



ThinkPHP Framework 1.0

ThinkTemplate Manual

ThinkPHP 1.0

模板指南

编写：ThinkPHP 文档组

最后更新：2008-05-26

目录

1	版权信息	4
2	模板指南	5
2.1	模板赋值	5
2.2	批量赋值	6
2.3	模板调用	6
2.4	加载标签库.....	7
2.5	模板注释	7
2.6	数组和对象.....	8
2.7	使用函数	8
2.8	系统变量	9
2.9	快捷输出	10
2.10	获取模版内容	10
2.11	包含外部文件	10
2.12	循环输出.....	11
2.13	Switch 标签	12
2.14	比较标签.....	13
2.15	If 标签.....	15
2.16	Present 标签.....	15
2.17	使用 PHP 代码	15
2.18	保持原样输出	16



2.19	便捷路径定义	16
2.20	导入外部 JS 和 CSS 文件	17
2.21	修改定界符	18
2.22	标签扩展.....	19

1 版权信息

发布本资料须遵守开放出版许可协议 1.0 或者更新版本。

未经版权所有者明确授权，禁止发行本文档及其被实质上修改的版本。

未经版权所有者事先授权，禁止将此作品及其衍生作品以标准（纸质）书籍形式发行。

如果有兴趣再发行或再版本手册的全部或部分内容，不论修改过与否，或者有任何问题，请联系版权所有者 liu21st@gmail.com。

对 ThinkPHP 有任何疑问或者建议，请进入官方论坛 [<http://bbs.thinkphp.cn>] 发布相关讨论。并在此感谢 ThinkPHP 团队的所有成员和所有关注和支持 ThinkPHP 的朋友。

有关 ThinkPHP 项目及本文档的最新资料，请及时访问 ThinkPHP 项目主站 <http://thinkphp.cn> 。

2 模板指南

ThinkPHP 内置了一个基于 XML 的性能卓越的模板引擎 ThinkTemplate ,这是一个专门为 ThinkPHP 服务的模板引擎。ThinkTemplate 是一个使用了 XML 标签库技术的编译型模板引擎，支持两种类型的模板标签，使用了动态编译和缓存技术，而且支持自定义标签库。其特点包括：

- 1、支持 XML 标签库和普通标签的混合定义；
- 2、编译一次，下次直接运行而无需重新编译；
- 3、模板文件更新后，自动更新模板缓存；
- 4、自动加载当前操作的模板缓存文件，无需指定模板文件名称；
- 5、支持编码转换和 Content-Type 更换；
- 6、模板变量输出前缀支持，避免变量名称冲突；
- 7、模板常量替换，无需设置模板变量；
- 8、支持变量组合调节器和格式化功能；
- 9、支持替换其它模板引擎插件使用；
- 10、支持获取模板页面内容

下面的内容描述了如何在 ThinkPHP 中使用内置模板引擎来完成模板文件的定义。

2.1 模板赋值

在 Action 类里面使用 assign 方法对模板变量赋值，无论何种变量类型都统一使用 assign 赋值。

```
$this->assign('name',$value);
```

系统只会输出设定的变量，其它变量不会输出，一定程度上保证了变量的安全性。

// 输出模版文件

```
$this->display();
```

//然后在模版文件定义

```
{ $name }
```

注意，不同的模板变量类型选择不同的模板标签来输出。

2.2 批量赋值

如果要同时输出多个模板变量，可以使用下面的方式：

```
$array = array();  
$array['name']      =    'thinkphp';  
$array['email']     =    'liu21st@gmail.com';  
$array['phone']     =    '12335678';  
$this->assign($array);
```

这样，就可以在模板文件中同时输出 name、email 和 phone 三个变量。

2.3 模板调用

模板变量赋值后就需要调用模板文件来输出相关的变量，模板调用通过 display 方法来实现，下面给出了 display 方法的基本使用。

```
// 假设当前操作是 User 模块下面的 read 操作  
  
// 调用 User 模块的 read 操作模版  
$this->display();  
  
// 调用 User 模块的 edit 操作模版  
$this->display('edit');  
  
// 调用 Member 模块的 read 操作模版  
$this->display('Member:read');  
  
// 调用 Xp 主题的 User 模块的 edit 操作模版  
$this->display('Xp@User:edit');  
  
// 直接指定模版文件的全名  
$this->display('../Member/read.html');
```

