<http://dbajun.iteye.com/blog/2066579>

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}   
${book.name?if\_exists }  //用于判断如果存在,就输出这个值   
${book.name?default(‘xxx’)}//默认值xxx   
${book.name!"xxx"}//默认值xxx   
${book.date?string('yyyy-MM-dd')} //日期格式   
${book?string.number}  20 //三种不同的数字格式   
${book?string.currency}--<#-- $20.00 -->   
${book?string.percent}—<#-- 20% -->   
  
<#assign foo=ture />   //声明变量,插入布尔值进行显示   
${foo?string("yes","no")} <#-- yes -->   
  
<等大小比较符号使用需要注意:(xml的原因),可以用于比较数字和日期   
使用lt、lte、gt和gte来替代<、<=、>和>= 也可以使用括号<#if (x>y)>   
  
内置函数: 调用区别于属性的访问,使用?代替.   
常见的一些内置函数   
对于字符串   
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   
trim－去掉字符串前后的空白字符   
  
对于Sequences(序列)   
size－获得序列中元素的数目   
  
对于数字   
int－取得数字的整数部分（如-1.9?int的结果是-1）   
  
对于集合,可以使用数组的方式,使用下标索引进行访问   
  
逻辑判断:   
if................   
  
<#if condition>...   
<#elseif condition2>...   
<#elseif condition3>......   
<#else>...   
Boolean类型的空值判断   
空值判断可以写成<#if book.name?? >   //注意${}为变量的渲染显示,而<>为定义等操作符的定义   
  
switch............   
<#switch value>   
<#case refValue1>   
    ...   
    <#break>   
<#case refValue2>   
    ...   
    <#break>   
...   
<#case refValueN>   
    ...   
    <#break>   
<#default>   
    ...   
</#switch>   
  
快速定义int区间的集合   
<#assign l=0..100/> //注意不需要[]   
  
3:循环读取集合:  注意/的使用   
<#list student as stu>   
    ${stu}<br/>   
</#list>   
与jstl循环类似,也可以访问循环的状态   
item\_index:当前变量的索引值   
item\_has\_next:是否存在下一个对象 其中item名称为as后的变量名,如stu   
  
集合长度判断   
<#if student?size != 0></#if>  判断=的时候,注意只要一个=符号,而不是==   
  
宏/模板   
初步了解: 使用更像一个闭包closure,可以定义后,在脚本中任意地方引用,并原地起作用   
<#macro greet>   
<font size="+2">Hello Joe!</font>   
</#macro>   
使用的方式为:   
<@greet></@greet>  //同xml可以简写成<@greet/>   
  
宏的参数定义,类似js,在宏名后 带参数进行传递定义   
<#macro greet person color>   
${person}   
</#macro>   
  
调用带参数时,注意使用类似XML的属性格式进行传递,不需要关心顺序问题   
<@greet person="Fred" color="black"/>   
  
参数默认值定义,如果没有,就必须要求传递完整的参数列表   
<#macro greet person color="black">   
<font size="+2" color="${color}">Hello ${person}!</font>   
</#macro>   
  
使用xml的嵌套内容进行传递宏调用,关键标签 <#nested>   
<#macro border>   
<table border=4 cellspacing=0 cellpadding=4><tr><td>   
    <#nested>   
</tr></td></table>   
</#macro>   
  
调用时:   
<@border>The bordered text</@border>   
  
<#nested> 标签可以在宏中多次调用,也可以将多个宏组合进行嵌套   
  
for循环的精简版:   
<#list 1..count as x>   
</#list>   
  
宏的循环变量,配合嵌套标签进行参数传递,   
<#macro repeat count>   
<#list 1..count as x>   
    <#nested x, x/2, x==count>  //这里的三个参数,将会传递到嵌套内容中   
</#list>   
</#macro>   
  
<@repeat count=4 ; c, halfc, last>   
${c}. ${halfc}<#if last> Last!</#if> //这里的内容由macro中的<#nested>进行参数的传递,传递的数量任意,当注意需要宏接受这些   
</@repeat>   
上述还需要注意;的使用   
  
