Beetl 模板语言使用指南

---李家智 2012-11-18（要要要发版）

1. 什么是 Beetl 2
   1.1. Beetl 能为你做些什么 3
   1.2. Beetl 可靠么？性能如何？ 3
2. 基本用法 4
   2.1. Hello Beetl 4
   2.2. 控制语句和占位符号 5
   2.3. 定义变量 6
   2.4. 属性引用 8
   2.5. 算术表达式 8
   2.6. 逻辑表达式 8
   2.7. 循环语句 9
   2.8. 条件语句 10
   2.9. 函数调用 11
   2.10. 格式化 12
   2.11. 直接调用 class 方法和属性 12
   2.12. 错误处理 13
   2.13. 安全输出 14
   2.14. 注释 15
   2.15. 其他琐碎功能 15
3. 高级用法 17
   3.1. 总是使用 GroupTemplate 17
   3.2. 配置 GroupTemplate 18
   3.3. 允许优化，超越其他模板引擎 19
   3.4. 允许 Byte 直接输出 20
   3.5. 自定义函数 21
   3.6. 格式化函数 22
   3.7. 模板变量 23
   3.8. 严格 MVC 控制 24
   3.9. 虚拟属性 24
   3.10. 标签 25
   3.11. 宏 26
   3.12. 空格处理 27
   3.13. 自定义错误处理 28
   3.14. 缓存字符串模板 28
   3.15. 布局 29
   3.16. 测试模板 29
4. Spring MVC 30
   4.1. 配置 ViewResolver 30
   4.2. 模板中获取参数 31
5. Struts2 32

6. JFinal.....32
7. 在 Servlet 中使用 Beetl.....34
7.1. 配置 Linstener.....34
7.2. 编写 Servlet.....35
8. 附录：扩展包.....36
8.1. 函数.....36
8.2. 格式化函数.....38
8.3. 标签.....38
8.4. Freemarker 功能对比.....39
8.5. Freemarker 性能比较.....42
8.5.1. 单线程：.....42
8.5.2. 并发.....44
9. Beetl 历史.....44

感谢老婆和儿子，有时候容忍我回到家后还继续写代码

感谢公司和同事，都这么忙，还支持我完成 Beetl

感谢使用 Beetl 的陌生朋友，亲~没有你们，Beetl 也不可能做到最好

1. 什么是 Beetl

Beetl 是 Bee Template language，Bee 译为忙碌的人，意指忙碌中国的开发人员。目前版本 1.2Beta，整个大小**410K**，它具有如下特性：

1 **非常简单**：它的语法是 javascript 一个子集，只有少量的大家熟悉的符号。任何了解 java，或者 javascript 的人，都能快速学会。如果从未用过任何模板语言，用 Beetl 是非常很合适的.不需要在学习一种的新的语言语法

2 **功能齐全**：beetl 具有目前流行的模板引擎所支持的功能，如具备 freemarker 所提供的绝大部分功能（参考 附录 freemarker 功能对比）

3 **性能卓越**：在优化模式(包括动态编译成 class 和采用 byte 输出)下运行，每秒渲染模板数高于其他模板引擎，而消耗的系统资源低于其他模板引擎。（参考附录 Freemarker 性能比较）

4 **提供一系列其他模板语言没有提供的功能**，如自定义占位符号，控制语句符号，模板变量,虚拟属性，自定义函数，标签 tag 等，安全输出等。它们并不复杂，但有可能解决你在使用别的模

板语言时候遇到的一些不便捷的问题。

5 **同时支持较为松散的 MVC 和严格的 MVC**，如果在模板语言里嵌入计算表达式，复杂条件表达式，以及函数调用有干涉业务逻辑嫌疑，你可以禁止使用这些语法。关于这一点，可以参考[strictly enforces model-view separation]

6 **能与 Serlvet, Spring MVC, Struts2, JFinal 等 web 框架结合起来**

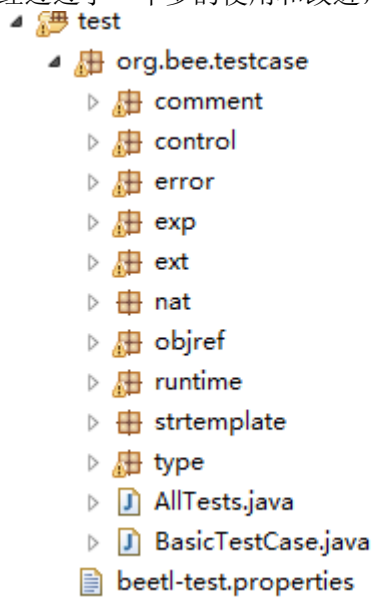
1.1. Beetl 能为你做些什么

作为模板语言，你可以用于任何适合在 MVC 的地方。如**代码生成**，或者超大规模访问量的互联网，电商的动态 **web 界面**，在功能上和性能上它完全能代替目前流行的 Freemarker，Velocity。如果有任何问题，可以访问 QQ 群219324263，或者发送邮件到 xiandafu@126.com

因为 Beetl 是基于 antlr 实现语法解析的，因此如果你仅仅对 antlr 感兴趣，beetl 仍然可以作为你的一个重要参考

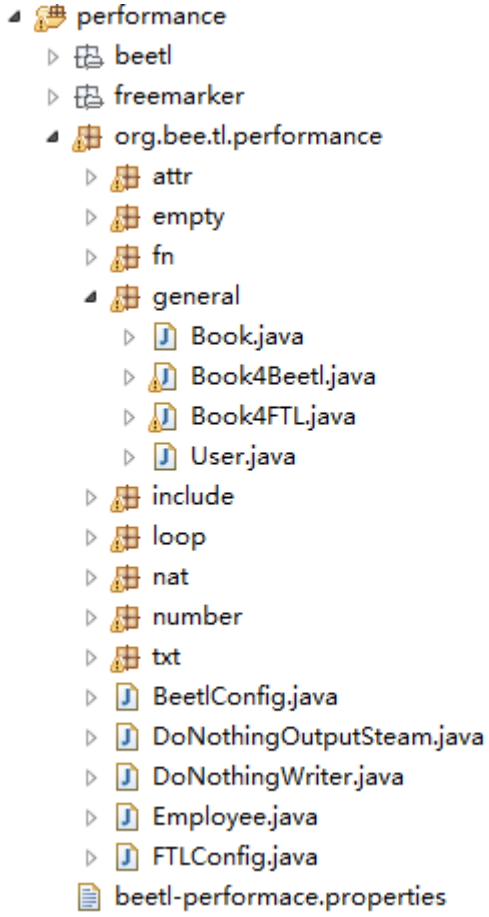
1.2. Beetl 可靠么？性能如何？

Beetl 已经有不少使用者，有电子商务系统，也有企业内部管理系统，也有大型门户网站，经过过了一年多的使用和改进，beetl 已经很稳定可靠了，如下式 Beetl 的单元测试。



Beetl 的性能也非常不错，针对流行的 Freemarker，做了单项和整体测试，每一项都能超过

Freemarker。



2. 基本用法

2.1. Hello Beetl

```
package org.bee.tl.samples;

import org.bee.tl.core.GroupTemplate;
import org.bee.tl.core.Template;

public class Helloworld
{
	public static void main(String[] args) throws Exception
	{

		String input = "hello,${name}!";
		GroupTemplate group = new GroupTemplate();/1
		Template template = group.getStringTemplate(input); /2
		template.set("name", "beetl");/3
		String output = template.getTextAsString();/4
		System.out.println(output);
	}
```

```
}
```

1 用空的构造函数创建 GroupTemplate 创建一个模板,能接收一个字符串模板。 更常见的是用模板文件根目录初始化 GroupTemplate 。如 GroupTemplate group = new GroupTemplate(“c:\templateroot”),能接收一个模板文件创建模板

2 模板是 String,通过 GroupTemplate 的 getStringTemplate 创建一个模板类 Template。“${”“}”是模板里的占位符号。里面的表达式计算后将输出。在此例子里,name 是一个变量

