# Laravel框架\_02

# 今日目标13

* 能够使用$this->validate()方式完成数据验证

此方法写在控制器类中，前3个参数我们一般都要去写

第1参数 Request对象

第2参数 规则

第3参数 验证不通过的信息 [可选]

laravel5.6 返回验证字段的信息

* 能够使用Validator::make()方式进行数据验证
* 第1参数 Request对象
* 第2参数 规则
* 第3参数 验证不通过的信息 [可选]

Validator 在门面或外观中已经定义 use Validator;

返回是一个对象 fails()方法，验证不通过则返回true

如果没有通过 return redirect()->back()->withErrors($validate);

验证器

php artisan make:request 验证器

类中的第一个方法我们设置为 true

rules 规则

messages 信息

在获取控制器方法中依赖注入进来

* 能够了解在laravel框架的模板视图中输出验证失败后的错误提示

@if($errors->any())

@foreach($errors->all() as $error)

{{ $error }}

@endif

* 能够掌握在.env文件中配置数据库的名称、账号和密码
* 能够掌握在config/database.php文件中配置数据库相关信息
* 能够使用laravel框架中的DB功能类查询数据

查询多条 get()

查询 first()

insert

update

delete

where

whereIn

whereBetween

Orwhere

when(条件为真才执行匿名函数,Closure)

* 能够说出创建的迁移文件所在的目录

database/migrations

* 能够说出创建的种子文件所在的目录

database/seeds

* 能够使用php artisan make:migration命令创建迁移文件

php artisan make:migration 创建迁移文件

php artisan migrate 执行迁移文件

php artisan migrate:rollback 回滚最后一次执行的执行文件

php artisan migrate:refresh --seed

* 能够使用php artisan make:seeder命令创建种子文件(数据填充)

php artisan make:seeder 创建种子文件

php artisan db:seed [--class=类名]

php artisan migrate:refresh --seed

* 能够掌握在同一个迁移文件里面，设置多张表的结构
* 能够使用php artisan make:model 命令创建模型
* 能够使用laravel模型进行数据添加

# 一、表单验证

表单验证是为了防止访问者跳过客户端验证而造成的系统安全问题，一但非法用户绕过客户端验证而服务器端没有加以验证，这样就是很不安全了，所以项目必须要进行服务器端表单验证。

Laravel 提供了多种不同的验证方法来对应用程序传入的数据进行验证。

注：多个验证规则可以通过 "|" 字符进行隔开

|  |  |
| --- | --- |
| 规则名 | 说明 |
| required | 不能为空 |
| max:value | 字段值必须小于或等于value,对于字符串来说，value 为字符数 |
| min:value | 字段值必须大于或等于value,对于字符串来说，value 为字符数 |
| email | 验证邮箱是否合法 |
| url | 验证字段必需是有效的 URL 格式 |
| confirmed | 验证两个字段是否相同，如果验证的字段是password,则必须输入一个与之匹配的password**\_confirmation**字段 |
| integer | 验证字段必须是整型 |
| ip | 验证字段必须是IP地址 |
| numeric | 验证字段必须是数值 |
| size:value | value 验证字段必须有和给定值value想匹配的尺寸，对字符串而言，value是相应的字符数目，对数值而言，value是给定整型值；对文件而言，value是相应的文件字节数 |
| string | 验证字段必须是字符串 |
| unique | 表名，字段，需要排除的ID |
| between:min,max | 验证字段值的大小是否介于指定的 min 和 max 之间。字符串、数值或是文件大小的计算方式和 size 规则相同 |

## 1.1、添加用户验证

表单验证前，我们要对表单的csrf验证要加以处理



laravel写法

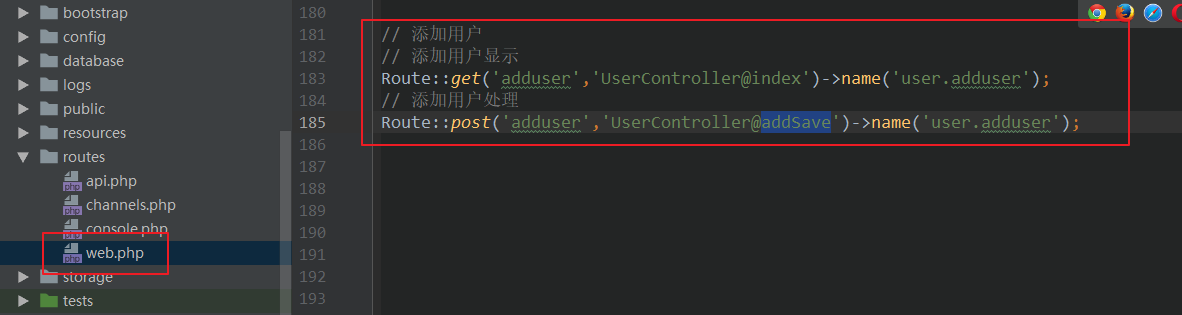
{{ csrf\_token() }} 只生成token值没有html代码 场景 ajax

