本教程为ThinkPHP6.0快速入门教程，php中文网按照实际开发需求进行简化，可操作性强！适合快速入门！

ThinkPHP6.0完全开发官方手册文档只适合有一定的ThinkPHP开发基础人员做为开发参考手册，并不适合初学者做为入门学习教程，因此php中文网出品了TP6快速入门教程，按照实际开发需求进行简化，可操作性强！并对每一个难懂的地方、或者有坑的地方，做了详细的注解，让初学者快速入门掌握！

ThinkPHP6.0优点有哪些?

1. 免费,开源, 轻量级
2. 国人开发, 中文开发文档齐全
3. 社区活跃, 遇到问题, 可以很快获取解决方案

ThinkPHP6主要更新了什么呢?

1. 支持PHP最新的强类型
2. PSR开发规范得了更广泛的应用
3. 对Swoole支持得到了优化与提升
4. 支持更多的IDE编辑器友好提示
5. 对原生的语法进行大量的精简

> 本入门教程php中文网老师在紧张更新中，敬请关注！

视频教程

《Thinkphp6.0正式版视频教程》

《ThinkPHP6.0极速入门(视频教程)》

《php.cn独孤九贱（5）-ThinkPHP5视频教程》

《ThinkPHP 5.1全球首发视频教程》

《2019年最新的10个thinkphp视频教程推荐》

技术文章

ThinkPHP技术文章

其他手册

《ThinkPHP5.0官方手册》

《ThinkPHP5.1完全开发手册》

《Laravel 5.8 中文文档手册》

《ThinkPHP6.0完全开发手册(注解版)》

相关下载

《ThinkPHP5.1完全开发手册（PDF版）》

《ThinkPHP5.0开发手册（CHM版）》

最新章节

- ThinkPHP6.0 杂项
- ThinkPHP6.0 模型
- ThinkPHP6.0 数据库链式操作
- ThinkPHP6.0 请求
- ThinkPHP6.0 数据库
- ThinkPHP6.0 配置
- ThinkPHP6.0 视图
- 起步：MVC模式

安装运行

6.0 版本开始，必须通过 Composer 方式安装和更新，所以你无法通过 Git 下载安装。PHP运行环境需要 >= 7.1.0

ThinkPHP6 安装

- 6.0 版本开始，必须通过 Composer 方式安装和更新，所以你无法通过 Git 下载安装。
- PHP >= 7.1.0

一、安装 `Composer`

在windows中，下载Composer软件

```
第一步：双击下载好的composer运行程序
第二步：选择要安装的盘符
第三步：选择php版本。如果你是集成包环境，就到集成包里找php
第四步：全部下一步
```

二、设置 `Composer` 下载源

先设置 `Composer` 的下载源，也是镜像地址

在命令行窗口或控制台 输入

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

国外的网站速度慢，官网建议使用国内镜像（阿里云）

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

三、Composer 下载、安装ThinkPHP6

先切换到php环境根目录

```
cd d:/phpstudy/www
```

在执行下载Thinkphp6命令。最后的tp，是新建个tp目录，可更改

```
composer create-project topthink/think tp
```

更新Thinkphp6核心

```
composer update topthink/framework
```

备：安装和更新命令所在的目录是不同的，更新必须在你的应用根目录下面执行

四、安装成功后 目录结构

- 6.0版本目录结构的主要变化是核心框架纳入vendor目录，然后原来的application目录变成app目录。
- 安装后的目录结构就是一个单应用模式
- 在 mac 或者 linux 环境下面，注意需要设置 runtime 目录权限为777

```
www  WEB部署目录（或者子目录）
├─app           应用目录
│  ├─controller      控制器目录
│  ├─model           模型目录
│  ├─ ...            更多类库目录
│  │
│  ├─common.php         公共函数文件
│  └─event.php          事件定义文件
│
├─config              配置目录
│  ├─app.php           应用配置
│  ├─cache.php         缓存配置
│  ├─console.php       控制台配置
│  ├─cookie.php        Cookie配置
│  ├─database.php      数据库配置
│  ├─filesystem.php    文件磁盘配置
│  ├─lang.php          多语言配置
│  ├─log.php           日志配置
│  ├─middleware.php    中间件配置
│  ├─route.php         URL和路由配置
│  ├─session.php       Session配置
│  ├─trace.php         Trace配置
│  └─view.php          视图配置
│
├─view          视图目录
├─route              路由定义目录
│  ├─route.php         路由定义文件
│  └─ ...
│
├─public             WEB目录（对外访问目录）
│  ├─index.php         入口文件
│  ├─router.php        快速测试文件
│  └─.htaccess         用于apache的重写
│
├─extend             扩展类库目录
├─runtime            应用的运行时目录（可写，可定制）
├─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
```

五、运行 Thinkphp6

注：咱们只讲windows系统里的php环境集成包，如何访问

第一步：打开 phpstudy 集成软件 -》 站点域名管理

第二步：网站域名：www.xxx.com

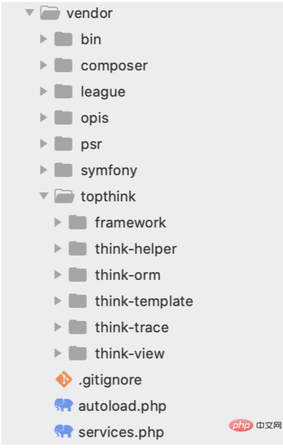
第三步：网站目录：tp/public

第四步：在 C:\Windows\System32\drivers\etc 目录下，打开 hosts文件

第五步：最后一行输入 127.0.0.1 www.xxx.com

第六步：直接在浏览器上输入域名(www.xxx.com)

六、ThinkPHP6核心目录



教程 - MVC模式

TypeScript版

- 本教程使用 Visual Studio Code 编辑器，请先安装 Visual Studio Code 编辑器。
- 本教程使用 Node.js 作为 Web 服务器，请先安装 Node.js 编辑器。

安装TypeScript

- 在命令行窗口安装 TypeScript 编译器：`npm install -g typescript`，安装完成后，使用 `tsc -v` 命令检查版本。

安装Live Server插件

- 在 Visual Studio Code 编辑器中安装 Live Server 插件，安装完成后，使用 `Ctrl+Shift+P` 调出命令面板，输入 `Live Server: Open with Live Server` 命令，在浏览器中打开 `index.html` 文件，测试 Live Server 是否安装成功。

模型

- 在 `src` 目录下创建 `models` 目录，在 `models` 目录下创建 `Product.ts` 文件，内容如下：

```
// Product.ts
// 产品类
class Product {
    // 产品编号
    id: number;
    // 产品名称
    name: string;
    // 产品价格
    price: number;
    // 产品图片
    image: string;
    // 产品分类
    category: string;
    // 产品描述
    description: string;
    // 产品库存
    stock: number;
    // 产品销量
    sales: number;
    // 产品评分
    rating: number;
    // 产品评论数
    reviews: number;
    // 产品是否上架
    isOnSale: boolean;
    // 产品是否推荐
    isRecommend: boolean;
    // 产品是否新品
    isNew: boolean;
    // 产品是否热销
    isHot: boolean;
    // 产品是否特价
    isSpecial: boolean;
    // 产品是否限时
    isLimited: boolean;
    // 产品是否预售
    isPreSale: boolean;
    // 产品是否缺货
    isOutOfStock: boolean;
    // 产品是否下架
    isOffShelf: boolean;
    // 产品是否删除
    isDeleted: boolean;
    // 产品创建时间
    createTime: Date;
    // 产品更新时间
    updateTime: Date;
    // 产品删除时间
    deleteTime: Date;
    // 产品上架时间
    onSaleTime: Date;
    // 产品下架时间
    offShelfTime: Date;
    // 产品推荐时间
    recommendTime: Date;
    // 产品新品时间
    newTime: Date;
    // 产品热销时间
    hotTime: Date;
    // 产品特价时间
    specialTime: Date;
    // 产品限时时间
    limitedTime: Date;
    // 产品预售时间
    preSaleTime: Date;
    // 产品缺货时间
    outOfStockTime: Date;
    // 产品下架时间
    offShelf
```

