<http://www.blogjava.net/jzone/articles/284300.html>

http://freemarker.org/docs/index.html   官方手册

FreeMarker的模板文件并不比HTML页面复杂多少,FreeMarker模板文件主要由如下4个部分组成:  
**1,文本:直接输出的部分  
2,注释:<#-- ... -->格式部分,不会输出  
3,插值:即${...}或#{...}格式的部分,将使用数据模型中的部分替代输出  
4,FTL指令:FreeMarker指定,和HTML标记类似,名字前加#予以区分,不会输出**

下面是一个FreeMarker模板的例子,包含了以上所说的4个部分  
<html><br>  
<head><br>  
<title>Welcome!</title><br>  
</head><br>  
<body><br>  
<#-- 注释部分 --><br>  
<#-- 下面使用插值 -->  
<h1>Welcome ${user} !</h1><br>  
<p>We have these animals:<br>  
<u1><br>  
<#-- 使用FTL指令 -->  
<#list animals as being><br>  
   <li>${being.name} for ${being.price} Euros<br>  
<#list><br>  
<u1><br>  
</body><br>  
</html>

**1, FTL指令规则**

在FreeMarker中,使用FTL标签来使用指令,FreeMarker有3种FTL标签,这和HTML标签是完全类似的.  
1,开始标签:<#directivename parameter>  
2,结束标签:</#directivename>  
3,空标签:<#directivename parameter/>

实际上,使用标签时前面的符号#也可能变成@,如果该指令是一个用户指令而不是系统内建指令时,应将#符号改成@符号.  
使用FTL标签时,应该有正确的嵌套,而不是交叉使用,这和XML标签的用法完全一样.如果全用不存在的指令,FreeMarker不会使用模板输出,而是产生一个错误消息.FreeMarker会忽略FTL标签中的空白字符.值得注意的是< , /> 和指令之间不允许有空白字符.

**2, 插值规则**

FreeMarker的插值有如下两种类型:1,通用插值${expr};2,数字格式化插值:#{expr}或#{expr;format}

**2.1 通用插值**

对于通用插值,又可以分为以下4种情况:  
1,插值结果为字符串值:直接输出表达式结果  
2,插值结果为数字值:根据默认格式(由#setting指令设置)将表达式结果转换成文本输出.可以使用内建的字符串函数格式化单个插值,如下面的例子:  
<#settion number\_format="currency"/>  
<#assign answer=42/>  
${answer}  
${answer?string} <#-- the same as ${answer} -->  
${answer?string.number}  
${answer?string.currency}  
${answer?string.percent}  
${answer}  
输出结果是:  
$42.00  
$42.00  
42  
$42.00  
4,200%  
3,插值结果为日期值:根据默认格式(由#setting指令设置)将表达式结果转换成文本输出.可以使用内建的字符串函数格式化单个插值,如下面的例子:  
${lastUpdated?string("yyyy-MM-dd HH:mm:ss zzzz")}  
${lastUpdated?string("EEE, MMM d, ''yy")}  
${lastUpdated?string("EEEE, MMMM dd, yyyy, hh:mm:ss a '('zzz')'")}  
输出结果是:  
2008-04-08 08:08:08 Pacific Daylight Time  
Tue, Apr 8, '03  
Tuesday, April 08, 2003, 08:08:08 PM (PDT)  
4,插值结果为布尔值:根据默认格式(由#setting指令设置)将表达式结果转换成文本输出.可以使用内建的字符串函数格式化单个插值,如下面的例子:  
<#assign foo=true/>  
${foo?string("yes", "no")}  
输出结果是:  
yes

**2.2 数字格式化插值**

数字格式化插值可采用#{expr;format}形式来格式化数字,其中format可以是:  
mX:小数部分最小X位  
MX:小数部分最大X位  
如下面的例子:  
<#assign x=2.582/>  
<#assign y=4/>  
#{x; M2} <#-- 输出2.58 -->  
#{y; M2} <#-- 输出4 -->  
#{x; m2} <#-- 输出2.6 -->  
#{y; m2} <#-- 输出4.0 -->  
#{x; m1M2} <#-- 输出2.58 -->  
#{x; m1M2} <#-- 输出4.0 -->

**3, 表达式**

表达式是FreeMarker模板的核心功能,表达式放置在插值语法${}之中时,表明需要输出表达式的值;表达式语法也可与FreeMarker标签结合,用于控制输出.实际上FreeMarker的表达式功能非常强大,它不仅支持直接指定值,输出变量值,也支持字符串格式化输出和集合访问等功能.

