# TP5备忘

# 目录结构详解

## Application目录

<应用目录，项目目录>

|-> index <模块目录>

| |-> controller <控制器目录>

| |-> model <模型目录>

| |-> view <视图目录>

|-> command.php <使用命令行运行项目时，会读取这里的配置信息>

|-> common.php <项目通用文件，全局函数，在项目任何地方都可调用>

|-> config.php <应用配置文件，整个应用的配置文件>

|-> database.php <数据库配置文件>

|-> route.php <路由配置文件，优化路由相关信息>

|-> tags.php <应用行为扩展文件，thinkphp中有很多的钩子，可以修改框架行为，但不用修改框架源码>

## Extend目录

<第三方库文件，非composer库文件>

## public目录

<网站根目录>

|->index.php <网站入口文件>

|->robots.txt <搜索引擎爬虫文件，可以配置禁止搜索引擎读取的信息>

|->router.php <通过命令行，执行这个文件可以启动一个类似apache的服务>

<php -S localhost:8080 router.php 启动服务>

|->static <静态文件目录/css/js/image等>

## Runtime目录

<网站运行的缓存文件，包括日志、缓存、编译文件等，temp里放的是编译后的html文件>

## Thinkphp目录

<thinkphp框架目录>

|-> base.php <定义一些常量>

|-> composer.json <composer的配置文件>

|-> console.php <控制台的配置文件>

|-> convention.php <框架默认的配置文件>

|-> helper.php <助手配置文件>

|-> license.txt <整个项目的说明文件>

|-> phpunit.xml <phpunit的配置文件>

|-> readme.md <说明文件>

|-> start.php <框架启动文件>

|-> lang目录 <语言包>

|-> library目录

| |-> traits <类库的扩展>

| |-> think <整个项目的核心文件>

|-> tpl<框架默认的模板目目录>

|-> default\_index.tpl <控制器的模板文件>

|-> dispatch\_jump.tpl <跳转成功或失败的跳转文件>

|-> page\_trace.tpl <调试的显示文件>

|-> think\_exception.tpl

## vendor目录

<composer类库文件夹>

# 开发规范

1. 目录使用小写+下划线的方式命名
2. 类的文件名均以命名空间定义，且命名空间和类库文件所在路径一致

< namespace app\index\controller >

1. 类文件采用驼峰命名法，首字母大写，其余文件小写 加 下划线命名

< Index.php >

1. 类名和文件名保持一致，大驼峰命名法

< class Index >

1. 函数名采用小驼峰命名法。

< function index() >

1. 属性采用小驼峰命名。
2. 常量使用大写 + 下划线连接
3. 数据库表名使用全小写 + 下划线连接，不能使用 下划线开头

# 模块设计

<?php

// application目录下\admin目录\controller目录

Namespace app\admin\controller

// Index与文件名相同

Class Index

{

Public function index(){}

}

## common模块

Thinkphp的公用模块，不能直接访问。TP5有自动加载功能，会根据use自动寻找文件。

### 使用方法

New类方法

1. application目录下用common模块
2. 在指定文件

//使用application目录下的common目录下的Index.php文件，别名为commonIndex

use app\common\Index as commonIndex

1. 初始化类

$com = new commonIndex();

-----------------------------------

继承使用

1. application目录下用common模块、
2. 在指定文件

//使用application目录下的common目录下的Index.php文件，别名为commonIndex

use app\common\Index as commonIndex

1. 继承类

Class Index extends commonIndex

1. 在指定函数中使用

$this->common中的函数名

# TP配置

使用config配置文件或使用config助手函数

惯例配置

位置 thinkphp\convention.php

应用配置

位置

扩展目录

需要特定的目录名 ”extra”

模块配置

位置 app\index\config.php

# Config类和助手函数

Use think\config;

或

$res = \think\config::get();

默认的助手文件位置

Thinkphp\helper.php助手文件

系统环境变量的配置使用

在根目录创建.env文件，文件中使用 email=1111111 的方式编写

Use think\env

$res = new env::get(“email”);

# url和路由

跳转

隐藏入口文件

配置apache重写路径，

\public\.htaccess文件定义了重写规则，

RewriteRule ^(.\*)$ index.php\$1 [QSA,TP,L]