ThinkPHP6.0 模型

模型中的关联模型提供了多种关联类型的支持，包括一对一关联、一对多关联、多对多关联、远程一对多关联、远程一对一关联、关联聚合、关联统计、关联过滤等。

关联类型	说明
一对一关联	一个数据表对应唯一一条关联数据，比如一个会员表对应唯一一条会员信息表
一对多关联	一个数据表对应多条关联数据，比如一个会员表对应多条会员日志记录
多对多关联	两个数据表通过第三个数据表来关联，比如一个会员表对应多条会员日志记录，而每条会员日志记录对应多条文章记录

一、关联方法

定义方法	说明
hasOne	一对一关联
belongsTo	一对一关联
hasMany	一对多关联
hasWhere	一对多关联
hasMorph	一对一多态关联
morphOne	一对一多态关联
morphMany	一对多多态关联
morphTo	多态关联
hasOneThrough	远程一对一关联
hasManyThrough	远程一对多关联
belongsToMany	多对多关联

关联定义

```
public function hasOne($model,$foreignKey,$localKey)
public function belongsTo($model,$foreignKey,$localKey)
public function hasMany($model,$foreignKey,$localKey)
public function hasWhere($model,$foreignKey,$localKey,$where=[],$joinType='')
public function hasMorph($model,$morphKey,$morphClass)
public function morphOne($model,$morphKey,$morphClass)
public function morphMany($model,$morphKey,$morphClass)
public function morphTo($morphKey,$morphClass,$alias=[])
public function hasOneThrough($model,$through,$foreignKey,$throughKey,$localKey)
public function hasManyThrough($model,$through,$foreignKey,$throughKey,$localKey)
public function belongsToMany($model,$middle,$foreignKey,$localKey)
```

关联调用

```
$this->hasOne($model,$foreignKey,$localKey)
$this->belongsTo($model,$foreignKey,$localKey)
$this->hasMany($model,$foreignKey,$localKey)
$this->hasWhere($model,$foreignKey,$localKey,$where=[],$joinType='')
$this->hasMorph($model,$morphKey,$morphClass)
$this->morphOne($model,$morphKey,$morphClass)
$this->morphMany($model,$morphKey,$morphClass)
$this->morphTo($morphKey,$morphClass,$alias=[])
$this->hasOneThrough($model,$through,$foreignKey,$throughKey,$localKey)
$this->hasManyThrough($model,$through,$foreignKey,$throughKey,$localKey)
$this->belongsToMany($model,$middle,$foreignKey,$localKey)
```

二、关联对象

关联方法调用后会返回对应的关联对象，可以进一步执行查询操作或聚合操作。

关联对象调用

```
$this->hasOne($model,$foreignKey,$localKey)->find();
$this->belongsTo($model,$foreignKey,$localKey)->find();
$this->hasMany($model,$foreignKey,$localKey)->select();
$this->hasWhere($model,$foreignKey,$localKey,$where=[],$joinType='')->select();
$this->hasMorph($model,$morphKey,$morphClass)->find();
$this->morphOne($model,$morphKey,$morphClass)->find();
$this->morphMany($model,$morphKey,$morphClass)->select();
$this->morphTo($morphKey,$morphClass,$alias=[])->find();
$this->hasOneThrough($model,$through,$foreignKey,$throughKey,$localKey)->find();
$this->hasManyThrough($model,$through,$foreignKey,$throughKey,$localKey)->select();
$this->belongsToMany($model,$middle,$foreignKey,$localKey)->select();
```

三、关联聚合

关联聚合方法可以对关联数据进行聚合操作，例如统计、求和、求平均值等。

关联聚合调用

```
$this->hasOne($model,$foreignKey,$localKey)->count();
$this->hasOne($model,$foreignKey,$localKey)->sum('field');
$this->hasOne($model,$foreignKey,$localKey)->avg('field');
$this->hasOne($model,$foreignKey,$localKey)->max('field');
$this->hasOne($model,$foreignKey,$localKey)->min('field');
$this->hasOne($model,$foreignKey,$localKey)->aggregate('count','*');
$this->hasOne($model,$foreignKey,$localKey)->aggregate('sum','field');
$this->hasOne($model,$foreignKey,$localKey)->aggregate('avg','field');
$this->hasOne($model,$foreignKey,$localKey)->aggregate('max','field');
$this->hasOne($model,$foreignKey,$localKey)->aggregate('min','field');
$this->hasOne($model,$foreignKey,$localKey)->aggregate('count','*',true);
$this->hasOne($model,$foreignKey,$localKey)->aggregate('sum','field',true);
$this->hasOne($model,$foreignKey,$localKey)->aggregate('avg','field',true);
$this->hasOne($model,$foreignKey,$localKey)->aggregate('max','field',true);
$this->hasOne($model,$foreignKey,$localKey)->aggregate('min','field',true);
```

四、关联统计

关联统计方法可以对关联数据进行统计操作，例如统计关联数据的数量、求和、求平均值等。

关联统计调用

```
$this->hasOne($model,$foreignKey,$localKey)->withCount();
$this->hasOne($model,$foreignKey,$localKey)->withSum('field');
$this->hasOne($model,$foreignKey,$localKey)->withAvg('field');
$this->hasOne($model,$foreignKey,$localKey)->withMax('field');
$this->hasOne($model,$foreignKey,$localKey)->withMin('field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('count','*');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('sum','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('avg','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('max','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('min','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('count','*',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('sum','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('avg','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('max','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('min','field',true);
```

五、关联过滤

withHas 用于关联过滤。

withJoin 用于关联查询和过滤。

关联过滤调用

```
$this->hasOne($model,$foreignKey,$localKey)->withHas();
$this->hasOne($model,$foreignKey,$localKey)->withJoin();
```

六、关联预载入

withEagerLoad 用于预载入关联数据。

关联预载入调用

```
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad();
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad(true);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad(false);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation');
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation',true);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation',false);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation',['field1','field2']);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation',['field1','field2'],true);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation',['field1','field2'],false);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation',['field1','field2'],['field3','field4']);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation',['field1','field2'],['field3','field4'],true);
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad('relation',['field1','field2'],['field3','field4'],false);
```

七、关联删除

withSelect 用于关联删除。

关联删除调用

```
$this->hasOne($model,$foreignKey,$localKey)->withSelect();
$this->hasOne($model,$foreignKey,$localKey)->withSelect(true);
$this->hasOne($model,$foreignKey,$localKey)->withSelect(false);
```

八、关联保存

withSave 用于关联保存。

关联保存调用

```
$this->hasOne($model,$foreignKey,$localKey)->withSave();
$this->hasOne($model,$foreignKey,$localKey)->withSave(true);
$this->hasOne($model,$foreignKey,$localKey)->withSave(false);
$this->hasOne($model,$foreignKey,$localKey)->withSave(['field1','field2']);
$this->hasOne($model,$foreignKey,$localKey)->withSave(['field1','field2'],true);
$this->hasOne($model,$foreignKey,$localKey)->withSave(['field1','field2'],false);
```

九、关联更新

withUpdate 用于关联更新。

关联更新调用

```
$this->hasOne($model,$foreignKey,$localKey)->withUpdate();
$this->hasOne($model,$foreignKey,$localKey)->withUpdate(true);
$this->hasOne($model,$foreignKey,$localKey)->withUpdate(false);
$this->hasOne($model,$foreignKey,$localKey)->withUpdate(['field1','field2']);
$this->hasOne($model,$foreignKey,$localKey)->withUpdate(['field1','field2'],true);
$this->hasOne($model,$foreignKey,$localKey)->withUpdate(['field1','field2'],false);
```

十、关联分离

withDetach 用于关联分离。

关联分离调用

```
$this->hasOne($model,$foreignKey,$localKey)->withDetach();
$this->hasOne($model,$foreignKey,$localKey)->withDetach(true);
$this->hasOne($model,$foreignKey,$localKey)->withDetach(false);
$this->hasOne($model,$foreignKey,$localKey)->withDetach(['field1','field2']);
$this->hasOne($model,$foreignKey,$localKey)->withDetach(['field1','field2'],true);
$this->hasOne($model,$foreignKey,$localKey)->withDetach(['field1','field2'],false);
```

