# Laravel

目录机构-------解析

**App -----**

**Http -----控制器**

**Exceptions----异常处理代码**

**Console ----- 控制台信息**

**User.php ------ 模型类**

**Config-------配置文件**

**Database -------模型的定义，配置**

Public -------静态资源文件（入口文件）

**Resources -----视图**

**Assets------未处理的css，js文件**

**Lang------语言包**

**Views -----试图实现（.php文件内html）**

**Routes ------路由**

**Web.php访问路由控制**

**Storage--------缓存**

**Tests -------单元测试**

# **常用命令**

1. **php artisan route:list查看已经注册的路由**
2. php artisan make:model Models/Article -m创建模型和迁移文件
3. php artisan make:controller UsersController --resource生产资源管理器
4. php artisan make:controllerStaticPagesController创建控制器
5. **php artisan make:policyUserPolicy授权策略类文件，用于管理用户模型的授权,所有生成的授权策略文件都会被放置在 app/Policies 文件夹下，例如用户a只能修改a自己的用户资料，防止用户修改路由参数，从而修改他人的资料**
6. **php artisan make:auth //脚手架 自动生产登陆注册页面，登陆后跳转home 主页注意路由问题**

# **Routes（路由详情）**

一．请求方式

1. Get：Route::get('/about', function () {

return 'about';

});

1. post：Route::post('/about', function () {

return 'about';

});

1. delete：Route::delete ('/about', function () {

return 'about';

});

1. patch：Route::patch('/about', function () {

return 'about';

});

二．路由传参

1. 必填参数

Get：Route::get('/about/{id}', function ($id) {

return 'id=>'.$id;

});

2. 选填参数

Get：Route::get('/about/{id?}', function ($id=’默认的参数’) {

return 'id=>'.$id;

});

3. 参数验证 id必须是数字

Route::*get*('/about/{id}', **function** ($id='默认的') {  
 **return** 'id=>'.$id;  
})->where('id','[0-9]+');

/\*基础控制器如何使用！\*/

4. Laravel 会自动解析定义在控制器方法（变量名匹配路由片段）中的 Eloquent 模型类型声明。在上面代码中，由于 show() 方法传参时声明了类型 —— Eloquent 模型 User，对应的变量名 $user 会匹配路由片段中的 {user}，这样，Laravel 会自动注入与请求 URI 中传入的 ID 对应的用户模型实例。

Route::get('/users/{user}', 'UsersController@show')->name('users.show');

public function show(User $user){

return view('users.show', compact('user'));

}

问题1：**前台最好不使用该方法用户有可能，篡改路由参数，适合后台做管理用户时使用，为解决用户修改路由参数需一下处理**

1. 用户只能编辑自己的资料
2. php artisan make:policyUserPolicy我们可以使用以下命令来生成一个名为UserPolicy 的授权策略类文件，用于管理用户模型的授权。
3. 所有生成的授权策略文件都会被放置在 app/Policies 文件夹下。
4. public function update(User $currentUser, User $user)

{

return $currentUser->id === $user->id;

}

1. update 方法接收两个参数，第一个参数默认为当前登录用户实例，第二个参数则为要进行授权的用户实例。当两个 id 相同时，则代表两个用户是相同用户，用户通过授权，可以接着进行下一个操作。如果 id 不相同的话，将抛出 403 异常信息来拒绝访问。
2. 配置
   1. 接下来我们还需要在AuthServiceProvider 类中对授权策略进行设置。AuthServiceProvider 包含了一个 policies 属性，该属性用于将各种模型对应到管理它们的授权策略上。我们需要为用户模型 User 指定授权策略 UserPolicy。
   2. 例如:

protected $policies = [

'App\Models' => 'App\Policies\ModelPolicy',

\App\Models\User::class=>\App\Policies\UserPolicy::class,

\App\Models\Status::class => \App\Policies\StatusPolicy::class,

];

