**Flask中Jinja2模板引擎详解(一)–控制语句和表达式**

Flask Python代码：

from flask import Flask,render\_template

app = Flask(\_\_name\_\_)

@app.route('/hello')

@app.route('/hello/<name>')

def hello(name=None):

return render\_template('hello.html', name=name)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', debug=True)

模板代码：

<!doctype html>

<title>Hello Sample</title>

{% if name %}

<h1>Hello {{ name }}!</h1>

{% else %}

<h1>Hello World!</h1>

{% endif %}

我们了解到，模板的表达式都是包含在分隔符{{ }}内的；控制语句都是包含在分隔符{% %}内的；另外，模板也支持注释，都是包含在分隔符{# #}内，支持块注释。

**表达式**

表达式一般有这么几种：

* 最常用的是变量，由Flask渲染模板时传过来，比如上例中的name
* 也可以是任意一种Python基础类型，比如字符串{{ "Hello" }}，用引号括起；或者数值，列表，元祖，字典，布尔值。直接显示基础类型没啥意义，一般配合其他表达式一起用
* 运算。包括算数运算，如{{ 2 + 3 }}；比较运算，如{{ 2 > 1 }}；逻辑运算，如{{ False and True }}
* [过滤器](http://www.bjhee.com/jinja2-filter.html)|和[测试器](http://www.bjhee.com/jinja2-test.html)is。这个在后面会介绍
* 函数调用，如{{ current\_time() }}；数组下标操作，如{{ arr[1] }}
* in操作符，如{{ 1 in [1,2,3] }}
* 字符串连接符~，作用同Python中的+一样，如{{ "Hello " ~ name ~ "!" }}
* if关键字，如{{ 'Hi, %s' % name if name }}。这里的if不是条件控制语句。

有没有觉得，这里的表达式很像Python的语法呀？

**控制语句**

Jinja2的控制语句主要就是条件控制语句if，和循环控制语句for，语法类似于Python。我们可以改动下上节的模板代码：

{% if name and name == 'admin' %}

<h1>This is admin console</h1>

{% elif name %}

<h1>Welcome {{ name }}!</h1>

{% else %}

<h1>Please login</h1>

{% endif %}

上面是一个条件控制语句的例子，注意if控制语句要用{% endif %}来结束。模板中无法像代码中一样靠缩进来判断代码块的结束。再来看个循环的例子，我们先改下Python代码中的”hello”函数，让其传两个列表进模板。

def hello(name=None):

return render\_template('hello-1.html', name=name, digits=[1,2,3,4,5],

users=[{'name':'John'},

{'name':'Tom', 'hidden':True},

{'name':'Lisa'}

{'name':'Bob'}])

然后在模板中加上：

{% for digit in digits %}

{{ digit }}

{% endfor %}

是不是列表被显示出来了？同if语句一样，for控制语句要用{% endfor %}来结束。页面上，每个元素之间会有空格，如果你不希望有空格，就要在for语句的最后，和endfor语句的最前面各加上一个-号。如：

{% for digit in digits -%}

{{ digit }}

{%- endfor %}

现在，你可以看到数字”12345”被一起显示出来了。我们再来看个复杂的循环例子：

<dl>

{% for user in users if not user.hidden %}

{% if loop.first %}

<div>User List:</div>

{% endif %}

<div class="{{ loop.cycle('odd', 'even') }}">

<dt>User No {{ loop.index }}:</dt>

<dd>{{ user.name }}</dd>

</div>

{% if loop.last %}

<div>Total Users: {{ loop.length }}</div>

{% endif %}

{% else %}

<li>No users found</li>

{% endfor %}

</dl>

这里有三个知识点。首先for循环支持else语句，当待遍历的列表”users”为空或者为None时，就进入else语句。

其次，在for语句后使用if关键字，可以对循环中的项作过滤。本例中，所有hidden属性为True的”user”都会被过滤掉。

另外，for循环中可以访问Jinja2的循环内置变量。本例中，我们会在第一项前显示标题，最后一项后显示总数，每一项显示序号。另外，奇偶项的HTML div元素会有不同的class。如果我们加入下面的CSS style，就能看到斑马线。

