# 一、路由的介绍

## 什么是路由

简单的说就是将以后的请求转发给php去处理。

## 路由的作用

路由的作用就是建立url和php程序之间的映射。在laravel框架里路由是**必须的**，并且每个路由是**唯一的**，laravel是一个重路由的框架。

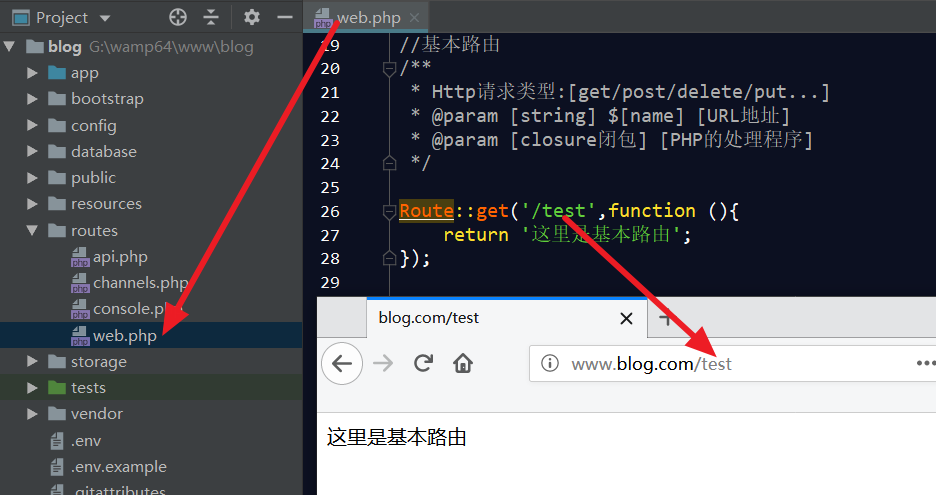
## Http请求的种类

Httpd请求共7种，但不是所有的请求web服务器都能支持，我们常用的是get、post。

【get、post、put、delete、patch、tarce、options】

# 二、路由的使用

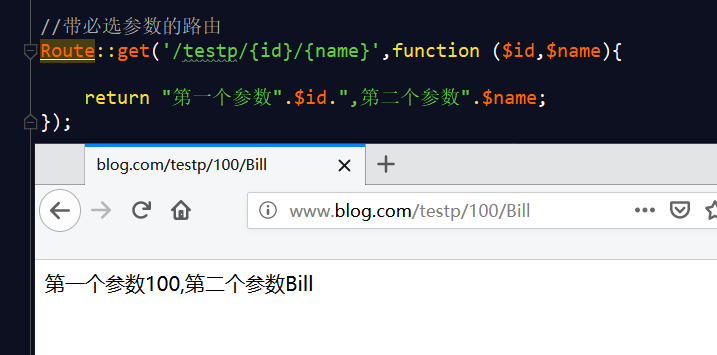
## 1.基本路由



## 2.必选参数路由

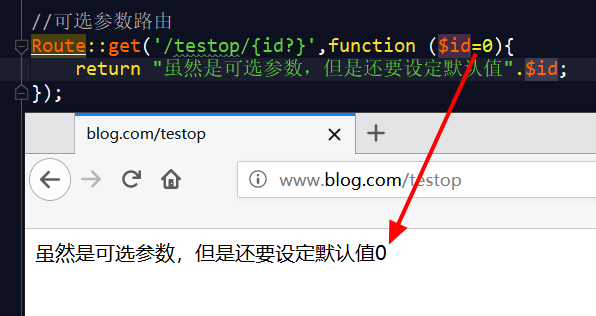
1.这种路由在使用时候如果没有参数就会报错。

2.路由里的{id}和闭包里的参数不必同名，顺序会一一对应。



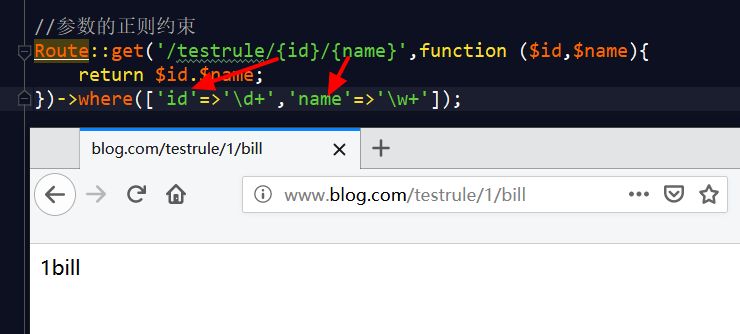
## 可选参数路由

这样的路由在地址栏传递的时候是可以没有参数的。



## 参数的正则约束

多个参数可以使用关联数组的形式写正则.



我们发现这个闭包函数可以完成很多事物,但是随着项目的越做越大，闭包就不够用啦。所以我们可以**路由到控制器**里的方法去处理。

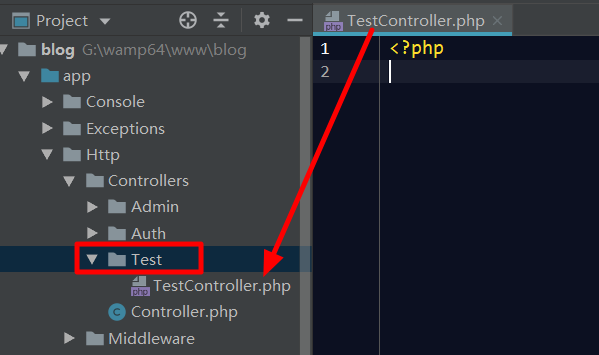
# 三、控制器

除了在路由文件中以闭包的形式定义所有的请求处理逻辑外，你可能还想使用控制器类来组织此类操作。控制器能够将相关的请求处理逻辑组成一个单独的类。控制器被存放在 app/Http/Controllers 目录下。

## 控制器--手动生成

**Step1.新建文件**

在app/Http/Controllers目录下新建一个文件TestController.php，为了方便管理，我又创建了Test文件夹来存放TestController.php文件。注意在laravel框架中类是不需要Class中间缀的。



**Step2.编写内容**

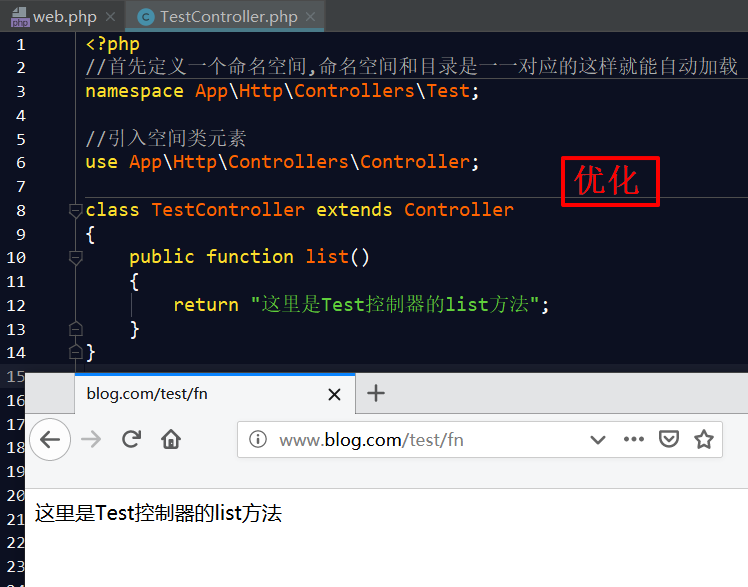


**定义一个路由到Test\TestController类的list方法的路由。**



**Step3.优化**

使用use引入空间类元素

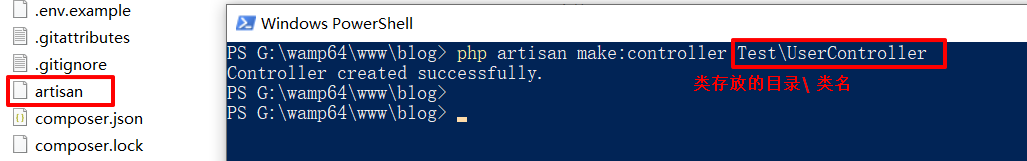


## 2.控制器--脚本生成

使用artisan脚本生成控制器

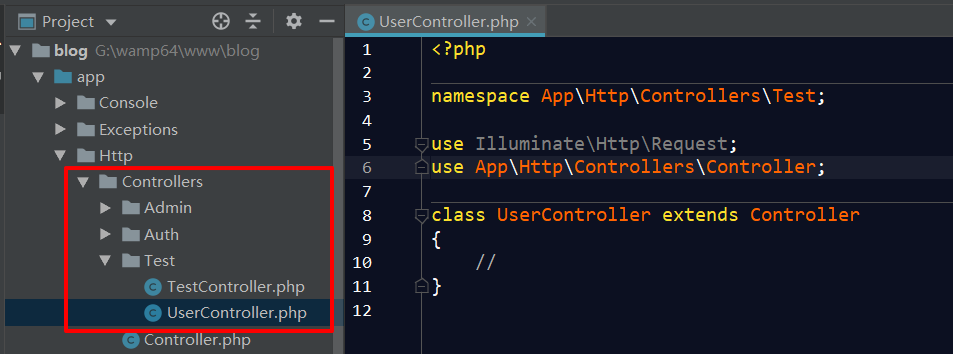
**step1：进入artisan的脚本目录，执行命令**

**php artisan make:controller Test\UserController**

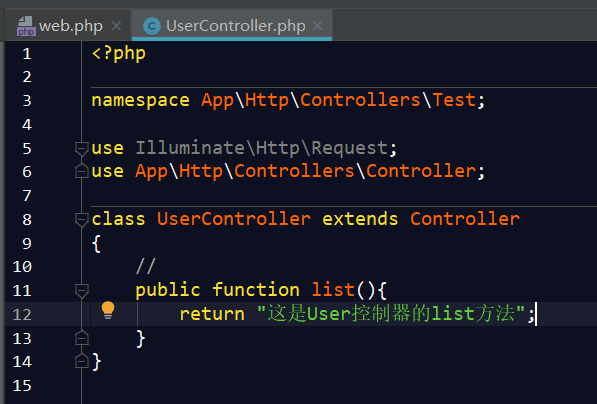


提示创建成功之后会在Cotroller目录下创建Test目录并在里面生成UserTroller.php。

可以发现和我们手动创建的控制器格式一样



Step2：编写业务代码、编写路由。



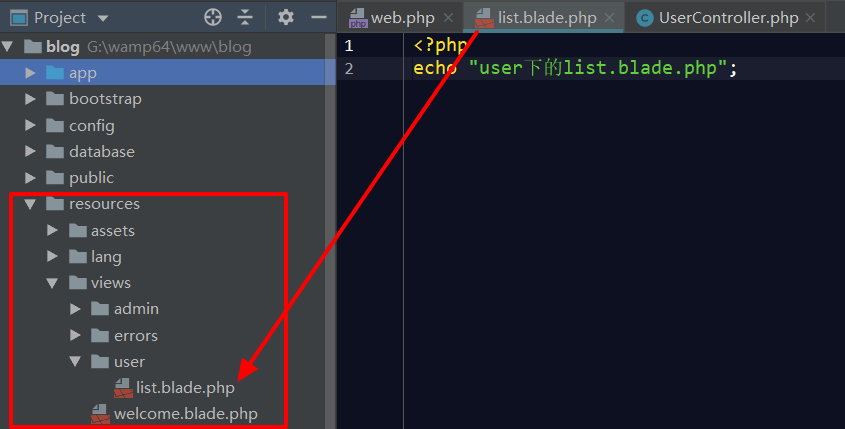


# 四、视图

## 1.创建视图

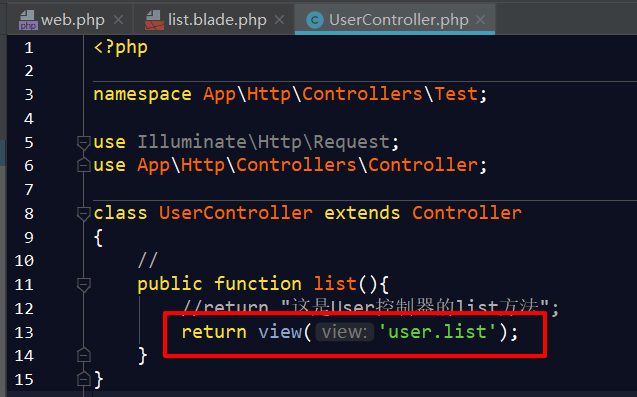
视图包含你应用程序的HTML内容，laravel使用blade模板引擎。视图文件存放于 resources/views目录下 。模板使用 .blade.php文件命名。

