# Laravel环境搭建

## Laravel框架介绍

Laravel 的官网地址为：<https://laravel.com/>

英文文档地址：<https://laravel.com/docs>

中文文档地址：<https://learnku.com/docs/laravel>

包管理地址：<https://packagist.org/>

Laravel包地址：

<https://packagist.org/packages/laravel/framework>

Laravel成长之路：

<https://learnku.com/docs/study-laravel/1>

Laravel社区Wiki：<https://learnku.com/laravel/wikis>

本文档适用的版本为：Laravel 7.10.\* 版本

<https://packagist.org/packages/laravel/framework#v7.10.0>

## Laravel框架安装

### 安装条件

按照文档的说明，在安装 Laravel 之前，需要开启如下 PHP 相关配置：

* PHP >= 7.2.0
* BCMath PHP Extension
* Ctype PHP Extension
* JSON PHP Extension
* Mbstring PHP Extension
* OpenSSL PHP Extension
* PDO PHP Extension
* Tokenizer PHP Extension
* XML PHP Extension

在Windows系统中，我们使用PHPStudy，在Linux系统中，我们使用宝塔面板。

### 安装过程

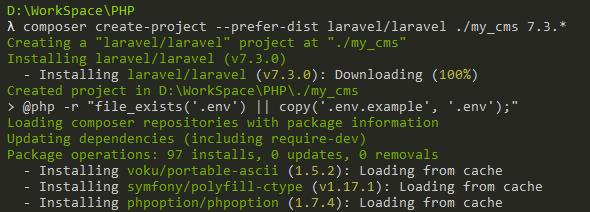
使用composer来安装Laravel框架，代码如下：

|  |
| --- |
| 在当前目录下安装最新版 |
| composer create-project --prefer-dist laravel/laravel ./ |
| 在当前目录的指定目录下安装指定版本 |
| composer create-project --prefer-dist laravel/laravel ./my\_cms 7.3.\* |



GitHub地址：<https://github.com/laravel/laravel/releases/tag/v7.3.0>

过程截图如下：

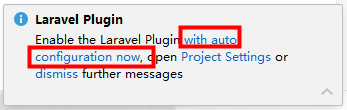
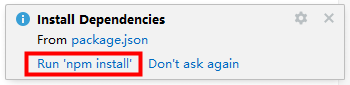


安装完成后，出现如下代码：

|  |
| --- |
| Package manifest generated successfully.  > @php artisan key:generate --ansi  Application key set successfully. |

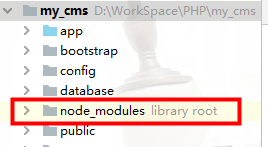
表示安装成功。

然后用PHPStorm打开项目，此时IDE提示安装插件，如下所示：

为了方便开发，这两个提示安装的插件最好都安装上。

安装Laravel Plugin才能正常使用代码跟踪功能，运行npm install会在项目下多出node\_modules，如下图所示：



一般我的开发流程：

|  |
| --- |
| 自己定义SQL文件来创建表结构。        不使用关联查询来查询数据库。 |

创建表的SQL语句为：

|  |
| --- |
| # 创建数据库  drop database if exists `my\_cms`;  create database `my\_cms` default character set utf8mb4 collate utf8mb4\_general\_ci;  # 创建 user 表结构  drop table if exists `user`;  create table `user`  (  `id` int(10) unsigned not null auto\_increment comment '用户ID',  `username` varchar(50) not null default '' comment '用户名',  `email` varchar(50) not null default '' comment '用户邮箱',  `email\_is\_verified` tinyint(1) unsigned not null default 0 comment '邮箱是否校验: 0 未校验 1 已校验',  `email\_verified\_at` int(10) unsigned not null default 0 comment '邮箱校验时间戳',  `password` char(32) not null default '' comment '用户密码, MD5 固定长度 32 位',  `mobile` varchar(30) not null default '' comment '用户手机号',  `fax` varchar(30) not null default '' comment '用户传真号',  `wechat` varchar(30) not null default '' comment '用户微信号',  `gender` tinyint(1) unsigned not null default 0 comment '用户性别: 0 男 1 女',  `address` varchar(50) not null default '' comment '用户住址',  `reg\_date` int(10) unsigned not null default 0 comment '用户注册时间戳',  `last\_login\_date` int(10) unsigned not null default 0 comment '用户最后登录时间戳',  primary key (id)  );  # 创建 category 表结构  drop table if exists `category`;  create table `category`  (  `id` smallint(6) unsigned not null auto\_increment comment '分类ID',  `reid` smallint(6) unsigned not null default 0 comment '父类ID',  `name` varchar(250) not null default '' comment '分类名称',  `pinyin` varchar(250) not null default '' comment '分类拼音',  `title` varchar(250) not null default '' comment '分类标题',  `keywords` varchar(250) not null default '' comment '分类关键词',  `description` varchar(250) not null default '' comment '分类描述',  `introduction` varchar(250) not null default '' comment '分类介绍',  `is\_show` tinyint(1) unsigned not null default 1 comment '是否展示: 1 展示 0 不展示',  `litpic` varchar(100) not null default '' comment '分类缩略图',  `backpic` varchar(100) not null default '' comment '分类背景图',  `content` text comment '分类正文',  primary key (id) comment '分类主键'  );  # 创建 article 文章表  drop table if exists `article`;  create table `article`  (  `id` int(10) unsigned not null auto\_increment comment '文章ID',  `cate\_id` smallint(6) unsigned not null default 0 comment '分类ID',  `title` varchar(250) not null default '' comment '文章标题',  `keywords` varchar(250) not null default '' comment '文章关键词',  `description` varchar(250) not null default '' comment '文章描述',  `pubdate` int(10) unsigned not null default 0 comment '文章发布时间戳',  `click` mediumint(8) unsigned not null default 0 comment '文章点击量',  `writer` varchar(50) not null default '' comment '文章作者',  `source` varchar(50) not null default '' comment '文章来源',  `is\_show` tinyint(1) unsigned not null default 1 comment '是否展示: 1 展示 0 不展示',  `litpic` varchar(100) not null default '' comment '文章缩略图',  `content` text comment '文章正文',  primary key (id) comment '文章主键'  );  # 创建 tag 表结构  drop table if exists `tag`;  create table `tag`  (  `id` int(10) unsigned not null auto\_increment comment '标签ID',  `name` varchar(250) not null default '' comment '标签名称',  `pinyin` varchar(250) not null default '' comment '标签拼音',  `title` varchar(250) not null default '' comment '标签标题',  `keywords` varchar(250) not null default '' comment '标签关键词',  `description` varchar(250) not null default '' comment '标签描述',  `introduction` varchar(250) not null default '' comment '标签简介',  `pubdate` int(10) unsigned not null default 0 comment '标签发布时间戳',  `jump\_url` varchar(50) not null default '' comment '标签跳转地址',  primary key (id) comment '标签主键',  key `pinyin` (`pinyin`) comment '标签索引键'  );  # 创建 article 和 category、tag 的中间表  drop table if exists `article\_tag\_category`;  create table `article\_tag\_category`  (  `article\_id` int(10) unsigned not null default 0 comment '文章ID',  `cate\_id` smallint(5) unsigned not null default 0 comment '分类ID',  `tag\_id` int(10) unsigned not null default 0 comment '标签ID'  );  # 创建 topic 表结构  drop table if exists `topic`;  create table `topic`  (  `id` int(10) unsigned not null auto\_increment comment '专题ID',  `aids` varchar(250) not null default '' comment '专题相关联的文章ID',  `name` varchar(250) not null default '' comment '专题名称',  `pinyin` varchar(250) not null default '' comment '专题拼音',  `title` varchar(250) not null default '' comment '专题标题',  `keywords` varchar(250) not null default '' comment '专题关键词',  `description` varchar(250) not null default '' comment '专题描述',  `introduction` varchar(250) not null default '' comment '专题简介',  `pubdate` int(10) unsigned not null default 0 comment '专题发布时间戳',  `jump\_url` varchar(50) not null default '' comment '专题跳转地址',  `writer` varchar(50) not null default '' comment '专题作者',  `source` varchar(50) not null default '' comment '专题来源',  `is\_show` tinyint(1) unsigned not null default 1 comment '是否展示: 1 展示 0 不展示',  `click` mediumint(8) unsigned not null default 0 comment '专题点击量',  `litpic` varchar(100) not null default '' comment '专题缩略图',  `content` text comment '专题正文',  primary key (id) comment '专题主键',  key `pinyin` (`pinyin`) comment '专题索引建'  );  # 创建 search 表结构  drop table if exists `search`;  create table `search`  (  `id` int(10) unsigned not null auto\_increment comment '搜索ID',  `name` varchar(250) not null default '' comment '搜索名称',  `pinyin` varchar(250) not null default '' comment '搜索拼音',  `title` varchar(250) not null default '' comment '搜索标题',  `keywords` varchar(250) not null default '' comment '搜索关键词',  `description` varchar(250) not null default '' comment '搜索描述',  `introduction` varchar(250) not null default '' comment '搜索简介',  `pubdate` int(10) unsigned not null default 0 comment '搜索发布时间戳',  `jump\_url` varchar(50) not null default '' comment '搜索跳转地址',  primary key (id) comment '搜索主键',  key `pinyin` (`pinyin`) comment '搜索索引键'  );  # 创建 sysconf 表结构  drop table if exists `sysconf`;  create table `sysconf`  (  `id` int(10) unsigned not null auto\_increment comment '配置项ID',  `varname` varchar(50) not null default '' comment '配置项变量名',  `value` varchar(50) not null default '' comment '配置项变量值',  `info` varchar(100) not null default '' comment '配置项文字说明',  primary key (id) comment '配置项主键'  ); |

附件：



### 网站示例

Laravel安装完成之后，打开 hosts 文件，增加如下域名解析代码：

|  |
| --- |
| 127.0.0.1 www.mycms.flow |

然后打开Apache配置文件，增加如下域名绑定文件：

|  |
| --- |
| <VirtualHost \*:80>  DocumentRoot "D:/WorkSpace/PHP/MyCMS/public"  ServerName www.mycms.flow  <Directory "D:/WorkSpace/PHP/MyCMS/public">  AllowOverride all  Order allow,deny  Allow from all  </Directory>  </VirtualHost> |

注意，在配置文件中文档地址要指向到Laravel程序的public目录中才正确。

重启 Apache 服务器，在浏览器输入<http://www.mycms.flow/> 出现如下信息：



说明 Laravel 安装成功。

### 查看版本

方法一：查看 Laravel 版本：进入 D:\WorkSpace\PHP\MyCMS目录，在该目录下有一个 artisan 文件，打开终端，在终端输入如下代码：

|  |
| --- |
| D:\WorkSpace\PHP\MyCMS  λ php artisan --version  Laravel Framework 6.3.0 |

方法二：打开 vendor/laravel/framework/src/Illuminate/Foundation/

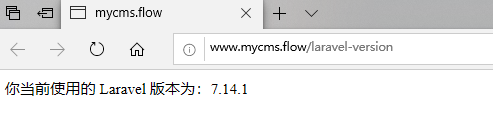
Application.php 文件进行查看，如下代码所示：

|  |
| --- |
| class Application extends Container implements ApplicationContract, HttpKernelInterface  {  /\*\*  \* The Laravel framework version.  \*  \* @var string  \*/  const VERSION = '5.8.36'; |

方法三：通过路由显示版本信息：在 routes/web.php 文件中，增加如下代码：

|  |
| --- |
| // 打印 Laravel 版本信息  Route::get('/laravel-version', function () {  $lara\_ver = app();  return "你当前使用的 Laravel 版本为：" . $lara\_ver::VERSION;  }); |

然后打开浏览器，输入<http://www.mycms.flow/laravel-version>显示结果如下图所示：



以上就是查看 Laravel 版本的三种方法了。

### LaraDock

在线文档：<https://laradock-docs.linganmin.cn/>

一套完整的基于Docker的PHP开发环境。包含了预先打包的Docker镜像，所有预先配置都是为了提供一个完美的PHP开发环境。

Laradock是在Laravel社区众所周知的，因为这个项目最开始只关注在Docker上运行的Laravel项目。后来，由于PHP社区的大量使用，它开始支持比如Symfony、CodeIgniter、WordPress、Drupal等其他的PHP项目。

#### LaraDock入门配置

文档介绍：

<https://learnku.com/articles/15197/laradock-portal-configuration>

### 目录结构

|  |
| --- |
| 这里以Laravel7.14.1为例说明 |
| ├─app  │ ├─Console  │ ├─Exceptions 异常处理  │ ├─Http Http请求  │ │ ├─Controllers 控制器目录  │ │ └─Middleware 中间件目录  │ ├─Providers  │ └─User.php User模型文件  ├─bootstrap  │ └─cache  ├─config 项目配置文件目录  ├─database 数据库配置目录  │ ├─factories 工厂类相关操作  │ ├─migrations 迁移相关操作  │ └─seeds 数据填充相关种子  ├─public 网站访问绑定目录  ├─resources  │ ├─js 存放js文件的目录  │ ├─lang 语言包目录  │ ├─sass  │ └─views 视图文件存放目录  │ └─user user相关的视图文件夹  ├─routes 路由文件夹  ├─storage 存储目录  │ ├─app  │ ├─debugbar  │ ├─framework  │ │ ├─cache  │ │ ├─sessions  │ │ ├─testing  │ │ └─views  │ └─logs  ├─tests 测试存放目录  │ ├─Feature  │ └─Unit  ├─vendor 框架核心库存放目录  ├─.env 运行环境配置文件  ├─artisan PHP命令行工具  ├─composer.json 配置PHP依赖包  ├─package.json 配置Node.js依赖包  └─phpunit.xml PHP测试相关 |

如果资源是放在storage目录下，则必须执行：php artisan storage:link，创建快捷方式，然后通过快捷方式来访问。

## Laravel开发环境

### PHPStorm 开发工具

我们使用PHPStorm作为Laravel框架的开发工具。

### PHPStorm 安装插件

在使用PHPStorm时，为了更好的开发Laravel，推荐安装以下插件：

|  |  |
| --- | --- |
| Laravel | Daniel Espendiller |
| LaravelStorm | SmartBit8 |
| Laravel Snippets | Alicannklc |
| Blade | JetBrains |

### PHPStorm 导入Laravel项目

启动 PHPStorm IDE → File → Open... → MyCMS → Attach 进行开发。

### 安装laravel-ide-helper代码提示库

库文件地址：

<https://packagist.org/packages/barryvdh/laravel-ide-helper>

步骤一：打开项目目录下的composer.json文件，给require变量增加行：

|  |
| --- |
| "barryvdh/laravel-ide-helper": "dev-master" |

步骤二：打开\config\app.php文件，给providers变量增加行：

|  |
| --- |
| Barryvdh\LaravelIdeHelper\IdeHelperServiceProvider::class, |

步骤三：在项目目录下执行如下代码进行依赖包更新：

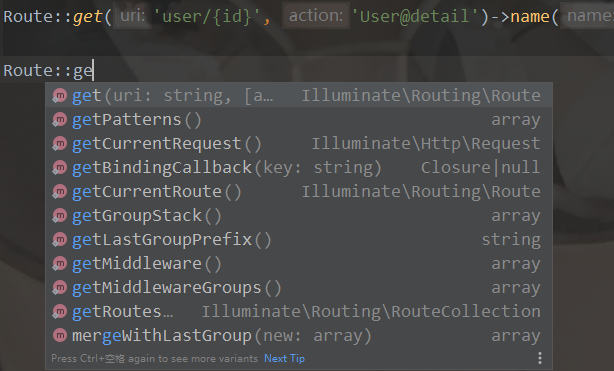
|  |
| --- |
| D:\WorkSpace\PHP\MyCMS>composer update |

步骤四：生成配置文件，在artisan所在目录执行如下命令：

|  |
| --- |
| D:\WorkSpace\PHP\MyCMS>php artisan ide-helper:generate |

此时，在项目根目录下多了一个\_ide\_helper.php文件。

步骤五：测试代码提示，打开\routes\web.php文件，输入Route::get时，出现了代码提示，之前是没有的，如下图所示：



以上就是给 PHPStorm 安装 Laravel 代码提示的完整步骤了。

代码参考：[Laravel 超好用代码提示工具 Laravel IDE Helper](https://learnku.com/articles/10172/laravel-super-good-code-prompt-tool-laravel-ide-helper)

