第五章 视图和模板

5.1 模板变量与模板渲染

5.1.1 模板变量

新版框架默认只能支持PHP原生模板(模板中只能写PHP原生代码，才能被解析)，如果需要使用`thinkTemplate`模板引擎，需要安装`think-view`扩展（该扩展会自动安装`think-template`依赖库）。

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

配置文件

安装完成后，在配置目录的`view.php`文件中进行模板引擎相关参数的配置，例如：

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

模板赋值

模板中的变量（除了一些系统变量外）必须先进行模板赋值后才能使用，可以使用`assign`方法进行全局模板变量赋值

```
namespace app\controller;

use think\facade\View;

class Index
{
    public function index()
    {
        // 模板变量赋值
```

```
        View::assign('name','ThinkPHP');
        View::assign('email','thinkphp@qq.com');
        // 或者批量赋值
        View::assign([
            'name'  => 'ThinkPHP',
            'email' => 'thinkphp@qq.com'
        ]);
        // 模板输出
        return View::fetch('index');
    }
}
```

助手函数

如果使用`view`助手函数渲染输出的话，可以使用下面的方法进行模板变量赋值：

```
return view('index', [
    'name'  => 'ThinkPHP',
    'email' => 'thinkphp@qq.com'
]);
```

5.1.2 模板渲染

默认情况下，框架会自动定位你的模板文件路径，优先定位应用目录下的`view`目录，这种方式的视图目录下就是应用的控制器目录。

单应用模式

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

多应用模式

如果是多应用模式的话，这种方式下`view`目录下面首先是应用子目录。

```
├─view                    视图文件目录
|  ├─index（应用视图目录）
|  |   ├─index             index控制器目录
|  |   |  └─index.html     index模板文件
|  |   └─ ...              更多控制器目录
```

模板渲染的最典型用法是直接使用`fetch`方法，不带任何参数：

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

use think\facade\View;

class Index
{
    public function index()
    {
        // 不带任何参数 自动定位当前操作的模板文件
        return View::fetch();
```

```
    }
}

注意：  表示系统会按照默认规则自动定位视图目录下的模板文件，其规则是：
        控制器名（小写+下划线）/操作名.html
        默认的模板文件名规则改为实际操作方法名的小写+下划线写法
        控制器（小写+下划线）_操作.html
```

如果没有按照模板定义规则来定义模板文件（或者需要调用其他控制器下面的某个模板），可以使用：

```
// 指定模板输出
return View::fetch('edit');
```

表示调用当前控制器下面的edit模板

```
return View::fetch('member/read');
```

跨应用渲染模板

```
return View::fetch('admin@member/edit');
```

5.2 变量输出

5.2.1 变量输出

在模板中输出变量的方法很简单，例如，在控制器的方法中我们给模板变量赋值：

```
$this->assign('name', 'thinkphp');
return $this->fetch();
```


然后就可以在模板中使用：

```
Hello,{$name}!
```

模板编译后的结果就是：

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

这样，运行的时候就会在模板中显示： Hello,ThinkPHP!
注意模板标签的{和$之间不能有任何的空格，否则标签无效。所以，下面的标签

```
Hello,{ $name}!   // 将不会正常输出name变量，而是直接保持不变输出： Hello,{ $name}!
```

模板标签的变量输出根据变量类型有所区别，刚才我们输出的是字符串变量，如果是数组变量，

```
$data['name'] = 'ThinkPHP';
$data['email'] = 'thinkphp@qq.com';
$this->assign('data',$data);
```

5.2.2 其他

默认值

我们可以给变量输出提供默认值，例如：

```
{$user.nickname|default="这家伙很懒，什么也没留下"}
```

对系统变量依然可以支持默认值输出，例如：

```
{$Request.get.name|default="名称为空"}
```

系统变量输出

普通的模板变量需要首先赋值后才能在模板中输出，但是系统变量则不需要，可以直接在模板中输出，系统变量的输出通常以`{$Request.`（大小写一致）打头，例如：

```
{$Request.server.script_name} // 输出$_SERVER['SCRIPT_NAME']变量
{$Request.session.user_id} // 输出$_SESSION['user_id']变量
{$Request.get.page} // 输出$_GET['page']变量
{$Request.cookie.name}  // 输出$_COOKIE['name']变量
```

支持输出 $*SERVER*、$ENV、 $*POST*、 $GET、 $*REQUEST*、$SESSION和 $_COOKIE变量。
如果在ThinkPHP6.0中使用的话，模板还支持直接输出Request请求对象的方法，用法如下：

```
$Request.方法名.参数
例子：{$Request.param.name}

其他：
// 调用Request对象的param方法 传入参数为name
{$Request.param.name}
// 调用Request对象的param方法 传入参数为user.nickname
{$Request.param.user.nickname}
// 调用Request对象的root方法
{$Request.root}
// 调用Request对象的root方法，并且传入参数true
{$Request.root.true}
// 调用Request对象的path方法
{$Request.path}
// 调用Request对象的module方法
{$Request.module}
// 调用Request对象的controller方法
{$Request.controller}
// 调用Request对象的action方法
{$Request.action}
// 调用Request对象的ext方法
{$Request.ext}
// 调用Request对象的host方法
{$Request.host}
// 调用Request对象的ip方法
{$Request.ip}
// 调用Request对象的header方法
{$Request.header.accept-encoding}

