

# Freemarker研究

## 1 FreeMarker 研究

### 1.1 FreeMarker介绍

1、freemarker是一个用Java开发的模板引擎

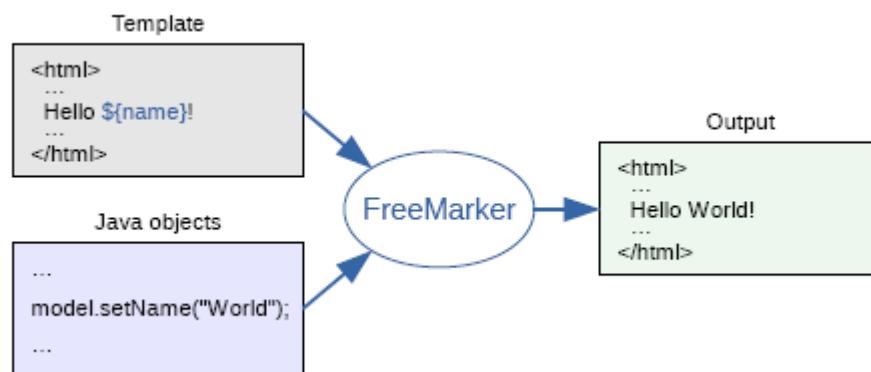
 |  收藏 |  861 |  91

### freemarker

 编辑

FreeMarker是一款模板引擎：即一种基于模板和要改变的数据，并用来生成输出文本（[HTML网页](#)、[电子邮件](#)、[配置文件](#)、[源代码](#)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是免费的，基于Apache许可证2.0版本发布。其模板编写为FreeMarker Template Language（FTL），属于简单、专用的语言。需要准备数据在真实编程语言中来显示，比如[数据库](#)查询和业务运算，之后模板显示已经准备好的数据。在模板中，主要用于如何展现数据，而在模板之外注意于要展示什么数据 [\[1\]](#)。



常用的java模板引擎还有哪些？

[Jsp](#)、[Freemarker](#)、[Thymeleaf](#)、[Velocity](#) 等。

### 2、模板+数据模型=输出

freemarker并不关心数据的来源，只是根据模板的内容，将数据模型在模板中显示并输出文件（通常为html，也可以生成其它格式的文本文件）

#### 1、数据模型

数据模型在java中可以是基本类型也可以List、Map、Pojo等复杂类型。

2、来自官方的例子：([https://freemarker.apache.org/docs/dgui\\_quickstart\\_basics.html](https://freemarker.apache.org/docs/dgui_quickstart_basics.html))

数据模型：



## DATA MODEL

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

模板：

## TEMPLATE

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

输出：

## OUTPUT

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

## 1.2 FreeMarker快速入门

freemarker作为springmvc一种视图格式，默认情况下SpringMVC支持freemarker视图格式。

需要创建Spring Boot+Freemarker工程用于测试模板。

### 1.2.1 创建测试工程

创建一个freemarker 的测试工程专门用于freemarker的功能测试与模板的测试。



pom.xml如下

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>xc-framework-parent</artifactId>
        <groupId>com.xuecheng</groupId>
        <version>1.0-SNAPSHOT</version>
        <relativePath>../xc-framework-parent/pom.xml</relativePath>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>test-freemarker</artifactId>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-freemarker</artifactId>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <dependency>
            <groupId>com.squareup.okhttp3</groupId>
            <artifactId>okhttp</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-io</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
    </dependencies>
</project>
```

## 1.2.2 配置文件

配置application.yml和logback-spring.xml，从cms工程拷贝这两个文件，进行更改，logback-spring.xml无需更改，application.yml内容如下：



```
server:  
  port: 8088 #服务端口  
  
spring:  
  application:  
    name: test-freemarker #指定服务名  
  freemarker:  
    cache: false #关闭模板缓存，方便测试  
    settings:  
      template_update_delay: 0 #检查模板更新延迟时间，设置为0表示立即检查，如果时间大于0会有缓存不方便  
      进行模板测试
```

### 1.2.3 创建模型类

在freemarker的测试工程下创建模型类型用于测试

```
package com.xuecheng.test.freemarker.model;  
  
import lombok.Data;  
import lombok.ToString;  
  
import java.util.Date;  
import java.util.List;  
  
@Data  
@ToString  
public class Student {  
  private String name;//姓名  
  private int age;//年龄  
  private Date birthday;//生日  
  private Float money;//钱包  
  private List<Student> friends;//朋友列表  
  private Student bestFriend;//最好的朋友  
}
```

### 1.2.3 创建模板

在 src/main/resources下创建templates，此目录为freemarker的默认模板存放目录。[springBoot中默认存放目录](#)

在templates下创建模板文件test1.ftl，模板中的\${name}最终会被freemarker替换成具体的数据。