例如我们在resources/views下创建user目录并创建list.blade.php

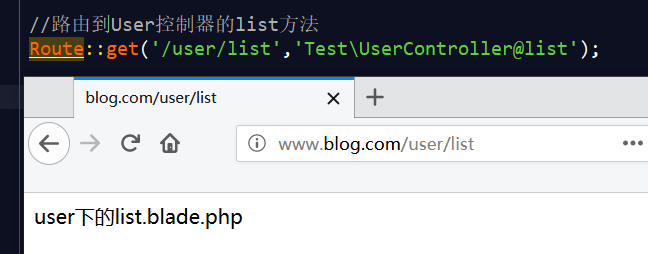


## 加载视图

我们在刚刚的UserCotroller类里面加载视图。通过view(目录.文件)或(目录\文件)

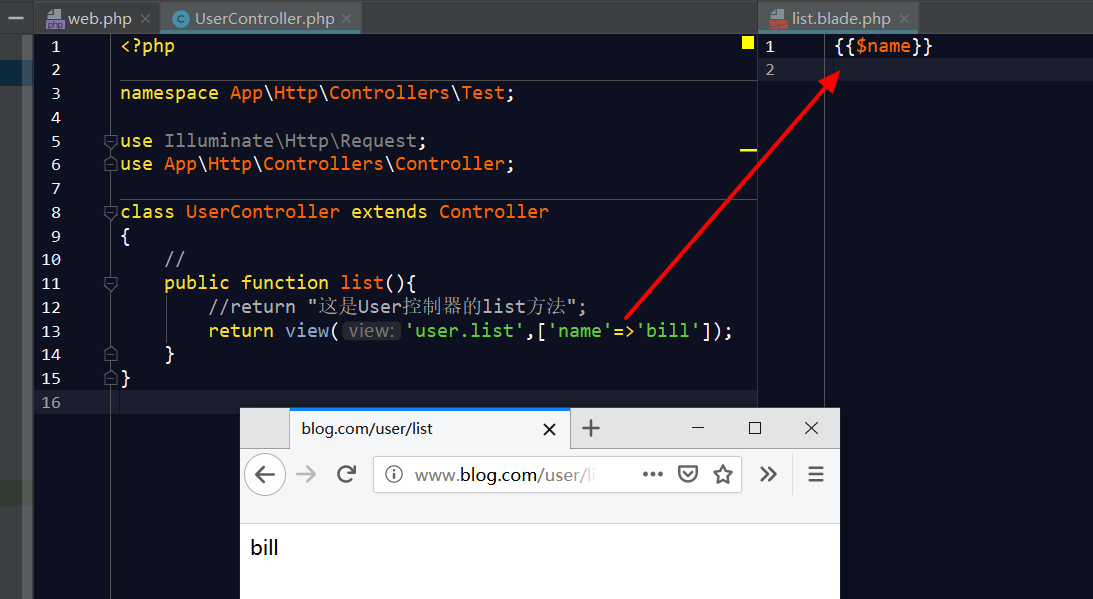


通过路由访问



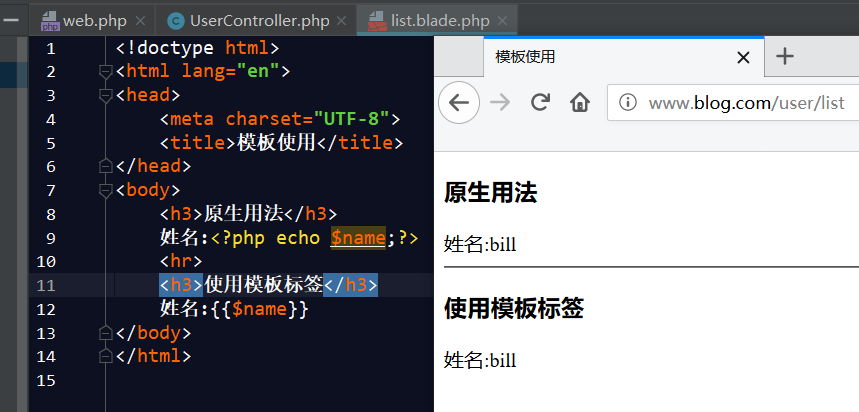
## 传递数据到视图

使用view的第二个参数，当用这种方式传递数据时，第二个参数($data）必须是键\值对数组。



## 模板标签语法

1. 使用原生语法
2. 使用模板标签(差值表达式) {{$键}}



1. 使用遍历

1.先定义一个数组

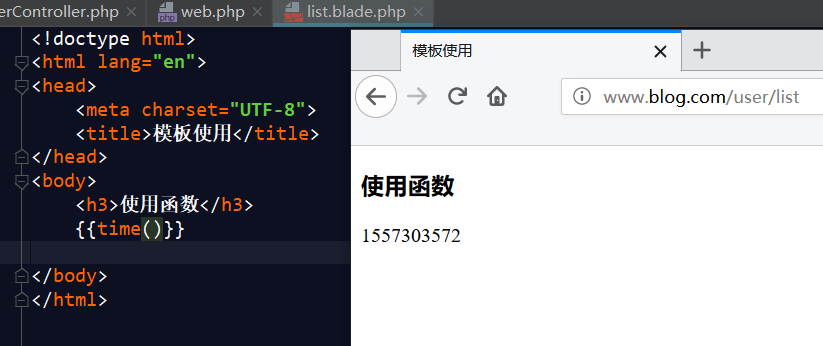


必须先给name一个属性，否则报错。

使用模板遍历



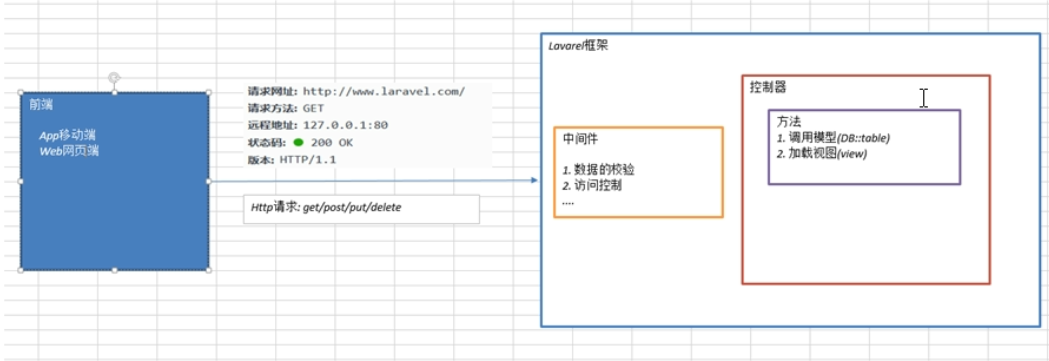
## 使用函数



# 五、中间件

## 1.什么是中间件

中间件位置：Http请求之后，控制器处理之前。



## 2.中间件作用

Laravel 中间件提供了一种方便的机制来**过滤**进入应用的 HTTP **请求**。例如，Laravel 内置了一个中间件来验证用户的身份认证。如果用户没有通过身份认证，中间件会将用户重定向到登录界面。但是，如果用户被认证，中间件将允许该请求进一步进入该应用。

当然，除了身份认证以外，还可以编写另外的中间件来执行各种任务。例如：CORS 中间件可以负责为所有离开应用的响应添加合适的头部信息；日志中间件可以记录所有传入应用的请求。

Laravel 自带了一些中间件，包括身份验证、CSRF 保护等。所有这些中间件都位于 **app/Http/Middleware** 目录。

作用：

数据过滤

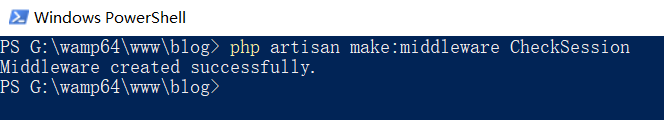
权限验证

## 3.中间件的使用

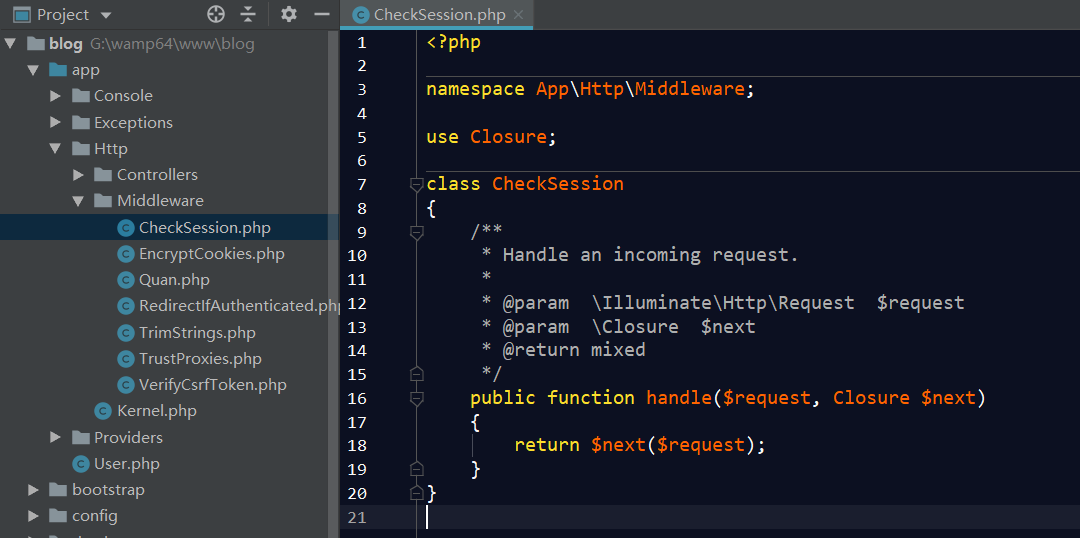
**Step1.生成中间件**

通过artisan脚本生成中间件

本次我们生成一个检测session的中间件.

