现有文件:

```
<?php

$file->move(WRITEPATH . 'uploads', null, true);
```

检查文件是否移动

一旦文件被移动，临时文件将被删除。你可以使用 `hasMoved()` 方法来检查文件是否已经被移动，该方法返回一个布尔值:

```
<?php

if ($file->isValid() && ! $file->hasMoved()) {
    $file->move($path);
}
```

移动失败时

在几种情况下，移动上传的文件可能会失败，并抛出一个 `HTTPException`:

- 文件已经移动过
- 文件上传不成功
- 文件移动操作失败 (例如权限不正确)

存储文件

每个文件都可以使用同名的 `store()` 方法移动到新位置。

使用最简单的用法, 单个文件可能这样提交:

```
<input type="file" name="userfile">
```

默认情况下, 上传的文件将保存在 **writable/uploads** 目录下。会创建 **YYYYMMDD** 文件夹和随机文件名。返回文件路径:

```
<?php

$path = $this->request->getFile('userfile')->store();
```

你可以指定一个目录作为第一个参数来移动文件。通过第二个参数指定一个新文件名:

```
<?php

$path = $this->request->getFile('userfile')->store('head_img/',
↪'user_name.jpg');
```

在几种情况下, 移动上传的文件可能会失败, 并抛出一个 `HTTPException`:

- 文件已经移动过
- 文件上传不成功
- 文件移动操作失败 (例如权限不正确)

7.1.19 使用 URI

CodeIgniter 提供了面向对象的方式来在你的应用中使用 URI。这使得确保 URI 结构始终正确变得很简单, 无论 URI 有多复杂, 都可以安全正确地添加相对 URI 到现有的 URI。

- *创建 URI 实例*
  - *当前 URI*
- *URI 字符串*
- *URI 各部分*
  - *Scheme(方案)*
  - *Authority(权限)*
  - *UserInfo(用户信息)*
  - *Host(主机)*
  - *Port(端口)*
  - *Path(路径)*
  - *Query(查询)*
  - *Fragment(片段)*
- *URI 段*
- *禁用抛出异常*

创建 URI 实例

创建一个 URI 实例就像创建一个新的类实例一样简单。

当你创建新实例时，可以在构造函数中传递完整或部分 URL，并将其解析为相应的部分:

```
$uri = new \CodeIgniter\HTTP\URI('http://www.example.com/some/path
↪');
```

或者, 你可以使用`service()` 函数来获取一个实例:

```
$uri = service('uri', 'http://www.example.com/some/path');
```

自 v4.4.0 起，如果你没有传递 URL，则返回当前的 URI：

```
$uri = service('uri'); // returns the current SiteURI instance.
```

备注： 上述代码返回 `SiteURI` 实例，它扩展了 `URI` 类。`URI` 类用于一般的 URI，而 `SiteURI` 类用于你的站点 URI。

当前 URI

当你需要一个表示当前请求 URL 的对象时，你可以使用*URL 辅助函数* 中提供的*current_url()* 函数：

```
$uri = current_url(true);
```

你必须传递 `true` 作为第一个参数, 否则它会返回当前 URL 的字符串表示。

这个 URI 基于当前请求对象和你在 `Config\App` 中的设置 (`baseURL`、`indexPage` 和 `forceGlobalSecureRequests`) 确定的相对路径。

假设你在一个扩展了 `CodeIgniter\Controller` 的控制器中，你还可以获取当前的 SiteURI 实例：

```
$uri = $this->request->getUri();
```

URI 字符串

很多时候, 你真正想要的只是获取一个 URI 的字符串表示。把 URI 转换为字符串就可以简单地做到这一点:

```
<?php

$uri = current_url(true);
echo (string) $uri;  // http://example.com/index.php
```

如果你知道 URI 的各个部分, 只是想确保它们都格式化正确, 可以使用 URI 类的静态 `createURIString()` 方法生成一个字符串:

```
<?php

use CodeIgniter\HTTP\URI;

$uriString = URI::createURIString($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');
```

重要: 当 `URI` 被转换为字符串时, 它会尝试根据 `Config\App` 中定义的设置调整项目 URL。如果你需要完全不变的字符串表示, 请改用 `URI::createURIString()`。

URI 各部分

一旦你有了一个 URI 实例, 你就可以设置或检索 URI 的各个部分。本节将详细介绍这些部分是什么, 以及如何使用它们。

Scheme(方案)

Scheme 常常是 ‘http’ 或 ‘https’ , 但任何 scheme 都是被支持的, 包括 ‘file’、’ mailto’ 等。

```
<?php

$uri = new \CodeIgniter\HTTP\URI('http://www.example.com/some/path
↪');

echo $uri->getScheme(); // 'http'
$uri->setScheme('https');
```

Authority(权限)

许多 URI 包含一些统称为 'authority' 的元素。这包括任何用户信息、主机和端口号。你可以使用 `getAuthority()` 方法作为一个字符串检索所有这些部分, 或者可以操作各个部分。

```
<?php

$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, 它不会显示密码部分, 因为你不会想把它展示给任何人。如果你想展示密码, 可以使用 `showPassword()` 方法。这个 URI 实例会一直展示密码, 直到你再次关闭它, 所以一定要在使用完以后立即关闭它:

```
<?php

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, 请只传入 `true` 作为唯一参数:

```
<?php

echo $uri->getAuthority(true);  // user@example.com
```

备注: 如果当前端口是 scheme 的默认端口则不会显示。

UserInfo(用户信息)

userinfo 部分简单就是你在 FTP URI 中可能看到的用户名和密码。虽然你可以作为 Authority 的一部分获取它, 但你也可以自己获取它:

```
<?php

echo $uri->getUserInfo();    // user
```

默认情况下, 它不会显示密码, 但是你可以用 `showPassword()` 方法覆盖:

```
<?php

echo $uri->showPassword()->getUserInfo();    // user:password
$uri->showPassword(false);
```

Host(主机)

URI 的 host 部分通常是 URL 的域名。可以使用 `getHost()` 和 `setHost()` 方法简单设置和获取它:

```
<?php

$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)

端口是一个介于 0 和 65535 之间的整数。每个 scheme 都有一个默认值与之关联。

```
<?php

$uri = new \CodeIgniter\HTTP\URI('ftp://user:password@example.
```

(续下页)

(接上页)

```
↪com:21/some/path');

echo $uri->getPort();    // 21
echo $uri->setPort(2201)->getPort(); // 2201
```

使用 `setPort()` 方法时, 会检查端口是否在有效范围内, 然后进行分配。

Path(路径)

路径是站点内部的所有段。正如预期的那样，可以使用 `getPath()` 和 `setPath()` 方法来操作它：

```
<?php

$uri = new \CodeIgniter\HTTP\URI('http://www.example.com/some/path
↪');

echo $uri->getPath();                          // '/some/path'
echo $uri->setPath('/another/path')->getPath();  // '/another/path'
```

备注: 当设置路径时，它会被清理以编码任何危险字符，并移除“点段”（dot segment）以确保安全。

备注: 自 v4.4.0 起，`SiteURI::getRoutePath()` 方法返回相对于 baseURL 的 URI 路径，而 `SiteURI::getPath()` 方法始终返回带有前导 / 的完整 URI 路径。

Query(查询)

可以通过类使用简单的字符串表示来操作查询数据。

获取/设置查询

当前查询值只能作为字符串进行设置。

```
<?php

$uri = new \CodeIgniter\HTTP\URI('http://www.example.com?foo=bar');

echo $uri->getQuery();  // 'foo=bar'
$uri->setQuery('foo=bar&bar=baz');
```

`setQuery()` 方法会覆盖现有的查询变量。

备注: 查询值不能包含片段。如果包含, 会抛出一个 InvalidArgumentException。

从数组设置查询

你可以使用数组设置查询值:

```
<?php

$uri->setQueryArray(['foo' => 'bar', 'bar' => 'baz']);
```

`setQueryArray()` 方法会覆盖现有的查询变量。

添加查询值

你可以使用 `addQuery()` 方法向查询变量集合中添加一个值, 而不会破坏现有的查询变量。第一个参数是变量名称, 第二个参数是值:

```
<?php

$uri->addQuery('foo', 'bar');
```

过滤查询值

你可以通过向 `getQuery()` 方法传递一个选项数组来过滤返回的查询值, 包含一个 *only* 键或一个 *except* 键:

```
<?php

$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']]);
```

这只改变此次调用返回的值。

更改查询值

如果你需要更永久地修改 URI 的查询值, 你可以使用 `stripQuery()` 和 `keepQuery()` 方法来更改实际对象的查询变量集合:

```
<?php

$uri = new \CodeIgniter\HTTP\URI('http://www.example.com?foo=bar&
↪bar=baz&baz=foz');

// Leaves just the 'baz' variable
$uri->stripQuery('foo', 'bar');

// Leaves just the 'foo' variable
$uri->keepQuery('foo');
```

备注: 默认情况下, `setQuery()` 和 `setQueryArray()` 方法使用原生的 `parse_str()` 函数来准备数据。如果你想使用更宽松的规则 (允许键名包含点), 你可以先使用特殊的 `useRawQueryString()` 方法。

Fragment(片段)

片段是 URL 末尾以井号 (#) 开头的部分。在 HTML URL 中它们链接到页面内的锚点。媒体 URI 可以以各种其他方式使用它们。

```
<?php

$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 段

路径之间的每个斜杠之间的部分都是单个段。

备注: 对于你的站点 URI 来说,URI 段仅指相对于 baseURL 的 URI 路径部分。如果你的 baseURL 包含子文件夹, 则值会与当前 URI 路径不同。

URI 类提供了一个简单的方法来确定段的值。段从最左边的路径开始编号 1。

```
<?php

// URI = http://example.com/users/15/profile

// Prints '15'
if ($uri->getSegment(1) === 'users') {
    echo $uri->getSegment(2);
}
```

你也可以通过 `getSegment()` 方法的第二个参数为特定段设置不同的默认值。默认值为空字符串。

```
<?php
```

(续下页)

(接上页)

```
// URI = http://example.com/users/15/profile

// will print 'profile'
echo $uri->getSegment(3, 'foo');
// will print 'bar'
echo $uri->getSegment(4, 'bar');
// will throw an exception
echo $uri->getSegment(5, 'baz');
// will print 'baz'
echo $uri->setSilent()->getSegment(5, 'baz');
// will print '' (empty string)
echo $uri->setSilent()->getSegment(5);
```

备注: 你可以获取最后的 +1 段。当你试图获取最后的 +2 或更多段时, 默认情况下会抛出异常。你可以使用 `setSilent()` 方法来防止抛出异常。

你可以获取总段数:

```
<?php

$total = $uri->getTotalSegments(); // 3
```

最后, 你可以检索所有段的数组:

```
<?php

$segments = $uri->getSegments();

/*
 * Produces:
 * [
 *     0 => 'users',
 *     1 => '15',
 *     2 => 'profile',
 * ]
 */
```

禁用抛出异常

默认情况下, 此类的某些方法可能会抛出异常。如果你要禁用它, 可以设置一个特殊标志来防止抛出异常。

```
<?php

// Disable throwing exceptions
$uri->setSilent();

// Enable throwing exceptions (default)
$uri->setSilent(false);
```

7.1.20 User Agent 类

User Agent 类提供了一些有助于识别访问你网站的浏览器、移动设备或机器人的信息的函数。

- *使用 User Agent 类*
  - *初始化类*
  - *User Agent 定义*
  - *示例*
- *类参考*

使用 User Agent 类

初始化类

User Agent 类总是可以直接从当前的*IncomingRequest* 实例获取。默认情况下, 在你的控制器中会有一个请求实例, 你可以从中获取 User Agent 类:

```
<?php

$agent = $this->request->getUserAgent();
```

User Agent 定义

User Agent 名称定义位于以下配置文件中:**app/Config/UserAgents.php**。如果需要的话你可以在各种 User Agent 数组中添加项。

示例

当 User Agent 类被初始化时, 它会试图确定是否正在浏览你的网站的 User Agent 是网页浏览器、移动设备还是机器人。如果可用的话, 它也会收集平台信息:

```
<?php

$agent = $this->request->getUserAgent();

if ($agent->isBrowser()) {
    $currentAgent = $agent->getBrowser() . ' ' . $agent->
↪getVersion();
} elseif ($agent->isRobot()) {
    $currentAgent = $agent->getRobot();
} elseif ($agent->isMobile()) {
    $currentAgent = $agent->getMobile();
} else {
    $currentAgent = 'Unidentified User Agent';
}

echo $currentAgent;

echo $agent->getPlatform(); // Platform info (Windows, Linux, Mac,␣
↪etc.)
```

类参考

class CodeIgniter\HTTP**UserAgent**

isBrowser([*$key = null*])

参数

- **$key** (string) –可选的浏览器名称

返回

如果 User Agent 是 (指定的) 浏览器则为 true, 否则为 false

返回类型

`bool`

如果 User Agent 是已知的网页浏览器, 则返回 true/false(布尔值)。

```
<?php

if ($agent->isBrowser('Safari')) {
    echo 'You are using Safari.';
} elseif ($agent->isBrowser()) {
    echo 'You are using a browser.';
}
```

备注: 这个示例中的 "Safari" 字符串是浏览器定义列表中的一个数组键。如果你想添加新浏览器或更改字符串, 可以在 **app/Config/UserAgents.php** 文件中找到这个列表。

---

isMobile([*$key = null*])

参数

- **$key** (`string`) –可选的移动设备名称

返回

如果 User Agent 是 (指定的) 移动设备则为 true, 否则为 false

返回类型

`bool`

如果 User Agent 是已知的移动设备, 则返回 true/false(布尔值)。

```
<?php

if ($agent->isMobile('iphone')) {
    echo view('iphone/home');
} elseif ($agent->isMobile()) {
    echo view('mobile/home');
} else {
```

(续下页)

(接上页)

```
    echo view('web/home');
}
```

isRobot([*$key* = *null*])

参数

- **$key** (string) –可选的机器人名称

返回

如果 User Agent 是 (指定的) 机器人则为 true, 否则为 false

返回类型

bool

如果 User Agent 是已知的机器人, 则返回 true/false(布尔值)。

备注: User Agent 库只包含最常见的机器人定义。这不是一个完整的机器人列表。有成百上千个, 逐个搜索每一个效率不高。如果你发现一些常访问你网站但列表中缺失的机器人, 可以添加到你的 **app/Config/UserAgents.php** 文件中。

---

isReferral()

返回

如果 User Agent 来自其他网站的推荐则为 true, 否则为 false

返回类型

bool

如果 User Agent 来自其他网站的推荐, 则返回 true/false(布尔值)。

getBrowser()

返回

检测到的浏览器或空字符串

返回类型

string

返回查看你网站的网页浏览器的名称字符串。

getVersion()

返回

检测到的浏览器版本或空字符串

返回类型

`string`

返回查看你网站的网页浏览器的版本号字符串。

getMobile()

返回

检测到的移动设备品牌或空字符串

返回类型

`string`

返回查看你网站的移动设备的名称字符串。

getRobot()

返回

检测到的机器人名称或空字符串

返回类型

`string`

返回查看你网站的机器人的名称字符串。

getPlatform()

返回

检测到的操作系统或空字符串

返回类型

`string`

返回查看你网站的平台 (Linux、Windows、OS X 等) 的字符串。

getReferrer()

返回

检测到的引用网站或空字符串

返回类型

`string`

如果 User Agent 来自其他网站的推荐, 返回推荐网站。通常会像这样测试:

```
<?php

if ($agent->isReferral()) {
    echo $agent->getReferrer();
}
```

getAgentString()

返回

完整的 User Agent 字符串或空字符串

返回类型

string

返回包含完整 User Agent 字符串的字符串。通常看起来像这样:

```
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) – 自定义 User Agent 字符串

返回类型

void

解析自定义 User Agent 字符串, 不同于当前访问者报告的字符串。

7.1.21 数据验证

CodeIgniter 提供了全面的数据验证类，帮助你减少需要编写的代码量。

- 概述
- 表单验证教程
  - 表单
  - 成功页面

  - 控制器
  - 路由
  - 尝试运行!
  - 说明
  - 添加验证规则
- 验证配置
  - 传统规则与严格规则
- 加载库
- 验证的工作原理
- 设置验证规则
  - 设置单个规则
  - 设置多个规则
  - 为数组数据设置规则
  - *withRequest()*
- 使用验证
  - 运行验证
  - 运行多次验证
  - 验证单个值
  - 获取已验证数据
  - 将验证规则保存到配置文件
  - 验证占位符
- 处理错误
  - 设置自定义错误信息
  - 信息与验证标签的翻译
  - 获取所有错误
  - 获取单个错误

  - 检查是否存在错误
  - 重定向与验证错误
- 自定义错误显示
  - 创建视图
  - 配置
  - 指定模板
- 创建自定义规则
  - 使用规则类
  - 使用闭包规则
  - 使用可调用规则
- 可用规则
  - 通用规则
  - 文件上传规则

概述

在解释 CodeIgniter 的数据验证方法之前，我们先描述理想场景：

1. 显示表单。
2. 填写并提交表单。
3. 如果提交了无效内容，或可能遗漏了必填项，表单会重新显示包含你的数据和描述问题的错误信息。
4. 此过程持续直到提交有效表单。

在接收端，脚本必须：

1. 检查必填数据。
2. 验证数据类型正确并符合正确标准。例如，如果提交用户名，必须验证其仅包含允许字符。必须满足最小长度且不超过最大长度。用户名不能是他人已存在的用户名，甚至可能是保留字等。
3. 对数据进行安全过滤。

4. 必要时预处理数据格式。
5. 准备数据以便插入数据库。

尽管上述过程并不复杂，但通常需要大量代码，并且为了显示错误信息，通常会在表单 HTML 中放置各种控制结构。表单验证虽然创建简单，但实现起来通常非常混乱和繁琐。

表单验证教程

以下是实现 CodeIgniter 表单验证的实践教程。

要实现表单验证，你需要三样东西：

1. 包含表单的视图 文件。
2. 包含成功提交后显示的“成功”信息的视图文件。
3. 接收和处理提交数据的控制器 方法。

让我们以会员注册表单为例创建这三个内容。

表单

使用文本编辑器创建名为 **signup.php** 的表单。在其中放置以下代码并保存到 **app/Views/** 文件夹:

```
<html>
<head>
    <title>我的表单</title>
</head>
<body>

    <?= validation_list_errors() ?>

    <?= form_open('form') ?>

        <h5>用户名</h5>
        <input type="text" name="username" value="<?= set_value(
↪'username') ?>" size="50">
```

(续下页)

(接上页)

```
        <h5>密码</h5>
        <input type="text" name="password" value="<?= set_value(
↪'password') ?>" size="50">

        <h5>确认密码</h5>
        <input type="text" name="passconf" value="<?= set_value(
↪'passconf') ?>" size="50">

        <h5>邮箱地址</h5>
        <input type="text" name="email" value="<?= set_value('email
↪') ?>" size="50">

        <div><input type="submit" value="提交"></div>

    <?= form_close() ?>

</body>
</html>
```

成功页面

使用文本编辑器创建名为 **success.php** 的表单。在其中放置以下代码并保存到 **app/Views/** 文件夹:

```
<html>
<head>
    <title>我的表单</title>
</head>
<body>

    <h3>你的表单已成功提交！</h3>

    <p><?= anchor('form', '再试一次！') ?></p>

</body>
</html>
```

控制器

使用文本编辑器创建名为 **Form.php** 的控制器。在其中放置以下代码并保存到 **app/Controllers/** 文件夹：

```
<?php

namespace App\Controllers;

class Form extends BaseController
{
    protected $helpers = ['form'];

    public function index()
    {
        if (! $this->request->is('post')) {
            return view('signup');
        }

        $rules = [
            // @TODO
        ];

        $data = $this->request->getPost(array_keys($rules));

        if (! $this->validateData($data, $rules)) {
            return view('signup');
        }

        // If you want to get the validated data.
        $validData = $this->validator->getValidated();

        return view('success');
    }
}
```

备注: 自 v4.3.0 起可使用*$this->request->is()* 方法。在早期版本中，需使用 `if`

`(strtolower($this->request->getMethod()) !== 'post')`。

备注: *$this->validator->getValidated()* 方法自 v4.4.0 起可用。

路由

然后在 **app/Config/Routes.php** 中添加控制器的路由:

```
// ...

$routes->get('form', 'Form::index');
$routes->post('form', 'Form::index');

// ...
```

尝试运行!

要测试表单，使用类似以下 URL 访问你的站点:

```
example.com/index.php/form/
```

如果提交表单，你应会看到表单重新加载。这是因为你尚未在*$this->validateData()* 中设置任何验证规则。

`validateData()` 是控制器中的一个方法，它在内部使用 **验证类**。详见*$this->validateData()*。

备注: 由于尚未告知 `validateData()` 方法验证任何内容，默认情况下它会返回 false（布尔值）。只有当所有规则成功应用且未失败时，`validateData()` 方法才会返回 true。

说明

你会注意到上述页面的一些特点。

signup.php

表单（**signup.php**）是标准网页表单，但有几点例外：

1. 使用*表单辅助函数* 创建表单开头和结尾。技术上这不是必需的，你可以使用标准 HTML 创建表单。但使用辅助函数的优点是根据配置中的 URL 生成 action URL，提高应用在 URL 变更时的可移植性。
2. 在表单顶部你会注意到以下函数调用：

   ```
   <?= validation_list_errors() ?>
   ```

   此函数返回验证器返回的所有错误信息。若无信息则返回空字符串。

Form.php

控制器（**Form.php**）有一个属性 `$helpers`，它加载视图文件使用的表单辅助函数。

控制器有一个方法 `index()`。当非 POST 请求时返回 **signup** 视图显示表单。否则使用控制器提供的*$this->validateData()* 方法运行验证流程。根据验证结果展示表单或成功页面。

添加验证规则

然后在控制器（**Form.php**）中添加验证规则：

```
// ...

$rules = [
    'username' => 'required|max_length[30]',
    'password' => 'required|max_length[255]|min_length[10]',
    'passconf' => 'required|max_length[255]|matches[password]',
    'email'    => 'required|max_length[254]|valid_email',
];
```

(续下页)

(接上页)

```
// ...
```

提交表单后，你将看到成功页面或带有错误信息的表单。

验证配置

传统规则与严格规则

CodeIgniter 4 有两种验证规则类。

默认规则类（**严格规则**）使用命名空间 `CodeIgniter\Validation\StrictRules`，提供严格验证。

传统规则类（**传统规则**）使用命名空间 `CodeIgniter\Validation`，仅为向后兼容保留。它们可能无法正确验证非字符串值，新项目无需使用。

备注： 自 v4.3.0 起默认使用 **严格规则**以提高安全性。

严格规则

在 4.2.0 版本加入.

严格规则不使用隐式类型转换。

传统规则

重要： 传统规则仅为向后兼容存在。新项目请勿使用。即使已在使用的项目也建议切换至严格规则。

警告： 当验证包含非字符串值（如 JSON 数据）时，应使用 **严格规则**。

传统规则隐式假设验证的是字符串值，输入值可能隐式转换为字符串。这对大多数基本用例（如验证 POST 数据）有效。

但例如使用 JSON 输入数据时，可能是布尔/空/数组类型。用传统规则验证布尔 `true` 会转换为字符串 `'1'`。用 `integer` 规则验证时，`'1'` 会通过验证。

使用传统规则

> **警告：传统规则**仅为向后兼容保留。新项目不建议使用，可能无法正确验证非字符串值。

若要使用传统规则，需修改 **app/Config/Validation.php** 中的规则类：

```
<?php

namespace Config;

// ...

class Validation extends BaseConfig
{
    // ...

    public array $ruleSets = [
        \CodeIgniter\Validation\CreditCardRules::class,
        \CodeIgniter\Validation\FileRules::class,
        \CodeIgniter\Validation\FormatRules::class,
        \CodeIgniter\Validation\Rules::class,
    ];

    // ...
}
```

加载库

验证库作为名为 **validation** 的服务加载：

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

这段代码会自动加载 `Config\Validation` 文件，其中包含用于引入多个规则集的设置以及可轻松复用的规则集合。

备注： 你可能永远不需要使用此方法，因为控制器 和模型 都提供了简化验证的方法。

验证的工作原理

- 验证过程永不更改待验证数据。
- 根据设置的验证规则依次检查每个字段。若任何规则返回 false，该字段检查即终止。
- 格式规则不允许空字符串。若要允许空字符串，需添加 `permit_empty` 规则。
- 若待验证数据中不存在某字段，其值视为 `null`。要检查字段是否存在，需添加 `field_exists` 规则。

备注： `field_exists` 规则自 v4.5.0 起可用。

设置验证规则

CodeIgniter 允许为字段设置多个验证规则并按顺序级联。要设置验证规则，需使用 `setRule()`、`setRules()` 或 `withRequest()` 方法。

设置单个规则

setRule()

此方法设置单个规则，方法签名为:

```
setRule(string $field, ?string $label, array|string $rules[, array
↪$errors = []])
```

$rules 接受管道分隔的规则列表或规则数组:

```
$validation->setRule('username', 'Username', 'required|max_
↪length[30]|min_length[3]');
$validation->setRule('password', 'Password', ['required', 'max_
↪length[255]', 'min_length[8]', 'alpha_numeric_punct']);
```

传递给 $field 的值必须与发送的数据数组键名匹配。若数据直接来自 $_POST，则必须与表单输入名称完全匹配。

> **警告:** v4.2.0 之前，此方法的第三个参数 $rules 类型限定为 string。v4.2.0 及之后版本移除了类型限定以支持数组。为避免扩展类中重写此方法时破坏 LSP，子类方法也应移除类型限定。

设置多个规则

setRules()

类似 setRule()，但接受字段名和规则的数组:

```
$validation->setRules([
    'username' => 'required|max_length[30]',
    'password' => 'required|max_length[255]|min_length[10]',
]);
// or
$validation->setRules([
    'username' => ['required', 'max_length[30]'],
```

(续下页)

(接上页)

```
    'password' => ['required', 'max_length[255]', 'min_length[10]'],
]);
```

要设置带标签的错误信息：

```
$validation->setRules([
    'username' => ['label' => 'Username', 'rules' => 'required|max_
↪length[30]'],
    'password' => ['label' => 'Password', 'rules' => 'required|max_
↪length[255]|min_length[10]'],
]);
// or
$validation->setRules([
    'username' => ['label' => 'Username', 'rules' => 'required|max_
↪length[30]'],
    'password' => ['label' => 'Password', 'rules' => ['required',
↪'max_length[255]', 'min_length[10]']],
]);
```

备注： `setRules()` 会覆盖之前设置的规则。要为现有规则集添加多个规则，需多次使用 `setRule()`。

为数组数据设置规则

若数据是嵌套关联数组，可使用“点数组语法”轻松验证：

```
/*
 * The data to test:
 * [
 *     'contacts' => [
 *         'name' => 'Joe Smith',
 *         'friends' => [
 *             [
 *                 'name' => 'Fred Flinstone',
 *             ],
```

(续下页)

(接上页)

```
 *              [
 *                  'name' => 'Wilma',
 *              ],
 *          ]
 *      ]
 * ]
 */

// Joe Smith
$validation->setRules([
    'contacts.name' => 'required|max_length[60]',
]);
```

可使用通配符 * 匹配数组的任意一级：

```
// Fred Flintsone & Wilma
$validation->setRules([
    'contacts.friends.*.name' => 'required|max_length[60]',
]);
```

备注： v4.4.4 之前存在 bug，通配符 * 会错误验证数据维度。详见升级指南。

“点数组语法”对单维数组数据也很有用。例如多选下拉返回的数据：

```
/*
 * The data to test:
 * [
 *     'user_ids' => [
 *         1,
 *         2,
 *         3,
 *     ]
 * ]
 */

// Rule
$validation->setRules([
```

(续下页)

(接上页)

```
    'user_ids.*' => 'required|max_length[19]',
]);
```

withRequest()

重要: 此方法仅为向后兼容存在。新项目请勿使用。即使已在使用，也建议改用其他更合适的方法。

> **警告:** 若仅需验证 POST 数据，请勿使用 `withRequest()`。此方法使用*$request->getVar()*，根据 php.ini 的 request-order 返回 `$_GET`、`$_POST` 或 `$_COOKIE` 数据（按顺序）。新值覆盖旧值。若同名，Cookie 可能覆盖 POST 值。

最常见的验证场景是验证来自 HTTP 请求的输入数据。若需要，可传递当前 Request 对象实例，它会将所有输入数据设为待验证数据：

```
$validation = service('validation');
$request    = service('request');

if ($validation->withRequest($request)->run()) {
    // If you use the input data, you should get it from the␣
↪getValidated() method.
    // Otherwise you may create a vulnerability.
    $validData = $validation->getValidated();

    // ...
}
```

> **警告:** 使用此方法时，应使用*getValidated()* 获取已验证数据。因为此方法在处理 JSON 请求（`Content-Type: application/json`）时通过*$request->getJSON()* 获取 JSON 数据，或处理非表单提交的 PUT、PATCH、DELETE 请求时通过*$request->getRawInput()* 获取原始数据，攻击者可能改变验证数据。

备注: *getValidated()* 方法自 v4.4.0 起可用。

使用验证

运行验证

`run()` 方法运行验证，方法签名为:

```
run(?array $data = null, ?string $group = null, ?string $dbGroup =␣
↪null): bool
```

`$data` 是待验证数据数组。可选参数 `$group` 是要应用的预定义规则组。可选参数 `$dbGroup` 是使用的数据库组。

验证成功时返回 true。

```
if (! $validation->run($data)) {
    // handle validation errors
}
// or
if (! $validation->run($data, 'signup')) {
    // handle validation errors
}
```

运行多次验证

备注: `run()` 方法不会重置错误状态。若前次运行失败，`run()` 将始终返回 false，且 `getErrors()` 返回所有先前错误直到显式重置。

若需运行多次验证（例如不同数据集或不同规则），可能需要在每次运行前调用 `$validation->reset()` 清除先前错误。注意 `reset()` 会清空之前设置的任何数据、规则或自定义错误，因此需重复调用 `setRules()`、`setRuleGroup()` 等:

```
foreach ($userAccounts as $user) {
    $validation->reset();
    $validation->setRules($userAccountRules);

    if (! $validation->run($user)) {
        // handle validation errors
    }
}
```

验证单个值

check() 方法根据规则验证单个值。第一个参数 $value 是待验证值，第二个参数 $rule 是验证规则，可选第三个参数 $errors 是自定义错误信息。

```
if ($validation->check($value, 'required')) {
    // $value is valid.
}
```

备注： v4.4.0 之前，此方法的第二个参数 $rule 类型限定为 string。v4.4.0 及之后版本移除了类型限定以支持数组。

备注： 此方法内部调用 setRule() 设置规则。

获取已验证数据

在 4.4.0 版本加入.

实际已验证数据可通过 getValidated() 方法获取。此方法返回仅包含通过验证规则的元素数组。

```
$validation = service('validation');
$validation->setRules([
    'username' => 'required',
    'password' => 'required|min_length[10]',
```

(续下页)

(接上页)

```
]);

$data = [
    'username'   => 'john',
    'password'   => 'BPi-$Swu7U5lm$dX',
    'csrf_token' => '8b9218a55906f9dcc1dc263dce7f005a',
];

if ($validation->run($data)) {
    $validatedData = $validation->getValidated();
    // $validatedData = [
    //     'username' => 'john',
    //     'password' => 'BPi-$Swu7U5lm$dX',
    // ];
}
```

```
// In Controller.

if (! $this->validateData($data, [
    'username' => 'required',
    'password' => 'required|min_length[10]',
])) {
    // The validation failed.
    return view('login', [
        'errors' => $this->validator->getErrors(),
    ]);
}

// The validation was successful.

// Get the validated data.
$validData = $this->validator->getValidated();
```

将验证规则保存到配置文件

验证类的一个优点是可以将所有应用的验证规则存储在配置文件中。你可以将规则组织成”组”，每次运行验证时指定不同组。

如何保存规则

要存储验证规则，只需在 `Config\Validation` 类中创建新的公共属性，属性名即组名。该元素将保存包含验证规则的数组。如前所示，验证数组原型如下：

```
<?php

namespace Config;

// ...

class Validation extends BaseConfig
{
    // ...

    public array $signup = [
        'username'     => 'required|max_length[30]',
        'password'     => 'required|max_length[255]',
        'pass_confirm' => 'required|max_
↪length[255]|matches[password]',
        'email'        => 'required|max_length[254]|valid_email',
    ];

    // ...
}
```

如何指定规则组

调用 `run()` 方法时指定要使用的组:

```
$validation->run($data, 'signup');
```

如何保存错误信息

也可在配置文件中通过创建与组同名并附加 `_errors` 的属性来存储自定义错误信息。使用该组时会自动应用这些错误信息:

```
<?php

namespace Config;

// ...

class Validation extends BaseConfig
{
    // ...

    public array $signup = [
        'username'     => 'required|max_length[30]',
        'password'     => 'required|max_length[255]',
        'pass_confirm' => 'required|max_
↪length[255]|matches[password]',
        'email'        => 'required|max_length[254]|valid_email',
    ];

    public array $signup_errors = [
        'username' => [
            'required' => 'You must choose a username.',
        ],
        'email' => [
            'valid_email' => 'Please check the Email field. It does␣
↪not appear to be valid.',
        ],
    ];
```

(续下页)

(接上页)

```
    // ...
}
```

或通过数组传递所有设置:

```
<?php

namespace Config;

// ...

class Validation extends BaseConfig
{
    // ...

    public array $signup = [
        'username' => [
            'rules'  => 'required|max_length[30]',
            'errors' => [
                'required' => 'You must choose a Username.',
            ],
        ],
        'email' => [
            'rules'  => 'required|max_length[254]|valid_email',
            'errors' => [
                'valid_email' => 'Please check the Email field. It␣
↪does not appear to be valid.',
            ],
        ],
    ];

    // ...
}
```

数组格式详见设置自定义错误信息。

获取和设置规则组

获取规则组

此方法从验证配置获取规则组：

```
$validation->getRuleGroup('signup');
```

设置规则组

此方法将验证配置中的规则组设置到验证服务：

```
$validation->setRuleGroup('signup');
```

验证占位符

验证类提供了一种基于传入数据替换规则部分内容的简便方法。这听起来可能有些晦涩，但在使用 `is_unique` 验证规则时特别有用。

占位符是用花括号包裹的传入数据字段名（或数组键）。它将被匹配传入字段的 **值**替换。以下示例可阐明此概念：

```
$validation->setRules([
    'id'    => 'max_length[19]|is_natural_no_zero',
    'email' => 'required|max_length[254]|valid_email|is_
↪unique[users.email,id,{id}]',
]);
```

> **警告：** 自 v4.3.5 起，出于安全考虑必须为占位符字段（上例中的 `id` 字段）设置验证规则。因为攻击者可能向应用发送任意数据。

在此规则集中，声明邮箱地址应在数据库中唯一，除了 id 与占位符值匹配的行。假设表单 POST 数据如下：

```
$_POST = [
    'id'    => 4,
```

(续下页)

(接上页)

```
    'email' => 'foo@example.com',
];
```

则 {id} 占位符会被替换为数字 **4**，形成调整后的规则：

```
$validation->setRules([
    'id'    => 'max_length[19]|is_natural_no_zero',
    'email' => 'required|max_length[254]|valid_email|is_
↪unique[users.email,id,4]',
]);
```

因此验证邮箱唯一性时将忽略数据库中 id=4 的行。

备注： 自 v4.3.5 起，若占位符（如 id）值未通过验证，占位符将不被替换。

只要确保传入的动态键名不与表单数据冲突，这也可用于在运行时创建更动态的规则。

处理错误

验证库提供多种方法帮助你设置错误信息、提供自定义错误信息及获取一个或多个错误信息用于显示。

默认情况下，错误信息来自 **system/Language/en/Validation.php** 中的语言字符串，每个规则对应一个条目。若要更改默认信息，可创建 **app/Language/en/Validation.php** 文件（和/或对应语言环境文件夹），并在其中放置要修改的错误信息键值对。

设置自定义错误信息

setRule() 和 setRules() 方法均可接受自定义错误信息数组作为最后一个参数，为每个字段提供特定错误信息。若未提供自定义信息，则使用默认值。

有两种方式提供自定义错误信息：

作为最后一个参数：

```
$validation->setRules(
    [
```

(续下页)

(接上页)

```
        'username' => 'required|max_length[30]|is_unique[users.
↪username]',
        'password' => 'required|max_length[254]|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?',
        ],
    ],
);
```

或标签式:

```
$validation->setRules([
    'username' => [
        'label'  => 'Username',
        'rules'  => 'required|max_length[30]|is_unique[users.
↪username]',
        'errors' => [
            'required' => 'All accounts must have {field} provided',
        ],
    ],
    'password' => [
        'label'  => 'Password',
        'rules'  => 'required|max_length[255]|min_length[10]',
        'errors' => [
            'min_length' => 'Your {field} is too short. You want to␣
↪get hacked?',
        ],
    ],
]);
```

若要在信息中包含字段”人类可读”名称、规则允许的可选参数（如 max_length）或验证值，可在信息中添加 `{field}`、`{param}` 和 `{value}` 标签:

```
'min_length' => '提供的值 ({value}) 对于 {field} 必须至少 {param}␣
↪个字符。'
```

对于人类名称为”用户名”、规则为 `min_length[6]`、值为“Pizza”的字段，错误信息将显示:”提供的值 (Pizza) 对于用户名必须至少 6 个字符。”

> **警告:** 通过 `getErrors()` 或 `getError()` 获取错误信息时，信息未进行 HTML 转义。若使用用户输入数据（如 `{value}`）生成错误信息，可能包含 HTML 标签。若未转义直接显示，可能导致 XSS 攻击。

备注: 使用标签式错误信息时，若传递第二个参数给 `setRules()`，它会被第一个参数的值覆盖。

信息与验证标签的翻译

要使用语言文件中的翻译字符串，可使用点语法。假设翻译文件位于 **app/Languages/en/Rules.php**，可如下使用定义的语言行:

```
$validation->setRules([
    'username' => [
        'label'  => 'Rules.username',
        'rules'  => 'required|max_length[30]|is_unique[users.
↪username]',
        'errors' => [
            'required' => 'Rules.username.required',
        ],
    ],
    'password' => [
        'label'  => 'Rules.password',
        'rules'  => 'required|max_length[255]|min_length[10]',
        'errors' => [
            'min_length' => 'Rules.password.min_length',
        ],
    ],
```

(续下页)

(接上页)

```
]);
```

获取所有错误

要获取所有失败字段的错误信息，可使用 `getErrors()` 方法:

```
$errors = $validation->getErrors();
/*
 * Produces:
 * [
 *     'field1' => 'error message',
 *     'field2' => 'error message',
 * ]
 */
```

若无错误，返回空数组。

使用通配符（`*`）时，错误将指向特定字段，用相应键替换星号:

```
// 数据
'contacts' => [
    'friends' => [
        [
            'name' => 'Fred Flinstone',
        ],
        [
            'name' => '',
        ],
    ]
]

// 规则
'contacts.friends.*.name' => 'required'

// 错误为
'contacts.friends.1.name' => 'contacts.friends.*.name 字段是必填项。
↪'
```

获取单个错误

可用 `getError()` 方法获取单个字段错误。唯一参数是字段名：

```
$error = $validation->getError('username');
```

若无错误，返回空字符串。

备注： 使用通配符时，所有匹配通配符的错误将合并为一行，用换行符分隔。

检查是否存在错误

可用 `hasError()` 方法检查是否存在错误。唯一参数是字段名：

```
if ($validation->hasError('username')) {
    echo $validation->getError('username');
}
```

指定带通配符的字段时，检查所有匹配错误：

```
/*
 * For errors:
 * [
 *     'foo.0.bar'   => 'Error',
 *     'foo.baz.bar' => 'Error',
 * ]
 */

// returns true
$validation->hasError('foo.*.bar');
```

重定向与验证错误

PHP 请求间不共享数据。因此若验证失败后重定向，新请求中不会有验证错误，因为验证在上次请求中运行。

此时需使用表单辅助函数*validation_errors()*、*validation_list_errors()*和*validation_show_error()*。这些函数检查存储在会话中的验证错误。

要将验证错误存入会话，需将 `withInput()` 与*redirect()* 结合使用：

```
// In Controller.
if (! $this->validateData($data, $rules)) {
    return redirect()->back()->withInput();
}
```

自定义错误显示

调用 `$validation->listErrors()` 或 `$validation->showError()` 时，会在后台加载视图文件决定错误显示方式。默认情况下，用 `errors` class 包裹 div。可轻松创建新视图并在应用中复用。

创建视图

第一步是创建自定义视图。这些视图可放在 `view()` 方法能定位的任何位置，如标准视图目录或命名空间视图文件夹。例如，可在 **app/Views/_errors_list.php** 创建新视图：

```
<?php if (! empty($errors)): ?>
    <div class="alert alert-danger" role="alert">
        <ul>
        <?php foreach ($errors as $error): ?>
            <li><?= esc($error) ?></li>
        <?php endforeach ?>
        </ul>
    </div>
<?php endif ?>
```

视图中可用 `$errors` 数组包含错误列表，键为字段名，值为错误信息：

```
$errors = [
    'username' => 'The username field must be unique.',
    'email'    => 'You must provide a valid email address.',
];
```

实际上有两种视图类型。第一种包含所有错误数组，如前所述。第二种更简单，仅包含单个变量 `$error`，用于 `showError()` 方法指定字段时:

```
<span class="help-block"><?= esc($error) ?></span>
```

配置

创建视图后，需让验证库知晓这些视图。打开 **app/Config/Validation.php** 文件，你会找到 `$templates` 属性。在此可以列出任意数量的自定义视图，并为其指定可引用的简短别名。如果添加上文示例文件，配置示例如下:

```
<?php

namespace Config;

// ...

class Validation extends BaseConfig
{
    // ...

    public array $templates = [
        'list'    => 'CodeIgniter\Validation\Views\list',
        'single'  => 'CodeIgniter\Validation\Views\single',
        'my_list' => '_errors_list',
    ];

    // ...
}
```

指定模板

通过别名指定模板作为 `listErrors()` 的第一个参数:

```
<?= $validation->listErrors('my_list') ?>
```

显示字段特定错误时，可将别名作为 `showError()` 的第二个参数:

```
<?= $validation->showError('username', 'my_single') ?>
```

创建自定义规则

使用规则类

规则存储在简单的命名空间类中，只要自动加载器能找到即可。这些文件称为规则集。

添加规则集

要添加新规则集，请编辑 **app/Config/Validation.php** 文件并将新文件加入 `$ruleSets` 数组:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;
use CodeIgniter\Validation\CreditCardRules;
use CodeIgniter\Validation\FileRules;
use CodeIgniter\Validation\FormatRules;
use CodeIgniter\Validation\Rules;

class Validation extends BaseConfig
{
    // ...

    public array $ruleSets = [
        Rules::class,
        FormatRules::class,
```

(续下页)

(接上页)

```
        FileRules::class,
        CreditCardRules::class,
    ];

    // ...
}
```

可通过两种方式添加：使用完全限定类名的字符串形式，或如示例所示使用 `::class` 后缀。使用 `::class` 后缀的主要优势是在高级 IDE 中能提供额外的导航功能。

创建规则类

在该文件内，每个方法即代表一条规则，必须将待验证值作为第一个参数，且必须返回布尔值 true 或 false 表示验证是否通过：

```
<?php

class MyRules
{
    public function even($value): bool
    {
        return (int) $value % 2 === 0;
    }
}
```

默认情况下，系统会在 **system/Language/en/Validation.php** 中查找错误信息使用的语言字符串。要为自定义规则提供默认错误信息，可将其置于 **app/Language/en/Validation.php**（和/或替代 `en` 的其他语言环境对应文件夹）。若需使用默认 **Validation.php** 之外的其他语言文件，可通过在第二个参数（若规则需要处理参数，则为第四个参数）以引用方式接受 `&$error` 变量来提供错误信息：

```
<?php

class MyRules
{
    public function even($value, ?string &$error = null): bool
    {
```

(续下页)

(接上页)

```
        if ((int) $value % 2 !== 0) {
            $error = lang('myerrors.evenError');

            return false;
        }

        return true;
    }
}
```

使用自定义规则

新自定义规则可像其他规则一样使用：

```
$validation->setRules([
    'foo' => 'required|max_length[19]|even',
]);
```

允许参数

若方法需要参数，函数至少需要三个参数：

1. 待验证值（$value）
2. 参数字符串（$params）
3. 表单提交的所有数据数组（$data）
4. （可选）自定义错误字符串（&$error）

> **警告：** $data 中的字段值未经验证（或可能无效）。使用未验证输入数据是漏洞来源。在使用数据前必须在自定义规则中执行必要验证。

$data 数组对于像 required_with 这样需要检查其他字段值的规则特别有用：

```
<?php
```

(续下页)

(接上页)

```
class MyRules
{
    public function required_with($value, string $params, array
↪$data): bool
    {
        $params = explode(',', $params);

        // 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 = $this->required($value ?? '');

        if ($present) {
            return true;
        }

        // Still here? Then we fail this test if
        // any of the fields are present in $data
        // as $fields in the list
        $requiredFields = [];

        foreach ($params 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, static fn (
↪$item) => ! empty($data[$item]));

        return empty($requiredFields);
    }
}
```

使用闭包规则

在 4.3.0 版本加入.