```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World!</title>
</head>
<body>
Hello ${name}!
</body>
</html>
```

## 1.2.4 创建controller

创建Controller类，向Map中添加name，最后返回模板文件。

```
package com.xuecheng.test.freemarker.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.client.RestTemplate;

import java.util.Map;

@RequestMapping("/freemarker")
@Controller
public class FreemarkerController {
    @Autowired
    RestTemplate restTemplate;

    @RequestMapping("/test1")
    public String freemarker(Map<String, Object> map){
        map.put("name", "黑马程序员");
        //返回模板文件名称
        return "test1";
    }
}
```

## 1.2.5 创建启动类

```
@SpringBootApplication
public class FreemarkerTestApplication {
    public static void main(String[] args) {
        SpringApplication.run(FreemarkerTestApplication.class, args);
    }
}
```



## 1.2.6 测试

调用模板文件，渲染成html文件，在页面上显示，但是还没有生成静态化文件，仅仅在内存中，类似jsp

请求：<http://localhost:8088/freemarker/test1>

屏幕显示：Hello 黑马程序员！

## 1.3 FreeMarker基础

### 1.3.1 核心指令

#### 1.3.1.1 数据模型

Freemarker静态化依赖数据模型和模板，下边定义数据模型：

下边方法形参map即为freemarker静态化所需要的数据模型，在map中填充数据：

```
@RequestMapping("/test1")
public String freemarker(Map<String, Object> map){
    //向数据模型放数据
    map.put("name", "黑马程序员");
    Student stu1 = new Student();
    stu1.setName("小明");
    stu1.setAge(18);
    stu1.setMondy(1000.86f);
    stu1.setBirthday(new Date());
    Student stu2 = new Student();
    stu2.setName("小红");
    stu2.setMondy(200.1f);
    stu2.setAge(19);
    // stu2.setBirthday(new Date());
    List<Student> friends = new ArrayList<>();
    friends.add(stu1);
    stu2.setFriends(friends);
    stu2.setBestFriend(stu1);
    List<Student> stus = new ArrayList<>();
    stus.add(stu1);
    stus.add(stu2);
    //向数据模型放数据
    map.put("stus", stus);
    //准备map数据
    HashMap<String, Student> stuMap = new HashMap<>();
    stuMap.put("stu1", stu1);
    stuMap.put("stu2", stu2);
    //向数据模型放数据
    map.put("stu1", stu1);
    //向数据模型放数据
    map.put("stuMap", stuMap);
    //返回模板文件名称
    return "test1";
}
```



### 1.3.1.2 List指令

本节定义freemarker模板，模板中使用freemarker的指令，关于freemarker的指令需要知道：

- 1、注释，即`<!--和-->`，介于其之间的内容会被freemarker忽略
- 2、插值 ( Interpolation )：即`${..}`部分，freemarker会用真实的值代替`${..}`
- 3、FTL指令：和HTML标记类似，名字前加#予以区分，Freemarker会解析标签中的表达式或逻辑。
- 4、文本，仅文本信息，这些不是freemarker的注释、插值、FTL指令的内容会被freemarker忽略解析，直接输出内容。

在test1.ftl模板中使用list指令遍历数据模型中的数据：

```
<table>
    <tr>
        <td>序号</td>
        <td>姓名</td>
        <td>年龄</td>
        <td>钱包</td>
    </tr>
    <#list stus as stu>
        <tr>
            <td>${stu_index + 1}</td>
            <td>${stu.name}</td>
            <td>${stu.age}</td>
            <td>${stu.mondy}</td>
        </tr>
    </#list>
</table>
```

3、输出：

Hello 黑马程序员! 序号 姓名 年龄 钱包 1 小明 18 1,000.86 2 小红 19 200.1

说明：

\_index：得到循环的下标，使用方法是在stu后边加“\_index”，它的值是从0开始

### 1.3.1.3 遍历Map数据

1、数据模型

使用map指令遍历数据模型中的stuMap。

2、模板

输出stu1的学生信息：  
姓名 : \${stuMap['stu1'].name}  
年龄 : \${stuMap['stu1'].age}

类似json



```
输出stu1的学生信息：  
姓名：${stuMap.stu1.name}<br/>  
年龄：${stuMap.stu1.age}<br/>  
遍历输出两个学生信息。  
<table>  
    <tr>  
        <td>序号</td>  
        <td>姓名</td>  
        <td>年龄</td>  
        <td>钱包</td>  
    </tr>  
<#list stuMap?keys as k>  
<tr>  
    <td>${k_index + 1}</td>  
    <td>${stuMap[k].name}</td>  
    <td>${stuMap[k].age}</td>  
    <td>${stuMap[k].mondy}</td>  
</tr>  
</#list>  
</table>
```

1. json对象.键名
2. json对象["键名"]

遍历Map

### 3、输出

