# Laravel框架\_04

# 今日目标7

* 能够了解laravel文件上传功能

Request对象来上传

# 文件是否上传

hasFile()

# 得到文件对象

file()

# 获取扩展名

getClientOriginalExtensions()

# 上传

move(path,filename)

iframe不刷新上传

* 能够了解在模型文件中创建一对一关联模型的方法

hasOne(关联模型，外键，本表ID)

hasMany

belongsToMany(关联模型，中间表，本模型对应中间表ID，关联模型对应中间件ID)

belongsTo

* 能够使用Cache类存储缓存数据

添加

add(key,value,expire);

# 设置

put

# 设置永久

forever(key,value)

# 判断是否存在

has

# 获取

get

remember(key,expire,function())

# 删除

forget(key)

flush();

* 能够使用composer方式安装验证码功能类

百度找到你想要的

到资源网站中去搜索它

composer require 安装

相关配置

模板中调用

验证器中去验证

* 能够使用captcha验证规则对错误的验证码进行验证

验证器中验证 captcha

* 能够在中间件中限制用户翻墙操作
* 能够完成rbac三张数据表的创建

用户表 用户表中有一个角色ID字段

角色表 用一个权限字段，以逗号隔开来保存权限

权限表

# 一、验证码

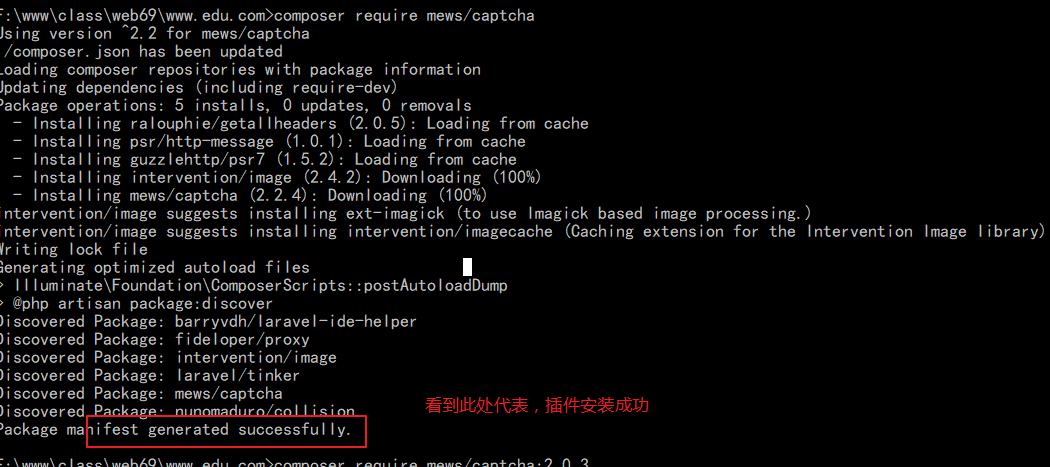
## 1.1、验证码的安装

我们通过 Composer 安装 Captcha 扩展包：

composer require mews/captcha

# 可以指定想要安装的版本

composer require mews/captcha:版本号



## 1.2、使用配置

使用Captcha服务提供者之前还需要在config/app.php中注册服务提供者：

# 在服务提供者中添加此项

'providers' => [

Mews\Captcha\CaptchaServiceProvider::class,

]

# 同时注册下相应门面：

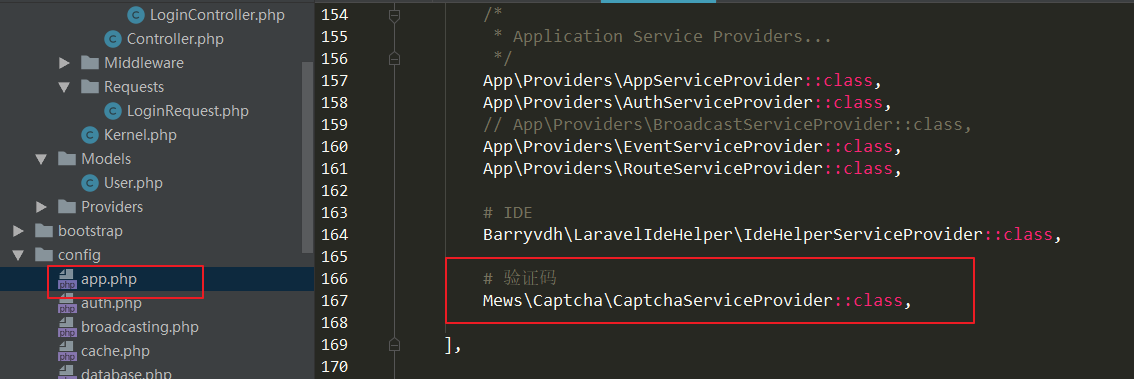
'aliases' => [

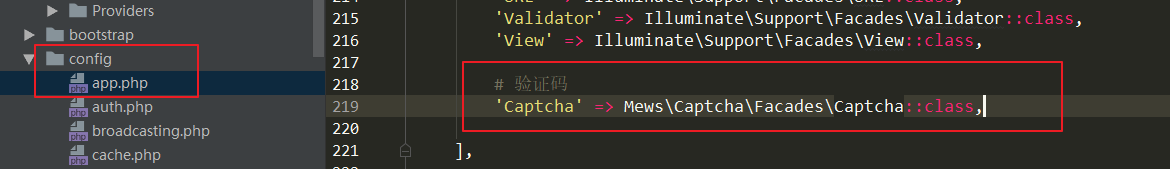
'Captcha' => Mews\Captcha\Facades\Captcha::class,

]

