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 ${user} !</h1><br>
<p>We have these animals:<br>
<ul><br>
<#-- 使用 FTL 指令 -->
<#list animals as being><br>
    <li>${being.name} for ${being.price} Euros<br>
```

<#list>

</body>

</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 & Jerry　　TOM & 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">

spring

j2ee

</@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.

if, else, elseif

switch, case, default, break

list, break

include

Import

compress

escape, noescape

assign

global

setting

macro, nested, return

t, lt, rt

3 一些常用方法或注意事项

表达式转换类

数字循环

对浮点取整数

给变量默认值

判断对象是不是 null

常用格式化日期

添加全局共享变量数据模型

直接调用 java 对象的方法

字符串处理(内置方法)

在模板里对 sequences 和 hashes 初始化

注释标志

sequences 内置方法

hashes 内置方法

4 freemarker 在 web 开发中注意事项

web 中常用的几个对象

view 中值的搜索顺序

在模板里 ftl 里使用标签

如何初始化共享变量

与 webwork 整合配置

5 高级方法

自定义方法

自定义 Transforms

1 概念

最常用的 3 个概念

sequence　序列，对应 java 里的 list、数组等非键值对的集合

hash　　键值对的集合

namespace 对一个 ftl 文件的引用,利用这个名字可以访问到该 ftl 文件的资源

2 指令

if, else, elseif

语法

<#if condition>

...

<#elseif condition2>

...

<#elseif condition3>

...

...

<#else>

...

</#if>

用例

<#if x = 1>

x is 1

</#if>

<#if x = 1>

x is 1

<#else>

x is not 1

</#if>

switch, case, default, break

语法

<#switch value>

```
    <#case refValue1>
        ...
        <#break>
    <#case refValue2>
        ...
        <#break>
    ...
    <#case refValueN>
        ...
        <#break>
    <#default>
        ...
</#switch>
```

用例

字符串

```
<#switch being.size>
    <#case "small">
          This will be processed if it is small
          <#break>
    <#case "medium">
          This will be processed if it is medium
          <#break>
```

```
    <#case "large">
          This will be processed if it is large
          <#break>
    <#default>
          This will be processed if it is neither
</#switch>
```

数字

```
<#switch x>
    <#case x = 1>
        1
    <#case x = 2>
        2
    <#default>
        d
</#switch>
```

如果 x=1 输出 1 2, x=2 输出 2, x=3 输出 d

list, break

语法

```
<#list sequence as item>
...
<#if item = "spring"><#break></#if>
...
```

</#list>

关键字

item_index:是 list 当前值的下标

item_has_next:判断 list 是否还有值

用例

<#assign seq = ["winter", "spring", "summer", "autumn"]>

<#list seq as x>

  ${x_index + 1}. ${x}<#if x_has_next>,</#if>

</#list>

输出

1. winter,
2. spring,
3. summer,
4. autumn

include

语法

<#include filename>

or

<#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”

用例

/common/copyright.ftl 包含内容

Copyright 2001-2002 ${me}

All rights reserved.

模板文件

<#assign me = "Juila Smith">

<h1>Some test</h1>

<p>Yeah.

<hr>

<#include "/common/copyright.ftl" encoding=”GBK”>

输出结果

<h1>Some test</h1>

<p>Yeah.

<hr>

Copyright 2001-2002 Juila Smith

All rights reserved.

Import

语法

<#import path as hash>

类似于 java 里的 import,它导入文件，然后就可以在当前文件里使用被导入文件里的宏组件

用例

假设 mylib.ftl 里定义了宏 copyright 那么我们在其他模板页面里可以这样使用

<#import "/libs/mylib.ftl" as my>

<@my.copyright date="1999-2002"/>

"my"在 freemarker 里被称作 namespace

compress

语法

<#compress>

  ...

</#compress>

用来压缩空白空间和空白的行

用例

<#assign x = "   moo   \n\n   ">

(<#compress>

  1 2 3   4   5

  ${moo}

  test only

  I said, test only

</#compress>)

输出

(1 2 3 4 5

moo

test only

I said, test only)

escape, noescape

语法

<#escape identifier as expression>

...

<#noescape>...</#noescape>

...

</#escape>

用例

主要使用在相似的字符串变量输出，比如某一个模块的所有字符串输出都必须是 html 安全的，这个时候就可以使用该表达式

<#escape x as x?html>

First name: ${firstName}

<#noescape>Last name: ${lastName}</#noescape>

Maiden name: ${maidenName}

</#escape>

相同表达式

```
   First name: ${firstName?html}
   Last name: ${lastName }
   Maiden name: ${maidenName?html}
```

assign

语法

