

# ThinkPHP6.0

辰风沐阳



# 目 录

序言

术语

模型关联

应用目录

应用根目录

pathinfo方式路由

安装

创建项目

nginx伪静态

阿里云镜像

composer.phar

测试运行 ThinkPHP6.0

PhpStudy-v8.1 运行TP6.0

开启伪静态

PhpStudy2018 运行 TP6.0

配置虚拟域名

基础

单应用模式

多应用模式

多应用模式扩展

配置

database.php

filesystem.php

view.php

route.php

配置目录

自定义配置文件

路由

路由简介

注册路由

路由别名

多应用路由

域名绑定路由

中间件

中间件简介

中间件文件

中间件定义

中间件别名

注册中间件

全局中间件  
应用中间件  
路由中间件  
控制器中间件

## 控制器

### 基础控制器

初始化方法  
注入的对象  
数据验证功能

### 创建控制器

### 控制器命名

### 控制器目录

### 控制器定义

### 控制器后缀

### 多级控制器

### 渲染输出

## 请求

### 请求对象

### 请求信息

### 请求类型

## 数据库

### 连接数据库

### 指定数据表

### 查询

find  
select  
paginate

### 更新

save  
update

### 链式操作

field  
join  
where  
whereIn

字符串条件  
查询表达式

fetchSql

### 事务操作

使用示例

### 聚合查询

count

## 分页查询

- [分页参数](#)
- [分页样式](#)
- [分页样式代码](#)
- [自定义分页驱动](#)
- [默认分页代码](#)

## 模型

- [定义](#)
- [新增](#)
- [简介](#)
- [save](#)
- [获取自增ID](#)
- [create](#)
- [删除](#)
- [简介](#)
- [delete](#)
- [destroy](#)

## 更新

- [静态方法直接更新](#)

## 查询

- [find](#)
- [获取器](#)
  - [应用场景](#)
  - [获取器定义](#)
  - [触发获取器](#)
  - [追加获取器](#)
  - [获取原始数据](#)
    - [getData](#)
  - [应用示例](#)

## 修改器

- [触发修改器](#)

## 搜索器

- [应用场景](#)
- [搜索器定义](#)
- [使用示例](#)
- [withSearch](#)

## 只读字段

- [只读字段简介](#)

## 模型关联

- [一对关联](#)
  - [数据表](#)
  - [基础使用](#)

- 关联查询
  - hasOne
  - hasWhere
  - belongsTo
- 关联保存更新
- 预载入查询
  - with
  - withJoin
- 绑定属性到父模型
  - bind
  - 属性别名
  - bindAttr
- 应用示例
- 一对多关联
  - 基础使用
  - 关联定义
  - 关联查询
    - 基本查询
    - 关联条件查询
- 远程一对多关联
  - 简介
  - 数据表
  - hasManyThrough
- 关联预载入
  - 简介
  - 数据表
- 关联统计
  - withCount
- 关联输出
  - hidden
  - visible
  - append
- 关联删除
- 自动时间戳
  - 配置
  - 触发
  - 写入格式
  - 自定义字段
- 模型输出
  - 模板输出
  - 数组转换
- 模型事件

[模型属性](#)

[视图](#)

[视图扩展](#)

[模板赋值](#)

[视图过滤](#)

[视图目录](#)

[模板渲染](#)

[条件判断标签](#)

[模板输出使用函数](#)

[调试](#)

[调试模式](#)

[变量调试](#)

[验证](#)

[生成验证器类](#)

[验证器类成员](#)

[验证规则](#)

[错误信息](#)

[验证场景](#)

[应用示例](#)

[内置验证规则](#)

[require](#)

[file](#)

[执行数据验证](#)

[杂项](#)

[Session](#)

[默认状态](#)

[开启Session](#)

[Session初始化](#)

[Session基础用法](#)

[Cookie](#)

[基本使用](#)

[文件上传](#)

[上传简介](#)

[上传规则](#)

[上传对象](#)

[方法描述](#)

[上传验证](#)

[多文件上传](#)

[获取磁盘配置](#)

[获取默认磁盘名称](#)

[自定义命名规则](#)

[layui](#)

文件上传

layuiAdmin单页版

部署方案

接口数据

隐藏 trace

数据表格中的删除

修改数据

layuiAdmin iframe版

iframe版部署

放入TP6.0视图

登陆页面验证码

引入authtree扩展组件

2020.1.24版 主页成背景

经验分享

章节停更说明

PhpStudy-v8.1 无法解析PHP代码

屏蔽Sublime的提示

正则匹配img标签

Sublime 3.x 激活码

windows dos 窗口命令 type

windows dos 窗口命令 start

apache配置主机

navicat 闲置时间过久会卡死

命令行

简介

生成控制器

生成模型类

生成验证器

生成中间件

查看框架版本

清除缓存文件

生成应用目录

模板引擎

运算符

变量输出

输出替换

模板引擎

模板注释

模板分离

模板继承

基础模板

区块设计

[内置标签](#)

[volist](#)

[资源文件](#)

[助手函数](#)

[app](#)

[url](#)

[input](#)

[redirect](#)

[validate](#)

[validate\(\) 简介](#)

[文件上传验证](#)

[传入验证器类名](#)

[传入验证规则数组](#)

[cookie](#)

[常用功能](#)

[无限极分类](#)

[继承基础控制器的登陆验证](#)

[常用扩展](#)

[flc/dysms](#)

[qiniu/php-sdk](#)

[简介](#)

[上传示例](#)

[firebase/php-jwt](#)

[symfony/var-dumper](#)

[phpmailer/phpmailer](#)

[liliuwei/thinkphp-jump](#)

[扩展介绍](#)

[下载扩展](#)

[使用方法](#)

[topthink/think-captcha](#)

[安装扩展](#)

[验证码显示](#)

[更换验证码](#)

[验证码校验](#)

[验证码配置](#)

[自定义验证码](#)

[phpoffice/phpspreadsheet](#)

[数据写入表格](#)

[读取表格数据](#)

[更新日志](#)

[2020-06](#)

[2020-05](#)

2020-04

2020-03

2020-02

2020-01

2019-12

# 序言

## 关于文档

- 本文档基于 ThinkPHP6.0.\*
- 个人编写 凭自己的理解而编写的文档
- 文档免费, ThinkPHP 技术群付费入群
- 仅供参考 , 请以 [ThinkPHP6.0官方手册](#) 为准
- 文档变动内容会记录在 [文档更新日志](#) 章节

## ThinkPHP 官方手册

thinkphp 6.0 框架官方手册

[https://www.kancloud.cn/manual/thinkphp6\\_0](https://www.kancloud.cn/manual/thinkphp6_0)

think-orm 扩展 , 数据库扩展官方手册

<https://www.kancloud.cn/manual/think-orm>

think-template 扩展 , 模板引擎官方手册

<https://www.kancloud.cn/manual/think-template>

## ThinkPHP 常用扩展

- 下载框架

```
composer create-project topthink/think=6.0.* tp6
```

- 视图扩展

```
composer require topthink/think-view
```

- 验证码扩展

```
composer require topthink/think-captcha
```

- 多应用模式扩展

```
composer require topthink/think-multi-app
```

# 术语

---

[模型关联](#)

[应用目录](#)

[应用根目录](#)

[pathinfo方式路由](#)

# 模型关联

```
/**
 * profile 关联方法名
 */
public function profile()
{
    // \app\model\AdminInfo::class 关联模型类名
    // 'aid' 外键字段名, 如:用户资料表的aid字段
    // 'tid' 主键字段名, 如:用户表的tid字段
    return $this->hasOne(\app\model\AdminInfo::class, 'aid', 'tid');
}
```

# 应用目录

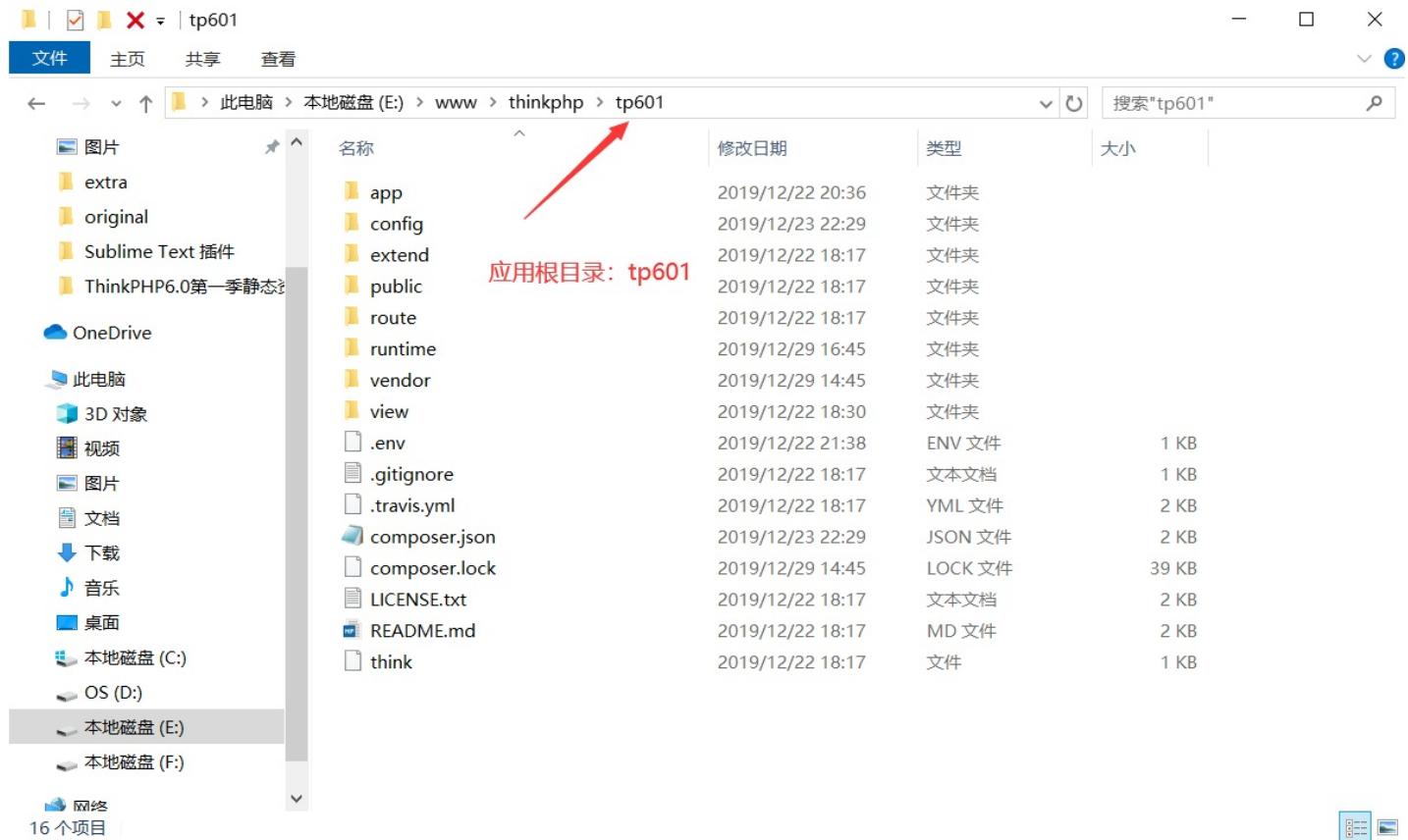
## 应用目录

- 应用目录：app目录
- 因为所有的应用都在该目录下，所以才叫应用目录

# 应用根目录

## 应用根目录

应用根目录指的就是框架根目录，示例如下：



## 在控制器中你可以使用以下方法获取应用根目录路径

```
// 示例：E:\www\thinkphp\tp6.0
$root = app()->getRootPath();
```

# pathinfo方式路由

## pathinfo 方式路由

pathinfo：即全路径的访问控制器方法，如：域名/应用名/控制器/操作名

## pathinfo 方式路由示例

```
// index应用下index控制器index操作方法  
tp6.com/index/index/index
```

# 安装

---

[创建项目](#)

[nginx伪静态](#)

[阿里云镜像](#)

[composer.phar](#)

[测试运行 ThinkPHP6.0](#)

[PhpStudy-v8.1 运行TP6.0](#)

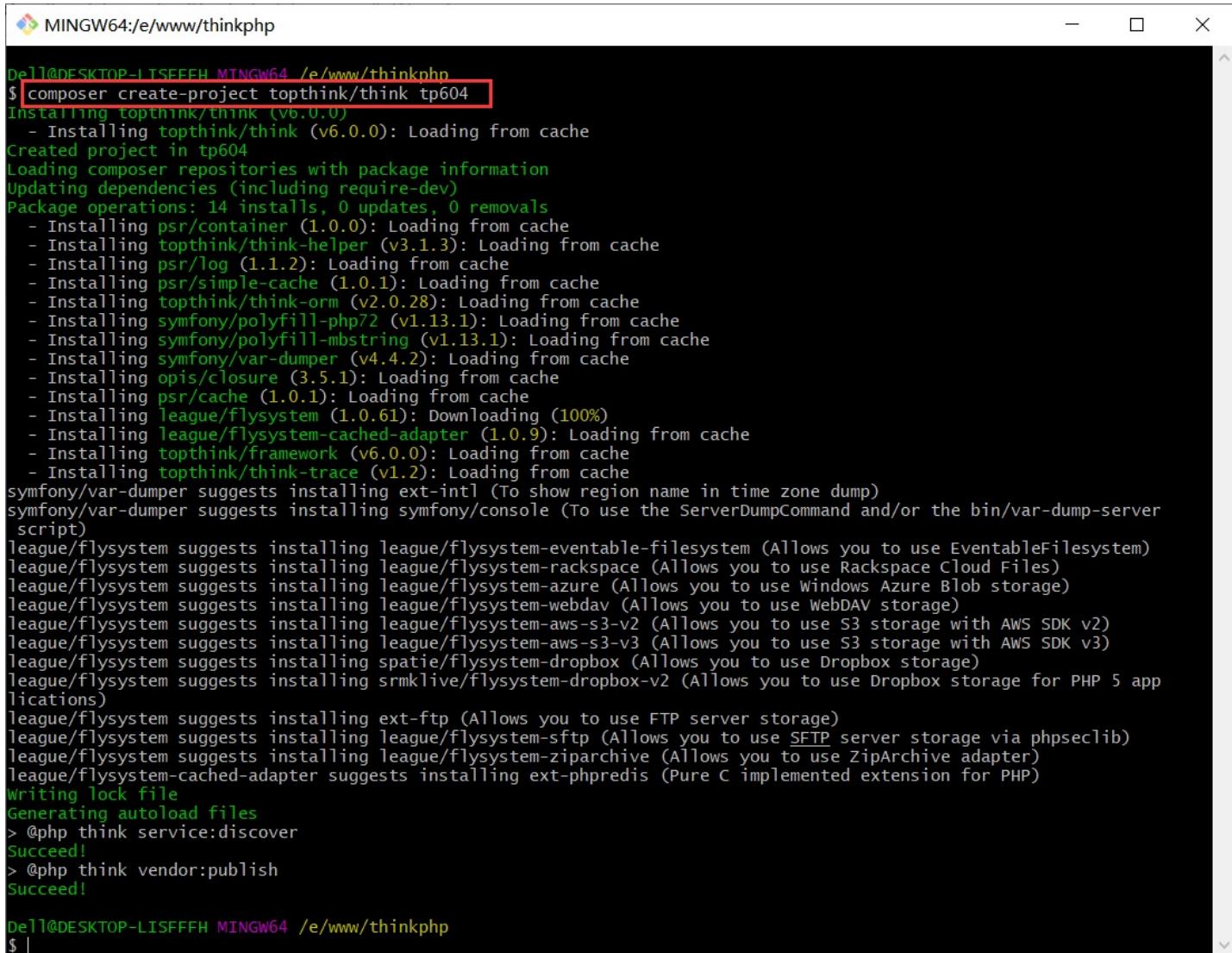
[PhpStudy2018 运行 TP6.0](#)

# 创建项目

## 通过 composer 下载 ThinkPHP6.0.x 最新稳定版

- 默认目录名：项目名称
- `topthink/think` 中 `topthink` 是供应商名称，`think` 是项目名称

```
composer create-project topthink/think=6.0.* [目录名=think]
```



```
Dell@DESKTOP-LISFFFH MINGW64 /e/www/thinkphp
$ composer create-project topthink/think tp604
Installing topthink/think (v6.0.0)
- Installing topthink/think (v6.0.0): Loading from cache
Created project in tp604
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 14 installs, 0 updates, 0 removals
- Installing psr/container (1.0.0): Loading from cache
- Installing topthink/think-helper (v3.1.3): Loading from cache
- Installing psr/log (1.1.2): Loading from cache
- Installing psr/simple-cache (1.0.1): Loading from cache
- Installing topthink/think-orm (v2.0.28): Loading from cache
- Installing symfony/polyfill-php72 (v1.13.1): Loading from cache
- Installing symfony/polyfill-mbstring (v1.13.1): Loading from cache
- Installing symfony/var-dumper (v4.4.2): Loading from cache
- Installing opis/closure (3.5.1): Loading from cache
- Installing psr/cache (1.0.1): Loading from cache
- Installing league/flysystem (1.0.61): Downloading (100%)
- Installing League/flysystem-cached-adapter (1.0.9): Loading from cache
- Installing topthink/framework (v6.0.0): Loading from cache
- Installing topthink/think-trace (v1.2): Loading from cache
symfony/var-dumper suggests installing ext-intl (To show region name in time zone dump)
symfony/var-dumper suggests installing symfony/console (To use the ServerDumpCommand and/or the bin/var-dump-server script)
league/flysystem suggests installing league/flysystem-eventable-fs (Allows you to use EventableFilesystem)
league/flysystem suggests installing league/flysystem-rackspace (Allows you to use Rackspace Cloud Files)
league/flysystem suggests installing league/flysystem-azure (Allows you to use Windows Azure Blob storage)
league/flysystem suggests installing league/flysystem-webdav (Allows you to use WebDAV storage)
league/flysystem suggests installing league/flysystem-aws-s3-v2 (Allows you to use S3 storage with AWS SDK v2)
league/flysystem suggests installing league/flysystem-aws-s3-v3 (Allows you to use S3 storage with AWS SDK v3)
league/flysystem suggests installing spatie/flysystem-dropbox (Allows you to use Dropbox storage)
league/flysystem suggests installing srmklive/flysystem-dropbox-v2 (Allows you to use Dropbox storage for PHP 5 applications)
league/flysystem suggests installing ext-ftp (Allows you to use FTP server storage)
league/flysystem suggests installing league/flysystem-sftp (Allows you to use SFTP server storage via phpseclib)
league/flysystem suggests installing league/flysystem-ziparchive (Allows you to use ZipArchive adapter)
league/flysystem-cached-adapter suggests installing ext-phppredis (Pure C implemented extension for PHP)
Writing lock file
Generating autoload files
> @php think service:discover
Succeed!
> @php think vendor:publish
Succeed!
```

Dell@DESKTOP-LISFFFH MINGW64 /e/www/thinkphp
\$ |

## 补充说明

- 以上命令中的 `6.0.*` 代表下载 `6.0.x` 版本中的最新稳定版
- 若想要下载指定版本，指定具体的版本号即可，如：`topthink/think=6.0.1`
- 不指定目录名时，默认目录名为 `think`，若目录存在，保证目录中无任何内容，也就是空目录，否则将无法下载，如下图所示

```
Dell@DESKTOP-LISFFFH MINGW64 /f/download/asaf
$ composer create-project topthink/think=5.1.* 02
Installing topthink/think (v5.1.39)

[InvalidArgumentException]
Project directory 02/ is not empty.

create-project [-s|--stability STABILITY] [--prefer-source] [--prefer-dist] [--repository REPOSITORY] [--url REPOSITORY-URL] [--dev] [--no-dev] [--no-custom-installers] [--no-scripts] [--no-progress] [--keep-vcs] [--remove-vcs] [--no-install] [--ignore-platform-reqs] [--] [<package>] [<directory>]
```

```
Dell@DESKTOP-LISFFFH MINGW64 /f/download/asaf
$ ll
total 8
drwxr-xr-x 1 Dell 197121 0 1月 3 21:57 01/
drwxr-xr-x 1 Dell 197121 0 1月 3 21:58 02/
```

### 删除框架附带的无用文件

README.md 文件仅用于说明，实际部署的时候可以删除

# nginx伪静态

## 快速使用

```
if (!-e $request_filename) {  
    rewrite ^(.*)$ /index.php?s=$1 last;  
    break;  
}
```

## nginx 伪静态配置

```
server {  
    listen 80;  
    server_name all.bjed.com;  
    root "F:\www\asdata";  
    location / {  
        index index.html index.htm index.php;  
        #autoindex on;  
  
        # 新增内容开始  
        if (!-e $request_filename) {  
            rewrite ^(.*)$ /index.php?s=$1 last;  
            break;  
        }  
        # 新增内容结束  
    }  
}
```

# 阿里云镜像

## 个人推荐的镜像

阿里云镜像传送门

<https://developer.aliyun.com/composer>

阿里云镜像的使用

- 配置全局阿里云镜像

```
composer config -g repo.packagist composer https://mirrors.aliyun.com/composer/
```

- 取消全局镜像配置

```
composer config -g --unset repo.packagist
```

查看 composer 全局配置中的镜像是否已经成功修改

```
composer config -gl
```

```
MINGW64:/f/download/asaf
[data-dir] C:/Users/Dell/AppData/Roaming/Composer
$ composer config -g1
[repositories.packagist.org.type] composer
[repositories.packagist.org.url] https://mirrors.aliyun.com/composer/
[process-timeout] 300
[use-include-path] false
[preferred-install] auto
[notify-on-install] true
[github-protocols] [https, ssh]
[vendor-dir] vendor (F:\download\asaf\vendor)
[bin-dir] {$vendor-dir}/bin (F:\download\asaf\vendor/bin)
[cache-dir] C:/Users/Dell/AppData/Local/Composer
[data-dir] C:/Users/Dell/AppData/Roaming/Composer
[cache-files-dir] {$cache-dir}/files (C:/Users/Dell/AppData/Local/Composer/files)
[cache-repo-dir] {$cache-dir}/repo (C:/Users/Dell/AppData/Local/Composer/repo)
[cache-vcs-dir] {$cache-dir}/vcs (C:/Users/Dell/AppData/Local/Composer/vcs)
[cache-ttl] 15552000
[cache-files-ttl] 15552000
[cache-files-maxsize] 300Mib (314572800)
[bin-compat] auto
[discard-changes] false
[autoloader-suffix]
[sort-packages] false
[optimize-autoloader] false
[classmap-authoritative] false
[apcu-autoloader] false
[prepend-autoloader] true
[github-domains] [github.com]
[bitbucket-expose-hostname] true
[disable-tls] false
[secure-http] true
[cafile]
[capath]
```

# composer.phar

从 ThinkPHP6.0 开始只支持通过Composer下载，  
所以想要自己下载TP6必须先准备好Composer的环境  
本章节介绍如何通过 composer.phar 下载 thinkphp6.0

## Windows安装 composer 的两种方式

- 方法1：下载 [Composer-Setup.exe](#) 安装程序
- 方法2：下载 [composer.phar](#) (本章节介绍的用法)

## 下载 composer.phar 最新的稳定版

Composer commit, you can use the `--snapshot` flag.

### Manual Download

If you prefer to download the phar manually, here are the available versions:

Version	Release Date	SHA-256	Download Link
1.9.1	2019-11-01	<a href="#">sha256</a>	<a href="#">1f210b9037fc82670d75892dfc44400f13fe9ada7af9e787f93e50e3b764111</a>
1.9.0	2019-08-02	<a href="#">sha256</a>	<a href="#">c9dff69d092bdec14dee64df6677e7430163509798895fb54891c166c5e0875</a>
1.8.6	2019-06-11	<a href="#">sha256</a>	<a href="#">b66fb53db72c5117408defe8a1e00515fe749e97ce1b0ae8bdaa6a5a43dd542</a>
1.8.5	2019-04-09	<a href="#">sha256</a>	<a href="#">4e4c1cd74b54a20618699f3190e6f5fc63bb308b13fa660f71f2a2df047c0e17</a>
1.8.4	2019-02-11	<a href="#">sha256</a>	<a href="#">1722826c8fbeaf2d8cd31c9af38694d6383a0f2bf476fe6bbd30939de058a</a>
1.8.3	2019-01-30	<a href="#">sha256</a>	<a href="#">5733ae9516e9185b7c3328d16dac75f3475f8ef137572dfb497f0f298157df33</a>
1.8.2	2019-01-29	<a href="#">sha256</a>	<a href="#">489025eb489a2a70b8cd7acd53f530e7be22a342b8c971a9d73091b898c7aed</a>
1.8.1	2019-01-29	<a href="#">sha256</a>	<a href="#">ded05cb32100648146a264eaf4b4919308b3f1fd8e38adde8406396860dfab</a>
1.8.0	2018-12-03	<a href="#">sha256</a>	<a href="#">0901a84d84d56f6d6ae8f8b96b0c131d4f51ccaf169d491813d2bcdf2a6e4cef6</a>
1.7.3	2018-11-01	<a href="#">sha256</a>	<a href="#">bc6cbcd2c0fbc03c7ab87442b5f1fbc9407f0b9900bddc10d755bcd81bbe7b6e</a>

<https://getcomposer.org/composer-stable.phar>

composer.phar 60.9 KB

## 前提：放在可以运行 php 命令的目录

运行 `php -v` 测试是否可以运行 `php` 命令

如何提示不是内部命令，说明没有配置PHP的环境变量

配置PHP环境变量：将PHP的安装目录路径添加到Path环境变量中

```
C:\Windows\System32\cmd.exe
C:\Users\De11\Desktop\新建文件夹>php -v
PHP 7.3.7 (cli) (built: Jul 3 2019 14:34:10) ( ZTS MSVC15 (Visual C++ 2017) x64 )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.3.7, Copyright (c) 1998-2018 Zend Technologies

C:\Users\De11\Desktop\新建文件夹>dir
驱动器 C 中的卷没有标签。
卷的序列号是 09C1-B27D

C:\Users\De11\Desktop\新建文件夹 的目录

2020/01/05 23:02 <DIR> .
2020/01/05 23:02 <DIR> ..
2020/01/05 22:58 1,933,813 composer.phar
               1 个文件      1,933,813 字节
               2 个目录 64,782,974,976 可用字节

C:\Users\De11\Desktop\新建文件夹>
```

## 下载 thinkphp6.0.x 最新稳定版

```
php composer.phar create-project tothink/think=6.0.*
```

```
C:\Windows\System32\cmd.exe
选择C:\Windows\System32\cmd.exe
1 个文件      1,933,813 字节
2 个目录 64,782,974,976 可用字节

C:\Users\De11\Desktop\新建文件夹>php composer.phar create-project tothink/think=6.0.0
Installing tothink/think (v6.0.0)
- Installing tothink/think (v6.0.0): Loading from cache
Created project in C:\Users\De11\Desktop\新建文件夹\think
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 14 installs, 0 updates, 0 removals
- Installing psr/container (1.0.0): Loading from cache
- Installing tothink/think-helper (v3.1.3): Loading from cache
- Installing psr/log (1.1.2): Loading from cache
- Installing psr/simple-cache (1.0.1): Loading from cache
- Installing tothink/think-orm (v2.0.30): Downloading (100%)
- Installing symfony/polyfill-php72 (v1.13.1): Loading from cache
- Installing symfony/polyfill-mbstring (v1.13.1): Loading from cache
- Installing symfony/var-dumper (v4.4.2): Loading from cache
- Installing opis/closure (3.5.1): Loading from cache
- Installing psr/cache (1.0.1): Loading from cache
- Installing league/flysystem (1.0.63): Downloading (100%)
- Installing league/flysystem-cached-adapter (1.0.9): Loading from cache
- Installing tothink/framework (v6.0.0): Loading from cache
- Installing tothink/think-trace (v1.2): Loading from cache
symfony/var-dumper suggests installing ext-intl (To show region name in time zone dump)
symfony/var-dumper suggests installing symfony/console (To use the ServerDumpCommand and/or the bin/var-dump-server script)
league/flysystem suggests installing league/flysystem-eventable-fs (Allows you to use EventableFilesystem)
league/flysystem suggests installing league/flysystem-rackspace (Allows you to use Rackspace Cloud Files)
league/flysystem suggests installing league/flysystem-azure (Allows you to use Windows Azure Blob storage)
league/flysystem suggests installing league/flysystem-webdav (Allows you to use WebDAV storage)
league/flysystem suggests installing league/flysystem-aws-s3-v2 (Allows you to use S3 storage with AWS SDK v2)
league/flysystem suggests installing league/flysystem-aws-s3-v3 (Allows you to use S3 storage with AWS SDK v3)
league/flysystem suggests installing spatie/flysystem-dropbox (Allows you to use Dropbox storage)
league/flysystem suggests installing srmklive/flysystem-dropbox-v2 (Allows you to use Dropbox storage for PHP 5 applications)
league/flysystem suggests installing ext-ftp (Allows you to use FTP server storage)
league/flysystem suggests installing league/flysystem-sftp (Allows you to use SFTP server storage via phpseclib)
league/flysystem suggests installing league/flysystem-ziparchive (Allows you to use ZipArchive adapter)
league/flysystem-cached-adapter suggests installing ext-phppredis (Pure C implemented extension for PHP)
Writing lock file
Generating autoload files
> @php think service:discover
Succeed!
> @php think vendor:publish
Succeed!
```

# 测试运行 ThinkPHP6.0

在应用根目录下执行命令

```
php think run
```

```
C:\Windows\System32\cmd.exe - php think run
C:\Users\DELL\Desktop\test>php think run
ThinkPHP Development server is started On <http://127.0.0.1:8000/>
You can exit with CTRL-C
Document root is: C:\Users\DELL\Desktop\test\think\public
[Sun Mar 15 15:54:25 2020] PHP 7.4.1 Development Server (http://0.0.0.0:8000) started
```

浏览器访问



指定端口号

```
php think run -p 8010
```

```
C:\Windows\System32\cmd.exe - php think run -p 8010
```

```
C:\Users\DELL\Desktop\test\think>php think run -p 8010
ThinkPHP Development server is started On <http://127.0.0.1:8010/>
You can exit with `CTRL-C`
Document root is: C:\Users\DELL\Desktop\test\think\public
[Sun Mar 15 15:56:15 2020] PHP 7.4.1 Development Server (http://0.0.0.0:8010) started
```

浏览器访问



退出测试运行

在命令行窗口按下 `ctrl + c`

```
[Sun Mar 15 15:56:38 2020] 127.0.0.1:53841 Closed without se  
preconnection  
[Sun Mar 15 15:56:38 2020] 127.0.0.1:53841 Closing  
C:\Users\DEll\Desktop\test\think>
```

# PhpStudy-v8.1 运行TP6.0

---

[开启伪静态](#)

# 开启伪静态

## 未开启伪静态



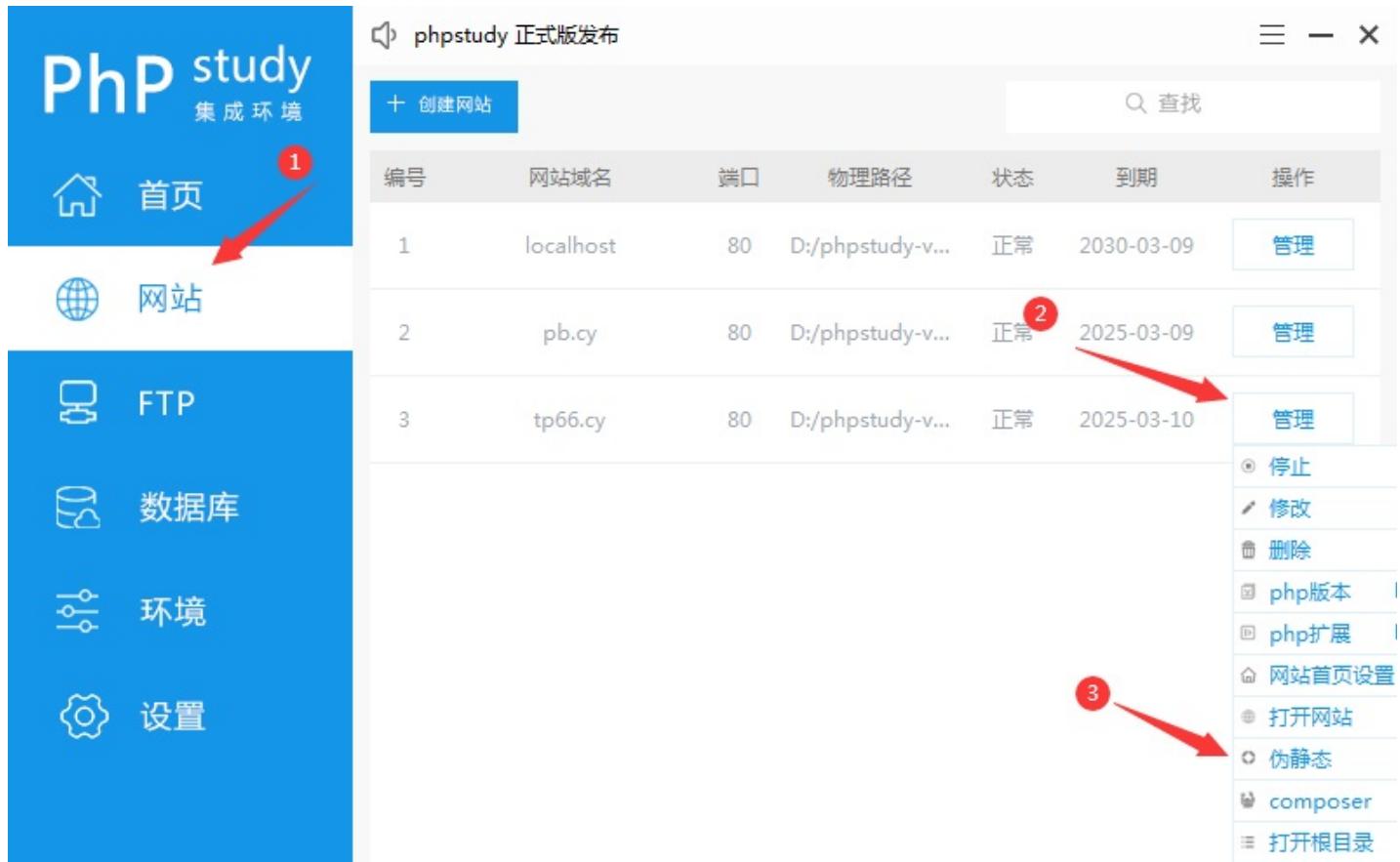
## 默认重写规则

```
<IfModule mod_rewrite.c>
    Options +FollowSymlinks -Multiviews
    RewriteEngine On

    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php/$1 [QSA, PT, L]
</IfModule>
```

## 修改重写规则

在 `index.php` 后面加上一个英文的问号



修改后的如下所示

The screenshot shows the PhP study integrated environment interface. On the left sidebar, there are icons for Home, Website, FTP, Database, Environment, and Settings. The main area displays a virtual host named 'localhost' on port 80, located at 'D:/phpstudy-v...'. A modal window titled '伪静态规则' (Rewrite Rules) is open, showing the following configuration:

```
<IfModule mod_rewrite.c>
    Options +FollowSymlinks -Multiviews
    RewriteEngine On

    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php?/$1 [QSA,PT,L]
</IfModule>
```

A red arrow points from the text 'index.php后面添加一个英文问号' (Add an English question mark after index.php) to the '\$1' placeholder in the RewriteRule line.

