# ThinkPHP框架\_01

# 今日目标13

* 能够理解并使用composer

PHP一个包管工具 解决源码包 依赖问题

* 能够安装Thinkphp5.1框架

composer create-project --prefer-dist topthink/think=[版本号] 目录名称

安装之前一定要切换国内源，否则成功的概率很小。

* 能够说出Thinkphp5.1重要目录作用
* 能够理解Thinkphp5.1框架中请求的生命周期(执行流程)
* 能够理解并使用路由注册功能

Route::get post put delete any不推荐

分组

路由参数

* 能够在Thinkphp5框架中定义控制器
* 能够使用命令行创建分组和控制器

php think build --module 分组名

php think make:controller --plain 控制器名称（首字母大写）

* 能够理解参数绑定和依赖注入
* 能够理解Request请求类的基本使用
* 能够返回json格式数据
* 能够进行模板的渲染
* 能够在模板中进行变量输出
* 能够在模板中使用foreach等内置标签

# 一、Thinkphp框架概述

## 1.1、介绍

在web领域，PHP是所有编程语言中比较受欢迎的一门语言！ PHP已经诞生出几十种编程框架！但国内最热门和使用率最好的框架有Thinkphp和Laravel这两款PHP框架！

## 1.2、为何选择thinkphp

ThinkPHP是国内应用程度最高的一个PHP框架，简单易学是他的特点。很多人和企业选择它的原因，是因为它是国人开发，拥有非常丰富的中文文档和中文社区，有问题可以第一时间找到解决方案，可查询到的资料多。

## 1.3、市场薪资



# 二、composer简介

## 2.1、什么是composer

composer英文单词意思：音乐指挥者

在windows系统里边安装一个软件，通常比较方便的方式是去”软件管家”里边找到软件并下载安装使用，同理其它的系统也有类型的软件管理工具来方便我们查询、安装所需要的软件，让安装软件工具便于的快捷方便。

php语言项目本身也提供类似”应用市场”的功能，就是composer，我们可以去composer中查找各种可以使用的功能(例如验证码类、分页类等等)，并引入到自己的项目中使用，而不用关心它们的引用和依赖，从而让我们开发变得更加的方便和快速。

composer是PHP中用来管理依赖关系的工具

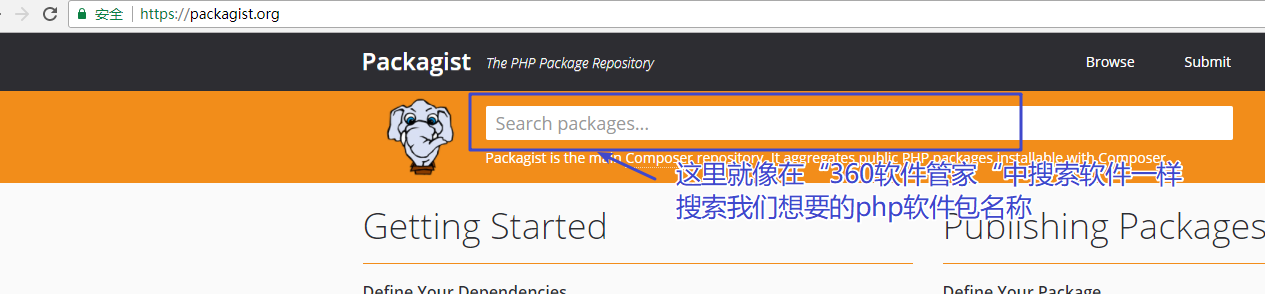
**一句话，composer是一个工具，是为php项目准备的软件管家。**

官网网址：<https://getcomposer.org/>

中文网址：<http://www.phpcomposer.com/>

composer资源仓库或应用市场

网址：<https://packagist.org/>



## 2.2、工作原理



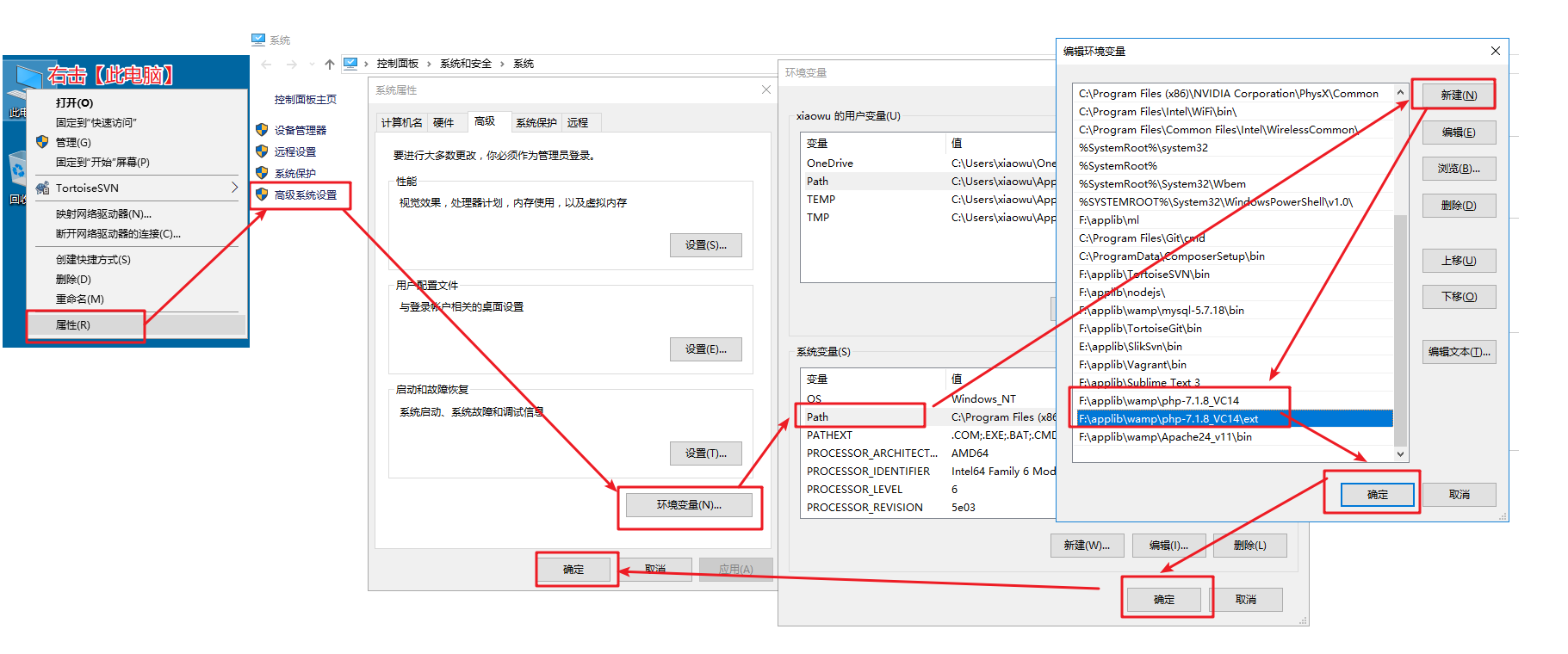
如上图，composer可以去packagist应用市场里边下载软件，但是该市场只给返回软件的地址，对应的软件都是在github里边存储的，最终下载的软件是从github返回的。