原文地址：<https://github.com/barryvdh/laravel-ide-helper>

### 安装laravel-debugbar调试工具库

官网地址：

<https://laravel-news.com/laravel-debugbar>

库文件地址：

<https://packagist.org/packages/barryvdh/laravel-debugbar>

步骤一：打开项目目录下的composer.json文件，给require变量增加行：

|  |
| --- |
| "barryvdh/laravel-debugbar": "dev-master" |

步骤二：打开\config\app.php文件，给providers变量增加行：

|  |
| --- |
| Barryvdh\Debugbar\ServiceProvider::class, |

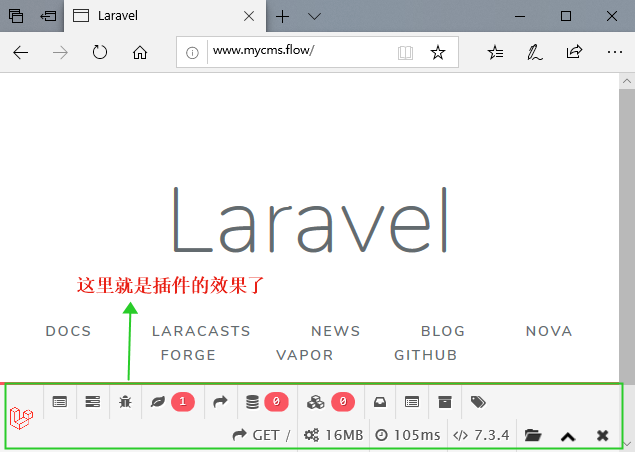
然后给**aliases**变量增加行：

|  |
| --- |
| 'Debugbar' => Barryvdh\Debugbar\Facade::class, |

步骤三：在项目目录下执行如下代码进行依赖包更新：

|  |
| --- |
| D:\WorkSpace\PHP\MyCMS>composer update |

步骤四：在浏览器输入网站访问网址，在浏览器的状态栏显示相关信息：



说明Laravel Debugbar安装没有问题，可以正常使用。

如果要关闭Debug的话，可以打开.env 把APP\_DEBUG设置为false即可。

### 安装汉字转拼音插件

插件地址：

<https://github.com/overtrue/laravel-pinyin>

步骤一：打开项目目录下的composer.json文件，给require变量增加行：

|  |
| --- |
| "overtrue/laravel-pinyin": "^4.0" |

步骤二：打开\config\app.php文件，给providers变量增加行：

|  |
| --- |
| Overtrue\LaravelPinyin\ServiceProvider::class, |

然后给**aliases**变量增加行：

|  |
| --- |
| 'Pinyin' => Overtrue\LaravelPinyin\Facades\Pinyin::class, |

步骤三：在项目目录下执行如下代码进行依赖包更新：

|  |
| --- |
| D:\WorkSpace\PHP\MyCMS>composer update |

在[MyCMS](#_编写数据填充代码)项目的database/factories/CategoryFactory.php有使用到转拼音功能。

### 安装图片验证码

图片验证码的地址：

<https://packagist.org/packages/gregwar/captcha>

步骤一：打开项目目录下的composer.json文件，给require变量增加行：

|  |
| --- |
| "gregwar/captcha": "^1.1.8" |

步骤二：在项目目录下执行如下代码进行依赖包更新：

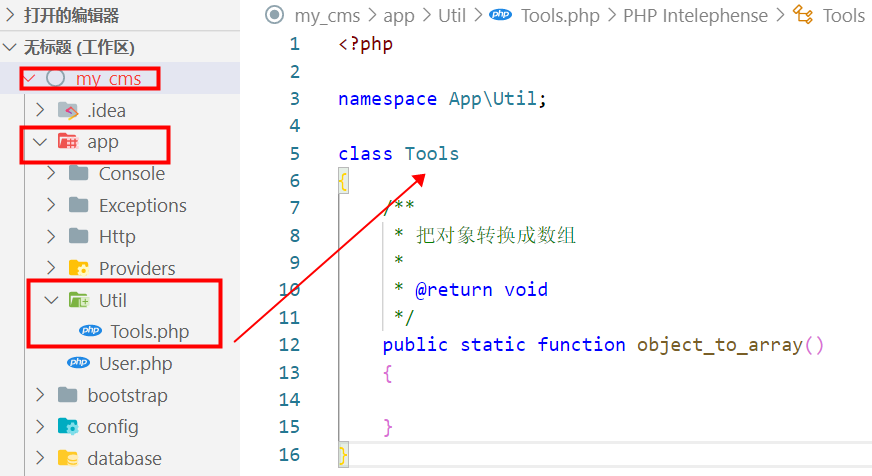
|  |
| --- |
| D:\WorkSpace\PHP\MyCMS>composer update |

### PHPUnit测试工具

<https://baijunyao.com/article/222>

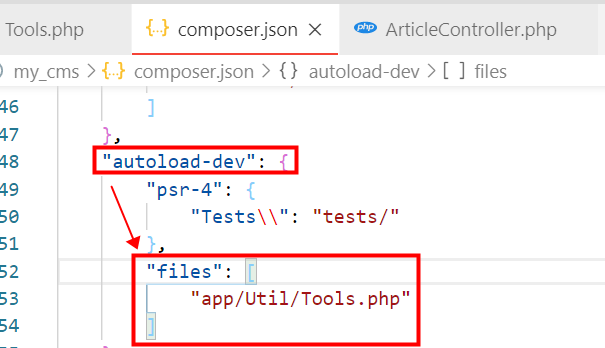
### 添加自定义工具类

在app目录下创建Util文件夹，然后在该文件夹下创建Tools类，如下图所示：



在该类里面添加静态方法。

然后我们需要在composer.json添加配置代码：



然后执行如下代码进行加载：

|  |
| --- |
| composer dump-autoload |

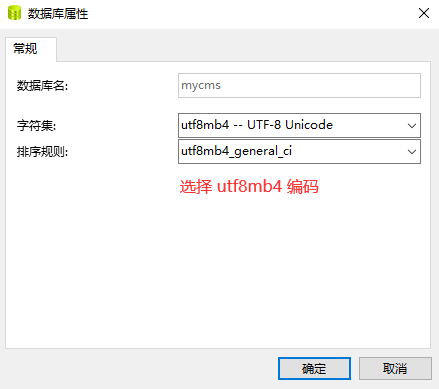
# Laravel开发网站

在进行Laravel开始时，需要先手工创建数据库，然后再做数据库配置，然后才开始进入编码开发。

## 入手增删改查

### 创建用户表结构

我们首先定义一个数据库，示例如下图所示：



然后，我们需要定义一个user的表结构，SQL定义语句如下所示：

|  |
| --- |
| DROP TABLE IF EXISTS `user`;  CREATE TABLE `user` (  `id` int(10) unsigned NOT NULL AUTO\_INCREMENT COMMENT '主键',  `name` varchar(50) NOT NULL DEFAULT '' COMMENT '用户名',  `passwd` char(32) NOT NULL DEFAULT '' COMMENT '密码',  PRIMARY KEY (`id`)  ) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4; |

### 添加用户信息

在routes/web.php添加如下代码：

|  |
| --- |
| //返回填写用户信息的界面  Route::view('/user/add', 'user.add');  //存储用户信息  Route::post('/user/store', 'UserController@store')->name('user.store'); |

创建resources/views/user/add.blade.php文件并添加如下代码：

|  |
| --- |
| <form method="post" action="{{ route('user.store') }}">  @csrf  <br>  <label for="username">账号</label>  <input id="username" name="username" value="orange">  <br>  <label for="passwd">密码</label>  <input id="passwd" name="passwd" type="password" value="123456">  <br>  <input type="submit" value="提交">  </form> |

在使用post方法提交时，一定要使用@csrf字段。另外，在action时，使用了命名路由的方式来填写表单提交的地址。

创建UserController控制器

|  |
| --- |
| php artisan make:controller UserController |

然后在app/Http/Controllers/UserController.php文件中添加如下代码：

|  |
| --- |
| function store(Request $request){  /\*\*  \* 把 \_token 之外的其它字段都取出来存放到 $input 变量中，结构如下所示：  \* array:2 [  \* "username" => "orange"  \* "passwd" => "a5s7sh4u"  \* ]  \*/  $input = $request->except('\_token');  dd($input);  } |

这里还没有做好保存。我们需要先把User模型和数据库的表结构做好映射，然后再使用模型的相关方法来保存数据会更方便点。

User模型映射user表结构

在app/Http/Controllers/UserController.php文件中的store方法修改成如下代码：

现在可以正常存储用户信息了，但是一直手工输入用户信息实在太慢了，怎么办呢？

我们可以使用Laravel的数据填充操作来填充user表，方便我们进行查询、分页、删除、修改等相关操作。

### 填充用户信息

### 查询用户信息

#### 处理分页伪静态

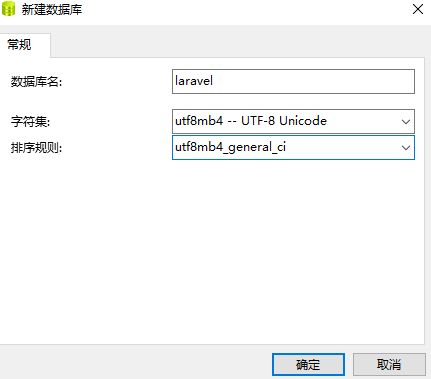
### 修改用户信息

### 删除用户信息

## 数据库操作

### 创建数据库

比如创建名为 laravel 的数据库：



一定要选择utf8mb4编码。

### 配置数据库

我们以MySQL数据库配置为例子，编辑 .env 文件，配置 MySQL 如下：

|  |
| --- |
| DB\_CONNECTION=mysql  DB\_HOST=127.0.0.1  DB\_PORT=3306  DB\_DATABASE=laravel  DB\_USERNAME=root  DB\_PASSWORD=root  DB\_CHARSET = utf8mb4 |

### 创建表结构

1 创建表结构的迁移文件

|  |
| --- |
| php artisan make:migration create\_users\_table |

新的迁移位于database/migrations目录下，每个迁移文件名都包含**时间戳**从而允许 Laravel 判断其顺序。

**-table**和**-create**选项可以用于指定表名以及该迁移是否要创建一个新的数据表。

这些选项只需要简单放在上述迁移命令后面并指定表名：

|  |
| --- |
| php artisan make:migration create\_users\_table –create=users  php artisan make:migration add\_votes\_to\_users\_table –table=users |

如果你想要指定生成迁移的自定义输出路径，在执行 make:migration 命令时可以使用**-path**选项，提供的路径应该是相对于应用根目录的。

2 在迁移文件里面定义相关字段

迁移类包含了两个方法：up 和 down。

up 方法用于新增表，列或者索引到数据库，而 down 方法就是 up 方法的反操作，和 up 里的操作相反。

在这两个方法中你都要用到 Laravel 的 schema 构建器来创建和修改表，要了解更多 Schema 构建器提供的方法，参考其文档。

下面让我们先看看创建 flights 表的简单示例：

|  |
| --- |
| <?php  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Database\Migrations\Migration;  class CreateFlightsTable extends Migration{  /\*\*  \* 运行迁移  \*  \* @return void  \*/  public function up()  {  Schema::create('flights', function (Blueprint $table) {  $table->increments('id');  $table->string('name');  $table->string('airline');  $table->timestamps();  });  }  /\*\*  \* 撤销迁移  \*  \* @return void  \*/  public function down()  {  Schema::drop('flights');  }  if (Schema::hasTable('users')) {  // 检测表是否存在  }  if (Schema::hasColumn('users', 'email')) {  // 检测列是否存在  }  } |

3 执行迁移文件创建表结构

|  |
| --- |
| php artisan migrate |

强制运行迁移文件：

|  |
| --- |
| php artisan migrate --force |

回滚迁移操作：

|  |
| --- |
| php artisan migrate:rollback |

### 数据库迁移

在执行数据库迁移操作时出现如下错误：

|  |
| --- |
| λ php artisan migrate  Migration table created successfully.  Migrating: 2014\_10\_12\_000000\_create\_users\_table  Illuminate\Database\QueryException : SQLSTATE[42000]: Syntax error or access violation: **1071 Specified key was too long; max key length is 1000 bytes** (SQL: alter table `users` add unique `users\_email\_unique`(`email`))  …… |

解决方法一：

修改\config\database.php的如下代码：

把代码：

|  |
| --- |
| 'mysql' => [  ......  'charset' => 'utf8mb4',  'collation' => 'utf8mb4\_unicode\_ci', |

修改为：

|  |
| --- |
| 'mysql' => [  ......  'charset' => 'utf8',  'collation' => 'utf8\_unicode\_ci', |

然后执行数据库迁移操作：

|  |
| --- |
| D:\WorkSpace\LaravelBlog>php artisan migrate  Migration table created successfully.  Migrating: 2014\_10\_12\_000000\_create\_users\_table  Migrated: 2014\_10\_12\_000000\_create\_users\_table (0.08 seconds)  Migrating: 2014\_10\_12\_100000\_create\_password\_resets\_table  Migrated: 2014\_10\_12\_100000\_create\_password\_resets\_table (0.18 seconds)  Migrating: 2019\_08\_19\_000000\_create\_failed\_jobs\_table  Migrated: 2019\_08\_19\_000000\_create\_failed\_jobs\_table (0.13 seconds)  Migrating: 2020\_01\_16\_072625\_create\_posts\_table  Migrated: 2020\_01\_16\_072625\_create\_posts\_table (0.08 seconds)  D:\WorkSpace\LaravelBlog> |

然后再把\config\database.php修改的utf8编码还原为utf8mb4编码。

解决方法二：

打开 2014\_10\_12\_000000\_create\_users\_table.php 文件，修改如下代码：

|  |
| --- |
| $table->string('email')->unique();  修改为  $table->string('email', 100)->unique(); |

执行迁移：

|  |
| --- |
| D:\WorkSpace\php\laravel\_study  λ php artisan migrate |

发现没有问题了，再执行一次迁移命令：

|  |
| --- |
| D:\WorkSpace\php\laravel\_study  λ php artisan migrate  Nothing to migrate. |

**当没有新的迁移文件时，提示没有东西迁移，如果有新的迁移文件，则执行新的迁移**，对于已经迁移过的文件，不做任何提示。

**解决方法三：（推荐该方法）**

打开app/Providers/AppServiceProvider.php文件，然后在boot()中添加如下代码：

|  |
| --- |
| Schema::defaultStringLength(191); |

//这里跟数据库编码有关。utf8字符串最大长度为255，而utf8mb4字符串最大长度为191，所以这里设置成191。

使用时，记得导入相关包：

|  |
| --- |
| use Illuminate\Support\Facades\Schema; |

### 填充数据表

#### Laravel获取Faker实例

|  |
| --- |
| use Faker;  $faker = app(Faker\Generator::class); |

#### Faker填充数据库

数据库迁移之后，我们可以通过填充器来往数据库表结构进行测试数据的填充。

方式一：通过编写填充器类填充数据

我们为users表创建一个填充器类：UsersTableSeeder

|  |
| --- |
| php artisan make:seeder UsersTableSeeder |

然后在/database/seeds/目录下可以看到生成了一个UsersTableSeeder.php文件，我们打开该文件进行编辑，我们需要把填充的逻辑代码定义在run方法里面：

|  |
| --- |
| <?php  use Illuminate\Database\Seeder;  use Illuminate\Support\Facades\DB;  use Illuminate\Support\Str;  class UsersTableSeeder extends Seeder  {  /\*\*  \* Run the database seeds.  \*  \* @return void  \*/  public function run()  {  $start\_timestamp = strtotime('2020-01-01 00:00:00');  $end\_timestamp = strtotime('2020-12-31 23:59:59');  for ($i = 0; $i < 1000; $i++) {  $created\_at = random\_int($start\_timestamp, $end\_timestamp);  DB::table('users')->insert([  'name' => Str::random(10),  'email' => Str::random(10) . '@gmail.com',  'email\_verified\_at' => date("Y-m-d H:i:s", $created\_at + 100),  'password' => md5(Str::random(10)),  'remember\_token' => Str::random(10),  'created\_at' => date("Y-m-d H:i:s", $created\_at),  'updated\_at' => date("Y-m-d H:i:s", $created\_at + 1000),  ]);  }  }  } |

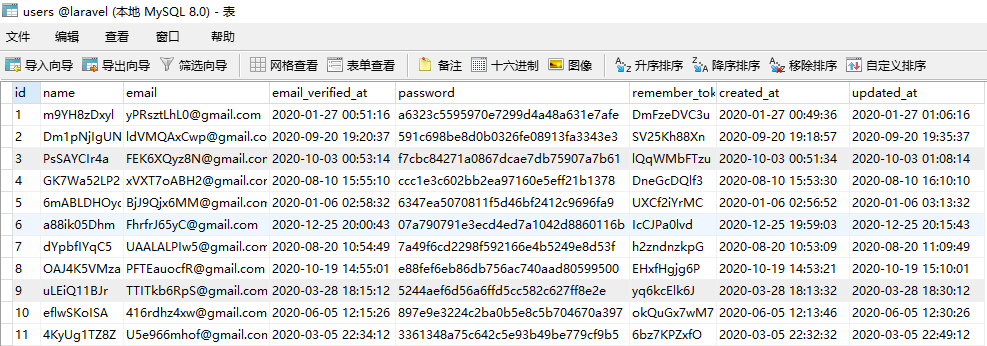
然后打开/database/seeds目录下的DatabaseSeeder.php文件，在run方法中，添加如下语句：

|  |
| --- |
| public function run()  {  $this->call(UsersTableSeeder::class);  } |