测试访问 正常运行 配置成功

:) 2020新春快乐

ThinkPHP V6.0.2

14载初心不改 - 你值得信赖的PHP框架

[ V6.0 版本由 [亿速云](#) 独家赞助发布 ]

[官方入门系列教程](#)

# PhpStudy2018 运行 TP6.0

---

[配置虚拟域名](#)

# 配置虚拟域名

使用 composer 下载 TP6.0.\* 的最新版

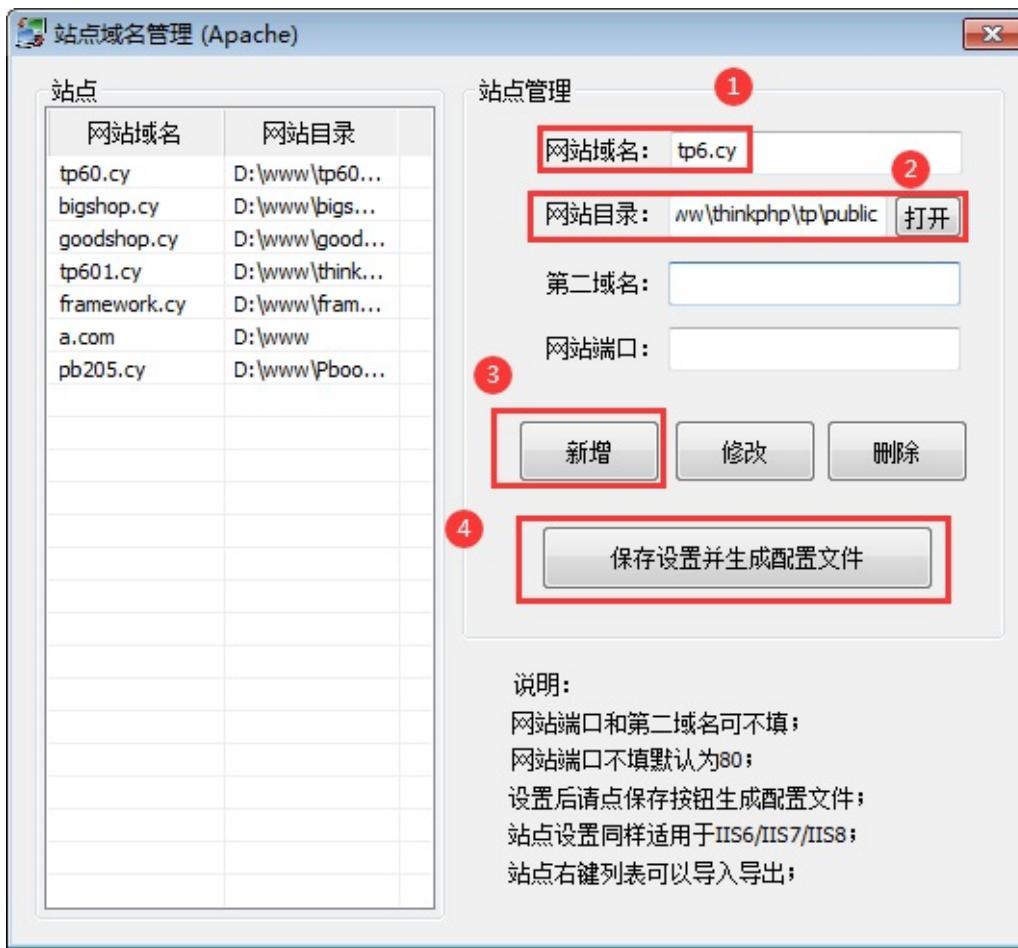
```
composer create-project topthink/think=6.0.* tp
```

```
D:\www\thinkphp>composer create-project topthink/think=6.0.* tp
Installing topthink/think (v6.0.2)
  - Installing topthink/think (v6.0.2): Loading from cache
Created project in tp
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 14 installs, 0 updates, 0 removals
  - Installing psr/container (1.0.0): Loading from cache
  - Installing topthink/think-helper (v3.1.3): Loading from cache
  - Installing psr/log (1.1.2): Loading from cache
  - Installing psr/simple-cache (1.0.1): Loading from cache
  - Installing topthink/think-orm (v2.0.31): Loading from cache
  - Installing symfony/polyfill-php72 (v1.14.0): Loading from cache
  - Installing symfony/polyfill-mbstring (v1.14.0): Loading from cache
  - Installing symfony/var-dumper (v4.4.5): Loading from cache
  - Installing opis/closure (3.5.1): Loading from cache
  - Installing psr/cache (1.0.1): Loading from cache
  - Installing league/flysystem (1.0.46): Loading from cache
  - Installing league/flysystem-cached-adapter (1.0.9): Loading from cache
  - Installing topthink/framework (v6.0.2): Loading from cache
  - Installing topthink/think-trace (v1.2): Loading from cache
symfony/var-dumper suggests installing ext-intl (To show region name in time zone dump)
symfony/var-dumper suggests installing symfony/console (To use the ServerDumpCommand and/or the bin/var-dump)
league/flysystem suggests installing ext-fileinfo (Required forMimeType)
league/flysystem suggests installing league/flysystem-eventable-fs (Allows you to use EventableFiles)
league/flysystem suggests installing league/flysystem-rackspace (Allows you to use Rackspace Cloud Files)
league/flysystem suggests installing league/flysystem-azure (Allows you to use Windows Azure Blob storage)
league/flysystem suggests installing league/flysystem-webdav (Allows you to use WebDAV storage)
league/flysystem suggests installing league/flysystem-aws-s3-v2 (Allows you to use S3 storage with AWS SDK v2)
league/flysystem suggests installing league/flysystem-aws-s3-v3 (Allows you to use S3 storage with AWS SDK v3)
league/flysystem suggests installing spatie/flysystem-dropbox (Allows you to use Dropbox storage)
league/flysystem suggests installing srmklive/flysystem-dropbox-v2 (Allows you to use Dropbox storage for PHP)
league/flysystem suggests installing ext-ftp (Allows you to use FTP server storage)
league/flysystem suggests installing league/flysystem-sftp (Allows you to use SFTP server storage via phpseclib)
league/flysystem suggests installing league/flysystem-ziparchive (Allows you to use ZipArchive adapter)
league/flysystem-cached-adapter suggests installing ext-phppredis (Pure C implemented extension for PHP)
Writing lock file
Generating autoload files
> @php think service:discover
Succeed!
> @php think vendor:publish
Succeed!

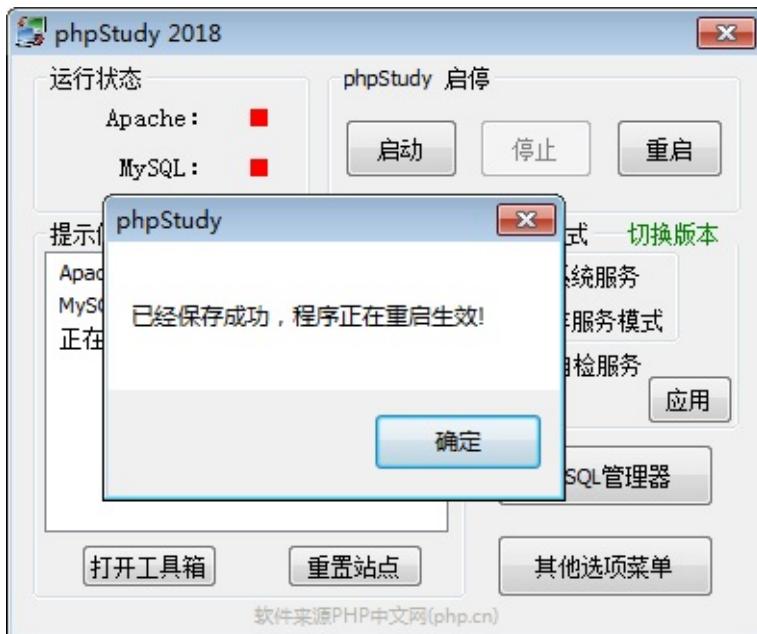
D:\www\thinkphp>
```

点击 站点域名管理

- 填写 网站域名 (虚拟域名)
- 填写 网站目录 : 指向public目录
- 点击 新增
- 点击 保存设置并生成配置文件



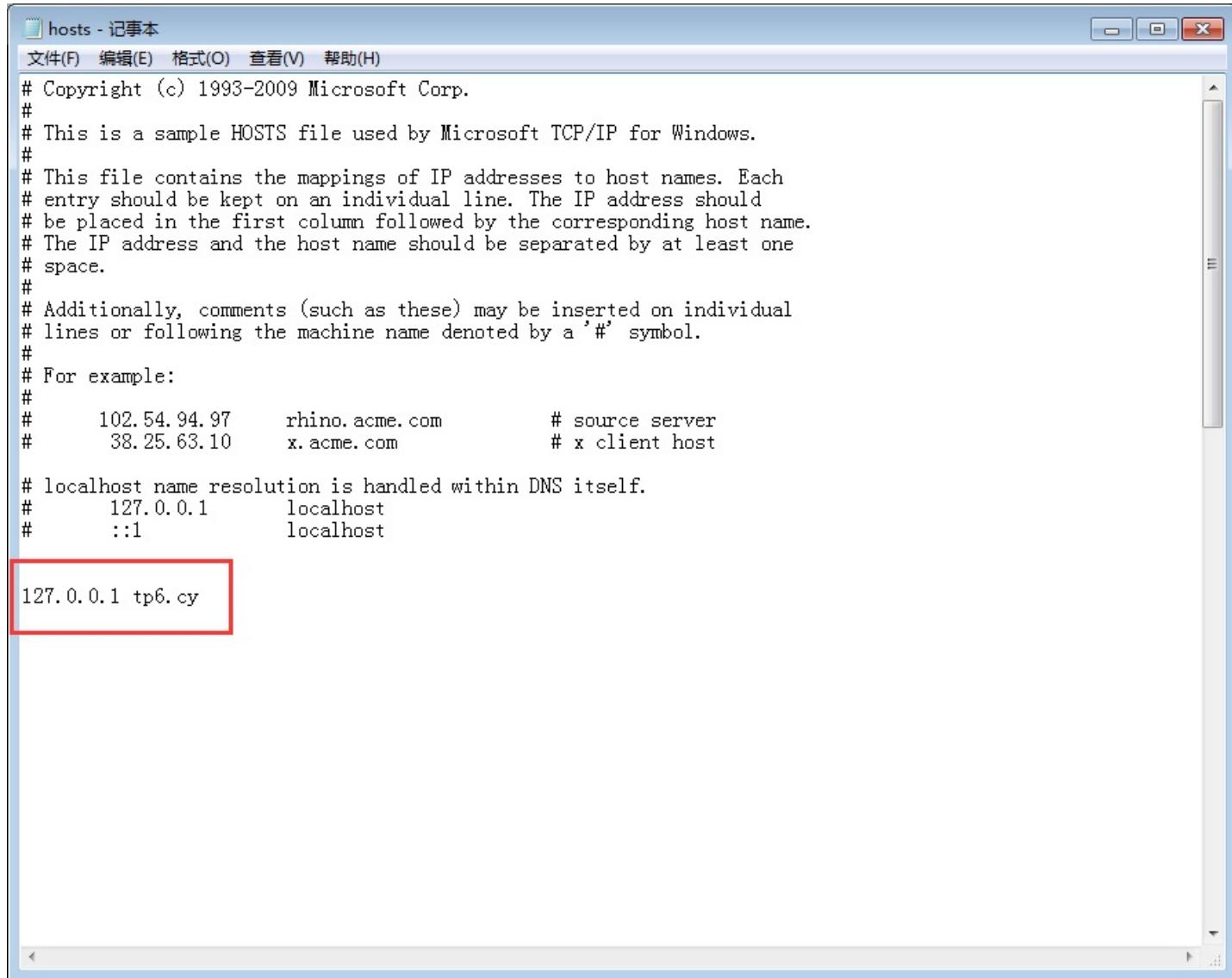
## 会弹出提示



将配置的虚拟域名添加到 hosts 文件



将 127.0.0.1 tp6.cy 添加到文件中



The screenshot shows the Windows hosts file in Notepad. The file contains comments explaining its purpose and syntax, followed by several example mappings. A new entry, "127.0.0.1 tp6.cy", has been added at the bottom and is highlighted with a red rectangular box.

```
# Copyright (c) 1993-2009 Microsoft Corp.  
#  
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.  
#  
# This file contains the mappings of IP addresses to host names. Each  
# entry should be kept on an individual line. The IP address should  
# be placed in the first column followed by the corresponding host name.  
# The IP address and the host name should be separated by at least one  
# space.  
#  
# Additionally, comments (such as these) may be inserted on individual  
# lines or following the machine name denoted by a '#' symbol.  
#  
# For example:  
#  
#      102.54.94.97    rhino.acme.com        # source server  
#      38.25.63.10    x.acme.com            # x client host  
  
# localhost name resolution is handled within DNS itself.  
#      127.0.0.1    localhost  
#      ::1          localhost  
  
127.0.0.1 tp6.cy
```

测试域名是否配置成功



# 基础

---

[单应用模式](#)

[多应用模式](#)

[多应用模式扩展](#)

# 单应用模式

## 单应用模式目录结构

-app 应用目录	
-controller	控制器目录
-model	模型目录
-view	视图目录
- ...	更多类库目录
-public	网站运行目录、入口目录、WEB目录、对外访问目录
-admin.php	后台入口文件
-index.php	入口文件
-router.php	快速测试文件
- .htaccess	用于apache的重写
-config	配置目录
-database.php	数据库配置文件
-runtime	运行时目录
-vendor	Composer类库目录
- .example.env	环境变量示例文件
-composer.json	composer 定义文件
-LICENSE.txt	授权说明文件
-README.md	README 文件
-think	命令行入口文件

# 多应用模式

## 多应用模式目录结构

-app 应用目录	
-index	index应用目录
-controller	控制器目录
-model	模型目录
-view	视图目录
-config	应用配置目录
-route	应用路由目录
-validate	应用验证器目录
-middleware	应用中间件目录
└ ...	更多类库目录
-public	网站运行目录、入口目录、WEB目录、对外访问目录
-admin.php	后台入口文件
-index.php	入口文件
-router.php	快速测试文件
└ .htaccess	用于apache的重写
-config	全局配置目录
-database.php	数据库配置文件
-runtime	运行时目录
-index	index应用运行时目录
-log	日志文件目录
-202004	以年月命名（当前时间： <b>2020-04-06</b> ）
-06.log	以几号命名（当前时间： <b>2020-04-06</b> ）
-temp	模板编译文件目录
-xxx.php	编译文件，将模板文件中的模板标签解析成了原生PHP代码
-session	session信息文件存储目录
-vendor	Composer类库目录
-example.env	环境变量示例文件
-composer.json	composer 定义文件
-LICENSE.txt	授权说明文件
-README.md	README 文件
-think	命令行入口文件

# 多应用模式扩展

## 多应用模式

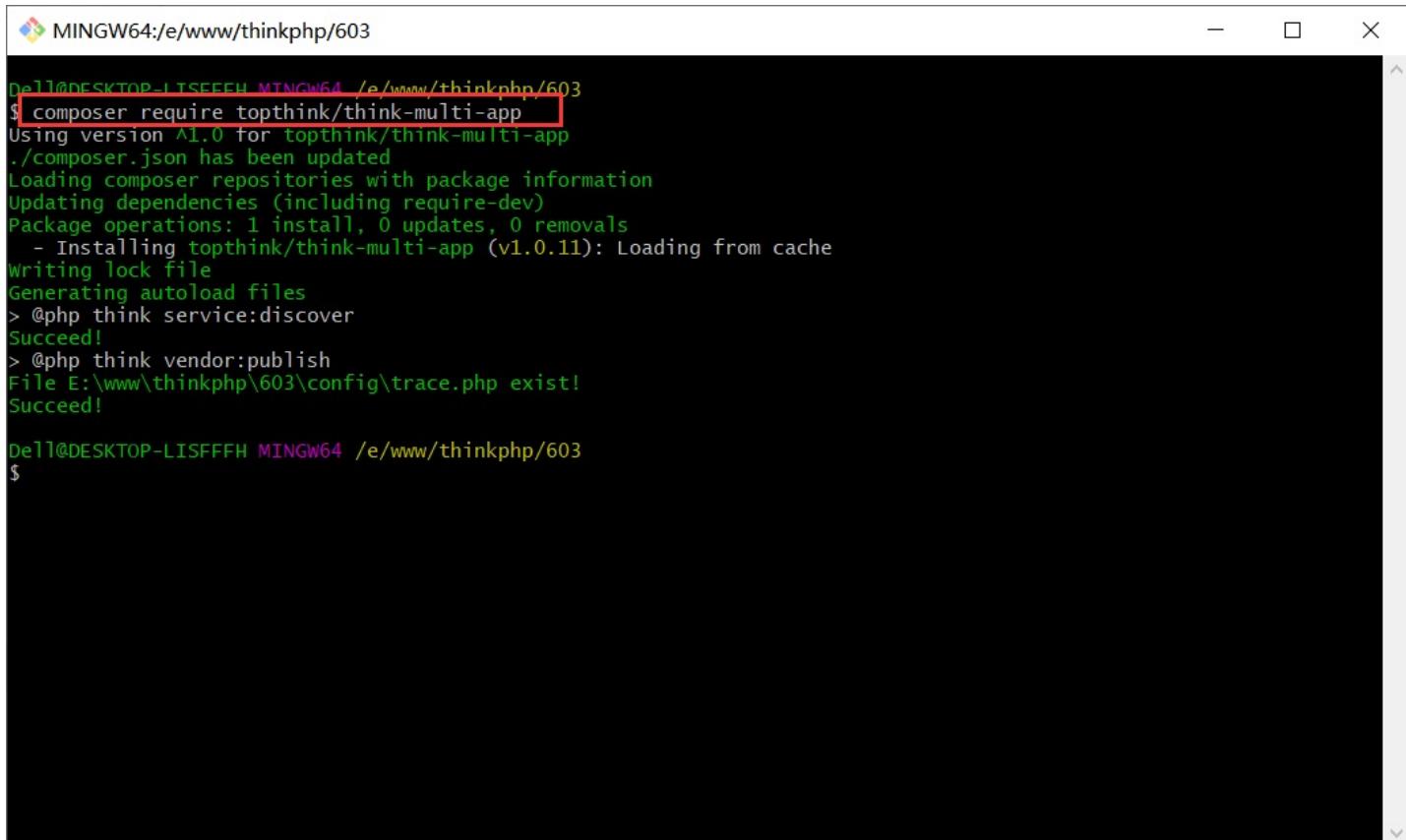
- TP6.0中的应用指的就是TP5.x中的模块
  - 多应用模式官方文档介绍
  - 应用即模块，只是叫法不同
- composer下载下来TP6.0默认是单应用模式
  - 一些情况下不需要多应用模式，如：给APP做后台时，单应用足以
  - ThinkPHP官方也许就是考虑到这点吧，所以默认采用单应用模式
- 如果网站有前台和后台，此时一般采用多应用模式
  - 使用多应用模式，必须先引入多应用模式扩展

### 通过命令行指令安装多应用模式扩展

ThinkPHP6.0 安装后默认是单应用模式

如果要使用多应用模式，需要单独安装多应用支持扩展

```
composer require topthink/think-multi-app
```

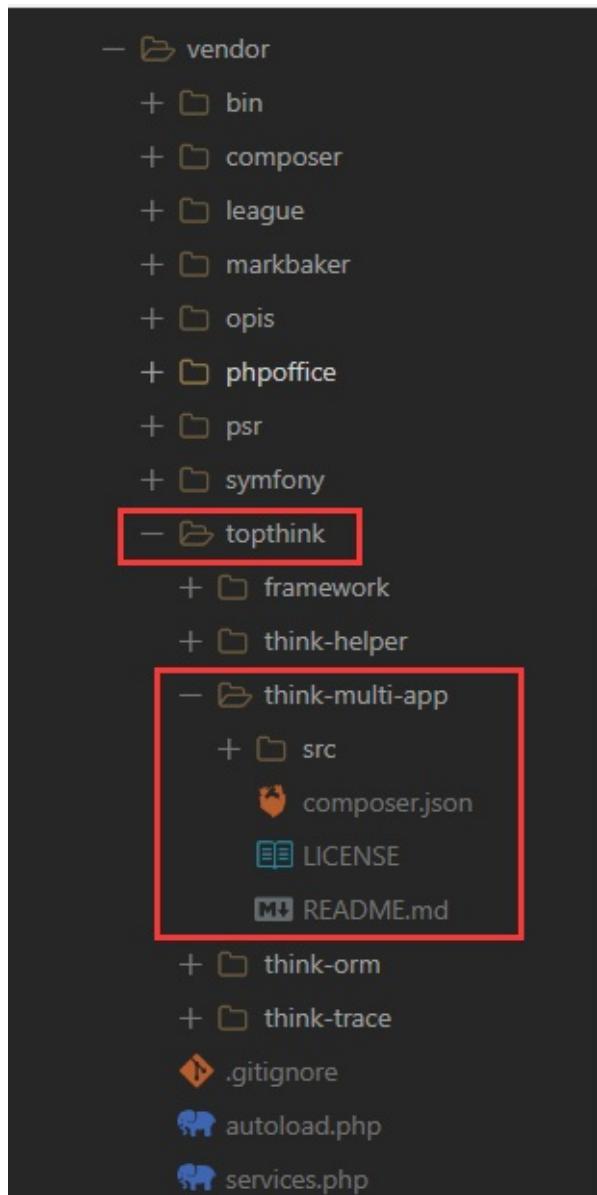


```
Dell@DESKTOP-LTSEEEH MINGW64 /e/www/thinkphp/603
$ composer require topthink/think-multi-app
Using version ^1.0 for topthink/think-multi-app
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing topthink/think-multi-app (v1.0.11): Loading from cache
Writing lock file
Generating autoload files
> @php think service:discover
Succeed!
> @php think vendor:publish
File E:\www\thinkphp\603\config\trace.php exist!
Succeed!

Dell@DESKTOP-LISFFFH MINGW64 /e/www/thinkphp/603
$
```

多应用扩展安装后 vendor/topthink 目录下会多出一个 think-multi-app 目录

vendor/topthink/think-multi-app



此时如果 app 目录下没有多应用模式目录 单应用模式的控制器仍然可以使用

没有必要这样混合使用 没有意义 删除单应用模式目录结构即可

# 配置

---

[database.php](#)

[filesystem.php](#)

[view.php](#)

[route.php](#)

[配置目录](#)

[自定义配置文件](#)

# database.php

## 数据库配置文件

config/database.php

## 各配置项的解释说明

- env('databse.type', 'mysql') 读取应用根目录下的.env文件内容,读取不到时默认值为 mysql (第二个参数)

## 数据库配置文件内容

```
return [
    // 默认使用的数据库连接配置
    'default'          => env('database.driver', 'mysql'),

    // 自定义时间查询规则
    'time_query_rule' => [],

    // 自动写入时间戳字段
    // true为自动识别类型 false关闭
    // 字符串则明确指定时间字段类型 支持 int timestamp datetime date
    'auto_timestamp'   => true,

    // 时间字段取出后的默认时间格式
    'datetime_format' => 'Y-m-d H:i:s',

    // 数据库连接配置信息
    'connections'      => [
        'mysql' => [
            // 数据库类型
            'type'           => env('database.type', 'mysql'),
            // 服务器地址
            'hostname'       => env('database.hostname', '127.0.0.1'),
            // 数据库名
            'database'        => env('database.database', ''),
            // 用户名
            'username'       => env('database.username', 'root'),
            // 密码
            'password'        => env('database.password', ''),
            // 端口
            'hostport'        => env('database.hostport', '3306'),
            // 数据库连接参数
            'params'          => [],
            // 数据库编码默认采用utf8
        ]
    ]
]
```

```
'charset'          => env('database.charset', 'utf8'),
// 数据库表前缀
'prefix'           => env('database.prefix', ''),

// 数据库部署方式:0 集中式(单一服务器),1 分布式(主从服务器)
'deploy'            => 0,
// 数据库读写是否分离 主从式有效
'rw_separate'       => false,
// 读写分离后 主服务器数量
'master_num'        => 1,
// 指定从服务器序号
'slave_no'           => '',
// 是否严格检查字段是否存在
'fields_strict'     => true,
// 是否需要断线重连
'break_reconnect'   => false,
// 监听SQL
'trigger_sql'        => env('app_debug', true),
// 开启字段缓存
'fields_cache'       => false,
// 字段缓存路径
'schema_cache_path'  => app()->getRuntimePath() . 'schema' . DIRECTORY_SEPARATOR,
],
],
// 更多的数据库配置信息
];
};
```

# filesystem.php

## 文件磁盘配置: 定义文件上传时的上传规则

全局配置文件(文件磁盘配置文件): config/filesystem.php

```
return [
    // 默认磁盘
    'default' => env('filesystem.driver', 'local'),
    // 磁盘列表
    'disks' => [
        'local' => [
            'type' => 'local',
            'root' => app()->getRuntimePath() . 'storage',
        ],
        'public' => [
            // 磁盘类型
            'type' => 'local',
            // 磁盘路径
            'root' => app()->getRootPath() . 'public/storage',
            // 磁盘路径对应的外部URL路径
            'url' => '/storage',
            // 可见性
            'visibility' => 'public',
        ],
        // 更多的磁盘配置信息
    ],
];
```

## default 默认磁盘配置

用于设置默认的磁盘，没有指定磁盘时的默认磁盘

```
$savename = \think\facade\Filesystem::putFile('topic', $file);
```

## disks 磁盘配置列表

- local 是第一个磁盘配置的配置名
- public 是第二个磁盘配置的配置名

```
// 上传到本地服务器
$savename = \think\facade\Filesystem::disk('磁盘名')->putFile('topic', $file);
```

示例：

```
$savename = \think\facade\Filesystem::disk('public')->putFile( 'topic', $file);
```

# view.php

## 视图配置文件

```
return [
    // 模板引擎类型使用Think
    'type'          => 'Think',
    // 默认模板渲染规则 1 解析为小写+下划线 2 全部转换小写 3 保持操作方法
    'auto_rule'     => 1,
    // 模板目录名
    'view_dir_name' => 'view',
    // 模板后缀
    'view_suffix'   => 'html',
    // 模板文件名分隔符
    'view_depr'     => DIRECTORY_SEPARATOR,
    // 模板引擎普通标签开始标记
    'tpl_begin'     => '{',
    // 模板引擎普通标签结束标记
    'tpl_end'       => '}',
    // 标签库标签开始标记
    'taglib_begin'  => '{',
    // 标签库标签结束标记
    'taglib_end'    => '}',

    // 模板输出替换
    'tpl_replace_string' => [
        '__STATIC__' => '/static',
    ],
];
```

# route.php

## 生成路由地址

```
// /index/index.html  
echo url('index/index') . '<br>';  
  
// /index/index.html?id=1  
echo url('index/index', ['id' => 1]) . '<br>';  
  
// /index/index.html?id=1&name=admin  
echo url('index/index', ['id' => 1, 'name' => 'admin']) . '<br>';
```

## 修改配置文件

- 将默认的true改为false

```
// URL普通方式参数 用于自动生成  
'url_common_param' => false,
```

```
/index/index.html  
/index/index/id/1.html  
/index/index/id/1/name/admin.html
```

# 配置目录

## 多应用模式下的全局配置和应用配置

- 在多应用模式下配置分为全局配置和应用配置
- 相同的配置参数 应用配置会覆盖全局配置
- 全局配置：config目录下的文件都是全局配置文件
- 应用配置：应用目录下的config目录下的文件都是应用配置文件

# 自定义配置文件

## 自定义配置文件

在配置目录下新建 mail.php

The screenshot shows a file tree on the left and a code editor on the right. The file tree under 'config' includes app.php, cache.php, console.php, cookie.php, database.php, filesystem.php, lang.php, log.php, and mail.php, which is highlighted with a red box. The code editor shows the following PHP code:

```
<?php  
return [  
    'host' => 'stmp.qq.com',  
];
```

## 读取配置文件内容

```
// 返回数组  
$mail = \think\facade\Config::get('mail');  
// 返回 stmp.qq.com  
$host = \think\facade\Config::get('mail.host');
```

# 路由

---

[路由简介](#)

[注册路由](#)

[路由别名](#)

[多应用路由](#)

[域名绑定路由](#)

# 路由简介

## 路由的作用

- 简化 URL 地址
- 隐藏真实的路径，提供安全性

# 注册路由

## 引入路由类

```
use think\facade\Route;
```

## 注册路由

```
Route::rule('路由表达式', '路由地址', '请求类型');
```

## 注册对应请求类型的快捷方法路由

```
Route::快捷方法名('路由表达式', '路由地址');
```

类型	描述	快捷方法
GET	GET请求	get
POST	POST请求	post
PUT	PUT请求	put
DELETE	DELETE请求	delete
PATCH	PATCH请求	patch
*	任何请求类型	any

# 路由别名

## 路由别名

```
// 文章详情页 {:url('index.article.detail', ['id' => 1])}
Route::get('art/:id', 'article/detail')->name('index.article.detail');

// /index/art/1.html
echo url('index.article.detail', ['id' => 1]);
```

# 多应用路由

## 应用的路由规则

- 应用的路由是定义的入口文件（或者应用名）后面的URL部分，而不包含应用。

### 应用路由示例

- 默认的pathinfo方式：/index/user/login
- 注册路由：Route::get('login', 'user/login');
- 访问方式：由 /index/user/login 变为 /index/login，在使用 /index/user/login 提示 非法请求
- index 应用的路由定义在 app/index/route 目录下

### index应用路由定义示例

```
app/index/route/app.php
```

```
use think\facade\Route;

// 路由aa 对应index应用的index控制器 hello方法
Route::get('aa', 'index/hello');
```

浏览器访问：域名/index/aa

# 域名绑定路由

## 域名路由

ThinkPHP支持完整域名、子域名和IP部署的路由和绑定功能，同时还可以起到简化URL的作用。

### 全局配置文件 app.php

```
admin => 'admin'
```

如果当前域名是 tp.com

则二级域名 admin.tp.com 会指向 admin 应用

本文档使用 看云 构建

# 中间件

---

[中间件简介](#)

[中间件文件](#)

[中间件定义](#)

[中间件别名](#)

[注册中间件](#)

[全局中间件](#)

[应用中间件](#)

[路由中间件](#)

[控制器中间件](#)

# 中间件简介

## 中间件的作用

主要用于拦截或过滤应用的 `HTTP` 请求，并进行必要的业务处理

## 中间件的使用步骤

- 第一步：生成中间件文件，定义中间件
- 第二步：在全局配置文件中定义中间件别名(可省略)
- 第三步：注册中间件（全局、应用、路由、控制器中间件）

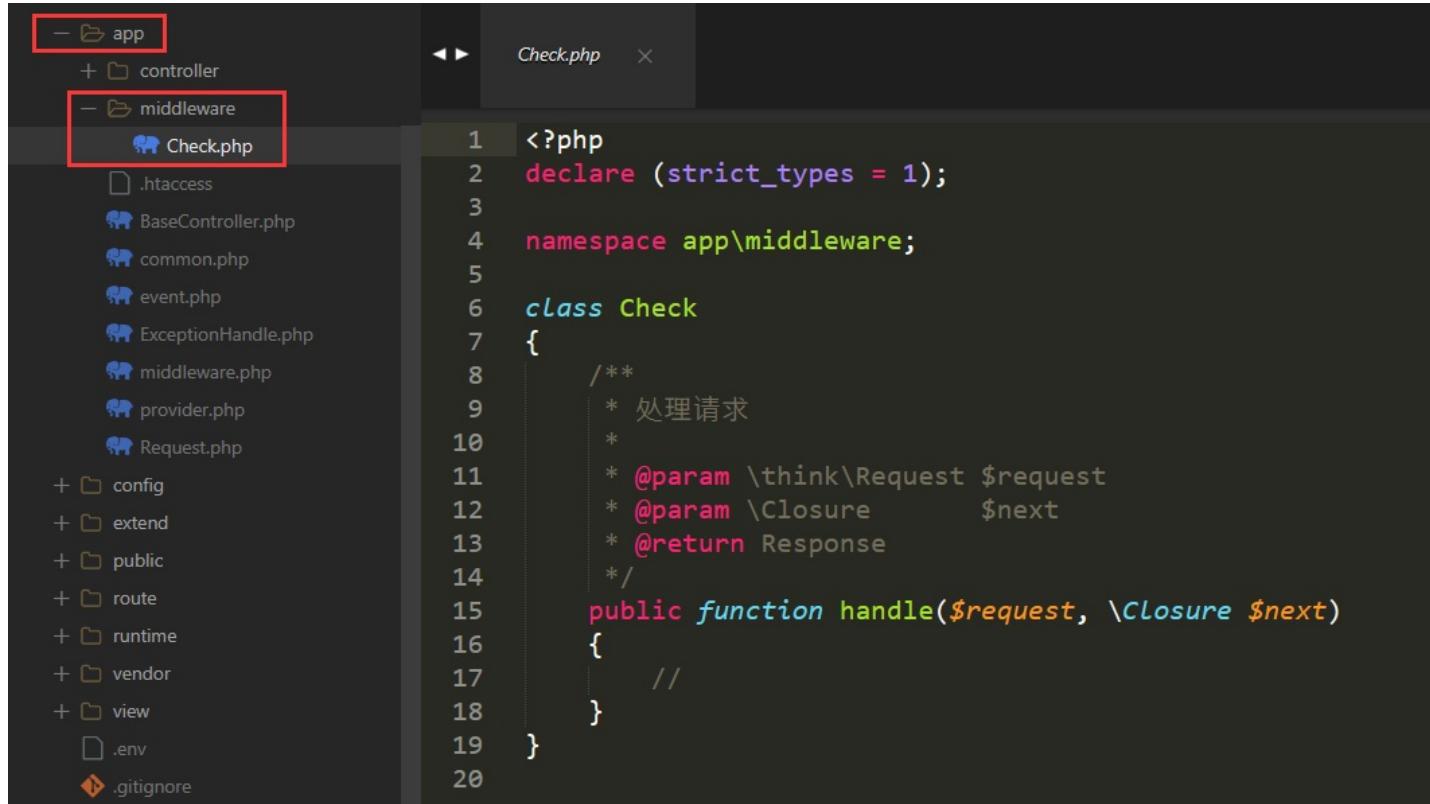
# 中间件文件

通过命令行指令快速生成中间件

```
php think make:middleware Check
```

这个指令会"app/middleware"目录下面生成一个"Check"中间件。

通过指令创建中间件文件默认根目录 : "app/middleware" middleware目录不存在会自动创建



The screenshot shows a code editor interface with a sidebar on the left displaying the project structure of a ThinkPHP application. The 'app' directory is expanded, showing 'controller', 'middleware', 'view', and other sub-directories. Inside 'middleware', a file named 'Check.php' is selected and highlighted with a red border. The main editor area shows the generated PHP code for the middleware:

```
1 <?php
2 declare (strict_types = 1);
3
4 namespace app\middleware;
5
6 class Check
7 {
8     /**
9      * 处理请求
10     *
11     * @param \think\Request $request
12     * @param \Closure $next
13     * @return Response
14     */
15    public function handle($request, \Closure $next)
16    {
17        //
18    }
19}
20
```

# 中间件定义

## 中间件定义

中间件的入口执行方法必须是 "handle" 方法,  
而且第一个参数是 "Request" 对象, 第二个参数是一个闭包

```
<?php

namespace app\middleware;

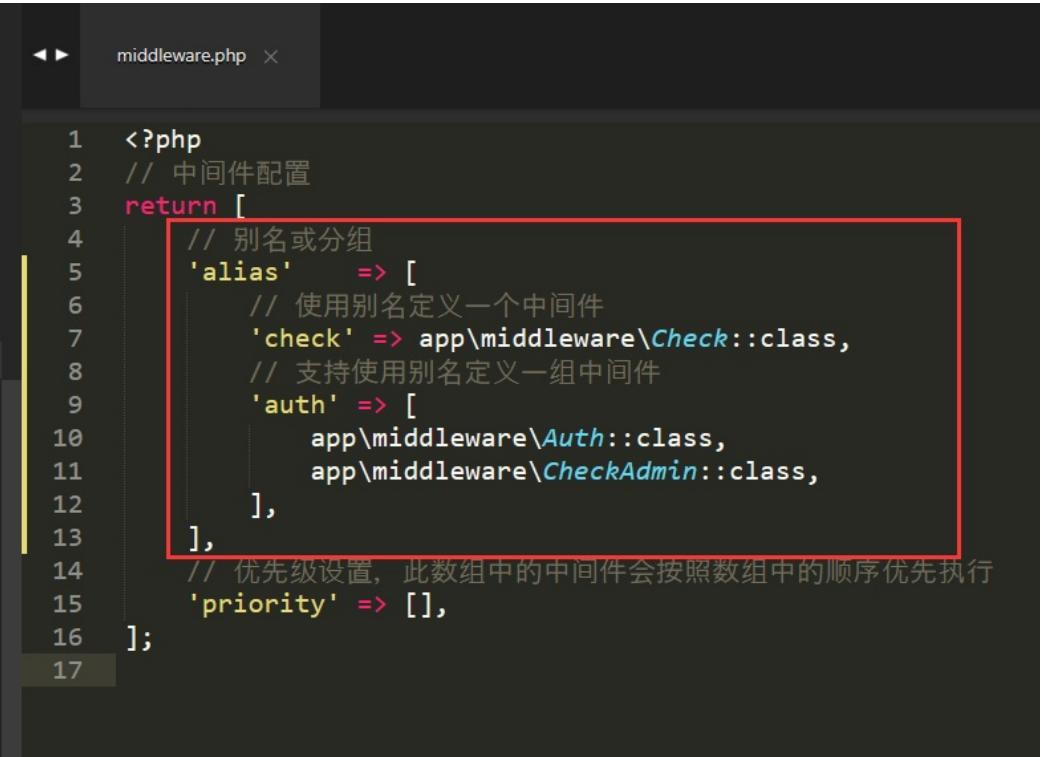
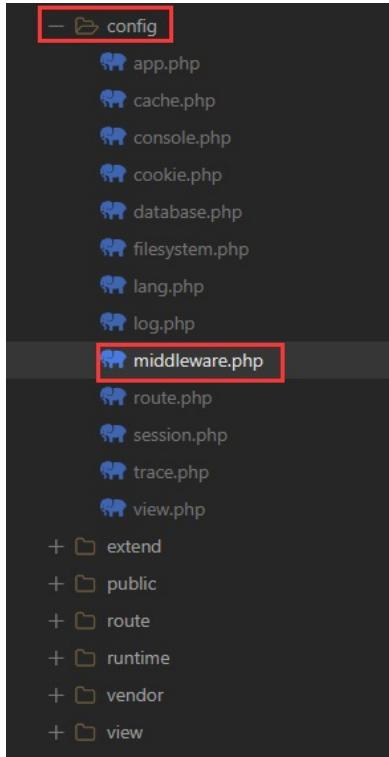
class Check
{
    public function handle($request, \Closure $next)
    {
        if ($request->param('name') == 'think') {
            return redirect('index/think');
        }

        return $next($request);
    }
}
```

# 中间件别名

## 定义中间件别名

在全局配置目录下的 "middleware.php" 中定义中间件别名



```
1 <?php
2 // 中间件配置
3 return [
4     // 别名或分组
5     'alias' => [
6         // 使用别名定义一个中间件
7         'check' => app\middleware\Check::class,
8         // 支持使用别名定义一组中间件
9         'auth' => [
10             app\middleware\Auth::class,
11             app\middleware\CheckAdmin::class,
12         ],
13     ],
14     // 优先级设置，此数组中的中间件会按照数组中的顺序优先执行
15     'priority' => [],
16 ];
17 ];
```

# 注册中间件

## 中间件执行顺序

全局中间件 -> 应用中间件(多应用模式下有效) -> 路由中间件 -> 控制器中间件

# 全局中间件

## 注册全局中间件

app/middleware.php

中间件的注册应该使用完整的类名

如果已经定义了中间件别名（或者分组）则可以直接使用定义的中间件别名

全局中间件的执行顺序就是定义顺序：数组元素靠前的先执行

```
1 <?php
2 // 全局中间件定义文件
3 return [
4     // 全局请求缓存
5     // \think\middleware\CheckRequestCache::class,
6     // 多语言加载
7     // \think\middleware\LoadLangPack::class,
8     // Session初始化
9     // \think\middleware\SessionInit::class,
10    // 自定义中间件
11    // 中间件文件位置 : app/middleware/Check.php
12    app\middleware\Check::class,
13];
14];
15];
```

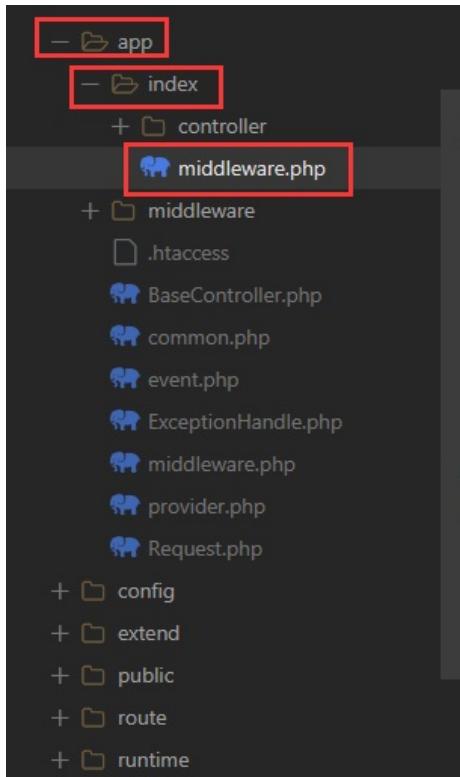
# 应用中间件

## 注册应用中间件

如果使用了多应用模式，则支持应用中间件定义

直接在应用目录下面增加 "middleware.php" 文件

定义方式和全局中间件定义一样，只是只会在该应用下面生效



The screenshot shows a file explorer interface with the following directory structure:

- app
- index
- + controller
- middleware.php** (highlighted with a red box)
- + middleware
  - .htaccess
  - BaseController.php
  - common.php
  - event.php
  - ExceptionHandle.php
  - middleware.php
  - provider.php
  - Request.php
- + config
- + extend
- + public
- + route
- + runtime

The right pane shows the contents of the selected `middleware.php` file:

```
1 <?php
2
3 // index 应用中间件定义文件
4
5 return [
6     // 自定义中间件
7     // 中间件文件位置 : app\middleware\Check.php
8     app\middleware\Check::class,
9 ];
10
```

# 路由中间件

注册路由中间件

# 控制器中间件

## 注册控制器中间件

- 控制器中间件必须使用中间件别名
- 在控制器中定义 `middleware` 属性

```
<?php
namespace app\controller;

class Index
{
    // 注册控制器中间件
    protected $middleware = [
        // 对所有方法有效
        'auth',
        // 仅对hello方法和world方法有效
        'check' => ['only' => ['hello', 'world']],
        // 仅对create方法和save方法无效
        'check' => ['except' => ['create', 'save']],
    ];
}
```

# 控制器

---

[基础控制器](#)

[创建控制器](#)

[控制器命名](#)

[控制器目录](#)

[控制器定义](#)

[控制器后缀](#)

[多级控制器](#)

[渲染输出](#)

# 基础控制器

---

[初始化方法](#)

[注入的对象](#)

[数据验证功能](#)

# 初始化方法

## 基础控制器的初始化方法 `initialize()`

- 初始化方法在构造方法中被调用
- 继承基础控制器的控制器初始化操作应重写父类方法 `initialize()`，而不是重写构造方法
- 重写基础控制器构造方法(不推荐)：传入参数 App \$app

```
/**  
 * 构造方法  
 * @access public  
 * @param App $app 应用对象  
 */  
public function __construct(App $app)  
{  
    $this->app = $app;  
    $this->request = $this->app->request;  
  
    // 控制器初始化  
    $this->initialize();  
}  
  
// 初始化  
protected function initialize()  
{}
```