**3.1 直接指定值**

使用直接指定值语法让FreeMarker直接输出插值中的值,而不是输出变量值.直接指定值可以是字符串,数值,布尔值,集合和MAP对象.

1,字符串  
直接指定字符串值使用单引号或双引号限定,如果字符串值中包含特殊字符需要转义,看下面的例子:  
${"我的文件保存在C:\\盘"}  
${'我名字是\"annlee\"'}  
输出结果是:   
我的文件保存在C:\盘  
我名字是"annlee"

FreeMarker支持如下转义字符:  
\";双引号(u0022)  
\';单引号(u0027)  
\\;反斜杠(u005C)  
\n;换行(u000A)  
\r;回车(u000D)  
\t;Tab(u0009)  
\b;退格键(u0008)  
\f;Form feed(u000C)  
\l;<  
\g;>  
\a;&  
\{;{  
\xCode;直接通过4位的16进制数来指定Unicode码,输出该unicode码对应的字符.

如果某段文本中包含大量的特殊符号,FreeMarker提供了另一种特殊格式:可以在指定字符串内容的引号前增加r标记,在r标记后的文件将会直接输出.看如下代码:  
${r"${foo}"}  
${r"C:\foo\bar"}  
输出结果是:  
${foo}  
C:\foo\bar

2,数值  
表达式中的数值直接输出,不需要引号.小数点使用"."分隔,不能使用分组","符号.FreeMarker目前还不支持科学计数法,所以"1E3"是错误的.在FreeMarker表达式中使用数值需要注意以下几点:  
1,数值不能省略小数点前面的0,所以".5"是错误的写法  
2,数值8 , +8 , 8.00都是相同的

3,布尔值  
直接使用true和false,不使用引号.

4,集合  
集合以方括号包括,各集合元素之间以英文逗号","分隔,看如下的例子:  
<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as x>  
${x}  
</#list>  
输出结果是:  
星期一  
星期二  
星期三  
星期四  
星期五  
星期六  
星期天

除此之外,集合元素也可以是表达式,例子如下:  
[2 + 2, [1, 2, 3, 4], "whatnot"]

还可以使用数字范围定义数字集合,如2..5等同于[2, 3, 4, 5],但是更有效率.注意,使用数字范围来定义集合时无需使用方括号,数字范围也支持反递增的数字范围,如5..2

5,Map对象  
Map对象使用花括号包括,Map中的key-value对之间以英文冒号":"分隔,多组key-value对之间以英文逗号","分隔.下面是一个例子:  
{"语文":78, "数学":80}  
Map对象的key和value都是表达式,但是key必须是字符串

**3.2 输出变量值**

FreeMarker的表达式输出变量时,这些变量可以是顶层变量,也可以是Map对象中的变量,还可以是集合中的变量,并可以使用点(.)语法来访问Java对象的属性.下面分别讨论这些情况

1,顶层变量  
所谓顶层变量就是直接放在数据模型中的值,例如有如下数据模型:  
Map root = new HashMap();   //创建数据模型  
root.put("name","annlee");   //name是一个顶层变量

对于顶层变量,直接使用${variableName}来输出变量值,变量名只能是字母,数字,下划线,$,@和#的组合,且不能以数字开头号.为了输出上面的name的值,可以使用如下语法:  
${name}

2,输出集合元素  
如果需要输出集合元素,则可以根据集合元素的索引来输出集合元素,集合元素的索引以方括号指定.假设有索引:  
["星期一","星期二","星期三","星期四","星期五","星期六","星期天"].该索引名为week,如果需要输出星期三,则可以使用如下语法:  
${week[2]}   //输出第三个集合元素

此外,FreeMarker还支持返回集合的子集合,如果需要返回集合的子集合,则可以使用如下语法:  
week[3..5]   //返回week集合的子集合,子集合中的元素是week集合中的第4-6个元素

3,输出Map元素  
这里的Map对象可以是直接HashMap的实例,甚至包括JavaBean实例,对于JavaBean实例而言,我们一样可以把其当成属性为key,属性值为value的Map实例.为了输出Map元素的值,可以使用点语法或方括号语法.假如有下面的数据模型:  
Map root = new HashMap();  
Book book = new Book();  
Author author = new Author();  
author.setName("annlee");  
author.setAddress("gz");  
book.setName("struts2");  
book.setAuthor(author);  
root.put("info","struts");  
root.put("book", book);

为了访问数据模型中名为struts2的书的作者的名字,可以使用如下语法:  
book.author.name    //全部使用点语法  
book["author"].name  
book.author["name"]    //混合使用点语法和方括号语法  
book["author"]["name"]   //全部使用方括号语法

使用点语法时,变量名字有顶层变量一样的限制,但方括号语法没有该限制,因为名字可以是任意表达式的结果.