若自定义规则仅需在应用中使用一次，可使用闭包代替规则类。

需为验证规则使用数组：

```
$validation->setRules(
    [
        'foo' => [
            'required',
            static fn ($value) => (int) $value % 2 === 0,
        ],
    ],
    [
        // Errors
        'foo' => [
            // Specify the array key for the closure rule.
            1 => 'The value is not even.',
        ],
    ],
);

if (! $validation->run($data)) {
    // handle validation errors
}
```

必须为闭包规则设置错误信息。指定错误信息时，需设置闭包规则的数组键。上例中，`required` 规则键为 `0`，闭包为 `1`。

或使用以下参数：

```
$validation->setRules([
    'foo' => [
        'required',
        static function ($value, $data, &$error, $field) {
            if ((int) $value % 2 === 0) {
                return true;
            }
```

(续下页)

(接上页)

```
            $error = 'The value is not even.';

            return false;
        },
    ],
]);
```

使用可调用规则

在 4.5.0 版本加入.

若想用数组回调作为规则，可替代闭包规则。

需为验证规则使用数组：

```
namespace App\Controllers;

class Form extends BaseController
{
    // Define a custom validation rule.
    public function _ruleEven($value): bool
    {
        return (int) $value % 2 === 0;
    }

    public function process()
    {
        // ...

        $validation = service('validation');
        $validation->setRules(
            [
                'foo' => [
                    'required',
                    // Specify the method in this controller as a␣
↪rule.
                    [$this, '_ruleEven'],
                ],
```

(续下页)

(接上页)

```
            ],
            [
                // Errors
                'foo' => [
                    // Specify the array key for the callable rule.
                    1 => 'The value is not even.',
                ],
            ],
        );

        if (! $validation->run($data)) {
            // handle validation errors
        }

        // ...
    }
}
```

必须为可调用规则设置错误信息。指定错误信息时，需设置可调用规则的数组键。上例中，required 规则键为 0，可调用规则为 1。

或使用以下参数：

```
namespace App\Controllers;

use Config\Services;

class Form extends BaseController
{
    // Define a custom validation rule.
    public function _ruleEven($value, $data, &$error, $field): bool
    {
        if ((int) $value % 2 === 0) {
            return true;
        }

        $error = 'The value is not even.';
```

(续下页)

(接上页)

```
        return false;
    }

    // ...
}
```

可用规则

备注: 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` 前后也不能有空格。

```
// is_unique[table.field,ignore_field,ignore_value]

$validation->setRules([
    'name' => "max_length[36]|is_unique[supplier.name,uuid, {$uuid}]
↪", // is not ok
    'name' => "max_length[36]|is_unique[supplier.name,uuid,{$uuid} ]
↪", // is not ok
    'name' => "max_length[36]|is_unique[supplier.name,uuid,{$uuid}]
↪",  // is ok
    'name' => 'max_length[36]|is_unique[supplier.name,uuid,{uuid}]',
↪   // is ok - see "Validation Placeholders"
]);
// Warning: If `$uuid` is a user input, be sure to validate the␣
↪format of the value before using it.
// Otherwise, it is vulnerable.
```

通用规则

以下是所有可用的原生规则列表:

规则	参数	描述
alpha	否	若字段包含非 ASCII 字母字符则失败。

规则	参数	描述
alpha_dash	否	若字段包含非字母数字、下划线或短横线（ASCII）则失败。
alpha_numeric	否	若字段包含非 ASCII 字母数字字符则失败。
alpha_numeric_punct	否	若字段包含非字母数字、空格或以下标点则失败：~ （波浪符）、！（
alpha_numeric_space	否	若字段包含非字母数字或空格（ASCII）则失败。
alpha_space	否	若字段包含非字母或空格（ASCII）则失败。
decimal	否	若字段包含非十进制数则失败（允许 +/- 符号）。
differs	是	若字段值与参数字段相同则失败。
exact_length	是	若字段长度不等于参数值则失败（支持逗号分隔多值）。
field_exists	是	若字段不存在则失败（v4.5.0 新增）。
greater_than	是	若字段值小于等于参数值或非数字则失败。
greater_than_equal_to	是	若字段值小于参数值或非数字则失败。
hex	否	若字段包含非十六进制字符则失败。
if_exist	否	仅当字段存在于验证数据中时才进行验证。
in_list	是	若字段值不在指定列表中则失败。
integer	否	若字段包含非整数值则失败。
is_natural	否	若字段包含非自然数（0,1,2…）则失败。
is_natural_no_zero	否	若字段包含非自然数（1,2,3…）则失败。
is_not_unique	是	检查数据库中是否存在给定的值。可以通过字段/值过滤器忽略记录（
is_unique	是	检查字段值是否存在于数据库中。可以可选地设置要忽略的列和值, 在
less_than	是	若字段值大于等于参数值或非数字则失败。
less_than_equal_to	是	若字段值大于参数值或非数字则失败。
matches	是	值必须与参数字段值匹配。
max_length	是	若字段长度超过参数值则失败。
min_length	是	若字段长度短于参数值则失败。
not_in_list	是	若字段值在指定列表中则失败。
numeric	否	若字段包含非数字字符则失败。
permit_empty	否	允许字段为空数组、空字符串、null 或 false。
regex_match	是	若字段不匹配正则表达式则失败。
required	否	若字段为空数组、空字符串、null 或 false 则失败。
required_with	是	当其他任一字段非空时本字段必填。
required_without	是	当其他任一字段为空时本字段必填。
string	否	通用字符串验证（替代 **alpha*** 系列规则）。
timezone	否	若字段不符合时区标识符则失败（基于 timezone_identifiers_list()）。

规则	参数	描述
valid_base64	否	若字段包含非合法 Base64 字符则失败。
valid_cc_number	是	验证信用卡号是否与指定提供程序使用的格式匹配。当前支持的提供程
valid_date	是	如果字段不包含有效的日期，则验证失败。任何 strtotime() 接受的字符
valid_email	否	若字段不符合邮箱格式则失败。
valid_emails	否	若逗号分隔列表中任一邮箱无效则失败。
valid_ip	是	如果提供的 IP 无效，则失败。接受一个可选参数 `ipv4` 或 `ipv6` 来指
valid_json	否	若字段非合法 JSON 字符串则失败。
valid_url	否	如果字段不包含（宽松意义上的）URL，则验证失败。包括可能作为主
valid_url_strict	是	如果字段不包含有效的 URL，则验证失败。你可以选择性地指定一个

备注： 你还可以使用任何返回布尔值且至少接受一个参数（待验证字段数据）的原生 PHP 函数。

重要： 验证库 **从不修改**待验证数据。

文件上传规则

验证上传文件时，必须使用专门针对文件验证的规则。

重要： 只能使用下表列出的规则验证文件。若在文件验证规则数组或字符串中添加通用规则（如 `permit_empty`），将导致文件验证失效。

由于文件上传字段的值不存在于常规数据中，而是存储在 `$_FILES` 全局变量，因此输入字段名需要被使用两次：一次作为常规字段名，另一次作为文件相关规则的第一个参数:

```
// 在 HTML 中
<input type="file" name="avatar">

// 在控制器中
```

(续下页)

(接上页)

```
$this->validateData([], [
    'avatar' => 'uploaded[avatar]|max_size[avatar,1024]',
]);
```

另见文件上传表单教程。

规则	参数	描述	示例
uploaded	是	如果参数的名称与任何已上传文件的名称不匹配，则验证失败。如果你希望文件上传是可选的（非必需的），请不要定义此规则。	uploaded[字段名]
max_size	是	若文件大小超过参数值（单位 KB）或 php.ini 中 upload_max_filesize 限制则失败。	max_size[字段名,2048]
max_dims	是	若图片尺寸超过指定宽高则失败，参数依次为字段名、最大宽度、最大高度。若非图片文件也会失败。	max_dims[字段名,300,150]
min_dims	是	若图片尺寸未达最小宽高则失败，参数依次为字段名、最小宽度、最小高度。(此规则在 v4.6.0 版本新增)	min_dims[字段名,300,150]
mime_in	是	若文件 MIME 类型不在参数列表中则失败。	mime_in[字段名,image/png,image/jpeg]
ext_in	是	若文件扩展名不在参数列表中则失败。	ext_in[字段名,png,jpg,gif]
is_image	是	若文件无法被识别为图片则失败。	is_image[字段名]

文件验证规则同时适用于单文件和多文件上传场景。

7.2 辅助函数

辅助函数是一些程序功能的集合。另请参阅辅助函数。

7.2.1 数组辅助函数

数组辅助函数提供了几个函数来简化数组的更复杂用法。它不打算重复 PHP 提供的任何现有功能 - 除非是为了极大地简化它们的用法。

- 加载此辅助函数
- 可用函数

加载此辅助函数

使用以下代码加载此辅助函数:

```
helper('array');
```

可用函数

以下函数可用:

dot_array_search(*string $search*, *array $values*)

参数

- **$search** (string) –用点表示法描述如何在数组中搜索的字符串
- **$values** (array) –要搜索的数组

返回

在数组中找到的值, 如果没有找到则为 null

返回类型

mixed

该方法允许你使用点表示法在数组中搜索特定键, 并允许使用通配符 *。给定以下数组:

```
$data = [
    'foo' => [
        'buzz' => [
            'fizz' => 11,
        ],
        'bar' => [
            'baz' => 23,
        ],
    ],
];
```

我们可以使用搜索字符串 `foo.buzz.fizz` 定位 `fizz` 的值。类似地，可以使用 `foo.bar.baz` 找到 `baz` 的值:

```
// Returns: 11
$fizz = dot_array_search('foo.buzz.fizz', $data);

// Returns: 23
$baz = dot_array_search('foo.bar.baz', $data);
```

你可以使用星号（`*`）作为通配符来替换任何段。找到时, 它将搜索所有子节点直到找到它。如果你不知道值, 或如果你的值具有数值索引, 这很方便:

```
// Returns: 23
$baz = dot_array_search('foo.*.baz', $data);
```

如果数组键包含点（`.`），则可以用反斜杠（`\`）转义键:

```
$data = [
    'foo' => [
        'bar.baz' => 23,
    ],
    'foo.bar' => [
        'baz' => 43,
    ],
];

// Returns: 23
$barBaz = dot_array_search('foo.bar\.baz', $data);
```

(续下页)

(接上页)

```
// Returns: 43
$fooBar = dot_array_search('foo\.bar.baz', $data);
```

备注: 在 v4.2.0 之前, 由于一个 bug, `dot_array_search('foo.bar.baz', ['foo' => ['bar' => 23]])` 返回的是 23。v4.2.0 及更高版本返回 `null`。

array_deep_search(*$key*, *array $array*)

参数

- **$key** (mixed) –目标键
- **$array** (array) –要搜索的数组

返回

在数组中找到的值, 如果没有找到则为 null

返回类型

mixed

从一个深度不确定的数组返回具有键值的元素的值

array_sort_by_multiple_keys(*array &$array*, *array $sortColumns*)

参数

- **$array** (array) –要排序的数组 (通过引用传递)。
- **$sortColumns** (array) –要排序的数组键及各自的 PHP 排序标志的关联数组

返回

排序是否成功

返回类型

bool

此方法以分层方式根据一个或多个键的值对多维数组的元素进行排序。例如, 从某个模型的 `find()` 函数返回以下数组:

```
$players = [
    0 => [
```

(续下页)

(接上页)

```
        'name'     => 'John',
        'team_id'  => 2,
        'position' => 3,
        'team'     => [
            'id'    => 1,
            'order' => 2,
        ],
    ],
    1 => [
        'name'     => 'Maria',
        'team_id'  => 5,
        'position' => 4,
        'team'     => [
            'id'    => 5,
            'order' => 1,
        ],
    ],
    2 => [
        'name'     => 'Frank',
        'team_id'  => 5,
        'position' => 1,
        'team'     => [
            'id'    => 5,
            'order' => 1,
        ],
    ],
];
```

现在按两个键对该数组进行排序。请注意, 该方法支持使用点表示法访问更深层数组级别中的值, 但不支持通配符:

```
array_sort_by_multiple_keys($players, [
    'team.order' => SORT_ASC,
    'position'   => SORT_ASC,
]);
```

现在 `$players` 数组已根据每个球员 `team` 子数组中的 `order` 值排序。如果对几个球员此值相等, 则这些球员将根据其 `position` 进行排序。结果数组为:

```
$players = [
    0 => [
        'name'     => 'Frank',
        'team_id'  => 5,
        'position' => 1,
        'team'     => [
            'id'    => 5,
            'order' => 1,
        ],
    ],
    1 => [
        'name'     => 'Maria',
        'team_id'  => 5,
        'position' => 4,
        'team'     => [
            'id'    => 5,
            'order' => 1,
        ],
    ],
    2 => [
        'name'     => 'John',
        'team_id'  => 2,
        'position' => 3,
        'team'     => [
            'id'    => 1,
            'order' => 2,
        ],
    ],
];
```

同样, 该方法也可以处理对象数组。在上面的示例中, 每个 `player` 都可能由一个数组表示, 而 `teams` 是对象。该方法将检测每个嵌套级别中的元素类型并相应处理。

array_flatten_with_dots(*iterable $array*[, *string $id* = '']) → array

参数

- **$array** (`iterable`) –要打平的多维数组
- **$id** (`string`) –可选的 ID 以添加到外部键前面。内部使用以展

平键。

返回类型

array

返回

打平的数组

此函数使用点作为键的分隔符, 将多维数组展平为单个键值对数组。

```
$arrayToFlatten = [
    'personal' => [
        'first_name' => 'john',
        'last_name'  => 'smith',
        'age'        => '26',
        'address'    => 'US',
    ],
    'other_details' => 'marines officer',
];

$flattened = array_flatten_with_dots($arrayToFlatten);
```

检查后, $flattened 等于:

```
[
    'personal.first_name' => 'john',
    'personal.last_name'  => 'smith',
    'personal.age'        => '26',
    'personal.address'    => 'US',
    'other_details'       => 'marines officer',
];
```

用户可以自己使用 $id 参数, 但不需要这样做。该函数在内部使用此参数来跟踪展平后的键。如果用户将提供初始 $id, 它将添加到所有键前面。

```
// using the same data from above
$flattened = array_flatten_with_dots($arrayToFlatten, 'foo_');
/*
 * $flattened is now:
 * [
 *     'foo_personal.first_name' => 'john',
```

(续下页)

(接上页)

```
*     'foo_personal.last_name'  => 'smith',
*     'foo_personal.age'        => '26',
*     'foo_personal.address'    => 'US',
*     'foo_other_details'       => 'marines officer',
* ]
*/
```

array_group_by(*array $array*, *array $indexes*[, *bool $includeEmpty = false*]) → array

参数

- **$array** (array) –数据行（很可能来自查询结果）
- **$indexes** (array) –要按索引值分组的索引。遵循点语法
- **$includeEmpty** (bool) –如果为 true，则不过滤掉 `null` 和 `''` 值

返回类型

array

返回

按索引值分组的数组

该函数允许你按索引值将数据行分组在一起。返回的数组的深度等于作为参数传递的索引数。

以下示例显示了一些数据（例如从 API 加载的数据）和嵌套数组。

```
$employees = [
    [
        'id'         => 1,
        'first_name' => 'Urbano',
        'gender'     => null,
        'hr'         => [
            'country'    => 'Canada',
            'department' => 'Engineering',
        ],
    ],
    [
        'id'         => 2,
        'first_name' => 'Case',
```

(续下页)

(接上页)

```
        'gender'     => 'Male',
        'hr'         => [
            'country'    => null,
            'department' => 'Marketing',
        ],
    ],
    [
        'id'         => 3,
        'first_name' => 'Emera',
        'gender'     => 'Female',
        'hr'         => [
            'country'    => 'France',
            'department' => 'Engineering',
        ],
    ],
    [
        'id'         => 4,
        'first_name' => 'Richy',
        'gender'     => null,
        'hr'         => [
            'country'    => null,
            'department' => 'Sales',
        ],
    ],
    [
        'id'         => 5,
        'first_name' => 'Mandy',
        'gender'     => null,
        'hr'         => [
            'country'    => 'France',
            'department' => 'Sales',
        ],
    ],
    [
        'id'         => 6,
        'first_name' => 'Risa',
        'gender'     => 'Female',
        'hr'         => [
```

(续下页)

(接上页)

```
            'country'    => null,
            'department' => 'Engineering',
        ],
    ],
    [
        'id'         => 7,
        'first_name' => 'Alfred',
        'gender'     => 'Male',
        'hr'         => [
            'country'    => 'France',
            'department' => 'Engineering',
        ],
    ],
    [
        'id'         => 8,
        'first_name' => 'Tabby',
        'gender'     => 'Male',
        'hr'         => [
            'country'    => 'France',
            'department' => 'Marketing',
        ],
    ],
    [
        'id'         => 9,
        'first_name' => 'Ario',
        'gender'     => 'Male',
        'hr'         => [
            'country'    => null,
            'department' => 'Sales',
        ],
    ],
    [
        'id'         => 10,
        'first_name' => 'Somerset',
        'gender'     => 'Male',
        'hr'         => [
            'country'    => 'Germany',
            'department' => 'Marketing',
```

(续下页)

(接上页)

```
        ],
    ],
];
```

我们首先想要按 gender 分组，然后按 hr.department 分组（最大深度为 2）。首先排除空值的结果如下：

```
$result = array_group_by($employees, ['gender', 'hr.department
↪']);

$result = [
    'Male' => [
        'Marketing' => [
            [
                'id'         => 2,
                'first_name' => 'Case',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => null,
                    'department' => 'Marketing',
                ],
            ],
            [
                'id'         => 8,
                'first_name' => 'Tabby',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => 'France',
                    'department' => 'Marketing',
                ],
            ],
            [
                'id'         => 10,
                'first_name' => 'Somerset',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => 'Germany',
                    'department' => 'Marketing',
```

(续下页)

(接上页)

```
                ],
            ],
        ],
        'Engineering' => [
            [
                'id'         => 7,
                'first_name' => 'Alfred',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => 'France',
                    'department' => 'Engineering',
                ],
            ],
        ],
        'Sales' => [
            [
                'id'         => 9,
                'first_name' => 'Ario',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => null,
                    'department' => 'Sales',
                ],
            ],
        ],
    ],
    'Female' => [
        'Engineering' => [
            [
                'id'         => 3,
                'first_name' => 'Emera',
                'gender'     => 'Female',
                'hr'         => [
                    'country'    => 'France',
                    'department' => 'Engineering',
                ],
            ],
            [
```

(续下页)

(接上页)

```
                'id'         => 6,
                'first_name' => 'Risa',
                'gender'     => 'Female',
                'hr'         => [
                    'country'    => null,
                    'department' => 'Engineering',
                ],
            ],
        ],
    ],
];
```

这里是相同的代码，但这次我们想要包括空值：

```
$result = array_group_by($employees, ['gender', 'hr.department
↪'], true);

$result = [
    '' => [
        'Engineering' => [
            [
                'id'         => 1,
                'first_name' => 'Urbano',
                'gender'     => null,
                'hr'         => [
                    'country'    => 'Canada',
                    'department' => 'Engineering',
                ],
            ],
        ],
        'Sales' => [
            [
                'id'         => 4,
                'first_name' => 'Richy',
                'gender'     => null,
                'hr'         => [
                    'country'    => null,
                    'department' => 'Sales',
```

(续下页)

(接上页)

```
                ],
            ],
            [
                'id'         => 5,
                'first_name' => 'Mandy',
                'gender'     => null,
                'hr'         => [
                    'country'    => 'France',
                    'department' => 'Sales',
                ],
            ],
        ],
    ],
    'Male' => [
        'Marketing' => [
            [
                'id'         => 2,
                'first_name' => 'Case',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => null,
                    'department' => 'Marketing',
                ],
            ],
            [
                'id'         => 8,
                'first_name' => 'Tabby',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => 'France',
                    'department' => 'Marketing',
                ],
            ],
            [
                'id'         => 10,
                'first_name' => 'Somerset',
                'gender'     => 'Male',
                'hr'         => [
```

(续下页)

(接上页)

```
                    'country'    => 'Germany',
                    'department' => 'Marketing',
                ],
            ],
        ],
        'Engineering' => [
            [
                'id'         => 7,
                'first_name' => 'Alfred',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => 'France',
                    'department' => 'Engineering',
                ],
            ],
        ],
        'Sales' => [
            [
                'id'         => 9,
                'first_name' => 'Ario',
                'gender'     => 'Male',
                'hr'         => [
                    'country'    => null,
                    'department' => 'Sales',
                ],
            ],
        ],
    ],
    'Female' => [
        'Engineering' => [
            [
                'id'         => 3,
                'first_name' => 'Emera',
                'gender'     => 'Female',
                'hr'         => [
                    'country'    => 'France',
                    'department' => 'Engineering',
                ],
```

(续下页)

(接上页)

```
            ],
            [
                'id'         => 6,
                'first_name' => 'Risa',
                'gender'     => 'Female',
                'hr'         => [
                    'country'    => null,
                    'department' => 'Engineering',
                ],
            ],
        ],
    ],
];
```

7.2.2 Cookie 辅助函数

Cookie 辅助函数文件包含了帮助处理 cookie 的函数。

- 加载此辅助函数
- 可用函数

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('cookie');
```

可用函数

以下函数可用:

set_cookie(*$name*[, *$value* = ''[, *$expire* = *0*[, *$domain* = ''[, *$path* = '/'[, *$prefix* = ''[, *$secure* = *false*[, *$httpOnly* = *false*[, *$sameSite* = '']]]]]]]])

参数

- **$name** (array|Cookie|string) –Cookie 名称 或此函数可用的所有参数的关联数组 或 `CodeIgniter\Cookie\Cookie` 的实例
- **$value** (string) –Cookie 值
- **$expire** (int) –到期秒数。如果设置为 `0` 则 cookie 仅在浏览器打开时有效
- **$domain** (string) –Cookie 域名 (通常:.yourdomain.com)
- **$path** (string) –Cookie 路径
- **$prefix** (string) –Cookie 名称前缀。如果为 `''`, 则使用 **app/Config/Cookie.php** 中的默认值
- **$secure** (bool) –是否仅通过 HTTPS 发送 cookie。如果为 `null`, 则使用 **app/Config/Cookie.php** 中的默认值
- **$httpOnly** (bool) –是否从 JavaScript 隐藏 cookie。如果为 `null`, 则使用 **app/Config/Cookie.php** 中的默认值
- **$sameSite** (string) –SameSite cookie 参数的值。如果为 `null`, 则使用 **app/Config/Cookie.php** 中的默认值

返回类型

`void`

备注: 在 v4.2.7 之前, 由于一个 bug, `$secure` 和 `$httpOnly` 的默认值是 `false`, 从不使用 **app/Config/Cookie.php** 中的值。

该辅助函数为设置浏览器 cookie 提供了更友好的语法。有关其用法的描述, 请参阅*Response 库*, 因为此函数是`CodeIgniter\HTTP\Response::setCookie()`的别名。

备注: 这个辅助函数只设置全局响应实例的浏览器 Cookie（由 `Services::response()` 返回）。所以，如果你创建并返回另一个响应实例（例如，如果你调用*redirect()*），这里设置的 Cookie 不会自动发送。

get_cookie(*$index*[, *$xssClean = false*[, *$prefix = ''*]])

参数

- **$index** (`string`) –Cookie 名称
- **$xssClean** (`bool`) –是否对返回的值应用 XSS 过滤
- **$prefix** (`string|null`) –Cookie 名称前缀。如果设置为 `''`，将使用 **app/Config/Cookie.php** 中的默认值。如果设置为 `null`，则没有前缀

返回

cookie 值, 如果未找到则为 null

返回类型

`mixed`

备注: 从 v4.2.1 开始, 引入了第三个参数 `$prefix`, 并且由于一个错误修复, 行为发生了一些变化。详见升级。

这个辅助函数为你提供了更友好的语法来获取浏览器的 Cookie。有关其使用的详细描述，请参考*IncomingRequest 库*，因为这个函数的行为与*CodeIgniter\HTTP\IncomingRequest::getCookie()* 非常相似，只是它还会在前面添加你在 **app/Config/Cookie.php** 文件中设置的 `Config\Cookie::$prefix`。

警告: 使用 XSS 过滤是一个不好的做法。它不能完美地防止 XSS 攻击。在视图中建议使用正确 `$context` 的*esc()*。

delete_cookie(*$name*[, *$domain = ''*[, *$path = '/'*[, *$prefix = ''*]]])

参数

- **$name** (`string`) –Cookie 名称

- **$domain** (string) –Cookie 域名 (通常:.yourdomain.com)
- **$path** (string) –Cookie 路径
- **$prefix** (string) –Cookie 前缀

返回类型

void

允许你删除一个 cookie。除非你设置了自定义路径或其他值, 否则只需要 cookie 的名称。

```
<?php

delete_cookie('name');
```

此函数与*set_cookie()* 其他方面相同, 只是它没有 value 和 expire 参数。

这也只是为删除全局响应实例（由 Services::response() 返回）的浏览器 Cookie 设置浏览器 Cookie。

备注: 当你使用*set_cookie()* 时, 如果 value 设置为空字符串且 expire 设置为 0, 则 cookie 将被删除。如果 value 设置为非空字符串且 expire 设置为 0, 则 cookie 仅在浏览器打开时有效。

你可以在第一个参数中提交值数组, 也可以设置离散参数。

```
<?php

delete_cookie($name, $domain, $path, $prefix);
```

has_cookie(*string $name*[, *?string $value = null*[, *string $prefix = ''*]])

参数

- **$name** (string) –Cookie 名称
- **$value** (string|null) –Cookie 值
- **$prefix** (string) –Cookie 前缀

返回类型

bool

检查在全局响应实例中（由 `Services::response()` 返回）是否存在同名的 Cookie。这是:php:meth::*CodeIgniter\HTTP\Response::hasCookie()* 的别名。

7.2.3 日期辅助函数

日期辅助函数文件包含了帮助处理日期的函数。

- 加载此辅助函数
- 可用函数

备注: 许多之前在 CodeIgniter 3 `date_helper` 中找到的函数已移到 CodeIgniter 4 的*Time* 类中。

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('date');
```

可用函数

以下函数可用:

now([*$timezone = null*])

参数

- **$timezone** (string) –时区

返回

UNIX 时间戳

返回类型

int

备注: 建议使用*Time* 类。使用 `Time::now()->getTimestamp()` 来获取当前的 UNIX 时间戳。

如果未提供时区, 它将通过 `time()` 返回当前的 UNIX 时间戳。

```
<?php

echo now();
```

如果提供任何 PHP 支持的时区, 它将返回一个由时差偏移的时间戳。它与当前的 UNIX 时间戳不同。

如果你不打算将主时间参考设置为任何其他 PHP 支持的时区 (如果你运行一个允许每个用户设置自己的时区设置的站点, 通常会这样做), 那么使用这个函数不会比 PHP 的 `time()` 函数有更多的好处。

timezone_select([*$class = '', $default = '', $what = \DateTimeZone::ALL, $country = null*])

参数

- **$class** (`string`) –可选的要应用于选择字段的类
- **$default** (`string`) –初始选择的默认值
- **$what** (`int`) –DateTimeZone 类常量 (参见 listIdentifiers)
- **$country** (`string`) –一个与 ISO 3166-1 兼容的两字母国家代码 (参见 listIdentifiers)

返回

预格式化的 HTML 选择字段

返回类型

`string`

生成可用时区的 *select* 表单字段 (可选择通过 `$what` 和 `$country` 过滤)。你可以提供一个选项 class 以方便格式化应用于字段, 以及一个默认选择值。

```
<?php

echo timezone_select('custom-select', 'America/New_York');
```

7.2.4 文件系统辅助函数

文件系统辅助函数文件包含了帮助处理文件和目录的函数。

- 加载此辅助函数
- 可用函数

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('filesystem');
```

可用函数

以下函数可用:

directory_map(*$sourceDir*[, *$directoryDepth = 0*[, *$hidden = false*]])

参数

- **$sourceDir** (string) –源目录路径
- **$directoryDepth** (int) –遍历的目录深度 (0 = 完全递归, 1 = 当前目录, 等等)
- **$hidden** (bool) –是否包含隐藏路径

返回

文件数组

返回类型

array

示例:

```
<?php

$map = directory_map('./mydirectory/');
```

备注: 路径几乎总是相对于你的主 **index.php** 文件。

包含在目录中的子文件夹也将被映射。如果你希望控制递归深度, 可以使用第二个参数 (整数)。深度为 1 只会映射顶级目录:

```
<?php

$map = directory_map('./mydirectory/', 1);
```

默认情况下, 返回的数组中不包括隐藏文件, 跳过隐藏目录。要覆盖此行为, 可以将第三个参数设置为 true (布尔值):

```
<?php

$map = directory_map('./mydirectory/', 0, true);
```

每个文件夹名称将是一个数组索引, 其包含的文件将以数字索引。这里是一个典型数组的示例:

```
Array (
    [libraries] => Array
        (
            [0] => benchmark.html
            [1] => config.html
            ["database/"] => Array
                (
                    [0] => query_builder.html
                    [1] => binds.html
                    [2] => configuration.html
                    [3] => connecting.html
                    [4] => examples.html
                    [5] => fields.html
                    [6] => index.html
                    [7] => queries.html
                )
            [2] => email.html
            [3] => file_uploading.html
            [4] => image_lib.html
```

(续下页)

(接上页)

```
            [5] => input.html
            [6] => language.html
            [7] => loader.html
            [8] => pagination.html
            [9] => uri.html
        )
)
```

如果未找到结果, 它将返回一个空数组。

directory_mirror(*$original*, *$target*[, *$overwrite = true*])

参数

- **$original** (string) –原始源目录
- **$target** (string) –目标目的目录
- **$overwrite** (bool) –是否在冲突时覆盖单个文件

递归复制源目录的文件和目录到目标目录, 即“镜像”其内容。

例子:

```
<?php

try {
    directory_mirror($uploadedImages, FCPATH . 'images/');
} catch (\Throwable $e) {
    echo 'Failed to export uploads!';
}
```

你可以通过第三个参数选择更改覆盖行为。

write_file(*$path*, *$data*[, *$mode = 'wb'*])

参数

- **$path** (string) –文件路径
- **$data** (string) –要写入文件的数据
- **$mode** (string) –fopen() 模式

返回

如果写入成功则为 `true`, 如果有错误则为 `false`

返回类型

`bool`

将数据写入路径中指定的文件。如果文件不存在, 则该函数将创建它。

例子:

```
<?php

$data = 'Some file data';

if (! write_file('./path/to/file.php', $data)) {
    echo 'Unable to write the file';
} else {
    echo 'File written!';
}
```

你可以通过第三个参数可选地设置写入模式:

```
<?php

write_file('./path/to/file.php', $data, 'r+');
```

默认模式为 `'wb'`。写入模式选项请参阅 PHP 用户指南的 *fopen() <https://www.php.net/manual/en/function.fopen.php>*。

备注: 为了使此函数能够将数据写入文件, 必须设置其权限以使其可写。如果文件不存在, 则包含它的目录必须可写。

备注: 该路径是相对于你的主站点 **index.php** 文件, 而不是你的控制器或视图文件。CodeIgniter 使用前端控制器, 因此路径始终相对于主站点 index。

备注: 此函数在写入文件时对该文件进行排他锁定。

---

delete_files(*$path*[, *$delDir = false*[, *$htdocs = false*[, *$hidden = false*]]])

参数

- **$path** (string) –目录路径
- **$delDir** (bool) –是否也删除目录
- **$htdocs** (bool) –是否跳过删除.htaccess 和索引页面文件
- **$hidden** (bool) –是否也删除隐藏文件 (以句点开头的文件)

返回

成功为 true, 错误为 false

返回类型

bool

删除提供的路径中包含的所有文件。

例子:

```
<?php

delete_files('./path/to/directory/');
```

如果第二个参数设置为 true, 则提供的根路径中包含的任何目录也将被删除。

例子:

```
<?php

delete_files('./path/to/directory/', true);
```

备注: 文件必须可写或由系统拥有才能被删除。

get_filenames(*$sourceDir*[, *$includePath = false*[, *$hidden = false*[, *$includeDir = true*]]])

参数

- **$sourceDir** (string) –目录路径

- **$includePath** (bool|null) –是否将路径作为文件名的一部分包含;false 不包含路径,null 包含相对于 $sourceDir 的路径,true 包含完整路径
- **$hidden** (bool) –是否包含隐藏文件 (以句点开头的文件)
- **$includeDir** (bool) –是否在数组输出中包含目录

返回

文件名数组

返回类型

array

获取一个服务器路径作为输入, 返回一个包含其中包含的所有文件名的数组。通过将第二个参数设置为‘relative’获取相对路径, 或任何其他非空值以获取完整文件路径, 可以选择将文件路径添加到文件名中。

备注: 在 v4.4.4 之前，由于一个错误，这个函数并未跟随文件夹的符号链接。

示例:

```
<?php

$controllers = get_filenames(APPPATH . 'Controllers/');
```

get_dir_file_info(*$sourceDir*[, *$topLevelOnly = true*])

参数

- **$sourceDir** (string) –目录路径
- **$topLevelOnly** (bool) –是否仅查看指定的目录 (不包括子目录)

返回

包含有关提供目录内容信息的数组

返回类型

array

读取指定的目录并构建一个包含文件名、文件大小、日期和权限的数组。仅当通过将第二个参数设置为 false 强制时, 才读取指定路径中包含的子文件夹, 因为这可能是一个密集操作。

示例:

```
<?php

$models_info = get_dir_file_info(APPPATH . 'Models/');
```

get_file_info(*$file[, $returnedValues = ['name', 'server_path', 'size', 'date']]*)

参数

- **$file** (string) –文件路径
- **$returnedValues** (array|string) –要作为数组或逗号分隔字符串返回的信息类型

返回

包含指定文件的信息的数组, 失败为 false

返回类型

array

根据文件和路径, 返回 (可选地) 名称、路径、大小和 修改日期信息属性。第二个参数允许你明确声明你想要返回的信息。

有效的 $returnedValues 选项有:name、size、date、readable、writeable、executable 和 fileperms。

symbolic_permissions(*$perms*)

参数

- **$perms** (int) –权限

返回

符号权限字符串

返回类型

string

获取数字权限 (例如 fileperms() 返回的) 并返回标准符号表示法的文件权限。