# 注入的对象

## 基础控制器

- 框架默认提供了一个基础控制器类 `app/BaseController.php`
- 基础控制器的位置可以任意放置，也可以随便修改，只需要更改命名空间即可

## 基础控制器注入的对象

在 `app/BaseController.php` 中的构造方法中

注入了 `think\App` 和 `think\Request` 对象

因此继承基础控制器的控制器可以通过`app`属性和`request`属性分别调用 `think\App` 和 `think\Request` 对象实例

```
/**
 * 构造方法
 * @access public
 * @param App $app 应用对象
 */
public function __construct(App $app)
{
    $this->app      = $app;
    $this->request = $this->app->request;

    // 控制器初始化
    $this->initialize();
}
```

# 数据验证功能

基础控制器提供了数据验证的功能

app/BaseController.php 的 validate() 方法

```
/*
 * 验证数据
 * @access protected
 * @param array      $data    数据
 * @param string|array $validate 验证器名或者验证规则数组
 * @param array      $message 提示信息
 * @param bool       $batch   是否批量验证
 * @return array|string|true
 * @throws ValidateException
 */
protected function validate(array $data, $validate, array $message = [], bool $batch = false)
{
    if (is_array($validate)) {
        $v = new Validate();
        $v->rule($validate);
    } else {
        if (strpos($validate, '.')) {
            // 支持场景
            [$validate, $scene] = explode('.', $validate);
        }
        $class = false !== strpos($validate, '\\') ? $validate : $this->app->parseClass('validate', $validate);
        $v     = new $class();
        if (!empty($scene)) {
            $v->scene($scene);
        }
    }
    $v->message($message);

    // 是否批量验证
    if ($batch || $this->batchValidate) {
        $v->batch(true);
    }

    return $v->failException(true)->check($data);
}
```

继承基础控制器的控制器可以使用以下验证

```
try {
    $this->validate( [
        'name' => 'thinkphp',
        'email' => 'thinkphp@qq.com',
    ], 'app\index\validate\User');
} catch (\think\exception\ValidateException $e) {
    // 验证失败 输出错误信息
    halt($e->getError());
}
```

指定验证场景：add 是验证场景名称

```
try {
    $this->validate($data, 'app\admin\validate\Node.add');
} catch (\think\exception\ValidateException $e) {
    // 验证失败 输出提示信息
    $this->error($e->getError());
}
```

# 创建控制器

## 命令行创建控制器

- 单应用下创建控制器

```
php think make:controller Index
```

- 在 admin 应用下创建控制器

```
php think make:controller admin@Index
```

# 控制器命名

## 控制器命名规范

- 采用大驼峰命名法
- 类名和文件名保持大小写一致

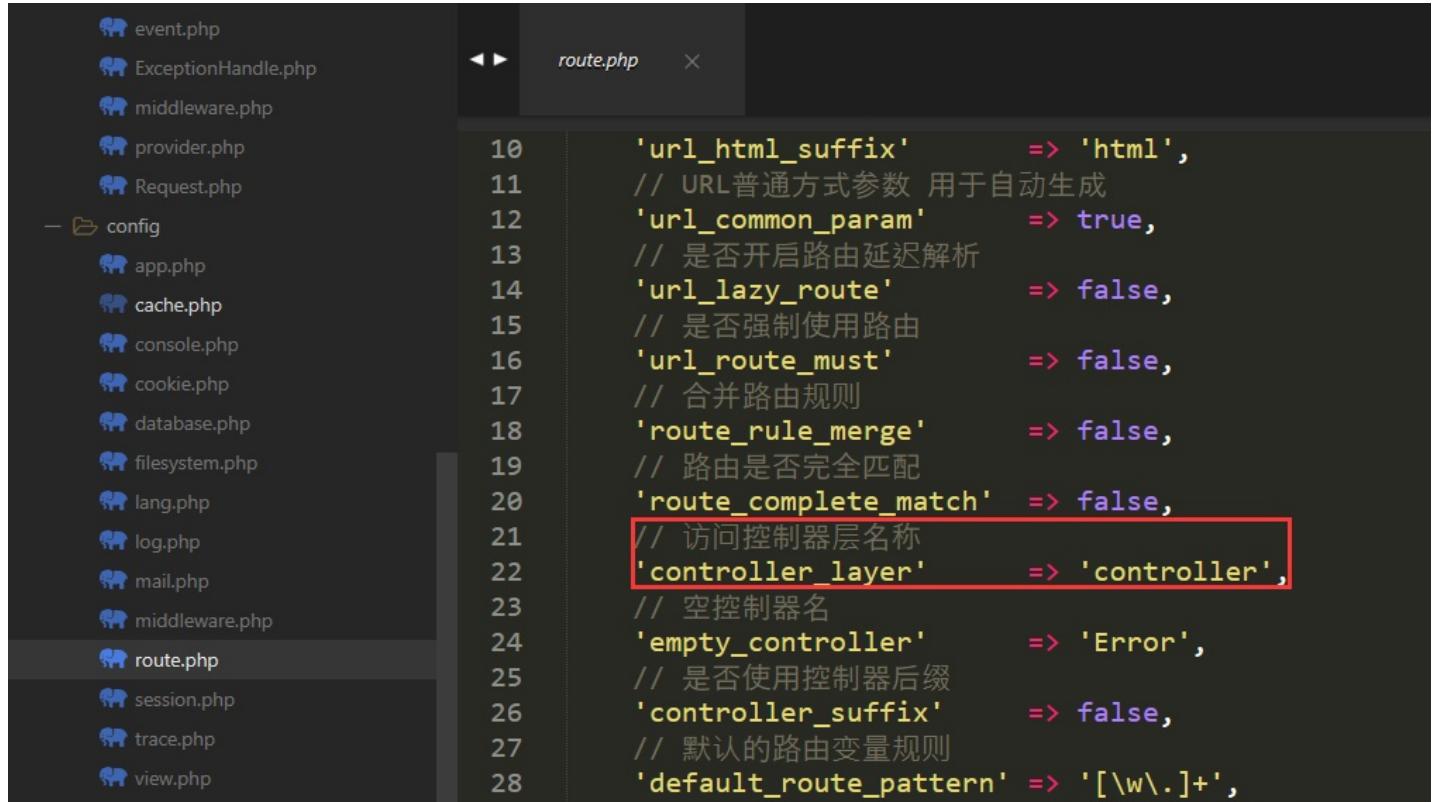
# 控制器目录

## 自定义控制器目录名

控制器文件通常放在controller目录下面

如果要改变controller目录名，需要在route.php配置文件中设置

config/route.php controller\_layer



```
event.php
ExceptionHandle.php
middleware.php
provider.php
Request.php
config
  app.php
  cache.php
  console.php
  cookie.php
  database.php
  filesystem.php
  lang.php
  log.php
  mail.php
  middleware.php
  route.php
  session.php
  trace.php
  view.php

route.php
10      'url_html_suffix'      => 'html',
11      // URL普通方式参数 用于自动生成
12      'url_common_param'     => true,
13      // 是否开启路由延迟解析
14      'url_lazy_route'       => false,
15      // 是否强制使用路由
16      'url_route_must'       => false,
17      // 合并路由规则
18      'route_rule_merge'     => false,
19      // 路由是否完全匹配
20      'route_complete_match' => false,
21      // 访问控制器层名称
22      'controller_layer'     => 'controller',
23      // 空控制器名
24      'empty_controller'     => 'Error',
25      // 是否使用控制器后缀
26      'controller_SUFFIX'    => false,
27      // 默认的路由变量规则
28      'default_route_pattern'=> '[\w\.\]+',
```

# 控制器定义

- [单应用模式控制器](#)
- [多应用模式控制器](#)

## 单应用模式控制器

控制器类文件

app\controller\User.php

```
<?php
namespace app\controller;

class User
{
    public function login()
    {
        return 'login';
    }
}
```

访问URL地址是（假设没有定义路由）

http://localhost/user/login

如果控制器名是 HelloWorld 有个hello()方法

访问URL地址：http://localhost/hello\_world/hello

## 多应用模式控制器

控制器类文件

app\shop\controller\User.php

```
<?php
namespace app\shop\controller;

class User
{
```

## 控制器定义

```
public function login()
{
    return 'login';
}
```

访问URL地址是（假设没有定义路由）

<http://localhost/index.php/shop/user/login>

# 控制器后缀

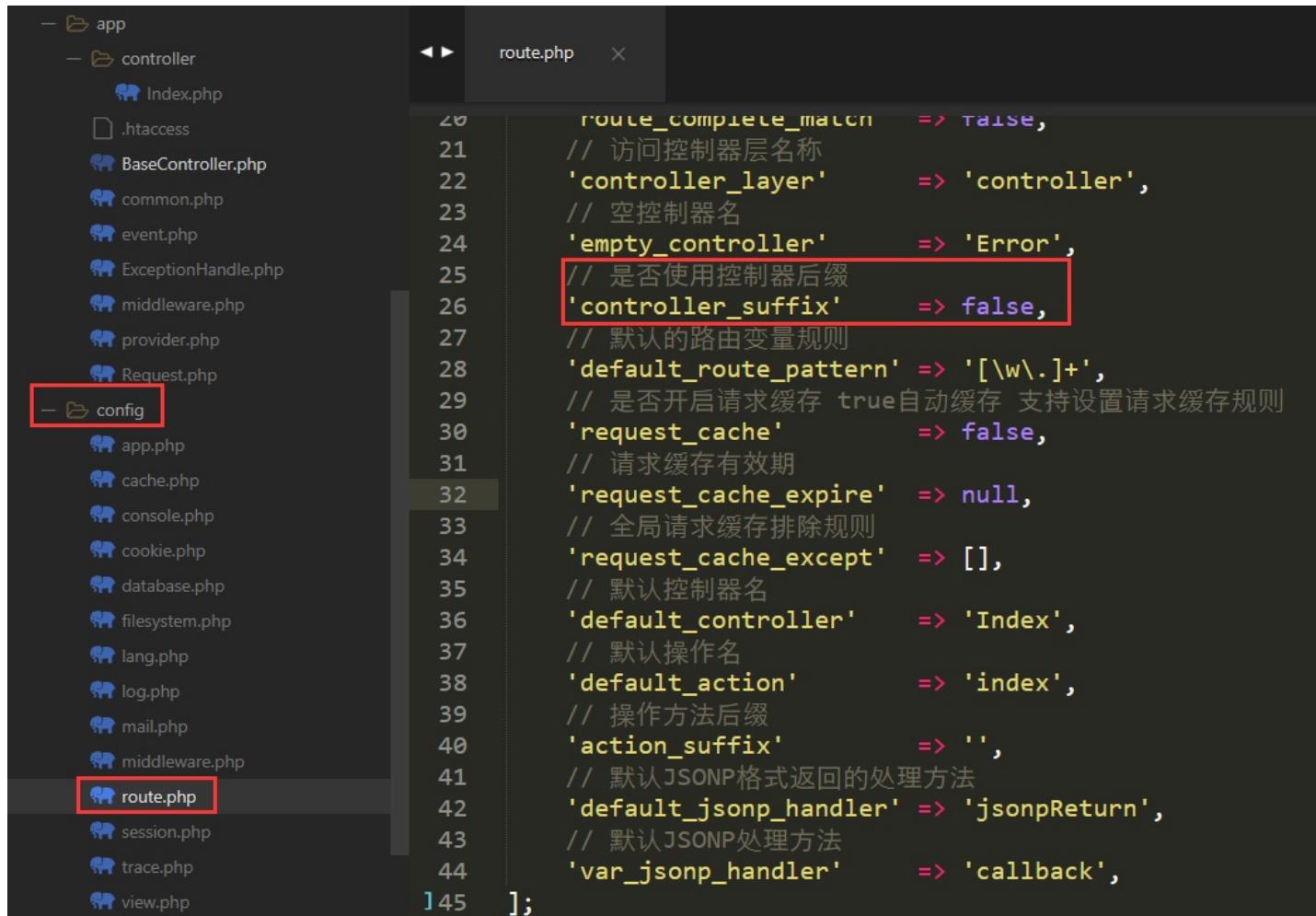
## 自定义控制器后缀

如果你希望避免引入同名模型类的时候冲突，可以在 `route.php` 配置文件中设置

```
// 使用控制器后缀  
'controller_suffix'      => true,
```

`app\controller\UserController.php`

```
<?php  
namespace app\controller;  
  
class UserController  
{  
    public function login()  
    {  
        return 'login';  
    }  
}
```



```
route_complete_match => false,
// 访问控制器层名称
'controller_layer' => 'controller',
// 空控制器名
'empty_controller' => 'Error',
// 是否使用控制器后缀
'controller_suffix' => false,
// 默认的路由变量规则
'default_route_pattern' => '[\w\.\.]+',
// 是否开启请求缓存 true自动缓存 支持设置请求缓存规则
'request_cache' => false,
// 请求缓存有效期
'request_cache_expire' => null,
// 全局请求缓存排除规则
'request_cache_except' => [],
// 默认控制器名
'default_controller' => 'Index',
// 默认操作名
'default_action' => 'index',
// 操作方法后缀
'action_SUFFIX' => '',
// 默认JSONP格式返回的处理方法
'default_jsonp_handler' => 'jsonpReturn',
// 默认JSONP处理方法
'var_jsonp_handler' => 'callback',
];

```

# 多级控制器

## 多级控制器

支持任意层次级别的控制器，并且支持路由

# 渲染输出

- `return` 渲染输出
- 调试并中止执行

## return 渲染输出

默认情况下，控制器的输出全部采用 `return` 的方式，  
无需进行任何的手动输出，系统会自动完成渲染内容的输出

## 调试并中止执行

```
halt('输出测试');
```

# 请求

---

[请求对象](#)

[请求信息](#)

[请求类型](#)

# 请求对象

## 获取请求对象的方式

- 基础控制器的 `request` 属性
- 构造方法依赖注入（没有基础控制器可以使用该方式）
- `request()` 助手函数 [请求对象官方文档](#)

当前类文件继承了 `BaseController` 类

```
use app\BaseController;

class Index extends BaseController
{
    public function index()
    {
        // 请求对象
        $request = $this->request;
    }
}
```

构造方法注入请求对象

一般适用于没有继承系统的控制器类的情况。

```
<?php

namespace app\index\controller;

use think\Request;

class Index
{
    /**
     * @var \think\Request Request实例
     */
    protected $request;

    /**
     * 构造方法
     * @param Request $request Request对象
     * @access public
     */
}
```

## 请求对象

```
public function __construct(Request $request)
{
    $this->request = $request;
}

public function index()
{
    return $this->request->param('name');
}
```

## Request 类

```
\think\Facade\Request::instance();
```

## request() 助手函数

```
// 返回 Request 对象
request();
```

# 请求信息

通过 Request 类获取当前请求信息

方法	描述
host	获取当前域名 示例：当前url地址为 http://tp6.cy，返回的是 tp6.cy
scheme	获取当前网络协议 示例：当前url地址为 http://tp6.cy，返回的是 http
ip	获取用户IP 示例：返回 127.0.0.1

调用示例

```
// Request 静态调用
\think\facade\Request::host();

// request() 助手函数调用
request()->host();

// 继承了基础控制器的控制器类通过request属性调用
$this->request->host();
```

# 请求类型

## 判断当前请求类型

用途	方法
获取当前请求类型	method
判断是否GET请求	isGet
判断是否POST请求	isPost
判断是否PUT请求	isPut
判断是否DELETE请求	isDelete
判断是否AJAX请求	isAjax
判断是否PJAX请求	isPjax
判断是否JSON请求	isJson
判断是否手机访问	isMobile
判断是否HEAD请求	isHead
判断是否PATCH请求	isPatch
判断是否OPTIONS请求	isOptions
判断是否为CLI执行	isCli
判断是否为CGI模式	isCgi

# 数据库

---

[连接数据库](#)

[指定数据表](#)

[查询](#)

[更新](#)

[链式操作](#)

[事务操作](#)

[聚合查询](#)

[分页查询](#)

# 连接数据库

使用 connect() 指定使用的 数据库连接配置信息

- connect() 的参数

```
return [
    // 默认使用的数据库连接配置
    'default' => env('database.driver', 'mysql'),
    // 自定义时间查询规则
    'time_query_rule' => [],
    // 自动写入时间戳字段
    // true为自动识别类型 false关闭
    // 字符串则明确指定时间字段类型 支持 int timestamp date
    'auto_timestamp' => true,
    // 时间字段取出后的默认时间格式
    'datetime_format' => 'Y-m-d H:i:s',
    // 数据库连接配置信息
    'connections' => [
        'mysql' => [
            ...
        ],
        'db2' => [
            ...
        ],
    ],
];
```

- 使用示例

```
\think\facade\Db::connect('db2')->table('user')->find();
```

# 指定数据表

table 必须指定完整的数据表名称

```
\think\facade\Db::table('完整表名')->select();
```

name 省略表前缀的表名

- 在数据库配置文件中设定表前缀 .env 或 config/database.php

```
// 数据库表前缀  
'prefix' => ''
```

# 查询

## 查询多条数据

```
use think\facade\Db;  
  
// 返回值：对象  
Db::table('think_user')->select();  
  
// 返回值：二维数组  
Db::table('ay_config')->select()->toArray();
```

## 查询单条数据

find()方法 查询到数据返回一维数组，查不到数据返回 NULL

```
Db::table('user')->find(1);  
  
Db::table('user')->where('username', '张三')->find();
```

## find()方法 必须加查询条件，否则返回NULL

```
Db::table('user')->find(); // 返回 NULL
```

# find

## find 快速使用

- 根据主键查询

```
Db::table('user')->find(1);
```

- 指定字段的值查询

```
Db::table('user')->where('username', '张三')->find();
```

## find 简述

- find 查询必须有查询条件，否则返回 NULL
- 用于查询单条数据，查到数据返回一维的关联数组，查询不到返回NULL

## find 查询必须有查询条件

- 必须有查询条件 否则返回 NULL
- 没有查询条件，即使表中有数据也会返回 NULL
- 经测试，以下查询根本就没有去连接数据库，此时数据库配置是错误的也不会提示错误

```
Db::table('user')->find();
```

## 和 find 相关的方法

```
// 和find的区别：查询不到数据返回空数组，而 find() 返回的是NULL  
Db::table('user')->findOrEmpty(1);
```

```
// 和find的区别：查询不到数据抛出异常，而 find() 返回的是NULL  
Db::table('user')->findOrFail(1);
```

# select

## select 简述

- 用于查询多条数据

### 查询数据集，返回数据集对象

```
Db::table('user')->select();
```

### 查询数据集，返回数组

```
// 返回空数组或二维的索引关联数组  
Db::table('user')->select()->toArray();
```

## 错误示范

无论是否查到数据都返回数据集对象，所有不要使用以下方式判断是否查到数据

```
$data = Db::table('art')->select();  
  
if ($data) {  
    echo '查到数据';  
} else {  
    echo '没有查到数据';  
}
```

# paginate

## 分页查询

在查询的时候调用 paginate 方法

```
Db::table('articles')->paginate(1);
```

## 分页传参

更新

# 更新

---

[save](#)

[update](#)

# save

## save() 更新数据

- save() 方法既可以添加数据，也可以更新数据
- save() 方法执行的是添加还是更新，看的是第二个参数中是否含有主键字段
- 第二个参数中含有主键字段就是更新，否则执行的是添加操作

```
// 执行更新操作时返回影响行数 1 或 0  
// 1 更新成功 0 数据不存在或新数据和旧数据相同  
Db::table('admin')->save(['id' => 13, 'username' => '张三']);
```

# update

## update() 更新数据

- 假设 `id` 为主键字段，以下操作可以更新

```
// $table 表名 $data 更新的数据
// Db::table(string $table)->update(array $data);

// 返回影响行数，没有修改任何数据或当前更新条件下无数据返回 0
Db::table('admin')->update(['id' => 1, 'username' => '张三']);
```

- 如果更新的数据中没有主键字段，将提示以下信息

#0 [10500]DbException in BaseQuery.php line 1029

## 缺少更新条件

```
1020.         $this->parseUpdateData($this->options['data']);
1021.     }
1022.
1023.     if (empty($this->options['where']) && $this->model) {
1024.         $this->where($this->model->getWhere());
1025.     }
1026.
1027.     if (empty($this->options['where'])) {
1028.         // 如果没有任何更新条件则不执行
1029.         throw new Exception('miss update condition');
1030.     }
1031.
1032.     return $this->connection->update($this);
1033. }
1034.
1035. /**
1036. * 删除记录
1037. * @access public
1038. * @param mixed $data 表达式 true 表示强制删除
```

# 链式操作

---

[field](#)

[join](#)

[where](#)

[fetchSql](#)

# field

## field 简述

- 指定要查询的字段，也可以取别名
- 参数类型既可以是字符串也可以是数组

## 字符串参数

```
$data = Db::table('user')->field('id, nickname as name')->select();
```

相当于执行以下SQL语句

```
SELECT id, nickname as name FROM user;
```

## 数组参数

```
$data = Db::table('articles')->field(['id', 'nickname' => 'name'])->select();
```

相当于执行以下SQL语句

```
SELECT id, nickname as name FROM user;
```

## 显式调用所有字段

field(true) 会将所有表的字段显式调用

```
Db::table('user')->field(true)->select();
```

假设user表只有 `id` , `username` , `password` 三个字段，则以上SQL语句相当于

```
SELECT id, username, password FROM user;
```

# join

```
$data = Db::table('articles')
    // as 可省略
    ->alias('as a')
    ->join('categories c', 'a.cate_id = c.id')
    ->field('a.id, a.title, a.description, a.add_time, a.click, a.thumb, c.cate_
name')
    ->fetchSql(true)
    ->select();
```

```
SELECT `a`.`id`, `a`.`title`, `a`.`description`, `a`.`add_time`, `a`.`click`, `a`.`t
humb`, `c`.`cate_name` FROM `articles` as a INNER JOIN `categories` `c` ON `a`.`
cate_id` = `c`.`id`;
```

where

# where

---

[whereIn](#)

[字符串条件](#)

[查询表达式](#)

# whereIn

## whereIn() 快捷查询方法

- 以下查询等价

```
self::where('id', 'in', '1,2,3')->select();
self::where('id', 'in', [1, 2, 3])->select();

self::whereIn('id', [1, 2, 3])->select();
self::whereIn('id', [1, 2, 3])->select();
```

- 查看执行的SQL

```
self::whereIn('id', [1, 2, 3])->fetchSql(true)->select();
```

- 执行的SQL

```
SELECT * FROM `user` WHERE `id` IN (1, 2, 3)
```

# 字符串条件

## where 字符串条件查询

- 一个 where

```
self::where('mobile', '<>', 1)->select();
```

```
SELECT * FROM `user` WHERE `mobile` <> 1
```

- 多个 where

```
self::where('name', '张三')->where('mobile', '<>', 10086)->select();
```

```
SELECT * FROM `user` WHERE `name` = '张三' AND `mobile` <> 10086
```

# 查询表达式

## where() 方法使用格式

```
where('字段名', '查询表达式', '查询条件');
```

# fetchSql

在模型中的使用

```
// 返回SQL语句  
$sql = User::fetchSql(true)->find(10);
```

在分页查询时无效，以下用法虽然没有报错但是 fetchSql(true) 无效

```
// 返回对象而不是SQL语句  
User::fetchSql(true)->paginate(['list_rows' => 10]);
```

# 事务操作

---

[使用示例](#)

# 使用示例

## 事务操作示例

```
// 启动事务
Db::startTrans();
try {
    Db::table('think_user')->find(1);
    Db::table('think_user')->delete(1);
    // 提交事务
    Db::commit();
} catch (\Exception $e) {
    // 回滚事务
    Db::rollback();
    // 事务执行中有错误
    die('错误信息: ' . $e->getMessage());
}
```

# 聚合查询

---

[count](#)

# Count

## count() 用于统计数量

- 可用于数据库操作和模型方法
- 还可以用于统计子模型数据总数

## Db类的使用

```
Db::table('user')->count();
```

## 在模型中的使用

```
self::where('id', '<', 10)->count();
```

# 分页查询

---

[分页参数](#)

[分页样式](#)

[分页样式代码](#)

[自定义分页驱动](#)

[默认分页代码](#)

# 分页参数

## 分页参数

参数	描述
list_rows	每页数量
page	指定获取第n页的数据
path	url路径
query	url额外参数
fragment	url锚点
var_page	分页变量

## 常用参数

```
$list = Db::table('user')->paginate([
    'list_rows' => 10,
    'query'      => request()->param(),
]);
```

# 分页样式

## Db 类查询的时候调用 paginate 方法

- 控制器方法

```
<?php
namespace app\controller;

use app\BaseController;

class Index extends BaseController
{
    public function index()
    {
        // 分页查询 每页2条数据
        $list = \think\facade\Db::name('user')->paginate(2);

        // 渲染模板 模板赋值
        return view('', compact('list'));
    }
}
```

- 模板文件

```
<ul>
    {volist name='list' id='user'}
        <li> {$user.username}</li>
    {/volist}
</ul>

<!-- 显示分页 -->
{$list|raw}
```

- 页面效果

分页没有样式的原因

默认使用的是bootstrap的分页

`{$list|raw}` 只是提供html标签和类名，并没有提供样式代码

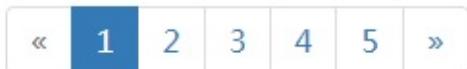
将bootstrap的分页样式引入，则可以显示bootstrap的分页效果

引入分页样式代码：css代码在本章节的同级章节 “[分页样式代码](#)”

- 张三
- 李四
- «
- 1
- 2
- 3
- 4
- 5
- »

### 引入 bootstrap 分页样式代码之后的效果

- 张三
- 李四



### 单独赋值分页输出的变量

```
$list = Db::table()->paginate(10);

// 渲染模板输出
return view('index', ['list' => $list]);

// 默认显示分页需要使用 :{$list|raw}

// 修改分页显示变量

// 获取分页显示
$page = $list->render();
// 显示分页则可以使用 :{$page|raw}
```

# 分页样式代码

## bootstrap 分页样式

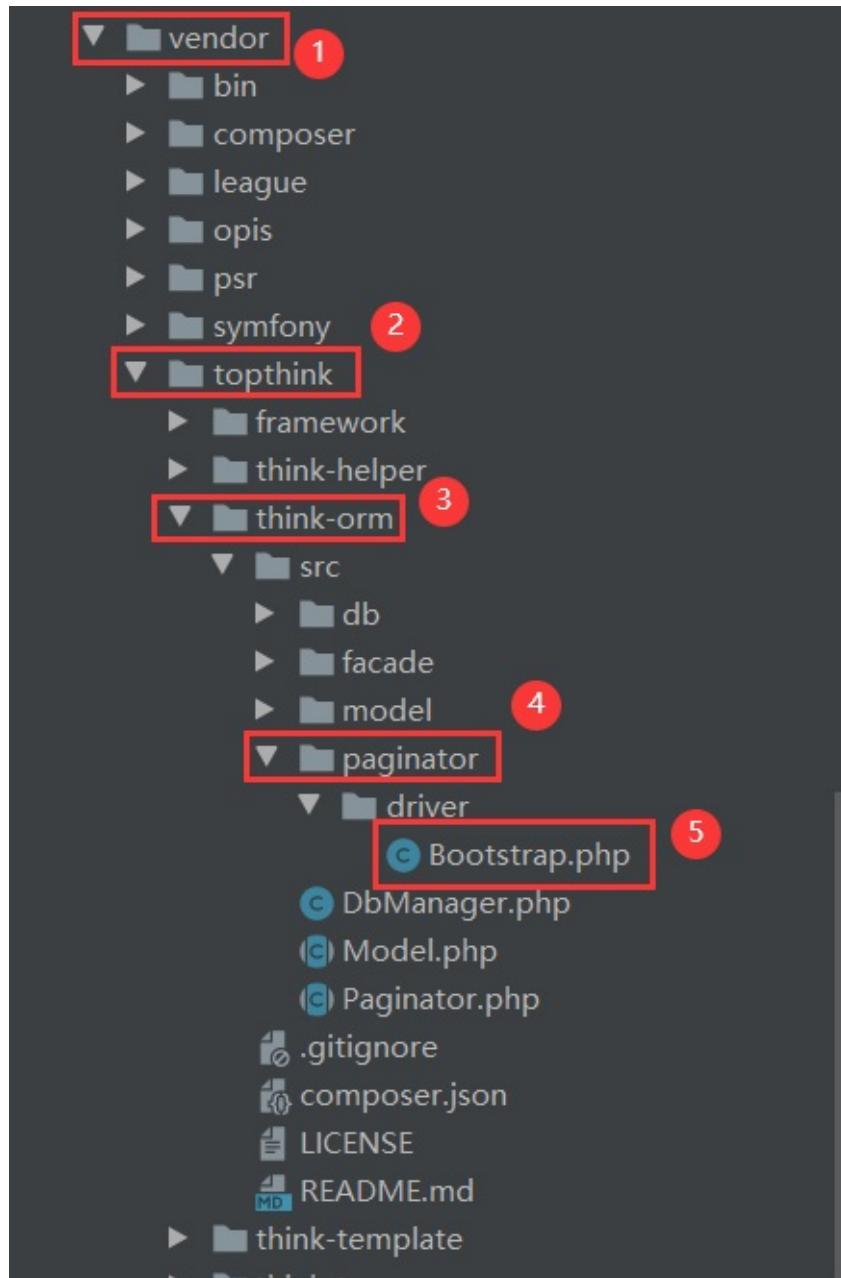
```
<style>
    .pagination {
        display: inline-block;
        padding-left: 0;
        margin: 20px 0;
        border-radius: 4px;
    }
    .pagination > li {
        display: inline;
    }
    .pagination > li > a,
    .pagination > li > span {
        position: relative;
        float: left;
        padding: 6px 12px;
        margin-left: -1px;
        line-height: 1.42857143;
        color: #337ab7;
        text-decoration: none;
        background-color: #fff;
        border: 1px solid #ddd;
    }
    .pagination > li:first-child > a,
    .pagination > li:first-child > span {
        margin-left: 0;
        border-top-left-radius: 4px;
        border-bottom-left-radius: 4px;
    }
    .pagination > li:last-child > a,
    .pagination > li:last-child > span {
        border-top-right-radius: 4px;
        border-bottom-right-radius: 4px;
    }
    .pagination > li > a:hover,
    .pagination > li > span:hover,
    .pagination > li > a:focus,
    .pagination > li > span:focus {
        z-index: 2;
        color: #23527c;
        background-color: #eee;
        border-color: #ddd;
    }
    .pagination > .active > a,
```

```
.pagination > .active > span,
.pagination > .active > a:hover,
.pagination > .active > span:hover,
.pagination > .active > a:focus,
.pagination > .active > span:focus {
    z-index: 3;
    color: #fff;
    cursor: default;
    background-color: #337ab7;
    border-color: #337ab7;
}
.pagination > .disabled > span,
.pagination > .disabled > span:hover,
.pagination > .disabled > span:focus,
.pagination > .disabled > a,
.pagination > .disabled > a:hover,
.pagination > .disabled > a:focus {
    color: #777;
    cursor: not-allowed;
    background-color: #fff;
    border-color: #ddd;
}
</style>
```

# 自定义分页驱动

复制默认分页驱动文件

```
vendor\topthink\think orm\src\paginator\driver\Bootstrap.php
```



粘贴到当前应用下(位置不固定)

```

1 <?php
2 // -----
3 // | ThinkPHP [ WE CAN DO IT JUST T
4 // +-----
5 // | Copyright (c) 2006~2019 http://
6 // +-----
7 // | Licensed ( http://www.apache.o
8 // +-----
9 // | Author: zhangyajun <448901948@
10 // +----- 修改为正确的命名空间 -----
11
12 namespace app\driver;
13
14 use think\Paginator;
15
16 /**
17 * Bootstrap 分页驱动
18 */
19 class MyPaginate extends Paginator
20 {
21 }

```

## 注册驱动

app/provider.php

```

1 <?php
2 use app\ExceptionHandle;
3 use app\Request;
4
5 // 容器Provider定义文件 ① 添加这一段内容
6 return [
7     'think\Request'      => Request::class,
8     'think\exception\Handle' => ExceptionHandle::class,
9
10    'think\Paginator'    => 'app\driver\MyPaginate'
11];

```

## 修改自定义的分页驱动

- 分页代码盒子

```

    /**
     * 渲染分页html
     * @return mixed
     */
    public function render()
    {
        if ($this->hasPages()) {
            if ($this->simple) {
                return sprintf(
                    format: '<ul class="pager">%s %s</ul>',
                    $this->getPreviousButton(),
                    $this->getNextButton()
                );
            } else {

                return sprintf(
                    format: '<nav class="navigation pagination" role="navigation"><div class="nav-links">%s %s %s</div></nav>',
                    $this->getPreviousButton(),
                    $this->getLinks(),
                    $this->getNextButton()
                );
            }
        }
    }

```

- 修改具体样式

```

    /**
     * 生成一个可点击的按钮
     * @param string $url
     * @param string $page
     * @return string
     */
    protected function getAvailablePageWrapper(string $url, string $page): string{...}

    /**
     * 生成一个禁用的按钮
     * @param string $text
     * @return string
     */
    protected function getDisabledTextWrapper(string $text): string{...}

    /**
     * 生成一个激活的按钮
     * @param string $text
     * @return string
     */

```

## 效果示例

上一页 1 **2** 3 4 5 6 7 8 ... 19 20 下一页

## 默认分页代码

```
<ul class="pagination">
    <li class="disabled"><span><</span></li>
    <li class="active"><span>1</span></li>
    <li><a href="/?page=2">2</a></li>
    <li><a href="/?page=2">>></a></li>
</ul>
```

# 模型

---

[定义](#)

[新增](#)

[删除](#)

[更新](#)

[查询](#)

[获取器](#)

[修改器](#)

[搜索器](#)

[只读字段](#)

[模型关联](#)

[自动时间戳](#)

[模型输出](#)

[模型事件](#)

[模型属性](#)

# 定义

## 模型定义

- 继承 `\think\Model` 类

```
<?php
namespace app\model;

use think\Model;

class User extends Model
{}
```

- 不想写 `use` 也可以这样写

```
<?php
namespace app\model;

class User extends \think\Model
{}
```

# 新增

---

简介

[save](#)

获取自增ID

[create](#)

# 简介

## 模型新增数据 和 数据库新增数据 区别

- 数据库的新增只是单纯的写入给定的数据
- 模型的数据写入会包含 修改器 、 自动完成 以及模型事件等环节

# Save

```
// 返回true 默认只会写入数据表中有的字段  
// $result = self::save(['username' => '张三', 'password' => 123456]);  
  
// 返回true allowField() 指定可以写入的字段  
// $result = self::allowField(['username', 'password'])->save(['username' => '张三', 'password' => 123456, 'age' => 20]);  
  
// 官方推荐：模型数据赋值之前就进行数据过滤  
$data = request()->only(['username', 'password']);  
$result = self::save($data);
```

# 获取自增ID

## 获取自增 ID

```
$model = new self;
$model->username = '张三';
$model->password = 123456;
$result = $model->save();
// 自增id
dump($model->id);
```

## 如果表的主键不是 id 字段

```
// 添加模型属性
protected $pk = 'aid'

$model = new self;
$model->username = '张三';
$model->password = 123456;
$result = $model->save();
// 自增aid
dump($model->aid);
```

# create

## create() 静态方法直接添加数据

- 会自动过滤非数据表字段

```
User::create([
    'username' => input('post.username', '', 'trim'),
    'password' => input('post.password', '', 'trim'),
])
```

- 字段没有默认值且插入一个null值会报错，捕获错误的发生

```
try {
    self::create($data);
} catch (\think\db\exception\PDOException $e) {
    halt($e->getMessage());
}
```

- 添加成功，返回模型数据对象

```
// 添加成功, 返回模型数据对象
// 模型数据对象中包含 $data 数据和自增的id值
$res = User::create($data);
```

# 删除

---

[简介](#)

[delete](#)

[destroy](#)

# 简介

## 模型删除

- `delete()` 用于查询后删除、可用于非主键条件删除
- `destroy()` 适用于根据主键直接删除

# delete

## delete() 删除当前模型数据

- 模型删除数据，可以再查询后删除数据
- 应用场景：数据存在则删除，不存在返回提示信息

```
/**
 * 删除分类逻辑处理
 * 200 删除成功 201 分类不存在
 */
public function delCategory()
{
    // 分类id
    $id = input('get.cid', 0, 'intval');

    // 根据主键查询数据
    $cate = self::find($id);

    // 判断分类是否存在
    if ($cate === null) {
        return $this->layuiRes(201, '删除失败, 分类不存在');
    }

    // 执行删除 返回 true
    $result = $cate->delete();

    // 删除成功
    return $this->layuiRes(200, '删除成功');
}
```

# destroy

## destroy() 静态方法根据主键直接删除

```
// 主键字段默认为id
// 如果不是id，需要添加模型属性$pk指定主键字段
protected $pk = 'cid';

// 删除一条数据
self::destroy(1);

// 根据主键批量删除
self::destroy([1, 2, 3]);
```

## 使用示例

```
// 删除主键为1的数据
User::destroy(1);

// 删除主键为1或2的数据
User::destroy([1, 2]);
```

## 特别注意

```
// 以下方式不是删除主键为1或2的数据，而仅仅删除主键为1的数据
User::destroy('1,2');

// 如果要删除多个主键值的数据，必须使用以下方式
User::destroy([1, 2, 3]);
```

## 返回值

- destroy() 方法返回 true 或 false

当参数传入空值：空字符串和空数组时返回 **false**，其他情况返回 **true**

# 更新

---

[静态方法直接更新](#)

# 静态方法直接更新

## update() 直接更新

```
// 返回模型对象  
// 三个参数依次为：更新的数据 更新条件 更新的字段  
self::update(array $data[, array $where[, array $field]]);  
  
// 更新的数据中包含主键使用该方式(主键就是更新条件)  
// self::update(array $data);  
self::update(['name' => 'thinkphp', 'id' => 1]);  
  
// 更新的数据中不包含主键使用该方式(第二个参数是更新条件)  
// self::update(array $data, array $where);  
self::update(['name' => 'thinkphp'], ['id' => 1]);  
  
// 需要限定更新的字段时使用该方式(第三个参数限定需要更新的字段)  
// self::update(array $data, array $where, array $field);  
self::update(['name' => 'thinkphp', 'email' => 'thinkphp@qq.com'], ['id' => 1],  
['name']);  
  
// 更新的数据包含主键 并指定更新的字段  
self::update(['name' => 'thinkphp', 'id' => 1], [], ['content', 'status']);
```

# 查询

---

[find](#)

# find

模型调用 find() 返回模型数据对象

```
<?php
namespace app\model;

use think\Model;

class User extends Model
{
    public function getUserInfo()
    {
        // 查询到数据返回模型数据对象,查不到数据返回NULL
        $data = self::find(10);

        halt($data);
    }
}
```

# 获取器

---

[应用场景](#)

[获取器定义](#)

[触发获取器](#)

[追加获取器](#)

[获取原始数据](#)

# 应用场景

## 获取器应用场景

- 状态值的转换
  - status 字段：1 启用 2 禁用
  - gender 字段：1 转为男 0 转为女

# 获取器定义

## 定义获取器

```
public function getSexAttr($value, $data)
{
    $status = [ '0' => '女', '1' => '男'];
    return $status[$value];
}
```

## 获取器方法名定义规则

字段名转为大驼峰命名法

```
// 字段名 : user
public function getUserAttr($value) {}

// 字段名 : nick_name
public function getNickNameAttr($value) {}
```

## 获取器的第二个参数

- 获取器的第二个参数就是当前行的数据数组

```
public function getSexAttr($value, $data)
{
    // $data 一维的关联数组 就是一行数据
    // 示例 ['id' => 1, 'name' => '张三']
    return $data;
}
```

# 触发获取器

## 模型的序列化输出

- `toArray()` 触发获取器

```
$data = self::where('id', 1)->find()->toArray();  
  
$data = self::where('id', 1)->select()->toArray();
```

## 模型数据对象取值

- 模型数据对象->字段名 触发获取器

```
$data = self::where('id', 1)->find();  
var_dump($data->nick_name);  
  
$data = self::where('id', 1)->select();  
foreach ($data as $v) {  
    //  
    var_dump($v->nick_name);  
}
```

