freemaker 的基本语法

freemaker 的基本语法:

<# ... > 中存放所有 freemaker 的内容，之外的内容全部原样输出。

<@ ... /> 是函数调用

两个定界符内的内容中，第一个符号表示指令或者函数名，其后的跟随参数。

freemaker 提供的控制包括如下：

<#if condition><#elseif condition><#else></#if> 条件判断

<#list hash_or_seq as var></#list> 遍历 hash 表或者 collection

（freemaker 称作 sequence）的成员

<#macro name param1 param2 ... ><#nested param></#macro> 宏，无返回参数

<#function name param1 param2><#return val></#function> 函数，有返回参数

var?member_function(...) 用函数对 var 进行转换，freemaker 称为

build-ins。实际内部实现类似 member_function(var, ...)

stringA[M .. N] 取子字符串，类似 substring(stringA, M, N)

{key:value, key2:value2 ...} 直接定义一个 hash 表

[item0, item1, item2 ...] 直接定义一个序列

hash0[key0] 存取 hash 表中 key 对应的元素

seq0[5] 存取序列指定下标的元素

<@function1 param0 param1 ... /> 调用函数 function1

<@macro0 param0 param1 ; nest_param0 nest_param1 ...>

nest_body < /@macro> 调用宏，并处理宏的嵌套

<#assign var = value > 定义变量并初始化

<#local var = value> 在 macro 或者 function 中定义局部变量并初始化

<#global var = value > 定义全局变量并初始化

${var} 输出并替换为表达式的值

<#visit xmlnode> 调用 macro 匹配 xmlnode 本身及其子节点

<#recurse xmlnode> 调用 macro 匹配 xmlnode 的子节点

FreeMaker 一篇通[【转】

FreeMaker 一篇通[【转】
2007-08-09 19:38
FreeMaker 一篇通

前言

Freemaker 是一个强大的模板引擎，相比 velocity 而言，其强大的过程调用、递归和闭包回调功能让 freemaker 可以完成几乎所有我们所想的功能。从个人看法而言，freemaker 完全有能力作为 MDA 的代码辅助生成工具。
本文试图越过传统的概念性介绍，通过一组例子直接把读者带入到 Freemaker 应用的较高层阶。

正文

大家看文章标题就应该知道，我想用一篇文章，把大家从对 freemaker 的陌生直接带入到比较深入的境界，所以不想说一些基础性的东西，如果大家不习惯我的表达方法，大可通过 google 去找习惯于自己阅读方式的相关文章。

我用过 velocity，最近才用 freemaker，才知道我以前的选择是错了，因为 velocity 不支持过程的调用，所以我为 velocity 增加了很多的东西，写了很多代码，而且脚本也累赘得要命。freemaker 首先吸引我的是它强大的过程调用和递归处理能力，其次则是 xml 风格的语法结构有着明显的边界，不象 velocity 要注意段落之间要留空格。所以我建议大家直接使用 Freemaker，虽然 freemaker 没有.net 版本，我想不嵌入程序中使用的话，freemaker 是绝对的首选。（题外话，谁有兴趣移植一个 NFreeMaker？）

在使用之前我们先要设置运行环境，在使用 Freemaker 的时候，我们需要下载相关的程序：

freemaker: http://freemarker.sourceforge.net/
fmpp: http://fmpp.sourceforge.net/

其中 fmpp 是一个 freemaker 的辅助工具，有了它，我们可以实现更多的功能。以下例子必须 fmpp 辅助。

这里我们首先提出问题。大家看如下的一个 xml 文件，虽然 freemaker 的能力不仅在于处理 xml 文件，但是用 xml 作为例子更直观一些：

```
<?xml version='1.0' encoding="gb2312" ?>
<types xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:DruleForm-Lite.xsd">
        <type name="Type1" >
            <labels>
                <label lang="zh-CN" value="投保单"/>
            </labels>
            <field name="Field11" type="Float" lbound="1" ubound="1" >
                <labels>
                    <label lang="zh-CN" value="投保单 ID"/>
                </labels>
            </field>
            <field name="Field12" type="String" lbound="1"
ubound="*"/>
            <field name="Field13" type="Integer" lbound="1"
ubound="*"/>
            <field name="Field14" type="Type2" lbound="1" ubound="*">
                <type name="Type2">
                    <field name="Field21" type="String" lbound="1"
ubound="*"/>
                    <field name="Field22" type="Integer" lbound="1"
ubound="*"/>
                </type>
            </field>
            <field name="Field15" type="InsuranceProduct" lbound="1"
ubound="*"/>
        <type>
        <type name="Type3">
            <field name="Field31" type="Type1" lbound="1" ubound="*"
/>
        </type>
    </types>
```

[代码 1]

我们的任务是把这个文件转化为相应的 C#代码。大家先看转换模板的代码：

```
1<#ftl ns_prefixes={"ns": "urn:DruleForm-Lite.xsd"}>
2<#-- 定义 xml namespace，以便在以下代码中使用,注意，ftl 指令必须使用单独的行 -->
3<@pp.setOutputEncoding encoding="gb2312" /> <#-- 使用 fmpp 提供的函数来设置输出编码 -->
4
5<#recurse doc> <#-- 根入口，代码 1 部分的 xml 存放在变量 doc 中，doc 变量的填充由 fmpp 根据 config.fmpp 中的配置进行 -->
6
7<#macro "ns:types"> <#-- xslt 风格的匹配处理入口 -->
8<#recurse> <#-- 直接进行 types 节点内的匹配 -->
9</#macro>
10
11<#macro "ns:type"> <#-- 匹配 type 节点 -->
12 class ${.node.@name} <#-- 其中.node 是保留字，表示当前节点，引用的 @name 是 xslt 风格 -->
13 {
14   <#recurse> <#-- 继续匹配 -->
15 }
16</#macro>
17
18<#macro "ns:field">
19   public ${.node.@type} ${.node.@name};
20</#macro>
21
22<#macro @element> <#-- 所有没有定义匹配的节点到这里处理 -->
23</#macro>
24
25
```

[代码 2]

我们使用的配置文件设置如下：

```
sourceRoot: src
outputRoot: out
logFile: log.fmpp
modes: [
copy(common/**/*.*, resource/*.*)
     execute(*.ftl)
ignore(templates/*.*, .project, **/*.xml, xml/*.*, *.js)
]
removeExtensions: ftl
```

```
sourceEncoding: gb2312
data: {
doc: xml(freemaker.xml)
}
```

[代码 3]

然后我们在 dos 模式下运行指令：
E:\work\blogs\freemaker>f:\download\freemaker\fmpp\bin\fmpp

最后的输出结果是这样的，存放在文件 out\freemaker.中：

```
    class Type1
    {
        public Float Field11;
        public String Field12;
        public Integer Field13;
        public Type2 Field14;
        public Float Field15;
    }

    class Type3
    {
        public Type1 Field31;
    }
```

[代码 4]

先来解释一下 freemaker 的基本语法了，
<# ... > 中存放所有 freemaker 的内容，之外的内容全部原样输出。
<@ ... /> 是函数调用
两个定界符内的内容中，第一个符号表示指令或者函数名，其后的跟随参数。
freemaker 提供的控制包括如下：
<#if condition><#elseif condition><#else></#if> 条件判断
<#list hash_or_seq as var></#list> 遍历 hash 表或者 collection(freemaker 称作 sequence) 的成员
<#macro name param1 param2 ... ><#nested param></#macro> 宏，无返回参数
<#function name param1 param2><#return val></#function>函数，有返回参数
var?member_function(...) 用函数对 var 进行转换,freemaker 称为 build-ins。实际内部实现类似 member_function(var, ...)
stringA[M .. N] 取子字符串，类似 substring(stringA, M, N)
{key:value, key2:value2 ...} 直接定义一个 hash 表

[item0, item1, item2 ...] 直接定义一个序列
hash0[key0] 存取 hash 表中 key 对应的元素
seq0[5] 存取序列指定下标的元素
<@function1 param0 param1 ... /> 调用函数 function1
<@macro0 param0 param1 ; nest_param0 nest_param1 ...> nest_body </@macro> 调用宏，并处理宏的嵌套
<#assign var = value > 定义变量并初始化
<#local var = value> 在 macro 或者 function 中定义局部变量并初始化
<#global var = value > 定义全局变量并初始化
${var} 输出并替换为表达式的值
<#visit xmlnode> 调用 macro 匹配 xmlnode 本身及其子节点
<#recurse xmlnode> 调用 macro 匹配 xmlnode 的子节点

[表 1]

大家仔细对比 xml 文件，发现少了什么吗？对了，少了一个 Type2 定义，我们把代码 2 中的 ns:type 匹配（第 11 行）修改一下：

```
<#macro "ns:field">
   public ${.node.@type} ${.node.@name};
   <#recurse > <#-- 深入处理子节点 -->
</#macro>
```

[代码 5]

结果输出文件中的内容就变为如下：

```
class Type1
{
    public Float Field11;
    public String Field12;
    public Integer Field13;
    public Type2 Field14;
    class Type2
    {
        public String Field21;
        public Integer Field22;
    }
    public Float Field15;
}

class Type3
```

```
    {
        public Type1 Field31;
    }
```

[代码 6]

如果各位有意向把 Type2 提到跟 Type1 和 Type3 同一级别的位置，那么我们要继续修改代码了。把代码 2 的 <#recurse doc>行（第 5 行）修改成如下：

```
<#assign inner_types=pp.newWritableHash()> <#-- 调用 fmpp 功能函数，生成一个可写的 hash -->
<#recurse doc> <#-- 根入口，代码 1 部分的 xml 存放在变量 doc 中，doc 变量的填充由 fmpp 根据 config.fmpp 中的配置进行 -->
<#if inner_types?size gt 0 > <#-- 如果存放有类型 -->
<#list inner_types?values as node> <#-- 遍历哈西表的值 -->
   <#visit node> <#-- 激活相应的 macro 处理，类似于 xslt 的 apply-template。大家把 visit 改成 recurse 看一下不同的效果 -->
</#list>
</#if>
```

[代码 7]

同时把 macro ns:field（第 18 行）修改成如下：

```
<#macro "ns:field">
   public ${.node.@type} ${.node.@name};
   <#if .node["ns:type"]?has_content > <#-- 如果当前节点下存在 type 节点 -->
    <#local t = .node["ns:type"] >
    <@pp.set hash=inner_types key="${t.@name}" value=t /> <#-- 哈西表中增加内容，key 为嵌套类型的 name 属性，value 为该类型节点 -->
   </#if>
</#macro>
```

[代码 8]

运行得到输出文件类似这样：

```
    class Type1
    {
        public Float Field11;
        public String Field12;
        public Integer Field13;
        public Type2 Field14;
        public Float Field15;
```

```
    }

    class Type3
    {
        public Type1 Field31;
    }

    class Type2
    {
        public String Field21;
        public Integer Field22;
    }
```

[代码 9]

大家比较一下，看看我们修改的地方出现了哪些效果？然后记得大家要做另外 2 件事情，
1。把第一行修改成为<#ftl ns_prefixes={"D": "urn:DruleForm-Lite.xsd"}> ，然后把所有的 <#macro "ns:type"> 修改成<#macro type>，把所有的.node["ns:type"]修改成 .node.type，看看能不能运行？是不是觉得简单方便些了？记住，第一行的那个 D 表示是 default namespace 的意思哦。
2。在第二行插入<#compress>，在最后一行添加</#compress>。再运行一下看看结果有什么不同？

一个例子下来，大家基本对 freemaker 有了一些感觉了，为了纠正大家认为 freemaker 就是一个 xml 处理工具的误解，我们再来做一个简单的实验。这次我们要做的是一个正常的编程题目，做一个 100 以内的 Fibonacci 数列的程序。程序如下：

```
迭代次数:
<#list 1 .. 10 as n>
${n} = ${fibo(n)}
</#list>

<#compress>
<#function fibo n>
<#if n lte 1>
   <#return 1>
<#elseif n = 2>
   <#return 1>
<#else>
   <#return fibo(n-1) + fibo(n-2)>
</#if>
</#function>
```