```
<?php

echo symbolic_permissions(fileperms('./index.php'));  // -rw-r--
↪r--
```

octal_permissions(*$perms*)

参数

- **$perms** (int) –权限

返回

八进制权限字符串

返回类型

string

获取数字权限 (例如 fileperms() 返回的) 并返回三字符八进制表示法的文件权限。

```
<?php

echo octal_permissions(fileperms('./index.php')); // 644
```

same_file(*$file1*, *$file2*)

参数

- **$file1** (string) –第一个文件的路径
- **$file2** (string) –第二个文件的路径

返回

两个文件是否具有相同的哈希值并存在

返回类型

boolean

比较两个文件是否相同 (基于它们的 MD5 哈希)。

```
<?php

echo same_file($newFile, $oldFile) ? 'Same!' : 'Different!';
```

set_realpath(*$path*[, *$checkExistence = false*])

参数

- **$path** (string) –路径
- **$checkExistence** (bool) –是否检查路径是否实际存在

返回

绝对路径

返回类型

string

此函数将返回没有符号链接或相对目录结构的服务器路径。可选的第二个参数将在无法解析路径时触发错误。

示例:

```
<?php

$file = '/etc/php5/apache2/php.ini';
echo set_realpath($file); // Prints '/etc/php5/apache2/php.ini'

$non_existent_file = '/path/to/non-exist-file.txt';
echo set_realpath($non_existent_file, true);    // Shows an␣
↪error, as the path cannot be resolved
echo set_realpath($non_existent_file, false);   // Prints '/
↪path/to/non-exist-file.txt'

$directory = '/etc/php5';
echo set_realpath($directory);  // Prints '/etc/php5/'

$non_existent_directory = '/path/to/nowhere';
echo set_realpath($non_existent_directory, true);   // Shows an␣
↪error, as the path cannot be resolved
echo set_realpath($non_existent_directory, false);  // Prints '/
↪path/to/nowhere'
```

7.2.5 表单辅助函数

表单辅助函数文件包含了帮助处理表单的函数。

- 配置
- 加载此辅助函数
- 转义字段值

- 可用函数

配置

从 v4.3.0 开始, `form_helper` 函数中的空 HTML 元素 (如 `<input>`) 默认为兼容 HTML5, 如果你需要兼容 XHTML, 必须在 **app/Config/DocTypes.php** 中将 `$html5` 属性设置为 `false`。

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('form');
```

转义字段值

你可能需要在表单元素中使用 HTML 和诸如引号之类的字符。为了安全地做到这一点, 你需要使用*common function* `esc()`。

考虑以下示例:

```
<?php

$string = 'Here is a string containing "quoted" text.';

?>

<input type="text" name="myfield" value="<?= $string ?>">
```

由于上面的字符串包含一组引号, 它会导致表单中断。`esc()` 函数将 HTML 特殊字符转换, 以便可以安全使用:

```
<input type="text" name="myfield" value="<?= esc($string) ?>">
```

备注: 如果使用此页面上列出的任何表单辅助函数, 并以关联数组的形式传递值, 则表

单值将自动转义, 因此不需要调用此函数。只有在以字符串形式创建自己的表单元素时, 才需要调用它。

可用函数

以下函数可用:

form_open([*$action* = ''[, *$attributes* = ''[, *$hidden* = *[]*]]])

参数

- **$action** (string) –表单操作/目标 URI 字符串
- **$attributes** (mixed) –HTML 属性, 作为数组或转义的字符串
- **$hidden** (array) –隐藏字段定义的数组

返回

一个 HTML 表单开启标签

返回类型

string

创建一个开启的 form 标签, 其 action 的 base 使用你的 `Config\App::$baseURL` 值。它将可选地允许你添加表单属性和隐藏输入字段, 并且将始终根据你的 **app/Config/App.php** 配置文件中的 `$charset` 属性添加 *accept-charset* 属性。

与硬编码你自己的 HTML 相比, 使用此标签的主要好处在于, 如果你的 URL 改变, 它允许你的站点更便携。

这里是一个简单的例子:

```
<?php

echo form_open('email/send');
```

上面的示例将创建一个指向你的站点 URL 加上 "email/send" URI 段的表单, 如下:

```
<form action="http://example.com/index.php/email/send" method=
↪"post" accept-charset="utf-8">
```

你也可以像下面这样添加 `{locale}`:

```
<?php

echo form_open('{locale}/email/send');
```

上面的示例将创建一个指向你的站点 URL 加上当前请求区域设置和 “email/send” URI 段的表单, 如下:

```
<form action="http://example.com/index.php/en/email/send"␣
↪method="post" accept-charset="utf-8">
```

添加属性

可以通过将关联数组作为第二个参数传递来添加属性, 如下所示:

```
<?php

$attributes = ['class' => 'email', 'id' => 'myform'];
echo form_open('email/send', $attributes);
```

或者, 你可以将第二个参数指定为字符串:

```
<?php

echo form_open('email/send', 'class="email" id="myform"
↪');
```

上面的示例将创建一个类似以下的表单:

```
<form action="http://example.com/index.php/email/send"␣
↪class="email" id="myform" method="post" accept-
↪charset="utf-8">
```

如果*CSRF* 过滤器已打开，`form_open()` 将在表单开头生成 CSRF 字段。你可以通过传递 **csrf_id** 作为 `$attribute` 数组的一个元素来指定此字段的 ID:

```
<?php

echo form_open('/u/sign-up', ['csrf_id' => 'my-id']);
```

将返回:

```
<form action="http://example.com/index.php/u/sign-up"␣
↪method="post" accept-charset="utf-8">
<input type="hidden" id="my-id" name="csrf_test_name"␣
↪value="964ede6e0ae8a680f7b8eab69136717d">
```

备注: 要使用 CSRF 字段的自动生成，你需要打开 **app/Config/Filters.php** 文件中的*CSRF 过滤器*。在大多数情况下，表单页面是使用 GET 方法请求的。通常，POST/PUT/DELETE/PATCH 请求需要 CSRF 保护，但即使是 GET 请求，对于显示表单的页面也必须启用 CSRF 过滤器。

如果你使用*$globals* 启用 CSRF 过滤器，它将对所有请求类型生效。但如果你使用 `public array $methods = ['POST' => ['csrf']];` 启用 CSRF 过滤器，那么在 GET 请求中不会添加隐藏的 CSRF 字段。

添加隐藏输入字段

可以通过将关联数组作为第三个参数传递来添加隐藏字段, 如下所示:

```
<?php

$hidden = ['username' => 'Joe', 'member_id' => '234'];
echo form_open('email/send', '', $hidden);
```

你可以通过向它传递任何 false 值来跳过第二个参数。

上面的示例将创建一个类似以下的表单:

```
<form action="http://example.com/index.php/email/send"␣
↪method="post" accept-charset="utf-8">
    <input type="hidden" name="username" value="Joe">
    <input type="hidden" name="member_id" value="234">
```

form_open_multipart([*$action* = ''[, *$attributes* = ''[, *$hidden* = []]]])

参数

- **$action** (string) –表单操作/目标 URI 字符串

- **$attributes** (mixed) –HTML 属性, 作为数组或转义的字符串
- **$hidden** (array) –隐藏字段定义的数组

返回

一个 HTML 多部分表单开启标签

返回类型

string

此函数与上面的 *form_open()* 完全相同, 除了它添加了一个 *multipart* 属性, 如果你想使用表单上传文件, 这是必需的。

form_hidden(*$name*[, *$value* = ''])

参数

- **$name** (string) –字段名称
- **$value** (string) –字段值

返回

一个 HTML 隐藏 input 元素

返回类型

string

让你生成隐藏的输入字段。你可以提交名称/值字符串以创建一个字段:

```
<?php

form_hidden('username', 'johndoe');
// Would produce: <input type="hidden" name="username" value=
↪"johndoe">
```

…或者你可以提交一个关联数组来创建多个字段:

```
<?php

$data = [
    'name'  => 'John Doe',
    'email' => 'john@example.com',
    'url'   => 'http://example.com',
```

(续下页)

(接上页)

```
];

echo form_hidden($data);
/*
 * Would produce:
 * <input type="hidden" name="name" value="John Doe">
 * <input type="hidden" name="email" value="john@example.com">
 * <input type="hidden" name="url" value="http://example.com">
 */
```

你也可以将关联数组传递给值字段:

```
<?php

$data = [
    'name'  => 'John Doe',
    'email' => 'john@example.com',
    'url'   => 'http://example.com',
];

echo form_hidden('my_array', $data);
/*
 * Would produce:
 * <input type="hidden" name="my_array[name]" value="John Doe">
 * <input type="hidden" name="my_array[email]" value=
↪"john@example.com">
 * <input type="hidden" name="my_array[url]" value="http://
↪example.com">
 */
```

如果你想要带有额外属性的隐藏输入字段:

```
<?php

$data = [
    'type'  => 'hidden',
    'name'  => 'email',
    'id'    => 'hiddenemail',
```

(续下页)

(接上页)

```
    'value' => 'john@example.com',
    'class' => 'hiddenemail',
];

echo form_input($data);
/*
 * Would produce:
 * <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
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性
- **$type** (string) –输入字段的类型。即,'text'、'email'、'number'等

返回

一个 HTML 文本 input 元素

返回类型

string

允许你生成标准的文本输入字段。你可以至少在第一个和第二个参数中传递字段名称和值:

```
<?php

echo form_input('username', 'johndoe');
/*
 * Would produce:
 * <input type="text" name="username" value="johndoe">
 */
```

或者你可以传递一个包含你希望表单包含的任何数据的关联数组:

```
<?php

$data = [
    'name'      => 'username',
    'id'        => 'username',
    'value'     => 'johndoe',
    'maxlength' => '100',
    'size'      => '50',
    'style'     => 'width:50%',
];
echo form_input($data);
/*
 * Would produce:
 * <input type="text" name="username" value="johndoe" id=
↪"username" maxlength="100" size="50" style="width:50%">
 */
```

如果你想要布尔属性, 请传递布尔值 (`true`/`false`)。在这种情况下, 布尔值无关紧要:

```
<?php

$data = [
    'name'     => 'username',
    'id'       => 'username',
    'value'    => '',
    'required' => true,
];
echo form_input($data);
/*
 * Would produce:
 * <input type="text" name="username" value="" id="username"␣
↪required>
 */
```

如果你希望你的表单包含一些额外的数据, 如 JavaScript, 你可以将其作为第三个参数中的字符串传递:

```
<?php

$js = 'onClick="some_function ()"';
echo form_input('username', 'johndoe', $js);
/*
 * Would produce:
 * <input type="text" name="username" value="johndoe" onClick=
↪"some_function ()">
 */
```

或者你可以传递它作为数组:

```
<?php

$js = ['onClick' => 'some_function ();'];
echo form_input('username', 'johndoe', $js);
/*
 * Would produce:
 * <input type="text" name="username" value="johndoe" onClick=
↪"some_function ();">
 */
```

为了支持扩展的 HTML5 输入字段范围, 你可以将输入类型作为第四个参数传递:

```
<?php

echo form_input('email', 'joe@example.com', ['placeholder' =>
↪'Email Address...'], 'email');
/*
 * Would produce:
 * <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) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML 密码 input 元素

返回类型

string

此函数在所有方面与上面的 *form_input()* 函数相同, 只是它使用 "password" 输入类型。

form_upload([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

参数

- **$data** (array) –字段属性数据
- **$value** (string) –字段值
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML 文件上传 input 元素

返回类型

string

此函数在所有方面与上面的 *form_input()* 函数相同, 只是它使用 "file" 输入类型, 允许它用于上传文件。

form_textarea([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

参数

- **$data** (array) –字段属性数据
- **$value** (string) –字段值
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML textarea 元素

返回类型

string

此函数在所有方面与上面的 *form_input()* 函数相同, 只是它生成一个 "textarea" 类型。

备注: 与上面的示例中的 *maxlength* 和 *size* 属性不同, 你将指定 *rows* 和 *cols*。

form_dropdown([*$name* = ''[, *$options* = []][, *$selected* = []][, *$extra* = '']]])

参数

- **$name** (string) –字段名称
- **$options** (array) –要列出的选项的关联数组
- **$selected** (array) –要标记为 *selected* 属性的字段列表
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML select(下拉框) 元素

返回类型

string

允许你创建一个标准的下拉字段。第一个参数将包含字段的名称, 第二个参数将包含选项的关联数组, 第三个参数将包含你希望选中的值。你还可以通过第三个参数传递多个项的数组, 辅助函数将为你创建一个多选字段。

示例:

```
<?php

$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');
/*
 * Would produce:
 * <select name="shirts">
 *     <option value="small">Small Shirt</option>
 *     <option value="med">Medium Shirt</option>
```

(续下页)

(接上页)

```
 *     <option value="large" selected="selected">Large Shirt</
↪option>
 *     <option value="xlarge">Extra Large Shirt</option>
 * </select>
 */

echo form_dropdown('shirts', $options, $shirts_on_sale);
/*
 * Would produce:
 * <select name="shirts" multiple="multiple">
 *     <option value="small" selected="selected">Small Shirt</
↪option>
 *     <option value="med">Medium Shirt</option>
 *     <option value="large" selected="selected">Large Shirt</
↪option>
 *     <option value="xlarge">Extra Large Shirt</option>
 * </select>
 */
```

如果你希望打开的 <select> 包含额外的数据, 如 id 属性或 JavaScript, 你可以将其作为第四个参数中的字符串传递:

```
<?php

$js = 'id="shirts" onChange="some_function();"';
echo form_dropdown('shirts', $options, 'large', $js);
```

或者你可以传入它作为数组:

```
<?php

$js = [
    'id'       => 'shirts',
    'onChange' => 'some_function();',
];
echo form_dropdown('shirts', $options, 'large', $js);
```

如果作为 `$options` 传递的数组是多维数组, 那么 `form_dropdown()` 将使用数组键作为标签生产一个 <optgroup>。

form_multiselect([*$name* = ''[, *$options* = *[]*[, *$selected* = *[]*[, *$extra* = '']]]])

参数

- **$name** (string) –字段名称
- **$options** (array) –要列出的选项的关联数组
- **$selected** (array) –要标记为 *selected* 属性的字段列表
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML 带有 multiple 属性的 select 元素

返回类型

string

允许你创建一个标准的多选字段。第一个参数将包含字段的名称, 第二个参数将包含选项的关联数组, 第三个参数将包含你希望选中的值或值。

参数用法与使用上面的 *form_dropdown()* 相同, 当然字段名称需要使用 POST 数组语法, 例如 foo[]。

form_fieldset([*$legend_text* = ''[, *$attributes* = *[]*]])

参数

- **$legend_text** (string) –要放入 <legend> 标签中的文本
- **$attributes** (array) –要在 <fieldset> 标签上设置的属性

返回

一个 HTML fieldset 开启标签

返回类型

string

允许你生成 fieldset/legend 字段。

示例:

```
<?php

echo form_fieldset('Address Information');
echo "<p>fieldset content here</p>\n";
echo form_fieldset_close();
```

(续下页)

(接上页)

```
?>

<!-- Produces: -->
<fieldset>
    <legend>Address Information</legend>
    <p>form content here</p>
</fieldset>
```

与其他函数类似, 如果你希望设置其他属性, 可以在第二个参数中提交关联数组:

```
<?php

$attributes = [
    'id'    => 'address_info',
    'class' => 'address_info',
];

echo form_fieldset('Address Information', $attributes);
echo "<p>fieldset content here</p>\n";
echo form_fieldset_close();

?>

<!-- Produces: -->
<fieldset id="address_info" class="address_info">
    <legend>Address Information</legend>
    <p>form content here</p>
</fieldset>
```

form_fieldset_close([*$extra* = ''])

参数

- **$extra** (string) –在关闭标签后要追加的任何内容, 原样

返回

一个 HTML fieldset 关闭标签

返回类型

string

产生一个关闭的 `</fieldset>` 标签。使用此函数的唯一优点是它允许你向其传递将添加在标签下方的数据。例如

```
<?php

$string = '</div></div>';
echo form_fieldset_close($string);
// Would produce: </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
- **$checked** (bool) –是否将复选框标记为 *checked*
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML 复选框 input 元素

返回类型

string

允许你生成一个复选框字段。简单示例:

```
<?php

echo form_checkbox('newsletter', 'accept', true);
// Would produce:  <input type="checkbox" name="newsletter"␣
↪value="accept" checked="checked">
```

第三个参数包含一个布尔值 true/false 以确定是否应选中该框。

与此辅助函数中的其他表单函数类似, 你也可以在第一个参数中传递属性的关联数组:

```
<?php
```

(续下页)

(接上页)

```
$data = [
    'name'    => 'newsletter',
    'id'      => 'newsletter',
    'value'   => 'accept',
    'checked' => true,
    'style'   => 'margin:10px',
];

echo form_checkbox($data);
// Would produce: <input type="checkbox" name="newsletter" id=
↪"newsletter" value="accept" checked="checked" style=
↪"margin:10px">
```

与其他函数一样, 如果你希望标签包含其他数据, 如 JavaScript, 可以将其作为第四个参数中的字符串传递:

```
<?php

$js = 'onClick="some_function()"';
echo form_checkbox('newsletter', 'accept', true, $js);
```

或者你可以传递它作为数组:

```
<?php

$js = ['onClick' => 'some_function();'];
echo form_checkbox('newsletter', 'accept', true, $js);
```

form_radio([*$data* = ''[, *$value* = ''[, *$checked* = *false*[, *$extra* = '']]]])

参数

- **$data** (array) –字段属性数据
- **$value** (string) –字段值
- **$checked** (bool) –是否将单选按钮标记为 *checked*
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML 单选按钮 input 元素

返回类型

string

此函数在所有方面与上面的 *form_checkbox()* 函数相同, 只是它使用 "radio" 输入类型。

form_label([*$label_text* = ''[, *$id* = ''[, *$attributes* = []]]])

参数

- **$label_text** (string) –要放入 <label> 标签中的文本
- **$id** (string) –我们为其创建标签的表单元素的 ID
- **$attributes** (string) –HTML 属性

返回

一个 HTML label 元素

返回类型

string

生成一个 <label>。简单示例:

```
<?php

echo form_label('What is your Name', 'username');
// Would produce:  <label for="username">What is your Name</
↪label>
```

与其他函数类似, 如果你更喜欢设置其他属性, 可以在第三个参数中提交关联数组。

例子:

```
<?php

$attributes = [
    'class' => 'mycustomclass',
    'style' => 'color: #000;',
];

echo form_label('What is your Name', 'username', $attributes);
// Would produce:  <label for="username" class="mycustomclass"␣
↪style="color: #000;">What is your Name</label>
```

form_submit([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

参数

- **$data** (string) –按钮名称
- **$value** (string) –按钮值
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML 提交 input 元素

返回类型

string

允许你生成一个标准的提交按钮。简单示例:

```
<?php

echo form_submit('mysubmit', 'Submit Post!');
// Would produce:  <input type="submit" name="mysubmit" value=
↪"Submit Post!">
```

与其他函数类似, 如果你更喜欢设置自己的属性, 可以在第一个参数中提交关联数组。第三个参数允许你向表单添加额外数据, 如 JavaScript。

form_reset([*$data* = ''[, *$value* = ''[, *$extra* = '']]])

参数

- **$data** (string) –按钮名称
- **$value** (string) –按钮值
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML reset input 元素

返回类型

string

允许你生成一个标准的重置按钮。用法与 *form_submit()* 相同。

form_button([*$data* = ''[, *$content* = ''[, *$extra* = '']]])

参数

- **$data** (string) –按钮名称
- **$content** (string) –按钮标签
- **$extra** (mixed) –要作为数组或字符串添加到标记的额外属性

返回

一个 HTML 按钮元素

返回类型

string

允许你生成一个标准的按钮元素。你可以至少在第一个和第二个参数中传递按钮名称和内容:

```
<?php

echo form_button('name', 'content');
// Would produce: <button name="name" type="button">Content</
↪button>
```

或者你可以传递一个关联数组, 其中包含你希望表单包含的任何数据:

```
<?php

$data = [
    'name'    => 'button',
    'id'      => 'button',
    'value'   => 'true',
    'type'    => 'reset',
    'content' => 'Reset',
];

echo form_button($data);
// Would produce: <button name="button" id="button" value="true
↪" type="reset">Reset</button>
```

如果你希望表单包含一些额外的数据, 如 JavaScript, 可以将其作为第三个参数中的字符串传递:

```
<?php
```

(续下页)

(接上页)

```
$js = 'onClick="some_function()"';
echo form_button('mybutton', 'Click Me', $js);
```

form_close([*$extra* = ''])

参数

- **$extra** (string) –在关闭标签后要追加的任何内容, 原样

返回

一个 HTML 表单关闭标签

返回类型

string

产生一个关闭的 `</form>` 标签。使用此函数的唯一优点是它允许你向其传递将添加在标签下方的数据。例如:

```
<?php

$string = '</div></div>';
echo form_close($string);
// Would produce:  </form> </div></div>
```

set_value(*$field*[, *$default* = ''[, *$html_escape* = *true*]])

参数

- **$field** (string) –字段名称
- **$default** (string) –默认值
- **$html_escape** (bool) –是否关闭 HTML 转义值

返回

字段值

返回类型

string

允许你设置输入表单或 textarea 元素的值。你必须通过函数的第一个参数提供字段名称。第二个 (可选) 参数允许你为字段设置默认值。第三个 (可选) 参数允许你关闭值的 HTML 转义, 以防你需要将此函数与 `form_input()` 结合使用以避免双重转义。

示例:

```
<input type="text" name="quantity" value="<?= set_value(
↪'quantity', '0') ?>" size="50">
```

上面的表单在首次加载时将显示 “0”。

set_select(*$field*[, *$value* = ''[, *$default* = *false*]])

参数

- **$field** (string) –字段名称
- **$value** (string) –要检查的值
- **$default** (string) –值是否也是默认值

返回

‘selected’ 属性或空字符串

返回类型

string

如果使用 <select> 菜单, 此函数允许你显示所选菜单项。

第一个参数必须包含选择菜单的名称, 第二个参数必须包含每个项目的值, 第三个 (可选) 参数允许你将某个项目设置为默认值 (使用布尔值 true/false)。

示例:

```
<select name="myselect">
    <option value="one" <?= set_select('myselect', 'one', true)␣
↪?>>One</option>
    <option value="two" <?= set_select('myselect', 'two') ?>>Two
↪</option>
    <option value="three" <?= set_select('myselect', 'three') ?>
↪>Three</option>
</select>
```

set_checkbox(*$field*[, *$value* = ''[, *$default* = *false*]])

参数

- **$field** (string) –字段名称
- **$value** (string) –要检查的值

- **$default** (string) –值是否也是默认值

返回

'checked' 属性或空字符串

返回类型

string

允许你以提交的状态显示复选框。

第一个参数必须包含复选框的名称, 第二个参数必须包含它的值, 第三个 (可选) 参数允许你将某个项目设置为默认值 (使用布尔值 true/false)。

示例:

```
<input type="checkbox" name="mycheck[]" value="1" <?= set_
↪checkbox('mycheck', '1') ?>>
<input type="checkbox" name="mycheck[]" value="2" <?= set_
↪checkbox('mycheck', '2') ?>>
```

set_radio(*$field*[, *$value* = ''[, *$default* = *false*]])

参数

- **$field** (string) –字段名称
- **$value** (string) –要检查的值
- **$default** (string) –值是否也是默认值

返回

'checked' 属性或空字符串

返回类型

string

允许你以提交的状态显示单选按钮。这个函数在所有方面与上面的*set_checkbox()* 函数相同。

示例:

```
<input type="radio" name="myradio" value="1" <?= set_radio(
↪'myradio', '1', true) ?>>
<input type="radio" name="myradio" value="2" <?= set_radio(
↪'myradio', '2') ?>>
```

validation_errors()

在 4.3.0 版本加入.

返回

验证错误

返回类型

array

返回验证错误。首先, 此函数检查存储在会话中的验证错误。要在会话中存储错误, 需要与*redirect()* 一起使用 withInput()。

返回的数组与 Validation::getErrors() 相同。详见验证。

备注: 此函数与模型内验证 不兼容。如果你想在模型验证中获取验证错误，请参阅获取验证错误。

示例:

```
<?php $errors = validation_errors(); ?>
```

validation_list_errors(*$template = 'list'*)

在 4.3.0 版本加入.

参数

- **$template** (string) –验证模板名称

返回

验证错误的渲染 HTML

返回类型

string

返回验证错误的渲染 HTML。

参数 $template 是一个验证模板名称。详见自定义错误显示。

此函数在内部使用*validation_errors()*。

备注: 此函数与模型内验证 不兼容。如果你想在模型验证中获取验证错误，请参阅获取验证错误。

示例:

```
<?= validation_list_errors() ?>
```

validation_show_error(*$field*, *$template = 'single'*)

在 4.3.0 版本加入.

参数

- **$field** (string) –字段名称
- **$template** (string) –验证模板名称

返回

格式化的验证错误 HTML

返回类型

string

为指定字段以格式化的 HTML 返回单个错误。

参数 $template 是一个验证模板名称。详见自定义错误显示。

此函数在内部使用*validation_errors()*。

备注: 此函数与模型内验证 不兼容。如果你想在模型验证中获取验证错误，请参阅获取验证错误。

示例:

```
<?= validation_show_error('username') ?>
```

7.2.6 HTML 辅助函数

HTML 辅助函数文件包含了帮助处理 HTML 的函数。

- 配置
- 加载此辅助函数
- 可用函数

配置

从 v4.3.0 开始, html_helper 函数中的空 HTML 标签 (如 <img>) 默认为兼容 HTML5, 如果你需要兼容 XHTML, 必须在 **app/Config/DocTypes.php** 中将 $html5 属性设置为 false。

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('html');
```

可用函数

以下函数可用:

img([*$src* = ''[, *$indexPage* = *false*[, *$attributes* = '']]])

参数

- **$src** (string|array) –图像源 URI, 或属性和值的数组
- **$indexPage** (bool) – 是否应该将 Config\App::$indexPage 添加到源路径中
- **$attributes** (mixed) –其他 HTML 属性

返回

一个 HTML img 标签

返回类型

string

允许你创建 HTML <img> 标签。第一个参数包含图像源。示例:

```
<?php

echo img('images/picture.jpg');
// <img src="http://site.com/images/picture.jpg">
```

有一个可选的第二个参数，一个 true/false 值，用于指定是否应在创建的地址中添加 `Config\App::$indexPage` 到 *src*。假设你在使用一个媒体控制器：

```
<?php

echo img('images/picture.jpg', true);
// <img src="http://site.com/index.php/images/picture.jpg" alt="
↪">
```

此外，可以将一个关联数组作为第一个参数传递，以完全控制所有属性和值。如果没有提供 *alt* 属性，CodeIgniter 将生成一个空字符串的 *alt* 属性。

示例:

```
<?php

$imageProperties = [
    'src'    => 'images/picture.jpg',
    'alt'    => 'Me, demonstrating how to eat 4 slices of pizza␣
↪at one time',
    'class'  => 'post_images',
    'width'  => '200',
    'height' => '200',
    'title'  => 'That was quite a night',
    'rel'    => 'lightbox',
];

img($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">
```

img_data(*$path*[, *$mime = null*])

参数

- **$path** (string) –图像文件路径
- **$mime** (string|null) –要使用的 MIME 类型, 如果为 null 将猜测

返回

base64 编码的二进制图像字符串

返回类型

`string`

使用 "数据:" 协议从图像生成 src 就绪字符串。示例:

```
<?php

$src = img_data('public/images/picture.jpg'); // data:image/jpg;
↪base64,R0lGODl...
echo img($src);
```

有一个可选的第二个参数来指定 MIME 类型, 否则该函数将使用你的 MIME 配置进行猜测:

```
<?php

$src = img_data('path/img_without_extension', 'image/png'); //␣
↪...
```

注意 `$path` 必须存在并且是一个 数据: 协议支持的可读图像格式。对于非常大的文件不推荐使用此函数, 但它提供了一种方便的方法来从你的应用程序中获取图像, 这些图像并非 Web 可访问的 (例如在 **public/** 中)。

link_tag([*$href* = ''[, *$rel* = 'stylesheet'[, *$type* = 'text/css'[, *$title* = ''[, *$media* = ''[, *$indexPage* = *false*[, *$hreflang* = '']]]]]]])

参数

- **$href** (`string`) –链接文件源
- **$rel** (`string`) –关系类型
- **$type** (`string`) –相关文档的类型
- **$title** (`string`) –链接标题
- **$media** (`string`) –媒体类型
- **$indexPage** (`bool`) –是否应该将 indexPage 添加到链接路径中
- **$hreflang** (`string`) –Hreflang 类型

返回

一个 HTML link 标签

返回类型

`string`

允许你创建 HTML `<link>` 标签。这对于样式表链接很有用，也用于其他链接。参数是 *href*，可选的 *rel*、*type*、*title*、*media*、*indexPage* 和 *hreflang*。

indexPage 是一个布尔值，指定 *href* 是否应该添加由 `$config['indexPage']` 指定的页面地址。

示例:

```
<?php

echo link_tag('css/mystyles.css');
// <link href="http://site.com/css/mystyles.css" rel="stylesheet
↪" type="text/css">
```

更多示例:

```
<?php

echo link_tag('favicon.ico', 'shortcut icon', 'image/ico');
// <link href="http://site.com/favicon.ico" rel="shortcut icon"␣
↪type="image/ico">

echo link_tag('feed', 'alternate', 'application/rss+xml', 'My␣
↪RSS Feed');
// <link href="http://site.com/feed" rel="alternate" type=
↪"application/rss+xml" title="My RSS Feed">
```

或者, 可以将关联数组传递给 `link_tag()` 函数, 以完全控制所有属性和值:

```
<?php

$link = [
    'href'  => 'css/printer.css',
    'rel'   => 'stylesheet',
    'type'  => 'text/css',
```

(续下页)

(接上页)

```
    '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

- **$src** (array|string) –JavaScript 文件的源名称或 URL, 或指定属性的关联数组
- **$indexPage** (bool) –是否将 $src 视为路由的 URI 字符串

返回

一个 HTML script 标签

返回类型

string

允许你创建 HTML <script> 标签。参数是 *src* 和可选的 *indexPage*。

indexPage 是一个布尔值, 指定 *src* 是否应该添加由 $config['indexPage'] 指定的页面地址。

示例:

```
<?php

echo script_tag('js/mystyles.js');
// <script src="http://site.com/js/mystyles.js"></script>
```

或者, 可以将关联数组传递给 script_tag() 函数, 以完全控制所有属性和值:

```
<?php

$script = ['src' => 'js/printer.js', 'defer' => null];

echo script_tag($script);
// <script src="http://site.com/js/printer.js" defer></script>
```

ul (*$list*[, *$attributes* = ''])

参数

- **$list** (array) –列表项
- **$attributes** (array) –HTML 属性

返回

一个 HTML 无序列表标签

返回类型

string

允许你从简单或多维数组生成无序 HTML 列表。示例:

```
<?php

$list = [
    'red',
    'blue',
    'green',
    'yellow',
];

$attributes = [
    'class' => 'boldlist',
    'id'    => 'mylist',
];

echo ul($list, $attributes);
```

以上代码将生成:

```
<ul class="boldlist" id="mylist">
    <li>red</li>
    <li>blue</li>
    <li>green</li>
    <li>yellow</li>
</ul>
```

这是一个更复杂的示例, 使用多维数组:

```php
<?php

$attributes = [
    'class' => 'boldlist',
    'id'    => 'mylist',
];

$list = [
    'colors' => [
        'red',
        'blue',
        'green',
    ],
    'shapes' => [
        'round',
        'square',
        'circles' => [
            'ellipse',
            'oval',
            'sphere',
        ],
    ],
    'moods' => [
        'happy',
        'upset' => [
            'defeated' => [
                'dejected',
                'disheartened',
                'depressed',
            ],
            'annoyed',
            'cross',
            'angry',
        ],
    ],
];

echo ul($list, $attributes);
```

以上代码将生成:

```
<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>square</li>
            <li>circles
                <ul>
                    <li>ellipse</li>
                    <li>oval</li>
                    <li>sphere</li>
                </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>
```

(续下页)

(接上页)

```
            </li>
        </ul>
    </li>
</ul>
```

ol (*$list*, *$attributes* = ”)

参数

- **$list** (array) –列表项
- **$attributes** (array) –HTML 属性

返回

一个 HTML 有序列表标签

返回类型

string

与*ul()* 相同, 只是它生成 <ol> 标签用于有序列表, 而不是 <ul>。

video (*$src*[, *$unsupportedMessage* = ”[, *$attributes* = ”[, *$tracks* = *[]*[, *$indexPage* = *false*]]]])

参数

- **$src** (mixed) –源字符串或源数组。参见*source()* 函数
- **$unsupportedMessage** (string) –如果浏览器不支持 video 标签应显示的消息
- **$attributes** (string) –HTML 属性
- **$tracks** (array) –在数组内使用 track 函数。参见*track()* 函数
- **$indexPage** (bool) –是否应该将 indexPage 添加到视频源路径中

返回

一个 HTML video 标签

返回类型

string

允许你从一个源字符串或一个源数组生成一个 HTML video 标签。示例:

```
<?php

$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,
);
```

以上代码将生成:

```
<video src="test.mp4" controls>
  你的浏览器不支持视频标签。
</video>

<video src="http://www.codeigniter.com/test.mp4" controls>
```

(续下页)

(接上页)

```
  <track src="subtitles_no.vtt" kind="subtitles" srclang="no"␣
↪label="Norwegian No" />
  <track src="subtitles_yes.vtt" kind="subtitles" srclang="yes"␣
↪label="Norwegian Yes" />
  你的浏览器不支持视频标签。
</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" />
  你的浏览器不支持视频标签。
</video>
```

audio(*$src*[, *$unsupportedMessage* = ''[, *$attributes* = ''[, *$tracks* = *[]*[, *$indexPage* = *false*]]]])

参数

- **$src** (mixed) –源字符串或源数组。参见*source()* 函数
- **$unsupportedMessage** (string) –如果浏览器不支持 audio 标签应显示的消息
- **$attributes** (string) –
- **$tracks** (array) –在数组内使用 track 函数。参见*track()* 函数
- **$indexPage** (bool) –是否应该将 indexPage 添加到音频源路径中

返回

一个 HTML audio 标签

返回类型

string

与 *video()* 相同, 只是它生成 <audio> 标签而不是 <video>。

source(*$src*, *$type = 'unknown'*, *$attributes = ''*, *$indexPage = false*)

参数

- **$src** (string) –媒体资源路径
- **$type** (bool) –资源的 MIME 类型, 可选编解码器参数
- **$attributes** (array) –HTML 属性

返回

一个 HTML source 标签

返回类型

string

允许你创建 HTML <source> 标签。第一个参数包含资源的路径。示例:

```
<?php

echo source('movie.mp4', 'video/mp4', 'class="test"');
// <source src="movie.mp4" type="video/mp4" class="test">
```

embed(*$src = ''*[, *$type = false*[, *$attributes = ''*[, *$indexPage = false*]]])

参数

- **$src** (string) –要嵌入的资源路径
- **$type** (bool) –MIME 类型
- **$attributes** (array) –HTML 属性
- **$indexPage** (bool) –是否应该将 indexPage 添加到源路径中

返回

一个 HTML embed 标签

返回类型

string

允许你创建 HTML <embed> 标签。第一个参数包含 embed 源。示例:

```
<?php

echo embed('movie.mov', 'video/quicktime', 'class="test"');
// <embed src="movie.mov" type="video/quicktime" class="test">
```

object(*$data*[, *$type = 'unknown'*[, *$attributes = ''*[, *$params = []*[, *$indexPage = false*]]]])

参数

- **$data** (string) –资源 URL
- **$type** (bool) –资源的内容类型
- **$attributes** (array) –HTML 属性
- **$indexPage** (bool) –是否应该将 indexPage 添加到资源 URL 中
- **$params** (array) –在数组中使用 param 函数。参见*param()* 函数

返回

一个 HTML object 标签

返回类型

string

允许你创建 HTML <object> 标签。第一个参数包含对象数据。示例:

```
<?php

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"'),
```

(续下页)

(接上页)

```
    ],
);
```

以上代码将生成:

```
<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 (*$name*, *$value*[, *$type = 'ref'*[, *$attributes = ''*]])

参数

- **$name** (string) –参数名称
- **$value** (string) –参数值
- **$attributes** (array) –HTML 属性

返回

一个 HTML param 标签

返回类型

string

允许你创建 HTML <param> 标签。第一个参数包含 param 源。示例:

```
<?php

echo param('movie.mov', 'video/quicktime', 'class="test"');
// <param src="movie.mov" type="video/quicktime" class="test">
```

track (*$src*, *$kind*, *$srcLanguage*, *$label*)

参数

- **$src** (string) –track 文件 (.vtt 文件) 的路径
- **$kind** (string) –定时 track 的类型
- **$srcLanguage** (string) –定时 track 的语言

- **$label** (string) –定时 track 的用户可读标题

返回

一个 HTML track 标签

返回类型

string

生成用于指定定时轨道的 track 标签。轨道以 WebVTT 格式格式化。示例:

```
<?php

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) –文档类型名称

返回

一个 HTML DocType 标签

返回类型

string

帮助你生成文档类型声明（DTD）。默认使用 HTML 5，但也有许多其他可用的文档类型。

示例:

```
<?php

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">
```

以下是预定义的文档类型列表。这些文档类型从 **app/Config/DocTypes.php** 中提

取，或者可以在你的 **.env** 配置中重写。

文档类型	$type 参数	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" >

7.2.7 Inflector 辅助函数

Inflector 辅助函数文件包含了允许你将 **英语**单词更改为复数、单数、驼峰式等的函数。

- 加载此辅助函数
- 可用函数

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('inflector');
```

可用函数

以下函数可用:

singular(*$string*)

参数

- **$string**(string) –输入字符串

返回

单数词

返回类型

string

将复数词变为单数。示例:

```
<?php

echo singular('dogs'); // Prints 'dog'
```

plural(*$string*)

参数

- **$string** (string) –输入字符串

返回

复数词

返回类型

string

将单数词变为复数。示例:

```
<?php

echo plural('dog'); // Prints 'dogs'
```

counted(*$count*, *$string*)

参数

- **$count** (int) –项目数量
- **$string** (string) –输入字符串

返回

单数或复数短语

返回类型

string

将词及其计数更改为短语。示例:

```
<?php

echo counted(3, 'dog'); // Prints '3 dogs'
```

camelize(*$string*)

参数

- **$string** (string) –输入字符串

返回

驼峰字符串

返回类型

string

将由空格或下划线分隔的词字符串更改为驼峰式。示例:

```
<?php

echo camelize('my_dog_spot'); // Prints 'myDogSpot'
```

pascalize(*$string*)

参数

- **$string**(string) –输入字符串

返回

帕斯卡式字符串

返回类型

string

将由空格或下划线分隔的词字符串更改为帕斯卡式, 即首字母大写的驼峰式。示例:

```
<?php

echo pascalize('my_dog_spot'); // Prints 'MyDogSpot'
```

underscore(*$string*)

参数

- **$string**(string) –输入字符串

返回

包含下划线而不是空格的字符串

返回类型

string

获取多个由空格分隔的词并在其下添加下划线。示例:

```
<?php

echo underscore('my dog spot'); // Prints 'my_dog_spot'
```

decamelize(*$string*)

参数

- **$string** (string) –输入字符串

返回

在词中间包含下划线的字符串

返回类型

string

获取多个驼峰或帕斯卡单词并将它们转换为下划线分隔的单词。示例:

```
<?php

echo decamelize('myDogSpot'); // Prints 'my_dog_spot'
```

humanize(*$string*[, *$separator* = '_'])

参数

- **$string** (string) –输入字符串
- **$separator** (string) –输入分隔符

返回

人性化字符串

返回类型

string

获取多个由下划线分隔的词并在它们之间添加空格。每个单词的首字母大写。

示例:

```
<?php

echo humanize('my_dog_spot'); // Prints 'My Dog Spot'
```

要使用破折号代替下划线:

```
<?php

echo humanize('my-dog-spot', '-'); // Prints 'My Dog Spot'
```

is_pluralizable(*$word*)

