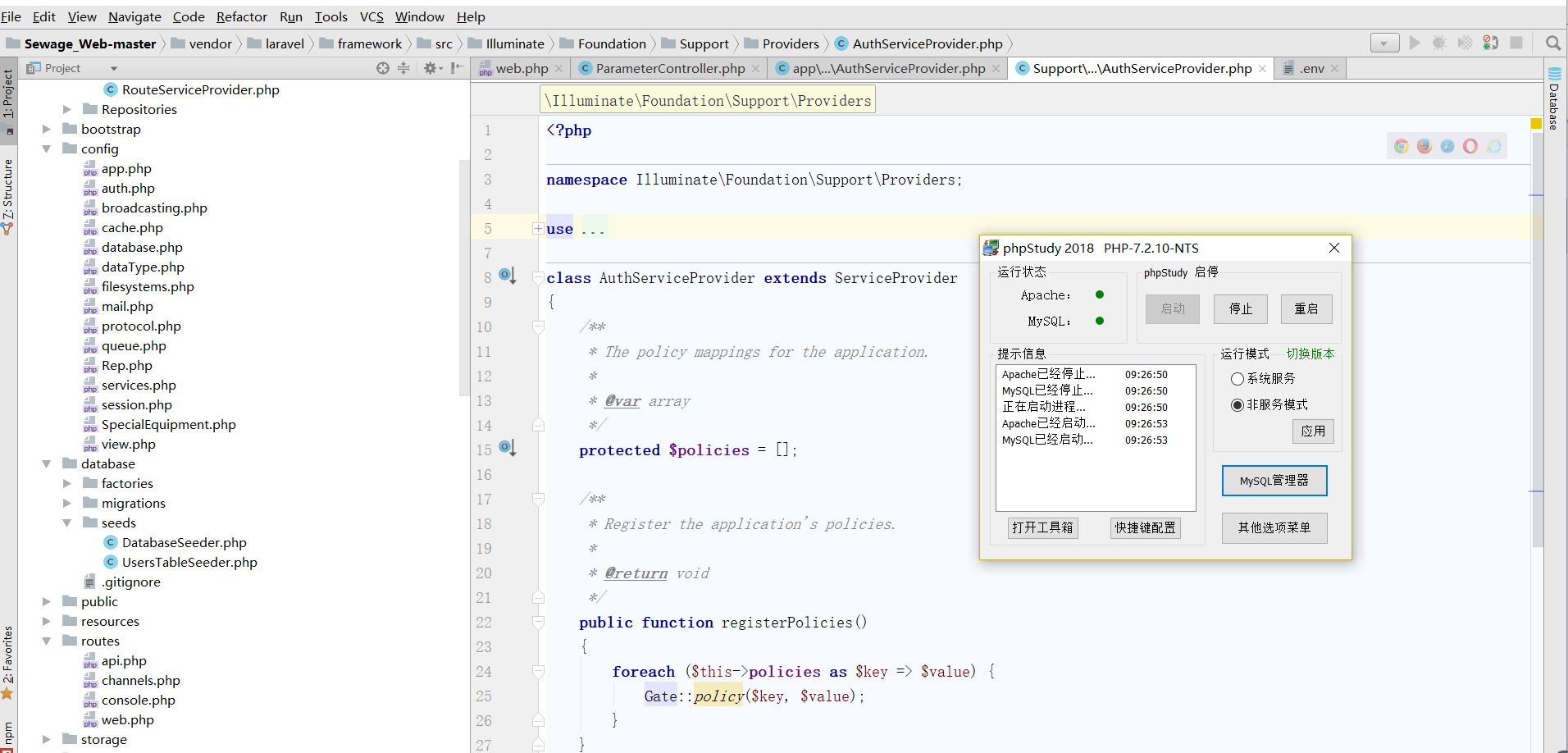
**实践周报告**

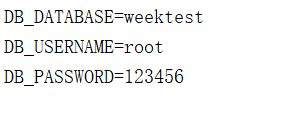
## 服务器搭建与项目本地配置

### 1.1本地服务器搭建：

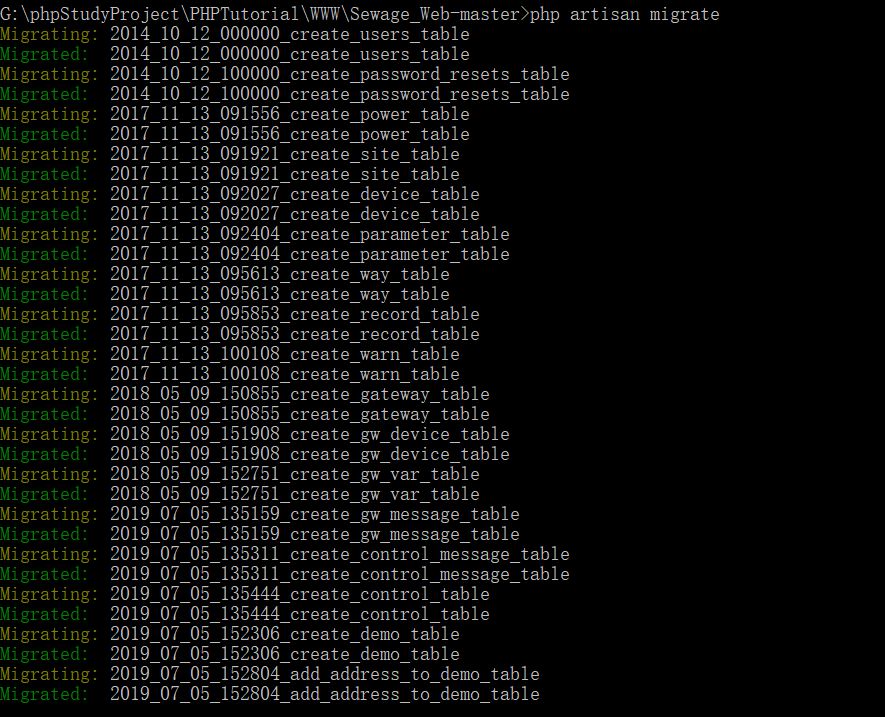
phpstudy+phpstorm

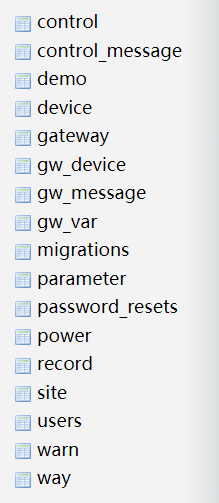


### 1.2数据库配置：

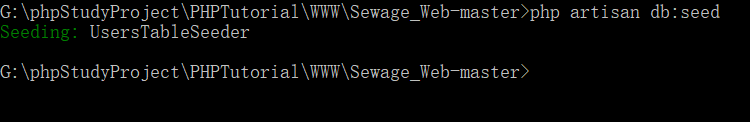