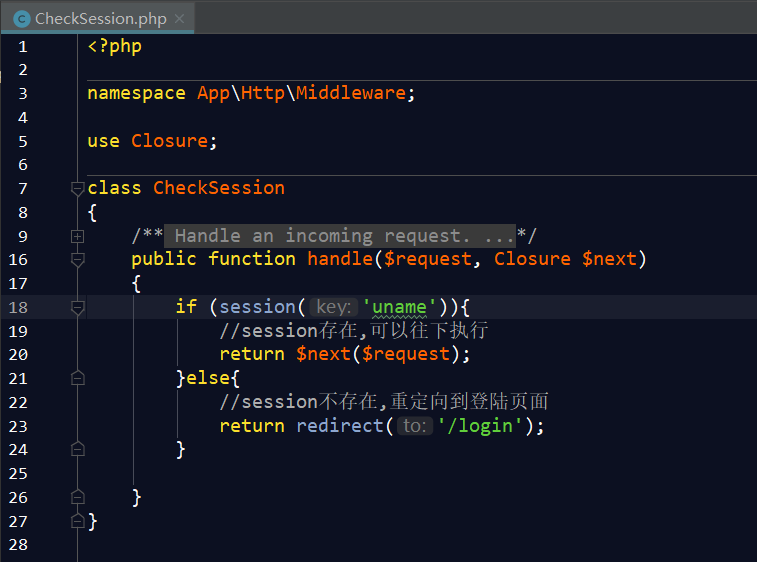


会在 app\Http\Middleware目录下生成中间件



**Step2.编写中间件**

先介绍一个框架的辅助函数session(), 函数可以用来获取或者设置 Session 值：



**Step3.注册中间件**

系统现在还不知道有这样一个中间件存在,所以我们还要去注册一下这个中间件。

所在路径app\Http\Kernel.php



注册内容:



**Step4.编写路由**



流程:

访问/cart

Route::get('/cart',function ()

一看有中间件,那就先走middleware(‘login’)

})->middleware('login');

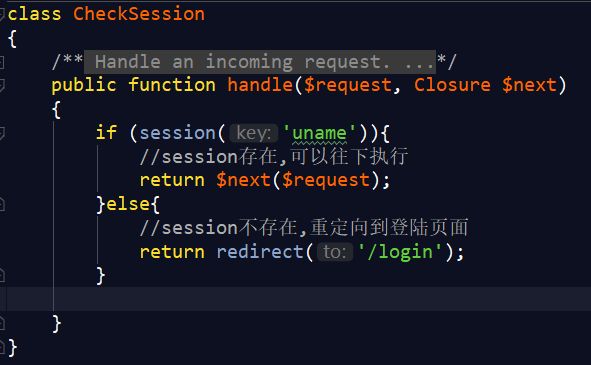
根据login中间件名去Kernel.php找它所指向的类的路径

'login' => \App\Http\Middleware\CheckSession::class,

根据路径找CheckSession类

\App\Http\Middleware\CheckSession::class,

按照类里面的代码执行



# 六、Http请求

**依赖注入（设计模式）了解**

Q：什么叫依赖？

A：如果A类的实现需要B类的支持，就叫做A类依赖于B类。

Q：什么叫注入？

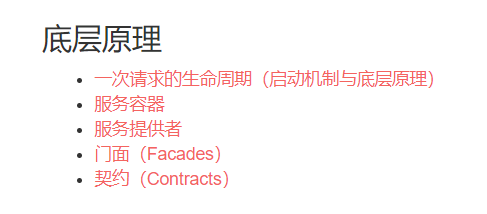
A：一般是在A类的构造方法里传入另B类的对象作为参数。

Q：什么叫容器？

A：容器一般包含两个方法：bind和make

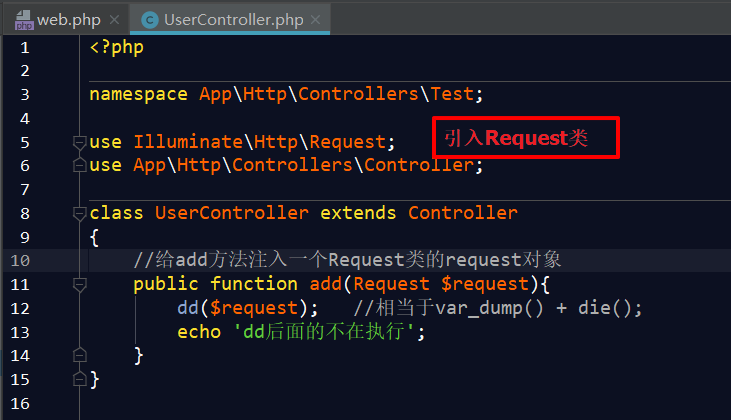
Bind：建立类与一个闭包的联系

Make：产生一个对象，并解决依赖关系

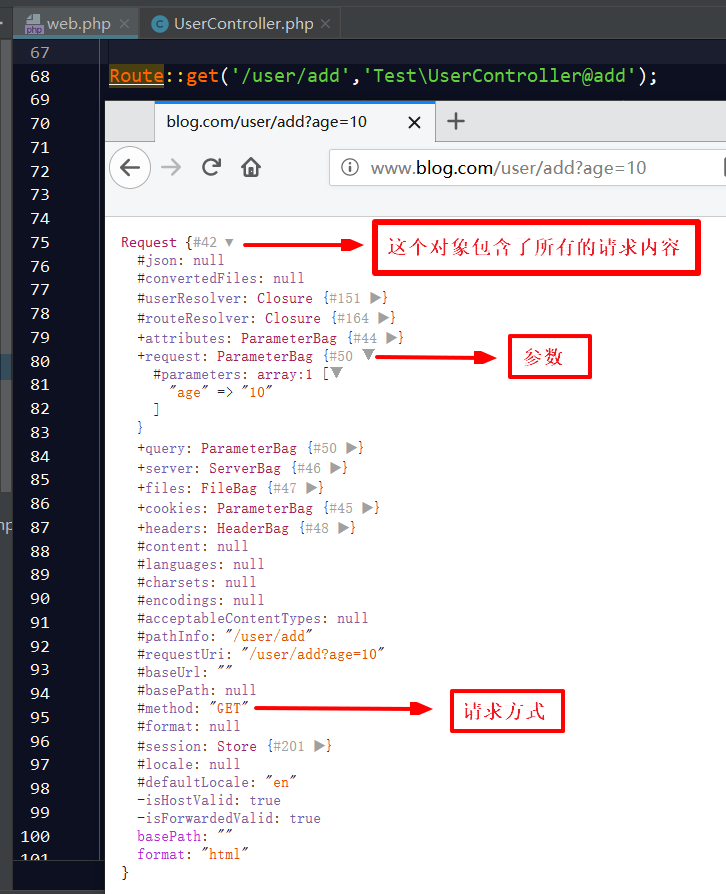


## Http请求的的简单案例

### 1.使用依赖注入Request对象

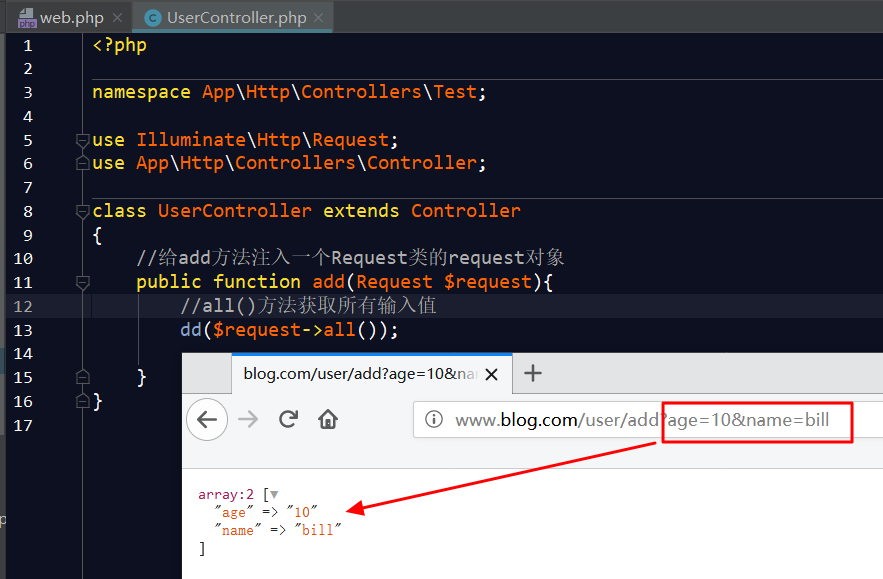


这样就不需要在add方法里面实例化对象了。



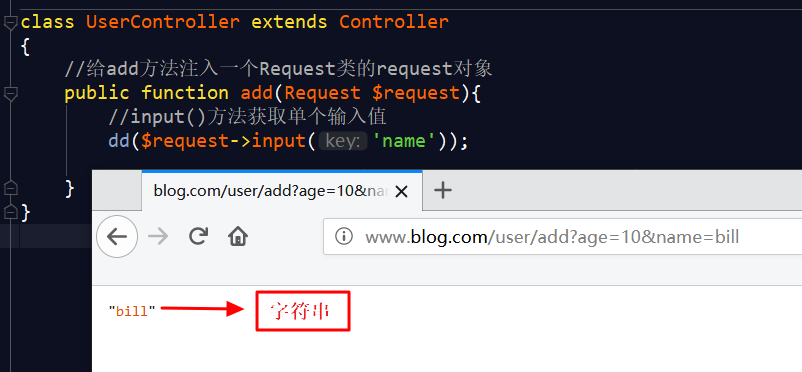
### 2.Request的all()方法

all()方法获取所有输入的值，以数组的格式。



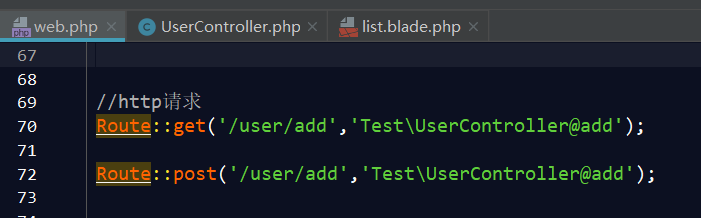
### 3.Request的input方法

input()方法获取单个输入值



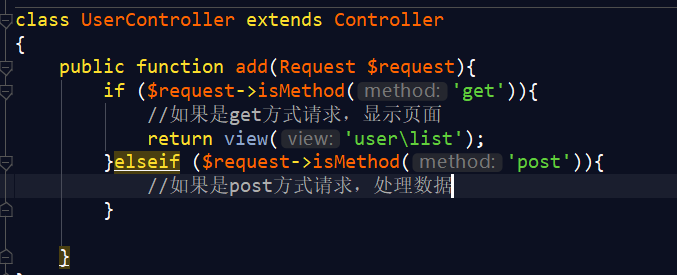
### method()方法

先看看这个路由

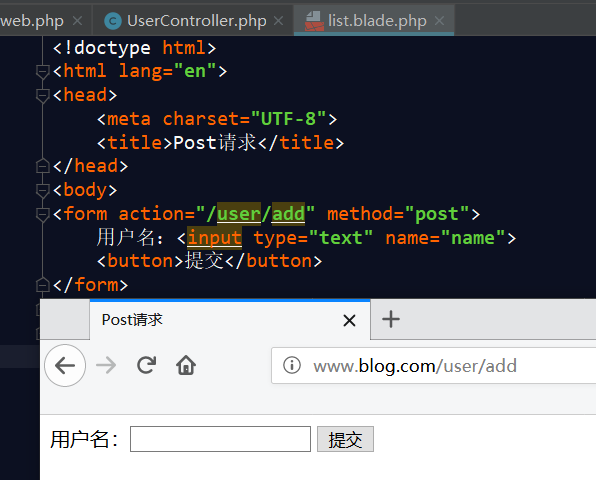


这是add方法

使用isMethod()进行判断



这是页面



点击提交后报错，这是因为框架做了csrf攻击的防范。

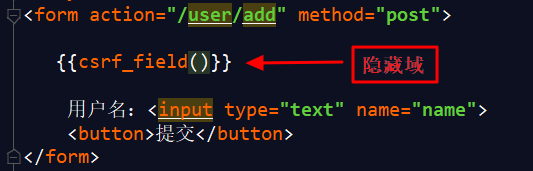
Csrf攻击：跨站脚本攻击，别的域名提交了这个地址。



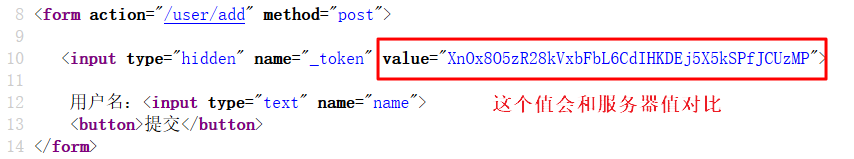
**怎样防范呢？**

在请求的cookie里面都会带一个XSCR-TOKEN值，这是唯一值，它会我们服务器存在的一个session值去对比，如果能够比较成功，就表明这是我们自己家域名提交的内容。