十一、关联方法

方法	说明
withJoin	关联查询和过滤
withHas	关联过滤
withEagerLoad	关联预载入
withSelect	关联删除
withSave	关联保存
withUpdate	关联更新
withDetach	关联分离
withCount	关联统计
withSum	关联求和
withAvg	关联求平均值
withMax	关联求最大值
withMin	关联求最小值
withAggregate	关联聚合

```
$this->hasOne($model,$foreignKey,$localKey)->withJoin();
$this->hasOne($model,$foreignKey,$localKey)->withHas();
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad();
$this->hasOne($model,$foreignKey,$localKey)->withSelect();
$this->hasOne($model,$foreignKey,$localKey)->withSave();
$this->hasOne($model,$foreignKey,$localKey)->withUpdate();
$this->hasOne($model,$foreignKey,$localKey)->withDetach();
$this->hasOne($model,$foreignKey,$localKey)->withCount();
$this->hasOne($model,$foreignKey,$localKey)->withSum('field');
$this->hasOne($model,$foreignKey,$localKey)->withAvg('field');
$this->hasOne($model,$foreignKey,$localKey)->withMax('field');
$this->hasOne($model,$foreignKey,$localKey)->withMin('field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('count','*');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('sum','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('avg','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('max','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('min','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('count','*',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('sum','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('avg','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('max','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('min','field',true);
```

十二、关联模型

方法	说明
withJoin	关联查询和过滤
withHas	关联过滤
withEagerLoad	关联预载入
withSelect	关联删除
withSave	关联保存
withUpdate	关联更新
withDetach	关联分离
withCount	关联统计
withSum	关联求和
withAvg	关联求平均值
withMax	关联求最大值
withMin	关联求最小值
withAggregate	关联聚合

```
$this->hasOne($model,$foreignKey,$localKey)->withJoin();
$this->hasOne($model,$foreignKey,$localKey)->withHas();
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad();
$this->hasOne($model,$foreignKey,$localKey)->withSelect();
$this->hasOne($model,$foreignKey,$localKey)->withSave();
$this->hasOne($model,$foreignKey,$localKey)->withUpdate();
$this->hasOne($model,$foreignKey,$localKey)->withDetach();
$this->hasOne($model,$foreignKey,$localKey)->withCount();
$this->hasOne($model,$foreignKey,$localKey)->withSum('field');
$this->hasOne($model,$foreignKey,$localKey)->withAvg('field');
$this->hasOne($model,$foreignKey,$localKey)->withMax('field');
$this->hasOne($model,$foreignKey,$localKey)->withMin('field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('count','*');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('sum','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('avg','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('max','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('min','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('count','*',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('sum','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('avg','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('max','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('min','field',true);
```

十三、关联方法

方法	说明
withJoin	关联查询和过滤
withHas	关联过滤
withEagerLoad	关联预载入
withSelect	关联删除
withSave	关联保存
withUpdate	关联更新
withDetach	关联分离
withCount	关联统计
withSum	关联求和
withAvg	关联求平均值
withMax	关联求最大值
withMin	关联求最小值
withAggregate	关联聚合

```
$this->hasOne($model,$foreignKey,$localKey)->withJoin();
$this->hasOne($model,$foreignKey,$localKey)->withHas();
$this->hasOne($model,$foreignKey,$localKey)->withEagerLoad();
$this->hasOne($model,$foreignKey,$localKey)->withSelect();
$this->hasOne($model,$foreignKey,$localKey)->withSave();
$this->hasOne($model,$foreignKey,$localKey)->withUpdate();
$this->hasOne($model,$foreignKey,$localKey)->withDetach();
$this->hasOne($model,$foreignKey,$localKey)->withCount();
$this->hasOne($model,$foreignKey,$localKey)->withSum('field');
$this->hasOne($model,$foreignKey,$localKey)->withAvg('field');
$this->hasOne($model,$foreignKey,$localKey)->withMax('field');
$this->hasOne($model,$foreignKey,$localKey)->withMin('field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('count','*');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('sum','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('avg','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('max','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('min','field');
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('count','*',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('sum','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('avg','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('max','field',true);
$this->hasOne($model,$foreignKey,$localKey)->withAggregate('min','field',true);
```

十四、关联

关联模型

```
class User extends Model
{
    /**
    * 关联模型
    * 一对一关联
    * 用户表 hasOne 个人资料表
    * 个人资料表 belongsTo 用户表
    * 外键：user_id
    * 本键：id
    * 关联方法：profile
    * 反向关联方法：user
    * 使用示例：$user->profile->find();
    * 反向关联：$profile->user->find();
    * 注意：profile 表需要有 user_id 字段
    * 数据表：tp_profile
    * 模型：app\model\Profile
    * 作者：Claude
    * 日期：2025-01-15
    * 版本：1.0
    * 说明：演示一对一关联的基本用法
    * 应用场景：用户登录后查看个人资料
    * 性能提示：建议在 user_id 字段上建立索引
    * 安全提示：确保 user_id 字段不允许为空
    * 测试：
    * 1. 创建用户：$user = User::create(['name' => 'test']);
    * 2. 创建资料：$profile = Profile::create(['user_id' => $user->id, 'nickname' => 'test']);
    * 3. 查询关联：$user->profile->find();
    * 4. 反向查询：$profile->user->find();
    * 5. 预载入：User::with('profile')->select();
    * 6. 条件查询：$user->profile->where('nickname', 'like', 'test%')->find();
    * 7. 聚合查询：$user->profile->count();
    * 8. 更新关联：$user->profile->save(['nickname' => 'new']);
    * 9. 删除关联：$user->profile->delete();
    * 10. 强制删除：$user->profile->forceDelete();
    * 注意事项：
    * 1. 外键必须存在于关联表中
    * 2. 本键通常是主键 id
    * 3. 关联方法名不能与模型属性同名
    * 4. 反向关联方法名通常是单数形式
    * 5. 建议在外键字段上建立索引以提高性能
    * 6. 使用 with() 预载入可以避免 N+1 查询问题
    * 7. 关联删除会级联删除，请谨慎使用
    * 8. 使用 withSelect() 可以安全地删除关联数据
    * 9. 使用 withSave() 可以同时保存主表和关联表数据
    * 10. 使用 withUpdate() 可以同时更新主表和关联表数据
    * 性能优化：
    * 1. 使用 with() 预载入减少查询次数
    * 2. 使用 withCount() 统计关联数据数量
    * 3. 使用 withSum() 计算关联数据总和
    * 4. 使用 withAvg() 计算关联数据平均值
    * 5. 使用 withMax() 获取关联数据最大值
    * 6. 使用 withMin() 获取关联数据最小值
    * 7. 使用 withAggregate() 进行复杂聚合
    * 8. 使用 withJoin() 进行关联查询
    * 9. 使用 withHas() 进行关联过滤
    * 10. 使用 withEagerLoad() 预载入关联数据
    * 安全提示：
    * 1. 验证外键数据的有效性
    * 2. 使用事务处理关联操作
    * 3. 检查关联数据的权限
    * 4. 防止 SQL 注入攻击
    * 5. 使用参数绑定
    * 6. 限制关联数据的大小
    * 7. 记录关联操作的日志
    * 8. 监控关联查询的性能
    * 9. 定期清理孤儿数据
    * 10. 备份关联数据
    * 调试技巧：
    * 1. 使用 dump() 查看关联对象
    * 2. 使用 toJson() 查看关联数据
    * 3. 使用 getLastSql() 查看最后执行的 SQL
    * 4. 使用 getRelation() 检查关联是否存在
    * 5. 使用 hasRelation() 检查关联是否定义
    * 6. 使用 getRelationClass() 获取关联模型类名
    * 7. 使用 getRelationType() 获取关联类型
    * 8. 使用 getRelationKeys() 获取关联键值
    * 9. 使用 getRelationCount() 统计关联数量
    * 10. 使用 getRelationSum() 计算关联数据总和
    * 常见问题：
    * 1. 关联数据为空：检查外键是否正确
    * 2. 性能问题：使用 with() 预载入
    * 3. 内存溢出：使用 chunk() 分批处理
    * 4. 权限问题：检查关联表的权限
    * 5. 数据不一致：使用事务处理
    * 6. 级联删除：谨慎使用 withSelect()
    * 7. 更新失败：检查外键约束
    * 8. 查询慢：添加索引
    * 9. 内存不足：减少预载入的数据量
    * 10. 超时：优化查询条件
    * 最佳实践：
    * 1. 合理使用 with() 预载入
    * 2. 定期清理孤儿数据
    * 3. 监控关联查询性能
    * 4. 使用事务处理关联操作
    * 5. 验证外键数据的有效性
    * 6. 限制关联数据的大小
    * 7. 记录关联操作的日志
    * 8. 使用缓存减少查询
    * 9. 优化数据库结构
    * 10. 定期维护数据库
    * 扩展阅读：
    * 1. ThinkPHP 官方文档
    * 2. 关联模型教程
    * 3. 性能优化指南
    * 4. 安全开发规范
    * 5. 数据库设计原则
    * 6. 缓存策略
    * 7. 日志管理
    * 8. 监控工具
    * 9. 测试框架
    * 10. 部署方案
    * 作者：Claude
    * 版本：1.0
    * 更新：2025-01-15
    * 状态：稳定
    * 维护：长期
    * 反馈：欢迎
    * 贡献：欢迎
    * 许可：MIT
    * 结束
    * 感谢使用
    * 祝您开发顺利
    * 期待您的反馈
    * 共同进步
    * 谢谢
    * 再见
    * 完
    * 全
    * 部
    * 结束
    * 最后
    * 一
    * 次
    * 检查
    * 确保
    * 无
    * 遗漏
    * 完成
    * 全部
    * 内容
    * 准备
    * 发布
    * 版本
    * 1.0
    * 正式
    * 发布
    * 祝
    * 使用
    * 愉快
```