# 发布配置

php artisan vendor:publish





## 1.3、模板中调用

常用方法

# 返回url

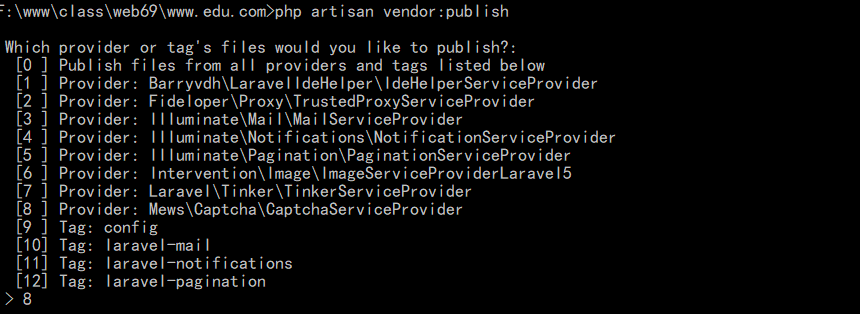
{!! captcha\_src() !!}

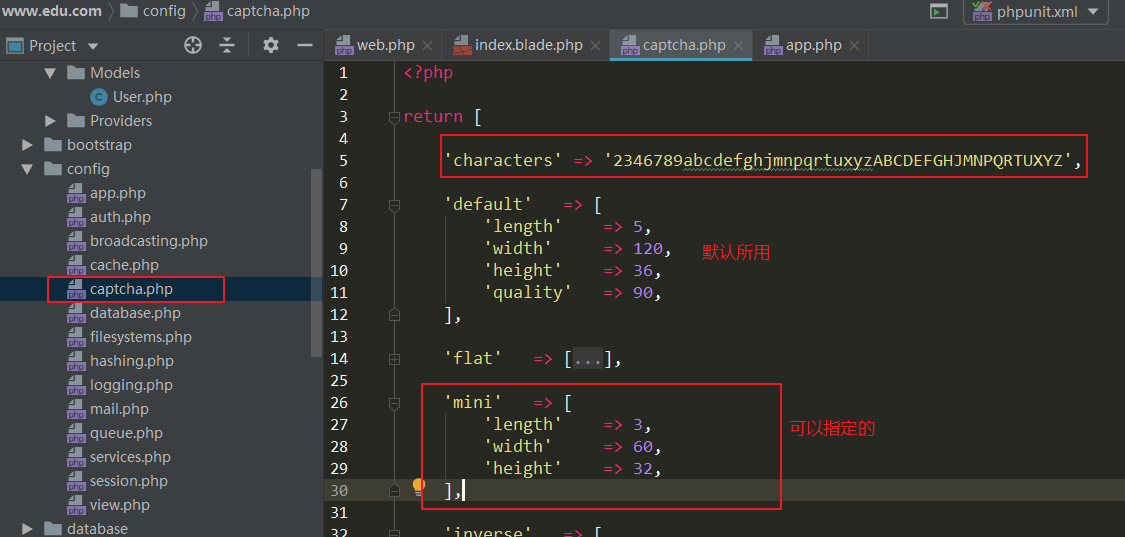
# 返回 img 的html

{!! captcha\_img () !!}

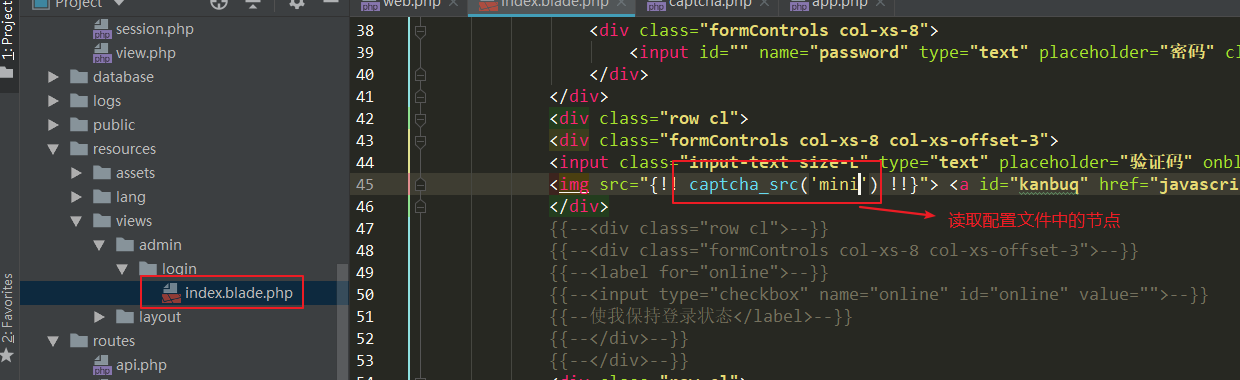
# 生成配置

php artisan vendor:publish





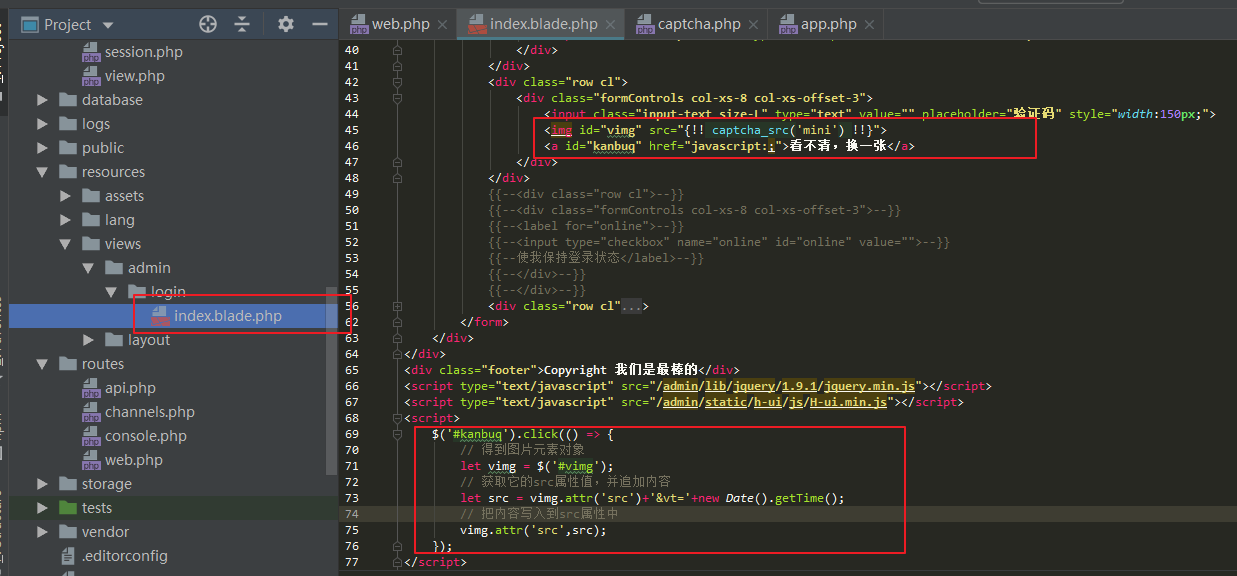
模板中



效果



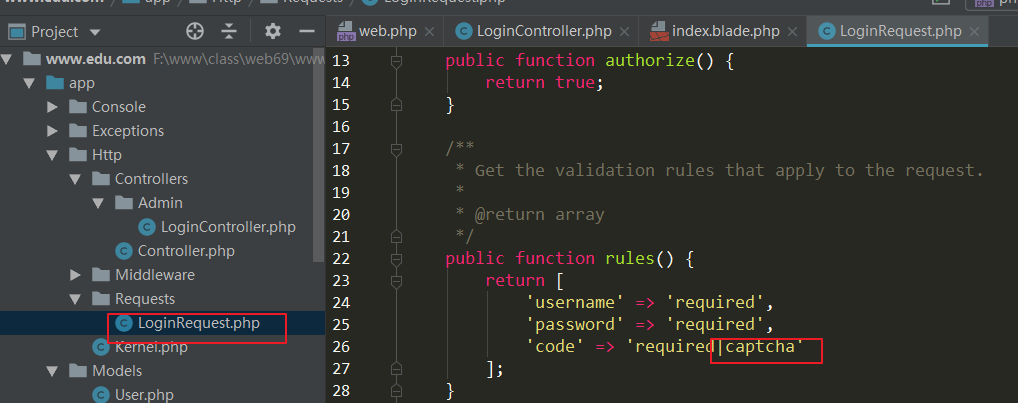
点击更换验证码图片



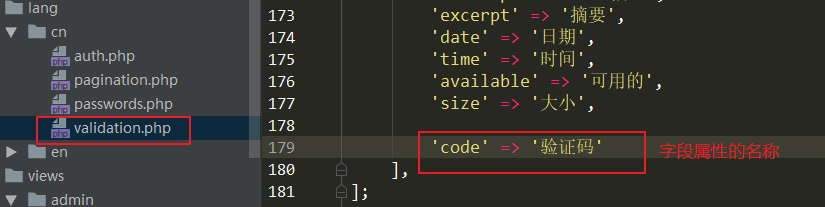
## 1.4、验证码的输入验证

解决验证提示中文

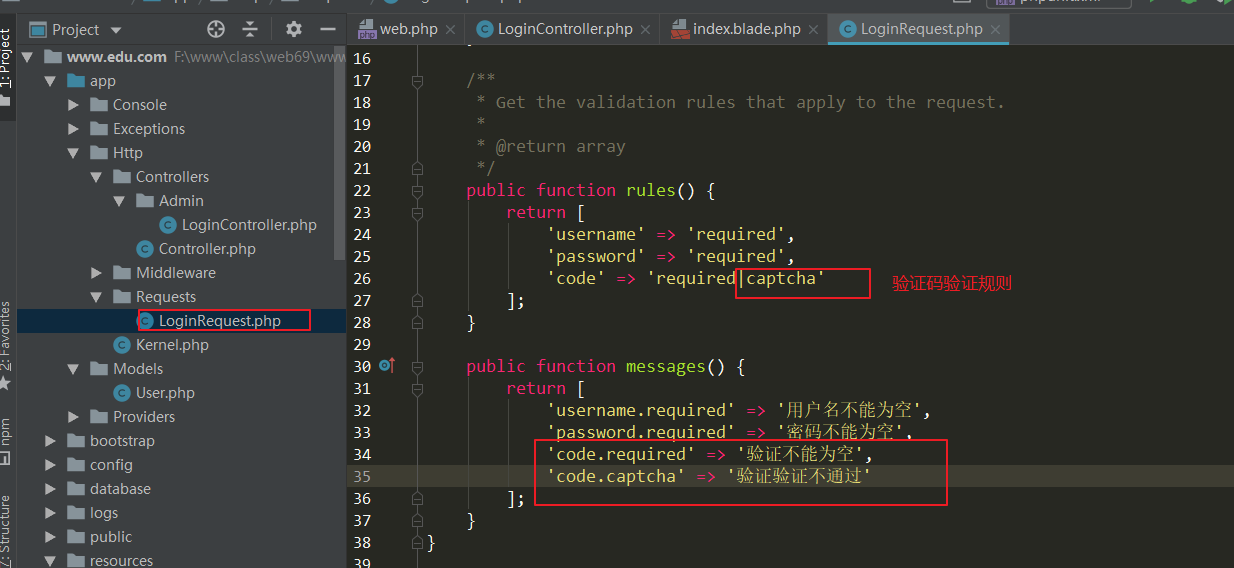
方案1 了解







方案2 推荐



效果



<http://www.h-ui.net/Hui-4.8-alert.shtml> h-ui框架手册

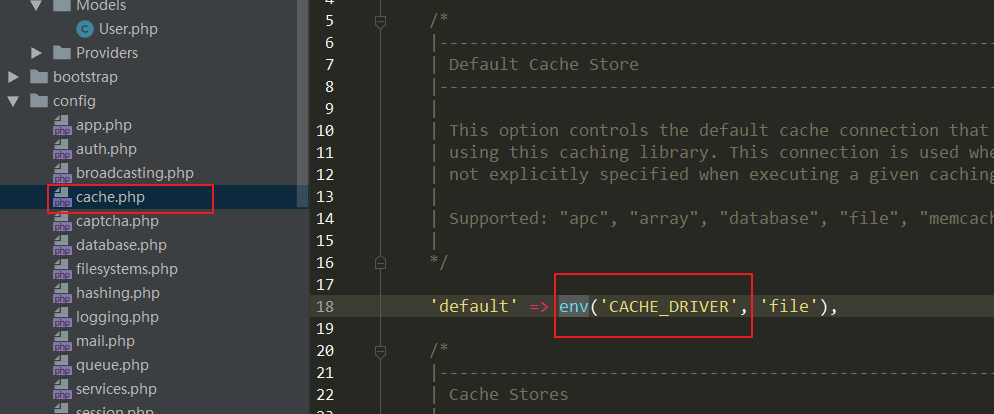
# 二、缓存操作

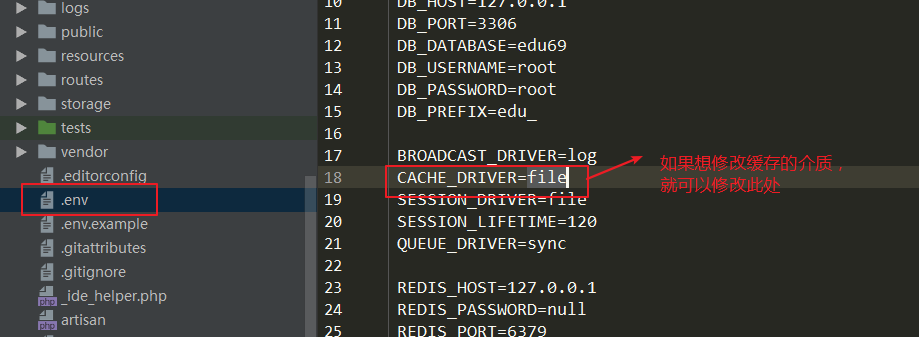
Laravel 为不同的缓存系统提供了统一的 API。缓存配置位于 config/cache.php。