如果模板是一个文件,则需要 getFileTemplate,传入一个相对 root 的路径来创建一个模板类,如

```
GroupTemplate  group = new GroupTemplate(“c:/templateroot”)
Template template  = group.getFileTemplate(“/user/userList.html”);
```

GroupTemplate 会解析 c:/templateroot/user/userList.html

3 template.set("name", "beetl") 是定义模板变量,set 方法允许 字符串-对象作对为参数.字符串为模板里的变量名,因此字符串必须合乎 java 变量名规则,如字母开头,不能有空格等。 通过 set 方法设置变量后,可以再模板里任何地方直接引用。 如果需要引用对象的属性,则用小数点,如${user.name},如果对象是个数组或者 List 集合,可以用[索引],如${user.friends[0]},如果对象是 Map 集合,使用[key],key 为任何对象,如${books[‘thinking in java’].author}

4 调用 template.getTextAsString() 或者 template.getText(Writer) 或者 template.getText(OutputStream os) 都可以获得模板渲染的结果(如果设置允许 byte 直接输出,则只能用 OutputStream,参考高级特性允许 byte 直接输出)。本例子中,输出是 hello,beetl!

2.2. 控制语句和占位符号

Beetl 默认情况下,模板采用<%作为控制语句开始,%>作为控制语句结束

```
<% for(user in userList){ %>

hello,${user.name}

<%}%>
```

默认情况下，占位符号使用"${"作为开始和"}"结尾占位符号

```
${users[index]}
```

然而，Beetl 支持自定义控制语句和占位符号，以适应不同类型模板文件,如果有你喜欢的风格，譬如 html 模板中，使用<!--: -->，则可以采用如下定义

```
public static void main(String[] args) {
	GroupTempate group = new GroupTemplate(root);
	group.setStatementStart("<!--:");
	group.setStatementEnd("-->");
	group.setPlaceholderStart("${");
	group.setPlaceholderStart("}");

	// 或者可以直接调用
	//group.config("<!--","-->","${","}");


	File child = …..
	Template template =group.getFileTemplate(child);

	String output = template.getTextAsString();



}
```

此代码可以解析如下模板文件

```
<!--: var name=” lijz” ;

var greeting = “morning” ;

-->

<p>Hello ${name} ${greeting~}</p>
```

以下内容如果不做特殊说明，控制语句使用<% %>和占位符号采用${}

2.3. 定义变量

Beet 有俩种变量，一种是通过 template.set(...)中传入的，这个是只读全局变量，在模板中任何一处，包括子模板等都可以被引用，但不能更改，另外，Beetl 也允许定义临时变量，(在模

板页面，使用临时变量并不值得推荐，但 Beetl 能支持），如下所示

```
<%var name='lijz',loopCount=100+other ,girlName;%>
```

关键字 var 是必须的，这点不同于 javascript

Beelt 中得变量同 javascript 不一样，没有自己的作用域，这点跟 java 类似，如下会报错，“变量已经定义”。早起版本支持 javascript 作用域，但显然这并不是一个好特性

```
<%var name=' lijz' ,i=1; %>

<%if(i>=1){%>

 <%var name='lucy';%>

 Hello,${name}

<%}%>
```

Beetl 支持定义 Json 变量，虽然此功能并不是很实用，推荐复杂的数据结构，还是要再控制层设置比较好

```
<% var usersList=[{“name”:”lijz”,”age”:18},{“name”:”lucy”,”age”:16}];%>

她得名字是 ${userList[1][“name”]}
```

Beetl 将上述 Json 变量对应于 java 的 Map 或者 List,如上例子则是一个 List,包含了三个 Map，第一个 Map 的 name 健值对应于“lijz”

对于 Beetl 中出现的整形或者浮点型，对应于 java 中 BeeNumber,是一个支持长精度型的类.beetl 并不试图 wrap java 对象，这不同于别的模板语言，但为了支持长精度运算，模板语言中的数字型都用 BeeNumber 代表

变量命名规则同 javascript 或者 java，但不允许以俩个下划线开头"__",这是因为以此开头的多为 Beetl 内部的一些临时变量

2.4. 属性引用

属性引用是模板中的重要一部分，beetl 支持属性引用如果 javascript 的支持方式一样，如下

1 Beetl 支持通过”.”号来访问对象的的属性，如果 javascript 一样。如果 User 对象有个 getName()方法，那么在模板中，可以通过${xxx.name}来访问

2 如果模板变量是数组或者 List 类，这可以通过[] 来访问，如${userList[0]}

3 如果模板变量是 Map 类，这可以通过[]来访问，如${map[“name”]},如果 key 值是字符串类型，也可以使用${map.name}.但不建议这么使用，因为会让模板阅读者误以为是一个 Pojo 对象

4 Beetl 也支持 **Generic Get** 方式，即如果对象有一个 public Object get(String key)方法，可以通过”.”号或者[]来访问，譬如
${activityRecord.name}或者${activityRecord[“name”] }都将调用 activityRecord 的 get(String key)方法。如果对象既有具体属性，又有 Generic get（这种模型设计方式是不值得鼓励），则以具体属性优先级高.

5 Beetl 也可以通过[]来引用属性，如${user[“name”]} 相当于${user.name}.这跟 javascript 保持一致。但建议不这么做，因为容易让阅读模板的人误认为这是一个 Map 类型

2.5. 算术表达式

Beetl 支持类似 javascript 的算术表达式和条件表达式，如+ - * / % 以及（），如下例子

```
<%:var a=1,b=2,c=3; %>

the result is ${(a+b)*c-0.75}
```

注：算数表达式无需担心计算精度问题，beetl 底层用 BeeNumber 来实现，支持长精度计算

2.6. 逻辑表达式

Beetl 支持类似 Javascript,java 的条件表达式 如>， <， == ，!=，>= , <= 以及 !,如下例子

```
<% var a=1,b=2,c=3;if((b-c)>=1){ %>
```

```
Hello big!

<%}else{%>

:( ,small!

<%}%>
```

2.7.循环语句

Beetl 支持 for in 循环格式，和 while 循环，以及 break，continue，return (实际上可以出现在任何地方)，如下例子

```
//java 代码

tempalte.set("userList",userList);

//模板

总共 ${userList.~size}

<%for(user in userList){%>

${user_index} . Welcome ${user.name}!

<%}%>
```

循环中可以使用数组，List 或者 Map 的子类，当使用 Map 子类的时候，如下例子

```
<% var softMap={'beetl':'very good ','freemarker':'very good!'};%>

Compare the ${softMap~size} soft ware as follows

<%for(soft in softMap){%>

${soft_index} / ${soft_size} . ${soft.key} is ${soft.value}

<%}%>
```

key 既是 Map 里的 key 值

value 既是 Map 里的 Value

如果循环中，需要引用当前索引,变量名加上后缀“_index”,代表当前循环的计数器，从0开始。在循环体里还可以变量名加上后最_size 表示长度

Beetl 也可以在集合变量名后加上"!" 用以安全输出，即如果集合为 null，则也不进入循环体，如

```
<%for(soft in softMap!){%>

.................

<%}%>
```

Beetl 也支持 while 循环，其格式是 while(条件表达式){}，如下例子，循环4次输出

```
<%
var a = 1;
while(a<5){
%>
${a}
<%
a=a+1;
}

%>
```

2.8. 条件语句

Beetl 支持同 javascript，java 一样的 if 语句，和 switch. 如下例子

```
<%var isGirl = true;%>

<%if(isGir){%>
```

```
姑娘，你好

<%}else{%>

小伙子，你好

<%}%>
```

Switch 和 case 可以使用任何表达式，下列中，switch 将变量 score 与 case 中得各个常量比较，如果没有找到，则执行 default

```
<%var score = 1;%>

<%switch(score){%>

  <%case 1:{%>

Get 1 score!

  <%break;}%>

  <%default:{%>

Default here

  <%}

}%>
```

