**模版讲解：**

* {{ ... }}：装载一个变量，模板渲染的时候，会使用传进来的同名参数这个变量代表的值替换掉。
* {% ... %}：装载一个控制语句。
* {# ... #}：装载一个注释，模板渲染的时候会忽视这中间的值。

**变量**

1）在模板中添加变量，可以使用（set）语句。

{% set name='xx' %}

之后就可以在页面文件中使用name这个变量了。在解释性语言中，变量的类型时运行时确定的，因此，这里的变量可以赋任何类型的值。

上面的语句创建的是全局变量，从定义之后的文件部分中都可以访问 。

2）局部变量

可以使用with语句来创建一个内部的作用域，将set语句放在其中，这样创建的变量只在with代码块中才有效。

{% with foo = 42 %}

{{ foo }}

{% endwith %}

这样，foo变量就只能在with标签间可以使用。

**控制语句**

控制语句都是放在{% ... %}中，并且有一个语句{% endxxx %}来进行结束。

1:if语句

{% if kenny.sick %}

Kenny is sick.

{% elif kenny.dead %}

You killed Kenny! You bastard!!!

{% else %}

Kenny looks okay --- so far

{% endif %}

2:for循环

1）普通用法

<ul>

{% for user in users %}<li>{{ user.username|e }}</li>

{% endfor %}</ul>

2）遍历字典

{% for key, value in **my\_dict.iteritems()** %}<dt>{{ key|e }}</dt><dd>{{ value|e }}</dd>

{% endfor %}

3）在循环中加入else

<ul>

{% for user in users %}<li>{{ user.username|e }}</li>

{% else %}<li><em>no users found</em></li>

{% endfor %}</ul>

4）Jinja2中for循环内置常量

|  |  |
| --- | --- |
| loop.index | 当前迭代的索引（从1开始） |
| loop.index0 | 当前迭代的索引（从0开始） |
| loop.first | 是否是第一次迭代，返回True\/False |
| loop.last | 是否是最后一次迭代，返回True\/False |
| loop.length | 序列的长度 |

**注意：不可以使用**continue**和**break**表达式来控制循环的执行。**

4）Jinja2中for循环内置常量

|  |  |
| --- | --- |
| loop.index | 当前迭代的索引（从1开始） |
| loop.index0 | 当前迭代的索引（从0开始） |
| loop.first | 是否是第一次迭代，返回True\/False |
| loop.last | 是否是最后一次迭代，返回True\/False |
| loop.length | 序列的长度 |

**运算符**

* +号运算符：可以完成数字相加，字符串相加，列表相加。但是并不推荐使用+运算符来操作字符串，字符串相加应该使用~运算符。
* -号运算符：只能针对两个数字相减。
* /号运算符：对两个数进行相除。
* %号运算符：取余运算。
* \*号运算符：乘号运算符，并且可以对字符进行相乘。
* \*\*号运算符：次幂运算符，比如2\*\*3=8。
* in操作符：跟python中的in一样使用，比如{{1 in [1,2,3]}}返回true。
* ~**号运算符：拼接多个字符串**，比如{{"Hello" ~ "World"}}将返回HelloWorld。

**Jiaja2模版最重要的部分——宏**

宏相当于一个搭建好的页面一部分，可以被引入，可以往宏传递参数。可以将一些经常用到的代码片段放到宏中，然后把一些不固定的值抽取出来当成一个变量，在使用宏时传递参数，从而将宏渲染成为页面的一部分。

比如：定义一个input标签宏

{% macro input(name, value='', type='text') %}

<input type="{{ type }}" name="{{ name }}" value="{{value|e }}">

{% endmacro %}

在其它地方使用这个宏快速创建出符合要求的input标签：

<p>{{ input('username') }}</p>

<p>{{ input('password', type='password') }}</p>

1）页面文件中导入宏——import

在开发中，会将一些常用的宏单独放在一个文件中，在需要使用的时候，再从这个文件中进行导入。

import语句的用法跟python中的import类似，可以直接import...as...，也可以from...import...或者from...import...as...。

{% import 'forms.html' as forms %} //导入宏文件

<dl>

<dt>Username</dt>

<dd>{{ forms.input('username') }}</dd> //使用宏

<dt>Password</dt>

<dd>{{ forms.input('password',type='password') }}</dd>

</dl>

<p>{{ forms.textarea('comment') }}</p>

导入模板并不会把当前上下文中的变量添加到被导入的模板中，我们可以在导入的时候使用with context 把上下文传进去：

{% from '\_helpers.html' import my\_macro with context %}

**模版文件的继承**

模板可以继承，通过继承可以把模板中许多重复出现的元素抽取出来，放在父模板中，并且父模板通过定义block给子模板开一个口，子模板根据需要，再实现这个block进行具体内容定义。

比如：父模版base.html如下：

<!DOCTYPE html>

<html lang="en">

<head>

{% block head %} //开放一个地方，以待具体赋值

<link rel="stylesheet" href="style.css" />

<title>{% block title %}{% endblock %} - My Webpage</title>

{% endblock %}

</head>

<body>

<div id="content">{% block content %}{% endblock %}</div>

<div id="footer">

{% block footer %}

© Copyright 2008 by <a href="http://domain.invalid/">you</a>.