Laravel 目前支持主流的缓存后端如 File、Memcached 和 Redis 等，默认是使用文件缓存。

* .env文件配置 ，推荐修改这里
* config/cache.php 文件，不建议直接修改

默认laravel支持缓存介质："apc", "array", "database", "file", "memcached", "redis"





## 2.1、设置缓存

use Cache; # 门面

# 添加 如果key不存在则添加成功，如果key存在，返回false

Cache::add('key', 'value', $minutes);

# 设置 不管key值是否存，都可以设置成功

Cache::put('key', 'value', $minutes);

# 设置永久缓存

Cache::forever('key', 'value');

## 2.2、获取缓存数据

$value = Cache::get('key');

$value = Cache::get('key', 'default');

$value = Cache::get('key', function(){});

# 获取的同时并存储

$value = Cache::remember('users', $*minutes*, function () {

return 'key不存的时候返回的数据';

});

## 2.3、检查缓存项是否存在

# 检查缓存的key值是否存在，存在返回true否则返回false

Cache::has('key')

## 2.4、删除缓存数据

# 先获取再删除

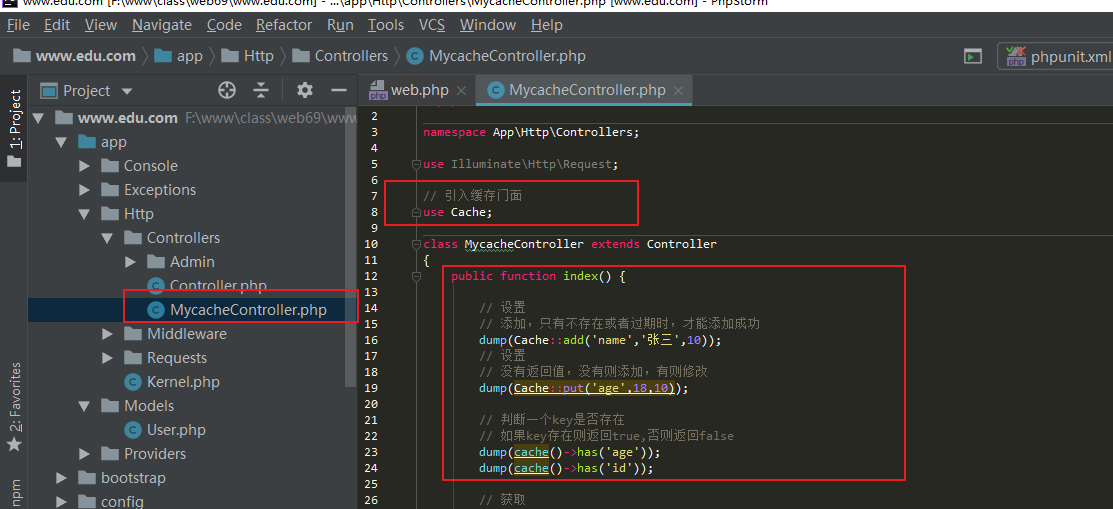
$value = Cache::pull('key');

