FreeMarker是一款[模板引擎](https://baike.baidu.com/item/%E6%A8%A1%E6%9D%BF%E5%BC%95%E6%93%8E/907667)： 即一种基于模板和要改变的数据， 并用来生成输出文本（[HTML](https://baike.baidu.com/item/HTML)网页、[电子邮件](https://baike.baidu.com/item/%E7%94%B5%E5%AD%90%E9%82%AE%E4%BB%B6/111106)、[配置文件](https://baike.baidu.com/item/%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6/286550)、[源代码](https://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81/3969)等）的通用工具。 它不是面向最终用户的，而是一个Java类库，是一款程序员可以嵌入他们所开发产品的组件。

freemarker的基本语法及入门基础  
  
一、freemarker模板文件(\*.ftl)的基本组成部分  
        1. 文本：直接输出的内容部分  
        2. 注释：不会输出的内容，格式为<#--  注释内容  -->  
        3. 取值(插值)：代替输出数据模型的部分，格式为${数据模型}或#{数据模型}  
        4. ftl指令：Freemarker指令，类似于HTML标记。  
                             内建指令：开始标签:<#directivename parameter>       结束标签:</#directivename>      空标签:<#directivename parameter/>     
                             自定义指令：开始标签:<@directivename parameter>       结束标签:</@directivename>      空标签:<@directivename parameter/>   
                             至于什么是内建指令，什么是自定义指令   我会在下面叙述到。

  
  
