——高性能PHP应用开发之

第8讲 Laravel框架

目录

CONTENTS

目录

CONTENTS

Laravel框架简介

■ PHP框架

- ✓ 轻量级框架：TP5、CodeIgniter
- ✓ 重量级框架：Laravel、Zend Framework、YII、Symfony
- ✓ 高性能框架：Yaf、Phalcon

■ Laravel框架安装环境

- ✓ Laravel5.6要求PHP7.1以上，Laravel5.5要求PHP7.0以上，Laravel5.4要求PHP5.6以上
- ✓ PHP扩展： OpenSSL、PDO、Mbstring、Tokenizer
- ✓ Composer和命令行工具
- ✓ PHPStorm、Sublime Text、Visual Studio Code
- ✓ 官方开发文档： https://laravel.com/docs/5.6
- ✓ 国内汉化开发文档： http://laravelacademy.org/tags/laravel

目录

CONTENTS

安装Laravel

■ 安装Laravel框架的方式

- ✓ 直接下载Laravel框架骨架结构压缩包，解压缩到指定目录即可
- ✓ 使用Laravel官方提供的 Laravel Homestead 环境
- ✓ 使用Composer安装Laravel应用程序（推荐使用）

■ 使用Composer安装Laravel

- ✓ 命令：composer create-project laravel/laravel 项目名称
- ✓ 也可以首先使用Composer下载 laravel安装器，再安装Laravel

```
> composer create-project laravel/laravel blog
Installing laravel/laravel (v5.4.9)
  - Installing laravel/laravel (v5.4.9)
    Downloading: 100%
```

当前Laravel最新版本为5.6，建议使用最新版本进行开发。

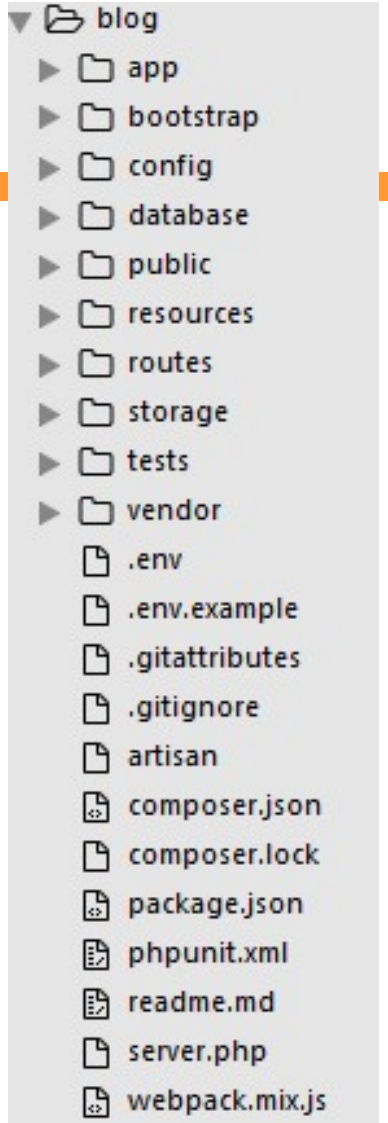
访问Laravel应用程序

■ 访问Laravel程序的方法

- ✓ Laravel程序的入口文件是 public/index.php，因此在浏览器中输入 http://127.0.0.1/应用程序目录/public/index.php 即可访问应用程序。
- ✓ 修改 public/目录下的 .htaccess 文件，设置端口号虚拟主机，通过 http://127.0.0.1:端口号 访问。
- ✓ 使用PHP内置服务器，在 public目录下，开启内置服务器，浏览器里访问。（推荐）
- • 开启内置服务器命令： php -S 127.0.0.1:88 （注意：在 public 目录下）
- • 使用Artisan命令：**php artisan serve** （注意：在 根目录下，端口默认为 8000）

目录结构剖析

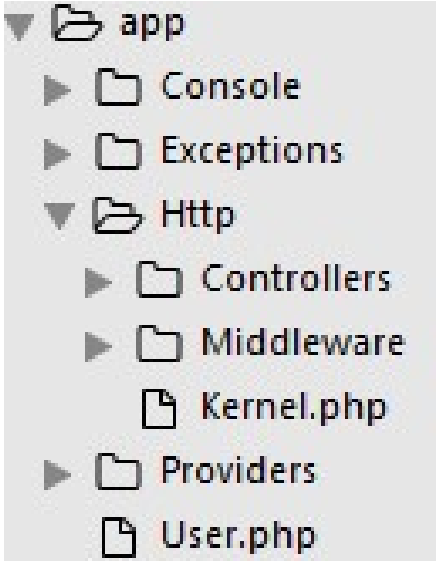
- app目录：应用程序开发
- config目录：配置文件
- database目录：数据库迁移及填充文件
- public目录：入口文件，静态资源
- resources目录：视图，多语言支持
- routes目录：路由文件
- storage目录：编译后模板文件，session数据等
- tests目录：单元测试
- vendor目录：开源库（Laravel源代码）
- .env文件：当前加载的配置项



