# thinkphp

* 基础

入口文件主要完成：

* 定义框架路径、项目路径（可选）
* 定义调试模式和应用模式（可选）
* 定义系统相关常量（可选）
* 载入框架入口文件（必须）

define('APP\_PATH','./Application/');

require './ThinkPHP/ThinkPHP.php';

如果你改变了项目目录（例如把Application更改为Apps），只需要在入口文件更改**APP\_PATH**常量定义即可：

define('APP\_PATH','./Apps/');

require './ThinkPHP/ThinkPHP.php';

THINK\_PATH 框架目录

APP\_PATH 应用目录

RUNTIME\_PATH 应用运行时目录（可写）

APP\_DEBUG 应用调试模式（默认false）

STORAGE\_TYPE 存储类型（默认为File）

APP\_MODE 应用模式（默认为common）

所有路径常量都必须以“/”结尾

例如，我们可以在入口文件中重新定义相关目录并且开启调试模式：

1. // 定义应用目录
2. define('APP\_PATH','./Apps/');
3. // 定义运行时目录
4. define('RUNTIME\_PATH','./Runtime/');
5. // 开启调试模式
6. define('APP\_DEBUG',True);
7. // 更名框架目录名称，并载入框架入口文件
8. require './Think/ThinkPHP.php';

最终应用目录结构

1. www WEB部署目录（或者子目录）
2. ├─index.php 应用入口文件
3. ├─Apps 应用目录
4. ├─Public 资源文件目录
5. ├─Runtime 运行时目录
6. └─Think 框架目录

为了避免某些服务器开启了目录浏览权限后可以直接在浏览器输入URL地址查看目录，系统默认开启了目录安全文件机制，会在自动生成目录的时候生成空白的index.html文件，当然安全文件的名称可以设置，例如你想给安全文件定义为default.html可以在入口文件中添加：

1. define('DIR\_SECURE\_FILENAME', 'default.html');
2. define('APP\_PATH','./Application/');
3. require './ThinkPHP/ThinkPHP.php';

如果你的环境足够安全，不希望生成目录安全文件，可以在入口文件里面关闭目录安全文件的生成，例如：

1. define('BUILD\_DIR\_SECURE', false);

* 配置

所有配置文件的定义格式均采用返回**PHP数组**的方式

配置参数不区分大小写，建议大写

1. //项目配置文件
2. return array(
3. 'DEFAULT\_MODULE' => 'Index', //默认模块
4. 'URL\_MODEL' => '2', //URL模式
5. 'SESSION\_AUTO\_START' => true, //是否开启session
6. //更多配置参数
7. //...
8. );

可以使用二维数组来配置更多的信息，例如：

1. //项目配置文件
2. return array(
3. 'DEFAULT\_MODULE' => 'Index', //默认模块
4. 'URL\_MODEL' => '2', //URL模式
5. 'SESSION\_AUTO\_START' => true, //是否开启session
6. 'USER\_CONFIG' => array(
7. 'USER\_AUTH' => true,
8. 'USER\_TYPE' => 2,
9. ),
10. //更多配置参数
11. //...
12. );

配置格式：php , yaml , ini , xml , xml , json

通过在入口文件中定义CONF\_EXT常量来指定格式，例如：

1. define('CONF\_EXT','.ini');

定义后，应用的配置文件（包括模块的配置文件）后缀都统一采用.ini。

配置加载顺序：



惯例配置->应用配置->模式配置->调试配置->状态配置->模块配置->扩展配置->动态配置

读取配置：**C('参数名称')**（Config） 获取已经设置的参数值，如：

1. $model = C('URL\_MODEL');

可以读取到系统的调试模式的设置值，同样，由于配置参数不区分大小写，因此

1. $model = C('url\_model');

是等效的，但是建议使用大写方式的规范。

如果url\_model尚未存在设置，则返回NULL。

1. C('my\_config',null,'default\_config');

如果my\_config尚未设置的话，则返回default\_config。

C方法也可以用于读取二维配置：

1. //获取用户配置中的用户类型设置
2. C('USER\_CONFIG.USER\_TYPE');