```
<#assign name=value>
```

or

```
<#assign name1=value1 name2=value2 ... nameN=valueN>
```

or

```
<#assign same as above... in namespacehash>
```

or

```
<#assign name>
   capture this
</#assign>
```

or

```
<#assign name in namespacehash>
   capture this
</#assign>
```

用例

生成变量,并且给变量赋值

给 seasons 赋予序列值

```
<#assign seasons = ["winter", "spring", "summer", "autumn"]>
```

给变量 test 加 1

<#assign test = test + 1>

给 my namespage 赋予一个变量 bgColor,下面可以通过 my.bgColor 来访问这个变量

<#import "/mylib.ftl" as my>

<#assign bgColor="red" in my>

将一段输出的文本作为变量保存在 x 里

下面的阴影部分输出的文本将被赋值给 x

<#assign x>

  <#list 1..3 as n>

    ${n} <@myMacro />

  </#list>

</#assign>

Number of words: ${x?word_list?size}

${x}

<#assign x>Hello ${user}!</#assign> error

<#assign x=" Hello ${user}!" > true

同时也支持中文赋值，如：

<#assign 语法>

  java

</#assign>

${语法}

打印输出:

java

global

语法

<#global name=value>

or

<#global name1=value1 name2=value2 ... nameN=valueN>

or

<#global name>

   capture this

</#global>

全局赋值语法，利用这个语法给变量赋值，那么这个变量在所有的 namespace 中是可见的,如果这个变量被当前的 assign 语法覆盖 如 <#global x=2><#assign x=1> 在当前页面里 x=2 将被隐藏，或者通过${.global.x}来访问

setting

语法

<#setting name=value>

用来设置整个系统的一个环境

locale

number_format

boolean_format

date_format, time_format, datetime_format

time_zone

classic_compatible

用例

假如当前是匈牙利的设置，然后修改成美国

${1.2}

<#setting locale="en_US">

${1.2}

输出

1,2

1.2

因为匈牙利是采用“,”作为十进制的分隔符，美国是用“.”

macro, nested, return

语法

```
<#macro name param1 param2 ... paramN>
   ...
   <#nested loopvar1, loopvar2, ..., loopvarN>
   ...
   <#return>
   ...
</#macro>
```

用例

```
<#macro test foo bar="Bar" baaz=-1>
   Test text, and the params: ${foo}, ${bar}, ${baaz}
</#macro>
<@test foo="a" bar="b" baaz=5*5-2/>
<@test foo="a" bar="b"/>
<@test foo="a" baaz=5*5-2/>
<@test foo="a"/>
```

输出

```
   Test text, and the params: a, b, 23
   Test text, and the params: a, b, -1
   Test text, and the params: a, Bar, 23
   Test text, and the params: a, Bar, -1
```

定义循环输出的宏

```
<#macro list title items>
   <p>${title?cap_first}:
   <ul>
       <#list items as x>
           <li>${x?cap_first}
       </#list>
   </ul>
</#macro>
```

<@list items=["mouse", "elephant", "python"] title="Animals"/>

输出结果

<p>Animals:

Mouse

Elephant

Python

包含 body 的宏

<#macro repeat count>

<#list 1..count as x>

<#nested x, x/2, x==count>

</#list>

</#macro>

<@repeat count=4 ; c halfc last>

${c}. ${halfc}<#if last> Last!</#if>

</@repeat >

输出

1. 0.5

2. 1

3. 1.5

4. 2 Last!

t, lt, rt

语法

<#t> 去掉左右空白和回车换行

<#lt>去掉左边空白和回车换行

<#rt>去掉右边空白和回车换行

<#nt>取消上面的效果

3 一些常用方法或注意事项

表达式转换类

${expression}计算 expression 并输出

#{ expression }数字计算#{ expression ;format}安格式输出数字

format 为 M 和 m

M 表示小数点后最多的位数,m 表示小数点后最少的位数如

#{121.2322;m2M2}输出 121.23

数字循环

1..5 表示从 1 到 5，原型 number..number

对浮点取整数

${123.23?int} 输出 123

给变量默认值

${var?default(“hello world
”)?html}如果 var is null 那么将会被 hello world
替代

判断对象是不是 null

```
        <#if mouse?exists>
            Mouse found
<#else>
```

也可以直接${mouse?if_exists})输出布尔形

常用格式化日期

openingTime 必须是 Date 型,详细查看 freemarker 文档 Reference->build-in referece->build-in for date

${openingTime?date}

${openingTime?date_time}

${openingTime?time}

添加全局共享变量数据模型

在代码里的实现

