《Velocity1.4 模板使用指南中文版》中文版

源文见 http://velocity.apache.org

本文属 NetJava.cn 中的 Velocity 中文系列，本系包含如下文章：
《Velocity Java 开发指南中文版》(Developer`s Guide)
《Velocity 模板使用指南中文版》(User`s Guide)
《Velocity Web 应用开发指南中文版》(Web Application Guide)
《VTL 语法参考指南中文版》(VTL Reference)
《DB4O 中文系列之起步篇》
. . .
更多资料请访问 http://www.netjava.cn/ 下载.

****

译者：javaFound
Mail: javafound@gmail.com
NetJava.cn@gmail.com

****

目　录

1. 本文目地和使用对象

本文主要介绍如何在模板中使用 Velocity 功能强大的模板语言 VTL(Velocity Template Language)用法有一个全面的认识，并掌握如何在模板中有效使用。同时，本文提供较多的例子帮您来学习它。.

感谢您选择 Velocity 帮助您实现纯正的 MVC 系统构架。

2.什么是 Velocity?

Velocity 是一个基于 java 的模板引擎(template engine). 它可以让视图的设计者在 web 页面中引用 java 代码中定义的数据对象和命令。从而使 Web designers 和 java 开发者依照 MVC 思想(Model-View-Controller)开发系统,这意味着 Web designers 只须将精力注用于良好表现力的视图外观设计，而Java 程序员则只要关心着如何写出高效简洁的 java 对象以实现业务逻辑-----Velocity 会将他们组装到一起. 相比传统的 jsp、PHP 等脚本语言，Velocity 彻底的将避免了在视图设计中出现的 java 代码, 从而保证了 web site 的长期可维护性.

一定要理解，Velocity 是一个 template engine 的意思，它还可以从模板中生成 SQL 语句或其它脚本提供给 web pages. 它也可以独立使用---做为一个工具类(utility class)用来生成源代码、报表、邮件模板等---在有需要重复的视图情况下，你应想到使用 Velocity.Apache 站点提供的另外一个框架 Turbine 可以和 Velocity 有效结合以实现 true MVC model.

3.Velocity 能为你做什么?

1.一个 Mud Store Example

假设你是一个 page designer 在为一个在线商店设计页面. 我们称这个项目为 "The Online Mud Store".业务发展还不错，客户会订购不同类型的 MuD,每个客户都会用自己的帐号密码 login,查看选择他们订购的 MuD,查看订单，但还有些忠诚用户会购卖不太流行的 MuD----这些不需要出现在页面中显眼的地方。当然，The Online MuD Store 必须把每个客户资料及订购信息记录到 DB 中，现在的问题是，如何让某个客户 login 后就看到他感兴趣的信息?

使用 Velocity!我们为每个客户定制一个页面！这听起来工作量巨大，让我们试试：.

使用 Velocity 的 VTL 如下来设计 web page:

```
<HTML>
<BODY>
##指定用户名字
欢迎你： $customer.Name!
<table>
```

```
###输出用户喜好的 MuD
#foreach( $mud in $mudsOnSpecial )
   #if ( $customer.hasPurchased($mud) )
      <tr>
        <td>
          $flogger.getPromo( $mud )
        </td>
      </tr>
   #end
#end
</table>
```

使用 VTL 设计页面就是这么简单！文档《VTL 参考中文版》中有更全面的 VTL 语言介绍，掌握这些，你将会全面体会到 Velocity 的威力.

4.Velocity Template Language (VTL): 介绍

The Velocity Template Language (VTL)目标是提供一个简洁，易学的方法将动态内容展现到 web page 上. a web page 设计者可以没有任何编程经验就可以在一天内学会使用它增强你的站点的展示力！.

VTL 使用引用(*references*)这种方式将 dynamic content(动态内容，一般指 java 代码生成的数据对象)加入到你的 web site,Velocity 中的变量(variable)只是 refernce 中的一种. Variables 是用来描述从引入到视图模板中的 java 数据对象。当然，java 代码也可以从模板的 VTL 中获取数据.以下是一个写在 HTML 中的 VTL 变量:

```
#set( $a = "Velocity" )
```

VTL 声明(statement),所以的 VTL statement 都是以#开头，且包含一个指示符（这里是 *set*）,当客户访问你的页面时, the Velocity Templating Engine 将搜索页面中的所有 # 符号,如果确定这是一个 VTL 声明时就按一定规则处理动态内容, 符号#仅仅只是表明这可能是一个 VTL 声明.

符号# 所跟的 *set* 我们用“指示符”这一名词来称呼它(随后介绍更多的指示符), *set* 指示符使用一个表达式(expression) (包含在一对括号里) - 将一个值 *value* (这里是 Velocity)付给变量 a,（变量名在左边，值在右边，用=组合起来）.

在以上的例子中,变量是 *a* ,而符号“$”表明它是一个变量,Velocity 中所有变量以符号”$”开头，所付的值要用双引号括起， 这个值中还可以再添加 Velocity 变量，如"Hello $name"，输出的将是 name 变量所付的值。

这是理解 VTL 基础的规则:

以$开头的表示“引用”意思是取得一些东东.而”指示”(Directives)则以#开头来表示，有点“做些什么动作”的意思.

如上便, *#set* 用来指定值给一个变量名*$a*, 以“$”标示的变量名 a 的值就是"Velocity".

5.输出第一个 VTL 页面!

有你的 HTML 文档的任何地方，都可以引用一个变量名来输出值，如下例，先给变量名 *foo* 赋值为 Velocity,然后将它输出到页面中.

```
<html>
<body>

#set( $foo = "Velocity" )
Hello $foo World!
</body>
<html>
```

在这个页面上，你看到的将是 "Hello Velocity World!".

