FreeMarker详解

FreeMarker是一个模板引擎，一个基于模板生成文本输出的通用工具，使用纯Java编写。FreeMarker被设计用来生成HTML Web页面，特别是基于MVC模式的应用程序。虽然FreeMarker具有一些编程的能力，但通常由Java程序准备要显示的数据，由FreeMarker生成页面，通过模板显示准备的数据。FreeMarker不是一个Web应用框架，而适合作为Web应用框架一个组件。FreeMarker与容器无关，因为它并不知道HTTP或Servlet；FreeMar  
ker同样可以应用于非Web应用程序环境。FreeMarker更适合作为Model2框架（如springMVC）的视图组件，你也可以在模板中使用JSP标记库。

# FreeMarker特性

## 通用目标

|  |
| --- |
| 能够生成各种文本：HTML、XML、RTF、Java源代码等等  易于嵌入到你的产品中：轻量级；不需要Servlet环境  插件式模板载入器：可以从任何源载入模板，如本地文件、数据库等等  你可以按你所需生成文本：保存到本地文件；作为Email发送；从Web应用程序发送它返回给Web浏览器 |

## 强大的模板语言

|  |
| --- |
| 所有常用的指令：include、if/elseif/else、循环结构  在模板中创建和改变变量  几乎在任何地方都可以使用复杂表达式来指定值  命名的宏，可以具有位置参数和嵌套内容  名字空间有助于建立和维护可重用的宏库，或者将一个大工程分成模块，而不必担心名字冲突  输出转换块：在嵌套模板片段生成输出时，转换HTML转义、压缩、语法高亮等等；你可以定义自己的转换 |

## 通用数据模型

|  |
| --- |
| FreeMarker不是直接反射到Java对象，Java对象通过插件式对象封装，以变量方式在模板中显示  你可以使用抽象（接口）方式表示对象（JavaBean、XML文档、SQL查询结果集等等），告诉模板开发者使用方法，使其不受技术细节的打扰 |

## 为Web准备

|  |
| --- |
| 在模板语言中内建处理典型Web相关任务（如HTML转义）的结构  能够集成到Model2 Web应用框架中作为JSP的替代  支持JSP标记库  为MVC模式设计：分离可视化设计和应用程序逻辑；分离页面设计员和程序员 |

## 智能的国际化和本地化

|  |
| --- |
| 字符集智能化（内部使用UNICODE）  数字格式本地化敏感  日期和时间格式本地化敏感  非US字符集可以用作标识（如变量名）  多种不同语言的相同模板 |

## 强大的XML处理能力

|  |
| --- |
| <#recurse> 和<#visit>指令（2.3版本）用于递归遍历XML树。在模板中清楚和直觉的访问XML对象模型。开源论坛 JForum 就是使用了 FreeMarker 做为页面模板。 |

# java 程序解析FreeMarker模板

第一步： 建立一个普通的java项目：testFreeMarker

第二步：引入freemarker.jar包

第三步：在项目目录下建立模板目录：templates

第四步：在templates目录下，建立a.ftl模板文件，内容如下：

|  |
| --- |
| 你好啊，${user}，今天你的精神不错！ |

第五步：建立com.test.freemarker包，然后建立Test1.java文件，内容如下：

|  |
| --- |
| public class Test1 {  public static void main(String[] args) throws Exception {  //创建Freemarker配置实例  Configuration cfg = new Configuration();  cfg.setDirectoryForTemplateLoading(new File("templates"));    //创建数据模型  Map root = new HashMap();  root.put("user", "老高");    //加载模板文件  Template t1 = cfg.getTemplate("a.ftl");  //显示生成的数据,//将合并后的数据打印到控制台  Writer out = new OutputStreamWriter(System.out);  t1.process(root, out);  out.flush();    //显示生成的数据,//将合并后的数据直接返回成字符串！  // StringWriter out = new StringWriter();  // t1.process(root, out);  // out.flush();  // String temp = out.toString();  // System.out.println(temp); }  } |