我们只需在form表单里面添加一个隐藏域



下面是页面源代码



再次提交



路由代码优化

Match匹配get和post请求

# 

# 七、Http响应

## 1、什么是http响应

http响应就是服务器返回给浏览器的数据

包括三个方面

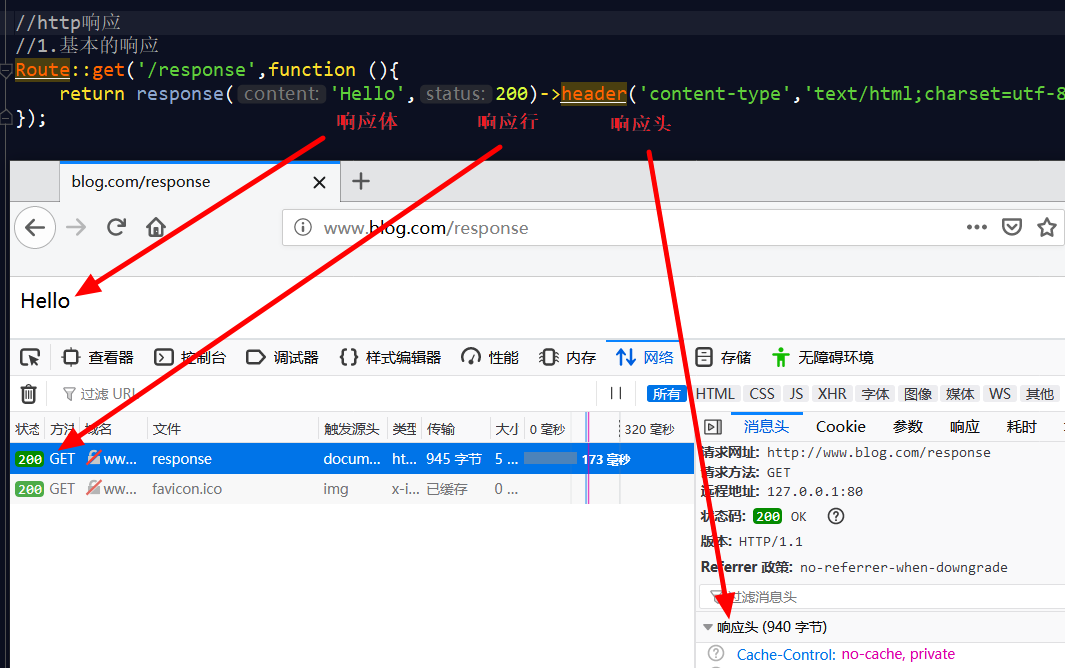
响应行

响应头

响应体（数据）

## 2、http响应案例

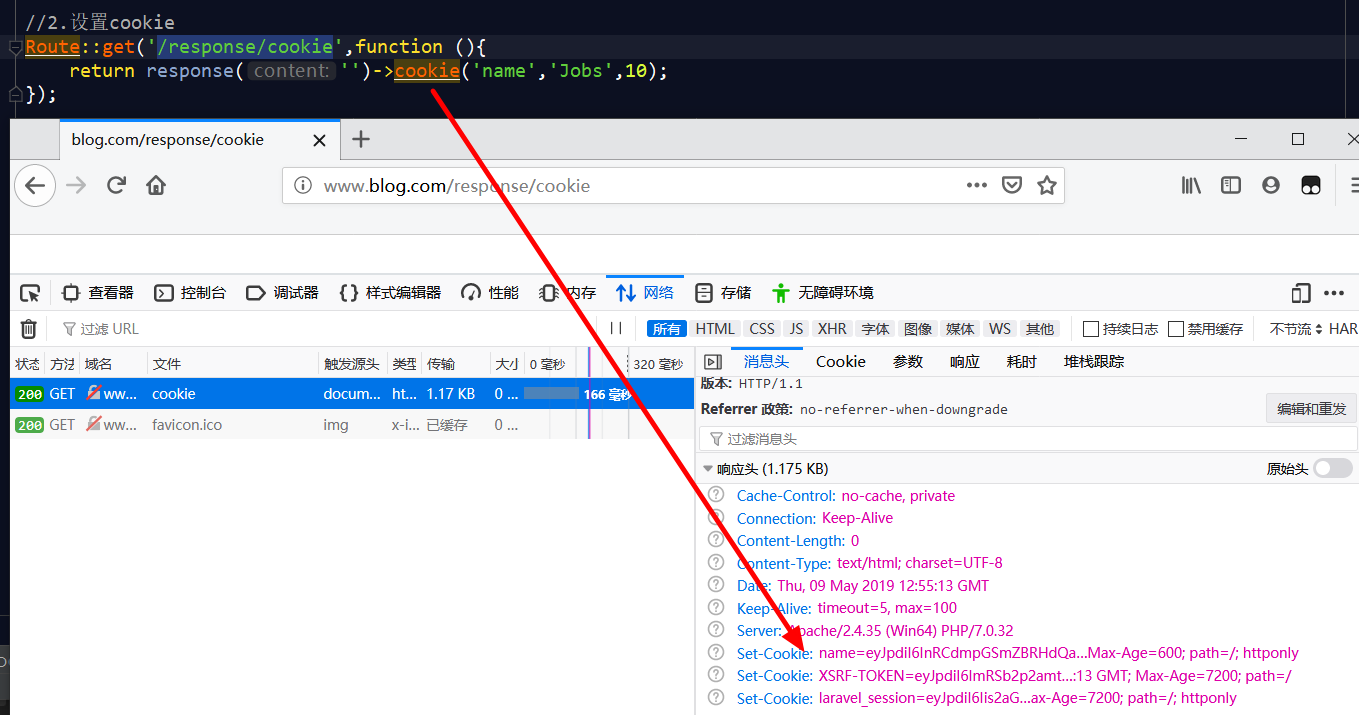
1、设置响应头



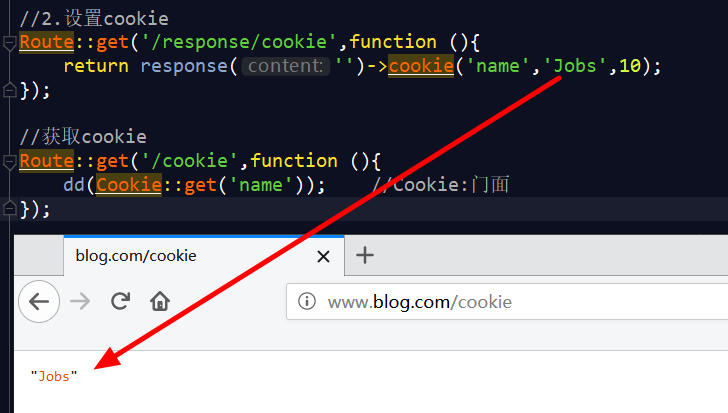
另外可以设置为空，但是不能什么都不写。



2.设置cookie

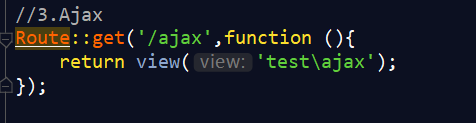


获取cookie

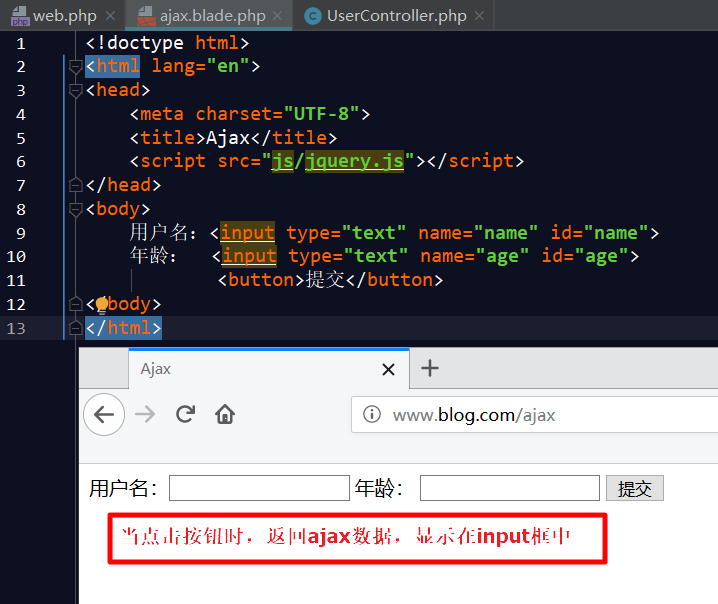


## 3、Ajax返回JSON数据（重点）

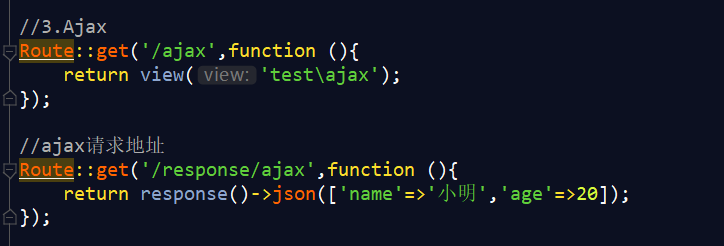
Step1、编写路由



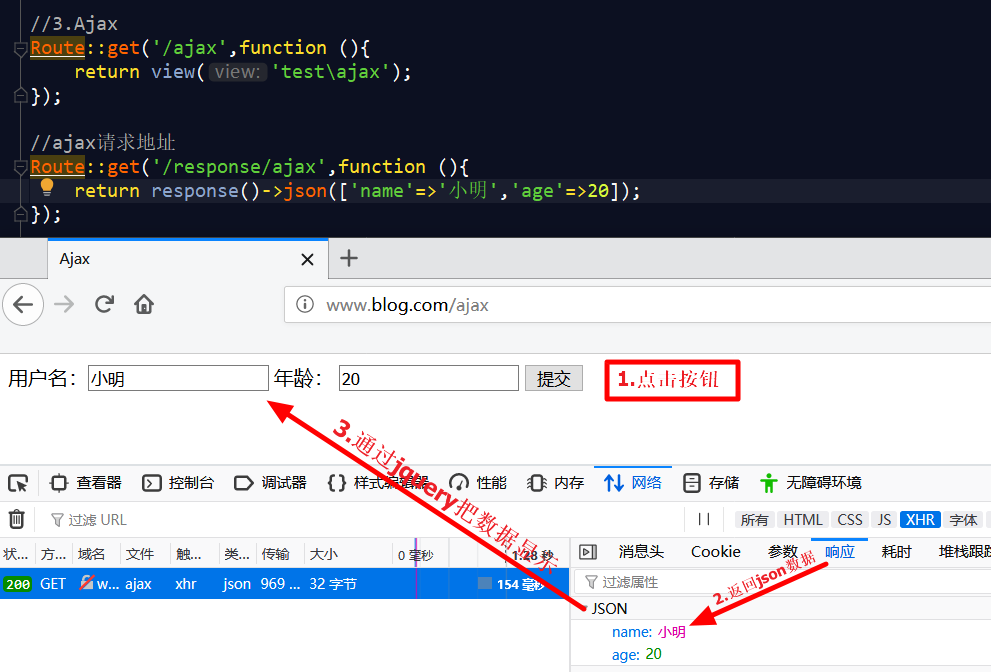
Step2、编写ajax模板



Step3、通过ajax发送get请求



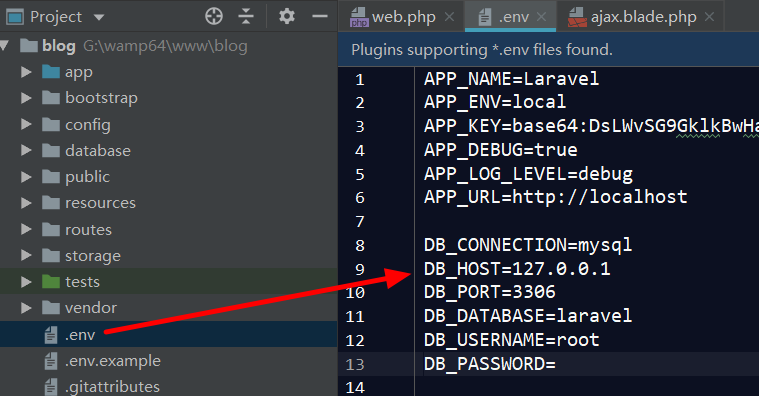
Step4、测试