参数

- **$word** (string) –输入字符串

返回

如果单词可数则为 true, 如果不可数则为 false

返回类型

bool

检查给定单词是否有复数形式。示例:

```
<?php

is_pluralizable('equipment'); // Returns false
```

dasherize(*$string*)

参数

- **$string** (string) –输入字符串

返回

短划线字符串

返回类型

string

用破折号替换字符串中的下划线。示例:

```
<?php

dasherize('hello_world'); // Returns 'hello-world'
```

ordinal(*$integer*)

参数

- **$integer** (int) –确定后缀的整数

返回

序数后缀

返回类型

string

返回应添加到数字以表示位置的后缀, 例如 1st、2nd、3rd、4th。示例:

```
<?php

ordinal(1); // Returns 'st'
```

ordinalize(*$integer*)

参数

- **$integer**(int) –要转为序数的整数

返回

序数整数

返回类型

string

将数字转换为用于表示位置的序数字符串, 如 1st、2nd、3rd、4th。示例:

```
<?php

ordinalize(1); // Returns '1st'
```

7.2.8 数字辅助函数

数字辅助函数文件包含了帮助你以与区域设置相关的方式处理数字数据的函数。

- 加载此辅助函数
- 当事情出错时
- 可用函数

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('number');
```

当事情出错时

如果 PHP 的国际化和本地化逻辑无法使用给定的区域设置和选项处理所提供的值, 则会抛出 `BadFunctionCallException()`。

可用函数

以下函数可用:

number_to_size(*$num*[, *$precision* = *1*[, *$locale* = *null*]])

参数

- **$num** (mixed) –字节数
- **$precision** (int) –浮点精度

返回

格式化的数据大小字符串, 如果提供的值不是数字则返回 false

返回类型

string

根据大小格式化数字, 并添加适当的后缀。示例:

```
<?php

echo number_to_size(456); // Returns 456 Bytes
echo number_to_size(4567); // Returns 4.5 KB
echo number_to_size(45678); // Returns 44.6 KB
echo number_to_size(456789); // Returns 447.8 KB
echo number_to_size(3456789); // Returns 3.3 MB
echo number_to_size(12345678912345); // Returns 1.8 GB
echo number_to_size(123456789123456789); // Returns 11,228.3 TB
```

可选的第二个参数允许你设置结果的精度:

```
<?php

echo number_to_size(45678, 2); // Returns 44.61 KB
```

可选的第三个参数允许你指定在生成数字时应使用的区域设置, 这可能会影响格式设置。如果未指定区域设置, 则将分析请求并从标头或应用程序默认值中获取适

当的区域设置:

```
<?php

// Generates 11.2 TB
echo number_to_size(12345678912345, 1, 'en_US');
// Generates 11,2 TB
echo number_to_size(12345678912345, 1, 'fr_FR');
```

备注: 此函数生成的文本位于以下语言文件中:*language/<your_lang>/Number.php*

number_to_amount(*$num[, $precision = 1[, $locale = null]*)

参数

- **$num** (mixed) –要格式化的数字
- **$precision** (int) –浮点精度
- **$locale** (string) –用于格式化的区域设置

返回

字符串的人类可读版本, 如果提供的值不是数字则返回 false

返回类型

string

将数字转换为人类可读版本, 如 **123.4 万亿**用于高达四次方的数字。示例:

```
<?php

echo number_to_amount(123456); // Returns 123 thousand
echo number_to_amount(123456789); // Returns 123 million
echo number_to_amount(1234567890123, 2); // Returns 1.23␣
↪trillion
echo number_to_amount('123,456,789,012', 2); // Returns 123.46␣
↪billion
```

可选的第二个参数允许你设置结果的精度:

```
<?php

echo number_to_amount(45678, 2); // Returns 45.68 thousand
```

可选的第三个参数允许指定区域设置:

```
<?php

echo number_to_amount('123,456,789,012', 2, 'de_DE'); //
↪Returns 123,46 billion
```

number_to_currency(*$num*, *$currency*[, *$locale = null*[, *$fraction = 0*]])

参数

- **$num** (float) –要格式化的数字
- **$currency** (string) –货币类型, 即 USD、EUR 等
- **$locale** (string|null) –用于格式化的区域设置
- **$fraction** (integer) –小数点后小数位数

返回

适用于该区域设置的货币格式的数字

返回类型

string

将数字转换为常见的货币格式, 如 USD、EUR、GBP 等:

```
<?php

echo number_to_currency(1234.56, 'USD', 'en_US', 2);  //
↪Returns $1,234.56
echo number_to_currency(1234.56, 'EUR', 'de_DE', 2);  //
↪Returns 1.234,56 €
echo number_to_currency(1234.56, 'GBP', 'en_GB', 2);  //
↪Returns £1,234.56
echo number_to_currency(1234.56, 'YEN', 'ja_JP', 2);  //
↪Returns YEN 1,234.56
```

如果你不指定区域设置, 将使用请求区域设置。

number_to_roman(*$num*)

参数

- **$num** (int|string) –要转换的数字

返回

从给定参数转换后的罗马数字

返回类型

string|null

将数字转换为罗马数字:

```
<?php

echo number_to_roman(23);     // Returns XXIII
echo number_to_roman(324);    // Returns CCCXXIV
echo number_to_roman(2534);   // Returns MMDXXXIV
```

此函数仅处理 1 到 3999 范围内的数字。对于该范围之外的任何值, 它都将返回 null。

7.2.9 安全辅助函数

安全辅助函数文件包含安全相关函数。

- 加载此辅助函数
- 可用函数

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('security');
```

可用函数

以下函数可用:

sanitize_filename(*$filename*)

参数

- **$filename** (string) –文件名

返回

安全的文件名

返回类型

string

提供对目录遍历的保护。

此函数是 \CodeIgniter\Security::sanitizeFilename() 的别名。更多信息, 请参阅安全库 文档。

strip_image_tags(*$str*)

参数

- **$str** (string) –输入字符串

返回

不包含图像标签的输入字符串

返回类型

string

这是一个安全函数, 用于从字符串中剥离图像标签。它将图像 URL 作为纯文本保留。

例子:

```
<?php

$string = strip_image_tags($string);
```

encode_php_tags(*$str*)

参数

- **$str** (string) –输入字符串

返回

安全格式化的字符串

返回类型

`string`

这是一个安全函数, 用于将 PHP 标签转换为实体。

例子:

```
<?php

$string = encode_php_tags($string);
```

7.2.10 测试辅助函数

测试辅助函数文件包含帮助测试项目的函数。

- 加载此辅助函数
- 可用函数

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('test');
```

可用函数

以下函数可用:

fake(*$model*, *array $overrides = null*)

参数

- **$model** (`Model|object|string`) –要与 Fabricator 一起使用的模型的实例或名称

- **$overrides** (array|null) – 要传递给 Fabricator::setOverrides() 的覆盖数据

返回

Fabricator 创建的随机假数据并添加到数据库中的项目

返回类型

object|array

使用 CodeIgniter\Test\Fabricator 创建一个随机项并将其添加到数据库中。

使用示例:

```
<?php

use CodeIgniter\Test\CIUnitTestCase;

final class MyTestClass extends CIUnitTestCase
{
    public function testUserAccess()
    {
        $user = fake('App\Models\UserModel');

        $this->assertTrue($this->userHasAccess($user));
    }
}
```

7.2.11 文本辅助函数

文本辅助函数文件包含用于处理文本的辅助函数。

- 加载辅助函数
- 可用函数

加载辅助函数

使用以下代码加载本辅助函数:

```
<?php

helper('text');
```

可用函数

以下函数可用:

random_string([$type = 'alnum'[, $len = 8]])

参数

- **$type** (string) –随机化类型
- **$len** (int) –输出字符串长度

返回

随机字符串

返回类型

string

根据你指定的类型和长度生成随机字符串。适用于创建密码或生成随机哈希值。

> **警告:** 对于 **basic**、**md5** 和 **sha1** 类型，生成的字符串不具备加密安全性。因此这些类型不能用于加密用途或需要不可预测返回值的场景。自 v4.3.3 起这些类型已弃用。

第一个参数指定字符串类型，第二个参数指定长度。可用选项包括:

- **alpha**: 仅包含大小写字母的字符串
- **alnum**: 包含大小写字母和数字的字符串

- **basic**: [已弃用] 基于 `mt_rand()` 的随机数（长度参数无效）
- **numeric**: 纯数字字符串
- **nozero**: 不含零的数字字符串
- **md5**: [已弃用] 基于 `md5()` 加密的随机数（固定长度 32）
- **sha1**: [已弃用] 基于 `sha1()` 加密的随机数（固定长度 40）
- **crypto**: 基于 `random_bytes()` 的加密安全随机字符串

备注: 使用 **crypto** 类型时，第二个参数必须设置为偶数。自 v4.2.2 起，如果传入奇数将抛出 `InvalidArgumentException`。

备注: 自 v4.3.3 起，**alpha**、**alnum** 和 **nozero** 改用 `random_byte()`，**numeric** 改用 `random_int()`。旧版本使用不具备加密安全性的 `str_shuffle()`。

使用示例:

```
<?php

echo random_string('alnum', 16);
```

increment_string(*$str*[, *$separator* = '_'[, *$first* = *1*]])

参数

- **$str** (string) –输入字符串
- **$separator** (string) –重复数字的分隔符
- **$first** (int) –起始数字

返回

递增后的字符串

返回类型

string

通过在字符串末尾追加数字或递增现有数字来实现字符串递增。适用于创建文件”副本”或复制具有唯一标题/slug 的数据库内容。

使用示例：

```
<?php

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) –可变数量参数

返回

交替输出的字符串

返回类型

mixed

允许在循环中交替输出两个或多个项目。示例：

```
<?php

for ($i = 0; $i < 10; $i++) {
    echo alternator('string one', 'string two');
}
```

可以添加任意数量的参数，每次循环迭代会返回下一个项目。

```
<?php

for ($i = 0; $i < 10; $i++) {
    echo alternator('one', 'two', 'three', 'four', 'five');
}
```

备注： 要初始化多次独立调用，只需不带参数调用本函数即可重置状态。

reduce_double_slashes(*$str*)

参数

- **$str** (string) –输入字符串

返回

标准化斜杠后的字符串

返回类型

string

将字符串中的双斜杠转换为单斜杠，URL 协议前缀中的双斜杠除外（例如 http://）。

示例：

```
<?php

$string = 'http://example.com//index.php';
echo reduce_double_slashes($string); // results in "http://
↪example.com/index.php"
```

strip_slashes(*$data*)

参数

- **$data** (mixed) –输入字符串或字符串数组

返回

去除斜杠后的字符串或数组

返回类型

mixed

从字符串数组中移除所有斜杠。

示例：

```
<?php

$str = [
    'question' => "Is your name O\\'reilly?",
    'answer'   => "No, my name is O\\'connor.",
];

$str = strip_slashes($str);
```

上述代码将返回：

```
<?php

[
    'question' => "Is your name O'reilly?",
    'answer'   => "No, my name is O'connor.",
];
```

备注: 由于历史原因，本函数也接受并处理字符串输入，这使得它成为 `stripslashes()` 的别名。

reduce_multiples(*$str*[, *$character* = ', '[, *$trim* = *false*]])

参数

- **$str** (string) –输入文本
- **$character** (string) –需要缩减的字符
- **$trim** (bool) –是否同时去除首尾字符

返回

缩减后的字符串

返回类型

string

缩减连续出现的重复字符。示例:

```
<?php

$string = 'Fred, Bill,, Joe, Jimmy';
$string = reduce_multiples($string); // results in "Fred, Bill,␣
↪Joe, Jimmy"
```

如果第三个参数设为 `true`，将同时去除字符串首尾的指定字符。示例:

```
<?php

$string = ',Fred, Bill,, Joe, Jimmy,';
$string = reduce_multiples($string, ',', true); // results in
↪"Fred, Bill, Joe, Jimmy"
```

quotes_to_entities(*$str*)

参数

- **$str** (string) –输入字符串

返回

引号转换为 HTML 实体后的字符串

返回类型

string

将字符串中的单双引号转换为对应的 HTML 实体。示例:

```
<?php

$string = "Joe's \"dinner\"";
$string = quotes_to_entities($string); // results in "Joe&#39;s␣
↪&quot;dinner&quot;"
```

strip_quotes(*$str*)

参数

- **$str** (string) –输入字符串

返回

去除引号后的字符串

返回类型

string

移除字符串中的单双引号。示例:

```
<?php

$string = "Joe's \"dinner\"";
$string = strip_quotes($string); // results in "Joes dinner"
```

word_limiter(*$str*[, *$limit = 100*[, *$endChar = '…'*]])

参数

- **$str** (string) –输入字符串
- **$limit** (int) –单词数量限制

- **$endChar** (string) –结尾字符（通常为省略号）

返回

单词截断后的字符串

返回类型

string

将字符串截断为指定数量的单词。示例：

```
<?php

$string = 'Here is a nice text string consisting of eleven␣
↪words.';
$string = word_limiter($string, 4);
// Returns:  Here is a nice
```

第三个参数是可选的结尾后缀，默认添加省略号。

character_limiter (*$string*[, *$limit = 500*[, *$endChar = '…'*]])

参数

- **$string** (string) –输入字符串
- **$limit** (int) –字符数量
- **$endChar** (string) –结尾字符（通常为省略号）

返回

字符截断后的字符串

返回类型

string

将字符串截断为指定数量的 字符，同时保持单词完整性，实际字符数可能略多于或少于指定值。

示例：

```
<?php

$string = 'Here is a nice text string consisting of eleven␣
↪words.';
```

(续下页)

(接上页)

```
$string = character_limiter($string, 20);
// Returns:  Here is a nice text string
```

第三个参数是可选的结尾后缀，未声明时使用省略号。

备注: 如需精确截断到指定字符数，请参考下方`ellipsize()` 函数。

---

ascii_to_entities(*$str*)

参数

- **$str** (string) –输入字符串

返回

ASCII 转换为实体后的字符串

返回类型

string

将 ASCII 值转换为字符实体，包括可能引发网页显示问题的高位 ASCII 和 MS Word 字符，确保在不同浏览器设置下显示一致或可靠存储于数据库。由于依赖服务器支持的字符集，在极少数情况下可能不完全可靠，但能正确识别常规范围外的字符（如带重音符号的字符）。

示例:

```
<?php

$string = ascii_to_entities($string);
```

entities_to_ascii(*$str*[, *$all = true*])

参数

- **$str** (string) –输入字符串
- **$all** (bool) –是否转换不安全实体

返回

实体转换回 ASCII 后的字符串

返回类型

string

本函数功能与*ascii_to_entities()* 相反，将字符实体转换回 ASCII。

convert_accented_characters(*$str*)

参数

- **$str** (string) –输入字符串

返回

转换重音字符后的字符串

返回类型

string

将高位 ASCII 字符转写为对应的低位 ASCII 字符。适用于需要在仅支持标准 ASCII 的场合（如 URL）使用非英文字符的场景。

示例：

```
<?php

$string = convert_accented_characters($string);
```

备注： 本函数使用配置文件 **app/Config/ForeignCharacters.php** 来定义转写对照表。

---

word_censor(*$str*, *$censored*[, *$replacement* = ''])

参数

- **$str** (string) –输入字符串
- **$censored** (array) –需屏蔽的敏感词列表
- **$replacement** (string) –替换内容

返回

敏感词过滤后的字符串

返回类型

string

对文本中的敏感词进行过滤替换。第一个参数是原始字符串，第二个是敏感词数组，第三个（可选）是替换内容（默认使用井号 #### 替换）。

示例:

```
<?php

$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

代码高亮后的 HTML 字符串

返回类型

string

对代码字符串（PHP、HTML 等）进行语法高亮。示例:

```
<?php

$string = highlight_code($string);
```

本函数使用 PHP 的 highlight_string()，颜色方案取决于 php.ini 中的设置。

highlight_phrase(*$str*, *$phrase*[, *$tag_open* = '<mark>'[, *$tag_close* = '</mark>']])

参数

- **$str** (string) –输入字符串
- **$phrase** (string) –需高亮的短语
- **$tag_open** (string) –高亮起始标签
- **$tag_close** (string) –高亮结束标签

返回

短语高亮后的 HTML 字符串

返回类型

string

在文本中高亮指定短语。第一个参数是原始字符串，第二个是要高亮的短语，第三、四个参数是包裹短语的 HTML 标签。

示例:

```
<?php

$string = 'Here is a nice text string about nothing in␣
↪particular.';
echo highlight_phrase($string, 'nice text', '<span style="color:
↪#990000;">', '</span>');
```

上述代码输出:

```
Here is a <span style="color:#990000;">nice text</span> string␣
↪about nothing in particular.
```

备注: 本函数曾默认使用 `<strong>` 标签。如需支持旧版浏览器，建议在样式表中添加以下 CSS:

```
mark {
    background: #ff0;
    color: #000;
};
```

---

word_wrap(*$str*[, *$charlim = 76*])

参数

- **$str** (string) –输入字符串
- **$charlim** (int) –字符限制数

返回

自动换行后的字符串

返回类型

string

在指定字符数处进行换行，同时保持单词完整。

示例:

```
<?php

$string = 'Here is a simple string of text that will help us␣
↪demonstrate this function.';
echo word_wrap($string, 25);
/*
 * Would produce:
 * Here is a simple string
 * of text that will help us
 * demonstrate this
 * function.
 *
 * Excessively long words will be split, but URLs will not be.
 */
```

ellipsize(*$str*, *$max_length*[, *$position* = *1*[, *$ellipsis* = *'…'*]])

参数

- **$str** (string) –输入字符串
- **$max_length** (int) –最大长度限制
- **$position** (mixed) –分割位置（整数或浮点数）
- **$ellipsis** (string) –省略符号

返回

添加省略号后的字符串

返回类型

string

本函数会去除标签，按最大长度分割字符串，并插入省略号。

第一个参数是待处理字符串，第二个是最终字符串长度，第三个是省略号位置（0-1 表示从左到右，例如 1 在右侧，0.5 在中部，0 在左侧），第四个可选参数指定省略符号（默认使用 …）。

示例：

```
<?php
```

(续下页)

(接上页)

```
$str = 'this_string_is_entirely_too_long_and_might_break_my_
↪design.jpg';
echo ellipsize($str, 32, .5);
```

输出:

```
this_string_is_e&hellip;ak_my_design.jpg
```

excerpt (*$text*, *$phrase = false*, *$radius = 100*, *$ellipsis = '...'*)

参数

- **$text** (string) –待提取文本
- **$phrase** (string) –中心短语
- **$radius** (int) –前后提取字符数
- **$ellipsis** (string) –省略符号

返回

提取的摘要

返回类型

string

本函数以指定短语为中心，提取前后各 $radius 个字符的文本，并在两端添加省略号。

第一个参数是源文本，第二个是中心短语，第三个是提取半径，第四个是省略符号。如果未传入短语，则提取前 $radius 个字符并在末尾加省略号。

示例:

```
<?php

$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␣
```

(续下页)

(接上页)

```
↪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
arcu. Cras ac ipsum sit amet augue laoreet laoreet. Aenean a␣
↪risus lacus.
Sed ut tortor diam.';

echo excerpt($text, '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 tortor. ...
```

7.2.12 URL 辅助函数

URL 辅助函数文件包含帮助使用 URL 的函数。

- 加载此辅助函数
- 可用函数

加载此辅助函数

此辅助函数由框架在每个请求上自动加载。

可用函数

以下函数可用:

site_url([*$uri* = ''[, *$protocol* = *null*[, *$altConfig* = *null*]]])

参数

- **$uri** (array|string) –URI 字符串或 URI 段数组
- **$protocol** (string) –协议，例如 'http' 或 'https'。如果设置为空字符串 ''，则返回一个 protocol-relative 链接。
- **$altConfig** (\Config\App) –要使用的备用配置

返回

站点 URL

返回类型

string

备注: 从 v4.3.0 开始, 如果你设置了 Config\App::$allowedHostnames, 并且当前 URL 匹配, 则会返回主机名设置了的 URL。

返回配置文件中指定的你的站点 URL。**index.php** 文件 (或你在配置文件中设置为站点 Config\App::$indexPage 的任何内容) 都将添加到 URL 中, 就像你传递给函数的任何 URI 段一样。

每当你需要生成本地 URL 时, 都建议使用此函数, 以便在 URL 改变的情况下使页面更便携。

段可以可选地作为字符串或数组传递给函数。这是字符串示例:

```
<?php

echo site_url('news/local/123');
```

上面的示例将返回类似内容: **http://example.com/index.php/news/local/123**

以下是作为数组传递段的示例:

```
<?php

$segments = ['news', 'local', '123'];
echo site_url($segments);
```

如果为不同于你自己的站点生成 URL, 其中包含不同的配置首选项, 则备用配置可能很有用。我们对框架本身使用它进行单元测试。

base_url([*$uri* = ''[, *$protocol* = *null*]])

参数

- **$uri** (array|string) –URI 字符串或 URI 段数组
- **$protocol** (string) –协议，例如 'http' 或 'https'。如果设置为空字符串 ''，则返回一个 protocol-relative 链接。

返回

Base URL

返回类型

string

备注: 从 v4.3.0 开始, 如果你设置了 Config\App::$allowedHostnames, 并且当前 URL 匹配, 则会返回主机名设置了的 URL。

备注: 在以前的版本中, 如果不带参数调用, 此函数返回没有尾随斜杠 (/) 的基本 URL。该错误已修复, 从 v4.3.2 开始, 它返回带有尾随斜杠的基本 URL。

返回配置文件中指定的你的站点基础 URL。示例:

```
<?php

echo base_url();
```

此函数返回与不附加 Config\App::$indexPage 的*site_url()* 相同的内容。

与*site_url()* 类似, 你可以将段作为字符串或数组提供。这是一个字符串示例:

```
<?php

// Returns like `http://example.com/blog/post/123`
echo base_url('blog/post/123');
```

上面的示例将返回类似内容: **http://example.com/blog/post/123**

如果你传递一个空字符串 `''` 作为第二个参数，它会返回 protocol-relative 链接:

这很有用, 因为与*site_url()* 不同, 你可以为文件 (如图像或样式表) 提供字符串。例如:

```
<?php

echo base_url('images/icons/edit.png');
```

这将给你类似的内容: **http://example.com/images/icons/edit.png**

current_url([*$returnObject* = *false*[, *$request* = *null*]])

参数

- **$returnObject** (boolean) –如果希望返回 URI 实例而不是字符串, 则为 True。
- **$request** (IncomingRequest|null) –用于路径检测的替代请求; 用于测试。

返回

当前 URL

返回类型

string|\CodeIgniter\HTTP\URI

返回当前正在查看的页面的完整 URL。返回字符串时, 会删除 URL 的查询和片段部分。返回 URI 时, 会保留查询和片段部分。

但是, 出于安全原因, 它基于 `Config\App` 设置创建, 而不是旨在匹配浏览器 URL。

从 v4.3.0 开始, 如果你设置了 `Config\App::$allowedHostnames`, 并且当前 URL 匹配, 则会返回主机名设置了的 URL。

备注: 调用 `current_url()` 与这样做相同:

```
site_url(uri_string());
```

重要: 在 v4.1.2 之前, 此函数有一个错误, 导致它忽略对 `Config\App::$indexPage` 的配置。

---

previous_url([*$returnObject = false*])

参数

- **$returnObject** (boolean) –如果希望返回 URI 实例而不是字符串, 则为 True。

返回

用户之前所在的 URL

返回类型

`string|\CodeIgniter\HTTP\URI`

返回用户之前完整的 URL(包括段)。

备注: 由于盲目信任 `HTTP_REFERER` 系统变量存在安全问题, 如果可用,CodeIgniter 会将以前访问的页面存储在会话中。这确保我们始终使用已知和可信的来源。如果尚未加载会话或否则不可用, 则将使用经过清理的 `HTTP_REFERER` 版本。

---

uri_string()

返回

URI 字符串

返回类型

`string`

返回相对于 baseURL 的当前 URL 的路径部分。

例如, 当你的 baseURL 为 **http://some-site.com/** , 当前 URL 为:

```
http://some-site.com/blog/comments/123
```

函数将返回:

```
blog/comments/123
```

当你的 baseURL 为 **http://some-site.com/subfolder/** , 当前 URL 为:

```
http://some-site.com/subfolder/blog/comments/123
```

函数将返回:

```
blog/comments/123
```

备注: 以前的版本中定义了参数 `$relative = false`。然而, 由于一个错误, 此函数总是返回相对于 baseURL 的路径。从 v4.3.2 开始, 该参数已被删除。

备注: 在以前的版本中, 当你导航到 baseURL 时, 此函数返回 `/`。从 v4.3.2 开始, 错误已修复, 它返回一个空字符串 (`''`)。

---

index_page([*$altConfig = null*])

参数

- **$altConfig** (`\Config\App`) –要使用的备用配置

返回

`indexPage` 值

返回类型

`string`

返回配置文件中指定的你的站点 **indexPage**。例如:

```
<?php

echo index_page();
```

与*site_url()* 一样, 你可以指定备用配置。如果为不同于你自己的站点生成 URL, 其中包含不同的配置首选项, 则备用配置可能很有用。我们对框架本身使用它进行单元测试。

anchor([$uri = ''[, $title = ''[, $attributes = ''[, $altConfig = null]]]])

参数

- **$uri** (array|string) –URI 字符串或 URI 段数组
- **$title** (string) –锚点标题
- **$attributes** (array|object|string) –HTML 属性
- **$altConfig** (\Config\App|null) –要使用的备用配置

返回

HTML 链接 (锚点标签)

返回类型

string

基于你的本地站点 URL 创建标准的 HTML 锚点链接。

第一个参数可以包含你希望附加到 URL 的任何段。与上面的*site_url()* 函数一样, 段可以是字符串或数组。

备注: 如果你正在构建应用程序内部的链接, 请不要包含基本 URL (http://...)。这将从配置文件中指定的信息自动添加。只包含你希望附加到 URL 的 URI 段。

第二段是你希望链接说的文本。如果留空, 将使用 URL。

第三个参数可以包含你希望添加到链接的属性列表。属性可以是简单的字符串或关联数组。

这里有一些示例:

```
<?php

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', ['title' => 'The best␣
↪news!']);
// Prints: <a href="http://example.com/index.php/news/local/123
```

(续下页)

(接上页)

```
↪" title="The best news!">My News</a>

echo anchor('', 'Click here');
// Prints: <a href="http://example.com/index.php">Click here</a>
```

如上所述, 你可以指定备用配置。如果为不同于你自己的站点生成链接, 其中包含不同的配置首选项, 则备用配置可能很有用。我们对框架本身使用它进行单元测试。

备注: 传递给 anchor 函数的属性会自动转义, 以防止 XSS 攻击。

---

anchor_popup([*$uri* = ''[, *$title* = ''[, *$attributes* = *false*[, *$altConfig* = *null*]]]])

参数

- **$uri** (string) –URI 字符串
- **$title** (string) –锚点标题
- **$attributes** (array|false|object|string) –HTML 属性
- **$altConfig** (\Config\App) –要使用的备用配置

返回

弹出式超链接

返回类型

string

几乎与*anchor()* 函数完全相同, 除了它在新窗口中打开 URL。你可以在第三个参数中指定 JavaScript 窗口属性以控制窗口的打开方式。如果未设置第三个参数, 它将简单地用你自己的浏览器设置打开新窗口。

这里是一个带有属性的示例:

```
<?php

$atts = [
    'width'       => 800,
    'height'      => 600,
```

(续下页)

(接上页)

```
    'scrollbars'  => 'yes',
    'status'      => 'yes',
    'resizable'   => 'yes',
    'screenx'     => 0,
    'screeny'     => 0,
    'window_name' => '_blank',
];

echo anchor_popup('news/local/123', 'Click Me!', $atts);
```

如上所述, 你可以指定备用配置。如果为不同于你自己的站点生成链接, 其中包含不同的配置首选项, 则备用配置可能很有用。我们对框架本身使用它进行单元测试。

备注: 上述属性是函数默认值, 所以你只需要设置与你需要的不同的那些。如果你希望函数使用所有默认值, 只需在第三个参数中传递一个空数组:

```
<?php

echo anchor_popup('news/local/123', 'Click Me!', []);
```

备注: **window_name** 实际上不是一个属性, 而是 window.open() 方法接受的一个参数, 它接受窗口名称或窗口目标。

备注: 除上述之外的任何其他属性都将作为 HTML 锚点标记的属性进行解析。

备注: 传递给 anchor_popup 函数的属性会自动转义, 以防止 XSS 攻击。

---

mailto(*$email*[, *$title* = ''[, *$attributes* = '']])

参数

- **$email** (string) –电子邮件地址

- **$title** (string) –锚点标题
- **$attributes** (array|object|string) –HTML 属性

返回

"发送邮件到" 超链接

返回类型

string

创建标准的 HTML 电子邮件链接。使用示例:

```
<?php

echo mailto('me@my-site.com', 'Click Here to Contact Me');
```

如上面的*anchor()* 选项卡一样, 你可以使用第三个参数设置属性:

```
<?php

$attributes = ['title' => 'Mail me'];
echo mailto('me@my-site.com', 'Contact Me', $attributes);
```

备注: 传递给 mailto 函数的属性会自动转义, 以防止 XSS 攻击。

---

safe_mailto(*$email*[, *$title* = ''[, *$attributes* = '']])

参数

- **$email** (string) –电子邮件地址
- **$title** (string) –锚点标题
- **$attributes** (array|object|string) –HTML 属性

返回

防垃圾邮件的 "发送邮件到" 超链接

返回类型

string

与*mailto()* 函数完全相同, 除了它使用序数数字与 JavaScript 编写的隐写版本来帮助防止垃圾邮件机器人收集电子邮件地址。

auto_link(*$str*[, *$type = 'both'*[, *$popup = false*]])

参数

- **$str** (string) –输入字符串
- **$type** (string) –链接类型 ('email'、'url' 或 'both')
- **$popup** (bool) –是否创建弹出链接

返回

链接化的字符串

返回类型

string

自动将字符串中包含的 URL 和电子邮件地址转换为链接。示例:

```
<?php

$string = auto_link($string);
```

第二个参数确定是转换 URL 和电子邮件还是仅转换其中一个。如果未指定参数,默认行为是两者都转换。电子邮件链接编码为上面显示的*safe_mailto()*。

仅转换 URL:

```
<?php

$string = auto_link($string, 'url');
```

仅转换电子邮件地址:

```
<?php

$string = auto_link($string, 'email');
```

第三个参数确定是否在新窗口中显示链接。值可以为 true 或 false(布尔值):

```
<?php

$string = auto_link($string, 'both', true);
```

备注: 仅识别以 `www.` 或 `://` 开头的 URL。

url_title(*$str*[, *$separator* = '-'[, *$lowercase* = *false*]])

参数

- **$str** (string) –输入字符串
- **$separator** (string) –单词分隔符 (通常为 '-' 或 '_')
- **$lowercase** (bool) –是否将输出字符串转换为小写

返回

URL 格式化的字符串

返回类型

string

获取一个字符串作为输入, 并创建一个人性化的 URL 字符串。例如, 如果你有一个博客, 希望在 URL 中使用条目的标题。示例:

```
<?php

$title     = "What's wrong with CSS?";
$url_title = url_title($title);
// Produces: Whats-wrong-with-CSS
```

第二个参数确定单词分隔符。默认使用破折号。首选选项是: - (破折号) 或 _ (下划线)。

示例:

```
<?php

$title     = "What's wrong with CSS?";
$url_title = url_title($title, '_');
// Produces: Whats_wrong_with_CSS
```

第三个参数确定是否强制使用小写字符。默认不强制。选项是布尔值 true/false。

示例:

```
<?php

$title     = "What's wrong with CSS?";
$url_title = url_title($title, '-', true);
// Produces: whats-wrong-with-css
```

mb_url_title(*$str*[, *$separator* = '-'[, *$lowercase* = *false*]])

参数

- **$str** (string) –输入字符串
- **$separator** (string) –单词分隔符 (通常为 '-' 或 '_')
- **$lowercase** (bool) –是否将输出字符串转换为小写

返回

URL 格式化的字符串

返回类型

string

此函数的工作方式与 *url_title()* 相同, 但它会自动转换所有重音字符。

prep_url([*$str* = ''[, *$secure* = *false*]])

参数

- **$str** (string) –URL 字符串
- **$secure** (boolean) –true 为 https://

返回

带协议前缀的 URL 字符串

返回类型

string

如果 URL 中缺少协议前缀, 此函数将添加 http:// 或 https://。

如下传入 URL 字符串给函数:

```
<?php

$url = prep_url('example.com');
```

url_to(*$controller*[, ...*$args*])

参数

- **$controller** (string) –路由名称或 Controller::method
- **...$args** (int|string) –要传递给路由的一个或多个参数。最后一个参数允许你设置区域设置。

返回

绝对 URL

返回类型

string

备注: 此函数要求在 **app/Config/Routes.php** 中为控制器/方法定义路由。

在你的应用程序中构建指向控制器方法的绝对 URL。示例:

```
<?php

// The route is defined as:
$routes->get('/', 'Home::index');

?>

<a href="<?= url_to('Home::index') ?>">Home</a>
<!-- Result: 'http://example.com/' -->
```

你还可以向路由添加参数。这是一个示例:

```
<?php

// The route is defined as:
$routes->get('pages/(:segment)', 'Page::index/$1');

?>

<a href="<?= url_to('Page::index', 'home') ?>">Home</a>
<!-- Result: 'http://example.com/pages/home' -->
```

这很有用, 因为即使在将链接放入视图后, 你仍然可以更改路由。

从 v4.3.0 开始, 当你在路由中使用 `{locale}` 时, 你可以可选地将区域设置值指定为最后一个参数。

```
<?php

// The route is defined as:
$routes->add(
    '{locale}/users/(:num)/gallery/(:num)',
    'Galleries::showUserGallery/$1/$2',
    ['as' => 'user_gallery'],
);

?>

<a href="<?= url_to('user_gallery', 15, 12, 'en') ?>">View␣
↪Gallery</a>
<!-- Result: 'http://example.com/en/users/15/gallery/12' -->
```

有关完整详细信息, 请参阅反向路由 和使用命名路由。

url_is(*$path*)

参数

- **$path** (string) –要比较当前 URI 路径的相对于 baseURL 的 URL 路径。

返回类型

boolean

将当前 URL 的路径与给定路径进行比较, 以查看它们是否匹配。示例:

```
<?php

if (url_is('admin')) {
    // ...
}
```

这将匹配 **http://example.com/admin**。如果你的 baseURL 是 `http://example.com/subdir/`, 它将匹配 **http://example.com/subdir/admin**。

你可以使用 * 通配符来匹配 URL 中的任何其他可应用字符:

```
<?php

if (url_is('admin*')) {
    // ...
}
```

这将匹配以下任何一个:

- /admin
- /admin/
- /admin/users
- /admin/users/schools/classmates/…

7.2.13 XML 辅助函数

XML 辅助函数文件包含帮助处理 XML 数据的函数。

- 加载此辅助函数
- 可用函数

加载此辅助函数

使用以下代码加载此辅助函数:

```
<?php

helper('xml');
```

可用函数

以下函数可用:

xml_convert(*$str*[, *$protect_all = false*])

参数

- **$str** (string) –要转换的文本字符串
- **$protect_all** (bool) –是否保护看起来像潜在实体的所有内容, 而不仅仅是编号的实体, 例如 &foo;

返回

XML 转换后的字符串

返回类型

string

接受一个字符串作为输入, 并将以下保留的 XML 字符转换为实体:

- 和号: &
- 小于号和大于号: < >
- 单引号和双引号: ‘“
- 破折号: -

如果它们是现有编号字符实体的一部分, 此函数会忽略和号, 例如 {。示例:

```
<?php

$string = '<p>Here is a paragraph & an entity (&#123;).</p>';
$string = xml_convert($string);
echo $string;
```

输出:

```
&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 测试

CodeIgniter 从一开始就尽可能地简化了框架和应用程序的测试。对 PHPUnit 的支持是内置的，框架还提供了许多方便的辅助方法，使得对应用程序的各个方面的测试变得尽可能轻松。

- 系统设置
  - *安装 PHPUnit*
    - *Composer*
    - *Phar*
- 测试应用程序
  - *PHPUnit 配置*

- 测试类
  - 搭建环境
  - *Traits*
  - 其他断言
  - 访问 *Protected/Private* 属性
- 模拟服务
  - *Services::injectMock()*
  - *Services::reset()*
  - *Services::resetSingle(string $name)*
- 模拟 *Factory* 实例
- 测试和时间

系统设置

安装 PHPUnit

CodeIgniter 使用 PHPUnit 作为所有测试的基础。有两种在系统内安装 PHPUnit 的方法。

Composer

推荐的方法是使用 Composer 在项目中安装它。尽管可以全局安装, 但我们不建议这样做, 因为随着时间的推移, 它可能与系统上的其他项目造成兼容性问题。

确保系统中安装了 Composer。从项目根目录 (包含应用程序和系统目录的目录) 命令行输入以下命令:

```
composer require --dev phpunit/phpunit
```

这将为当前 PHP 版本安装正确的版本。完成后, 可以通过输入以下命令来运行此项目的所有测试:

```
vendor/bin/phpunit
```

如果使用 Windows, 请使用以下命令:

```
vendor\bin\phpunit
```

Phar

另一种选择是从 PHPUnit 站点下载.phar 文件。这是一个独立的文件, 应该放在项目根目录中。

测试应用程序

PHPUnit 配置

在你的 CodeIgniter 项目根目录中，有一个 `phpunit.xml.dist` 文件。这个文件控制着你的应用程序的单元测试。如果你提供了自己的 `phpunit.xml`，它将覆盖默认文件。

默认情况下，测试文件放置在项目根目录下的 **tests** 目录中。

测试类

为了利用提供的额外工具，你的测试必须继承 `CodeIgniter\Test\CIUnitTestCase`。

对于测试文件的放置位置没有硬性规定。然而，我们建议你提前制定放置规则，以便你能快速了解测试文件的位置。

在本文档中，与 **app** 目录中的类对应的测试文件将放置在 **tests/app** 目录中。要测试一个新的库 **app/Libraries/Foo.php**，你需要在 **tests/app/Libraries/FooTest.php** 创建一个新文件：

```
<?php

namespace App\Libraries;

use CodeIgniter\Test\CIUnitTestCase;

class FooTest extends CIUnitTestCase
{
    public function testFooNotBar()
```

(续下页)

(接上页)

```
    {
        // ...
    }
}
```

要测试你的某个模型 **app/Models/UserModel.php**，你可能会在 **tests/app/Models/UserModelTest.php** 中得到如下内容：

```
<?php

namespace App\Models;

use CodeIgniter\Test\CIUnitTestCase;

class UserModelTest extends CIUnitTestCase
{
    public function testFooNotBar()
    {
        // ...
    }
}
```

你可以创建任何适合测试风格或需求的目录结构。在给测试类加命名空间时，请记住 **app** 目录是 `App` 命名空间的根目录，因此所使用的任何类都必须与 `App` 具有正确的相对命名空间。

备注： 对测试类使用命名空间不是强制的，但它有助于确保类名不冲突。

在测试数据库结果时，必须在类中使用*DatabaseTestTrait*。

搭建环境

大多数测试都需要一些准备才能正确运行。PHPUnit 的 `TestCase` 提供了四个方法来帮助搭建环境和清理:

```
public static function setUpBeforeClass(): void
public static function tearDownAfterClass(): void

protected function setUp(): void
protected function tearDown(): void
```

静态方法 `setUpBeforeClass()` 和 `tearDownAfterClass()` 分别在整个测试用例之前和之后运行, 而受保护的方法 `setUp()` 和 `tearDown()` 在每个测试之间运行。