### 1.3 php artisan migrate数据库迁移

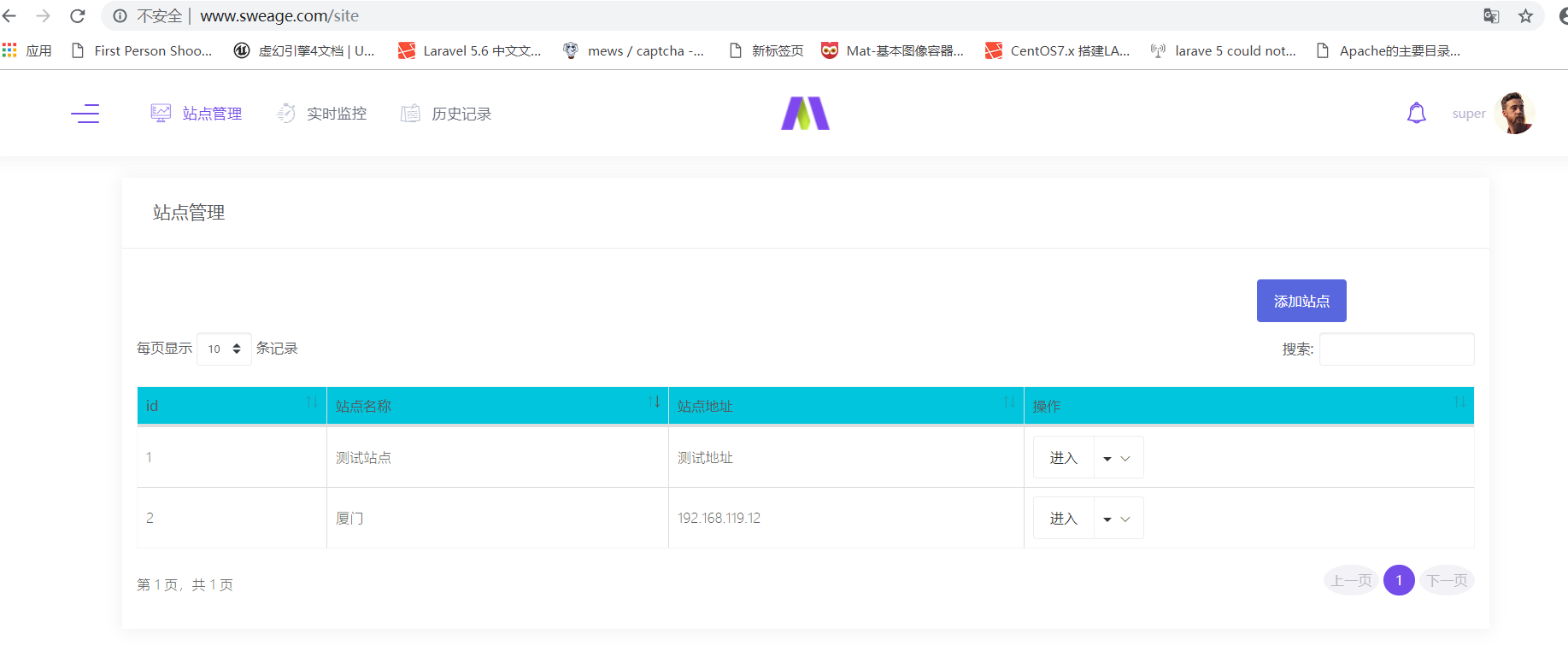




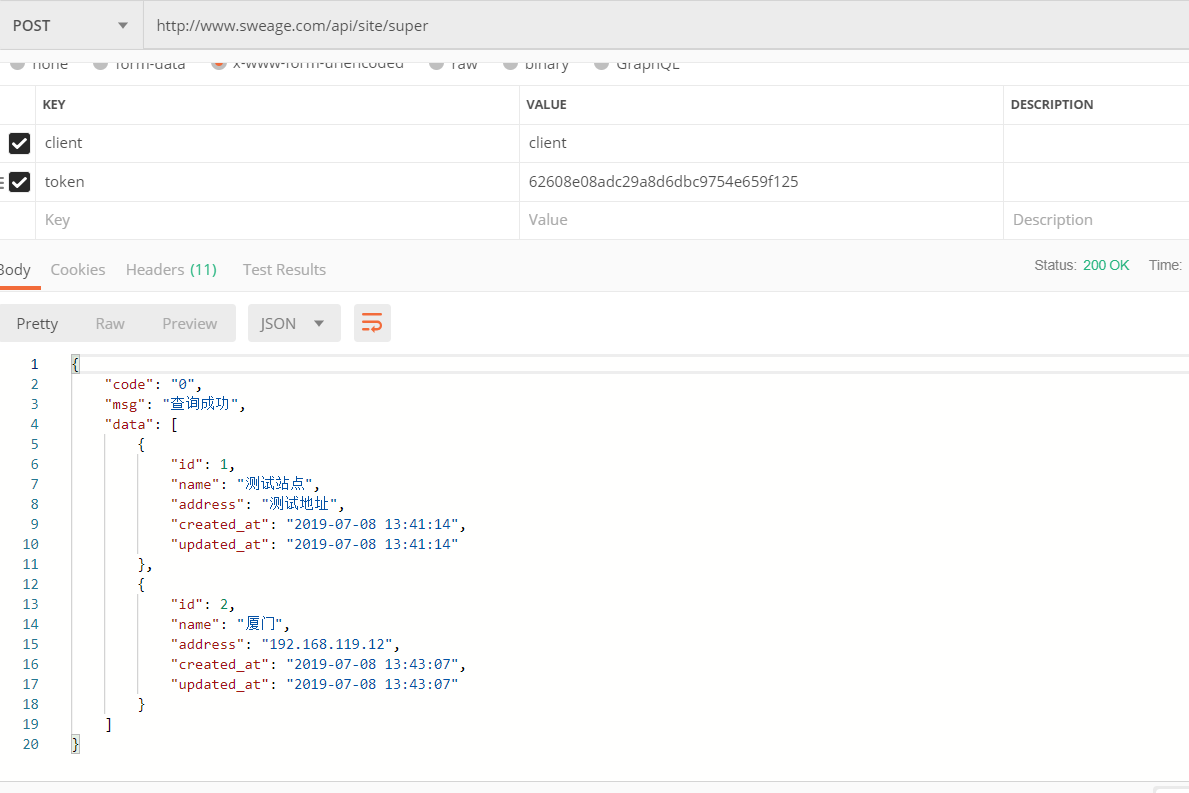
#### 1.31基本数据填充：



#### 1.32面效果：



### 1.4接口测试：



## 2.laravel使用Repository Pattern(仓库模式)实现数据逻辑处理分离

之前自学laravel框架代码写得都很随意，没有用过Repository Pattern(仓库模式)，下面就自己这段时间的学习理解，做个笔记。

### 2.1什么是仓库模式

在一般的框架中我们大多是用的是MVC模式将应用程序分层开发，View层是界面，Controller层是业务逻辑，Model层是数据库访问。这种分层结构对于小型的项目而言完全没有问题。但对于中大型的公司项目而言，大量的数据逻辑放在Model层会使项目变得臃肿，增删改查代码重复性高，耦合度高，不利于后期的项目测试。因此为了避免这些我们使用Repository模式在MVC的基础上独立出一个层,用于处理数据进行对的数据库的增删改查。