{% endblock %}

</div>

</body>

</html>

然后定义子模版，对父模板中的block部分进行**覆盖书写**：

{% extends "base.html" %}//1:继承父模板

{% block title %}Index{% endblock %}／／2:书写title block

{% block head %}//3：书写head block

{{ super() }}//调用父模板中的内容，如果不调用，则此处会被子模板中书写的内容覆盖掉

<style type="text/css">

.important { color: #336699; }

</style>

{% endblock %}

{% block content %}//4:书写content block

<h1>Index</h1>

<p class="important">

Welcome to my awesome homepage.

</p>

{% endblock %}

另外：模板文件中对block内容的调用，可以使用 self.blockName 的方式。

<title>{% block title %}{%endblock %}</title>

<h1>{{ self.title() }}</h1>//调用title block的内容

注意：在子模板中，所有的标签和代码都要添加到从父模板中继承的block中。否则，这些文本和标签将不会被渲染。（因为子模板相当于把内容嵌入到父模板到block中，而没有写到block中的内容当然不会被嵌入，也就不会被渲染。）

**过滤器**

过滤器是通过（|）符号进行使用的，例如：{{ name|length }}：将返回name的长度。

过滤器相当于是一个函数，把当前的变量传入到过滤器中，然后过滤器根据自己的功能，再返回相应的值，之后再将结果渲染到页面中。

Jinja2拥有许多过滤器：（转自：http://www.jianshu.com/p/31a75d3d9270）

abs(value)：返回一个数值的绝对值。示例：-1|abs

default(value,default\_value,boolean=false)：如果当前变量没有值，则会使用参数中的值来代替。示例：name|default('xiaotuo')——如果name不存在，则会使用xiaotuo来替代。boolean=False默认是在只有这个变量为undefined的时候才会使用default中的值，如果想使用python的形式判断是否为false，则可以传递boolean=true。也可以使用or来替换。

escape(value)或e：转义字符，会将<、>等符号转义成HTML中的符号。示例：content|escape或content|e。

first(value)：返回一个序列的第一个元素。示例：names|first

format(value,\*arags,\*\*kwargs)：格式化字符串。

比如：{{ "%s" - "%s"|format('Hello?',"Foo!") }}将输出：Helloo? - Foo!

last(value)：返回一个序列的最后一个元素。示例：names|last。

length(value)：返回一个序列或者字典的长度。示例：names|length。

join(value,d=u'')：将一个序列用d这个参数的值拼接成字符串。

safe(value)：如果开启了全局转义，那么safe过滤器会将变量关掉转义。示例：content\_html|safe。

int(value)：将值转换为int类型。

float(value)：将值转换为float类型。

lower(value)：将字符串转换为小写。

upper(value)：将字符串转换为小写。

replace(value,old,new)： 替换将old替换为new的字符串。

truncate(value,length=255,killwords=False)：截取length长度的字符串。striptags(value)：删除字符串中所有的HTML标签，如果出现多个空格，将替换成一个空格。

trim：截取字符串前面和后面的空白字符。

string(value)：将变量转换成字符串。

wordcount(s)：计算一个长字符串中单词的个数。

**测试器**

测试器主要用来**判断一个值是否满足某种类型，语法是：**if...is...：

{% **if variable is escaped**%}

value of variable: {{ escaped }}

{% else %}

variable is not escaped

{% endif %}

Jinja2中测试器有：

* callable(object)：是否可调用。
* defined(object)：是否已经被定义了。
* escaped(object)：是否已经被转义了。
* upper(object)：是否全是大写。
* lower(object)：是否全是小写。
* string(object)：是否是一个字符串。
* sequence(object)：是否是一个序列。
* number(object)：是否是一个数字。
* odd(object)：是否是奇数。
* even(object)：是否是偶数。

**转义**

在模板渲染字符串的时候，字符串有可能包括一些非常危险的字符比如<、>等，这些字符会破坏掉原来HTML标签的结构，更严重的可能会发生XSS跨域脚本攻击，因此如果碰到<、>这些字符的时候，应该转义成HTML能正确表示这些字符的写法。

对于一些不信任的字符串，可以通过{{ content\_html|e }}或者是{{ content\_html|escape }}的方式进行转义。

如果想关闭自动转义，可以通过{{ content\_html|safe }}的方式关闭自动转义。

{%autoescape true/false%}...{%endautoescape%}可以将**一段代码**块放在中间，来关闭或开启自动转义：

{% autoescape false %}<p>autoescaping is disabled here<p>{{ will\_not\_be\_escaped }}

{% endautoescape %}

**模版页面中引入静态文件**

静态文件主要包括有CSS样式文件、JavaScript脚本文件、图片文件、字体文件等静态资源。

在Jinja中加载静态文件只需要通过url\_for全局函数就可以实现：

<link href="{{ url\_for('static',filename='about.css') }}">