2.4 加载标签库

要在模板页面中使用 TagLib 标签库功能，需要在开始时候使用 taglib 标签导入需要使用的标签，防止以

后标签库大量扩展后增加解析工作量，用法如下：

```
<tagLib name="cx,html" />
```

引入标签库后，就可以使用标签库定义的标签来定义模板了，例如：

//可以使用下面的模板标签定义。

```
<cx:var name='user' property='name' />
<cx:present name='user.name' ></cx:present>
<html:select options='name' selected='value' />
```

标签库使用的时候忽略大小写，因此下面的方式一样有效：

```
<CX:VAR NAME='user' property='name' />
```

实际上，ThinkPHP 框架模板引擎会默认加载 CX 标签库，所以下面的方式效果相同：

```
<tagLib name='html' />
```

并且，默认加载的 CX 库可以不使用 CX 命名空间前缀，也就是说

```
<var name='user' property='name' />
```

//等效于 <cx:var name='user' property='name' />

要使用 Html 标签库，需要添加 html 命名前缀，如

```
<html:link href='/path/to/common.js' />
```

2.5 模板注释

模板支持注释功能，该注释文字在最终页面不会显示，仅供模板制作人员参考和识别。

格式：**`{/* 注释内容 */}`** 或 **`{// 注释内容 }`**

说明：在显示页面的时候不会显示模板注释，仅供模板制作的时候参考。

注意{和注释标记之间不能有空格。



例如：

```
{// 这是模板注释内容 }
```

2.6 数组和对象

如果要输出数组变量，可以使用下面的方式

```
{ $user['name'] }
```

如果要输出对象的属性，可以使用下面的方式

```
{ $user:name }
```

为了方便模板定义，无论输出的模板变量是数组还是对象，都可以用下列统一方式输出：

```
{ $user.name }
```

系统会自动判断要输出的变量，如果是多维数组或者多层对象属性的输出，请使用下面的定义方式：

```
{ $user['sub']['name'] }  
{ $user:sub:name }
```

2.7 使用函数

内置模板引擎支持对模板变量使用函数，并支持多个函数同时使用。

模板变量的函数调用格式为

格式：{ \$varname|function1|function2=arg1,arg2,### }

说明：

{ 和 \$ 符号之间不能有空格，后面参数的空格就没有问题

###表示模板变量本身的参数位置

支持多个函数，函数之间支持空格

支持函数屏蔽功能，在配置文件中可以配置禁止使用的函数列表

支持变量缓存功能，重复变量字串不多次解析

使用例子：

```
{ $webTitle|md5|strtoupper|substr=0,3}  
{ $number|number_format=2}
```

2.8 系统变量

除了常规变量的输出外，模板引擎还支持系统变量和系统常量、以及系统特殊变量的输出。它们的输出不需要对模板变量赋值。

系统变量（依然支持函数使用和大小写、空格），以 Think.打头，如

```
{ $Think.server.script_name } //取得$_SERVER 变量  
  
{ $Think.session.session_id|md5 } // 获取$_SESSION 变量  
  
{ $Think.get.pageNumber } //获取$_GET 变量  
  
{ $Think.cookie.name } //获取$_COOKIE 变量
```

系统常量

```
{ $Think.const.__FILE__ }  
{ $Think.const.MODULE_NAME }
```

特殊变量，由 ThinkPHP 系统定义的常量

```
{ $Think.version } //版本  
  
{ $Think.now } //现在时间  
  
{ $Think.template|basename } //模板页面  
  
{ $Think.LDELIM } //模板标签起始符号  
  
{ $Think.RDELIM } //模板标签结束符号
```

配置参数

```
{ $Think.config.db_charset }
```

语言变量

```
{Think.lang.page_error}
```

2.9 快捷输出

为了使得模板定义更加简洁，系统还支持一些常用的变量输出快捷标签，包括：

```
{:function(...)} //执行方法并输出返回值
```

```
{~function} //执行方法不输出
```

```
{@var} //输出 Session 变量
```

```
{&var} //输出配置参数
```

```
{%var} //输出语言变量
```

```
{.var} //输出 GET 变量
```

```
{^var} //输出 POST 变量
```

```
{*var} //输出常量
```

2.10 获取模版内容

// 在 Action 类里面使用

```
$content = $this->fetch();
```

fetch 的参数可以参考 Display 方法，基本一致

2.11 包含外部文件

可以使用 Include 标签来包含外部文件，使用方法如下：

// 使用一个完整的文件名包含

```
<include file="../../../Public/header.html" />
```


```
// 包含 Public 模块的 header 操作模版
<include file="Public:header" />

// 包含 blue 主题的 User 模块的 read 操作模版
<include file="blue@User:read" />

// 导入当前模块下面的 read 操作模版
<include file="read" />

// 用变量控制要导入的模版
<include file="$tplName" />

// 下面的方式等效于使用 include 标签
{include:"Public:header"}
```