# FreeMarker数值和数据类型

## 直接指定值

直接指定值可以是字符串、数值、布尔值、集合及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。字符串中可以使用转义字符"\"。如果字符串内有大量的特殊字符，则可以在引号的前面加上一个字母r，则字符串内的所有字符都将直接输出。${r"$good''"})  **#2. 数值**  数值可以直接输入，不需要引号。FreeMarker不支持科学计数法。  **#3. 布尔值**  直接使用true或false，不使用引号。  **#4. 集合**  集合用中括号包括，集合元素之间用逗号分隔。  使用数字范围也可以表示一个数字集合，如1..5等同于集合[1, 2, 3, 4, 5]；同样也可以用5..1来表示[5, 4, 3, 2, 1]。  **#5. Map对象**  Map对象使用花括号包括，Map中的key-value对之间用冒号分隔，多组key-value对之间用逗号分隔。  注意：Map对象的key和value都是表达式，但key必须是字符串。    **#6. 时间对象**  root.put("date1", new Date());  ${date1?string("yyyy-MM-dd HH:mm:ss")} |

## 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。

|  |
| --- |
| **#1. 顶层变量**  所谓顶层变量就是直接放在数据模型中的值。输出时直接用${variableName}即可。  **#2. javaBean**  Freemarker中对于javabean的处理跟EL表达式一致，类型可自动转化！非常方便！${bean.property},${bean["property"]}  **#3. 输出集合元素**  可 以根据集合元素的索引来输出集合元素，索引用中括号包括。如： 输出["1"， "2"， "3"]这个名为number的集合，可以用${number[0]}来输出第一个数字。FreeMarker还支持用number[1..2]来表示原 集合的子集合["2"， "3"]。  **#4. 输出Map元素**  对于JavaBean实例，FreeMarker一样把它看作属性为key，属性值为value的Map对象。输出Map对象时，可以使用点语法或中括号语法，如下面的几种写法的效果是一样的：  book.author.name  book.author["name"]  book["author"].name  book["author"]["name"]  **注意**：使用点语法时，变量名字有和顶层变量一样的限制，但中括号语法没有任何限制。 |

## 字符串操作

|  |
| --- |
| **#1. 字符串连接**  字符串连接有两种语法：  （1）使用${..}或#{..}在字符串常量内插入表达式的值；  （2)） 直接使用连接运算符"+"连接字符串。  如，下面两种写法等效： ${"Hello,${user}"} ， ${"Hello, " + user +"!"}    注意： ${..}只能用于文本部分作为插值输出，而不能用于比较等其他用途，如：  <#if ${isBig}>Wow!</#if>  <#if"${isBig}">Wow!</#if>  应该写成：  <#ifisBig>Wow!</#if>  **#2. 截取子串**  截取子串可以根据字符串的索引来进行，如果指定一个索引值，则取得字符串该索引处的字符；如果指定两个索引值，则截取两个索引中间的字符串子串。如：  <#assign number="01234">  ${number[0]} <#-- 输出字符0-->  ${number[0..3]}<#-- 输出子串"0123"--> |

# FreeMarker运算符

## 算术运算符

FreeMarker表达式中支持"+"、"－"、"\*"、"/"、"%"运算符。(除法保留小数)

## 连接运算符

+：可用于字符，集合，map的连接。

## 比较运算符

表达式中支持的比较运算符有如下几种：

|  |
| --- |
| **=（或者==）：**判断两个值是否相等；  **!=**：判断两个值是否不相等；  注： =和!=可以用作字符串、数值和日期的比较，但两边的数据类型必须相同。而且FreeMarker的比较是精确比较，不会忽略大小写及空格。  **>（或者gt）：**大于  **>=（或者gte）：**大于等于  **<（或者lt）：**小于  **<=（或者lte）：**小于等于  注： 上面这些比较运算符可以**用于数字和日期**，但不能用于字符串。大部分时候，使用gt比>有更好的效果，因为FreeMarker会把>解释成标签的结束字符。可以使用括号来避免这种情况，如：<#if (x>y)>。  if else 语句测试：  <#if num0 gt 18> <#--不是使用>，大部分时候，freemarker会把>解释成标签结束！ -->  及格！  <#else>  不及格！  </#if>  root.put("num0", 18); |

## 逻辑运算符

|  |
| --- |
| 1. &&： 逻辑与；  2. ||：逻辑或；  3. !：逻辑非  注意：逻辑运算符只能用于布尔值。 |

## 空值处理运算符

FreeMarker的变量必须赋值，否则就会抛出异常。而对于FreeMarker来说，null值和不存在的变量是完全一样的，因为FreeMarker无法理解null值。

