## Laravel支持多种请求方式

get、put、post等

|  |
| --- |
| <form method=’PUT’ action=’’>  <input type=’hidden’ name=’\_method’ value=’PUT’>//第一种方式  {{ method\_field(‘PUT’) }}//第二种方式  </form> |

## 快速向数据库填充测试数据

使用database/factories/ModelFactory.php

* 效仿例子即可，此处使用了faker，详情：<https://github.com/fzaninotto/Faker>
* return 数组之后，在tinker里面，执行：

|  |
| --- |
| factory（App\Post::class，20）->make()//仅在当前页面显示，不插入进数据库  factory（App\Post::class，20）->create()//插入到数据库  factory（）第二个参数是需要插入多少条记录 |

## 辅助方法（仅仅是laravel提供的）

str\_limit($value, $limit = 100, $end = '...')//限制字符串显示数量，$limit最大显示的字符数，$end 超出显示范围显示的内容

request（）返回当前请求实例,返回一个对象

request()->all() 返回请求传递过来的数组；

request(‘title’) 或者获取一个输入项

request([‘title­’,’content’]) 获取提交过来的两个字段的值

## 依赖注入

定义路由Route::put('posts/{post}','PostController@update');

//红色部分的post应该对应。

在控制器中就可以将模型注入：

|  |
| --- |
| public function update(Request $request,Post $post)  {  //$post可以通过注入，直接使用，它即是通过路由传入的那个id的Post实例,省去了find一步。  if ($post->update(\request()->all())) {  return redirect('/posts/'.$post->id);  } else {  echo '文章更新失败';  }  //若只是用$request,应该如下：  $post = Post::find($request->id);  if ($post->update(\request()->all())) {  return redirect('/posts/'.$post->id);  } else {  echo '文章更新失败';  }  } |

## 查找一个门脸或注入类有哪些方法可以使用

* 1. 使用tinker

app(‘门脸名|注入类’) 返回对应的类文件名

注入类开头不能有反斜杠

* 1. 使用官方api接口文档

<http://d.laravel-china.org/api/5.4/Illuminate/Log/Writer.html>

* 1. 在这类文件里使用phpstrom的structure视图（左侧栏）

## Auth门面

在模型类里引入use Illuminate\Foundation\Auth\User as Authenticatable;

config/auth.php中：

|  |
| --- |
| 'defaults' => [  'guard' => 'web',  'passwords' => 'users', ],  'guards' => [  'web' => [  'driver' => 'session',使用session  'provider' => 'users',  ],  ]  'providers' => [  'users' => [  'driver' => 'eloquent',模型驱动  'model' => App\User::class,验证的模型是User  ],  ] |

应用：

|  |
| --- |
| $userInfo = request(['email','password']); $is\_remember = *boolval*(\request('is\_remember')); if (Auth::attempt($userInfo, $is\_remember)) {  return redirect('posts'); } else {  return redirect()->back()->withErrors('登录失败')->withInput(); }  Auth::attempt()第一个参数，是要验证的邮箱和密码的数组，第二个参数，是否需要记住密码  密码加密必须使用bcrypt（）  //下面两个方法可以取代读取session  Auth::id() 获取当前登录用户的id；  Auth::user()获取当前登录用户的对象实例，可以进一步获取Auth::user()->name等属性  Auth::check()判断用户是否已登录 |

## 用户授权

### Gate

gate相似于路由，Gates 提供了一个简单、基于闭包的方式来授权认证。

Gates 是用来决定用户是否授权访问给定的动作的闭包函数。App\Providers\AuthServiceProvider 类中使用 Gate facade 定义。

|  |
| --- |
| \*\*  \* 注册任意用户认证 、用户授权服务。  \*  \* @return void  \*/  public function boot()  {  $this->registerPolicies();  Gate::define('update-post', function ($user, $post) {  return $user->id == $post->user\_id;  });  或者直接关联策略  Gate::define('update-post', 'PostPolicy@update');  } |