**3.3, 字符串操作**

FreeMarker的表达式对字符串操作非常灵活,可以将字符串常量和变量连接起来,也可以返回字符串的子串等.

字符串连接有两种语法:  
1,使用${..}或#{..}在字符串常量部分插入表达式的值,从而完成字符串连接.  
2,直接使用连接运算符+来连接字符串

例如有如下数据模型:  
Map root = new HashMap(); root.put("user","annlee");  
下面将user变量和常量连接起来:  
${"hello, ${user}!"}   //使用第一种语法来连接  
${"hello, " + user + "!"} //使用+号来连接  
上面的输出字符串都是hello,annlee!,可以看出这两种语法的效果完全一样.

值得注意的是,${..}只能用于文本部分,不能用于表达式,下面的代码是错误的:  
<#if ${isBig}>Wow!</#if>  
<#if "${isBig}">Wow!</#if>  
应该写成:<#if isBig>Wow!</#if>

截取子串可以根据字符串的索引来进行,截取子串时如果只指定了一个索引值,则用于取得字符串中指定索引所对应的字符;如果指定两个索引值,则返回两个索引中间的字符串子串.假如有如下数据模型:  
Map root = new HashMap(); root.put("book","struts2,freemarker");  
可以通过如下语法来截取子串:  
${book[0]}${book[4]}   //结果是su  
${book[1..4]}     //结果是tru

**3.4 集合连接运算符**

这里所说的集合运算符是将两个集合连接成一个新的集合,连接集合的运算符是+,看如下的例子:  
<#list ["星期一","星期二","星期三"] + ["星期四","星期五","星期六","星期天"] as x>  
${x}  
</#list>  
输出结果是:星期一 星期二 星期三 星期四 星期五 星期六 星期天

**3.5 Map连接运算符**

Map对象的连接运算符也是将两个Map对象连接成一个新的Map对象,Map对象的连接运算符是+,如果两个Map对象具有相同的key,则右边的值替代左边的值.看如下的例子:  
<#assign scores = {"语文":86,"数学":78} + {"数学":87,"Java":93}>  
语文成绩是${scores.语文}  
数学成绩是${scores.数学}  
Java成绩是${scores.Java}  
输出结果是:  
语文成绩是86  
数学成绩是87  
Java成绩是93

**3.6 算术运算符**

FreeMarker表达式中完全支持算术运算,FreeMarker支持的算术运算符包括:+, - , \* , / , % 看如下的代码:  
<#assign x=5>  
${ x \* x - 100 }  
${ x /2 }  
${ 12 %10 }  
输出结果是:  
-75   2.5   2

在表达式中使用算术运算符时要注意以下几点:  
1,运算符两边的运算数字必须是数字  
2,使用+运算符时,如果一边是数字,一边是字符串,就会自动将数字转换为字符串再连接,如:${3 + "5"},结果是:35

使用内建的int函数可对数值取整,如:  
<#assign x=5>  
${ (x/2)?int }  
${ 1.1?int }  
${ 1.999?int }  
${ -1.1?int }  
${ -1.999?int }  
结果是:2 1 1 -1 -1

**3.7 比较运算符**

表达式中支持的比较运算符有如下几个:  
1,=或者==:判断两个值是否相等.  
2,!=:判断两个值是否不等.  
3,>或者gt:判断左边值是否大于右边值  
4,>=或者gte:判断左边值是否大于等于右边值  
5,<或者lt:判断左边值是否小于右边值  
6,<=或者lte:判断左边值是否小于等于右边值

注意:=和!=可以用于字符串,数值和日期来比较是否相等,但=和!=两边必须是相同类型的值,否则会产生错误,而且FreeMarker是精确比较,"x","x ","X"是不等的.其它的运行符可以作用于数字和日期,但不能作用于字符串,大部分的时候,使用gt等字母运算符代替>会有更好的效果,因为FreeMarker会把>解释成FTL标签的结束字符,当然,也可以使用括号来避免这种情况,如:<#if (x>y)>