绑定入口文件

但模块绑定后，不能访问其他模块

\public\index.php目录下

定义define(“BIND\_MODULE”,”admin”)

------------------------------

设置第2个入口文件，\public\ 新建api.php

Define(“APP\_PATH”,\_\_DIR\_\_ . “/../app/’);

Define(“CONF\_PATH”, \_\_DIR\_\_ . ‘/../conf/’);

//这里可以定义指定的模块，如果没有指定模块，会报错

Require(\_\_DIR\_\_ . ‘/../thinkphp/start.php’);

---------------------

可以使用config配置文件，当没有指定模块，可以访问其他模块

新建指定的api.php TP会去找app目录下的api模块，并访问controller\index.php

----------

开启路由

配置

url\_route\_on = true

Url\_route\_must = false

新建路由文件与config文件同级目录

Route.php

Return [

//标识/变量名 =》 要访问的路径

‘new/:id’ => “index/index/index

]

好处 可以直接打url路径

Echo url(‘index\index\info’, [‘id’=>10]

# 请求对象request

## 获取对象

建议使用3

1------------

Use think\Request;

Public function index(Request $request)

{

Print\_r($request);

}

2----------

Use think\request;

Public function index()

{

$res = request();

}

3---------

Use think\Request;

Public function index()

{

$res = request::instance();

}

得到参数

获取域名

$request->domain();

获取后缀

$request->pathinfo()；

$request->path

请求类型 get post

$resquest->method();

$resquest->isGet()

$resquest->isPost();

$resquest->isAjax();

请求参数

$resquest->get() 获取请求参数

$resquest->post()

$resquest->param()

获取session中的值

$resquest->session()

开启session

在配置信息设置session的

Session(“name”,”ssssss”);

Cookie(“name”,”dddd”);

读取cookel

$request->cookie();

获取模块控制器

$request->module() 模块名

$request->controller() 控制器名

$request->action() 方法名

查看request类源码

/think/request.php

# View

## 使用view对象

Return View();

## 默认模板位置

#application\模块名\view目录\index目录\index.html

#app\index\view\index\index.html

View();

## view(‘upload’);

#这里找的目录位置为

#application\模块名\view目录\控制器名目录\upload.html

## View(‘public/upload’)

#public会替换 控制名目录

#这里找的目录位置为

#application\模块名\view目录\public\upload.html

## View(‘./index.html’)

#这里会去找

#项目文件夹/public/index.html

#必须带.html ?

## View(‘index’,[‘xx’=>’ss’])

#这里会替换index模板上{$xx}，将{$xx}替换为ss

## View(‘index’,[‘xx’=>’ss’],[‘STATIC’=>’stass’])

#这里的第3个参数会替换掉页面中所有的 STATIC ，并将结果替换为stass

#在html页面中写 STATIC 即可

## 通过继承使用视图(推荐)

Use think\Controller;

Class Index extends Controller

1. 调用视图
   1. $this->fetch()
2. Fetch(‘upload’);

这里会去找 view\index\upload.html

1. Fetch(‘pulic\upload’)

这里会去找 view\public\upload.html

1. Fetch(‘./index.html’)

这里会去找 项目目录/public/index.hmtl

1. Fetch(‘index’,[‘xx’=>’ss’]

会替换页面中xx值

1. Fetch同样有第3个参数，用法与view一样

## Assign

需要继承controller

$this->assign(‘xxx’,’ssss’);

替换页面中{$xxx}为ssss

## Display

需要继承controller

#display不需要视图

$this->display(‘返回一条字符串’);

1. display(‘这是一{$xx}个支付’,[‘xx’=>’xxxxxxxxxx’])

这里会将{$xx}替换为xxxxxxxxxxxxx

1. Display也可以与assign一起使用

$this->assign(“xxxx”,”ssssss”);

$this->display(‘返回一条{$xxxx}字符串’)