目录结构剖析

■ app目录

- ✓ Console目录：应用所有自定义的Artisan命令
- ✓ Exceptions目录：应用的异常处理器
- ✓ Http目录：控制器、中间件以及表单请求等，几乎所有进入应用的请求处理都在这里进行
- ✓ Providers目录：应用的所有服务提供者
- ✓ User.php文件：模型类文件；在此特别强调，Laravel没有提供 Model 目录，完全可以自己创建 Model 目录用来表示MVC中的模型类。



Laravel程序的执行过程

- 初始化 Application类
  - ✓ 注册基本绑定
  - ✓ 注册基本服务提供者
  - ✓ 注册核心类别名
  - ✓ 设置根路径
- 注册共享的Kernel和异常处理器

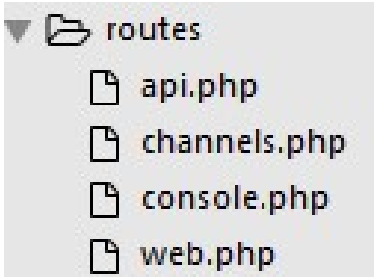
- 处理请求和响应

目录

CONTENTS

Laravel路由文件

■ Laravel路由文件位于 /routes 目录下



- ✓ **web.php**：文件包含的路由都会应用web中间件组，具备Session、CSRF防护以及Cookie加密功能，网站中采用的路由绝大多数都在此文件中定义。
- ✓ api.php：包含的路由应用了api中间件组，具备频率限制功能，这些路由是无状态的，所以请求通过这些路由进入应用需要通过token进行认证并且不能访问Session状态。一般使用在为客户端提供API接口的场景中。
- ✓ console.php：用于定义所有基于闭包的控制台命令，主要是用于Artisan控制台命令中使用该路由。
- ✓ channels.php：注册应用程序所提供的广播频道。

定义路由

■ 路由定义形式：使用 Route 门面 提供的方法来定义

- ✓ 定义路由采用静态方法的形式进行定义；
- ✓ 可以定义不同HTTP请求类型的路由；
- ✓ 路由可以由闭包函数响应，也可以由控制器动作响应。

```
static void get(string $uri, \Closure|array|string $action)
static void post(string $uri, \Closure|array|string $action)
static void put(string $uri, \Closure|array|string $action)
static void delete(string $uri, \Closure|array|string $action)
static void patch(string $uri, \Closure|array|string $action)
static void options(string $uri, \Closure|array|string $action)
static void match(array|string $methods, string $uri, \Closure|array|string $action)
static void resource(string $name, string $controller, array $options = [])
static void group(array $attributes, \Closure $callback)
```

定义路由

- 闭包路由：由闭包函数响应该路由请求

```
// 匹配GET请求
Route::get('/', function () {
    return view('welcome');
});
// 匹配任意HTTP请求
Route::any('/list', function() {
    return 'every http request type!';
});
```

- 控制器动作：由控制器的指定动作响应该路由请求

```
// 由IndexController的index动作响应请求
Route::get('', 'IndexController@index');
```

- 资源控制器路由：创建RESTful风格的路由，由资源控制器响应路由请求

```
// RESTful路由
Route::resource('photos', 'PhotoController');
```

路由参数

有些情况下，需要在路由URL中给出参数，现在讨论如何把给定参数和响应代码绑定。

- 必需参数：必须在URL给定参数，否则路由解析错误

```
// 必需参数：必须在URL中给定参数
Route::get('user/{id}', function ($id) {
    return 'User '.$id;
});
Route::get('user/{id}', 'IndexController@index');
```

- 可选参数：在URL中可以给定参数，也可以不给定参数（若无参数，使用默认值参数）

✓ 在控制器方法中要给出参数的默认值。

```
// 可选参数：可以给出参数，也可以不给（使用默认值）
Route::get('user/{name?}', function ($name = 'John') {
    return $name;
});
Route::get('user/{name?}', 'IndexController@index');
```

命名路由

命名路由为**生成 URL** 或**重定向**提供了便利。实现也很简单，在路由定义之后使用 name 方法链的方式来实现。

■ 定义命名路由：

✓ 使用 name( )方法链实现

✓ 使用 “as” 键实现

■ 使用命名路由：

✓ 生成URL

✓ 重定向

```php
Route::get('/', function() {
    return view('welcome');
})->name('root');

Route::get('news', [
    'as'    =>  'news',
    'uses'   =>  function() {
        return 'news route';
    }
]);

$url = route('news');
redirect()->route('news');
```

路由群组

路由群组允许我们在多个路由中共享路由属性，比如中间件和命名空间等，而不必为每一个路由单独定义属性。共享属性以数组的形式作为第一个参数被传递给 **Route::group** 方法。

- 共享中间件
- 路由前缀
- 共享命名空间

✓ 默认使用的控制器命名空间为**App\Http\Controllers**

✓ 可以指定子命名空间

```php
Route::group(['middleware' => 'auth'], function () {
    Route::get('/', function () {
        // 使用 Auth 中间件
    });

    Route::get('user/profile', function () {
        // 使用 Auth 中间件
    });
});
```

```php
Route::group(['prefix' => 'admin'], function () {
    Route::get('users', function () {
        // 匹配 "/admin/users" URL
    });
});
```

```php
Route::group(['namespace' => 'Admin'], function(){
    // 控制器在 "App\Http\Controllers\Admin" 命名空间下
    Route::group(['namespace' => 'User'], function(){
        // 控制器在 "App\Http\Controllers\Admin\User" 命名空间下
    });
});
```