## 2.3、composer安装【重点】

### 2.3.1、配置PHP环境变量

因为我们马上要使用的thinkphp5.1需要最低的PHP版本为PHP >= 5.6.0所以，这里你所需要安装的PHP版本一定比这个版本要高。

**windows设置环境变量(windows 10)**



设置完毕后，重新打开cmd窗口，输入 php –v



### 2.3.2、开启相关php扩展

同时也要开启如下扩展 php.ini文件

**extension=php\_openssl.dll**

**extension=php\_pdo\_mysql.dll**

**extension=php\_mbstring.dll**

**extension=php\_fileinfo.dll**

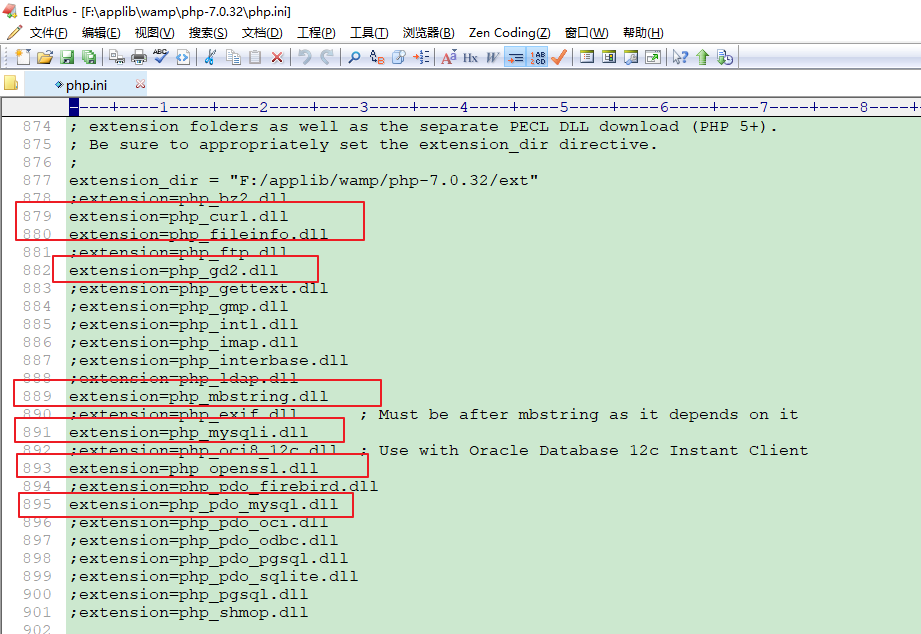
**extension=php\_curl.dll**

**extension=php\_gd2.dll**

**extension=php\_soap.dll**

**extension=php\_mysqli.dll**

**extension=php\_sockets.dll 【可选】**



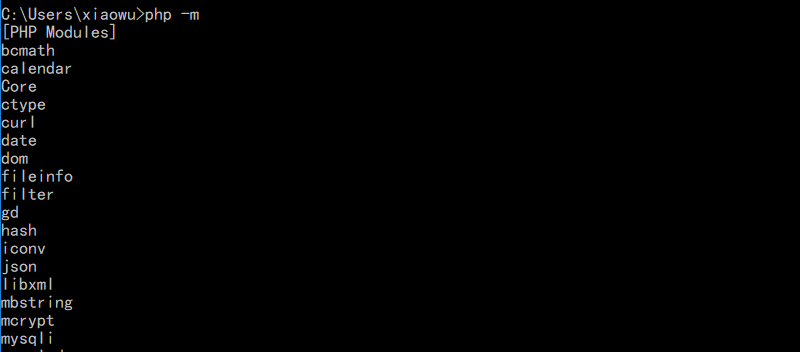
注：命令行下则打开后就生效，apache或nginx就需要重启相关的服务。

**查看命令行下可用扩展**

php –m

加管道过滤

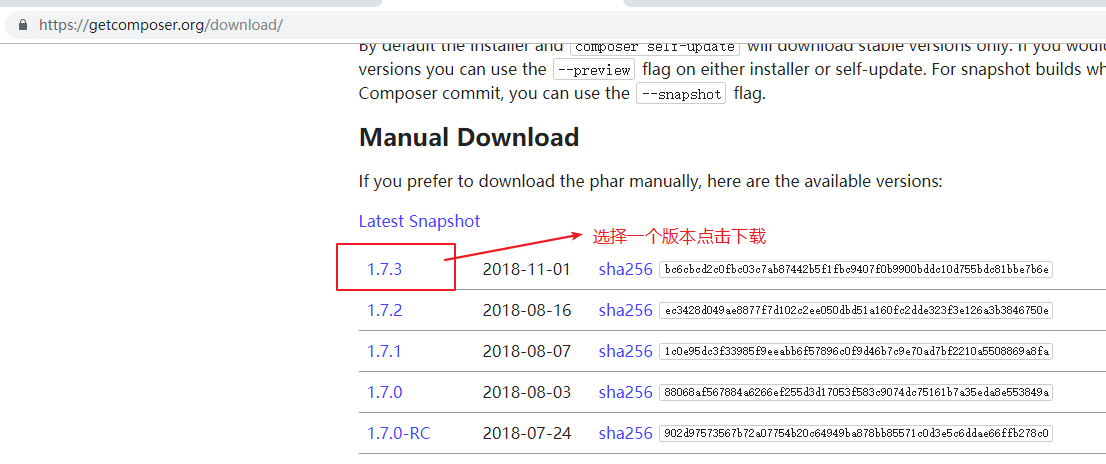
php –m | findstr gd # window



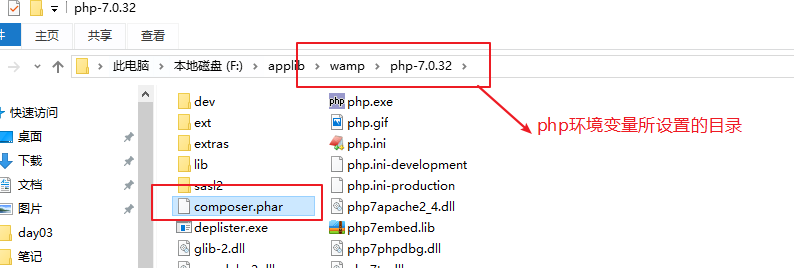


### 2.3.3、安装composer

下载composer.phar：<https://getcomposer.org/download/>

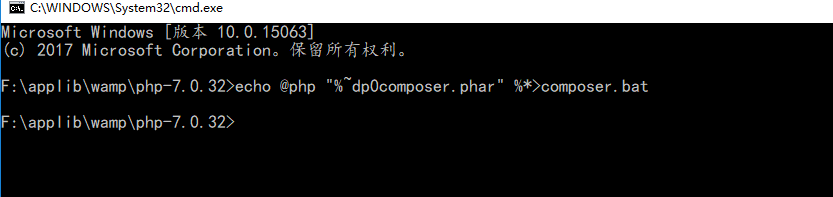


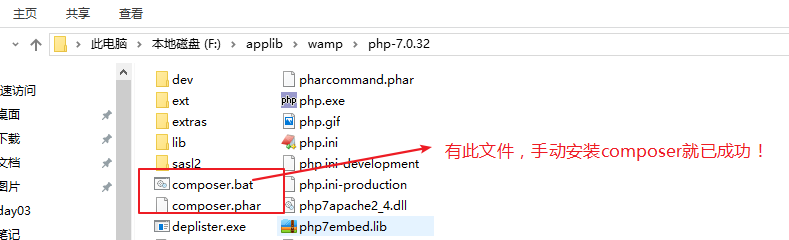
把下载好的composer.phar文件复制到PHP环境变量指定的目录中



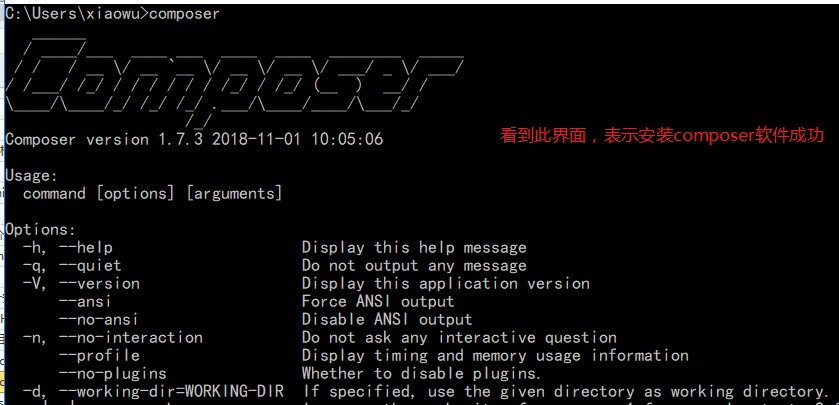
**命令创建bat文件**

echo @php "%~dp0composer.phar" %\*>composer.bat





### 2.3.4、检查composer是否安装成功



## 2.4、切换composer镜像

由于众所周知的原因，国外的网站连接速度很慢。因此安装的时间可能会比较长或是安装不成功，我们建议通过切换composer国内镜像来解决此类问题。当然有钱的情况下，可以购买相关的渠道来访问国外网络。