然后在命令行执行如下命令：

|  |
| --- |
| php artisan db:seed |

生成的测试数据如下图所示：



以上就是通过编写填充器类填充数据的操作过程了。

方式一：通过模型工厂填充数据

## Eloquent数据库操作

### 建立模型

|  |
| --- |
| class User extends Authenticatable  {  use Notifiable;  //指定表名，默认值为模型名的复数  protected $table = 'users';  //指定主键，默认主键为id  protected $primaryKey = 'id';  //是否维护时间戳，默认维护  //在数据库中会创建created\_at和updated\_at字段  public $timestamps = true;  //指定允许批量赋值的字段  //fillable为白名单，表示该字段可被批量赋值  protected $fillable = ['name', 'email', 'password',];  //guarded为黑名单，表示该字段不可被批量赋值  protected $guarded = [];  //维护时间的时候保存为时间戳格式  protected function getDateFormat() { return time(); }  //查询的时候返回时间戳格式的时间  protected function asDateTime($value) { return $value; }  //设置隐藏属性的字段  protected $hidden = ['password', 'remember\_token',];  //设置显示属性的字段  protected $visible = ['first\_name', 'last\_name',];  //对字段值进行数据类型转换  protected $casts = ['email\_verified\_at' => 'datetime',];  } |

### ORM相关操作

#### 插入数据

|  |
| --- |
| //save()  $student=new Student();  $student->name='vbb';  $student->age=34;  $rs=$student->save();  dd($rs);  //create()  $rs=Student::create([  'name'=>'momo',  'age'=>23  ]);  dd($rs);  //firstOrCreate()以属性查询数据 如果没有 新建数据  $rs=Student::firstOrCreate(  ['name'=>'vbb4']  );  //firstOrNew() 以属性查询数据 如果没有 新建实例 如果想保存调用save()  $rs=Student::firstOrNew(  ['name'=>'vbb4']  );  $bool=$rs->save();  dd($rs); |

#### 删除数据

|  |
| --- |
| //通过模型删除  $student = Student::find(2);  $bool = $student->delete();  var\_dump($bool);  //通过主键删除  $num = Student::destroy(3, 4, 5);  $num = Student::destroy([3, 4, 5]);  var\_dump($num);  //删除指定条件  $num = Student::where('id', '>', 7)->delete();  var\_dump($num); |

#### 修改数据

|  |
| --- |
| //通过模型更新数据  $student = Student::find(2);  $student->age = 2;  $bool = $student->save();  var\_dump($bool);  //批量更新  $num = Student::where('id', '>', 5)->update(  ['age' => 41]  );  var\_dump($num); |

#### 查询数据

|  |
| --- |
| //all() 查询所有数据 查询数据为集合  $students = Student::all();  dd($students);  //根据主键查询 查询一条数据  $student = Student::find(2);  dd($student);  //findOrFail() 根据主键查询 如果没有查到 报错  $student = Student::findOrFail(2);  dd($student);  //get() 查询所有数据  $students = Student::get();  dd($students);  //first() 查询第一条  $students = Student::where('id', '>', '1')->orderBy('age', 'desc')->first();  dd($students);  //chunk() 每次查询一定条数  Student::chunk(2, function ($students) {  var\_dump($students);  });  //聚合函数  //count() 条数  $num = Student::count();  var\_dump($num);  //max() 查询最大值  $max = Student::where('id', '>', 1)->max('age');  var\_dump($max); |

## 查询和分页

一般操作步骤如下：

创建路由

|  |
| --- |
| // 用户列表  Route::get('users/{id?}', 'UserController@list')->where(['id' => '[0-9]+']); |

编写控制器方法

|  |
| --- |
| class UserController extends Controller  {  public function list($id = null)  {  if ($id == null) {  $id = 1;  }  // $users = DB::select("SELECT id,name,email,created\_at FROM users LIMIT " . ($id \* 10) . ",10");  //$users = DB::table('users')->select(['id','name','email','created\_at'])->paginate(15);  //  $users = User::select(['id', 'name', 'email', 'created\_at'])->orderBy('id', 'desc')->user\_seo\_paginate(10, ['id', 'name'], 'users', $id);  return view('users.list', ['users' => $users]);  }  } |

自定义分类页

|  |
| --- |
| <?php  namespace App\Utils;  use Illuminate\Container\Container;  use Illuminate\Database\Eloquent\Builder;  use Illuminate\Support\Str;  use Illuminate\Pagination\Paginator;  use Illuminate\Pagination\LengthAwarePaginator as BasePaginator;  // 自定义 User 列表分页链接  class UserPaginator extends BasePaginator  {  /\*\*  \* 将新增的分页方法注册到查询构建器中，以便在模型实例上使用  \* 注册方式：  \* 在 AppServiceProvider 的 boot 方法中注册：UserPaginator::rejectIntoBuilder();  \* 使用方式：  \* 将之前代码中在模型实例上调用 paginate 方法改为调用 seoPaginate 方法即可：  \* Article::where('status', 1)->seoPaginate(15, ['\*'], 'page', page);  \*/  public static function injectIntoBuilder()  {  /\*\*  \* $perPage 表示每页显示多少条数据  \* $columns 选择需要的列  \*/  Builder::macro('user\_seo\_paginate', function ($perPage, $columns, $pageName, $page) {  $perPage = $perPage ?: $this->model->getPerPage();  $items = ($total = $this->toBase()->getCountForPagination())  ? $this->forPage($page, $perPage)->get($columns)  : $this->model->newCollection();  $options = [  'path' => Paginator::resolveCurrentPath(),  'pageName' => $pageName,  ];  return Container::getInstance()->makeWith(UserPaginator::class, compact(  'items', 'total', 'perPage', 'page', 'options'  ));  });  }  /\*\*  \* 重写页面 URL 实现代码，去掉分页中的问号，实现伪静态链接  \* @param int $page  \* @return string  \*/  public function url($page){  if ($page <= 0) {  $page = 1;  }  // 移除路径尾部的/  $path = rtrim($this->path, '/');  // 如果路径中包含分页信息则正则替换页码，否则将页码信息追加到路径末尾  if (preg\_match('/\/users\/\d+/', $path)) {  if($page == 1){  /\*\*  \* 链接格式为 http://www.laravel-study.com/users  \*/  //$path = preg\_replace('/\/users\/\d+/', '/users' . '', $path);  /\*\*  \* 链接格式为 http://www.laravel-study.com/users/1  \*/  $path = preg\_replace('/\/users\/\d+/', '/users/' . $page, $path);  }else{  $path = preg\_replace('/\/users\/\d+/', '/users/' . $page, $path);  }  } else {  $path .= '/' . $page;  }  $this->path = $path;  if ($this->query) {  $url = $this->path . (Str::contains($this->path, '?') ? '&' : '?')  . http\_build\_query($this->query, '', '&')  . $this->buildFragment();  } elseif ($this->fragment) {  $url = $this->path . $this->buildFragment();  } else {  $url = $this->path;  }  return $url;  }  /\*\*  \* 重写当前页设置方法  \*  \* @param int $currentPage  \* @param string $pageName  \* @return int  \*/  protected function setCurrentPage($currentPage, $pageName)  {  if (!$currentPage && preg\_match('/\/users\/(\d+)/', $this->path, $matches)) {  $currentPage = $matches[1];  }  return $this->isValidPageNumber($currentPage) ? (int) $currentPage : 1;  }  } |

注册自定义的分页类

|  |
| --- |
| <?php  namespace App\Providers;  use Illuminate\Support\ServiceProvider;  use App\Utils\UserPaginator;  class AppServiceProvider extends ServiceProvider  {  /\*\*  \* Register any application services.  \*  \* @return void  \*/  public function register()  {  // 注册 User 的分页链接类  UserPaginator::injectIntoBuilder();  } |

编写视图文件 users/list.blade.php

|  |
| --- |
| <!doctype html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport"  content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">  <meta http-equiv="X-UA-Compatible" content="ie=edge">  <title>Document</title>  <style type="text/css">  ul.pagination li{  list-style-type: none;  display: inline-block;  float: left;  padding: 0 10px;  }  ul.pagination li a {  text-decoration: none;  }  </style>  </head>  <body>  @foreach($users as $user)  <li>ID: {{ $user->id }} - 名字: {{ $user->name }} - 邮箱: {{ $user->email }} - 注册日期: {{ $user->created\_at }}</li>  @endforeach  {{ $users->links() }}  </body>  </html> |

以上就是查询和分页操作了。

参考地址：

<https://xueyuanjun.com/post/9694>

<https://www.cnblogs.com/Richard-Tang/p/11150892.html>

### 自定义分页伪静态

|  |
| --- |
| <?php  namespace App\Utils;  use Illuminate\Container\Container;  use Illuminate\Database\Eloquent\Builder;  use Illuminate\Support\Str;  use Illuminate\Pagination\Paginator;  use Illuminate\Pagination\**LengthAwarePaginator** as BasePaginator;  use Illuminate\Pagination\**AbstractPaginator**;//定义分页链接的格式  /\*\*  \* Class AcademyPaginator  \* @package App\Utils  \*  \* 由于 **LengthAwarePaginator** 继承了 **AbstractPaginator**  \* 所以，我们这里继承 **LengthAwarePaginator** 即可  \* 然后重写 url 方法即可  \*  \*/  class AcademyPaginator extends BasePaginator  {  /\*\*  \* 重写页面URL格式的实现代码  \* 去掉分页中的问号，实现伪静态链接  \* @param int $page  \* @return string|void  \*/  public function url($page)  {  if ($page <= 0) {  $page = 1;  }  //移除路径尾部的 / 斜杠  $path = rtrim($this->path, '/');  //如果路径中包含分页信息则正则替换页码  //否则将页码信息追加到路径末尾  if (preg\_match('/\/page\/\d+/', $path)) {  $path = preg\_replace('/\/page\/\d+/', '/page/' . $page, $path);  } else {  $path .= '/page/' . $page;  }  $this->path = $path;  if ($this->query) {  $url = $this->path . (Str::contains($this->path, '?') ? '&' : '?')  . http\_build\_query($this->query, '', '&')  . $this->buildFragment();  } elseif ($this->fragment) {  $url = $this->path . $this->buildFragment();  } else {  $url = $this->path;  }  return $url;  }  /\*\*  \* 重置当前页设置方法  \* @param int $currentPage  \* @param string $pageName  \* @return int  \*/  protected function setCurrentPage($currentPage, $pageName)  {  if (!$currentPage && preg\_match('/\/page\/(\d+)/', $this->path, $matches)) {  $currentPage = $matches[1];  }  return $this->isValidPageNumber($currentPage) ? (int)$currentPage : 1;  }  /\*\*  \* 将新增的分页方法注册到查询构建器中，以便在模型实例上使用  \* 注册方式：  \* 在 AppServiceProvider 的 boot 方法中注册：AcademyPaginator::rejectIntoBuilder();  \* 使用方式：  \* 将之前代码中在模型实例上调用 paginate 方法改为调用 seoPaginate 方法即可：  \* Article::where('status', 1)->seoPaginate(15, ['\*'], 'page', page);  \*/  public static function injectIntoBuilder()  {  /\*\*  \* seoPaginate 就是调用分页链接的方法  \*/  Builder::macro('seoPaginate', function ($perPage, $columns, $pageName, $page) {  $perPage = $perPage ?: $this->model->getPerPage();  $items = ($total = $this->toBase()->getCountForPagination())  ? $this->forPage($page, $perPage)->get($columns)  : $this->model->newCollection();  $options = [  'path' => Paginator::resolveCurrentPath(),  'pageName' => $pageName,  ];  return Container::getInstance()->makeWith(AcademyPaginator::class, compact(  'items', 'total', 'perPage', 'page', 'options'  ));  });  }  /\*\*  \* 接下来  \* 在 AppServiceProvider 的 boot 方法中全局调用这个注入：  \* // 为查询构建器注入自己实现的分页器方法  \* AcademyPaginator::injectIntoBuilder();  \* 调用方法：  \* $articles = Article::with('author', 'category')  \* ->public()  \* ->orderBy('id', 'desc')  \* ->seoPaginate($pageSize, $this->listColumns, 'page', $page);  \*/  } |

参考地址：

<https://xueyuanjun.com/post/9661.html>

## 定义路由

路由定义的位置为/route/web.php文件中。因此，我们说的路由文件一般情况下就是指/route/web.php文件。

### 基本路由

基本路由的代码格式如下两种：

一个URL与一个闭包：

|  |
| --- |
| Route::get('/hello',function (){  return 'Hello World!';  }); |

一个URL与一个控制器：

打开\routes\web.php文件，添加如下代码：

|  |
| --- |
| Route::get('/user', 'UserController@index'); |

在 app\Http\Controllers\ 目录下创建 UserController.php 文件，并添加如下代码：

|  |
| --- |
| <?php  namespace App\Http\Controllers;  class UserController extends Controller {  public function index(){  return "UserController@index Say Hello World！";  }  } |

对于常见的HTTP请求，Laravel有以下几种路由：

|  |
| --- |
| Route::get($uri, $callback);  Route::post($uri, $callback);  Route::put($uri, $callback);  Route::patch($uri, $callback);  Route::delete($uri, $callback);  Route::options($uri, $callback);  Route::match(['get', 'post'], $uri, $callback);//响应get和post请求  Route::any($uri, $callback);//响应所有请求 |
| $callback要么是闭包，要么是一个控制器方法。 |

#### 返回无参数视图的路由

在/resources/views/目录下增加hello.blade.php视图文件，代码如下：

|  |
| --- |
| <?php  echo "Hello World!我是返回视图的路由配置哦！！！"; |

在路由文件中增加如下代码：

|  |
| --- |
| Route::get('/hello',function (){  return view('hello');  }); |

#### 返回带参数视图的路由

在/resources/views/目录下增加name.blade.php视图文件，代码如下：

|  |
| --- |
| <?php  echo "{$name} 你好，你是通过参数路由传递过来的吗？"; |

在路由文件中增加如下代码：

|  |
| --- |
| Route::get('/user/{name}', function ($name) {  return view('name', ['name' => $name]);  }); |

在浏览器输入：<http://www.mycms.flow/user/orange>

打印如下结果：orange 你好，你是通过参数路由传递过来的吗？

#### 返回使用视图组的路由

在/resources/views/目录下增加user/目录，然后在user/目录下增加add.blade.php视图文件，代码如下：

|  |
| --- |
| <?php  echo "我是一个 add 增加用户的操作界面！"; |

在路由文件中增加如下代码：

|  |
| --- |
| Route::get('/user/add',function (){  return view("user.add");  }); |

在浏览器访问：http://www.mycms.flow/user/add

输出如下结果：我是一个 add 增加用户的操作界面！

### 视图路由

视图路由是指通过路由访问某个URL的时候，直接返回一个view视图。

基本格式如下：

|  |
| --- |
| Route::view('路由', '视图名称', ['可选的数组数据']); |

打开\routes\web.php文件，添加如下代码：

|  |
| --- |
| Route::view('/hello', 'hello', ['name'=>'orange', 'age'=>20]); |

在resources\views\目录下创建hello.blade.php文件，添加代码如下：

|  |
| --- |
| <p>我的名字是：</p>  <p>{{$name}}</p>  <p>我的年龄是：</p>  <p>{{$age}}</p> |

然后打开浏览器，在浏览器地址栏输入：<http://www.mycms.flow/hello> ，显示如下结果：



以上是视图路由示例。

### 命名路由

常见命名路由格式：

|  |
| --- |
| //闭包的命名路由  Route::get('user', array('as' => 'alia', function () {}));  //闭包的方法链路由  Route::get('user', function () {})->name('alia');  //控制器方法的命名路由  Route::get('user',array('as'=>'alia','uses'=>'AController@index'));  //控制器方法的方法链路由  Route::get('user', 'AController@index')->name('alia'); |

#### 定义命名路由

定义命名路由有两种方式，一种是as，另一种是name。

如下示例代码：

|  |
| --- |
| Route::get('api/user',['as'='web.user'],'messageController@userInformation'); |
| Route::get('api/user','messageController@userInformation')->name('web.user'); |

#### 使用命名路由

使用命名路由都是用route方法，参数值为路由的名称。

在PHP代码中可以这样使用：

|  |
| --- |
| this->visit(route('web.user')) |

在blade.php模板文件中这样使用：

|  |
| --- |
| <a href="{{route('web.user')}}">user</a> |

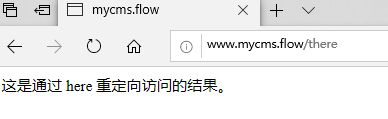
### 路由重定向

#### 302 临时重定向

打开\routes\web.php文件，添加如下代码：

|  |
| --- |
| Route::get('/there', function(){  return "这是通过 here 重定向访问的结果。";  });  Route::redirect('/here', '/there'); |