{{ csrf\_field() }} 生成了token同时还生成了html代码 场景 一般的表单提交

laravel5.6之后我们对

{{ csrf\_field() }} 对它还可以简写 @csrf

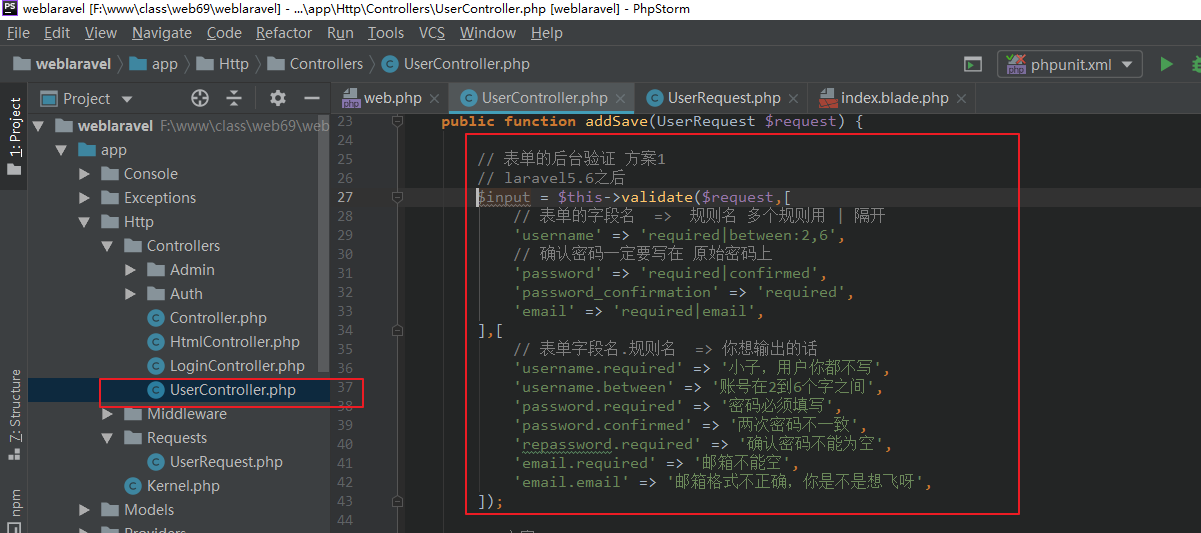
第1步：定义路由



方式一，用控制器中的$this->validate方式去验证表单

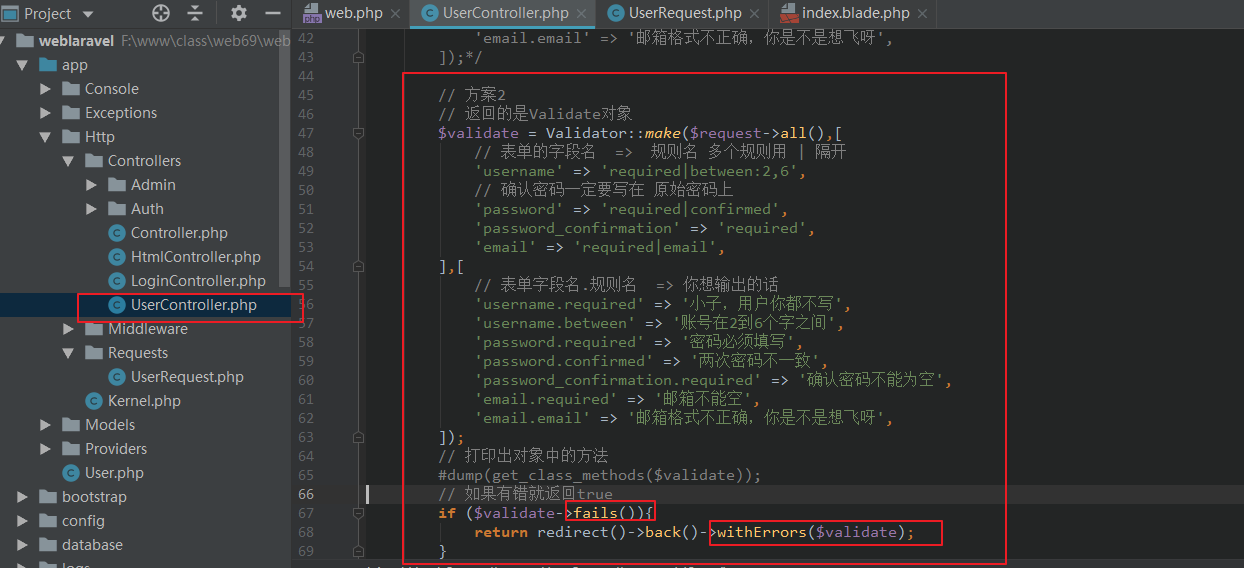
首先要有控制器

php artisan make:controller 控制器名



方式二，独立方式去验证表单

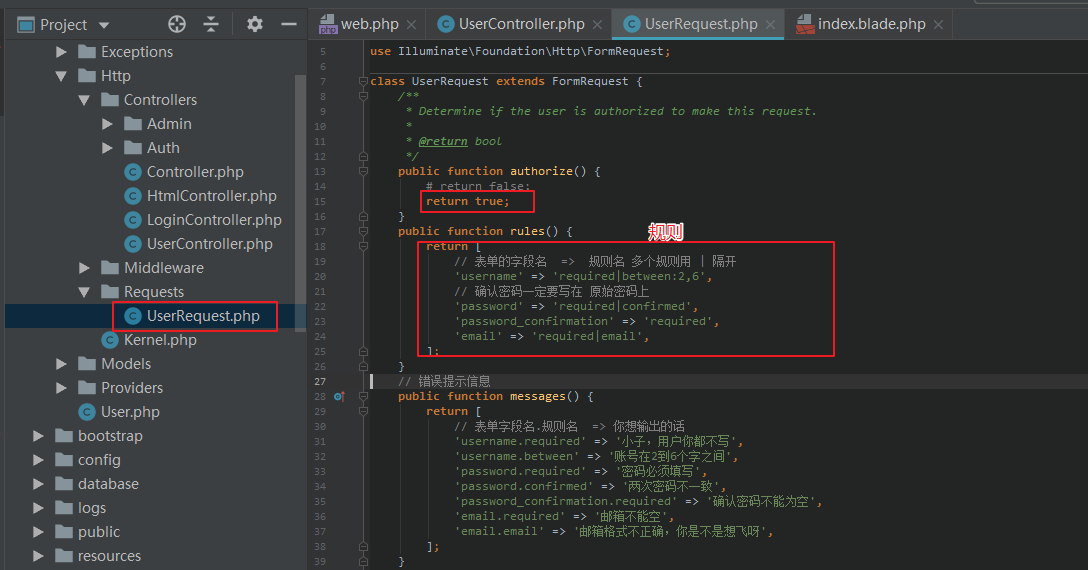




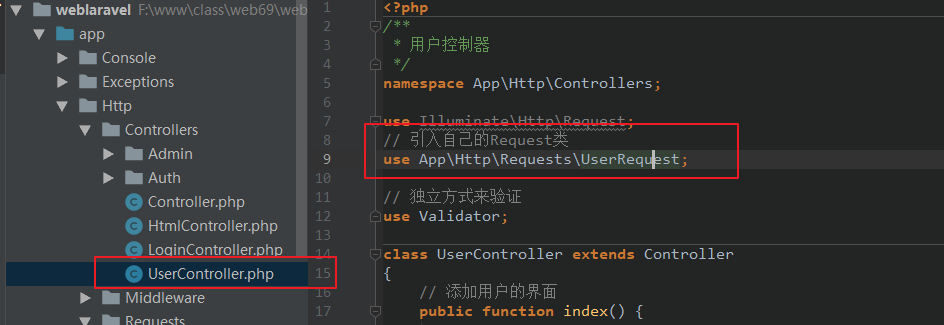
方式三、验证器

php artisan make:request 验证器的名称



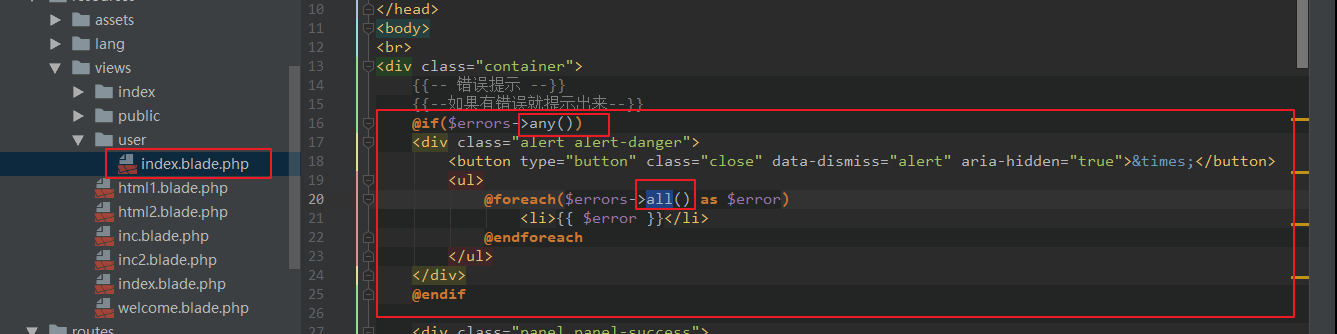


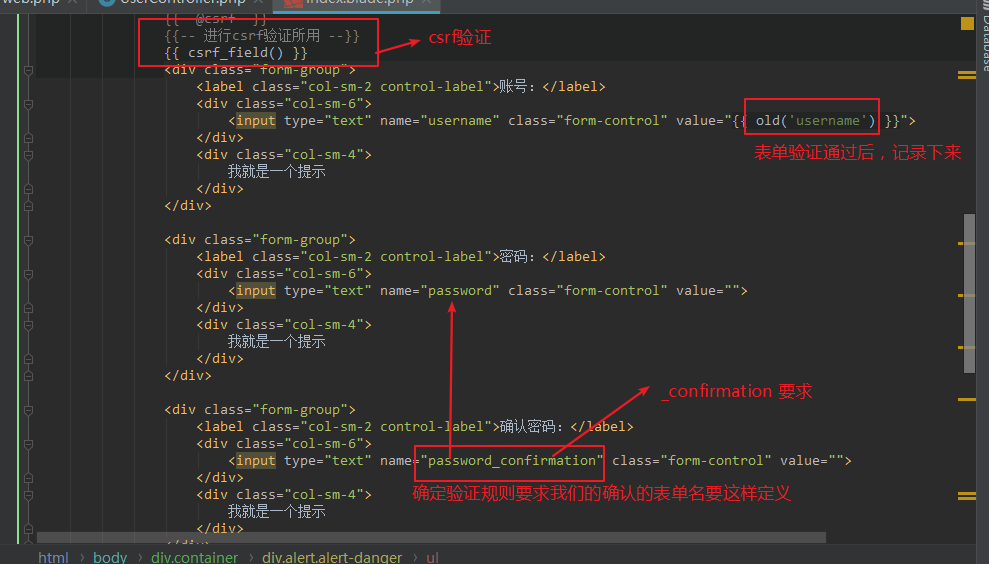
调用，控制器中用依赖注入的方式来使用



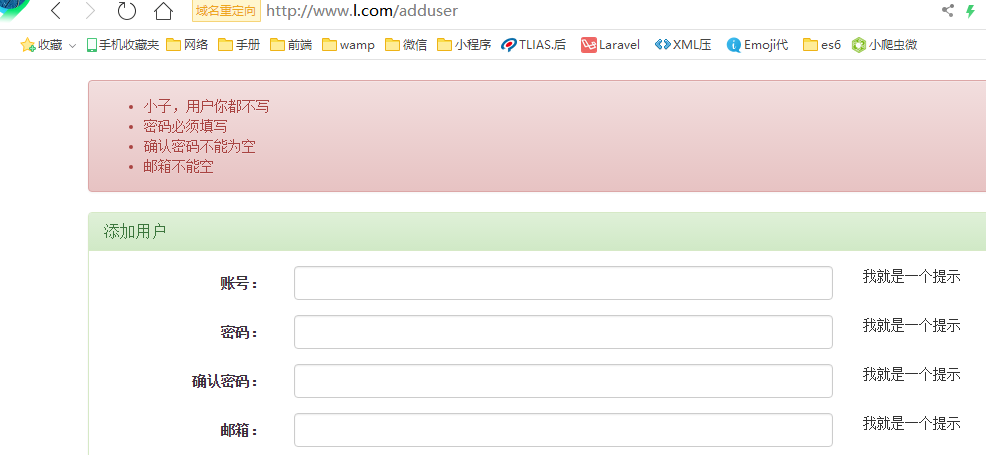


模板中是如何让错误信息输出