# 八、数据库

## 1、修改配置文件

.env



Config->database.php 配置文件值如果不存在，使用默认值



## 2、运行原生sql语句

（以后很少使用原生方法，这里学习一下，所以直接在闭包里面测试）

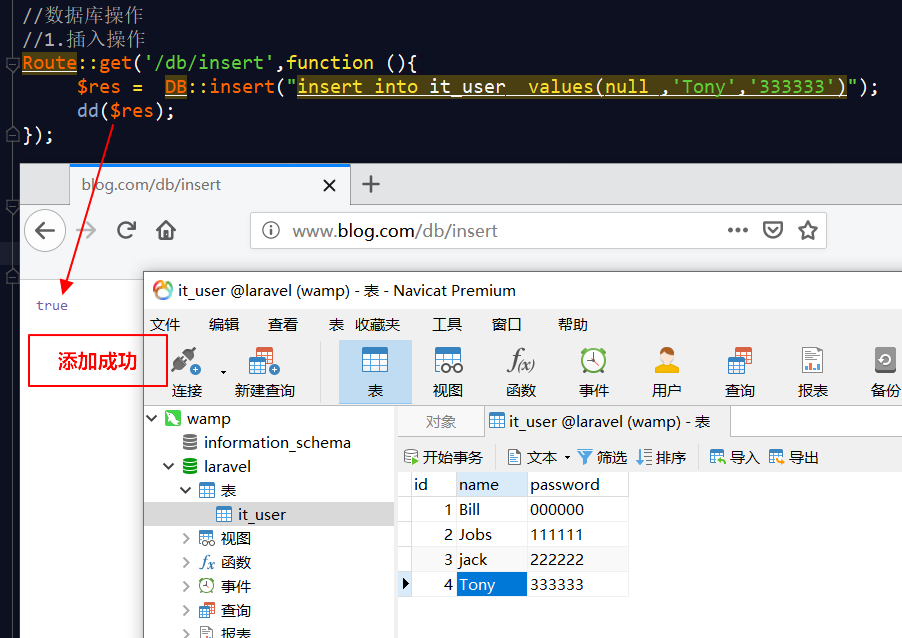
**DB门面**：

相当于PDO为上层提供了统一的操作。

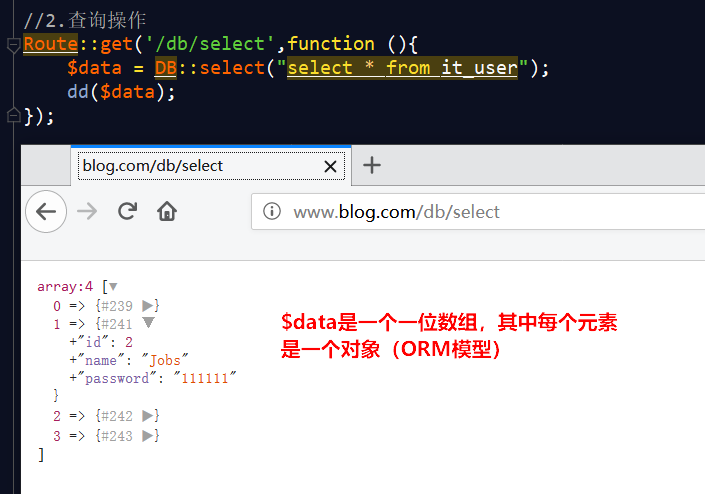
**注意：在控制器里面使用DB，必须先引入DB，**

**在闭包里面不需要事先引入DB门面。**

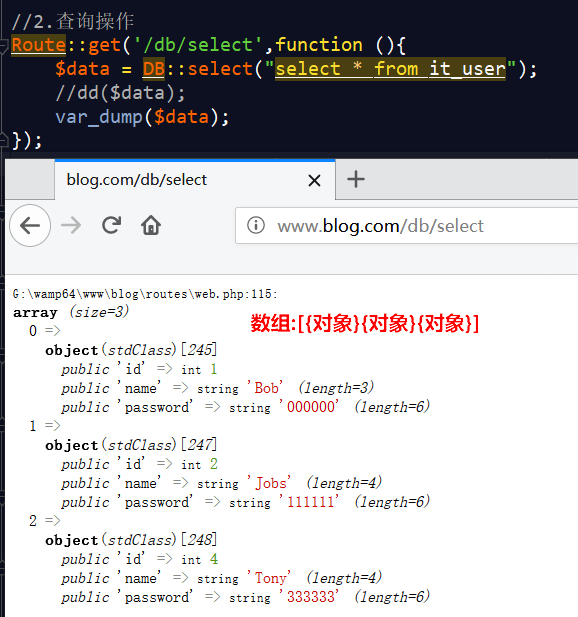
### 1.插入操作



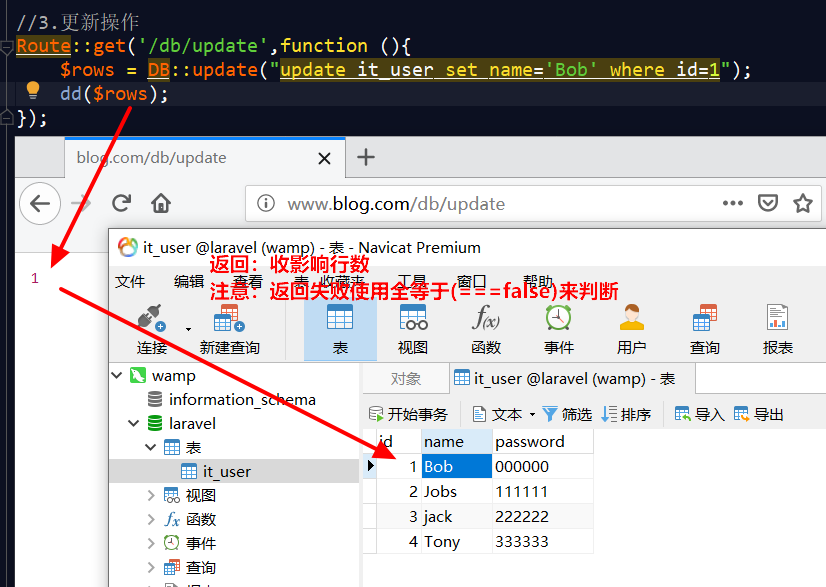
### 2.查询操作



换var\_dump打印一下



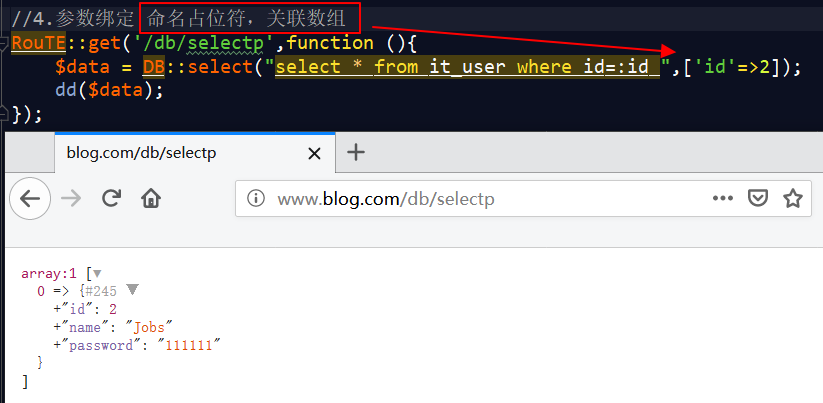
### 3.更新操作



### 4.参数绑定

目的：防止sql注入

命名占位符



### 5.删除delete

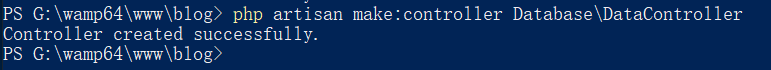


## 3.构造器操作数据库

### 1.什么是构造器

在查询之前，使用DB门面的table方法，**table方法**为给定表**返回一个查询构造器**，允许你在查询上链接更多约束条件并最终返回查询结果。本次在控制器里测试。