注意：支持Request类的大部分方法，但只支持方法的第一个参数。
```


配置输出

仅用于输出ThinkPHP`6.0+`中的配置参数使用：

```
{$Think.config.app.app_host}
{$Think.config.session.name}
```

5.3 函数与运算符

5.3.1 使用函数

需要对模板输出使用函数进行过滤或其它处理的时候，可以使用：

```
{$data.name|md5}
// 编译后的结果
<?php echo htmlentities(md5($data['name'])); ?>  // 其中htmlentities方法是系统默认添加的（无需手动指定。

// 如果你不需要转义（例如你需要输出html表格等内容），可以使用：
{$data.name|raw}

// 编译后的结果
<?php echo $data['name']; ?>
```

系统内置了下面几个固定的过滤规则（不区分大小写）

过滤方法	描述
date	日期格式化（支持各种时间类型）
format	字符串格式化
upper	转换为大写
lower	转换为小写
first	输出数组的第一个元素
last	输出数组的最后一个元素
default	默认值
raw	不使用（默认）转义

还可以支持多个函数过滤，多个函数之间用“|”分割即可，例如：

```
{$name|md5|upper|substr=0,3}
```

如果你觉得这样写起来比较麻烦，也可以直接这样写：

```
{:substr(strtoupper(md5($name)),0,3)}

注意：使用该方法输出的值不会使用默认的过滤方法进行转义。
```

5.3.2 运算符

我们可以对模板输出使用运算符，包括如下支持。

运算符	使用示例
+	{$a+$b}
-	{$a-$b}
*	{$a*$b}
/	{$a/$b}
%	{$a%$b}
++	{$a++} 或 {++$a}
--	{$a--} 或{--$a}
综合运算	{$a+$b*10+$c}

在使用运算符的时候，不再支持前面提到的函数过滤用法，例如：

```
{$user.score+10} //正确的
{$user['score']+10} //正确的
{$user['score']*$user['level']} //正确的
{$user['score']|myFun*10} //错误的
{$user['score']+myFun($user['level'])} //正确的
```

三元运算

```
{$status? '正常' : '错误'}
{$info['status']? $info['msg'] : $info['error']}
{$info.status? $info.msg : $info.error }

// 还支持如下写法
{$name ?? '默认值'}

// 表示如果有设置$name则输出$name,否则输出默认值。
{$name?='默认值'}
```


5.4 原样输出和模板注释

5.4.1 原样输出

可以使用`literal`标签来防止模板标签被解析，例如：

```
{literal}

Hello,{$name}!

{/literal}
```

上面的`{$name}`标签被`literal`标签包含，因此并不会被模板引擎解析，而是保持原样输出。

`literal`标签还可以用于页面的JS代码外层，确保JS代码中的某些用法和模板引擎不产生混淆。

总之，所有可能和内置模板引擎的解析规则冲突的地方都可以使用`literal`标签处理。

5.4.2 模板注释

```
多行注释
格式：
{// 这是模板注释内容 }              // 注意：注意{和注释标记之间不能有空格。

多行注释
{/* 这是模板
注释内容*/ }

 注意：模板注释支持多行，模板注释在生成编译缓存文件后会自动删除，这一点和Html的注释不同。
```

5.5 模板布局和模板继承

5.5.1 模板布局

ThinkPHP支持三种模板布局，在这里讲述其中的一种。

在讲解模板布局之前先了解包含文件，有助于理解模板布局。

在当前模版文件中包含其他的模版文件使用include标签，标签用法：

```
{include file='模版文件1,模版文件2,...' /}
```

使用模板表达式

```
模版表达式的定义规则为：模块@控制器/操作
例如：
{include file="public/header" /}          // 包含头部模版header
{include file="public/menu" /}            // 包含菜单模版menu
{include file="blue/public/menu" /}       // 包含blue主题下面的menu模版

可以一次包含多个模版，例如
{include file="public/header,public/menu" /}
```


定义布局文件 layout/layout.html（名称并不固定）

布局模板的写法和其他模板的写法类似，本身也可以支持所有的模板标签以及包含文件，区别在于有一个特定的输出替换变量`{__CONTENT__}`，例如，下面是一个典型的layout.html模板的写法

```
{include file="public/header" /}
 {__CONTENT__}
{include file="public/footer" /}
```

在模板文件中指定布局模板

比如当前模板是 index.html,在头部增加下面的布局标签（记得首先关闭前面的`layout_on`设置，否则可能出现布局循环）：