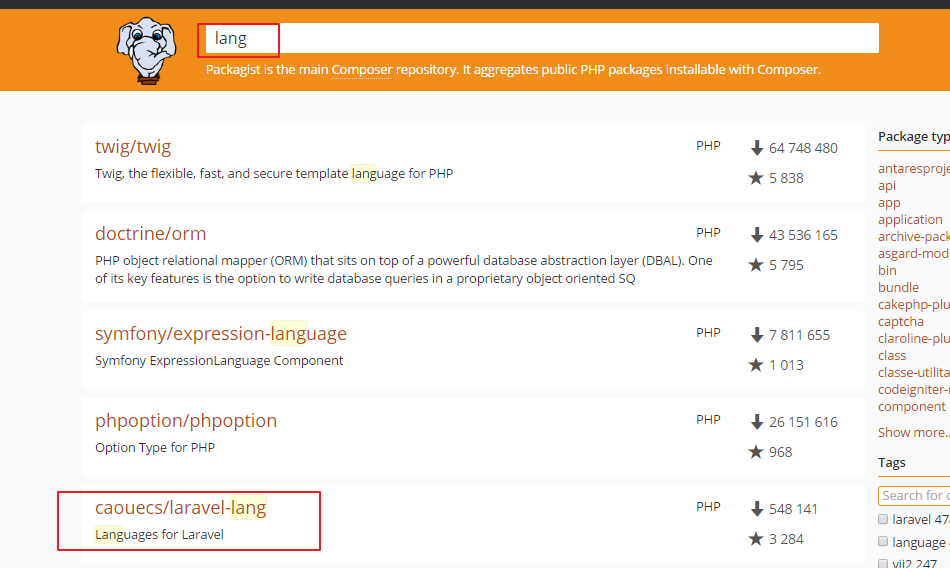


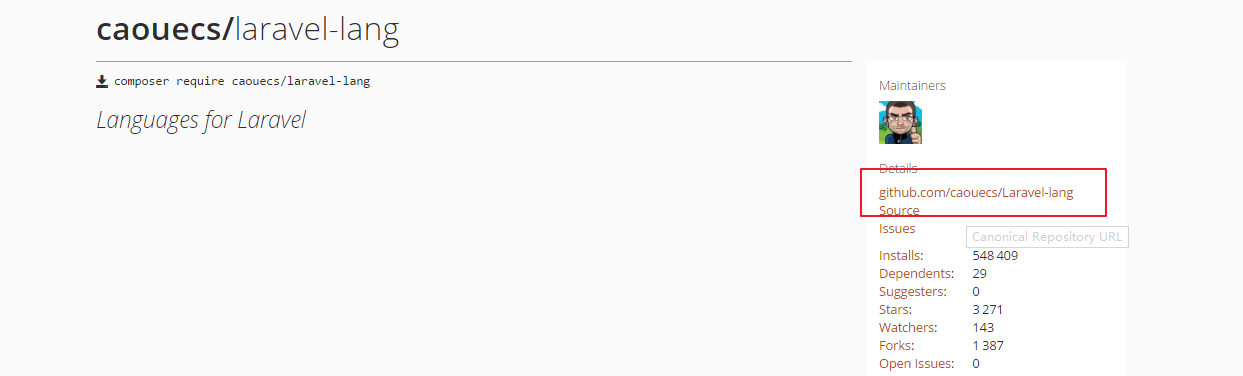
效果

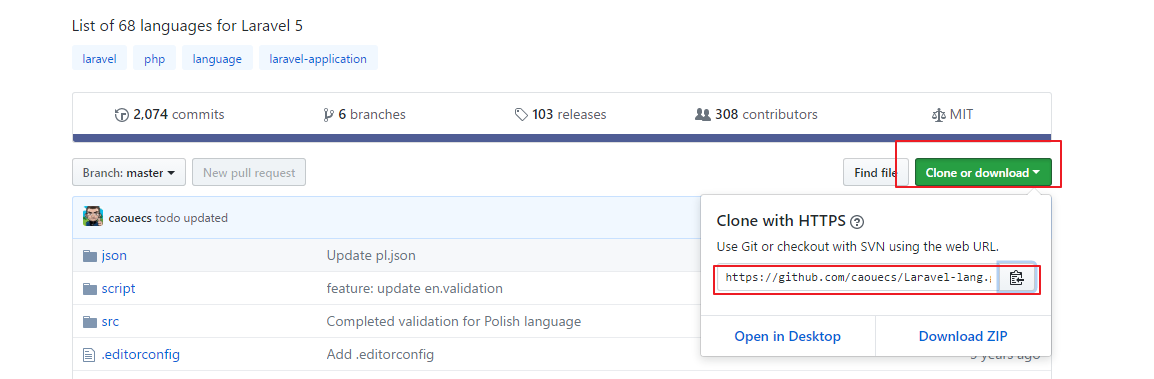


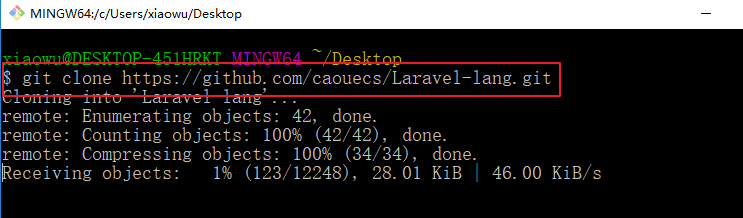
## 1.2、把默认信息换成中文

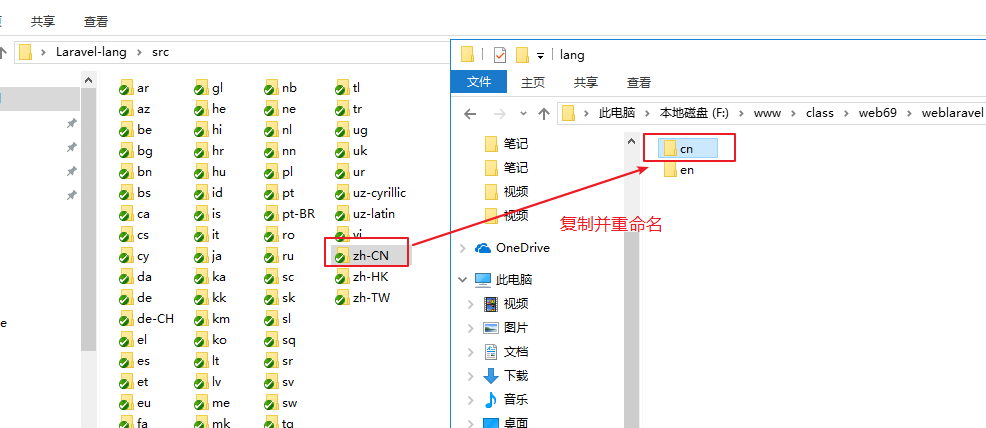
在此网站：<https://packagist.org>

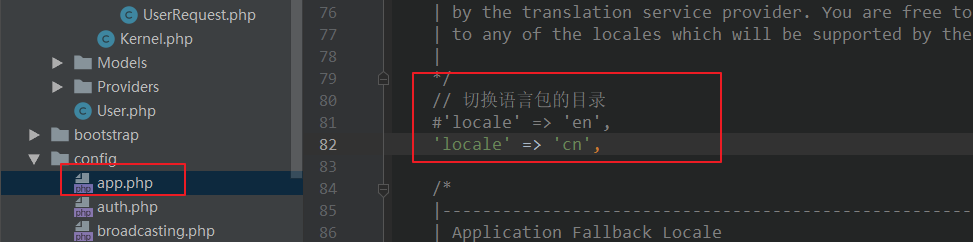


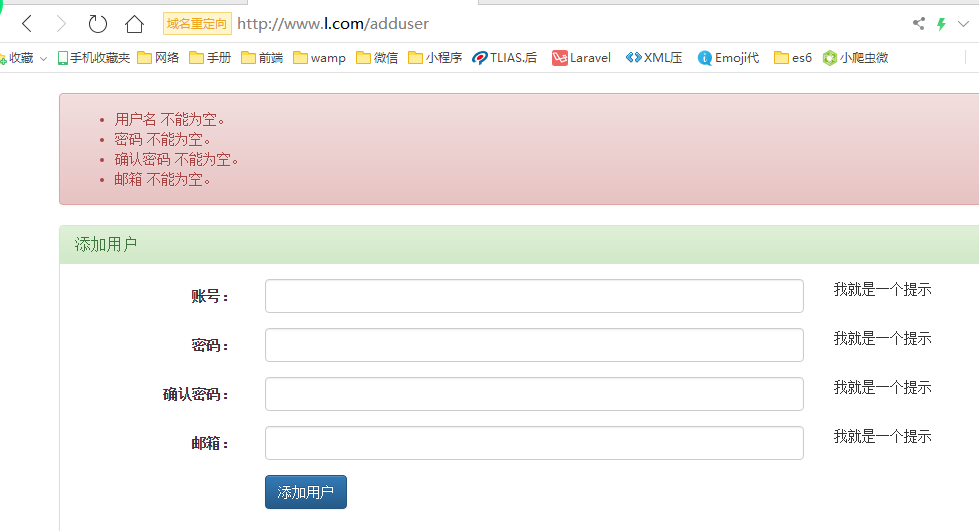












# 二、DB类操作数据库

## 2.1、laravel支持的数据库类型

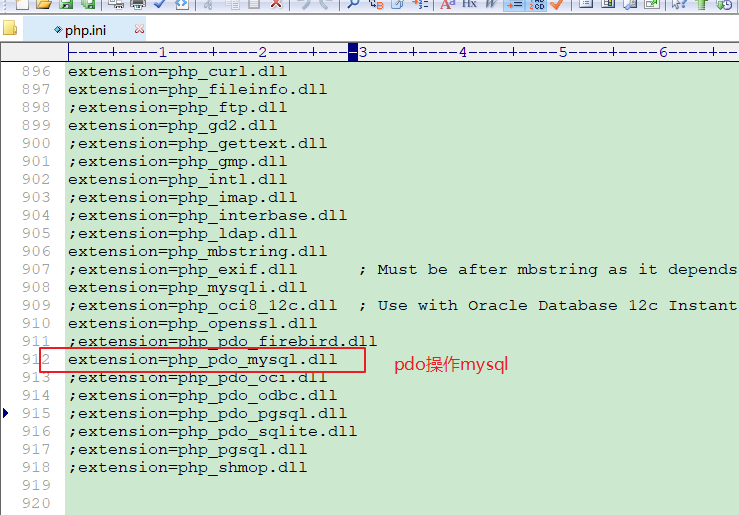
**MySQL**

Pgsql

SQLite

SQL Server

laravel操作数据库所使用的扩展 pdo，一定要开启pdo扩展



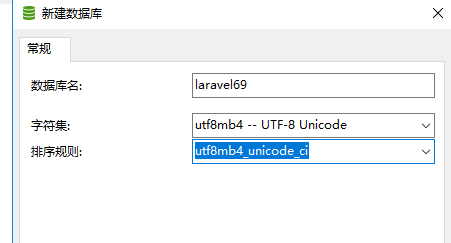
## 2.2、配置连接数据库信息

在laravel中修改连接数据库的文件有两处

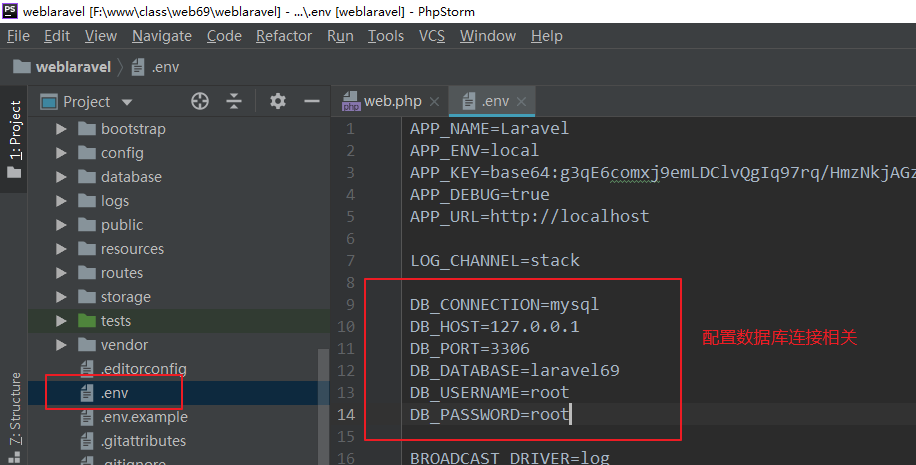
* 修改 .env文件
* 修改config/database.php文件。

laravel5.4之后对我们的数据库的编码就进行了默认改变，默认使用 utf8mb4

创建数据库



修改.env文件来连接数据库



通过配置后，发现，如果我们的数据表有前缀的话，在.env文件中不能去设置



到此我们的数据库的配置就完成了

注：表前缀，如果有则添加，没有可以不修改。

检查是否配置好





## 2.3、DB类执行原生SQL语句

// 添加操作

$ret = DB::insert('insert into test\_member (name,age,email) values (:name,:age,:email)', [':name'=>'张三',':age'=>20,':email'=>'admin@admin.com']);

// 修改操作

$ret = DB::update('update test\_member set age=:age where id=:id', [':age' => 12, ':id' => 1]);

// 查询 select

# 查询单条

$ret = DB::selectOne("select \* from web64\_user where id=:id",[':id'=>$id]);