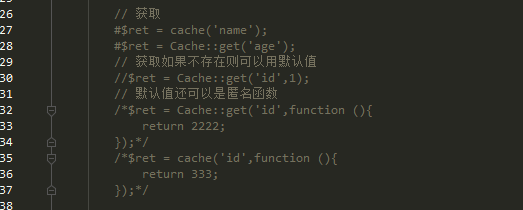
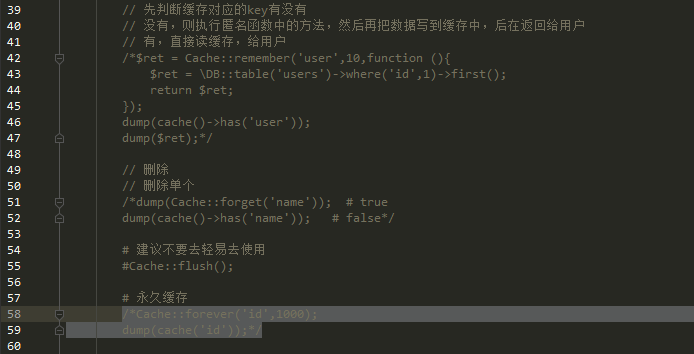
# 删除单个

Cache::forget('key');

# 清除所有的缓存

Cache::flush();



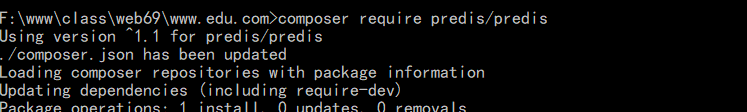
文件所生成的地方在



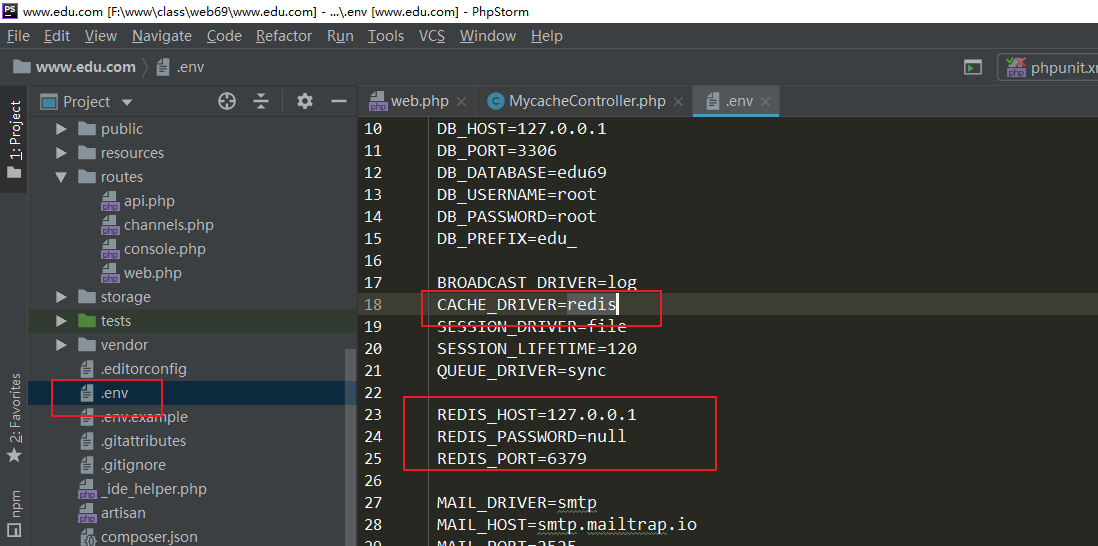
## 2.5、缓存设置到redis中

第1步：安装laravel支持的redis扩展

composer require predis/predis



第2步：在.env文件中修改存储介质



第3步：开启redis服务

# 三、文件上传

在laravel里面实现文件的上传是很简单的，不用引入第三方的类库，直接通过Request对象就可以获取到上传文件资源后进行保存。

# 获取上传的文件 $\_FILES[文件域表单名称]

$file = $request->file('文件表单名称');

# 验证文件是否存在

$request->hasFile('文件表单名称');

# 获取上传文件的后缀

$request->file('文件表单名称')->getClientOriginalExtension();

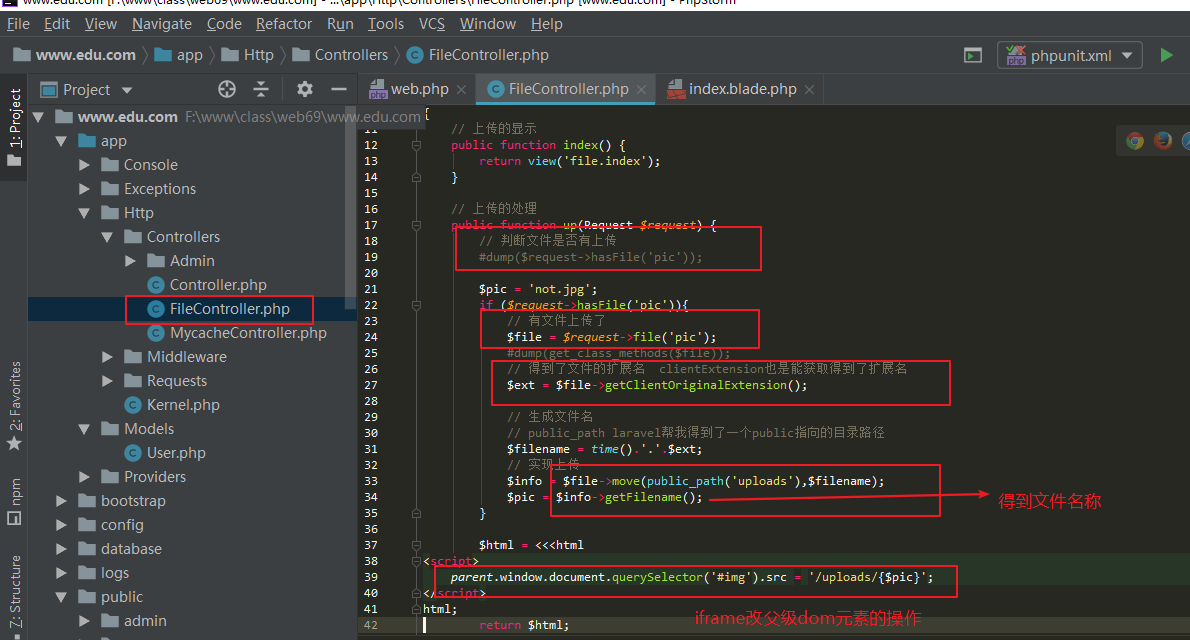
# 将文件移动到服务器指定的位置 上传方式1

$request->file('文件表单名称')->move(路径,文件名)