<style type="text/css">

.odd {

background-color: #BDF;

}

</style>

Jinja2的循环内置变量主要有以下几个：

| **变量** | **内容** |
| --- | --- |
| loop.index | 循环迭代计数（从1开始） |
| loop.index0 | 循环迭代计数（从0开始） |
| loop.revindex | 循环迭代倒序计数（从len开始，到1结束） |
| loop.revindex0 | 循环迭代倒序计数（从len－1开始，到0结束） |
| loop.first | 是否为循环的第一个元素 |
| loop.last | 是否为循环的最后一个元素 |
| loop.length | 循环序列中元素的个数 |
| loop.cycle | 在给定的序列中轮循，如上例在”odd”和”even”两个值间轮循 |
| loop.depth | 当前循环在递归中的层级（从1开始） |
| loop.depth0 | 当前循环在递归中的层级（从0开始） |

关于递归循环，大家看看下面的例子，我就不多介绍了：

{% for item in [[1,2],[3,4,5]] recursive %}

Depth: {{ loop.depth }}

{% if item[0] %}

{{ loop(item) }}

{% else %}

Number: {{ item }} ;

{% endif %}

{% endfor %}

另外，如果你启用了jinja2.ext.loopcontrols扩展的话，你还可以在循环中使用{% break %}和{% continue %}来控制循环执行。关于Jinja2的扩展，我们会在本系列的[第七篇](http://www.bjhee.com/jinja2-i18n.html)和[第八篇](http://www.bjhee.com/jinja2-extension.html)中介绍。

**其他常用语句**

**忽略模板语法**

有时候，我们在页面上就是要显示{{ }}这样的符号怎么办？Jinja2提供了raw语句来忽略所有模板语法。

{% raw %}

<ul>

{% for item in items %}

<li>{{ item }}</li>

{% endfor %}

</ul>

{% endraw %}

**自动转义**

我们将本文一开始的Flask代码hello()方法改动下：

@app.route('/hello')

@app.route('/hello/<name>')

def hello(name=None):

if name is None:

name = '<em>World</em>'

return render\_template('hello.html', name=name)

此时，访问http://localhost:5000/hello，页面上会显示Welcome <em>World</em>!，也就是这个HTML标签<em>被自动转义了。正如我们曾经[提到过的](http://www.bjhee.com/flask-ad3.html)，Flask会对”.html”, “.htm”, “.xml”, “.xhtml”这四种类型的模板文件开启HTML格式自动转义。这样也可以防止HTML语法注入。如果我们不想被转义怎么办？

{% autoescape false %}

<h1>Hello {{ name }}!</h1>

{% endautoescape %}

将autoescape开关设为false即可，反之，设为true即开启自动转义。使用autoescape开关前要启用jinja2.ext.autoescape扩展，在Flask框架中，这个扩展默认已启用。

**赋值**

使用set关键字给变量赋值：

{% set items = [[1,2],[3,4,5]] %}

**with语句**

类似于Python中的with关键字，它可以限制with语句块内对象的作用域：

{% with foo = 1 %}

{% set bar = 2 %}

{{ foo + bar }}

{% endwith %}

{# foo and bar are not visible here #}

使用with关键字前要启用jinja2.ext.with\_扩展，在Flask框架中，这个扩展默认已启用。

**执行表达式**

{% with arr = ['Sunny'] %}

{{ arr.append('Rainy') }}

{{ arr }}

{% endwith %}

看上面这段代码，我们想执行列表的append操作，这时使用{{ arr.append('Rainy') }}页面会输出None，换成{% %}来执行，程序会报错，因为这是个表达式，不是语句。那怎么办？我们可以启用jinja2.ext.do扩展。然后在模板中执行do语句即可：

{% with arr = ['Sunny'] %}

{% do arr.append('Rainy') %}

{{ arr }}

{% endwith %}