然后打开浏览器，在浏览器地址栏输入：<http://www.mycms.flow/here> ，显示如下结果：



从截图中可以看出，本来访问的是 here 路径，结果却重定向到了 there 路径了。

以上的重定向是 302 方式重定向，也就是临时重定向。

如果要使用 301 永久重定向怎么配置？

#### 301 永久重定向

在Laravel中，实现301永久重定向有两种方式：

方式一：

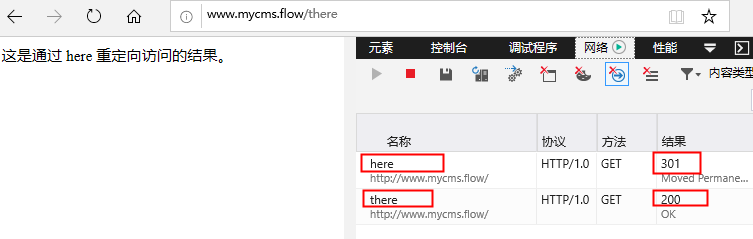
给Route::redirect设置第三个参数，如下代码所示：

|  |
| --- |
| Route::redirect('/here', '/there', 301); |

示例代码：

|  |
| --- |
| Route::get('/there', function(){  return "这是通过 here 重定向访问的结果。";  });  Route::redirect('/here', '/there', 301); |

然后打开浏览器，在浏览器地址栏输入：<http://www.mycms.flow/here> ，显示如下结果：



从截图可以看出here经过301重定向到了there页面了。

方式二：

使用Route::permanentRedirect方法，如下代码所示：

|  |
| --- |
| Route::permanentRedirect('/here', '/there'); |

示例代码：

|  |
| --- |
| Route::get('/there', function(){  return "这是通过 here 重定向访问的结果。";  });  Route::permanentRedirect('/here', '/there'); |

访问结果与上面相同，这里不再重复。

### 路由参数

在Laravel的路由参数中，如果路由参数**不带?**则为**必选**参数，如果**带?**则为**可选**参数。

|  |
| --- |
| 路由参数常见使用情况： |
| //单个路由参数  Route::get('user/{id}', function ($id) {  return 'User ' . $id;  });  //多个路由参数  Route::get('posts/{post}/comments/{comment}', function ($postId, $commentId) {  //  });  //单个路由参数（可选）  Route::get('user/{id?}', function ($id = 1) {  return 'User ' . $id;  });  //多个路由参数（可选）  Route::get('posts/{post}/comments/{comment?}', function ($postId, $commentId = 1) {  //  });  //**注意：多个参数时，只可以对最后一个参数设置可选，其他位置设置可选会解析错误**  // 正则约束单个参数  Route::get('user/{name?}', function ($name = 'Jone') {  return $name;  })->where('name', '\w+'); //约束参数为单词字符（数字、字母、下划线）  // 正则约束多个参数  Route::get('user/{id}/{name}', function ($id, $name) {  //  })->where(['id' => '[0-9]+', 'name' => '[a-z]+']); |

#### 必选参数

打开\routes\web.php文件，添加如下代码：

|  |
| --- |
| Route::get('user/{id}', function($userId){  return "当前访问的 User 的 ID 值为：" . $userId;  }); |

定义多个必选参数：打开\routes\web.php文件，添加如下代码

|  |
| --- |
| Route::get('posts/{postId}/comments/{commentId}', function($postId,$commentId){  return "通过 ID 为 " . $postId ." 的 postID 和 ID 为 " . $commentId . " 的 commentId 获取数据";  }); |

#### 可选参数

打开\routes\web.php文件，添加如下代码：

|  |
| --- |
| Route::get('username/{name?}', function($name = null){  return $name;  }); |

#### 可选参数默认值

我们可以把上面的代码修改成如下，再试试浏览结果：

|  |
| --- |
| Route::get('username/{name?}', function($name = "John"){  return $name;  }); |

#### 使用规则

1、路由参数和形式参数的值是按照顺序一一对应，与参数的名称无关。

2、路由参数名称只能包含字母，如果需要包含 - 则由 \_ 替代。

3、**可选参数**对应的变量**必须**设置默认值，不建议使用 null 默认值，最好是具体的业务相关的词。

4、设置多个路由参数时，最多只可对最后一个参数设置为可选，否则会解析错误。

### 正则参数

正则表达式参数是用于**对路由参数的格式进行约束**，换句话说就是路由参数的参数名必须满足正则表达式设置的格式才允许访问路由。

通过where方法来配置正则表达式。格式如下：

|  |
| --- |
| where('参数名称', '正则表达式'); |

示例代码，打开\routes\web.php文件，添加如下代码：

|  |
| --- |
| Route::get('username/{name?}', function($name = "John"){  return $name;  })->where('name','[a-zA-Z]\*');  Route::get('username/{id}/{name?}', function($name = 'Orange'){  return "ID And Name is " . $id ." " . $name;  })->where(['name'=>'[a-zA-Z]\*', 'id'=>'[0-9]+']); |

通过where进行过滤的方式为局部过滤，对应的，我们还可以设置**全局过滤**，在app/Providers/**RouteServiceProvider**.php文件的**boot**()方法的parent::boot();**语句之前**添加参数过滤规则，比如：

|  |
| --- |
| public function boot()  {  Route::pattern('id','[0-9]+');  parent::boot();  } |

### 路由分组

|  |
| --- |
| // 第一个数组参数中，  // prefix键定义URI的公共部分  // namespace键定义方法名(命名空间语法)的公共部分  Route::group(array('prefix' => 'admin', 'namespace' => 'Admin'), function () {  Route::get('login', 'IndexController@login');  Route::get('index', 'IndexController@index');  }); |

可以嵌套操作哦，比如：

|  |
| --- |
| Route::group(array('prefix' => 'admin', 'namespace' => 'Admin'), function () {  Route::group(array('prefix' => 'article', 'namespace' => 'Article'), function () {  });  Route::get('index', 'IndexController@index');  }); |

### 资源路由

在web.php中添加如下代码：

|  |
| --- |
| routes/web.php |
| Route::resource('article','ActicleController'); |

对应的路由情况如下图所示：



代码如下所示：

|  |
| --- |
| +---------+----------------------+---------------+--------------------------+  | Method | URI | Name | Action |  +---------+----------------------+---------------+--------------------------+  |POST |article |article.store |\ActicleController@store |  |GET|HEAD |article |article.index |\ActicleController@index |  |GET|HEAD |article/create |article.create |\ActicleController@create|  |PUT|PATCH|article/{article} |article.update |\ActicleController@update |  |GET|HEAD |article/{article} |article.show |\ActicleController@show |  |DELETE |article/{article} |article.destroy |\ActicleController@destroy|  |GET|HEAD |article/{article}/edit |article.edit |\ActicleController@edit |  +---------+----------------------+---------------+--------------------------+ |

### 路由列表

查看所有路由情况，命令：

|  |
| --- |
| php artisan route:list |

## CSRF 保护

### CSRF 用法

使用还是很简单的，只需要在 form 表单中使用@csrf的Blade指令即可。

示例代码：

|  |
| --- |
| <form method="" action="">  @csrf  ......  </form> |

一般情况下，当提交表单时，自动会把@csrf字段的值传输到服务器。

但是如果使用的是JS、jQuery等脚本处理表单时，需要在脚本代码里面提交@csrf字段的值，示意代码如下：

|  |
| --- |
| <form method="" action="">  <input type="hidden" name="\_token" value="......">  </form>  <script type="text/javascript">  function doLogin(){  ...  var \_token = $('input[name="\_token"]').val(); // 固定表单字段名  $.post('/admins',{...,\_token:\_token},function(res){  ...  },'json');  }  </script> |

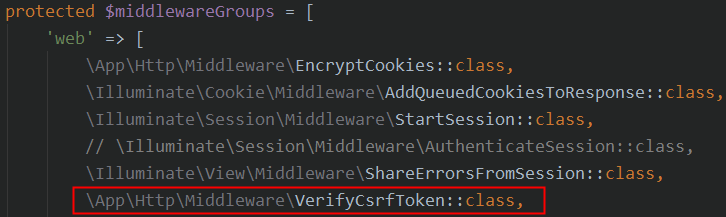
带@csrf的表单提交之后，包含在web中间件组里的VerifyCsrfToken中间件会自动验证请求里的令牌是否与存储在会话中的令牌相匹配。

如果在表单中不写@csrf代码会触发状态码为419的请求错误。

### 关闭 CSRF 防护

#### 关闭全部CSRF防护

在文件**app/Http/Kernel.php**中有如下配置：



把红框选中的代码删除即可关闭全部的CSRF防护功能。

#### 关闭部分CSRF防护

关闭部分CSRF防护，也就是通常所说的CSRF白名单。

CSRF白名单是指不需要CSRF保护的那些URL的配置单。

当URL不需要CSRF保护时，可以设置CSRF白名单，方式是通过将这类URL添加到VerifyCsrfToken中间件的$except属性来排除对这类路由的CSRF保护，示意代码如下所示：

|  |
| --- |
| <?php  namespace App\Http\Middleware;  use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken as Middleware;  class VerifyCsrfToken extends Middleware  {  /\*\*  \* 从CSRF验证中排除的 URI  \*  \* @var array  \*/  protected $except = [  'stripe/\*',  'http://example.com/foo/bar',  'http://example.com/foo/\*',  ];  } |

简单说就是找到app\Http\Middleware\VerifyCsrfToken.php文件，在文件属性$except中设置那些不想做csrf验证防护的路由，比如我想要api开头的路由都不做 csrf验证，只需要添加api/\* 就可以了。

### X-CSRF-Token

在使用Ajax请求时，如何添加CSRF保护呢？

首先把CSRF代码存放在meta里面：

|  |
| --- |
| <meta name="csrf-token" content="{{ csrf\_token() }}"> |

然后在发起请求时，带上meta信息：

|  |
| --- |
| $.ajax({  headers: { 'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content') }  }); |

### X-XSRF-Token

Laravel 默认会把 CSRF\_TOKEN 的值写在一个叫 XCRF-TOKEN 的 cookie 中，其实每次访问这个值都会发生变化，因此我们只需要用这个值即可。

示例代码：

|  |
| --- |
| $.ajaxSetup({  headers: {  'X-XSRF-TOKEN': $.cookie('XSRF-TOKEN')  }  }); |

我们在某个全局的文件里面进行调用就可以了。

## 定义中间件

### 中间件作用

中间件的作用是用来过滤进入应用程序的HTTP请求，简单说就是过滤HTTP请求。

Laravel自带的中间件包括身份验证中间件、CSRF中间件、日志中间件等，所有的这些中间件都位于app\Http\Middleware目录中。

注意控制器Controllers和中间件Middleware同属于app\Http命名空间下的同层目录中。

### 中间件定义

在app/Http/Middleware目录下定义两个中间件，一个作为全局中间件，一个作为路由中间件。

创建PrintDateTime.php文件，作为全局中间件，代码如下：

|  |
| --- |
| <?php  namespace App\Http\Middleware;  use Illuminate\Http\Request;  use Closure;  class PrintDateTime {  public function handle(Request $req, Closure $next){  // 在网页打印当前日期和时间  print(date("Y-m-d h:i:s") . "<br/>");  return $next($req);  }  } |

创建CheckAge.php文件，作为路由中间件，代码如下：

|  |
| --- |
| <?php  namespace App\Http\Middleware;  use Illuminate\Http\Request;  use Closure; # 必需  class CheckAge {  // handle 方法是 中间件进行数据处理的入口函数。  public function handle(Request $req, Closure $next){  // 在网页打印字符串  print("我是路由中间件，输入的年龄为 " .$req->id." 岁！<br/>");  if($req->id >= 200){  print("我大于 200 岁哦！<br/>");  return redirect('shouye');  }else{  print("我小于 200 岁哦!<br/>");  return $next($req);  }  }  } |

由于所有的中间件都是通过**服务容器**，因此，可以在中间件的构造函数中输入我们需要的任何依赖。

根据**中间件处理代码**在\$next($req)之前或者之后，可以分为**前置中间件**和**后置中间件**，如下示意代码：

|  |
| --- |
| 前置中间件 |
| <?php  namespace App\Http\Middleware;  use Closure;  // 前置中间件  class BeforeMiddleware{  public function handle($request, Closure $next){  // 执行一些任务  return $next($request);  }  } |

|  |
| --- |
| 后置中间件 |
| <?php  namespace App\Http\Middleware;  use Closure;  // 后置中间件  class AfterMiddleware{  public function handle($request, Closure $next){  $response = $next($request);  // 执行一些任务  return $response;  }  } |

### 中间件注册

中间件在定义完成之后，必须进行注册才可以使用。在注册中间件的时候，根据中间件执行的周期，可以分为：**全局中间件**和**路由中间件**。

#### 全局中间件

如果我们想让**应用在处理每一个 HTTP 请求时，都运行一次中间件**，则需要把该中间件注册为全局中间件，也就是在**app\Http\Kernel.php**文件中的**$middleware**属性中列出该中间件，比如我们把上面定义的**PrintDateTime**注册为全局中间件，则需要在**$middleware**列出，示意代码如下：

|  |
| --- |
| protected $middleware = [  \App\Http\Middleware\TrustProxies::class,  ......  // 自定义的 PrintDateTime 全局中间件  \App\Http\Middleware\PrintDateTime::class,  ]; |

在这里注册的中间件，将会在每一个 HTTP 请求中都执行一次，请求一次就执行一次，请求 N 次就执行 N 次。

#### 路由中间件

现在，我们把上面定义的CheckAge注册为路由中间件，则需要在$routeMiddleware列出，示意代码如下：

|  |
| --- |
| protected $routeMiddleware = [  'auth' => \App\Http\Middleware\Authenticate::class,  ......  // 自定义的 CheckAge 路由中间件  'check.age' => \App\Http\Middleware\CheckAge::class,  ]; |

接下来，我们编写相关路由和控制器代码，来测试上面的两个中间件。

打开 routes\web.php 文件，编写如下代码：

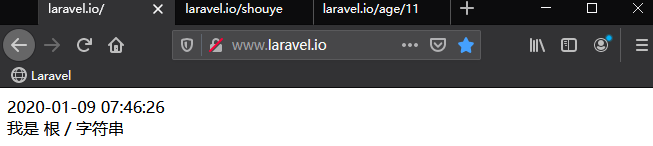
|  |
| --- |
| <?php  Route::get('/', function(){  return "我是 根 / 字符串";  });  Route::get('shouye', function(){  return "我是 shouye 字符串";  });  Route::get('age/{id}', 'UserController@getAge')->where('id','[0-9]+')->middleware('check.age'); |

在 app\Http\Controllers 目录下创建 UserController.php 文件，代码如下：

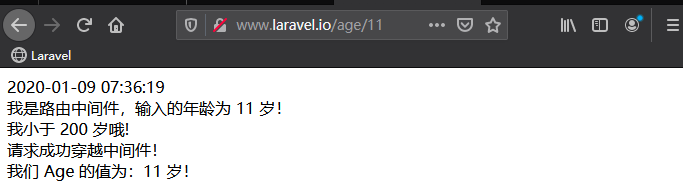
|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Illuminate\Http\Request;  class UserController extends Controller {  public function getAge($id){  return "请求成功穿越中间件！<br/>我们 Age 的值为：" . $id ." 岁！<br/>";  }  } |

接下来我们在浏览器进行访问：

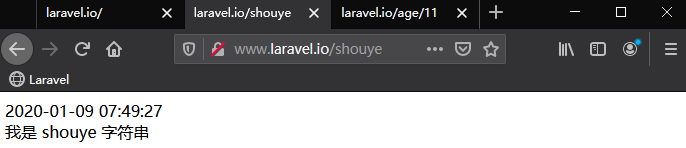
1. 在浏览器输入 http://www.laravel.io/ 结果如下图所示：



1. 在浏览器输入 http://www.laravel.io/age/11 结果如下图所示：



1. 在浏览器输入 http://www.laravel.io/age/888 结果如下图所示，发生了重定向：



从输入的 URL 地址和显示结果可以看出每次访问都打印了当前的日期和时间，说明全局中间件 PrintDateTime 起作用了。

而从第二张和第三张图对比，可以发现路由中间件 CheckAge 起作用了。

另外，在 CheckAge.php 文件中的代码：

|  |
| --- |
| return redirect('shouye'); # 这些写的是路由的 URL 地址，而不是路由的 name 名称。 |

**分配多个中间件**，示例代码：

|  |
| --- |
| Route::get('age/{id}', 'UserController@getAge')->middleware('first', 'second'); |

**分配路由中间件时，直接传递完整类名**，示例代码：

|  |
| --- |
| use App\Http\Middleware\CheckAge;  Route::get('age/{id}', 'UserController@getAge')->middleware(CheckAge::class); |

### 中间件群组

就是使用一个键把多个中间件打包成一个组，以便更容易地分配给路由。

在**app\Http\Kernel.php**文件中的**$middlewareGroups**属性中列出了中间件群组，代码如下：

|  |
| --- |
| protected $middlewareGroups = [  'web' => [  \App\Http\Middleware\EncryptCookies::class,  ......  \Illuminate\Routing\Middleware\SubstituteBindings::class,  ],  'api' => [  'throttle:60,1',  'bindings',  ],  ]; |

**中间件群组分配给路由的方式**与**单个中间件分配给路由的方式**是一样的，比如我们要使用上面定义的**web**中间件群组，则使用方式如下：

|  |
| --- |
| Route::get('/', 'UserController@getAge')->middleware('web'); |
| Route::group(['middleware' => ['web']], 'UserController@getAge'); |