**3.8 逻辑运算符**

逻辑运算符有如下几个:  
逻辑与:&&  
逻辑或:||  
逻辑非:!  
逻辑运算符只能作用于布尔值,否则将产生错误

**3.9 内建函数**

FreeMarker还提供了一些内建函数来转换输出,可以在任何变量后紧跟?,?后紧跟内建函数,就可以通过内建函数来轮换输出变量.下面是常用的内建的字符串函数:  
html:对字符串进行HTML编码  
cap\_first:使字符串第一个字母大写  
lower\_case:将字符串转换成小写  
upper\_case:将字符串转换成大写  
trim:去掉字符串前后的空白字符

下面是集合的常用内建函数  
size:获取序列中元素的个数

下面是数字值的常用内建函数  
int:取得数字的整数部分,结果带符号

例如:  
<#assign test="Tom & Jerry">  
${test?html}  
${test?upper\_case?html}  
结果是:Tom &amp; Jerry   TOM &amp; JERRY

**3.10 空值处理运算符**

FreeMarker对空值的处理非常严格,FreeMarker的变量必须有值,没有被赋值的变量就会抛出异常,因为FreeMarker未赋值的变量强制出错可以杜绝很多潜在的错误,如缺失潜在的变量命名,或者其他变量错误.这里所说的空值,实际上也包括那些并不存在的变量,对于一个Java的null值而言,我们认为这个变量是存在的,只是它的值为null,但对于FreeMarker模板而言,它无法理解null值,null值和不存在的变量完全相同.

为了处理缺失变量,FreeMarker提供了两个运算符:  
!:指定缺失变量的默认值  
??:判断某个变量是否存在

其中,!运算符的用法有如下两种:  
variable!或variable!defaultValue,第一种用法不给缺失的变量指定默认值,表明默认值是空字符串,长度为0的集合,或者长度为0的Map对象.

使用!指定默认值时,并不要求默认值的类型和变量类型相同.使用??运算符非常简单,它总是返回一个布尔值,用法为:variable??,如果该变量存在,返回true,否则返回false

**3.11 运算符的优先级**

FreeMarker中的运算符优先级如下(由高到低排列):  
1,一元运算符:!  
2,内建函数:?  
3,乘除法:\*, / , %  
4,加减法:- , +  
5,比较:> , < , >= , <= (lt , lte , gt , gte)  
6,相等:== , = , !=  
7,逻辑与:&&  
8,逻辑或:||  
9,数字范围:..

实际上,我们在开发过程中应该使用括号来严格区分,这样的可读性好,出错少

**4 FreeMarker的常用指令**

FreeMarker的FTL指令也是模板的重要组成部分,这些指令可实现对数据模型所包含数据的抚今迭代,分支控制.除此之外,还有一些重要的功能,也是通过FTL指令来实现的.

**4.1 if指令**

这是一个典型的分支控制指令,该指令的作用完全类似于Java语言中的if,if指令的语法格式如下:  
<#if condition>...  
<#elseif condition>...  
<#elseif condition>...  
<#else> ...  
</#if>

例子如下:  
<#assign age=23>  
<#if (age>60)>老年人  
<#elseif (age>40)>中年人  
<#elseif (age>20)>青年人  
<#else> 少年人  
</#if>  
输出结果是:青年人  
上面的代码中的逻辑表达式用括号括起来主要是因为里面有>符号,由于FreeMarker会将>符号当成标签的结束字符,可能导致程序出错,为了避免这种情况,我们应该在凡是出现这些符号的地方都使用括号.