 注意：由于模板解析的特点，从入口模板开始解析，如果外部模板有所更改，模板引擎并不会重新编译模板，除非缓存已经过期。如果遇到比较大的更改，您可以尝试把模块的缓存目录清空，系统就会重新编译，并解析到最新的外部文件了。

2.12 循环输出

iterate 循环输出标签

模版赋值

```
$User = D('User');
$list = $User->findAll();
$this->assign('list',$list);
```

模版定义

```
<iterate name="list" id="vo">
{$vo.name}
</iterate>
```

注意 name 和 id 表示的含义

// 输出 list 的第 5~15 条记录

```
<iterate name="list" id="vo" offset="5" length='10'>
{$vo.name}
</iterate>
```

// 输出偶数记录

```
<iterate name="list" id="vo" mod="2" >
<eq name="mod" value="1">
{$vo.name}
</eq>
</iterate>
```

// 输出 key

```
<iterate name="list" id="vo" key="k" >
{$k}.{$vo.name}
</iterate>
```

iterate 还有其它的别名，包括 volist,resultset,sublist

子循环输出

```
<volist name="list" id="vo">
<sublist name="vo['sub']" id="sub">
{$sub.name}
</sublist>
</volist>
```

注意子循环标签里面不能使用数组和对象自动判断的点语法

2.13 Switch 标签

模板引擎支持 Switch 标签，使用方法如下：

```
<switch name="name">
<case value="1">value1</case>
<case value="2">value2</case>
<default />default
</switch>
```

其中 name 属性可以使用函数以及系统变量，例如：

```
<switch name="Think.get.userId|abs">
<case value="1">admin</case>
<default />default
</switch>
```

也可以对 case 的 value 属性使用变量，例如：

```
<switch name="userId">
<case value="$adminId">admin</case>
<case value="$memberId">member</case>
<default />default
</switch>
```

2.14 比较标签

模板引擎提供了丰富的判断标签

// name 变量的值等于 value 就输出

```
<eq name="name" value="value">value</eq>
```

//或者

```
<equal name="name" value="value">value</equal>
```

// name 变量的值不等于 value 就输出

```
<neq name="name" value="value">value</neq>
```

//或者

```
<notequal name="name" value="value">value</notequal>
```

// name 变量的值大于 5 就输出

```
<gt name="name" value="5">value</gt>
```

// name 变量的值不小于 5 就输出

```
<egt name="name" value="5">value</egt>
```

// name 变量的值小于 5 就输出

```
<lt name="name" value="5">value</lt>
```

// name 变量的值不大于 5 就输出

```
<elt name="name" value="5">value</elt>
```

// name 变量的值恒等于 5 就输出

```
<heq name="name" value="5">value</heq>
```

// name 变量的值不恒等于 5 就输出

```
<nheq name="name" value="5">value</nheq>
```

或者统一使用 compare 标签，其实上面的所有标签都是 compare 标签的别名

// name 变量的值等于 5 就输出

// 其中 type 属性的值就是上面列出的判断标签名称

```
<compare name="name" value="5" type="eq">value</compare>
```

判断标签的高级用法

判断对象的属性或者数组

// 当 vo 对象的属性（或者数组，自动判断）等于 5 就输出

```
<eq name="vo.name" value="5">{$vo.name}</eq>
```

// 当 vo 对象的属性等于 5 就输出

```
<eq name="vo:name" value="5">{$vo.name}</eq>
```

// 当 \$vo['name'] 等于 5 就输出

```
<eq name="vo['name']" value="5">{$vo.name}</eq>
```

值是一个模版变量

// 当 vo 对象的属性等于 \$a 就输出

```
<eq name="vo:name" value="$a">{$vo.name}</eq>
```

使用函数

// 当 vo 对象的属性值的字符串长度等于 5 就输出

```
<eq name="vo:name|strlen" value="5">{$vo.name}</eq>
```

2.15 If 标签

复杂的条件判断可以使用 if 标签，例如：

```
<if condition="$name eq 1 "> value1
<elseif condition="$name eq 2" />value2
<else /> value3
</if>
```