注意：**RouteServiceProvider**默认会把web中间件组自动应用到routes/web.php。

### 中间件排序

在开发时，可能需要中间件按照特定的顺序来执行，在这种情况下，可以使用app/Http/Kernel.php文件的$middlewarePriority属性指定中间件优先级：

|  |
| --- |
| <?php  /\*\*  \* 中间件的优先级排序列表，越是上面则越先执行：  \* 比如，定义了一个 CustomMiddelware ，希望在 Authenticate 执行之前优先执行，  \* 则把 CustomMiddelware 写在 Authenticate 前面，  \* 如下代码所示：  \*/  protected $middlewarePriority = [  \Illuminate\Session\Middleware\StartSession::class,  \Illuminate\View\Middleware\ShareErrorsFromSession::class,  \App\Http\Middleware\CustomMiddelware::class,  \App\Http\Middleware\Authenticate::class,  \Illuminate\Session\Middleware\AuthenticateSession::class,  \Illuminate\Routing\Middleware\SubstituteBindings::class,  \Illuminate\Auth\Middleware\Authorize::class,  ]; |

不在$middlewarePriority列表中的中间件，则是按照在路由配置里编写的顺序来依次调用执行：

* 在 Route::group 里定义的，外层先执行，内层后执行；
* 在数组里定义的，数组前面的先执行，数组后面的后执行；

如下代码所示：

|  |
| --- |
| Route::group(['middleware' => 'outer'], function () {  Route::group(['middleware' => 'middle'], function () {  Route::group(['middleware' => ['array1', 'array2', 'auth:api']], function () {  Route::get('test', function () {  return;  });  });  });  }); |

这里就是按照 outer、middle、array1、array2 的顺序执行，这个比较好理解，也比较容易记住，规则就是按照从上到下、从左到右的顺序，先碰到哪个就先执行哪个。

### 中间件参数

中间件也是可以接收参数的。

1. 在app\Http\Middleware目录下创建CheckRole.php中间件文件，代码如下：

|  |
| --- |
| <?php  namespace App\Http\Middleware;  use Illuminate\Http\Request;  use Closure;  class CheckRole {  // $role 表示角色  // $rank 表示等级  public function handle(Request $req, Closure $next, $roleName, $rank){  if($req->role == $roleName){  echo "输入的角色 " . $roleName . " 符合要求！<br/>";  echo "角色的等级为： " . $rank . " 级！！！<br/>";  return $next($req);  }  }  } |

1. 注册中间件，打开app\Http\Kernel.php文件，在$routeMiddleware属性中注册中间件，代码如下所示：

|  |
| --- |
| protected $routeMiddleware = [  'auth' => \App\Http\Middleware\Authenticate::class,  ...  // 自定义的 CheckRole 中间件  'check.role' => \App\Http\Middleware\CheckRole::class,  ]; |

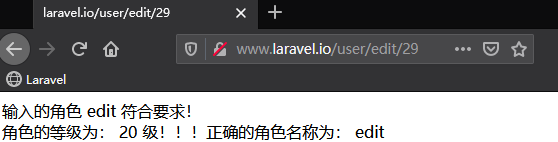
1. 创建控制器，在app\Http\Controllers目录下创建UserController.php文件，代码如下：

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Illuminate\Http\Request;  class UserController extends Controller {  public function getRole($role){  return "正确的角色名称为： " . $role;  }  } |

1. 编写路由，打开routes\web.php文件，添加如下代码：

|  |
| --- |
| <?php  Route::get('user/{role}/{rank}', 'UserController@getRole')->where(['role'=>'[a-z]+','rank'=>'[0-9]+'])->middleware('check.role:edit,20'); |

1. 访问路由，在浏览器输入 http://www.laravel.io/user/edit/29 ，结果如下所示：



以上就是使用中间件参数的过程了。

从结果可以看出：

1、中间件接收参数的形参是放在 \$next 这个形参后面的。比如 \$roleName, \$rank 就放在 \$next 之后。

2、路由在使用中间件时，通过**:**来分隔中间件名称和传递给中间件的参数，如果有多个参数，则用**,**隔开。比如，在路由中的如下代码：

|  |
| --- |
| middleware('check.role:edit,20');  // check.role: 中间件名称  // edit,20: 依次传递给 handle 方法中的 $roleName, $rank |

### Terminable中间件

其实就是在中间件中添加 terminate 方法，该方法同时接收$request、$response参数，示意代码如下所示：

|  |
| --- |
| <?php  namespace Illuminate\Session\Middleware;  use Closure;  class StartSession{  public function handle($request, Closure $next){  ......  return $next($request);  }  public function terminate($request, $response){  // Store the session data...  ......  }  } |

如果我们的 PHP 使用的是 FastCGI 并且我们在中间件中定义了 terminate 方法，那么该 terminate 方法将会在响应准备发送到浏览器时（只是准备好要发送了，但是还没有发送）自动调用。

中间件上调用 terminate 方法的时候，Laravel 将从**服务容器**中解析出一个新的中间件实例。

如果在调用 handle 和 terminate 方法的同时使用相同的中间件实例，请使用容器的 singleton 方法在容器中注册中间件，防止产生两个一样的中间件。

### 中间件原理

Laravel中间件原理解析:

<https://www.jianshu.com/p/9432f6e980c1>

Laravel框架的中间件middleware详解:

<https://www.cnblogs.com/it-3327/p/11794804.html>

## 定义控制器

### 控制器位置

一般我们是在app/Http/Controllers目录中定义控制器。

但是为了更好的管理控制器，我们常常需要对控制器进行分组。因此，我们需要在app/Http/Controllers目录下定义新的目录，然后在新目录里面定义控制器。

### 基础控制器

控制器定义的基础代码：

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use App\User; // 导入模型  use App\Http\Controllers\Controller; // 导入基类控制器  class UserController extends Controller {  /\*\*  \* 显示用户信息  \*  \* @param int $id  \* @return view  \*  \*/  public function show($id){  return view("user.profile", ['user'=>User::findOrFail($id)]);  }  } |

下面我们定义一个路由来访问UserController控制器中的show方法，代码如下：

|  |
| --- |
| <?php  Route::get('user/{id}', 'UserController@show')->where('id', '[0-9]+'); |

当一个请求与指定路由的URI匹配时，UserController控制器中的show方法将被执行，路由参数也将会被传递给该show方法。

注意：控制器并不是必需继承基础类，然而，如果控制器没有继承基础类，你将无法使用一些便捷的功能，比如middleware中间件、validate校验和dispatch分发方法。

### 资源控制器

在创建控制器的基础上加上--resource参数即可，如下命令所示：

|  |
| --- |
| php artisan make:controller ActicleController --resource |

## 角色权限

## Blade 模板

参考资料:

<https://www.jianshu.com/p/4b195e5f241b>

### loop 变量



# 模块实现

## 定时任务

## Laravel队列使用

### 队列原理

### 队列选项和配置

Laravel队列的代码文件为：config/queue.php

我们打开该文件，可以看到如下代码：

|  |
| --- |
| 'connections' => [  'sync' => [  ……  ],  'database' => [  ……  ],  'beanstalkd' => [  ……  ],  'sqs' => [  ……  ],  'redis' => [  ……  ],  ], |

这部分的代码就是定义我们队列可使用的队列驱动，每种驱动说明如下：

|  |  |
| --- | --- |
| sync | 同步 |
| database | 数据库 |
| beanstalkd | 第三方队列服务 |
| sqs | 第三方队列服务 |
| redis | 第三方队列服务 |

那么在使用的时候，如何确定使用哪个队列驱动呢？很简单，在.env文件中的QUEUE\_CONNECTION这个配置项就是用来指定我们的队列采用的驱动类型，比如配置项的值如下：

|  |
| --- |
| QUEUE\_CONNECTION=sync |

说明，我们的队列采用的是sync同步方式的队列驱动。

### 队列的使用

一般经常使用的队列驱动为database和redis两种，所以分别以这两种来说明队列的使用方式。

<https://www.jianshu.com/p/3371666b7880>

<https://www.jianshu.com/p/a53351b6810a>

#### database驱动

#### redis驱动

## 自定义404页面

在Laravel中，404状态码属于NotFoundHttpException，而NotFoundHttpException本身也是对应404状态码，所以我们需要判断异常是否属于404异常就可以了，如果当前异常确实是404异常，则返回自定义的404页面。

打开app/Exceptions/Handler.php页面，修改render方法的代码：

|  |
| --- |
| 修改前 |
| public function render($request, Throwable $exception)  {  return parent::render($request, $exception);  } |
| 修改后 |
| use Symfony\Component\HttpKernel\Exception\NotFoundHttpException; |
| public function render($request, Throwable $exception)  {  /\*\*  \* 增加实现自定义 404 页面的代码  \*/  if ($exception instanceof NotFoundHttpException) {  return response()->view('error.404');  }  return parent::render($request, $exception);  } |

然后在views目录下创建error目录，并在error目录下创建404.blade.php文件，该文件代码如下：

|  |
| --- |
|  |

## webpack.mix.js

该文件用来把js、css文件打包到public目录下。

该文件的初始代码如下：

|  |
| --- |
| const mix = require('laravel-mix');  mix.js('resources/js/app.js', 'public/js')  .sass('resources/sass/app.scss', 'public/css'); |

打包时，执行如下命令：

|  |
| --- |
| npm run dev |

第一次打包时，会安装Mix模块的代码。打包执行过程的代码如下所示：

|  |
| --- |
| 98% after emitting SizeLimitsPlugin  DONE Compiled successfully in 5598ms 11:32:03  Asset Size Chunks Chunk Names  /css/app.css 0 bytes /js/app [emitted] /js/app  /js/app.js 592 KiB /js/app [emitted] /js/app |

然后在页面上通过asset就可以访问相关文件了，比如：

|  |
| --- |
| <link rel="stylesheet" href="{{ asset('/css/app.css') }}">  <script src="{{ asset('js/app.js') }}"></script> |

进一步学习参考网址：

<https://segmentfault.com/a/1190000010437630>

## 发送邮件

### 通过Gmail发送邮件

### 通过QQ邮箱发送邮件

## 发送短信

参考代码：

<http://www.bcty365.com/content-153-5885-1.html>

## 搜索模块

主要是学习如何在Laravel中使用Elasticsearch搜索模块。

Laravel集成搜索功能的话，需要使用Elasticsearch以及laravel/scout模块。

## 缓存模块

### 文件缓存

use Illuminate\Support\Facades\Cache;

一 存缓存

1. Cache::put(key,val,time);//存缓存

2. Cache::add(key,val,time);//缓存不存在时才能存

3.Cache::forever(key,val)//永久存入缓存，不需要时间

二 取缓存

1. Cache::get(key);

2. Cache::pulll(key);//取出缓存并删除此缓存

3. Cache::has(key);//判断此缓存是否存在

4. Cache::forget(key);//删除缓存

Laravel 文件缓存也可以快得飞起，tmpfs 了解一下：

<https://learnku.com/laravel/t/43651>

## 广播模块

<https://learnku.com/articles/35597>

## 辅助函数

### Arr::pluck()

用来遍历多维数组并检索给定键的所有值。

示例代码：

|  |
| --- |
| use Illuminate\Support\Arr;  $array = [  ['website' => ['id' => 1, 'url' => 'reddit.com']],  ['website' => ['id' => 2, 'url' => 'twitter.com']],  ['website' => ['id' => 3, 'url' => 'dev.to']],  ];  $names = Arr::pluck($array, 'website.url');  // ['reddit.com', 'twitter.com', 'dev.to'] |

### Str::contains()

判断一个字符串中是否包含另一个字符串。

示例代码：

|  |
| --- |
| use Illuminate\Support\Str;  $contains = Str::contains('My name is Inigo Montoya.', 'Inigo');  // true  $contains = Str::contains('My name is Inigo Montoya.', 'Andrew');  // false |

随着最近的rfc的批准，PHP将很快拥有自己的str\_contains方法，这将使该方法过时。不过在此之前，它是Laravel中最有用的辅助函数之一。

### Str::between()

返回两个字符串之间的内容。如果没有找到指定字符串，返回null。

示例代码：

|  |
| --- |
| use Illuminate\Support\Str;  $slice = Str::between('My name is Inigo Montoya.', 'My name is ', '.');  // 'Inigo Montoya' |

### Str::limit()

用于获取一个字符串并用一个设定的字符长度限制截断它，也可以传入第三个可选参数来控制返回的字符串后显示什么。

### head()和last()

head()返回数组的第一个元素。

示例代码：

|  |
| --- |
| $array = [100, 200, 300];  $first = head($array);  // 100 |

last()返回数组的最后一个元素。

示例代码：

|  |
| --- |
| $array = [100, 200, 300];  $last = last($array);  // 300 |

这两个方法不会影响原始数组。

### blank()

它根据你传入的参数是否包含实际数据，来返回一个布尔值。

示例代码：

|  |
| --- |
| // all of these return true  blank('');  blank(' ');  blank(null);  blank(collect());  // all of these return false  blank(true);  blank(false);  blank(0); |

这个辅助函数在结合表单验证、除去API中没有规范输入的数据时，特别有用。它比empty(trim())更好用。

### collect()

用于将数组转换为集合。

示例代码：

|  |
| --- |
| $collection = collect(['Keys', 'Krates']);  return $collection->map(function ($value) {  return Str::upper($value);  });  // ['KEYS', 'KRATES']  return $collection->filter(function ($value) {  return strlen($value) > 4;  });  // ['Krates'] |

当我们要处理的不是来自数据库模型的庞大而复杂的数据集时以及CSV数据、外部API请求、目录结构等都可以从放入集合中使用。

## 获取常用目录

### base\_path()

base\_path函数返回**项目根目录**的**绝对路径**：

$path = base\_path();

你还可以使用base\_path函数为相对于应用目录的给定文件生成**绝对路径**：

$path = base\_path('vendor/bin');

### app\_path()

app\_path函数返回**app目录**的**绝对路径**：

$path = app\_path();

你还可以使用app\_path函数为相对于app目录的给定文件生成**绝对路径**：

$path = app\_path('Http/Controllers/Controller.php');

### config\_path()

config\_path函数返回应用**配置目录**的**绝对路径**：

$path = config\_path();

### database\_path()

database\_path函数返回应用**数据库目录**的**绝对路径**：

$path = database\_path();

### public\_path()

public\_path函数返回**public目录**的**绝对路径**：

$path = public\_path();

### storage\_path()

storage\_path函数返回**storage目录**的**绝对路径**：

$path = storage\_path();

还可以使用storage\_path函数生成相对于storage目录的给定文件的绝对路径：

$path = storage\_path('app/file.txt');

# 版本区别

## Laravel5与Laravel6的区别

把 str\_slug 替换为 Str::slug() 函数，使用时需要导入 Str，如下操作：

|  |
| --- |
| use Illuminate\Support\Str; |

# 案例学习

## 相关项目

### LaraCMS

GitHub地址：<https://github.com/wanglelecc/laracms>

开发手册：<https://www.kancloud.cn/wanglelecc/laracms/>

官方网站：<https://www.laracms.cn/>

演示地址：<https://www.56br.com/>

### KKCMS

KKCMS是用于搭建自动采集影视的程序。

程序介绍：<https://wslmf.com/post/319.html>

GitHub地址：<https://github.com/Mengfanpro/kkcms>

### EQCMS

EQCMS是一套CMS建站系统。可以用于搭建影视站点。

系统介绍：<https://wslmf.com/522.html>

GitHub地址：<https://gitee.com/wslmf/eqcms>

采集脚本：<https://wslmf.com/post/529.html>

### LightCMS

LightCMS是一个基于Laravel开发的轻量级CMS系统，也可以作为一个通用的后台管理框架使用。

项目介绍：<https://segmentfault.com/a/1190000018801955>

GitHub地址：<https://github.com/eddy8/lightCMS>

演示站点：<http://lightcms.bituier.com/admin/login>

## 学习地址

学习和参考网址：

<https://xueyuanjun.com/>

GitHub 地址：

<https://github.com/laravel/laravel>

Laravel 视频地址：

<https://www.sixstaredu.com/laravel/index.html>

# 使用技巧

## 向公共模板传值

在网站开发中，往往有很多地方需要公共部分数据传递，比如菜单、侧边栏、热门文章、推荐文章、友情链接之类的。

实现 Laravel 向公共模板传值需要通过 Laravel 的 View Composer (视图合成器) 来实现。

操作步骤:

1、在 app\Http\ 下创建 ViewComposers 目录: 专门用于存放 View Composer

2、在 app\Http\ViewComposers\ 中创建需要的 View Composer:

|  |
| --- |
| namespace App\Http\ViewComposers;  use Illuminate\View\View;  use App\Models\Category;  class CategoryComposer{  private $category;  // 使用 laravel 的依赖注入自动注入我们需要的类  public function \_\_construct(Category $category) {  $this->category = $category;  }  // 这里的 view 就是我们的公共视图文件  public function compose(View $view) {  // 使用 with 方法注入变量  $view->with('navs', $this->category->getNavs());  }  } |
| 说明一下:  1、这里，我们只是要读取数据库表信息，所以不是非得用模型才行，用 DB:select 进行操作也是可以的。  2、另外，在写 View Composer 的时候，我们可以判断是否有缓存，如果有则取缓存中的数据，如果没有，则查询数据库。 |