FreeMarker提供两个运算符来避免空值：

* !：指定缺失变量的默认值；
* ??：判断变量是否存在。

**!运算符**有两种用法：variable!或variable!defaultValue。第一种用法不给变量指定默认值，表明默认值是空字符串、长度为0的集合、或长度为0的Map对象。使用!运算符指定默认值并不要求默认值的类型和变量类型相同。

|  |
| --- |
| 测试空值处理：  ${var} <#--没有定义var变量，会报异常！ -->  ${sss!} <#--没有定义var变量，默认值是空字符串！ -->  ${sss!"abc"} <#--没有定义var变量，默认值是字符串abc！ --> |

**??运算符**返回布尔值，如：variable??，如果变量存在，返回true，否则返回false。多层次调用使用：**(product.color)??**

|  |
| --- |
| **<#if mouse??>**  Mouse found  <#else>  No mouse found  </#if> |

## 运算符的优先级

由上往下，优先级减小。

|  |  |
| --- | --- |
| **运算符说明** | **运算符** |
| 最高优先级运算符 | [subvarName][subStringRange].?(methodParams) expr! expr?? |
| 一元前缀运算符 | +expr -expr !expr |
| 乘除法，求模 | \* / % |
| 加减法 | + - |
| 关系运算符 | < > <= >= (相当于: gt, lt, 等) |
| 相等，不等 | == (也可以是: =) != |
| 逻辑与 | && |
| 逻辑或 | || |
| 数字范围 | .. |

# FreeMarker内建函数

FreeMarker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转换输出变量。字符串相关常用的内建函数：

|  |
| --- |
| **#1. 字符串内建函数**  **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**：将字符串转成小写；  **upper\_case**：将字符串转成大写；  **#2 . 集合常用的内建函数**：  **size**：获得集合中元素的个数；  **#3. 数字值常用的内建函数**  **int**：取得数字的整数部分。  举例：  ${test?html} // Tom &amp; Jerry |

# FreeMarker模板结构

实际上你用程序语言编写的程序就是模板，模板也被称为FTL（代表FreeMarker模板语言）。这是为编写模板设计的非常简单的编程语言。

模板（FTL编程）是由如下部分混合而成的：

* **Text文本**：文本会照着原样来输出。
* **Interpolation插值**：这部分的输出会被计算的值来替换。插值由${和}所分隔（或者#{和}，这种风格已经不建议再使用了）。
* **FTL tags标签**：FTL标签和HTML标签很相似，但是它们却是给FreeMarker的指示，而且不会打印在输出内容中。
* **Comments注释**：FTL的注释和HTML的注释也很相似，但它们是由<#--和-->来分隔的。注释会被FreeMarker所忽略，更不会在输出内容中显示。

|  |
| --- |
| <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> |

## Interpolation插值

插值的使用语法是：${expression}，expression可以是所有种类的表达式（比如${100 + x}）。插值是用来给插入具体值然后转换为文本（字符串）。插值仅仅可以在两种位置使用：

* 文本区（如<h1>Hello ${name}!</h1>）
* 字符串表达式（如<#include "/footer/${company}.html">）中。

注意：变量可以直接在标签中使用，不用插值。典型的错误就是<#if ${isBig}>Wow!</#if>，这是语法上的错误。只要写为<#if isBig>Wow!  
</#if>就对了，而且<#if "${isBig}">Wow!</#if>也是错误的，因为这样参数就是字符串类型了，但是if指令的参数要求是布尔值，所以运行时就会发生错误。插值表达式的结果必须是字符串，数字或日期类型的，因为只有数字和日期类型可以自动转换为字符串类型，其他类型的值（如布尔，序列）只能手动转换为字符串类型，否则就会发生错误导致模板执行中止。

## FreeMarker标签

|  |
| --- |
| #1. if指令  <#if random gte 90>  优秀！  <#elseif random gte 80>  良好！  <#else>  一般！  </#if>  #2. list指令  <ul>  **<#list animals as being>**  <li>${being.name} for ${being.price} Euros  **</#list>**  </ul>  #3. include指令  <#include "foot.tl" />  #4. assign自定义变量指令  <#assign x = 1> <#-- 创建变量 x -->  ${x}  #5. 自定义指令(macro指令)  **注意**：替换（方括号）：这个特性从FreeMarker 2.3.4版本后才可用。FreeMarker支持一个替换的语法。就是在FreeMarker的指令和注释中用[和]来代替<  **调用预定义指令**：[#list animals as being]...[/#list] **注释**：[#-- the comment --] |