**4.2 switch , case , default , break指令**

这些指令显然是分支指令,作用类似于Java的switch语句,switch指令的语法结构如下:  
<#switch value>  
<#case refValue>...<#break>  
<#case refValue>...<#break>  
<#default>...  
</#switch>

**4.3 list, break指令**

list指令是一个迭代输出指令,用于迭代输出数据模型中的集合,list指令的语法格式如下:  
<#list sequence as item>  
...  
</#list>  
上面的语法格式中,sequence就是一个集合对象,也可以是一个表达式,但该表达式将返回一个集合对象,而item是一个任意的名字,就是被迭代输出的集合元素.此外,迭代集合对象时,还包含两个特殊的循环变量:  
item\_index:当前变量的索引值  
item\_has\_next:是否存在下一个对象  
也可以使用<#break>指令跳出迭代

例子如下:  
<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as x>  
${x\_index + 1}.${x}<#if x\_has\_next>,</if>  
<#if x="星期四"><#break></#if>  
</#list>

**4.4 include指令**

include指令的作用类似于JSP的包含指令,用于包含指定页.include指令的语法格式如下:  
<#include filename [options]>  
在上面的语法格式中,两个参数的解释如下:  
filename:该参数指定被包含的模板文件  
options:该参数可以省略,指定包含时的选项,包含encoding和parse两个选项,其中encoding指定包含页面时所用的解码集,而parse指定被包含文件是否作为FTL文件来解析,如果省略了parse选项值,则该选项默认是true.

**4.5 import指令**

该指令用于导入FreeMarker模板中的所有变量,并将该变量放置在指定的Map对象中,import指令的语法格式如下:  
<#import "/lib/common.ftl" as com>  
上面的代码将导入/lib/common.ftl模板文件中的所有变量,交将这些变量放置在一个名为com的Map对象中.

**4.6 noparse指令**

noparse指令指定FreeMarker不处理该指定里包含的内容,该指令的语法格式如下:  
<#noparse>...</#noparse>

看如下的例子:  
<#noparse>  
<#list books as book>  
   <tr><td>${book.name}<td>作者:${book.author}  
</#list>  
</#noparse>  
输出如下:  
<#list books as book>  
   <tr><td>${book.name}<td>作者:${book.author}  
</#list>

**4.7 escape , noescape指令**

escape指令导致body区的插值都会被自动加上escape表达式,但不会影响字符串内的插值,只会影响到body内出现的插值,使用escape指令的语法格式如下:  
<#escape identifier as expression>...  
<#noescape>...</#noescape>  
</#escape>

看如下的代码:  
<#escape x as x?html>  
First name:${firstName}  
Last name:${lastName}  
Maiden name:${maidenName}  
</#escape>  
上面的代码等同于:  
First name:${firstName?html}  
Last name:${lastName?html}  
Maiden name:${maidenName?html}

escape指令在解析模板时起作用而不是在运行时起作用,除此之外,escape指令也嵌套使用,子escape继承父escape的规则,如下例子:  
<#escape x as x?html>  
Customer Name:${customerName}  
Items to ship;  
<#escape x as itemCodeToNameMap[x]>  
   ${itemCode1}  
   ${itemCode2}  
   ${itemCode3}  
   ${itemCode4}  
</#escape>  
</#escape>  
上面的代码类似于:  
Customer Name:${customerName?html}  
Items to ship;  
${itemCodeToNameMap[itemCode1]?html}  
${itemCodeToNameMap[itemCode2]?html}  
${itemCodeToNameMap[itemCode3]?html}  
${itemCodeToNameMap[itemCode4]?html}

对于放在escape指令中所有的插值而言,这此插值将被自动加上escape表达式,如果需要指定escape指令中某些插值无需添加escape表达式,则应该使用noescape指令,放在noescape指令中的插值将不会添加escape表达式.

**4.8 assign指令**

assign指令在前面已经使用了多次,它用于为该模板页面创建或替换一个顶层变量,assign指令的用法有多种,包含创建或替换一个顶层变量,或者创建或替换多个变量等,它的最简单的语法如下:<#assign name=value [in namespacehash]>,这个用法用于指定一个名为name的变量,该变量的值为value,此外,FreeMarker允许在使用assign指令里增加in子句,in子句用于将创建的name变量放入namespacehash命名空间中.

assign指令还有如下用法:<#assign name1=value1 name2=value2 ... nameN=valueN [in namespacehash]>,这个语法可以同时创建或替换多个顶层变量,此外,还有一种复杂的用法,如果需要创建或替换的变量值是一个复杂的表达式,则可以使用如下语法格式:<#assign name [in namespacehash]>capture this</#assign>,在这个语法中,是指将assign指令的内容赋值给name变量.如下例子:  
<#assign x>  
<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as n>  
${n}  
</#list>  
</#assign>  
${x}  
上面的代码将产生如下输出:星期一 星期二 星期三 星期四 星期五 星期六 星期天