### 2.2仓库模式的好处

A：把数据处理逻辑代码独立成一个层，使得代码容易维护

B：减少代码重复

C：降低出错率

D:使代码更简洁明了

### 2.3仓库模式在laravel中的实现

如果我们想要在服务提供者中注册视图 composer 该怎么做？这就要用到 boot 方法了。**该方法在所有服务提供者被注册以后才会被调用**，这就是说我们可以在其中访问框架已注册的所有其它服务：

步骤：

1. 创建Interface类：

在Interface中我们定义了我们必须要实现的方法，一般就是我们对数据库进行增删改查的代码。在接口类中我们可以清晰的看到我们有的所有方法。

1. 创建repository类

在repository仓库类中我们继承了interface类用于实现具体的的方法。

1. 在服务容器中绑定Interface类和repository类
2. Controller依赖注入

代码如下：

**TestInterface**

<?php

/\*\*

\* Created by PhpStorm.

\* User: SWESWE

\* Date: 2019/7/3

\* Time: 9:28

\*/

namespace App\Repository\Interfaces;

interface TestInterface

{

public function findAll();

public function findOne($id);

public function test();

}

**TestRepository：**

<?php

/\*\*

\* Created by PhpStorm.

\* User: SWESWE

\* Date: 2019/7/3

\* Time: 9:28

\*/

namespace App\Repository\Repositories;

use App\Models\Test;

use App\Repository\Interfaces\TestInterface;

class TestRepository implements TestInterface {

public function findOne($id)

{

// TODO: Implement findOne() method.

return Test::find($id)->toArray();

}

public function findAll()

{

// TODO: Implement findAll() method.

return Test::all()->toArray();

}

public function test(){

echo"接口测试类方法，绑定接口到实现";

}

}

**Model:**

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class test extends Model

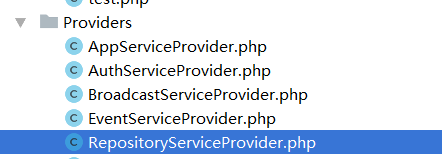
{

//

public $timestamps = false;

protected $table = 'test';

}

php artisan make:provider RepositoryServiceProvider  
在provider下生成一个新的Service文件  


在register方法中绑定接口类和仓库类public function register()

{

//绑定接口和测试类

//测试models

$this->app->bind(

'App\Repository\Interfaces\TestInterface',

'App\Repository\Repositories\TestRepository'

);

}



2.4效果：



### 2.4遇到问题

多个仓库类同时使用一个接口，如何识别每个仓库类

#### 2.41解决办法：

使用上下文绑定方法

$this->app->when(TestController::class)

->needs(TestInterface::class)

->give(TestRepository::class);

$this->app->when(TestController3::class)

->needs(TestInterface::class)

->give(TestRepository3::class);