以上两个步骤属于必须的步骤，接下来的步骤可以分成**两种方式**：

* **3.1、方式一 在 AppServiceProvider 的 boot 方法使用 View Composer:**

|  |
| --- |
| public function boot()  {  /\* 自定义类方式 \*/  // 复杂的预设变量通过自定义类实现，导航栏视图共享  view()->composer('layouts.main',CategoryComposer::class);  /\* 使用闭包方式 \*/  // 多个页面配置信息视图共享，如果是是全部页面可以使用 \* 代替  view()->composer(['layouts.main','login.create','users.create'], function ($view) {  $view->with('cfg',(object)Config::getAll());  });  // 当 Laravel 渲染 goods.index 和 goods.show 模板时，就会使ExampleComposer 这个来注入分类变量  // 同时 Laravel 还支持通配符，例如 goods.\* 即代表当渲染 goods 目录下的模板时都执行这个 ViewComposer  \View::composer(['goods.index', 'goods.show'],  \App\Http\ViewComposers\ExampleComposer::class  );  } |

我们可以通过这段代码取代之前的闭包函数定义的 View Composer，但是除非预设逻辑很复杂，否则推荐使用闭包函数的方式来实现，既简洁又减少了不必要的类初始化和方法调用对性能的损耗。

然后，我们就可以在视图上使用变量了，如下代码所示:

|  |
| --- |
| <meta name="keywords" content="{{ $cfg->keyword }}"/>  <meta name="description" content="{{ $cfg->description }}" /> |

* **3.2、方式二 自定义 ServiceProvider 并注册服务:**

|  |
| --- |
| <?php  namespace App\Providers;  use Illuminate\Support\ServiceProvider;  use Illuminate\Support\Facades\View;  class ViewComposerServiceProvider extends ServiceProvider {  public function boot() {  // 在这里，我们可以写多条的 View::composer 语句  View::composer('widget.categories', 'App\Http\ViewComposers\CategoriesComposer');  }  public function register(){}  } |

然后在 config/app.php 的 providers 中添加如下代码:

|  |
| --- |
| App\Providers\ViewComposerServiceProvider::class |

以上两种方式为最常见的方式，**还有一种方式是采用把数据共享给所有视图**:

在服务提供器 AppServiceProvider 中的 boot 方法调用 Illuminate\Support\Facades\View 的 share 方法，代码如下：

|  |
| --- |
| <?php  namespace App\Providers;  use Illuminate\Support\Facades\View;  class AppServiceProvider extends ServiceProvider {  public function boot() {  View::share('key', 'value');  }  public function register() {}  { |

但是这种方式太浪费资源，不合理，所以很少使用。

参考资料:

https://learnku.com/laravel/t/3094/some-practical-laravel-tips

https://learnku.com/articles/26010

https://www.jianshu.com/p/cdf494b844dd

## 简单的子模板传值

在当前视图文件中向子模板传值：

|  |
| --- |
| @include("message",['msg'=>'中国']) |

然后在子模板（也就是这里的message模板）使用变量：在 message 子模板中调用 msg 的值: {{msg}}

但是这种方式的话，在每个使用message子模板的视图文件中都需要给message赋值，这样操作就太麻烦了。

**不推荐使用这种方式**。

## 自定义函数和自定义类

### 自定义函数

首先，创建 app\helpers.php

|  |
| --- |
| <?php  function foo(){  return "Hello World";  } |

然后，修改 composer.json 文件，在 autoload 的 files 字段加入该文件:

|  |
| --- |
| "autoload": {  "files": {  "app/helpers.php"  }  } |

最后，运行：

|  |
| --- |
| composer dump-autoload -o |

### 自定义类

首先，创建自定义类 app/Libraries/Test.php

|  |
| --- |
| <?php  namespace App\Libraries;    class Test{  public static function index(){  echo 123;  }  } |

然后，修改 composer.json 文件，在 autoload 的 classmap 字段加入该文件:

|  |
| --- |
| "autoload": {  "classmap": {  "app/Libraries"  }  } |

最后，运行：

|  |
| --- |
| composer dump-autoload -o |

使用自定义类:

|  |
| --- |
| use App\Libraries\Test;  class IndexController extends Controller  {  public function ttt(){  Test::index();  }  } |

### 使用总结

为什么需要运行：

|  |
| --- |
| composer dump-autoload -o |

如果不运行的话，虽然不影响使用，但是无法进行代码跳转跟踪操作，也就是说会影响开发效率。

# 项目实战

## mycms

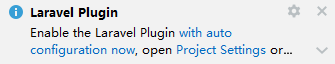
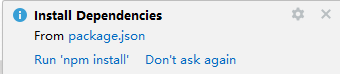
### 环境准备

Laravel版本：

使用的是Laravel 7.12.\* 版本。

|  |
| --- |
| composer create-project --prefer-dist laravel/laravel ./mycms 7.12.\* |

创建完成之后，启动PHPStorm，并使用PHPStorm打开项目，打开项目之后，PHPStorm右下角提示安装如下两个插件：



根据提示安装上面与项目相关的插件和依赖。其中package.json依赖安装之后，在项目里面会生成**node\_modules**目录。

然后安装如下两个插件：

[安装laravel-ide-helper代码提示库](#_安装laravel-ide-helper代码提示库)。

[安装laravel-debugbar调试工具库](#_安装laravel-debugbar调试工具库)。

[安装汉字转拼音插件](#_安装汉字转拼音插件)

[安装图片验证码](#_安装图片验证码)

创建数据库：

数据库名称为mycms，字符集编码为utf8mb4 -- UTF-8 Unicode，排序规则编码为utf8mb4\_unicode\_ci，两个编码规则都是Unicode。

修改.env数据库配置：

|  |
| --- |
| DB\_CONNECTION=mysql  DB\_HOST=127.0.0.1  DB\_PORT=3306  DB\_DATABASE=mycms  DB\_USERNAME=root  DB\_PASSWORD=root |

为了成功执行迁移命令，需要修改下源代码：

打开app/Providers/AppServiceProvider.php文件，然后在boot()中添加如下代码：

|  |
| --- |
| Schema::defaultStringLength(191); |

//这里跟数据库编码有关。utf8字符串最大长度为255，而utf8mb4字符串最大长度为191，所以这里设置成191。

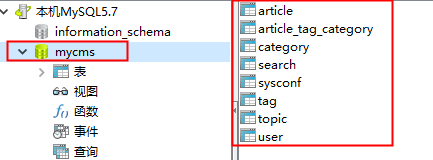
使用时，记得导入相关包：

|  |
| --- |
| use Illuminate\Support\Facades\Schema; |

### 创建表结构



打开mycms.sql文件，执行创建表结构的SQL语句，在数据库中创建8张表结构，如下图所示：



### 插入测试的数据

创建控制器：

|  |
| --- |
| php artisan make:controller TestController |

给控制器添加代码：

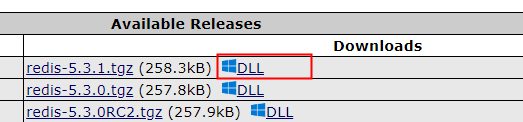
|  |
| --- |
| TestController.php |
| <?php  namespace App\Http\Controllers;  use DB;  use Faker;  use Str;  use Cache;  class TestController  {  // 添加 category 测试数据  public function addCategory()  {  $cates = [  '热点资讯', '新浪新闻', '搜狐新闻', '腾讯新闻', '网易新闻', '凤凰资讯', '参考消息',  '澎湃新闻', '厦门新闻', '社会资讯', '生活服务', '国际动态', '国内民生', '汽车新闻',  ];  foreach ($cates as $cate) {  /\*\*  \* @noinspection SqlResolve  \*/  DB::insert(  "INSERT INTO `category`(`name`,`pinyin`) VALUE (?,?)",  [$cate, implode('', pinyin($cate, null))]  );  }  return "category 数据添加完成";  }  // 添加 article 测试数据  public function addArticle()  {  $faker = app(Faker\Generator::class);  for ($i = 0; $i < 5000; $i++) {  $cate\_id = random\_int(1, 14);  $title = $faker->sentence(mt\_rand(1, 3));  $keywords = join(",", $faker->words(mt\_rand(3, 5)));  $description = join(",", $faker->sentences(mt\_rand(1, 2)));  $pubdate = $faker->dateTimeBetween('-6 month', '+6 days')->getTimestamp();  $click = random\_int(1000, 10000);  $writer = $faker->name;  $source = $faker->name;  $is\_show = random\_int(0, 1);  $litpic = $faker->imageUrl();  $content = join("\n\n", $faker->paragraphs(mt\_rand(4, 6)));  /\*\*  \* @noinspection SqlResolve  \*/  DB::insert(  "  INSERT INTO `article`(`cate\_id`,`title`,`keywords`,`description`,`pubdate`,`click`,`writer`,`source`,`is\_show`,`litpic`,`content`)  VALUE (?,?,?,?,?,?,?,?,?,?,?)",  [$cate\_id, $title, $keywords, $description, $pubdate, $click, $writer, $source, $is\_show, $litpic, $content]  );  }  return "article 数据添加完成";  }  // 添加 tag 测试数据  public function addTag()  {  $faker = app(Faker\Generator::class);  $tags = [  "爱卡汽车", "爱卡汽车网", "宝沃汽车", "北京汽车", "标志图解", "纯电动汽车", "大众汽车", "电动汽车",  "东风汽车", "东南汽车", "丰田汽车", "福特汽车", "福田汽车", "共享汽车", "海马汽车", "红旗汽车", "环球汽车网",  "吉利汽车", "几何汽车", "江淮汽车", "江铃汽车", "理想汽车", "力帆汽车", "莲花汽车", "猎豹汽车", "林肯汽车",  "铃木汽车", "领克汽车", "哪吒汽车", "奇瑞汽车", "启辰汽车", "汽车", "汽车报价", "汽车标志", "汽车大全",  "汽车购置税", "汽车故障灯", "汽车简笔画", "汽车票", "汽车票查询", "汽车品牌", "汽车图片", "汽车网", "汽车销量",  "汽车摇号", "汽车仪表盘", "汽车之家", "三菱汽车", "太平洋汽车", "特斯拉汽车", "威马汽车", "蔚来汽车", "沃尔沃汽车",  "现代汽车", "小鹏汽车", "小汽车摇号", "新能源汽车", "易车网汽车", "长安汽车", "长城汽车", "众泰汽车"  ];  foreach ($tags as $tag) {  $name = $tag;  $pinyin = implode('', pinyin($tag, null));  $title = $faker->sentence(mt\_rand(3, 6));  $keywords = join(",", $faker->words(mt\_rand(3, 5)));  $description = join(",", $faker->sentences(mt\_rand(1, 2)));  $introduction = join(",", $faker->sentences(mt\_rand(1, 2)));  $pubdate = $faker->dateTimeBetween('-6 month', '+6 days')->getTimestamp();  $jump\_url = $faker->imageUrl();  /\*\*  \* @noinspection SqlResolve  \*/  DB::insert(  "  INSERT INTO `tag`(`name`,`pinyin`,`title`,`keywords`,`description`,`introduction`,`pubdate`,`jump\_url`)  VALUE (?,?,?,?,?,?,?,?)",  [$name, $pinyin, $title, $keywords, $description, $introduction, $pubdate, $jump\_url]  );  }  return "tag 数据添加完成";  }  // 添加 article category tag 中间表数据  public function addArticleCategoryTag()  {  // 查询所有的分类ID  /\*\*  \* @noinspection SqlResolve  \*/  $cates = DB::select("SELECT \* FROM `category`");  $cate\_ids = [];  foreach ($cates as $cate) {  $cate\_ids[] = $cate->id;  }  // 查询所有的标签ID  /\*\*  \* @noinspection SqlResolve  \*/  $tags = DB::select("SELECT \* FROM `tag`");  $tag\_ids = [];  foreach ($tags as $tag) {  $tag\_ids[] = $tag->id;  }  // 查询所有的文章  /\*\*  \* @noinspection SqlResolve  \*/  $articles = DB::select("SELECT \* FROM `article`");  $article\_ids = [];  foreach ($articles as $article) {  $article\_ids[] = $article->id;  }  // 添加 article category tag 中间表数据  /\*\*  \* 每篇文章有 3~5 个标签  \*/  ini\_set('memory\_limit', '2560M'); // 临时设置内存  set\_time\_limit(0); // 临时设置脚本最大执行时间为永不过期  foreach ($article\_ids as $article\_id) {  // 获取 category 的 ID  $cate\_id\_key = array\_rand($cate\_ids, 1);  $cate\_id = $cate\_ids[$cate\_id\_key];  // 获取 tag 的 IDs  $nums = random\_int(2, 4);  $temp\_tags = array\_rand($tag\_ids, $nums);  foreach ($temp\_tags as $temp\_tag) {  $tag\_id = $tag\_ids[$temp\_tag];  /\*\*  \* @noinspection SqlResolve  \*/  DB::insert(  'INSERT INTO article\_tag\_category (article\_id, cate\_id, tag\_id) VALUE (?, ?, ?)',  [$article\_id, $cate\_id, $tag\_id]  );  }  }  return "article category tag 中间表数据添加完成";  }  // 添加 user 数据  /\*\*  \* @noinspection SqlResolve  \*/  public function addUser()  {  $faker = app(Faker\Generator::class);  for ($i = 0; $i < 100; $i++) {  $username = $faker->name;  $email = $faker->unique()->safeEmail;  $email\_is\_verified = random\_int(0, 1);  $email\_verified\_at = now()->getTimestamp();  $pwd = md5('a5s7sh4u');  $mobile = $faker->phoneNumber;  $fax = $faker->phoneNumber;  $wechat = Str::random(10);  $gender = random\_int(0, 2);  $address = $faker->address;  $reg\_date = now()->getTimestamp();  $last\_login\_date = now()->getTimestamp();  DB::insert(  'INSERT INTO user (username, email, email\_is\_verified,email\_verified\_at,pwd,mobile,fax,wechat,gender,address,reg\_date,last\_login\_date) VALUE (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',  [$username, $email, $email\_is\_verified, $email\_verified\_at, $pwd, $mobile, $fax, $wechat, $gender, $address, $reg\_date, $last\_login\_date]  );  }  return "user 数据添加完成";  }  // redis 使用测试  public function testRedis()  {  $key = 'appname';  // Cache::set('appname','我是新的CMS系统');  if (Cache::has('appname')) {  $value = Cache::get($key);  var\_dump($value);  } else {  var\_dump('缓存数据不存在');  }  }  } |

通过路由来执行填充测试数据的方法：

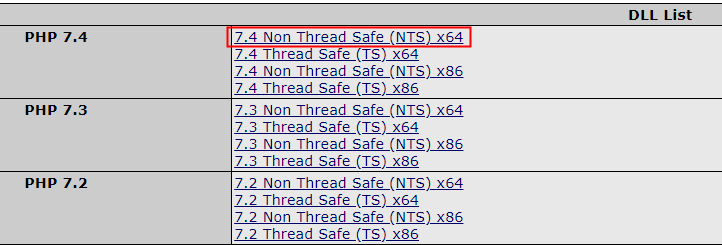
|  |
| --- |
| /\*\* 添加测试数据 \*/  Route::prefix('test')->name('test::date::')->group(function () {  Route::get('/category/', 'TestController@addCategory')->name('add::category');  Route::get('/article/', 'TestController@addArticle')->name('add::article');  Route::get('/tag/', 'TestController@addTag')->name('add::tag');  Route::get('/actmiddle/', 'TestController@addArticleCategoryTag')->name('add::actmiddle');  Route::get('/user/', 'TestController@addUser')->name('add::user');  Route::get('/redis/', 'TestController@testRedis')->name('redis');  }); |

### PHP安装 Redis 扩展

打开<https://pecl.php.net/package/redis> 下载 Windows 版本的Redis扩展，注意版本对应关系。



点击之后进入<https://pecl.php.net/package/redis/5.3.1/windows>页面，然后选择：



查看phpinfo()函数，可以看到如下信息：



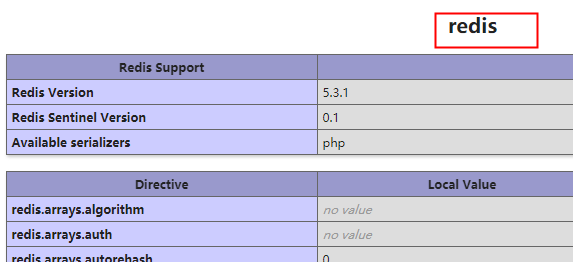
说明我们的PHP是NTS版本，所以上面下载的Redis扩展为NTS版本。

把扩展下载下来之后进行解压缩，然后把**php\_redis.dll**和**php\_redis.pdb**两个文件复制粘贴到PHP的ext目录下，遇到同名文件则覆盖即可。