二、Freemarker语法及使用方法  
        1. 取值(插值)指令及适用类型：  
                (1) ${var}  
                        适用类型：java中常用的八大基本类型以及我们的String引用类型，但是，freemarker中boolean类型显示时true==yes  false==no  
                        示例：  
                                在后台文件中定义变量  
                                        String strVar = "世界你好";  
                                        int intVar = 10;  
                                        boolean booVar = true;  
                                在页面中获取变量：  
                                        String获取：<font color="red"> ${strVar} </font><br>  
                                        int获取：<font color="red"> ${intVar} </font><br>  
                                        boolean获取：<font color="red"> ${booVar?string("yes","no")} </font>  
                                展示结果：  
                                        String获取：世界你好  
                                        int获取：10  
                                        boolean获取：yes  
                (2)${var!}   
                        适用类型：对 null 或者不存在的对象进行取值，可以设置默认值，例：${var!'我是默认值'}    即，有值时显示正常值，无值时显示默认值  
                        示例：  
                                在后台文件中定义变量  
                                        String strVar = "世界你好";  
                                        String str = null;  
                                在页面中获取变量：  
                                        String获取：<font color="red"> ${strVar!"我是空"} </font><br>  
                                        str获取：<font color="red"> ${str!} </font><br>  
                                        str获取：<font color="red"> ${str!"默认"} </font><br>  
                                展示结果：  
                                        String获取：世界你好  
                                        str获取：  
                                        str获取：默认  
                (3)${封装对象.属性}  
                        适用类型：对封装对象进行取值，例：${User.name}  
                        示例：  
                                在后台文件中封装对象User[ name,  age ]  
                                        String name = "姓名";  
                                        int age = 18;  
                                在页面中获取变量：  
                                        name获取：<font color="red"> ${User.name} </font><br>  
                                        age获取：<font color="red"> ${User.age} </font><br>  
                                展示结果：  
                                        name获取：姓名  
                                        age获取：18  
                (4)${date?String('yyyy-MM-dd')}  
                        适用类型：对日期格式进行取值，在这里我要强调的是，定义Date类型的变量时，java.util.Date无法输出日期，须使用java.sql.Date  
                        示例：  
                                在后台文件中定义变量  
                                        java.sql.Date date = new Date().getTime();  
                                        java.sql.Date time = new Date().getTime();  
                                        java.sql.Date datetime = new Date().getTime();  
                                在页面中获取变量：  
                                        date获取：<font color="red"> ${date?string('yyyy-MM-dd')} </font><br>  
                                        time获取：<font color="red"> ${date?string('HH:mm:ss')} </font><br>  
                                        datetime获取：<font color="red"> ${date?string('yyyy-MM-dd HH:mm:ss')} </font><br>  
                                展示结果：  
                                        name获取：姓名  
                                        age获取：18  
                (5)${var?html}  
                        适用类型：转义HTML内容  
                        示例：  
                                在后台文件中封装变量Menu[ name, model ]  
                                        Menu m = new Menu();   
                                        m.setName(" freemarker ");  
                                        m.setModel("<font color = 'red'>我只是个菜单</font>");  
                                在页面中获取变量：  
                                        非转义获取：<font color="red"> ${m.model} </font><br>  
                                        转义获取： ${m.model?html} </font><br>  
                                展示结果：  
                                        非转义获取：我只是个菜单  
                                        转义获取：<font color = 'red'>我只是个菜单</font>  
                (6)<#assign num = 100 />  
                        适用类型：定义变量，支持计算和赋值  
                        示例：  
                                在页面中定义变量：  
                                        <#assign num = 100 />  
                                        num获取：<font color="red"> ${num)} </font><br>  
                                        计算结果：<font color="red"> ${num \* 10} </font><br>  
                                展示结果：  
                                        num获取：100  
                                        计算结果：1000  
                (7)对List集合进行取值  
                        <#list  list集合  as  item>   
                               ${item}    --取值  
                        </#list>  
                        示例：  
                                在后台文件中定义变量  
                                        List<String> strList = new ArrayList<String>();  
                                        strList.add("第一个值");  
                                        strList.add("第二个值");  
                                        strList.add("第三个值");  
                                在页面中获取变量：  
                                        <#list  strList  as  item>   
                                               ${item!}<br/>    --取值  
                                        </#list>  
                                展示结果：  
                                        第一个值  
                                        第二个值  
                                        第三个值  
                (8)对Map集合进行取值  
                        <#list map?keys as key>  
                               ${key}:${map[key]}  
                        </#list>  
                        示例：  
                                在后台文件中定义变量  
                                        Map<String, Object> m = new HashMap<String, Object>();  
                                        m.put("name","姓名");  
                                        m.put("age",18);  
                                        m.put("sex","男");  
                                在页面中获取变量：  
                                         <#list m?keys as key>  
                                                ${key}:${m[key]}  
                                         </#list>  
                                展示结果：  
                                        name：姓名  
                                        age：18  
                                        sex：男  
        2. 条件判断指令：  
                (1) if   
                        格式：<#if 条件>  
                                        输出  
                                     </#if>  
                        示例：  
                                在页面中定义变量并判断条件：  
                                        <#assign age = 18 /><br>  
                                        <#if age == 18>  
                                                <font color="red"> age = 18</font>  
                                        </#if>  
                                展示结果：  
                                        age = 18  
                (2) if - else   
                        格式：<#if 条件>  
                                        输出  
                                    <#else>  
                                        输出  
                                    </#if>  
                        示例：  
                                在页面中定义变量并判断条件：  
                                        <#assign age = 20 /><br>  
                                        <#if age == 18>  
                                                <font color="red"> age = 18</font>  
                                        <#else>  
                                                <font color="red"> age != 18</font>  
                                        </#if>  
                                展示结果：  
                                        age != 18  
                (3) if - elseif - else  
                        格式：<#if 条件1>  
                                        输出  
                                    <#elseif 条件2>  
                                        输出  
                                    <#else>  
                                        输出  
                                    </#if>  
                        示例：  
                                在页面中定义变量并判断条件：  
                                        <#assign age = 20 /><br>  
                                        <#if age &gt; 18>  
                                                <font color="red">青年</font>  
                                        <#elseif age == 18>  
                                                <font color="red"> 成年</font>  
                                        <#else>  
                                                <font color="red"> 少年</font>  
                                        </#if>  
                                展示结果：  
                                        成年  
                (4) switch  --常与case break default一起使用  参数可为字符串  
                        格式：<#switch var>  
                                    <#case 条件1>  
                                          输出  
                                    <#break>  
                                    <#case 条件2>  
                                          输出  
                                    <#break>  
                                    <#default>  
                                          输出  
                                    </#switch>  
                        示例：  
                                在页面中定义变量并判断：  
                                       <#switch var="星期一">  
                                       <#case "星期一">  
                                               油焖大虾  
                                       <#break>  
                                       <#case "星期二">  
                                               炸酱面  
                                       <#break>  
                                       <#default>  
                                               肯德基  
                                       </#switch>  
                                展示结果：  
                                        油焖大虾  
        3. 自定义函数、自定义指令：  
                (1) 自定义函数  
                        实现TemplateMthodModelEx  
                (2) 自定义指令  
                        实现TemplateDirectiveModel  
                        示例：  
                                <@自定义指令名称 入参(key-value格式) ; 出参(list格式)>  
                                        运行条件  
                                </@自定义指令名称>  
  