```
        cfg = Configuration.getDefaultConfiguration();
cfg.setSharedVariable("global", "you good");
```

页面实现可以通过 global 指令,具体查看指令里的 global 部分

直接调用 java 对象的方法

${object.methed(args)}

字符串处理(内置方法)

html 安全输出

“abc<table>sdfsf”?html

返回安全的 html 输出,替换掉 html 代码

xml 安全输出

var?xml

substring 的用法

<#assign user=”hello jeen”>

${user[0]}${user[4]}

${user[1..4]}

输出 :

ho

ello

类似 String.split 的用法

“abc;def;ghi”?split(“;”)返回 sequence

将字符串按空格转化成 sequence,然后取 sequence 的长度

var?word_list 效果同 var?split(“ ”)

var?word_list?size

取得字符串长度

var?length

大写输出字符

var?upper_case

小写输出字符

var?lower_case

首字符大写

var?cap_first

首字符小写

var?uncap_first

去掉字符串前后空格

var?trim

每个单词的首字符大写

var?capitalize

类似 String.indexof:

“babcdabcd”?index_of(“abc”) 返回 1

“babcdabcd”?index_of(“abc”,2) 返回 5

类似 String.lastIndexOf

last_index_of 和 String.lastIndexOf 类似,同上

下面两个可能在代码生成的时候使用（在引号前加”\”）

j_string: 在字符串引号前加”\”

<#assign beanName = 'The "foo" bean.'>

String BEAN_NAME = "${beanName?j_string}";

打印输出:

String BEAN_NAME = "The \"foo\" bean.";

js_string:

```
  <#assign user = "Big Joe's \"right hand\"."> 
<script>
   alert("Welcome ${user}!");
</script>
```

打印输出

```
  alert("Welcome Big Joe\'s \"right hand\"!");
```

替换字符串 replace

${s?replace(‘ba’, ‘XY’)}

${s?replace(‘ba’, ‘XY’ , ‘规则参数’)}将 s 里的所有的 ba 替换成 xy 规则参数包含: i r m s c f 具体含义如下:

- i: 大小写不区分.
- f: 只替换第一个出现被替换字符串的字符串
- r: XY 是正则表达式
- m: Multi-line mode for regular expressions. In multi-line mode the expressions ^ and $ match just after or just before, respectively, a line terminator or the end of the string. By default these expressions only match at the beginning and the end of the entire string.
- s: Enables dotall mode for regular expressions (same as Perl singe-line mode). In dotall mode, the expression . matches any character, including a line terminator. By default this

expression does not match line terminators.

- c: Permits whitespace and comments in regular expressions.

在模板里对 sequences 和 hashes 初始化

sequences

1. [“you”,”me”,”he”]

2. 1..100

3. [{“Akey”:”Avalue”},{“Akey1”:”Avalue1”},

{“Bkey”:”Bvalue”},{“Bkey1”:”Bvalue1”},

]

hashes {“you”:”a”,”me”:”b”,”he”:”c”}

注释标志

<#--

这里是注释

-->

旧版本的 freemarker 采用的是<#comment> 注释 </#comment>方法

sequences 内置方法

sequence?first

返回 sequence 的第一个值;前提条件 sequence 不能是 null

sequence?last

返回 sequence 最后一个值

sequence?reverse

反转 sequence 的值

sequence?size

返回 sequence 的大小

sequence?sort

对 sequence 按里面的对象 toString()的结果进行排序

sequence?sort_by(value)

对 sequence 按里面的对象的属性 value 进行排序

如: sequence 里面放入的是 10 个 user 对象，user 对象里面包含 name,age 等属性

sequence?sort_by(name) 表示所有的 user 按 user.name 进行排序

hashes 内置方法

hash?keys

返回 hash 里的所有 keys, 返回结果类型 sequence

hash?values

返回 hash 里的所有 value, 返回结果类型 sequence

4 freemarker 在 web 开发中注意事项

freemarker 与 webwork 整合

web 中常用的几个对象

Freemarker 的 ftl 文件中直接使用内部对象:

${Request ["a"]}

${RequestParameters["a"]}

${Session ["a"]}

${Application ["a"]}

${JspTaglibs ["a"]}

与 webwork 整合之后 通过配置的 servlet 已经把 request,session 等对象置入了数据模型中

在 view 中存在下面的对象

我们可以在 ftl 中${req}来打印 req 对象

- req - the current HttpServletRequest
- res - the current HttpServletResponse
- stack - the current OgnlValueStack
- ognl - the OgnlTool instance
- webwork - an instance of FreemarkerWebWorkUtil
- action - the current WebWork action
- exception - optional the Exception instance, if the view is a JSP exception or Servlet exception view

view 中值的搜索顺序

${name}将会以下面的顺序查找 name 值

- freemarker variables
- value stack
- request attributes
- session attributes

- servlet context attributes

在模板里 ftl 里使用标签

注意，如果标签的属性值是数字，那么必须采用 nubmer=123 方式给属性赋值

JSP 页面