1. 生成一个控制器



1. 写路由，为了分辨，每次换了不同的路由。也可以不换。



1. 给控制器引入DB，打印控制器

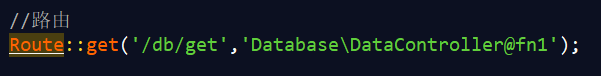


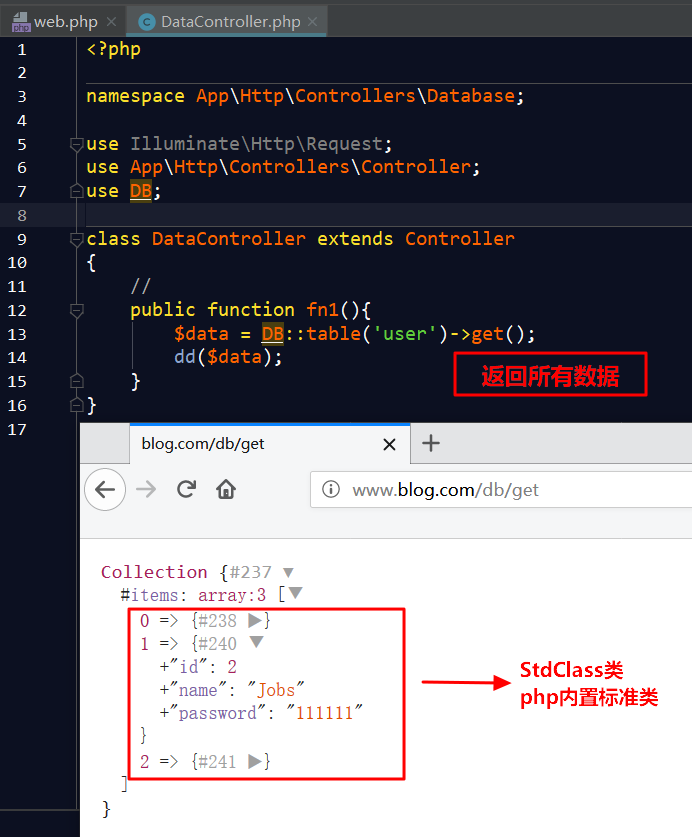
1. **构造器Builder是一个对象，+号的是属性。**



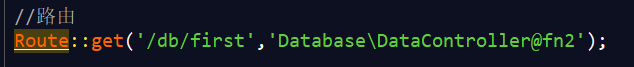
### 2.构造器的方法

#### 1. get获取所有记录。构造器用get



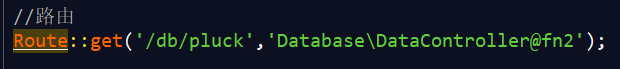


#### 2. first方法获取一行记录





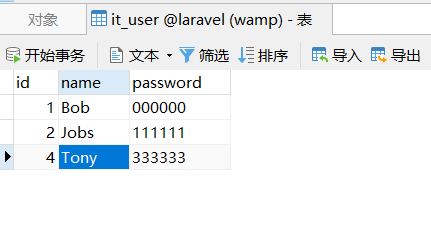
#### 3 pluck方法获取一列记录





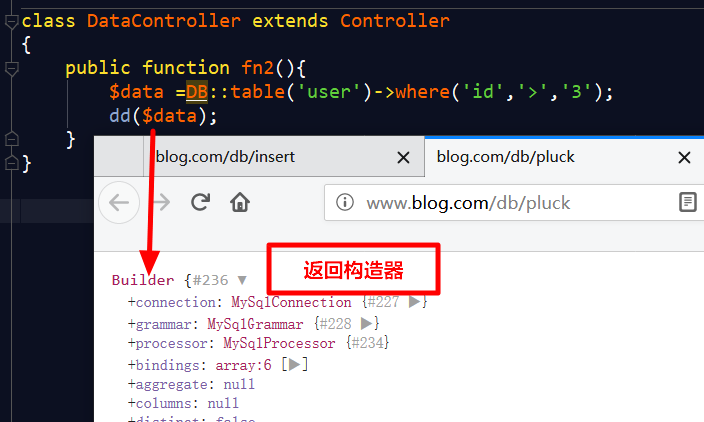
还可以通过第二个参数让结果根据参数排序，不过这个参数必须是本表的字段。





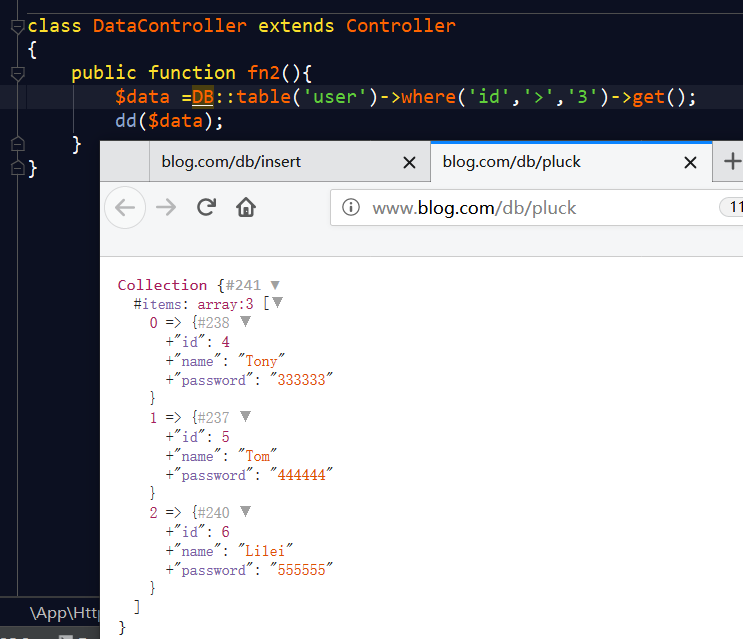
#### 4.where子句

使用查询构建起上的where方法可以添加where子句到查询中，调用where最基本的方法需要传递三个参数。



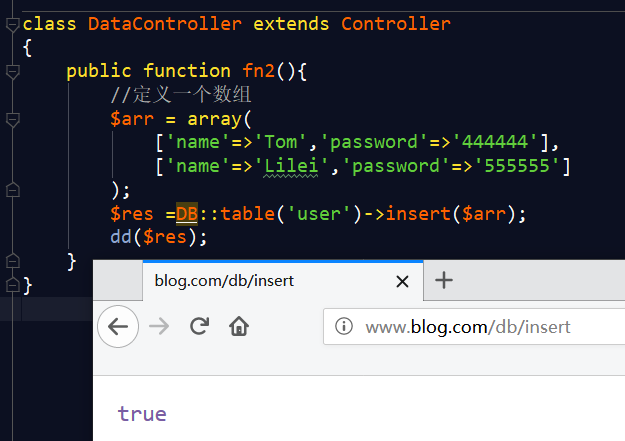
**那么怎么获取里面的值呢？**

**获取构造器的值，当然还是使用get方法了。**

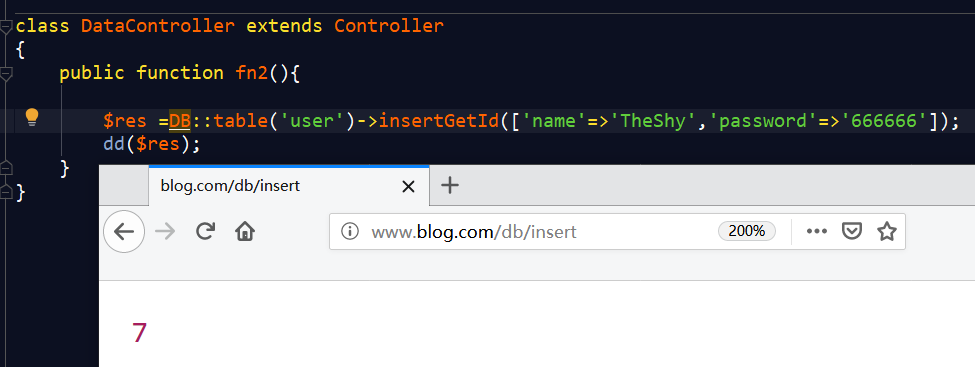


#### 5.插入insert

Insert的参数是数组



插入数据并返回id



1. 更新update
2. 先用where子句找到要更新的记录。
3. 再用update更新记录



1. 删除delete

1. 先用where子句找到要删除的记录。

2. 再用delete删除

返回0是没有删除记录。返回false是删除失败。



# 九、数据库迁移 了解

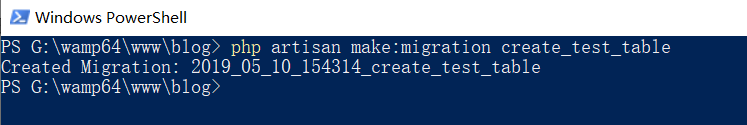
本质就是通过artisan脚本建表，建字段。

底层的实现就是使用PDO执行crate table的SQL语句。

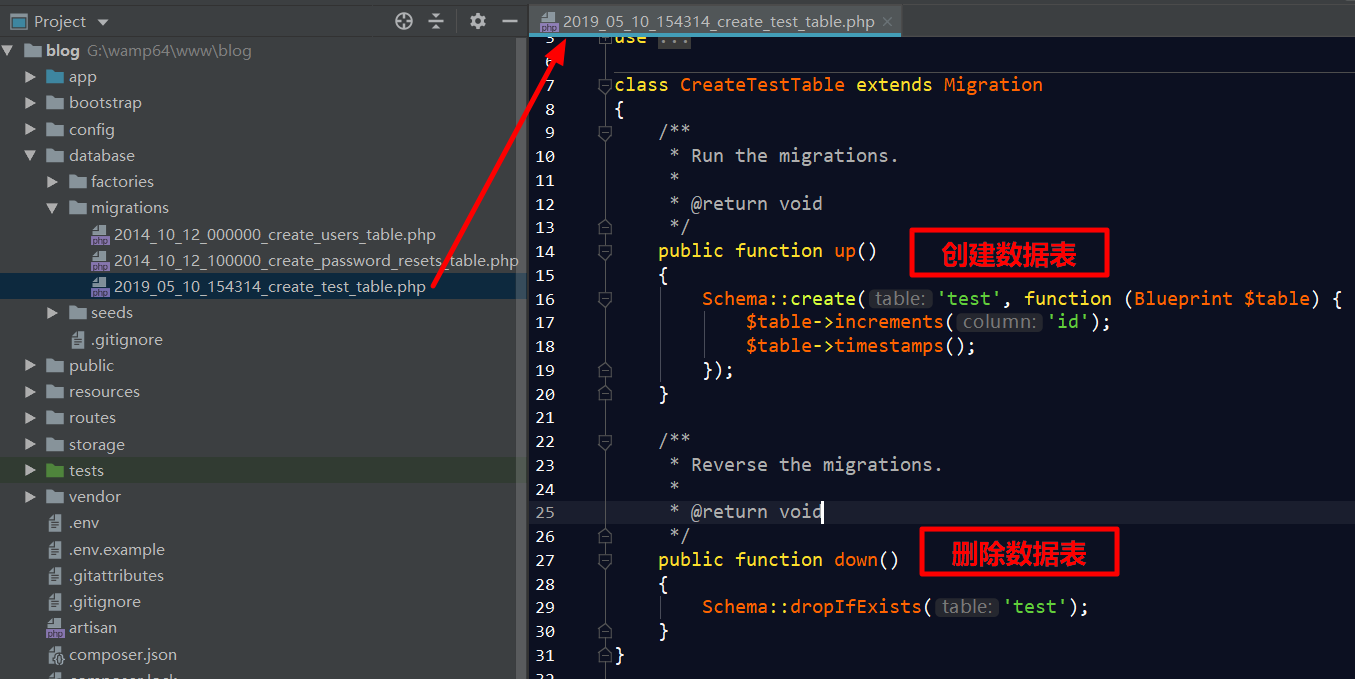
1. 使用migration命令创建迁移文件

切换到artisan目录，执行artisan命令

**php artisan make:migration create\_test\_table**



会在database/migration目录下生成对应的文件，如下



1. 编写迁移文件
2. 执行migrate命令
3. 查看数据库

略。。。

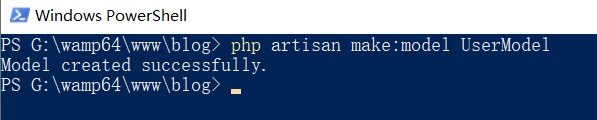
# 十、Elopuent模型

简介：Laravel自带的Eloquent ORM提供了一个简洁的于数据库交互的方式，每张表都对应一个与该表进行交互的“模型”，模型允许在表中进行数据增、删、改、查。