# 追加获取器

## 追加获取器

- 模型查询默认只会自动触发 以字段名命名 的获取器方法
- 可以通过 `append()` 方法追加获取器 参考 [官方手册-模型输出](#)

特别注意：必须要首先调用一次Db类的方法后才能调用 `append` 方法。

- 以下方式虽然没有报错，但是追加无效，因为没有调用Db类的方法

```
$data = self::append(['abc'])->select();
```

- 解决方案：调用一个Db类的方法，使其追加有效

```
$data = self::where('id', '>', '0')  
->append(['abc'])->select();
```

# 获取原始数据

---

[getData](#)

[应用示例](#)

# getData

## 获取原始数据示例

- 通过模型数据对象的 `getData()` 方法获取原始数据

```
/**  
 * 获取原始数据示例  
 */  
public function showData()  
{  
    // toArray() 触发获取器  
    $data = self::where('id', 1)->select();  
  
    foreach ($data as $key => $value) {  
        // 获取原始数据  
        var_dump($value->getData('nick_name'));  
        // 获取器处理后的数据  
        var_dump($value->nick_name);  
    }  
}  
  
/**  
 * 定义nick_name字段的获取器  
 * @param {[type]} $value nick_name字段值  
 * @return {[type]}       经过获取器转换后的值  
 */  
public function getNickNameAttr($value)  
{  
    $status = [ '0' => '女', '1' => '男'];  
    return $status[$value];  
}
```

# 应用示例

## 获取原始数据的应用示例

- 获取器定义

```
/**
 * 获取器 性别
 * @param [type] $value [description]
 * @return [type] [description]
 */
public function getGenderAttr($value)
{
    $status = [0 => '女', 1 => '男', 2 => '保密'];
    return $status[$value];
}
```

- 单选框的选中判断（在模板文件中经常使用）

```
// 模型对象
$member = self::find($id);

{if $member->getData('gender') == 1} checked {/if}
{if $member->getData('gender') == 0} checked {/if}

<input type="radio" name="gender" value="1" {if $member->getData('gender') == 1} checked {/if}>
<input type="radio" name="gender" value="0" {if $member->getData('gender') == 0} checked {/if}>
```

- 下拉框的选中判断

```
<select name="rank">
    <option value="-1" {if $data->getData('rank') == -1} selected {/if}>私有</option>
    <option value="0" {if $data->getData('rank') == 0} selected {/if}>待审核</option>
    <option value="1" {if $data->getData('rank') == 1} selected {/if}>公开浏览</option>
</select>
```

# 修改器

---

[触发修改器](#)

# 触发修改器

添加

```
self::create($data);
```

更新

```
self::update($data, ['id' => $aid], $field);
```

# 搜索器

---

[应用场景](#)

[搜索器定义](#)

[使用示例](#)

[withSearch](#)

# 应用场景

## 搜索器

用于封装字段的查询表达式 [TP6.0 搜索器 官方文档](#)

### 搜索器应用场景

- 限制和规范表单的搜索条件
- 预定义查询条件简化查询

# 搜索器定义

## 搜索器定义规范

- 访问类型：public
- 方法名：searchFieldNameAttr
- `FieldName` 是数据表字段名的大驼峰命名转换

# 使用示例

## 使用示例

- 定义搜索器

```
// title 字段搜索器 模糊查询
public function searchTitleAttr($query, $value, $data)
{
    $query->where('title', 'like', '%' . $value . '%');
}

// create_time 字段搜索器 时间范围查询
public function searchCreateTimeAttr($query, $value, $data)
{
    $query->whereBetweenTime('create_time', $value[0], $value[1]);
}
```

- 在模型中执行使用搜索器的查询

```
// 字段限定 title create_time
$field = ['title', 'create_time'];

// 传入的数据
$data = [
    'name'        => 'think',
    'create_time' => ['2018-8-1', '2018-8-5'],
];

// 执行使用搜索器的查询
$res = self::withSearch($field, $data)->select();

// 实际执行的SQL语句
// SELECT * FROM `admin` WHERE `title` LIKE '%think%' AND `create_time` BETWEEN
// '2018-8-1' AND '2018-8-5'
```

# withSearch

## withSearch() 模型调用搜索器时使用

- \$field : 索引数组，键值是字段名，限定使用的获取器
- \$data : 关联数组，['键名' => '获取器的第三个参数的值']

```
withSearch(array $field, array $data)
```

## 搜索器方法的参数

```
// create_time 字段搜索器 时间范围查询
public function searchCreateTimeAttr($query, $value, $data)
{
    // $query
    // $query是一个对象,用于实现链式操作
    //
    // $value
    // 假设withSearch的第二参数为$arr
    // 则 $value = $arr['create_time']
    $query->whereBetweenTime('create_time', $value[0], $value[1]);
}
```

# 只读字段

---

[只读字段简介](#)

# 只读字段简介

## 只读字段

只读字段用来保护某些特殊的字段值不被更改，  
如：用户名，这个字段的值一旦写入，就无法更改

## 工作原理

- 当执行更新方法之前会自动过滤到只读字段的值，避免更新到数据库
- 只读字段仅针对模型的更新方法，如果使用数据库的更新方法则无效

# 模型关联

---

[一对关联](#)

[一对多关联](#)

[远程一对多关联](#)

[关联预载入](#)

[关联统计](#)

[关联输出](#)

[关联删除](#)

# 一对一关联

---

[数据表](#)

[基础使用](#)

[关联查询](#)

[hasOne](#)

[hasWhere](#)

[belongsTo](#)

[关联保存更新](#)

[预载入查询](#)

[绑定属性到父模型](#)

# 数据表

用于做示例的数据表

## 用户表

```
mysql> select * from admin;
+---+---+---+
| uid | tid | name   |
+---+---+---+
| 1   | 3   | admin  |
| 2   | 4   | zhangsan |
+---+---+---+
2 rows in set (0.00 sec)
```

## 用户信息表

```
mysql> select * from admin_info;
+---+---+---+---+
| aid | nickname           | money | phone |
+---+---+---+---+
| 1   | 淘气的松鼠          | 100.00 | NULL  |
| 2   | 进击的巨人          | 200.00 | NULL  |
| 3   | PHP是世界上最好的语言 | 300.00 | NULL  |
| 4   | TP6.0模型一对-关联  | 400.00 | NULL  |
+---+---+---+---+
4 rows in set (0.00 sec)
```

# 基础使用

## 一对一模型关联

一个用户 对应一个 个人资料 [一对一关联官方文档](#)

```
hasOne('关联模型类名' [, '外键'] [, '主键']);
```

- 关联模型类名，必填参数，指向要关联的数据表模型

```
/**
 * 在admin表模型中定义一对一模型关联
 * @return [type] [description]
 */
public function profile()
{
    return $this->hasOne(\app\model\AdminInfo::class);
}
```

- 外键：可选参数，用户资料表的字段名，此字段和用户表对应主键
- 外键省略时默认为当前模型名拼接上 `_id`
- 如：当前模型名为 admin，此时外键默认为 admin\_id

```
/**
 * 在admin表模型中定义一对一模型关联
 * @return [type] [description]
 */
public function profile()
{
    // 因为 admin_info 表和 admin 对应的字段不是 admin_id
    // 所以此时第二个参数必须指定为aid
    return $this->hasOne(\app\model\AdminInfo::class, 'aid');
}
```

- 主键：可选参数，用户表的字段名，和用户资料表的外键对应

```
/**
 * 在admin表模型中定义一对一模型关联
 * @return [type] [description]
 */
public function profile()
{
    // hasOne的三个参数
```

```
//  
// 第一个参数：必写参数，关联模型类名  
//  
// 第二个参数：可选参数，外键字段名  
// 因为 admin_info 表和 admin 对应的字段不是 admin_id  
// 所以此时第二个参数必须指定为aid  
//  
// 第三个参数：可选参数，主键字段名  
// 此时没有指定主键字段(第三个参数) 默认是模型属性$pk的值，  
// 而$pk属性如果没有手动指定，框架默认值为：id  
// 也就是说：如果主键不是$pk的属性值再填写该参数  
return $this->hasOne(\app\model\AdminInfo::class, 'aid', 'tid');  
}
```

# 关联查询

## 关联查询

- 定义好一对关联后，使用以下方式获取关联数据
- 模型数据对象 -> 关联方法名 -> 字段名

```
// PHP是世界上最好的语言  
dump(dump(self::find(1)->profile->nickname));
```

## 如果没有关联到数据

```
// 关联到数据返回外键表模型对象  
// 没有关联外键表数据返回 null  
self::find(1)->profile;
```

# hasOne

## hasOne() 一对关联

```
<?php
namespace app\model;

use think\Model;

/**
 * 用户表模型
 */
class User extends Model
{
    /**
     * 关联用户资料表模型
     */
    public function profile()
    {
        // 用户资料表的uid 对应用户表的主键id
        return $this->hasOne(Profile::class, 'uid');
    }
}
```

## 模型查询

```
// $id 用户id
$data = self::find($id);

// 没有关联到数据返回null
// 关联到数据返回用户资料表模型数据对象
$profile = $data->profile;
```

```
// $data 用户表数据集模型对象
$data = self::select();
foreach ($data as $value) {
    // $value 用户表数据模型对象
    // $profile null或用户资料表模型对象
    $profile = $value->profile;
}
```

# hasWhere

## hasWhere() 根据关联数据查询

- 根据关联条件来查询当前模型对象数据
- 根据用户资料表的字段的限定条件查询用户表数据

```
/**
 * 模型关联测试
 */
public function test()
{
    // $data 用户表数据集模型对象
    // hasWhere('关联方法名', ['用户资料表的字段' => '值'])
    // 相当于在关联方法的基础上新增一个关联条件：用户资料表mobile字段值为10086
    $data = self::hasWhere('profile', ['mobile' => '10086'])->select();

    foreach ($data as $value) {
        // 用户资料表模型
        dump($value->profile);
    }
}

/**
 * 关联用户资料表模型
 */
public function profile()
{
    // 用户资料表的uid 对应用户表的主键id
    return $this->hasOne(Profile::class, 'uid');
}
```

## 如果关联条件比较复杂，可以使用闭包查询

```
$name = 'PHP';
// 可以使用闭包查询
self::hasWhere('profile', function($query){
    global $name;
    $query->where('nickname', 'like', '%' . $name . '%');
})->select();
```

## 可以通过 `fetchSql(true)` 查看执行的SQL语句

```
$sql = self::haswhere('profile', function($query){  
    $query->where('mobile', 10086);  
})->fetchSql(true)->select();  
  
halt($sql);
```

# belongsTo

## belongsTo 相对关联

- 在用户资料表中可以定义相对关联
- 前面的关联方法都是定义在用户表模型，相对关联定义在用户资料表模型

```
public function ttt()
{
    $admin = self::find(1);

    // 用户表的 tid 字段
    dump($admin->admindd->tid);
}

/**
 * 相对关联 关联方法
 */
public function admindd()
{
    // belongsTo('关联模型类名', '外键', '主键')
    //
    // 第一个参数
    // \app\model\Admin::class 用户表模型类名
    //
    // 第二个参数
    // 'aid' 用户资料表外键字段, 也就是当前表的字段
    // 默认是当前模型名拼接上 _id, 而不是当前模型$pk的值
    //
    // 第三个参数
    // 'uid' 用户表主键字段 默认是 关联模型类(Admin用户表)的$pk属性值
    return $this->belongsTo(\app\model\Admin::class, 'aid');
}
```

## 应用场景

在用户资料表模型中想要获取对应的用户表的数据

# 关联保存更新

## 关联保存更新

```
// 如果还没有关联数据 则进行新增
$admin = self::select();
foreach ($admin as $value) {
    // 关联保存更新 返回json字符串
    // 没关联到外键表数据就给外键表添加数据
    // 关联到就更新外键表 nickname 字段的值
    $value->profile()->save(['nickname' => '关联自增']);
}
```

# 预载入查询

---

[with](#)

[withJoin](#)

# with

## 预载入查询

将外键表的数据查出来一起返回

### 示例

```
// 类似左外查询 用户表完全显示
dump(self::with('profile')->select()->toArray());
```

```
^ array:2 [▼
  0 => array:4 [▼
    "uid" => 1
    "tid" => 3
    "name" => "admin"
    "profile" => array:4 [▼
      "aid" => 1
      "nickname" => "淘气的松鼠"
      "money" => "100.00"
      "phone" => null
    ]
  ]
  1 => array:4 [▼
    "uid" => 2
    "tid" => 4
    "name" => "zhangsan"
    "profile" => array:4 [▼
      "aid" => 2
      "nickname" => "进击的巨人"
      "money" => "200.00"
      "phone" => null
    ]
  ]
]
```

### 如果要对关联模型进行约束，可以使用闭包的方式

```
$result = self::with(['profile' => function($query){
  $query->field('aid,nickname')->where('aid', '>', 0);
}])->select()->toArray();
```

# withJoin

需要使用 join 方式查询时使用

```
// withJoin('关联方法名', 'join类型')
// join类型 默认使用的是 INNER JOIN 内连接 可取值: left right
$result = self::withJoin('profile', 'left')->select()->toArray();

dump($result);
```

约束关联字段

```
$result = self::withJoin([
    // 外键表 用户资料表 显示的字段
    'profile' => ['aid', 'nickname']
], 'left')->select()->toArray();
```

# 绑定属性到父模型

---

[bind](#)

[属性别名](#)

[bindAttr](#)

[应用示例](#)

# bind

## bind() 绑定属性到父模型

- 配合 `预载入查询` 使用
- 在定义关联的使用`bind()`方法可以将属性绑定到父模型上
- 如：将用户资料表的属性绑定到用户表模型上

## 示例

### 关联方法

```
/**
 * profile 关联方法名
 */
public function profile()
{
    // \app\model\AdminInfo::class 关联模型类名
    // 'aid' 外键字段名
    // 'tid' 主键字段名
    return $this->hasOne(\app\model\AdminInfo::class, 'aid')
        ->bind(['nickname']);
}
```

### 绑定属性到父模型上

```
$admin = self::with('profile')->find(1);

dump($admin->toArray());

/*
array:4 [▼
    "uid" => 1
    "tid" => 3
    "username" => "112"
    "nickname" => "富商大贾"
]
*/
```

// 如果定义关联的时候没有bind方法

```
/*
array:4 [▼
    "uid" => 1
```

```
"tid" => 3
"username" => "112"
"profile" => array:4 [▼
    "aid" => 1
    "nickname" => "富商大贾"
    "money" => "10.00"
    "phone" => "123"
]
]
*/
```

### 考虑没有关联到数据的情况

- 在模板中可以这样使用 没有关联时 显示空字符串

```
{$v.nickname ?? ''}
```

# 属性别名

## 属性别名

- 给 bind() 传入关联数组就是给属性起别名
- 特别注意：键名才是新名

```
/**
 * profile 关联方法名
 */
public function profile()
{
    return $this->hasOne(\app\model\AdminInfo::class, 'aid')
        ->bind([
            // admin_name 新名 nickname 字段名
            // 新名在前 字段名在后
            // 这里不像sql起别名新名在后
            'admin_name' => 'nickname',
        ]);
}
```

# bindAttr

## bindAttr() 动态绑定关联属性

- bind() 用于在定义关联的时候使用
- 定义关联时没有使用bind()，则模型对象可以调用 bindAttr() 方法动态绑定关联属性
- 给键值加上键名就是起新名

```
$admin = self::with('profile')
    ->find(1)
    // bindAttr('关联方法名', ['用户资料表字段', '新名' => '用户资料表字段'])
    ->bindAttr('profile', ['nickname', 'new_phone' => 'phone']);
```

# 应用示例

## 绑定属性到父模型使用示例

```
// with() 预载入查询 bindAttr() 绑定属性到父模型
$data = self::with(['member', 'sentence'])
    ->page($page, $limit)
    ->select()
    ->bindAttr('member', ['member_username' => 'username'])
    ->bindAttr('sentence', ['juzi_content' => 'juzi']);
```

# 一对多关联

---

[基础使用](#)

[关联定义](#)

[关联查询](#)

# 基础使用

## 一对多模型关联

[一篇文章](#) [对应多个](#) [评论](#) [一对多关联官方文档](#)

# 关联定义

## hasMany() 一对多关联

在 文章模型 中定义

```
/**  
 * 一对多关联  
 *  
 * 文章对应多个评论  
 */  
public function comments()  
{  
    // hasMany('关联模型', '外键', '主键')  
    //  
    // 外键字段名 默认值：当前模型名拼接上 _id  
    //  
    // 主键字段名 默认值：$pk属性的值  
    return $this->hasMany(Comment::class, 'aid');  
}
```

# 关联查询

## 获取关联数据

```
public function test()
{
    $art = self::find(1);

    // $art->comments 模型对象 -> 关联方法名
    // 返回模型数据集对象,需要查询条件时可以此方式
    dump($art->comments);

    // 需要添加查询条件时就不要调用comments属性了
    // 调用关联方法 链式操作添加查询条件 c_id 是评论表的字段
    $result = $art->comments()->where('c_id', '>', 1)->select();
    dump($result);
}
```

# 基本查询

## 获取关联数据

```
public function test()
{
    $art = self::find(1);

    // $art->comments 模型对象 -> 关联方法名
    // 返回模型数据集对象,需要查询条件时可以此方式
    dump($art->comments);

    // 需要添加查询条件时就不要调用comments属性了
    // 调用关联方法 链式操作添加查询条件 c_id 是评论表的字段
    $result = $art->comments()->where('c_id', '>', 1)->select();
    dump($result);
}
```

# 关联条件查询

## 根据关联条件查询

- 根据关联条件来查询当前模型对象数据
- 例如：在文章模型中查询评论大于3条数据的文章

```
// 查询评论超过3个的文章  
// has('关联方法名', '操作符', '值')  
$list = self::has('comments', '>', 3)->select();
```

```
// 查询评论状态正常的文章  
// hasWhere('关联方法名', '操作符', '值') status 评论表字段  
$list = self::hasWhere('comments', ['status' => 1])->select();  
dump($list);
```

# 远程一对多关联

---

[简介](#)

[数据表](#)

[hasManyThrough](#)

# 简介

## 远程一对多关联

远程：指代 跨表

## 应用场景举例

- 每个城市有多个用户
- 每个用户有多个话题
- 城市和话题之间并无关联

可以通过远程一对多关联获取每个城市的多个话题

# 数据表

用于测试 远程一对多关联 的数据表

```
CREATE TABLE `city` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `city_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) COMMENT='城市表';

CREATE TABLE `user` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT COMMENT '用户id',
  `city_id` int(11) DEFAULT NULL,
  `username` varchar(255) NOT NULL COMMENT '用户名',
  `nickname` varchar(255) DEFAULT NULL COMMENT '昵称',
  `password` char(32) DEFAULT NULL COMMENT '登陆密码',
  PRIMARY KEY (`id`) USING BTREE
) COMMENT='用户表';

CREATE TABLE `topic` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `user_id` int(11) DEFAULT NULL,
  `title` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) COMMENT='话题表';
```

# hasManyThrough

## hasManyThrough() 方法参数

```
hasManyThrough('关联模型', '中间模型', '外键', '中间表关联键', '当前模型主键', '中间模型主键');
```

- 关联模型（必须）：关联模型类名
- 中间模型（必须）：中间模型类名
- 外键：中间表和当前表关联的外键字段名，默认的外键名规则是当前模型名+ `_id`
- 中间表关联键：远程表和中间表关联的外键字段名，默认的中间表关联键名的规则是中间模型名+ `_id`
- 当前模型主键字段名：一般会自动获取也可以指定传入
- 中间模型主键字段名：一般会自动获取也可以指定传入

# 关联预载入

---

[简介](#)

[数据表](#)

# 简介

## 什么是关联预载入

- 将关联的数据查出来一起返回

## 预载入

预载入：预先加载

将关联的数据预先加载到当前模型中

# 数据表

## 用户资料表

```
CREATE TABLE `profile` (
  `id` int(11) NOT NULL COMMENT '资料id',
  `uid` int(11) DEFAULT NULL COMMENT '用户id',
  `mobile` int(11) DEFAULT NULL COMMENT '手机号',
  `gender` tinyint(4) DEFAULT NULL COMMENT '性别 1 男 0 女',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='用户资料表';
```

# 关联统计

---

[withCount](#)

# withCount

## withCount() 关联数量统计

应用场景: 不需要获取关联数据, 只需要关联数据的统计

## 使用示例

```
^ array:2 [▼
  0 => array:4 [▼
    "id" => 1
    "title" => "张三的面"
    "content" => "全场八折"
    "comm_count" => 2
  ]
  1 => array:4 [▼
    "id" => 2
    "title" => "2"
    "content" => "3"
    "comm_count" => 1
  ]
]
```

```
<?php
namespace app\model;

/**
 * 文章模型
 */
class Article extends \think\Model
{
    public function test()
    {
        // self::withCount('关联方法名')->select();
        // self::withCount(['关联方法名' => '自定义属性名'])->select();
        $data = self::withCount(['comments' => 'comm_count'])->select();

        // 查看数据
        dump($data->toArray());
    }

    /**
     * 一对多关联 文章关联多个评论
     */
    public function comments()
    {
        return $this->hasMany(Comments::class, 'aid');
    }
}
```

}

## 关联预载入 和 关联统计 同时使用

```

^ array:2 [▼
0 => array:5 [▼
  "id" => 1
  "title" => "张三的面"
  "content" => "全场八折"
  "comm_count" => 2
  "comments" => array:2 [▼
    0 => array:3 [▼
      "id" => "1"
      "aid" => 1
      "remark" => "你说的对"
    ]
    1 => array:3 [▼
      "id" => "3"
      "aid" => 1
      "remark" => "真棒"
    ]
  ]
]
1 => array:5 [▼
  "id" => 2
  "title" => "2"
  "content" => "3"
  "comm_count" => 1
  "comments" => array:1 [▼
    0 => array:3 [▼
      "id" => "2"
      "aid" => 2
      "remark" => "厉害啊"
    ]
  ]
]
]

```

```

$data = self::with('comments')
->withCount(['comments' => 'comm_count'])
->select();

```

## 关联统计属性名

关联统计会自动添加一个以 `关联方法名 + _count` 为名称的属性保存到模型数据对象中

## 查询多个统计字段

```

// 统计一个关联数据
$data = self::withCount('comments')->select();
// 统计多个关联数据
$data = self::withCount(['comments', 'likes'])->select();

```

## 使用闭包的方式对关联统计进行条件过滤

```
$data = self::withCount(['comments' => function($query) {
    // comments评论表id大于3
    $query->where('id', '>', 3);

    // 没有使用闭包时 自定义关联统计属性名
    // withCount(['关联方法' => '自定义属性名'])

    // 官方手册：使用了闭包方式 自定义属性名的方法如下
    // 但是测试结果 定义属性名无效
    $alias = 'comm_count';
}, 'likes'])->select();
```

## 所有的关联统计方法都可以多次调用

```
$data = self::withCount('comments')
->withCount('likes')
->select();
```

# 关联输出

---

[hidden](#)

[visible](#)

[append](#)

# hidden

## hidden() 隐藏关联属性

- 预载入默认显示关联数据的所有字段

```
$data = self::with('comments')->select();
dump($data->toArray());
```

```
^ array:2 [▼
  0 => array:4 [▼
    "id" => 1
    "title" => "张三的面"
    "content" => "全场八折"
    "comments" => array:2 [▼
      0 => array:3 [▼
        "id" => "1"
        "aid" => 1
        "remark" => "你说的对"
      ]
      1 => array:3 [▼
        "id" => "3"
        "aid" => 1
        "remark" => "真棒"
      ]
    ]
  ]
  1 => array:4 [▼
    "id" => 2
    "title" => "2"
    "content" => "3"
    "comments" => array:1 [▼
      0 => array:3 [▼
        "id" => "2"
        "aid" => 2
        "remark" => "厉害啊"
      ]
    ]
  ]
]
```

隐藏关联数据中的 id

- 隐藏后的效果

```
$data = self::with('comments')
  ->hidden(['comments.id'])
  ->select();
dump($data->toArray());
```

```

^ array:2 [▼
  0 => array:4 [▼
    "id" => 1
    "title" => "张三的面"
    "content" => "全场八折"
    "comments" => array:2 [▼
      0 => array:2 [▼
        "aid" => 1
        "remark" => "你说的对"
      ]
      1 => array:2 [▼
        "aid" => 1
        "remark" => "真棒"
      ]
    ]
  ]
  1 => array:4 [▼ id 已经隐藏了
    "id" => 2
    "title" => "2"
    "content" => "3"
    "comments" => array:1 [▼
      0 => array:2 [▼
        "aid" => 2
        "remark" => "厉害啊"
      ]
    ]
  ]
]
]

```

- 隐藏多个关联数据的属性

```

$data = self::with('comments')
// 隐藏多个关联数据的属性
->hidden(['comments' => ['id', 'aid']])
->select();

dump($data->toArray());

```

```
^ array:2 [▼
  0 => array:4 [▼
    "id" => 1
    "title" => "张三的面"
    "content" => "全场八折"
    "comments" => array:2 [▼
      0 => array:1 [▼
        "remark" => "你说的对"
      ]
      1 => array:1 [▼
        "remark" => "真棒"
      ]
    ]
  ]
  1 => array:4 [▼
    "id" => 2
    "title" => "2"
    "content" => "3"
    "comments" => array:1 [▼
      0 => array:1 [▶]
    ]
  ]
]
```

# visible

## visible() 设定显示的关联数据属性

- 用法同 hidden()
- hidden() 设定隐藏的属性
- visible() 设定显示的属性

```
$data = self::with('comments')
    // visible 指定显示的属性
    // visible(['comments.aid']) // 单个属性
    // visible(['comments' => ['id', 'aid']]) // 多个属性
->visible(['comments' => ['id', 'aid']])
->select();
```

# append

## append() 追加关联属性

- 文章模型

```
<?php
namespace app\model;

use think\Model;

/**
 * 文章模型
 */
class Article extends Model
{
    public function test()
    {
        $data = self::with('comments')
            ->visible(['comments' => ['id', 'aid']])
            ->append(['likes.status'])
            ->select();

        dump($data->toArray());
    }

    public function comments()
    {
        return $this->hasMany(Comments::class, 'aid');
    }

    public function likes()
    {
        return $this->hasMany(Likes::class, 'aid');
    }
}
```

- Likes 模型

```
<?php
declare (strict_types = 1);

namespace app\model;

use think\Model;
```

```
/**
 * Likes模型
 */
class Likes extends Model
{
    public function getStatusAttr($value)
    {
        return 123;
    }
}
```

- likes 表没有 status 字段
- 如果关联到数据获取器会自动追加一个属性

```
^ array:2 [▼
  0 => array:5 [▼
    "id" => 1
    "title" => "张三的面"
    "content" => "全场八折"
    "comments" => array:2 [▶]
    "likes" => array:2 [▼
      0 => array:4 [▼
        "id" => 1
        "aid" => 1
        "likes" => 2
        "status" => 123
      ]
      1 => array:4 [▼
        "id" => 2
        "aid" => 1
        "likes" => 3
        "status" => 123
      ]
    ]
  ]
  1 => array:5 [▼
    "id" => 2
    "title" => "2"
    "content" => "3"
    "comments" => array:1 [▼
      0 => array:2 [▼
        "id" => "2"
        "aid" => 2
      ]
    ]
    "likes" => []
  ]
]
```

# 关联删除

## together() 关联删除

```
$article = Article::find(1);

// 删除文章的同时删除下面的评论
// 删除 comments 方法关联到的数据
// comments 一对多关联方法
$article->together(['comments'])->delete();

// 删除多个关联方法关联的数据
// comments、likes 一对多关联方法
$article->together(['comments', 'likes'])->delete();
```

## 注意事项

- together() 是模型数据对象的方法，不是数据集对象的方法

```
$data = self::with(['topics'])->select();

foreach ($data as $v) {
    $v->together(['topics'])->delete();
}
```

- 远程一对多关联

```
/**
 * hasManyThrough() 远程一对多关联
 */
public function topics()
{
    // hasManyThrough('远程表模型类', '中间表模型类', '中间表外键字段', '远程表外键字段')
);
    return $this->hasManyThrough(Topic::class, User::class, 'city_id', 'user_id')
;
}
```

# 自动时间戳

---

[配置](#)

[触发](#)

[写入格式](#)

[自定义字段](#)

# 配置

自定时间戳在 `数据库配置文件` 中定义

- 自动时间戳官方手册

`config/database.php`

```
<?php

return [
    // 默认使用的数据库连接配置
    'default'          => env('database.driver', 'mysql'),

    // 自定义时间查询规则
    'time_query_rule' => [],

    // 自动写入时间戳字段
    // true为自动识别类型 false关闭
    // 字符串则明确指定时间字段类型 支持 int timestamp datetime date
    'auto_timestamp'   => true, // 自动写入时间戳字段

    // 时间字段取出后的默认时间格式
    'datetime_format' => 'Y-m-d H:i:s',
]
```

官方手册上说自动时间戳默认关闭？

- 自动时间戳官方手册上说是 `默认关闭` 的
- 但是在全局数据库配置文件中默认是 `true` (开启)
- 如果是我理解的不到位，可在本章节下方留下评论

系统支持自动写入创建和更新的时间戳字段（默认关闭），有两种方式配置支持。

第一种方式是全局开启，在数据库配置文件中进行设置：

```
// 开启自动写入时间戳字段  
'auto_timestamp' => true,
```

第二种是在需要的模型类里面单独开启：

### 自动时间戳的两种配置支持

- 在配置文件中定义 `auto_timestamp` 配置项
  - 如上图所示：在 config/database.php 中修改
- 在模型类中定义 `protected $autoWriteTimestamp` 属性

```
// 开启自动时间戳  
// 自动识别: true 指定字段类型: int timestamp datetime date  
protected $autoWriteTimestamp = true;  
  
// 关闭自动时间戳  
protected $autoWriteTimestamp = false;
```

# 触发

## 触发自动时间戳

触发条件：使用模型方法添加和更新数据才会触发时间戳

### 触发自动时间戳的举例

- 以下都是模型方法添加或更新数据，所以会触发自动时间戳

```
// 添加数据 给添加时间和更新时间两个字段同时写入数据
self::save(['username' => '张三', 'password' => 123456]);

// 更新数据 更新update_time字段
self::update($data, ['id' => $aid], $field);
```

### 不会触发自动时间戳的举例

- 以下方式不会触发自动时间戳，因为不是模型方法

```
Db::table('admin')->insert(['username' => '张三', 'password' => 123456])
```

# 写入格式

## 自动时间写入格式

```
// 自动识别: true  
// 指定字段类型: int timestamp datetime date  
protected $autoWriteTimestamp = true;  
  
// 不管字段类型是什么, 都写入 2020-04-09 00:06:53 格式的数据  
// 如果字段类型是 int, 就会报错  
protected $autoWriteTimestamp = 'datetime';
```

- true: 自动识别字段类型, 根据不同的字段类型写入不同格式数据
  - 字段类型是 int, 则写入时间戳
  - 字段类型是datetime, 则写入如 2020-04-09 00:06:53 的日期格式

# 自定义字段

## 开启自动时间戳

```
// 自动识别: true (常用)
// 指定字段类型: int timestamp datetime date
protected $autoWriteTimestamp = true;

// 关闭自动时间戳
protected $autoWriteTimestamp = false;
```

## 自定义自动时间戳字段名

```
// 添加时间字段名
protected $createTime = 'create_at';

// 更新时间字段名
protected $updateTime = 'update_at';
```

## 单独关闭某个字段

```
// 关闭添加时间字段的自动写入
protected $createTime = false;

// 关闭更新时间字段的自动写入
protected $updateTime = false;
```

# 模型输出

---

[模板输出](#)

[数组转换](#)

# 模板输出

## 模式数据对象数组输出

- 模型数据对象通过数组键名访问数据

```
$data = self::find(1);  
  
halt($data['city_name']); // 输出结果：北京
```

- 所以在模板文件中可以使用以下方式输出模型属性

```
// 模板赋值  
return view('', ['data' => $data]);  
  
// 模板文件  
{$data.city_name}  
  
// 模板文件编译后  
<?php echo htmlentities($data['city_name']); ?>
```

# 数组转换

## 设置不输出的字段属性

- `hidden()` 指定不输出模型数据对象的属性

```
$data = self::find(1);

$array = $data->hidden(['id', 'city_name'])->toArray();
```

- `visible()` 指定输出的模型数据对象属性

```
$data = self::find(1);

$array = $data->visible(['id', 'city_name'])->toArray();
```

## 追加获取器定义

- 追加 `status_text` 字段获取器

```
$data = self::select();

$array = $data->append(['status_text'])->toArray();

halt($array);
```

- 添加获取器方法

```
/**
 * 追加的获取器定义
 */
public function getStatusTextAttr($value)
{
    return true;
}
```

# 模型事件

## 删除后执行

- 应用场景：删除文章后删除文章的缩略图和内容中的图片

```
public static function onAfterDelete($user)
{
}
```

# 模型属性

## 常用模型属性

属性	描述
\$connection	指定数据库连接配置信息
\$table	指定当前模型对应的数据表
\$pk	指定数据表主键字段，默认是 id
\$connection	指定数据库连接配置信息
\$autoWriteTimestamp	自动时间戳的开关、字段类型设定
\$createTime	指定添加时间字段名、是否开启写入
\$updateTime	指定更新时间字段名、是否开启写入

## 使用示例

```
/**
 * 指定主键字段
 * $pk : primary key 主键的缩写
 * 主键字段默认为id
 * 如果不是id, 可以添加模型属性$pk指定主键字段
 */
protected $pk = 'cid';

/**
 * 自动时间戳设定
 * false 关闭 true 自动识别
 * 指定字段类型: int timestamp datetime date
 */
protected $autoWriteTimestamp = false;

/**
 * 自定义添加时间字段名、更新时间字段名
 * 当取值为false时, 代表关闭指定字段的自动写入
 */
protected $createTime = 'create_at';
protected $updateTime = 'update_at';
```

# 视图

- 视图模板引擎驱动
- 视图助手函数 view()
  - view() 函数封装文件位置
- 视图目录及优先级

## 视图模板引擎驱动

默认的视图仅支持PHP原生模板

```
composer require topthink/think-view
```

```
[0] InvalidArgumentException in Manager.php line 103
Driver [Think] not supported.

94. {
95.     if ($this->namespace || false !== strpos($type, '\\')) {
96.         $class = false !== strpos($type, '\\') ? $type : $this->namespace . Str::studly($type);
97.
98.         if (class_exists($class)) {
99.             return $class;
100.        }
101.    }
102.
103.    throw new InvalidArgumentException("Driver [$type] not supported.");
104.
105.
106. /**
107. * 获取驱动参数
108. * @param $name
109. * @return array
110. */
111. protected function resolveParams($name): array
112. {
```

```
C:\WINDOWS\System32\WindowsPowerShell\v1.0\powershell.exe
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。
尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS E:\www\thinkphp\tp601> composer require tophink/think-view
Using version 1.0 for tophink/think-view
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 2 installs, 0 updates, 0 removals
- Installing tophink/think-template (v2.0.7): Downloading (100%)
- Installing tophink/think-view (v1.0.13): Downloading (100%)
Writing lock file
Generating autoload files
> @php think service:discover
Succeed!
> @php think vendor:publish
File E:\www\thinkphp\tp601\config\trace.php exist!
Succeed!
PS E:\www\thinkphp\tp601>
```

## 视图助手函数 view()

无需引入任何类 可以直接使用

```
return view();
return view('index');
return view('index/index');
return view('index/index/index');
```

## view() 函数封装文件位置

vendor/topthink/framework/src/helper.php 551行

```
if (!function_exists('view')) {
    /**
     * 渲染模板输出
     * @param string $template 模板文件
     * @param array $vars 模板变量
     * @param int $code 状态码
     * @param callable $filter 内容过滤
     * @return \think\response\View
    */
    function view(string $template = '', $vars = [], $code = 200, $filter = null
): View
{
    return Response::create($template, 'view', $code)->assign($vars)->filter
    ($filter);
}
}
```



## 视图目录及优先级

`view()` / `view('方法名')`  
默认路径 : view/模块名/控制器名/方法名.html

`view('/index');` 视图根目录下的直接子文件  
路径 : view/index.html

```
public function index()
{
    return view();
}
```

# 视图扩展

## 视图扩展 tophink/think-view

- 在 ThinkPHP5.\*系列中，不需要引入视图扩展
- 在 ThinkPHP6.0中要渲染模板必须先引入视图扩展

### 引入视图扩展

```
composer require tophink/think-view
```

常用错误：没有引入视图扩展无法进行模板渲染

```
return view();
return \think\facade\View::fetch();
```

- 如果使用以上方式渲染模板，提示以下错误信息，是因为没有引入视图扩展
- 如果提示的是 `页面错误` 是因为没有开启调试模式：参考 [调试模式](#) 使其显示错误信息

← → C ⓘ 不安全 | tp6.0.cy

#0 [0]InvalidArgumentException in Manager.php line 103

Driver [Think] not supported.



```

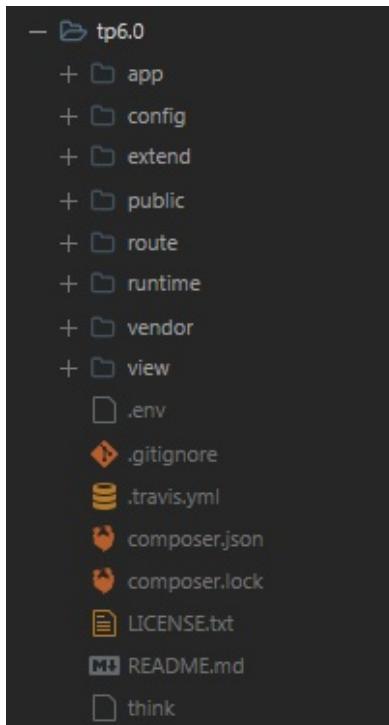
94.     {
95.         if ($this->namespace || false !== strpos($type, '\\')) {
96.             $class = false !== strpos($type, '\\') ? $type : $this->namespace . Str::studly($type);
97.
98.             if (class_exists($class)) {
99.                 return $class;
100.            }
101.        }
102.
103.        throw new InvalidArgumentException("Driver [$type] not supported.");
104.    }
105.
106. /**
107. * 获取驱动参数
108. * @param $name
109. * @return array
110. */
111. protected function resolveParams($name): array
112. {

```

## 引入 `topthink/think-view` 视图扩展

解决方法：在应用根目录下执行引入视图的composer命令

- 目录结构，特别注意：在应用根目录（我的应用根目录是tp6.0）执行该命令



- 执行引入视图扩展的命令

```
composer require topthink/think-view
```

```
管理员: C:\Windows\System32\cmd.exe
Microsoft Windows [版版本本 6.1.7601]
版版权权所所有有 <c> 2009 Microsoft Corporation。保保留所所有有权利利。。
D:\phpstudy-v8.1\phpstudy_pro\WWW\tp6.0>composer require topthink/think-view
Using version ^1.0 for topthink/think-view
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 2 installs, 0 updates, 0 removals
  - Installing topthink/think-template (<v2.0.7>): Loading from cache
  - Installing topthink/think-view (<v1.0.14>): Loading from cache
Writing lock file
Generating autoload files
> @php think service:discover
Succeed!
> @php think vendor:publish
File D:\phpstudy-v8.1\phpstudy_pro\WWW\tp6.0\config\trace.php exist?
Succeed!

D:\phpstudy-v8.1\phpstudy_pro\WWW\tp6.0>
```

# 模板赋值

- 使用View类的 `assign()` 静态方法 进行模板赋值
- 使用View类的 `fetch()` 静态方法 进行模板赋值

## 使用View类的 `assign()` 静态方法 进行模板赋值

```
// 引入View类
use think\facade\View;

public function index()
{
    // 单个赋值
    View::assign('name', "张三");

    // 批量赋值
    View::assign([
        'author' => 'liang',
        'email'   => '23426945@qq.com',
    ]);

    // 模板渲染
    return View::fetch();
}
```

## 使用View类的 `fetch()` 静态方法 进行模板赋值

```
// 引入View类
use think\facade\View;

public function index()
{
    // 模板输出并模板赋值
    return View::fetch('', [
        'name' => '张三',
        'pass' => '123456',
    ]);
}
```

# 视图过滤

- 视图过滤
- [View类的视图过滤](#)
- [view\(\) 助手函数是视图过滤](#)

## 视图过滤

对视图的渲染输出进行过滤

将视图中的指定内容进行替换

## View类的视图过滤

以下方法会将模板文件中的 `\r\n` 替换成 `<br/>`

```
// 使用视图输出过滤
return View::filter(function($content){
    return str_replace("\r\n", '<br>', $content);
})->fetch();
```

## view() 助手函数是视图过滤

```
// 使用视图输出过滤
return view()->filter(function($content){
    return str_replace("\r\n", '<br>', $content);
});
```

# 视图目录

## 视图目录定位

- 关键点：当前应用下 view 目录是否存在
- 优先定位当前应用下的 view 目录
- 如果应用下的 view 目录不存在，则定位根目录下的 view 目录中的模板文件

## 单应用模式的两种视图目录路径

视图目录和控制器目录、模型目录同级

```
|-- app
  |-- view (视图目录)
  |   |-- index           index控制器目录
  |   |   |-- index.html    index模板文件
  |   |   |-- ...
  |   |
  |   |-- ...             更多控制器目录
```

视图目录在根目录下 view 目录是根目录的直接子目录

```
|-- view                视图文件目录
  |-- index              index控制器目录
  |   |-- index.html      index模板文件
  |   |-- ...
  |
  |-- ...                更多控制器目录
```

## 自定义视图目录名称

在全局配置文件 view.php 中自定义视图目录名

```
config/view.php     view_dir_name
```

The screenshot shows a file explorer on the left and a code editor on the right. The file explorer displays the directory structure of an application:

- app
  - + controller
  - .htaccess
  - BaseController.php
  - common.php
  - event.php
  - ExceptionHandle.php
  - middleware.php
  - provider.php
  - Request.php
- config
  - app.php
  - cache.php
  - console.php
  - cookie.php
  - database.php
  - filesystem.php
  - lang.php
  - log.php
  - mail.php
  - middleware.php
  - route.php
  - session.php
  - trace.php
- + extend
- + public

The code editor window title is "view.php". The code content is as follows:

```
1 <?php
2 // -----
3 // | 模板设置
4 // -----
5
6 return [
7     // 模板引擎类型使用Think
8     'type' => 'Think',
9     // 默认模板渲染规则 1 解析为小写+下划线 2 全部转
10    'auto_rule' => 1,
11    // 模板目录名
12    'view_dir_name' => 'view',
13    // 模板后缀
14    'view_suffix' => 'html',
15    // 模板文件名分隔符
16    'view_depr' => DIRECTORY_SEPARATOR,
17    // 模板引擎普通标签开始标记
18    'tpl_begin' => '{',
19    // 模板引擎普通标签结束标记
20    'tpl_end' => '}',
21    // 标签库标签开始标记
22    'taglib_begin' => '{',
23    // 标签库标签结束标记
24    'taglib_end' => '}',
25];
26
```

The line 'view\_dir\_name' => 'view' is highlighted with a red box.