```
{layout name="layout" /}
```

原理解析

当渲染`index.html`模板文件的时候，如果读取到layout标签，则会把当前模板的解析内容替换到layout布局模板的`{CONTENT}`特定字符串。

5.5.2 模板继承

模板继承是一项更加灵活的模板布局方式，模板继承不同于模板布局，甚至来说，应该在模板布局的上层。模板继承其实并不难理解，就好比类的继承一样，模板也可以定义一个基础模板（或者是布局），并且其中定义相关的区块（block），然后继承（extend）该基础模板的子模板中就可以对基础模板中定义的区块进行重载。

block标签必须指定name属性来标识当前区块的名称，这个标识在当前模板中应该是唯一的，block标签中可以包含任何模板内容，包括其他标签和变量，例如：

```
{block name="title"}<title>{$web_title}</title>{/block}
```

你甚至还可以在区块中加载外部文件：

```
{block name="include"}{include file="Public:header" /}{/block}
```

一个模板中可以定义任意多个名称标识不重复的区块，例如下面定义了一个`base.html`基础模板：

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>{block name="title"}标题{/block}</title>
</head>
<body>
{block name="menu"}菜单{/block}
{block name="left"}左边分栏{/block}
{block name="main"}主内容{/block}
{block name="right"}右边分栏{/block}
{block name="footer"}底部{/block}
</body>
</html>
```


然后我们在子模板（其实是当前操作的入口模板）中使用继承：

```
{extend name="base" /}

{block name="title"}{$title}{/block}

{block name="menu"}
<a href="/" >首页</a>
<a href="/info/" >资讯</a>
<a href="/bbs/" >论坛</a>
{/block}

{block name="left"}{/block}

{block name="main"}
{volist name="list" id="vo"}
<a href="/new/{$vo.id}">{$vo.title}</a><br/>
 {$vo.content}
{/volist}
{/block}

{block name="right"}
 最新资讯：
{volist name="news" id="new"}
<a href="/new/{$new.id}">{$new.title}</a><br/>
```

```
{/volist}
{/block}

{block name="footer"}
{__block__}
 @ThinkPHP 版权所有
{/block}
```

5.6 输出替换

支持对模板文件输出的内容进行字符替换，定义后在渲染模板或者内容输出的时候就会自动根据设置的替换规则自动替换。
如果需要全局替换的话，可以直接在配置文件中添加：

```
'tpl_replace_string'  =>  [
    '__STATIC__'=>'/static',
    '__JS__' => '/static/javascript',
]
```

5.7 标签库

变量输出使用普通标签就足够了，但是要完成其他的控制、循环和判断功能，就需要借助模板引擎的标签库功能了，系统内置标签库的所有标签无需引入标签库即可直接使用。

内置标签主要包括：

标签名	作用	包含属性
include	包含外部模板文件（闭合）	file
load	导入资源文件（闭合 包括js css import别名）	file,href,type,value,basepath
volist	循环数组数据输出	name,id,offset,length,key,mod
foreach	数组或对象遍历输出	name,item,key
for For	循环数据输出	name,from,to,before,step
switch	分支判断输出	name
case	分支判断输出（必须和switch配套使用）	value,break
default	默认情况输出（闭合 必须和switch配套使用）	无
compare	比较输出（包括eq neq lt gt egt elt heq nheq等别名）	name,value,type
range	范围判断输出（包括in notin between notbetween别名）	name,value,type
present	判断是否赋值	name
。。。。。。。。。。		

内置的标签很多，我们不能每一个都细讲，在这里我们详解几个常用且具有代表性的。

5.7.1 循环标签

foreach 标签

`foreach`标签的用法和PHP语法非常接近，用于循环输出数组或者对象的属性，用法如下：

```
$list = User::all();
View::assign('list',$list);
```

模板文件中可以这样输出

```
{foreach $list as $key=>$vo }
    {$vo.id}:{$vo.name}
{/foreach}
```

volist 标签

```
{volist name="list" id="vo"}
{$vo.id}:{$vo.name}<br/>
{/volist}
```

可以直接使用函数设定数据集，而不需要在控制器中给模板变量赋值传入数据集变量，如：

```
{volist name=":model('user')->all()" id="vo"}
{$vo.name}
{/volist}
```

支持输出查询结果中的部分数据，例如输出其中的第5～15条记录

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

输出偶数记录

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


5.7.2 条件判断

switch 标签

```
{switch 变量 }
    {case value1 }输出内容1{/case}
    {case value2}输出内容2{/case}
    {default /}默认情况
{/switch}
```

使用实例：

```
{switch User.level}
    {case 1}value1{/case}
    {case 2}value2{/case}
    {default /}default
{/switch}
```

if 标签

```
{if 表达式}value1
{elseif 表达式 /}value2
{else /}value3
{/if}
```

用法示例：

```
{if ( $name == 1) OR ( $name > 100) } value1
{elseif $name == 2 /}value2
{else /} value3
{/if}

// 可以使用PHP代码
{if strtoupper($user['name']) == 'THINKPHP' }ThinkPHP
{else /} other Framework
{/if}
```

5.7.3 资源文件

传统方式的导入外部 `js` 和 `css` 文件的方法是直接在模板文件使用：

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

系统提供了专门的标签来简化上面的导入：

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

并且支持同时加载多个资源文件，例如：

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

各位同学，以上就是视图模板的全部内容。