在 condition 属性中可以支持 eq 等判断表达式，同上面的判断标签，但是不支持带有">”、”<”等符号的用法，因为会混淆模板解析。注意除判断表达式其他的模板标签使用将不被支持，所以请不要在

condition 属性里面使用点语法或者其他标签，但是可以直接书写 php 代码，例如：

```
<if condition="$user['name'] neq 'ThinkPHP' "> ThinkPHP
<else /> other Framework
</if>
```

2.16 Present 标签

可以使用 present 标签来判断模板变量是否已经赋值，例如：

```
<present name="name">name 已经赋值</present>
```

如果判断没有赋值，可以使用：

```
<notpresent name="name">name 还没有赋值</notpresent>
```

2.17 使用 PHP 代码

可以在模板文件里面书写任意的 PHP 语句代码，包括下面两种方式：

第一种使用 php 标签

```
<php>echo 'Hello,world!';</php>
```

或者直接使用原始的 php 代码

```
<?php echo 'Hello,world!'; ?>
```

2.18 保持原样输出

可以使用 literal 标签来防止模板标签被解析，例如：

```
<literal>
<if condition="$name eq 1 "> value1
<elseif condition="$name eq 2" />value2
<else /> value3
</if>
</literal>
```

上面的 if 标签被 literal 标签包含，因此 if 标签里面的内容并不会被模板引擎解析，而是保持原样输出。

2.19 便捷路径定义

如果使用了内置模板引擎的话，还可以在模板文件里面使用一些已经定义好的特殊字符串，系统在解析

模板的时候会自动替换成相关的系统常量。这些可替换的字符串包括：

```
../public //项目公共目录
__PUBLIC__ //网站公共目录
__ROOT__ //网站根目录
__APP__ //当前项目地址
__URL__ //当前模块地址
__ACTION__ //当前操作地址
__SELF__ //当前页面地址
```

使用例子：

例如你的应用地址是 <http://yourdomain.com/myApp/>

我们以 User 模块的 add 操作为例

<http://yourdomain.com/myApp/index.php/User/add>

上面的操作对应的模板文件应该是 add.html 文件

如果我们要在 add.html 模板文件里面调用项目公共目录的 css 文件，可以使用

```
<link rel='stylesheet' type='text/css' href='../Public/Css/style.css'>
```


那么，实际的解析后的模板文件应该是

```
<link rel='stylesheet' type='text/css' href='/myApp/Tpl/default/Public/Css/style.css'>
```

如果我们要调用网站公共目录下面的 JS 文件，可以使用

```
<script type="text/javascript" src="__PUBLIC__/Js/moo.fx.js"></script>
```

解析后的模板文件应该是

```
<script type="text/javascript" src="/Public/Js/moo.fx.js"></script>
```

如果我们需要在 add 操作页面提交表单到 User 模块的 insert 操作，我们可以在模板文件里面使用

```
<form action="__URL__/insert" method="post" >
```

解析后的模板文件就是

```
<form action="/myApp/index.php/User/insert" method="post" >
```

如果要提交到其他模块的操作，可以使用

```
<form action="__APP__/Member/insert" method="post" >
```

上面的表单提交到 Member 模块的 insert 操作

其他特殊字符串的使用方法类似，只是代表的含义不同。

2.20 导入外部 JS 和 CSS 文件

导入外部 Js 和 Css 文件除了使用上面提到的传统方式来导入外，系统还提供了两个标签用于专门导入。

Html 标签库的两个标签可以用于导入外部 JS 和 CSS 文件

第一个是 import 标签 用以导入网站的公共 JS 或者 CSS

例如：

```
<html:import type='js' file="Js.Util.Array" />
```

// 默认的 type 是 js 所以下面的效果等同

```
<html:import file="Js.Util.Array" />
```

```
<html:import type='css' file="Css.common" />
```

//上面的方式默认的 import 的 base 路径是网站的 Public 目录，如果需要指定其他的目录，可以使

用 basepath 属性，例如：

```
<html:import type='js' file="Js.Util.Array" basepath="./Common" />
```

Html link 标签

第二个是 link 标签 用以导入当前项目的公共 JS 或者 CSS

例如：

```
<html:link type="js" href="../../../Public/Js/Common.js" />
```

如果没有 type 属性，就会自动判断文件的后缀

```
<html:link href="../../../Public/Js/Common.js" />
<html:link href="../../../Public/Css/common.css" />
```

2.21 修改定界符

模版开始和结束标签定义

模板文件可以包含普通模板标签和 XML 模板标签，ThinkPHP 系统的普通模板标签默认以{ 和 } 作为开始和结束标识，并且在开始标记紧跟标签的定义，如果之间有空格或者换行则被视为非模板标签直接输出。