2.9. 函数调用

Beetl 内置了少量实用函数，可以在 Beetl 任何地方调用，一般情况是在占位符里调用。

如下例子是调用 date 函数，不传参数情况下，返回当前日期

```
Today  is ${date()}
```

Beetl 允许用户自定义函数以适合特定领域使用，请参考高级用法。也欢迎有人把他认为能公用的函数发给我，我将作为核心函数放入 beetl 里

2.10.格式化

几乎所有的模板语言都支持格式化，Beetl 也不列外，如下例子 Beetl 提供的内置日期格式

```
<%var date = date(); %>

Today is ${date,dateFormat="yyyy-MM-dd"}.

Today is $date,dateFormat$
```

格式化函数只需要一个字符串作为参数放在=号后面，如果没有为格式化函数输入参数，则使用默认值，dateFormat 格式化函数默认值是 local

Beetl 允许用户自定义格式化函数以适合特定领域，请参考高级用户，也欢迎有人把他认为能公用的格式化函数发给我，我将作为核心函数放入 beetl 里

2.11. 直接调用 class 方法和属性

默认情况，是不允许直接调用对象的方法的，必须先调用 groupTemplate.enableNativeCall()；

```
${@user.getMaxFriend(“lucy”)}
${@user.maxFriend}
${@com.xxxx.constants.Order.getMaxNum()}
<%
var max = @com.xxxx.constants.Order.MAX_NUM;
%>
```

可以调用 instance 的 public 方法和属性，也可以调用静态类的属性和方法 ，需要加一个@指示此调用是直接调用 class

目前 Beetl 不支持连续调用 Class 方法和属性，**如下列是非法的**

@com.xxxx.constants.Order.getMaxNum().intValue()

2.12.错误处理

Beetl 默认情况使用 org.bee.tl.core.DefaultErrorHandler 来处理语法解析错误和运行时刻的错误，在解释执行的认情况下，会显示如下信息

- **错误行号**
- **错误原因以及异常栈**
- **错误的关键字**
- **以及错误上下3行**
- **错误所在的模板文件**

如下所示

```
Template t = new BeeTemplate("<%:if(!isGirl){var c=1;}%>");

//使用严格 MVC 限制，因此不允许变量定义
t.makeStrict(true);
t.set("isGirl", false);
t.getTextAsString() ;

会导致如下编译错误



STRICK_MVC 位于1行，符号 var
1|<%if(!isGirl){var c=1;} %>
```

关于如何自定义错误处理，请参考高级用法

如果是在编译执行，则显示的信息没有错误关键字，这是因为已经编译成 class，java 报错不支持错误关键字

2.13.安全输出

安全输出是任何一个模板引擎必须重视的问题，否则，将极大困扰模板开发者。Beetl 中，如果要输出的模板变量为 null，则 beetl 将不做输出，这点不同于 JSP，JSP 输出 null，也不同于 Feemarker，如果没有用!,它会报错.

模板中还有俩中情况会导致模板输出异常

- 有时候模板变量并不存在（譬如子模板里）
- 模板变量为 null，但输出的是此变量的一个属性，如${user.wife.name}

针对前俩种种情况，可以在变量引用后加上！以提醒 beetl 这是一个安全输出的变量。
如${{user.wife.name! },即使 user 不存在，或者 user 为 null，或者 user.wife 为 null，或者 user.wife.name 为 null beetl 都不将输出
可以在!后增加一个常量（字符串，数字类型），或者另外一个变量，方法，本地调用，作为默认输出，譬如
${user.wife.name!”单身”}，如果 user 为 null，或者 user.wife 为 null，或者 user.wife.name 为 null，输出”单身”
譬如${user. birthday!@System.constants.DefaultBir}，表示如果 user 为 null，或者 user. birthday 为 null，输出 System.constants.DefaultBir

- 还有一种情况很少发生，但也有可能，输出模板变量发生的任何异常，如变量内部抛出的一个异常

这需要使用格式${!(变量)},这样，在变量引用发生任何异常情况下，都不作输出，譬如${!(user.name)},，beetl 将会调用 user.getName()方法，如果发生异常，beetl 将不会忽略此异常，继续渲染

如下是预编译后的代码

```
try{
    out.write(user.getName());
  }catch(Exception ex){
 }
```

值得注意的是，在变量后加上!不仅仅可以应用于占位符输出(但主要是应用于占位符输出)，也可以用于表达式中，如：
<%
var k = user.name!'N/A'+user.age!;
%>
${k}
如果 user 为 null，则 k 值将为 N/A

”

2.14.注释

Beetl 采用 javascript 一样的注释风格，即单行使用"//" ,多行可以使用/* */
如下代码

```
<%
var a = 3 ;
// 单行注释
/* 这是多行注释 */
var b = 2;
/* 以下内容页被注释了
.....
%>
内容.....
<%
*/
%>
```

Beetl 现在暂时不支持占位符的注释

2.15.其他琐碎功能

对齐： 我发现别的模板语言要是做到对齐，非常困难,Beetl 你完全不用担心，比如 velocty，stringtemlate，freemarker 例子都出现了不对齐的情况，影响了美观，Beetl 完全无需担心输出对齐

包含其他模板文件：在 Beetl 中，这不是一个特殊的功能，通过调用函数 includeFileTemplate,或者 includeStringTemplate 都可以实现，前者是包含一个文件模板，后者是将一个 string 模板作为输入。详细使用请参考高级用法

Escape：可以使用\ 做escape 符号，如\$monkey\$ 将作为一个普通的文本，输出为$monkey$.再如为了在钱后加上美元符号（占位符恰好又是美元符号）可以用这俩种方式hello,it's $money$\$, 或者Hello,it's $money+"\$"$ 。如果要输出\符号本生，则需要用俩个\\,这点与 javascript，java 语义一致.

空值策略： 如果一个变量为空，则输出的时候默认为空字符串，也可以制定输出

```
${u.wife.name!"N/A"}
```

如果 u 为空，或者 u.wife 为空，输出都是"N/A"

标签：类似 jsp 的标签 Tag，允许你将模板文件中的一段文件内容作为输入，经过函数操作，变成特定的输出，如下标签

```
<%includeFileTemplate ("/footer.html") { %>

This is static footer,will be replaced by fotter.html

<%}%>
```

则模板文件里等于号后的字符串将被以此替换.

表达式计算： org.bee.tl.core.SimpleRuleEval .扩展了 BeeTemplate，可以用来计算表达式，如下代码

```
SimpleRuleEval eval = new SimpleRuleEval("a+15");

eval.set("a", 3);

int o = eval.calcInt();

此时 o 为 18
```

表达式可以是任何计算表达式或者条件表达式，如(12+5)*3,或者 a>12&&a<=18

其他方法还有

calc()返回 Object，(对于算数计算，Object 为 BeeNumber 类型)
calcInt() 返回 Int
calcBoolean() 返回 boolean
calcDouble() 返回 double

其他详细参考 API

3. 高级用法

3.1. 总是使用 GroupTemplate

无论你是在学习使用 Beetl，还是在项目中使用 Beetl，好的习惯是正确初始化 GroupTemplate，并通过 GroupTemplate 获得 Template，如下代码

```
import org.bee.tl.core.GroupTemplate;
public class GroupTemplateUtil {

	static GroupTemplate group = new GroupTemplate("/home/template");
	static {
		group.setPlaceholderStart("<%");
		group.setPlaceholderEnd("%>");
		group.setStatementStart("${");
		group.setStatementEnd("}");
		group.enableOptimize(); //1
		group.enableNativeCall(); //2
		group.enableChecker(10); //3
		addCommonFunction(); //4
		addCommonFormat(); //5


	}
	public static GroupTemplate getGroup (){
		return group;
	}
	public static void addCommonFunction(){

	}

	public static void addCommonFormat(){

	}

}
```