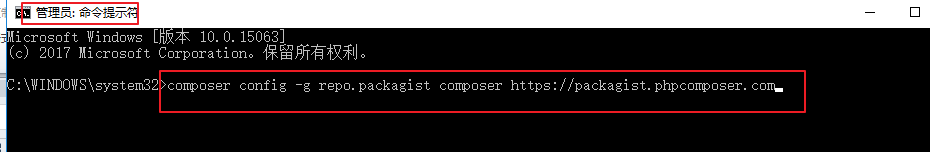
以管理员身份打开cmd命令行窗口（windows用户）或控制台（Linux、Mac 用户）并执行如下命令：

composer config -g repo.packagist composer <https://packagist.phpcomposer.com>

注意：最近phpcomposer镜像存在问题，可以改成

备选方案

composer config -g repo.packagist composer <https://packagist.laravel-china.org>



执行完毕，源切换为国内镜像，下载的速度会有所提升。

注：

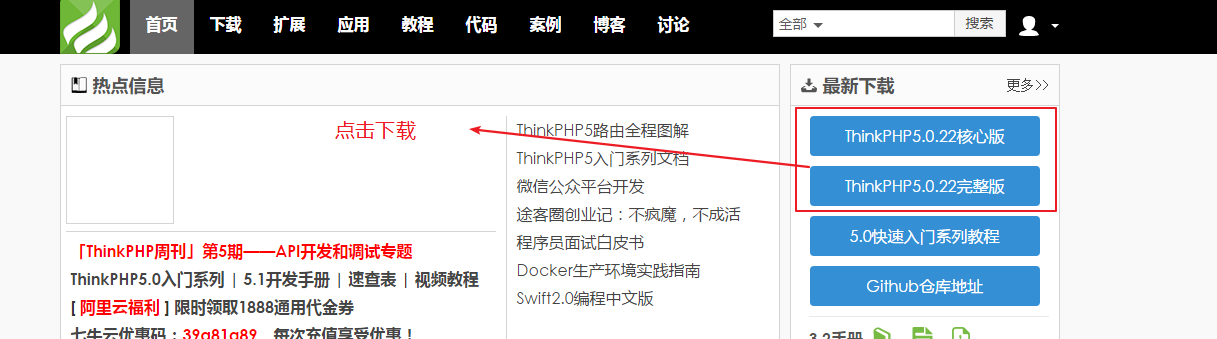
composer网站问题，一定要切换为国内镜像，如有条件可使用vpn

windows 7以上，mac或linux系统，一定要考虑到权限的问题。

# 三、安装与启动Thinkphp

## 3.1、安装thinkphp5.1

如果安装tp5.1之前的版本，则可以直接在官网下载即可



### 3.1.1、环境要求

ThinkPHP5.1的环境要求如下：

PHP >= 5.6.0

PDO PHP Extension

MBstring PHP Extension

第一个要求 命令行下PHP版本要达到要求



apache环境中PHP版本达到要求



两个终端中的环境要保持一致

### 3.1.2、安装

thinkphp5.1之前可以在thinkphp官方网站进行下载源码包，但是thinkphp5.1版本开始，官网不再提供下载版本，请使用Composer或者git方式安装和更新。

composer create-project [--prefer-dist] topthink/think=[版本号] tp5

如果不写版本号，则下载 php版本所支持的最高的稳定版本

例：

composer create-project --prefer-dist topthink/think=5.0.\* tp5

注：

composer 表示执行composer程序

create-project 通过composer去创建项目

topthink/think 框架的名称

--prefer-dist 优先下载压缩包方式，而不是直接从github上下载源码

~5.1 >=5.1 < 6.0

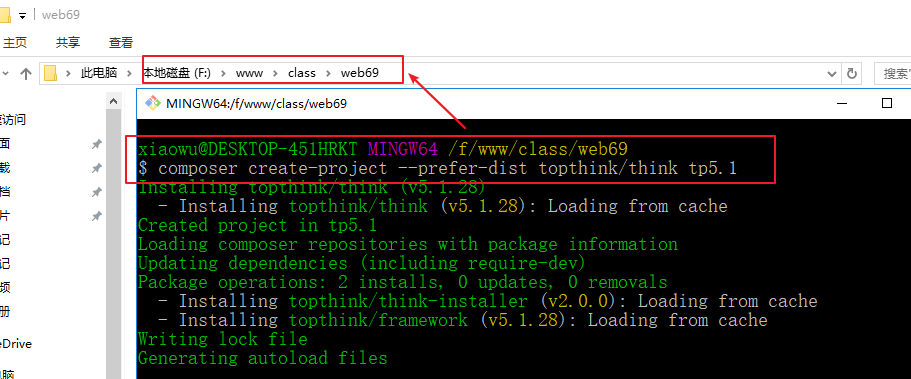
^5.1 >=5.1 < 6.0

~5.1.0 >=5.1.0 < 5.2.0

^5.1.0 >=5.1.0 < 6.0

最终方案

composer create-project --prefer-dist topthink/think tp5.1





## 3.2、启动thinkphp

### 3.2.1、虚拟主机 方案一

配置apache2.4

# 设置虚拟主机

<VirtualHost \*:80>

# 设置网站目录

DocumentRoot "F:/www/class/web69/tp5.1/public"

# 设置网站域名

ServerName www.tp.com

# 错误日志

ErrorLog "F:/www/class/web69/tp5.1/logs/error.log"

# 成功日志

CustomLog "F:/www/class/web69/tp5.1/logs/access.log" combined

# 设置目录访问权限

<Directory "F:/www/class/web69/tp5.1/public">

# 指定目录启用特怀

Options Indexes FollowSymLinks

# 是否允许使用.htaccess文件

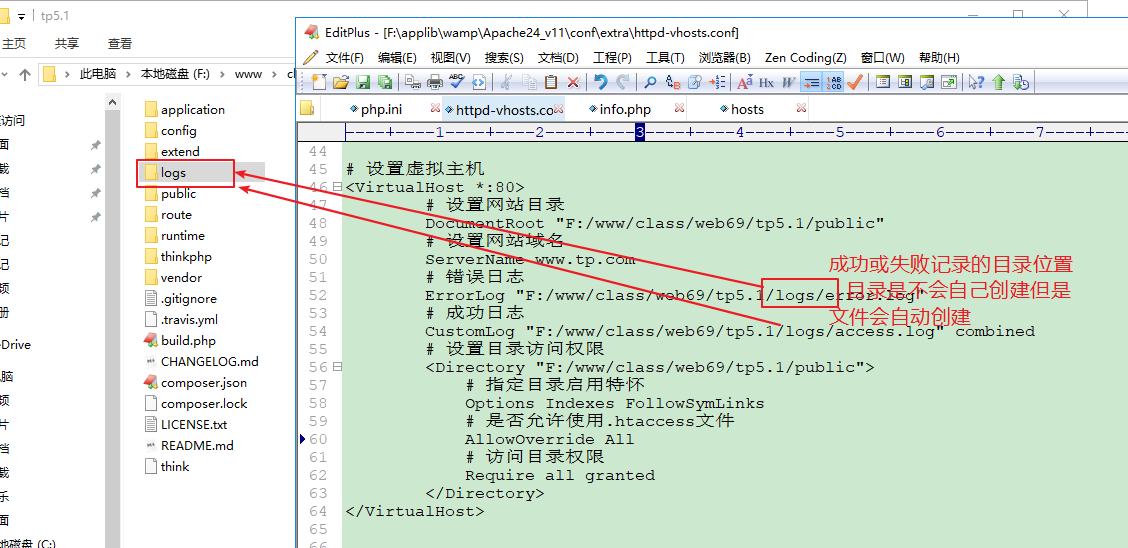
AllowOverride All

# 访问目录权限

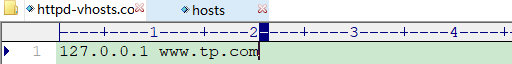
Require all granted

</Directory>

</VirtualHost>

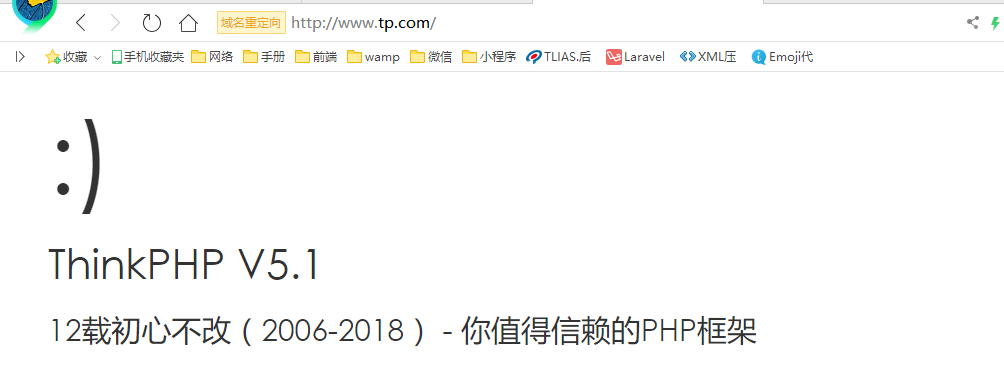


设置hosts文件



设置完成后，一定要重启一下apache服务，让配置生效

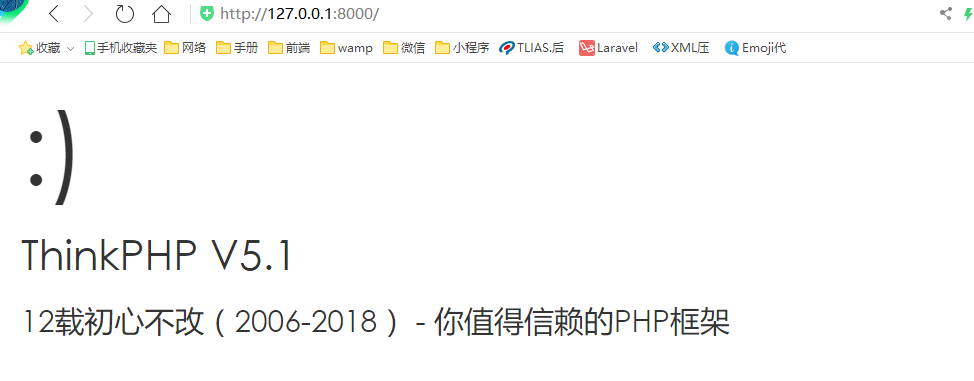
效果：通过浏览器看到如下图所示效果，表示，thinkphp5.1运行已经配置完成



