Sample: 快速搭建 LISP 学习环境

我是副标题什么的 Lisp

WisdomFusion

https://github.com/WisdomFusion

摘要

Laravel 5.6 在 Laravel 5.5 的基础上继续进行优化，包括**日志系统**、单机任务调度、模型序列化优化、动态频率限制、广播频道类、API 资源控制器生成、Eloquent 日期格式化优化、Blade 组件别名、Argon2 密码哈希支持、引入 Collision 扩展包等等等等。此外，所有的前端脚手架代码都已升级到 Bootstrap 4，Laravel 底层使用的 Symfony 组件都已升级到 Symfony ~4.0 版本。

关键字：LISP　PHP　学习环境

1 新版特性

Laravel 5.6 在 Laravel 5.5 的基础上继续进行优化，包括**日志系统**、单机任务调度、模型序列化优化、动态频率限制、广播频道类、API 资源控制器生成、Eloquent 日期格式化优化、Blade 组件别名、Argon2 密码哈希支持、引入 Collision 扩展包等等等等。此外，所有的前端脚手架代码都已升级到 Bootstrap 4，Laravel 底层使用的 Symfony 组件都已升级到 Symfony ~4.0 版本。

</> CODE 1: test/test.json

```
'channels' => [
  'stack' => [
  'driver' => 'stack',
  'channels' => ['syslog', 'slack'],
  ],
],
```

```
cd my-app
ng serve --open
```

Laravel 5.6 版本的发布恰逢 Spark 6.0 的发布，所以这也是自 Laravel Spark 发布以来第一次重大版本升级。Spark 6.0 为 Stripe 和 Braintree 引入了按座定价功能，以及本地化、Bootstrap 4、增强 UI 和 Stripe Elements 支持。

此外，现在可以使用日志系统的新 “tap” 功能很轻松地自定义已存在的日志频道。想要了解更多细节，请查看完整日志文档。

```
{{Talk|open|date=March 28, 2017}}

Hi Handbook team,

I constructively think FOO part of the Handbook can be enhanced in
   BAR way. Here is an example of the code/text that will make the
   improvement:

(insert code or text to be improved here.)

Thank you for considering my suggestion. --~~~~
```

如果你的应用运行在多个服务器上，现在可以限定只在一台机器上运行调度任务。例如，假设你有一个在每周五晚上生成新报告的调度任务，如果任务调度器运行在三个服务器上，这个调度任务就会在三台机器上运行并生成同样的报告三次，这样很不优雅，甚至很糟糕！

要指定任务只在一台机器上运行，可以在定义调度任务时使用 `onOneServer` 方法，第一台获取到任务的机器会给这个任务上一把原子级别的锁来阻止其他服务器同时运行同一个任务：

2 核心概念

2.1 请求的生命周期

当我们使用现实世界中的任何工具时，如果理解了该工具的工作原理，那么用起来就会得心应手，应用开发也是如此。当你理解了开发工具如何工作，用起来就会更加游刃有余。

这篇文档的目标就是从更高层面向你阐述 Laravel 框架的工作原理。通过对框架更全面的了解，一切都不再那么神秘，你将会更加自信地构建应用。如果你不能马上理解所有这些条款，不要失去信心！先试着掌握一些基本的东西，你的知识水平将会随着对文档的探索而不断提升。

2.1.1 第一件事

Laravel 应用的所有请求入口都是 `public/index.php` 文件，所有请求都会被 Web 服务器（Apache/Nginx）导向这个文件。index.php 文件包含的代码并不多，但是，这里是加载框架其它部分的起点。

`index.php` 文件载入 Composer 生成的自动加载设置，然后从 `bootstrap/app.php` 脚本获取 Laravel 应用实例，Laravel 的第一个动作就是创建服务容器实例。

2.1.2 HTTP/Console 内核

接下来，请求被发送到 HTTP 内核或 Console 内核（分别用于处理 Web 请求和 Artisan 命令），这取决于进入应用的请求类型。这两个内核是所有请求都要经过的中央处理器，现在，就让我们