C(‘’)读取全部配置项

C(‘name’)读取配置项名称为name的配置值

C(‘name’,null,’default’)读取 配置项name的配置值，如未配置返回default

C(‘parent.sub’)读取二维数组配置

配置的名称不区分大小写，但二维数组配置读取的参数，parent是严格区分大小写

动态配置：C('参数名称','新的参数值');

例如，我们需要动态改变数据缓存的有效期的话，可以使用

1. // 动态改变缓存有效期
2. C('DATA\_CACHE\_TIME',60);
3. // 获取已经设置的参数值
4. C('USER\_CONFIG.USER\_TYPE');
5. //设置新的值
6. C('USER\_CONFIG.USER\_TYPE',1);

C(‘name’,$value)设置配置项名称为name的配置值

C($config\_array)批量配置

$config\_array中 Key为配置项名称，对应Value为配置值

扩展配置：

通过设置LOAD\_EXT\_CONFIG配置项的值来加载当前模块的自定义配置文件

* 控制器

1. 就是一个php类，继承thinkphp的核心类Controller
2. public外部访问
3. ACTION\_SUFFIX操作方法定义修改
4. 空控制器、空操作方法
5. 操作绑定到类
6. 调用四种方式：自动定位、new实例化、A函数、R函数

URL模式

四种：通过URL\_MODEL配置

0：普通模式 ( http://domain/index.php?m=m&c=c&a=a )

1：PathInfo ( http://domain/index.php?m/c/a )

2：ReWrite ( http://domain/m/c/a )

3：兼容模式

PathInfo和Rewrite需要运行环境的支持

URL动态生成

使用U函数可以动态生成模式匹配的URL:

U(‘地址表达式’,[‘参数’],[‘伪静态后缀’],[‘显示域名’])

-地址表达式：[模块/控制器/操作#锚点@域名]?参数1=值1&参数2=值2…

-第二个参数支持数组和字符串两种定义方式

-类unix系统对URL大小写是敏感的，通过配置URL\_CASE\_INSENSITIVE实现不区分大小写的URL访问

伪静态

为了满足更好的SEO效果，使用：

URL\_HTML\_SUFFIX 配置伪静态后缀

URL\_DENY\_SUFFIX 配置禁止访问的URL后缀

本地Apache服务器实现Rewrite模式

Rewrite是通过URL重写隐藏应用的入口文件index.php:

Apache开启mod\_rewrite.so模块

AllowOverride 设置为 All

定义分布式配置文件 htaccess

重启Apache服务后生效

实现更接近生产环境的URL

通过Apache开启mod\_vhost\_alias.so模块

配置httpd-vhosts.conf

添加hosts主机头实现转发

重启Apache服务后生效

跳转与AJAX数据返回

以下几种跳转和响应：

redirect重定向

参数（地址表达式 附加参数 跳转时间s 提示信息）

success成功提示跳转

参数（提示信息 跳转地址 跳转时间s）

error错误提示跳转

参数（提示信息 跳转地址 跳转时间s）

ajaxReturn实现Ajax数据返回

参数（返回数据 数据格式）

控制器定义了六个常量，用来快速判断请求的类型：

IS\_GET、IS\_POST、IS\_PUT、IS\_DELETE、IS\_AJAX、REQUEST\_METHOD（枚举）

不使用jquery的话，AJAX请求要添加识别参数：通过VAR\_AJAX\_SUBMIT配置

获取输入变量

以下几种方式：

$\_GET/$\_POST/$\_SERVER(不安全)

I函数（Input单词的首字母），方便实现输入变量读取，并自动进行数据安全过滤

Input函数：

I(‘变量类型.变量名’,[‘变量不存在默认值’],[‘过滤方法’],[‘额外数据源‘])