### 3.2.1、命令行运行 方案二

php think run





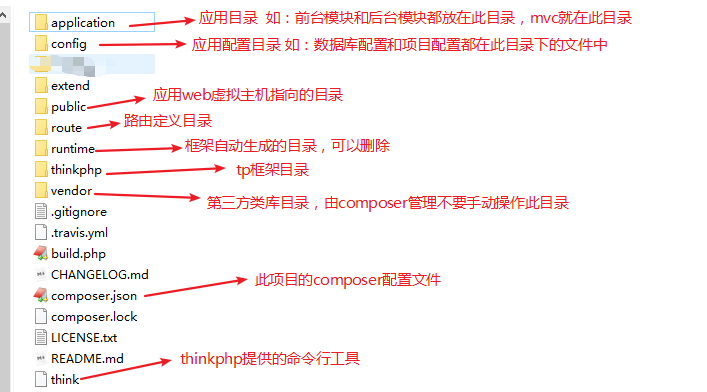
注：生产环境下一定要运行在apache或nginx环境下

命令行模式启动不支持热加载配置。需要手动停止再次启动

# 四、框架基础

## 4.1、目录结构【重点】

<https://www.kancloud.cn/manual/thinkphp5_1/353950>



**runtime目录在linux和mac下面一定要设置可写的权限**

**重点的目录和文件**

|  |  |
| --- | --- |
| **目录** | **功能** |
| application | 应用目录，MVC就在此目录中，也是实现业务代码的所在的目录 |
| application/common | 公共模块目录，在application可以自定创建自己的模块，但在common定义的函数和模型都是公用的 |
| application/common.php | 公共函数库文件 |
| config/app.php | 应用主配置文件 |
| route | 路由文件目录 |
| public | 虚拟主机指向的目录 |
| application/[index或admin] | 表示index模块和admin模块 [自定义创建的] |

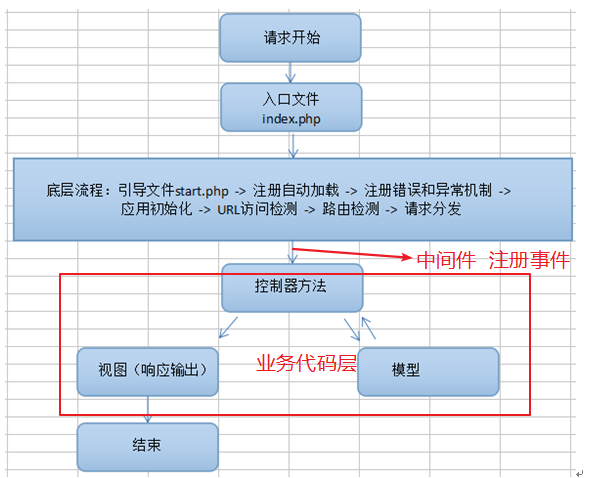
## 4.2、thinkphp命名规范

<https://www.kancloud.cn/thinkphp/php-fig-psr/3141>



## 4.3、请求的生命周期

框架的生命周期，也就是请求的执行流程：



# 五、路由

**什么是路由**

将用户的请求按照事先规划的方案提交给指定的控制器和 方法来进行处理。

Thinkphp框架提供了两种路由规则

* pathinfo模式
* 自定义路由规则【推荐】

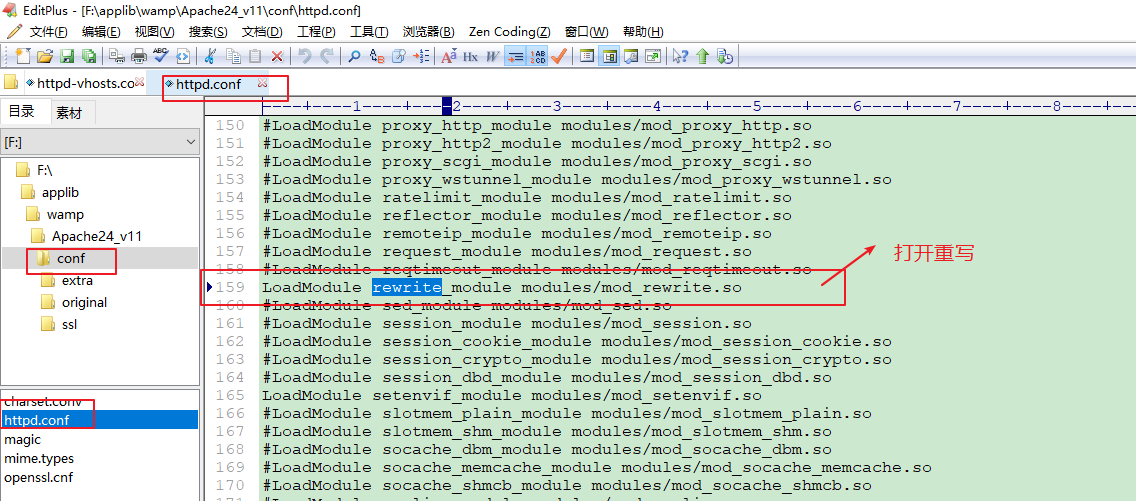
进而可以让URL更规范以及优雅，提高网站的安全和网站URL访问的友好度。

Route类注册使用think\facade\Route类静态调用 think\Route.php

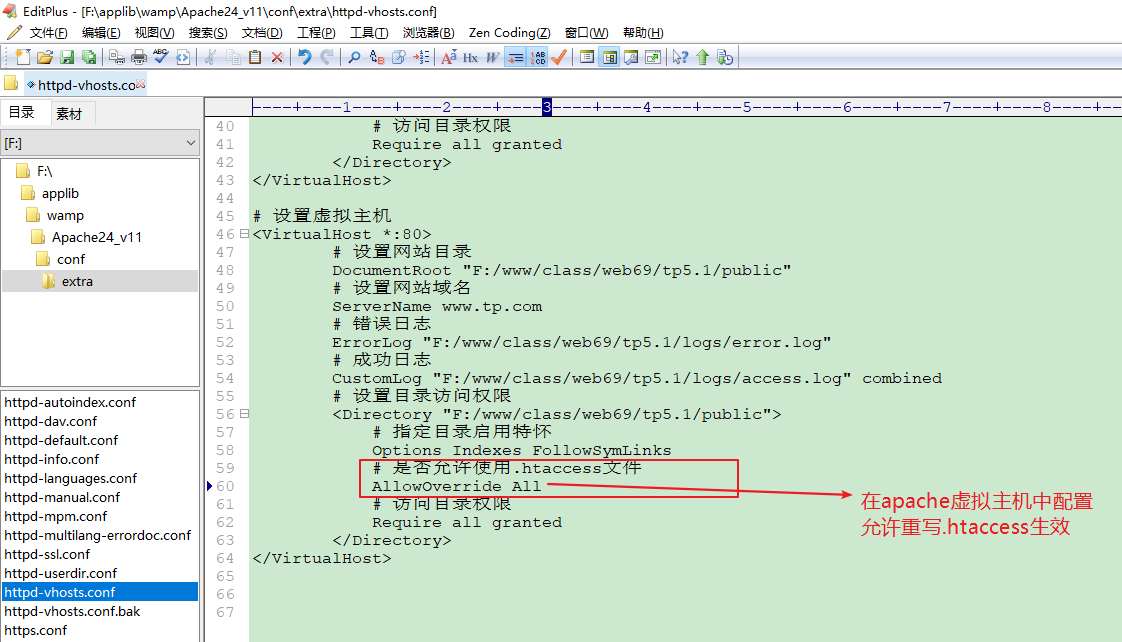
注：ThinkPHP5.1的路由定义更加对象化，并且默认开启路由（不能关闭），如果一个URL没有定义路由，则采用默认的PATH\_INFO 模式访问URL。