</#compress>

[代码 10]

这个例子里边有一些问题需要注意，大家看我的 #if n lte 1 这一行，为什么我这么写？因为常规的大于小于号和 xml 的节点有冲突，为了避免问题，所以用 gt(>) gte(>=) lt(<) lte(<=) 来代表。

另外，复杂的字符串处理如何来做？就留作家庭作业吧，大家记得取 substr 的方法是 str[first .. last] 就可以了。如下的例子可能会给你一点提示：

```
<#assign str = "hello!$world!">
<#assign mid = (str?length + 1)/2-1 >
<#list mid .. 0 as cnt>
${str[(mid - cnt) .. (mid + cnt)]?left_pad(mid*2)}
</#list>
```

[代码 11]

最后，说一下非常有用的 macro 的 nested 指令，没有它，也许 freemaker 会失去大部分的魅力。我个人认为这也是 freemaker 全面超越 velocity 的地方。大家先看一下代码：

```
<#assign msg = "hello">
<@macro0 ; index >
${msg} ${index}
</@macro0>

<#macro macro0>
<#list 0 .. 10 as number>
   <#nested number>
</#list>
</#macro>
```

[代码 12]

这段代码的作用就是一个闭包（closure）。我们用 java 的匿名类实现相同的功能就是这样：

```
interface ICallback
{
public void call(int index);
```

```
}

void Main()
{
String msg = "hello";
macro0(
   new ICallback()
   {
    public void call(int index)
    {
     System.out.println(msg + index.toString());
    }
   }
);
}

void macro0(ICallback callback)
{
for(int i = 0; i < 10; ++i)
{
   callback.call(i);
}
}
```

freemaker 学习笔记--设计指导

<# ... > 中存放所有 freemaker 的内容，之外的内容全部原样输出。

<@ ... /> 是函数调用

两个定界符内的内容中，第一个符号表示指令或者函数名，其后的跟随参数。

freemaker 提供的控制包括如下：

<#if condition><#elseif condition><#else></#if> 条件判断

<#list hash_or_seq as var></#list> 遍历 hash 表或者 collection（freemaker 称作 sequence）的成员

<#macro name param1 param2 ... ><#nested param></#macro> 宏，无返回参数

<#function name param1 param2><#return val></#function>函数，有返回参数

var?member_function(...) 用函数对 var 进行转换，freemaker 称为 build-ins。实际内部实现类似 member_function(var, ...)

stringA[M .. N] 取子字符串，类似 substring(stringA, M, N)

{key:value, key2:value2 ...} 直接定义一个 hash 表

[item0, item1, item2 ...] 直接定义一个序列

hash0[key0] 存取 hash 表中 key 对应的元素

seq0[5] 存取序列指定下标的元素

<@function1 param0 param1 ... /> 调用函数 function1

<@macro0 param0 param1 ; nest_param0 nest_param1 ...>
nest_body < /@macro > 调用宏，并处理宏的嵌套

<#assign var = value > 定义变量并初始化

<#local var = value> 在 macro 或者 function 中定义局部变量并初始化

<#global var = value > 定义全局变量并初始化

${var} 输出并替换为表达式的值

<#visit xmlnode> 调用 macro 匹配 xmlnode 本身及其子节点

<#recurse xmlnode> 调用 macro 匹配 xmlnode 的子节点

<#if *condition* > </#if>

<#list *SequenceVar* as *variable* > *repeatThis* </#list>

<#include "/copyright_footer.html">

一个 ftl 标记不能放在另外一个 ftl 标记里面，但是注释标记能够放在 ftl 标

记里面。

系统预定义指令采用<#...></#>

用户自定义指令采用<@...></@>

hash 片段可以采用： products[10..19] or products[5..] 的格式。

序列也可以做加法计算：passwords + {"joe":"secret42"}

缺省值：name!"unknown" 或者 (user.name)!"unknown" 或者 name! 或者 (user.name)!

null 值检查：name?? or (user.name)??

转义列表：

Escape sequence	Meaning
\"	Quotation mark (u0022)
\'	Apostrophe (a.k.a. apostrophe-quote) (u0027)
\\	Back slash (u005C)
\n	Line feed (u000A)
\r	Carriage return (u000D)
\t	Horizontal tabulation (a.k.a. tab) (u0009)
\b	Backspace (u0008)
\f	Form feed (u000C)
\l	Less-than sign: <
\g	Greater-than sign: >
\a	Ampersand: &
\{	Curly bracket: {

Escape sequence	Meaning
\x*Code*	Character given with its hexadecimal Unicode code (UCS code)

如果想打印${，则需要将{转义，可以写成"$\{user}"，或者可以用生字符（r 指令）：$(r "${xx}"}

序列构成：<#list **["winter", "spring", "summer", "autumn"]** as x>${x}</#list>

不同的对象可以存放在一个序列里面，比如：[2 + 2, [1, 2, 3, 4], "whatnot"]. 第一个是数字，第二个是序列，第三个是字符串。

可用采用 start..end 的方式来定义一个数字序列，start 可以小于 end，同时，end 也可以省略。

hash 取值支持一下四种模式：book.author.name, book[″author″].name, book.author.[″name″], book[″author″][″name″].

特殊变量是指 freemaker 引擎本身定义的变量。访问时，以.variable_name 的语法访问。

变量表达式支持嵌套模式，比如：${"Hello ${user}!"}。

变量表达式在指令中的使用情况：
变量表达式可以在指令中，用“”的方式存在，不如：<#include "/footer/${company}.html">.
但是不允许下面的方式存在： <#if ${isBig}>Wow!</#if>, 正确写法是：<#if

isBig>Wow!</#if>.

而且 <#if "${isBig}">Wow!</#if>写法也不正确，因为"${isBig}"返回的是字符串，不是 boolean 类型。

字符串中取字符或字符串采用以下语法：${user[0]},${user[0..2]}

${user[4..]},${user?string(4)}

序列操作：

加法：<#list ["Joe", "Fred"] + ["Julia", "Kate"] as user>

但要注意串联之后的读取速度变慢。

子序列：seq[1..4]

序列和 hash 的串联都只能用于两个相加，不能有多个相加的模式，hash 相加，如果两个相加的 hash 存在相同的 key，则后面会覆盖前面的。

在使用>=或者>时，需要注意一些问题，因为 freemaker 会将>解释成标记的关闭符，为了解决这个问题，需要在表达式加上括号，比如：<#if (x > y)>. 或者使用 > and &lt 符号来代替。

无值变量（包括无该变量，null，返回 void，无属性等）：

unsafe_expr!*default_expr* or *unsafe_expr*! or

(*unsafe_expr*)!*default_expr* or (*unsafe_expr*)!

缺省值可以是任何类型，不一定是数字，比如：hits!0 或者 colors!["red", "green", "blue"].

如果缺省值忽略，那么将会默认为空串、空序列或者空 hash，因为 freemarker 支持多类型的值。不过要让默认值为 0 或 false，则不能省略缺省值。

非顶层变量的无值处理：

product.color!"red"：只处理 product 不为空，color 为空的缺省值处理，如果 product 为空，则 freemaker 会抛出异常。

(product.color)!"red"：则会处理 product 为空，color 为空，或者没有 color 属性的无值情况。

无值变量的判断操作：*unsafe_expr*?? or (*unsafe_expr*)??

判断变量是否是无值。

普通变量插入方式：${*expression*},${3+5};

数字变量插入方式：#{*expression*} or #{*expression*; *format*}：过期。

变量只能用于文本区或者是字符串里面，比如：<h1>Hello ${name}!</h1>以及<#include "/footer/${company}.html">

数字值的插入：根据缺省的 number_format 输出，以及可以通过 setting 来达到设置数字格式的目的，也可以通过内置函数 string 来改变输出格式。

日期类型的格式设置：date_format, time_format 和 datetime_format

定义宏：

不带参数：**<#macro 宏名>...</#macro>，引用<@宏名 />**

带参数：**<#macro 宏名 参数...>...</#macro>，引用<@宏名 参数 1=值 1.../>，带有参数的宏，调用是参数的值必须和参数的个数相同。当然也可以在宏定义时给参数一些默认值。比如：**<#macro greet person color="black">

宏里面的嵌套内容：

```
<#macro border>
  <table border=4 cellspacing=0 cellpadding=4><tr><td>
    <#nested>
  </tr></td></table>
</#macro>
```

在宏的定义 body 中加入<#nested>指令。嵌套的内容可以是任何正确的 ftl 块。

宏的本地变量在嵌套内容中是不可见的。

宏定义时，<#nest>指令相当于调用定义的内容，而使用宏时，nest body 相当于定义。

```
<#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>
```

定义变量：

在模板中定义的变量将会隐藏（不是更改）数据模型根下面的同名的变量。

模板中的 3 种类型变量：

1：plain variables，能够在模板中的任何地方访问，一个模板 include 另外一个模板，也可以访问被包含模板的变量。可以通过 assign 或者 macro 指令产生或替换变量。

如果要访问数据模型中的变量，则可以通过.global 来访问：

```
<#assign user = "Joe Hider">
${user}          <#-- prints: Joe Hider -->
${.globals.user} <#-- prints: Big Joe-->
```

2：Local variables，宏定义 body 中用 local 指令创建或者替换。

3：Loop variables:由 list 指令产生。

namespaces:

```
<#import "/lib/my_test.ftl" as my> <#-- the hash called "my" will be the
"gate" -->
<@my.copyright date="1999-2002"/>
${my.mail}
```

设置命名空间里面的变量：<#assign mail="jsmith@other.com" in my>

命名空间与数据模型：命名空间的 ftl 可以访问数据模型的变量。同样命名空间的变量也会隐藏数据模型中同名的变量。

空白问题：

1：White-space stripping，默认为 enabled，清除 ftl 标记带来的空白以及缩进。处理模板的空白。

2：t, rt, lt 指令。

3：ftl 的参数 strip_text.

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。

可替换语法：
freemarker 可用"["代替"<".在模板的文件开头加上[#ftl].

Trackback: http://tb.blog.csdn.net/TrackBack.aspx?PostId=1791781

Spring MVC 使用 Freemarker

- Freemaker 是取代 JSP 的又一种视图技术，和 Velocity 非常类似，但是它比 Velocity 多了一个格式化的功能，因此使用上较 Velocity 方便一点，但语法也稍微复杂一些。

将 Velocity 替换为 Freemarker 只需要改动一些配置文件，同样，在 Spring 中使用 Freemarker 也非常方便，根本无须与 Freemarker 的 API 打交道。我们将 Spring_Velocity 工程复制一份，命名为 Spring_Freemarker，结构如图 7-44 所示。

图 7-44

修改 dispatcher-servlet.xml，将 velocityConfig 删除，修改 viewResolver 为 FreeMarker ViewResolver，并添加一个 freemarkerConfig。

```
<!-- 使用 Freemarker 视图解析器 -->
<bean id="viewResolver" class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver">
    <property name="contentType" value="text/html;charset=UTF-8" />
    <property name="prefix" value="/" />
    <property name="suffix" value=".html" />
</bean>
<!-- 配置 Freemarker -->
<bean id="freemarkerConfig" class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">
    <!-- 视图资源位置 -->
    <property name="templateLoaderPath" value="/" />
    <property name="defaultEncoding" value="UTF-8" />
</bean>
```