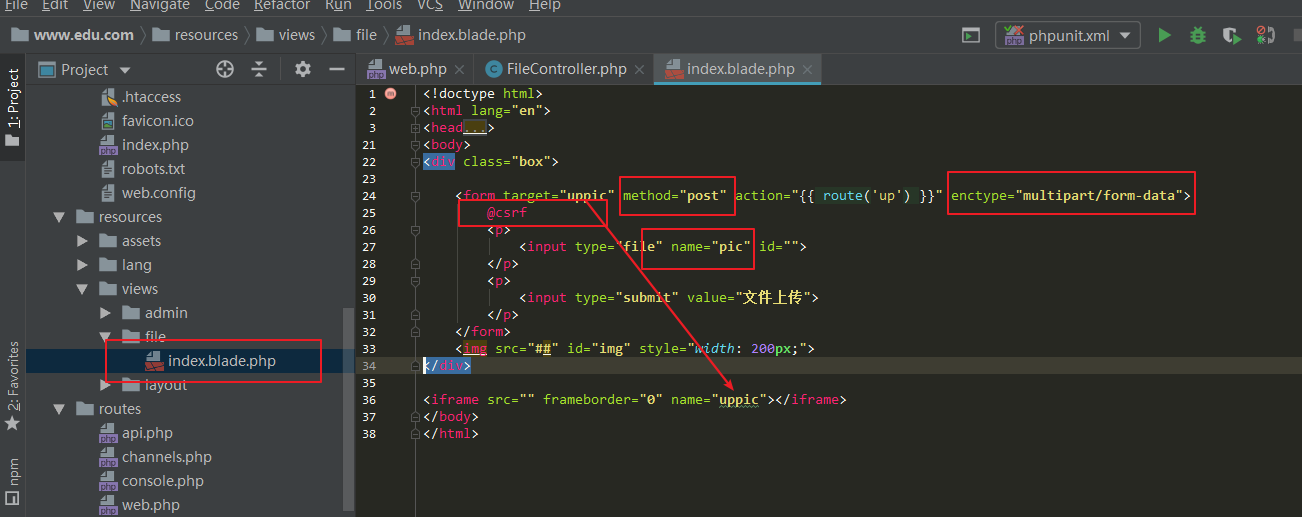
路由



控制器



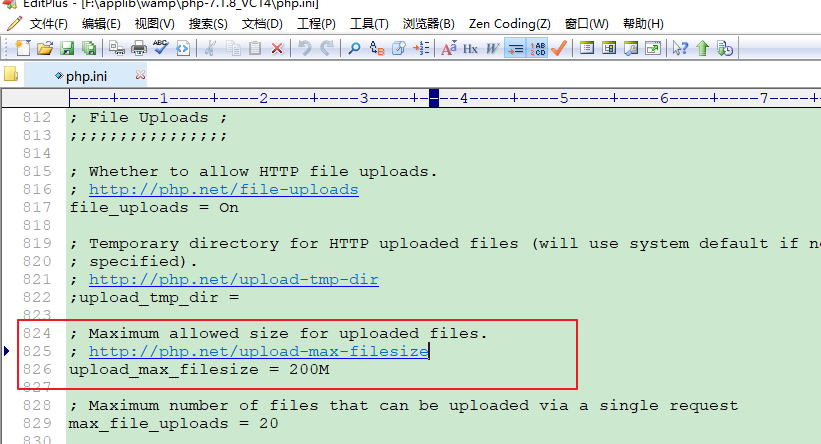
html元素

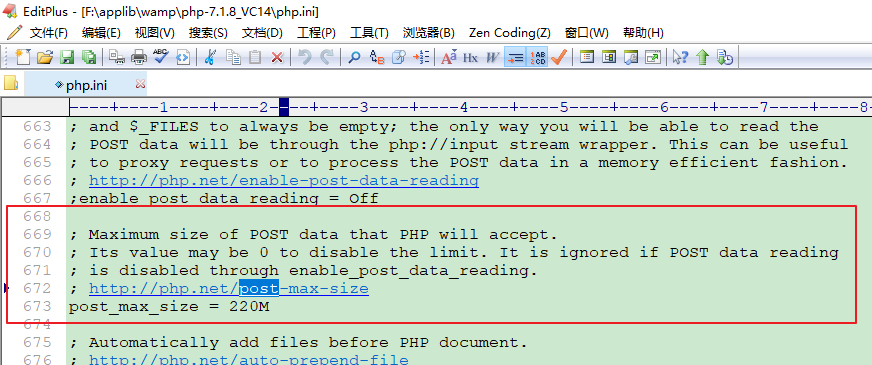


效果



修改配置文件，让上传支持大文件,修改php.ini文件





# 四、使用Auth类来进行登录验证

Laravel 中实现用户认证非常简单。实际上，几乎所有东西都已经为你配置好了。配置文件位于 config/auth.php，其中包含了用于调整认证服务行为的、标注好注释的选项配置。