1. 允许优化成 class 代码运行

2 运行直接调用 java 类

3 每10秒检查一下模板文件是否更新,如果未调用，或者设置为0，则不检查更新

4 增加项目自定义方法

5 增加项目自定义的格式化函数

然后你可以在代码里调用

```
Template t = GroupTemplateUtil .getGroup
().getFileTemplate("/exp/string_add_template.html");

T.set("user",new User());

String str = t.getTextAsString();
```

3.2. 配置 GroupTemplate

GroupTemplate 有很多选项，可以通过 Config 类来完成创建 GroupuTemplate，Config 默认会先装载 /org/bee/tl/core/beetl-default.properties, 然后如果 classpath 下存在 beetl.properties,再装载或者覆盖其属性。beetl-default.properties 内容如下

```
#######默认配置
DELIMITER_PLACEHOLDER_START=${
DELIMITER_PLACEHOLDER_END=}
DELIMITER_STATEMENT_START=<%
DELIMITER_STATEMENT_END=%>
NATIVE_CALL = TRUE
COMPILE_CLASS=FALSE
DIRECT_BYTE_OUTPUT = FALSE
TEMPLATE_ROOT=
TEMPLATE_CHARSET = GBK
TEMPLATE_CACHE_CHECK_PERIOD = 2
TEMPLATE_CLASS_FOLDER=
ERROR_HANDLER = org.bee.tl.core.DefaultErrorHandler
MVC_STRICT = FALSE
#内部使用
DEBUG=FALSE
#######默认配置结束
```

```
#######性能最佳配置开始
#COMPILE_CLASS=true
#DIRECT_BYTE_OUTPUT = true

########性能最佳配置结束

#######严格MVC配置开始
#MVC_STRICT = TRUE
#NATIVE_CALL = FALSE
########严格MVC配置结束


#######编译成class的其他选项开始，未完全测试，暂时不支持
#COMPILE_CLASS_KEEP_SOURCE=FALSE
#OPTIMIZE_COMPILE_LATTER=FALSE
#OPTIMIZE_COMPILE_WORKER_NUM=2

#######编译成 class 的其他选项结束
```

如下例子是 beetl 单元测试的配置

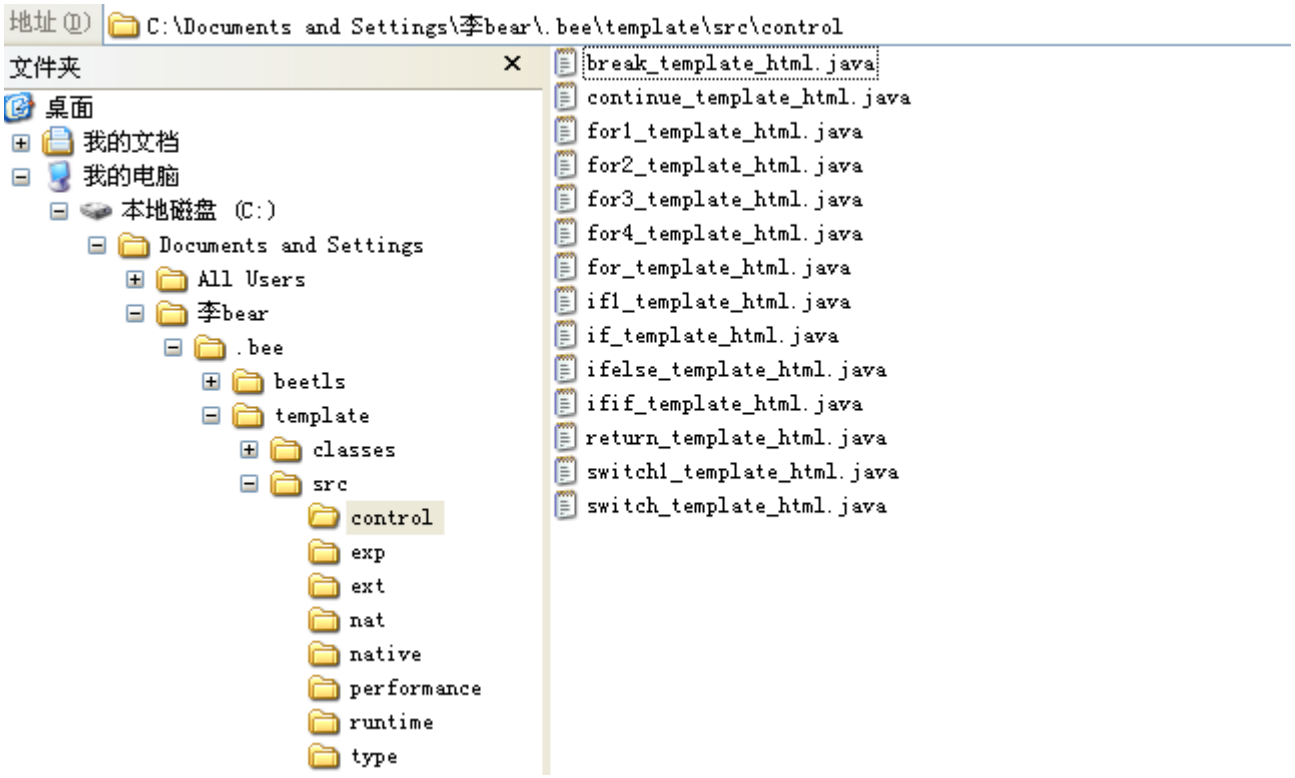
```
Config config = new Config();
config.load("/beetl-test.properties");
home = System.getProperty("user.dir") + File.separator + "template" +
File.separator;
config.put(Config.TEMPLATE_ROOT, home);

GroupTemplate base = config.createGroupTemplate();
```

3.3. 允许优化，超越其他模板引擎

默认情况下，Beetl 采用解释执行，性能略低于其他模板语言，且同其他模板语言一样，消耗了较大的系统资源。Beetl1.0版本后可以在运行时编译成 class，并且通过运行时刻推测出其模板变量的 java 类型，通过杜绝反射操作从而获得最好的性能

只需要调用 groupTemplate.enableOptimize();

默认情况下，所以预编译的类都将放在 user.home 的.bee 目录下，如下示例



如果想要保存在指定的目录，可以调用 groupTemplate.setTempFolder(String path),传入路径

目前并不是所有的模板都能优化成 class 代码。请参考代码优化了解如何编写模板代码。但无论如何，如果优化失败，beetls 将会解释执行。

注意：新版本中默认不生成 java 源代码，如果想保持生成的 java 代码，则需要如下初始化 GroupTemplate

```
Map config  = new HashMap();
config.put(GroupTemplate.OPTIMIZE_KEEP_SOURCE, true);
group.enableOptimize(config);
```

3.4. 允许 Byte 直接输出

如果你确定输出的字符集总是固定的，比如总是 GBK，那么，允许 byte 输出将对性能进一步提供质的优化。加速渲染速度以及降低对 CPU 的消耗。这是因为，byte 直接输出，省去了模板中文本的转码过程。允许 byte 字节输出，性能会提供一倍，需要做的仅仅是如下俩个设置:

groupTemplate.enableDirectOutputByte();
再输出的时候，提供 OutputStream 作为输出流，如 web 开发里

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
            response.setContentType("text/html;charset=UTF-8");
            Template t =
ServletGroupTemplate.intance().getTemplate("/news.html", request,
response);
            PortalService portalService = new PortalService();
            List<News> list =
portalService.getNews(News.GNENERAL_NEWS);
            t.set("newsList", list);
            try {
                  t.getText(response.getOutputStream());
            } catch (BeeException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
      }
```

3.5. 自定义函数

Beetl 允许提供自定义函数以适合特定业务需求，自定义函数需要实现 org.bee.tl.core.Function。如下定义了一个 date 函数仅仅返回一个 java.util.Date 实例