#### 2.42效果：





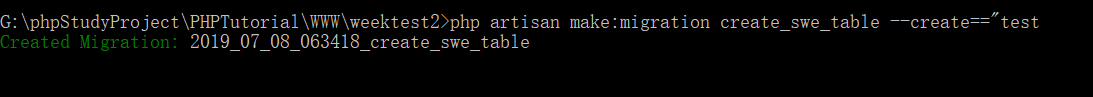
## 3.php artisan数据库迁移

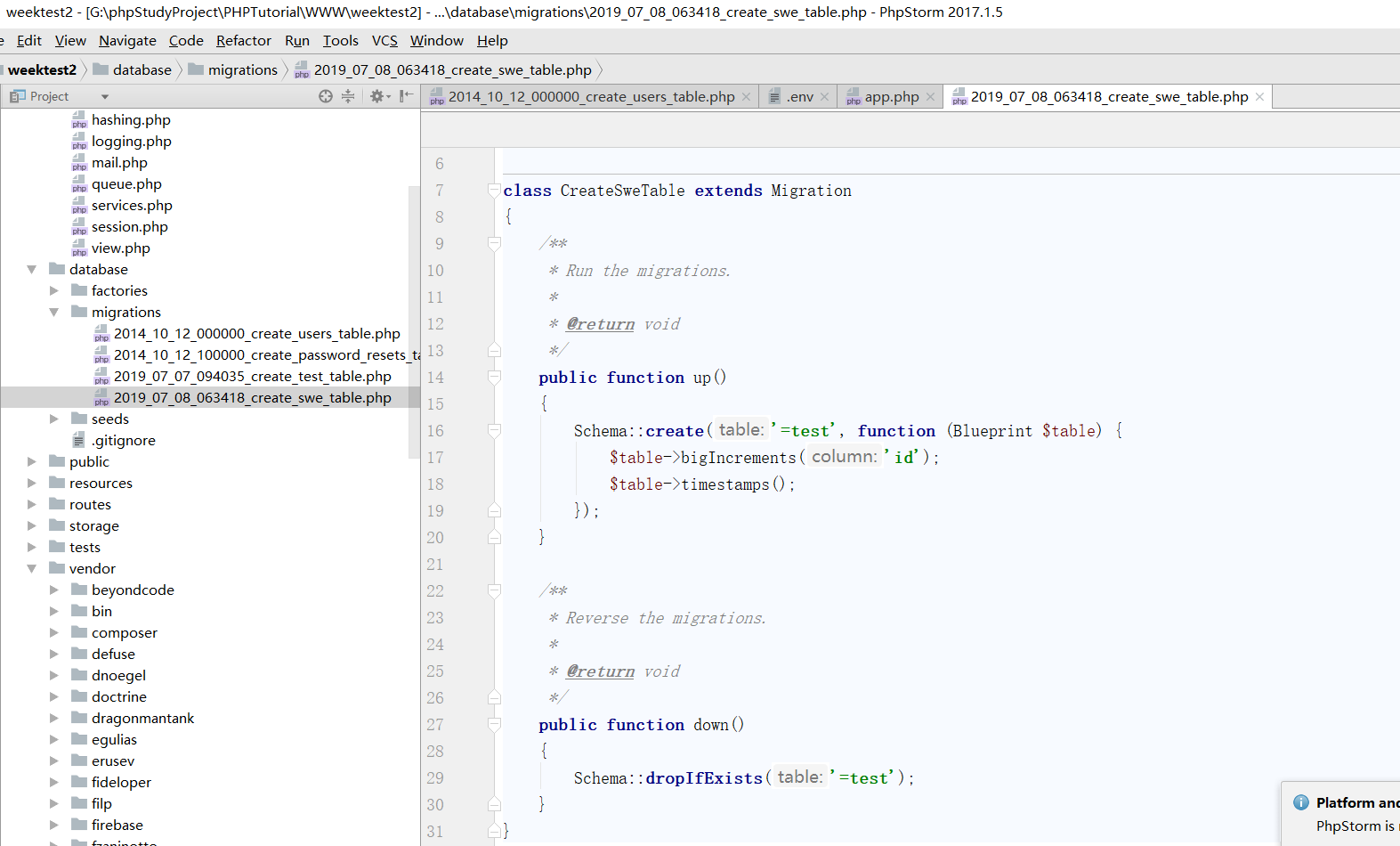
简单的说数据库迁移就是在团队合作中，把数据库中的表变成一个个类放在本地代码中，当其他成员要使用的时候就将代码导入数据库生成表，这样就保证了数据库版本的一致性。

#### 3.1指令

##### 3.12php artisan make:migration create\_swe\_table --create=“test”

生成一张test表,执行这个命令之后本地多了一个类





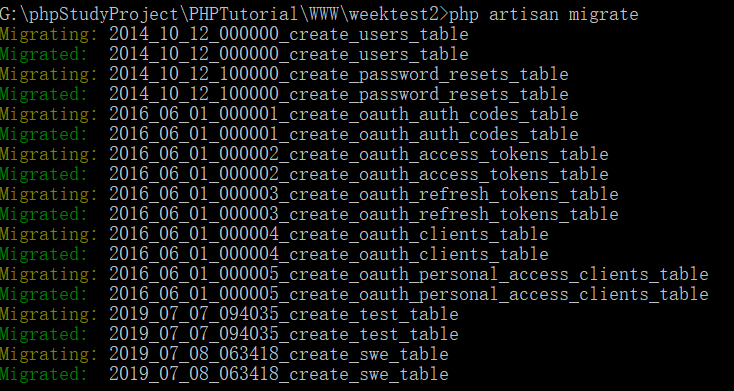
Up()方法用于创建，更改数据表，down()方法用于回滚。

##### 3.11php artisan migrate:install

在数据库中生成一张migration的表,用来跟踪哪些迁移已经运行。

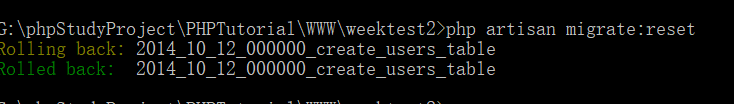
##### 3.13php artisan migrate

将本地的数据表类迁移到数据库



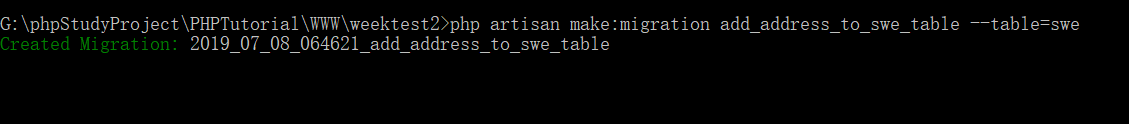
##### 3.14php artisan migrate:reset

回滚操作，将数据库中通过php artisan 指令插入的数据表删除



##### 3.15 php artisan make:migration add\_address\_to\_swe\_table --table=swe

为表新增字段





还可以在 up()方法中进行其他操作



##### 设置默认数据

php artisan make:seeder UserTableSeeder

在database/seeds/下生成UserTableSeeder.php

在run方法下插入默认数据

**public function** run()  
{  
 *//* DB::**table**(**'users'**)->insert([**'name'**=>env(**'SuperUser'**),  
 **'password'**=>env(**'SuperPassword'**),**'phone'**=>env(**'SuperPhone'**),**'is\_check'**=>2]);  
}