```
输出stu1的学生信息：  
姓名：小明  
年龄：18  
输出stu1的学生信息：  
姓名：小明  
年龄：18  
遍历输出两个学生信息：  
序号 姓名 年龄 钱包  
1 小红 19 200.1  
2 小明 18 1,000.86
```

#### 1.3.1.4 if指令

if 指令即判断指令，是常用的FTL指令，freemarker在解析时遇到if会进行判断，条件为真则输出if中间的内容，否则跳过内容不再输出。

##### 1、数据模型：

使用list指令中测试数据模型。

##### 2、模板：

```
<table>  
    <tr>  
        <td>姓名</td>  
        <td>年龄</td>  
        <td>钱包</td>  
    </tr>
```



```
<#list stus as stu>
<tr>
    <td <#if stu.name =='小明'>style="background:red;"</#if>>${stu.name}</td>
    <td>${stu.age}</td>
    <td>${stu.mondy}</td>
</tr>
</#list>

</table>
```

通过阅读上边的代码，实现的功能是：如果姓名为“小明”则背景色显示为红色。

3、输出：

通过测试发现 姓名为小明的背景色为红色。

## 1.3.2 其它指令

### 1.3.2.1 运算符

- 1、算数运算符 FreeMarker表达式中完全支持算术运算,FreeMarker支持的算术运算符包括:+, -, \*, /, %
- 2、逻辑运算符 逻辑运算符有如下几个:

逻辑与:&& 逻辑或:|| 逻辑非:! 逻辑运算符只能作用于布尔值,否则将产生错误

- 3、比较运算符 表达式中支持的比较运算符有如下几个:

- 1 =或者==:判断两个值是否相等.
- 2 !=:判断两个值是否不等.
- 3 >或者gt:判断左边值是否大于右边值
- 4 >=或者gte:判断左边值是否大于等于右边值
- 5 <或者lt:判断左边值是否小于右边值
- 6 <=:或者lte:判断左边值是否小于等于右边值

注意: =和!=可以用于字符串,数值和日期来比较是否相等,但=和!=两边必须是相同类型的值,否则会产生错误,而且FreeMarker是精确比较,"x","x","X"是不等的.

其它的运行符可以作用于数字和日期,但不能作用于字符串,

大部分的时候,使用gt等字母运算符代替>会有更好的效果,  
因为 FreeMarker会把>解释成FTL标签的结束字符,  
当然,也可以使用括号来避免这种情况,如:<#if (x>y)>

### 1.3.2.2 空值处理

- 1、判断某变量是否存在使用 “??” 用法为:variable??,如果该变量存在,返回true,否则返回false

例：为防止stus为空报错可以加上判断如下：  
<#if stus??>  
<#list stus as stu>  
.....  
</#list>  
</#if>

- 2、缺失变量默认值使用 “!” 使用!要以指定一个默认值 , 当变量为空时显示默

认值。例： \${name!""}表示如果name为空显示空字符串。  
如果是嵌套对象则建议使用( ) 括起



例： \${stu.bestFriend.name}"}表示，如果stu或bestFriend或name为空默认显示空字符串。

### 1.3.2.3 内建函数

内建函数语法格式：变量+?+函数名称

1、和到某个集合的大小

\${集合名?size}

2、日期格式化

```
显示年月日： ${today?date}  
显示时分秒： ${today?time}  
显示日期+时间： ${today?datetime} <br>  
自定义格式化： ${today?string("yyyy年MM月")}
```

3、转成字符串

map.put("point", 102920122);

point是数字型，使用\${point}会显示这个数字的值，不并每三位使用逗号分隔。

如果不想显示为每三位分隔的数字，可以使用c函数将数字型转成字符串输出

\${point?c}

4、将json字符串转成对象

一个例子：

其中用到了 assign标签，assign的作用是定义一个变量。

```
<#assign text="{'bank':'工商银行','account':'10101920201920212'}" />  
<#assign data=text?eval />  
开户行：${data.bank} 账号：${data.account}
```

### 1.3.2.4 完整的模板

上边测试的模板内容如下，可自行进行对照测试。

```
<!DOCTYPE html>  
<html>  
<head>  
    <meta charset="utf-8">  
    <title>Hello World!</title>  
</head>  
<body>  
Hello ${name}!  
<br/>  
<table>
```



```
<tr>
    <td>序号</td>
    <td>姓名</td>
    <td>年龄</td>
    <td>钱包</td>
</tr>
<#list stus as stu>
    <tr>
        <td>${stu_index + 1}</td>
        <td <#if stu.name =='小明'>style="background:red;"</#if>>${stu.name}</td>
        <td>${stu.age}</td>
        <td >${stu.mondy}</td>
    </tr>
</#list>