# 模板

View\_replace\_str定义替换的字符串

\_\_CSS\_\_ 定义路径为static/css

模板的路径：

\_\_STATIC\_\_ 路径为/public/static

# 系统变量

在模板中使用，需要添加$Think

$think.server系统变量

$think.evn 参数

$Think.session 使用session

$Think.cookie 使用cookie

$Think.get.get键名 获取get值

$think.post.post键名 获取post值

$Think.const.APP\_PATH 获取系统常量

# 模板运算

{$aa | md5} 对字符串进行MD5加密

{$aa | substr=0,8}

substr为函数名

=后面是参数

{/\*这样注释不会显示在html中，右击参看源代码，看不到注释\*/}

# 模板标签

# 模板

## 使用模板

-----------------------------

可以在建立在view目录下建立common文件夹，这个目录可以修改

在需要引入的位置输入

// file是引入文件的路径

{include file=”common/nav” /}

------------------------

例：

<!--头部-->

<div class=”header”>

{include file=”common/nav” /}

</div>

## 模板继承

一、灵活性更高

1、新建base.html文件，这个文件是布局的基础文件（也可以说是layout），其他文件都继承这个文件

2、在控制器加载的index文件进行添加如下代码，即可完成继承

{extend name=”common/base” /}

----------------------------------------

二、在继承基础上修改不同的页面，需要使用{block}标签

1、在base.html页面中添加需要修改的模块：

{block name=”title”} ...内容1(默认输出内容)....{/block}

2、在需要修改的页面中使用，如index控制器调用的index.html页面

{block name=”title”}...要替换的内容...{/block}

如果没有替换内容，会输出base文件中默认内容

-----------------------------------------

三、在已存在的内容后进行添加，需要用{\_\_block\_\_}标签

//在控制器调用的Index.html页面进行添加

{block name=”title”}

//{\_\_block\_\_}是被继承模板的默认内容

添加内容{\_\_block\_\_}

{/block}

## Layout布局

灵活性不如模板灵活

可以使用include 标签

修改配置文件

Template项的

layout\_on => true

Layout\_name => ‘layout’

默认layout布局的模板位置为view目录下

{\_\_CONTENT\_\_} 会引入默认的index.html文件的内容 fatch()方法

# 数据库操作

## 数据库连接方法

修改batabase.php文件

Use think\Db

Db::connect(

[database中的返回值]

) 连接方法

## 查询方法

### query方法

DB::query(“ sql语句 ”)

### Execute方法

DB::execute(“ sql 字段=? 字段2=2”,[‘aa’,’BB’]);

### Table与name

Db::table() 需要填写表前缀

DB::name() 只使用表名即可，但需要配置表前缀

可以调用 where、select、find、value、column等方法

### select方法

#查询所有记录，返回结果是一个二维数组

#如果结果不存在，返回一个空数组

Db::table(‘表名’) -> select();

### Find方法

#查询1条记录，如果不使用where条件，默认查询ID为1的数据

#结果不存在，返回null

DB::table(‘表名’)->find();

### value方法

#返回1条记录

#结果不存在返回null

DB::table(‘表名’) -> value( ‘字段内容’ );

### Column方法

#返回一个数组，

#如果存在第2参数，就使用第2个参数作为结果的key值

#结果不存在，返回空

DB::table(‘表名’)->column( ‘字段内容’，’字段名’)

## 添加数据

#返回值是影响的行数

$db = Db::name(“表名”);

### 插入1条

$db->insert([

“字段名” => “值”

])

### 返回自增ID

$db->insertGetID([

‘字段名’ => ‘值’

])

### 插入多条数据

#$data为2维数组

#返回值为影响的行数

$db->insertAll($data);

## 数据库更新

$db = Db::name(“user’);

### Update

#更新id为1的字段，

#返回影响的行数

$db->where([

‘id’ => ‘1’

])->update([

‘更新的字段’ => ‘更新后的值’a,

‘更新字段2’ => ‘数据’

])

### setField

#只更新1个字段

#返回影响的行数

$db->where([

‘id’=>’2’

])->setField(‘字段名’,’值’)

### SetInc

#对指定字段进行加1

#返回影响数据的行数

$db->where([

‘id’=>1

])->setInc(‘num’);

------------------

#设置setInc的第2个值为5，则指定字段每次自增5

$db->where([

‘id’=>1

])->setInc(‘num’，5);

### SetDec

#返回影响行数

#对指定字段进行减1

$db->where([

‘id’=>1

])->setDec(‘num’);