路由相关artisan命令

- 查看当前已经定义的所有路由：php artisan **route:list**
- 路由缓存：php artisan **route:cache**
  - ✓ 使用路由缓存将会极大减少注册所有应用路由所花费的时间开销，在某些案例中，路由注册速度甚至能提高100倍!
  - ✓ 路由缓存不会作用于基于闭包的路由。要使用路由缓存，必须将闭包路由转化为控制器路由。
- 清除路由缓存：php artisan **route:clear**

基本控制器

■ 创建控制器

- ✓ 控制器目录：/app/Http/Controllers 目录下，可以把控制器类组织到不同的子目录中
- ✓ 控制器类文件：控制器类名.php
- ✓ 控制器类名：控制器类名不是必须带有 Controller 后缀，只要符合PHP规范即可
- ✓ 命名空间：每一个控制器类均要指明其所在的命名空间，命名空间要符合 PSR-4 规范；根命名空间为 \App
- ✓ 继承父类：控制器类一般要继承 Illuminate\Routing\Controller 类或其子类；若不继承该类，则不能直接使用Laravel提供的常用控制器方法
- ✓ 实例：创建UserController类

基本控制器

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

use Illuminate\Routing\Controller as BaseController;

class UserController extends BaseController {
    /**
     * 为指定用户显示详情
     *
     * @param int $id
     * @return Response
     */
    public function showProfile($id) {
        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}
```

使用命令行创建控制器

控制器类可以纯手动创建，也可以使用Artisan命令行工具创建，使用命令行创建控制器更为简单和方便。

■ Artisan命令创建控制器

✓ 创建空控制器： php artisan make:controller NewsController

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

use Illuminate\Http\Request;

class NewsController extends Controller {
    //
}
```

✓ 创建资源控制器：

php artisan make:controller NewsController -r

资源控制器

- 资源控制器主要为了响应 **RESTful** 风格的请求而产生的，只需要在路由中定义资源控制器路由，Laravel就自动把不同请求类型的URL匹配资源控制器的不同方法。

方法	路径	动作	路由名称
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

资源控制器

- 资源控制器主要为了响应 **RESTful** 风格的请求而产生的，只需要在路由中定义资源控制器路由，Laravel就自动把不同请求类型的URL匹配资源控制器的不同方法。
- 定义资源控制器路由：**Route::resource( );**
- 补充资源控制器路由：如果有必要在默认资源路由之外添加额外的路由到资源控制器，应该在调用 Route::resource 之前定义这些路由；否则，通过 resource 方法定义的路由可能无意中优先于补充的额外路由：

```
Route::get('photos/popular', 'PhotoController@method');
Route::resource('photos', 'PhotoController');
```

控制器相关artisan命令

- 创建空控制器：php artisan make:controller 控制器名

  php artisan make:controller MsgController
- 创建资源控制器：php artisan make:controller 控制器名 -r

  php artisan make:controller PhotoController -r
- 指定位置创建控制器：php artisan make:controller 完整类名（带命名空间）

  php artisan make:controller Admin\IndexController

控制器类相关方法

- Illuminate\Routing\Controller （Class）
- Illuminate\Foundation\Bus\DispatchesJobs （Trait）
- Illuminate\Foundation\Validation\ValidatesRequests （Trait）
- Illuminate\Foundation\Auth\Access\AuthorizesRequests （Trait）

CSRF保护

跨站请求伪造（CSRF）是一种通过伪装授权用户的请求来利用授信网站的恶意漏洞。Laravel 让应用避免遭到跨站请求伪造攻击变得简单。

Laravel 自动为每一个被应用管理的有效用户会话生成一个 CSRF “令牌”，该令牌用于验证授权用户和发起请求者是否是同一个人。

任何时候在 Laravel 应用中定义 HTML 表单，都需要在表单中引入 CSRF 令牌字段，这样 CSRF 保护中间件才能够正常验证请求。

```
<form method="POST" action="/profile">
    {{ csrf_field() }}
    ...
</form>
```

■ 在HTML表单中添加CSRF隐藏域：

✓ **csrf_token**（）函数：获取当前 CSRF 令牌的值

✓ **csrf_field**（）函数：生成一个包含 CSRF 令牌值的 HTML 隐藏域

表单伪造

- 默认情况下，在HTML中不能发送 PUT、DELETE、PATCH请求，Laravel框架提供了表单伪造机制，允许HTML表单模拟发送这类请求。

✓ 直接使用表单隐藏域：

```
<form action="/foo/bar" method="POST">
    <input type="hidden" name="_method" value="PUT">
</form>
```

✓ 在视图中使用Laravel提供的辅助函数： **method_field( )**

```
<form action="/news" method="post">
    {{ method_field('DELETE') }}
</form>
```

感谢聆听！

THANK YOU FOR YOUR ATTENTION