```
public class DateFunction implements Function {
      public Date call(Object[] paras,Context ctx) {
            return new Date();
      }
      public  static void main(String[] args) throws IOException{
            GroupTemplate group = new GroupTemplate();
            group.registerFunction("date", new DateFunction());
            Template t = group.getStringTemplate("today is $date()$");
            System.out.println(t.getTextAsString());
      }


}
```

注册的方法名可以带"."，如下
group.registerFunction("date.now", new DateFunction());则在模板里，可以使用today is ${date.now()}.

另外一种自定义 beetl 方法的是使用 GroupTemplate.registerFunctionPackage(String packageName,Object o);

Beetl 将找到 o 所属的所有 public 非 Static 方法，并依次按照 packageName+"."+方法名 作为 beetl 方法名，如下例子

```
group.registerFunctionPackage("string", new Object(){
```

```
                    /**
                     * @param str
                     * @param from d对于数字型，beetl里都使用BeeNumber
                     * @param to
                     * @return
                     * @throws Exception
                     */
                    public    String    sub(String    str,BeeNumber
from,BeeNumber to) throws Exception{

                        if(str==null){
                            throw      new      Exception("error
paramter");
                        }
                        return      str.substring(from.intValue(),
to.intValue());
                    }

                    public void print(String str,Context ctx){
                        Writer w = (Writer)ctx.getVar("__pw");
                        try {
                            w.write(str);
                        } catch (IOException e) {
                            throw                                new
RuntimeException(e.getMessage());
                        }

                    }

            });
```

如上注册了俩个方法，分别是 string.sub 方法以及 string.print 函数。

3.6. 格式化函数

Beetl 允许提供自定义格式化函数，用于格式化输出。 格式化函数需要实现 org.bee.tl.core.Format

```
public class DateFormat extends Format  {
    public Object format(Object data,String pattern){
        SimpleDateFormat sdf = null;
        if(pattern==null){

            sdf = new SimpleDateFormat();
        }else{
            sdf = new SimpleDateFormat(pattern);
```

```
            }

            return sdf.format(data);
      }
      public static void main(String[] args)throws IOException {
            GroupTemplate group = new GroupTemplate();
            group.registerFunction("now", new DateFunction());
            group.registerFormat("df", new DateFormat());
            Template t = group.getStringTemplate("today is $now(),df=' yyyy-MM-dd'
$");
            System.out.println(t.getTextAsString());
      }
}
```

其中,也可以直接用 today is ${now(),shortDate} 如果没有=号，format(Object data, String pattern) 中的 pattern 为空

注册的格式化函数名可以带".", 如下：

group.registerFormat("date.short", new DateFormat());则在模板里，可以使用 today is ${now(),date.short}.

3.7. 模板变量

模板变量定义和赋值跟其他变量没有区别，唯一不同的时他允许的内容是一段模板，他的值是模板所对应的字符串（字节模式下是字节数组），格式如下
var varibaleName = {….}.，模板变量可以一旦定义好后，可以用在其任何地方，譬如重复使用模板变量，将模板变量用在子模板里：

```
<%
var a = {
        var k= 123;
%>
        ${k}.
<%
        };
%>
${a}
```

如上例，a 是一个模板变量，结果是"123."
使用模板变量一个常见的地方是更加灵活的布局

```
<%
var part1 = {

%>
```

```
这是内容一
<%
};
%>

<%
var part2 = {

%>
这是内容二
<%
};
%>
inlcudeFileTemplate("/common/_layout.html",{"part1":part1,"part2":part2})
```

3.8. 严格 MVC 控制

如果设置了严格 MVC，则以下语法将不在模板文件里允许，否则将报出 STRICK_MVC 错误

定义变量，为变量赋值,如 var a = 12是非法的

算术表达式 如${user.age+12}是非法的

除了只允许布尔以外，不允许逻辑表达式和方法调用 如 if(user.gender==1)是非法的

方法调用，如${subString(string,1)}是非法的

Class 方法和属性调用，如${@user.getName()}是非法的

严格的 MVC，非常有助于逻辑与视图的分离，特别当逻辑与视图是由俩个团队来完成的。如果你嗜好严格 MVC，可以调用 groupTemplate.enableStrict()

3.9. 虚拟属性

无需为 java 对象定义额外的属性用于辅助显示，虚拟属性可以轻易做到，如 Beetl 为 java.util.Collection 定义的一个虚拟属性 size，用于表示集合大小

```
group.registerVirtualAttributeEval(new VirtualAttributeEval(){
      public Object eval(Object o,String attributeName,Context ctx){
```

```
            if(attributeName.equals("size")){
                  return ((Collection)o).size();
            }else{
                  throw new IllegalArgumentException();
            }

      }
      public boolean isSuppoert(Class c,String attributeName){

             if(Collection.class.isAssignableFrom(c)&&attributeName.equals("size
             ")){
                  return true;
            }else{
                  return false;
             }
      }

});
```

这样，所以 Collection 子类都有虚拟属性 size。$userList.~size$ 输出 userList 集合长度

实现虚拟属性，必须实现接口俩个方法，一个是 isSupport,这让 Beetl 用于找到 VirtualAttributeEval，eval 方法用于计算虚拟属性

3.10.标签

所谓标签，即允许处理模板文件里的一块内容，功能等于同 jsp tag。如下{}的内容在 beetl 运行的时候将会被删除

```
<%del(){%>

This content will be deleted

<%}%>
```

自定义标签 org.bee.tl.core.Tag，需要实现 requriedInput，用于告诉 Beetl，是否需要先渲染文本体。

setInput 是把渲染的结果传回给标签函数

getOutput 最后用于返回文本处理函数的结果

如下是 Beetl 提供的内置的 del 文本处理函数实现

```
public class DeleteFunction extends Tag{
	public String getOutput(){
		return "";
	}
	@Override
	public  boolean requriedInput(){
		return false;
	}


}
```

可以通过父类属性 args,获取输入参数，详细可以参考 API

由于标签将会处理标签内的渲染结果，如果在使用 byte 输出的话，最好继承 ByteSupportTag，同时实现 getOutput(),getOutputAsByte()俩个接口，具体用法可以查看源码。如果是允许直接输出，Beetl 将会调用 getOutputAsByte () 而不是 getOutput(),然而，如果你继承 Tag，也可以的，只是 byte 输出的性能将在此处会打一点折扣.因为 beetl 判断只实现了 Tag 接口，就会调用 getOutput()方法，将得到输出再转一次 Byte 作为渲染结果。

详情参考IncludeFileTemplateTag 和 LayoutTag的实现。

3.11.宏

Beetl 不直接支持宏定义，但可以间接通过内置的 includeFileTemplate 标签来做到，这正如 JSP 做的那样，如下是一个 print.html 文件，将定义个宏

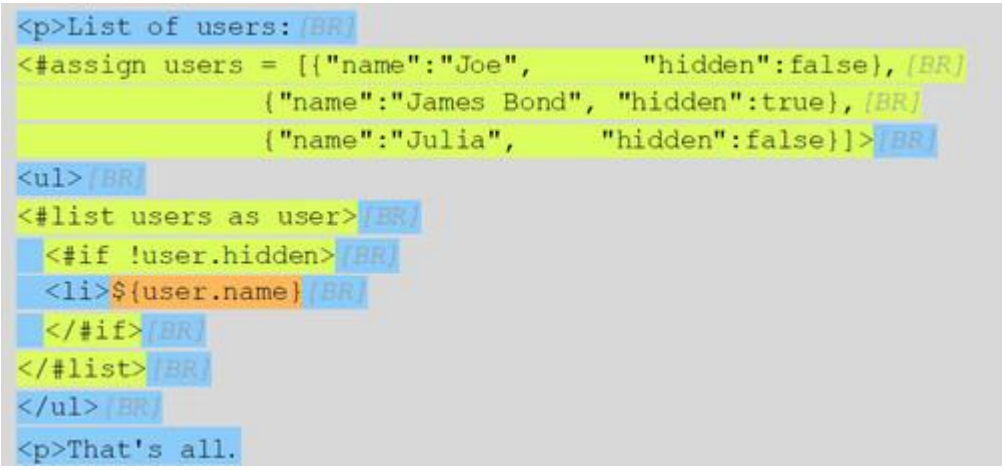
```
Debug info is <% println(object);%>
```

则在任何文件中，都可以使用 includeFileTemplate 使用此宏