如果你实现了这些特殊函数中的任何一个，请确保你也运行它们的父级函数，以免扩展的测试用例干扰到分阶段测试:

```
<?php

namespace App\Models;

use CodeIgniter\Test\CIUnitTestCase;

final class UserModelTest extends CIUnitTestCase
{
    protected function setUp(): void
    {
        parent::setUp(); // Do not forget

        helper('text');
    }

    // ...
}
```

Traits

一种常见的增强测试的方法是使用 traits 来整合不同测试用例中的准备工作。`CIUnitTestCase` 会检测任何类的 traits，并查找以 trait 本身命名的准备方法来运行（即 *setUp{NameOfTrait}()* 和 *tearDown{NameOfTrait}()*）。

例如, 如果你需要在某些测试用例中添加认证, 可以创建一个具有假登录用户设置方法的认证 trait:

```
<?php

namespace App\Traits;

trait AuthTrait
{
    protected function setUpAuthTrait()
    {
        $user = $this->createFakeUser();
        $this->logInUser($user);
    }

    // ...
}
```

```
<?php

namespace Tests;

use App\Traits\AuthTrait;
use CodeIgniter\Test\CIUnitTestCase;

final class AuthenticationFeatureTest extends CIUnitTestCase
{
    use AuthTrait;

    // ...
}
```

其他断言

CIUnitTestCase 提供了你可能会发现有用的其他单元测试断言。

assertLogged($level, $expectedMessage)

确保预期的内容确实已记录到日志:

assertLogContains($level, $logMessage)

确保日志中存在包含消息片段的记录:

```
<?php

$config = new \Config\Logger();
$logger = new \CodeIgniter\Log\Logger($config);

// check verbatim the log message
$logger->log('error', "That's no moon");
$this->assertLogged('error', "That's no moon");

// check that a portion of the message is found in the logs
$exception = new \RuntimeException('Hello world.');
$logger->log('error', $exception->getTraceAsString());
$this->assertLogContains('error', '{main}');
```

assertEventTriggered($eventName)

确保预期触发的事件确实被触发了:

```
<?php

use CodeIgniter\Events\Events;

Events::on('foo', static function ($arg) use (&$result) {
    $result = $arg;
});
```

(续下页)

(接上页)

```
Events::trigger('foo', 'bar');

$this->assertEventTriggered('foo');
```

assertHeaderEmitted($header, $ignoreCase = false)

确保标头或 Cookie 已实际发送:

```
<?php

$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');
```

备注: 测试用例应作为单独的进程运行（使用 @runInSeparateProcess annotation 或 RunInSeparateProcess attribute）在 PHPUnit 中。

assertHeaderNotEmitted($header, $ignoreCase = false)

确保标头或 Cookie 没有被发送:

```
<?php

$response->setCookie('foo', 'bar');

ob_start();
$this->response->send();
$output = ob_get_clean(); // in case you want to check the actual␣
```

(续下页)

(接上页)

```
↪body

$this->assertHeaderNotEmitted('Set-Cookie: banana');
```

备注: 测试用例应作为单独的进程运行（使用 @runInSeparateProcess annotation 或 RunInSeparateProcess attribute）在 PHPUnit 中。

assertCloseEnough($expected, $actual, $message = '', $tolerance = 1)

对于延长执行时间的测试, 测试预期时间与实际时间之间的绝对差值是否在允许公差范围内:

```
<?php

use CodeIgniter\Debug\Timer;

$timer = new Timer();
$timer->start('longjohn', strtotime('-11 minutes'));
$this->assertCloseEnough(11 * 60, $timer->getElapsedTime('longjohn
↪'));
```

上面的测试将允许实际时间为 660 或 661 秒。

assertCloseEnoughString($expected, $actual, $message = '', $tolerance = 1)

对于延长执行时间的测试, 测试格式化为字符串的预期和实际时间之间的绝对差值是否在允许公差范围内:

```
<?php

use CodeIgniter\Debug\Timer;

$timer = new Timer();
$timer->start('longjohn', strtotime('-11 minutes'));
```

(续下页)

(接上页)

```
$this->assertCloseEnoughString(11 * 60, $timer->getElapsedTime(
↪'longjohn'));
```

上面的测试将允许实际时间为 660 或 661 秒。

访问 Protected/Private 属性

在测试期间, 可以使用以下 setter 和 getter 方法访问要测试类中的 protected 和 private 方法和属性。

getPrivateMethodInvoker($instance, $method)

启用你从类外调用私有方法。它返回一个可调用的函数。第一个参数是要测试的类的实例。第二个参数是要调用的方法名称。

```
<?php

use App\Libraries\Foo;

// 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)

从类的实例中检索私有/受保护类属性的值。第一个参数是要测试的类的实例。第二个参数是属性名称。

```
<?php

use App\Libraries\Foo;
```

(续下页)

(接上页)

```
// 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)

在类实例中设置受保护的值。第一个参数是要测试的类的实例。第二个参数是要设置值的属性名称。第三个参数是要设置的值:

```
<?php

use App\Libraries\Foo;

// Create an instance of the class to test
$obj = new Foo();

// Set the value
$this->setPrivateProperty($obj, 'baz', 'oops!');

// Do normal testing...
```

模拟服务

在测试中, 你经常会发现需要模拟 **app/Config/Services.php** 中定义的服务之一, 以将测试限制于仅检查相关代码, 同时模拟服务的各种响应。这在测试控制器和其他集成测试中尤其如此。**Services** 类提供了以下方法来简化此操作。

Services::injectMock()

此方法允许你定义 Services 类将返回的确切实例。你可以使用它来设置服务的属性, 以使其以某种方式运行, 或将服务替换为模拟类。

```
<?php

namespace Tests;

use CodeIgniter\HTTP\CURLRequest;
use CodeIgniter\Test\CIUnitTestCase;
use Config\Services;

final class SomeTest extends CIUnitTestCase
{
    public function testSomething()
    {
        $curlrequest = $this->getMockBuilder(CURLRequest::class)
            ->onlyMethods(['request'])
            ->getMock();
        Services::injectMock('curlrequest', $curlrequest);

        // Do normal testing here....
    }
}
```

第一个参数是要替换的服务。名称必须与 Services 类中的函数名称完全匹配。第二个参数是要替换的实例。

Services::reset()

从 Services 类中删除所有模拟类, 将其恢复到原始状态。

你也可以使用 `CIUnitTestCase` 提供的 `$this->resetServices()` 方法。

备注: 此方法会重置所有服务的状态, 并且 `RouteCollection` 将不包含任何路由。如果你想要使用加载的路由, 你需要调用 `loadRoutes()` 方法, 例如 `Services::routes()->loadRoutes()`。

Services::resetSingle(string $name)

通过名称删除单个服务的所有模拟和共享实例。

备注: Cache、Email 和 Session 服务默认进行模拟, 以防止侵入式测试行为。要阻止模拟, 请从类属性中删除方法回调:`$setUpMethods = ['mockEmail', 'mockSession'];`

模拟 Factory 实例

与 Services 类似, 在测试期间你可能需要提供预先配置的类实例用于 `Factories`。像 **Services** 一样使用相同的 `Factories::injectMock()` 和 `Factories::reset()` 静态方法, 但它们需要在前面附加组件名称作为额外参数:

```
<?php

namespace Tests;

use App\Models\UserModel;
use CodeIgniter\Config\Factories;
use CodeIgniter\Test\CIUnitTestCase;
use Tests\Support\Mock\MockUserModel;

final class SomeTest extends CIUnitTestCase
{
    protected function setUp(): void
    {
        parent::setUp();

        $model = new MockUserModel();
        Factories::injectMock('models', UserModel::class, $model);
    }
}
```

备注: 所有组件工厂在每个测试之间默认重置。如果需要实例持久化, 请修改测试用例

的 $setUpMethods。

测试和时间

测试依赖于时间的代码可能会很有挑战性。然而，当使用*Time*类时，可以在测试期间随意固定或更改当前时间。

下面是一个固定当前时间的样本测试代码：

```
<?php

namespace Tests;

use CodeIgniter\I18n\Time;
use CodeIgniter\Test\CIUnitTestCase;

final class TimeDependentCodeTest extends CIUnitTestCase
{
    protected function tearDown(): void
    {
        parent::tearDown();

        // Reset the current time.
        Time::setTestNow();
    }

    public function testFixTime(): void
    {
        // Fix the current time to "2023-11-25 12:00:00".
        Time::setTestNow('2023-11-25 12:00:00');

        // This assertion always passes.
        $this->assertSame('2023-11-25 12:00:00', (string)␣
↪Time::now());
    }
}
```

你可以使用 Time::setTestNow() 方法来固定当前时间。可选地，你可以指定一个语言环境作为第二个参数。

不要忘记在测试后调用该方法（不带参数）来重置当前时间。

8.1.2 测试数据库

- 测试类
- 设置测试数据库
  - 迁移和种子
- 帮助方法
  - 更改数据库状态
  - 从数据库获取数据
  - 断言

测试类

为了利用 CodeIgniter 为测试提供的内置数据库工具，你的测试必须扩展 `CIUnitTestCase` 并使用 `DatabaseTestTrait`:

```
<?php

namespace App\Database;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\DatabaseTestTrait;

class MyTests extends CIUnitTestCase
{
    use DatabaseTestTrait;

    // ...
}
```

由于在 `setUp()` 和 `tearDown()` 阶段执行了特殊功能，所以如果你需要使用这些方法，必须确保调用父类的方法，否则你将失去这里描述的大部分功能：

```php
<?php

namespace App\Database;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\DatabaseTestTrait;

class MyTests extends CIUnitTestCase
{
    use DatabaseTestTrait;

    protected function setUp(): void
    {
        parent::setUp();

        // Do something here....
    }

    protected function tearDown(): void
    {
        parent::tearDown();

        // Do something here....
    }
}
```

设置测试数据库

运行数据库测试时, 你需要提供可在测试期间使用的数据库。框架提供了特定于 CodeIgniter 的工具, 而不是使用 PHPUnit 内置的数据库功能。第一步是确保你在 **app/Config/Database.php** 中设置了 `tests` 数据库组。这指定了仅在运行测试时使用的数据库连接, 以保持其他数据的安全。

如果团队中有多个开发人员, 你可能希望将凭证保存在 **.env** 文件中。要这样做, 请编辑文件以确保存在以下行并具有正确的信息:

```
database.tests.hostname = localhost
database.tests.database = ci4_test
```

(续下页)

(接上页)

```
database.tests.username = root
database.tests.password = root
database.tests.DBDriver = MySQLi
database.tests.DBPrefix =
database.tests.port = 3306
```

迁移和种子

运行测试时, 你需要确保数据库具有正确的 schema 设置并且对每个测试处于已知状态。你可以使用迁移和种子来设置数据库, 方法是在测试中添加一些类属性。

```php
<?php

namespace App\Database;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\DatabaseTestTrait;

class MyTests extends CIUnitTestCase
{
    use DatabaseTestTrait;

    // For Migrations
    protected $migrate     = true;
    protected $migrateOnce = false;
    protected $refresh     = true;
    protected $namespace   = 'Tests\Support';

    // For Seeds
    protected $seedOnce = false;
    protected $seed     = 'TestSeeder';
    protected $basePath = 'path/to/database/files';

    // ...
}
```

迁移

$migrate

此布尔值确定是否在测试之前运行数据库迁移。默认情况下, 始终将数据库迁移到 `$namespace` 定义的最新可用状态。如果为 `false`, 则不运行迁移。如果要禁用迁移, 请设置为 `false`。

$migrateOnce

此布尔值确定是否只运行一次数据库迁移。如果要在首次测试之前运行一次迁移, 请设置为 `true`。如果不存在或为 `false`, 则在每次测试之前运行迁移。

$refresh

此布尔值确定是否在测试之前完全刷新数据库。如果为 `true`, 则所有迁移都会回滚到版本 0。

$namespace

默认情况下,CodeIgniter 将在 **tests/_support/Database/Migrations** 中查找在测试期间应运行的迁移。你可以在 `$namespace` 属性中指定新命名空间来更改此位置。这不应包括 **Database\Migrations** 子命名空间, 而只是基本命名空间。

重要: 如果将此属性设置为 `null`, 则像 `php spark migrate --all` 一样从所有可用的命名空间运行迁移。

种子

$seed

如果存在且非空, 则指定在测试运行之前用来向数据库填充测试数据的种子文件的名称。

$seedOnce

此布尔值确定是否只运行一次数据库种子。如果要在首次测试之前运行一次数据库种子, 请设置为 `true`。如果不存在或为 `false`, 则在每次测试之前运行数据库种子。

$basePath

默认情况下,CodeIgniter 将在 **tests/_support/Database/Seeds** 中查找在测试期间应运行的种子。你可以通过指定 `$basePath` 属性来更改此目录。这不应包括 **Seeds** 目录, 而是保存子目录的单个目录的路径。

帮助方法

DatabaseTestTrait 类提供了几个帮助方法来帮助测试数据库。

更改数据库状态

regressDatabase()

在上述 `$refresh` 期间调用, 如果需要手动重置数据库, 此方法可用。

migrateDatabase()

在 `setUp()` 期间调用, 如果需要手动运行迁移, 此方法可用。

seed($name)

允许你手动将 Seed 加载到数据库中。唯一的参数是要运行的种子的名称。种子必须存在于 `$basePath` 中指定的路径内。

hasInDatabase($table, $data)

将新行插入数据库中。此行在当前测试运行后被删除。$data 是一个包含要插入表中的数据的关联数组。

```
<?php

$data = [
    'email' => 'joe@example.com',
    'name'  => 'Joe Cool',
];
$this->hasInDatabase('users', $data);
```

从数据库获取数据

grabFromDatabase($table, $column, $criteria)

返回在行与 $criteria 匹配的指定表中的 $column 的值。如果找到多行, 它只会返回第一行。

```
<?php

$username = $this->grabFromDatabase('users', 'username', ['email' =>
↪ 'joe@example.com']);
```

断言

dontSeeInDatabase($table, $criteria)

断言与 $criteria 中的键/值对匹配的行在数据库中不存在。

```
<?php

$criteria = [
    'email'  => 'joe@example.com',
    'active' => 1,
```

(续下页)

(接上页)

```
];
$this->dontSeeInDatabase('users', $criteria);
```

seeInDatabase($table, $criteria)

断言与 `$criteria` 中的键/值对匹配的行在数据库中存在。

```
<?php

$criteria = [
    'email'  => 'joe@example.com',
    'active' => 1,
];
$this->seeInDatabase('users', $criteria);
```

seeNumRecords($expected, $table, $criteria)

断言在数据库中找到的与 `$criteria` 匹配的行数。

```
<?php

$criteria = [
    'active' => 1,
];
$this->seeNumRecords(2, 'users', $criteria);
```

8.1.3 生成测试数据

通常，你需要示例数据来运行应用程序的测试。`Fabricator` 类使用 Faker 将模型转换为随机数据生成器。在你的种子或测试用例中使用 fabricators 来为单元测试准备虚假数据。

- *支持的模型*
- *加载 Fabricator*

- 定义 *Formatter*
  - 高级格式化
- 设置修饰符
- 本地化
- 伪造数据
- 指定测试数据
- 测试辅助函数
- 表计数
  - 方法

支持的模型

Fabricator 支持任何扩展框架核心模型 CodeIgniter\Model 的模型。你可以通过确保自己的自定义模型实现 CodeIgniter\Test\Interfaces\FabricatorModel 接口来使用它们:

```
<?php

namespace App\Models;

use CodeIgniter\Test\Interfaces\FabricatorModel;

class MyModel implements FabricatorModel
{
    public function find($id = null)
    {
        // TODO: Implement find() method.
    }

    public function insert($row = null, bool $returnID = true)
    {
        // TODO: Implement insert() method.
    }
```

(续下页)

(接上页)

```
    // ...
}
```

备注: 除了方法之外, 接口还概述了目标模型所需的一些必要属性。请参阅接口代码以获取详细信息。

加载 Fabricator

最基本的 fabricator 只需要模型进行操作:

```
<?php

use App\Models\UserModel;
use CodeIgniter\Test\Fabricator;

$fabricator = new Fabricator(UserModel::class);
```

参数可以是指定模型名称的字符串, 也可以是模型实例本身:

```
<?php

use App\Models\UserModel;
use CodeIgniter\Test\Fabricator;

$model = new UserModel($testDbConnection);

$fabricator = new Fabricator($model);
```

定义 Formatter

Faker 通过从 formatter 请求数据来生成数据。如果没有定义 formatter, `Fabricator` 将尝试根据字段名称和它所表示的模型的属性来猜测最合适的匹配, 如果找不到则回退到 `$fabricator->defaultFormatter`。如果字段名称与常用 formatter 对应, 或者你不太关心字段的内容, 这可能就可以了, 但大多数情况下你会想指定要使用的 formatter, 可以将它们作为构造函数的第二个参数:

```
<?php

use App\Models\UserModel;
use CodeIgniter\Test\Fabricator;

$formatters = [
    'first'  => 'firstName',
    'email'  => 'email',
    'phone'  => 'phoneNumber',
    'avatar' => 'imageUrl',
];

$fabricator = new Fabricator(UserModel::class, $formatters);
```

你也可以在初始化 fabricator 后使用 `setFormatters()` 方法更改 formatter。

高级格式化

有时 formatter 的默认返回值是不够的。Faker 提供者允许大多数 formatter 使用参数来进一步限制随机数据的范围。fabricator 将检查其代表模型的 `fake()` 方法, 在其中你可以定义伪造的数据应该是什么样的:

```
<?php

namespace App\Models;

use Faker\Generator;

class UserModel
{
    // ...

    public function fake(Generator &$faker)
    {
        return [
            'first'  => $faker->firstName(),
            'email'  => $faker->email(),
            'phone'  => $faker->phoneNumber(),
```

(续下页)

(接上页)

```
            'avatar' => \Faker\Provider\Image::imageUrl(800, 400),
            'login'  => config('Auth')->allowRemembering ? date('Y-
↪m-d') : null,
        ];

        /*
         * Or you can return a return type object.

        return new User([
            'first'  => $faker->firstName(),
            'email'  => $faker->email(),
            'phone'  => $faker->phoneNumber(),
            'avatar' => \Faker\Provider\Image::imageUrl(800, 400),
            'login'  => config('Auth')->allowRemembering ? date('Y-
↪m-d') : null,
        ]);

        */
    }
}
```

请注意，在这个例子中，前三个值等效于之前的 formatter。但是对于 `avatar` 我们请求了与默认不同的图像大小，`login` 使用基于应用配置的条件，这两者在使用 `$formatters` 参数时都是不可能的。

你可能希望将测试数据与生产模型分开，所以最好是在测试支持文件夹中定义一个子类：

```
<?php

namespace Tests\Support\Models;

use App\Models\UserModel;
use Faker\Generator;

class UserFabricator extends UserModel
{
    public function fake(Generator &$faker)
```

(续下页)

(接上页)

```
    {
        // ...
    }
}
```

设置修饰符

在 4.5.0 版本加入.

Faker 提供了三个特殊的提供者，`unique()`, `optional()` 和 `valid()`，可以在任何提供者之前调用。Fabricator 通过提供专用方法完全支持这些修饰符。

```
<?php

use App\Models\UserModel;
use CodeIgniter\Test\Fabricator;

$fabricator = new Fabricator(UserModel::class);
$fabricator->setUnique('email'); // sets generated emails to be␣
↪always unique
$fabricator->setOptional('group_id'); // sets group id to be␣
↪optional, with 50% chance to be `null`
$fabricator->setValid('age', static fn (int $age): bool => $age >=␣
↪18); // sets age to be 18 and above only

$users = $fabricator->make(10);
```

在字段名称之后传递的参数会直接按原样传递给修饰符。你可以参考 Faker 的修饰符文档 了解详细信息。

如果你在模型上使用 `fake()` 方法，你可以直接使用 Faker 的修饰符，而不是在 Fabricator 上调用每个方法。

```
<?php

namespace App\Models;

use CodeIgniter\Test\Fabricator;
```

(续下页)

(接上页)

```
use Faker\Generator;

class UserModel
{
    protected $table = 'users';

    public function fake(Generator &$faker)
    {
        return [
            'first'    => $faker->firstName(),
            'email'    => $faker->unique()->email(),
            'group_id' => $faker->optional()->passthrough(mt_rand(1,
↪ Fabricator::getCount('groups'))),
        ];
    }
}
```

本地化

Faker 支持许多不同的语言环境。请查看其文档以确定哪些提供程序支持你的语言环境。在初始化 fabricator 时, 可以在第三个参数中指定语言环境:

```
<?php

use App\Models\UserModel;
use CodeIgniter\Test\Fabricator;

$fabricator = new Fabricator(UserModel::class, null, 'fr_FR');
```

如果未指定语言环境, 它将使用在 **app/Config/App.php** 中定义为 `defaultLocale` 的语言环境。你可以使用其 `getLocale()` 方法查看现有 fabricator 的语言环境。

伪造数据

正确初始化 fabricator 后, 使用 `make()` 命令生成测试数据很容易:

```
<?php

use CodeIgniter\Test\Fabricator;
use Tests\Support\Models\UserFabricator;

$fabricator = new Fabricator(UserFabricator::class);
$testUser   = $fabricator->make();
print_r($testUser);
```

你可能会得到这样的返回:

```
<?php

[
    'first'  => 'Maynard',
    'email'  => 'king.alford@example.org',
    'phone'  => '201-886-0269 x3767',
    'avatar' => 'http://lorempixel.com/800/400/',
    'login'  => null,
];
```

你也可以通过提供数量来获取更多数据:

```
<?php

$users = $fabricator->make(10);
```

`make()` 的返回类型模拟代表模型中定义的类型, 但你可以使用方法直接强制类型:

```
<?php

$userArray  = $fabricator->makeArray();
$userObject = $fabricator->makeObject();
$userEntity = $fabricator->makeObject('App\Entities\User');
```

`make()` 的返回可在测试中使用或插入到数据库中。另一方面, `Fabricator` 包含 `create()` 命令可以帮你插入, 并返回结果。由于模型回调、数据库格式化和特殊键

(如主键和时间戳), `create()` 的返回可能与 `make()` 不同。你可能会得到这样的返回:

```
<?php

[
    'id'         => 1,
    'first'      => 'Rachel',
    'email'      => 'bradley72@gmail.com',
    'phone'      => '741-241-2356',
    'avatar'     => 'http://lorempixel.com/800/400/',
    'login'      => null,
    'created_at' => '2020-05-08 14:52:10',
    'updated_at' => '2020-05-08 14:52:10',
];
```

与 `make()` 类似, 你可以提供数量来插入和返回对象数组:

```
<?php

$users = $fabricator->create(100);
```

最后, 有时你可能希望使用完整的数据库对象进行测试, 但实际上你没有使用数据库。`create()` 的第二个参数允许模拟对象, 在不实际接触数据库的情况下返回带有额外数据库字段的对象:

```
<?php

$user = $fabricator(null, true);

$this->assertIsNumeric($user->id);
$this->dontSeeInDatabase('user', ['id' => $user->id]);
```

指定测试数据

生成的数据很好, 但有时你可能希望在不损害 formatter 配置的情况下为测试提供特定字段的值。与为每个变体创建新的 fabricator 相比, 你可以使用 `setOverrides()` 来指定任何字段的值:

```
<?php

$fabricator->setOverrides(['first' => 'Bobby']);
$bobbyUser = $fabricator->make();
```

现在通过 `make()` 或 `create()` 生成的任何数据将始终对 `first` 字段使用 “Bobby” :

```
<?php

[
    'first'  => 'Bobby',
    'email'  => 'latta.kindel@company.org',
    'phone'  => '251-806-2169',
    'avatar' => 'http://lorempixel.com/800/400/',
    'login'  => null,
];

[
    'first'  => 'Bobby',
    'email'  => 'melissa.strike@fabricon.us',
    'phone'  => '525-214-2656 x23546',
    'avatar' => 'http://lorempixel.com/800/400/',
    'login'  => null,
];
```

`setOverrides()` 可以带一个第二个参数来指示这是否应该是持久化的覆盖或者仅用于单个操作:

```
<?php

$fabricator->setOverrides(['first' => 'Bobby'], $persist = false);
$bobbyUser = $fabricator->make();
$bobbyUser = $fabricator->make();
```

请注意, 在第一次返回后,fabricator 停止使用覆盖:

```
<?php

[
```

(续下页)

(接上页)

```
    'first'  => 'Bobby',
    'email'  => 'belingadon142@example.org',
    'phone'  => '741-857-1933 x1351',
    'avatar' => 'http://lorempixel.com/800/400/',
    'login'  => null,
];

[
    'first'  => 'Hans',
    'email'  => 'hoppifur@metraxalon.com',
    'phone'  => '487-235-7006',
    'avatar' => 'http://lorempixel.com/800/400/',
    'login'  => null,
];
```

如果没有提供第二个参数, 则传递的值默认持久化。

测试辅助函数

通常你只需要一个一次性的伪对象用于测试。测试辅助函数提供了 `fake($model, $overrides, $persist = true)` 来实现这一目的:

```
<?php

helper('test');
$user = fake('App\Models\UserModel', ['name' => 'Gerry']);
```

这相当于:

```
<?php

use CodeIgniter\Test\Fabricator;

$fabricator = new Fabricator('App\Models\UserModel');
$fabricator->setOverrides(['name' => 'Gerry']);
$user = $fabricator->create();
```

如果你只需要一个不保存到数据库的伪对象, 可以将 persist 参数设置为 false。

表计数

经常地, 你的伪数据将依赖于其他伪数据。Fabricator 为每个表提供了已创建的伪项数的静态计数。考虑以下例子:

你的项目有用户和组。在测试用例中, 你想创建具有不同组大小的各种场景, 所以你使用 Fabricator 来创建一些组。现在你想要创建伪用户, 但不想分配给不存在的组 ID。你的模型的 fake 方法可以这样:

```
<?php

namespace App\Models;

use CodeIgniter\Test\Fabricator;
use Faker\Generator;

class UserModel
{
    protected $table = 'users';

    public function fake(Generator &$faker)
    {
        return [
            'first'    => $faker->firstName(),
            'email'    => $faker->email(),
            'group_id' => mt_rand(1, Fabricator::getCount('groups
↪')),
        ];
    }
}
```

现在创建新用户将确保它属于有效的组:`$user = fake(UserModel::class);`

方法

`Fabricator` 在内部处理计数, 但你也可以访问这些静态方法来帮助使用它们:

getCount(string $table): int

返回特定表的当前值 (默认值:0)。

setCount(string $table, int $count): int

手动设置特定表的值, 例如, 如果你创建了一些没有使用 fabricator 的测试项, 但仍想将它们计入最终计数。

upCount(string $table): int

将特定表的值递增 1 并返回新的值。(这是在 `Fabricator::create()` 内部使用的)。

downCount(string $table): int

将特定表的值递减 1 并返回新的值, 例如, 如果你删除了一个伪对象, 但想跟踪更改。

resetCounts()

重置所有计数。在测试用例之间调用这一方法是很不错的主意 (虽然使用 `CIUnitTestCase::$refresh = true` 会自动完成此操作)。

8.1.4 测试控制器

几个新帮助类和 trait 使得测试控制器变得方便。在测试控制器时, 你可以执行控制器中的代码, 而无需先运行整个应用程序引导过程。通常, 使用功能测试工具 将更简单, 但如果需要, 此功能仍可提供。

备注: 由于整个框架没有启动, 所以有时你无法以这种方式测试控制器。

- 帮助 Trait
- 帮助方法
  - controller($class)
  - execute(string $method, ...$params)
  - withConfig($config)
  - withRequest($request)
  - withResponse($response)
  - withLogger($logger)
  - withUri(string $uri)
  - withBody($body)
- 检查响应
- 过滤器测试
  - Helper Trait
  - 配置
  - 检查路由
  - 调用过滤器方法
  - 断言

帮助 Trait

要启用控制器测试, 你需要在测试中使用 `ControllerTestTrait` trait:

```
<?php

namespace App\Controllers;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\ControllerTestTrait;
use CodeIgniter\Test\DatabaseTestTrait;
```

(续下页)

(接上页)

```
class FooControllerTest extends CIUnitTestCase
{
    use ControllerTestTrait;
    use DatabaseTestTrait;
}
```

一旦包含了 trait, 你就可以开始设置环境, 包括请求和响应类、请求体、URI 等。你可以使用 `controller()` 方法指定要使用的控制器, 传入控制器的完全限定类名。最后, 用要运行的方法名作为参数调用 `execute()` 方法:

```
<?php

namespace App\Controllers;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\ControllerTestTrait;
use CodeIgniter\Test\DatabaseTestTrait;

class ForumControllerTest extends CIUnitTestCase
{
    use ControllerTestTrait;
    use DatabaseTestTrait;

    public function testShowCategories()
    {
        $result = $this->withUri('http://example.com/categories')
            ->controller(ForumController::class)
            ->execute('showCategories');

        $this->assertTrue($result->isOK());
    }
}
```

帮助方法

controller($class)

指定要测试的控制器的类名。第一个参数必须是完全限定的类名 (即包含命名空间):

```
<?php

$this->controller(\App\Controllers\ForumController::class);
```

execute(string $method, ...$params)

在控制器内执行指定的方法。第一个参数是要运行的方法名:

```
<?php

$results = $this->controller(\App\Controllers\
↪ForumController::class)
    ->execute('showCategories');
```

通过指定第二个和后续参数, 你可以将它们传递给控制器方法。

这将返回一个新的帮助类, 它提供了许多用于检查响应本身的例程。有关详细信息, 请参阅下文。

withConfig($config)

允许你传入修改后的 **app/Config/App.php** 以使用不同设置进行测试:

```
<?php

$config               = new \Config\App();
$config->appTimezone = 'America/Chicago';

$results = $this->withConfig($config)
    ->controller(\App\Controllers\ForumController::class)
    ->execute('showCategories');
```

如果未提供, 将使用应用程序的 App 配置文件。

withRequest($request)

允许你提供适合测试需求的 **IncomingRequest** 实例:

```
<?php

$request = new \CodeIgniter\HTTP\IncomingRequest(
    new \Config\App(),
    new \CodeIgniter\HTTP\URI('http://example.com'),
    null,
    new \CodeIgniter\HTTP\UserAgent(),
);

$request->setLocale($locale);

$results = $this->withRequest($request)
    ->controller(\App\Controllers\ForumController::class)
    ->execute('showCategories');
```

如果未提供, 将使用具有默认应用程序值的新的 IncomingRequest 实例传入控制器。

withResponse($response)

允许你提供 **Response** 实例:

```
<?php

$response = new \CodeIgniter\HTTP\Response(new \Config\App());

$results = $this->withResponse($response)
    ->controller(\App\Controllers\ForumController::class)
    ->execute('showCategories');
```

如果未提供, 将使用具有默认应用程序值的新的 Response 实例传入控制器。

withLogger($logger)

允许你提供 **Logger** 实例:

```
<?php

$logger = new \CodeIgniter\Log\Handlers\FileHandler();

$results = $this->withResponse($response)
    ->withLogger($logger)
    ->controller(\App\Controllers\ForumController::class)
    ->execute('showCategories');
```

如果未提供, 将使用具有默认配置值的新的 Logger 实例传入控制器。

withUri(string $uri)

允许你提供新的 URI, 模拟客户端访问此控制器时的 URL。如果你需要在控制器中检查 URI 片段, 这很有帮助。唯一的参数是一个表示有效 URI 的字符串:

```
<?php

$results = $this->withUri('http://example.com/forums/categories')
    ->controller(\App\Controllers\ForumController::class)
    ->execute('showCategories');
```

在测试期间始终提供 URI 可以避免意外情况, 这是一种好的实践。

备注: 从 v4.4.0 版本开始, 该方法使用 URI 创建一个新的 Request 实例。因为 Request 实例应该具有 URI 实例。如果 URI 字符串中的主机名与 `Config\App` 中的设置不匹配, 将会设置有效的主机名。

---

withBody($body)

允许你为请求提供自定义主体。当测试 API 控制器并需要将 JSON 值设置为主体时, 这很有用。唯一的参数是一个表示请求主体的字符串:

```
<?php

$body = json_encode(['foo' => 'bar']);

$results = $this->withBody($body)
    ->controller(\App\Controllers\ForumController::class)
    ->execute('showCategories');
```

检查响应

`ControllerTestTrait::execute()` 返回 `TestResponse` 的一个实例。请参见测试响应 以了解如何使用此类在测试用例中执行其他断言和验证。

过滤器测试

与控制器测试类似, 框架提供了工具来帮助针对自定义过滤器 及项目中的使用方式进行测试。

Helper Trait

与控制器测试器一样, 你需要在测试用例中包含 `FilterTestTrait` 来启用这些功能:

```
<?php

namespace App\Filters;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\FilterTestTrait;

class FooFilterTest extends CIUnitTestCase
{
```

(续下页)

(接上页)

```
    use FilterTestTrait;
}
```

配置

由于与控制器测试的逻辑重叠, FilterTestTrait 旨在与 ControllerTestTrait 一起使用, 如果同一个类需要两者。一旦包含了 trait, CIUnitTestCase 将检测其 setUp 方法并准备测试所需的所有组件。如果需要特殊配置, 可以在调用支持方法之前更改任何属性:

- $request 准备好的默认 IncomingRequest 服务的版本
- $response 准备好的默认 ResponseInterface 服务的版本
- $filtersConfig 默认的 Config\Filters 配置 (注意: 发现由 Filters 处理, 所以不会包括模块别名)
- $filters 使用上述三个组件的 CodeIgniter\Filters\Filters 实例
- $collection 准备好的 RouteCollection 版本, 其中包括 Config\Routes 的发现

默认配置通常对测试最有利, 因为它最接近 "实时" 项目, 但是 (例如) 如果要模拟过滤器意外触发未过滤的路由, 可以将其添加到 Config 中:

```
<?php

namespace App\Filters;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\FilterTestTrait;

final class FooFilterTest extends CIUnitTestCase
{
    use FilterTestTrait;

    protected function testFilterFailsOnAdminRoute()
    {
        $this->filtersConfig->globals['before'] = ['admin-only-
```

(续下页)

(接上页)

```
↪filter'];

        $this->assertHasFilters('unfiltered/route', 'before');
    }

    // ...
}
```

检查路由

第一个帮助方法是 `getFiltersForRoute()`，它将模拟提供的路由并返回将为给定位置（“before” 或 “after”）运行的所有过滤器列表（按其别名），而不实际执行任何控制器或路由代码。这比控制器和 HTTP 测试具有很大的性能优势。

getFiltersForRoute(*$route*, *$position*)

参数

- **$route** (string) – 要检查的 URI
- **$position** (string) – 要检查的过滤器方法，“before” 或 “after”

返回

将运行的每个过滤器的别名

返回类型

string[]

用法示例:

```
<?php

$result = $this->getFiltersForRoute('/', 'after'); // ['toolbar
↪']
```

调用过滤器方法

配置中描述的属性都设置好了，以确保不干扰或不受其他测试的干扰的最大性能。下一个帮助方法将使用这些属性返回一个可调用的方法来安全地测试过滤器代码并检查结果。

getFilterCaller(*$filter*, *$position*)

参数

- **$filter** (FilterInterface|string) –过滤器实例、类或别名
- **$position** (string) –要运行的过滤器方法，"before" 或 "after"

返回

模拟过滤器事件的可调用方法

返回类型

Closure

用法示例:

```
<?php

namespace App\Filters;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\FilterTestTrait;

final class FooFilterTest extends CIUnitTestCase
{
    use FilterTestTrait;

    protected function testUnauthorizedAccessRedirects()
    {
        $caller = $this->getFilterCaller('permission', 'before
↪');
        $result = $caller('MayEditWidgets');

        $this->assertInstanceOf('CodeIgniter\HTTP\
↪RedirectResponse', $result);
```

(续下页)

(接上页)

```
    }
}
```

请注意, `Closure` 可以接受输入参数, 这些参数会传入过滤器方法。

断言

除了上面的帮助方法之外, `FilterTestTrait` 还带有一些断言来简化测试方法。

assertFilter()

`assertFilter()` 方法检查给定路由在指定位置使用了过滤器 (按其别名):

```
<?php

// Make sure users are logged in before checking their account
$this->assertFilter('users/account', 'before', 'login');
```

assertNotFilter()

`assertNotFilter()` 方法检查给定路由在指定位置没有使用过滤器 (按其别名):

```
<?php

// Make sure API calls do not try to use the Debug Toolbar
$this->assertNotFilter('api/v1/widgets', 'after', 'toolbar');
```

assertHasFilters()

`assertHasFilters()` 方法检查给定路由在指定位置至少设置了一个过滤器:

```
<?php

// Make sure that filters are enabled
$this->assertHasFilters('filtered/route', 'after');
```

assertNotHasFilters()

`assertNotHasFilters()` 方法检查给定路由在指定位置没有设置任何过滤器:

```
<?php

// Make sure no filters run for our static pages
$this->assertNotHasFilters('about/contact', 'before');
```

8.1.5 HTTP 功能测试

功能测试允许你查看对应用程序的单次调用的结果。这可能是返回单个网页表单的结果, 访问 API 端点等等。这很方便, 因为它允许你测试单个请求的整个生命周期, 确保路由工作正常, 响应格式正确, 分析结果等等。

- 测试类
- 请求页面
  - 缩写方法
  - 设置不同的路由
  - 设置会话值
  - 设置标头
  - 绕过事件
  - 格式化请求
  - 设置 *Body*
- 检查响应

测试类

功能测试要求所有测试类使用 `CodeIgniter\Test\DatabaseTestTrait` 和 `CodeIgniter\Test\FeatureTestTrait` traits。由于这些测试工具依赖于适当的数据库准备，如果实现自己的方法，必须始终确保调用 `parent::setUp()` 和 `parent::tearDown()`。

```
<?php

namespace Tests\Feature;

use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\DatabaseTestTrait;
use CodeIgniter\Test\FeatureTestTrait;

class FooTest extends CIUnitTestCase
{
    use DatabaseTestTrait;
    use FeatureTestTrait;

    protected function setUp(): void
    {
        parent::setUp();

        $this->myClassMethod();
    }

    protected function tearDown(): void
    {
        parent::tearDown();

        $this->anotherClassMethod();
    }
}
```

请求页面

基本上，功能测试允许你调用应用程序上的一个端点，并获取结果返回。为此，你可以使用 `call()` 方法。

1. 第一个参数是要使用的 HTTP 方法（通常是 `GET` 或 `POST`）。
2. 第二个参数是要测试的站点上的 URI 路径。
3. 第三个参数 `$params` 接受一个数组，用于填充你正在使用的 HTTP 动词的超全局变量。因此，**GET** 方法将填充 `$_GET` 变量，而 **POST** 请求将填充 `$_POST` 数组。`$params` 也用于格式化请求。

> **备注：** `$params` 数组并不适用于每个 HTTP 动词，但为了保持一致性而包含在内。

```
// Get a simple page
$result = $this->call('GET', '/');

// Submit a form
$result = $this->call('post', 'contact', [
    'name'  => 'Fred Flintstone',
    'email' => 'flintyfred@example.com',
]);
```

缩写方法

为每个 HTTP 动词提供了缩写方法, 以减少输入并增加清晰度:

```
$this->get($path, $params);
$this->post($path, $params);
$this->put($path, $params);
$this->patch($path, $params);
$this->delete($path, $params);
$this->options($path, $params);
```

设置不同的路由

你可以通过将 “routes” 数组传递到 `withRoutes()` 方法来使用自定义路由集合。这将覆盖系统中的任何现有路由:

```
$routes = [
    ['GET', 'users', 'UserController::list'],
];

$result = $this->withRoutes($routes)->get('users');
```

每个 “routes” 都是一个包含 HTTP 动词 (或 “add” 表示全部)、要匹配的 URI 和路由目的地的 3 元素数组。

设置会话值

你可以使用 `withSession()` 方法在单次测试期间设置自定义会话值。这需要一个键/值对数组, 在发出此请求时, 它应存在于 `$_SESSION` 变量中, 或者为 `null` 表示应使用 `$_SESSION` 的当前值。这在测试认证等方面很有用。

```
$values = [
    'logged_in' => 123,
];

$result = $this->withSession($values)->get('admin');

// Or...

$_SESSION['logged_in'] = 123;

$result = $this->withSession()->get('admin');
```