模板 test.html 可以稍做修改，加入 Freemarker 内置的格式化功能来定制 Date 类型的输出格式。

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Spring_Freemarker</title>
</head>
<body>
    <h3>Hello, ${name}, it is ${time?string("yyyy-MM-dd HH:mm:ss")}</h3>
</body>
</html>
```

添加 freemarker.jar 到 web/WEB-INF/lib 目录后，启动 Resin，可以看到由 Freemarker 渲染的页面

FreeMarker 入门文章引用自：

1、快速入门

（1）模板 + 数据模型 = 输出

l FreeMarker 基于设计者和程序员是具有不同专业技能的不同个体的观念

l 他们是分工劳动的：设计者专注于表示——创建 HTML 文件、图片、Web 页面的其它可视化方面；程序员创建系统，生成设计页面要显示的数据

l 经常会遇到的问题是：在 Web 页面（或其它类型的文档）中显示的信息在设计页面时是无效的，是基于动态数据的

l 在这里，你可以在 HTML（或其它要输出的文本）中加入一些特定指令，FreeMarker 会在输出页面给最终用户时，用适当的数据替代这些代码

l 下面是一个例子：

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome ${user}!</h1>
  <p>Our latest product:
  <a href="${latestProduct.url}">${latestProduct.name}</a>!
</body>
</html>
```

l 这个例子是在简单的 HTML 中加入了一些由**${…}**包围的特定代码，这些特定代码是 FreeMarker 的指令，而包含 FreeMarker 的指令的文件就称为模板（Template）

l 至于 user、latestProduct.url 和 latestProduct.name 来自于数据模型(data model)

l 数据模型由程序员编程来创建，向模板提供变化的信息，这些信息来自于数据库、文件，甚至于在程序中直接生成

l 模板设计者不关心数据从那儿来，只知道使用已经建立的数据模型

l 下面是一个可能的数据模型：

```
(root)
  |
  +- user = "Big Joe"
  |
  +- latestProduct
      |
      +- url = "products/greenmouse.html"
```

```
      |
      +- name = "green mouse"
```

l 数据模型类似于计算机的文件系统，latestProduct 可以看作是目录，而 user、url 和 name 看作是文件，url 和 name 文件位于 latestProduct 目录中（这只是一个比喻，实际并不存在）

l 当 FreeMarker 将上面的数据模型合并到模板中，就创建了下面的输出：

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome Big Joe!</h1>
  <p>Our latest product:
  <a href="products/greenmouse.html">green mouse</a>!
</body>
</html>
```

（2）数据模型

l 典型的数据模型是树型结构，可以任意复杂和深层次，如下面的例子：

```
(root)
  |
  +- animals
  |   |
  |   +- mouse
  |   |   |
  |   |   +- size = "small"
  |   |   |
  |   |   +- price = 50
  |   |
  |   +- elephant
  |   |   |
  |   |   +- size = "large"
  |   |   |
  |   |   +- price = 5000
  |   |
  |   +- python
  |       |
  |       +- size = "medium"
  |       |
  |       +- price = 4999
  |
  +- test = "It is a test"
  |
  +- whatnot
      |
```

```
      +- because = "don't know"
```

l 类似于目录的变量称为 **hashes**，包含保存下级变量的唯一的查询名字

l 类似于文件的变量称为 **scalars**，保存单值

l **scalars** 保存的值有两种类型：字符串（用引号括起，可以是单引号或双引号）和数字（不要用引号将数字括起，这会作为字符串处理）

l 对 **scalars** 的访问从 root 开始，各部分用“.”分隔，如 animals.mouse.price

l 另外一种变量是 **sequences**，和 **hashes** 类似，只是不使用变量名字，而使用数字索引，如下面的例子：

```
(root)
  |
  +- animals
  |   |
  |   +- (1st)
  |   |   |
  |   |   +- name = "mouse"
  |   |   |
  |   |   +- size = "small"
  |   |   |
  |   |   +- price = 50
  |   |
  |   +- (2nd)
  |   |   |
  |   |   +- name = "elephant"
  |   |   |
  |   |   +- size = "large"
  |   |   |
  |   |   +- price = 5000
  |   |
  |   +- (3rd)
  |       |
  |       +- name = "python"
  |       |
  |       +- size = "medium"
  |       |
  |       +- price = 4999
  |
  +- whatnot
      |
      +- fruits
          |
          +- (1st) = "orange"
          |
          +- (2nd) = "banana"
```

l 这种对 **scalars** 的访问使用索引，如 animals[0].name

（3）模板

l 在 FreeMarker 模板中可以包括下面三种特定部分：

? ${…}：称为 interpolations，FreeMarker 会在输出时用实际值进行替代

? FTL 标记（FreeMarker 模板语言标记）：类似于 HTML 标记，为了与 HTML 标记区分，用#开始（有些以@开始，在后面叙述）

? 注释：包含在<#--和-->（而不是<!--和-->）之间

l 下面是一些使用指令的例子：

? if 指令

```
<#if animals.python.price < animals.elephant.price>
  Pythons are cheaper than elephants today.
<#else>
  Pythons are not cheaper than elephants today.
</#if>
```

? list 指令

```
<p>We have these animals:
<table border=1>
  <tr><th>Name<th>Price
  <#list animals as being>
  <tr><td>${being.name}<td>${being.price} Euros
  </#list>
</table>
```

输出为：

```
<p>We have these animals:
<table border=1>
  <tr><th>Name<th>Price
  <tr><td>mouse<td>50 Euros
  <tr><td>elephant<td>5000 Euros
  <tr><td>python<td>4999 Euros
</table>
```

? include 指令

```
<html>
<head>
  <title>Test page</title>
</head>
<body>
  <h1>Test page</h1>
  <p>Blah blah...
<#include "/copyright_footer.html">
</body>
</html>
```

? 一起使用指令

```
<p>We have these animals:
<table border=1>
  <tr><th>Name<th>Price
```

```
  <#list animals as being>
  <tr>
    <td>
      <#if being.size = "large"><b></#if>
      ${being.name}
      <#if being.size = "large"></b></#if>
    <td>${being.price} Euros
  </#list>
</table>
```

FreeMarker 设计指南

快速入门

（1）模板 + 数据模型 = 输出
FreeMarker 基于设计者和程序员是具有不同专业技能的不同个体的观念他们是分工劳动的：
设计者专注于表示——创建 HTML 文件、图片、Web 页面的其它可视化方面；
程序员创建系统，生成设计页面要显示的数据。
经常会遇到的问题是：在 Web 页面（或其它类型的文档）中显示的信息在设计页面时是无效的，是基于动态数据的。在这里，你可以在 HTML（或其它要输出的文本）中加入一些特定指令，FreeMarker 会在输出页面给最终用户时，用适当的数据替代这些代码。

先来解释一下 freemaker 的基本语法了，
<# ... > 中存放所有 freemaker 的内容，之外的内容全部原样输出。
<@ ... /> 是函数调用
两个定界符内的内容中，第一个符号表示指令或者函数名，其后的跟随参数。
freemaker 提供的控制包括如下：
<#if condition><#elseif condition><#else> 条件判断
<#list hash_or_seq as var> 遍历 hash 表或者 collection（freemaker 称作 sequence）的成员
<#macro name param1 param2 ... ><#nested param> 宏，无返回参数
<#function name param1 param2><#return val>函数，有返回参数
var?member_function(...) 用函数对 var 进行转换，freemaker 称为 build-ins。实际内部实现类似 member_function(var, ...)
stringA[M .. N] 取子字符串，类似 substring(stringA, M, N)
{key:value, key2:value2 ...} 直接定义一个 hash 表
[item0, item1, item2 ...] 直接定义一个序列
hash0[key0] 存取 hash 表中 key 对应的元素
seq0[5] 存取序列指定下标的元素

<@function1 param0 param1 ... /> 调用函数 function1
<@macro0 param0 param1 ; nest_param0 nest_param1 ...> nest_body </@macro>
调用宏，并处理宏的嵌套
<#assign var = value > 定义变量并初始化
<#local var = value> 在 macro 或者 function 中定义局部变量并初始化
<#global var = value > 定义全局变量并初始化
${var} 输出并替换为表达式的值
<#visit xmlnode> 调用 macro 匹配 xmlnode 本身及其子节点
<#recurse xmlnode> 调用 macro 匹配 xmlnode 的子节点

下面是一个例子：

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome ${user}!</h1>
  <p>Our latest product:
  <a href="${latestProduct.url}">${latestProduct.name}</a>!
</body>
</html>
```

这个例子是在简单的 HTML 中加入了一些由${…}包围的特定代码，这些特定代码是FreeMarker 的指令，而包含 FreeMarker 的指令的文件就称为模板(Template)。至于 user、latestProduct.url 和 latestProduct.name 来自于数据模型（data model）。
数据模型由程序员编程来创建，向模板提供变化的信息，这些信息来自于数据库、文件，甚至于在程序中直接生成。
模板设计者不关心数据从那儿来，只知道使用已经建立的数据模型。
下面是一个可能的数据模型：

```
(root)
  |
  +- user = "Big Joe"
  |
  +- latestProduct
      |
      +- url = "products/greenmouse.html"
      |
      +- name = "green mouse"
```

数据模型类似于计算机的文件系统，latestProduct 可以看作是目录。
2、数据模型
（1）基础
在快速入门中介绍了在模板中使用的三种基本对象类型：scalars、hashes 和 sequences，其实还可以有其它更多的能力：
scalars：存储单值

hashes：充当其它对象的容器，每个都关联一个唯一的查询名字
sequences：充当其它对象的容器，按次序访问
方法：通过传递的参数进行计算，以新对象返回结果
用户自定义 FTL 标记：宏和变换器
通常每个变量只具有上述的一种能力，但一个变量可以具有多个上述能力，如下面的例子：

```
(root)
 |
 +- mouse = "Yerri"
      |
      +- age = 12
      |
      +- color = "brown">
```

mouse 既是 scalars 又是 hashes，将上面的数据模型合并到下面的模板：

```
${mouse}          <#-- use mouse as scalar -->
${mouse.age}    <#-- use mouse as hash -->
${mouse.color} <#-- use mouse as hash -->
```

输出结果是：

```
Yerri
12
brown
```

（2）Scalar 变量
Scalar 变量存储单值，可以是：
字符串：简单文本，在模板中使用引号（单引号或双引号）括起
数字：在模板中直接使用数字值
日期:存储日期/时间相关的数据,可以是日期、时间或日期-时间(Timestamp)；通常情况，日期值由程序员加到数据模型中，设计者只需要显示它们
布尔值：true 或 false，通常在<#if …>标记中使用
（3）hashes 、sequences 和集合
有些变量不包含任何可显示的内容,而是作为容器包含其它变量,者有两种类型：
hashes：具有一个唯一的查询名字和它包含的每个变量相关联
sequences：使用数字和它包含的每个变量相关联，索引值从 0 开始
集合变量通常类似 sequences，除非无法访问它的大小和不能使用索引来获得它的子变量；集合可以看作只能由<#list …>指令使用的受限 sequences
（4）方法
方法变量通常是基于给出的参数计算值。
下面的例子假设程序员已经将方法变量 avg 放到数据模型中，用来计算数字平均值：

```
The average of 3 and 5 is: ${avg(3, 5)}
The average of 6 and 10 and 20 is: ${avg(6, 10, 20)}
The average of the price of python and elephant is:
    ${avg(animals.python.price, animals.elephant.price)}
```

（5）宏和变换器

宏和变换器变量是用户自定义指令（自定义 FTL 标记），会在后面讲述这些高级特性
（6）节点
节点变量表示为树型结构中的一个节点，通常在 XML 处理中使用，会在后面的专门章节中讲
3、模板
（1）整体结构
模板使用 FTL（FreeMarker 模板语言）编写，是下面各部分的一个组合：
文本：直接输出
Interpolation：由${和}，或#{和}来限定，计算值替代输出
FTL 标记：FreeMarker 指令，和 HTML 标记类似，名字前加#予以区分，不会输出
注释：由<#--和-->限定，不会输出
下面是以一个具体模板例子：

```
<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <#-- Greet the user with his/her name -->
  <h1>Welcome ${user}!</h1>
  <p>We have these animals:
  <ul>
  <#list animals as being>
    <li>${being.name} for ${being.price} Euros
  </#list>
  </ul>
</body>
</html>
```