```
<% inlucdeFileTemplate("/print.html",{"object": realValue}){}%>
```

3.12.空格处理

大多数模板语言都会输出额外的空格或者回车，JSP 也如此，freemaker 还专门有一删除多余空格的函数，**在 beetl 中，是不需要此功能的**



上图来源于南磊翻译的中文版 Freemarker，说是以上代码会有过多的换行（br），必须使用删除多余空行函数才能使模板输出达到我 们想得样子。Beetl 没有此额外函数做这事情，因为 Beetl 自动就能分辨出这些额外空行。

为什么 beetl 无需担心额外空行呢，其实很简单，beetl 碰到只有 beetl 语句的行，无论是前有空格还是后有回车，都会忽略的，看过 beetl 在优化模式下生成的 class 代码就很容易能看出来。

但在一种情况下，模板尾部，Beetl 是无非判断是否空行的，如下：

```
${name}

<% …… beetl 语句%>
```

在第一行占位符 name 后面有个回车，beetl 无法判断这个回车是否是用户需要的回车还是仅仅为了其后的控制语句。目前情况下，beetl 认为用户需要输出空格行，因此，模板的输出会在尾部多出一个空格。

3.13.自定义错误处理

groupTemplate..setErrorHandler(h)用于设置你自定义的错误处理，譬如，默认情况下，错误处理会显示错误所在行，错误原因，以及模板上下3行的内容，如果你不希望客户看到你的模板内容，你可以自定义错误处理，请参考 org.bee.tl.core.DefaultErrorHandler

如果 groupTemplate.setErrorHandler，传入 null，则表示不希望 groupTemplate 处理错误，其后模板的任何错误都将抛出,如下例子：

```
try{

Template template = …… ;

template.set(…..);

t.getText (writer)

}catch(BeeException bee){

DefaultErrorHandler  h = new DefaultErrorHandler ();

h.processExcption(bee);

//or  打印到前台，h.processExcption(bee,writer);

throw new YourExcepion(bee.getMessage(),bee);

}
```

3.14.缓存字符串模板

字符串模板通常用于 CMS 系统，将模板页面生成静态页面，此时字符串模板来源于 CMS 的数据库。通常情况下，生成静态页面可以不缓存模板，但如果对性能也有要求，可以缓存字符串模板，可以调用

```

```

```
Template t = groupTemplate.getStringTemplate(input, key);
```

Input 是字符串模板，key 是此模板缓存的唯一标示，必须是合法的 java 类名（这样在允许编译成 class 的时候，此 key 就是 class 类名）。如果需要删除缓存，则调用如下语句即可

```
groupTemplate. removeTemplateCache(key);
```

除了 getStringTemplate 方法外，还可以调用 getReaderTemplate，输入是 java.io.Reader

3.15.布局

模板语言必须支持布局，否则就不是一个好的模板语言，Beetl 支持用标签函数 layout 做简单布局，也支持联合使用模板变量和 include 标签函数来做复杂布局。

Layout 布局请参考8.3的标签函数

复杂布局参考3.7的模板变量

3.16.测试模板

在后台没有准备好模型的情况下，如何验证模板是否正确呢，可以采用 json 变量来代替 pojo 的 java 对象来完成测试，beetl 扩展包提供了 org.bee.tl.ext.SimpleTemplateTestUtil 来帮助测试
如下代码：

```
String input = "this is template,${user.name},${sessions['userId']}";
String json = "var user = {'name':'joel'},sessions={'userId':'12345'};";
Writer w = new StringWriter();
SimpleTemplateTestUtil util = new SimpleTemplateTestUtil(input, json, w);
util.run();
System.out.println(util.isOk());
System.out.println(w);
```

输出是

```
true
this is template,joel,12345
```

可以看到尽管后台模型（user 对象，和 sessions 会话还没有准备好），模板仍然能够运行测试，如果模板有错误，错误也将输出 Writer 里

4. Spring MVC

4.1. 配置 ViewResolver

为了能在 Spring MVC 中使用 Beetl，必须配置 ViewResolver，如下

```
 <bean                                                id="beetlConfig"
class="org.bee.tl.ext.spring.BeetlGroupUtilConfiguration"
init-method="init">
            <property name="root" value="/"/>
            <property name="optimize" value="true"/>
            <property name="nativeCall" value="true"/>
            <property name="check" value="2"/>
      </bean>

      <bean                                           id="viewResolver"
class="org.bee.tl.ext.spring.BeetlSpringViewResolver">


      </bean>
```

Root 属性告诉 Beetl 模板文件未 WebRoot 的哪个目录下，通常是/ ,默认是/

optimize 属性允许优化，预编译成 class。默认是 true

nativeCall 运行本地调用,默认是 true

check 是每隔多少秒检测一下文件是否改变，设置较短时间有利于开发，在线上环境，设置为0，则不检查模板更新，默认是2秒

其他属性还有

tempFolder：预编译生成的源文件以及 class 的位置，默认是 WebRoot/WEB-INF/.temp 目录下

占位符指定：statementStart，statementEnd，placeholderStart，placeholderEnd 默认分别是 <% %> ${ }

charset：模板字符集，默认是 GBK

开发者也可以继承 BeetlGroupUtilConfiguration，实现 initOther 方法，为 GroupTemplate 增加更多的特性，如下代码

```
      public     class     MyBeetlGroupConfigration     extends
BeetlGroupUtilConfiguration
{
  Protected void initOther(){
       group.registerFunction(......)
  }
}
```

4.2. 模板中获取参数

在 Spring MVC 中，任何在 ModelMap 中的变量都可以直接在 Beetl 中引用，在 Session 中的变量，需要使用 session["变量名"]

如下 HelloWorldController 代码

```
@Controller
@SessionAttributes("currUser")
public class HelloWorldController {
      @RequestMapping("/hello")
      public ModelAndView helloWorld(ModelMap model ) {
            String message = "Hello World, Spring 3.0!";
            model.addAttribute("name","joel");
            model.addAttribute("currUser","libear");
            return new ModelAndView("/hello.html", "message",
message);
      }

}
```

则在模板中，访问 name,message，currUser 分别采用如下方式

${name},${message},${session["currUser"]}

除了用户设置的得变量外，其他能访问的变量还有

ctxPath ： web 应用的上下文环境路径

servlet：包含了 request,response,session

5. Struts2

只需要在 Struts 配置文件里增加如下配置，就可以使用 Beetl，此时 Beetl 的属性仍然是读取 classpath 的 beetl.properties.如果没有，就使用默认

```
<result-types>
    <result-type  name="beetl"  class="org.bee.tl.ext.struts2.Struts2BeetlActionResult"  default="true"/>

</result-types>
```

在视图层，可以直接访问 Action 定义好的属性，也可以访问如下变量。
session，可以获取 session 会话
request,是 HTTPRequest
ctxPath,是 request.getContextPath()

如果你需要初始化 GroupTemplate，如增加一些自定义方法等，可以创建一个心累继承 Struts2BeetlActionResult，并覆盖 public void initGroupTemplate(GroupTemplate group)
如果你还需要在模板渲染前做一些处理，可以覆盖public void addCommonProperty(Template t, HttpServletRequest req,HttpServletResponse rsp)

6. JFinal

Jfinal 是国内很有潜力的 Web 开发框架，Beetl 可以很容易作为其框架的视图层技术。代码如下

```
import org.bee.tl.ext.jfinal.BeetlRenderFactory
public class DemoConfig extends JFinalConfig
{
      public void configConstant(Constants me)
      {
```