                                PS:不同的返回值用逗号( , )间隔开  
  
        4.常用内建函数、macro(宏指令)、function(函数指令)：  
                (1) 常用内建函数  
                        处理字符串：   
                             substring                                          截取字符串，包头不包尾（下标）  
                             cap\_first                                          第一个字母大写   
                             end\_with                                           以什么字母结尾      
                             contains                                            是否包含目标字符串  
                             date  datetime  time                       转换成日期格式  
                             starts\_with                                      以什么字母开头  
                             index\_of                                          返回某个指定的字符串值在字符串中首次出现的位置（下标）  
                             last\_index\_of                                  获取指定字符出现的最后位置（下标）  
                             split                                                  分隔  
                             trim                                                  去两端空格  
                        处理数字：  
                             string                                                
                             x?string("0.##")                          变成小数点后几位  
                             round                                              四舍五入  
                             floor                                               去掉小数点  
                             ceiling                                             近1   变成整数  
                        处理list:  
                              first:                                              取List值第一个值  
                              last:                                                取List值最后一个值  
                              seq\_contains:                                是否包含指定字符  
                              seq\_index\_of:                               指定字符所在位置  
                              size:                                                集合大小  
                               reverse:                                          集合倒序排列  
                              sort:                                                对集合进行排序  
                              sort\_by:                                         根据某一个属性排序  
                              chunk:                                            分块处理  
                        其他:  
                              is\_string:                                      是否为字符类型  
                              is\_number:                                    是否为整数类型  
                              is\_method:                                   是否为方法  
                              ():                                                  判断整个变量  
                              has\_content:                                判断对象是否为空或不存在  
                              eval：                                           求值  
                (2) macro(宏指令)  
                        调用：<@macro\_name param />  
                        语法：<#macro  变量名  参数>    
                                            <#nested/>    
                                    </#macro>  
                (3) function(函数指令)  
                        调用：${function\_name(param)}  
                        语法：<#function  变量名  参数>    
                                    <#return>    
                                    </#function>

FTL指令常用标签及语法

注意：使用freemaker，要求所有标签必须闭合，否则会导致freemaker无法解析。

freemaker注释:<#-- 注释内容 -->格式部分,不会输出

---------------------------------- 基础语法 ----------------------------------

1、字符输出

${emp.name?if\_exists}　　　　　　// 变量存在，输出该变量，否则不输出

${emp.name!}　　　　　　　　　　 // 变量存在，输出该变量，否则不输出

${emp.name?default("xxx")} // 变量不存在，取默认值xxx

${emp.name!"xxx"} 　　　　　　// 变量不存在，取默认值xxx

常用内部函数：

${"123<br>456"?html} 　　// 对字符串进行HTML编码，对html中特殊字符进行转义

${"str"?cap\_first} 　　 // 使字符串第一个字母大写

${"Str"?lower\_case} // 将字符串转换成小写

${"Str"?upper\_case} // 将字符串转换成大写

${"str"?trim} // 去掉字符串前后的空白字符

字符串的两种拼接方式拼接：

${"hello${emp.name!}"} // 输出hello+变量名

${"hello"+emp.name!} // 使用+号来连接，输出hello+变量名

可以通过如下语法来截取子串:

<#assign str = "abcdefghijklmn"/>

// 方法1

${str?substring(0,4)} // 输出abcd

// 方法2

${str[0]}${str[4]} // 结果是ae

${str[1..4]} 　　　 // 结果是bcde

// 返回指定字符的索引

${str?index\_of("n")}

2、日期输出

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

3、数字输出(以数字20为例)