# 模板渲染

## 修改模板渲染规则

在全局配置文件view.php中修改模板渲染规则

```
config/view.php    auto_rule
```

1 解析为小写+下划线 【默认规则】

2 全部转换小写

3 保持操作方法

```

1 <?php
2 // -----
3 // | 模板设置
4 // -----
5
6 return [
7     // 模板引擎类型使用Think
8     'type'      => 'Think',
9     // 默认模板渲染规则 1 解析为小写+下划线 2 全部转换小写 3 保持操作方法
10    'auto_rule'   => 1,
11    // 模板目录名
12    'view_dir_name' => 'view',
13    // 模板后缀
14    'view_suffix'   => 'html',
15    // 模板文件名分隔符
16    'view_depr'     => DIRECTORY_SEPARATOR,
17    // 模板引擎普通标签开始标记
18    'tpl_begin'     => '{',
19    // 模板引擎普通标签结束标记
20    'tpl_end'       => '}',
21    // 标签库标签开始标记
22    'taglib_begin'  => '{',
23    // 标签库标签结束标记
24    'taglib_end'    => '}';
25];
26

```

## View::display() 渲染内容

如果希望直接解析内容而不通过模板文件的话，可以使用 `display` 方法

也就是直接输出内容 不渲染模板文件 类似于 `echo` 输出语句

```
// 直接渲染内容
$content = "{$name}-{$email}";
return View::display($content, ['name' => 'thinkphp', 'email' => 'thinkphp@qq.com']);
```

# 条件判断标签

## 选中判断

```
<input type="radio" name="status" {if condition="$roleInfo.status == 0"} checked{/if} value="0">
```

## 应用场景

下拉选择框 判断选中的选项

- \$categories 二维数组，分类选项
- \$data 一维数组，根据id查询出的数据

```
<select name="cate_id" id="cate_id" class="form-control">  
    <option value="">选择分类</option>  
    {volist name="categories" id="v"}  
        <option value="{{$v.id}}" {if $v.id == $data.cate_id} selected{/if}>{{$v.catename}}</option>  
    {/volist}  
</select>
```

# 模板输出使用函数

模板输出使用函数进行过滤或其它处理的时候，可以使用

- | 左右两边的空格可有可无

```
{$data.name | md5}
```

## 应用场景

将时间戳格式化为日期时间格式

```
{$data.create_time|date='Y-m-d H:i:s'}
```

假设 \$data.content 是ueditor编辑器内容，使用下面函数才能正常展示

```
{$data.content|raw}
```

# 调试

---

[调试模式](#)

[变量调试](#)

# 调试模式

## 调试模式和部署模式

- 调试模式：当有错误发生时可以看到具体的错误信息
- 部署模式(默认)：当有错误发生时只会提示页面有错误，看不到具体的错误信息
- 应用默认是部署模式,可修改环境变量APP\_DEBUG开启调试模式

## 开启调试模式

重命名应用根目录下的 `.example.env` 为 `.env` 即可

因为 `.example.env` 中的 APP\_DEBUG 默认就是 true

```
APP_DEBUG = true

[APP]
DEFAULT_TIMEZONE = Asia/Shanghai

[DATABASE]
TYPE = mysql
HOSTNAME = 127.0.0.1
DATABASE = test
USERNAME = username
PASSWORD = password
HOSTPORT = 3306
CHARSET = utf8
DEBUG = true

[LANG]
default_lang = zh-cn
```

## 部署模式下仍然希望看到具体的错误信息

修改配置文件: config/app.php

```
// 部署模式下显示错误信息
'show_error_msg' => true,
```

# 变量调试

## 变量调试

### 变量调试官方手册

TP6.0 引入了第三方扩展 `symfony/var-dumper` 优雅的PHP调试扩展库

`dump()` 的输出效果正是来自该扩展库

输出/打印某个变量，TP6 推荐使用 `dump()` `halt()`

用法和PHP内置的 `var_dump` 一致：

```
// 打印一个或多个变量
dump($var1, ...$varN)
```

`dump()` 使用示例

```
$data = [
    'name' => '张三',
    'age'  => 20
];

$address = ['北京', '上海', '广州'];

dump($data, $address);
```

```
^ array:2 [▼
  "name" => "张三"
  "age"  => 20
]
```

```
^ array:3 [▼
  0 => "北京"
  1 => "上海"
  2 => "广州"
]
```

调试变量并中止程序的执行

`halt()` 用法和 `dump()` 相同 只是在输出变量后会终止程序的执行

# 验证

---

[生成验证器类](#)

[验证器类成员](#)

[内置验证规则](#)

[执行数据验证](#)

# 生成验证器类

ThinkPHP6.0 支持通过指令生成验证器类

## 在app目录下生成验证器类

app/validate/User.php

```
php think make:validate User
```

- app/validate/admin/User.php

```
php think make:validate admin/User
```

## 在应用目录下生成验证器类

- app/admin/validate/User.php

```
php think make:validate admin@User
```

# 验证器类成员

---

[验证规则](#)

[错误信息](#)

[验证场景](#)

[应用示例](#)

# 验证规则

以下两种方式等价

```
protected $rule = [  
    'username' => 'require|min:3',  
];
```

```
protected $rule = [  
    'username' => [  
        'require',  
        'min' => 3,  
    ],  
];
```

# 错误信息

## 自定义验证错误信息

- 没有定义错误信息，框架有默认错误信息

```
/**
 * 定义错误信息
 * 格式：'字段名.规则名' => '错误信息'
 *
 * @var array
 */
protected $message = [
    'username.require' => '用户名不能为空',
    'password.require' => '登录密码不能为空',
    'file.fileSize'     => '文件太大',
    'file.fileExt'      => '不支持的文件后缀',
];
```

# 验证场景

## 定义验证场景

```
/**  
 * 验证场景定义  
 * 只验证password字段的数据  
 */  
public function sceneAdd()  
{  
    return $this->only(['password']);  
}
```

# 应用示例

## 验证器类成员

```
<?php
declare (strict_types = 1);

namespace app\validate;

use think\Validate;

class User extends Validate
{
    /**
     * 定义验证规则
     * 格式：'字段名' => ['规则1', '规则2'...]
     *
     * @var array
     */
    protected $rule = [];

    /**
     * 定义错误信息
     * 格式：'字段名.规则名' => '错误信息'
     *
     * @var array
     */
    protected $message = [];

    /**
     * 定义验证场景
     * 格式：'场景名' => '字段名'
     *
     * @var array
     */
    protected $scene = [
        'add' => ['title'],
    ];

    // edit 验证场景定义
    public function sceneEdit()
    {
        return $this->only(['name', 'age'])
            ->append('name', 'min:5')
            ->remove('age', 'between')
            ->append('age', 'require|max:100');
    }
}
```

## 应用示例

}

# 内置验证规则

---

`require`

`file`

# require

require 某个字段是必须的

```
'name' => 'require'
```

require 验证原理

```
// $value 是name的键值
// $result 是true验证通过 false验证失败
$result = !empty($value) || '0' == $value;
```

require 特别注意

如果验证规则没有添加require就表示没有值的话不进行验证

```
// 如果name的值是空(除'0'外),不执行min验证
// 如果name的值不是空(除'0'外),才会执行min验证
'name' => 'min:3'
```

require 验证 框架源码

vendor/topthink/framework/src/think/Validate.php

822行的 is方法() 中的 827行

```
/**
 * 验证字段值是否为有效格式
 * @access public
 * @param mixed $value 字段值
 * @param string $rule 验证规则
 * @param array $data 数据
 * @return bool
 */
public function is($value, string $rule, array $data = []): bool
{
    switch (Str::camel($rule)) {
        case 'require':
            // 必须
            $result = !empty($value) || '0' == $value;
            break;
    }
}
```

require

```
case 'accepted':
    // 接受
    $result = in_array($value, ['1', 'on', 'yes']);
    break;
case 'date':
    // 是否是一个有效日期
    $result = false !== strtotime($value);
    break;
case 'activeUrl':
    // 是否为有效的网址
    $result = checkdnsrr($value);
    break;
case 'boolean':
case 'bool':
    // 是否为布尔值
    $result = in_array($value, [true, false, 0, 1, '0', '1'], true);
    break;
case 'number':
    $result = ctype_digit((string) $value);
    break;
case 'alphaNum':
    $result = ctype_alnum($value);
    break;
case 'array':
    // 是否为数组
    $result = is_array($value);
    break;
case 'file':
    $result = $value instanceof File;
    break;
case 'image':
    $result = $value instanceof File && in_array($this->getImageType($value->getRealPath()), [1, 2, 3, 6]);
    break;
case 'token':
    $result = $this->token($value, '__token__', $data);
    break;
default:
    if (isset($this->type[$rule])) {
        // 注册的验证规则
        $result = call_user_func_array($this->type[$rule], [$value]);
    } elseif (function_exists('ctype_' . $rule)) {
        // ctype验证规则
        $ctypeFun = 'ctype_' . $rule;
        $result = $ctypeFun($value);
    } elseif (isset($this->filter[$rule])) {
        // Filter_var验证规则
        $result = $this->filter($value, $this->filter[$rule]);
    } else {
        // 正则验证
    }
}
```

```
require
```

```
    $result = $this->regex($value, $rule);  
}  
}  
  
return $result;  
}
```

# file

```
/**
 * 定义验证规则
 * 格式：'字段名' => ['规则1', '规则2'...]
 *
 * @var array
 */
protected $rule = [
    'file' => [
        // 限制文件大小(单位b), 这里限制为4M
        'fileSize' => 4 * 1024 * 1024,
        // 限制文件后缀, 多个后缀以英文逗号分割
        'fileExt' => 'gif,png,txt',
        // 验证是不是图片文件
        'image',
        // 验证 mime 类型
        'fileMime' => 'image/jpeg,image/png',
    ],
];

/**
 * 定义错误信息
 * 格式：'字段名.规则名' => '错误信息'
 *
 * @var array
 */
protected $message = [
    'file.fileSize' => '文件太大',
    'file.fileExt' => '不支持的文件后缀',
    'file.image' => '不是有效的图片文件',
    'file.fileMime' => '不允许该文件MIME类型',
];
```

# 执行数据验证

## 基础控制器数据验证

- 参考：控制器 - 基础控制器 - 数据验证功能 [传送门](#)

继承基础控制器的控制器可以使用 `$this->validate()` 进行数据验证

## validate() 助手函数数据验证

- 参考：助手函数 - validate [传送门](#)

# 杂项

---

[Session](#)

[Cookie](#)

[文件上传](#)

[layui](#)

[layuiAdmin单页版](#)

[layuiAdmin iframe版](#)

[经验分享](#)

# Session

---

[默认状态](#)

[开启Session](#)

[Session初始化](#)

[Session基础用法](#)

# 默认状态

## 默认没有开启 Session

**index()**方法：

没有开启Session时 也可以将数据存入Session

**test()**方法：

没有开启Session时 数据可以存入Session 但不能跨方法传递

```
<?php
namespace app\admin\controller;

use app\BaseController;

class Index extends BaseController
{
    public function index()
    {
        session('name', '张三');

        var_dump(session('name'));//123
    }

    public function test()
    {
        var_dump(session('name'));//NULL
    }
}
```

# 开启Session

Session 功能默认是没有开启的

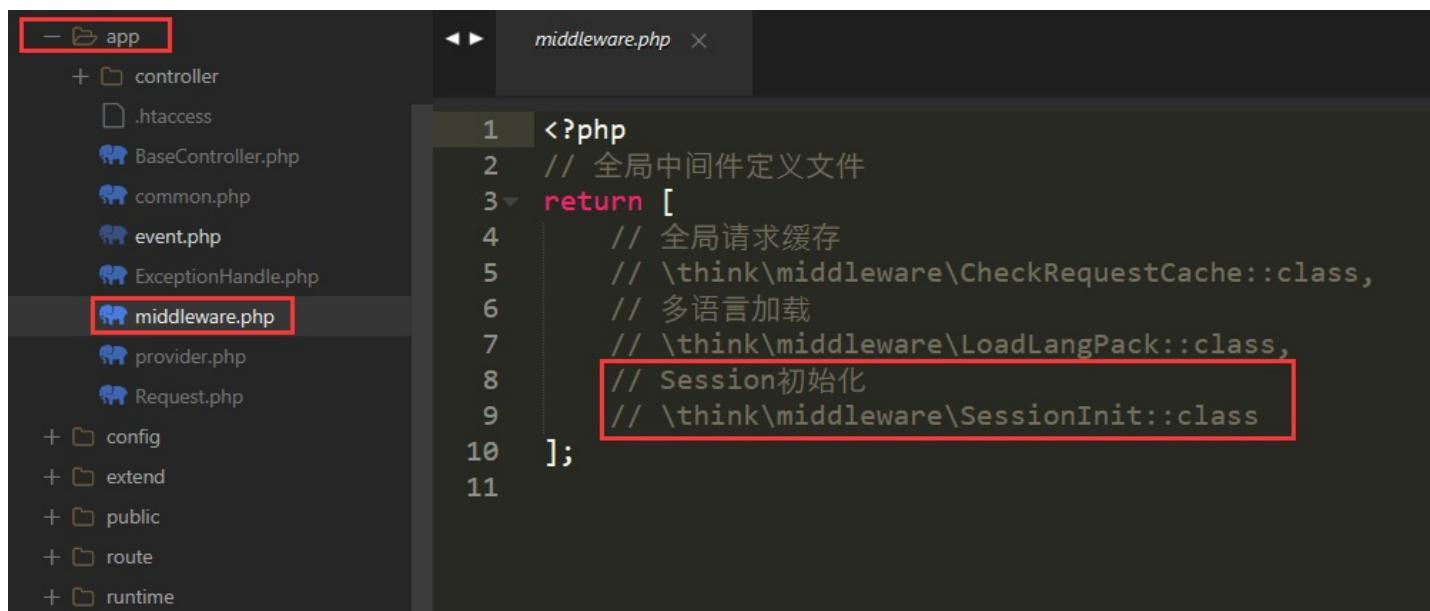
原因：API应用通常不需要使用 Session

开启Session

在全局中间件定义文件中加上下面的中间件定义(把这行注释打开就行了)

think\middleware\SessionInit

全局中间件定义文件：app/middleware.php



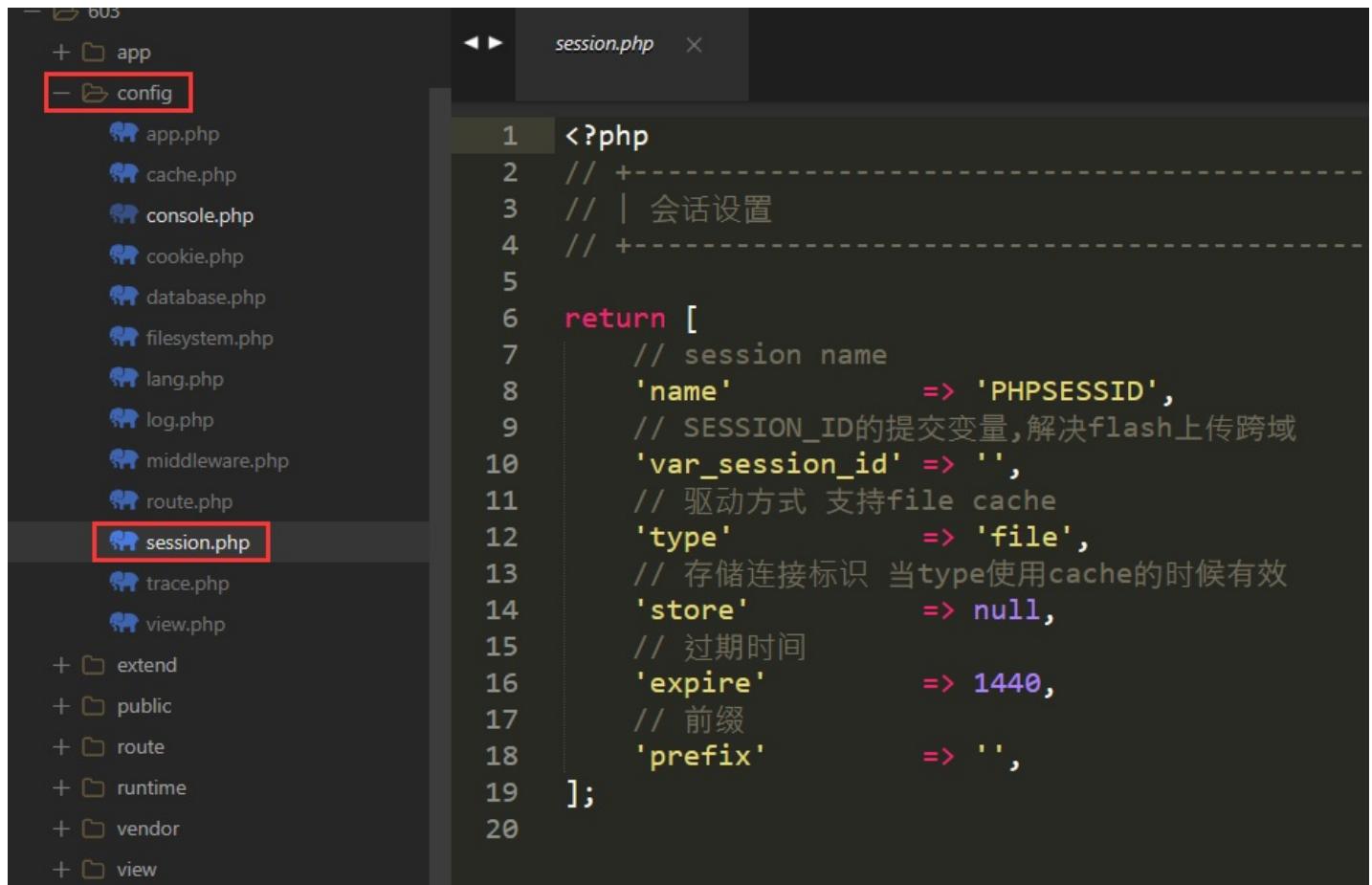
```
1 <?php
2 // 全局中间件定义文件
3 return [
4     // 全局请求缓存
5     // \think\middleware\CheckRequestCache::class,
6     // 多语言加载
7     // \think\middleware\LoadLangPack::class,
8     // Session初始化
9     // \think\middleware\SessionInit::class
10 ];
11
```

# Session初始化

## Session 初始化

框架会自动按照 `session.php` 配置的参数自动初始化 Session

## 全局配置文件：config/session.php



The screenshot shows a code editor interface with a sidebar on the left displaying a file tree. The tree includes a 'config' folder which contains several PHP files: app.php, cache.php, console.php, cookie.php, database.php, filesystem.php, lang.php, log.php, middleware.php, route.php, and session.php. The 'session.php' file is currently selected and highlighted with a red box. The main editor area shows the contents of the session.php file:

```
1 <?php
2 // -----
3 // | 会话设置
4 // -----
5
6 return [
7     // session name
8     'name'          => 'PHPSESSID',
9     // SESSION_ID的提交变量,解决flash上传跨域
10    'var_session_id' => '',
11    // 驱动方式 支持file cache
12    'type'           => 'file',
13    // 存储连接标识 当type使用cache的时候有效
14    'store'          => null,
15    // 过期时间
16    'expire'         => 1440,
17    // 前缀
18    'prefix'         => '',
19];
20
```

# Session基础用法

使用 `think\facade\Session` 类操作 `Session`

```
use think\facade\Session;
```

赋值

```
Session::set('name', 'thinkphp');
```

取值

```
// 如果值不存在, 返回null  
Session::get('name');  
// 如果值不存在, 返回空字符串  
Session::get('name', '');  
// 获取全部数据  
Session::all();
```

删除

```
Session::delete('name');
```

判断是否存在

```
Session::has('name');
```

# Cookie

---

[基本使用](#)

# 基本使用

ThinkPHP6.0 提供 `think\facade\Cookie` 类使其可以操作 `Cookie`

- cookie 相关配置在全局配置目录下 `config/cookie.php` 修改

## 设置 cookie

- `Cookie::set('名称', '值', 有效期);`
- `Cookie`数据不支持数组，如果需要请自行序列化后存入 `serialize()`

```
// 设置Cookie 有效期为 3600秒  
Cookie::set('name', 'value', 3600);
```

- 永久保存cookie，即cookie永不过期，除非 手动删除 或 清除浏览器的cookie

```
// 永久保存 Cookie  
Cookie::forever('name', 'value');
```

## 删除 cookie

```
// 删除cookie  
Cookie::delete('name');
```

## 获取 cookie

```
// 读取某个cookie数据  
Cookie::get('name');
```

# 文件上传

---

[上传简介](#)

[上传规则](#)

[上传对象](#)

[方法描述](#)

[上传验证](#)

[多文件上传](#)

[获取磁盘配置](#)

[获取默认磁盘名称](#)

[自定义命名规则](#)

# 上传简介

## 文件上传

TP6.0 内置的上传只是上传到本地服务器，  
上传到远程或者第三方平台(如: 七牛云)的话需要安装额外的扩展。

# 上传规则

## 默认的上传规则

```
<form action="{:url('index/upload')}" enctype="multipart/form-data" method="post">
    <input type="file" name="image" /> <br><br><br>
    <input type="submit" value="上传" />
</form>
```



```
// $file: \think\File 对象
$file = request()->file('image');

// 上传到本地服务器
$savename = \think\facade\Filesystem::putFile('topic', $file);
```

- 默认情况下是上传到本地服务器
- 文件上传默认根目录: runtime/storage (单应用模式)、 runtime/应用名/storage (多应用模式)

## 自定义上传规则

在全局配置文件 config/filesystem.php

# 上传对象

## 文件上传对象

```
$file = request()->file('pic');

// 有该文件字段域返回对象 否则返回 null
dump($file);
```

```
^ think\file\UploadedFile {#50 ▼
-#test: false
-#originalName: "01.jpg"
-#mimeType: "image/jpeg"
-#error: 0
#hash: []
#hashName: null
path: "C:\Windows\Temp"
filename: "phpECC8. tmp"
basename: "phpECC8. tmp"
pathname: "C:\Windows\Temp\phpECC8. tmp"
extension: "tmp"
realPath: "C:\Windows\Temp\phpECC8. tmp"
aTime: 2020-05-01 13:48:25
mTime: 2020-05-01 13:48:25
cTime: 2020-05-01 13:48:25
inode: 0
size: 32620
perms: 0100666
owner: 0
group: 0
type: "file"
writable: true
readable: true
executable: false
file: true
dir: false
link: false
linkTarget: "C:\Windows\Temp\phpECC8. tmp"
}
```

- 源码位置，该对象有什么可用的方法去看源码即可

```
vendor\topthink\framework\src\think\file\UploadedFile.php
```

The screenshot shows a code editor with a sidebar on the left displaying a file tree. A red box highlights the file 'UploadedFile.php' under the 'file' directory. A red circle with the number '1' is placed next to this highlighted file. Another red box highlights the class 'File' in the code, and a red circle with the number '2' is placed next to it, with a red arrow pointing from the highlighted 'File.php' in the sidebar to this circle.

```
12 namespace think\file;
13
14
15 use think\exception\FileException;
16 use think\File;
17
18 class UploadedFile extends File
19 {
20
21     private $test = false;
22     private $originalName;
23     private $mimeType;
24     private $error;
25
26     public function __construct(string $path, string $name)
27     {
28         $this->originalName = $originalName;
29         $this->mimeType      = $mimeType ?: 'application/octet-stream';
30         $this->test          = $test;
31         $this->error         = $error ?: UPLOAD_ERR_OK;
32     }
33 }
```

# 方法描述

\think\file\UploadedFile 对象方法

方法	作用
getRealPath()	获取临时文件路径
getOriginalName()	获取原始文件名称
getOriginalExtension()	获取原始文件后缀名

getRealPath()

- 相当于 `$_FILES['pic']['tmp_name']`
- 获取临时文件路径，将图片上传到七牛云上时可使用该方法

```
$file = request()->file('pic');
dump($file->getRealPath());
dump($_FILES);
```

^ "C:\Windows\Temp\php6946.tmp"

```
^ array:1 [▼
  "pic" => array:5 [▼
    "name" => "01.jpg"
    "type" => "image/jpeg"
    "tmp_name" => "C:\Windows\Temp\php6946.tmp"
    "error" => 0
    "size" => 32620
  ]
]
```

getOriginalName()

- 获取上传的文件的原始名称
- 假设上传的是本地桌面的 `01.jpg`，返回的就是 `01.jpg`

```
$file = request()->file('pic');
dump($file->getOriginalName());
```

getOriginalExtension()

- 假设上传的是本地桌面的 `01.jpg`，返回的就是 `.jpg`

## 方法描述

- 假设上传的是本地桌面的 `01.text` , 返回的就是 `txt`

# 上传验证

文件上传可用的验证

验证参数	描述
fileSize	上传文件的最大字节
fileExt	文件后缀，多个用逗号分割或者数组
fileMime	文件MIME类型，多个用逗号分割或者数组
image	验证图像文件的尺寸和类

# 多文件上传

## 多文件上传

```
<form action="{:url('index/upload')}" enctype="multipart/form-data" method="post">
    <input type="file" name="image[]" /> <br>
    <input type="file" name="image[]" /> <br>
    <input type="submit" value="上传" />
</form>
```

```
public function upload()
{
    // 获取表单上传文件
    // 数组对象 [对象1, 对象2, ...]
    $files = request()->file('image');

    $savename = [];

    foreach($files as $file){
        $savename[] = \think\facade\Filesystem::putFile('topic', $file);
    }

    /**
     * array(3) {
     *     [0]=>
     *         string(52) "topic/20200131\fd9a546c5a7c2cf36d43fe76f520bf1c.jpeg"
     *     [1]=>
     *         string(51) "topic/20200131\00f446e1e4d501a7e1f3d6b2fef6ef34.jpg"
     * }
     */
    var_dump($savename);
}
```

# 获取磁盘配置

## 参数说明

- \$disk 磁盘配置名称
- \$name 可选参数,磁盘配置项
- \$default 可选参数,磁盘配置项默认值

```
getDiskConfig($disk, $name = null, $default = null)
```

## 使用示例

```
// 读取磁盘配置名为public的所有配置项 返回数组
$public = \think\Facade\Filesystem::getDiskConfig('public');

// 读取磁盘配置名为public下的url配置项
$url = \think\Facade\Filesystem::getDiskConfig('public', 'url');

// 磁盘配置项 path 不存在时默认值为 /static
$default = \think\Facade\Filesystem::getDiskConfig('public', 'path', '/static');

dump($public);
dump($url);
dump($default);
```

```
^ array:4 [▼
  "type" => "local"
  "root" => "D:\phpstudy-v8.1\phpstudy_pro\WWW\tp6.0\public/storage"
  "url" => "/storage"
  "visibility" => "public"
]

^ "/storage"

^ "/static"
```

## 框架源码

```
vendor\topthink\framework\src\think\Filesystem.php
```

```
/**
```

## 获取磁盘配置

```
* 获取磁盘配置
* @param string $disk
* @param null   $name
* @param null   $default
* @return array
*/
public function getDiskConfig($disk, $name = null, $default = null)
{
    if ($config = $this->getConfig("disks.{${$disk}}")) {
        return Arr::get($config, $name, $default);
    }

    throw new InvalidArgumentException("Disk [$disk] not found.");
}
```

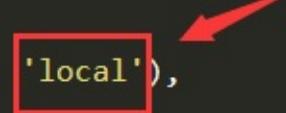
# 获取默认磁盘名称

## 获取默认磁盘名称

```
// 返回默认磁盘名称  
\think\Facade\Filesystem::getDefaultDriver();
```

config/filesystem.php

```
return [  
    // 默认磁盘  
    'default' => env('filesystem.driver', 'local'),  
    // 磁盘列表  
    'disks' => [  
        'local' => [  
            'type' => 'local',  
            'root' => app()->getRuntimePath() . 'storage',  
        ],  
        'public' => [  
            // 磁盘类型  
            'type' => 'local',  
            // 磁盘路径  
            'root' => app()->getRootPath() . 'public/storage',  
            // 磁盘路径对应的外部URL路径  
            'url' => '/storage',  
            // 可见性  
            'visibility' => 'public',  
        ],  
        // 更多的磁盘配置信息  
    ],  
];
```



# 自定义命名规则

## 传入函数

- `putFile` 第三个参数为文件命名规则

```
\think\facade\Filesystem::disk('public')->putFile('topic', $file, 'md5');
```

- 自定义函数：在 `app/common.php` 中定义函数

```
function generateName()  
{  
    // 不带目录  
    // 示例：topic/文件名.jpg  
    // return mt_rand(100, 999);  
  
    // 带目录  
    // 示例：topic/20200501/文件名.jpg  
    return date('Ymd') . '/' . mt_rand(100, 999);  
}
```

- 第三个参数传入函数名称即可

```
$savename = \think\facade\Filesystem::disk('public')->putFile('topic', $file,  
'generateName');
```

## 闭包函数

```
\think\facade\Filesystem::disk('public')->putFile('topic', $file, function(){  
    return date('Ymd') . '/' . mt_rand(100, 999);  
});
```

# layui

---

[文件上传](#)

# 文件上传

Layui 文件上传组件官方文档

<https://wwwlayui.com/doc/modules/upload.html>

HTML部分

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>upload模块快速使用</title>
    <link rel="stylesheet" href="https://www.layuicdn.com/layui-v2.5.6/css/layui.css">
    <script src="https://www.layuicdn.com/layui-v2.5.6/layui.js"></script>
</head>
<body>

    <button type="button" class="layui-btn" id="test1">
        <i class="layui-icon">上传图片
    </button>

    <script>
        layui.use(['upload', 'layer'], function(){
            var upload = layui.upload
            ,layer = layui.layer;

            //执行实例
            var uploadInst = upload.render({
                //绑定元素
                elem: '#test1'
                //上传接口
                ,url: '{:url("index/upload")}'
                // 文件域的字段名 该属性省略时默认值为: file
                ,field: 'img'
                //上传完毕回调
                ,done: function(res){
                    // 返回结果
                    console.log(res)
                }
                //请求异常回调
                ,error: function(){

                }
            });
        });
    </script>

```

```

    });
</script>
</body>
</html>

```

## 控制器方法

```

<?php
namespace app\controller;

use app\BaseController;

/**
 * layui 文件上传 在 TP6.0 中的简单示例
 *
 * @author liang QQ : 23426945
 *
 * @datetime 2020-04-03
 *
 */
class Index extends BaseController
{
    /**
     * 渲染首页模板文件
     * @return [type] [description]
     */
    public function index()
    {
        return view();
    }

    /**
     * layui 文件上传接口
     */
    public function upload()
    {
        // file('文件域的字段名')
        $file = request()->file('img');

        // 上传到本地服务器 返回文件存储位置
        //
        // disk('磁盘配置名称') 该配置 在 config/filesystem.php中的 disks 中查看
        // disk('public') 代表使用的是 disks 中的 public 键名对应的磁盘配置
        // putFile('目录名', $file);
        //
        // $savename 执行上传 返回文件存储位置
        //
        // 当前文件存储位置：public/storage/topic/当前时间/文件名
    }
}

```

## 文件上传

```
$savename = \think\facade\Filesystem::disk('public')->putFile('topic',  
$file);  
  
// 将上传后的文件位置返回给前端  
return json(['code' => 0, 'path' => $savename]);  
  
}  
}
```

# layuiAdmin单页版

---

[部署方案](#)

[接口数据](#)

[隐藏 trace](#)

[数据表格中的删除](#)

[修改数据](#)

# 部署方案

将 src 下的所有文件粘贴到 pubic/static/layuiadmin 目录下

layuiAdmin.pack > layuiAdmin.pro-v1.4.0 > src

名称	修改日期	类型	大小
controller	2020/5/12 20:12	文件夹	
lib	2020/5/12 20:12	文件夹	
style	2020/5/12 20:12	文件夹	
views	2020/5/12 20:12	文件夹	
config.js	2020/4/1 17:31	JavaScript 文件	5 KB
index.js	2020/4/1 17:31	JavaScript 文件	7 KB

dmin-v1.4.2

将 start 下的两个目录粘贴到 pubic/static/layuiadmin 目录下

layuiAdmin.pack > layuiAdmin.pro-v1.4.0 > start

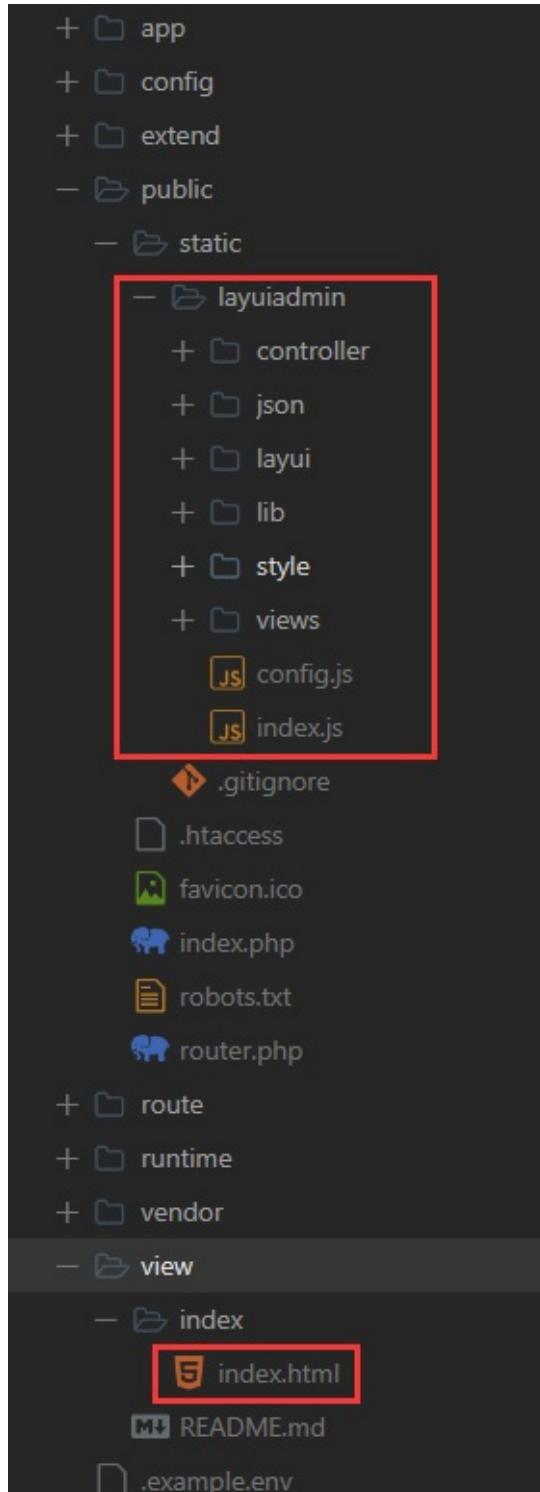
名称	修改日期	类型	大小
json	2020/5/12 20:12	文件夹	
layui	2020/5/12 20:12	文件夹	
index.html	2020/4/1 17:31	Chrome HTML D...	1 KB

将 start/index.html 放入视图目录

layuiAdmin.pack > layuiAdmin.pro-v1.4.0 > start

名称	修改日期	类型	大小
json	2020/5/12 20:12	文件夹	
layui	2020/5/12 20:12	文件夹	
index.html	2020/4/1 17:31	Chrome HTML D...	1 KB

目录结构示例



### 配置模板输出替换

- config/view.php

```
// 模板输出替换
'tpl_replace_string' => [
    '__LAYUIADMIN__' => '/staticlayuiadmin',
],
```

## 修改宿主页面

- view/index/index.html
- 修改静态资源路径，并配置version属性使其实时更新缓存

```
version: new Date().getTime()
```

## 修改 config.js

如：自定义响应字段名

# 接口数据

```
/**
 * 侧边菜单
 * index/menu
 */
public function menu()
{
    $data = [
        [
            'title' => '主页',
            'icon'  => 'layui-icon-home',
            'list'  => [
                [
                    'title' => '控制台',
                    'jump'  => '/',
                ],
                [
                    'name'  => 'homepage1',
                    'title' => '主页一',
                    'jump'  => 'home/homepage1',
                ],
                [
                    'name'  => 'homepage2',
                    'title' => '主页二',
                    'jump'  => 'home/homepage2',
                ],
            ],
        ],
        [
            'name' => 'user',
            'title' => '用户管理',
            'icon'  => 'layui-icon-home',
            'list'  => [
                [
                    'name'  => 'admin',
                    'title' => '后台管理员',
                    'jump'  => 'admin/index',
                ],
                [
                    'name'  => 'admin1',
                    'title' => '后台管理员',
                    'jump'  => 'aa',
                ],
            ],
        ],
    ];
}
```

```
    return rps('左侧菜单', $data);
}

/**
 * 用户信息
 * index/userSession
 */
public function userSession()
{
    $data = [
        'username' => '贤心',
        'sex'       => '男',
        'role'      => 1,
    ];