例如：{\$name} {\$vo.name} {\$vo['name']|strtoupper} 都属于普通模板标签

要更改普遍模板的起始标签和结束标签，请使用下面的配置参数：

TMPL_L_DELIM //模板引擎普通标签开始标记

TMPL_R_DELIM //模板引擎普通标签结束标记

普通模板标签主要用于模板变量输出、模板注释和公共模板包含。如果要使用其它功能，请使用 XML 模板标签，ThinkPHP 包含了一个基于 XML 和 TagLib 技术的模板标签，包含了普通模板有的功能，并且有一些方面的增强和补充，更重要的一点是新的标签库模板技术更加具有扩展性。新的 TagLib 标签库具有命名空间功能，ThinkPHP 框架内置了两个小型的标签库实现：CX 和 Html。

如果你觉得 XML 标签无法在正在使用的编辑器里面无法编辑，还可以更改 XML 标签库的起始和结束标签，请修改下面的配置参数：

```
TAGLIB_BEGIN      //标签库标签开始标签
```

```
TAGLIB_END        //标签库标签结束标记
```

注意 XML 标签和普通标签的定界符不能冲突，否则会导致解析错误。

XML 模板标签可以用于模板变量输出、文件包含、模板注释、条件控制、循环输出等功能，而且完全可以自己扩展功能。

2.22 标签扩展

任何一个模板引擎的功能都不是为你量身定制的，ThinkTemplate 具有一个良好的可扩展机制，Smarty 采用的是插件方法来实现扩展，ThinkTemplate 由于采用了标签库技术，比 Smarty 提供了更为强大的定制功能，和 Java 的 TagLibs 一样可以支持自定义标签库和标签，每个 XML 标签都有独立的解析方法，所以可以根据标签库的定义规则来增加和修改标签解析规则。

系统已经支持的标签库包括 CX 标签库和 HTML 标签库，你还可以根据自己需要扩展或者增加标签库，或者在原有的标签库基础上增加新的标签。

标签库由定义文件和解析类构成。每个标签库存在一个 XML 定义文件，用来定义标签库中的标签和属性。并且一个标签库文件对应一个标签库解析类，每个标签就是解析类中的一个方法。

要增加自己的标签库定义，需要首先在系统目录下面的 ThinkTemplate/Template/Tags/ 目录下面增加标签库定义文件，例如：

我们对 MyTags 标签库的定义文件格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>

<taglib>
```



```
<shortname>Mytags</shortname>

<tag>

<name>var</name>

<bodycontent>empty</bodycontent>

<attribute>

<name>name</name>

<required>true</required>

</attribute>

</tag>
```

标签库的名称和文件名一致，每个 tag 标签对定义了标签库中的一个标签，例如，下面的代码定义了

MyTags 标签库的 var 标签：

```
<tag>

<name>var</name>

<bodycontent>empty</bodycontent>

<attribute>

<name>name</name>

<required>true</required>

</attribute>

</tag>
```

Bodycontent 属性表示该标签是否属于闭合标签，然后

```
<attribute>

<name>name</name>
```

```
<required>true</required>
```

```
</attribute>
```

定义了其中的一个 name 属性，required 表示该属性是否为必须定义属性。

标签库定义文件完成后，我们就需要给标签库增加解析类。

标签库解析类的作用其实就是把某个标签定义解析成为有效的模版文件（可以包括 PHP 语句或者 HTML 标签）。每个标签的解析方法就是标签解析类的一个方法，为了不和系统的关键字冲突，所以在方法名前加上了_前缀，因此，假如要定义 MyTags:Var 的标签解析，就需要定义一个 _var 方法。

我们在 ThinkTemplate/Template/TagLib/ 目录下面增加一个 TagLibMyTags.class.php 文件：

```
<?php

import('Think.Template.TagLib');
Class TagLibMytags extends TagLib
{
    public function _var ($attr,$content)
    {
        $tag = $this->parseXmlAttr($attr,'var');
        $name = $tag['name'];
        $parseStr = "<?php echo $name ?> ";
        return $parseStr;
    }
}
```

```
?>
```

标签解析方法必须使用 return 返回一个字符串变量，才有页面输出。定义完成后，我们就可以使用这个标签库了，首先在模板文件的开头加上标签库引入

```
<tagLib name="myTags" />
```

然后在模板页中调用：

```
<mytags:var name='ThinkPHP' />
```

执行后可以看到页面输出的结果是

ThinkPHP