------------------

#设置setInc的第2个值为5，则指定字段每次自减5

$db->where([

‘id’=>1

])->setDec(‘num’，5);

## 数据库删除操作

$db = Db::name();

### Delete

#删除指定表中id为12的数据

$db->where([

‘id’=>’12’

])->delete()

### 删除主键

$db->delete(‘主键’)

### 删除所有记录

$db->where(‘1=1’)->delete();

## 显示sql语句

$db->buildSql(); //可以打印即将执行的sql语句

## 条件构造器(where)

Param1 字段名

Param2 表达式

Param3 对比的值

Where(param1,param2,param3)

表达式备注

EQ =

NEQ != <>

LT <

ELT <=

GT >

EGT >=

#当使用betwween时，第3个参数应是“1，4”这种

BETWEEN BETWEEN \* AND \*

#第3个参数应是“[1,5]”

NOTBETWEEN

#第3个参数可以用[‘1’,’3’,’4’] 也可以用“1，3，4”

In

#Exp

WhereOr 会构造or sql语句

## 链式操作

# 模型（model）

Namespace app\index\model;

Use think\model;

Class User extends Model{}

## 命名规范

1. model名、文件名、与表名相同
2. 类名使用大驼峰
3. 当表名有下划线时，使用大驼峰命名法。例如：表明user\_info : model名UserInfo.php

## 控制器中调用model

1、//引入model，推荐使用，可以增强代码的可读性

Use app\index\model\User;

//直接使用静态方法，get是model中的方法

Use::get(3)

//new

$user = new User;

$user -> get(3) -> toArray();

----------------------------------------------------------

1. //使用Loader，加载多个model时只需要use 1次即可

Use think\Loader;

$us = Loader::model(“模型名”);

$us::get(4)

-------------------------------------------------------

1. //使用助手函数，有可能被覆盖

//不用use任何模型

$us = model(“ model名 “)

## 模型查询

User表名

### 获取ID=1条数据

$res = user::get(1)

-----------------------------

### 获取指定条件

$res = User::get(function($query{

$query->where();

})

------------------------------

### Find

$res = User::where(“id”,12)

->field(“id,username”) //获取指定字段

->find

--------------------------

### 获取多条数据-------------------

### All函数

#注意这里的结果是一个对象，不能使用toArray进行转换

#通过ID进行查询

$res = User:: all(“1，2，3”)

$res = User::All([‘1’,’2’,’3’]

#转换$res的结果为数组

Foreach($res as $val){

$val->toArray();

}

-----------------------------

### All函数的闭包写法

$res = User::all(function($query){

$query->where(“id”,”<”,5)

->field(“id,”email”); //查询指定字段

})

----------------------------

### 删除指定数据

$res = User::where(“id”,”<”,”15”)

->field(“username,email”)

->limit(3) //只获取3条

->order(“id DESC”) //排序

->select();

-----------------------------

### Value与column一样可以使用

## 模型添加数据

### create方法

$res = User::create([

‘字段名’ => “值”

])

----------------------------------

### 当数据库表不存在某些字段时，不操作字段

$res = User::create([

“字段名” => “值”

],true)

-------------------------------

### 添加指定字段

$res = User::create([

“字段名” => “值”

],[‘字段名’,’字段名’])

### save方法

$userModel = new User;

$userModel->字段名 = “值”

$userModel->字段名 = “值”

//添加进数据库

$userModel->save()

//获取添加后的ID

$userModel->ID

-----------------------------------------

### Save的类的写法

#res是受影响的行数

$userModel = new User;

$res = $userModel->save([

‘字段名” => “值”

“字段名” => “值”

])

---------------------------------

$userModel = new User

$res = $userModel->allowField(true) //没有的字段不添加

$res = $userModel->allowField(‘字段1’，‘字段2’) //添加指点字段

->save([

‘字段名” => “值”

“字段名” => “值”

])

---------------------------------

### 添加2条数据

$userModel = new User;

$res = $userModel ->saveAll([

[‘字段1’=>’值1’],

[‘字段2’=>’值2’]

])

#解析结果

Foreach($res as $val){

//打印数组

$val->toArray();

//打印影响的ID

$val->id

}

## 模型删除

### 通过主键删除

#返回受影响的行数

$res = User::destroy( 主键 )