## 5.0、隐藏index.php文件

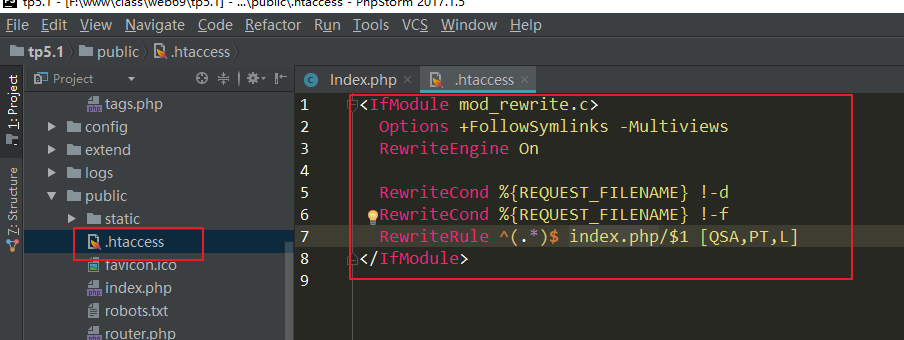
打开apache重写模块



配置虚拟主机中允许重写



在入口文件中要有.htaccess文件



重启apache服务，让配置生效。访问时就可以不用带index.php文件了。

## 5.1、路由相关配置

**强制路由**

在config/app.php配置文件中设置

'url\_route\_must' => true,

注：将开启强制使用路由，这种方式下面必须严格给每一个访问地址定义路由规则（包括首页），否则将抛出异常。

**路由缓存**

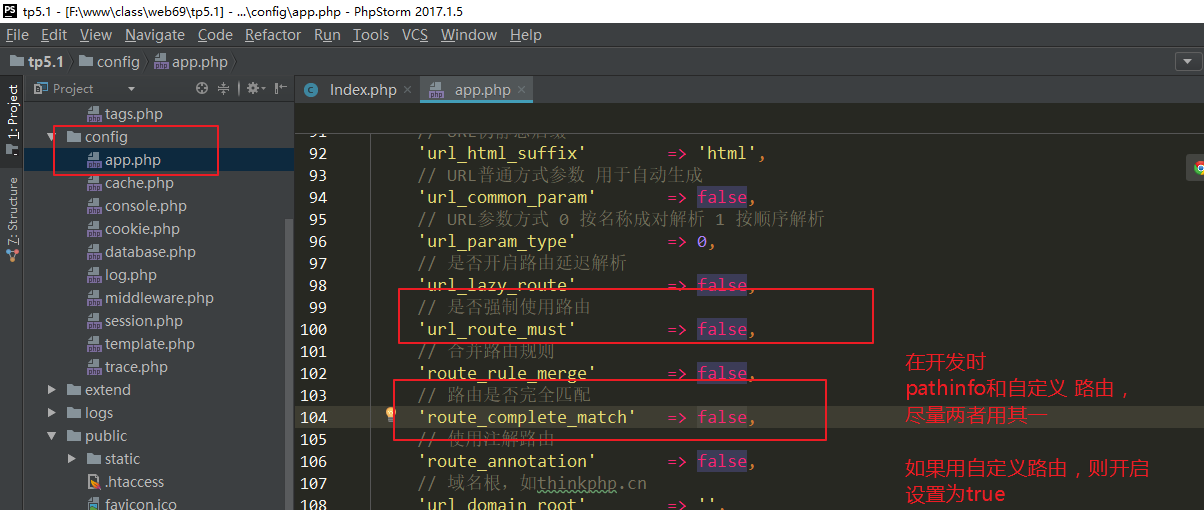
对于路由规则较多的应用可以大幅提升路由性能（仅部署模式有效）

'route\_check\_cache' => true,

注：如果路由定义中，有某个路由规则的路由地址使用了闭包的方式，那么路由缓存将会失效。

**完全匹配**

'route\_complete\_match' => true,



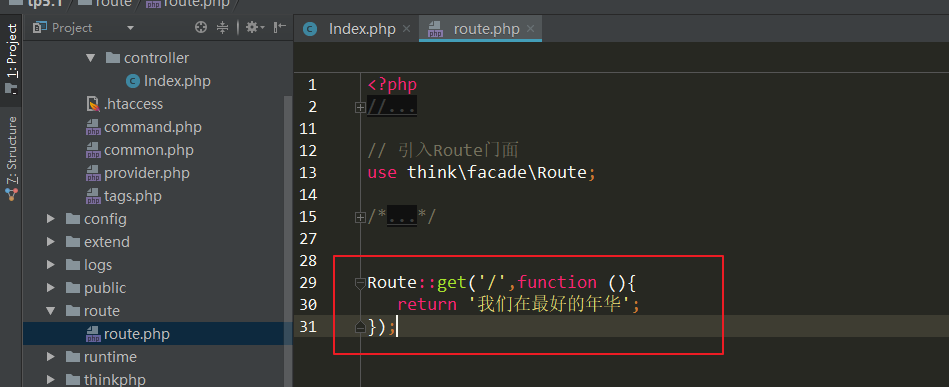
## 5.2、定义路由

路由文件的位置:项目目录/routes/route.php文件中进行如下格式的定义

Route::请求方式('路由表达式',匿名函数);

Route::请求方式('路由表达式','[@]模块名/控制器名/方法名'); # 后讲到控制器时使用

注：给控制器方法设置好了请求的路由规则后，原来的pathinfo请求则失效，请求就会报异常，只能通过自定义路由规则来请求



## 5.3、请求类型

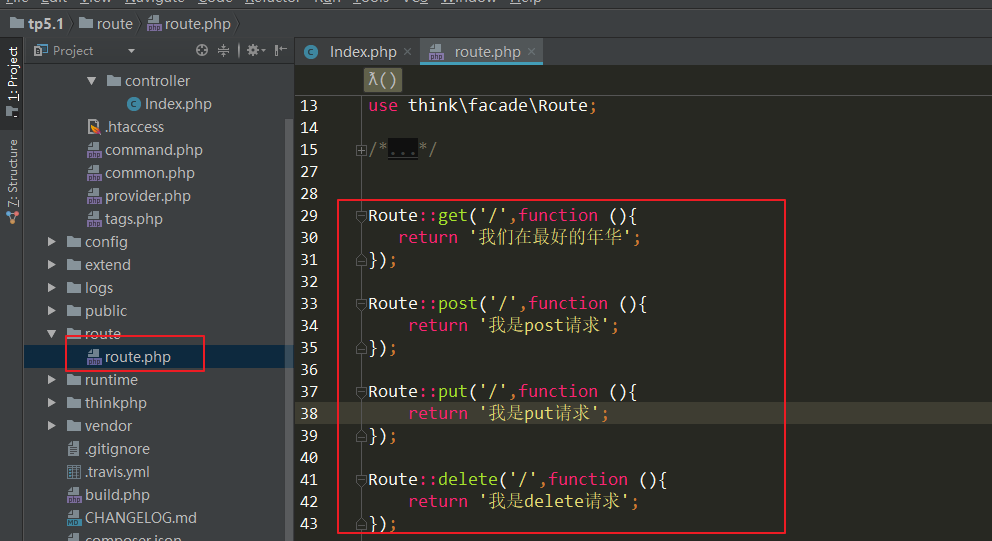
Route::get('new/:id','News/read'); // 定义GET请求路由规则 查询

Route::post('new/:id','News/update'); // 定义POST请求路由规则 添加

Route::put('new/:id','News/update'); // 定义PUT请求路由规则 修改

Route::delete('new/:id','News/delete'); // 定义DELETE请求路由规则 删除

Route::any('new/:id','News/read'); // 所有请求都支持的路由规则 框架提供 不推荐



测试定义好的请求路由规则类型，使用postman软件来测试



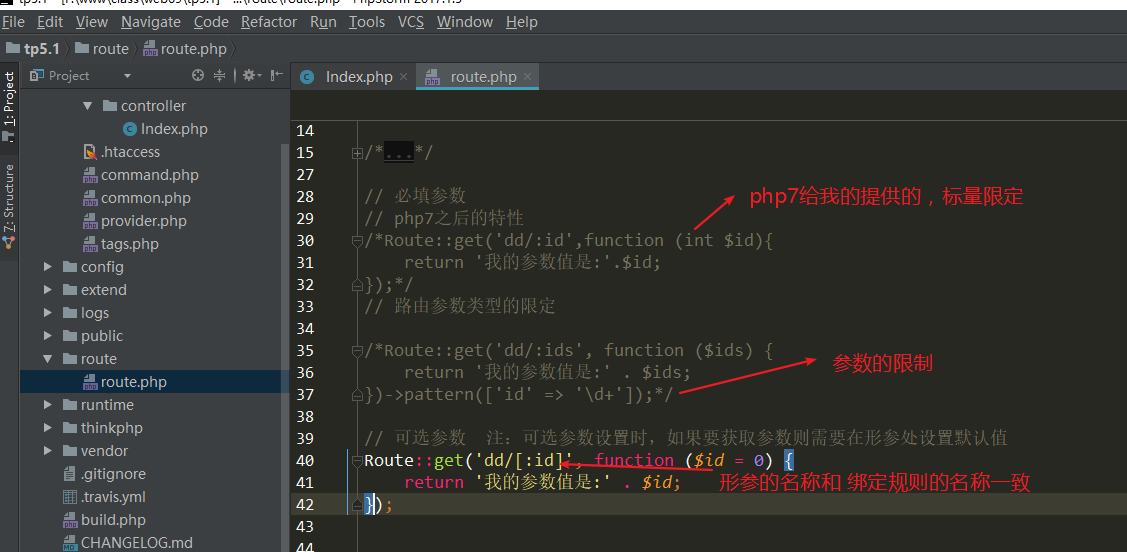
## 5.4、路由参数

# 必填参数

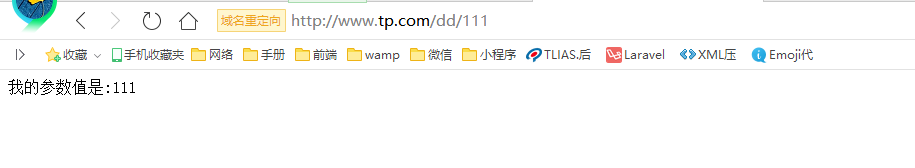
Route::请求方式('路由表达式/:参数',匿名函数);