${emp.name?string.number} 　// 输出20

${emp.name?string.currency} // ￥20.00

${emp.name?string.percent} // 20%

${1.222?int} 　　　　　　　　　　 // 将小数转为int，输出1

<#setting number\_format="percent"/> // 设置数字默认输出方式('percent',百分比)

<#assign answer=42/> 　　　　　　　　 // 声明变量 answer 42

#{answer} 　　　　　　　　 // 输出 4,200%

${answer?string} 　　　　 // 输出 4,200%

${answer?string.number} 　　// 输出 42

${answer?string.currency} // 输出 ￥42.00

${answer?string.percent} 　// 输出 4,200%

#{answer} 　　　　　　　　// 输出 42

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2.58

#{y; m2} // 输出4.0

#{x; m1M2} // 输出2.58

#{x; m1M2} // 输出4.0

4、申明变量

<#assign foo=false/> // 声明变量,插入布尔值进行显示,注意不要用引号

${foo?string("yes","no")} // 当为true时输出"yes",否则输出"no"

申明变量的几种方式

<#assign name=value>

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

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

<#assign name>

capture this

</#assign>

<#assign name in namespacehash>

capture this

</#assign>

5、比较运算符

表达式中支持的比较运算符有如下几个:

= 或 == ：判断两个值是否相等.

!= ：判断两个值是否不等.

> 或 gt ：判断左边值是否大于右边值

>= 或 gte ：判断左边值是否大于等于右边值

< 或 lt ：判断左边值是否小于右边值

<= 或 lte ：判断左边值是否小于等于右边值

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:+, - , \* , / , %

注意：

（1）、运算符两边必须是数字

（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

7、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

8、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

9、if 逻辑判断（注意：elseif 不加空格）

<#if condition>

...

<#elseif condition2>

...

<#elseif condition3>

...

<#else>

...

</#if>

if 空值判断

// 当 photoList 不为空时

<#if photoList??>...</#if>

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>

// 正确写法

<#if isBig>Wow!</#if>

10、switch (条件可为数字，可为字符串)

<#switch value>

<#case refValue1>

....

<#break>

<#case refValue2>

....

<#break>

<#case refValueN>

....

<#break>

<#default>

....

</#switch>

11、集合 & 循环

// 遍历集合:

<#list empList! as emp>

${emp.name!}

</#list>

// 可以这样遍历集合:

<#list 0..(empList!?size-1) as i>

${empList[i].name!}

</#list>

// 与jstl循环类似,也可以访问循环的状态。

empList?size 　　　// 取集合的长度

emp\_index: 　　　　// int类型，当前对象的索引值

emp\_has\_next: // boolean类型，是否存在下一个对象

// 使用<#break>跳出循环

<#if emp\_index = 0><#break></#if>

// 集合长度判断

<#if empList?size != 0></#if> // 判断=的时候,注意只要一个=符号,而不是==

<#assign l=0..100/> // 定义一个int区间的0~100的集合，数字范围也支持反递增,如100..2

<#list 0..100 as i> 　　// 等效于java for(int i=0; i <= 100; i++)

　　${i}

</#list>

// 截取子集合：

empList[3..5] //返回empList集合的子集合,子集合中的元素是empList集合中的第4-6个元素

// 创建集合：

<#list ["星期一", "星期二", "星期三", "星期四", "星期五", "星期六", "星期天"] as x>

// 集合连接运算,将两个集合连接成一个新的集合

<#list ["星期一","星期二","星期三"] + ["星期四","星期五","星期六","星期天"] as x>

// 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"]