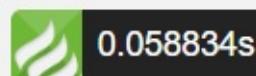
    return rps('用户数据', $data);
}

/**
 * 消息中心
 * index/messageNew
 */
public function messageNew()
{
    $data = [
        'newmsg' => 3
    ];

    return rps('用户数据', $data);
}
```

# 隐藏 trace

TP6 调试模式自带 trace



在宿主页面 start/index.html 修改为以下内容可以隐藏 trace

- 使用 jq 移除元素

```
<script>
layui.config({
    base: '__LAYUIADMIN__/'
    ,version: new Date().getTime()
}).use('index', function(){
    layui.$('#think_page_trace, #think_page_trace_open').remove()
});
</script>
```

- 右下角 trace 没了



# 数据表格中的删除

## 单个删除

```

if(obj.event === 'del'){
    layer.confirm('确定删除此链接？', {
        title: '删除链接',
    }, function(index){
        admin.req({
            url: '/admin/links/del',
            type: 'get',
            data: {id: data.id},
            success: res => {
                console.log(res)
                layer.msg(res.msg)
                if (res.status == 200) {
                    table.reload('link')
                }
            }
        })
        layer.close(index);
    });
}

```

## 批量删除

```

// 批量删除
batchdel: function(){
    var checkStatus = table.checkStatus('link');//数据表格id
    ,checkData = checkStatus.data; //得到选中的数据

    if(checkData.length === 0){
        return layer.msg('请选择数据');
    }

    var ids = [];

    for (var v of checkData) {
        ids.push(v.id)
    }

    layer.confirm('确定删除吗？', function(index) {
        admin.req({
            url: '/admin/links/del',
            type: 'post',

```

```

    data: {ids: ids},
    done: res => {
        // console.log(res)
        layer.msg(res.msg)
        if (res.status == 200) {
            table.reload('link');//数据表格id
        }
    }
});

layer.close(index)
});
}

```

## 模型方法

```

/**
 * layui数据表格删除
 * get请求 单个删除
 * post请求 多个删除
 */
public function del()
{
    // 如果是post请求执行批量删除
    if (request()->isPost()) {
        $ids = input('post.ids/a');
        self::destroy($ids);
        return res(200, '删除成功');
    } else {
        // 单个删除
        $id = input('get.id/d');
        // 查询到数据返回模型数据对象 查不到数据返回null
        $data = self::find($id);
        if ($data === null) {
            return res(201, '删除失败,数据不存在或已被删除');
        }
        // 删除当前模型数据
        $data->delete();
        return res(200, '删除成功');
    }
}

```

# 修改数据

```
/**
 * 修改颜色
 */
public function edit()
{
    $id = input('post.id/d');
    // 准备更新的数据
    $data = $this->_postData();
    // 避免新数据和其他数据名称重复
    $isExists = self::where('name', $data['name'])
        ->where('id', '<>', $id)
        ->find();
    if ($isExists !== null) {
        return res(203, '该颜色名称已存在');
    }
    // 数据验证
    try {
        validate('app\index\validate\DoorColor.edit')->check($data);
    } catch (\think\exception\ValidateException $e) {
        return res(201, $e->getError());
    }
    // 执行更新
    try {
        self::update($data, ['id' => $id]);
    } catch (\think\db\exception\PDOException $e) {
        return res(202, $e->getMessage());
    }

    return res(200, '修改成功');
}
```

# layuiAdmin iframe版

---

[iframe版部署](#)

[放入TP6.0视图](#)

[登陆页面验证码](#)

[引入authtree扩展组件](#)

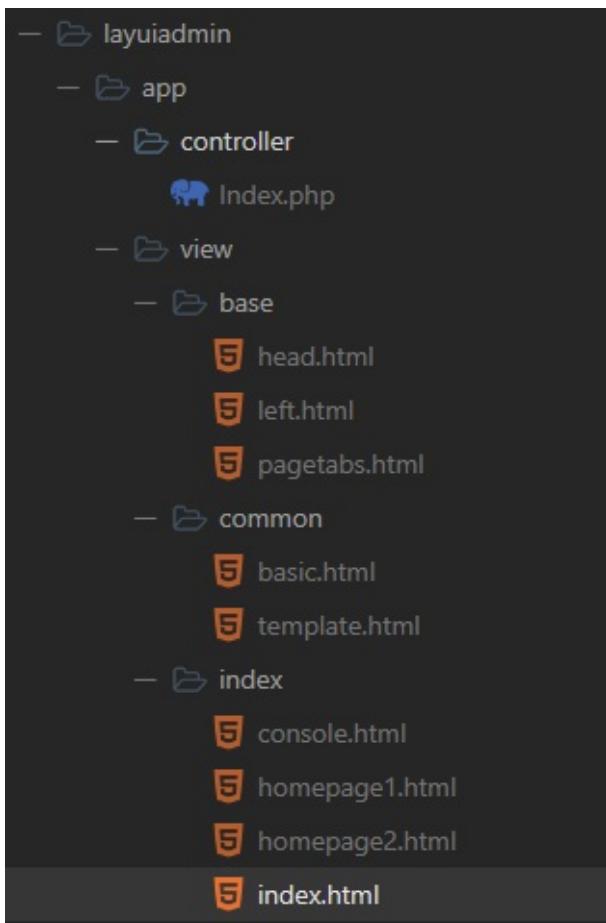
[2020.1.24版 主页成背景](#)

# iframe版部署

## layuiadmin iframe 模板布局方案分享

- 将layuiadmin静态资源放入框架中
- 定义模板输出替换
- 将主页的模板文件放入框架视图并替换静态资源路径
- 主页采用模板分离
- iframe页采用模板继承

## 目录结构



## 后台主页

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>layuiAdmin std - 通用后台管理模板系统 (iframe标准版) </title>
  <meta name="renderer" content="webkit">

```

```

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
<meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, maximum-scale=1.0, user-scalable=0">
    <link rel="stylesheet" href="__LAYUIADMIN__/layui/css/layui.css" media="all">
>
    <link rel="stylesheet" href="__LAYUIADMIN__/style/admin.css" media="all">
</head>
<body class="layui-layout-body">

    <div id="LAY_app">
        <div class="layui-layout layui-layout-admin">
            <!-- 头部区域 -->
            {include file="base/head" /}

            <!-- 侧边菜单 -->
            {include file="base/left" /}

            <!-- 页面标签 -->
            {include file="base/pagetabs" /}

            <!-- 主体内容 -->
            <div class="layui-body" id="LAY_app_body">
                <div class="layadmin-tabsbody-item layui-show">
                    <iframe src="{:url('index/console')}" frameborder="0" class="layadmin-iframe"></iframe>
                </div>
            </div>

            <!-- 辅助元素，一般用于移动设备下遮罩 -->
            <div class="layadmin-body-shade" layadmin-event="shade"></div>
        </div>
    </div>

    <script src="__LAYUIADMIN__/layui/layui.js"></script>

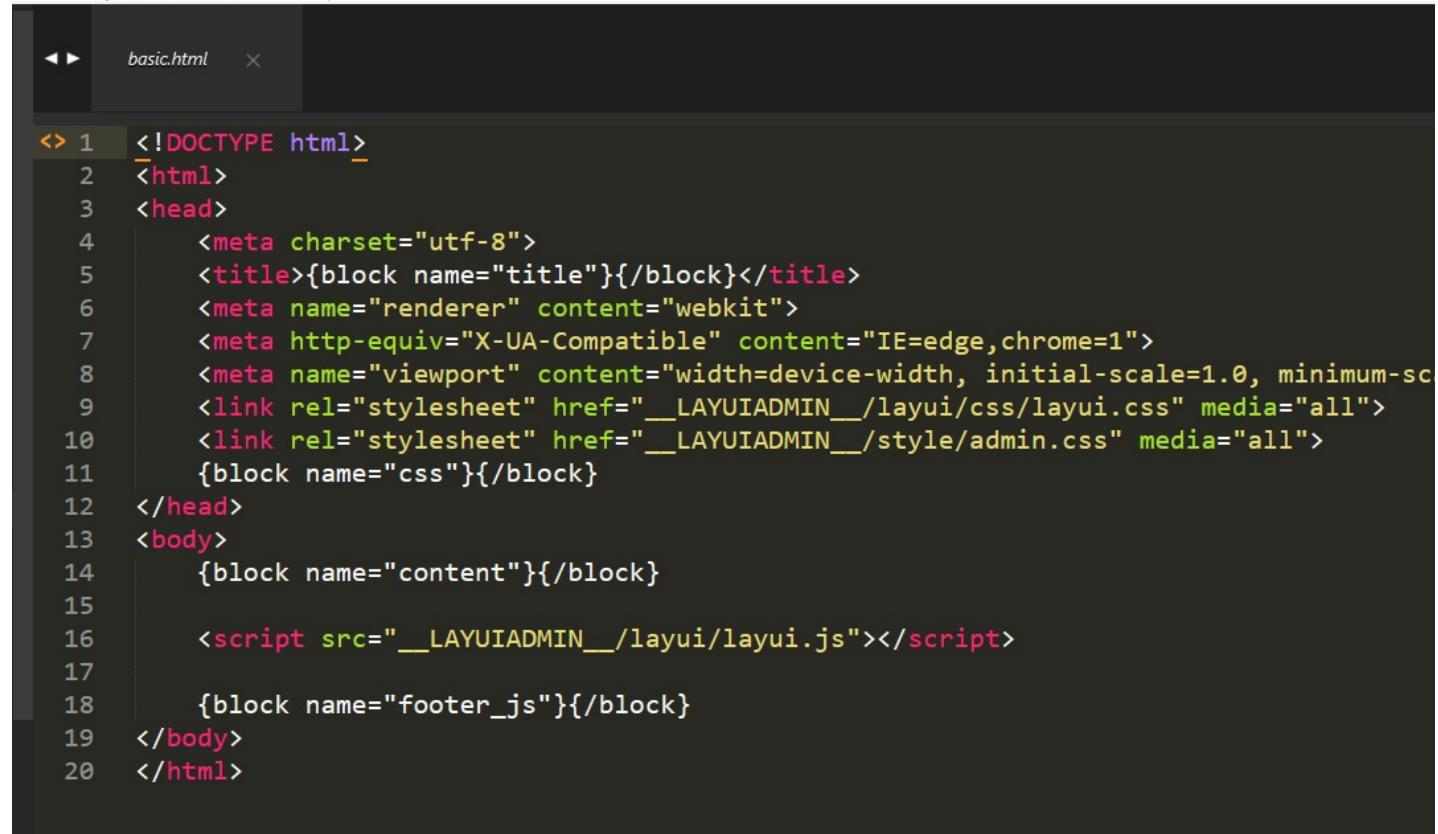
    <script>
        layui.config({
            base: '__LAYUIADMIN__/' //静态资源所在路径
            ,version: new Date().getTime() //实时更新缓存,开发环境使用
        }).extend({
            index: 'lib/index' //主入口模块
        }).use('index');
    </script>
</body>
</html>

```

iframe 页面

## 基础模板

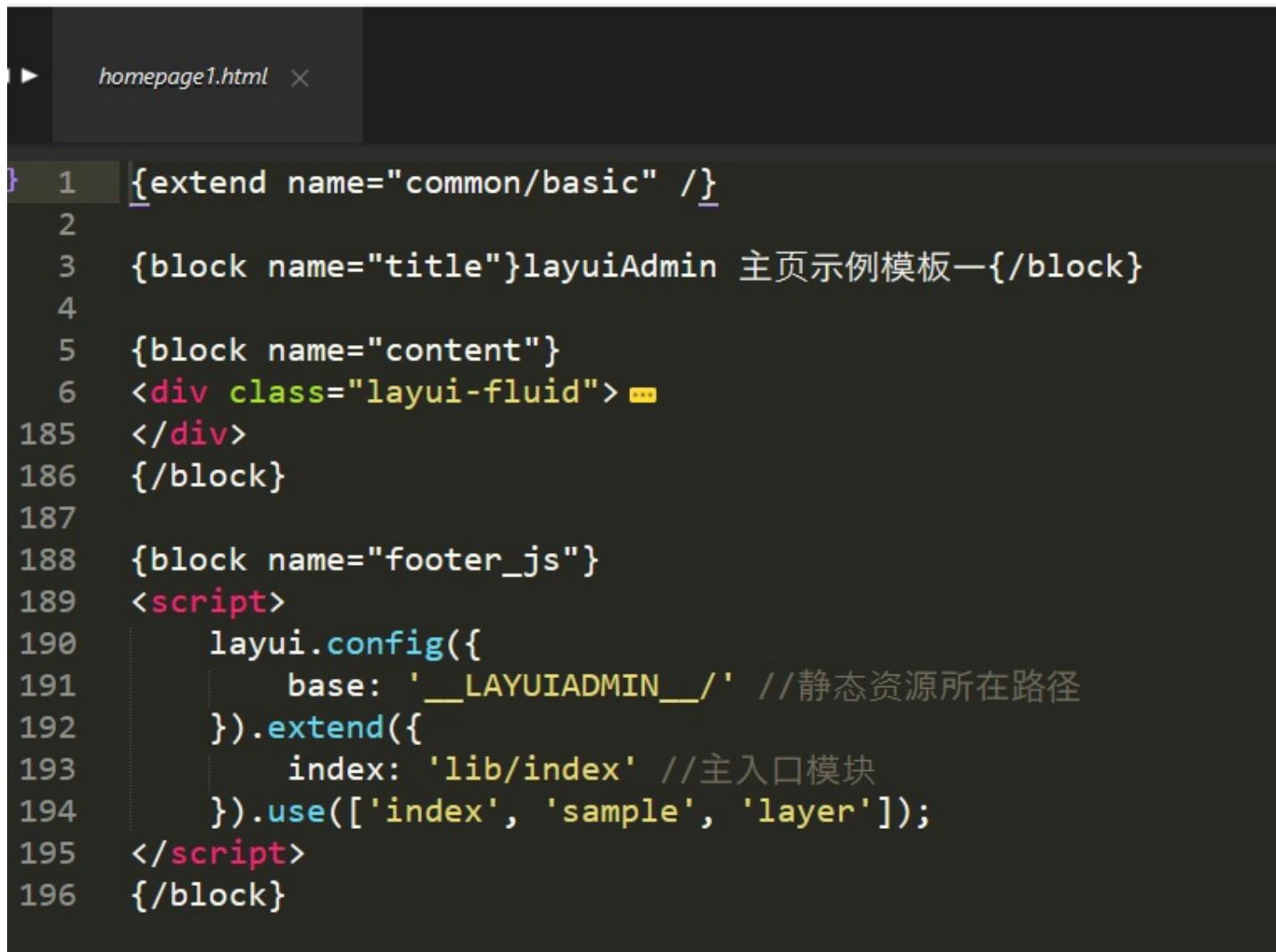
Tools Project Preferences Help



The screenshot shows a code editor window with a dark theme. At the top, there's a toolbar with icons for back, forward, and close. Below the toolbar, the file name "basic.html" is displayed. The main area contains the following HTML code:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>{block name="title"}{/block}</title>
    <meta name="renderer" content="webkit">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0, user-scalable=yes">
    <link rel="stylesheet" href="__LAYUIADMIN__/layui/css/layui.css" media="all">
    <link rel="stylesheet" href="__LAYUIADMIN__/style/admin.css" media="all">
    {block name="css"}{/block}
</head>
<body>
    {block name="content"}{/block}
    <script src="__LAYUIADMIN__/layui/layui.js"></script>
    {block name="footer_js"}{/block}
</body>
</html>
```

## 子模板



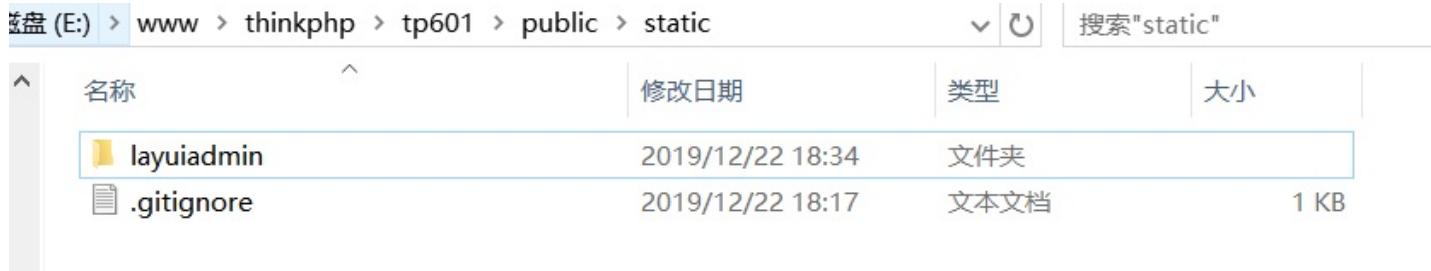
The screenshot shows a code editor window with the file 'homepage1.html' open. The code is a template for a Layui Admin page, featuring block-based logic and Layui configuration.

```
{1  {extend name="common/basic" /}
2
3  {block name="title"}layuiAdmin 主页示例模板一{/block}
4
5  {block name="content"}
6    <div class="layui-fluid">...
185 </div>
186 {/block}
187
188 {block name="footer_js"}
189 <script>
190   layui.config({
191     base: '__LAYUIADMIN__/' //静态资源所在路径
192   }).extend({
193     index: 'lib/index' //主入口模块
194   }).use(['index', 'sample', 'layer']);
195 </script>
196 {/block}
```

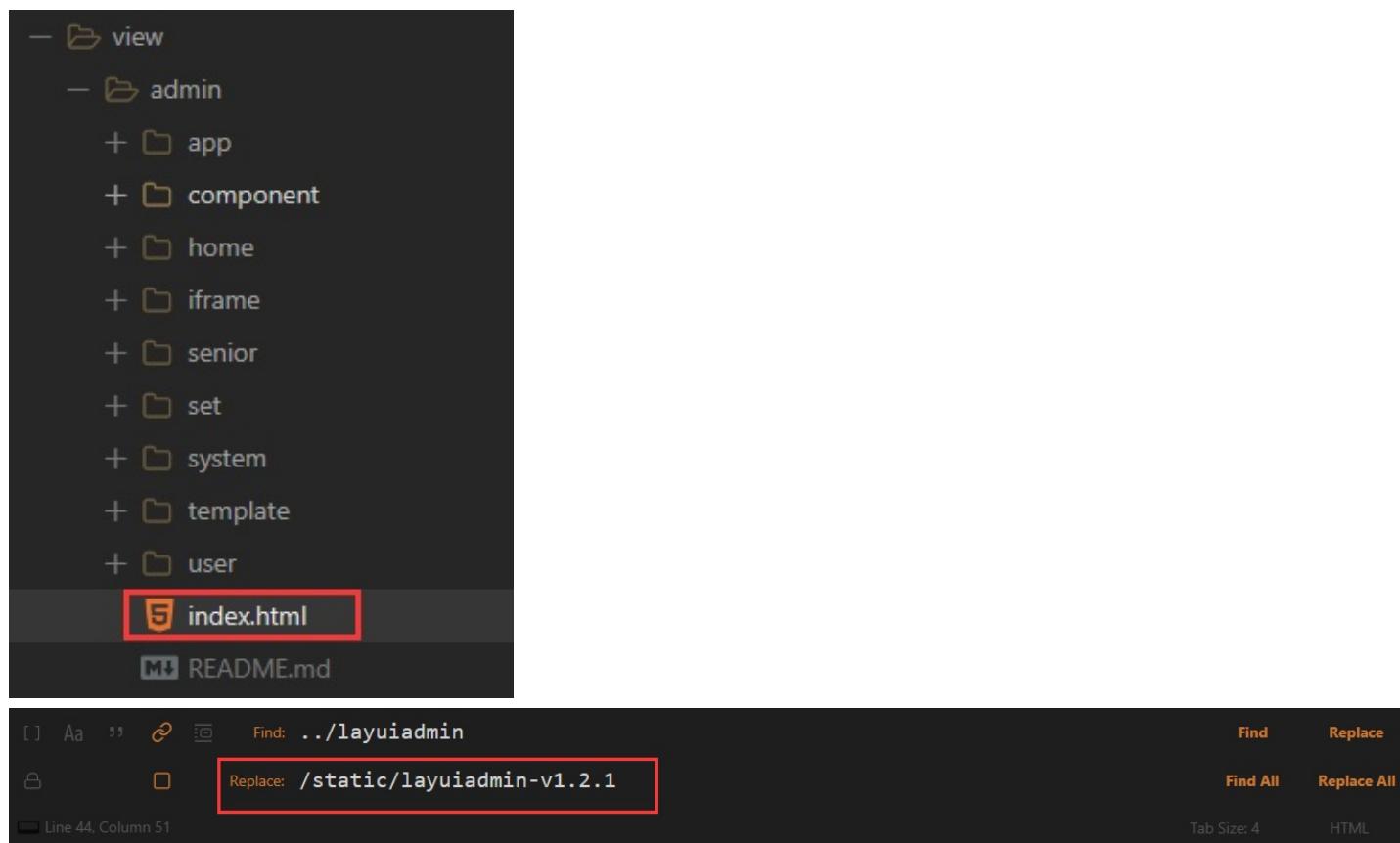
# 放入TP6.0视图

## 第一步：放入 layuiadmin 静态资源

将 "src" 目录下的 "layuiadmin" 文件夹粘贴到public目录下static文件夹中



## 第二步：修改后台主页面的静态资源路径



## 修改 iframe标签的src属性 未修改会报错

[0] **HttpException** in **Controller.php** line 76

**控制器不存在:app\controller\Home**

```

67.         ->setAction($this->actionName);
68.     }
69.
70.     public function exec()
71.     {
72.         try {
73.             // 实例化控制器
74.             $instance = $this->controller($this->controller);
75.         } catch (ClassNotFoundException $e) {
76.             throw new HttpException(404, "controller not exists:' . $e->getClass());
77.         }
78.
79.         // 注册控制器中间件
80.         $this->registerControllerMiddleware($instance);
81.
82.         return $this->app->middleware->pipeline('controller')
83.             ->send($this->request)
84.             ->then(function () use ($instance) {
85.                 // 获取当前操作名

```

**Call Stack**

1. in Controller.php line 76
2. at Controller->exec() in Dispatch.php line 96
3. at Dispatch->run() in Route.php line 747
4. at Route->think(\closure\object\Request)) in Pipeline.php line 59
5. at Pipeline->think(\closure\object\Request)) in Pipeline.php line 65
6. at Pipeline->then(\object(Closure)) in Route.php line 748

0.108626s

## 搜索关键字 iframe

index.html

```

439         <dd layadmin-event="closeThisTabs"><a href="#>
440         <dd layadmin-event="closeOtherTabs"><a href="#>
441         <dd layadmin-event="closeAllTabs"><a href="#>
442             </dl>
443         </li>
444     </ul>
445     </div>
446     <div class="layui-tab" lay-unauto lay-allowClose="true">
447         <ul class="layui-tab-title" id="LAY_app_tabsheader">
448             <li lay-id="home/console.html" lay-attr="home/console">
449         </ul>
450     </div>
451     </div>

454     <!-- 主体内容 -->
455     <div class="layui-body" id="LAY_app_body">
456         <div class="layadmin-tabsbody-item layui-show">
457             <iframe src="home/console.html" frameborder="0" class="layui-body-iframe">
458         </div>
459     </div>

461     <!-- 辅助元素，一般用于移动设备下遮罩 -->
462     <div class="layadmin-body-shade" layadmin-event="shade"></div>

```

# 登陆页面验证码

layuiadmin iframe 版 登陆页面验证码改为 tophink/think-captcha 验证码

## think-captcha 快速使用

```

```

## layuiAdmin 登陆页面使用 think-captcha 自定义验证码示例

```
<div class="layui-col-xs5">
    <div style="margin-left: 10px;">
        <!--  -->
        
    </div>
</div>
```

# 引入authtree扩展组件

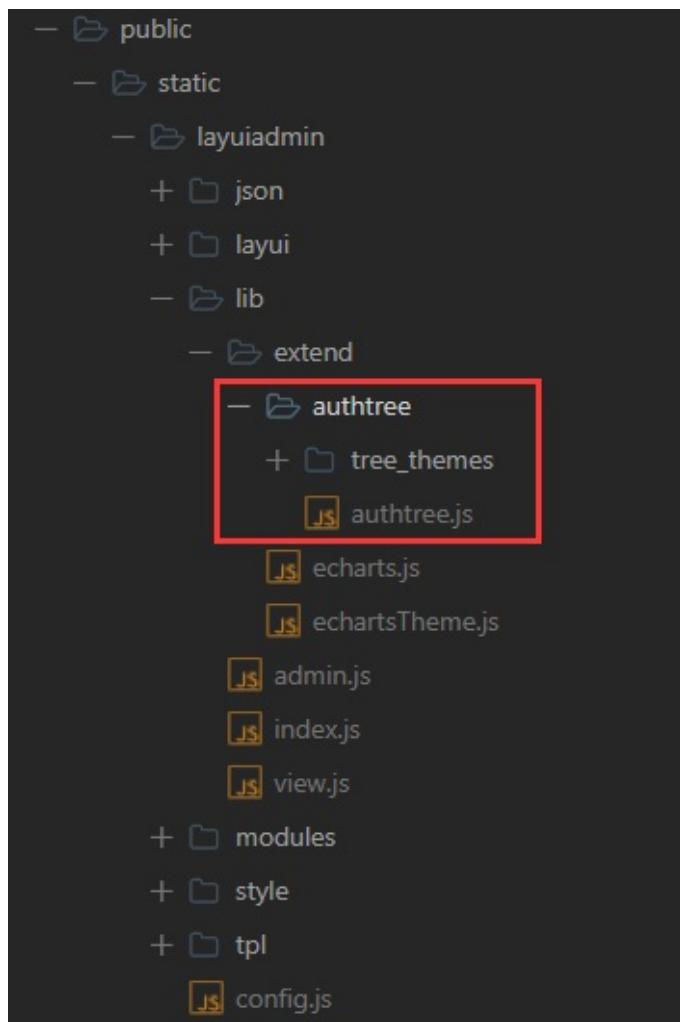
layuiadmin iframe 2020版引入 authtree 组件

- 引入其他扩展组件方法类似
- authtree 权限树组件 以此组件为例

## authtree 仓库地址

<https://github.com/wangerzi/layui-authtree>

将 authtree 中的 layui\_exts 目录放入 lib/extend 目录下并重命名为 authtree



## 配置 authtree组件 模板输出替换

```
// 模板输出字符串替换
'tpl_replace_string' => [
    // layuiadmin 静态资源
    '__LAYUIADMIN__' => '/static/admin/layuiadmin',
```

```
// authtree 权限树扩展组件
'__AUTHTREE__'    => '/static/layuiadmin/lib/extend/authtree',
],
```

## 渲染权限树

```
layui.config({
  base: '__LAYUIADMIN__/' //静态资源所在路径
}).extend({
  index: 'lib/index' //主入口模块

  ,authtree: '{/}__AUTHTREE__/authtree'

}).use(['index', 'form', 'upload', 'authtree'], function(){
  var $ = layui.$
  ,form = layui.form

  ,authtree = layui.authtree

  ,upload = layui.upload;

  $.ajax({
    url: '/tree.json',
    dataType: 'json',
    success: function(data){
      var trees = data.data.trees;
      // #LAY-auth-tree-index 权限树容器
      authtree.render('#LAY-auth-tree-index', trees, {
        inputname: 'authids[]',
        layfilter: 'lay-check-auth',
        autowidth: true
        ,theme: 'auth-skin-universal'
        ,themePath: '__AUTHTREE__/tree_themes/'
      });
    }
  });
});
```

## layuiadmin 更多使用笔记

参考文档的 layuiadmin 章节：<https://www.kancloud.cn/monday/phper>

# 2020.1.24版 主页成背景

## 问题描述

- layuiAdmin iframe 2020.1.24 版 背景混乱问题
- 官方已回复，这确实是个bug，下个版本会修复



## 解决方案

文件位置： src/layuiadmin/lib/index.js 67行左右

```

admin.tabsBodyChange(tabsPage.index, {
    url: url
    ,text: text
});

```

修改为以下内容，也就是给方法加个延迟就可以了

```

setTimeout(function(){
    admin.tabsBodyChange(tabsPage.index, {
        url: url
        ,text: text
    });
}, 10);

```

## 测试效果，没毛病

The screenshot shows a web browser window with the URL `blog.cy/admin`. The left sidebar is a dark-themed admin panel titled "layuiAdmin" with categories like Control Panel, Main Page 1, Main Page 2, Components, Pages, Applications, Users, and Settings. The "Main Page 1" item is highlighted in green. The main content area displays an error message:

#0 [0]HttpException in Controller.php line 76  
控制器不存在:app\controller\Home

```
67.         ->setAction($this->actionName);
68.     }
69.
70.     public function exec()
71.     {
72.         try {
73.             // 实例化控制器
74.             $instance = $this->controller($this->controller);
75.         } catch (ClassNotFoundException $e) {
76.             throw new HttpException(404, 'controller not exists:' . $e->getClass());
77.         }
78.
79.         // 注册控制器中间件
80.         $this->registerControllerMiddleware($instance);
81.
82.         return $this->app->middleware->pipeline('controller')
83.             ->send($this->request)
84.             ->then(function () use ($instance) {
85.                 // 获取当前操作名
```

Below the code, there is a "Call Stack (折叠)" section with the following entry:

1. in Controller.php line 76

# 经验分享

---

[章节停更说明](#)

[PhpStudy-v8.1 无法解析PHP代码](#)

[屏蔽Sublime的提示](#)

[正则匹配img标签](#)

[Sublime 3.x 激活码](#)

[windows dos 窗口命令 type](#)

[windows dos 窗口命令 start](#)

[apache配置主机](#)

[navicat 闲置时间过久会卡死](#)

# 章节停更说明

经验分享章节不再更新

请移步：<https://www.kancloud.cn/monday/phper/1544810>

# PhpStudy-v8.1 无法解析PHP代码

phpstudy v8.1 版本无法解析php代码

```
← → C ⓘ localhost/1.php

<?php

header (' content-type:image/jpeg' );

$b = imagecreatetruecolor(700, 500);

$blue = imagecolorallocate($b, 0, 0, 255);

imagefill($b, 0, 0, $blue);

imagejpeg($b);

?>
```

原因: localhost 站点状态 过期

The screenshot shows the PhpStudy v8.1 control panel interface. On the left, there is a sidebar with icons for Home, Website, FTP, Database, Environment, and Settings. The main area displays a list of websites. One website entry is shown in detail:

编号	网站域名	端口	物理路径	状态	到期	操作
1	localhost	80	D:/phpstudy-v...	过期	2020-03-09	<a href="#">管理</a>

A red arrow points to the "过期" (Expired) status column, which is highlighted with a red box. The status column header is also labeled "过期".

## 解决方案

点击 管理 中的 启用

The screenshot shows the main interface of the PhpStudy v8.1 control panel. On the left is a sidebar with icons for Home, Website, FTP, Database, Environment, and Settings. The main area is titled "phpstudy 正式版发布" and shows a table of websites. One row is selected, and a red arrow points to the "启用" (Enable) button in the context menu, which is highlighted with a red box. The table columns are: 编号 (ID), 网站域名 (Website Domain), 端口 (Port), 物理路径 (Physical Path), 状态 (Status), 到期 (Expiration Date), and 操作 (Operations).

编号	网站域名	端口	物理路径	状态	到期	操作
1	localhost	80	D:/phpstudy-v...	过期	2020-03-09	<input checked="" type="radio"/> 启用 <input type="checkbox"/> 修改 <input type="checkbox"/> 删除 <input type="checkbox"/> php版本 <input type="checkbox"/> php扩展 <input type="checkbox"/> 网站首页设置 <input type="checkbox"/> 打开网站 <input type="checkbox"/> 伪静态 <input type="checkbox"/> composer <input type="checkbox"/> 打开根目录

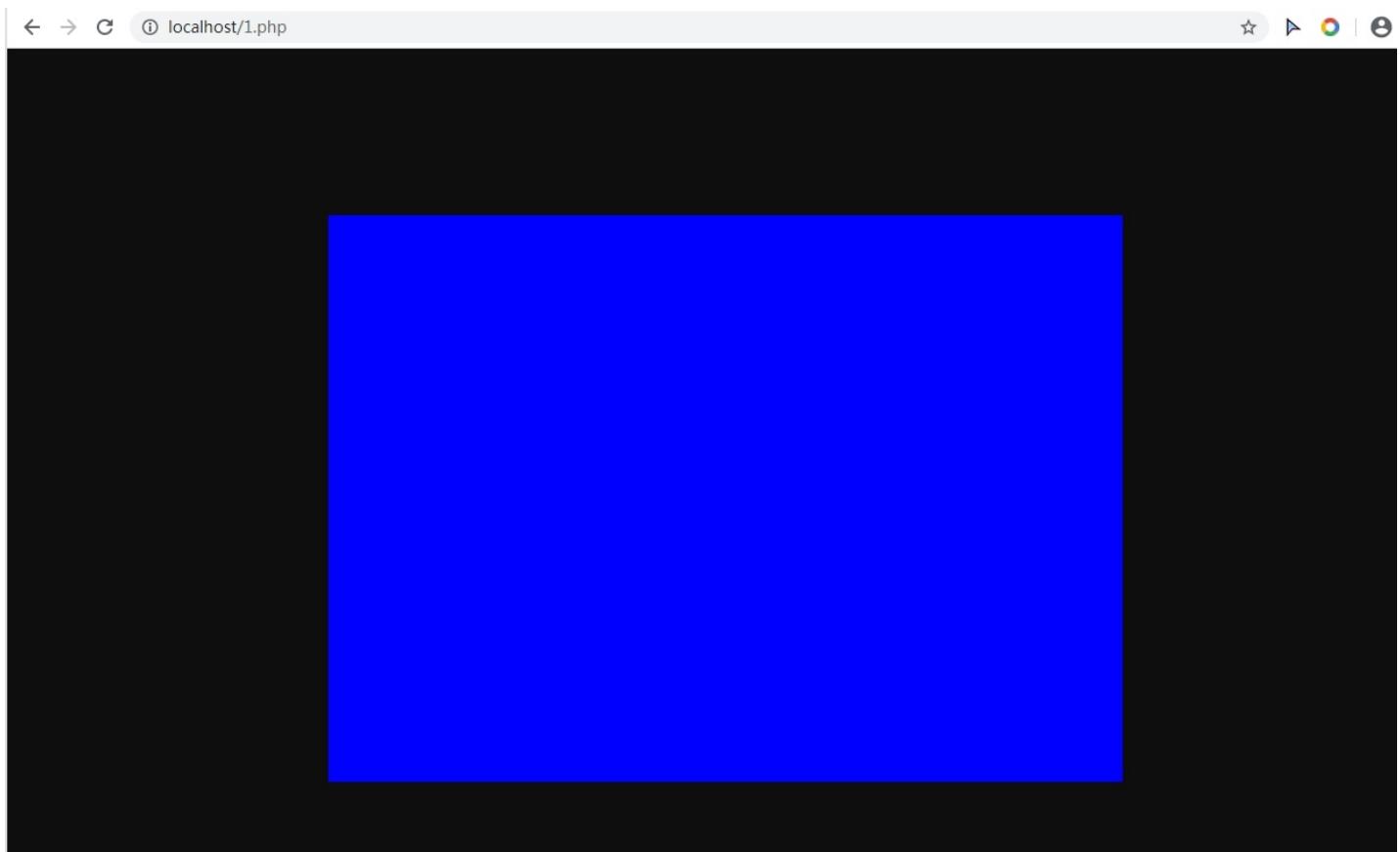
状态切换成正常就可以正常访问PHP文件了

The screenshot shows the main interface of the PhpStudy v8.1 integrated environment. On the left sidebar, there are icons for Home, Website, FTP, Database, Environment, and Settings. The main content area displays a table titled "phpstudy 正式版发布" (phpstudy Official Release) with the following data:

编号	网站域名	端口	物理路径	状态	到期	操作
1	localhost	80	D:/phpstudy-v...	正常	2030-03-09	<a href="#">管理</a>

A red box highlights the "正常" (Normal) status column, and a red arrow points from the text above to this box. At the bottom of the main panel, there are status indicators for Apache 2.4.39 (green) and MySQL (green), and a version information bar stating "版本 : 8.1.0.1".

测试访问

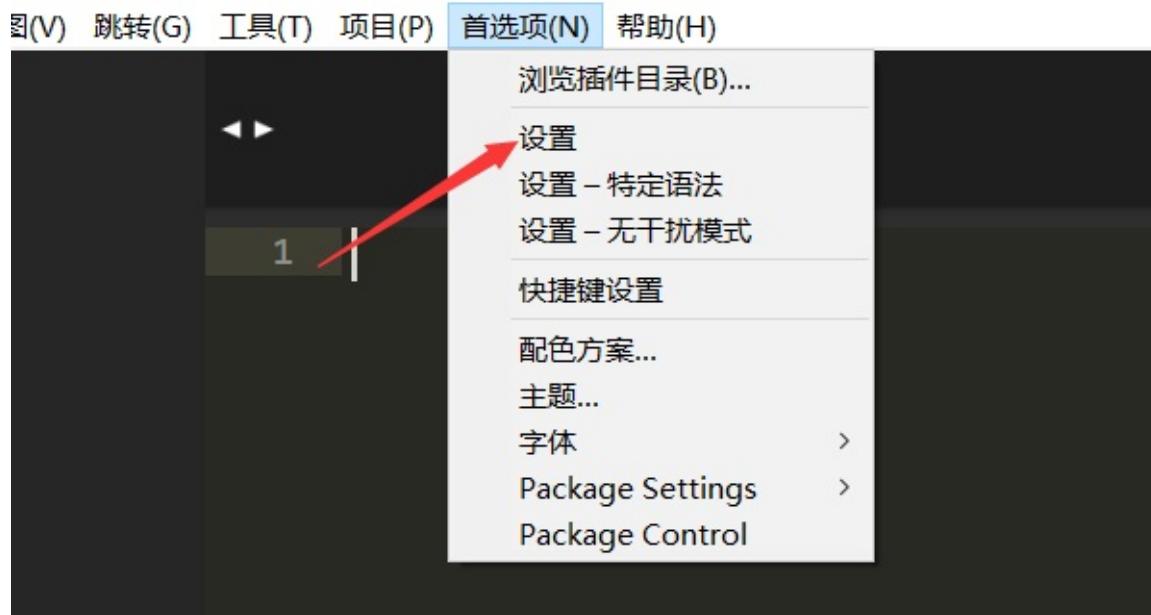


# 屏蔽Sublime的提示

关闭更新提醒

```
{  
    // 关闭自动换行  
    "word_wrap": false,  
    // 关闭更新提醒  
    "update_check": false,  
}
```

ink-template) - Sublime Text (LICENSE UPGRADE REQUIRED)



```

1 // Place your settings in the file "Packages",
2 // which overrides the settings in here.
3 //
4 // Settings may also be placed in syntax-specific
5 // example, in Packages/User/Python.sublime-
6 {
7     // Sets the colors used within the text editor.
8     "color_scheme": "Monokai.sublime-color-scheme"
9
10    // Note that the font_face and font_size
11    // specific settings file, for example,
12    // Because of this, setting them here will
13    // in your User File Preferences.
14    "font_face": "",
15    "font_size": 10,
16
17    // Valid options on all platforms are:
18    // - "no_bold": Disables bold text
19    // - "no_italic": Disables italic text
20    // Antialiasing options:
21    // - "no_antialias": Disables antialiasing
22    // - "gray_antialias": Uses grayscale antialiasing
23    // Ligature options:
24    // - "no_liga": Disables standard ligatures
25    // - "no_clig": Disables contextual ligatures
26    // - "no_calt": Disables contextual alternative characters
27    // - "dlig": Enables discretionary ligatures
28    // - "calt": Enables OpenType style substitutions

```

```

1 {
2     "color_scheme": "Packages/Color Scheme - Default.sublime-color-scheme",
3     "enable_autoreload": true,
4     "font_size": 13,
5     "ignored_packages": [
6         "Vintage"
7     ],
8     "theme": "Afterglow-orange.sublime-theme",
9
10    // 关闭自动换行
11    "word_wrap": false,
12    // 关闭更新提醒
13    "update_check": false,
14
15 }
16

```

Line 16, Column 1; Saved F:\software\sublime\Data\Packages\User\Preferences.sublime-settings (UTF-8) Tab Size: 4 JSON (Sublime)

## 另一种提示就是授权提示

授权码在本章节的下下章节

# 正则匹配img标签

## 正则匹配 img标签及src属性值

```
$content = file_get_contents('1.html');  
preg_match_all('/<img(..*?)src=(\"(.*?)\")(.*)>/is', $content, $matches);
```

```
matches[0] //整个img标签  
matches[2] //图片的url
```

## 修饰符：

- i 忽略大小写
- s 单行文本模式

# Sublime 3.x 激活码

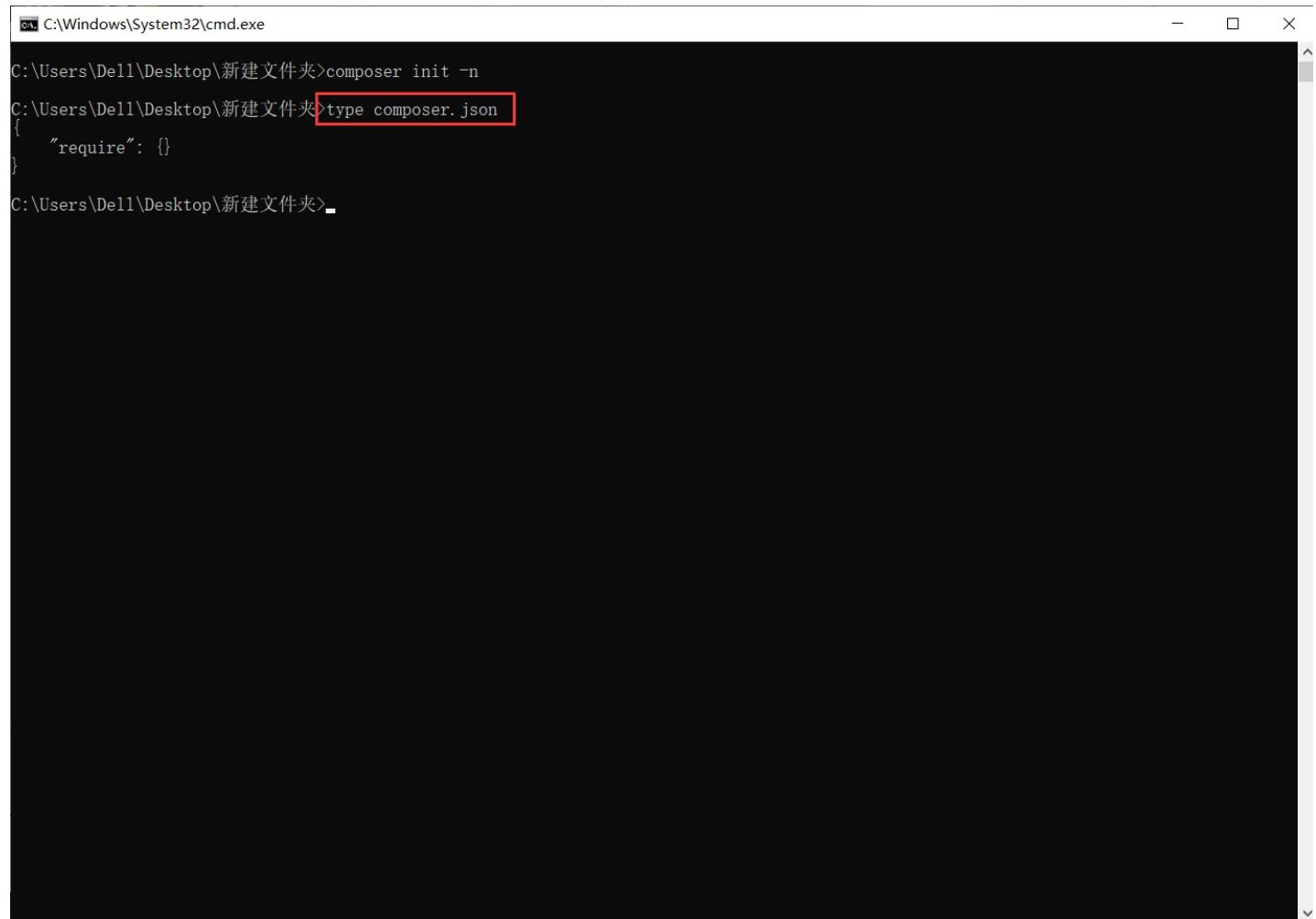
## Sublime 3 激活码

若激活码失效，联系QQ：[23426945](#) 更新时间：[2020-01-10](#)

```
----- BEGIN LICENSE -----
Member J2TeaM
Single User License
EA7E-1011316
D7DA350E 1B8B0760 972F8B60 F3E64036
B9B4E234 F356F38F 0AD1E3B7 0E9C5FAD
FA0A2ABE 25F65BD8 D51458E5 3923CE80
87428428 79079A01 AA69F319 A1AF29A4
A684C2DC 0B1583D4 19CBD290 217618CD
5653E0A0 BACE3948 BB2EE45E 422D2C87
DD9AF44B 99C49590 D2DBDEE1 75860FD2
8C8BB2AD B2ECE5A4 EFC08AF2 25A9B864
----- END LICENSE -----
```

# windows dos 窗口命令 type

window dos 命令窗口 type 可用于查询文件内容



The screenshot shows a Windows Command Prompt window titled 'cmd C:\Windows\System32\cmd.exe'. The command 'type composer.json' has been entered, and its output is displayed. The output shows a JSON object with a single key 'require' which has an empty value. The line 'type composer.json' is highlighted with a red rectangle.