然后在php.ini配置文件中开启redis扩展：extension=php\_redis.dll

然后重启下Apache或者Nginx服务器。

再次打印phpinfo()函数，在页面中看到如下信息：



说明我们的Redis库安装成功。

### 配置 Redis 缓存

首先，修改.env配置文件：

APP\_NAME=my\_blog 每个网站要取不同的名字

CACHE\_DRIVER=redis 如果是使用 redis 需要把这里修改为 redis

REDIS\_HOST=127.0.0.1 配置 redis 的相关信息

REDIS\_PASSWORD=a5s7sh4u

REDIS\_PORT=6379

然后修改config/cache.php 配置

'default' => env('CACHE\_DRIVER', 'redis'), 该配置项可以修改为 redis 也可以不修改

'redis' => [ 该配置项不用修改

'driver' => 'redis',

'connection' => 'default', 对应 config/database.php 中 redis 的 default 配置项

],

配置缓存键值对中键的前缀，可以根据需要修改。

'prefix' => env('CACHE\_PREFIX', Str::slug(env('APP\_NAME', 'laravel'), '\_').'\_cache'),

接着修改config/database.php 配置

'redis' => [ redis 的配置，注意，这里的 default 是对应上面的 connection' => 'default' 配置项

'cluster'=>false, 表示该 redis 不是集群

'default' => [

// 'url' => env('REDIS\_URL'),

'host' => env('REDIS\_HOST', '127.0.0.1'),

'password' => env('REDIS\_PASSWORD', null),

'port' => env('REDIS\_PORT', '6379'),

'database' => env('REDIS\_DB', '0'),

],

],

再接着，在控制器使用Cache代码：

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Cache;  class TestController extends Controller  {  public function test\_redis()  {  // dd('成功访问');  // phpinfo();  // Cache::set('name','我使用 Redis 作为 Cache 还曾',11000);  // Cache::set('age',38);  // Cache::set('collage','');  // Cache::put('collage', '大学');  // Cache::put('collage', '中学');  // $re = Cache::put('collage', '小学');  // dd($re);  Cache::putMany(['collage' => ['北京', '天津', '福建'], 'age' => 18, 'name' => 'orange']);  }  } |

在使用的时候，并没有注释 config/app.php 中的 'Redis' => Illuminate\Support\Facades\Redis::class, 且使用正常。

使用缓存的基本过程代码：

|  |
| --- |
| <?php  // 使用缓存的基本过程  $key = '3333';  if (Cache::has($key)){ //首先查寻 cache 的键是否存在  $values = Cache::get($key); //如果存在则直接读取 cache 里面的值  dd($values);  }else{ //如果 cache 里面没有，则先进行数据库查询操作，再放入缓存  $value = '4444';  Cache::put($key,$value,500);  }  dd(Cache::get($key)); |

### 配置后台路径

为了是后台路径不暴露出去，我们在.env中配置如下变量：

|  |
| --- |
| ADMIN\_DIR="zlslhxadmin" |

然后在config/app.php中添加配置：

|  |
| --- |
| 'admin\_dir' =>env('ADMIN\_DIR'), |

然后我们在web.php路由文件中就可以使用：

|  |
| --- |
| Route::prefix(config('app.admin\_dir')); |

### （可选）需要的时候再创建模型来关联表

在创建模型的同时创建迁移和控制器的命令格式：

|  |
| --- |
| php artisan make:model MySon -mc |
| Model created successfully.  Created Migration: 2020\_07\_20\_070524\_create\_my\_sons\_table  Controller created successfully. |

具体查看帮助命令：

|  |
| --- |
| php artisan make:model --help |

我们需要创建的模型包括：

|  |
| --- |
| php artisan make:model Admin/Category -mc  php artisan make:model Admin/Article -mc  php artisan make:model Admin/Tag -mc  php artisan make:model Admin/Topic -mc  php artisan make:model Admin/Search -mc  php artisan make:model Admin/Sysconf -mc |

### 前端操作

#### 展示分类列表

创建分类控制器

|  |
| --- |
| php artisan make:controller Home\CategoryController |

#### 展示文章和列表

创建文章控制器

|  |
| --- |
| php artisan make:controller Home\ArticleController |

在控制器中添加如下代码：

|  |
| --- |
| Http/Controllers/Home/ArticleController.php |
| <?php  namespace App\Http\Controllers\Home;  use App\Http\Controllers\Controller;  use DB;  use App\Util\Tools;  class ArticleController extends Controller  {  // 显示 article 列表  /\*\* @noinspection SqlResolve \*/  public function list($cid, $pid = 1)  {  // 查询分类  $category = DB::select('select \* from category where id = ?', [$cid])[0];  $category = Tools::object\_to\_array($category);  // 查询文章列表  $psize = 10; // 每页显示 10 条数据  $start = ($pid - 1) \* $psize;  $articles = DB::select("select id,title from article where cate\_id = {$cid} order by id limit {$start},10");  $articles = Tools::object\_array\_to\_array($articles);  return view('home.article.list', ['articles' => $articles, 'category' => $category]);  }  // 显示 article 详情页  /\*\* @noinspection SqlResolve \*/  public function detail($aid, $pid = 1)  {  $article = DB::select('select \* from article where id = ?', [$aid])[0];  $article = Tools::object\_to\_array($article);  $category = DB::select('select \* from category where id = ?', [$article['cate\_id']])[0];  $category = Tools::object\_to\_array($category);  return view('home.article.detail', ['article' => $article, 'category' => $category]);  }  } |

然后添加路由器：

### 后端操作

#### 后台登陆验证

### 全局变量

在Laravel中的全局变量是在config目录下创建的。

### 数据库表结构

#### 表结构长度限制问题

[首先解决字符长度错误问题](#_数据库迁移)。

#### 创建模型和迁移文件

|  |
| --- |
| php artisan make:model --migration Models\Article |
| articles |
| php artisan make:model --migration Models\ArticleTag |
| article\_tags |
| php artisan make:model --migration Models\Category |
| categories |
| php artisan make:model --migration Models\Search |
| searches |
| php artisan make:model --migration Models\Sysconf |
| sysconfs |
| php artisan make:model --migration Models\Tag |
| tags |
| php artisan make:model --migration Models\Topic |
| topics |

如果创建的模型不对，可以把相关的模型文件和迁移文件删除，然后重新执行上面的命令即可生成新的模型和迁移文件。

在所有模型文件中加入如下代码：

|  |
| --- |
| protected $dateFormat = 'U'; |

也就是统一用时间戳格式来保存所有日期时间。

#### 编写迁移文件代码

各模型迁移文件代码：

|  |
| --- |
| create\_articles\_table.php |
| Schema::create('articles', function (Blueprint $table) {  $table->integerIncrements('id')->nullable(false)->comment('主键ID');  $table->integer('cateid')->unsigned()->nullable(false)->default(0)->comment('栏目ID');  $table->string('title')->nullable(false)->default('')->comment('文章标题');  $table->string('keywords')->nullable(false)->default('')->comment('文章关键词');  $table->string('description')->nullable(false)->default('')->comment('文章描述');  /\* 时间戳 \*/  $table->integer('pubdate')->nullable(false)->default(0)->comment('时间戳格式的发布时间');  $table->mediumInteger('click')->unsigned()->nullable(false)->default(0)->comment('访问量');  $table->string('writer')->nullable(false)->default('')->comment('作者');  $table->string('source')->nullable(false)->default('')->comment('来源');  $table->boolean('is\_show')->unsigned()->nullable(false)->default(1)->comment('是否展示: 1 展示 0 不展示');  $table->string('litpic')->nullable(false)->default('')->comment('缩略图');  $table->text('content')->comment('文章正文');  }); |
| create\_article\_tags\_table.php |
| Schema::create('article\_tags', function (Blueprint $table) {  // integer: int 长度默认 10  $table->integer('tid')->unsigned()->nullable(false)->default(0)->comment('tag标签ID');  $table->integer('aid')->unsigned()->nullable(false)->default(0)->comment('文章ID');  // smallInteger: smallint 长度默认 5  $table->smallInteger('typeid')->unsigned()->nullable(false)->default(0)->comment('栏目ID');  // 联合主键  $table->primary(['tid', 'aid']);  }); |
| create\_categories\_table.php |
| Schema::create('categories', function (Blueprint $table) {  $table->smallIncrements('id')->nullable(false)->comment('主键ID');  $table->integer('reid')->unsigned()->nullable(false)->default(0)->comment('父栏目ID');  $table->string('name')->nullable(false)->default('')->comment('栏目名称');  $table->string('pinyin')->nullable(false)->default('')->comment('栏目拼音');  $table->string('title')->nullable(false)->default('')->comment('栏目SEO标题');  $table->string('keywords')->nullable(false)->default('')->comment('栏目SEO关键词');  $table->string('description')->nullable(false)->default('')->comment('栏目SEO描述');  $table->string('introduction')->nullable(false)->default('')->comment('栏目简介');  $table->boolean('is\_show')->unsigned()->nullable(false)->default(1)->comment('是否展示: 1 展示 0 不展示');  $table->string('normal\_pic')->nullable(false)->default('')->comment('栏目图片');  $table->text('content')->comment('栏目内容');  }); |
| create\_searches\_table.php |
| Schema::create('searches', function (Blueprint $table) {  $table->integerIncrements('id')->nullable(false)->comment('主键ID');  $table->string('name')->nullable(false)->default('')->comment('搜索名称');  $table->string('jianpin')->nullable(false)->default('')->comment('搜索简拼');  $table->string('quanpin')->nullable(false)->default('')->comment('搜索全拼');  $table->string('title')->nullable(false)->default('')->comment('搜索标题');  $table->string('keywords')->nullable(false)->default('')->comment('搜索关键词');  $table->string('description')->nullable(false)->default('')->comment('搜索描述');  $table->string('introduction')->nullable(false)->default('')->comment('搜索简介');  $table->string('jump\_url')->nullable(false)->default('')->comment('搜索页跳转地址');  $table->timestamp('addtime')->nullable(false)->default(DB::raw('CURRENT\_TIMESTAMP'))->comment('搜索添加的时间戳');  }); |
| create\_sysconfs\_table.php |
| Schema::create('sysconfs', function (Blueprint $table) {  /\* smallIncrements: smallint 无符号 自动增长 长度为 5 \*/  $table->smallIncrements('id')->nullable(false)->comment('配置项的ID');  $table->string('varname')->nullable(false)->default('')->comment('配置项中文名称');  $table->string('info')->nullable(false)->default('')->comment('配置项文字说明');  $table->text('value')->comment('配置项的值');  }); |
| create\_tags\_table.php |
| Schema::create('tags', function (Blueprint $table) {  $table->integerIncrements('id')->nullable(false)->comment('主键ID');  $table->timestamp('addtime')->nullable(false)->default(DB::raw('CURRENT\_TIMESTAMP'))->comment('标签添加时间戳');  $table->string('name')->nullable(false)->default('')->comment('标签名称');  $table->string('jianpin')->nullable(false)->default('')->comment('标签简拼');  $table->string('quanpin')->nullable(false)->default('')->comment('标签全拼');  $table->string('title')->nullable(false)->default('')->comment('标签标题');  $table->string('keywords')->nullable(false)->default('')->comment('标签关键词');  $table->string('description')->nullable(false)->default('')->comment('标签描述');  $table->string('introduction')->nullable(false)->default('')->comment('标签简介');  $table->string('jump\_url')->nullable(false)->default('')->comment('标签页跳转地址');  /\* 设置索引 \*/  $table->index(['jianpin'], 'tagjp');  $table->index(['quanpin'], 'tagquanpin');  }); |
|  |
| create\_topics\_table.php |
| Schema::create('topics', function (Blueprint $table) {  $table->integerIncrements('id')->nullable(false)->comment('主键ID');  $table->smallInteger('cateid')->unsigned()->nullable(false)->default(0)->comment('栏目ID');  $table->string('title')->nullable(false)->default('')->comment('专题标题');  $table->string('keywords')->nullable(false)->default('')->comment('专题关键词');  $table->string('description')->nullable(false)->default('')->comment('专题描述');  $table->string('introduction')->nullable(false)->default('')->comment('专题简介');  $table->timestamp('pubdate')->nullable(false)->default(DB::raw('CURRENT\_TIMESTAMP'))->comment('时间戳格式的发布时间');  $table->mediumInteger('click')->unsigned()->nullable(false)->default(0)->comment('访问量');  $table->string('writer')->nullable(false)->default('')->comment('作者');  $table->string('source')->nullable(false)->default('')->comment('来源');  $table->boolean('is\_show')->unsigned()->nullable(false)->default(1)->comment('是否展示: 1 展示 0 不展示');  $table->string('litpic')->nullable(false)->default('')->comment('缩略图');  $table->text('content')->comment('文章正文');  }); |
| create\_users\_table.php |
| Schema::create('users', function (Blueprint $table) {  /\* 修改部分字段 \*/  $table->id()->nullable(false)->comment('主键ID');;  $table->string('username')->nullable(false)->default('')->comment('账号');  $table->string('email')->nullable(false)->default('')->unique()->comment('邮箱');  $table->timestamp('email\_verified\_at')->nullable();  $table->string('password')->nullable(false)->default('')->comment('密码');  $table->rememberToken();  $table->timestamps();  /\* 添加如下字段 \*/  $table->string('mobile')->nullable(false)->default('')->comment('手机');  $table->string('wechat')->nullable(false)->default('')->comment('微信');  $table->boolean('sexy')->unsigned()->nullable(false)->default(1)->comment('性别: 0 男 1 女');  $table->timestamp('reg\_date')->nullable(false)->default(DB::raw('CURRENT\_TIMESTAMP'))->comment('时间戳格式的注册时间');  //由于没法设置默认值，所以默认值定义为当前时间的时间戳  $table->timestamp('last\_login')->nullable(false)->default(DB::raw('CURRENT\_TIMESTAMP'))->comment('最近登录的时间戳');  }); |

#### 运行所有迁移文件

运行所有迁移文件进行表结构的创建：

|  |
| --- |
| **运行指定单个迁移文件，方便测试各种数据类型的字段的定义是否正确。** |
| php artisan migrate --path=/database/migrations/文件名  例如  php artisan migrate --path=/database/migrations/2020\_06\_18\_022555\_create\_articles\_table.php |
| **运行所有迁移文件** |
| php artisan migrate |

#### 编写模型文件代码

编写模型文件代码，把我们的模型和数据库的表进行一一对应。

|  |
| --- |
| app/Models/Article.php |
| class Article extends Model  {  /\* 把 DATATIME 转换成 TIMESTAMP \*/  protected $dateFormat = 'U';  /\* 取消自动维护 created\_at 和 updated\_at 字段 \*/  public $timestamps = false;  } |
| app/Models/ArticleTag.php |
|  |
| app/Models/Category.php |
| class Category extends Model  {  /\* 取消自动维护 created\_at 和 updated\_at 字段 \*/  public $timestamps = false;  } |
| app/Models/Search.php |
|  |
| app/Models/Sysconf.php |
|  |
| app/Models/Tag.php |
| class Tag extends Model  {  public $timestamps = false;  } |
| app/Models/Topic.php |
|  |
| app/User.php |
|  |

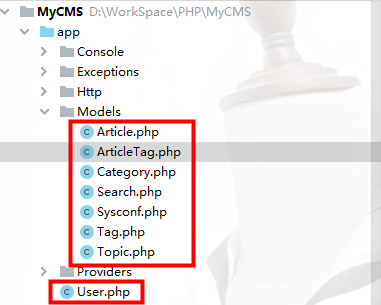
#### 编写数据填充代码

流程为：数据库填充器（调用） → 填充器（调用） → 模型工厂。

Faker填充字段：

<https://www.jianshu.com/p/d359e67fab91>

**步骤一**：创建填充器文件：创建填充器文件时，文件名可以随意定义，我喜欢在模型名称的基础上加上Seeder来命名，当前我们的模型名如下截图所示：

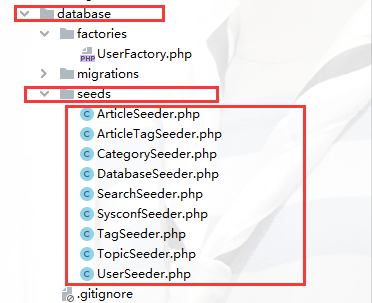


因此，我们创建填充器时，文件名如下代码所示：

|  |
| --- |
| php artisan make:seeder ArticleSeeder  php artisan make:seeder ArticleTagSeeder  php artisan make:seeder CategorySeeder  php artisan make:seeder SearchSeeder  php artisan make:seeder SysconfSeeder  php artisan make:seeder TagSeeder  php artisan make:seeder TopicSeeder  php artisan make:seeder UserSeeder |