虽然assign指定了这种复杂变量值的用法,但是我们也不要滥用这种用法,如下例子:<#assign x>Hello ${user}!</#assign>,以上代码改为如下写法更合适:<#assign x="Hello ${user}!">

**4.9 setting指令**

该指令用于设置FreeMarker的运行环境,该指令的语法格式如下:<#setting name=value>,在这个格式中,name的取值范围包含如下几个:  
locale:该选项指定该模板所用的国家/语言选项  
number\_format:指定格式化输出数字的格式  
boolean\_format:指定两个布尔值的语法格式,默认值是true,false  
date\_format,time\_format,datetime\_format:指定格式化输出日期的格式  
time\_zone:设置格式化输出日期时所使用的时区

**4.10 macro , nested , return指令**

macro可以用于实现自定义指令,通过使用自定义指令,可以将一段模板片段定义成一个用户指令,使用macro指令的语法格式如下:  
<#macro name param1 param2 ... paramN>  
...  
<#nested loopvar1, loopvar2, ..., loopvarN>  
...  
<#return>  
...  
</#macro>  
在上面的格式片段中,包含了如下几个部分:  
name:name属性指定的是该自定义指令的名字,使用自定义指令时可以传入多个参数  
paramX:该属性就是指定使用自定义指令时报参数,使用该自定义指令时,必须为这些参数传入值  
nested指令:nested标签输出使用自定义指令时的中间部分  
nested指令中的循环变量:这此循环变量将由macro定义部分指定,传给使用标签的模板  
return指令:该指令可用于随时结束该自定义指令.

看如下的例子:  
<#macro book>   //定义一个自定义指令  
j2ee  
</#macro>  
<@book />    //使用刚才定义的指令  
上面的代码输出结果为:j2ee

在上面的代码中,可能很难看出自定义标签的用处,因为我们定义的book指令所包含的内容非常简单,实际上,自定义标签可包含非常多的内容,从而可以实现更好的代码复用.此外,还可以在定义自定义指令时,为自定义指令指定参数,看如下代码:  
<#macro book booklist>     //定义一个自定义指令booklist是参数  
<#list booklist as book>  
   ${book}  
</#list>  
</#macro>  
<@book booklist=["spring","j2ee"] />   //使用刚刚定义的指令  
上面的代码为book指令传入了一个参数值,上面的代码的输出结果为:spring j2ee

不仅如此,还可以在自定义指令时使用nested指令来输出自定义指令的中间部分,看如下例子:  
<#macro page title>  
<html>  
<head>  
   <title>FreeMarker示例页面 - ${title?html}</title>  
</head>  
<body>  
   <h1>${title?html}</h1>  
   <#nested>      //用于引入用户自定义指令的标签体  
</body>  
</html>  
</#macro>  
上面的代码将一个HTML页面模板定义成一个page指令,则可以在其他页面中如此page指令:  
<#import "/common.ftl" as com>     //假设上面的模板页面名为common.ftl,导入页面  
<@com.page title="book list">  
<u1>  
<li>spring</li>  
<li>j2ee</li>  
</ul>  
</@com.page>

从上面的例子可以看出,使用macro和nested指令可以非常容易地实现页面装饰效果,此外,还可以在使用nested指令时,指定一个或多个循环变量,看如下代码:  
<#macro book>  
<#nested 1>      //使用book指令时指定了一个循环变量值  
<#nested 2>  
</#macro>  
<@book ;x> ${x} .图书</@book>  
当使用nested指令传入变量值时,在使用该自定义指令时,就需要使用一个占位符(如book指令后的;x).上面的代码输出文本如下:  
1 .图书    2 .图书

在nested指令中使用循环变量时,可以使用多个循环变量,看如下代码:  
<#macro repeat count>  
<#list 1..count as x>     //使用nested指令时指定了三个循环变量  
   <#nested x, x/2, x==count>  
</#list>  
</#macro>  
<@repeat count=4 ; c halfc last>  
${c}. ${halfc}<#if last> Last! </#if>  
</@repeat>  
上面的输出结果为:  
1. 0.5   2. 1   3. 1.5   4. 2 Last;

return指令用于结束macro指令,一旦在macro指令中执行了return指令,则FreeMarker不会继续处理macro指令里的内容,看如下代码:  
<#macro book>  
spring  
<#return>  
j2ee  
</#macro>  
<@book />  
上面的代码输出:spring,而j2ee位于return指令之后,不会输出.