注意事项：
FTL 区分大小写，所以 list 是正确的 FTL 指令，而 List 不是；${name}和${NAME}是不同的
Interpolation 只能在文本中使用
FTL 标记不能位于另一个 FTL 标记内部，例如：

```
<#if <#include 'foo'>='bar'>...</if>
```

注释可以位于 FTL 标记和 Interpolation 内部，如下面的例子：

```
<h1>Welcome ${user <#-- The name of user -->}!</h1>
<p>We have these animals:
<ul>
<#list <#-- some comment... --> animals as <#-- again... -->
being>
...
```

余的空白字符会在模板输出时移除
（2）指令

在 FreeMarker 中，使用 FTL 标记引用指令。有三种 FTL 标记，这和 HTML 标记是类似的：
开始标记：<#directivename parameters>
结束标记：</#directivename>
空内容指令标记：<#directivename parameters/>
有两种类型的指令：预定义指令和用户定义指令。
用户定义指令要使用@替换#，如<@mydirective>...</@mydirective>（会在后面讲述）。
FTL 标记不能够交叉，而应该正确的嵌套，如下面的代码是错误的：

```
<ul>
<#list animals as being>
  <li>${being.name} for ${being.price} Euros
  <#if use = "Big Joe">
     (except for you)
</#list>
</#if> <#-- WRONG! -->
</ul>
```

如果使用不存在的指令，FreeMarker 不会使用模板输出，而是产生一个错误消息。
FreeMarker 会忽略 FTL 标记中的空白字符，如下面的例子：

```
<#list
  animals         as
     being
>
${being.name} for ${being.price} Euros
</#list      >
```

但是，<、</和指令之间不允许有空白字符。
（3）表达式
直接指定值
字符串
使用单引号或双引号限定
如果包含特殊字符需要转义，如下面的例子：

```
${"It's \"quoted\" and
this is a backslash: \\"}

${'It\'s "quoted" and
this is a backslash: \\'}
```

输出结果是：

```
It's "quoted" and
this is a backslash: \

It's "quoted" and
this is a backslash: \
```

下面是支持的转义序列：

转义序列	含义
\"	双引号(u0022)
\'	单引号(u0027)
	反斜杠(u005C)
\n	换行(u000A)
\r	Return (u000D)
\t	Tab (u0009)
\b	Backspace (u0008)
\f	Form feed (u000C)
\l	<
\g	>
\a	&
\{	{
\xCode	4 位 16 进制 Unicode 代码

有一类特殊的字符串称为 raw 字符串，被认为是纯文本，其中的\和{等不具有特殊含义，该类字符串在引号前面加 r，下面是一个例子：

```
${r"${foo}"}

${r"C:\foo\bar"}
```

输出的结果是：

```
${foo}

C:\foo\bar
```

数字

直接输入，不需要引号

精度数字使用“.”分隔，不能使用分组符号

目前版本不支持科学计数法，所以“1E3”是错误的

不能省略小数点前面的 0，所以“.5”是错误的

数字 8、+8、08 和 8.00 都是相同的

布尔值

true 和 false，不使用引号

序列

由逗号分隔的子变量列表，由方括号限定，下面是一个例子：

```
<#list ["winter", "spring", "summer", "autumn"] as x>
${x}
</#list>
```

输出的结果是：

```
winter
spring
```

summer
autumn
列表的项目是表达式，所以可以有下面的例子：
[2 + 2, [1, 2, 3, 4], "whatnot"]
可以使用数字范围定义数字序列，例如 2..5 等同于[2, 3, 4, 5]，但是更有效率，注意数字范围没有方括号
可以定义反递增的数字范围，如 5..2
散列（hash）
由逗号分隔的键/值列表，由大括号限定，键和值之间用冒号分隔，下面是一个例子：
{"name":"green mouse", "price":150}
键和值都是表达式，但是键必须是字符串
获取变量
顶层变量： ${variable}，变量名只能是字母、数字、下划线、$、@和#的组合，且不能以数字开头
从散列中获取数据
可以使用点语法或方括号语法，假设有下面的数据模型：

```
(root)
 |
 +- book
 |    |
 |    +- title = "Breeding green mouses"
 |    |
 |    +- author
 |         |
 |         +- name = "Julia Smith"
 |         |
 |         +- info = "Biologist, 1923-1985, Canada"
 |
 +- test = "title"
```

下面都是等价的：
book.author.name
book["author"].name
book.author.["name"]
book["author"]["name"]
使用点语法，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，因为名字是任意表达式的结果
从序列获得数据：和散列的方括号语法语法一样，只是方括号中的表达式值必须是数字；注意：第一个项目的索引是 0
序列片断：使用[startIndex..endIndex]语法，从序列中获得序列片断（也是序列）；startIndex 和 endIndex 是结果为数字的表达式
特殊变量：FreeMarker 内定义变量，使用.variablename 语法访问
字符串操作
Interpolation（或连接操作）

可以使用${..}（或#{..}）在文本部分插入表达式的值，例如：
${"Hello ${user}!"}

${"${user}${user}${user}${user}"}
可以使用+操作符获得同样的结果
${"Hello " + user + "!"}

${user + user + user + user}
${..}只能用于文本部分，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>
子串
例子（假设 user 的值为“Big Joe”）：
${user[0]}${user[4]}

${user[1..4]}
结果是（注意第一个字符的索引是 0）：
BJ

ig J
序列操作
连接操作：和字符串一样，使用+，下面是一个例子：
<#list ["Joe", "Fred"] + ["Julia", "Kate"] as user>
- ${user}
</#list>
输出结果是：
- Joe
- Fred
- Julia
- Kate
散列操作
连接操作：和字符串一样，使用+，如果具有相同的 key，右边的值替代左边的值，例如：
<#assign ages = {"Joe":23, "Fred":25} + {"Joe":30, "Julia":18}>
- Joe is ${ages.Joe}
- Fred is ${ages.Fred}
- Julia is ${ages.Julia}
输出结果是：
- Joe is 30
- Fred is 25
- Julia is 18

算术运算
＋、－、×、／、％，下面是一个例子：
${x * x - 100}
${x / 2}
${12 % 10}
输出结果是（假设 x 为 5）：
-75
2.5
2
操作符两边必须是数字，因此下面的代码是错误的：
${3 * "5"} <#-- WRONG! -->
使用+操作符时，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，例如：
${3 + "5"}
输出结果是：
35
使用内建的 int（后面讲述）获得整数部分，例如：
${(x/2)?int}
${1.1?int}
${1.999?int}
${-1.1?int}
${-1.999?int}
输出结果是（假设 x 为 5）：
2
1
1
-1
-1
比较操作符
使用=（或==，完全相等）测试两个值是否相等，使用!= 测试两个值是否不相等
=和!=两边必须是相同类型的值，否则会产生错误，例如<#if 1 = "1">会引起错误
Freemarker 是精确比较，所以对"x"、"x "和"X"是不相等的
对数字和日期可以使用<、<=、>和>=，但不能用于字符串
由于 Freemarker 会将>解释成 FTL 标记的结束字符，所以对于>和>=可以使用括号来避免这种情况，例如<#if (x > y)>
另一种替代的方法是，**使用 lt、lte、gt 和 gte 来替代<、<=、>和>=**
逻辑操作符
&&（and）、||（or）、!（not），只能用于布尔值，否则会产生错误
例子：
```
<#if x < 12 && color = "green">
  We have less than 12 things, and they are green.
</#if>
<#if !hot> <#-- here hot must be a boolean -->
```

It's not hot.
</#if>
内建函数
内建函数的用法类似访问散列的子变量，只是使用“?”替代“.”，下面列出常用的一些函数

字符串使用的：
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

序列使用的：
size：获得序列中元素的数目

数字使用的：
int：取得数字的整数部分（如-1.9?int 的结果是-1）
例子（假设 test 保存字符串"Tom & Jerry"）：
${test?html}
${test?upper_case?html}
输出结果是：
Tom & Jerry
TOM & JERRY
操作符优先顺序

操作符组	操作符
后缀	[subvarName] [subStringRange] . (methodParams)
一元	+expr、-expr、!
内建	?
乘法	*、 / 、%
加法	+、-
关系	<、>、<=、>= (lt、lte、gt、gte)
相等	== (=) 、!=
逻辑 and	&&
逻辑 or	双竖线
数字范围	..

（4）Interpolation
Interpolation 有两种类型：
通用 Interpolation：${expr}
数字 Interpolation：#{expr}或#{expr; format}
注意：Interpolation 只能用于文本部分

通用 Interpolation
插入字符串值：直接输出表达式结果
插入数字值：根据缺省格式（由#setting 指令设置）将表达式结果转换成文本输出；可以使用内建函数 string 格式化单个 Interpolation，下面是一个例子：

```
<#setting number_format="currency"/>
<#assign answer=42/>
${answer}
${answer?string}  <#-- the same as ${answer} -->
${answer?string.number}
${answer?string.currency}
${answer?string.percent}
```

输出结果是：

```
$42.00
$42.00
42
$42.00
4,200%
```

插入日期值：根据缺省格式（由#setting 指令设置）将表达式结果转换成文本输出；可以使用内建函数 string 格式化单个 Interpolation，下面是一个使用格式模式的例子：

```
${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')'")}
```

输出的结果类似下面的格式：

```
2003-04-08 21:24:44 Pacific Daylight Time
Tue, Apr 8, '03
Tuesday, April 08, 2003, 09:24:44 PM (PDT)
```

插入布尔值：根据缺省格式（由#setting 指令设置）将表达式结果转换成文本输出；可以使用内建函数 string 格式化单个 Interpolation，下面是一个例子：

```
<#assign foo=true/>
${foo?string("yes", "no")}
```

输出结果是：

```
yes
```

数字 Interpolation 的#{expr; format}形式可以用来格式化数字，format 可以是：
mX：小数部分最小 X 位
MX：小数部分最大 X 位
例子：

```
<#-- If the language is US English the output is: -->
<#assign x=2.582/>
<#assign y=4/>
#{x; M2}   <#-- 2.58 -->
#{y; M2}   <#-- 4    -->
#{x; m1}   <#-- 2.6 -->
#{y; m1}   <#-- 4.0 -->
```

#{x; m1M2} <#-- 2.58 -->
#{y; m1M2} <#-- 4.0 -->

4、杂项

（1）用户定义指令

宏和变换器变量是两种不同类型的用户定义指令，它们之间的区别是宏是在模板中使用 macro 指令定义，而变换器是在模板外由程序定义，这里只介绍宏

基本用法

宏是和某个变量关联的模板片断,以便在模板中通过用户定义指令使用该变量,下面是一个例子：

```
<#macro greet>
  <font size="+2">Hello Joe!</font>
</#macro>
```

作为用户定义指令使用宏变量时，使用@替代 FTL 标记中的#

```
<@greet></@greet>
```

如果没有体内容，也可以使用：

```
<@greet/>
```

参数

在 macro 指令中可以在宏变量之后定义参数，如：

```
<#macro greet person>
  <font size="+2">Hello ${person}!</font>
</#macro>
```

可以这样使用这个宏变量：

```
<@greet person="Fred"/> and <@greet person="Batman"/>
```

输出结果是：

```
  <font size="+2">Hello Fred!</font>

 and   <font size="+2">Hello Batman!</font>
```

宏的参数是 FTL 表达式，所以下面的代码具有不同的意思：

```
<@greet person=Fred/>
```

这意味着将 Fred 变量的值传给 person 参数，该值不仅是字符串，还可以是其它类型，甚至是复杂的表达式

可以有多参数，下面是一个例子：

```
<#macro greet person color>
  <font size="+2" color="${color}">Hello ${person}!</font>
</#macro>
```

可以这样使用该宏变量：

```
<@greet person="Fred" color="black"/>
```

其中参数的次序是无关的，因此下面是等价的：

```
<@greet color="black" person="Fred"/>
```

只能使用在 macro 指令中定义的参数，并且对所有参数赋值，所以下面的代码是错误的：

```
<@greet person="Fred" color="black" background="green"/>
<@greet person="Fred"/>
```