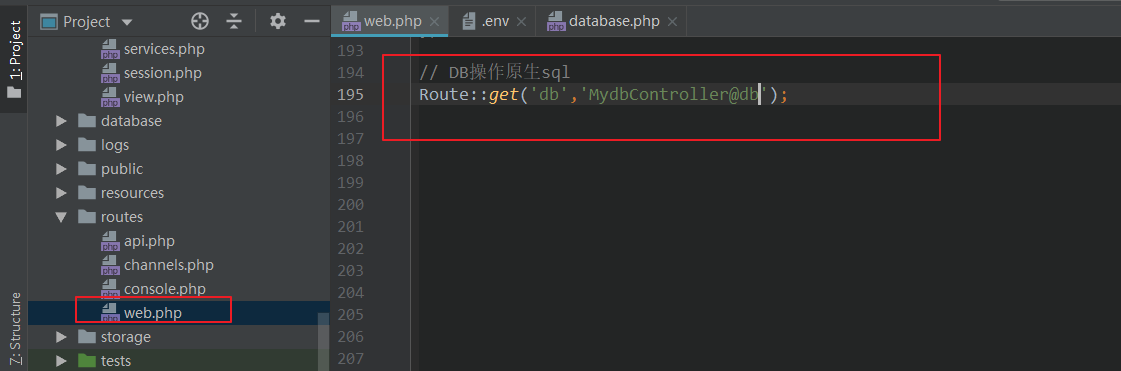
# 查询多条

$ret = DB::select("select \* from web64\_user");

// 删除操作

$ret = DB::delete("delete from test\_member where id=:id",[':id'=>1]);

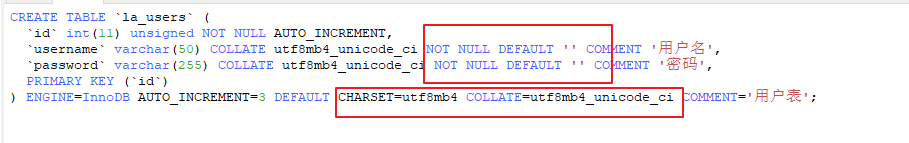
定义路由



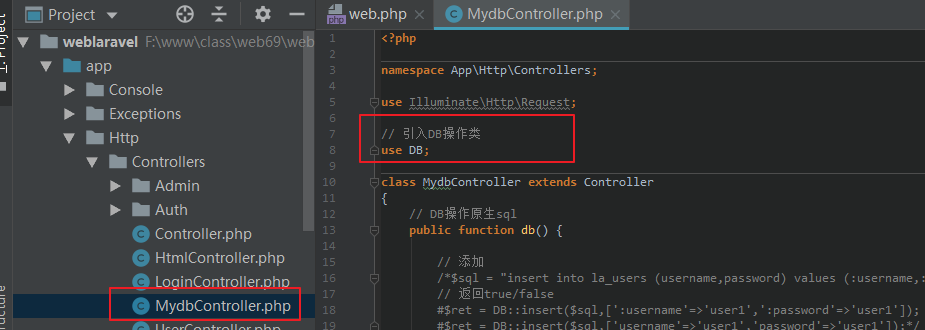
创建控制器

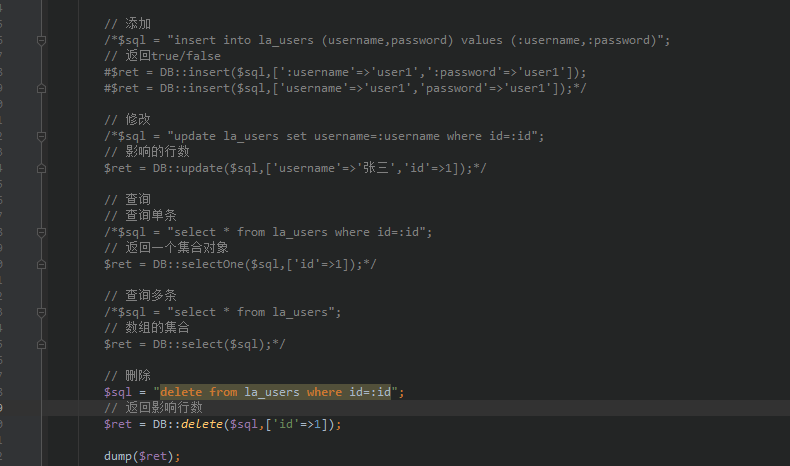
php artisan make:controller MydbController

数据表



控制器





## 2.4、DB构建器操作数据库

### 2.4.1、查询(get/first)

table(表名)，这里的表名，如查有前缀不用写。

# 查询多条 get all

# all不可以接where条件

// 查询所有无条件

$data = DB::table('user')->get(); // 默认 [‘\*’]

// 查询所有，并指定字段 推荐使第一种方式

$data = DB::table('user')->get(['name','age']);

// 获取ID大于5的数据

$ret = DB::table('member')->where('id','>=',5)->get();

// when 来进行条件的搜索

$kw = $request->get('kw','user');

// when 字段 如果为真则执行匿名函数中的操作

$ret = DB::table('users')->when($kw,function (Builder $query) use ($kw){

$query->where('username','like',"%{$kw}%");

})->get();

# 查询单条记录

$ret = DB::table('member')->where('id', 5)->first();

# 获取某个具体的值

$ret = DB::table('member')->where('id',5)->value('name');

# 获取一列数据

$ret = DB::table('member')->pluck('name');

# 以ID为下标 name为值

$ret = DB::table('member')->pluck('name', 'id');

# 排序操作

$ret = DB::table('member')->orderBy('id','desc')->get();

# 查询总记录数

echo DB::table('member')->count();

# 分页获取数据

$ret = DB::table('member')->orderBy('id','desc')->offset(0)->limit(2)->get();

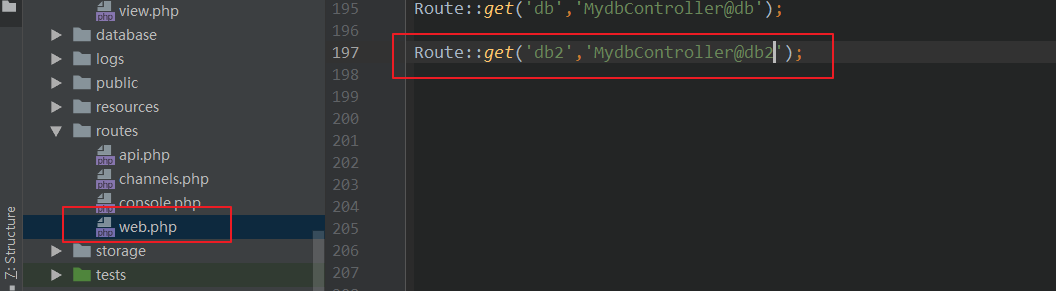
limit：表示限制输出的条数

offset：从什么地方开始,起始从0开始

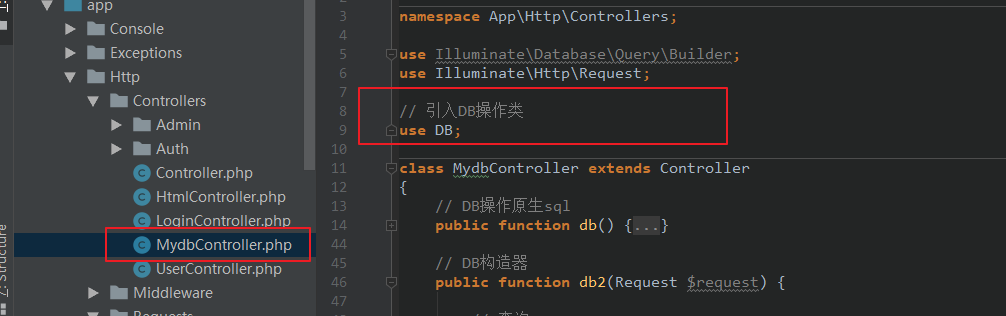
$ret = DB::table('users')->whereIn('id',[2,3,5])->get();