实现Auth认证登录步骤

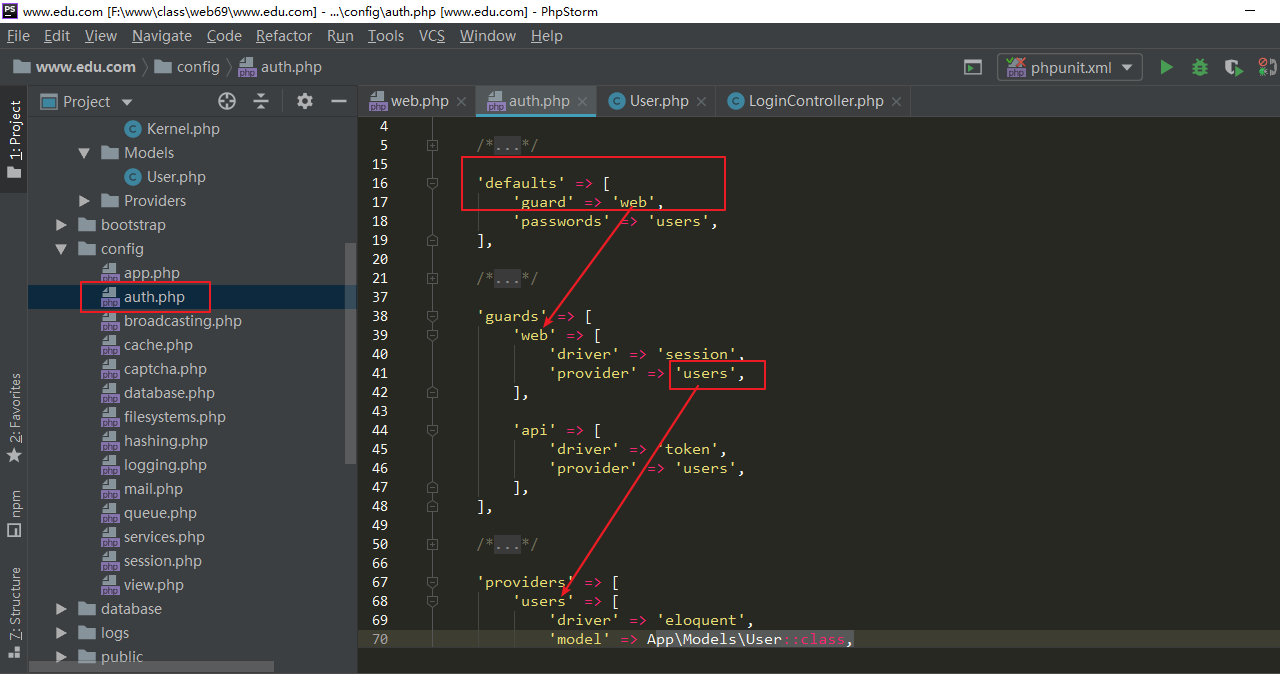
* 用户模型继承父类进行了更换

use Illuminate\Foundation\Auth\User

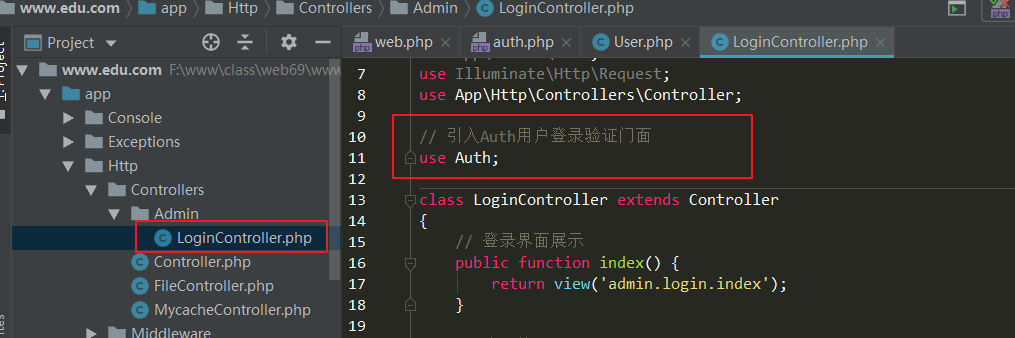
* 在控制器中引入Auth门面

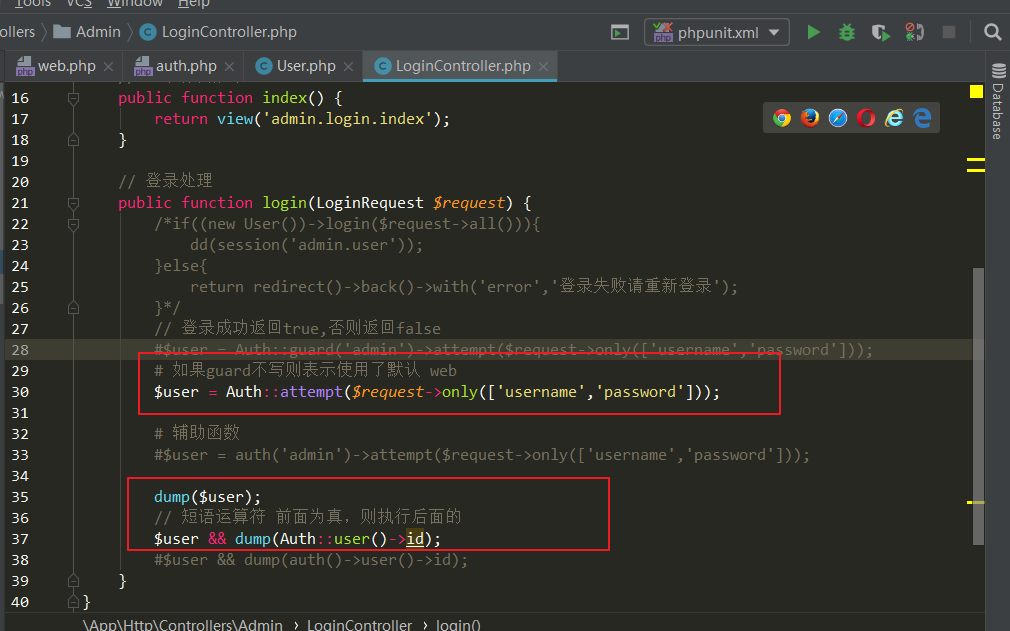
// 引入Auth用户登录验证门面  
use Auth;

* 修改config/auth.php文件



* 在控制器中实现登录功能





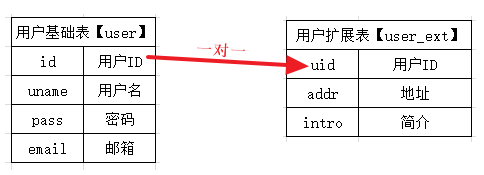
注：密码字段名必须要用password

# 五、多表关系关联模型

数据表之间经常会互相进行关联。例如，一篇博客文章可能会有多条评论，或是一张订单可能对应一个下单客户。Laravel中的ORM让管理和处理这些关联变得很容易。

## 5.1、一对一关系

在实际的项目中我们保存用户的信息往往进行分表来进行保存，把常用的基础数据存放在一张表中，而一些不用常的冷数据存放在另一张表中，而这2张表的关系就为一对一的关系。



模型中使用

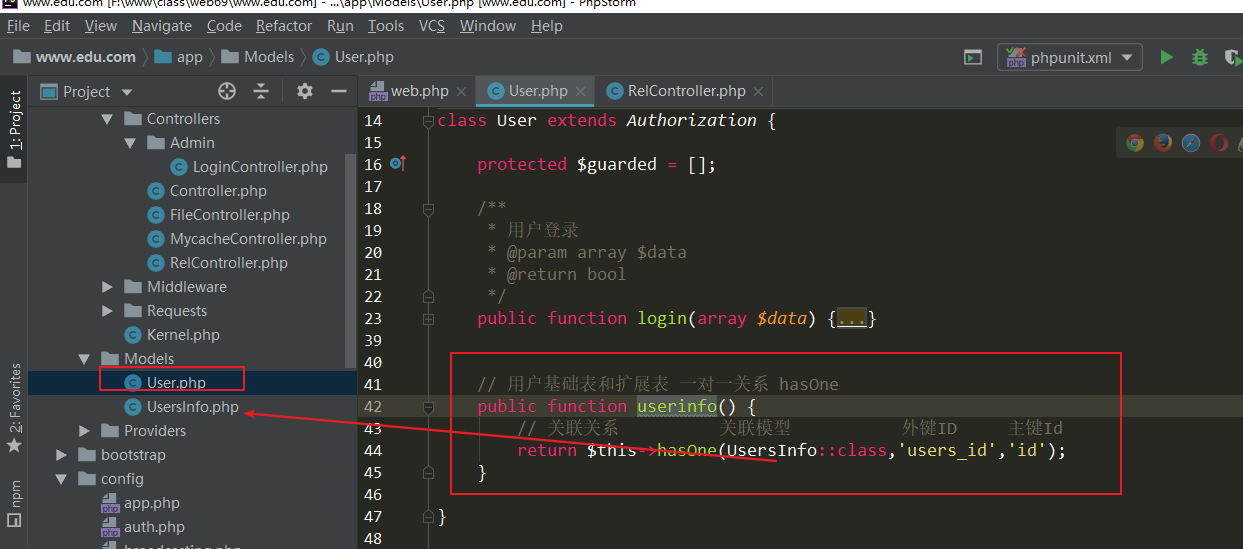
$this->hasOne(关联model，[关联model的联系键]，[本model的联系键]);

return $this->hasOne(Extuser::class, 'uid', 'id');

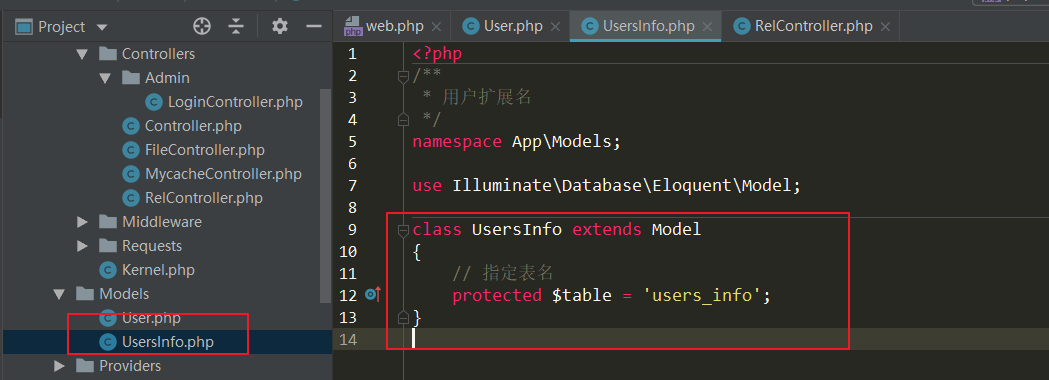
# 关联表中的外键名 user\_id 本表中的主键ID为id

return $this->hasOne(Extuser::class);