API补充

Action参数绑定是通过直接绑定URL地址中的变量作为操作方法的参数，可以简化方法的定义甚至路由的解析。

Action参数绑定功能默认是开启的，原理是把URL中的参数（不包括模块、控制器和操作名）和操作方法中的参数进行绑定。

* 视图

1.基本结构

*//入口文件绑定模块文件，新建一个模块  
define*(**'BIND\_MODULE'**,**'Views'**);  
*//关闭目录安全文件生成  
define*(**'BUILD\_DIR\_SECURE'**,**false**);

知识点：

1.1.默认视图层（DEFAULT\_V\_LAYER）

1.2.模板文件后缀（TMPL\_TEMPLATE\_SUFFIX）

1.3.连接符（TMPL\_FILE\_DEPR）

1.4.视图目录（VIEW\_PATH）

**'DEFAULT\_V\_LAYER'** => **'Template'**,  
**'TMPL\_TEMPLATE\_SUFFIX'** => **'.jike'**,  
**'TMPL\_FILE\_DEPR'** => **'\_'**,  
**'VIEW\_PATH'** => **'./Theme/Views/'**,

2.视图变量赋值和渲染(动态网页基础)

知识点：

* 1. 使用控制器的assign方法为视图变量赋值：

assign(‘name’,$value)

* 1. 渲染视图（控制器display、fetch和show方法）
  2. display(‘[模板文件]’[,’字符串编码’][,’输出类型’])
  3. fetch(‘[模板文件]’[,’字符串编码’][,’输出类型’])
  4. show(‘渲染内容’[,’字符串编码’][,’输出类型’])
  5. T函数：T([资源://][模块@][主题/][控制器/]操作,[视图分层])

3.主题

知识点：

3.1 设置默认主题（DEFAULT\_THEME）

3.2 控制器theme方法

3.3 多主题切换

3.4 差异主题（TMPL\_LOAD\_DEFAULTTHEME）

* 模板（上）

1. 变量输出

知识点：

1.1自定义变量输出：{$value}

1.2 默认值输出：{$value|default=’默认值’}

1.3系统变量($Think.server/env/post/get/cookie/session/request)

1.4使用运算符需要注意：数学运算，点语法失效

1.5模板解析定界符(TMPL\_L\_DELIM、TMPL\_R\_DELIM)

1. 使用函数

知识点：

2.1变量是第一个参数：{$value|function[=?,?,?]}，例如{$value|trim}

2.1变量不是第一个参数：{$value|str\_replace=?,?,###}}

2.3嵌套使用多个函数：{$value|func1|func2|func3}，优先级从左到右

2.4回归原生PHP：{:expression}

1. 包含文件和模板替换

知识点：

3.1包含文件：<include file=”模板表达式/文件路径[,…]” key=value/> [key]

如：<**include file="Public/navbar" home="主页"**/>

<**li class="active"**><**a href="#"**>[home]<**span class="src-only"**>(current)</**span**></**a**></**li**>

3.2导入资源文件：import、load、js、css

如<**import file="Js.bootstrap" type="js"** />  
<**load href="\_\_PUBLIC\_\_/js/jquery.js"**/>  
<**css href="\_\_PUBLIC\_\_/css/bootstrap.css"** />  
<**js href="\_\_CDN\_\_/js/bootstrap.js"** />

3.3 模板替换：TMPL\_PARSE\_STRING

如*// 模板替换规则定义* **'TMPL\_PARSE\_STRING'**=>**array**(  
 **'\_\_CDN\_\_'**=>**'./Cdn'**,  
 ),

3.4一些默认的替换规则：

\_\_ROOT\_\_，\_\_APP\_\_，\_\_MODULE\_\_，\_\_CONTROLLER\_\_，\_\_ACTION\_\_，\_\_SELF\_\_，\_\_PUBLIC\_\_

* 模型：定义与实例化

1模型的定义

知识点：

1.1模型是以PHP类的方式定义，继承自相应的模型类

1.2约定模型名称是不包括前缀的数据表名称，UserModel对应jike\_user数据表

1.3模型的属性：tablePrefix、tableName、trueTableName、dbName、connection

1.4实例化：new UserModel()、M(‘User’)、D(‘User’)和控模型

如：*//new*$user\_model=**new** \Models\Model\UserModel();  
  