$ret = DB::table('users')->whereBetween('id',[3,5])->get();

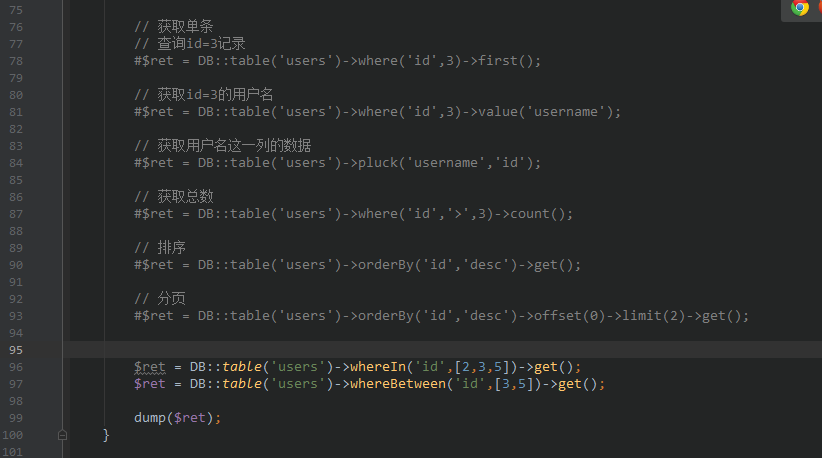
定义路由



控制器







### 2.4.2、添加数据操作(insert/ insertGetId)

对数据库中的某个表增加数据主要有两个方法可以实现，分别是insert()和insertGetId()

insert() 可以同时添加一条或多条，返回值是布尔类型。

insertGetId() 只能添加一条数据，返回自增的id。

// 添加数据

$ret = DB::table('member')->insert([

'name' => '张三丰',

'age' => 50,

'email' => 'ee@ee.com'

]);

// 添加数据并得到插入时的ID值

$id = DB::table('member')->insertGetId([

'name' => '小华',

'age' => 60,

'email' => 'ff@ff.com'

]);

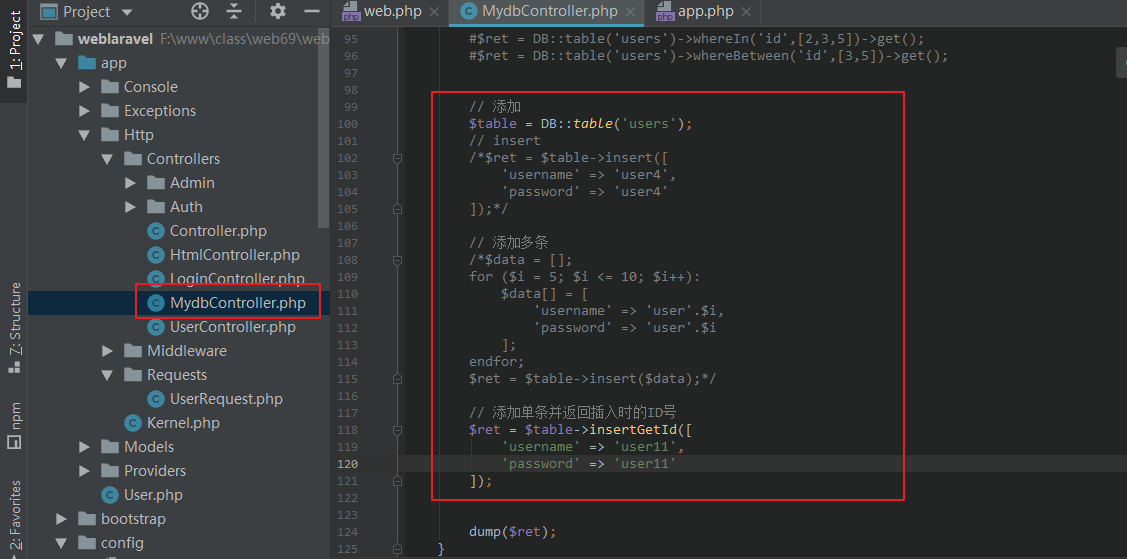
// 添加多条记录

$ret = DB::table('member')->insert([

['name'=>'AAAA','age'=>20,'email'=>'111@111.com'],

['name'=>'BBBB','age'=>30,'email'=>'222@222.com'],

]);



### 2.4.3、修改数据(update)

$ret = DB::table('member')->where('id', 2)->update([

'name' => '修改一下',

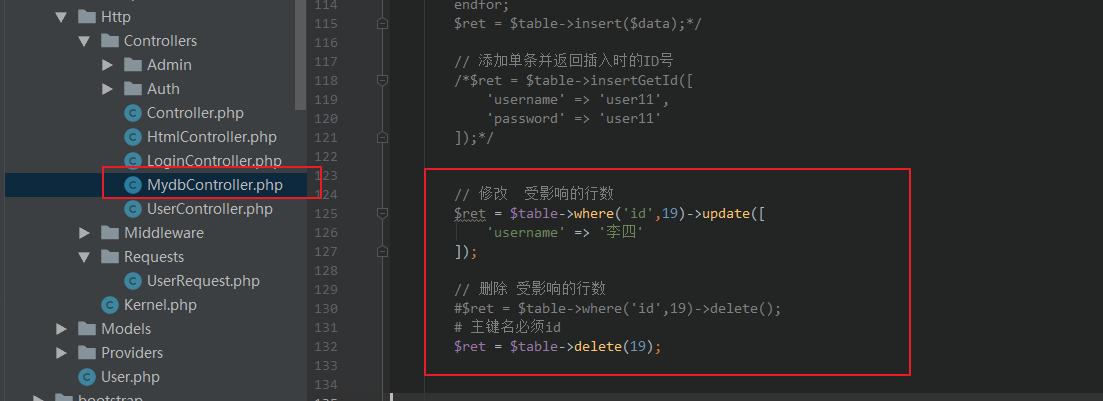
'age' => 50

]);

### 2.4.4、删除数据(delete)

// 删除数据

$ret = DB::table('member')->where('id',2)->delete();