# 可选参数

Route::请求方式('路由表达式/[:参数]',匿名函数);



效果



## 5.5、路由分组

路由分组功能允许把相同前缀的路由定义合并分组，这样可以简化路由定义，并且提高路由匹配的效率。

Route::group('分组名（字符串）或者分组路由参数（数组）','分组路由规则（数组或者闭包）');

Route::group('blog', function () {

Route::get(':id', 'read');

Route::post(':id', 'update');

Route::delete(':id', 'delete');

})->pattern(['id' => '\d+']);

# 分组的嵌套

Route::group(['method'=>'get'], function () {

Route::group('blog',function(){

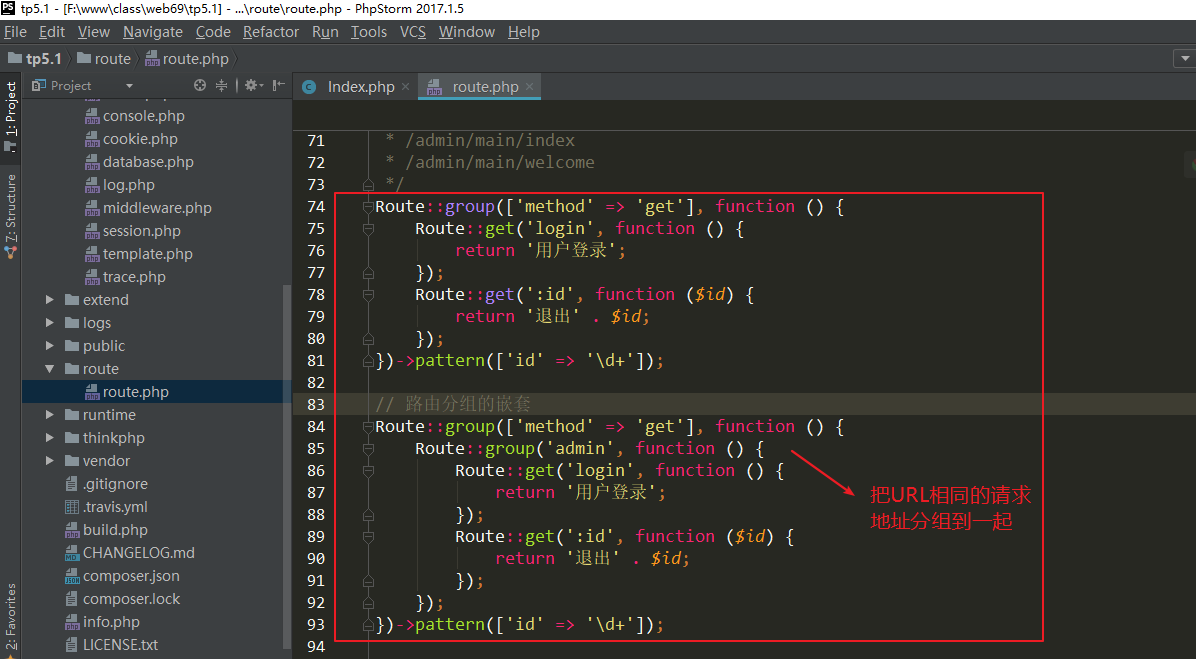
Route::get(':id', 'read');

Route::post(':id', 'update');

Route::delete(':id', 'delete');

});

})->pattern(['id' => '\d+']);





# 六、控制器

## 6.1、控制器的后缀

打开配置文件application/app.php，有如下配置

'controller\_suffix' => false,

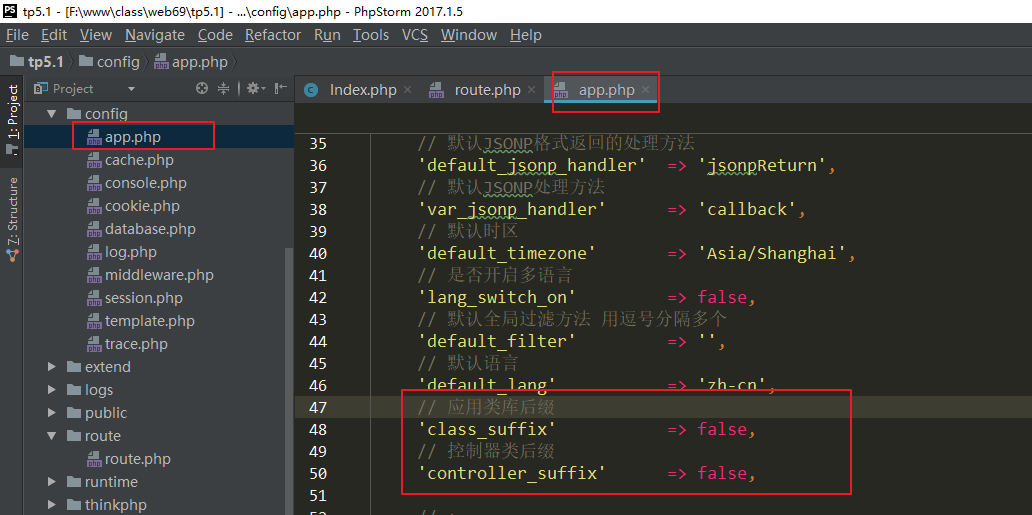
# 表示默认情况下，控制器无特殊后缀。例如 Index控制器，文件名为Index.php

如果需要进行设置，可以设置为

'controller\_suffix' => 'Controller',

# 表示控制器以Controller为后缀。例如Index控制器，文件名为IndexController.php

注：一般不建议进行修改，保持默认，框架默认给我们的配置。



## 6.2、使用命令行创建分组

php think build --module 分组名称



## 6.3、创建控制器

创建控制器的方式有两种：

* 手动创建

application/模块目录/controller/目录下

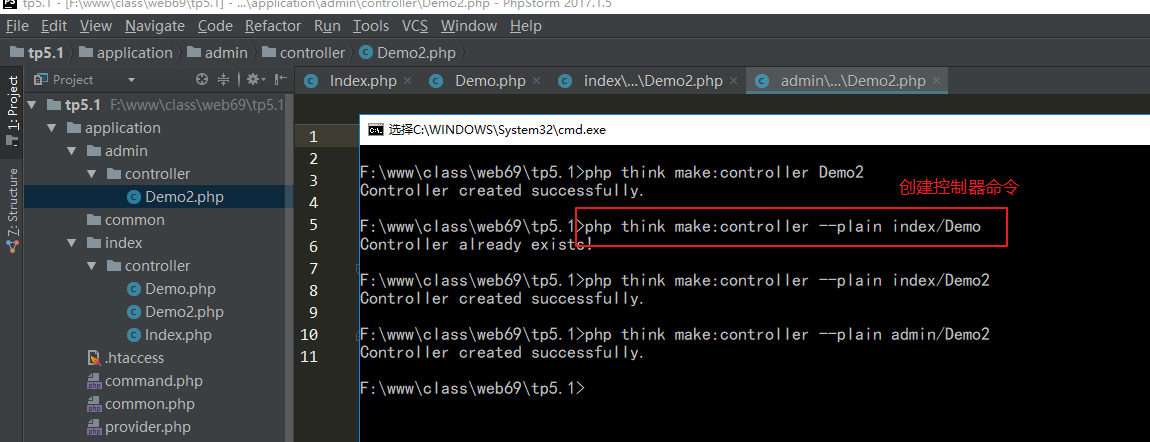
命名规则：控制器名称（首字母大写） + (控制器后缀，默认没有) + .php

* 命令行方式创建【推荐】

php think make:controller --plain 模块名/控制器名

# 参数说明

--plain 标准控制器 (默认创建的控制器是一个资源控制器，所以一般加上此选项)