*//M*$user\_m\_model=M(**'User'**);  
  
*//D*$user\_d\_model=D(**'User'**);  
  
*//控模型*$empty\_model=**new** \Think\Model();  
$empty\_m\_model=M();  
$empty\_d\_model=D();

2模型数据库信息配置

知识点：

2.1数据库信息配置

2.2切换数据库：db方法

2.3分布式数据库：DB\_DEPLOY\_TYPE

2.4读写分离：DB\_RW\_SEPARATE、DB\_MASTER\_NUM、DB\_SLAVE\_NO

如：*/\* 数据库设置 \*/***'DB\_TYPE'** => **'mysql'**, *// 数据库类型***'DB\_HOST'** => **'localhost'**, *// 服务器地址***'DB\_NAME'** => **'jike'**, *// 数据库名***'DB\_USER'** => **'root'**, *// 用户名***'DB\_PWD'** => **'root'**, *// 密码***'DB\_PORT'** => **'3306'**, *// 端口***'DB\_PREFIX'** => **'jike\_'**, *// 数据库表前缀***'DB\_CHARSET'** => **'utf8'**, *// 数据库编码默认采用utf8***'DB\_DEPLOY\_TYPE'** => 0, *// 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)***'DB\_RW\_SEPARATE'** => **false**, *// 数据库读写是否分离 主从式有效***'DB\_MASTER\_NUM'** => 1, *// 读写分离后 主服务器数量***'DB\_SLAVE\_NO'** => **''**, *// 指定从服务器序号***'DB\_2'**=>**array**(  
 **'DB\_TYPE'** => **'mysql'**, *// 数据库类型* **'DB\_HOST'** => **'localhost'**, *// 服务器地址* **'DB\_NAME'** => **'mysql'**, *// 数据库名* **'DB\_USER'** => **'root'**, *// 用户名* **'DB\_PWD'** => **'root'**, *// 密码* **'DB\_PORT'** => **'3306'**, *// 端口*),

3模型的字段定义

知识点：

3.1字段缓存

为什么要缓存字段信息：数据库中字段信息存入缓存

保存在哪里：缓存目录下的Data/\_fields/

以什么形式保存：数据库名称[.表前缀]+模型名称（小写）.php

如何关闭：DB\_FIELDS\_CACHE

获取模型的字段信息：getDbFields方法

如：**'DB\_FIELDS\_CACHE'** => **true**, *//启用字段缓存*

3.2字段定义

模型的fields属性

使用\_type定义字段数据类型

通过pk属性定义表的主键

* 模型：数据库操作

1. CURD操作

知识点：

* 1. Create（新增insert），使用add方法，注意并不是create方法
  2. Update（更新），使用save方法
  3. Read（读取），使用find、select
  4. Delete（删除），使用delete方法

1. 连贯操作

实例化的模型类每调用一次连贯操作方法，便会修改本身的属性值，最后以一个终结方法来结束连贯操作，对于模型来说就是进行数据库操作然后返回结果。

如：$topicModel->order(‘view desc’)->limit(10)->select()

order和limit是连贯方法，执行它们会返回模型本身，而select属于终结方法，执行数据库读取操作，返回符合条件的数据

连贯方法：

约束类：where、table、field(查询字段)、order、limit、page(分页)、group、having、index

多表辅助类：alias(指定别名)、join、union(合并查询)、relation(关联查询)

工具辅助类：distinct、lock、token(设置表单令牌)、fetchSql(返回sql语句)、cache、strict(长置检查字段)、comment(添加注释)、data