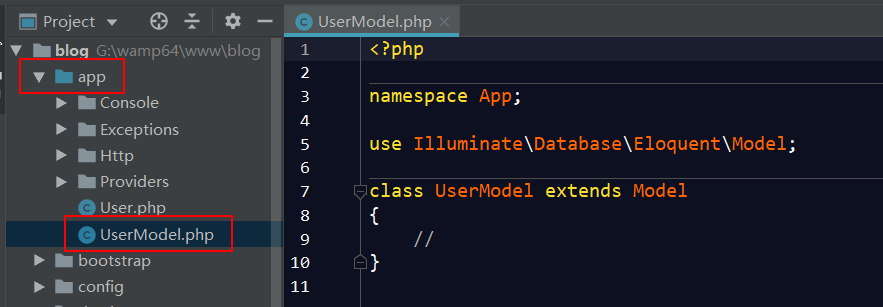
在开始之前，确保在config/database.php文件配置好了与数据库的链接。

## 创建自定义模型

使用artisan脚本创建自定义模型，命令如下：



会在app目录下生成UserModel.php



1. 编写自定义模型

一般要书写这样几个参数

protected $table //真实表名

protected $primaryKey //主键字段，默认为id

protected $fillable //可以操作的字段

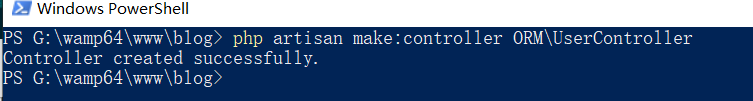


## 2、（案例）用户的CURD

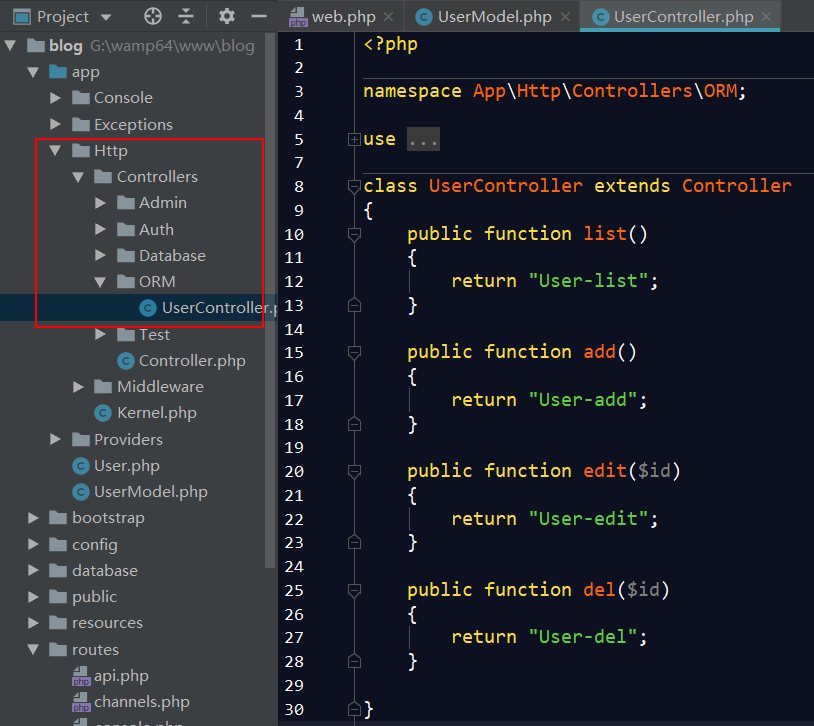
### 1.路由规划



### 2.创建控制器并测试



**编写ORM目录下的这个控制器**



### 3.显示用户信息

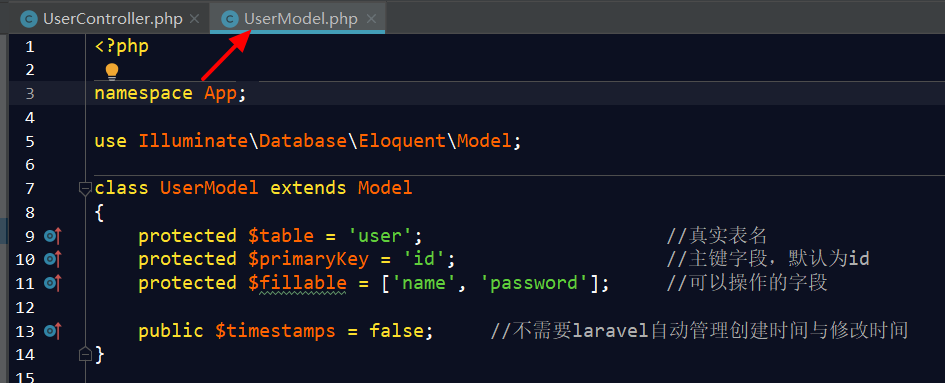
**Step1.创建模板**

创建view/orm/list.php



**Step2.获取数据（重点）**

该使用UserModel了，注意：这个文件的命名空间在App下。



所以我们还需要在UserContriller里面引入UserModel。



**#默认情况下，Eloquent 查询的结果总是返回 Collection 实例。**

**Collection是个集合对象，保存了所有数据**

**可以使用foreach遍历，因为里面实现了foreach接口。**

**可以看作是个高级数组，比数组更好用的对象。**

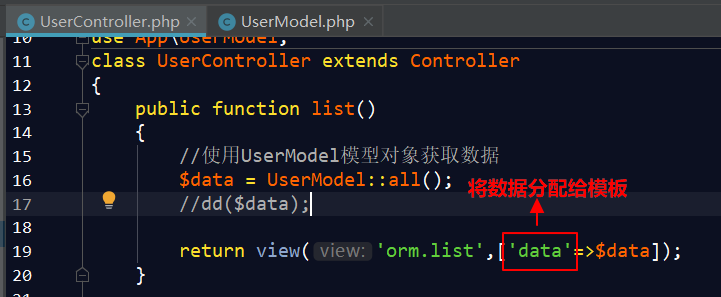
这里collenction一共返回了三个元素，

每个元素都是UserModel模型对象，

模型对象的attributes属性里面的值就是表的数据。



**把php里面的$data变量分配给模板。**



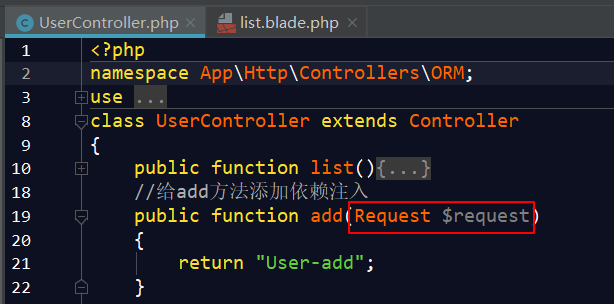