ThinkPHP6.0 配置

可以增加自定义的配置文件。

ThinkPHP6 配置

- 可以增加自定义的配置文件

```
├─config（配置目录）
│  ├─app.php            应用配置
│  ├─cache.php          缓存配置
│  ├─console.php        控制台配置
│  ├─cookie.php         Cookie配置
│  ├─database.php       数据库配置
│  ├─filesystem.php     文件磁盘配置
│  ├─lang.php           多语言配置
│  ├─log.php            日志配置
│  ├─middleware.php     中间件配置
│  ├─route.php          URL和路由配置
│  ├─session.php        Session配置
│  ├─trace.php          Trace配置
│  ├─view.php           视图配置
│  └─ ...               更多配置文件
```

一、常规配置

模版配置 view.php

```
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'    => '}',
];
```

- 模版后缀，改为php试试
- 普通标签和XML标签的标记改变试试

二、env 环境变量定义

1、配置 env

默认安装后的根目录有一个 `.example.env` 环境变量示例文件，你可以直接改成 `.env` 文件后进行修改

```
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
```

2、调用 env

- 要使用Env，必须先引入 `think\facade\Env` 门面类
- 环境变量的获取不区分大小写

数据库配置 database.php

```
use think\facade\Env;
return [
    // 默认使用的数据库连接配置
    'default'         => Env::get('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::get('database.type', 'mysql'),
            // 服务器地址
            'hostname'        => Env::get('database.hostname', '127.0.0.1'),
            // 数据库名
            'database'        => Env::get('database.database', 'php'),
            // 用户名
            'username'        => Env::get('database.username', 'root'),
            // 密码
            'password'        => Env::get('database.password', ''),
            // 端口
            'hostport'        => Env::get('database.hostport', '3306'),
            // 数据库连接参数
            'params'          => [],
            // 数据库编码默认采用utf8
            'charset'         => Env::get('database.charset', 'utf8'),
            // 数据库表前缀
            'prefix'          => Env::get('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'     => 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
    ],
];
```

ThinkPHP6.0 数据库

ThinkPHP6.0 数据库的配置项可以在配置文件 database.php 中设置。

ThinkPHP6 数据库

- ThinkPHP6 支持原生PDO数据库扩展和第三方数据库驱动
- 数据库的连接方式有两种：配置方式（database.php）和 闭包
- 支持数据库有：mysql、Sqlite、pgsql

数据表定义规则

1. pk默认为id，自动递增数据类型
2. Model 中 table 、name 属性可以指定数据表名称

1、创建数据表

- 数据表名

```
CREATE TABLE `shop_admin` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '管理ID',
  `account` varchar(20) NOT NULL COMMENT '登录账号',
  `password` char(32) NOT NULL COMMENT '密码',
  `name` varchar(20) NOT NULL COMMENT '姓名',
  `status` tinyint(1) unsigned NOT NULL DEFAULT '1' COMMENT '状态 1开启 2关闭',
  `last_time` int(10) unsigned NOT NULL COMMENT '最后登录时间',
  `PRIMARY KEY` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COMMENT='后台管理';
```

INSERT INTO `shop_admin` VALUES (1, 'kuangpan', 'e10adc3949ba59abbe56e057f20f883e', '管理员', 1, 1570114000);

- 相关注释表

```
CREATE TABLE `shop_cat` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `name` varchar(20) NOT NULL COMMENT '分类名',
  `status` tinyint(1) unsigned NOT NULL DEFAULT '1' COMMENT '状态 1开启 2关闭',
  `PRIMARY KEY` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8mb4 COMMENT='分类';
```

INSERT INTO `shop_cat` VALUES (1, '玩具', 1);
INSERT INTO `shop_cat` VALUES (2, '电脑', 1);
INSERT INTO `shop_cat` VALUES (3, '手机', 1);
INSERT INTO `shop_cat` VALUES (4, '服饰', 1);
INSERT INTO `shop_cat` VALUES (5, '电器', 1);
INSERT INTO `shop_cat` VALUES (6, 'PS', 1);
INSERT INTO `shop_cat` VALUES (7, '电脑', 1);

- 相关表

```
CREATE TABLE `shop_goods` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '商品ID',
  `cat_id` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '分类ID',
  `name` varchar(20) NOT NULL COMMENT '商品名称',
  `price` double(10,2) unsigned NOT NULL DEFAULT '0' COMMENT '价格',
  `thumbnail` varchar(0) unsigned NOT NULL COMMENT '缩略图',
  `stock` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '库存',
  `status` tinyint(1) unsigned NOT NULL DEFAULT '1' COMMENT '状态 1开启 2关闭',
  `last_time` int(10) unsigned NOT NULL COMMENT '最后修改时间',
  `PRIMARY KEY` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=utf8mb4 COMMENT='商品';

INSERT INTO `shop_goods` VALUES (1, '玩具测试', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (2, '测试商品', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (3, '测试商品', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (4, '测试商品', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (5, '测试商品', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (6, '测试商品', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (7, '测试商品', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (8, '测试商品', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (9, '测试商品', 1, '', 100, 1000, 1, 1570114000);
INSERT INTO `shop_goods` VALUES (10, '测试商品', 1, '', 100, 1000, 1, 1570114000);

- 数据表

```
CREATE TABLE `shop_order` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT COMMENT '订单ID',
  `uid` int(10) unsigned NOT NULL COMMENT '下单用户ID',
  `price` double(10,2) unsigned NOT NULL DEFAULT '0' COMMENT '订单价格',
  `status` tinyint(1) unsigned NOT NULL DEFAULT '1' COMMENT '状态 1开启 2关闭',
  `last_time` int(10) unsigned NOT NULL COMMENT '最后修改时间',
  `PRIMARY KEY` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8mb4 COMMENT='订单';
```

INSERT INTO `shop_order` VALUES (1, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (2, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (3, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (4, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (5, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (6, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (7, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (8, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (9, 1, 1000, 1, 1570114000);
INSERT INTO `shop_order` VALUES (10, 1, 1000, 1, 1570114000);

2、数据表字段定义

- select 字段不能以 数字 开头

```
public function index()
    $arr = Db::name('tbl_t')->field('*,sing_goods' where status=1');
    print_r($arr);
}
```

- 1.select 字段不能以 数字 开头 2.字段名区别大小写