填充器的文件名是**模型名 + Seeder**组合而成的。

代码执行完成后，填充器位置如下图所示：

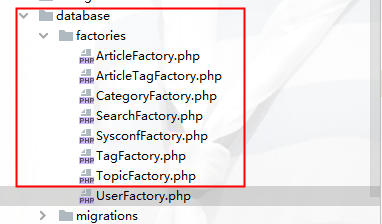


**步骤二**：创建模型工厂文件：创建模型工厂时，模型工厂文件的文件名也可以随意取，但是这里我们保持与填充器文件一样的命名规则，也就是由**模型名 + Factory**组合而成。

另外，由于UserFactory.php系统已经自带，我们就不用再次创建了，这里我们创建我们定义的其它模型的模型工厂文件，代码如下所示：

|  |
| --- |
| php artisan make:factory ArticleFactory  php artisan make:factory ArticleTagFactory  php artisan make:factory CategoryFactory  php artisan make:factory SearchFactory  php artisan make:factory SysconfFactory  php artisan make:factory TagFactory  php artisan make:factory TopicFactory |

代码执行之后，模型工厂文件位置如下图所示：



**步骤三**：编写填充器文件代码

|  |
| --- |
| ArticleSeeder.php |
| public function run()  {  \App\Models\Article::truncate();  factory(\App\Models\Article::class, 100)->create();  } |
| ArticleTagSeeder.php |
|  |
| CategorySeeder.php |
| class CategorySeeder extends Seeder  {  public function run()  {  \App\Models\Category::truncate();  factory(\App\Models\Category::class, 8)->create();  }  } |
| SearchSeeder.php |
|  |
| SysconfSeeder.php |
|  |
| TagSeeder.php |
| public function run()  {  \App\Models\Tag::truncate();  factory(\App\Models\Tag::class, 100)->create();  } |
| TopicSeeder.php |
|  |
| UserSeeder.php |
| public function run()  {  // 清空记录的同时把 ID 重置为 1  \App\User::truncate();  factory(\App\User::class, 10)->create();  } |

**步骤四**：编写模型工厂文件代码

|  |
| --- |
| ArticleFactory.php |
| use App\Models\Article;  use Faker\Generator as Faker;  $factory->define(Article::class, function (Faker $faker) {  return [  'cateid' => random\_int(1, 8),  'title' => $faker->sentence(mt\_rand(1, 3)),  'keywords' => join(",", $faker->words(mt\_rand(3, 5))),  'description' => join(",", $faker->sentences(mt\_rand(1, 2))),  /\* 把 DATETIME 对象转换成时间戳 \*/  'pubdate' => $faker->dateTimeBetween('-6 month', '+6 days')->getTimestamp(),  'click' => random\_int(1000, 10000),  'writer' => $faker->name,  'source' => $faker->name,  'is\_show' => random\_int(0, 1),  'litpic' => $faker->imageUrl(),  'content' => join("\n\n", $faker->paragraphs(mt\_rand(4, 6))),  ];  }); |
| ArticleTagFactory.php |
|  |
| CategoryFactory.php |
| use App\Models\Category;  use Faker\Generator as Faker;  use Overtrue\LaravelPinyin\Facades\Pinyin;  /\*  在 public 目录下创建 categories.txt 文件，  然后把下面八个分类名复制到 categories.txt 文件。  快速入门  核心架构  基础组件  前端开发  后端服务  安全系列  国内信息  海外风云  \*/  $factory->define(Category::class, function (Faker $faker, Array $cats) {  $cats\_file\_path = "public/categories.txt";  $cats = file($cats\_file\_path, FILE\_IGNORE\_NEW\_LINES | FILE\_SKIP\_EMPTY\_LINES);  $cat = $faker->randomElement($cats);  $cat\_key = array\_search($cat, $cats);  unset($cats[$cat\_key]);  file\_put\_contents($cats\_file\_path,implode(PHP\_EOL,$cats));  return [  'reid' => random\_int(0, 2),  'name' => $cat,  'pinyin' => implode('', pinyin($cat, null)),  'title' => $faker->sentence(mt\_rand(1, 3)),  'keywords' => join(",", $faker->words(mt\_rand(3, 5))),  'description' => join(",", $faker->sentences(mt\_rand(1, 2))),  'introduction' => join(",", $faker->sentences(mt\_rand(1, 2))),  'is\_show' => random\_int(0, 1),  'normal\_pic' => $faker->imageUrl(),  'content' => join("\n\n", $faker->paragraphs(mt\_rand(4, 6))),  ];  }); |
| SearchFactory.php |
|  |
| SysconfFactory.php |
|  |
| TagFactory.php |
| use App\Models\Tag;  use Faker\Generator as Faker;  $factory->define(Tag::class, function (Faker $faker) {  $tags\_file\_path = "public/tags.txt";  $tags = file($tags\_file\_path, FILE\_IGNORE\_NEW\_LINES | FILE\_SKIP\_EMPTY\_LINES);  $tag = $faker->randomElement($tags);  $tag\_key = array\_search($tag, $tags);  unset($tags[$tag\_key]);  file\_put\_contents($tags\_file\_path, implode(PHP\_EOL, $tags));  return [  'addtime' => $faker->dateTimeBetween('-6 month', '+6 days'),  'name' => $tag,  'jianpin' => pinyin\_abbr($tag),  'quanpin' => implode('', pinyin($tag, null)),  'title' => $faker->sentence(mt\_rand(3, 6)),  'keywords' => join(",", $faker->words(mt\_rand(3, 5))),  'description' => join(",", $faker->sentences(mt\_rand(1, 2))),  'introduction' => join(",", $faker->sentences(mt\_rand(1, 2))),  'jump\_url' => $faker->imageUrl(),  ];  }); |
| TopicFactory.php |
|  |
| UserFactory.php |
| $factory->define(User::class, function (Faker $faker) {  return [  'username' => $faker->name,  'email' => $faker->unique()->safeEmail,  'email\_verified\_at' => now(),  'password' => md5('a5s7sh4u'), // password  'remember\_token' => Str::random(10),  'mobile' => $faker->phoneNumber,  'wechat' => Str::random(10),  'sexy' => random\_int(0, 1),  'reg\_date' => now(),  'last\_login' => now(),  ];  }); |

**步骤五**：编写DatabaseSeeder.php代码

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

4、运行填充器代码

首先运行如下代码：

|  |
| --- |
| composer dump-autoload |

然后再运行如下代码：

|  |
| --- |
| 运行所有填充器，此时以database/seeds/DatabaseSeeder.php的run为入口 |
| php artisan db:seed |
| 运行指定填充器 |
| php artisan db:seed --class=ArticleSeeder  php artisan db:seed --class=ArticleTagSeeder  php artisan db:seed --class=CategorySeeder  php artisan db:seed --class=SearchSeeder  php artisan db:seed --class=SysconfSeeder  php artisan db:seed --class=TagSeeder  php artisan db:seed --class=TopicSeeder  php artisan db:seed --class=UserSeeder |

在执行如下代码：

|  |
| --- |
| php artisan db:seed --class=ArticleSeeder |

时，如果出错，提示没有设置created\_at和updated\_at字段的值，则需要修改模型文件，在模型文件中添加如下代码：

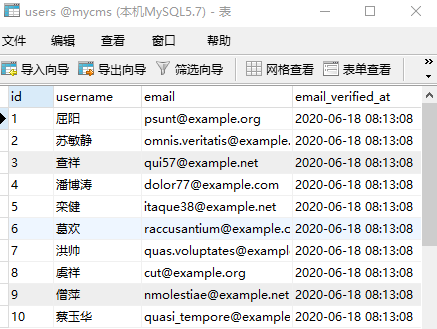
|  |
| --- |
| class Article extends Model  {  /\* 取消自动维护 created\_at 和 updated\_at 字段 \*/  public $timestamps = false;  } |

#### 配置中文填充数据

很简单，打开config/app.php文件，修改如下代码即可：

|  |
| --- |
| 'faker\_locale' => 'en\_US', |
| 修改为 |
| 'faker\_locale' => 'zh\_CN', |

这样就可以了，表数据如下所示：



### 后台管理

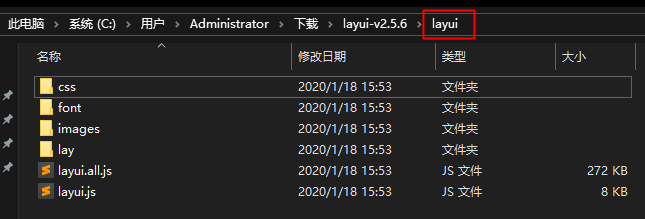
#### 后台UI框架

我们在后台使用LayUI框架。

LayUI地址为：<https://www.layui.com/>

文件下载地址：<https://res.layui.com/static/download/layui/layui-v2.5.6.zip?v=2>

下载下来进行解压，然后把解压后的layui文件夹，如下图所示：



复制并粘贴到项目的如下目录，如下图所示：



放在public目录下更方便访问。

#### 登陆界面

登陆的几种方式：

|  |  |  |
| --- | --- | --- |
| 账号、密码、验证码 | 微信登陆 | QQ登陆 |

#### 执行认证代码

打开Laravel官方文档：

<https://laravel.com/docs/7.x/authentication>

根据文档信息，需要在composer.json的require中添加如下代码：

|  |
| --- |
| "laravel/ui": "^2.0.3" |

然后执行更新操作：

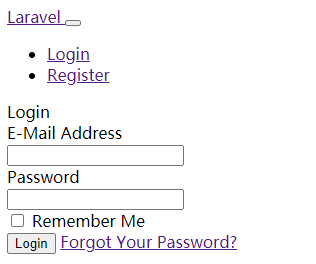
|  |
| --- |
| composer update |

然后执行如下命令：

|  |
| --- |
| php artisan ui vue --auth |

安装认证相关代码。

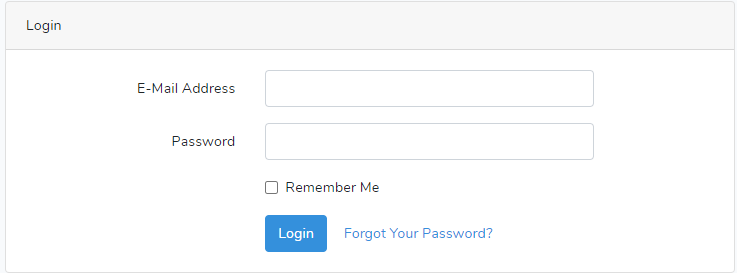
然后访问<http://www.blog.io/home> 界面显示如下：



说明我们的CSS没有起作用，这是因为没有找到对应的文件，我们需要执行如下代码：

|  |
| --- |
| npm run dev |

现在正常显示了，效果如下图所示：



#### 初始化账号

|  |
| --- |
| D:\WorkSpace\PHP\MyCMS>php artisan tinker  Psy Shell v0.10.4 (PHP 7.3.4 — cli) by Justin Hileman  >>> $user = new App\User;  => App\User {#3094}  >>> $user->username = 'shanhai'  => "shanhai"  >>> $user->email = 'shanhai@gaoding.com'  => "shanhai@gaoding.com"  >>> $user->password = md5('a5s7sh4u')  => "01fa9e493e3ca3bbb84aae31a35bab66"  >>> $user->save();  => true  >>> |

#### 注册和登陆修改为MD5加密

步骤一：

在app目录下创建Libraries目录，然后在Libraries目录下创建MD5.php类文件，该文件代码如下所示：

|  |
| --- |
| app/Libraries/MD5.php |
| <?php  namespace App\Libraries;  use Illuminate\Contracts\Hashing\Hasher;  class MD5 implements Hasher  {  public function make($value, array $options = [])  {  /\*\*  \* Hash the given value.  \*  \* @param string $value  \*  \* @return array $options  \* @return string  \*/  return md5($value);  }  /\*\*  \* Check the given plain value against a hash.  \*  \* @param string $value  \* @param string $hashedValue  \* @param array $options  \*  \* @return bool  \*/  public function check($value, $hashedValue, array $options = [])  {  // TODO: Implement check() method.  if (empty($hashedValue)) {  return true;  }  return $this->make($value) === $hashedValue;  }  /\*\*  \* Check if the given hash has been hashed using the given options.  \*  \* @param string $hashedValue  \* @param array $options  \*  \* @return bool  \*/  public function needsRehash($hashedValue, array $options = [])  {  // TODO: Implement needsRehash() method.  return false;  }  /\*\*  \* @inheritDoc  \*/  public function info($hashedValue)  {  // TODO: Implement info() method.  }  } |

步骤二：

在app/Providers目录下创建MD5ServiceProvider.php类文件，该文件代码如下所示：

|  |
| --- |
| app/Providers/MD5ServiceProvider.php |
| <?php  namespace App\Providers;  use Illuminate\Auth\EloquentUserProvider;  use Illuminate\Contracts\Hashing\Hasher as HasherContract;  class MD5ServiceProvider extends EloquentUserProvider  {  // 继承EloquentUserProvider类，调用父类的构造函数  public function \_\_construct(HasherContract $hasher, $model)  {  parent::\_\_construct($hasher, $model);  }  /\*\*  \* Bootstrap the application services.  \*  \* @return void  \*/  public function boot()  {  }  /\*\*  \* Register the application services.  \*  \* @return void  \*/  public function register()  {  }  } |

步骤三：

在app/Providers/AuthServiceProvider.php文件里面的boot方法里修改代码，过程如下：

|  |
| --- |
| app/Providers/AuthServiceProvider.php |
| use Auth;//导入  use App\Libraries\MD5;//导入 |
| public function boot()  {  $this->registerPolicies();  //  } |
| 修改为 |
| public function boot()  {  $this->registerPolicies();  //  Auth::provider('MD5', function ($app) {  $model = config('auth.providers.users.model');  return new MD5ServiceProvider(new MD5, $model);  });  } |

步骤四：

打开config/auth.php文件，修改providers的代码块：

|  |
| --- |
| config/auth.php |
| 'providers' => [  'users' => [  'driver' => 'eloquent',  'model' => App\User::class,  ], |
| 修改为 |
| 'providers' => [  'users' => [  // 'driver' => 'eloquent',  'driver' => 'MD5',  'model' => App\User::class,  ], |

到这里，我们注册登录就都可以使用MD5进行加密验证了。

#### 整个框架修改为MD5加密

步骤一：

在app下创建MD5目录，在MD5目录下创建MD5Hasher.php类文件，该文件代码如下：

|  |
| --- |
| app/MD5/MD5Hasher.php |
| <?php  namespace App\MD5;  use Illuminate\Contracts\Hashing\Hasher;  class MD5Hasher implements Hasher  {  public function info($hashedValue)  {  }  public function make($value, array $options = [])  {  return md5($value);  }  public function check($value, $hashedValue, array $options = [])  {  if (empty($hashedValue)) {  return true;  }  return $this->make($value) === $hashedValue;  }  public function needsRehash($hashedValue, array $options = [])  {  return false;  }  } |

步骤二：

在app/Providers目录下创建MD5HasherServiceProvider.php类文件，该文件代码如下所示：

|  |
| --- |
| app/Providers/MD5HasherServiceProvider.php |
| <?php  namespace App\Providers;  use Illuminate\Support\ServiceProvider;  use App\MD5\MD5Hasher;  class MD5HasherServiceProvider extends ServiceProvider  {  public function boot()  {  }  public function register()  {  $this->app->singleton('hash', function () {  return new MD5Hasher();  });  }  } |

步骤三：

打开config/app.php文件，在providers代码块进行如下修改：

|  |
| --- |
| 注释掉 |
| Illuminate\Hashing\HashServiceProvider::class, |
| 添加上 |
| App\Providers\MD5HasherServiceProvider::class, |

步骤四：测试

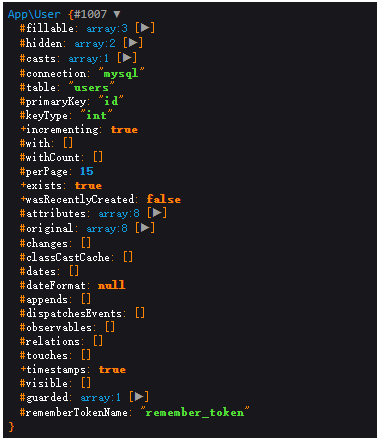
执行php artisan make:controller LoginController创建一个控制器，代码如下：

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Illuminate\Http\Request;  use Auth;  class LoginController extends Controller  {  public function login()  {  $input = [  'email' => 'shanhai@gaoding.com',  'password' => 'a5s7sh4u'  ];  Auth::attempt($input);  // 如果验证没有问题，会打印相应的用户信息  // 如果用户不存在会打印 null  // 如有用户存在会打印 App\User {#1007 ▶} 格式类似的信息  dd(Auth::user());  }  } |

然后在routes/web.php添加路由代码：

|  |
| --- |
| // 添加访问路由  Route::get('/login', 'LoginController@login'); |

接着在浏览器输入：<http://www.blog.io/login> ，浏览器显示如下信息：



说明访问没有问题，也就是说我们修改为MD5加密的过程是正确的。

#### 修改后台管理界面

既然后台的登陆已经没有问题了，那么接下来，我们开始更换后台的登陆界面。

在这里我们将使用LayUI。[LayUI安装过程](#_后台UI框架)在上面有说过。

下面开始使用哦。

# 常用扩展库

## Sitemap网站地图

<https://github.com/Laravelium/laravel-sitemap>