1. 需要在用户访问的路由使用该方法$this->authorize('update', $user);验证用户是否修改路由参数，如果修改，页面会报错

# 数据库迁移

1. 原理：利用php生产mysql数据表
2. 迁移类目录 /database/migrations
3. 生产数据表的命令：php artisan migrate
4. php artisan make:model Models/Article -m//创建一个模型和他的迁移文件

# Controllers

1. 路由绑定控制器：

Route::get('member/info','MemberController@info');

Route::get('member/{id}',[

'uses'=>'MemberController@info',

'as'=>'cs/{id}'

])->where('id','[0-9]');

1. 多模块控制器

绑定方法：

Route::get('/admin','Admin\IndexController@index');

1. 资源控制器（用于注册实例类控制器路由增加删除修改）

命令生生成资源控制器：

php artisan make:controllerUserController–resource

资源管理器提供命名规范：

1. index ---资源列表
2. create---添加资源表单页面
3. store-----添加功能的实现
4. show-----查看单个实例
5. edit -----修改实例页面
6. update -----修改功能的实现
7. destore -----删除功能的实现

路由注册

Route::resource('/user','Admin\UserController', ['only' => ['show', 'update', 'edit']]);

视图传参：

$nam=’’;

$list=’’;

return view('\Admin\User\index',compact('list','name'));

视图{{$name}}绑定参数

1. 静态页面 控制器：

命令生静态页面 控制器：

php artisan make:controllerStaticPagesController

1. 数据验证：
   1. $this->validate(1,2) 第一个参数需要验证的数据，2规则
   2. $this->validate($request, [

'name' => 'required|max:50',

'email' => 'required|email|unique:users|max:255',

'password' => 'required|confirmed|min:6'

]);

* 1. 验证规则：
     1. Required必须
     2. 'min:3|max:50'长度
     3. Email邮箱
     4. unique:users数据库唯一性 这个例子是针对users表的
     5. confirmed重复一致 例如 password和password\_confirmation中password\_confirmation的验证规则就是 confirmed
     6. unique:users,name,' . Auth::id() —— unique 数据库唯一，在 users 数据表里，字段为 name，Auth::id() 指示将此 ID 排除在外。
  2. 表单验证在视图中 使用 {{ csrf\_field() }}会帮我们自动生成一个 隐藏域的表单<input type="hidden" name="\_token" value="fhcxqT67dNowMoWsAHGGPJOAWJn8x5R5ctSwZrAq">

1. 回话
   1. Auth::attempt(1,2) 用户登陆函数自动查询sql判断账号密码是否错误，1用户信息2为是否开启记住我功能 coock功能
   2. session()->flash('success', '欢迎，您将在这里开启一段新的旅程~');使用flash发方法定义键名为success的回话
   3. Auth::check()判断用户是否登陆 返回true和false
2. http请求
   1. $request->has('name')**要判断请求是否存在某个值，可以使用 has 方法。如果请求中存在该值，has 方法就会返回 true**
3. 重定向
   1. redirect()->route('users.show', [$user]);**这里的$user为用户的实例 router会自动获取user的id**相当于redirect()->route('users.show', [$user->id])***;这是一个约定***
   2. redirect()->back();返回前一个页面
4. 消息提示
   1. danger', 'warning', 'success', 'info’在视图中有不同的样式

@foreach (['danger', 'warning', 'success', 'info'] as $msg)

@if(session()->has($msg))

<div class="flash-message">

<p class="alert alert-{{ $msg }}">

{{ session()->get($msg) }}

</p>

</div>

@endif

@endforeach

1. 自动登陆
   1. Auth::login($user);//注册后自动登陆
   2. Auth::logout();用户退出
2. **中间件 所有的中间件存在放 Http/Middleware 目录下，中间件用于验证用户是否登陆等**
3. **权限验证 session登陆过滤 改条件放在 父类中**

public function \_\_construct()

{

$this->middleware('auth', [

'except' => ['show', 'create', 'store']

]);