```
            me.setDevMode(true);
            me.setMainRenderFactory(new BeetlRenderFactory());

    }                                       }
```

默认情况下 BeetlrednerFacotry 会装载 classpath 路径/ beetl.properties 的配置，配置里未指定属性 TEMPLATE_ROOT 的值，这 BeetlrednerFacotry 默认的模板根目录在 WebRoot/WEB-INF/template 目录下。如果指定了，譬如 TEMPLATE_ROOT=/beetl/template 则模板跟目录在 WebRoot/WEB-INF/beetl/template.

在控制层，通过 setAttr 方法设置的变量，可以再在模板文件中直接引用，模板中还内置了如下变量
session，可以获取 session 会话
request,是 HTTPRequest
ctxPath,是 request.getContextPath()

如下模板例子

```
Hello，${session[“user”]}

<%

for(order in orderList){}

%>

<image src=${ctxPath}/images/user?${@request.getParameter(“action”)}>
```

7. 在 Servlet 中使用 Beetl

7.1. 配置 Linstener

无论在何种框架下使用 Beetl,都必须保证 GroupTemplate 是单例，且被正确初始化，所以 Beetl 扩展包提供了一种在 Servlet 中使用 Beetl 的例子，使用用户监听器初始化 GroupTemplate，如下代码

```
public class MyTestListener implements ServletContextListener {


    public void contextInitialized(ServletContextEvent arg0) {

       ServletGroupTemplate.intance().init(arg0.getServletConte
xt());
    }

    public void contextDestroyed(ServletContextEvent arg0) {

    }


}
```

ServletGroupTemplate 位于扩展包 org.bee.tl.ext 下，此类将从 web.xml 读取如下变量初始化 GroupTemplate:

GroupTemplate.Root 属性告诉 Beetl 模板文件未 WebRoot 的哪个目录下，通常是/,默认是/

GroupTemplate.Optimize 属性允许优化，预编译成 class。默认是 true

GroupTemplate.NativeCall 运行本地调用,默认是 true

GroupTemplate.Check 是每隔多少秒检测一下文件是否改变，设置较短时间有利于开发，在线上环境，设置为0，则不检查模板更新，默认是2秒

GroupTemplate.DirectByteOutput 设置为 true,允许 byte 输出从而提升性能,默认是 false

其他属性还有

GroupTemplate.TempFolder：预编译生成的源文件以及 class 的位置，默认是 WebRoot/WEB-INF/.temp 目录下。但是由于部署方式以及 web 服务器不一样，如使用 war 方式部署，Beetl 不能从 ServletContext 得到正确的绝对路径。你最好看看是否是在默认目录下，否则，请指定一个绝对路径。

占位符指定：GroupTemplate.StatementStart，GroupTemplate..StatementEnd，GroupTemplate.PlaceholderStart，GroupTemplate.PlaceholderEnd 默认分别是 <% %> ${ }

GroupTemplate.Charset：模板字符集，默认是 GBK

因此 Web.xml 文件看起来像这样

```
  <listener>
    <listener-class>com.MyTestListener</listener-class>
  </listener>
  <context-param>
    <param-name>GroupTemplate.Root</param-name>
    <param-value>/</param-value>
  </context-param>
  <context-param>
    <param-name>GroupTemplate.Optimize</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <param-name>GroupTemplate.NativeCall</param-name>
    <param-value>true</param-value>
  </context-param>
  <context-param>
    <param-name>GroupTemplate.Check</param-name>
    <param-value>2</param-value>
 </context-param>
```

7.2. 编写 Servlet

只需要在 Servlet 中调用 ServletGroupTemplate.instance().getTemplate(child,request,response)就可以获取 Template

如下代码

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
		request.getSession().setAttribute("user", "joelli");
		Template t = ServletGroupTemplate.intance().getTemplate("/hello.html", request, response);
		t.set("message", "hello");
		try {
			response.setCharacterEncoding("UTF-8");
			t.getText(response.getWriter());
		} catch (BeeException e) {
			//既然没有设置错误处理器，GroupTemplate将使用默认处理，此异常不会被抛出
		}


	}
```

getTemplate 将会获取 child 对应的模板，并还做如下赋值操作

将 Session 赋值给 session,将 ContextPath 赋值给 ctxPath, 以及所有 request 中的属性都赋值给 Template，因此可以在模板使用
${session["user"]} ,${ctxPath},${meesage} 来访问 Servelt 中赋值的变量

8. 附录：扩展包

扩展包现在提供的功能有限，主要是提供常用扩展，以及作为例子参考。如果你有很好的扩展，可以参与到扩展包的开发来

8.1. 函数

函数名称	使用说明
date	返回一个 java.sql.Date 类型的变量，如 ${date()} 返回一个当前时间 ${date('2011-1-1','yyyy-MM-dd')} 返回指定日期

print	打印一个对象 <% print(user.name);%>
println	打印一个对象以及回车换行符号，回车换号符号使用的是模板本身的，而不是本地系统的 <% print(user.name);%>
printf	格式化输出，如 <%printf("Hello, %s. Next year, you'll be %d", name, age)%> 具体格式请参考 http://docs.oracle.com/javase/6/docs/api/java/util/Formatter.html
nvl	函数 nvl，如果对象为 null，则返回第二个参数，否则，返回自己 ${nvl(user,"不存在"}
debug	在控制台输出 debug 指定的对象以及所在模板文件中的行数，采用 System.out.println("<line "+line+">"+debugObject.toString); <% debug("whatever");%>
exist	判断全局变量是否存在（不能判断临时变量） <%if(exist(“user”,”sessions”)){……}%>
assert	如果表达式为 false，则抛出异常，异常信息可以第二个参数指定 assert(exist(“userList”)),或者 assert(exist(“userList”),”缺少用户信息”); Beetl 将不再渲染模板，退出
trunc	截取数字，保留指定的小数位，如 ${trunk(12.456,2)} 输出是 12.45 ${trunk(12.456,0)} 输出是 12 ${trunk(12.456)} 输出是 12
decode	一个简化的 if else 结构，如 ${decode(a,1,"a=1",2,"a=2","不知道了")} 如果 a 是 1，这 decode 输出"a=1",如果 a 是 2，则输出"a==2", 如果是其他值，则输出"不知道了"

函数名称	使用说明
strutil.startWith	${ strutil.startWith(“hello”,”he”) 输出是 true
strutil.endWith	${ strutil.endWith(“hello”,”o”) 输出是 true
strutil.length	${ strutil. length (“hello”),输出是 5
strutil.subString	${ strutil.subString (“hello”,1),输出是“ello”
strutil.subStringTo	${ strutil.subStringTo (“hello”,1,2),输出是“el”
strutil.split	${ strutil.split (“hello,joeli”,”,”),输出是数组，有俩个元素，第一个是 hello，第二个是 joelli”
strutil.contain	${ strutil.split (“hello,”el”),输出是 true
strutil.toUpperCase	${ strutil.toUpperCase (“hello”),输出是 HELLO
strutil.toLowerCase	${ strutil.toLowerCase (“Hello”),输出是 hello
strutil.replace	${ strutil.replace (“Hello”,”lo”,”loooo”),输出是 helloooo
strutil.format	${ strutil.format (“hello,{0}, my age is {1}”,”joeli”,15),输出是 hello,joelli, my age is 15. 具体请参考 http://docs.oracle.com/javase/6/docs/api/java/text/MessageFormat.html

8.2. 格式化函数

函数名称	使用说明
dateFormat	日期格式化函数，如 ${date,dateFormat='yyyy-Mm-dd'} 等于符号后的参数也可以没有，则使用本地默认来做格式化 如 ${date,dateFormat}
numberFormat	${0.345,numberFormat="##.#%"} 输出是 34.5%,具体请参考文档 http://docs.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html