在DatabaseSeeder.php下添加

**public function** run()  
{  
 $this->call(UserTableSeeder::***class***);  
 $this->call(RoleTableSeeder::***class***);  
 $this->call(UserRoleTableSeeder::***class***);  
}

执行：

php artisan db:seed

单个执行：

php artisan db:seed --class=ProcessTableSeeder

重新执行

php artisan migrate:refresh --seed

##### 3.16其他

// 回滚上一次的迁移

php artisan migrate:rollback

// 回滚所有迁移并且再执行一次

php artisan migrate:refresh

//修改字段名称

php artisan make:migration alter\_tablename\_table --table=tablename

#### 3.2自定义artisan命令

Php artisan make:command command\_name

## 4.laravel 辅助函数

### 4.1 config()

config 函数获取配置变量的值，配置值可以通过使用”.”号访问，包含文件名以及你想要访问的选项。如果配置选项不存在的话默认值将会被指定并返回：

$value = config('app.timezone');

$value = config('app.timezone', $default);

辅助函数 config 还可以用于在运行时通过传递键值对数组设置配置变量值：

config(['app.debug' => true]);

### 4.2 route()函数

**为命名路由生成 URL**

正如上面代码所展示的，为给定路由分配名称之后，就可以通过辅助函数 route 为该命名路由生成 URL 或者通过 redirect 函数进行重定向：

// 生成URL

$url = route('profile');

// 生成重定向

return redirect()->route('profile');

### 4.3 asset()

$url = asset('img/photo.jpg');

为前端资源生成一个 URL,不用写明具体的文件路径，asset会自动找到完整的路径。

asset()方法用于引入 CSS/JavaScript/images 等文件,文件必须存放在public文件目录下。

## 5.trait

简单来说，trait就是把重复的方法拆分出去，是一组可以被不同类调用的方法集合，用的时候，使用use引入从而带到代码复用的效果。Trait不是类不能被实例化。

Trait的优先级：当前类 > trait > 父类

**<?php  
  
namespace** App\Http\Controllers;  
  
**use** Illuminate\Http\Request;  
  
**trait** traittest{  
 **public function** name()  
 {  
 **echo"我的名字叫trait<br>"**;  
  
 }  
 **public function** study(){  
 **echo "trait不需要学习<br>"**;  
  
 }  
}

**<?php  
  
namespace** App\Http\Controllers;  
  
**use** Illuminate\Http\Request;  
  
**class** people **extends** Controller  
{  
 **protected $name**;  
 **public function** \_\_construct($name)  
 {  
 $this->**name**=$name;  
 }  
 **public function** name(){  
 **echo "我的名字叫"**+$this->**name**;  
 }  
  
}

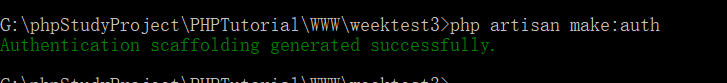
**<?php  
  
namespace** App\Http\Controllers;  
  
**use** Illuminate\Http\Request;  
  
**class** student **extends** people  
{  
 **use** traittest;  
 **public function** study(){  
 **echo** $this->**name**.**"是学生需要学习<br>"**;  
 }  
}

**<?php  
  
namespace** App\Http\Controllers;  
  