```
C:\Users\DELL\Desktop>composer init -n
C:\Users\DELL\Desktop>type composer.json
{
    "require": {}
}
C:\Users\DELL\Desktop>
```

# windows dos 窗口命令 start

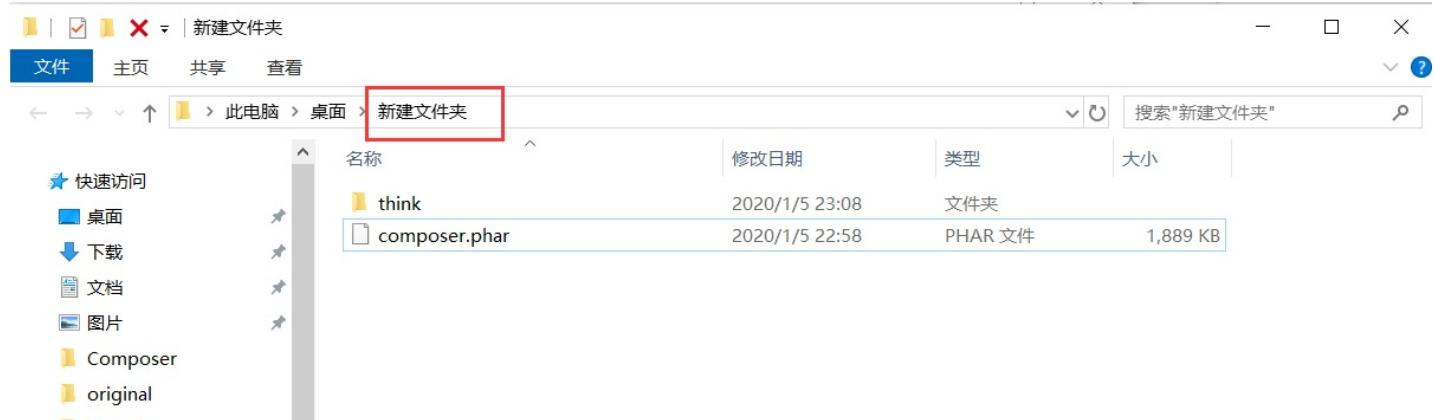
windows dos 窗口命令 start

在资源管理器中打开当前目录

```
start .
```



执行 start . 会弹出以下内容

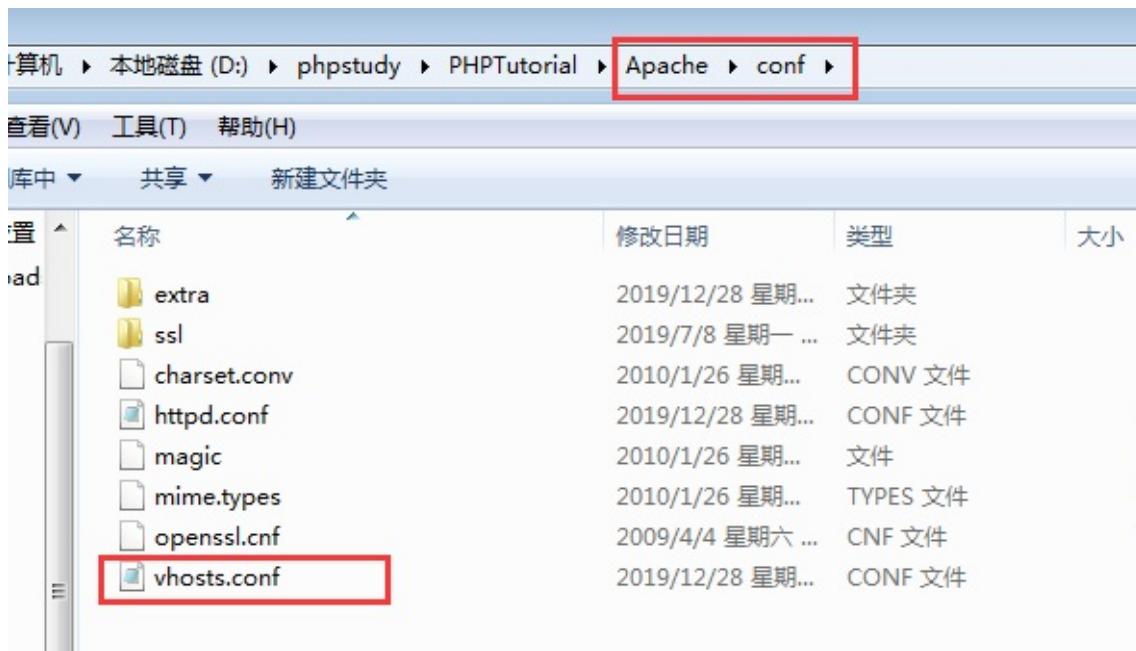


# apache配置主机

## 本章节说明

针对单独配的PHP环境, 前提: apache 已开启重写模块

## 打开虚拟主机配置文件



## 指向 ThinkPHP 的public目录

```
<VirtualHost *:80>
    DocumentRoot "D:\www\thinkphp\think6.0.1/public"
    ServerName tp601.cy
    ServerAlias admin.tp601.cy admin2.tp601.cy
    <Directory "D:\www\thinkphp\think6.0.1/public">
        Options FollowSymLinks ExecCGI
        AllowOverride All
        Order allow,deny
        Allow from all
        Require all granted
    </Directory>
</VirtualHost>
```

## host 文件中添加虚拟域名

```
C:\Windows\System32\drivers\etc\hosts
```

```
127.0.0.1 tp601.cy  
127.0.0.1 admin.tp601.cy  
127.0.0.1 admin2.tp601.cy
```

## 重启 apache 服务

因为修改了 apache的配置文件(vhosts.conf) , 所以重启apache才会生效

修改 hosts 不需要重启apache就可以生效 , 因为 hosts 不是apache的配置文件

# navicat 闲置时间过久会卡死

## 问题描述

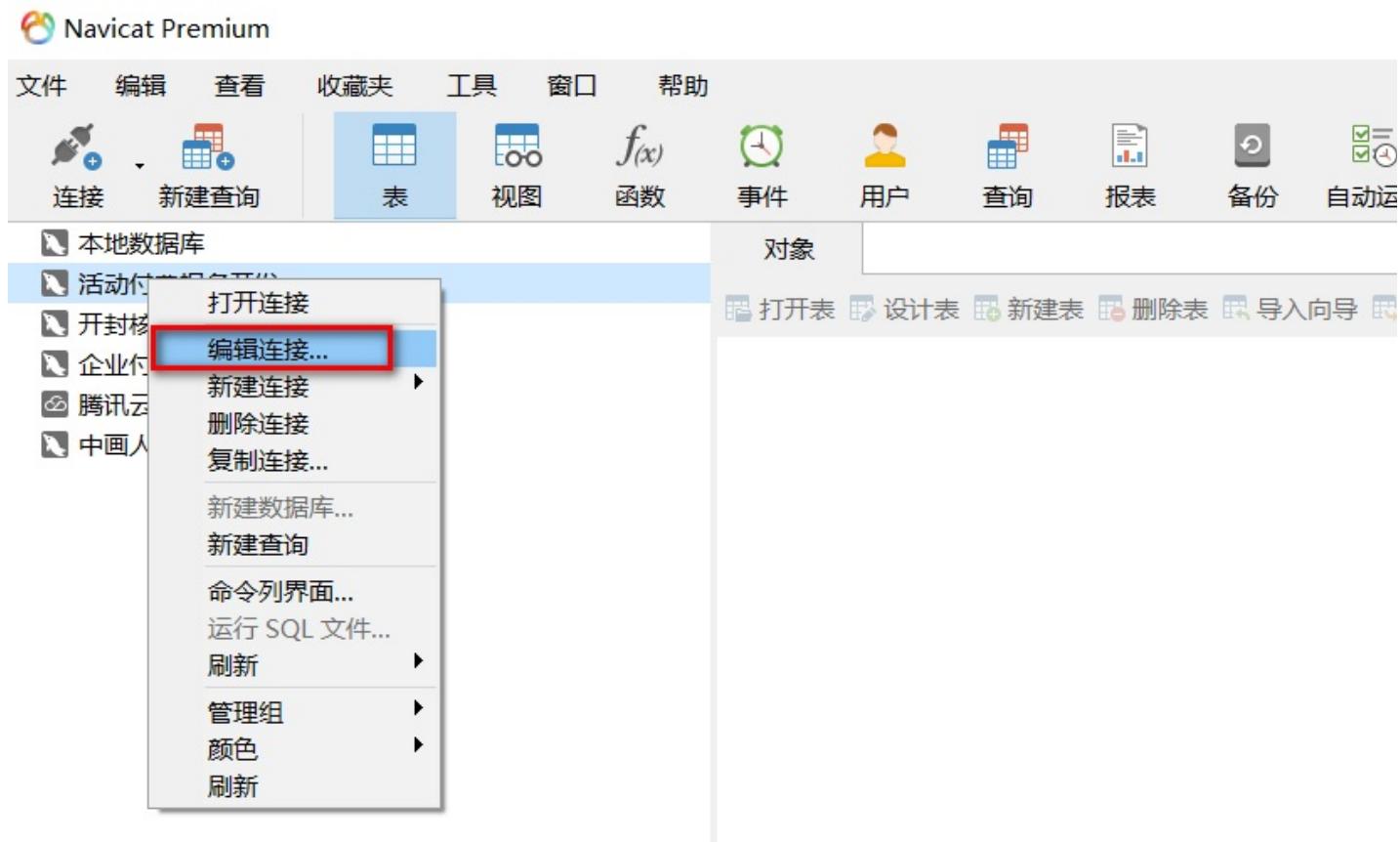
navicat 保持连接状态时 停滞久了在使用navicat会卡死

## 解决方案

先关闭连接，再右键点击所需要设置的链接，

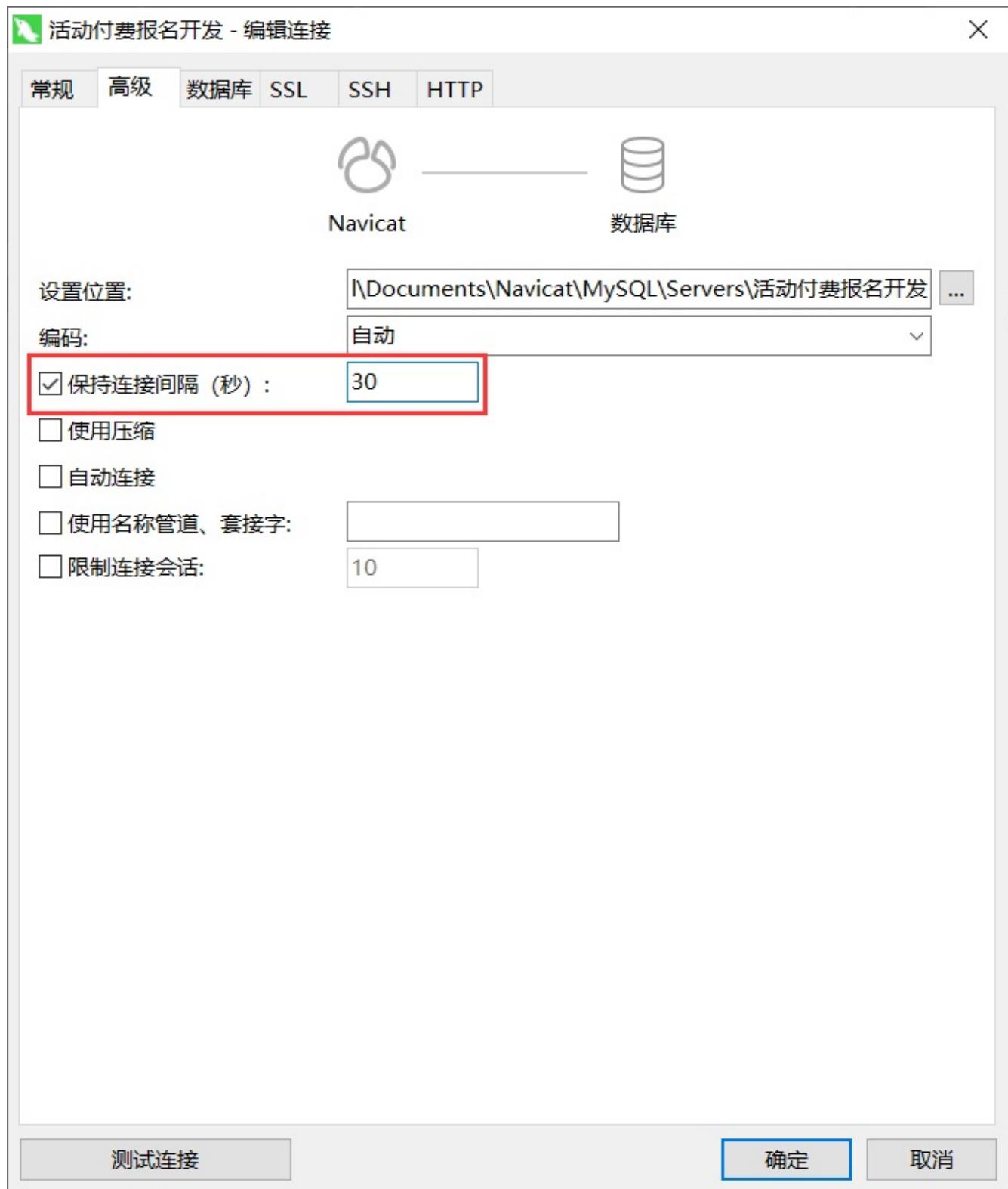
进入编辑连接，进入高级项，勾选保持连续间隔(秒)：时间设置短一些，比如：30秒

先关闭连接 才能编辑



勾选上保持连接间隔

我设置的 30秒 看心情自己定义即可 时间短一点



# 命令行

---

[简介](#)

[生成控制器](#)

[生成模型类](#)

[生成验证器](#)

[生成中间件](#)

[查看框架版本](#)

[清除缓存文件](#)

[生成应用目录](#)

# 简介

## 命令行指令

在cmd命令行中切换到 框架根目录 可以执行相关指令

## 命令行指令格式

```
php think 指令 参数
```

# 生成控制器

## 命令行创建控制器

- 单应用模式 `app/controller/User.php`

```
php think make:controller User
```

- 多应用模式 `app/admin/controller/User.php`

```
php think make:controller admin@User
```

## 默认生成的控制器类带有资源操作方法

- 生成不带有资源操作方法的控制器类文件

```
php think make:controller User --plain
```

# 生成模型类

## 命令行创建模型

- 单应用模式 `app/model/User.php`

```
php think make:model User
```

- 多应用模式 `app/admin/model/User.php`

```
php think make:model admin@User
```

# 生成验证器

## 命令行生成验证器类文件

- 单应用模式 `app/validate/User`

```
php think make:validate User
```

- 多应用模式 `app/index/validate/User`

```
php think make:validate index@User
```

# 生成中间件

## 命令行生成中间件

- app/middleware/Auth

```
php think make:middleware Auth
```

- app/middleware/admin/Auth

```
php think make:middleware admin/Auth
```

## 补充

- 无法通过命令行将中间件直接创建到应用下
- 但是可以手动放到应用目录下，可以正常使用（修改命名空间）

# 查看框架版本

version 指令 查看框架版本

```
php think version
```



A screenshot of a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The window shows the command 'E:\www\thinkphp\tp6.0>php think version' being run, followed by the output 'v6.0.2'. The window has standard minimize, maximize, and close buttons at the top right.

```
E:\www\thinkphp\tp6.0>php think version
v6.0.2
E:\www\thinkphp\tp6.0>
```

# 清除缓存文件

clear 指令

经测试发现单应用模式下才可以用，多应用模式下无效

# 生成应用目录

## 快速生成应用

- 使用前提：已经引入多应用模式扩展

```
// 快速生成一个demo应用  
php think build demo
```

```
// 快速生成一个index应用  
php think build index
```

```
// 快速生成一个admin应用  
php think build admin
```

## 自定义应用结构

- 新增一个 `app/build.php` 文件，文件内容如下
- 自定义命令行生成应用时需要创建的文件 [官方手册传送门](#)

```
return [  
    // 需要自动创建的文件  
    '__file__' => [],  
    // 需要自动创建的目录  
    '__dir__' => ['controller', 'model', 'view'],  
    // 需要自动创建的控制器  
    'controller' => ['Index'],  
    // 需要自动创建的模型  
    'model' => ['User'],  
    // 需要自动创建的模板  
    'view' => ['index/index'],  
];
```

# 模板引擎

---

[运算符](#)

[变量输出](#)

[输出替换](#)

[模板引擎](#)

[模板注释](#)

[模板分离](#)

[模板继承](#)

[内置标签](#)

# 运算符

## 三元运算符

- 模板中的写法

```
{$arr ?? 123}  
{$arr ?: 123}  
{$arr ? 123 : 456}
```

- 模板经过编译后

```
<?php echo isset($arr) ? htmlentities($arr) : 123; ?>  
<?php echo !empty($arr) ? htmlentities($arr) : 123; ?>  
<?php echo !empty($arr) ? 123 : 456; ?>
```

# 变量输出

## 普通变量

```
{$name}
```

```
<?php echo htmlentities($name); ?>
```

## 数组变量

```
 {$arr.name.email}
```

```
<?php echo htmlentities($arr['name']['email']); ?>
```

# 输出替换

## 模板输出替换 快速使用

```
// 模板输出字符串替换
'tpl_replace_string' => [
    '__STATIC__'      => '/static',
    '__INDEX__'       => '/static/index',
    '__ADMIN__'        => '/static/admin',
    // layuiadmin 静态资源
    '__LAYUIADMIN__'  => '/static/admin/layuiadmin',
    // authtree 权限树扩展组件
    '__AUTHTREE__'    => '/static/layuiadmin/lib/extend/authtree',
],
]
```

## 什么是模板输出替换

对模板文件输出的内容进行字符替换，

定义后在渲染模板或者内容输出的时候就会自动根据设置的替换规则自动替换。

## 全局模板字符串替换

常用格式：大写英文字母（前后各两个下划线）

在全局配置文件 config/view.php 中新增 tpl\_replace\_string 配置项

```

19     // 模板引擎普通标签结束标记
20     'tpl_end'      => '}',
21     // 标签库标签开始标记
22     'taglib_begin'  => '{',
23     // 标签库标签结束标记
24     'taglib_end'    => '}',
25
26
27
28     // 模板输出字符串替换
29     'tpl_replace_string' => [
30         '__STATIC__' => '/static'
31     ];
32

```

### 修改输出替换配置未生效

原因：框架的缓存问题，删除runtime目录下的temp目录即可

temp目录位置

单应用模式：runtime/temp  
多应用模式：runtime/应用名/temp

### 使用示例

- 配置输出替换

```

// 模板输出字符串替换
'tpl_replace_string' => [
    '__STATIC__' => '/static'
]

```

- 模板文件内容



A screenshot of a browser window. The address bar shows '127.0.0.1/thinkphp/tp603/public/'. The main content area displays the text 'location: \_\_STATIC\_\_' enclosed in a red rectangular box. The browser interface includes tabs for 'view.php' and 'index.html'.

- 浏览器输出内容

---

← → ⌛ ⓘ 127.0.0.1/thinkphp/tp603/public/

location: /static

# 模板引擎

- 安装 `think-view` 扩展会自动安装 `think-template` 依赖库
- 模板引擎相关参数配置
- `View::engine()` 动态切换模板引擎
- `View::config` 动态定义模板引擎配置

## 安装 `think-view` 扩展会自动安装 `think-template` 依赖库

默认只能支持PHP原生模板

如果需要使用 `thinkTemplate` 模板引擎，

需要安装 `think-view` 扩展（该扩展会自动安装 `think-template` 依赖库）。

## 模板引擎相关参数配置

在配置目录的 `view.php` 中自定义模板引擎相关配置项

```
return [
    // 模板引擎类型
    'type'          => 'Think',
    // 模板路径
    'view_path'     => './template/',
    // 模板后缀
    'view_suffix'   => 'html',
    // 模板文件名分隔符
    'view_depr'     => '/',
    // 模板引擎普通标签开始标记
    'tpl_begin'     => '{',
    // 模板引擎普通标签结束标记
    'tpl_end'       => '}',
    // 标签库标签开始标记
    'taglib_begin'  => '{',
    // 标签库标签结束标记
    'taglib_end'    => '}',
];
```

## `View::engine()` 动态切换模板引擎

```
// 使用内置PHP模板引擎渲染模板输出
// 表示当前视图的模板文件使用原生php进行解析
// 此时模板引擎不会被解析 被当做普通的文本内容
return View::engine('php')->fetch();
```

## View::config 动态定义模板引擎配置

```
// 改变当前操作的模板路径
View::config(['view_path' => 'mypath']);
return View::fetch();
```

# 模板注释

## 单行注释

```
{// 文字说明}
```

## 多行注释

```
/*  
文字说明  
*/
```

## 补充说明

- { 和注释标记 // 、 /\* 之间不能有空格
- 模板注释在生成编译缓存文件后会自动删除

# 模板分离

## 模板包含快速使用

```
{include file='public/left' /}  
{include file='public/header' /}
```

## 什么是模板分离

- 模板分离又称：模板包含、包含文件
- 在当前模版文件中包含其他的模版文件
- {include file='模块@控制器/操作,模块@控制器/操作' ...}

## 模板包含注意事项

- 包含文件根目录当前应用(模块)的视图目录
- 包含的模板文件中不能再使用模板布局或者模板继承

# 模板继承

---

[基础模板](#)

[区块设计](#)

# 基础模板

## 基础模板

基础模板即父模板，基础模板是用来被子模板继承的

## 模板继承

在基础模板中定义基础布局和区块（block）

子模板通过（extend）继承父模板，就可以对基础模板中定义的区块进行重载

## 模板继承的优势

设计基础模板中的区块（block）和子模板中替换这些区块

# 区块设计

## 区块 用法

在父模板中的定义区块，子模板重载父模板中的区块

# 内置标签

---

[volist](#)

[资源文件](#)

# volist

## volist 标签属性

属性	描述
name	必写属性，模板赋值的变量名称
id	必写属性，当前的循环变量
key	当前循环序号变量名称，默认为\$i，从1开始
offset	偏移量，默认为0，也就是从第一条数据开始显示
length	显示数量，默认显示所有

## volist 标签中内置的变量

变量	描述
\$i	当前循环序号，可通过key属性指定变量名称
\$key	当前循环变量的下标

# 资源文件

传统引入css、js文件都是使用 `link` 和 `script` 引入

```
<script type='text/javascript' src='/static/js/common.js'>
<link rel="stylesheet" type="text/css" href="/static/css/style.css" />
```

ThinkPHP 提供了专门的标签来简化此方式

- `css` 、 `js` 、 `load` 互为标签别名，作用一样

```
{css href="/static/css/style.css" /}

{js href="/static/js/common.js" /}

{load href="/static/js/common.js" /}
{load href="/static/css/style.css" /}
```

同时加载多个资源文件使用 逗号 分割即可

```
{load href="/static/js/common.js,/static/css/style.css" /}
```

补充说明

- `css` 标签也可以引入js文件（根据引入文件的后缀名判断的）但是不要这么用

```
{css href="/static/js/common.js" /}
```

# 助手函数

---

[app](#)

[url](#)

[input](#)

[redirect](#)

[validate](#)

[cookie](#)

# app

## app()

快速获取容器中的实例 支持依赖注入

- App类源码位置

```
vendor\topthink\framework\src\think\App.php
```

## 应用场景

- 获取当前ThinkPHP版本号

```
// app() 助手函数  
app()->version(); // 返回示例：6.0.2  
  
// 如果继承了基础控制器可以使用以下方式  
$this->app->version(); // 返回示例： 6.0.2  
  
// 在模板文件中可以直接调用  
{:app()->version()}
```

## App对象方法

方法	描述
version()	获取框架版本号，示例： 6.0.2
getRootPath()	获取应用根目录，即框架根目录 示例： E:\www\thinkphp\tp6.0

# url

## 快速使用

```
{:url('admin/article/edit')}
```

```
{:url('admin/article/edit', ['id' => $v.id])}
```

```
{:url('admin/article/edit', ['id' => $v['id']])}
```

## url 助手函数

```
public function list()
{
    // /admin/article/index.html
    echo url('admin/article/index'), '<br>';

    // /admin/article/index.html?id=10
    echo url('admin/article/index', ['id' => 10]), '<br>';
}
```

## 在模板文件中你可能会这么用

```
<a href="{:url('admin/article/edit', ['id' => $v.id])}">编辑</a>
```

```
<a href="{:url('admin/article/edit', ['id' => $v['id']])}">编辑</a>
```

# input

## input 助手函数

```
input('参数名[/变量修饰符]', '默认值', '过滤方法');
```

### 获取请求变量

```
// 获取任何请求类型的name参数值  
$name = input('name');  
  
// 获取get请求类型的所有参数及其参数值  
// 返回值：一维数组  
// 键名：参数名，键值：参数值  
$array = input('get.');//  
  
// 获取post请求类型的所有参数及其参数值  
// 返回值：一维数组  
// 键名：参数名，键值：参数值  
$array = input('post.');
```

### 变量修饰符

```
// 获取指定参数的值并将转为数字  
$id = input('id/d');
```

### 参数默认值

```
// 获取指定参数的值 没有获取到将返回默认值  
// 示例：如果id参数不存在，返回 666  
$id = input('id', 666);
```

### 过滤方法

```
// 获取指定参数的值再经过intval函数进行过滤  
$id = input('id', '', 'intval');
```

# redirect

## redirect 重定向助手函数

```
public function index()
{
    return redirect('index/login');
}
```

## 经测试

该重定向助手函数在 `BaseController.php` 的初始化方法 `initialize()` 中无法跳转

# validate

---

[validate\(\) 简介](#)

[文件上传验证](#)

[传入验证器类名](#)

[传入验证规则数组](#)

# validate() 简介

## validate() 助手函数

- 控制器和模型中都可以使用该助手函数
- 在模型中：推荐使用该函数
- 控制器中：推荐使用基础控制器提供的数据验证功能

## 源码位置

vendor\topthink\framework\src\helper.php

```
if (!function_exists('validate')) {
    /**
     * 生成验证对象
     * @param string|array $validate      验证器类名或者验证规则数组
     * @param array         $message       错误提示信息
     * @param bool          $batch         是否批量验证
     * @param bool          $failException 是否抛出异常
     * @return Validate
    */
    function validate($validate = '', array $message = [], bool $batch = false,
bool $failException = true): Validate
    {
        if (is_array($validate) || '' === $validate) {
            $v = new Validate();
            if (is_array($validate)) {
                $v->rule($validate);
            }
        } else {
            if (strpos($validate, '.')) {
                // 支持场景
                [$validate, $scene] = explode('.', $validate);
            }
        }

        $class = false !== strpos($validate, '\\') ? $validate : app()->par
seClass('validate', $validate);

        $v = new $class();

        if (!empty($scene)) {
            $v->scene($scene);
        }
    }
}
```

```
    return $v->message($message)->batch($batch)->failException($failException);
}
}
```

# 文件上传验证

## 传入验证规则数组

```
// img 是文件字段域名
$file = request()->file('img');

try {
    // 使用验证器验证上传的文件
    validate(
        [
            'file' => [
                // 限制文件大小(单位b), 这里限制为4M
                'fileSize' => 4 * 1024 * 1024,
                // 限制文件后缀, 多个后缀以英文逗号分割
                'fileExt' => 'gif,jpg'
            ]
        ],
        [
            [
                'file.fileSize' => '文件太大',
                'file.fileExt' => '不支持的文件后缀',
            ]
        ]
    )->check(['file' => $file]);

    // 上传到本地服务器
    // .....
    $savename = \think\facade\Filesystem::disk('public')->putFile('img', $file);

    $path = \think\Facade\Filesystem::getDiskConfig('public', 'url') . str_replace('\\', '/', $savename);
} catch (\think\exception\ValidateException $e) {
    return $e->getMessage();
}
```

## 传入验证器类名

- 验证器类如果还有其他字段的验证
- 则可以指定验证场景使其只进行文件上传验证

```
<?php
namespace app\validate;

use think\Validate;
```

```

class User extends Validate
{
    /**
     * 定义验证规则
     * 格式：'字段名' => ['规则1', '规则2'...]
     *
     * @var array
     */
    protected $rule = [
        'username' => 'require',
        'password' => 'require',
        'file' => [
            // 限制文件大小(单位b), 这里限制为4M
            'fileSize' => 4 * 1024 * 1024,
            // 限制文件后缀, 多个后缀以英文逗号分割
            'fileExt' => 'gif'
        ]
    ];

    /**
     * 定义错误信息
     * 格式：'字段名.规则名' => '错误信息'
     *
     * @var array
     */
    protected $message = [
        'username.require' => '用户名不能为空',
        'password.require' => '登录密码不能为空',
        'file.fileSize' => '文件太大',
        'file.fileExt' => '不支持的文件后缀',
    ];
}

/**
 * 文件上传
*/
public function sceneAvatar()
{
    return $this->only(['file']);
}
}

```

```

// pic 文件字段域名称
$file = request()->file('pic');
try {
    validate('app\validate\User')
        ->scene('avatar')
        ->check(['file' => $file]);
} catch (\think\exception\ValidateException $e) {

```

```
dump('文件上传验证失败: ' . $e->getMessage());
}
```

# 传入验证器类名

## 传入验证器类名进行验证

- 验证器类位置：`app\validate\User.php`

```
try {
    validate('app\validate\User')->check($data);
} catch (\think\exception\ValidateException $e) {
    die('验证失败: ' . $e->getError());
}
```

- 也可以使用以下方式
- 因为 `\app\validate\User::class` 返回 `app\validate\User`

```
try {
    validate(\app\validate\User::class)->check($data);
} catch (\think\exception\ValidateException $e) {
    die('验证失败: ' . $e->getError());
}
```

## 错误信息

- 第二个参数省略或传入空数组时使用的是验证器类的错误信息
- 第二个参数的错误信息和验证器类错误信息重复时，优先使用前者

```
try {
    validate('app\validate\User', [
        'password.require' => '抱歉，密码不能为空'
    ])->check($data);
} catch (\think\exception\ValidateException $e) {
    die('验证失败: ' . $e->getError());
}
```

## 批量验证

- 第三个参数为 `true`，代表进行批量验证

```
try {
    validate('app\validate\User.add', [], true)->check($data);
} catch (\think\exception\ValidateException $e) {
```

```
    dump($e->getError());
}
```

- 批量验证时 `$e->getError()` 返回关联数组

```
^ array:2 [▼
  "username" => "用户名不能为空"
  "password" => "登录密码不能为空"
]
```

### 验证场景 (示例 : add )

- 方案一：验证场景写在类名之后

```
try {
    validate('app\validate\User.add')->check($data);
} catch (\think\exception\ValidateException $e) {
    die('验证失败: ' . $e->getError());
}
```

- 方案二：使用 `sence()` 指定验证场景

```
try {
    validate('app\validate\User')->sence('add')->check($data);
} catch (\think\exception\ValidateException $e) {
    die('验证失败: ' . $e->getError());
}
```

# 传入验证规则数组

## 传入验证器规则数组进行验证

```
$rules = [
    'username' => 'require',
    'password' => 'require',
];

try {
    validate($rules)->check($data);
} catch (\think\exception\ValidateException $e) {
    dump('验证失败: ' . $e->getError());
}
```

## 错误信息

```
$rules = [
    'username' => 'require',
    'password' => 'require',
];
$messages = [
    'username.require' => '用户名不能为空',
];
try {
    validate($rules, $messages)->check($data);
} catch (\think\exception\ValidateException $e) {
    dump('验证失败: ' . $e->getError());
}
```

## 批量验证

- 第三个参数传入 true 代表进行批量验证 返回数组

```
try {
    validate($rules, $messages, true)->check($data);
} catch (\think\exception\ValidateException $e) {
    dump('错误信息数组: ', $e->getError());
}
```

## 是否抛出异常

- 第四个参数为false时，即使验证失败也不抛出异常

```
try {
    validate($rules, $messages, true, false)->check($data);
} catch (\think\exception\ValidateException $e) {
    dump('错误信息数组：', $e->getError());
}
```

### 验证场景

此方式不需要验证场景，也没有意义，传入什么验证规则就验证什么

# cookie

## cookie

系统提供了 `cookie` 助手函数用于基本的 `cookie` 操作

### 设置 cookie

```
// cookie('名称', '值', '有效期'); 有效期单位:秒  
cookie('name', 'value', 3600);
```

### 获取指定cookie名称的值

```
// 返回该cookie名称对应的值, 没有找到该名称返回 NULL  
cookie('name');
```

### 删除 cookie

```
// cookie值置为null  
cookie('name', null);
```

# 常用功能

---

[无限极分类](#)

[继承基础控制器的登陆验证](#)

# 无限极分类

## 无限级栏目查询

```
<?php
namespace app\model;

use think\Model;

class Category extends Model
{
    public function cateTree()
    {
        $cate = self::select();

        $cate = $this->_sort($cate);

        return $cate;
    }

    private function _sort($data, $pid = 0, $level = 0)
    {
        static $arr = [];

        foreach ($data as $k => $v) {
            if ($v['pid'] == $pid) {
                $v->level = $level;
                $arr[] = $v->toArray();
                $this->_sort($data, $v['id'], $level + 1);
            }
        }