可以在定义参数时指定缺省值，如：

```
<#macro greet person color="black">
  <font size="+2" color="${color}">Hello ${person}!</font>
</#macro>
```
这样<@greet person="Fred"/>就正确了
宏的参数是局部变量，只能在宏定义中有效
嵌套内容
用户定义指令可以有嵌套内容，使用<#nested>指令执行指令开始和结束标记之间的模板片断
例子：
```
<#macro border>
  <table border=4 cellspacing=0 cellpadding=4><tr><td>
    <#nested>
  </tr></td></table>
</#macro>
```
这样使用该宏变量：
```
<@border>The bordered text</@border>
```
输出结果：
```
  <table border=4 cellspacing=0 cellpadding=4><tr><td>
    The bordered text
  </tr></td></table>
```
<#nested>指令可以被多次调用，例如：
```
<#macro do_thrice>
  <#nested>
  <#nested>
  <#nested>
</#macro>
<@do_thrice>
  Anything.
</@do_thrice>
```
输出结果：
```
  Anything.
  Anything.
  Anything.
```
嵌套内容可以是有效的FTL，下面是一个有些复杂的例子：<@border>
<@do_thrice>
<@greet person="Joe"/></@do_thrice>
</@border>}}} 输出结果：
```
  <table border=4 cellspacing=0 cellpadding=4><tr><td>
      <ul>
    <li><font size="+2">Hello Joe!</font>
    <li><font size="+2">Hello Joe!</font>
    <li><font size="+2">Hello Joe!</font>
  </ul>
  </tr></td></table>
```
宏定义中的局部变量对嵌套内容是不可见的，例如：
```
<#macro repeat count>
  <#local y = "test">
```

```
  <#list 1..count as x>
    ${y} ${count}/${x}: <#nested>
  </#list>
</#macro>
<@repeat count=3>${y?default("?")} ${x?default("?")}
${count?default("?")}</@repeat>
```
输出结果：
```
    test 3/1: ? ? ?
    test 3/2: ? ? ?
    test 3/3: ? ? ?
```
在宏定义中使用循环变量
用户定义指令可以有循环变量，通常用于重复嵌套内容，基本用法是：作为 nested 指令的参数传递循环变量的实际值，而在调用用户定义指令时，在<@…>开始标记的参数后面指定循环变量的名字
例子：
```
<#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>
```
输出结果：
```
  1. 0.5
  2. 1
  3. 1.5
  4. 2 Last!
```
指定的循环变量的数目和用户定义指令开始标记指定的不同不会有问题
调用时少指定循环变量，则多指定的值不可见
调用时多指定循环变量，多余的循环变量不会被创建
（2）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；局部变量隐藏（而不是覆盖）同名的 plain 变量；循环变量隐藏同名的局部变量和 plain 变量，下面是一个例子：
```
<#assign x = "plain">
1. ${x}  <#-- we see the plain var. here -->
<@test/>
6. ${x}  <#-- the value of plain var. was not changed -->
<#list ["loop"] as x>
```

```
  7. ${x}  <#-- now the loop var. hides the plain var. -->
<#assign x = "plain2"> <#-- replace the plain var, hiding does not mater
here -->
 8. ${x}  <#-- it still hides the plain var. -->
</#list>
   9. ${x}  <#-- the new value of plain var. -->
<#macro test>
  2. ${x}  <#-- we still see the plain var. here -->
  <#local x = "local">
  3. ${x}  <#-- now the local var. hides it -->
  <#list ["loop"] as x>
    4. ${x}  <#-- now the loop var. hides the local var. -->
  </#list>
  5. ${x}  <#-- now we see the local var. again -->
</#macro>
```

输出结果：

```
1. plain
  2. plain
  3. local
    4. loop
  5. local
6. plain
    7. loop
    8. loop
9. plain2
```

内部循环变量隐藏同名的外部循环变量，如：

```
<#list ["loop 1"] as x>
  ${x}
  <#list ["loop 2"] as x>
    ${x}
    <#list ["loop 3"] as x>
      ${x}
    </#list>
    ${x}
  </#list>
  ${x}
</#list>
```

输出结果：

```
  loop 1
    loop 2
      loop 3
    loop 2
  loop 1
```

模板中的变量会隐藏（而不是覆盖）数据模型中同名变量，如果需要访问数据模型中的同名变量，使用特殊变量 global，下面的例子假设数据模型中的 user 的值是 Big Joe：

```
<#assign user = "Joe Hider">
${user}          <#-- prints: Joe Hider -->
${.globals.user} <#-- prints: Big Joe -->
```

（3）名字空间

通常情况，只使用一个名字空间，称为主名字空间

为了创建可重用的宏、变换器或其它变量的集合（通常称库），必须使用多名字空间，其目的是防止同名冲突

创建库

下面是一个创建库的例子（假设保存在 lib/my_test.ftl 中）：

```
<#macro copyright date>
  <p>Copyright (C) ${date} Julia Smith. All rights reserved.
  <br>Email: ${mail}</p>
</#macro>
<#assign mail = "jsmith@acme.com">
```

使用 import 指令导入库到模板中,Freemarker 会为导入的库创建新的名字空间，并可以通过 import 指令中指定的散列变量访问库中的变量：

```
<#import "/lib/my_test.ftl" as my>
<#assign mail="fred@acme.com">
<@my.copyright date="1999-2002"/>
${my.mail}
${mail}
```

输出结果：

```
  <p>Copyright (C) 1999-2002 Julia Smith. All rights reserved.
  <br>Email: jsmith@acme.com</p>
jsmith@acme.com
fred@acme.com
```

可以看到例子中使用的两个同名变量并没有冲突,因为它们位于不同的名字空间

可以使用 assign 指令在导入的名字空间中创建或替代变量，下面是一个例子：

```
<#import "/lib/my_test.ftl" as my>
${my.mail}
<#assign mail="jsmith@other.com" in my>
${my.mail}
```

输出结果：

```
jsmith@acme.com
jsmith@other.com
```

数据模型中的变量任何地方都可见，也包括不同的名字空间，下面是修改的库：

```
<#macro copyright date>
  <p>Copyright (C) ${date} ${user}. All rights reserved.</p>
</#macro>
<#assign mail = "${user}@acme.com">
```

假设数据模型中的 user 变量的值是 Fred，则下面的代码：

```
<#import "/lib/my_test.ftl" as my>
<@my.copyright date="1999-2002"/>
${my.mail}
```
输出结果：
```
  <p>Copyright (C) 1999-2002 Fred. All rights reserved.</p>
Fred@acme.com
```

补充（静态方法的调用）：

方法 1：
```
##定义配置文件 freemarkerstatic.properties
_Validator=com.longyou.util.Validator
_Functions=com.longyou.util.Functions
_EscapeUtils=com.longyou.util.EscapeUtils
/调用代码
${_Functions.toUpperCase("Hello")}<br>
${_EscapeUtils.escape("狼的原野")}
```

方法 2：
```
${stack.findValue("@package.ClassName@method")}
```

补充：常用语法

EG.一个对象 BOOK

1.输出 ${book.name}

空值判断：${book.name?if_exists },

${book.name?default(‘xxx’)}//默认值 xxx

${ book.name!"xxx"}//默认值 xxx

日期格式：${book.date?string('yyyy-MM-dd')}

数字格式：${book?string.number}--20

${book?string.currency}--<#-- $20.00 -->

${book?string.percent}—<#-- 20% -->

插入布尔值：

```
<#assign foo=ture />
${foo?string("yes","no")} <#-- yes -->
```

2. 逻辑判断

a:

```
<#if condition>...
```

<#elseif condition2>...
<#elseif condition3>......
<#else>...
其中空值判断可以写成<#if book.name?? >

b:
<#switch value>
<#case refValue1>
...
<#break>
<#case refValue2>
...
<#break>
...
<#case refValueN>
...
<#break>
<#default>
...

3. 循环读取
<#list sequence as item>
...
空值判断<#if bookList?size = 0>
e.g.
<#list employees as e>
${e_index}. ${e.name}
输出:
1. Readonly
2. Robbin

freemarker 中 Map 的使用

```
<#list testMap?keys as testKey>
       < option value="${testKey}" >
              ${testMap[testKey]}
     </option>
</#list>
```

freemarker 的 Eclipse 插件

If you use Eclipse 2.x:
Open the **Window** menu, then **Open Perspective** -> **Install/Update**
Click with the right mouse button on the **Feature Updates** view, then select **New** -> **Site Bookmark**

In the displayed dialog box, type **"FreeMarker"** for Name and **"http://www.freemarker.org/eclipse/update"** for URL. Leave the "Bookmark type" radio buttons on "Eclipse update site".
Click **Finish**
Open the tree node under the newly created update site named "FreeMarker", select the **"FreeMarker *X.Y.Z*"** feature, and install it using the **Install now** button in the preview pane.
If you use Eclipse 3.x:
Help -> **Software updates** -> **Find and install...**.
Choose "Search for new features to install".
Click **Add Update Site...**, and type **"FreeMarker"** for Name and **"http://www.freemarker.org/eclipse/update"** for URL.
Check the box of the "FreeMarker" feature.
"Next"-s until it is installed...

关键字：FreeMarker

引用地址：http://www.dlog.cn/html/trackback.do?id=13925&type=1 (复制地址)

FreeMarker 设计指南(2)

2、数据模型

（1）基础

在快速入门中介绍了在模板中使用的三种基本对象类型：scalars、hashes 和 sequences，其实还可以有其它更多的能力：

scalars：存储单值

hashes：充当其它对象的容器，每个都关联一个唯一的查询名字

sequences：充当其它对象的容器，按次序访问

方法：通过传递的参数进行计算，以新对象返回结果

用户自定义 FTL 标记：宏和变换器

通常每个变量只具有上述的一种能力，但一个变量可以具有多个上述能力，如下面的例子：

```
(root)
 |
 +- mouse = "Yerri"
      |
      +- age = 12
      |
      +- color = "brown">
```

mouse 既是 scalars 又是 hashes，将上面的数据模型合并到下面的模板：

```
${mouse}          <#-- use mouse as scalar -->
${mouse.age}      <#-- use mouse as hash -->
${mouse.color} <#-- use mouse as hash -->
```

输出结果是：

```
Yerri
```

12
brown
（2）Scalar 变量
Scalar 变量存储单值，可以是：
字符串：简单文本，在模板中使用引号（单引号或双引号）括起
数字：在模板中直接使用数字值
日期：存储日期/时间相关的数据，可以是日期、时间或日期-时间（Timestamp）；通常情况，日期值由程序员加到数据模型中，设计者只需要显示它们
布尔值：true 或 false，通常在<#if …>标记中使用
（3）hashes 、sequences 和集合
有些变量不包含任何可显示的内容，而是作为容器包含其它变量，者有两种类型：
hashes：具有一个唯一的查询名字和它包含的每个变量相关联
sequences：使用数字和它包含的每个变量相关联，索引值从 0 开始
集合变量通常类似 sequences，除非无法访问它的大小和不能使用索引来获得它的子变量；集合可以看作只能由<#list …>指令使用的受限 sequences
（4）方法
方法变量通常是基于给出的参数计算值
下面的例子假设程序员已经将方法变量 avg 放到数据模型中，用来计算数字平均值：
The average of 3 and 5 is: ${avg(3, 5)}
The average of 6 and 10 and 20 is: ${avg(6, 10, 20)}
The average of the price of python and elephant is: ${avg(animals.python.price, animals.elephant.price)}
（5）宏和变换器
宏和变换器变量是用户自定义指令（自定义 FTL 标记），会在后面讲述这些高级特性
（6）节点
节点变量表示为树型结构中的一个节点，通常在 XML 处理中使用，会在后面的专门章节中讲述