}

middleware 方法，该方法接收两个参数，**第一个为中间件的名称**，**第二个为要进行过滤的动作**。'except'**除了此处指定的动作以外，所有其他动作都必须登录用户才能访问**，类似于黑名单的过滤机制。相反的还有 only 白名单方法，将只过滤指定动作。我们提倡在控制器 Auth 中间件使用中，首选 except 方法，这样的话，当你新增一个控制器方法时，默认是安全的，此为最佳实践。

1. 数据分页
   1. 模型的方法 $users = User::paginate(1);
   2. 在模板中调用 {{ $users->render() }}方法render获取分页链接
2. 加密 如果要使用lavarel内置的加密方式必须 **正确配置 config/app.php 配置文件中*的* key 选项** ，命令 php artisan key:generate会自动它会使用 PHP 的安全随机字节生成器来构建密钥加密解密方式如下：
   1. use Illuminate\Support\Facades\Crypt;
   2. $encrypted = Crypt::encryptString('Hello world.');
   3. $decrypted = Crypt::decryptString($encrypted);
3. 获取用户实例 Auth::user()
4. 验证码：
   1. 安装拓展composer require "mews/captcha:~2.0"
   2. 生产配置文件 php artisan vendor:publish 选择confg 那个选项
   3. 验证码连接 {{ captcha\_src('flat') }}
   4. Js 动态切换图片 onclick="this.src='/captcha/flat?'+Math.random()"
   5. 后台验证return Validator::make($data, [

'captcha' => 'required|captcha',

], [

'captcha.required' => '验证码不能为空',

'captcha.captcha' => '请输入正确的验证码',

]);

1. 数据验证
   1. 验证类：Http/Requests 存放在该文件下

验证规则：public function rules()

{

return [

'name'=>'required|between:3,25|regex:/^[A-Za-z0-9\-\\_]+$/|unique:users,name,' . Auth::id(),

'email' => 'required|email',

'introduction' => 'max:80',

];

}

* 1. 提示信息

public function messages()

{

return [

'name.unique' => '用户名已被占用，请重新填写',

'name.regex' => '用户名只支持英文、数字、横杆和下划线。',

'name.between' => '用户名必须介于 3 - 25 个字符之间。',

'name.required' => '用户名不能为空。',

];

}

* 1. 异常捕获：
     1. 在模板中 会有对象 $errors ，包含错误信息
     2. 使用$errors->all();获取错误信息提示，该方法返回一个数组

1. 文件上传
   1. 获取文件：
      1. $file = $request->file('avatar');
      2. $file = $request->avatar; ***可读性更高***
   2. 文件验证 验证规则：

'avatar'=> 'mimes:jpeg,bmp,png,gif|dimensions:min\_width=200,min\_height=200',

'avatar.mimes' =>'头像必须是 jpeg, bmp, png, gif 格式的图片',

'avatar.dimensions' => '图片的清晰度不够，宽和高需要 200px 以上',

* 1. 图片处理
     1. 安装插件 composer require intervention/image
     2. 生成配置文件：php artisan vendor:publish --provider="Intervention\Image\ImageServiceProviderLaravel5"
     3. 调用类 use Image;

public function reduceSize($file\_path, $max\_width)

{

// 先实例化，传参是文件的磁盘物理路径

$image = Image::make($file\_path);

// 进行大小调整的操作

$image->resize($max\_width, null, function ($constraint) {

// 设定宽度是 $max\_width，高度等比例双方缩放

$constraint->aspectRatio();

// 防止裁图时图片尺寸变大

$constraint->upsize();

});

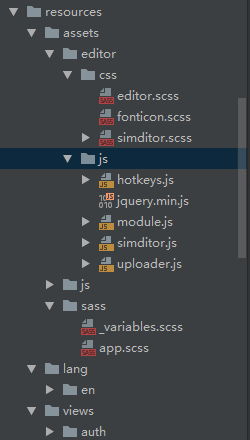
// 对图片修改后进行保存

$image->save();