终结方法：find、select、getField、add、addAll、save、delete

1. 模型中定义命名范围

命名范围作用

如何定义：模型的\_scope属性

通过模型的scope方法调用

如：**protected $\_scope**=**array**(  
 **'latest'**=>**array**(  
 **'order'**=>**'create\_time desc'**,  
 **'limit'**=>10  
 ),  
 **'vip'**=>**array**(  
 **'where'**=>**array**(  
 **'is\_vip'**=>1  
 ),  
 ),  
);  
  
*//获取最近注册的10个用户***public function** getLatestUsers(){  
 *//连贯操作方法  
 //$result=$this->order('create\_time desc')->limit(10)->select();* $result=$this->scope(**'latest'**,**array**(**'limit'**=>20))->select();  
 **return** $result;  
}

*//获取最近注册的10个已经通过VIP认证的用户***public function** getLatestVipUsers(){  
  
 *//$conditon=array(  
 // 'is\_vip'=>1  
 //);  
 //$result=$this->where($conditon)->order('create\_time desc')->limit(10)->select();  
 //$result=$this->scope('vip')->scope('latest')->select();* $result=$this->vip()->scope(**'latest'**)->select();  
 **return** $result;  
}  
  
*//获取10个VIP认证用户，要求2015年1月1日之前注册,并且按照积分score从高到低排序***public function** getSomeUsers(){  
 $condition=**array**(  
 **'is\_vip'**=>1  
 );  
 $result=$this  
 ->where($condition)  
 ->where(**'create\_time <'**.*strtotime*(2015-1-1))  
 ->order(**'score desc'**)  
 ->limit(10)  
 ->select();  
 **return** $result;

1. 使用查询语言

查询条件的三种定义方式：字符串、数组和对象

表达式查询

快捷查询

区间查询

组合查询：\_string、\_query、\_complex

统计查询

SQL查询

动态查询

如：*//查询条件的三种定义方式***public function** sql\_1(){  
 *//字符串* $conditionString=**'status=1 OR is\_vip=1'**;  
  
 *//数组* $conditionArray=**array**(  
 **'status'**=>1,  
 **'is\_vip'**=>1,  
 **'\_logic'**=>**'OR'** );  
  
 *//对象* $conditonClass=**new** \stdClass();  
 $conditonClass->**status**=1;  
 $conditonClass->**is\_vip**=1;  
 $conditonClass->**\_logic**=**'OR'**;  
}  
  
*//表达式查询***public function** sql\_2(){  
 *//eq: =  
 //neq: <>  
 //gt: >  
 //egt: >=  
 //lt: <  
 //elt: <=  
 //between  
 //not between  
 //in  
 //not in* $condition=**array**(  
 **'username'**=>**array**(**'eq'**,**'linda'**),  
 **'email'**=>**array**(**'jike'**,**'%jike%'**),  
 **'score'**=>**array**(**'egt'**,1000),  
 **'age'**=>**array**(**'between'**,**'20,35'**),  
 **'id'**=>**array**(**'not in'**,**'1,6,99'**),  
 *//表达式，支持sql语法函数* **'score'**=>**array**(**'exp'**=>**'score+2'**)  
 );  
  
}  
  
*//快捷查询***public function** sql\_3(){  
 $condition=**array**(  
 **'username|email|mobile'**=>I(**'post.account'**),  
 **'password'**=>*md5*(I(**'post.password'**))  
 );  
}  
  
*//区间查询***public function** sql\_4(){  
 $condition=**array**(  
 **'age'**=>**array**(  
 **array**(**'egt'**,18),  
 **array**(**'elt'**,35)  
 ),  
 **'score'**=>**array**(  
 **array**(**'elt'**,99),  
 **array**(**'egt'**,1000),  
 **'or'** )  
 );  
}  
  
*//组合查询***public function** sql\_5(){  
 $subCondition=**array**(  
 **'status'**=>1,  
 **'age'**=>**array**(**'gt'**,18),  
 **'\_logic'**=>**'and'** );  
 $condition=**array**(  
 **'\_string'**=>**'status=1 AND age>18'**,  
 **'\_query'**=>**'status=1&age>18&\_logic=and'**,  
 **'\_complex'**=>$subCondition  
 );  
}  
  