```
<%@page  contentType="text/html;charset=ISO-8859-2"
language="java"%>
<%@taglib  uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@taglib  uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<html>
   <body>
        <h1><bean:message key="welcome.title"/></h1>
        <html:errors/>
        <html:form action="/query">
            Keyword: <html:text property="keyword"/><br>
            Exclude: <html:text property="exclude"/><br>
            <html:submit value="Send"/>
        </html:form>
   </body>
</html>
```

模板 ftl 页面

```
<#assign html=JspTaglibs["/WEB-INF/struts-html.tld"]>
<#assign bean=JspTaglibs["/WEB-INF/struts-bean.tld"]>
<html>
   <body>
       <h1><@bean.message key="welcome.title"/></h1>
       <@html.errors/>
       <@html.form action="/query">
           Keyword: <@html.text property="keyword"/><br>
           Exclude: <@html.text property="exclude"/><br>
           <@html.submit value="Send"/>
       </@html.form  >
   </body>
</html>
```

如何初始化共享变量

1. 初始化全局共享数据模型

freemark 在 web 上使用的时候对共享数据的初始化支持的不够,不能在配置初始化的时候实现，而必须通过 ftl 文件来初始化全局变量。这是不能满主需求的，我们需要在 servlet init 的时候留出一个接口来初始化系统的共享数据

具体到和 webwork 整合,因为本身 webwork 提供了整合 servlet,如果要增加全局共享变量，可以通过修改

com.opensymphony.webwork.views.freemarker.FreemarkerServlet 来实现,我们可以在这个 servlet 初始化的时候来初始化全局共享变量

与 webwork 整合配置

配置 web.xml

```
<servlet>
        <servlet-name>freemarker</servlet-name>
        <servlet-class>com.opensymphony.webwork.views.freemarker.FreemarkerServlet</servlet-class>
        <init-param>
            <param-name>TemplatePath</param-name>
<param-value>/</param-value>
<!—模板载入文件夹，这里相对 context root，递归获取该文件夹下的所有模板-->
        </init-param>
        <init-param>
            <param-name>NoCache</param-name> <!—是否对模板缓存-->
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>ContentType</param-name>
```

```
            <param-value>text/html</param-value>
        </init-param>
        <init-param>
<param-name>template_update_delay</param-name>
<!—模板更新时间,0 表示每次都更新,这个适合开发时候-->
            <param-value>0</param-value>
        </init-param>
        <init-param>
            <param-name>default_encoding</param-name>
            <param-value>GBK</param-value>
        </init-param>
        <init-param>
            <param-name>number_format</param-name>
            <param-value>0.##########</param-value><!—数字
显示格式-->
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>freemarker</servlet-name>
        <url-pattern>*.ftl</url-pattern>
    </servlet-mapping>
```

5 高级方法

自定义方法

${timer("yyyy-MM-dd H:mm:ss", x)}

${timer("yyyy-MM-dd ", x)}

在模板中除了可以通过对象来调用方法外(${object.methed(args)})

也可以直接调用 java 实现的方法，java 类必须实现接口

TemplateMethodModel 的方法 exec(List args). 下面以把毫秒的时间转换成按格式输出的时间为例子

```
public class LongToDate implements TemplateMethodModel {


public TemplateModel exec(List args) throws
TemplateModelException {
SimpleDateFormat mydate = new SimpleDateFormat((String)
args.get(0)));
                return mydate.format(new
Date(Long.parseLong((String)args.get(1)));
        }
}
```

将 LongToDate 对象放入到数据模型中

root.put("timer", new IndexOfMethod());

ftl 模板里使用

<#assign x = "123112455445">

${timer("yyyy-MM-dd H:mm:ss", x)}

${timer("yyyy-MM-dd ", x)}

输出

2001-10-12 5:21:12

2001-10-12

自定义 Transforms

实现自定义的<@transform>文本或表达式</@transform >的功能,允许对中间的最终文本进行解析转换

例子：实现<@upcase>str</@upcase > 将 str 转换成 STR 的功能

代码如下：

```
import java.io.*;
import java.util.*;
import freemarker.template.TemplateTransformModel;
class UpperCaseTransform implements TemplateTransformModel {
        public Writer getWriter(Writer out, Map args) {
                return new UpperCaseWriter(out);
        }
        private class UpperCaseWriter extends Writer {

                private Writer out;

                UpperCaseWriter (Writer out) {
```

```
                this.out = out;
            }
            public void write(char[] cbuf, int off, int len)
                        throws IOException {
                out.write(new String(cbuf, off,
len).toUpperCase());
            }
            public void flush() throws IOException {
                out.flush();
            }
            public void close() {
            }
        }
}
```

然后将此对象 put 到数据模型中

```
root.put("upcase", new UpperCaseTransform());
```

在 view(ftl)页面中可以如下方式使用

```
<@upcase>
hello world
</@upcase  >
```

打印输出:

```
HELLO WORLD
```