</table>
<br/><br/>
输出stu1的学生信息 : <br/>
姓名 : ${stuMap['stu1'].name}<br/>
年龄 : ${stuMap['stu1'].age}<br/>
输出stu1的学生信息 : <br/>
姓名 : ${stu1.name}<br/>
年龄 : ${stu1.age}<br/>
遍历输出两个学生信息 : <br/>
<table>
    <tr>
        <td>序号</td>
        <td>姓名</td>
        <td>年龄</td>
        <td>钱包</td>
    </tr>
<#list stuMap?keys as k>
<tr>
    <td>${k_index + 1}</td>
    <td>${stuMap[k].name}</td>
    <td>${stuMap[k].age}</td>
    <td >${stuMap[k].mondy}</td>
</tr>
</#list>
</table>
<br>
<table>
    <tr>
        <td>姓名</td>
        <td>年龄</td>
        <td>出生日期</td>
        <td>钱包</td>
        <td>最好的朋友</td>
        <td>朋友个数</td>
        <td>朋友列表</td>
    </tr>
<#if stus??>
    <#list stus as stu>
```



```
<tr>
    <td>${stu.name}''}</td>
    <td>${stu.age}</td>
    <td>${(stu.birthday?date)'''}</td>
    <td>${stu.mondy}</td>
    <td>${(stu.bestFriend.name)'''}</td>
    <td>${(stu.friends?size)!0}</td>
    <td>
        <#if stu.friends??>
        <#list stu.friends as firend>
            ${firend.name}''}<br/>
        </#list>
        </#if>
    </td>
</tr>
</#list>
</#if>

</table>
<br/>
<#assign text="{'bank':'工商银行','account':'10101920201920212'}" />
<#assign data=text?eval />
开户行 : ${data.bank} 账号 : ${data.account}

</body>
</html>
```

### 1.3.3 静态化测试

在cms中使用freemarker将页面生成html文件，本节测试html文件生成的方法：

#### 1、使用模板文件静态化

定义模板文件，使用freemarker静态化程序生成html文件。

#### 2、使用模板字符串静态化

定义模板字符串，使用freemarker静态化程序生成html文件。

#### 1.3.3.1 使用模板文件静态化

在test下创建测试类，并且将main下的resource/templates拷贝到test下，本次测试使用之前我们在main下创建的模板文件。

```
//基于模板生成静态化文件 这个测试不需要SpringBoot支持
@SpringBootTest
public void testGenerateHtml() throws IOException, TemplateException {
    //创建配置类
    Configuration configuration=new Configuration(Configuration.getVersion());
```



```
//设置模板路径
String classpath = this.getClass().getResource("/").getPath();
configuration.setDirectoryForTemplateLoading(new File(classpath + "/templates/"));
//设置字符集
configuration.setDefaultEncoding("utf-8");
//加载模板
Template template = configuration.getTemplate("test1.ftl");
//数据模型
Map<String, Object> map = new HashMap<>();
map.put("name", "黑马程序员");
//静态化
String content = FreeMarkerTemplateUtils.processTemplateToString(template, map);
//静态化内容
System.out.println(content);
InputStream inputStream = IOUtils.toInputStream(content);
//输出文件
FileOutputStream fileOutputStream = new FileOutputStream(new File("d:/test1.html"));
int copy = IOUtils.copy(inputStream, fileOutputStream);
}
```

### 1.3.3.2 使用模板字符串静态化

```
//基于模板字符串生成静态化文件
@Test
public void testGenerateHtmlByString() throws IOException, TemplateException {
    //创建配置类
    Configuration configuration=new Configuration(Configuration.getVersion());
    //模板内容，这里测试时使用简单的字符串作为模板
    String templateString="" +
        "<html>\n" +
        "    <head></head>\n" +
        "    <body>\n" +
        "        名称 : ${name}\n" +
        "    </body>\n" +
        "</html>";

    //模板加载器
    StringTemplateLoader stringTemplateLoader = new StringTemplateLoader();
    stringTemplateLoader.putTemplate("template",templateString);
    configuration.setTemplateLoader(stringTemplateLoader);
    //得到模板
    Template template = configuration.getTemplate("template","utf-8");

    //数据模型
    Map<String, Object> map = new HashMap<>();
    map.put("name", "黑马程序员");
    //静态化
    String content = FreeMarkerTemplateUtils.processTemplateToString(template, map);
    //静态化内容
    System.out.println(content);
```



```
InputStream inputStream = IOUtils.toInputStream(content);
//输出文件
FileOutputStream fileOutputStream = new FileOutputStream(new File("d:/test1.html"));
IOUtils.copy(inputStream, fileOutputStream);
}
```