参数的数量是可变的,并不要求全部都有,但是效果不同   
  
在模板中定义变量   
在模板中定义的变量有三种类型：   
plain变量：可以在模板的任何地方访问，包括使用include指令插入的模板，使用assign指令创建和替换。   
局部变量：在宏定义体中有效，使用local指令创建和替换。   
循环变量：只能存在于指令的嵌套内容，由指令（如list）自动创建；宏的参数是局部变量，而不是循环变量   
  
<#assign x = "plain"> //全局的plain变量   
内部循环变量将会隐藏同名的外部循环变量   
  
外部导入的使用,可以用于模块化,并且提供公用性   
如:lib/my\_lib.ftl文件   
<#macro copyright date>   
<p>Copyright (C) ${date} Julia Smith. All rights reserved.   
<br>Email: ${mail}</p>   
</#macro>   
<#assign mail = "jsmith@acme.com">   
  
lib/my\_inc.ftl文件   
<#import "/lib/my\_test.ftl" as my>   
<#assign mail="fred@acme.com">   
<@my.copyright date="1999-2002"/>   
${my.mail}   
${mail}   
输出结果将不会出现冲突   
  
对于库中的变量修改,使用in关键字   
<#assign mail="jsmith@other.com" in my>   
  
函数定义:区别于宏对象,带返回值   
<#function name param1 param2><#return val></#function>函数，有返回参数   
  
stringA[M .. N] 取子字符串，类似substring(stringA, M, N)   
  
<#include "/copyright\_footer.html"> 导入其他页面元素   
<#include filename options>   
options包含两个属性   
encoding=”GBK” 编码格式   
parse=true 是否作为ftl语法解析,默认是true，false就是以文本方式引入.注意在ftl文件里布尔值都是直接赋值的如parse=true,而不是   
  
parse=”true”   
  
hash与list的定义   
<#assign c= {"a":"orz","b":"czs"}>   
${c.a}   
  
List片段可以采用： products[10..19] or products[5..] 的格式进行定义,当只局限于数字   
<#assign c= [1,2,3,4,5,6,6,7]>   
    <#list c[1..3] as v>   
    ${v}   
    </#list>   
  
对变量的缺省处理   
product.color!"red"   
  
用compress directive或者transform来处理输出。   
<#compress>...</#compress>：消除空白行。   
<@compress single\_line=true>...</@compress>将输出压缩为一行。都需要包裹所需文档   
  
freemarker可用"["代替"<".在模板的文件开头加上[#ftl].   
  
注释部分   
<#-- 注释部分 -->   
  
数字输出的另外一种方式   
#{c.a;m0} 区别于${},这个例子是用于输出数字的格式化,保留小数的位数,详细如下   
  
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   
  
在定义字符串的时候,可以使用''或者"",对特殊字符,需要使用\进行转义   
  
如果存在大量特殊字符,可以使用${r"..."}进行过滤   
${r"${foo}"}   
${r"C:\foo\bar"}   
  
Map对象的key和value都是表达式,但是key必须是字符串   
可以混合使用.和[""]访问   
book.author["name"] //混合使用点语法和方括号语法   
  
为了处理缺失变量,FreeMarker提供了两个运算符: 用于防止对象不存在而导致的异常   
!:指定缺失变量的默认值   
??:判断某个变量是否存在,返回boolean值   
  
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>   
  
${firstName?html} 使用html对字符进行格式化处理,对于<等的过滤   
  
escape , noescape指令,对body内的内容实用统一的表达式   
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}   
  
定义全局变量的方式   
<#assign name1=value1 name2=value2 / > // 可以同时定义多个变量,也可以使用循环来给变量赋值   
<#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}   
  
setting指令,用于动态设置freeMarker的运行环境:   
  
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   
  
<#return> 用于退出宏的运行   
  
?html 用于将字符串中可能包含的html字符,进行过滤.   
  
调用Java方法,需要使用实现TemplateMethodModel接口,但是好像会覆盖掉属性的访问