*//统计查询***public function** sql\_6(){  
 $this->count();  
 $this->max(**'age'**);  
 $this->min(**'age'**);  
 $this->avg(**'age'**);  
 $this->sum(**'age'**);  
}  
  
*//sql查询***public function** sql\_7(){  
 M()->query(**'select \* form jike\_user where status=1'**);  
 M()->excute(**'update jike\_user set status=0 where id=1'**);  
}  
  
*//动态查询***public function** sql\_8(){  
 $this->getFieldByUsername(**'linda'**,**'email'**);  
 $this->getByUsername(**'linda'**);  
}

1. ActiveRecord模式

什么是ActiveRecord

ORM对象关系映射（Object Relational Mapping）用于实现面向对象编程语言里不同类型系统的数据之间的转换

基于ActiveRecord的CURD操作

如：*//新增用户***private function** createUser(){  
  
 $userAttribute=**array**(  
 **'username'**=>**'tom'**,  
 **'password'**=>*md5*(**'111'**),  
 **'email'**=>**'tom@jike.com'**,  
 **'create\_time'**=>*time*(),  
 **'status'**=>1  
 );  
 D(**'User'**)->add($userAttribute);  
}  
*//activeRecord模式的新增用户***private function** createUserAR(){  
 $user=D(**'User'**);  
  
 $user->**username**=**'tom'**;  
 $user->**password**=*md5*(**'111'**);  
 $user->**email**=**'tom@jike.com'**;  
 $user->**create\_time**=*time*();  
 $user->**status**=1;  
  
 $user->add();  
}

*//更新用户***private function** updateUserStatus($userId){  
  
 $userUpdateAttribute=**array**(  
 **'id'**=>$userId,  
 **'status'**=>0  
 );  
 D(**'User'**)->save($userUpdateAttribute);  
}  
*//activeRecord模式的更新用户***private function** updateUserStatusAR($userId){  
 $user=D(**'User'**);  
  
 $user->**id**=$userId;  
 $user->**status**=0;  
  
 $user->save();  
}

1. 定义字段映射

为什么要映射字段（防攻击）

模型的\_map属性（调用create方法时自动映射）

通过READ\_DATA\_MAP配置开启读取数据时的字段映射

模型的parseFieldsMap方法

* 模板（下）

1. 模板标签

内置标签：include、import、volist、foreach、for、switch、compare、present、empty、defined、if/else

输出php代码：PHP标签

原样输出：literal标签

1. 标签库

标签库的加载：taglib标签，标签预加载配置（TAGLIB\_PRE\_LOAD）

配置内置标签库：TAGLIB\_BUILD\_IN

1. 模板布局和继承

模板布局的三种方式：

全局模式：

LAYOUT\_ON

LAYOUT\_NAME

TMPL\_LAYOUT\_ITEM

标签模式：

<layout name=”布局名称“ replace=”替换字符串”/>

layout方法：

layout(bool)开启和关闭

模板的继承

extend标签：

模板继承标签

<extend name=”模板表达式/模板地址”/>

block标签：

<block name=”区块名称”/>

block重载：如果字模板中不进行重载则沿用，如果置空则删除

* 模型数据验证和填充

1. 数据验证和填充的概念和实现方式

B/S架构应用的数据安全风险主要集中在数据交互环节；

不要相信用户提交的任何数据，进行过滤和验证是必须的；

对于按照自定义规则生成的数据，如订单编号、积分变更，只能在服务器端进行填充；

模型作为数据库的业务抽象层，非常适合用于做数据的统一处理；

1. 实现数据自动验证

自动验证的四大要素：

对象：验证那个对象的数据

规则：按照什么规则验证

时机：何时、何种情况下验证

反馈：如何反馈验证结果

静态定义和动态定义

批量验证

1. 实现数据自动填充