**Velocity 及其指令简介**

Velocity 是一个基于 Java 的模板引擎，它允许用户使用简单的模板语言来引用由 Java 代码定义的对象。当 Velocity 应用于 Web 开发时，界面设计人员可以和 Java 程序开发人员同步开发一个遵循 MVC 架构的 Web 站点。也就是说，页面设计人员可以只关注页面的显示效果，而 Java 程序开发人员关注后台业务逻辑的编码。 Velocity 将 Java 代码从 Web 页面中分离出来，这样为 Web 站点的长期维护提供了便利，同时也为我们在 JSP 和 PHP 之外又提供了一种可选的方案。

Velocity 的能力不仅仅用于 Web 开发领域，它也可以被当作一个独立工具来产生源代码和报告（例如，可以产生 SQL 和 PostScript、XML 等），或者作为其他系统的集成组件使用。

下面是一个简单的用 Velocity 编写的网页代码：

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<meta http-equiv="Content-Language" content="zh-CN"/>

<title>$page\_title</title>

<link rel="stylesheet" href="osc-global.css" type="text/css" />

</head>

<body>

<div id="OSC\_top">

#parse("header.vm")

</div>

<div id="OSC\_content">

<table>

#foreach($idx in [1..20])

<tr>

<td>$velocityCount</td>

<td>Now is $date.now()</td>

</tr>

#end

</table>

</div>

<div id="OSC\_bottom">

#include("bottom.vm")

</div>

</body>

</html>

在 Velocity 模板语言的语法中，以美元符 $ 开头的为变量的声明或者引用，而以井号 # 开头的语句则为 Velocity 的指令（Directive）。其中 Velocity 的指令又分为内置指令、自定义宏和自定义指令。 Velocity 包含下列这些内置指令：

| **#set** | **赋值指令** |
| --- | --- |
| #if/#elseif/#end | if 条件判断指令 |
| #foreach | 循环指令 |
| #include | 嵌入静态内容 |
| #parse | 嵌入动态内容 |
| #stop | 停止执行下面代码 |
| #evaluate | 执行指令 |
| #define | 定义脚本 |
| #marco | 宏定义 |

另外也可以通过宏来扩展指令，例如，请看下面这个宏的定义：

#macro(invoke $\_\_p\_page)

#if($\_\_p\_page.startsWith("/"))

#parse($\_\_p\_page)

#else

#set($\_\_uri = $resource.this\_vm())

#set($\_\_path = $\_\_uri.substring(0, $\_\_uri.lastIndexOf("/")))

#parse("$\_\_path/$\_\_p\_page")

#end

#end

上面这个名为 invoke 的宏的作用是以相对路径的方式嵌入某个动态的页面。使用的方法是 #invoke( “ hello.vm ” )。尽管 Velocity 的自定义宏可以用来扩展指令，但是宏有一个不足的地方，它更适合用来执行一些比较通用的代码嵌入和简单的功能处理。

由此引出来的问题是，我如何来编写一个类似于 #foreach 的指令呢，并具有逻辑判断的功能？

**Velocity 的指令类型简介**

在 Velocity 的指令定义上，有两种指令类型分别是行指令和块指令。行指令例如 #set($name= ” Winter Lau ” ) 赋值指令，只有一行，中间没有任何的代码；而块指令例如循环指令 #foreach($idx in [1..20]) $idx #end，块指令需要用 #end 来结束。在 Velocity 自带的指令中块指令包括有：#if #elseif #foreach #define 和 #macro 这几个指令，除此之外都是行指令。

**编写自定义的 Velocity 指令**

Velocity 允许您对指令系统进行扩展，在 Velocity 引擎初始化的时候会加载系统内置指令和用户的自定义指令。系统的内置指令已经在 Velocity 的 Jar 包中的 directive.properties 文件中定义，不建议直接修改该文件。而自定义的指令要求用户在 velocity.properties 文件中定义的，例如：userdirective=net.oschina.toolbox.CacheDirective。如果是多个自定义指令则使用逗号隔开。

所有的自定义指令要求扩展 org.apache.velocity.runtime.directive.Directive 这个类。为了更加形象直观的表现 Velocity 自定义指令的优点，接下来我们将以一个实际的应用场景进行讲解。

在该应用场景中，所有的页面请求直接指向 vm 文件，中间没经过任何的控制器。数据是通过 Velocity 的 toolbox 直接读取并显示在页面上。如果数据是来自数据库的，而且访问量非常大的时候，我们就需要对这些数据进行缓存以便快速响应用户请求和降低系统负载。一种方法是直接在 toolbox 的读取数据的方法中进行数据的缓存；另外一种就是我们接下来要介绍的，通过编写自定义的缓存指令来缓存页面上的某个 HTML 片段。

首先我们定义一个这样的块指令：#cache( “ CacheRegion ” , ” Key ” ) ，其中第一个参数为缓存区域、第二个参数为对应缓存数据的键值。该指令自动将包含在指令内部的脚本执行后的结构缓存起来，当第一次请求时检查缓存中是否存在此 HTML 片段数据，如果存在就直接输出到页面，否则执行块指令中的脚本，执行后的结果输出到页面同时保存到缓存中以便下次使用。使用方法如下所示：