Step3.编写模板



用 **$v->id**的方法访问值

### 4.添加用户信息

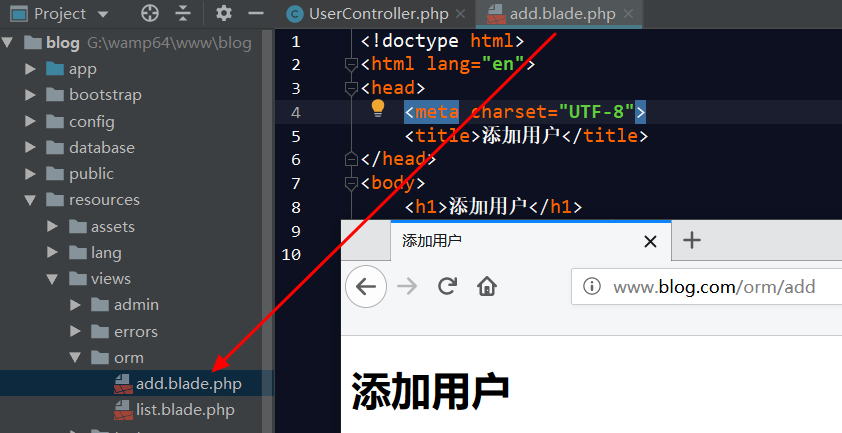
**Step1.添加依赖注入**



**Step2.判断请求模型**



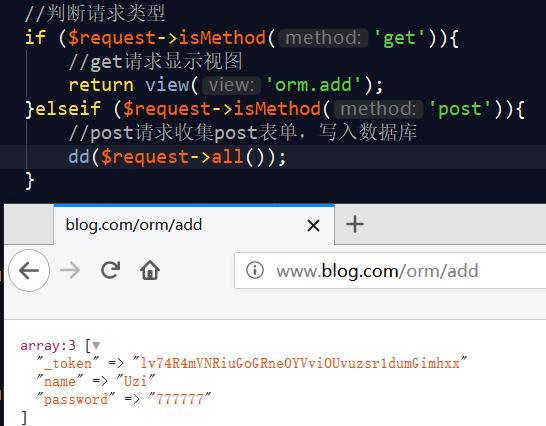
**Step3.为get请请编写模板**

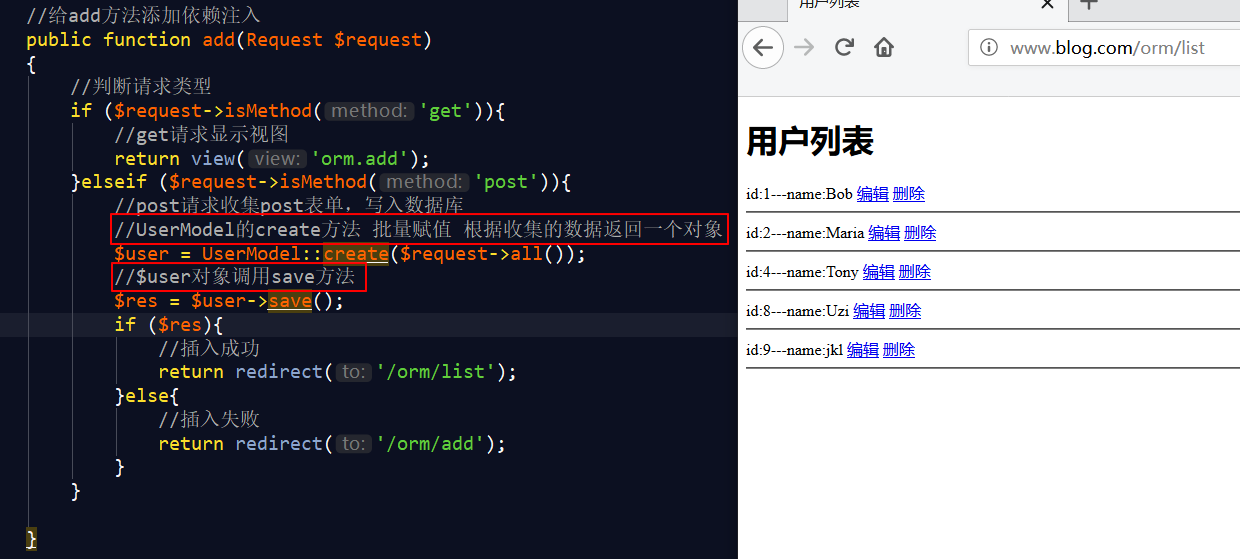




注意：一定不要忘记令牌，（不过你还是会经常忘记的。。。）

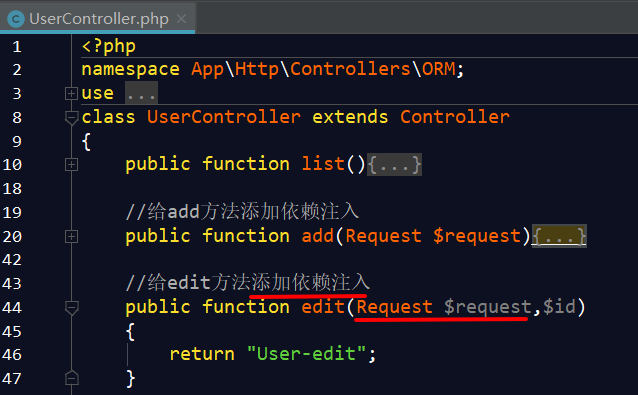
**Step4.为post请求写入库操作**



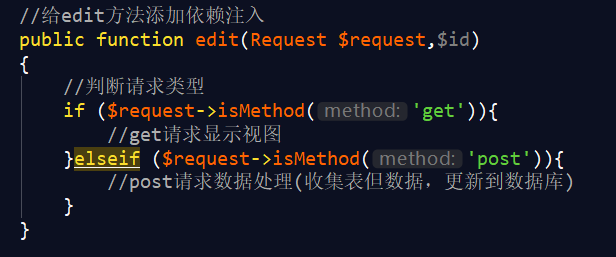


### 修改用户信息

**Step1.添加依赖注入**



**Step2.判断请求类型**



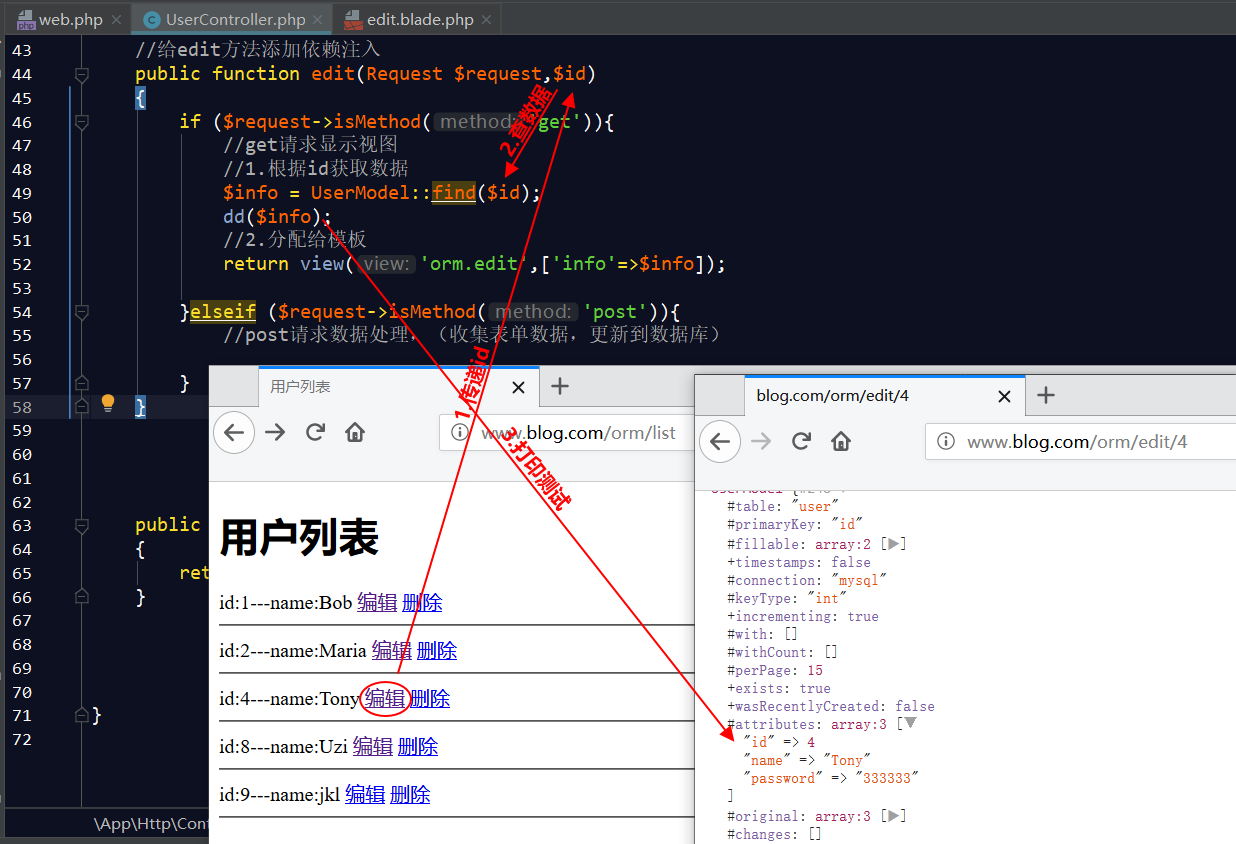
**Step3.为get请求编写模板**



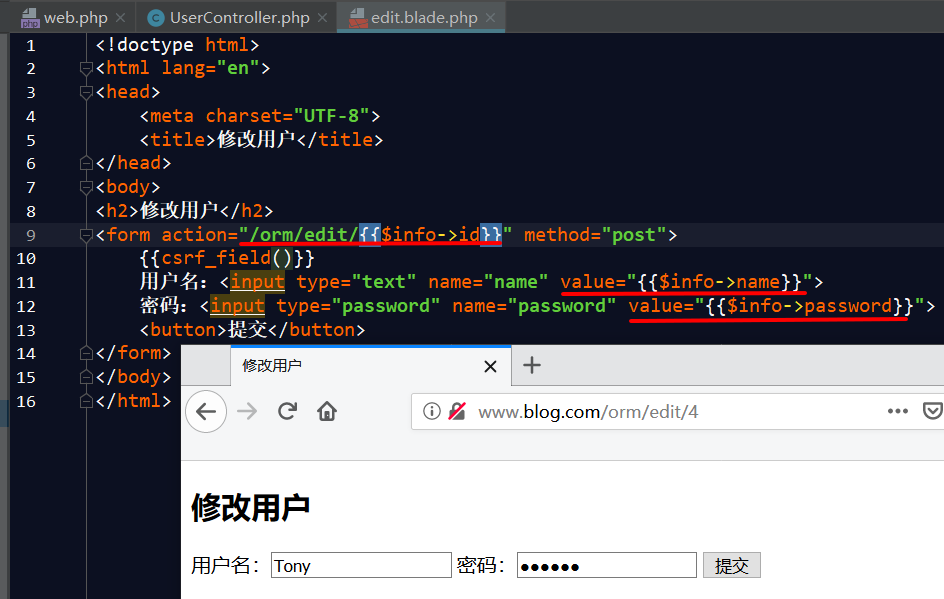
**Step4.为模板分配数据**

**find方法：获取单个模型，UserModel模型，它可以调用一些方法。**

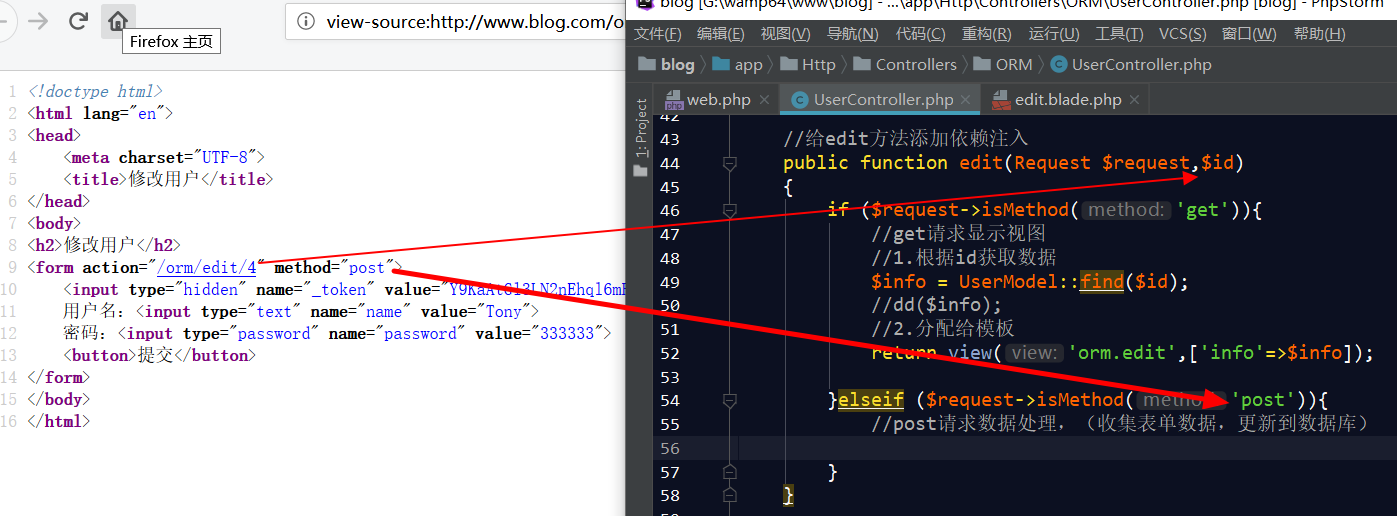
**单个模型是一条数据，用$info；多条数据用$data**



完善模板



是这样传递的



完善代码

**#input()方法获取input表单里面的值。**



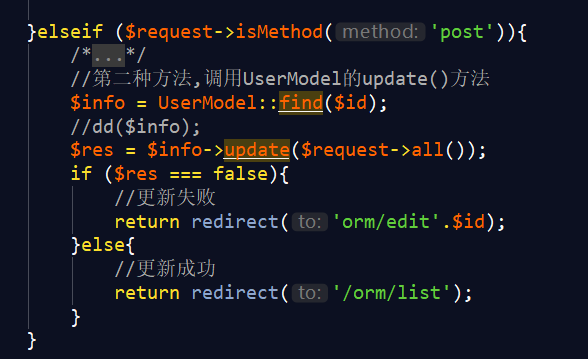
运行





第二种方法

UserModel模型调用update()方法来更新数据



运行

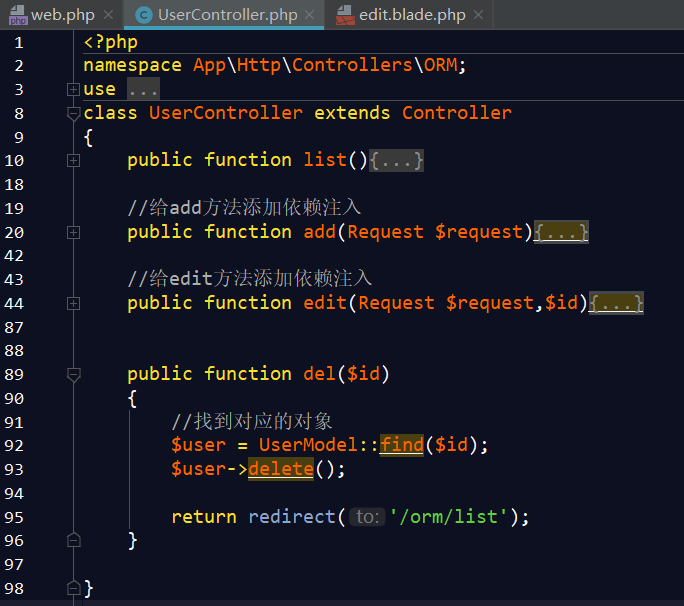




### 删除用户信息

**要删除一个模型(UserModel)调用UserModel的delete()方法。**

这是物理删除，还有个软删除。



**运行**