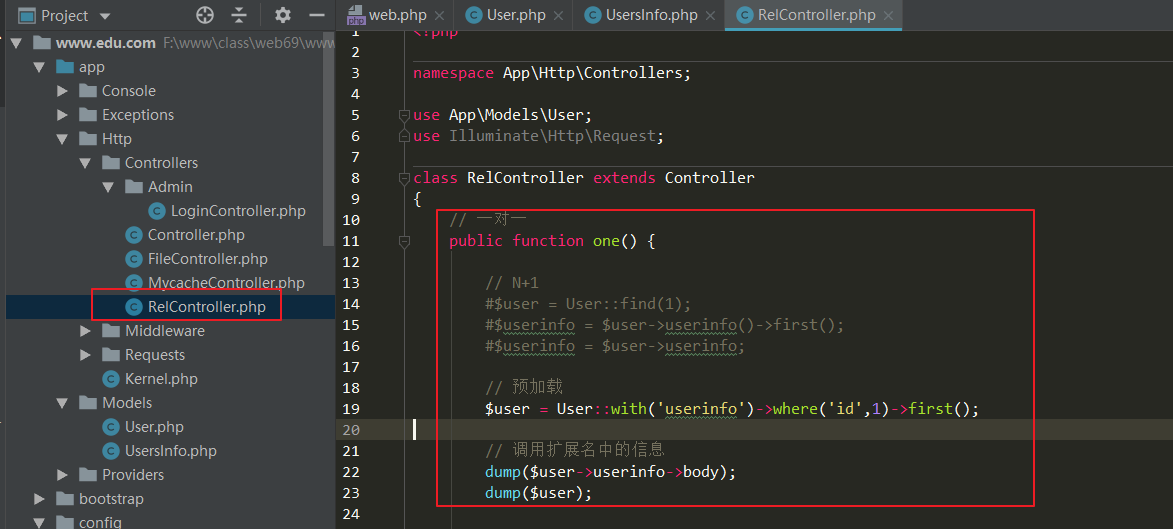
在主模型

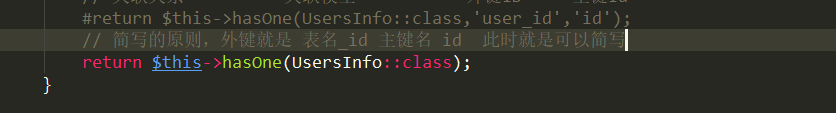


关联模型中



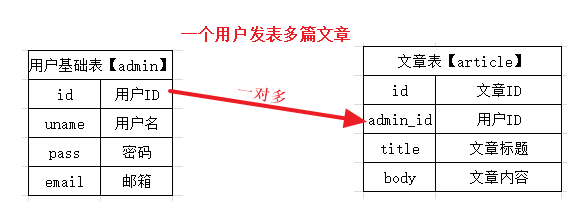
控制器





## 5.2、一对多关系

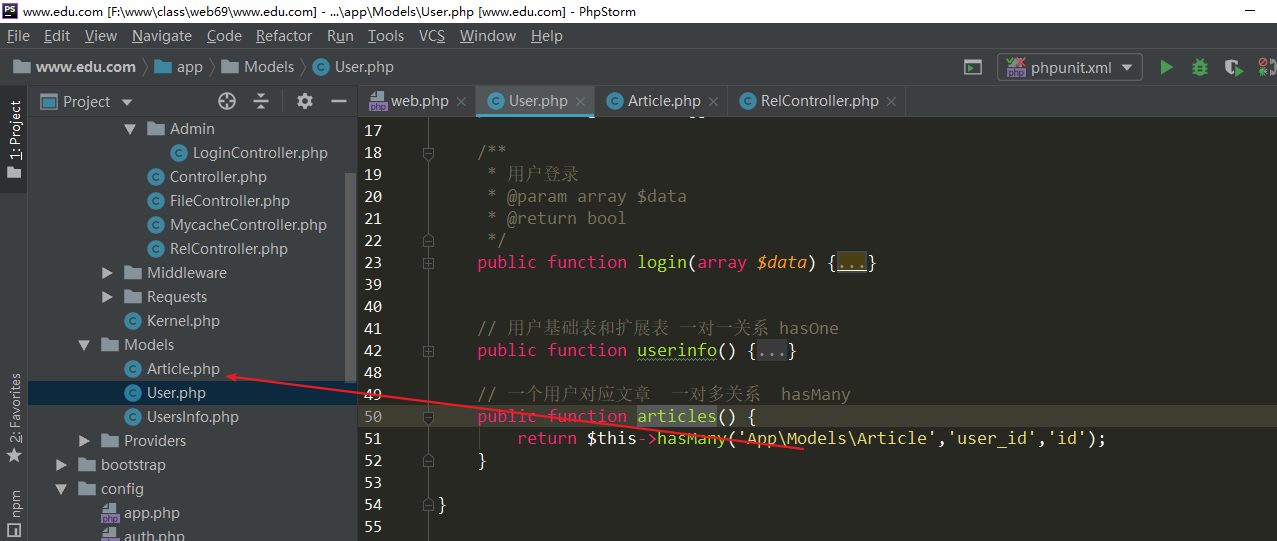
在实际的项目中一个用户发表多篇的文章，这样的关系就是一对多的关系。



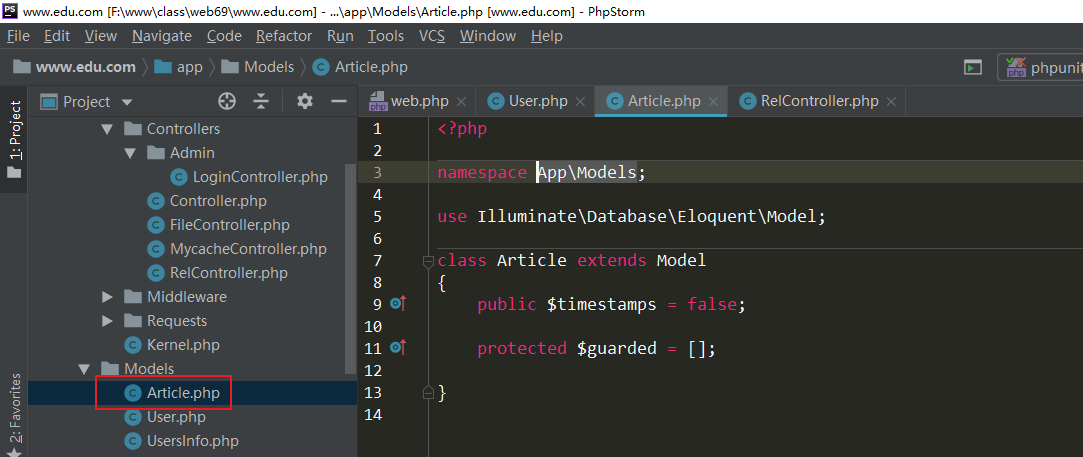
模型中使用

$this->hasMany(关联model，[关联model的联系键]，[本model的联系键]);