使用 gates 授权动作：

|  |
| --- |
| //使用allows时，可以不用传当前用户给allows（），laravel会自动处理将当前用户传给Gate闭包函数  if (Gate::allows('update-post', $post)) {  // 当前用户可以更新 post...  }  if (Gate::denies('update-post', $post)) {  // 当前用户不能更新 post...  }  --------------------------------------------------------------------------------------------  若需要指定一个特定的用户是否可以访问某个动作，使用forUser()方法：  if (Gate::forUser($user)->allows('update-post', $post)) {  // 指定用户可以更新 post...  }  if (Gate::forUser($user)->denies('update-post', $post)) {  // 指定用户不能更新 post...  } |

### Policy

policy相似于控制器，在特定模型或者资源中组织授权逻辑的类

* + 1. 生成策略

|  |
| --- |
| php artisan make:policy PostPolicy  //生成的类包含基本的CURD策略方法  php artisan make:policy PostPolicy --model=Post |

* + 1. 注册策略

AuthServiceProvider中：

|  |
| --- |
| /\*\*  \* 应用的策略映射。  \*  \* @var array  \*/  protected $policies = [  Post::class => PostPolicy::class,  ]; |

* + 1. 编写策略

例如：返回true或false来指明用户是否授权更新给定的post。（此处依据是该文章的作者是不是当前用户）

|  |
| --- |
| //参数：当前认证通过的用户，授权更新的模型实例  public function update(User $user, Post $post)  {  return $user->id === $post->user\_id;  } |

不传模型实例：

|  |
| --- |
| public function create(User $user)  {  //  } |

* + 1. 策略过滤器

before方法会在策略的其他方法之前执行。

例如管理员授权：

|  |
| --- |
| public function before($user, $ability)  {  if ($user->isSuperAdmin()) {  return true;  }  } |

* + 1. 判断是否被授权

通过模型的方法：

Laravel 应用内置的 User 模型包含 2 个有用的方法来授权动作：can 和 cant

|  |
| --- |
| if ($user->can('update', $post)) {  //  } |

需要传入授权动作和模型实例；若该模型的策略被注册过，can方法自动调用策略方法并且返回bool值。若模型的策略没被注册过，尝试调用和动作名匹配的gate闭包。

不需要指定模型的动作：只需传个类名即可

|  |
| --- |
| if ($user->can('create', Post::class)) {  // 执行相关策略中的「create」方法...  } |

通过中间件：

Illuminate\Auth\Middleware\Authorize 中间件被指定到 App\Http\Kernel 类中 can 键上：

|  |
| --- |
| use App\Post;  Route::put('/post/{post}', function (Post $post) {  // 当前用户可以更新博客...  })->middleware('can:update,post'); |

can有两参数：需要授权的动作名称（update），要传给策略方法的路由参数（对应/post/{post}）

例如：/post/1, 通过中间件会把id=1的Post实例（隐式参数绑定）和update这个动作，传递到策略类里面，把$post实例传入并验证update方法是否被授权。

不需要指定模型：

|  |
| --- |
| Route::post('/post', function () {  // 当前用户可以创建博客...  })->middleware('can:create,App\Post'); |

因为不需要传递模型实例，中间件的第二个参数只需传入App\Post类即可

通过控制器的辅助函数：

App\Http\Controllers\Controller 基类的控制器提供了一个有用的 authorize 方法

|  |
| --- |
| public function update(Request $request, Post $post)  {  $this->authorize('update', $post);  // 当前用户可以更新博客...  } |

authorize()两个参数：动作，模型实例

不需要指定模型的：

$this->authorize('create', ‘Post::class’);

通过blade模板：

需要给不同权限的用户展示不同的页面，可以这样在前端判断：

|  |
| --- |
| @can('update', $post)  <!-- 当前用户可以更新博客 -->  @elsecan('create', $post)  <!-- 当前用户可以新建博客 -->  @endcan |