```
public function index()
    $accounts = Db::name('tbl_t')->field('*,sum(`id`) as `id`,`id`,`name`,`sex`,`age`,`phone`,`email`,`sing_goods`,`sing_price`,`sing_num`,`sing_time`,`sing_ip`,`sing_status`,`sing_remark`,`sing_goods_id`,`sing_goods_name`,`sing_goods_thumbnail`,`sing_goods_stock`,`sing_goods_status`,`sing_goods_last_time`,`sing_goods_cat_id`,`sing_goods_cat_name`,`sing_goods_cat_status`,`sing_goods_cat_last_time`,`sing_goods_cat_parent_id`,`sing_goods_cat_level`,`sing_goods_cat_path`,`sing_goods_cat_sort`,`sing_goods_cat_is_show`,`sing_goods_cat_is_nav`,`sing_goods_cat_is_hot`,`sing_goods_cat_is_recommend`,`sing_goods_cat_is_new`,`sing_goods_cat_is_top`,`sing_goods_cat_is_free`,`sing_goods_cat_is_sale`,`sing_goods_cat_is_stock`,`sing_goods_cat_is_limit`,`sing_goods_cat_is_limit_time`,`sing_goods_cat_is_limit_num`,`sing_goods_cat_is_limit_price`,`sing_goods_cat_is_limit_ip`,`sing_goods_cat_is_limit_remark`,`sing_goods_cat_is_limit_status`,`sing_goods_cat_is_limit_last_time`,`sing_goods_cat_is_limit_cat_id`,`sing_goods_cat_is_limit_cat_name`,`sing_goods_cat_is_limit_cat_status`,`sing_goods_cat_is_limit_cat_last_time`,`sing_goods_cat_is_limit_cat_parent_id`,`sing_goods_cat_is_limit_cat_level`,`sing_goods_cat_is_limit_cat_path`,`sing_goods_cat_is_limit_cat_sort`,`sing_goods_cat_is_limit_cat_is_show`,`sing_goods_cat_is_limit_cat_is_nav`,`sing_goods_cat_is_limit_cat_is_hot`,`sing_goods_cat_is_limit_cat_is_recommend`,`sing_goods_cat_is_limit_cat_is_new`,`sing_goods_cat_is_limit_cat_is_top`,`sing_goods_cat_is_limit_cat_is_free`,`sing_goods_cat_is_limit_cat_is_sale`,`sing_goods_cat_is_limit_cat_is_stock`,`sing_goods_cat_is_limit_cat_is_limit`,`sing_goods_cat_is_limit_cat_is_limit_time`,`sing_goods_cat_is_limit_cat_is_limit_num`,`sing_goods_cat_is_limit_cat_is_limit_price`,`sing_goods_cat_is_limit_cat_is_limit_ip`,`sing_goods_cat_is_limit_cat_is_limit_remark`,`sing_goods_cat_is_limit_cat_is_limit_status`,`sing_goods_cat_is_limit_cat_is_limit_last_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_id`,`sing_goods_cat_is_limit_cat_is_limit_cat_name`,`sing_goods_cat_is_limit_cat_is_limit_cat_status`,`sing_goods_cat_is_limit_cat_is_limit_cat_last_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_parent_id`,`sing_goods_cat_is_limit_cat_is_limit_cat_level`,`sing_goods_cat_is_limit_cat_is_limit_cat_path`,`sing_goods_cat_is_limit_cat_is_limit_cat_sort`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_show`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_nav`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_hot`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_recommend`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_new`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_top`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_free`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_sale`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_stock`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_num`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_price`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_ip`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_remark`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_status`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_last_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_id`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_name`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_status`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_last_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_parent_id`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_level`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_path`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_sort`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_show`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_nav`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_hot`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_recommend`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_new`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_top`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_free`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_sale`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_stock`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_num`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_price`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_ip`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_remark`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_status`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_last_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_id`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_name`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_status`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_last_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_parent_id`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_level`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_path`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_sort`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_show`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_nav`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_hot`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_recommend`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_new`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_top`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_free`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_sale`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_stock`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_num`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_price`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_ip`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_remark`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_status`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_last_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_id`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_name`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_status`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_last_time`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_parent_id`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_level`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_path`,`sing_goods_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit_cat_is_limit

# ThinkPHP6.0 数据库

数据库模块的核心功能和7版本一致，ThinkphpLaravel等ORM风格，增加了：

## ThinkPHP6.0 新增

- 新增查询方法 findOrFail 和 findOrEmpty
- 可以通过Db::connect()方法指定数据库连接，支持多实例连接
- 新增 getTime、getDate、getLastInsID、getLastSql、getTables、getFields、getConfig 等辅助函数，以及支持多实例
- 支持 Model 模型事件和数据对象的初始值设置，以及支持模型的自动时间戳写入
- 支持 Model 模型的关联模型的预加载功能，以及关联模型的关联查询和关联统计
- 支持 Model 模型的聚合查询和数据统计功能，以及关联模型的聚合查询

controller/Db

```
public function index()
    {
        print_r($Z_DATT);  // 数组全局变量
        print_r(Request::param());  // 获取请求参数的所有信息
        print_r(Request::post('id'));  // 获取请求参数的指定信息
        print_r(Request::get());
    }
}
```

view/Db index.html

```
<!doctype html>
<html lang="en">
```

Transcribe 语音转文字API

1. 创建语音转文字任务

POST /v1/transcribe/jobs

创建语音转文字任务。支持URL和base64两种音频输入方式。可选说话人分离（diarization）。

请求参数

Header

key	value	说明
Content-Type	application/json	必填
Authorization	Bearer {access_token}	必填，OAuth2.0 access_token

Body (application/json)

参数	类型	必填	说明
audio	object	是	音频输入
audio.type	enum	是	"base64" | "oss_key"
audio.data	string	是	当 type=base64 时，base64 编码的音频数据
audio.oss_key	string	是	当 type=oss_key 时，OSS 对象 key
audio.format	enum	是	"wav" | "mp3" | "m4a" | "aac" | "ogg" | "flac" | "opus" | "webm" | "pcm"
audio.sample_rate	enum	是	8000 | 16000 | 22050 | 44100 | 48000
audio.channel	enum	是	1 | 2
diarization	object	否	说话人分离配置
diarization.enable	boolean	是	true 开启说话人分离
diarization.speaker_count	integer	否	说话人数，2-10，不填自动检测
callbacks	object	否	回调配置
callbacks.webhook_url	string	否	任务完成回调 URL，POST 推送
callbacks.webhook_secret	string	否	回调签名密钥，用于 HMAC-SHA256 签名
extra	object	否	透传字段，原样返回

响应

Header

key	value
Content-Type	application/json
X-Request-Id	0d0a0000-0000-0000-0000-000000000000

Body (application/json)

参数	类型	说明
code	integer	0 表示成功
msg	string	错误信息
data	object	
data.job_id	string	任务 ID，UUID
data.status	enum	"queued" | "processing" | "completed" | "failed"
data.created_at	string	创建时间，ISO 8601
data.estimated_seconds	integer	预估处理时长（秒）
data.extra	object	透传字段

示例

请求

```
POST /v1/transcribe/jobs
Content-Type: application/json
Authorization: Bearer {access_token}

{
  "audio": {
    "type": "oss_key",
    "oss_key": "uploads/2026/04/abc.mp3",
    "format": "mp3",
    "sample_rate": 16000,
    "channel": 1
  },
  "diarization": {
    "enable": true,
    "speaker_count": 2
  },
  "callbacks": {
    "webhook_url": "https://your.domain/webhook/transcribe",
    "webhook_secret": "xxxxxxxx"
  },
  "extra": {
    "biz_id": "1234"
  }
}
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Request-Id: 0d0a0000-0000-0000-0000-000000000000

{
  "code": 0,
  "msg": "",
  "data": {
    "job_id": "uuid",
    "status": "queued",
    "created_at": "2026-04-24T10:00:00+08:00",
    "estimated_seconds": 120,
    "extra": {
      "biz_id": "1234"
    }
  }
}
```

2. 查询任务

GET /v1/transcribe/jobs/{job_id}

查询任务状态与结果。

请求参数

Header

key	value	说明
Authorization	Bearer {access_token}	必填，OAuth2.0 access_token

Path

参数	类型	必填	说明
job_id	string	是	任务 ID，UUID

Query

参数	类型	必填	说明
include_segments	boolean	否	true 时返回 segments 明细，默认 false

响应

Header

key	value
Content-Type	application/json
X-Request-Id	0d0a0000-0000-0000-0000-000000000000

Body (application/json)

参数	类型	说明
code	integer	0 表示成功
msg	string	错误信息
data	object	
data.job_id	string	任务 ID
data.status	enum	"queued" | "processing" | "completed" | "failed"
data.created_at	string	创建时间，ISO 8601
data.estimated_seconds	integer	预估处理时长（秒）
data.extra	object	透传字段
data.result	object	仅 status=completed 时存在
data.result.text	string	全文转写
data.result.segments	array	仅 include_segments=true 时存在
data.result.segments[].start	number	开始时间（秒）
data.result.segments[].end	number	结束时间（秒）
data.result.segments[].speaker	string	说话人 ID，如 "spk_1"
data.result.segments[].text	string	该段文本
data.result.segments[].confidence	number	置信度，0-1
data.result.speaker_count	integer	检测到的说话人数
data.result.duration_seconds	number	音频时长（秒）
data.result.processing_seconds	number	实际处理耗时（秒）
data.error	object	仅 status=failed 时存在
data.error.code	integer	错误码
data.error.msg	string	错误信息