Trackback: http://tb.blog.csdn.net/TrackBack.aspx?PostId=164465

解析 FreeMarker 视图

- 解析 FreeMarker 视图

声明一个针对 FreeMarker 的视图解析器：

```
<bean id="viewResolver" class="org.springframework.
```

```
        ➥web.servlet.view.freemarker.FreeMarkerViewResolver">
    <property name="suffix"><value>.ftl</value></property>
  </bean>
```

FreeMarkerViewResolver 和 VelocityViewResolver 或 InternalResourceViewResolver 的工作机制相同。模板资源是通过在视图的逻辑名上增加 prefix 属性的值作为前缀，以及增加 suffix 属性的值作为后缀进行解析的。和 VelocityViewResolver 一样，在这里我们又一次只设置 suffix 属性，因为模板的路径已经在 FreeMarkerConfigurer 的 templateLoaderPath 属性中定义了。

暴露请求和会话属性

在第 9.1.3 节中，你看到如何告诉 VelocityViewResolver 将请求和会话属性复制到模型 map 中，从而它们能够在模板中作为变量使用。采用同样的方式配置 FreeMarkerViewResolver，可以将请求和会话属性作为变量暴露给 FreeMarker 模板使用。要做到这一点，可以设置 exposeRequestAttributes 或者 exposeSessionAttributes 为 true：

```
  <bean id="viewResolver" class="org.springframework.
          web.servlet.view.freemarker.FreeMarkerViewResolver">
  ...
    <property name="exposeRequestAttributes">
      <value>true</value>
    </property>
    <property name="exposeSessionAttributes">
      <value>true</value>
    </property>
  </bean>
```

这里，两个属性都被设置为 true。结果是请求和会话属性都被复制到模板的属性集中，可以使用 FreeMarker 的表达式语言来访问并显示。

FreeMarker 文章引用自：

发表时间：2007 年 11 月 2 日 16 时 54 分 13 秒 本文链接：http://user.qzone.qq.com/55117942/blog/1193993653 评论/阅读(0/5)

[顶]FreeMarker

包含 FreeMarker 的指令的文件就称为模板（Template）。
模板设计者不关心数据从那儿来，只知道使用已经建立的数据模型。

数据模型由程序员编程来创建，向模板提供变化的信息，这些信息来自于数据库、文件，甚至于在程序中直接生成。

数据类型：

一、基本：
1、scalars：存储单值

字符串：简单文本由单或双引号括起来。
数字：直接使用数值。
日期：通常从数据模型获得
布尔值：true 或 false，通常在<#if …>标记中使用

2、hashes：充当其它对象的容器，每个都关联一个唯一的查询名字

具有一个唯一的查询名字和他包含的每个变量相关联。

3、sequences：充当其它对象的容器，按次序访问

使用数字和他包含的每个变量相关联。索引值从 0 开始。

4、集合变量：

除了无法访问它的大小和不能使用索引来获得它的子变量:集合可以看作只能由<#list...>指令使用的受限 sequences。

5、方法：通过传递的参数进行计算，以新对象返回结果

方法变量通常是基于给出的参数计算值在数据模型中定义。

6、用户自定义 FTL 指令：宏和变换器

7、节点

节点变量表示为树型结构中的一个节点，通常在 XML 处理中使用。

模板：

使用 FTL（freeMarker 模板语言）编写

组成部分

一、整体结构

1、注释：<#--注释内容-->，不会输出。

2、文本：直接输出。

3、interpolation:由 ${var} 或 #{var} 限定，由计算值代替输出。

4、FTL 标记

二、指令：
freemarker 指令有两种：
1、预定义指令：引用方式为<#指令名称>
2、用户定义指令：引用方式为<@指令名称>，引用用户定义指令时须将#换为@。
注意：如果使用不存在的指令，FreeMarker 不会使用模板输出，而是产生一个错误消息。

freemarker 指令由 FTL 标记来引用，FTL 标记和 HTML 标记类似，名字前加#来加以区分。如 HTML 标记的形式为<h1></h1>则 FTL 标记的形式是<#list></#list>(此处 h1 标记和 list 指令没有任何功能上的对应关系，只是做为说明使用一下）。

有三种 FTL 标记：
1)、开始标记：<#指令名称>
2)、结束标记：</#指令名称>
3)、空标记：<#指令名称/>

注意：

1) FTL 会忽略标记之中的空格，但是，<#和指令 与 </#和指令 之间不能有空格。
2) FTL 标记不能够交叉，必须合理嵌套。每个开始标记对应一个结束标记，层层嵌套。 如：

```
<#list>
<li>
${数据}
<#if 变量>
<p>game over!</p>
</#if>
</li>
</#list>
```

注意事项：
1)、FTL 对大小写敏感。所以使用的标记及 interpolation 要注意大小写。name 与 NAME 就是不同的对象。<#list>是正确的标记，而<#List>则不是。
2)、interpolation 只能在文本部分使用，不能位于 FTL 标记内。如<#if ${var}>是错误的，正确的方法是：<#if var>,而且此处 var 必须为布尔值。
3)、FTL 标记不能位于另一个 FTL 标记内部，注释例外。注释可以位于标记及 interpolation 内部。

三、表达式

1、直接指定值：

1-1、字符串：
由双引号或单引号括起来的字符串，其中的特殊字符（如’ ″ \等）需要转义。

1-2、raw 字符串：
有一种特殊的字符串称为 raw 字符串，被认为是纯文本，其中的\和{等不具有特殊含义，该类字符串在引号前面加 r，下面是一个例子：
${r″/${data}″year″″}屏幕输出结果为：/${data}″year″

转义　　含义
序列

\″　双引号(u0022)

\’　单引号(u0027)

\\　反斜杠(u005C)

\n　换行(u000A)

\r　Return (u000D)

\t　Tab (u0009)

\b　Backspace (u0008)

\f　Form feed (u000C)

\l　<

\g　>

\a　&

\{　{

\xCode　4 位 16 进制 Unicode 代码

1-3、数字：直接输入，不需要引号

1)、精度数字使用“.”分隔，不能使用分组符号
2)、目前版本不支持科学计数法，所以“1E3”是错误的
3)、不能省略小数点前面的 0，所以“.5”是错误的
4)、数字 8、+8、08 和 8.00 都是相同的

1-4、布尔值：true 和 false，不使用引号

1-5、序列：由逗号分隔的子变量列表，由[]方括号限定。
1)、子变量列表可以是表达式
2)、可以使用数字范围定义数字序列,不需要方括号限定,例如 2..5 等同于[2, 3, 4, 5]，但是更有效率，可以定义反递增范围如：5..2。

1-6、散列(hash)
1)、由逗号分隔的键/值列表，由{}大括号限定，键和值之间用冒号分隔，如：
{"key1":valu1,"key2":"character string"....}
2)、键和值都是表达式，但是键必须是字符串。

2、获取变量：

2-1、顶层变量：${变量名}

变量名只能是字母、数字、下划线、$、#、@ 的组合，且不能以数字开头。

2-2、散列：有两种方法

1)、点语法：变量名字和顶层变量的名字受同样的限制
2)、方括号语法：变量名字无限制，可以是任意的表达式的结果
book.author.name
book.author.["name"]
book["author"].name
book["author"]["name"]
以上是等价的。

2-3、序列：使用散列的方括号语法获取变量，方括号中的表达式结果必须为数字。注意：第一个项目的索引为 0。可以使用
[startindex..endindex]语法获取序列片段。

2-4、特殊变量：FreeMarker 内定义变量，使用.variablename 语法访问。

3、字符串操作

3-1、interpolation:使用${}或#{}在文本部分插入表达式的值，例如：

${"hello${username}!"}
${"${username}${username}${username}"}

也可以使用+来获得同样的结果：
${"hello"+username+"!"}
${username+username+username}

注意：${}只能用于文本部分而不能出现于标记内。

<#if ${user.login}>或<#if "${user.login}">都是错误的;
<#if user.login>是正确的。
本例中 user.login 的值必须是布尔类型。

3-2、子串：
举例说明：假如 user 的值为"Big Joe"
${user[0]}${user[4]}结果是：BJ
${user[1..4]}结果是：ig J

4、序列操作

4-1、连接操作：可以使用+来操作，例如：
["title","author"]+["month","day"]

5、散列操作
5-1、连接操作：可以使用+来操作，如果有相同的 KEY，则右边的值会替代左边的值，例如：
{"title":散列,"author":"emma"}+{"month":5,"day":5}+{"month":6}结果 month 的值就是 6。

6、算术运算

6-1、操作符：+、-、*、/、%
除+号以外的其他操作符两边的数据，必须都是数字类型。
如果+号操作符一边有一个字符型数据，会自动将另一边的数据转换为字符型数据，运算结果为字符型数据。

6-2、比较操作符：
1)、=
2)、==
3)、!=
4)、<
5)、<=
6)、>
7)、>=
1-3 的操作符，两边的数据类型必须相同，否则会产生错误

4-7 的操作符，对于日期和数字可以使用，字符串不可以使用。

注意：
1)、FreeMarker 是精确比较，所以"x" "x " "X"是不等的。
2)、因为<和>对 FTL 来说是开始和结束标记，所以，可以用两种方法来避免这种情况：
一种是使用括号<#if (a<b)>
另一是使用替代输出，对应如下：
< lt
<= lte
> gt
>= gte

6-3、逻辑操作符：只能用于布尔值，否则会出现错误。

&&(and)与运算
||(or)或运算
!(not)非运算

6-4、内建函数：使用方法类似于访问散列的子变量，只是使用?代替.例如：
${test?upper_case?html}

常用的内建函数列举如下：

1)、字符串使用：

html:对字符串进行 HTML 编码
cap_first:字符串第一个字母大写
lower_first:字符串第一个字母小写
upper_case:将字符串转换成大写
trim:去年字符前后的空白字符

2)、序列使用：
size:获得序列中元素的数目

3)、数字使用：
int:取得数字的整数部分

7、操作符的优先顺序：

后缀：[subbarName][subStringRange].(mathodParams)
一元：+expr、-expr、! (not)
内建：?
乘法：*、/、%
加法：+、-

关系：<、<=、>、>= (lt、lte、gt、gte)
相等：=、==、!=
逻辑与：&& (and)
逻辑或：|| (or)
数字范围：..

四、interpolation

inperpolation 只能用于文本，有两种类型：通用 interpolation 及数字 interpolation

1、通用 interpolation

如${expr}

1-1、插入字符串值：直接输出表达式结果。
1-2、插入数字值：根据缺省格式(由 setting 指令设置)将表达式结果转换成文本输出;可以使用内建函数 string 来格式化单个 interpolation

如：
<#setting number_format="currency" />
<#assign answer=42 />
${answer} <#-- ￥42.00 -->
${answer?string} <#-- ￥42.00 -->
${answer?string.number} <#-- 42 -->
${answer?string.currency} <#-- ￥42.00 -->
${answer?string.percent} <#-- 42,00% -->

1-3、插入日期值：根据缺省格式(由 setting 指令设置)将表达式结果转换成文本输出;可以使用内建函数 string 来格式化单个 interpolation

如：

${lastupdata?string("yyyy-MM-dd HH:mm:ss zzzz")} <#-- 2003-04-08 21:24:44 Pacific Daylight Time -->
${lastupdata?string("EEE,MMM d, ''yy")} <#-- tue,Apr 8, '03 -->
${lastupdata?string("EEEE,MMMM dd, yyyy,hh:mm:ss a '('zzz')'")} <#-- Tuesday,April 08, 2003, 09:24:44 PM (PDT)-->

1-4、插入布尔值：根据缺省格式(由 setting 指令设置)将表达式结果转换成文本输出;可以使用内建函数 string 来格式化单个 interpolation

如：
<#assign foo=ture />
${foo?string("yes","no")} <#-- yes -->

2、数字 interpolation:

有两种形式：
1)、#{expr}
2)、#{expr;format}：format 可以用来格式化数字，format 可以是如下：
mX:小数部分最小 X 位
MX:小数部分最大 X 位

例如：
<#assign x=2.582 />
<#assign y=4 />
#{x;M2} <#-- 2.58 -->
#{y;M2} <#-- 4 -->
#{x;m1} <#-- 2.582 -->
#{y;m1} <#-- 4.0 -->
#{x;m1M2} <#-- 2.58 -->
#{y;m1M2} <#-- 4.0 -->

杂项

一、用户定义指令

宏和变换器变量是两种不同类型的用户自定义指令，他们的区别是：

宏可以在模板中用 macro 指令来定义
变换器是在模板外由程序定义

1、宏：和某个变量关联的模板片段，以便在模板中通过用户自定义指令使用该变量
1-1、基本用法：
例如：
<#macro greet>
 Hello JOE!
</#macro>

使用时：
<@greet></@greet>
如果没有体内容也可以用
<@greet />

1-2、变量：

1)、可以在宏定义之后定义参数，宏参数是局部变量，只在宏定义中有效。如：

<#macro greet person>
 Hello ${person}!
</#macro>
使用时：
<@greet person="emma"> and <@greet person="LEO">
输出为：
 Hello emma!
 Hello LEO!