设置标头

你可以使用 `withHeaders()` 方法设置标头值。这需要一个键/值对数组, 它将作为调用中的标头传递:

```
$headers = [
    'CONTENT_TYPE' => 'application/json',
];

$result = $this->withHeaders($headers)->post('users');
```

绕过事件

事件在应用程序中很有用, 但在测试中可能 problematic。特别是用于发送电子邮件的事件。你可以使用 `skipEvents()` 方法告诉系统跳过任何事件处理:

```
$result = $this->skipEvents()->post('users', $userInfo);
```

格式化请求

你可以使用 `withBodyFormat()` 方法设置请求体的格式。目前支持 `json` 或 `xml`。这在测试 JSON 或 XML API 时非常有用，因为你可以设置请求的格式，以符合控制器的预期。

这将接收传递给 `call()`, `post()`, `get()` …的参数，并将它们分配给请求体，以给定的格式。

这还将相应地设置请求的 *Content-Type* 标头。

```
// If your feature test contains this:
$result = $this->withBodyFormat('json')->post('users', $userInfo);

// Your controller can then get the parameters passed in with:
$userInfo = $this->request->getJson();
```

设置 Body

你可以使用 `withBody()` 方法设置请求的 Body。这允许你按照所需的格式设置请求 Body。如果你有更复杂的 XML 需要测试，建议使用此方法。

这不会为你设置 *Content-Type* 标头。如果需要，你可以使用 `withHeaders()` 方法设置它。

检查响应

`FeatureTestTrait::call()` 返回 `TestResponse` 的一个实例。请参阅测试响应以了解如何使用此类在测试用例中执行其他断言和验证。

8.1.6 测试响应

`TestResponse` 类提供了许多有用的函数来解析测试用例中的响应并对其进行测试。通常, `TestResponse` 将作为你的结果提供, 控制器测试 或*HTTP 功能测试*, 但你始终可以直接使用任何 `ResponseInterface` 创建自己的:

```
$result = new \CodeIgniter\Test\TestResponse($response);
$result->assertOK();
```


- 测试响应
  - 访问请求/响应
  - 检查响应状态
  - *Session* 断言
  - *Header* 断言
  - *Cookie* 断言
  - *DOM* 辅助函数
  - *DOM* 断言
  - 使用 *JSON*
  - 使用 *XML*

测试响应

无论你是从测试中获得 `TestResponse` 还是自己创建, 都可以在测试中使用许多新的断言。

访问请求/响应

request()

如果在测试期间设置了请求, 你可以直接访问请求对象:

```
$request = $results->request();
```

response()

这允许你直接访问响应对象:

```
$response = $results->response();
```

检查响应状态

isOK()

根据响应是否被视为“正常”返回布尔值 true/false。这主要由 200 或 300 范围内的响应状态代码确定。如果重定向, 空响应正文不被视为有效。

```
if ($result->isOK()) {
    // ...
}
```

assertOK()

此断言简单地使用 `isOK()` 方法来测试响应。`assertNotOK()` 是此断言的逆。

```
$result->assertOK();
```

isRedirect()

根据响应是否重定向返回布尔值 true/false。

```
if ($result->isRedirect()) {
    // ...
}
```

assertRedirect()

断言响应是 RedirectResponse 的一个实例。`assertNotRedirect()` 是此断言的逆。

```
$result->assertRedirect();
```

assertRedirectTo()

断言响应是一个 RedirectResponse 实例, 且目标与给定的 uri 匹配。

```
$result->assertRedirectTo('foo/bar');
```

getRedirectUrl()

返回设置为 RedirectResponse 的 URL, 如果失败则为 null。

```
$url = $result->getRedirectUrl();
$this->assertEquals(site_url('foo/bar'), $url);
```

assertStatus(int $code)

断言返回的 HTTP 状态码匹配 $code。

```
$result->assertStatus(403);
```

Session 断言

assertSessionHas(string $key, $value = null)

断言结果 Session 中存在一个值。如果传递了 $value, 还将断言变量的值与指定的相匹配。

```
$result->assertSessionHas('logged_in', 123);
```

assertSessionMissing(string $key)

断言结果 Session 不包括指定的 $key。

```
$result->assertSessionMissing('logged_in');
```

Header 断言

assertHeader(string $key, $value = null)

断言响应中存在一个名为 `$key` 的 Header。如果 `$value` 非空, 还将断言值匹配。

```
$result->assertHeader('Content-Type', 'text/html');
```

assertHeaderMissing(string $key)

断言响应中不存在名为 `$key` 的 Header。

```
$result->assertHeader('Accepts');
```

Cookie 断言

assertCookie(string $key, $value = null, string $prefix = '')

断言响应中存在一个名为 `$key` 的 Cookie。如果 `$value` 非空, 还将断言值匹配。如有必要, 可以通过第三个参数传入前缀来设置 Cookie 前缀。

```
$result->assertCookie('foo', 'bar');
```

assertCookieMissing(string $key)

断言响应中不存在名为 `$key` 的 Cookie。

```
$result->assertCookieMissing('ci_session');
```

assertCookieExpired(string $key, string $prefix = '')

断言一个名为 `$key` 的 Cookie 存在, 但已过期。如有必要, 可以通过第二个参数传入前缀来设置 Cookie 前缀。

```
$result->assertCookieExpired('foo');
```

DOM 辅助函数

你得到的响应包含许多帮助方法来检查响应中的 HTML 输出。这些在测试中的断言中很有用。

see()

根据页面上的文本是否存在，返回一个布尔值 true/false。可以通过 type、class 或 id 来指定文本所在的标签。

```
// Check that "Hello World" is on the page
if ($results->see('Hello World')) {
    // ...
```

(续下页)

(接上页)

```
}

// Check that "Hello World" is within an h1 tag
if ($results->see('Hello World', 'h1')) {
    // ...
}

// Check that "Hello World" is within an element with the "notice"␣
↪class
if ($results->see('Hello World', '.notice')) {
    // ...
}

// Check that "Hello World" is within an element with id of "title"
if ($results->see('Hello World', '#title')) {
    // ...
}
```

`dontSee()` 方法正好相反:

```
// Checks that "Hello World" does NOT exist on the page
if ($results->dontSee('Hello World')) {
    // ...
}

// Checks that "Hellow World" does NOT exist within any h1 tag
if ($results->dontSee('Hello World', 'h1')) {
    // ...
}
```

seeElement()

`seeElement()` 和 `dontSeeElement()` 与前面的方法非常相似, 但不检查元素的值。相反, 它们仅检查元素是否存在于页面上:

```
// Check that an element with class 'notice' exists
if ($results->seeElement('.notice')) {
```

(续下页)

(接上页)

```
    // ...
}

// Check that an element with id 'title' exists
if ($results->seeElement('#title')) {
    // ...
}

// Verify that an element with id 'title' does NOT exist
if ($results->dontSeeElement('#title')) {
    // ...
}
```

seeLink()

你可以使用 `seeLink()` 来确保页面上存在具有指定文本的链接:

```
// Check that a link exists with 'Upgrade Account' as the text::
if ($results->seeLink('Upgrade Account')) {
    // ...
}

// Check that a link exists with 'Upgrade Account' as the text, AND␣
↪a class of 'upsell'
if ($results->seeLink('Upgrade Account', '.upsell')) {
    // ...
}
```

seeInField()

`seeInField()` 方法检查是否存在具有给定名称和值的输入标签:

```
// Check that an input exists named 'user' with the value 'John Snow
↪'
if ($results->seeInField('user', 'John Snow')) {
    // ...
```

(续下页)

(接上页)

```
}

// Check a multi-dimensional input
if ($results->seeInField('user[name]', 'John Snow')) {
    // ...
}
```

seeCheckboxIsChecked()

最后, 你可以使用 `seeCheckboxIsChecked()` 方法检查复选框是否存在并已被选中:

```
// Check if checkbox is checked with class of 'foo'
if ($results->seeCheckboxIsChecked('.foo')) {
    // ...
}

// Check if checkbox with id of 'bar' is checked
if ($results->seeCheckboxIsChecked('#bar')) {
    // ...
}
```

seeXPath()

在 4.5.0 版本加入.

你可以使用 `seeXPath()` 来充分利用 xpath 提供的强大功能。此方法针对的是希望直接使用 DOMXPath 对象编写更复杂表达式的高级用户:

```
// Check that h1 element which contains class "heading" is on the␣
↪page
if ($results->seeXPath('//h1[contains(@class, "heading")]')) {
    // ...
}

// Check that h1 element which contains class "heading" have a text
↪"Hello World"
```

(续下页)

(接上页)

```
if ($results->seeXPath('//h1[contains(@class, "heading")][contains(.
↪,"Hello world")]')) {
    // ...
}
```

dontSeeXPath() 方法则完全相反:

```
// Check that h1 element which contains class "heading" does NOT␣
↪exist on the page
if ($results->dontSeeXPath('//h1[contains(@class, "heading")]')) {
    // ...
}

// Check that h1 element which contains class "heading" and text
↪"Hello World" does NOT exist on the page
if ($results->dontSeeXPath('//h1[contains(@class, "heading
↪")][contains(.,"Hello world")]')) {
    // ...
}
```

DOM 断言

你可以使用以下断言来测试响应正文中是否存在特定元素/文本等。

assertSee(string $search = null, string $element = null)

断言文本/HTML 存在于页面上, 无论是自身存在还是更具体地说是存在于由类型、类或 id 指定的标记内:

```
// Verify that "Hello World" is on the page
$result->assertSee('Hello World');

// Verify that "Hello World" is within an h1 tag
$result->assertSee('Hello World', 'h1');

// Verify that "Hello World" is within an element with the "notice"␣
```

(续下页)

(接上页)

```
↪class
$result->assertSee('Hello World', '.notice');

// Verify that "Hello World" is within an element with id of "title"
$result->assertSee('Hello World', '#title');
```

assertDontSee(string $search = null, string $element = null)

与 `assertSee()` 方法完全相反:

```
// Verify that "Hello World" does NOT exist on the page
$results->assertDontSee('Hello World');

// Verify that "Hello World" does NOT exist within any h1 tag
$results->assertDontSee('Hello World', 'h1');
```

assertSeeElement(string $search)

类似于 `assertSee()`, 但是这只检查存在的元素。它不检查特定文本:

```
// Verify that an element with class 'notice' exists
$results->assertSeeElement('.notice');

// Verify that an element with id 'title' exists
$results->assertSeeElement('#title');
```

assertDontSeeElement(string $search)

类似于 `assertSee()`, 但是这只检查缺失的现有元素。它不检查特定文本:

```
// Verify that an element with id 'title' does NOT exist
$results->assertDontSeeElement('#title');
```

assertSeeLink(string $text, string $details = null)

断言找到一个锚定标签, 其标签体匹配 `$text`:

```
// Verify that a link exists with 'Upgrade Account' as the text::
$results->assertSeeLink('Upgrade Account');

// Verify that a link exists with 'Upgrade Account' as the text,␣
↪AND a class of 'upsell'
$results->assertSeeLink('Upgrade Account', '.upsell');
```

assertSeeInField(string $field, string $value = null)

断言存在具有给定名称和值的输入标签:

```
// Verify that an input exists named 'user' with the value 'John␣
↪Snow'
$results->assertSeeInField('user', 'John Snow');

// Verify a multi-dimensional input
$results->assertSeeInField('user[name]', 'John Snow');
```

使用 JSON

响应通常会包含 JSON 响应, 特别是在使用 API 方法时。以下方法可以帮助测试响应。

getJSON()

此方法将以 JSON 字符串的形式返回响应正文:

```
/*
 * Response body is this:
 * ['foo' => 'bar']
 */

$json = $result->getJSON();
```

(续下页)

(接上页)

```
/*
 * $json is this:
 * {
 *     "foo": "bar"
 * }
`*/
```

你可以使用此方法来确定 `$response` 是否确实包含 JSON 内容:

```
// Verify the response is JSON
$this->assertTrue($result->getJSON() !== false);
```

备注: 请注意结果中的 JSON 字符串将美化打印。

assertJSONFragment(array $fragment)

断言 `$fragment` 在 JSON 响应中找到。它不需要匹配整个 JSON 值。

```
/*
 * Response body is this:
 * [
 *     'config' => ['key-a', 'key-b'],
 * ]
 */

// Is true
$result->assertJSONFragment(['config' => ['key-a']]);
```

assertJSONExact($test)

与 `assertJSONFragment()` 类似, 但检查整个 JSON 响应以确保完全匹配。

使用 XML

getXML()

如果应用程序返回 XML, 则可以通过此方法检索它。

8.1.7 测试 CLI 命令

- 使用 *MockInputOutput*
  - *MockInputOutput*
    - 辅助方法
  - 如何使用
- 不使用 *MockInputOutput*
  - 测试 *CLI* 输出
    - *StreamFilterTrait*
    - *CITestStreamFilter*
  - 测试 *CLI* 输入
    - *PhpStreamWrapper*

使用 MockInputOutput

在 4.5.0 版本加入.

MockInputOutput

MockInputOutput 提供了一种简单的方法来编写需要用户输入的命令的测试，例如 `CLI::prompt()`、`CLI::wait()` 和 `CLI::input()`。

你可以在测试执行期间用 `MockInputOutput` 替换 `InputOutput` 类来捕获输入和输出。

备注: 当你使用 `MockInputOutput` 时，你不需要使用*StreamFilterTrait*、*CITestStreamFilter* 和*PhpStreamWrapper*。

辅助方法

getOutput(?int $index = null): string

获取输出。

- 如果你像 `$io->getOutput()` 这样调用它，它会返回整个输出字符串。
- 如果你指定 `0` 或一个正数，它会返回输出数组项。每个项都有一个 `CLI::fwrite()` 调用的输出。
- 如果你指定一个负数 `-n`，它会返回输出数组的倒数第 `n` 项。

getOutputs(): array

返回输出数组。每个项都有一个 `CLI::fwrite()` 调用的输出。

如何使用

- `CLI::setInputOutput()` 可以将 `MockInputOutput` 实例设置到 `CLI` 类。
- `CLI::resetInputOutput()` 重置 `CLI` 类中的 `InputOutput` 实例。
- `MockInputOutput::setInputs()` 设置用户输入数组。
- `MockInputOutput::getOutput()` 获取命令输出。

以下测试代码用于测试命令 `spark db:table`:

```
<?php

use CodeIgniter\CLI\CLI;
use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\DatabaseTestTrait;
use CodeIgniter\Test\Mock\MockInputOutput;
```

(续下页)

(接上页)

```
final class DbTableTest extends CIUnitTestCase
{
    use DatabaseTestTrait;

    protected $migrateOnce = true;

    public function testDbTable(): void
    {
        // Set MockInputOutput to CLI.
        $io = new MockInputOutput();
        CLI::setInputOutput($io);

        // User will input "a" (invalid value) and "0".
        $io->setInputs(['a', '0']);

        command('db:table');

        // Get the whole output string.
        $output = $io->getOutput();

        $expected = 'Which table do you want to see? [0, 1, 2, 3, 4,
↪ 5, 6, 7, 8, 9]: a';
        $this->assertStringContainsString($expected, $output);

        $expected = 'Data of Table "db_migrations":';
        $this->assertStringContainsString($expected, $output);

        // Remove MockInputOutput.
        CLI::resetInputOutput();
    }
}
```

不使用 MockInputOutput

测试 CLI 输出

StreamFilterTrait

在 4.3.0 版本加入.

StreamFilterTrait 提供了这些辅助方法的替代方案。

你可能需要测试一些难以测试的东西。有时，捕获一个流，比如 PHP 自己的 STDOUT 或 STDERR，可能会有所帮助。`StreamFilterTrait` 帮助你捕获所选流的输出。

如何使用

- `StreamFilterTrait::getStreamFilterBuffer()` 从缓冲区获取捕获的数据。
- `StreamFilterTrait::resetStreamFilterBuffer()` 重置捕获的数据。

在你的一个测试用例中演示这一点的示例：

```
<?php

namespace Tests;

use CodeIgniter\CLI\CLI;
use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\StreamFilterTrait;

final class SomeTest extends CIUnitTestCase
{
    use StreamFilterTrait;

    public function testSomeOutput(): void
    {
        CLI::write('first.');

        $this->assertSame("\nfirst.\n", $this->
↪getStreamFilterBuffer());
```

(续下页)

(接上页)

```
        $this->resetStreamFilterBuffer();

        CLI::write('second.');

        $this->assertSame("second.\n", $this->
↪getStreamFilterBuffer());
    }
}
```

StreamFilterTrait 有一个自动调用的配置器。参见*Testing Traits*。

如果你在测试中重写了 setUp() 或 tearDown() 方法，那么你必须分别调用 parent::setUp() 和 parent::tearDown() 方法来配置 StreamFilterTrait。

CITestStreamFilter

CITestStreamFilter 用于手动/单次使用。

如果你只需要在一个测试中捕获流，那么可以手动向流添加过滤器，而不是使用 StreamFilterTrait trait。

如何使用

- CITestStreamFilter::registration() 过滤器注册。
- CITestStreamFilter::addOutputFilter() 向输出流添加过滤器。
- CITestStreamFilter::addErrorFilter() 向错误流添加过滤器。
- CITestStreamFilter::removeOutputFilter() 从输出流中移除过滤器。
- CITestStreamFilter::removeErrorFilter() 从错误流中移除过滤器。

```
<?php

namespace Tests;

use CodeIgniter\CLI\CLI;
use CodeIgniter\Test\CIUnitTestCase;
```

(续下页)

(接上页)

```
use CodeIgniter\Test\Filters\CITestStreamFilter;

final class SomeTest extends CIUnitTestCase
{
    public function testSomeOutput(): void
    {
        CITestStreamFilter::registration();
        CITestStreamFilter::addOutputFilter();

        CLI::write('first.');

        CITestStreamFilter::removeOutputFilter();
    }
}
```

测试 CLI 输入

PhpStreamWrapper

在 4.3.0 版本加入.

PhpStreamWrapper 提供了一种方法来编写需要用户输入的方法的测试，例如 `CLI::prompt()`、`CLI::wait()` 和 `CLI::input()`。

备注： PhpStreamWrapper 是一个流包装类。如果你不了解 PHP 的流包装器，请参见 PHP 手册中的 The streamWrapper class。

如何使用

- `PhpStreamWrapper::register()` 将 `PhpStreamWrapper` 注册到 `php` 协议。
- `PhpStreamWrapper::restore()` 将 php 协议包装器恢复为 PHP 内置包装器。
- `PhpStreamWrapper::setContent()` 设置输入数据。

重要: PhpStreamWrapper 仅用于测试 `php://stdin`。但是当你注册它时，它会处理所有 php 协议 流，例如 `php://stdout`、`php://stderr`、`php://memory`。因此强烈建议仅在需要时注册/注销 `PhpStreamWrapper`。否则，它在注册时会干扰其他内置的 php 流。

在你的一个测试用例中演示这一点的示例:

```
<?php

namespace Tests;

use CodeIgniter\CLI\CLI;
use CodeIgniter\Test\CIUnitTestCase;
use CodeIgniter\Test\PhpStreamWrapper;

final class SomeTest extends CIUnitTestCase
{
    public function testPrompt(): void
    {
        // Register the PhpStreamWrapper.
        PhpStreamWrapper::register();

        // Set the user input to 'red'. It will be provided as␣
↪`php://stdin` output.
        $expected = 'red';
        PhpStreamWrapper::setContent($expected);

        $output = CLI::prompt('What is your favorite color?');

        $this->assertSame($expected, $output);

        // Restore php protocol wrapper.
        PhpStreamWrapper::restore();
    }
}
```

8.1.8 模拟系统类

框架中的几个组件在测试期间提供了他们类的模拟版本。这些类可以在测试执行期间取代正常的类, 通常提供额外的断言来测试执行过程中是否发生了某些操作 (或没有发生)。这可能是检查数据是否正确缓存, 电子邮件是否正确发送等等。

- 缓存
  - 额外方法
  - 可用的断言

缓存

你可以使用 `mock()` 方法模拟缓存, 其唯一参数为 `CacheFactory`。

```
<?php

$mock = mock(\CodeIgniter\Cache\CacheFactory::class);
```

虽然这会返回可以直接使用的 `CodeIgniter\Test\Mock\MockCache` 实例, 但它也会将模拟插入到服务类中, 因此代码中的任何对 `service('cache')` 或 `Config\Services::cache()` 的调用都将在其位置使用模拟类。

如果在单个文件中的多个测试方法中使用这一点, 应该在测试 `setUp()` 期间调用 `clean()` 或 `bypass()` 方法, 以确保测试运行时拥有清晰的状态。

额外方法

你可以指示模拟的缓存处理程序永不执行任何缓存, 方法是使用 `bypass()` 方法。这将模拟使用 dummy 处理程序, 并确保测试不依赖于缓存的数据。

```
<?php

$mock = mock(\CodeIgniter\Cache\CacheFactory::class);
// Never cache any items during this test.
$mock->bypass();
```

可用的断言

在测试期间, 模拟类上提供了以下新的断言:

```
<?php

$mock = mock(\CodeIgniter\Cache\CacheFactory::class);

// Assert that a cached item named $key exists
$mock->assertHas($key);
// Assert that a cached item named $key exists with a value of
↪$value
$mock->assertHasValue($key, $value);
// Assert that a cached item named $key does NOT exist
$mock->assertMissing($key);
```

8.1.9 基准测试

CodeIgniter 提供了两种独立的工具来帮助你基准测试代码和测试不同选项: **Timer** 和 **Iterator**。Timer 允许你轻松计算脚本执行过程中的两点之间的时间。Iterator 允许你设置几种变体并运行这些测试, 记录性能和内存统计数据以帮助你决定哪种版本最佳。

Timer 类始终处于活动状态, 从框架被调用的那一刻开始, 直到向用户发送输出之前的最后一刻, 使整个系统执行时间非常准确。

- *使用 Timer*
  - *Timer::start()*
  - *Timer::stop()*
  - *timer()*
  - *Timer::record()*
  - *查看基准点*
  - *显示执行时间*
- *使用 Iterator*
  - *创建要运行的任务*

– 运行任务

使用 Timer

通过 Timer, 你可以测量应用程序执行过程中的两个时刻之间的时间。这使得测量应用程序不同方面的性能变得很简单。所有测量都使用 `start()` 和 `stop()` 方法完成。

Timer::start()

`start()` 方法接受一个参数: 计时器的名称。你可以使用任意字符串作为计时器的名称。它仅用于以后参考以知道哪个测量是哪个:

```
<?php

$benchmark = service('timer');
$benchmark->start('render view');
```

Timer::stop()

`stop()` 方法将要停止的计时器的名称作为唯一参数:

```
<?php

$benchmark->stop('render view');
```

名称不区分大小写, 但否则必须与启动计时器时给定的名称匹配。

timer()

或者, 你可以使用全局函数 `timer()` 来启动和停止计时器:

```
<?php

// Start the timer
timer('render view');
// Stop a running timer,
```

(续下页)

(接上页)

```
// if one of this name has been started
timer('render view');
```

Timer::record()

在 4.3.0 版本加入.

从 v4.3.0 开始, 如果你使用非常小的代码块进行基准测试, 也可以使用 `record()` 方法。它接受一个无参数的可调用对象并测量其执行时间。`start()` 和 `stop()` 方法将在函数调用周围自动调用。

```
<?php

$benchmark->record('slow_function', static function () { slow_
↪function('...'); });

/*
 * Same as:
 *
 * $benchmark->start('slow_function');
 * slow_function('...');
 * $benchmark->stop('slow_function');
*/
```

你也可以返回可调用对象的返回值以供进一步处理。

```
<?php

$length = $benchmark->record('string length', static fn () =>␣
↪strlen('CI4'));

/*
 * $length = 3
 *
 * Same as:
 *
 * $benchmark->start('string length');
```

(续下页)

(接上页)

```
 * $length = strlen('CI4');
 * $benchmark->stop('string length');
*/
```

将可调用对象作为第二个参数传递给 `timer()` 时, 也可使用相同的功能。

```
<?php

$length = timer('string length', static fn () => strlen('CI4'));

/*
 * $length = 3
 *
 * Same as:
 *
 * timer('string length');
 * $length = strlen('CI4');
 * timer('string length');
*/
```

查看基准点

当应用程序运行时,Timer 类会收集你设置的所有计时器。但它不会自动显示它们。你可以通过调用 `getTimers()` 方法检索所有计时器。它返回一个基准信息数组, 包括开始时间、结束时间和持续时间:

```
<?php

$timers = $benchmark->getTimers();
/*
 * Produces:
 * [
 *     'render view'  => [
 *         'start'    => 1234567890,
 *         'end'      => 1345678920,
 *         'duration' => 15.4315, // number of seconds
 *     ]
```

(续下页)

(接上页)

```
 * ]
 */
```

你可以通过作为唯一参数传递要显示的小数位数来更改计算出的持续时间的精度。默认值是小数点后 4 位数:

```
<?php

$timers = $benchmark->getTimers(6);
```

计时器会自动显示在*Debug 工具栏* 中。

显示执行时间

虽然 `getTimers()` 方法将为项目中的所有计时器提供原始数据, 但你可以使用 `getElapsedTime()` 方法以秒为单位检索单个计时器的持续时间。第一个参数是要显示的计时器的名称。第二个是要显示的小数位数。默认为 4:

```
<?php

echo timer()->getElapsedTime('render view');
// Displays: 0.0234
```

使用 Iterator

Iterator 是一个简单的工具, 旨在允许你尝试对一个解决方案的多个变体以查看速度差异和不同的内存使用模式。你可以添加任意数量的 "任务" 供它运行, 该类将在运行数百次或数千次任务后获得性能的更清晰图片。然后可以检索结果并由脚本使用, 或者以 HTML 表格显示。

创建要运行的任务

任务在 Closure 中定义。任务创建的任何输出都将自动丢弃。它们通过 `add()` 方法添加到 Iterator 类中。第一个参数是你要引用此测试的名称。第二个参数是 Closure 本身:

```
<?php

$iterator = new \CodeIgniter\Debug\Iterator();

$iterator->add('double', static function ($word = 'little') {
    "Some basic {$word} string test.";
});
```

运行任务

添加任务后, 可以使用 `run()` 方法循环任务多次。默认情况下, 它将每个任务运行 1000 次。对于大多数简单测试来说, 这可能就足够了。如果需要运行测试更多次, 可以将次数作为第一个参数传递:

```
<?php

// Run the tests 3000 times.
$htmlTable = $iterator->run(3000);
```

运行后, 它将返回一个包含测试结果的 HTML 表格。如果不想要结果, 可以将 `false` 作为第二个参数传递:

```
<?php

// Returns null.
$iterator->run(1000, false);
```

8.1.10 调试应用程序

- 检查日志
  - *CodeIgniter* 错误日志
  - 记录所有 *SQL* 查询
- 替换 *var_dump()*
  - 启用 *Kint*
  - 使用 *Kint*
- 调试工具栏
  - 启用工具栏
  - 设置基准点
  - 创建自定义收集器
  - 热重载

检查日志

CodeIgniter 错误日志

CodeIgniter 根据 **app/Config/Logger.php** 中的设置记录错误信息。

默认配置下，日志文件每天存储在 **writable/logs** 目录中。如果事情没有按预期进行，检查这些日志是个好主意！

你可以调整错误阈值以查看更多或更少的信息。详情请参见*Logging*。

记录所有 SQL 查询

CodeIgniter 发出的所有 SQL 查询都可以被记录。详情请参见*Database Events*。

替换 var_dump()

虽然使用 Xdebug 和一个好的 IDE 对调试你的应用程序是不可或缺的，但有时一个简单的 `var_dump()` 就足够了。CodeIgniter 通过捆绑优秀的 PHP 调试工具 Kint 使这一点变得更好。

这远远超出了你通常的工具，提供了许多替代数据，例如将时间戳格式化为可识别的日期，显示颜色的十六进制代码，将数组数据显示为易于阅读的表格，等等。

启用 Kint

默认情况下,Kint 仅在 **development** 和 **testing** 环境 中启用。当常量 `CI_DEBUG` 定义且值为 true 时就会启用它。常量定义在引导文件中 (例如 **app/Config/Boot/development.php**)。

使用 Kint

d()

`d()` 方法将传入的唯一参数的数据全部输出到屏幕, 允许脚本继续执行:

```
d($_SERVER);
```

dd()

这个方法与 `d()` 一样, 但会在输出数据后 `die()`, 请求不再执行后续代码。

trace()

这会以 Kint 独特的方式提供当前执行点的回溯:

```
trace();
```

更多信息请参考 Kint 文档。

调试工具栏

调试工具栏可以一目了然地查看当前页面请求的信息, 包括基准测试结果、执行的查询、请求和响应数据等。这在开发过程中有助于调试和优化。

备注: 调试工具栏仍在开发中, 许多计划的功能尚未实现。

启用工具栏

除 **production** 环境外, 调试工具栏在其他所有环境 下默认启用。当常量 `CI_DEBUG` 定义且值为 true 时就会显示。常量定义在引导文件中 (例如 **app/Config/Boot/development.php**), 可以在其中修改以确定显示的环境。

备注: 当你的 `baseURL` 设置 (在 **app/Config/App.php** 或 `app.baseURL` 在 **.env** 中) 与实际 URL 不匹配时, 不会显示调试工具栏。

工具栏本身显示为一个后置过滤器。你可以通过从 **app/Config/Filters.php** 文件的 `$required` (或 `$globals`) 属性中移除 `'toolbar'` 来阻止它运行。

备注: 在 v4.5.0 之前, 工具栏默认设置为 `$globals`。

选择显示内容

CodeIgniter 默认带有多个 Collector , 它们收集要在工具栏上显示的数据。你可以方便地自定义 Collector 。要确定显示哪些收集器, 请查看配置文件 **app/Config/Toolbar.php**:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Toolbar extends BaseConfig
```

(续下页)

(接上页)

```
{
    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,
    ];
    // ...
}
```

注释掉不想显示的收集器。通过提供完全限定类名, 这里可以添加自定义收集器。出现的收集器将决定显示的选项卡和时间线上的信息。

备注: 比如数据库和日志, 只有存在内容的选项卡才会显示。否则它们会在小屏幕上删除。

CodeIgniter 带有以下收集器:

- **Timers** 收集系统和应用程序的所有基准数据。
- **Database** 显示所有数据库连接执行的查询与耗时。
- **Logs** 显示记录的任何信息。长时间运行的系统中可能导致内存问题, 应禁用它。
- **Views** 在时间线显示视图渲染时间, 在单独选项卡显示传递给视图的数据。
- **Cache** 显示缓存命中、未命中信息和执行时间。
- **Files** 显示请求期间加载的所有文件的列表。
- **Routes** 显示当前和所有定义的路由信息。
- **Events** 显示请求期间加载的所有事件列表。

设置基准点

为展示定界器的基准数据, 必须使用特定语法命名标记点。

请参考基准测试库 文档中的说明。

创建自定义收集器

创建自定义收集器很简单。创建一个新的类, 使用命名空间完全限定以便自动加载, 它需要扩展 `CodeIgniter\Debug\Toolbar\Collectors\BaseCollector`。BaseCollector 提供可重写的方法, 并要求正确设置四个必需类属性以确定收集器的行为:

```
<?php

namespace MyNamespace;

use CodeIgniter\Debug\Toolbar\Collectors\BaseCollector;

class MyCollector extends BaseCollector
{
    protected $hasTimeline = false;

    protected $hasTabContent = false;

    protected $hasVarData = false;

    protected $title = '';
}
```

如果任何收集器要在时间线显示信息, 应将 `$hasTimeline` 设为 `true`。如果为 true, 需要实现 `formatTimelineData()` 方法来格式化返回显示的数据。

如果收集器要显示自定义选项卡和内容, 应将 `$hasTabContent` 设为 `true`。如果为 true, 需要提供 `$title`、实现 `display()` 渲染选项卡内容, 如果要在标题右侧显示额外信息, 还可能需要实现 `getTitleDetails()`。

如果要向 `Vars` 选项卡添加数据, 应将 `$hasVarData` 设为 `true`, 并实现 `getVarData()`。

`$title` 显示在打开的选项卡上。

显示工具栏选项卡

显示工具栏选项卡需要:

1. 在 `$title` 填入显示为工具栏和选项卡标题的文本。
2. 将 `$hasTabContent` 设为 `true`。
3. 实现 `display()` 方法渲染内容。
4. 可选地, 实现 `getTitleDetails()`。

`display()` 返回显示在选项卡中的 HTML, 不需要处理标题, 由工具栏自动处理。它应返回 HTML 字符串。

`getTitleDetails()` 返回在标题右侧显示的字符串, 可提供概述信息。比如, 数据库选项卡显示所有连接的查询总数, 文件选项卡显示总文件数。

提供时间线数据

提供时间线数据需要:

1. 将 `$hasTimeline` 设为 `true`。
2. 实现 `formatTimelineData()`。

`formatTimelineData()` 必须以时间线可以排序和显示的格式返回数组数组。内部数组必须包含:

```
<?php

$data[] = [
    'name'      => '',      // Name displayed on the left of the␣
↪timeline
    'component' => '',      // Name of the Component listed in the␣
↪middle of timeline
    'start'     => 0.00,    // start time, like microtime(true)
    'duration'  => 0.00,    // duration, like mircrotime(true) -␣
↪microtime(true)
];
```

提供 Vars

向 Vars 选项卡添加数据需要:

1. 将 `$hasVarData` 设为 `true`
2. 实现 `getVarData()`。

`getVarData()` 应返回包含要显示的键值对数组的数组。外部数组的键为 Vars 选项卡中的部分名称:

```
<?php

$data = [
    'section 1' => [
        'foo' => 'bar',
        'bar' => 'baz',
    ],
    'section 2' => [
        'foo' => 'bar',
        'bar' => 'baz',
    ],
];
```

热重载

在 4.4.0 版本加入.

调试工具栏包含一个名为热重载的功能，它允许你对应用程序的代码进行更改，并在浏览器中自动重新加载，而无需刷新页面。这在开发过程中非常省时。

在开发过程中启用热重载，你可以点击工具栏左侧的按钮，它看起来像一个刷新图标。这将在所有页面上启用热重载，直到你禁用它。

热重载通过每秒扫描 **app** 目录中的文件并查找更改来工作。如果发现任何更改，它将向浏览器发送消息以重新加载页面。它不会扫描任何其他目录，因此如果你对 **app** 目录之外的文件进行更改，你需要手动刷新页面。

如果你需要监视 **app** 目录之外的文件，或者由于项目的大小而导致速度较慢，你可以在 **app/Config/Toolbar.php** 配置文件的 `$watchedDirectories` 和 `$watchedExtensions` 属性中指定要扫描的目录和文件扩展名。

8.2 命令行使用

CodeIgniter 4 也可以与命令行程序一起使用。

8.2.1 CLI 概览

CodeIgniter 4 提供了内置命令 **spark** 和有用的命令与库。你还可以创建 spark 命令, 并通过 CLI 运行控制器。

- *什么是 CLI?*
- *为什么通过命令行运行?*
- *Spark 命令*
- *CLI 库*

什么是 CLI?

命令行接口是一种通过文本方式与计算机交互的方法。有关更多信息, 请查看 维基百科文章。

为什么通过命令行运行?

有许多原因可以通过命令行运行 CodeIgniter, 但这些原因并不总是明显的。

- 在不需要使用 *wget* 或 *curl* 的情况下运行 cron 作业。
- 制作交互式 “任务” , 可以执行设置权限、清理缓存文件夹、运行备份等操作。
- 与其他语言的其他应用程序集成。例如, 随机的 C++ 脚本可以调用一个命令并在模型中运行代码!

Spark 命令

CodeIgniter 提供了官方命令 **spark** 和内置命令。

你可以运行 spark 并查看帮助:

```
php spark
```

有关详细信息, 请参阅*Spark 命令* 页面。

CLI 库

CLI 库使使用 CLI 接口变得简单。它提供了将文本输出到终端窗口的多种颜色的简单方法。它还允许提示用户提供信息, 从而轻松构建灵活、智能的工具。

有关详细信息, 请参阅*CLI 库* 页面。

8.2.2 通过 CLI 运行控制器

除了通过浏览器的 URL 调用应用程序的控制器 外, 它们也可以通过命令行接口 (CLI) 加载。

备注: 建议使用 Spark 命令来编写 CLI 脚本, 而不是通过 CLI 调用控制器。有关详细信息, 请参阅*Spark 命令* 和创建 *Spark 命令* 页面。

- 让我们试一试:*Hello World!*
  - 创建控制器
  - 定义路由
  - 通过 *CLI* 运行
- 这就是基础知识!

让我们试一试:Hello World!

创建控制器

让我们创建一个简单的控制器, 这样你就可以看到它的实际效果。使用文本编辑器, 创建一个名为 Tools.php 的文件, 并添加以下代码:

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class Tools extends Controller
{
    public function message($to = 'World')
    {
        return "Hello {$to}!" . PHP_EOL;
    }
}
```

备注: 如果使用自动路由（改进版）, 请将方法名更改为 `cliMessage()`。

然后将该文件保存到 **app/Controllers/** 目录中。

定义路由

如果使用自动路由, 请跳过此步骤。

在 **app/Config/Routes.php** 文件中, 你可以轻松创建只能通过 CLI 访问的路由, 就像创建任何其他路由一样。与使用 `get()`、`post()` 或类似的方法不同, 你将使用 `cli()` 方法。其他所有内容的工作原理与正常的路由定义完全相同:

```
<?php

$routes->cli('tools/message/(:segment)', 'Tools::message/$1');
```

有关更多信息, 请参阅*Routes* 页面。

警告: 如果启用自动路由（传统版） 并将命令文件放在 **app/Controllers** 中, 任何人都可以在自动路由（传统版） 的帮助下通过 HTTP 访问该命令。

通过 CLI 运行

通常, 你会使用类似于以下内容的 URL 访问站点:

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

相反, 我们将在 Mac/Linux 上打开终端, 或者在 Windows 上转到运行窗口 > "cmd", 并导航到 CodeIgniter 项目的 web 根目录。

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

如果你操作正确, 应该会看到打印出 "Hello World!"。

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

这里我们以参数的方式传递内容, 就像 URL 参数的工作方式一样。"John Smith" 被作为参数传递, 输出是:

> Hello John Smith!

这就是基础知识!