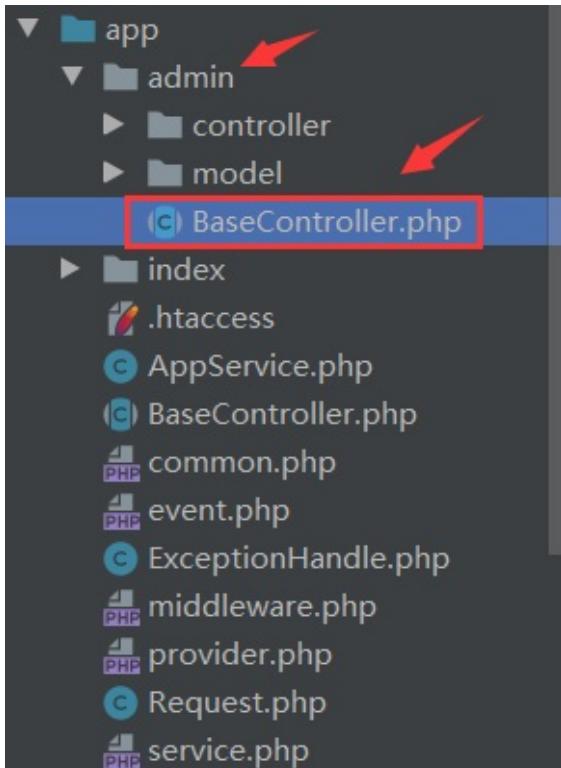
        return $arr;
    }
}
```

# 继承基础控制器的登陆验证

## 继承基础控制器的登陆验证

复制 `BaseController.php` 到后台模块下

- 别忘了修改 `BaseController.php` 的命名空间



在 `BaseController.php` 的初始化方法中校验登陆状态

```
// 初始化
protected function initialize()
{
    // 登陆状态校验
    if (empty(session('uid')) || empty(session('username'))) {
        die('请先登录');
    }
}
```

在需要校验登陆状态的控制器中继承 `BaseController.php`

```
<?php
namespace app\admin\controller;
```

```
use think\facade\Db;
use app\admin\BaseController;

/**
 * 管理员控制器 继承 BaseController
 */
class Admin extends BaseController
{
    /**
     * 管理员列表
     */
    public function index()
    {
        // 查询管理员表数据
        $data = Db::table('admin')->select();

        return view('list', ['data' => $data]);
    }
}
```

# 常用扩展

---

[flc/dysms](#)

[qiniu/php-sdk](#)

[firebase/php-jwt](#)

[symfony/var-dumper](#)

[phpmailer/phpmailer](#)

[liliwei/thinkphp-jump](#)

[topthink/think-captcha](#)

[phoffice/phpspreadsheet](#)

# flc/dysms

## 阿里大鱼短信验证

<https://packagist.org/packages/flc/dysms>

```
composer require flc/dysms
```

```
public function sendSms()
{
    $config = [
        'accessKeyId'      => 'LTAI4FxkezJcQyzMnixxxx',
        'accessKeySecret'  => 'ezh3hfslU01By31XJrel3lLxxxx',
    ];

    $client  = new Client($config);
    $sendSms = new SendSms;

    // 接受验证码的手机号
    $sendSms->setPhoneNumbers('150xxxx');

    // 签名
    $sendSms->setSignName('it社区');

    // 模板码
    $sendSms->setTemplateCode('SMS_178456xxx');

    // 模板变量
    $sendSms->setTemplateParam(['code' => rand(100000, 999999)]);

    $sendSms->setOutId('demo');

    // 执行发送验证码
    $result = $client->execute($sendSms);

    // 打印结果
    print_r($result);
}
```

# qiniu/php-sdk

---

# 简介

## 七牛云官方资源

- 七牛云官方SDK [七牛云官方文档](#)

```
composer require qiniu/php-sdk
```

# 上传示例

## 上传示例

```
use Qiniu\Auth;
use Qiniu\Storage\UploadManager;

/**
 * 文件上传到七牛云
 * @return 在七牛云上的文件名
 */
public function upload()
{
    // 临时文件路径 $_FILES['pic']['tmp_name']
    $tmp = request()->file('pic')->getRealPath();

    // 用于签名的公钥和私钥
    $accessKey = 'd5pCObiT0CXMLN9EKvjyQImSyqD3z2xbzxxxxx';
    $secretKey = 'KxncccQTEu-jZHLiUnfcVSXaVN3jAJSxxxxxx';

    // 初始化鉴权对象
    $auth      = new Auth($accessKey, $secretKey);

    // 存储空间名称
    $bucket    = 'test';
    // 生成uploadToken
    $token     = $auth->uploadToken($bucket);

    // 上传管理类 构建UploadManager对象
    $uploadMgr = new UploadManager();

    // token 七牛云图片名称 图片名称
    $name      = time() . '.jpg';
    $info      = $uploadMgr->putFile($token, $name, $tmp);

    // 上传到七牛云后的新名称
    return $info[0]['key'];
}
```

# firebase/php-jwt

JWT 接口鉴权扩展

<https://packagist.org/packages/firebase/php-jwt>

# symfony/var-dumper

symfony/var-dumper 传送门

<https://packagist.org/packages/symfony/var-dumper>

通过composer安装 symfony/var-dumper 扩展

```
composer require symfony/var-dumper
```

```
<?php
require_once 'vendor/autoload.php';

$data = [
    'name' => 'zhangsan',
    'password' => 123456
];
dump($data);
```

效果

```
^ array:2 [▼
  "name" => "zhangsan"
  "password" => 123456
]
```

# phpmailer/phpmailer

phpmail 发送邮件

<https://packagist.org/packages/phpmailer/phpmailer>

# liliuwei/thinkphp-jump

---

[扩展介绍](#)

[下载扩展](#)

[使用方法](#)

# 扩展介绍

liliuwei/thinkphp-jump

- TP5 的经典跳转提示在TP6中已经取消
- 通过composer下载该扩展可以在TP6中使用TP5的跳转提示操作

ThinkPHP的经典跳转提示

- \$this->success('登陆成功')



## 登陆成功

页面自动 跳转 等待时间： 2

- \$this->error('登陆失败');



## 登陆失败

页面自动 跳转 等待时间： 3

# 下载扩展

引入 liliuwei/thinkphp-jump 扩展

在应用根目录执行以下命令

```
composer require liliuwei/thinkphp-jump
```

```

- thinkphp
  - bbc
    + app
    + config
    + extend
    + public
    + route
    + runtime
    + vendor
    + view
      - .env
      - .gitignore
      - .travis.yml
      - composer.json
      - composer.lock
      - LICENSE.txt
      - README.md
      - think
MINGW64:/e/www/thinkphp/bbc
$ composer require liliuwei/thinkphp-jump
Using version ^1.4 for liliuwei/thinkphp-jump
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing liliuwei/thinkphp-jump (v1.4): Downloading (100%)
Writing lock file
Generating autoload files
> @php think service:discover
Succeed!
> @php think vendor:publish
File E:\www\thinkphp\bbc\config\captcha.php exist!
File E:\www\thinkphp\bbc\config\trace.php exist!
Succeed!
Dell@DESKTOP-LISFFFH MINGW64 /e/www/thinkphp/bbc
$ 
```

引入该扩展后会在全局配置目录生成 jump.php 文件

```

return [
    // 默认跳转页面对应的模板文件
    'dispatch_success_tmpl' => app()->getRootPath().'/vendor/liliuwei/thinkphp-
jump/src/tpl/dispatch_jump.tpl',
    'dispatch_error_tmpl'   => app()->getRootPath().'/vendor/liliuwei/thinkphp-
jump/src/tpl/dispatch_jump.tpl',
]; 
```

# 使用方法

## thinkphp-jump 的两种使用方法

- 方案一：在控制器中引入 Trait
- 方案二：在基础控制器中引入 Trait，控制器继承基础控制器

### 方案一：在控制器中引入 Trait

```
use \liliuwei\think\Jump;
```

```
<?php
namespace app\controller;

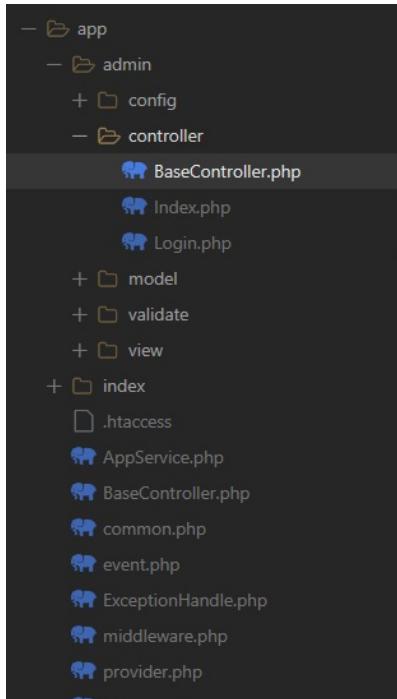
class Index
{
    use \liliuwei\think\Jump;

    public function index()
    {
        //return $this->error('error');
        //return $this->success('success', 'index/index');
        //return $this->redirect('/admin/index/index');
        return $this->result(['username' => 'liliuwei', 'sex' => '男']);
    }
}
```

### 方案二：在基础控制器 BaseController.php 中引入 Trait

如果继承了的 BaseController

在 BaseController.php 文件的最后添加以下内容即可



```
57     /**
58      * 验证数据
59      * @access protected
60      * @param array $data 数据
61      * @param string|array $validate 验证器名或者验证规则数组
62      * @param array $message 提示信息
63      * @param bool $batch 是否批量验证
64      * @return array|string|true
65      * @throws ValidateException
66      */
67     protected function validate(array $data, $validate, array
68     {
69     }
70 }
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94     use \liliuwei\think\Jump;
95 }
96 }
```

# topthink/think-captcha

---

[安装扩展](#)

[验证码显示](#)

[更换验证码](#)

[验证码校验](#)

[验证码配置](#)

[自定义验证码](#)

# 安装扩展

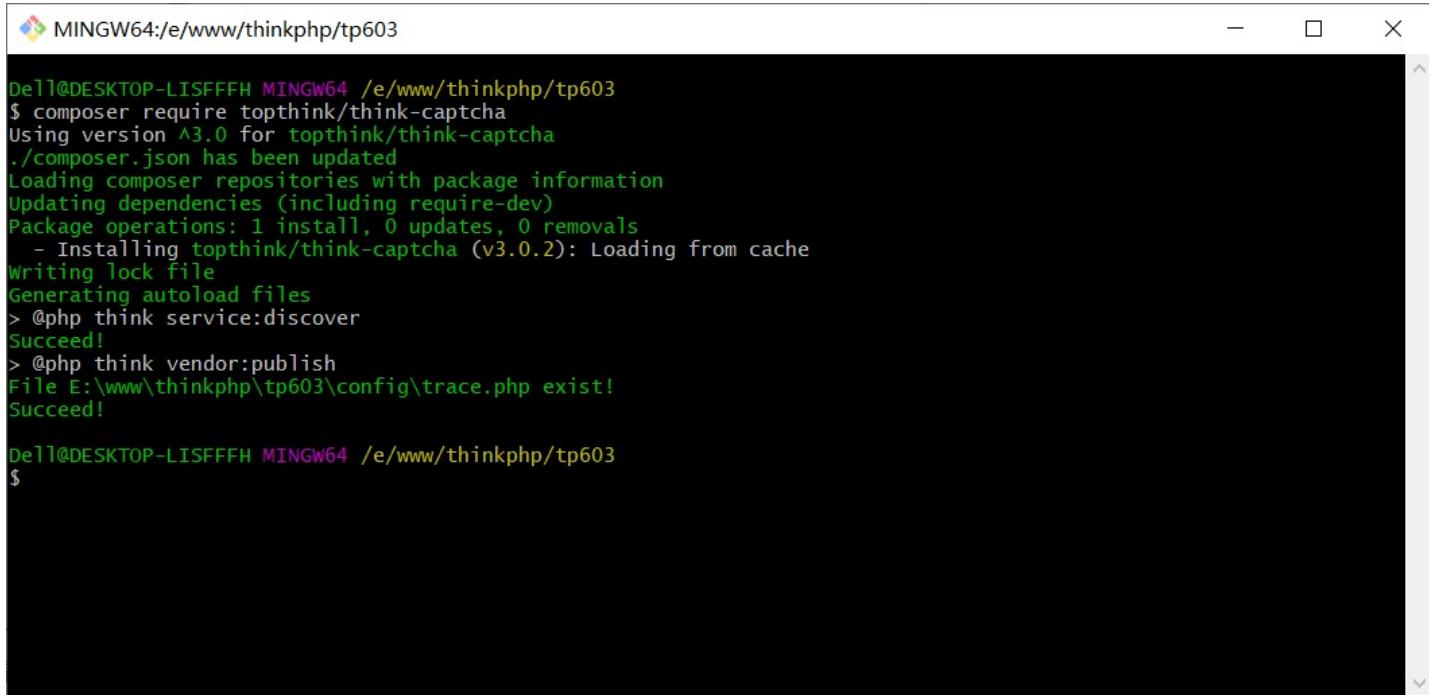
## topthink/think-captcha 简介

TThinkPHP 官方提供的验证码扩展包

topthink/think-captcha 传送门：<https://packagist.org/packages/topthink/think-captcha>

## 通过composer安装 think-captcha 扩展

```
composer require topthink/think-captcha
```

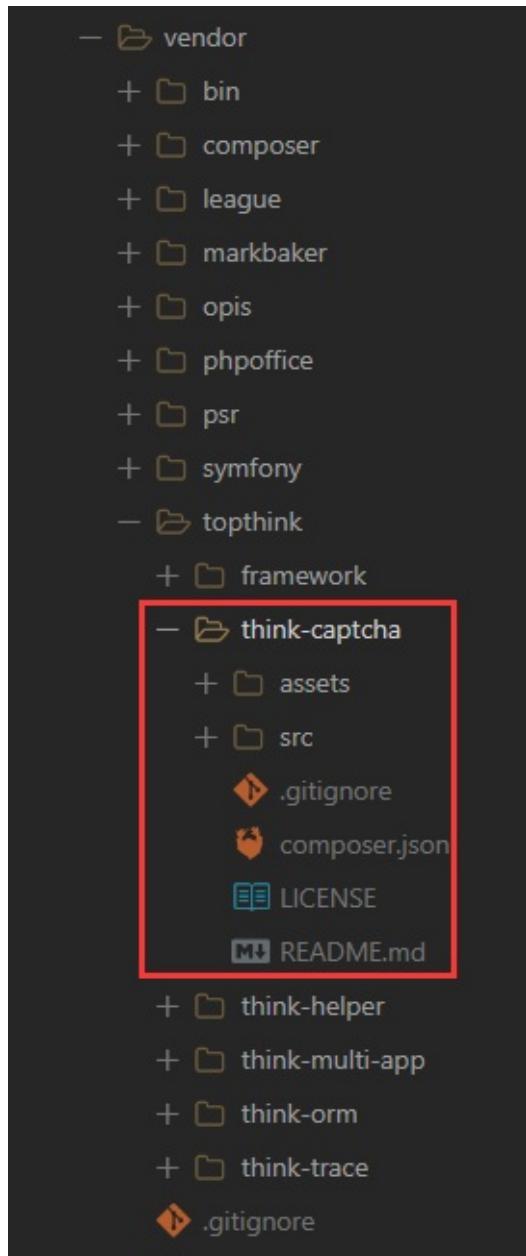


```
MINGW64:/e/www/thinkphp/tp603
Dell@DESKTOP-LISFFFH MINGW64 /e/www/thinkphp/tp603
$ composer require topthink/think-captcha
Using version ^3.0 for topthink/think-captcha
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Installing topthink/think-captcha (v3.0.2): Loading from cache
Writing lock file
Generating autoload files
> @php think service:discover
Succeed!
> @php think vendor:publish
File E:\www\thinkphp\tp603\config\trace.php exist!
Succeed!

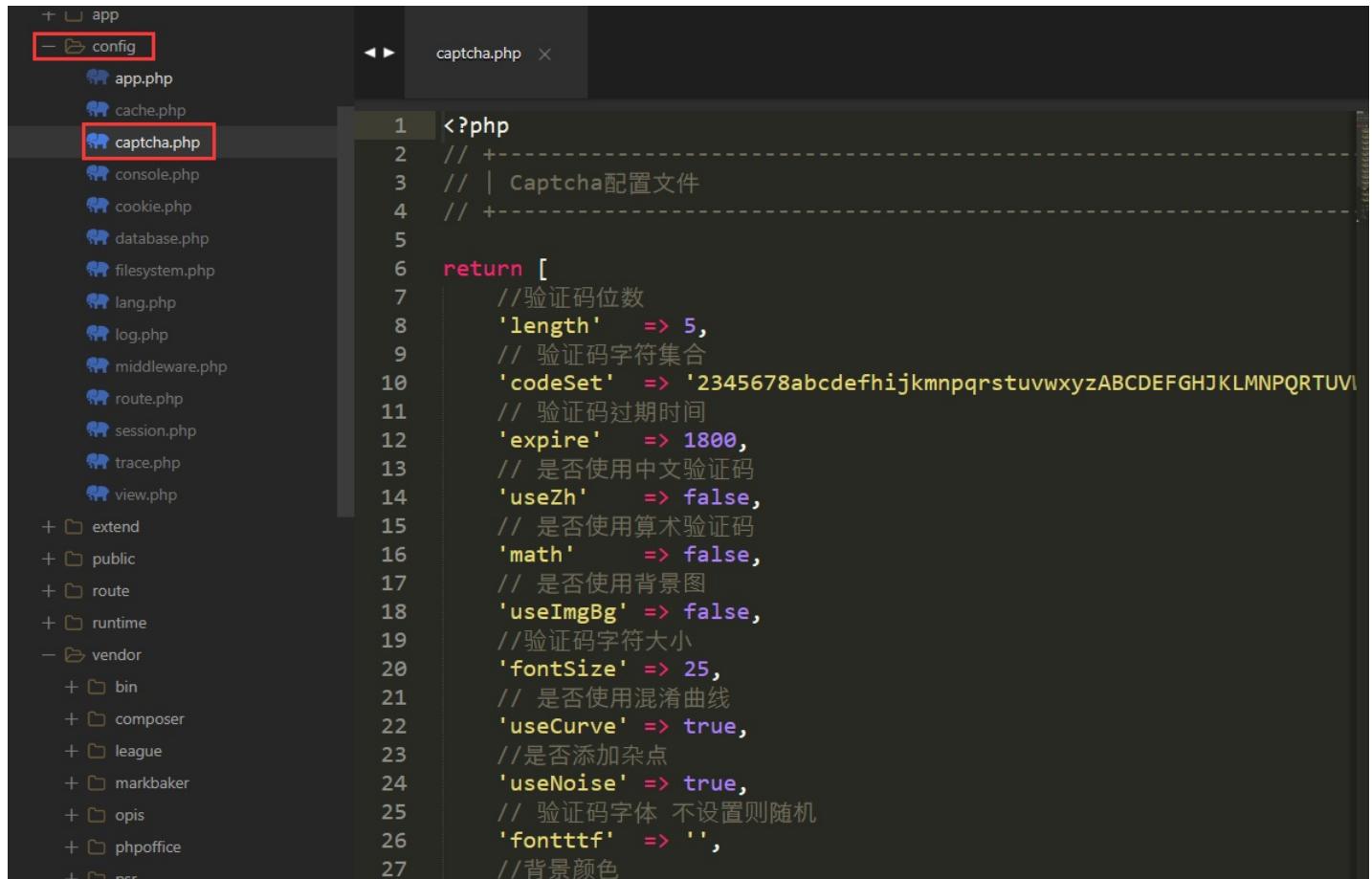
Dell@DESKTOP-LISFFFH MINGW64 /e/www/thinkphp/tp603
$
```

## think-captcha 扩展库安装后框架目录结构的变动

在 vendor/topthink 目录下生成 think-captcha 目录



在全局配置目录下生成 `captcha.php` 全局配置文件



```
1 <?php
2 // -----
3 // | Captcha配置文件
4 // -----
5
6 return [
7     //验证码位数
8     'length' => 5,
9     //验证码字符集合
10    'codeSet' => '2345678abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ',
11    //验证码过期时间
12    'expire' => 1800,
13    //是否使用中文验证码
14    'useZh' => false,
15    //是否使用算术验证码
16    'math' => false,
17    //是否使用背景图
18    'useImgBg' => false,
19    //验证码字符大小
20    'fontSize' => 25,
21    //是否使用混淆曲线
22    'useCurve' => true,
23    //是否添加杂点
24    'useNoise' => true,
25    //验证码字体 不设置则随机
26    'fontttf' => '',
27    //背景颜色
```

# 验证码显示

## 验证码显示的两种方法

方法一：自带 img 标签

```
<div>{:captcha_img()}</div>
```

方法二：使用验证码路径(推荐)

```
<div></div>
```

# 更换验证码

点击验证码图片更换验证码

如果使用的`{:captcha_src()}`方式显示验证码可给标签添加`onclick`属性

点击验证码图片自动在路径中拼接随机数，`cursor: pointer;` 鼠标滑过图片样式：小手

```
<div></div>
```

# 验证码校验

## think-captcha 验证码校验

- 方案一：captcha\_check() 函数验证
  - 参考TP6.0官方手册： 扩展库 - 验证码 章节 [点击进入](#)
  - 方案一适合只返回验证 成功和失败 或 在模型中校验验证码 的情况
  - 方案二和方案三更加灵活, 可以返回不同的验证码错误信息 , 但是必须在控制器中使用
- 方案二：基础控制器的数据验证功能
  - \$this->validate() 执行数据验证
  - \think\exception\ValidateException 捕获抛出的异常错误信息
- 方案三：基础控制器和验证器进行校验
  - 定义验证器
  - \$this->validate() 执行数据验证
  - \think\exception\ValidateException 捕获抛出的异常错误信息

### 校验 think-captcha 验证码的前提条件

必须开启 Session 否则即使验证码输入正确也无法校验成功

开启Session：在全局中间件定义文件 `app/middleware.php` 中取消 `Session初始化` 的注释

```

- app
  + admin
  + index
    .htaccess
    AppService.php
    BaseController.php
    common.php
    event.php
    ExceptionHandle.php
  middleware.php
  provider.php
  Request.php
  service.php
+ config
+ extend

      ◀ ▶ middleware.php ×

1 <?php
2 // 全局中间件定义文件
3 return [
4     // 全局请求缓存
5     // \think\middleware\CheckRequestCache::class,
6     // 多语言加载
7     // \think\middleware\LoadLangPack::class,
8     // Session初始化
9     \think\middleware\SessionInit::class
10 ];
11

```

### 方案一：captcha\_check() 函数验证

```

// $captcha 用户输入的验证码
// captcha_check($captcha) 成功 true 失败 false

```

```
if(!captcha_check($captcha)){
    // think-captcha 验证码校验失败
}
```

## 方案二：基础控制器的数据验证功能

```
try {
    $this->validate([
        'vercode' => '用户输入的验证码'
    ], [
        'vercode|验证码' => 'require|min:4|captcha'
    ], [
        'vercode.min'      => '验证码长度不能低于4位',
        'vercode.captcha' => '您输入的验证码有误',
    ]);
} catch (\think\exception\ValidateException $e) {
    return $e->getError();
}
```

## 方案三：基础控制器和验证器进行校验

- 定义验证器

```
<?php
namespace app\admin\validate;

use think\Validate;

class Admin extends Validate
{
    protected $rule = [
        'vercode|验证码'=>'require|min:4|captcha'
    ];

    protected $message = [
        'vercode.min'      => '验证码长度不能低于4位',
        'vercode.captcha' => '您输入的验证码有误',
    ];
}
```

- 执行校验

```
try {
    $this->validate(input('post.'), 'app\admin\validate\Admin');
} catch (\think\exception\ValidateException $e) {
```

```
    return $e->getError();  
}
```

# 验证码配置

## 验证码配置

全局验证码配置

```
config/captcha.php
```

应用验证码配置

```
app/应用/config/captcha.php
```

## 生成多个不同设置的验证码

```
<?php  
  
// verify 验证码名称  
return [  
    'verify' => [  
        'codeSet'=>'1234567890'  
    ]  
];
```

指定验证码名称

```
return Captcha::create('verify');
```

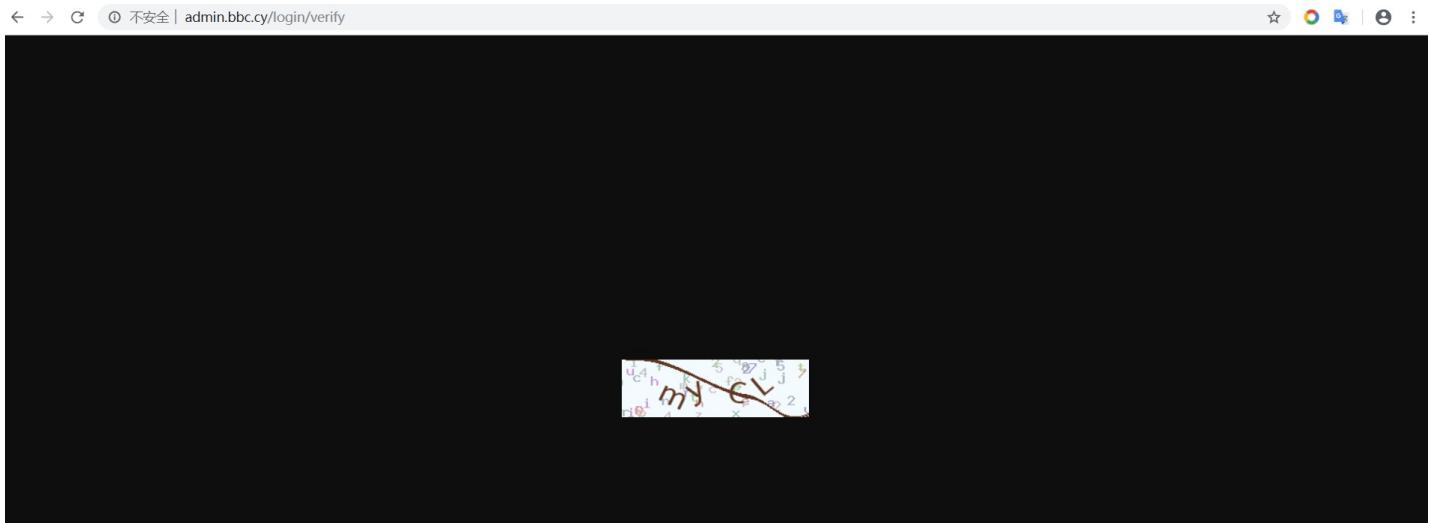
# 自定义验证码

## 自定义 think-captcha 验证码

- 在控制器方法中定义验证码方法

```
public function verify()  
{  
    return \think\captcha\facade\Captcha::create();  
}
```

- 访问控制器方法 参考TP6.0官方手册：[扩展库](#) - [验证码](#) 章节 [点击进入](#)



可以给验证码地址注册路由，使用路由显示验证码

```
Route::get('verify', 'index/verify');
```

```

```

# phponline/phpspreadsheet

---

[数据写入表格](#)

[读取表格数据](#)

# 数据写入表格

## phpspreadsheet

```
composer require phpoffice/phpspreadsheet
```

<https://packagist.org/packages/phpoffice/phpspreadsheet>

## 在安装前先查看当前环境 `fileinfo` 扩展有没有开启

如果没有开启 `fileinfo` 扩展安装该扩展库会报错

## 导出Excel表格封装

```
use PhpOffice\PhpSpreadsheet\IOFactory;
use PhpOffice\PhpSpreadsheet\Spreadsheet;
```

```
/**
 * 导出Excel表格 Xlsx格式(2007版)
 *
 * @author liang <23426945@qq.com>
 * @datetime 2019-12-22
 *
 * @param array $title 表头单元格内容
 * @param array $data 从第二行开始写入的数据
 * @param string $path Excel文件保存位置,路径中的目录必须存在
 *
 * @return null 没有设定返回值
 */
private function excel($title = [], $data = [], $path = '') {
    // 获取Spreadsheet对象
    $spreadsheet = new Spreadsheet();
    $sheet = $spreadsheet->getActiveSheet();

    // 表头单元格内容 第一行
    $titCol = 'A';
    foreach ($title as $value) {
        // 单元格内容写入
        $sheet->setCellValue($titCol . '1', $value);
        $titCol++;
    }
}
```

```
// 从第二行开始写入数据
$row = 2;
foreach ($data as $item) {
    $dataCol = 'A';
    foreach ($item as $value) {
        // 单元格内容写入
        $sheet->setCellValue($dataCol . $row, $value);
        $dataCol++;
    }
    $row++;
}

$writer = IOFactory::createWriter($spreadsheet, 'Xlsx');

$result = $writer->save($path . '.xlsx');
}
```

## 调用示例

```
$data = Db::table('ay_config')->select();

$title = ['id', '字段名', 'value值', '类型', '排序', '描述'];

$this->excel($title, $data, $filename = './excel/03');
```

# 读取表格数据

## 读取表格数据示例

```
use PhpOffice\PhpSpreadsheet\IOFactory;

// 创建读操作
$reader = IOFactory::createReader('Xlsx');

// 打开文件 载入excel表格
$spreadsheet = $reader->load($_FILES['excel']['tmp_name']);

// 获取活动工作簿
$sheet = $spreadsheet->getActiveSheet();

// 获取内容的最大列 如 D
$highest = $sheet->getHighestColumn();

// 获取内容的最大行 如 4
$row = $sheet->getHighestRow();

// 用于存储表格数据
$data = [];
for ($i = 2; $i <= $row; $i++) {
    // D 1234 C 123
    $t1 = $sheet->getCellByColumnAndRow(1, $i)->getValue();
    $t2 = $sheet->getCellByColumnAndRow(2, $i)->getValue();
    $t3 = $sheet->getCellByColumnAndRow(3, $i)->getValue();
    $t4 = $sheet->getCellByColumnAndRow(4, $i)->getValue();

    $data[] = [$t1, $t2, $t3, $t4];
}

var_dump($data);
```

# 更新日志

---

[2020-06](#)

[2020-05](#)

[2020-04](#)

[2020-03](#)

[2020-02](#)

[2020-01](#)

[2019-12](#)

# 2020-06

2020 - 06 - 08

## 细节调整

新增 杂项 - layuiAdmin单页版 - 数据表格中的删除

修改 模型 - 删除 - destroy

新增 模型 - 新增 - create

2020 - 06 - 04

新增 杂项 - layuiadmin单页版 - 隐藏trace

# 2020-05

2020 - 05 - 31

修改 序言 - \*

2020 - 05 - 28

新增 常用扩展 - firebase/php-jwt

细节调整

2020 - 05 - 26

新增 常用功能 - 无限级分类

细节调整

2020 - 05 - 20

修改 序言 - ThinkPHP技术群

2020 - 05 - 18

新增 模板引擎 - 内置标签 - 资源文件

修改 数据库 - 分页查询 - \*

2020 - 05 - 17

修改 模型 - 模型属性

新增 数据库 - 连接数据库

2020 - 05 - 16

新增 数据库 - 分页查询 - css代码

2020 - 05 - 14

新增 杂项 - layuiadmin - 单页版部署

删除 杂项 - layuiadmin - \*

2020 - 05 - 13

- 删除 杂项 - layuiadmin - 获取选中的id

◦ 移步此文档 [layuiadmin 章节](#)

2020 - 05 - 09

---

细节调整

2020 - 05 - 06

---

新增 验证 - 内置验证规则 - file

修改 验证 - 验证器类成员 - \*

修改 助手函数 - validate

修改 助手函数 - app

2020 - 05 - 05

---

新增 模板引擎 - 内置标签

2020 - 05 - 01

---

修改 杂项 - 文件上传 - \*

细节调整

# 2020-04

2020 - 04 - 28

新增 命令行 - 生成应用目录

新增 数据库 - 分页查询

2020 - 04 - 25

新增 路由 - 路由别名

2020 - 04 - 24

新增 常用扩展库 - phpoffice/phpspreadsheet - 读取表格数据

新增 杂项 - 文件上传 - 获取默认磁盘名称

新增 杂项 - 文件上传 - 获取磁盘配置

2020 - 04 - 21

新增 模型 - 获取器 - 追加获取器

新增 模型 - 模型关联 - 远程一对多关联

新增 模型 - 模型关联 - 关联删除

细节调整

2020 - 04 - 19

新增 数据库 - 聚合查询

新增 模型 - 模型输出

移动 视图 - \* 到 模板引擎 - \*

新增 模板引擎

细节调整

2020 - 04 - 15

新增 模型 - 模型关联 - 关联输出

新增 模型 - 模型关联 - 关联统计

2020 - 04 - 12

新增 模型 - 模型关联 - 关联预载入

新增 模型 - 模型属性

新增 模型 - 删除

新增 数据库 - 链式操作 - where

新增 模型关联 - 一对关联 - hasOne

2020 - 04 - 10

---

新增 模型 - 获取器 - 获取原始数据 - 应用示例

新增 模型关联 - 一对关联 - 绑定属性到父模型 - 应用示例

细节调整

2020 - 04 - 09

---

新增 命令行

2020 - 04 - 08

---

新增 模型 - 自动时间戳 - \*

新增 模型 - 更新 - \*

新增 模型 - 新增 - \*

细节调整

2020 - 04 - 07

---

新增 数据库 - 更新 - save

新增 数据库 - 更新 - update

细节调整

2020 - 04 - 06

---

删除 序言 - 关于QQ 群

移动 序言 - 感谢贡献者 纳入 序言 - 关于文档

新增 基础 - 多应用模式目录结构

新增 基础 - 单应用模式目录结构

新增 术语 - 应用目录

删除 镜像

移动 镜像 - 阿里云镜像 到 安装 - 阿里云镜像

新增 常用扩展库 - qiniu/php-sdk - 上传示例

重命名 架构 为 基础

## 细节调整

2020 - 04 - 05

删除 安装 - 服务器部署

重命名 安装 - 安装TP6.0 为 安装 - 创建项目

修改 序言 - 关于文档

新增 序言 - ThinkPHP技术群 为提高质量 设置付费入群

## 细节调整

2020 - 04 - 04

新增 数据库 - 事务操作

新增 模型 - 模型关联 - 一对多关联

新增 术语 - 模型关联

新增 模型 - 模型关联 - 一对一关联

新增 模型 - 搜索器

2020 - 04 - 03

修改 杂项 - layui - 文件上传

2020 - 04 - 01

重命名 第三方扩展 为 常用扩展库

重命名 更新日志 为 文档更新日志

## 细节调整

# 2020-03

2020-03-31

- 移动 layuiadmin 到 杂项 - layuiadmin
- 移动 layui 到 杂项 - layui
- 新增 请求 - 请求信息
- 新增 模型 - 只读字段
- 移动 文件上传 到 杂项 - 文件上传
- 移动 Session 到 杂项 - Session
- 新增 杂项 - Cookie
- 新增 助手函数 - cookie
- 修改 验证 - 执行数据验证 - validate助手函数验证
- 修改 助手函数 - validate
- 细节调整：重命名章节、修改某个小地方

2020-03-30

- 新增 layuiadmin - 引入authtree第三方扩展组件
- 新增 助手函数 - app
- 细节调整

2020-03-29

- 修改 序言
- 细节调整

2020-03-25

- 新增 序言 - 感谢贡献者
- 将 视图 - 视图渲染 中的 视图目录 独立为单独的章节：视图 - 视图目录
- 修改 视图 - 视图扩展
- 其他细节调整

2020-03-23

- 移动 第三方扩展 - topthink/think-captcha - layuiadmin 到 layuiAdmin -

登陆页面的think-captcha验证码

- 修改 序言 - 关于QQ群
- 修改 第三方扩展 - tophink/think-captcha - 自定义验证码

2020-03-22

- 修改 第三方扩展 - liliuwei/thinkphp-jump
- 修改 tophink/think-captcha - 验证码校验
  - think-captcha 验证码校验的三种方案
- 重命名 tophink/think-captcha - think-captcha 为 tophink/think-captcha - 安装扩展

2020-03-20

- 修改 路由 - 多应用路由
- 新增 术语 - pathinfo方式路由
- 新增 路由 - 注册路由
- 新增 路由 - 路由简介

2020-03-19

- 修改 安装 - 服务器部署
- 修改 安装 - nginx伪静态
- 修改 术语 - 应用根目录
- 调整 第三方扩展 - liliuwei/thinkphp-jump 目录结构

2020-03-16

- 新增 助手函数 - validate
- 新增 验证 - 执行数据验证 - 助手函数验证
- 新增 验证 - 内置验证规则
- 新增 layuiAdmin - layuiadmin iframe 布局方案

2020-03-15

- 新增 验证
- 新增 控制器 - 基础控制器
- 新增 安装 - 测试运行 ThinkPHP6.0

- 新增 视图 - 模板继承
- 其他细节调整

### 2020-03-11

- 新增 模型
- 新增 助手函数 - redirect
- 新增 数据库 - 指定数据表
- 新增 数据库 - 查询
- 新增 数据库 - 链式操作 - join
- 调整 数据库 目录结构

### 2020-03-10

- 修改 序言 - ThinkPHP 官方资源
- 删除 安装 - 安装常用扩展
- 新增 视图 - 模板输出使用函数
- 新增 数据库 - 链式操作 - field
- 修改 经验分享 - apache 配置虚拟域名
- 改名 小东西分享 修改为 经验分享

### 2020-03-09

- 调整 序言 目录结构
- 调整 术语 目录结构
- 调整 镜像 目录结构
- 新增 小东西分享 - PhpStudy-v8.1 无法解析PHP代码

### 2020-03-08

- 新增 常用功能 - 登陆验证
- 删除 think-orm
- 新增 配置 - view.php
- 新增 视图 - 条件判断标签

### 2020-03-07

- 新增 助手函数 - url

- 新增 安装 - PhpStudy2018 运行 TP6.0

2020-03-05

- 新增 配置 - database.php
- 新增 调试 - 调试模式
- 新增 layui - layui文件上传
- 新增 layui - layui数据表格接口
- 修改 序言

# 2020-02

2019-02-05

- 新增 文件上传 - layui文件上传

# 2020-01

- [2019-01-31](#)
- [2019-01-30](#)
- [2019-01-29](#)
- [2019-01-14](#)
- [2019-01-10](#)
- [2019-01-08](#)
- [2019-01-05](#)
- [2019-01-03](#)
- [2019-01-02](#)

## 2019-01-31

- 新增 配置 - `filesystem.php`
- 新增 文件上传 - \*

## 2019-01-30

- 新增 文件上传

## 2019-01-29

- 改动 中间件 - xxxx
- 删除 `think-template`
- 移动 `think-template` - 输出替换 移动到 视图 - 输出替换

## 2019-01-14

- 新增 小东西分享 - 正则匹配 `img`标签及`src`属性值
- 新增 小东西分享 - 屏蔽Sublime的提示

## 2019-01-10

- 重命名 小东西分享 为 杂项
- 新增 小东西分享 - Sublime 3.x 激活码

## 2019-01-08

- 新增 调试 - 变量调试

## 2019-01-05

- 新增 助手函数 - input
- 新增 安装 - composer.phar
- 新增 技巧分享 - windows dos 窗口命令 start
- 新增 技巧分享 - windows dos 窗口命令 type
- 调整 安装 目录结构

## 2019-01-03

- 新增 镜像 章节
- 调整 安装 章节

## 2019-01-02

- 新增 术语 章节
- 新增 think-orm 章节
- 调整 配置 章节目录结构

# 2019-12

- 2019-12-31

新增章节："第三方扩展库" - "liliuwei/thinkphp-jump"

在TP6中使用TP5的跳转提示，如：`$this->success();`

TP5中的跳转提示在TP6中已经取消，想要使用TP5跳转可下载本扩展

- 2019-12-30

修改章节："第三方扩展库" - "topthink/think-captcha"

完整章节内容 优化章节页面展示效果

- 2019-12-28

新增章节："路由" - "域名路由"

通过二级域名访问指定的应用，如：admin.tp.com访问的是admin应用

- 2019-12-27

新增章节："其他技巧分享" - "navicat 闲置时间过久会卡死"

解决方法：设定保持连接间隔为30秒 时间自己定 不要太久

新增章节："think-template" - "输出替换"

模板字符串输出替换的使用方法及可能遇到的缓存问题

- 2019-12-24

新增章节："第三方扩展库" - "symfony/var-dumper"

优雅的PHP调试扩展库

ThinkPHP6.0、Laravel都使用了这个扩展

修改：第三方扩展库下的目录名修改为安装包地址

- 2019-12-23

新增章节："layuiAdmin" - "layuiAdmin FAQ"

记录layuiadmin使用过程中遇到的一些问题及解决方案

新增章节："第三方扩展库" - "think-captcha"

ThinkPHP官方提供的验证码扩展库安装及使用示例