**use** Illuminate\Http\Request;  
  
**class** test **extends** Controller  
{  
  
 **public function** index(){  
 $student=**new** student(**"小明"**);  
 $student->name();  
 $student->study();  
 }  
}

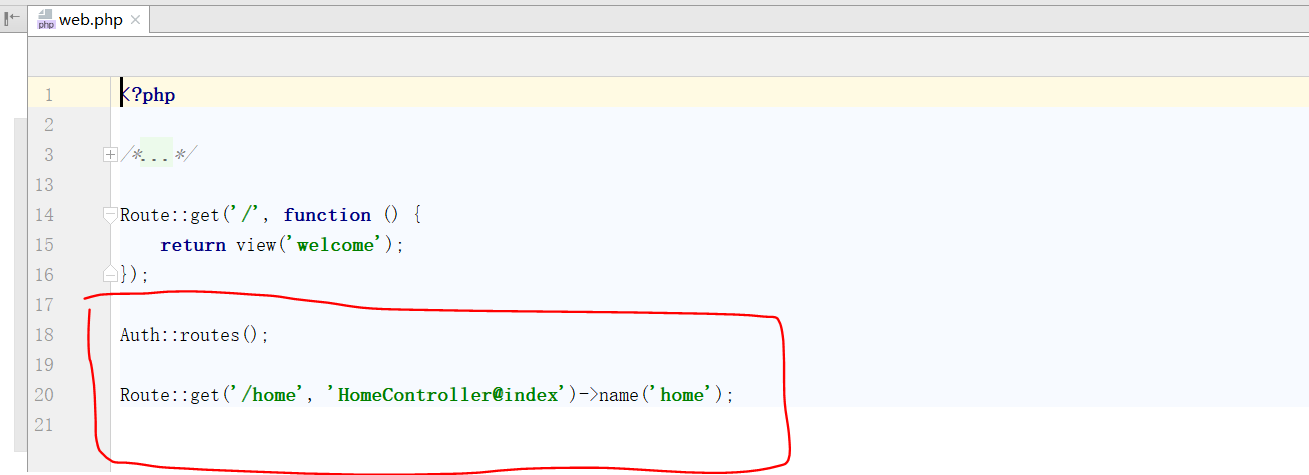
## 6.laravel的登录认证

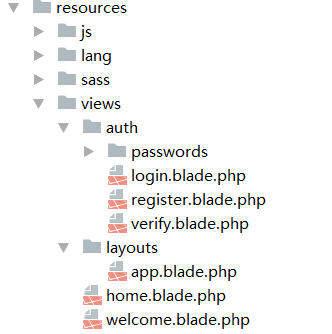
### 6.1 安装

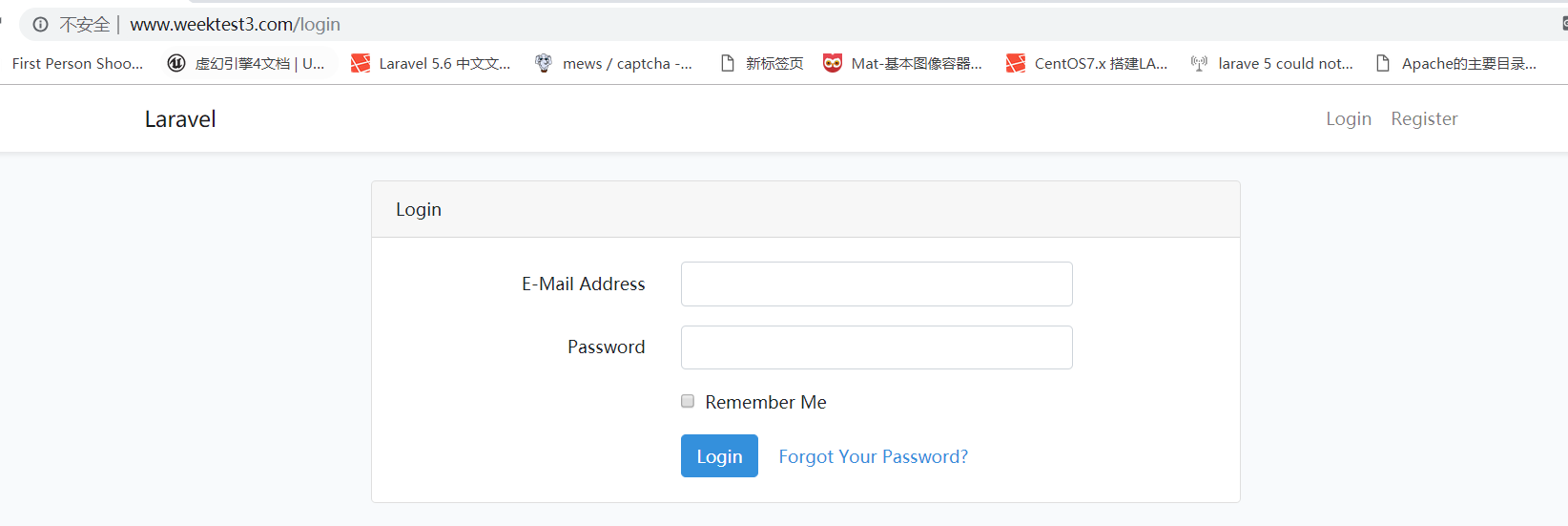
#### 6.11 php artisan make:auth



#### 6.12 结果



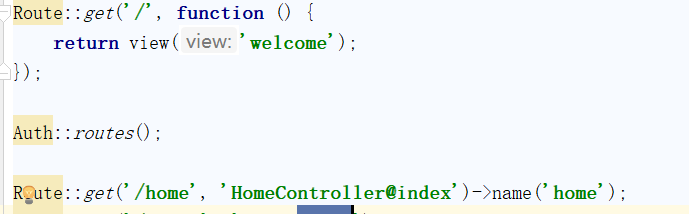




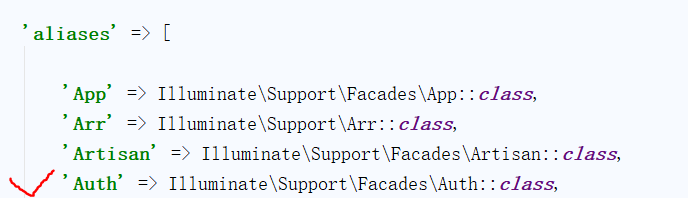
### 6.2页面跳转逻辑

参考：https://www.jianshu.com/p/7faa73e6ede3

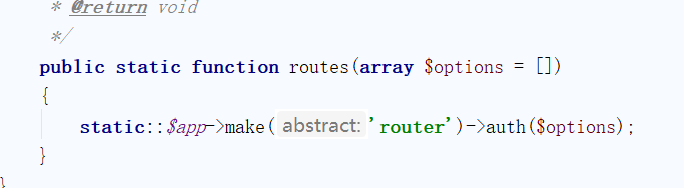
1.浏览器输入url

web.php中的Auth::routes()，Auth::routes()定义我们所需要的路由