简而言之, 这就是有关命令行上的控制器需要了解的全部内容。请记住, 这是一个正常的控制器, 因此路由和 `_remap()` 正常工作。

备注: `_remap()` 在自动路由（改进版） 中不起作用。

如果要确保通过 CLI 运行, 请检查*`is_cli()`* 的返回值。

但是,CodeIgniter 提供了其他工具, 可以使创建 CLI 可访问的脚本更加愉快, 包括 CLI 專用路由和一个可以帮助你使用 CLI 專用工具的库。

8.2.3 Spark 命令

CodeIgniter 提供了官方命令 **spark** 和内置命令。

- 运行命令
  - 通过 *CLI* 运行
    - 显示命令列表
    - 显示帮助
    - 运行命令
    - 抑制头部输出
  - 调用命令

运行命令

通过 CLI 运行

命令是从命令行中在项目根目录下运行的。提供了一个名为 **spark** 的命令文件，用于运行任何 CLI 命令。

显示命令列表

当不指定命令调用 **spark** 时，会显示一个简单的帮助页面，其中还提供了按类别排序的可用命令列表及其描述：

```
php spark
```

spark list

`php spark` 与 `list` 命令完全相同：

```
php spark list
```

你还可以使用 `--simple` 选项获取按字母顺序排序的所有可用命令的原始列表：

```
php spark list --simple
```

显示帮助

你可以使用 `help` 命令获取有关任何 CLI 命令的帮助，如下所示：

```
php spark help db:seed
```

自 v4.3.0 起，你还可以使用 `--help` 选项代替 `help` 命令：

```
php spark db:seed --help
```

运行命令

你应该将命令的名称作为第一个参数传递以运行该命令：

```
php spark migrate
```

某些命令接受附加参数，这些参数应该直接在命令之后用空格分隔提供：

```
php spark db:seed DevUserSeeder
```

对于 CodeIgniter 提供的所有命令，如果你没有提供所需的参数，系统将提示你提供运行所需的信息：

```
php spark make:controller

Controller 类名：
```

抑制头部输出

运行命令时，会输出包含 CodeIgniter 版本和当前时间的头部信息：

```
php spark env

CodeIgniter v4.3.5 Command Line Tool - Server Time: 2023-06-16␣
```

(续下页)

(接上页)

```
↪12:45:31 UTC+00:00

Your environment is currently set as development.
```

你可以始终传递 `--no-header` 以抑制头部输出，这对于解析结果很有帮助：

```
php spark env --no-header

Your environment is currently set as development.
```

调用命令

命令也可以从你自己的代码中运行。这通常在控制器中用于 cron 任务，但可以随时使用。你可以使用 `command()` 函数来实现。该函数始终可用。

```
<?php

echo command('make:migration TestMigration');
```

唯一的参数是字符串，即所调用的命令和任何参数。它的使用方式与从命令行调用完全相同。

当不从命令行运行时，所有运行的命令的输出都会被捕获。它会从命令中返回，以便你可以选择是否显示它。

8.2.4 创建 Spark 命令

虽然能够像其它路由一样通过 CLI 使用控制器很方便，但你可能会发现有时需要一些不同的东西。这就是 Spark 命令的用武之地。它们是简单的类，不需要为它们定义路由，使其成为构建可以帮助开发人员简化工作的工具的完美选择，无论是通过处理迁移或数据库填充，检查任务状态，甚至为你的公司构建定制代码生成器。

- 创建新命令
  - 文件位置
  - 一个示例命令

- *BaseCommand*

创建新命令

你可以非常轻松地创建新的命令供自己开发使用。每个类必须在其自己的文件中, 并且必须扩展 `CodeIgniter\CLI\BaseCommand`, 并实现 `run()` 方法。

应使用以下属性将命令列入 CLI 命令并添加帮助功能:

- `$group`: 描述命令分组的字符串。例如: `数据库`
- `$name`: 描述命令名称的字符串。例如:`make:controller`
- `$description`: 描述命令的字符串。例如: `生成一个新的控制器文件。`
- `$usage`: 描述命令用法的字符串。例如:`make:controller <name> [options]`
- `$arguments`: 描述每个命令参数的字符串数组。例如:`'name' => '控制器类名。'`
- `$options`: 描述每个命令选项的字符串数组。例如:`'--force' => '强制覆盖现有文件。'`

帮助描述将根据上述参数自动生成。

文件位置

命令必须存储在名为 **Commands** 的目录中。但是, 该目录必须位于 PSR-4 命名空间中, 以便自动加载程序 可以定位它。这可能在 **app/Commands** 中, 或者是一个用于所有项目开发的命令目录, 像 **Acme/Commands**。

备注: 当执行命令时, 会加载完整的 CodeIgniter CLI 环境, 使你可以获取环境信息、路径信息, 并使用控制器中会使用的任何工具。

一个示例命令

让我们逐步创建一个示例命令, 其唯一的功能是报告有关应用程序本身的一些基本信息, 以演示用途。首先在 **app/Commands/AppInfo.php** 中创建一个新文件。它应该包含以下代码:

```
<?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 = 'Displays basic application␣
↪information.';

    public function run(array $params)
    {
        // ...
    }
}
```

如果运行 **list** 命令, 你将在自己的 `Demo` 组下看到新命令被列出。如果仔细看, 应该可以相当容易地理解它的工作方式。`$group` 属性简单地告诉它如何组织此命令与所有其他存在的命令, 告诉它在哪个标题下列出它。

`$name` 属性是可以调用此命令的名称。唯一的要求是它不得包含空格, 并且所有字符在命令行本身必须有效。不过, 按照惯例, 命令应该是小写的, 并且通过在命令名称本身使用冒号进一步对命令进行分组, 以帮助防止多个命令发生命名冲突。

最后一个属性 `$description` 是一个简短的字符串, 在 **list** 命令中显示, 并应描述命令的作用。

run()

`run()` 方法是在运行命令时调用的方法。`$params` 数组是命令名称后面的任何 CLI 参数列表, 供你使用。如果 CLI 字符串是:

```
php spark foo bar baz
```

那么 **foo** 是命令名称, `$params` 数组将是:

```
<?php

$params = ['bar', 'baz'];
```

这也可以通过*CLI* 库访问, 但这里已经从字符串中删除了你的命令。这些参数可以用于自定义脚本的行为方式。

我们的演示命令可能有一个 `run()` 方法, 如下所示:

```
<?php

namespace App\Commands;

use CodeIgniter\CLI\BaseCommand;
use CodeIgniter\CLI\CLI;

class AppInfo extends BaseCommand
{
    // ...

    public function run(array $params)
    {
        CLI::write('PHP Version: ' . CLI::color(PHP_VERSION, 'yellow
↪'));
        CLI::write('CI Version: ' . CLI::color(\CodeIgniter\
↪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_
```

(续下页)

(接上页)

```
↪included_files()), 'yellow'));
    }
}
```

请参阅*CLI 库* 页面了解详细信息。

命令终止

默认情况下, 命令以成功代码 0 退出。如果在执行命令时遇到错误, 你可以通过在 `run()` 方法中使用 `return` 语句和退出代码来终止命令。

例如, `return EXIT_ERROR;`

这种方法可以帮助系统级调试, 如果命令例如通过 crontab 运行。

你可以使用 **app/Config/Constants.php** 文件中定义的 `EXIT_*` 退出代码常量。

BaseCommand

所有命令必须扩展的 `BaseCommand` 类有一些你应该熟悉的有用实用方法, 当创建自己的命令时。它还具有可以通过 `$this->logger` 访问的日志。

class `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`) –要传递给该命令的其他 CLI 参数。

此方法允许你在当前命令执行期间运行其他命令:

```
<?php

$this->call('command_one');
$this->call('command_two', $params);
```

showError(*Throwable $e*)

参数

- **$e** (`Throwable`) –用于报告错误的异常。

一种保持 CLI 错误输出一致且清晰的便捷方法:

```
<?php

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)

setPad(*string $item*, *int $max*, *int $extra = 2*, *int $indent = 0*) → string

参数

- **$item** (string) –字符串项目。
- **$max** (integer) –最大长度。
- **$extra** (integer) –在末尾添加的额外空格数。
- **$indent** (integer) –缩进空格数。

填充我们的字符串, 以便所有标题的长度相同, 以美观地排列描述:

```
use CodeIgniter\CLI\CLI;

$length = max(array_map('strlen', array_keys($this->
↪options)));

foreach ($this->options as $option => $description)
↪{
    CLI::write(CLI::color($this->setPad($option,
↪$length, 2, 2), 'green') . $description);
}
/*
 * Output will be:
 *  -n      Set migration namespace
 *  -g      Set database group
 *  --all   Set for all namespaces, will ignore (-n)␣
↪option
 */
```

getPad(*$array*, *$pad*)

自 4.0.5 版本弃用: 请使用*CodeIgniter\CLI*

`BaseCommand::setPad()`。

参数

- **$array** (array) – `$key => $value` 数组。
- **$pad** (integer) – 填充的空格数。

计算用于 `$key => $value` 数组输出的填充的方法。该填充可用于在 CLI 中输出格式良好的表格。

8.2.5 CLI 生成器

CodeIgniter4 现在配备了生成器, 以简化常规控制器、模型、实体等的创建。你还可以使用一个命令搭建一整套完整的文件。

- 简介
- 内置生成器
  - *make:cell*
  - *make:command*
  - *make:config*
  - *make:controller*
  - *make:entity*
  - *make:filter*
  - *make:model*
  - *make:seeder*
  - *make:test*
  - *make:migration*
  - *make:validation*
- 搭建一整套完整的代码
- *GeneratorTrait*
- 声明自定义生成器命令模板的位置

简介

所有内置生成器在使用 `php spark list` 列出时都位于 `Generators` 组下。要查看特定生成器的完整描述和使用信息, 请使用命令:

```
php spark help <生成器命令>
```

其中 `<生成器命令>` 将替换为要检查的命令。

备注: 你需要在子文件夹中生成代码吗? 例如, 如果你想在主 `Controllers` 文件夹的 `Admin` 子文件夹中创建一个控制器类, 你只需要在类名前加上子文件夹, 像这样:`php spark make:controller admin/login`。这个命令将在 `Controllers/Admin` 子文件夹中创建 `Login` 控制器, 命名空间为 `App\Controllers\Admin`。

备注: 在模块上工作? 代码生成将根命名空间默认设置为 `APP_NAMESPACE`。如果需要在模块命名空间的其他位置生成代码, 请确保在命令中设置 `--namespace` 选项, 例如 `php spark make:model blog --namespace Acme\\Blog`。

警告: 设置 `--namespace` 选项时, 请确保提供的命名空间是在 `Config\Autoload` 中的 `$psr4` 数组或你的 composer 自动加载文件中定义的有效命名空间。否则, 代码生成将中断。

内置生成器

CodeIgniter4 默认附带以下生成器。

make:cell

在 4.3.0 版本加入.

创建一个新的 Cell 文件及其视图。

用法:

```
make:cell <名称> [选项]
```

参数:

- 名称: 单元类的名称。应为 PascalCase。**[必需]**

选项:

- `--namespace`: 设置根命名空间。默认为 `APP_NAMESPACE` 的值。
- `--force`: 设置此标志以覆盖目标上的现有文件。

make:command

创建一个新的 spark 命令。

用法:

```
make:command <名称> [选项]
```

参数:

- 名称: 命令类的名称。**[必需]**

选项:

- `--command`: 在 spark 中运行的命令名称。默认为 `command:name`。
- `--group`: 命令的组/命名空间。对于基本命令默认为 `App`, 对于生成器命令默认为 `Generators`。
- `--type`: 命令类型, 可以是 `basic` 基本命令或 `generator` 生成器命令。默认为 `basic`。

- --namespace: 设置根命名空间。默认为 APP_NAMESPACE 的值。
- --suffix: 在生成的类名后附加组件后缀。
- --force: 设置此标志以覆盖目标上的现有文件。

make:config

创建一个新的配置文件。

用法:

```
make:config <名称> [选项]
```

参数:

- 名称: 配置类的名称。**[必需]**

选项:

- --namespace: 设置根命名空间。默认为 APP_NAMESPACE 的值。
- --suffix: 在生成的类名后附加组件后缀。
- --force: 设置此标志以覆盖目标上的现有文件。

make:controller

创建一个新的控制器文件。

用法:

```
make:controller <名称> [选项]
```

参数:

- 名称: 控制器类的名称。**[必需]**

选项:

- `--bare`: 扩展自 `CodeIgniter\Controller` 而不是 `BaseController`。
- `--restful`: 扩展自一个 RESTful 资源。可选 `controller` 和 `presenter`。默认为 `controller`。
- `--namespace`: 设置根命名空间。默认为 `APP_NAMESPACE` 的值。
- `--suffix`: 在生成的类名后附加组件后缀。
- `--force`: 设置此标志以覆盖目标上的现有文件。

备注: 如果使用 `--suffix`, 生成的控制器名称将类似于 `ProductController`。这与使用自动路由 的控制器命名约定相违背 (控制器类名必须以大写字母开头, 且只能大写第一个字符)。所以 `--suffix` 可与定义路由 一起使用。

---

make:entity

创建一个新的实体文件。

用法:

```
make:entity <名称> [选项]
```

参数:

- 名称: 实体类的名称。**[必需]**

选项:

- `--namespace`: 设置根命名空间。默认为 `APP_NAMESPACE` 的值。
- `--suffix`: 在生成的类名后附加组件后缀。
- `--force`: 设置此标志以覆盖目标上的现有文件。

make:filter

创建一个新的过滤器文件。

用法:

```
make:filter <名称> [选项]
```

参数:

- 名称: 过滤器类的名称。**[必需]**

选项:

- `--namespace`: 设置根命名空间。默认为 `APP_NAMESPACE` 的值。
- `--suffix`: 在生成的类名后附加组件后缀。
- `--force`: 设置此标志以覆盖目标上的现有文件。

make:model

创建一个新的模型文件。

用法:

```
make:model <名称> [选项]
```

参数:

- 名称: 模型类的名称。**[必需]**

选项:

- `--dbgroup`: 要使用的数据库组。默认为 `default`。
- `--return`: 设置返回类型, 可以是 `array`、`object` 或 `entity`。默认为 `array`。
- `--table`: 提供不同的表名。默认为将类名复数化。
- `--namespace`: 设置根命名空间。默认为 `APP_NAMESPACE` 的值。
- `--suffix`: 在生成的类名后附加组件后缀。
- `--force`: 设置此标志以覆盖目标上的现有文件。

make:seeder

创建一个新的种子文件。

用法:

```
make:seeder <名称> [选项]
```

参数:

- 名称: 种子类的名称。**[必需]**

选项:

- `--namespace`: 设置根命名空间。默认为 `APP_NAMESPACE` 的值。
- `--suffix`: 在生成的类名后附加组件后缀。
- `--force`: 设置此标志以覆盖目标上的现有文件。

make:test

在 4.5.0 版本加入.

创建一个新的测试文件。

用法:

```
make:test <name> [options]
```

参数:

- `name`: 测试类的名称。**[必需]**

选项:

- `--namespace`: 设置根命名空间。默认为 `Tests` 的值。
- `--force`: 设置此标志以覆盖目标上的现有文件。

make:migration

创建一个新的迁移文件。

用法:

```
make:migration <名称> [选项]
```

参数:

- 名称: 迁移类的名称。**[必需]**

选项:

- `--session`: 为数据库会话生成迁移文件。
- `--table`: 设置数据库会话的表名。默认为 `ci_sessions`。
- `--dbgroup`: 设置数据库会话的数据库组。默认为 `default` 组。
- `--namespace`: 设置根命名空间。默认为 `APP_NAMESPACE` 的值。
- `--suffix`: 在生成的类名后附加组件后缀。
- `--force`: 设置此标志以覆盖目标上的现有文件。

make:validation

创建一个新的验证文件。

用法:

```
make:validation <名称> [选项]
```

参数:

- 名称: 验证类的名称。**[必需]**

选项:

- `--namespace`: 设置根命名空间。默认为 `APP_NAMESPACE` 的值。
- `--suffix`: 在生成的类名后附加组件后缀。
- `--force`: 设置此标志以覆盖目标上的现有文件。

搭建一整套完整的代码

在开发阶段, 我们有时会分组创建功能, 比如创建一个 *Admin* 组。该组将包含自己的控制器、模型、迁移文件, 甚至实体。你可能会想一一在终端中输入每个生成器命令, 并希望有一个单一的生成器命令可以统治一切。

不要担心!CodeIgniter4 还配备了专用的 `make:scaffold` 命令, 它基本上是控制器、模型、实体、迁移和种子生成器命令的包装器。你只需要输入用于命名所有生成类的类名。另外, **每个生成器命令支持的单独选项**也会被脚手架命令识别。

在终端中运行此命令:

```
php spark make:scaffold user
```

将创建以下文件:

(1) **app/Controllers/User.php**

(2) **app/Models/User.php**

(3) **app/Database/Migrations/< 某日期 >_User.php** 和

(4) **app/Database/Seeds/User.php**

要在生成的文件中包含 `Entity` 类, 只需在命令中包含 `--return entity` 选项即可将其传递给模型生成器。

GeneratorTrait

所有生成器命令必须使用 `GeneratorTrait` 以充分利用其在代码生成中使用的方法。

声明自定义生成器命令模板的位置

生成器模板的默认查找顺序是 (1) **app/Config/Generators.php** 文件中定义的模板, 如果未找到, 则是 (2) 在 `CodeIgniter\Commands\Generators\Views` 命名空间下找到的模板。

要为自定义生成器命令声明模板位置, 需要将其添加到 **app/Config/Generators.php** 文件中。例如, 如果你有一个命令 `make:awesome-command`, 并且生成器模板位于 *app* 目录 **app/Commands/Generators/Views/awesomecommand.tpl.php** 中, 则需要按如下方式更新配置文件:

```
<?php

namespace Config;

use CodeIgniter\Config\BaseConfig;

class Generators extends BaseConfig
{
    public array $views = [
        // ..
        'make:awesome-command' => 'App\Commands\Generators\Views\
↪awesomecommand.tpl.php',
    ];
}
```

8.2.6 CLI 库

CodeIgniter 的 CLI 库可以轻松创建交互式命令行脚本, 包括:

- 提示用户提供更多信息
- 在终端上写入多彩文本
- 蜂鸣声 (要友好!)
- 在长时间任务期间显示进度条
- 将过长的文本行包装以适应窗口

- 初始化类
- 获取用户输入
  - *prompt()*
  - *promptByKey()*
  - *promptByMultipleKeys()*
- 提供反馈
  - *write()*
  - *print()*
  - *color()*
  - *error()*
  - *wrap()*
  - *newLine()*
  - *clearScreen()*
  - *showProgress()*
  - *table()*
  - *wait()*

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的。相反, 你只需要通过控制器顶部的 `use` 语句确保控制器可以定位它:

```
<?php

namespace App\Commands;

use CodeIgniter\CLI\BaseCommand;
use CodeIgniter\CLI\CLI;

class MyCommand extends BaseCommand
{
```

(续下页)

(接上页)

```
    // ...

    public function run(array $params)
    {
        // ...
    }
}
```

该类在首次加载文件时会自动初始化。

获取用户输入

有时你需要询问用户更多信息。他们可能没有提供可选的命令行参数, 或者脚本可能遇到现有文件并需要确认才能覆盖。这通过 `prompt()` 或 `promptByKey()` 方法来处理。

备注: 从 v4.3.0 开始, 你可以用 `PhpStreamWrapper` 为这些方法编写测试。请参阅*测试 CLI 输入*。

---

prompt()

你可以通过作为第一个参数传递问题来提供一个问题:

```
<?php

use CodeIgniter\CLI\CLI;

$color = CLI::prompt('What is your favorite color?');
```

你可以通过在第二个参数中传递默认值, 为用户只需按 Enter 提供默认答案:

```
<?php

use CodeIgniter\CLI\CLI;

$color = CLI::prompt('What is your favorite color?', 'blue');
```

你可以通过作为第二个参数传递允许答案的数组来限制可接受的答案:

```
<?php

use CodeIgniter\CLI\CLI;

$overwrite = CLI::prompt('File exists. Overwrite?', ['y', 'n']);
```

最后, 你可以将答案输入的验证 规则作为第三个参数传递:

```
<?php

use CodeIgniter\CLI\CLI;

$email = CLI::prompt('What is your email?', null, 'required|valid_
↪email');
```

验证规则也可以以数组语法编写:

```
<?php

use CodeIgniter\CLI\CLI;

$email = CLI::prompt('What is your email?', null, ['required',
↪'valid_email']);
```

promptByKey()

预定义的提示答案 (选项) 有时需要描述或过于复杂, 无法通过其值进行选择。promptByKey() 允许用户通过其键而不是值来选择选项:

```
<?php

use CodeIgniter\CLI\CLI;

$fruit = CLI::promptByKey('These are your choices:', ['The red apple
↪', 'The plump orange', 'The ripe banana']);
/*
 * These are your choices:
```

(续下页)

(接上页)

```
 *   [0]  The red apple
 *   [1]  The plump orange
 *   [2]  The ripe banana
 *
 * [0, 1, 2]:
 */
```

命名键也是可能的:

```
<?php

use CodeIgniter\CLI\CLI;

$fruit = CLI::promptByKey(['These are your choices:', 'Which would␣
↪you like?'], [
    'apple'  => 'The red apple',
    'orange' => 'The plump orange',
    'banana' => 'The ripe banana',
]);
/*
 * These are your choices:
 *   [apple]   The red apple
 *   [orange]  The plump orange
 *   [banana]  The ripe banana
 *
 * Which would you like? [apple, orange, banana]:
 */
```

最后, 你可以将答案输入的验证 规则作为第三个参数传递, 可接受的答案会自动限制为传入的选项。

promptByMultipleKeys()

在 4.3.0 版本加入.

这个方法与 `promptByKey()` 相同, 但它支持多个值。

```
<?php

use CodeIgniter\CLI\CLI;

$hobbies = CLI::promptByMultipleKeys('Select your hobbies:', [
↪'Playing game', 'Sleep', 'Badminton']);
/*
 * Select your hobbies:
 *   [0]  Playing game
 *   [1]  Sleep
 *   [2]  Badminton
 *
 * You can specify multiple values separated by commas.
 * [0, 1, 2]:
 *
 * if your answer is '0,2', the return is the key and the value of␣
↪the options :
 * [
 *   [0] => "Playing game",
 *   [2] => "Badminton"
 * ]
 */
```

重要: 与 `promptByKey()` 不同, `promptByMultipleKeys()` 方法不支持命名键或验证。

提供反馈

write()

提供了几种方法来向用户提供反馈。这可以是简单的单个状态更新, 也可以是复杂的信息表格, 会换行到用户的终端窗口。这其核心是 `write()` 方法, 它以要输出的字符串作为第一个参数:

```
<?php
```

(续下页)

(接上页)

```
use CodeIgniter\CLI\CLI;

CLI::write('The rain in Spain falls mainly on the plains.');
```

你可以通过在第二个参数中传递颜色名称来更改文本颜色:

```
<?php

use CodeIgniter\CLI\CLI;

CLI::write('File created.', 'green');
```

这可以用来按状态区分消息, 或通过使用不同的颜色创建 “标题”。你甚至可以通过将颜色名称作为第三个参数传递来设置背景颜色:

```
<?php

use CodeIgniter\CLI\CLI;

CLI::write('File overwritten.', 'light_red', 'dark_gray');
```

以下前景色可用:

- 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

并且有更小数量的背景色可用:

- black
- blue
- green
- cyan
- red
- yellow
- light_gray
- magenta

print()

`print()` 的作用与 `write()` 方法相同, 只是它不会在前后强制换行。相反, 它会将内容打印到光标当前所在的屏幕上。这允许你从不同的调用中在同一行上打印多个项目。当你想显示一个状态, 执行一些操作, 然后在同一行上打印 "Done" 时, 这特别有用:

```
<?php

use CodeIgniter\CLI\CLI;

for ($i = 0; $i <= 10; $i++) {
    CLI::print($i);
}
```

color()

虽然 `write()` 命令会将单行写入终端, 并在结束时带有 EOL 字符, 但你可以使用 `color()` 方法来制作一个字符串片段, 可以以相同的方式使用, 不同之处在于打印后不会强制 EOL。这允许你在同一行上创建多个输出。或者, 更常见的是, 你可以在 `write()` 方法中使用它来创建不同颜色的字符串:

```
<?php

use CodeIgniter\CLI\CLI;

CLI::write("fileA \t" . CLI::color('/path/to/file', 'white'),
↪'yellow');
```

这个示例将在窗口中写入一行, `fileA` 为黄色, 后跟一个制表符, 然后是白色的 `/path/to/file`。

error()

如果需要输出错误, 你应该使用适当命名的 `error()` 方法。这会将浅红色文本写入 STDERR, 而不是像 `write()` 和 `color()` 那样写入 STDOUT。如果你有监视错误的脚本, 这样可以方便它们不必筛选所有信息, 而只提取实际的错误消息。你可以像使用 `write()` 方法一样使用它:

```
<?php

use CodeIgniter\CLI\CLI;

CLI::error('Cannot write to file: ' . $file);
```

wrap()

这个命令将获取一个字符串, 开始在当前行打印它, 并将其换行到设置的长度。当显示你想要在当前窗口中换行而不是超出屏幕的选项及其描述时, 这可能很有用:

```
<?php
```

(续下页)

(接上页)

```
use CodeIgniter\CLI\CLI;

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, 字符串将换行到终端宽度。Windows 当前没有提供确定窗口大小的方法, 因此我们默认为 80 个字符。如果你想将宽度限制为一些可以相当确定适合窗口的较短长度, 请将最大行长度作为第二个参数传递。这将在最接近的词边界处中断字符串, 以免单词被断开。

```
<?php

use CodeIgniter\CLI\CLI;

// Wrap the text at max 20 characters wide
CLI::wrap($description, 20);
```

你可能会发现, 你想要标题、文件或任务的左边有一列, 而右边有描述文本的一列。默认情况下, 这将回绕到窗口的左边缘, 这不允许项目按列对齐。在这种情况下, 你可以传入换行后要填充的空格数, 以便在左边有一个整齐的列边界:

```
<?php

use CodeIgniter\CLI\CLI;

$titles = [
    'task1a',
    'task1abc',
];
$descriptions = [
    'Lorem Ipsum is simply dummy text of the printing and␣
↪typesetting industry.',
    "Lorem Ipsum has been the industry's standard dummy text ever␣
↪since the",
];

// Determine the maximum length of all titles
```

(续下页)

(接上页)

```
// to determine the width of the left column
$maxlen = max(array_map('strlen', $titles));

for ($i = 0; $i < count($titles); $i++) {
    CLI::write(
        // Display the title on the left of the row
        substr(
            $titles[$i] . str_repeat(' ', $maxlen + 3),
            0,
            $maxlen + 3,
        ) .
        // Wrap the descriptions in a right-hand column
        // with its left side 3 characters wider than
        // the longest item on the left.
        CLI::wrap($descriptions[$i], 40, $maxlen + 3),
    );
}
```

这将创建类似如下的内容:

```
task1a      Lorem Ipsum 只是印刷和排版
            行业的虚构文字
task1abc    Lorem Ipsum 从1500年代起
            就一直是行业的标准虚构文字
```

newLine()

`newLine()` 方法向用户显示一个空行。它不接受任何参数:

```
<?php

use CodeIgniter\CLI\CLI;

CLI::newLine();
```

clearScreen()

你可以使用 `clearScreen()` 方法清除当前的终端窗口。在大多数 Windows 版本中, 这只会插入 40 行空白行, 因为 Windows 不支持此功能。Windows 10 bash 集成应该会改变这一点:

```
<?php

use CodeIgniter\CLI\CLI;

CLI::clearScreen();
```

showProgress()

如果你有一个长时间运行的任务, 希望保持用户了解进度, 可以使用 `showProgress()` 方法, 它会显示类似以下内容:

```
[####......] 40% Complete
```

此块会就地进行动画以产生非常好的效果。

使用时, 请将当前步骤作为第一个参数传递, 将总步骤数作为第二个参数。完成百分比和显示长度将根据该数字确定。完成时, 请将 `false` 作为第一个参数传入, 进度条将被删除。

```
<?php

use CodeIgniter\CLI\CLI;

$totalSteps = count($tasks);
$currStep   = 1;

foreach ($tasks as $task) {
    CLI::showProgress($currStep++, $totalSteps);
    $task->run();
}

// Done, so erase it...
CLI::showProgress(false);
```

table()

```
<?php

use CodeIgniter\CLI\CLI;

$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       | 2017-11-15 10:35:02 | 1      |
| 8  | Another great item title | 2017-11-16 13:46:54 | 0      |
+----+--------------------------+---------------------+--------+
```

wait()

等待一定的秒数, 可选择显示等待消息并等待按键。

```
<?php

use CodeIgniter\CLI\CLI;

// wait for specified interval, with countdown displayed
CLI::wait($seconds, true);

// show continuation message and wait for input
CLI::wait(0, false);

// wait for specified interval
CLI::wait($seconds, false);
```

8.2.7 CLIRequest 类

如果请求来自命令行调用, 则请求对象实际上是一个 `CLIRequest`。它的行为与常规请求 相同, 但添加了一些方便的访问器方法。

额外的访问器

getSegments()

返回被视为路径一部分的命令行参数数组:

```
<?php

// command line: php index.php users 21 profile --foo bar
echo $request->getSegments();  // ['users', '21', 'profile']
```

getPath()

返回重构后的路径字符串:

```
<?php

// command line: php index.php users 21 profile --foo bar
echo $request->getPath();  // users/21/profile
```

getOptions()

返回被视为选项的命令行参数数组:

```
<?php

// command line: php index.php users 21 profile --foo bar
echo $request->getOptions();  // ['foo' => 'bar']
```

getOption($key)

返回被视为选项的特定命令行参数的值:

```
<?php

// command line: php index.php users 21 profile --foo bar
echo $request->getOption('foo');       // bar
echo $request->getOption('notthere'); // null
```

getOptionString()

返回重构后的命令行选项字符串:

```
<?php

// command line: php index.php users 21 profile --foo bar
echo $request->getOptionString();  // -foo bar
```

向第一个参数传递 `true` 将尝试使用两个破折号编写长选项:

```
<?php

// php index.php user 21 --foo bar -f
echo $request->getOptionString();      // -foo bar -f
echo $request->getOptionString(true); // --foo bar -f
```

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, 都会自动初始化几个基本类作为核心框架的一部分。但是, 可以用你自己的版本替换任何核心系统类, 或者只是扩展核心版本。

大多数用户都不需要这样做, 但对于那些想要显着改变 CodeIgniter 核心的人来说, 替换或扩展它们的选项确实存在。

重要: 与核心系统类打交道有很多影响, 所以在尝试之前, 请确保你知道你在做什么。

- 系统类列表
- 替换核心类
  - 创建你的类
  - 添加服务
- 扩展核心类

系统类列表

以下是每次 CodeIgniter 运行时都会调用的核心系统类列表:

- `CodeIgniter\Autoloader\Autoloader`
- `CodeIgniter\Autoloader\FileLocator`
- `CodeIgniter\Cache\CacheFactory`
- `CodeIgniter\Cache\Handlers\BaseHandler`
- `CodeIgniter\Cache\Handlers\FileHandler`
- `CodeIgniter\Cache\ResponseCache`
- `CodeIgniter\CodeIgniter`

- CodeIgniter\Config\BaseService
- CodeIgniter\Config\DotEnv
- CodeIgniter\Config\Factories
- CodeIgniter\Config\Services
- CodeIgniter\Controller
- CodeIgniter\Cookie\Cookie
- CodeIgniter\Cookie\CookieStore
- CodeIgniter\Debug\Exceptions
- CodeIgniter\Debug\Timer
- CodeIgniter\Events\Events
- CodeIgniter\Filters\Filters
- CodeIgniter\HTTP\CLIRequest (如果仅从命令行启动)
- CodeIgniter\HTTP\ContentSecurityPolicy
- CodeIgniter\HTTP\Header
- CodeIgniter\HTTP\IncomingRequest (如果通过 HTTP 启动)
- CodeIgniter\HTTP\Message
- CodeIgniter\HTTP\OutgoingRequest
- CodeIgniter\HTTP\Request
- CodeIgniter\HTTP\Response
- CodeIgniter\HTTP\SiteURI
- CodeIgniter\HTTP\SiteURIFactory
- CodeIgniter\HTTP\URI
- CodeIgniter\HTTP\UserAgent (如果通过 HTTP 启动)
- CodeIgniter\Log\Logger
- CodeIgniter\Log\Handlers\BaseHandler
- CodeIgniter\Log\Handlers\FileHandler
- CodeIgniter\Router\RouteCollection

- CodeIgniter\Router\Router
- CodeIgniter\Superglobals
- CodeIgniter\View\View

替换核心类

要使用自己的系统类代替默认类, 请确保:

1. 自动加载器 可以找到你的类,
2. 你的新类实现了适当的接口,
3. 并修改适当的服务 来加载你的类以替换核心类。

创建你的类

例如, 如果你有一个新的 App\Libraries\RouteCollection 类, 想用它代替核心系统类, 你会这样创建你的类:

```
<?php

namespace App\Libraries;

use CodeIgniter\Router\RouteCollectionInterface;

class RouteCollection implements RouteCollectionInterface
{
    // ...
}
```

添加服务

然后你需要在 **app/Config/Services.php** 中添加 routes 服务来加载你的类:

```
<?php

namespace Config;
```

(续下页)

(接上页)

```
use CodeIgniter\Config\BaseService;

class Services extends BaseService
{
    public static function routes(bool $getShared = true)
    {
        if ($getShared) {
            return static::getSharedInstance('routes');
        }

        return new \App\Libraries\RouteCollection(static::locator(),
↪ config(Modules::class), config(Routing::class));
    }

    // ...
}
```

扩展核心类

如果你只需要向现有的库中添加一些功能 - 可能是一两个方法 - 那么重新创建整个库就有些过度了。在这种情况下，更好的做法是简单地扩展这个类。扩展类与替换核心类 几乎相同，只有一个例外：

- 类声明必须扩展父类。

例如, 要扩展原生的 `RouteCollection` 类, 你需要用以下方式声明你的类:

```
<?php

namespace App\Libraries;

use CodeIgniter\Router\RouteCollection as BaseRouteCollection;

class RouteCollection extends BaseRouteCollection
{
    // ...
}
```

如果你的类中需要使用构造函数, 请确保扩展父类构造函数:

```
<?php

namespace App\Libraries;

use CodeIgniter\Router\RouteCollection as BaseRouteCollection;

class RouteCollection extends BaseRouteCollection
{
    public function __construct()
    {
        parent::__construct();

        // your code here
    }
}
```

提示: 你类中与父类方法同名的任何函数都会代替原生的方法 (这被称为 “方法重载”)。这允许你大幅改变 CodeIgniter 核心。

8.3.2 替换通用函数

CodeIgniter 中有很多函数需要提前加载, 以便在核心类中使用, 因此不能放入 helper 中。虽然大多数用户都不需要这样做, 但对于那些想要显着改变 CodeIgniter 核心的人来说, 替换这些函数的选项确实存在。在 **app/** 目录中有一个文件 **Common.php**, 其中定义的任何函数都会优先于 **system/Common.php** 中的版本。这也是一个机会, 可以创建在整个框架中打算使用的全局可用函数。

备注: 与核心系统类挂钩有很多含义, 所以在尝试之前, 请确保你知道你在做什么。

8.3.3 事件

CodeIgniter 的事件功能提供了一种无需修改核心文件即可介入和修改框架内部工作机制的方式。当 CodeIgniter 运行时，它遵循特定的执行流程。然而，有时你可能希望在执行流程的特定阶段触发某些操作。例如，你可能想在控制器加载之前或之后立即运行一个脚本，或者在某个其他位置触发你自己的脚本。

事件基于 * 发布/订阅 * 模式工作，在脚本执行期间的某个时刻触发一个事件。其他脚本可以通过向 Events 类注册来" 订阅" 该事件，以告知框架当该事件触发时它们希望执行某个操作。

- 启用事件
- 定义事件
  - 设置优先级
- 发布自定义事件
- 模拟事件
- 事件点
  - *Web* 应用
  - *CLI* 应用
  - 其他

启用事件

事件始终处于启用状态，并且全局可用。

定义事件

大多数事件定义在 **app/Config/Events.php** 文件中。你可以使用 `Events` 类的 `on()` 方法为事件订阅一个操作。第一个参数是要订阅的事件名称，第二个参数是当事件触发时要运行的可调用对象：

```
<?php

use CodeIgniter\Events\Events;

Events::on('pre_system', ['MyClass', 'myFunction']);
```

在这个例子中，每当 `pre_system` 事件被执行时，就会创建 `MyClass` 的实例并运行 `myFunction()` 方法。注意第二个参数可以是 PHP 识别的 * 任何 * 形式的 '可调用对象 <https://www.php.net/manual/en/function.is-callable.php>'_：

```
<?php

use CodeIgniter\Events\Events;

// Call a standalone function
Events::on('pre_system', 'some_function');

// Call on an instance method
$user = new \App\Libraries\User();
Events::on('pre_system', [$user, 'someMethod']);

// Call on a static method
Events::on('pre_system', 'SomeClass::someMethod');

// Use a Closure
Events::on('pre_system', static function (...$params) {
    // ...
});
```

设置优先级

由于多个方法可以订阅同一个事件，你需要一种方式来定义这些方法的调用顺序。你可以通过向 `on()` 方法的第三个参数传递优先级值来实现。数值越小优先级越高执行越早，1 为最高优先级，没有最低值限制：

```
<?php

use CodeIgniter\Events\Events;

Events::on('post_controller_constructor', 'some_function', 25);
```

具有相同优先级的订阅者将按照它们被定义的顺序执行。

自 v4.2.0 起，定义了三个类常量供你使用，这些常量设置了有用的数值范围。虽然不强制使用，但它们有助于提高可读性：

```
<?php
```

(续下页)

(接上页)

```
use CodeIgniter\Events\Events;

Events::PRIORITY_LOW;     // 200
Events::PRIORITY_NORMAL; // 100
Events::PRIORITY_HIGH;    // 10
```

重要: 常量 `EVENT_PRIORITY_LOW`、`EVENT_PRIORITY_NORMAL` 和 `EVENT_PRIORITY_HIGH` 已在 v4.6.0 中移除。

排序完成后，所有订阅者将按顺序执行。如果任何订阅者返回布尔值 false，则订阅者的执行将停止。

发布自定义事件

Events 类库也让你可以轻松在自己的代码中创建事件。要使用此功能，只需在 **Events** 类上调用 `trigger()` 方法并指定事件名称：

```
<?php

\CodeIgniter\Events\Events::trigger('some_event');
```

你可以通过添加额外参数将任意数量的参数传递给订阅者。订阅者将按照定义顺序接收这些参数：

```
<?php

use CodeIgniter\Events\Events;

Events::trigger('some_events', $foo, $bar, $baz);

Events::on('some_event', static function ($foo, $bar, $baz) {
    // ...
});
```

模拟事件

在测试期间，你可能不希望实际触发事件，因为每天发送数百封邮件既缓慢又适得其反。你可以使用 `simulate()` 方法让 Events 类仅模拟运行事件。当设置为 **true** 时，所有事件都将在触发方法中被跳过，其他操作仍会正常进行。

```
<?php

use CodeIgniter\Events\Events;

Events::simulate(true);
```

你可以通过传递 false 来停止模拟：

```
<?php

use CodeIgniter\Events\Events;

Events::simulate(false);
```

事件点

Web 应用

以下是 Web 应用程序中由 **public/index.php** 调用的可用事件点列表：

- **pre_system** 在系统执行的早期被调用。此时 URI、Request 和 Response 已实例化，但尚未进行页面缓存检查、路由和”before”控制器过滤器的执行。
- **post_controller_constructor** 在控制器实例化后立即调用，但在任何方法调用之前。
- **post_system** 在最终渲染页面发送到浏览器之前调用，在系统执行结束时，在”after”控制器过滤器执行之后。

CLI 应用

以下是针对*Spark 命令* 的可用事件点列表:

- **pre_command** 在命令代码执行之前立即调用。
- **post_command** 在命令代码执行之后立即调用。

其他

以下是各库可用的通用事件点列表:

- **email** 当 `CodeIgniter\Email\Email` 成功发送邮件后调用。接收 `Email` 类的属性数组作为参数。
- **DBQuery** 在数据库查询（无论成功与否）之后调用。接收 `Query` 对象。
- **migrate** 在成功调用 `latest()` 或 `regress()` 迁移方法后调用。接收 `MigrationRunner` 的当前属性以及方法名称。

8.3.4 扩展控制器

CodeIgniter 的核心控制器不应该改变, 但是在 **app/Controllers/BaseController.php** 提供了一个默认的类扩展。你创建的任何新控制器都应该扩展 `BaseController` 来利用预加载的组件和你提供的任何其他功能:

```
<?php

namespace App\Controllers;

class Home extends BaseController
{
    // ...
}
```

- 预加载组件
- 其他方法
- 其他选择

预加载组件

基础控制器是一个很好的地方, 可以加载你打算在项目每次运行时使用的任何辅助函数、模型、类库、服务等。辅助函数应该添加到预定义的 `$helpers` 数组中。例如, 如果你需要 HTML 和 Text 辅助函数在所有地方可用:

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

abstract class BaseController extends Controller
{
    // ...

    protected $helpers = ['html', 'text'];

    // ...
}
```

需要加载的任何其他组件或要处理的数据应该添加到构造函数 `initController()` 中。例如, 如果你的项目大量使用 Session 库, 你可以在这里初始化它:

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

abstract class BaseController extends Controller
{
    // ...

    /**
     * @var \CodeIgniter\Session\Session;
     */
    protected $session;
```

(续下页)

(接上页)

```
    public function initController(/* ... */)
    {
        // Do Not Edit This Line
        parent::initController($request, $response, $logger);

        $this->session = service('session');
    }
}
```

其他方法

基础控制器是不可路由的。作为一个额外的安全措施, **所有**你创建的新方法都应该声明为 `protected` 或 `private`, 并且只能通过扩展 `BaseController` 的控制器访问它们。

其他选择

你可能会发现需要多个基础控制器。只要其他控制器扩展正确的基础控制器, 就可以创建多个基础控制器。例如, 如果你的项目有复杂的公共接口和简单的管理门户, 可以考虑让公共控制器扩展 `BaseController`, 为任何管理控制器创建 `AdminController`。

如果你不想使用基础控制器, 可以通过让控制器扩展系统的 Controller 来绕过它:

```
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class Home extends Controller
{
    // ...
}
```

8.3.5 身份验证

CodeIgniter 为 CodeIgniter 4 提供了一个官方的身份验证和授权框架*CodeIgniter Shield*，它被设计为安全、灵活且易于扩展，以满足许多不同类型网站的需求。

为了在开发者之间保持一致，以下是一些推荐的准则。

推荐

- 处理登录和登出操作的模块在成功时应该触发 `login` 和 `logout` 事件
- 定义 "当前用户" 的模块应该定义 `user_id()` 函数以返回用户的唯一标识符，如果没有当前用户则返回 `null`

符合这些推荐的模块可以在 **composer.json** 中添加以下内容表示兼容:

```
"provide": {
    "codeigniter4/authentication-implementation": "1.0"
},
```

你可以在 Packagist 上查看提供该实现的模块列表。

8.3.6 创建 Composer 包

你可以将你创建的代码模块 转换为 Composer 包，或者为 CodeIgniter 4 创建一个 Composer 包。

- 文件夹结构
- 创建 *composer.json*
  - 包名称
  - 命名空间
  - 选择许可证
- 准备开发工具
  - 安装 *DevKit*
  - 配置 *Coding Standards Fixer*

- 配置文件
  - 允许用户覆盖设置
  - 在 *app/Config* 中覆盖设置
- 参考资料

文件夹结构

下面是一个典型的 Composer 包的目录结构示例:

```
your-package-name/
├── .gitattributes
├── .gitignore
├── LICENSE
├── README.md
├── composer.json
├── src/
│   └── YourClass.php
└── tests/
    └── YourClassTest.php
```

创建 composer.json

在你的包目录的根目录中，创建一个 **composer.json** 文件。该文件定义了关于你的包及其依赖项的元数据。

使用 `composer init` 命令可以帮助你创建它。

例如，**composer.json** 可能如下所示:

```
{
    "name": "your-vendor/your-package",
    "description": "Your package description",
    "type": "library",
    "license": "MIT",
    "autoload": {
        "psr-4": {
            "YourVendor\\YourPackage\\": "src/"
```

(续下页)

(接上页)

```
        }
    },
    "authors": [
        {
            "name": "Your Name",
            "email": "yourname@example.com"
        }
    ],
    "require": {
        // 在此处添加你的包所需的任何依赖项
    },
    "require-dev": {
        // 在此处添加开发所需的任何依赖项（例如 PHPUnit）
    }
}
```

包名称

name 字段在这里非常重要。包名称通常以 "vendor-name/package-name" 的格式书写，全部小写。以下是一个常见的示例：

- your-vendor-name：标识供应商（包的创建者）的名称，例如你的姓名或组织名称。
- your-package-name：你正在创建的包的名称。

因此，为了使名称唯一以区分其它包，使其与其他包区分开是非常重要的，尤其是在发布时。

命名空间

包名称决定了 autoload.psr4 中的供应商命名空间。

如果你的包名称是 your-vendor/your-package，那么供应商命名空间必须是 YourVendor。因此，你需要像下面这样编写:

```
"autoload": {
    "psr-4": {
```

(续下页)

(接上页)

```
        "YourVendor\\YourPackage\\": "src/"
    }
}
```

这个设置指示 Composer 自动加载你的包的源代码。

选择许可证

如果你对开源许可证不熟悉，请参考 https://choosealicense.com/。许多 PHP 包，包括 CodeIgniter，使用 MIT 许可证。

准备开发工具

有许多工具可以帮助确保代码质量。因此，你应该使用它们。你可以使用 CodeIgniter DevKit 轻松安装和配置此类工具。

安装 DevKit

在你的包目录的根目录中，运行以下命令：

```
composer config minimum-stability dev
composer config prefer-stable true
composer require --dev codeigniter4/devkit
```

DevKit 安装了各种 Composer 包，帮助你进行开发，并在 **vendor/codeigniter4/devkit/src/Template** 中为它们安装了模板。将其中的文件复制到你的项目根目录，并根据你的需求进行编辑。

配置 Coding Standards Fixer

DevKit 提供了基于 PHP-CS-Fixer 的 CodeIgniter Coding Standard 的 Coding Standards Fixer。

将 **vendor/codeigniter4/devkit/src/Template/.php-cs-fixer.dist.php** 复制到你的项目根目录。

为缓存文件创建 **build** 文件夹:

```
your-package-name/
├── .php-cs-fixer.dist.php
├── build/
```

打开你的编辑器中的 **.php-cs-fixer.dist.php** 文件，并修复文件夹路径:

```
--- a/.php-cs-fixer.dist.php
+++ b/.php-cs-fixer.dist.php
@@ -7,7 +7,7 @@ use PhpCsFixer\Finder;
 $finder = Finder::create()
     ->files()
     ->in([
-        __DIR__ . '/app/',
+        __DIR__ . '/src/',
         __DIR__ . '/tests/',
     ])
     ->exclude([
```

完成后，你可以运行 Coding Standards Fixer：

```
vendor/bin/php-cs-fixer fix --ansi --verbose --diff
```

如果你在 **composer.json** 中添加了 `scripts.cs-fix`，则可以使用 `composer cs-fix` 命令运行它:

```
{
    // ...
    },
    "scripts": {
        "cs-fix": "php-cs-fixer fix --ansi --verbose --diff"
    }
}
```

配置文件

允许用户覆盖设置

如果你的包有一个配置文件，并且你希望用户能够覆盖设置，可以使用 *config()* 函数与短类名（例如 `config('YourConfig')`）来调用配置文件。

然后，用户可以通过在 **app/Config** 中放置一个与短类名相同且扩展了包配置类的配置类（例如 `YourVendor\YourPackage\Config\YourConfig`）来覆盖包配置。

在 app/Config 中覆盖设置

如果你需要在 **app/Config** 文件夹中覆盖或添加已知配置，可以使用隐式注册器。

参考资料

我们已经发布了一些官方包。你可以在创建自己的包时使用这些包作为参考：

- https://github.com/codeigniter4/shield
- https://github.com/codeigniter4/settings
- https://github.com/codeigniter4/tasks
- https://github.com/codeigniter4/cache

8.3.7 为 CodeIgniter 做贡献

CodeIgniter 是一个社区驱动的项目，接受来自社区的代码和文档贡献。这些贡献是以 Issue 或 Pull Request 的形式在 GitHub 上的 CodeIgniter4 仓库 提出的。

如果你想贡献，请参阅我们代码仓库的 为 CodeIgniter4 做贡献 部分。

章节 9

官方包

9.1 官方包

CodeIgniter 框架无法解决开发者将遇到的所有问题。许多用户说他们喜欢框架小巧、快速的特点，所以我们不想让核心框架臃肿。为了弥合这一差距，我们正在发布官方包以提供不是每个网站都需要或想要的额外功能。

- *Shield*
- *Settings*
- *任务 (BETA)*
- *队列 (BETA)*
- *Cache*
- *DevKit*
- *编码标准*

9.1.1 Shield

CodeIgniter Shield 是 CodeIgniter 4 的身份验证和授权框架。它旨在安全、灵活且易于扩展以满足许多不同类型网站的需求。其中许多功能包括:

- 基于会话的身份验证
- 个人访问令牌认证
- 登录/注册后“动作”的框架 (如双因素认证等)
- 基于角色的访问控制, 具有简单、灵活的权限。
- 每个用户的权限覆盖
- 等等…

9.1.2 Settings

CodeIgniter Settings 是配置文件包装器, 允许任何配置设置保存到数据库中, 同时在没有自定义值存储时默认为配置文件。这允许应用程序与默认配置值一起发布, 但随着项目的发展或服务器的迁移而不必接触代码而适应。

9.1.3 任务 (BETA)

CodeIgniter 任务 是 CodeIgniter 4 的一个简单任务调度器。它允许你安排任务在特定时间运行，或者定期运行。它的设计目标是简单易用，但足够灵活以处理大多数使用场景。

9.1.4 队列 (BETA)

CodeIgniter 队列 是 CodeIgniter 4 的一个简单队列系统。它允许你将任务排队，以便稍后运行。

9.1.5 Cache

我们为 CodeIgniter 4 提供了带有 PSR-6 和 PSR-16 缓存适配器 的库。这不是必需的, 因为 CodeIgniter 4 自带完全功能的缓存组件。此模块仅用于集成依赖 PSR 接口规定的第三方包。

9.1.6 DevKit

CodeIgniter DevKit 提供了 CodeIgniter 用来帮助确保高质量代码的所有开发工具, 包括我们的编码标准、静态分析工具和规则、单元测试、数据生成、文件系统模拟、安全建议等等。这可以在你的任何个人项目或库中使用, 以快速设置 17 种不同的工具。

9.1.7 编码标准

CodeIgniter 编码标准 包含了基于 PHP CS Fixer 和 Nexus CS Config 的 CodeIgniter 的官方编码标准。这可以在你自己的项目中使用, 以形成一致的风格规则集合, 可以自动应用于你的代码。

PHP Namespace Index

C

CodeIgniter, 1017

CodeIgniter\Cache, 1091

CodeIgniter\CLI, 1556

CodeIgniter\Cookie, 1113

CodeIgniter\Database, 1077

CodeIgniter\Email, 1148

CodeIgniter\Encryption, 1165

CodeIgniter\HTTP, 789

CodeIgniter\View, 726

索引

非字母

() （方法）, 801807, 1258

__construct() (*CodeIgniter\Cookie\Cookie method*), 1114

__construct() (*CodeIgniter\Cookie\CookieStore method*), 1119

A

addColumn() (*CodeIgniter\Database\Forge method*), 1063

addField() (*CodeIgniter\Database\Forge method*), 1063

addForeignKey() (*CodeIgniter\Database\Forge method*), 1063

addHeader() (*CodeIgniter\HTTP\Message method*), 674

addKey() (*CodeIgniter\Database\Forge method*), 1064

addPrimaryKey() (*CodeIgniter\Database\Forge method*), 1065

addResponseHeaders() (*CodeIgniter\HTTP\Cors method*), 1126

addRow() (*CodeIgniter\View\Table method*), 776

addUniqueKey() (*CodeIgniter\Database\Forge method*), 1065

alternator() (*global function*), 1436

anchor() (*global function*), 1452

anchor_popup() (*global function*), 1454

app_timezone() (*global function*), 536

appendBody() (*CodeIgniter\HTTP\Message method*), 669

appendHeader() (*CodeIgniter\HTTP\Message method*), 673

APPPATH (*global constant*), 546

array_deep_search() (*global function*), 1353

array_flatten_with_dots() (*global function*), 1355

array_group_by() (*global function*), 1357

array_sort_by_multiple_keys() (*global function*), 1353

ascii_to_entities() (*global function*), 1441

attach() (*CodeIgniter\Email\Email

method), **1154**
audio() (*global function*), **1414**
auto_link() (*global function*), **1456**
autoTypography() (*global function*), **1280**

B

base_url() (*global function*), **1449**
BaseBuilder (CodeIgniter\Database 中的类), **942**
BaseCommand (CodeIgniter\CLI 中的类), **1556**
BaseResult (CodeIgniter\Database 中的类), **872**

C

cache() (*global function*), **529**
CacheInterface (CodeIgniter\Cache 中的类), **1091**
call() (*CodeIgniter\CLI\BaseCommand method*), **1556**
camelize() (*global function*), **1422**
character_limiter() (*global function*), **1440**
clean() (*CodeIgniter\Cache\CacheInterface method*), **1094**
clear() (*CodeIgniter\Cookie\CookieStore method*), **1121**
clear() (*CodeIgniter\Email\Email method*), **1152**
clear() (*CodeIgniter\View\Table method*), **779**
CodeIgniter (*namespace*), **1017**
CodeIgniter\Cache (*namespace*), **1091**
CodeIgniter\CLI (*namespace*), **1556**
CodeIgniter\Cookie (*namespace*), **1113**
CodeIgniter\Database (*namespace*), **872**, **942**, **1063**, **1077**
CodeIgniter\Email (*namespace*), **1148**
CodeIgniter\Encryption (*namespace*), **1165**
CodeIgniter\Encryption\EncrypterInterface (*interface in CodeIgniter\Encryption*), **1166**
CodeIgniter\HTTP (*namespace*), **669**, **676**, **694**, **789**, **1126**, **1307**
CodeIgniter\View (*namespace*), **726**, **763**, **774**
config() (*global function*), **530**
convert_accented_characters() (*global function*), **1442**
cookie() (*global function*), **530**
cookies() (*global function*), **531**
CookieStore (CodeIgniter\Cookie 中的类), **1119**
Cookie (CodeIgniter\Cookie 中的类), **1113**
Cors (CodeIgniter\HTTP 中的类), **1126**
countAll() (*CodeIgniter\Database\BaseBuilder method*), **943**
countAllResults() (*CodeIgniter\Database\BaseBuilder method*), **943**
counted() (*global function*), **1422**
createDatabase() (*CodeIgniter\Database\Forge method*), **1065**
createKey() (*CodeIgniter\Encryption\Encryption method*), **1165**

`createTable()` (*CodeIgniter\Database\Forge method*), **1066**
`csp_script_nonce()` (*global function*), **537**
`csp_style_nonce()` (*global function*), **537**
`csrf_field()` (*global function*), **538**
`csrf_hash()` (*global function*), **537**
`csrf_header()` (*global function*), **537**
`csrf_meta()` (*global function*), **538**
`csrf_token()` (*global function*), **537**
`current_url()` (*global function*), **1450**

D

`dasherize()` (*global function*), **1425**
`dataSeek()` (*CodeIgniter\Database\BaseResult method*), **875**
`DAY` (*global constant*), **546**
`db()` (*CodeIgniter\Database\BaseBuilder method*), **942**
`DECADE` (*global constant*), **546**
`decamelize()` (*global function*), **1423**
`decrement()` (*CodeIgniter\Cache\CacheInterface method*), **1094**
`decrement()` (*CodeIgniter\Database\BaseBuilder method*), **967**
`delete()` (*CodeIgniter\Cache\CacheInterface method*), **1092**
`delete()` (*CodeIgniter\Database\BaseBuilder method*), **966**
`delete_cookie()` (*global function*), **1367**
`delete_files()` (*global function*), **1374**
`deleteBatch()` (*CodeIgniter\Database\BaseBuilder method*), **966**
`deleteCookie()` (*CodeIgniter\HTTP\Response method*), **795**
`deleteMatching()` (*CodeIgniter\Cache\CacheInterface method*), **1092**
`directory_map()` (*global function*), **1371**
`directory_mirror()` (*global function*), **1373**
`dispatch()` (*CodeIgniter\Cookie\CookieStore method*), **1121**
`display()` (*CodeIgniter\Cookie\CookieStore method*), **1121**
`distinct()` (*CodeIgniter\Database\BaseBuilder method*), **946**
`doctype()` (*global function*), **1418**
`dot_array_search()` (*global function*), **1351**
`dropColumn()` (*CodeIgniter\Database\Forge method*), **1066**
`dropDatabase()` (*CodeIgniter\Database\Forge method*), **1066**
`dropKey()` (*CodeIgniter\Database\Forge method*), **1067**
`dropPrimaryKey()` (*CodeIgniter\Database\Forge method*), **1067**
`dropTable()` (*CodeIgniter\Database\Forge method*), **1068**

E

`ellipsize()` (*global function*), **1445**
`Email` (`CodeIgniter\Email` 中的类),

1148
embed() (*global function*), 1415
emptyTable() (*CodeIgniter\Database\BaseBuilder method*), 967
encode_php_tags() (*global function*), 1431
EncryptionEncrypterInterfacedecrypt() (*CodeIgniter\Encryption\CodeIgniterEncryptionEncrypterInterface method*), 1167
EncryptionEncrypterInterfaceencrypt() (*CodeIgniter\Encryption\CodeIgniterEncryptionEncrypterInterface method*), 1166
Encryption (CodeIgniter\Encryption 中的类), 1165
entities_to_ascii() (*global function*), 1441
env() (*global function*), 531
esc() (*global function*), 532
excerpt() (*global function*), 1446

F

fake() (*global function*), 1432
FCPATH (*global constant*), 546
fetchGlobal() (*CodeIgniter\HTTP\Request method*), 680
findMigrations() (*CodeIgniter\Database\MigrationRunner method*), 1077
force() (*CodeIgniter\Database\MigrationRunner method*), 1078
force_https() (*global function*), 538
Forge (CodeIgniter\Database 中的类), 1063
form_button() (*global function*), 1397
form_checkbox() (*global function*), 1394
form_close() (*global function*), 1399
form_dropdown() (*global function*), 1390
form_fieldset() (*global function*), 1392
form_fieldset_close() (*global function*), 1393
form_hidden() (*global function*), 1384
form_input() (*global function*), 1386
form_label() (*global function*), 1396
form_multiselect() (*global function*), 1391
form_open() (*global function*), 1381
form_open_multipart() (*global function*), 1383
form_password() (*global function*), 1388
form_radio() (*global function*), 1395
form_reset() (*global function*), 1397
form_submit() (*global function*), 1396
form_textarea() (*global function*), 1389
form_upload() (*global function*), 1389
formatCharacters() (*global function*), 1280
freeResult() (*CodeIgniter\Database\BaseResult method*), 878
from() (*CodeIgniter\Database\BaseBuilder method*), 946
fromCookieHeaders() (*CodeIgniter\Cookie\CookieStore method*), 1119
fromHeaderString() (*CodeIgniter\Cookie\Cookie method*), 1113
fromSubquery() (*CodeIgniter\Database\BaseBuilder method*), 947
function_usable() (*global function*), 539

G

generate() (*CodeIgniter\View\Table method*), 775

get() (*CodeIgniter\Cache\CacheInterface method*), 1091

get() (*CodeIgniter\Cookie\CookieStore method*), 1120

get() (*CodeIgniter\Database\BaseBuilder method*), 943

get_cookie() (*global function*), 1367

get_dir_file_info() (*global function*), 1376

get_file_info() (*global function*), 1377

get_filenames() (*global function*), 1375

getAgentString() (*CodeIgniter\HTTP\UserAgent method*), 1311

getBody() (*CodeIgniter\HTTP\Message method*), 669

getBrowser() (*CodeIgniter\HTTP\UserAgent method*), 1309

getCacheInfo() (*CodeIgniter\Cache\CacheInterface method*), 1095

getCompiledDelete() (*CodeIgniter\Database\BaseBuilder method*), 969

getCompiledInsert() (*CodeIgniter\Database\BaseBuilder method*), 968

getCompiledSelect() (*CodeIgniter\Database\BaseBuilder method*), 968

getCompiledUpdate() (*CodeIgniter\Database\BaseBuilder method*), 968

getCookie() (*CodeIgniter\HTTP\IncomingRequest method*), 698

getCookie() (*CodeIgniter\HTTP\Response method*), 797

getCookies() (*CodeIgniter\HTTP\Response method*), 797

getCustomResultObject() (*CodeIgniter\Database\BaseResult method*), 873

getCustomRowObject() (*CodeIgniter\Database\BaseResult method*), 875

getDomain() (*CodeIgniter\Cookie\Cookie method*), 1115

getEnv() (*CodeIgniter\HTTP\Request method*), 678

getExpiresString() (*CodeIgniter\Cookie\Cookie method*), 1115

getExpiresTimestamp() (*CodeIgniter\Cookie\Cookie method*), 1115

getFieldCount() (*CodeIgniter\Database\BaseResult method*), 877

getFieldData() (*CodeIgniter\Database\BaseResult method*), 877

getFieldNames() (*CodeIgniter\Database\BaseResult method*), 877

getFilterCaller() (*global function*), 1506

getFiltersForRoute() (*global function*), 1505

getFirstRow() (*CodeIgniter\Database\BaseResult method*), 876
getGet() (*CodeIgniter\HTTP\IncomingRequest method*), 695
getGetPost() (*CodeIgniter\HTTP\IncomingRequest method*), 698
getHeaderLine() (*CodeIgniter\HTTP\Message method*), 672
getId() (*CodeIgniter\Cookie\Cookie method*), 1114
getIPAddress() (*CodeIgniter\HTTP\Request method*), 676
getLastRow() (*CodeIgniter\Database\BaseResult method*), 876
getMaxAge() (*CodeIgniter\Cookie\Cookie method*), 1115
getMetadata() (*CodeIgniter\Cache\CacheInterface method*), 1095
getMethod() (*CodeIgniter\HTTP\Request method*), 677
getMobile() (*CodeIgniter\HTTP\UserAgent method*), 1310
getName() (*CodeIgniter\Cookie\Cookie method*), 1114
getNextRow() (*CodeIgniter\Database\BaseResult method*), 876
getNumRows() (*CodeIgniter\Database\BaseResult method*), 877
getOptions() (*CodeIgniter\Cookie\Cookie method*), 1115
getPad() (*CodeIgniter\CLI\BaseCommand method*), 1557
getPath() (*CodeIgniter\Cookie\Cookie method*), 1115
getPath() (*CodeIgniter\HTTP\IncomingRequest method*), 700
getPlatform() (*CodeIgniter\HTTP\UserAgent method*), 1310
getPost() (*CodeIgniter\HTTP\IncomingRequest method*), 696
getPostGet() (*CodeIgniter\HTTP\IncomingRequest method*), 697
getPrefix() (*CodeIgniter\Cookie\Cookie method*), 1114
getPrefixedName() (*CodeIgniter\Cookie\Cookie method*), 1115
getPreviousRow() (*CodeIgniter\Database\BaseResult method*), 876
getProtocolVersion() (*CodeIgniter\HTTP\Message method*), 675
getReasonPhrase() (*CodeIgniter\HTTP\Response method*), 790
getReferrer() (*CodeIgniter\HTTP\UserAgent method*), 1310
getResult() (*CodeIgniter\Database\BaseResult method*), 872
getResultArray()

(*CodeIgniter\Database\BaseResult method*), **873**
getResultObject() (*CodeIgniter\Database\BaseResult method*), **873**
getRobot() (*CodeIgniter\HTTP\UserAgent method*), **1310**
getRow() (*CodeIgniter\Database\BaseResult method*), **873**
getRowArray() (*CodeIgniter\Database\BaseResult method*), **874**
getRowObject() (*CodeIgniter\Database\BaseResult method*), **874**
getSameSite() (*CodeIgniter\Cookie\Cookie method*), **1115**
getServer() (*CodeIgniter\HTTP\IncomingRequest method*), **699**
getServer() (*CodeIgniter\HTTP\Request method*), **678**
getStatusCode() (*CodeIgniter\HTTP\Response method*), **789**
getUnbufferedRow() (*CodeIgniter\Database\BaseResult method*), **874**
getUserAgent() (*CodeIgniter\HTTP\IncomingRequest method*), **700**
getValue() (*CodeIgniter\Cookie\Cookie method*), **1115**
getVar() (*CodeIgniter\HTTP\IncomingRequest method*), **695**
getVersion() (*CodeIgniter\HTTP\UserAgent method*), **1309**
getWhere() (*CodeIgniter\Database\BaseBuilder method*), **943**
groupBy() (*CodeIgniter\Database\BaseBuilder method*), **959**
groupEnd() (*CodeIgniter\Database\BaseBuilder method*), **951**
groupStart() (*CodeIgniter\Database\BaseBuilder method*), **950**

H

handlePreflightRequest() (*CodeIgniter\HTTP\Cors method*), **1127**
has() (*CodeIgniter\Cookie\CookieStore method*), **1120**
has_cookie() (*global function*), **1368**
hasCookie() (*CodeIgniter\HTTP\Response method*), **796**
hasHeader() (*CodeIgniter\HTTP\Message method*), **671**
having() (*CodeIgniter\Database\BaseBuilder method*), **953**
havingGroupEnd() (*CodeIgniter\Database\BaseBuilder method*), **958**
havingGroupStart() (*CodeIgniter\Database\BaseBuilder method*), **958**
havingIn() (*CodeIgniter\Database\BaseBuilder method*), **955**
havingLike() (*CodeIgniter\Database\BaseBuilder*

method), 956
havingNotIn() (CodeIgniter\Database\BaseBuilder method), 955
header() (CodeIgniter\HTTP\Message method), 670
headers() (CodeIgniter\HTTP\Message method), 670
helper() (global function), 532
highlight_code() (global function), 1443
highlight_phrase() (global function), 1443
HOUR (global constant), 546
humanize() (global function), 1424

I

img() (global function), 1404
img_data() (global function), 1405
IncomingRequest (CodeIgniter\HTTP 中的类), 694
increment() (CodeIgniter\Cache\CacheInterface method), 1093
increment() (CodeIgniter\Database\BaseBuilder method), 967
increment_string() (global function), 1435
index_page() (global function), 1452
initialize() (CodeIgniter\Encryption\Encryption method), 1165
insert() (CodeIgniter\Database\BaseBuilder method), 961
insertBatch() (CodeIgniter\Database\BaseBuilder method), 961
is_cli() (global function), 539
is_pluralizable() (global function), 1424
is_really_writable() (global function), 539
is_windows() (global function), 539
isAJAX() (CodeIgniter\HTTP\IncomingRequest method), 694
isBrowser() (CodeIgniter\HTTP\UserAgent method), 1307
isCLI() (CodeIgniter\HTTP\IncomingRequest method), 694
isExpired() (CodeIgniter\Cookie\Cookie method), 1115
isHTTPOnly() (CodeIgniter\Cookie\Cookie method), 1115
isMobile() (CodeIgniter\HTTP\UserAgent method), 1308
isPreflightRequest() (CodeIgniter\HTTP\Cors method), 1127
isRaw() (CodeIgniter\Cookie\Cookie method), 1115
isReferral() (CodeIgniter\HTTP\UserAgent method), 1309
isRobot() (CodeIgniter\HTTP\UserAgent method), 1309
isSecure() (CodeIgniter\Cookie\Cookie method), 1115
isSecure() (CodeIgniter\HTTP\IncomingRequest method), 694
isSupported() (CodeIgniter\Cache\CacheInterface

method), 1091
isValidIP() (CodeIgniter\HTTP\Request method), 676

J

join() (CodeIgniter\Database\BaseBuilder method), 948

L

lang() (global function), 532
latest() (CodeIgniter\Database\MigrationRunner method), 1077
like() (CodeIgniter\Database\BaseBuilder method), 951
limit() (CodeIgniter\Database\BaseBuilder method), 959
link_tag() (global function), 1406
log_message() (global function), 540

M

mailto() (global function), 1455
makeColumns() (CodeIgniter\View\Table method), 777
mb_url_title() (global function), 1459
Message (CodeIgniter\HTTP 中的类), 669
MigrationRunner (CodeIgniter\Database 中的类), 1077
MINUTE (global constant), 546
model() (global function), 533
Model (CodeIgniter 中的类), 1017
modifyColumn() (CodeIgniter\Database\Forge method), 1068
MONTH (global constant), 546

N

nl2brExceptPre() (global function), 1281
noCache() (CodeIgniter\HTTP\Response method), 791
notGroupStart() (CodeIgniter\Database\BaseBuilder method), 951
notHavingGroupStart() (CodeIgniter\Database\BaseBuilder method), 958
notHavingLike() (CodeIgniter\Database\BaseBuilder method), 957
notLike() (CodeIgniter\Database\BaseBuilder method), 952
now() (global function), 1369
number_to_amount() (global function), 1428
number_to_currency() (global function), 1429
number_to_roman() (global function), 1429
number_to_size() (global function), 1427

O

object() (global function), 1416
octal_permissions() (global function), 1377
offset() (CodeIgniter\Database\BaseBuilder method), 960
ol() (global function), 1412
old() (global function), 533
onConstraint() (CodeIgniter\Database\BaseBuilder method), 964
orderBy() (CodeIgniter\Database\BaseBuilder method), 959
ordinal() (global function), 1425

ordinalize() (*global function*), 1426
orGroupStart() (*CodeIgniter\Database\BaseBuilder method*), 951
orHaving() (*CodeIgniter\Database\BaseBuilder method*), 954
orHavingGroupStart() (*CodeIgniter\Database\BaseBuilder method*), 958
orHavingIn() (*CodeIgniter\Database\BaseBuilder method*), 954
orHavingLike() (*CodeIgniter\Database\BaseBuilder method*), 956
orHavingNotIn() (*CodeIgniter\Database\BaseBuilder method*), 954
orLike() (*CodeIgniter\Database\BaseBuilder method*), 952
orNotGroupStart() (*CodeIgniter\Database\BaseBuilder method*), 951
orNotHavingGroupStart() (*CodeIgniter\Database\BaseBuilder method*), 958
orNotHavingLike() (*CodeIgniter\Database\BaseBuilder method*), 957
orNotLike() (*CodeIgniter\Database\BaseBuilder method*), 953
orWhere() (*CodeIgniter\Database\BaseBuilder method*), 948
orWhereIn() (*CodeIgniter\Database\BaseBuilder method*), 949
orWhereNotIn() (*CodeIgniter\Database\BaseBuilder method*), 949

P

param() (*global function*), 1417
parse() (*CodeIgniter\HTTP\UserAgent method*), 1311
Parser (CodeIgniter\View 中的类), 763
pascalize() (*global function*), 1423
plural() (*global function*), 1421
populateHeaders() (*CodeIgniter\HTTP\Message method*), 670
prep_url() (*global function*), 1459
prependHeader() (*CodeIgniter\HTTP\Message method*), 674
previous_url() (*global function*), 1451
printDebugger() (*CodeIgniter\Email\Email method*), 1156
processIndexes() (*CodeIgniter\Database\Forge method*), 1068
put() (*CodeIgniter\Cookie\CookieStore method*), 1120

Q

quotes_to_entities() (*global function*), 1438

R

random_string() (*global function*), 1434
redirect() (*global function*), 541

reduce_double_slashes() (*global function*), 1436
reduce_multiples() (*global function*), 1438
regress() (*CodeIgniter\Database\MigrationRunner method*), 1078
remember() (*CodeIgniter\Cache\CacheInterface method*), 1091
remove() (*CodeIgniter\Cookie\CookieStore method*), 1120
remove_invisible_characters() (*global function*), 541
removeHeader() (*CodeIgniter\HTTP\Message method*), 673
renameTable() (*CodeIgniter\Database\Forge method*), 1069
render() (*CodeIgniter\View\Parser method*), 763
render() (*CodeIgniter\View\View method*), 726
renderString() (*CodeIgniter\View\Parser method*), 763
renderString() (*CodeIgniter\View\View method*), 727
replace() (*CodeIgniter\Database\BaseBuilder method*), 966
request() (*global function*), 541
Request (CodeIgniter\HTTP 中的类), 676
resetQuery() (*CodeIgniter\Database\BaseBuilder method*), 942
response() (*global function*), 542
Response (CodeIgniter\HTTP 中的类), 789
ROOTPATH (*global constant*), 546
route_to() (*global function*), 542

S

safe_mailto() (*global function*), 1456
same_file() (*global function*), 1378
sanitize_filename() (*global function*), 1431
save() (*CodeIgniter\Cache\CacheInterface method*), 1092
script_tag() (*global function*), 1408
SECOND (*global constant*), 546
select() (*CodeIgniter\Database\BaseBuilder method*), 944
selectAvg() (*CodeIgniter\Database\BaseBuilder method*), 944
selectCount() (*CodeIgniter\Database\BaseBuilder method*), 945
selectMax() (*CodeIgniter\Database\BaseBuilder method*), 945
selectMin() (*CodeIgniter\Database\BaseBuilder method*), 945
selectSubquery() (*CodeIgniter\Database\BaseBuilder method*), 946
selectSum() (*CodeIgniter\Database\BaseBuilder method*), 945
send() (*CodeIgniter\Email\Email method*), 1153
send() (*CodeIgniter\HTTP\Response method*), 793

service() (*global function*), 544
session() (*global function*), 534
set() (*CodeIgniter\Database\BaseBuilder method*), 960
set_checkbox() (*global function*), 1400
set_cookie() (*global function*), 1366
set_radio() (*global function*), 1401
set_realpath() (*global function*), 1378
set_select() (*global function*), 1400
set_value() (*global function*), 1399
setAltMessage() (*CodeIgniter\Email\Email method*), 1151
setAttachmentCID() (*CodeIgniter\Email\Email method*), 1155
setBCC() (*CodeIgniter\Email\Email method*), 1150
setBody() (*CodeIgniter\HTTP\Message method*), 669
setCache() (*CodeIgniter\HTTP\Response method*), 792
setCaption() (*CodeIgniter\View\Table method*), 775
setCC() (*CodeIgniter\Email\Email method*), 1150
setConditionalDelimiters() (*CodeIgniter\View\Parser method*), 765
setContentType() (*CodeIgniter\HTTP\Response method*), 791
setCookie() (*CodeIgniter\HTTP\Response method*), 793
setData() (*CodeIgniter\Database\BaseBuilder method*), 965
setData() (*CodeIgniter\View\Parser method*), 764
setData() (*CodeIgniter\View\View method*), 727
setDate() (*CodeIgniter\HTTP\Response method*), 791
setDefaults() (*CodeIgniter\Cookie\Cookie method*), 1113
setDelimiters() (*CodeIgniter\View\Parser method*), 765
setEmpty() (*CodeIgniter\View\Table method*), 779
setFooting() (*CodeIgniter\View\Table method*), 776
setFrom() (*CodeIgniter\Email\Email method*), 1148
setGlobal() (*CodeIgniter\HTTP\Request method*), 679
setGroup() (*CodeIgniter\Database\MigrationRunner method*), 1079
setHeader() (*CodeIgniter\Email\Email method*), 1152
setHeader() (*CodeIgniter\HTTP\Message method*), 672
setHeading() (*CodeIgniter\View\Table method*), 776
setInsertBatch() (*CodeIgniter\Database\BaseBuilder method*), 962
setLastModified() (*CodeIgniter\HTTP\Response method*), 793
setMessage() (*CodeIgniter\Email\Email method*), 1151
setMethod() (*CodeIgniter\HTTP\Request*

method), 677
setNamespace() (CodeIgniter\Database\MigrationRunner method), 1079
setPad() (CodeIgniter\CLI\BaseCommand method), 1557
setPath() (CodeIgniter\HTTP\IncomingRequest method), 700
setProtocolVersion() (CodeIgniter\HTTP\Message method), 675
setQueryAsData() (CodeIgniter\Database\BaseBuilder method), 947
setReplyTo() (CodeIgniter\Email\Email method), 1149
setRow() (CodeIgniter\Database\BaseResult method), 875
setStatusCode() (CodeIgniter\HTTP\Response method), 790
setSubject() (CodeIgniter\Email\Email method), 1151
setSyncRowsWithHeading() (CodeIgniter\View\Table method), 780
setTemplate() (CodeIgniter\View\Table method), 778
setTo() (CodeIgniter\Email\Email method), 1149
setUpdateBatch() (CodeIgniter\Database\BaseBuilder method), 965
setValidationMessage() (CodeIgniter\Model method), 1018
setValidationMessages() (CodeIgniter\Model method), 1018
setValidationRule() (CodeIgniter\Model method), 1017
setValidationRules() (CodeIgniter\Model method), 1017
setVar() (CodeIgniter\View\Parser method), 765
setVar() (CodeIgniter\View\View method), 728
showError() (CodeIgniter\CLI\BaseCommand method), 1556
showHelp() (CodeIgniter\CLI\BaseCommand method), 1557
single_service() (global function), 544
singular() (global function), 1421
site_url() (global function), 1448
slash_item() (global function), 545
source() (global function), 1415
stringify_attributes() (global function), 545
strip_image_tags() (global function), 1431
strip_quotes() (global function), 1439
strip_slashes() (global function), 1437
symbolic_permissions() (global function), 1377
SYSTEMPATH (global constant), 546

T

Table (CodeIgniter\View 中的类), 774
timer() (global function), 534
timezone_select() (global function), 1370
toArray() (CodeIgniter\Cookie\Cookie method), 1119
toHeaderString()

(*CodeIgniter\Cookie\Cookie method*), **1118**
track() (*global function*), **1417**
truncate() (*CodeIgniter\Database\BaseBuilder method*), **967**

U

ul() (*global function*), **1408**
underscore() (*global function*), **1423**
union() (*CodeIgniter\Database\BaseBuilder method*), **960**
unionAll() (*CodeIgniter\Database\BaseBuilder method*), **960**
update() (*CodeIgniter\Database\BaseBuilder method*), **963**
updateBatch() (*CodeIgniter\Database\BaseBuilder method*), **963**
updateFields() (*CodeIgniter\Database\BaseBuilder method*), **964**
upsert() (*CodeIgniter\Database\BaseBuilder method*), **962**
upsertBatch() (*CodeIgniter\Database\BaseBuilder method*), **962**
uri_string() (*global function*), **1451**
url_is() (*global function*), **1461**
url_title() (*global function*), **1458**
url_to() (*global function*), **1459**
UserAgent (CodeIgniter\HTTP 中的类), **1307**

V

validateKey() (*CodeIgniter\Cache\CacheInterface method*), **1096**
validation_errors() (*global function*), **1401**
validation_list_errors() (*global function*), **1402**
validation_show_error() (*global function*), **1403**
video() (*global function*), **1412**
view() (*global function*), **535**
view_cell() (*global function*), **536**
View (CodeIgniter\View 中的类), **726**

W

WEEK (*global constant*), **546**
where() (*CodeIgniter\Database\BaseBuilder method*), **948**
whereIn() (*CodeIgniter\Database\BaseBuilder method*), **950**
whereNotIn() (*CodeIgniter\Database\BaseBuilder method*), **950**
withDomain() (*CodeIgniter\Cookie\Cookie method*), **1117**
withExpired() (*CodeIgniter\Cookie\Cookie method*), **1116**
withExpires() (*CodeIgniter\Cookie\Cookie method*), **1116**
withHTTPOnly() (*CodeIgniter\Cookie\Cookie method*), **1118**
withMethod() (*CodeIgniter\HTTP\Request method*), **677**
withName() (*CodeIgniter\Cookie\Cookie method*), **1116**

withNeverExpiring() (*CodeIgniter\Cookie\Cookie method*), **1117**
withPath() (*CodeIgniter\Cookie\Cookie method*), **1117**
withPrefix() (*CodeIgniter\Cookie\Cookie method*), **1115**
withRaw() (*CodeIgniter\Cookie\Cookie method*), **1115**
withSameSite() (*CodeIgniter\Cookie\Cookie method*), **1118**
withSecure() (*CodeIgniter\Cookie\Cookie method*), **1118**
withValue() (*CodeIgniter\Cookie\Cookie method*), **1116**
word_censor() (*global function*), **1442**
word_limiter() (*global function*), **1439**
word_wrap() (*global function*), **1444**
write_file() (*global function*), **1373**
WRITEPATH (*global constant*), **546**

X

xml_convert() (*global function*), **1463**

Y

YEAR (*global constant*), **546**