注意：宏的参数是 FTL 表达式，所以，person=emma 和上面的例子中具有不同的意义，这意味着将变量 emma 的值传给 person,这个值可能是任意一种数据类型，甚至是一个复杂的表达式。

宏可以有多个参数，使用时参数的次序是无关的，但是只能使用宏中定义的参数，并且对所有参数赋值。如：
<#macro greet person color>
 Hello ${person}!
</#macro>

使用时：
<@greet color="black" person="emma" />正确
<@greet person="emma" />错误，color 没有赋值，此时，如果在定义宏时为 color 定义缺省值
<#macro greet person color="black">这样的话，这个使用方法就是正确的。
<@greet color="black" person="emma" bgcolor="yellow" />错误，宏 greet 定义中未指定 bgcolor 这个参数

2、嵌套内容：

2-1、自定义指令可以有嵌套内容，使用<#nested>指令，执行自定义指令开始和结束标记之间的模板片段。例如：
<#macro greet>

<#nested>

</#macro>

<@greet>hello Emma!</@greet>

输出为

hello Emma!

2-2、<#nested>指令可以被多次调用，例如

```
<#macro greet>
<p>
<#nested>
<#nested>
<#nested>
<#nested>
</p>
</#macro>

<@greet>hello Emma!</@greet>
```

输出为

```
<p>
hello Emma!
hello Emma!
hello Emma!
hello Emma!
</p>
```

2-3、嵌套的内容可以是有效的FTL，例如：

```
<#macro welcome>
<p>
<#nested>
</p>
</#macro>

<#macro greet person color="black">
<font size="+2" color="${color}"> Hello ${person}!</font>
</#macro>

<@welcome>
<@greet person="Emma" color="red" />
<@greet person="Andrew" />
<@greet person="Peter" />
</@welcome>
```

输出为：

```
<p>
<font size="+2" color="red"> Hello Emma!</font>
<font size="+2" color="black"> Hello Andrew!</font>
<font size="+2" color="black"> Hello Peter!</font>
```

</p>

2-4、宏定义中的局部变量对嵌套内容是不可见的，例如：

```
<#macro repeat count>
<#local y="test" />
<#list 1..count as x>
${y}${count}/${x}:<#nested />
</#list>
</#macro>

<@repeat count=3>
${y?default("?")}
${x?default("?")}
${count?default("?")}
</@repeat>
```

输出结果为

```
test 3/1:???
test 3/2:???
test 3/3:???
```

2-5、在宏定义中使用循环变量，通常用来重复嵌套内容，基本用法为：作为 nested 指令的参数，传递循环变量的实际值，而在调用自定义指令时，在标记的参数后面指定循环变量的名字。
例如：

```
<#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>
```

输出结果是

```
1. 0.5
2. 1
3. 1.5
4. 2last!
```

注意：指定循环变量的数目和用户定义指令开始标记指定的不同不会有问题
调用时，少指定循环变量，多指定的值会不见
调用时，多指定循环变量，多余的循环变量不会被创建

二、在模板中定义变量

1、在模板中定义的变量有三种类型

1-1、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。
1-2、局部变量：在宏定义体中有效，使用 local 指令创建和替换。
1-3、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。

注意：
1)、宏的参数是局部变量，不是循环变量。
2)、局部变量隐藏同名的 plain 变量
3)、循环变量隐藏同名的 plain 变量和局部变量。

例如：

```
<#assign x="plain">
1. ${x} <#-- plain -->

<@test />

6. ${x}
<#list ["loop"] as x>
7. ${x} <#-- loop -->
<#assign x="plain2">
8. ${x} <#-- loop -->
</#list>
9. ${x} <#-- plain2 -->

<#macro test>
2. ${x} <#-- plain -->
<#local x="local">
3. ${x} <#-- local -->
<#list ["loop"] as x>
4. ${x} <#-- loop -->
</#list>
5. ${x} <#-- local -->
</#macro>
```

4)、内部循环变量隐藏同名的外部循环变量

```
<#list ["loop1"] as x>
${x} <#-- loop1 -->
<#list ["loop2"] as x>
${x} <#-- loop2 -->
<#list ["loop3"] as x>
${x} <#-- loop3 -->
</#list>
${x} <#-- loop2 -->
</#list>
${x} <#-- loop1 -->
</#list>
```

5)、模板中的变量会隐藏数据模型中的同名变量，如果需访问数据模型中的变量，使用特殊变量 global。

例如：
假设数据模型中的 user 值为 Emma

```
<#assign user="Man">
${user} <#-- Man -->
${.global.user} <#-- Emma -->
```

数据源+freemarker+servlet 生成 xml 文件

一.步骤:

1.在 server.xml 文件中建立数据源.

```
 <Service name="Cms">
     <Connector debug="0" enableLookups="false" port="8084" protocol="AJP/1.3" redir
ectPort="8443"/>
    <Connector acceptCount="100" connectionTimeout="20000" debug="0" disableUploa
dTimeout="true"

 enableLookups="false" maxSpareThreads="75" maxThreads="150" minSpareThreads="
```

```
25" port="8081" redirectPort="8443"/>
      <Engine defaultHost="localhost_Cms" name="Catalina_Cms">
       <Logger className="org.apache.catalina.logger.FileLogger" prefix="localhost_cmt_l
og." suffix=".txt" timestamp="true"/>
       <Realm className="org.apache.catalina.realm.UserDatabaseRealm"/>
       <Host autoDeploy="true" debug="0" name="localhost_Cms" unpackWARs="true" x
mlNamespaceAware="false" xmlValidation="false">
         <Context debug="0" docBase="D:WorkspaceCMSweb" path="/" reloadable="true
" workDir="D:WorkspaceCMSj2src">
          <Resource name="jdbc/news_DB" auth="Container" type="javax.sql.DataSource
"/>
         <ResourceParams name="jdbc/news_DB">
       <parameter>
        <name>factory</name>
        <!-- DBCP Basic Datasource Factory -->
        <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
       </parameter>

       <parameter>
        <name>maxActive</name>
        <value>1000</value>
       </parameter>
       <parameter>
         <name>validationQuery</name>
         <value>select 1+1</value>
       </parameter>
       <parameter>
```

```
  <name>maxIdle</name>
  <value>100</value>
</parameter>
<parameter>
  <name>maxWait</name>
  <value>10000</value>
</parameter>
<parameter>
  <name>removeAbandoned</name>
  <value>true</value>
</parameter>
<parameter>
  <name>removeAbandonedTimeout</name>
  <value>60</value>
</parameter>
<parameter>
  <name>logAbandoned</name>
  <value>false</value>
</parameter>

<parameter>
  <name>username</name>
  <value>aaaa</value>
</parameter>
<parameter>
  <name>password</name>
  <value>bbbbbb</value>
```

```
    </parameter>
    <parameter>
      <name>driverClassName</name>
      <value>net.sourceforge.jtds.jdbc.Driver</value>
    </parameter>
    <parameter>
      <name>url</name>
      <value>jdbc:jtds:sqlserver://111.111.111.111:1433/cms</value>
    </parameter>
     </ResourceParams>

  </Context>
    </Host>
</Engine>
 </Service>
```

2.在 web.xml 文件中配置 servlet

```
  <servlet>
     <description>generate xml file</description>
     <servlet-name>NewsXmlServlet</servlet-name>
     <servlet-class>xml.NewsXmlServlet</servlet-class>
  </servlet>
  <servlet-mapping>
     <servlet-name>NewsXmlServlet</servlet-name>
     <url-pattern>/xmlServlet</url-pattern>
```

```
    </servlet-mapping>
    <servlet>
```

3.newsXmlServlet.java

```
package xml;

import java.io.*;
import java.nio.charset.Charset;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

import freemarker.template.Configuration;
```

```
import freemarker.template.Template;
import freemarker.template.TemplateException;


import java.util.Locale;


public class NewsXmlServlet extends HttpServlet{

   protected void doGet(HttpServletRequest req, HttpServletResponse resp)
           throws ServletException, IOException {
       // TODO Auto-generated method stub
       try {
           Connection conn=null;
           Context ctx = new InitialContext();
           DataSource ds=(DataSource)ctx.lookup("java:comp/env/jdbc/news_DB");
           conn=ds.getConnection();
           Statement stmt=conn.createStatement();
           ResultSet rs=stmt.executeQuery("select url,updatetime,tpf_edu_contentTitle,tpf
_edu_contentneirong,tpf_edu_contentlaiyuan,tpf_edu_contentkeyword from tp5__edu_con
tent where url<>'' and url is not null and dateDiff(d,updatetime,getDate())=1 order by upd
atetime desc");
           Configuration cfg=new Configuration();
           cfg.setDirectoryForTemplateLoading(new File("E:/wwwroot/CMS/web/WEB-INF/c
lasses/xml"));
           Template tem=cfg.getTemplate("news.ftl");

           List list=new ArrayList();
           OutputStreamWriter out=new OutputStreamWriter(System.out);
```

```
            while(rs.next()){
                Map item=new HashMap();
                item.put("title",rs.getString(3));
                item.put("link","http://test.com.cn"+rs.getString(1));
                item.put("pubdate",rs.getTimestamp(2));
                item.put("content",DelHtml(rs.getString(4)));
                item.put("source",rs.getString(5));
                item.put("keywords",DelHtml(rs.getString(6)));
                list.add(item);
            }
            Map data=new HashMap();
            data.put("items",list);
            StringWriter writer=new StringWriter();
            tem.process(data,writer);
            String content=writer.toString();
            writer.close();
            createXml(content);
            out.close();
            //resp.setContentType("text/xml; charset=utf-8");
            //resp.getWriter().write(content);

        } catch (NamingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
        } catch (TemplateException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
    public String DelHtml(String content){
        String contents=content.replaceAll("<\/?\s*(\S+)(\s*[^>]*)?\s*\/?>","");
        contents=contents.replaceAll("&ldquo;", "“");
        contents=contents.replaceAll("&rdquo;","”");
        contents=contents.replaceAll("&ldquo;", "‘");
        contents=contents.replaceAll("&rdquo;","’");
        contents=contents.replaceAll("&middot;","·");
        contents=contents.replaceAll("&mdash;","—");
        contents=contents.replaceAll("&hellip;","…");
        contents=contents.replaceAll(" ","");
        contents=contents.replaceAll(","," ");
        return contents;
    }
    public void createXml(String fileContent){
        try {
            String filePath="E:/wwwroot/cmsHtml/education/news.xml";
            File fileXml=new File(filePath);
            if(!fileXml.exists()){
                fileXml.createNewFile();
            }