// seq\_contains：判断序列中的元素是否存在

<#assign x = ["red", 16, "blue", "cyan"]>

${x?seq\_contains("blue")?string("yes", "no")} // yes

${x?seq\_contains("yellow")?string("yes", "no")} // no

${x?seq\_contains(16)?string("yes", "no")} // yes

${x?seq\_contains("16")?string("yes", "no")} // no

// seq\_index\_of：第一次出现的索引

<#assign x = ["red", 16, "blue", "cyan", "blue"]>

${x?seq\_index\_of("blue")} // 2

// sort\_by：排序（升序）

<#list movies?sort\_by("showtime") as movie></#list>

// sort\_by：排序（降序）

<#list movies?sort\_by("showtime")?reverse as movie></#list>

// 具体介绍：

// 不排序的情况：

<#list movies as moive>

　　<a href="${moive.url}">${moive.name}</a>

</#list>

//要是排序，则用

<#list movies?sort as movie>

　　<a href="${movie.url}">${movie.name}</a>

</#list>

// 这是按元素的首字母排序。若要按list中对象元素的某一属性排序的话，则用

<#list moives?sort\_by(["name"]) as movie>

　　<a href="${movie.url}">${movie.name}</a>

</#list>

//这个是按list中对象元素的[name]属性排序的，是升序，如果需要降序的话，如下所示：

<#list movies?sort\_by(["name"])?reverse as movie>

　　<a href="${movie.url}">${movie.name}</a>

</#list>

12、Map对象

// 创建map

<#assign scores = {"语文":86,"数学":78}>

// Map连接运算符

<#assign scores = {"语文":86,"数学":78} + {"数学":87,"Java":93}>

// Map元素输出

emp.name // 全部使用点语法

emp["name"] // 使用方括号

13、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}"} // 输出 ${foo}

${r"C:/foo/bar"} // 输出 C:/foo/bar

14、include指令

// include指令的作用类似于JSP的包含指令:

<#include "/test.ftl" encoding="UTF-8" parse=true>

// 在上面的语法格式中,两个参数的解释如下:

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"

15、import指令

// 类似于jsp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

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

// 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，"my"在freemarker里被称作namespace

17、compress 压缩

// 用来压缩空白空间和空白的行

<#compress>

...

</#compress>

<#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

18、escape,noescape 对字符串进行HTML编码

// 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 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}

---------------------------------- 高级语法 ----------------------------------

1、global全局赋值语法

<#global name=value>

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

<#global name>

　　capture this

</#global>

// 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${.globals.x} 来访问

2、setting 语法

// 用来设置整个系统的一个环境

locale // zh\_CN 中文环境

number\_format

boolean\_format

date\_format , time\_format , datetime\_format

time\_zone

classic\_compatible

// 例1：

<#setting number\_format="percent"/> // 设置数字默认输出方式('percent',百分比)

// 例2：

// 假如当前是匈牙利的设置，然后修改成美国

${1.2} // 输出1,2

<#setting locale="en\_US">

${1.2} // 输出1.2,因为匈牙利是采用", "作为十进制的分隔符，美国是用". "

3、macro宏指令

例子1：

<#-- 定义宏 -->

<#macro test foo bar="Bar" baaz=-1>

　　Text: ${foo}, ${bar}, ${baaz}

</#macro>

<#-- 使用宏 -->

<@test foo="a" bar="b" baaz=5\*5/> // 输出：Text: a, b, 25

<@test foo="a" bar="b"/> 　　　　// 输出：Text: a, b, -1

<@test foo="a" baaz=5\*5-2/> 　　　　// 输出：Text: a, Bar, 23

<@test foo="a"/> // 输出：Text: a, Bar, -1

例子2：

<#-- 定义一个循环输出的宏 -->

<#macro list title items>

　　${title}

　　<#list items as x>

　　　　\*${x}

　　</#list>

</#macro>

<#-- 使用宏 -->

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

// 输出Animals \*mouse \*elephant \*python

例子3：

<#-- 嵌套宏 -->

<#macro border>

　　<table>

　　　　<#nested>

　　</table>

</#macro>

<#-- 嵌套宏使用 -->

<@border>

　　<tr><td>hahaha</td></tr>

</@border>

输出结果：

<table>

　　<tr><td>hahaha</td></tr>

</table>

例子4：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:

<#-- 循环嵌套宏 -->

<#macro repeat count>

　　<#list 1..count as x>

　　　　<#nested x, x/2, x==count> // 使用nested指令时指定了三个循环变量

　　</#list>

</#macro>

<#-- 使用宏 -->

<@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!

freemarker 宏嵌套nested 的使用:

http://blog.sina.com.cn/s/blog\_7e5699790100z59g.html

4、结束macro指令

// return指令用于结束macro指令

<#-- 创建宏 -->

<#macro book>

　　spring

　　<#return>

　　j2ee

</#macro>

<#-- 使用宏 -->

<@book />

// 上面的代码输出:spring,而j2ee位于return指令之后,不会输出.