聚焦在位于 app/Http/Kernel.php 的 HTTP 内核。

HTTP 内核还定义了一系列所有请求在处理前需要经过的 HTTP 中间件，这些中间件处理 HTTP 会话的读写、判断应用是否处于维护模式、验证 CSRF 令牌等等。

HTTP 内核的 handle 方法签名相当简单：获取一个 Request，返回一个 Response，可以把该内核想象作一个代表整个应用的大黑盒子，输入 HTTP 请求，返回 HTTP 响应。

HTTP/Console 内核

HTTP 内核继承自 Illuminate\Foundation\Http\Kernel 类，该类定义了一个 bootstrappers 数组，这个数组中的类在请求被执行前运行，这些 bootstrappers 配置了错误处理、日志、检测应用环境以及其它在请求被处理前需要执行的任务。

Note

HTTP 内核继承自 Illuminate\Foundation\Http\Kernel 类，该类定义了一个 bootstrappers 数组，这个数组中的类在请求被执行前运行，这些 bootstrappers 配置了错误处理、日志、检测应用环境以及其它在请求被处理前需要执行的任务。

Important

HTTP 内核继承自 Illuminate\Foundation\Http\Kernel 类，该类定义了一个 bootstrappers 数组，这个数组中的类在请求被执行前运行，这些 bootstrappers 配置了错误处理、日志、检测应用环境以及其它在请求被处理前需要执行的任务。

```
cd my-app
ng serve --open
```


```
cd my-app
ng serve --open
```


2.1.3 服务提供者

内核启动过程中最重要的动作之一就是为应用载入服务提供者，应用的所有服务提供者都被配置在 config/app.php 配置文件的 providers 数组中。首先，所有提供者的 register 方法被调用，然后，所有提供者被注册之后，boot 方法被调用。

服务提供者负责启动框架的所有各种各样的组件，比如数据库、队列、验证器，以及路由组件等，正是因为他们启动并配置了框架提供的所有特性，所以服务提供者是整个 Laravel 启动过程中最重要的部分。

2.1.4 分发请求

一旦应用被启动并且所有的服务提供者被注册，`Request` 将会被交给路由器进行分发，路由器将会分发请求到路由或控制器，同时运行所有路由指定的中间件。

2.1.5 聚焦服务提供者

服务提供者是启动 Laravel 应用中最关键的部分，应用实例被创建后，服务提供者被注册，请求被交给启动后的应用进行处理，整个过程就是这么简单！

对 Laravel 应用如何通过服务提供者构建和启动有一个牢固的掌握非常有价值，当然，应用默认的服务提供者存放在 `app/Providers` 目录下。

默认情况下，`AppServiceProvider` 是空的，这里是添加自定义启动和服务容器绑定的最佳位置，当然，对大型应用，你可能希望创建多个服务提供者，每一个都有着更加细粒度的启动。

3 服务容器 3

Laravel 服务容器是一个用于管理类依赖和执行依赖注入的强大工具。依赖注入听上去很花哨，其实质是通过构造函数或者某些情况下通过 `setter` 方法将类依赖注入到类中。

让我们看一个简单的例子：

</> CODE 2: PHP 代码样例

```php
<?php
namespace App\Http\Controllers;

use App\User;
use App\Repositories\UserRepository;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
  /**
  * The user repository implementation.
  *
  * @var UserRepository
  */
  protected $users;

  /**
  * Create a new controller instance.
  *
  * @param  UserRepository  $users
  * @return void
  */
```

```
    public function __construct(UserRepository $users)
    {
        $this->users = $users;
    }

    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        $user = $this->users->find($id);
        return view('user.profile', ['user' => $user]);
    }
}
```

在本例中，UserController需要从数据源获取用户，所以，我们注入了一个可以获取用户的服务UserRepository，其扮演的角色类似使用Eloquent从数据库获取用户信息。注入UserRepository后，我们可以在其基础上封装其他实现，也可以模拟或者创建一个假的UserRepository实现用于测试。

深入理解 Laravel 服务容器对于构建功能强大的大型 Laravel 应用而言至关重要，对于贡献代码到 Laravel 核心也很有帮助。