}

1. 代码生成器
   1. 安装拓展包composer require "summerblue/generator:~0.5" –dev
   2. 生成代码命令 php artisan make:scaffold Topic --schema="title:string:index,body:text,user\_id:integer:unsigned:index,category\_id:integer:unsigned:index,reply\_count:integer:unsigned:default(0),view\_count:integer:unsigned:default(0),last\_reply\_user\_id:integer:unsigned:default(0),order:integer:unsigned:default(0),excerpt:text,slug:string:nullable"
   3. 运行结果：创建话题的数据库迁移文件 —— 2017\_09\_26\_111713\_create\_topics\_table.php；
   4. 创建话题数据工厂文件 —— TopicFactory.php；
   5. 创建话题数据填充文件 —— TopicsTableSeeder.php；
   6. 创建模型基类文件 —— Model.php， 并创建话题数据模型；
   7. 创建话题控制器 —— TopicsController.php；
   8. 创建表单请求的基类文件 —— Request.php，并创建话题表单请求验证类；
   9. 创建话题模型事件监控器 TopicObserver 并在 AppServiceProvider 中注册；
   10. 创建授权策略基类文件 —— Policy.php，同时创建话题授权类，并在 AuthServiceProvider 中注册；
   11. 在 web.php 中更新路由，新增话题相关的资源路由新建符合资源控制器要求的三个话题视图文件，并存放于 resources/views/topics 目录中；
   12. 执行了数据库迁移命令 artisan migrate；
   13. 因此次操作新建了多个文件，最终执行 composer dump-autoload 来生成 classmap
2. 调试工具
   1. 安装命令：composer require "barryvdh/laravel-debugbar:~3.1" –dev
   2. 生成配置文件：php artisan vendor:publish --provider="Barryvdh\Debugbar\ServiceProvider"
   3. 修改其中的配置 'enabled' => env('APP\_DEBUG', false),
3. 类库 active（路由判断的类库） 的安装
   1. composer require "hieu-le/active:~3.5"
   2. 常用方法
      1. if\_route() - 判断当前对应的路由是否是指定的路由；
      2. if\_route\_param() - 判断当前的 url 有无指定的路由参数。
      3. if\_query() - 判断指定的 GET 变量是否符合设置的值；
      4. if\_uri() - 判断当前的 url 是否满足指定的 url；
      5. if\_route\_pattern() - 判断当前的路由是否包含指定的字符；
      6. if\_uri\_pattern() - 判断当前的 url 是否含有指定的字符；
4. 富文本编辑器
   1. 下载地址<https://github.com/mycolorway/simditor/releases/download/v2.3.6/simditor-2.3.6>.

Zip

* 1. 导入其中css和js目录如下 
  2. 修改webpack.mix.js 打包位置将css,js目录配置到打包目录

mix.js('resources/assets/js/app.js', 'public/js')

.sass('resources/assets/sass/app.scss', 'public/css')

.copyDirectory('resources/assets/editor/js', 'public/js')

.copyDirectory('resources/assets/editor/css', 'public/css')

* 1. 使用案例：
     1. 导入的css以及js
     2. js/module.js
     3. js/hotkeys.js
     4. js/uploader.js
     5. js/simditor.js
     6. css/simditor.css
     7. 使用方法：

图片上传的路径配置：

$(document).ready(function(){

var editor = new Simditor({

textarea: $('#editor'),

upload: {

url: '{{ route('topics.upload\_image') }}', //—— 处理上传图片的 URL；

params: { \_token: '{{ csrf\_token() }}' }, //表单提交的参数，Laravel 的 POST 请求必须带防止 CSRF 跨站请求伪造的 \_token 参数；

fileKey: 'upload\_file', //是服务器端获取图片的键值，我们设置为 upload\_file，（表单名）

connectionCount: 3,//最多只能同时上传 3 张图片；

leaveConfirm: '文件上传中，关闭此页面将取消上传。'

},

pasteImage: true,//设定是否支持图片黏贴上传，这里我们使用 true 进行开启；

});

});

1. 防xss攻击类库 [HTMLPurifier for Laravel](https://github.com/mewebstudio/Purifier)
   1. 安装命令 composer require "mews/purifier:~2.0"
   2. 配置生成