```

```
            /*FileWriter fileWriter=new FileWriter(fileXml);
            fileWriter.
            fileWriter.write(fileContent);
            fileWriter.close();*/
                OutputStreamWriter writer=new OutputStreamWriter(new FileOutputStream(f
ileXml), Charset.forName("utf-8"));
                writer.write(fileContent);
                writer.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
    }

    public void destroy() {
        // TODO Auto-generated method stub
        super.destroy();
    }

    public void init() throws ServletException {
        // TODO Auto-generated method stub
        super.init();
    }

}
```

4.news.ftl

```
<?xml version="1.0" encoding="utf-8" ?>
<document>
    <webSite>edu.aweb.com.cn</webSite>
    <webMaster>webmaster@aweb.com.cn</webMaster>
    <updatePeri>1440</updatePeri>
    <#list items as it>
    <item>
        <title><![CDATA[${it.title}]]></title>
        <link>${it.link}</link>
        <pubDate>${it.pubdate}</pubDate>
        <text><![CDATA[${it.content}]]></text>
      <image/>
        <source>${it.source}</source>
        <keywords><![CDATA[${it.keywords}]]></keywords>
    </item>
    </#list>
</document>
```

Trackback: http://tb.blog.csdn.net/TrackBack.aspx?PostId=1889578

FreeMarker 学习笔记 3

2007-10-18 13:11

（3）模板

1　　在 FreeMarker 模板中可以包括下面三种特定部分：

- Ø　　${…}：称为 interpolations，FreeMarker 会在输出时用实际值进行替代

- FTL 标记（FreeMarker 模板语言标记）：类似于 HTML 标记，为了与 HTML 标记区分，用#开始（有些以@开始，在后面叙述）
- 注释：包含在<#--和-->（而不是<!--和-->）之间

1 下面是一些使用指令的例子：

- if 指令

```
<#if animals.python.price < animals.elephant.price>
  Pythons are cheaper than elephants today.
<#else>
  Pythons are not cheaper than elephants today.
</#if>
```

- list 指令

```
<p>We have these animals:
<table border=1>
  <tr><th>Name<th>Price
  <#list animals as being>
  <tr><td>${being.name}<td>${being.price} Euros
  </#list>
</table>
```

玩过 C#的都一眼就看出来了,和 foreach 一样.用过 java 的 for(xx:xx)的家伙也清楚吧.其实就是遍历这个 being.

输出为：

```
<p>We have these animals:
<table border=1>
  <tr><th>Name<th>Price
  <tr><td>mouse<td>50 Euros
  <tr><td>elephant<td>5000 Euros
  <tr><td>python<td>4999 Euros
```

```
</table>
```

Ø include 指令

```
<html>
<head>
  <title>Test page</title>
</head>
<body>
  <h1>Test page</h1>
  <p>Blah blah...
<#include "/copyright_footer.html">
</body>
</html>
```

这个就不用解释了,嵌入个网页

Ø 一起使用指令

```
<p>We have these animals:
<table border=1>
  <tr><th>Name<th>Price
  <#list animals as being>
  <tr>
    <td>
      <#if being.size = "large"><b></#if>
      ${being.name}
      <#if being.size = "large"></b></#if>
    <td>${being.price} Euros
  </#list>
</table>
```

Spring 中使用 FreeMaker 或 Vilocity 模板发送邮件

本文以用户注册后为用户发送一封邮件为例子，讲述如何在 Spring 中使用 FreeMaker 或 Vilocity 发送邮件。

Spring 配置文件：

xml 代码

```
1.
2.      <bean id="mailSender" class="org.springframework.mail.javam
    ail.JavaMailSenderImpl">
3.          <property name="host" value="smtp.163.com"/>
4.          <property name="username" value="test"/>
5.          <property name="password" value="123456"/>
6.          <property name="javaMailProperties">
7.                  <props>
8.                    <prop key="mail.smtp.auth">trueprop>
9.                  props>
10.         property>
11.     bean>
12.
13.
14.     <bean id="freeMarkerConfigurer"                    class="o
    rg.springframework.web.servlet.view.freemarker.FreeMarkerConfig
    urer">
15.         <property name="templateLoaderPath" value="/WEB-INF/fre
    emakertemplate/" />
16.         <property name="freemarkerSettings">
17.             <props>
18.                 <prop key="template_update_delay">0prop>
19.                 <prop key="default_encoding">GBKprop>
20.                 <prop key="locale">zh_CNprop>
21.             props>
22.         property>
23.     bean>
24.
25.
26.     <bean id="velocityEngine" class="org.springframework.ui.vel
    ocity.VelocityEngineFactoryBean">
27.         <property name="resourceLoaderPath" value="/WEB-INF/vil
    ocitytemplate/" />
28.         <property name="velocityProperties">
```

```
29.             <props>
30.                 <prop key="velocimacro.library">*.vmprop>
31.                 <prop key="default.contentType">text/html; char
   set=utf-8prop>
32.                 <prop key="output.encoding">utf-8prop>
33.                 <prop key="input.encoding">utf-8prop>
34.             props>
35.         property>
36.     bean>
37.
38.      <bean id="mailMessage" class="org.springframework.mail.Sim
   pleMailMessage" singleton="false">
39.         <property name="from" value="test@163.com"/>
40.     bean>
41.
42.     <bean id="mailEngine" class="test.MailEngine">
43.         <property name="mailSender" ref="mailSender"/>
44.
45.         <property name="velocityEngine" ref="velocityEngine"/>

46.
47.        <property name="freeMarkerConfigurer" ref="freeMarkerCon
   figurer" />
48.     bean>
```

java 代码

```
1.  MailEngine类：
2.
3. public class MailEngine {
4.     protected static final Log log = LogFactory.getLog(MailEngi
   ne.class);
5.
6. //    private FreeMarkerConfigurer freeMarkerConfigurer;
7.     private VelocityEngine velocityEngine;
8.     private MailSender mailSender;
9.
10.//    public void setFreeMarkerConfigurer(
11.//            FreeMarkerConfigurer freeMarkerConfigurer) {
12.//        this.freeMarkerConfigurer = freeMarkerConfigurer;
13.//    }
```

```
14.
15.     public void setMailSender(MailSender mailSender) {
16.         this.mailSender = mailSender;
17.     }
18.
19.     public void setVelocityEngine(VelocityEngine velocityEngine
    ) {
20.         this.velocityEngine = velocityEngine;
21.     }
22.
23.     /**
24.      * 通过模板产生邮件正文
25.      * @param templateName    邮件模板名称
26.      * @param map             模板中要填充的对象
27.      * @return 邮件正文（HTML）
28.      */
29.     public String generateEmailContent(String templateName, Map
     map) {
30.         //使用 FreeMaker 模板
31. //         try {
32. //             Configuration configuration = freeMarkerConfigure
    r.getConfiguration();
33. //             Template t = configuration.getTemplate(templateNa
    me);
34. //             return FreeMarkerTemplateUtils.processTemplateInt
    oString(t, map);
35. //         } catch (TemplateException e) {
36. //             log.error("Error while processing FreeMarker temp
    late ", e);
37. //         } catch (FileNotFoundException e) {
38. //             e.printStackTrace();
39. //             //log.error("Error while open template file ", e)
    ;
40. //         } catch (IOException e) {
41. //             log.error("Error while generate Email Content ",
    e);
42. //         }
43.
44. //         使用 Vilocity 模板
45.         try {
46.            return VelocityEngineUtils.mergeTemplateIntoString(v
    elocityEngine, templateName, map);
47.         } catch (VelocityException e) {
```

```
48.             log.error("Error while processing Vilocity template
    ", e);
49.         }
50.
51.         return null;
52.     }
53.
54.     /**
55.      * 发送邮件
56.      * @param emailAddress         收件人Email地址的数组
57.      * @param fromEmail             寄件人Email地址, null为默认
    寄件人web@vnvtrip.com
58.      * @param bodyText             邮件正文
59.      * @param subject             邮件主题
60.      * @param attachmentName     附件名
61.      * @param resource             附件
62.      * @throws MessagingException
63.      */
64.     public void sendMessage(String[] emailAddresses, String fro
    mEmail,
65.             String bodyText, String subject, String attachmentN
    ame,
66.             ClassPathResource resource) throws MessagingExcepti
    on {
67.         MimeMessage message = ((JavaMailSenderImpl) mailSender)

68.                 .createMimeMessage();
69.
70.         // use the true flag to indicate you need a multipart m
    essage
71.         MimeMessageHelper helper = new MimeMessageHelper(messag
    e, true);
72.
73.         helper.setTo(emailAddresses);
74.         if(fromEmail != null){
75.             helper.setFrom(fromEmail);
76.         }
77.         helper.setText(bodyText, true);
78.         helper.setSubject(subject);
79.
80.         if(attachmentName!=null && resource!=null)
81.             helper.addAttachment(attachmentName, resource);
82.
83.         ((JavaMailSenderImpl) mailSender).send(message);
```

```
84.     }
85.
86.     /**
87.      * 发送简单邮件
88.      * @param msg
89.      */
90.     public void send(SimpleMailMessage msg) {
91.         try {
92.             ((JavaMailSenderImpl) mailSender).send(msg);
93.         } catch (MailException ex) {
94.             //log it and go on
95.             log.error(ex.getMessage());
96.         }
97.     }
98.
99.     /**
100.          * 使用模版发送 HTML 格式的邮件
101.          *
102.          * @param msg           装有 to,from,subject 信息的
    SimpleMailMessage
103.          * @param templateName 模版名,模版根路径已在配置文件定
    义于 freemakarengine 中
104.          * @param model         渲染模版所需的数据
105.          */
106.         public void send(SimpleMailMessage msg, String templat
    eName, Map model) {
107.             //生成 html 邮件内容
108.             String content = generateEmailContent(templateName
    , model);
109.             MimeMessage mimeMsg = null;
110.             try {
111.                 mimeMsg = ((JavaMailSenderImpl) mailSender).cr
    eateMimeMessage();
112.                 MimeMessageHelper helper = new MimeMessageHelp
    er(mimeMsg, true, "utf-8");
113.                 helper.setTo(msg.getTo());
114.
115.                 if(msg.getSubject()!=null)
116.                     helper.setSubject(msg.getSubject());
117.
118.                 if(msg.getFrom()!=null)
119.                     helper.setFrom(msg.getFrom());
120.
121.                 helper.setText(content, true);
```

```
122.
123.                 ((JavaMailSenderImpl) mailSender).send(mimeMsg
    );
124.             } catch (MessagingException ex) {
125.                 log.error(ex.getMessage(), ex);
126.             }
127.
128.         }
129.     }
130.
131.     发送邮件：
132.     SimpleMailMessage message = (SimpleMailMessage) getBean("m
    ailMessage");
133.                     message.setTo(user.getName() + "<" + user.
    getEmail() + ">");
134.
135.                     Map model = new HashMap();
136.                     model.put("user", user);
137.
138.                     MailEngine engine = (MailEngine)getBean("m
    ailEngine");
139.                     //Vilocity 模板
140.                     engine.send(message, "notifyUser.vm", mode
    l);
141.                     //FreeMaker 模板
142.                     //engine.send(message, "NotifyUser.ftl", m
    odel);
143.
144.     以上的 User 为用户类。
```

xml 代码

```
1. 模板：
2. <html>
3. <head>
4. <meta http-equiv="Content-Type" content="text/html;charset=UTF-
   8">
5. <title>用户注册通知 title>
6. head>
7. <body>
8. <p>${user.name} 您好，恭喜您，已经成为本站会员！ p>
9. <table>
```

```
10. <tr><td>用户名：td><td>${user.name}td>tr>
11. <tr><td>密码：td><td>${user.password}td>tr>
12. table>
13. body>
14. html>
```

FreeMaker 开发指南

本文转自福州 IT 信息网 (http://www.fzic.net), 详细出处参考：http://www.fzic.net/SrcShow.asp?Src_ID=1328

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

查看更多关于:FreeMaker　FreeMaker 开发指南
本 文 转 自 福 州　IT　信 息 网 (http://www.fzic.net), 详 细 出 处 参 考 ：
http://www.fzic.net/SrcShow.asp?Src_ID=1328