# 三、数据库迁移与数据填充

## 3.1、数据库迁移

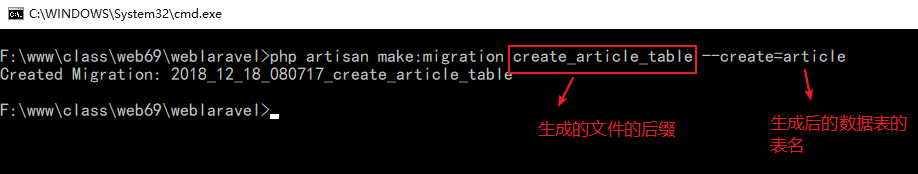
使用php文件编写的代码来进行数据库表结构的创建和修改。

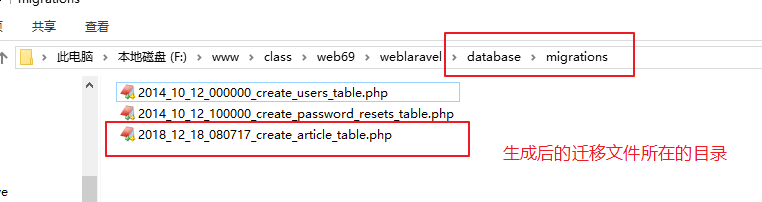
### 3.1.1、生成与编写迁移

php artisan **make:migration** create\_test\_table --create=test

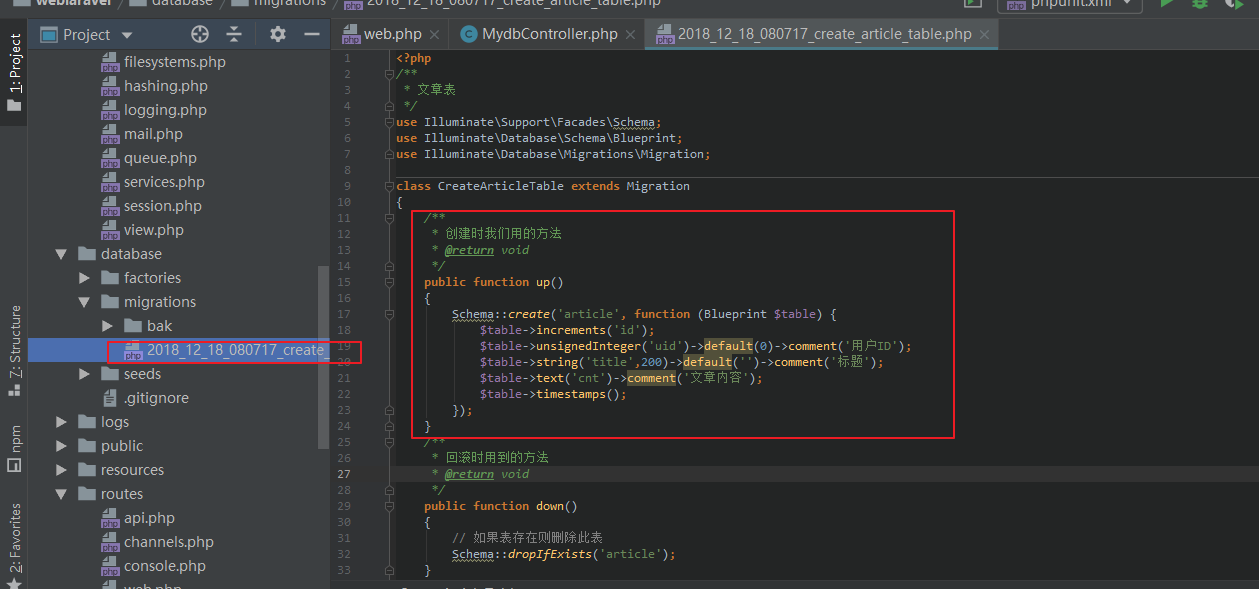
create\_test\_table 生成的文件后缀名称

--create=test 生成表名为test的数据表





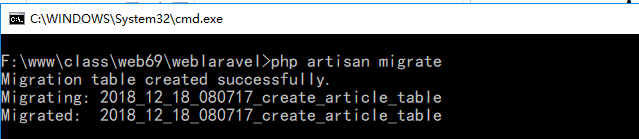
编写迁移文件，创建字段和表



### 3.1.2、执行数据迁移

php artisan migrate

# 直接执行迁移文件中的up方法



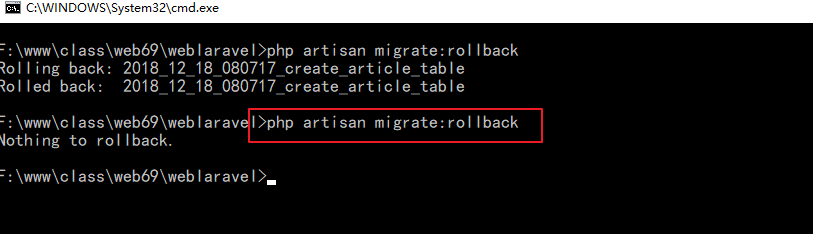


### 3.1.3、回滚迁移

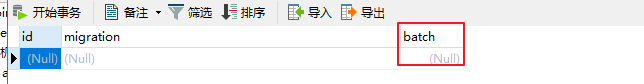
回滚最后一次的迁移操作, 删除（回滚）之后会删除迁移记录，并且数据表也会删除，但是迁移文件依旧存在，方便后期继续迁移（创建数据表）。