8.3. 标签

函数名称	使用说明
deleteTag	此标签体的内容将不作输出
includeFileTemplate	include 一个模板，如 <%includeFileTemplate("/header.html"){}%> 如果想往子模板中传入参数，则可以后面跟一个 json 变量，如 include 一个模板，如 <%includeFileTemplate("/header.html",{'user',user:'id',user.id}){}%>
inlucde	同上
layout	提供一个布局功能，每个页面总是由一定布局，如页面头，菜单，页面脚，以及正文 layout 标签允许为正文指定一个布局，如下使用方式 content.html 内容如下： <%layout("/org/bee/tl/samples/layout.html"){%> this is 正文 ..... <%%}%> layout.html 是布局文件，内容如下・ this is header this is content:${layoutContent} this is footer:

	运行 content.html 模板文件后，，正文文件的内容将被替换到 layoutContent 的地方，变成如下内容 this is header this is content:this is 正文 ..... this is footer: 如果想往 layout 页面传入参数，则传入一个 json 变量，如下往 layout.html 页面传入一个用户登录时间 <%layout("/org/bee/tl/samples/layout.html",{'date':user.loginDate}){%> this is 正文 ..... <%%}%>

cache	能 Cache 标签的内容，并指定多长时间刷新，如
	<%:cache('key2',10,false){ %>
	内容体
	<%%>
	需要指定三个参数，第一个是 cache 的 Key 值，第二个是缓存存在的时间，秒为单位，第三个表示是否强制刷新 •，false 表示不，true 表示强制刷新
	Cache 默认实现 org.bee.tl.ext.cache.SimpleCacheManager. 你可以设置你自己的 Cache 实现，通过调用 CacheTag. cacheManager= new YourCacheImplementation();
	
	可以在程序里调用如下方法手工删除 Cache：
	
	public void clearAll();
	
	public void clearAll(String key);
	
	public void clearAll(String... keys);

8.4. Freemarker 功能对比

参考了 Freemarker 官方文档（2.3.19） 第一列是官方文档目录，第二列是表示 beetl 中是否具有同样的功能
通过下列表格，可以看出绝大部分功能，beetl 都是支持的。

功能	是否支持	备注
1.1 简介	支持	

1.2模板 + 数据模型 = 输出	支持	
1.3 数据模型一览	支持	
1.4.1 简介	支持	
1.4.2 (.1--.4) 指令示例	支持	Beetl 中，采用includeFileTemplate标签来include一个文件。Beetl 中还支持switch/case
1.4.2.5 处理不存在的变量	支持	不支持询问变量是否存在,但可以通过扩展函数来支持
2.1 基本内容	支持	
2.2.1 类型 简介	大部分支持	很少使用的**节点类型**不支持
2.2.2 标量	支持	
2.2.3 容器	支持	
2.2.4 方法和函数	支持	可以通过扩展函数来间接支持
2.2.5 其它（节点类型）	不支持	
3.1 总体结构	支持	
3.2 指令	支持	
3.3 表达式	支持	集合操作不支持，但可以通过扩展函数来实现集合操作。 截取字符串也不是内置的，但可以通过扩展

		函数来实现，如 str.subString
3.3.3.1 字符串	支持	
3.3.3.2 数字	支持	
3.3.3.3 布尔值	支持	
3.3.3.4 序列	支持	采用 js 语法的 json 格式支持。但不包括支持 *start..end 这样的方式*
3.3.3.5 哈希表	支持	
3.3.4 检索变量	支持	
3.3.5 字符串操作	不支持	此语法很奇怪，只能说 freemaker 复杂了语法
3.3.5.2 获取一个字符	支持	通过扩展函数支持
3.3.6.1 序列操作 连接	支持	通过扩展函数支持
3.3.6.2 序列切分	支持	通过扩展函数支持
3.3.7 哈希表操作 连接	支持	
3.3.8 算数运算	支持	
3.3.9 比较运算	支持	
3.3.10 逻辑操作	支持	
3.3.11 内建函数	支持	
3.3.13 处理不存在的值	支持	通过扩展函数支持

3.3.13.1 默认值	部分支持	
3.3.13.2 检测不存在的值	支持	
3.3.14 括号	支持	
3.3.15 表达式中的空格	支持	
3.3.16 操作符的优先级	支持	无数字范围 这个概念
3.4 插值	支持	
4.1 自定义指令 （宏）	支持	间接通过标签等支持
4.1.4 嵌套内容	支持	Beetl 中主要用在 layout 标签里
4.1.5 宏和循环变量	支持	
4.2 在模板中定义变量	支持	
4.3 命名空间	部分支持	通过扩展函数支持，
4.4 空白处理	支持	Beetl 中不需要此额外功能
4.5替换（方括号）语法	支持	

8.5. Freemarker 性能比较

8.5.1.单线程：

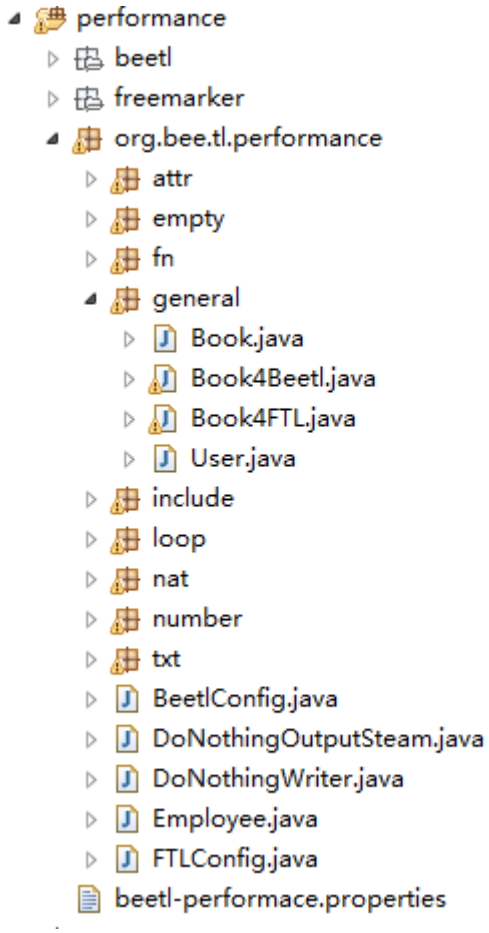
还是以常用模板为准（大小 6K），循环渲染 50000 次，需要时间（毫秒为单位）如下

Beetl1.2M1 解释执行方式（普通模式），运行三次，分别 1356,1365,1348
Beetl1.2M1 编译执行方式（普通模式），运行三次，分别 913,922,905
Freemarker 分别是 1155,1130,1122
Beetl1.2M1 允许字节流优化，解释方式执行三次，分别是 587,605,610
Beetl1.2M1 允许字节流优化，编译方式执行三次，分别是 385,355,370

简而言之，对 beetl 做运行时编译，byte 输出设定后，渲染 5 万次 6K 的模板，性能如下

Freemarker	Beetl
1140 毫秒	370 毫秒

及时不对 beetl 做任何优化，性能也与 Freemarker 持平.如下是单项测试的性能比较，数据不再列出，没项性能测试都超过了 Freemarker



8.5.2.并发

9. Beetl 历史

更新时间	更新内容
2012-11-18	重构完成，发布了1.2beta 版本
2012-10-24	文档完善和 bug，修复了一个会产生多余空格的轻微 bug
2012-10-5	性能优化，添加错误处理文档说明
2012-9-25	测试模板，修复 bug
2012-9-23	字符串模板
2012-9-9	安全输出做了调整，可以应用于表达式中
2012-8-25	通过使用 Config 来简化创建 GroupTemplate
2012-6-10	Beetl1.2M1发布，性能优化，文本合并输出，二进制直接输出
2012-1-8	Beetl1.1发布，编译 class 完善
2011-10-1	Beetl1.0发布，运行中编译成

	class
2011-6-1	Beetl 的0.6一个初始化版本发布
2011-4-1	开始设计和开发 beetl
2010-12-1	Java 模板引擎那么多，程序员必须用这些难用的模板引擎么，无法选择