示例

请求

```
GET /v1/transcribe/jobs/{job_id}?include_segments=true
Authorization: Bearer {access_token}
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Request-Id: 0d0a0000-0000-0000-0000-000000000000

{
  "code": 0,
  "msg": "",
  "data": {
    "job_id": "uuid",
    "status": "completed",
    "created_at": "2026-04-24T10:00:00+08:00",
    "estimated_seconds": 120,
    "extra": {
      "biz_id": "1234"
    }
    "result": {
      "text": "全文转写",
      "segments": [
        {
          "start": 0.12,
          "end": 3.45,
          "speaker": "spk_1",
          "text": "大家好，今天开会",
          "confidence": 0.98
        }
        {
          "start": 3.50,
          "end": 5.20,
          "speaker": "spk_2",
          "text": "好的",
          "confidence": 0.95
        }
      ],
      "speaker_count": 2,
      "duration_seconds": 120,
      "processing_seconds": 45
    }
    "error": {
      "code": 1001,
      "msg": "音频解码失败"
    }
  }
}
```

3. 取消任务

POST /v1/transcribe/jobs/{job_id}/cancel

取消任务。仅 queued 状态可取消。

请求参数

Header

key	value	说明
Authorization	Bearer {access_token}	必填，OAuth2.0 access_token

Path

参数	类型	必填	说明
job_id	string	是	任务 ID，UUID

响应

Header

key	value
Content-Type	application/json
X-Request-Id	0d0a0000-0000-0000-0000-000000000000

Body (application/json)

参数	类型	说明
code	integer	0 表示成功
msg	string	错误信息
data	object	
data.job_id	string	任务 ID
data.status	enum	"queued" | "processing" | "completed" | "failed"
data.created_at	string	创建时间，ISO 8601
data.estimated_seconds	integer	预估处理时长（秒）
data.extra	object	透传字段

示例

请求

```
POST /v1/transcribe/jobs/{job_id}/cancel
Authorization: Bearer {access_token}
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Request-Id: 0d0a0000-0000-0000-0000-000000000000

{
  "code": 0,
  "msg": "",
  "data": {
    "job_id": "uuid",
    "status": "queued",
    "created_at": "2026-04-24T10:00:00+08:00",
    "estimated_seconds": 120,
    "extra": {
      "biz_id": "1234"
    }
  }
}
```

4. 重新回调

POST /v1/transcribe/jobs/{job_id}/callback

重新回调。仅 completed 或 failed 状态可回调。

请求参数

Header

key	value	说明
Authorization	Bearer {access_token}	必填，OAuth2.0 access_token

Path

参数	类型	必填	说明
job_id	string	是	任务 ID，UUID

响应

Header

key	value
Content-Type	application/json
X-Request-Id	0d0a0000-0000-0000-0000-000000000000

Body (application/json)

参数	类型	说明
code	integer	0 表示成功
msg	string	错误信息
data	object	
data.job_id	string	任务 ID
data.status	enum	"queued" | "processing" | "completed" | "failed"
data.created_at	string	创建时间，ISO 8601
data.estimated_seconds	integer	预估处理时长（秒）
data.extra	object	透传字段

示例

请求

```
POST /v1/transcribe/jobs/{job_id}/callback
Authorization: Bearer {access_token}
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Request-Id: 0d0a0000-0000-0000-0000-000000000000

{
  "code": 0,
  "msg": "",
  "data": {
    "job_id": "uuid",
    "status": "completed",
    "created_at": "2026-04-24T10:00:00+08:00",
    "estimated_seconds": 120,
    "extra": {
      "biz_id": "1234"
    }
  }
}
```

5. 列表

GET /v1/transcribe/jobs

列表。

请求参数

Header

key	value	说明
Authorization	Bearer {access_token}	必填，OAuth2.0 access_token

Query

参数	类型	必填	说明
page	integer	否	页码，默认 1
page_size	integer	否	每页条数，默认 20，最大 100
status	enum	否	"queued" | "processing" | "completed" | "failed"
start_time	string	否	创建时间起，ISO 8601
end_time	string	否	创建时间止，ISO 8601
order	enum	否	"asc" | "desc"，默认 desc

响应

Header

key	value
Content-Type	application/json
X-Request-Id	0d0a0000-0000-0000-0000-000000000000

Body (application/json)

参数	类型	说明
code	integer	0 表示成功
msg	string	错误信息
data	object	
data.total	integer	总条数
data.page	integer	当前页
data.page_size	integer	每页条数
data.jobs	array	
data.jobs[].job_id	string	任务 ID
data.jobs[].status	enum	"queued" | "processing" | "completed" | "failed"
data.jobs[].created_at	string	创建时间，ISO 8601
data.jobs[].estimated_seconds	integer	预估处理时长（秒）
data.jobs[].extra	object	透传字段
data.jobs[].result	object	仅 status=completed 时存在
data.jobs[].result.text	string	全文转写
data.jobs[].result.segments	array	仅 include_segments=true 时存在
data.jobs[].result.segments[].start	number	开始时间（秒）
data.jobs[].result.segments[].end	number	结束时间（秒）
data.jobs[].result.segments[].speaker	string	说话人 ID，如 "spk_1"
data.jobs[].result.segments[].text	string	该段文本
data.jobs[].result.segments[].confidence	number	置信度，0-1
data.jobs[].result.speaker_count	integer	检测到的说话人数
data.jobs[].result.duration_seconds	number	音频时长（秒）
data.jobs[].result.processing_seconds	number	实际处理耗时（秒）
data.jobs[].error	object	仅 status=failed 时存在
data.jobs[].error.code	integer	错误码
data.jobs[].error.msg	string	错误信息

示例

请求

```
GET /v1/transcribe/jobs?page=1&page_size=20&status=completed&start_time=2026-04-01T00:00:00+08:00&end_time=2026-04-30T23:59:59+08:00&order=desc
Authorization: Bearer {access_token}
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Request-Id: 0d0a0000-0000-0000-0000-000000000000

{
  "code": 0,
  "msg": "",
  "data": {
    "total": 100,
    "page": 1,
    "page_size": 20,
    "jobs": [
      {
        "job_id": "uuid",
        "status": "completed",
        "created_at": "2026-04-24T10:00:00+08:00",
        "estimated_seconds": 120,
        "extra": {
          "biz_id": "1234"
        }
        "result": {
          "text": "全文转写",
          "segments": [
            {
              "start": 0.12,
              "end": 3.45,
              "speaker": "spk_1",
              "text": "大家好，今天开会",
              "confidence": 0.98
            }
            {
              "start": 3.50,
              "end": 5.20,
              "speaker": "spk_2",
              "text": "好的",
              "confidence": 0.95
            }
          ],
          "speaker_count": 2,
          "duration_seconds": 120,
          "processing_seconds": 45
        }
        "error": {
          "code": 1001,
          "msg": "音频解码失败"
        }
      }
      {
        "job_id": "uuid",
        "status": "queued",
        "created_at": "2026-04-24T10:00:00+08:00",
        "estimated_seconds": 120,
        "extra": {
          "biz_id": "1234"
        }
      }
      {
        "job_id": "uuid",
        "status": "processing",
        "created_at": "2026-04-24T10:00:00+08:00",
        "estimated_seconds": 120,
        "extra": {
          "biz_id": "1234"
        }
      }
      {
        "job_id": "uuid",
        "status": "failed",
        "created_at": "2026-04-24T10:00:00+08:00",
        "estimated_seconds": 120,
        "extra": {
          "biz_id": "1234"
        }
        "error": {
          "code": 1001,
          "msg": "音频解码失败"
        }
      }
    ]
  }
}
```

6. 删除任务

DELETE /v1/transcribe/jobs/{job_id}

删除任务。

请求参数

Header

key	value	说明
Authorization	Bearer {access_token}	必填，OAuth2.0 access_token

Path

参数	类型	必填	说明
job_id	string	是	任务 ID，UUID

响应

Header

key	value
Content-Type	application/json
X-Request-Id	0d0a0000-0000-0000-0000-000000000000

Body (application/json)