# Models

1. **在定义了关联视图时一定要使用with（）去预加载关联模型否者将产生大量的sql查询**
   1. **例 $topics = Topic::with('user', 'category')->paginate();**
   2. **用户with 去预加载 user和'category'模型**
2. 关联视图
   1. 拥有多个public function statuses()

{

return $this->hasMany(Status::class);

}

* 1. 1对1

public function user()

{

return $this->belongsTo(User::class);

}

* 1. 多对多

public function followers()

{

return $this->belongsToMany(User::Class, 'followers', 'user\_id', 'follower\_id');

}

public function followings()

{

return $this->belongsToMany(User::Class, 'followers', 'follower\_id', 'user\_id');

}

* 1. //判断用户时候相互关注

public function isFollowing($user\_id)

{

return $this->followings->contains($user\_id);

}

备注：这里使用followings而不是followings()的原因就是followings返回的是一个Collection集合，在该集合下可以使用contains方法判断集合内是否存在这个id

1. 关联写入：sync()
2. 关联删除：detach()
3. 授权 protected $fillable = ['content']; //允许模型更新的数据
4. 定义表名称 protected $table = 'my\_articles';
5. 插入数据
   1. User::create([

'name' => $request->name,

'email' => $request->email,

'password' =>bcrypt($request->password),

]);

* 1. 使用关联视图会自动关联用户idAuth::user()->statuses()->create([

'content' => $request['content']

]);

1. Sql：查询
   1. Auth::user()->pluck('id') 获取数据单一字段 返回：Illuminate\Support\Collection::\_\_set\_state(array( 'items' => array ( 0 => 1, 1 => 2, ), ))
2. 本地作用域 相当于一个约顶，相同的复用排序，封装成发放
   1. 约定规则：方法名加前缀scope

例如public function scopeOrdered($query)

{

return $query->orderBy('order', 'desc');

}

* 1. 控制器中调用规则 model->rdered(); 不用加前缀

1. 属性写入 fill
   1. 例如$topic->fill($request->all());
   2. $topic->user\_id = Auth::id();
   3. $topic->excerpt = '';
   4. $topic->save();

# 视图

1. 模板继承

@extends('layouts.default')继承 layouts文件夹下的default模板（只要名称。不要模板后缀）

@yield（1，2）定义一个父模板的 模块名1默认为2

@section（1）

在这里编写自己的模块内容

@stop

1. 在模板中使用函数：

{{route('help')}}输出help的完整路由

//局部模板传入参数

@include('shared.\_user\_info', ['user' => $user])

1. 表单数据保持
   1. {{ old('name') }}
   2. <input type="text" name="name" class="form-control" value="{{ old('name') }}">页面刷新但是这个表单的数据依然不变
2. 策略验证 @can('destroy', $user) html @endcan //直接在模板中验证用户是否有权看到这部分代码

# Eloquent数据交互

1. 命令行 php artisan tinker可以直接在命令行访问类下面的方法
   1. 例如：

App\Models\User::create(['name'=>'Aufree','email'=>'aufree@yousails.com','password'=>bcrypt('password')])通过模型创建一个用户

use App\Models\User引用类 **当使用了 use 对类进行引用之后，便可以对类进行直接调用。为了让接下来的命令可读性更高，我们使用 use 来引用 App\Models\User Eloquent 模型类**

User::find(1)查找一个 id 为 1 的用户

User::first()查找用户表中的首个用户

User::all()用 all 方法取出所有的用户数据

**修改用户**

1.

$user = User::find(1)

$user->update(['name'=>1])

2.

$user = User::find(1)

$user->name = 'Summer'

$user->save()

# 微博测试项目

1. 安装依赖（自带依赖 ）

bootstrap-sass —— Bootstrap NPM 扩展包；

jquery —— jQuery NPM 扩展包；

laravel-mix —— 由 Laravel 官方提供的静态资源管理工具；

vue —— VUE.js 前端框架；

1. 安装命令:
   1. yarn install --no-bin-links
   2. yarn add cross-env
   3. npm run watch-poll（监听resources/assets/sass/app.scss的变化，将其编译到public/css下）