php artisan migrate:rollback

# 直接执行迁移文件中的 down方法



回滚条件，根据数据表中的

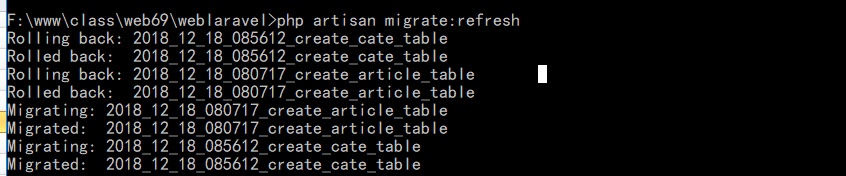


根据此值来找迁移文件，然后执行迁移文件中的down方法

### 3.1.4、清除表后再执行迁移

php artisan migrate:refresh

# 先执行所有的迁移文件的down方法，再执行所有的迁移文件的up方法



### 3.1.5、小结

* 创建

php artisan make:migration 文件名 --create=表名

* 编写迁移文件 [见文档]
* 执行

php artisan migrate

* 回滚

php artisan migrate:rollback

* 清除表并重新执行迁移

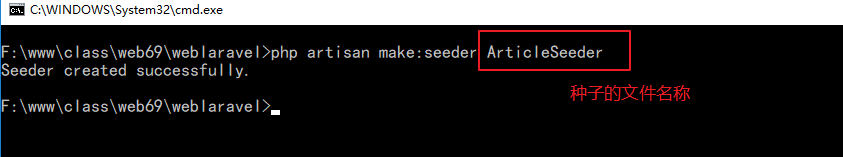
php artisan migrate:refresh

## 3.2、数据填充

填充操作就是往数据表中写测试数据的操作。

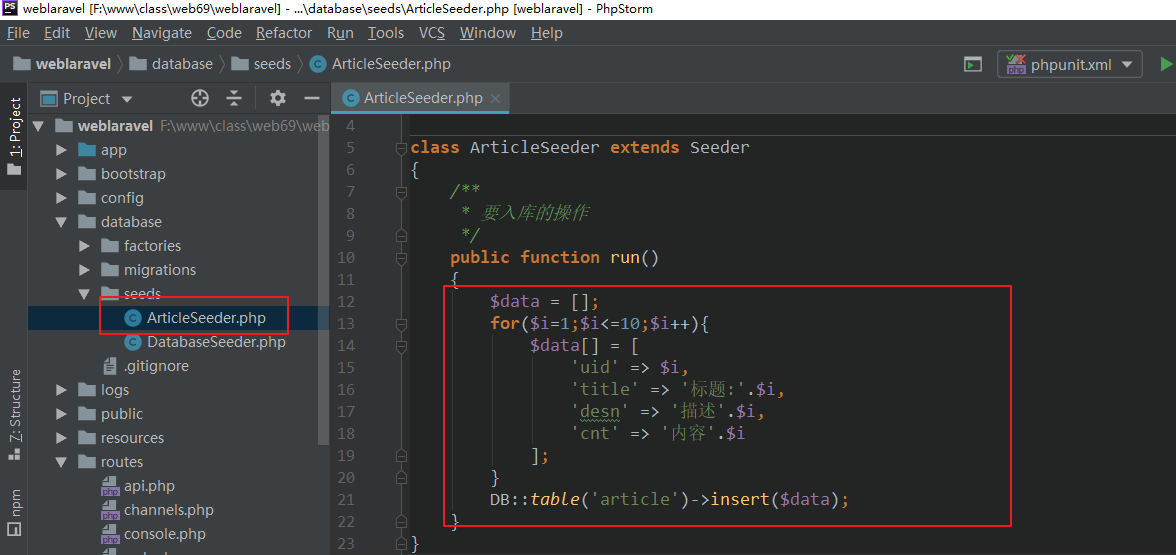
### 3.2.1、创建和编写种子文件

php artisan make:seeder TestTableSeeder{种子文件的名，文件名}





编写种子文件代码



### 3.2.2、执行种子文件

# 执行指定种子的文件

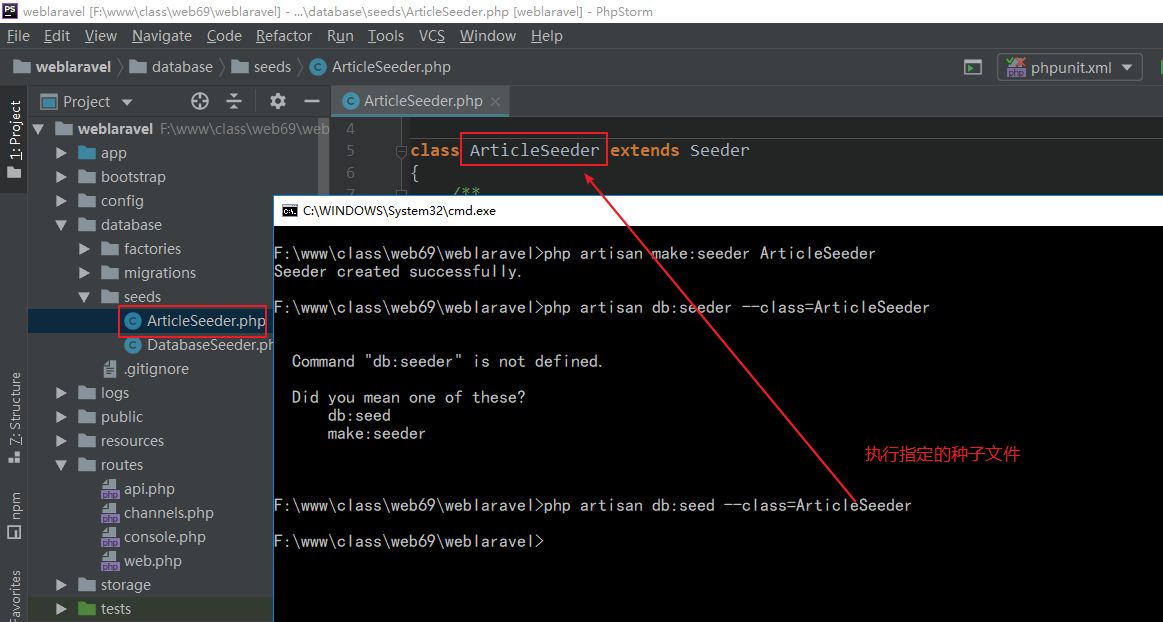
php artisan db:seed --class=需要执行的种子文件名

# 执行全部的种子文件

php artisan db:seed 必须在DatabaseSeeder文件中定义好 call

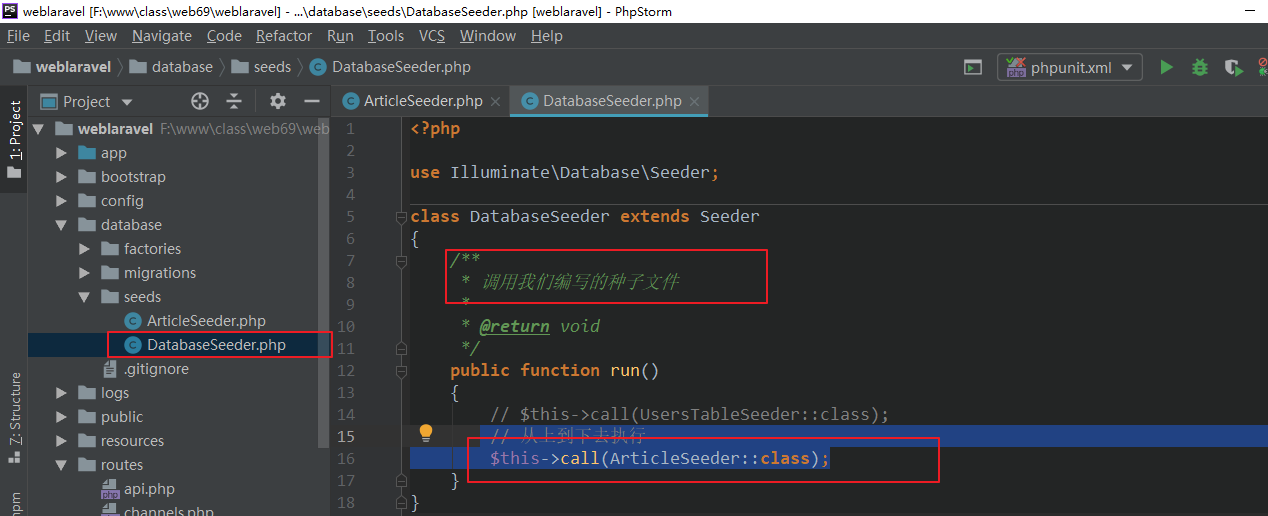
# 删除表并执行迁移和种子文件

php artisan migrate:refresh --seed 必须在DatabaseSeeder文件中定义好 call

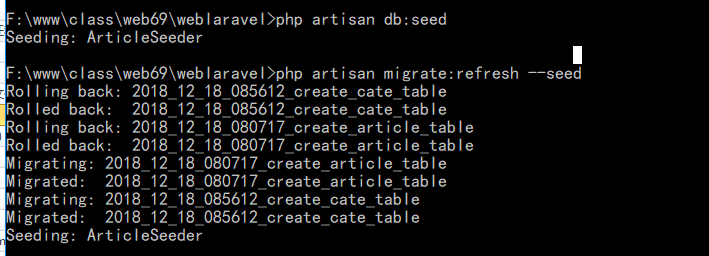


使用db:seed 执行不带参数

在DatabaseSeeder文件中run中的call方法定义我们的写的种子



这样就可以执行如下的两个命令



# 四、模型操作(Eloquent ORM)

Laravel 自带的 Eloquent ORM 提供了一个美观、简单的与数据库打交道的方案，每张数据表都对应一个与该表进行交互的“Model模型”，模型允许你在表中进行数据查询，以及插入、更新、删除等操作。

## 4.1、模型文件的位置

## 4.2、创建模型

php artisan **make:model** 模型文件名

# 分目录的

php artisan make:model Models/Member

# 创建模板并生成迁移文件

php artisan make:model Models/Article **–m**

-m 生成迁移文件

## 4.3、模型限制

### 4.3.1、定义表名($table)

模型所对应的默认表名是模型类名复数形式，如果表名不统一则需要指定一下。

// 指定表名

protected $table = 'member';

### 4.3.2、指定主键名($primaryKey)

laravel中默认的主键ID名为id，如果你创建的表字段中主键ID的名称不为id，则需要通过 $primaryKey 来指定一下。

// 指定主键的名称

protected $primaryKey = 'mid';

### 4.3.3、指定时间戳($timestamps)

默认情况下，模型操作会认为在你的数据库表有 created\_at 和 updated\_at 字段。如果你不希望让模型来自动维护这两个字段，可在模型内将 $timestamps 属性设置为 false。

// 时间戳 这里一定要注意它是用的public

**public** $timestamps = false;

### 4.3.4、批量赋值($fillable 与 $guarded)

当通过create方法来保存数据的时候，你需要先在你的模型上定义$fillable或 $guarded属性。

// 批量赋值

# 白名单

protected $fillable = ['允许添加的字段名'];

# 黑名单

protected $guarded = ['拒绝添加的字段名'];

## 4.4、CURD基本操作

### 4.4.1、添加数据

# 方法一

save() # 对象属性方式

# 方法二

insert() #数组方式

# 方法三

create() # 数组方式 需要在模型中设置好fillable或guarded属性

### 4.4.2、查询数据

# 查询单条

first()

# 查询多条

get()

# 条件

where()

# 排序

orderBy()

# 取指定的值

value()

# 取指定的列

pluck()

# 查询总记录数

count()

### 4.4.3、修改数据

# 方法一

save() # 对象属性方式

# 方法二

update() # 数组

### 4.4.4、删除数据

# 删除 一定要指定条件删除

delete()