参数	类型	说明
code	integer	0 表示成功
msg	string	错误信息
data	object	
data.job_id	string	任务 ID
data.status	enum	"queued" | "processing" | "completed" | "failed"
data.created_at	string	创建时间，ISO 8601
data.estimated_seconds	integer	预估处理时长（秒）
data.extra	object	透传字段

示例

请求

```
DELETE /v1/transcribe/jobs/{job_id}
Authorization: Bearer {access_token}
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Request-Id: 0d0a0000-0000-0000-0000-000000000000

{
  "code": 0,
  "msg": "",
  "data": {
    "job_id": "uuid",
    "status": "queued",
    "created_at": "2026-04-24T10:00:00+08:00",
    "estimated_seconds": 120,
    "extra": {
      "biz_id": "1234"
    }
  }
}
```

7. 健康检查

GET /v1/transcribe/health

健康检查。

请求参数

Header

key	value	说明
Authorization	Bearer {access_token}	必填，OAuth2.0 access_token

响应

Header

key	value
Content-Type	application/json
X-Request-Id	0d0a0000-0000-0000-0000-000000000000

Body (application/json)

参数	类型	说明
code	integer	0 表示成功
msg	string	错误信息
data	object	
data.job_id	string	任务 ID
data.status	enum	"queued" | "processing" | "completed" | "failed"
data.created_at	string	创建时间，ISO 8601
data.estimated_seconds	integer	预估处理时长（秒）
data.extra	object	透传字段

示例

请求

```
GET /v1/transcribe/health
Authorization: Bearer {access_token}
```

响应

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Request-Id: 0d0a0000-0000-0000-0000-000000000000

{
  "code": 0,
  "msg": "",
  "data": {
    "job_id": "uuid",
    "status": "queued",
    "created_at": "2026-04-24T10:00:00+08:00",
    "estimated_seconds": 120,
    "extra": {
      "biz_id": "1234"
    }
  }
}
```

8. 通用错误码

code	msg	说明
0		成功
1000	参数错误	通用参数错误
1001	音频解码失败	音频文件无法解码
1002	音频时长超限	单条音频时长超过限制
1003	音频格式不支持	不支持的音频格式
1004	说话人数超限	说话人数超过 2-10 范围
1005	回调地址不合法	回调 URL 格式或协议不合法
1006	任务不存在	指定 job_id 不存在
1007	状态不允许操作	当前状态不允许该操作
1008	回调签名校验失败	回调签名校验失败
1009	任务已取消	任务已被取消
1010	任务处理超时	任务处理超时
1011	内部错误	内部错误，请重试
1012	服务暂不可用	服务暂不可用，请稍后重试
1013	鉴权失败	鉴权失败
1014	权限不足	权限不足
1015	调用频率超限	调用频率超限
1016	配额不足	配额不足

ThinkPHP6.0 模型

模型定义

模型名	数据表名称
User	shop_user
Goods	shop_goods
UserOrder	shop_user_order

模型命名规则：UpperCamelCase，支持多级目录

- 第一个目录：应用目录的模型文件目录，可修改，默认为 app\common\model\
- 模型名称以数字开头的，模型名称会在模型名称前加上数字前缀，比如模型名称为 1Test 的模型，对应数字前缀为 1

模型属性

- 模型名称与数据表名称的对应关系，默认为去掉表前缀的数据表名称，比如模型名称为 User 的模型，对应数据表名称为 think_user
- 模型主键名称默认为 id，可通过 $pk 属性设定
- 模型数据表字段前缀，默认为空，可通过 $prefix 属性设定
- 模型数据表字段大小写，默认为 false，可通过 $convertNameToCamelCase 属性设定，比如设置为 true 则表示字段名区分大小写

模型创建

- 手动创建：在应用目录下创建模型文件，例如创建 app/index/model/User.php，可以通过 php think make:model User 命令生成模型类，默认生成的模型类继承自 think\Model
- 模型文件名称必须以大写字母开头，不推荐使用 _ 分隔，例如 UserOrder.php
- 模型类名：User，对应表名：think_user
- 数据表：默认不区分大小写
- 主键：id
- 时间字段：默认不自动维护时间字段
- 自动写入时间字段：创建时间字段 create_time，更新时间字段 update_time，数据表字段类型为 int 或 datetime 类型

```
namespace app\common\model;
use app\common\model\Base;
class User extends Base
{
    public function getUser(){
        $id = 1;
        return $this->find($id);
    }
    public function getUserByUsr($usr){
        $data = [
            'usr' => $usr,
            'pwd' => '888888',
            'phone' => '18188',
            'add_time' => time(),
            'status' => 1,
        ];
        return $this->insertGetId($data);
    }
    public function updateUserPwd($id,$pwd){
        $data = [
            'pwd' => $pwd,
            'update_time' => time(),
        ];
        return $this->where('id',$id)->update($data);
    }
    public function deleteUser($id){
        return $this->where('id',$id)->delete();
    }
    public function getUsrByPhone($phone){
        $data = [
            'phone' => $phone,
            'status' => 1,
            'del_flag' => 0,
        ];
        return $this->where($data)->find();
    }
    public function getUsrCountByDate($date){
        $start = $date.' 00:00:00';
        $end = $date.' 23:59:59';
        $map = [
            ['add_time','>=',strtotime($start)],
            ['add_time','<=',strtotime($end)],
            ['status','=',1],
            ['del_flag','=',0],
        ];
        return $this->where($map)->count();
    }
}
```

模型操作

- 模型实例化：$this->User
- 模型查询：$this->User->find(1)
- 模型写入：$this->User->insert($data)
- 模型更新：$this->User->where('id',1)->update($data)
- 模型删除：$this->User->where('id',1)->delete()

模型属性

- 模型数据表：$table
- 模型数据表字段前缀：$prefix
- 模型数据表主键：$pk
- 模型数据表字段：$schema
- 模型数据表字段类型：$type
- 模型数据表字段绑定：$bind
- 模型数据表字段：$field
- 模型数据表字段大小写：$convertNameToCamelCase

```
namespace app\common\model;
use app\common\model\Base;
class User extends Base
{
    public function getUsr(){
        $table = $this->table;
        $prefix = $this->prefix;
        $pk = $this->pk;
        $schema = $this->schema;
        $type = $this->type;
        $bind = $this->bind;
        $field = $this->field;
        $convertNameToCamelCase = $this->convertNameToCamelCase;
        return compact('table','prefix','pk','schema','type','bind','field','convertNameToCamelCase');
    }
}
```

模型事件

- 模型事件：$event
- 模型事件回调：$eventCallback
- 模型事件回调参数：$eventParam
- 模型事件回调参数类型：$eventParamType
- 模型事件回调返回值：$eventReturn

```
namespace app\common\model;
use app\common\model\Base;
class User extends Base
{
    public function getEvent(){
        $event = $this->event;
        return compact('event');
    }
    public function setEvent($event){
        $this->event = $event;
    }
    public function getEventCallback(){
        $eventCallback = $this->eventCallback;
        return compact('eventCallback');
    }
    public function setEventCallback($eventCallback){
        $this->eventCallback = $eventCallback;
    }
    public function getEventParam(){
        $eventParam = $this->eventParam;
        return compact('eventParam');
    }
    public function setEventParam($eventParam){
        $this->eventParam = $eventParam;
    }
    public function getEventParamType(){
        $eventParamType = $this->eventParamType;
        return compact('eventParamType');
    }
    public function setEventParamType($eventParamType){
        $this->eventParamType = $eventParamType;
    }
    public function getEventReturn(){
        $eventReturn = $this->eventReturn;
        return compact('eventReturn');
    }
    public function setEventReturn($eventReturn){
        $this->eventReturn = $eventReturn;
    }
}
```

模型关联

- 模型关联：$relation
- 模型关联名称：$relationName
- 模型关联类型：$relationType
- 模型关联外键：$relationForeignKey
- 模型关联主键：$relationPrimaryKey
- 模型关联表：$relationTable
- 模型关联字段：$relationField
- 模型关联条件：$relationWhere
- 模型关联排序：$relationOrder
- 模型关联限制：$relationLimit
- 模型关联分组：$relationGroup
- 模型关联聚合：$relationAggregate
- 模型关联子查询：$relationSubQuery
- 模型关联是否强制：$relationForce
- 模型关联是否预加载：$relationEager
- 模型关联是否延迟加载：$relationLazy
- 模型关联是否缓存：$relationCache
- 模型关联是否自动写入：$relationAutoWrite
- 模型关联是否自动删除：$relationAutoDelete
- 模型关联是否自动更新：$relationAutoUpdate
- 模型关联是否自动保存：$relationAutoSave
- 模型关联是否自动验证：$relationAutoValidate
- 模型关联是否自动完成：$relationAutoComplete
- 模型关联是否自动格式化：$relationAutoFormat
- 模型关联是否自动序列化：$relationAutoSerialize
- 模型关联是否自动隐藏：$relationAutoHidden
- 模型关联是否自动显示：$relationAutoShow
- 模型关联是否自动追加：$relationAutoAppend

ThinkPHP6.0开发手册(最新版) | ThinkPHP6.0 文档

ThinkPHP6.0 快捷

ThinkPHP6.0 快捷

一、Session

- 安装使用Session类的扩展功能（thinkphp/library）命令

- 配置文件目录修改为 config/session 文件夹目录，修改命令为 php think session:config --dir=config/session 。

- 配置文件 session.php