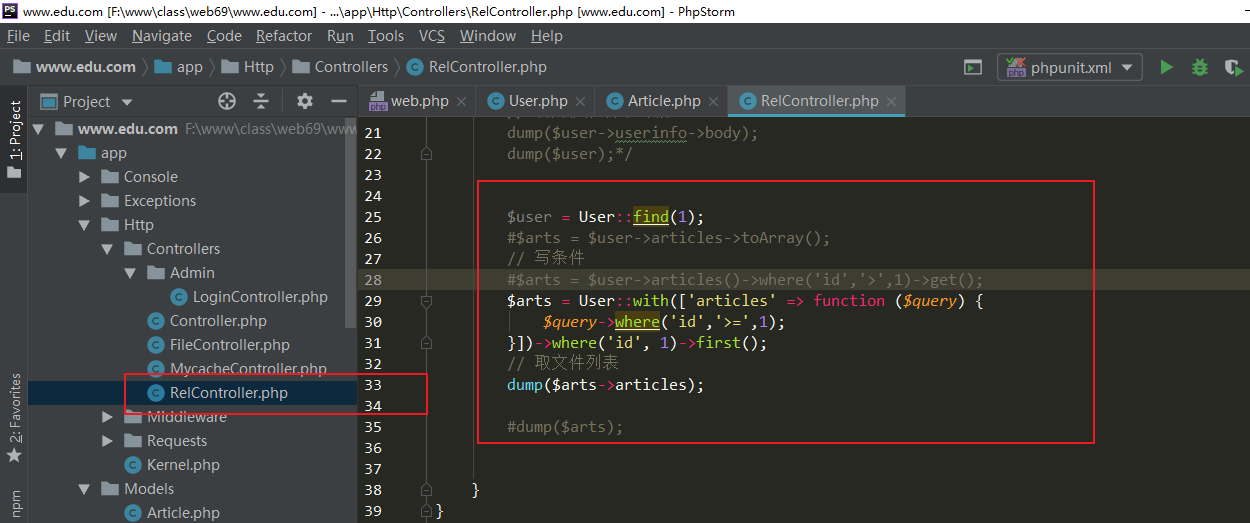
return $this->hasMany(App\Phone::class, 'foreign\_key', 'local\_key');



文章模型

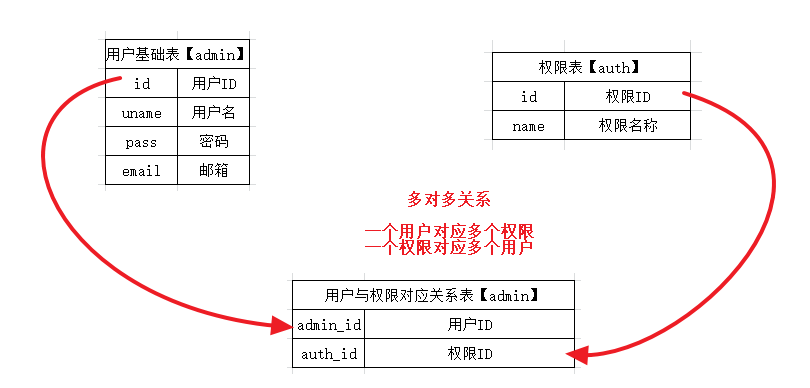


控制器中调用



## 5.3、多对多关系

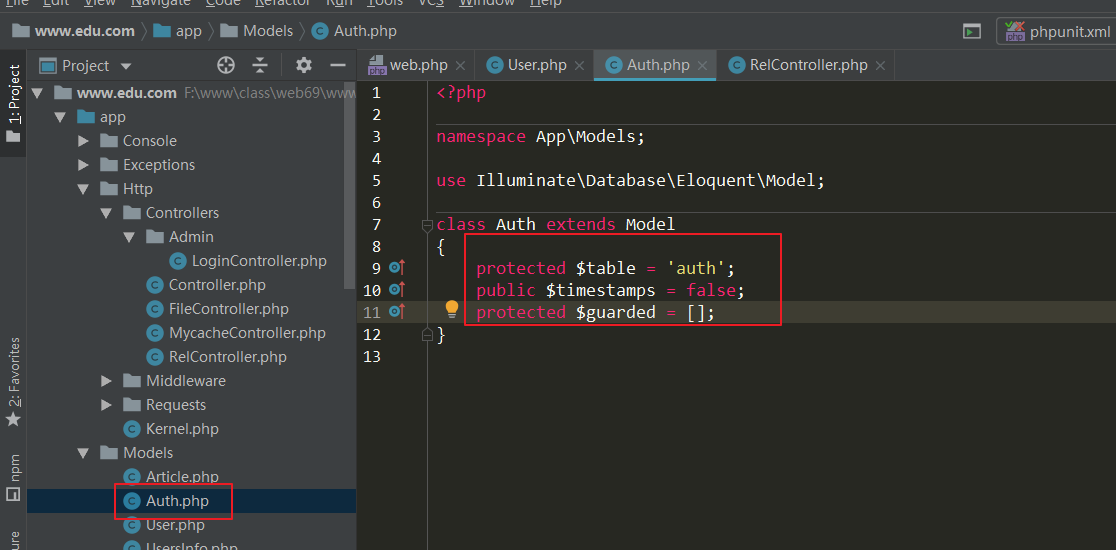
一个用户对应着多的权限，一个权限也对应多个用户，这样两者的关系就为多对多的关系，一般在做后台用户权限的时候会用涉及到。



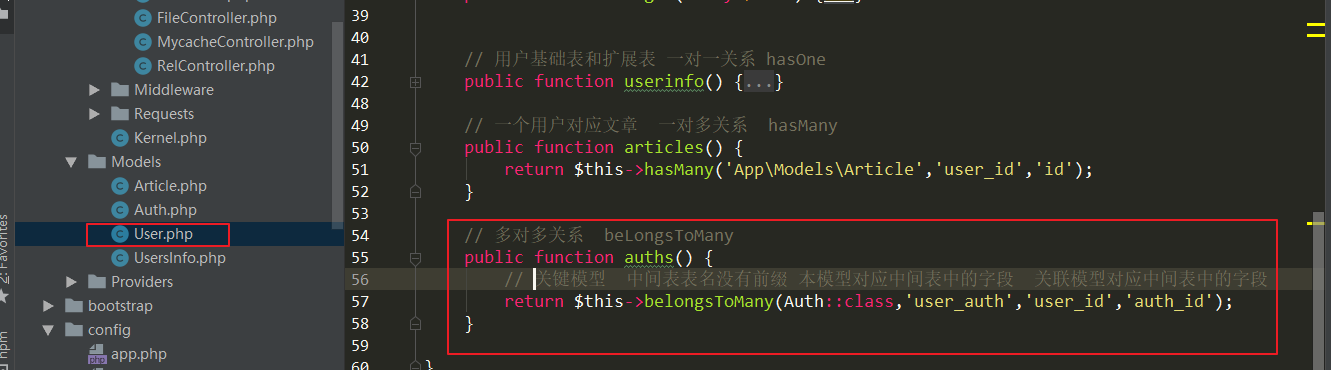
模型中使用

$this-> belongsToMany(关联表model，中间表表名，中间表中本model的关联ID，中间表中关联model的关联ID);

return $this->belongsToMany(App\User::class, 'user\_auth\_table', 'user\_id', 'auth\_id');



用户模型



控制器调用