1. 在config.php的app.php中找到Auth所在位置

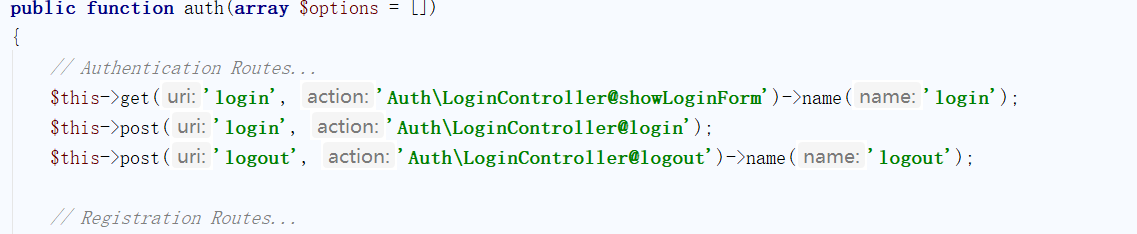


1. 找到这个Illuminate\Support\Facades\Auth类，看到其中有个routes()的静态方法

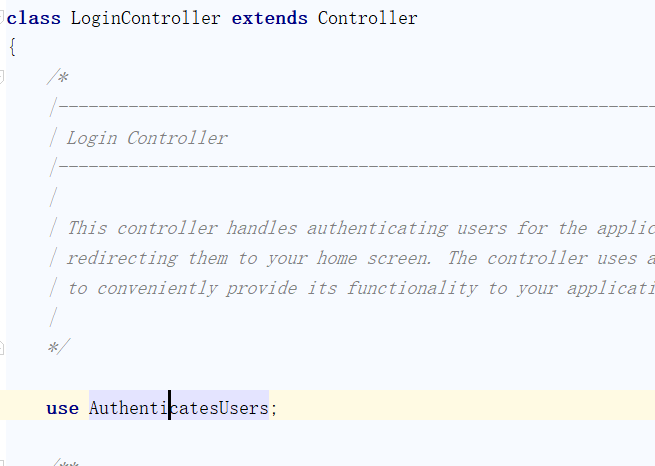


给应用注册标准的认证路由，方法中 static::$app->make('router')是获取router类的实例,而这个router类的实例则是在启动应用的时候通过注册基本服务者的时候已经注册过的。

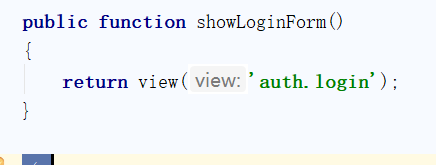
1. 在vender\laravel\framework\src\Illiminate\routing\rouer.php找到auth()方法

在这里可以看到laravel为我们定义好的路由。

5.回到loginController,发现没有**showLoginForm**方法，但是loginController引用了trait

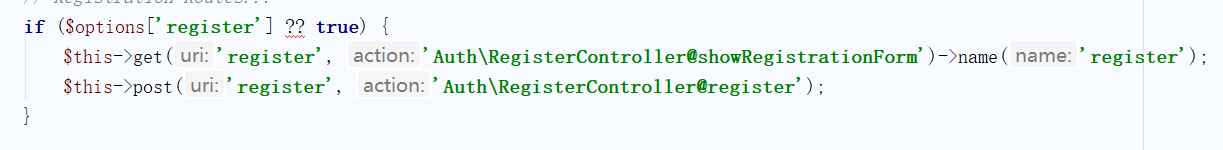


因此在AuthenticatesUsers中找到



返回登陆页面。

同理：



注册的时候，到registercontroller引入的trait中看register方法。



注册时先验证字段，再创建用户，$this->guard()进行基于session的认证guard,最后给用户登录操作