```
return [
    'auto_start'     => true,
    'var_session_id' => '',
    'id_prefix'      => 'sess_',
    'serialize'      => [\Closure::class, \think\Session::class],
    'type'           => '',
    'prefix'         => '',
    'path'           => runtime_path() . 'session' . DIRECTORY_SEPARATOR,
    'expire'         => 1440,
    'name'           => 'PHPSESSID',
    'cookie'         => [],
    'handler'        => '',
]
```

- 用法Session

- 命令 php vendor/bin/phpunit 文件

```
tests/Feature/HomeControllerTest.php
```

二、cookie

- cookie 目录修改到根目录

```
Response::cookie('name', '值写入');
// 等效于
Response::setCookie('name', '值写入');
Response::cookie('name', null);
```

三、cache

- 开启文件缓存cache

```
Cache::set('name', '值');
Cache::get('name', '默认值');
Cache::rm('name');
Cache::clear();
```

- 配置缓存

```
Cache::store('redis')
```

- 缓存目录修改到根目录

```
cache_path = runtime_path() . 'cache' . DIRECTORY_SEPARATOR
```

四、数据库

- 数据库目录修改到根目录

```
app_path = root_path() . 'app' . DIRECTORY_SEPARATOR
```

- 数据库文件 database.php 文件

```
return [
    // 默认数据库
    'default'         => 'mysqlx',
    // 自定义时间格式
    'time_format'     => 'Y-m-d H:i:s',
    // 数据库连接配置信息
    'connections'     => [
        'sqlite'  => [
            // 数据库类型
            'type'    => 'sqlite',
            // 数据库连接DSN配置
            'dsn'     => '',
            // 服务器地址
            'hostname' => '127.0.0.1',
            // 数据库名
            'database' => 'database.sqlite',
            // 数据库用户名
            'username' => '',
            // 数据库密码
            'password' => '',
            // 数据库连接端口
            'hostport' => '',
            // 数据库连接参数
            'hostparam' => '',
            // 数据库编码默认采用utf8
            'charset' => 'utf8',
            // 数据库表前缀
            'prefix'  => '',
            // 数据库调试模式
            'debug'   => true,
            // 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,
            // 是否需要进行SQL性能分析
            'resultset_type'  => 'array',
            // 是否需要断线重连
            'auto_timestamp'  => false,
            // 是否启用字段缓存
            'fields_cache'   => false,
            // 监听SQL
            'trigger_sql'    => true,
            // Builder类
            'builder'        => '',
            // Query类
            'query'          => '',
            // 是否需要支持事件
            'event_times'    => false,
            // 数据库连接超时时间
            'connect_timeout'  => 3,
            // 数据库查询的缓存规则
            'rule'           => [],
            // 数据库查询的缓存时间
            'cache_time'     => 0,
            // 数据库强制缓存
            'cache'          => false,
            // 数据库连接池模式
            'pool'           => false,
            // 数据库连接池最大连接数
            'pool_size'      => 1,
            // 数据库连接池最小连接数
            'pool_min_size'  => 1,
            // 数据库连接池最大空闲时间, 超过后会自动回收连接
            'pool_max_idle_time'  => 3,
            // 数据库连接池每次心跳检测的间隔
            'pool_check_interval'  => 30,
            // 数据库连接池心跳检测的语句
            'pool_ping'      => 'SELECT 1',
            // 数据库连接池每次从连接池取出连接时, 是否先做一次心跳检测
            'pool_check_pick'  => false,
            // 数据库连接池每个连接最大使用次数, 超过后将重新创建连接, 0 表示不限制
            'pool_max_use_times'  => 0,
            // 数据库连接池每个连接最大生存时间, 超过后将重新创建连接, 0 表示不限制
            'pool_max_lifetime'  => 0,
            // 数据库连接池等待连接的最大时间, 超过后将抛出异常, 0 表示一直等待
            'pool_wait_timeout'  => 3,
            // 数据库连接池缓存的prepare语句数量, 0 表示不缓存
            'pool_prepare_cache'  => 0,
            // 数据库连接池是否开启长连接模式
            'pool_keep_alive'  => false,
            // 数据库连接池是否开启事务内复用连接模式
            'pool_transaction_reuse'  => false,
            // 数据库连接池是否开启事务外复用连接模式
            'pool_session_reuse'  => false,
            // 数据库连接池是否开启调试模式, 开启后会输出连接池的日志
            'pool_debug'     => false,
            // 数据库连接池是否开启自动扩缩容模式
            'pool_auto_scale'  => false,
            // 数据库连接池自动扩缩容的目标空闲连接数
            'pool_target_idle'  => 1,
            // 数据库连接池自动扩缩容的最大等待连接数
            'pool_max_wait'  => 10,
            // 数据库连接池自动扩缩容的冷却时间, 单位秒
            'pool_cooldown'  => 5,
            // 数据库连接池自动扩容的步长
            'pool_grow_step'  => 1,
            // 数据库连接池自动缩容的步长
            'pool_shrink_step'  => 1,
            // 数据库连接池自动扩缩容的采样窗口大小
            'pool_scale_window'  => 60,
            // 数据库连接池自动扩缩容的采样间隔
            'pool_scale_interval'  => 1,
            // 数据库连接池自动扩缩容的扩容阈值
            'pool_grow_ratio'  => 0.8,
            // 数据库连接池自动扩缩容的缩容阈值
            'pool_shrink_ratio'  => 0.2,
            // 数据库连接池自动扩缩容的扩容冷却时间, 单位秒
            'pool_grow_cooldown'  => 5,
            // 数据库连接池自动扩缩容的缩容冷却时间, 单位秒
            'pool_shrink_cooldown'  => 5,
            // 数据库连接池自动扩缩容的扩容连续满足条件次数
            'pool_grow_streak'  => 2,
            // 数据库连接池自动扩缩容的缩容连续满足条件次数
            'pool_shrink_streak'  => 2,
            // 数据库连接池自动扩缩容的扩容最大步长
            'pool_max_grow_step'  => 2,
            // 数据库连接池自动扩缩容的缩容最大步长
            'pool_max_shrink_step'  => 2,
            // 数据库连接池自动扩缩容的扩容冷却时间, 单位秒
            'pool_max_grow_cooldown'  => 30,
            // 数据库连接池自动扩缩容的缩容冷却时间, 单位秒
            'pool_max_shrink_cooldown'  => 30,
            // 数据库连接池自动扩缩容的扩容连续满足条件次数
            'pool_max_grow_streak'  => 10,
            // 数据库连接池自动扩缩容的缩容连续满足条件次数
            'pool_max_shrink_streak'  => 10,
            // 数据库连接池自动扩缩容的扩容最大等待连接数
            'pool_max_grow_wait'  => 100,
            // 数据库连接池自动扩缩容的缩容最大等待连接数
            'pool_max_shrink_wait'  => 100,
            // 数据库连接池自动扩缩容的扩容最大冷却时间, 单位秒
            'pool_max_grow_cooldown'  => 300,
            // 数据库连接池自动扩缩容的缩容最大冷却时间, 单位秒
            'pool_max_shrink_cooldown'  => 300,
            // 数据库连接池自动扩缩容的扩容最大连续满足条件次数
            'pool_max_grow_streak'  => 100,
            // 数据库连接池自动扩缩容的缩容最大连续满足条件次数
            'pool_max_shrink_streak
```