-------------------------------------

### 删除指定字段

$res = User::destroy([‘字段名’=>‘值’])

----------------------------------

### 通过闭包函数删除

$res = user::destroy(function($query){

//where条件

$query->where(1);

})

### 使用对象删除

#返回值为影响的行数

$userModel = User::get(6);

$userModel->delete()

------------------------------

### 删除指定条件的值

#返回值是受影响的行数

$res = User::where( 条件 ) ->delete()

## 模型数据更新

### 不判断数据是否被更新

User：：update([

‘id’=> 1, //主键

‘键名’ => ‘值’

])

### 更新指定字段

#[‘id’=>2]更新id为2的字段

User::update([

‘键值’=>’值’

],[‘id’=>2])

----------------------------

#闭包函数

User::update([

‘键值’=>’值’

],function($query){

$query->where(‘id’,’<’,’5’)

})

-----------------------------

### 使用where条件

#返回值为影响的行数

User::where(“id”,”<”,6)

->update([

‘字段名’=>’值’

])

### 使用get

#根据主键获取指定对象

$userModel = User::get(1);

$userModel->字段1 = 值

$userModel->字段2 = 值

$userModel->save();

-------------------------

#更新iD为5的字段

$userModel = new User;

$res = $userModel->save([

‘键’ => ‘值’

],[‘id’=>5])

### 使用闭包函数更新

$userModel = new User;

$res = $userModel->save([

‘键’ => ‘值’

],function($query){

//更新ID小于5的字段

$query->where(“id”,”<”,”5”)

})

# 聚合模型

## 获取数据总条数

$res = User::count();

## 获取指定条件的条数

$res = User::where(“id”,”<”,5)->count();

## 获取指定字段的最大值

$res = User::max(‘字段名’)

## 获取某条件的字段值

$res = User::where(‘id’,’<’,’5’)->max(‘字段名’)

## 获取指定字段的和

$res = user::sum(‘ 字段名’)

## 获取某条件下的字段和

$res = user::where(‘id’,’<’,5)->sum(‘字段名’);

## 获取平均值

$res = User::avg(‘ 字段名’)

## 获取某条件下的平均值

$res = User::where(“id”,”<”,”5”)->avg(‘ 字段名’)

## 获取最小值

$res = User::min(‘字段名’)

# 模型获取器

自动对数据库中的值进行转换，例如0代表男，1代表女

#在模型中进行操作

#特定的方法名

#get 表明是模型获取器方法

#Sex 表中的字段名

#Attr 获取器设置

#这里可有第2个产生，第2个参数是传入的所有值

Public function getSexAttr($val){

操作。。。。。。

}

# 模型修改器+自动完成

自动完成有3个特殊的变量：

//自动完成，在数据新增修改的时候执行，操作指定字段

Protected $auto = [‘ 字段名1’，’字段名2’]

//在数据新增的时候执行，操作指定的字段

Protected $insert = [‘字段1’，’字段2’]

//在数据更新的时候执行，操作指定字段

Protected $update = [‘字段1’，‘字段2’]

--------------------------------------------

要实现自动完成，需要模型修改器的配合

-----------------------------------------------

#模型修改器 在模型中进行操作

#特定的方法名

#set 表明是模型修改器

#Sex 表中的字段名

#Attr 获取器设置

#这里可有第2个产生，第2个参数是传入的所有值

Public function getSexAttr($val){

操作。。。。。。

}

# 自动填写时间戳

TP框架提供自动填写时间戳的功能，

1. 在config.php文件中修改auto\_timestamp = true
2. 表中需要特殊的字段create\_time 与 update\_time

------------------------

指定Model中添加

//开启自动填写时间戳

1. Protected $autoWriteTimestamp = true;

//声明数据表中更新的字段名

1. Protected $createTime = ‘自定义字段名’
2. Protected $updateTime = ‘自定义字段名’

//关闭自动填写创建时间

4、Protected $ctrateTime = false

//关闭自动填写修改时间

1. Protected $updateTime = false

# 软删除（不全）

当配置软删除后，在使用get等model方法时无法获取值

User数据表/model

//获取经软删除的数据

User::withTrashed(true)->find( ‘id’)

//获取所有软删除的数据

User::onlyTrashed() -> select();