如果创建时，没有在模块名称，则默认创建到公共模块中 【common】

## 6.4、开启调试模式

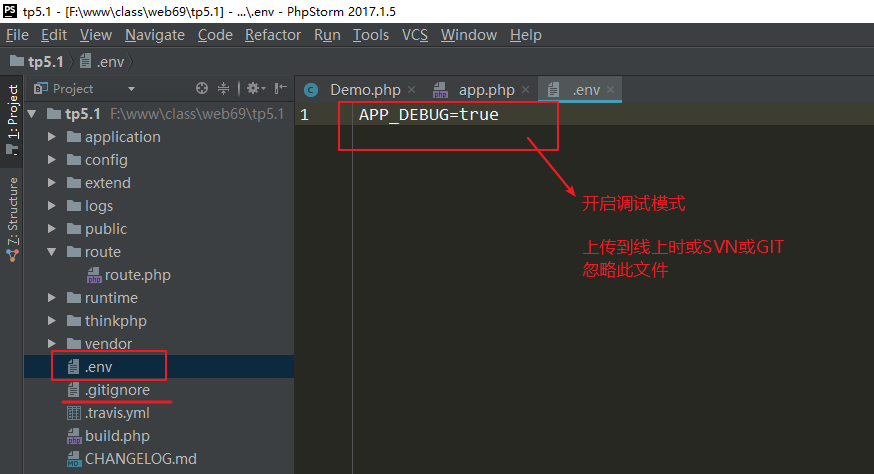
默认情况下，错误描述比较模糊，不方便进行错误调试。这种模式通常叫做“部署模式”。

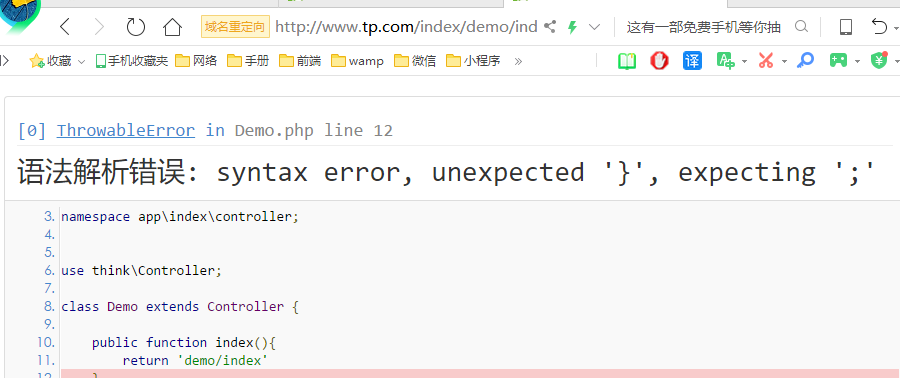
开发阶段可以将框架设置为调试模式，便于进行错误调试：

修改项目目录 config/app.php文件



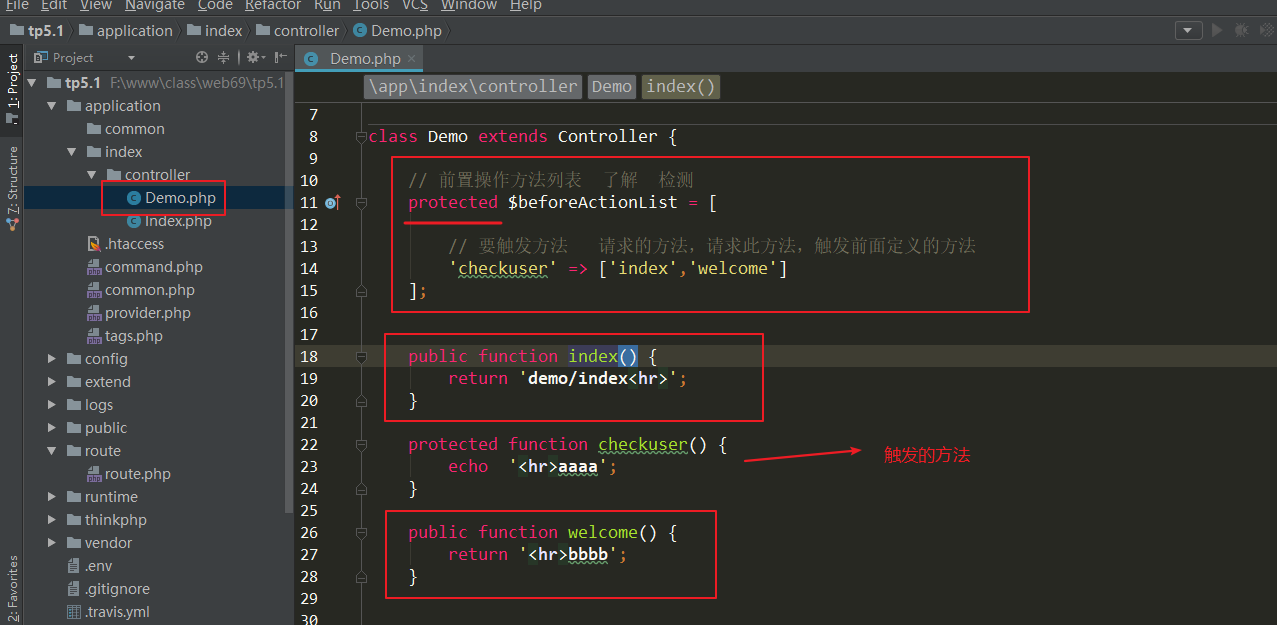
实际开发中，使用第2种方案





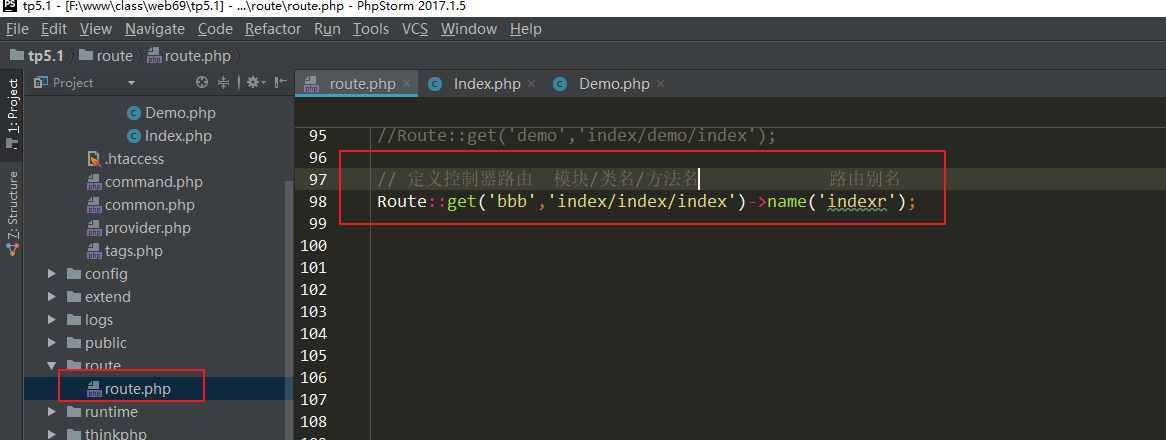
## 6.5、前置操作

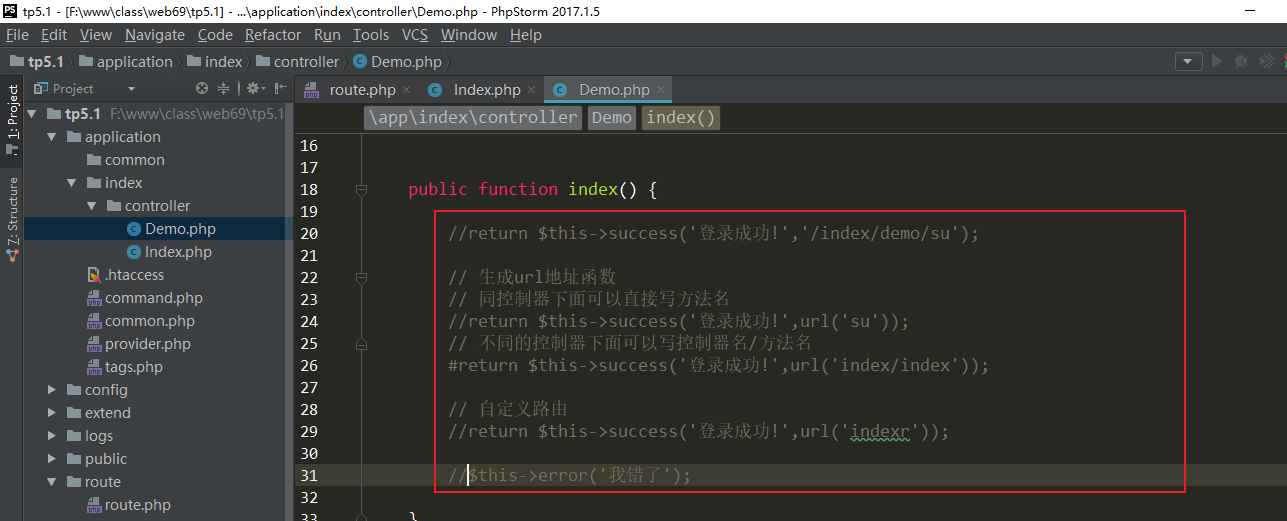
可以为某个或者某些操作指定前置执行的操作方法，设置 beforeActionList属性可以指定某个方法为其他方法的前置操作，数组键名为需要调用的前置方法名，无值的话为当前控制器下所有方法的前置方法。