#cache("News","home")

## 读取数据库中最新新闻并显示

<ul>

#foreach($news in $NewsTool.ListTopNews(10))

<li>

<span class='date'>

$date.format("yyyy-MM-dd",${news.pub\_time})

</span>

<span class='title'>${news.title}</span>

</li>

#end

</ul>

#end

其中 $NewsTool.ListTopNews(10) 是用来从数据库中读取最新发布的 10 条新闻信息。

接下来我们来看 #cache 这个指令对应的源码：

/\*\*

\* Velocity模板上用于控制缓存的指令

\* @author Winter Lau

\* @date 2009-3-16 下午04:40:19

\*/

public class CacheDirective extends Directive {

final static Hashtable<String,String> body\_tpls = new Hashtable<String, String>();

@Override

public String getName() { return "cache"; } //指定指令的名称

@Override

public int getType() { return BLOCK; } //指定指令类型为块指令

/\* (non-Javadoc)

\* @see org.apache.velocity.runtime.directive.Directive#render()

\*/

@Override

public boolean render(InternalContextAdapter context, Writer writer, Node node)

throws IOException, ResourceNotFoundException, ParseErrorException,

MethodInvocationException

{

//获得缓存信息

SimpleNode sn\_region = (SimpleNode) node.jjtGetChild(0);

String region = (String)sn\_region.value(context);

SimpleNode sn\_key = (SimpleNode) node.jjtGetChild(1);

Serializable key = (Serializable)sn\_key.value(context);

Node body = node.jjtGetChild(2);

//检查内容是否有变化

String tpl\_key = key+"@"+region;

String body\_tpl = body.literal();

String old\_body\_tpl = body\_tpls.get(tpl\_key);

String cache\_html = CacheHelper.get(String.class, region, key);

if(cache\_html == null || !StringUtils.equals(body\_tpl, old\_body\_tpl)){

StringWriter sw = new StringWriter();

body.render(context, sw);

cache\_html = sw.toString();

CacheHelper.set(region, key, cache\_html);

body\_tpls.put(tpl\_key, body\_tpl);

}

writer.write(cache\_html);

return true;

}

}

Directive 是所有指令的基类，Directive 是一个抽象类，它有三个方法必须实现的，分别是：

* getName：返回指令的名称
* getType：返回指令的类型，行指令：LINE、块指令：BLOCK
* render：指令执行的入口

其中 render 方法的最后一个参数 node 表示为该指定对应在 Velocity 模板中的节点对象，通过调用 node 的 jjtGetChild 方法可以获取到传递给该指令的参数以及包含在该指令的脚本内容。

上面的代码中，首先获取传递给指令的参数，也就是缓存的区域名和对应缓存数据的键值。接着判断距上次数据被缓存时，指令所包含的脚本代码是否有更改（以便页面开发人员修改了 vm 脚本时自动刷新缓存数据），然后判断缓存中是否已有数据。当缓存中无数据或者页面代码被修改时，重新执行块指令中的脚本并将执行的结果置入缓存，否则直接将缓存中的数据输出到页面。

上述例子中，传递给 #cache 指令的参数也可以是某个变量，例如

#set($region = "news")

#set($key = "home")

#cache("CACHE\_$region",$key)

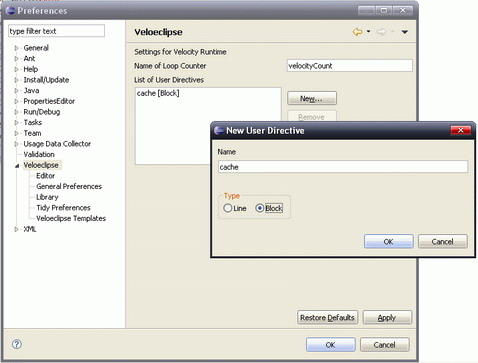
如此，便以很小的代码侵入，来实现页面的缓存。

**自定义指令在 Veloeclipse 插件中的使用事项**

Veloeclipse 是一个在 Eclipse 开发环境中编辑 Velocity 模板和 HTML 代码的插件，具有语法着色、自动代码完成以及错误提示等功能。如果在编辑 Velocity 模板时使用了自定义插件，则该插件会提示错误，我们可以通过下面的界面添加自定义的指令来使 Veloeclipse 支持这些指令。

点击菜单 **Windows - Preferences - Veloeclipse** ，通过如下的界面来添加自定义指令。

**图 1. 添加自定义指令的界面**



**总结**

Velocity 是一个非常高效、简洁的 Java 模板引擎，而且其强大的扩展性，使之特别适合在 Web 项目中使用。本文通过一个实际应用中的例子对 Velocity 的指令系统进行了介绍，欢迎大家跟我一起深入探讨 Velocity 的相关扩展问题。