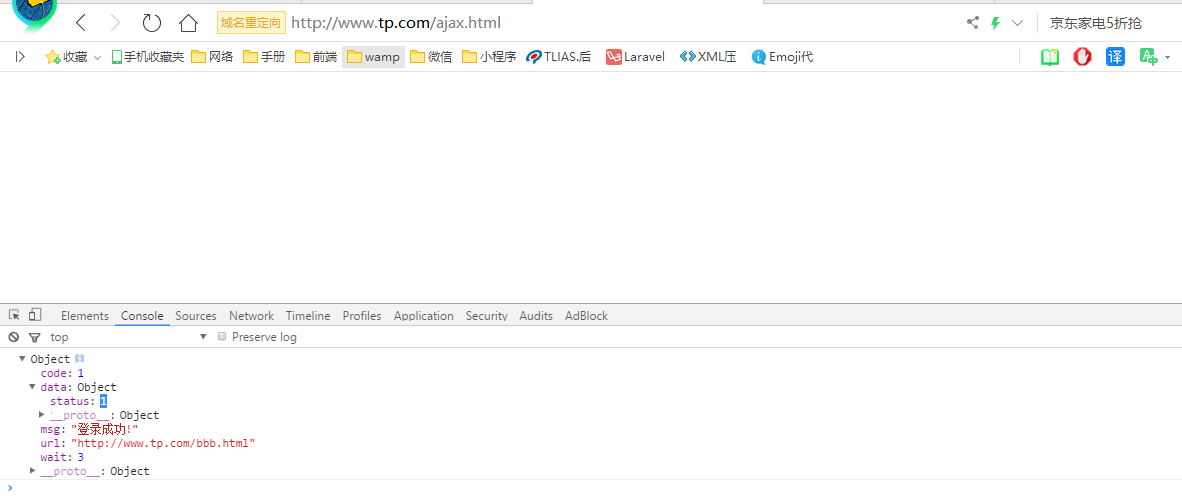
## 6.6、页面跳转

在应用开发中，经常会遇到一些带有提示信息的跳转页面，例如操作成功或者操作错误页面，并且自动跳转到另外一个目标页面。系统的\think\Controller类内置了两个跳转方法success和error，用于页面跳转提示。





此方法可根据不同的请求方式http标准请求还是ajax请求会自动返回数据(html/json)



# 七、请求

当浏览器向Web服务器发出请求时，它向服务器传递了一个数据块，也就是请求信息。在Thinkphp5.1中，获取请求对象数据，是由**think\Request**类负责，在很多场合下并不需要实例化调用，通常使用依赖注入即可，在其它场合（例如模板输出等）则可以使用think\facade\Request静态类操作。

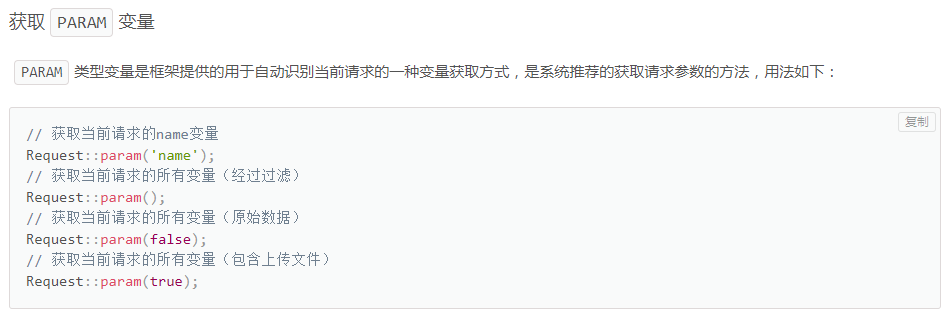
Request::param('name');

$request->param('name');

## 7.1、变量获取

变量类型方法('变量名/变量修饰符','默认值','过滤方法')





## 7.2、参数绑定

参数绑定是把当前请求的路由参数作为操作方法的参数直接传入，参数绑定并不区分请求类型。



## 7.3、依赖注入

依赖注入是一种软件设计思想，在传统软件中，上层代码依赖于下层代码，当下层代码有所改动时，上层代码也要相应进行改动，因此维护成本较高。而依赖注入原则的思想是，上层不应该依赖下层，应依赖接口。意为上层代码定义接口，下层代码实现该接口，从而使得下层依赖于上层接口，降低耦合度，提高系统弹性。

# 八、响应

## 8.1、输出响应

return 'PHP是世界上最好的语言';

## 8.2、json数据返回

由于默认是输出Html输出，所以直接以html页面方式输出响应内容，但也可以设置默认输出为json格式

'default\_return\_type' => 'json',

返回json数据

return json($data,http状态码);

return json($data)->code(201)->header(['Content-Type' => 'application/json']);

## 8.3、重定向

可以使用redirect助手函数进行重定向

redirect('地址或方法名',数组参数);

# 九、视图

## 9.1、视图的组成

此处所说的视图就是MVC中所说的V层，视图层也要展示层。thinkphp中视图层是由HTML模板文件组成的。

## 9.2、模板的定义

为了对模板文件更加有效的管理，ThinkPHP对模板文件进行目录划分，默认的模板文件定义规则是：视图目录/控制器名（小写）/操作名（小写）+ 模板后缀(框架的默认视图文件后缀是.html)

## 9.3、模板渲染和赋值

控制器中指定渲染的模板文件

在控制器中使用

方法一：

return $this->fetch('[模板文件]'[,'模板变量（数组）']);

方法二：

return view('[模板文件]'[,'模板变量（数组）']); # 辅助函数

模板赋值

方法一：

$this->assign('name','ThinkPHP');

方法二：

以关联数组的方式在渲染模板方法第二个参数填写

**全局赋值**

use think\facade\View;

# 赋值全局模板变量

View::share('name','value');

# 或者批量赋值

View::share(['name1'=>'value','name2'=>'value2']);

## 9.4、使用函数

需要对模板输出使用函数进行过滤或其它处理的时候，就可以使用一下函数来解决。

{$password|md5}

当然也可以写成如下的写法，推荐

{:md5($password)}

## 9.5、原样输出

因thinkphp5的标签语法和js的一致，所以有的时候，可能会使用到它。

{literal}

Hello,{$name}！

{/literal}

上面的{$name}标签被literal标签包含，因此并不会被模板引擎解析，而是保持原样输出。

## 9.6、循环标签【重点】

# foreach 【推荐写法】

{foreach $list as $key=>$vo }

{$vo.id}:{$vo.name}

{/foreach}

# volist

{volist name="list" id="vo"}

{$vo.id}:{$vo.name}<br/>

{/volist}

## 9.7、条件判断

{if ( $name == 1) OR ( $name > 100) } value1

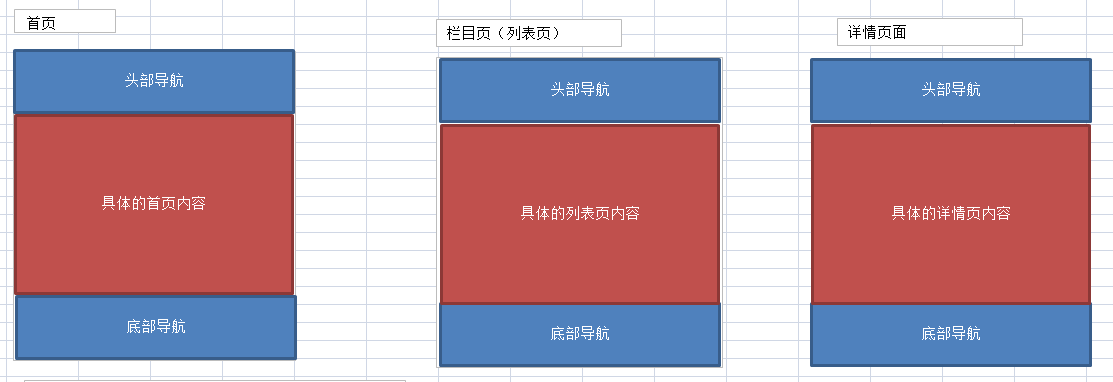
{elseif $name == 2 /} value2

{else /} value3

{/if}

## 9.8、模板继承

模板继承其实并不难理解，就好比类的继承一样，模板也可以定义一个基础模板（或者是布局），并且其中定义相关的区块（block），然后继承（extend）该基础模板的子模板中就可以对基础模板中定义的区块进行重载。



**定义一个基础模板**



**在子模板（其实是当前操作的入口模板）中使用继承**