为了让查板中的 VTL 指令更易读，我们强烈建议你每行一条 VTL 指令,当然这不是必须的. 关于 *set* directive 的更多功能我们随后再讨论.

6.Comments(注释)

Comments 可以让你在模板中包含对 VTL 或其它问题的说明描述以便与阅读和理解.---但它并不会在最终输出的 web pages 中看到.如下示例是 VTL 中的一行注释.

```
## This is a single line comment.
```

单行注释是以## 开头的一行文字.如要写下多行注释，就要像下面那样，将它们放入#* 和*#间:

```
This is text that is outside the multi-line comment.
Online visitors can see it.

#*
 Thus begins a multi-line comment. Online visitors won't
 see this text because the Velocity Templating Engine will
 ignore it.
*#
```

不需要太复杂了，这两种方式已足够你给自己的页面加上充分的说明。

7.References(引用)

VTL 中有三种 references:变量引用(variables),属性引用(properties)和命令引用(methods). 做为一个使用 VTL 的设计者，你和你的 java 软件工程师必须就模板中引用的特定名了(就是$后的名字)达成一致的协议！这样，模板和 java 代码才可按照你们的意图去结合以输出正确的内容.
所有的引用在模板中都表现为一个字符串. 假设一个引用变量 *$foo* 的值事实上是一个 int, Velocity

engine 在处理时将调用它的.toString() 去解析这个字符串所代表的对象(int).

1.Variables(变量)

简单的说变量以"$" 开后，后面跟一个 VTL 指示符(Identifier).一个合法的 VTL 指示符是以字母开头,后面可以是以下任意字符:
alphabetic (a .. z, A .. Z)
numeric (0 .. 9)
hyphen ("-")
underscore ("_")
以下是正确的 VTL 变量名:
$foo
$mudSlinger
$mud-slinger
$mud_slinger
$mudSlinger1
当 VTL 中的一个变量如 *$foo*, 这个变量可以在模板中提取它自己的值通过 *set* 指示符,或者从 java 代码中.比如，如果需 java 中的变量 foo 的值为 bar,给模板中所有输出$foo 声明的值都成为 bar.当然，在模板中使用如下 VTL 也可以达到这个目地。
#set($foo = "bar")

2.Properties(属性)

VTL 的第二种引用，属性引用,注意”属性”具有相对固定的格式. 它也是以$开头的合法 VTL 指示符,随后是”变量名字.变量属性”. 如下例:
$customer.Address
$purchase.Total
第一个例子， *$customer.Address*.我们设想可能在两种意思.首先它可能在查找 customer 引用的一个 hashtable 中的以“Address”为 key 关联的一个数据对象.另外他可能表示的是 java 对象 customer 中的 getAddress()这个命令取得的结果(当然也可写成 *$customer.getAddress()*). 当客户请求 Web 页面中 Velocity 将根据具体的 customer 类型输出.

3.Methods（命令）

在 java 代码中定义命令是最通常的事, 像执得一组方程式计算成本，读取一个文件等. 命令引用和其它引用一个也是一个一般的 VTL 声明,看如下的例子可能会更快的理解:

```
$customer.getAddress()
$purchase.getTotal()
$page.setTitle( "My Home Page" )
$person.setAttributes( ["Strange", "Weird", "Excited"] )
```

前两个例子 -- *$customer.getAddress()* and *$purchase.getTotal()* - 可以等同与属性引用情况：*$customer.Address* and *$purchase.Total*. 如你猜对了这点，呵，你还蛮聪明!同时，他们是等同与 java

对象 coustomer 的 getAddress（）命令调用。 后面两个引用呢，也会直接对应 java 对象的对应命令，不同的是传入了命令参数。

4.属性引用中的属性查找规则

前已提及,属性可以引用到对象的命令. Velocity 会使用合适的策略选择引用到的命令. 它会根据协定的命令命令格式查找. 无论属性引用的的名字是否大小写，Velocity 都有固定的查找规则.如在 *$customer.address* 引用时，查找顺序是：

1. getaddress()
2. getAddress()
3. get("address")
4. isAddress()

对于ＶＴＬ中大写的属性名 Address 引用，将是:

1. getAddress()
2. getaddress()
3. get("Address")
4. isAddress()

8.Formal Reference Notation(正规引用格式注意事项)

以上是简洁格式引用的介绍，规则的格式像下面，如你看到的变量名需要放到{}中:

```
${mudSlinger}
${customer.Address}
${purchase.getTotal()}
```

大多数情况下，简洁格式引用足以满足使用,但有些情况下，必须使用正规格式引用.

假设你构造的一个字符串中要包括有 *$vice* 变量的值. 你觉得以下两种写法会是同样的结果吗:

```
1.Jack is a $vicemaniac.
2.Jack is a ${vice}maniac.
```

这样，Velocity 就知你要的是 *$vice*, 而不是 *$vicemaniac* 变量，正规引用格式一般用于在模板中直接调整字符串内容.

9.Quiet Reference Notation(静态引用输出)

Velocity 遇到一个不能处理的引用时，一般他会直接输出这个引用$email 的写法,页面上会看到的是 $email，如下例,我们可以在$后面加上一个!号，那么就会输出空白:.

```
<input type="text" name="email" value="$email"/>

<input type="text" name="email" value="$!email"/>
```

正式的写法是:.

```
<input type="text" name="email" value="$!{email}"/>
```

10.Getting literal(语义问题)

一般情况下，velocity 使用$,#字符来标志它的声明，但有时，ＨＴＭＬ中因为某种其它意图，也会写出这样的字符，我们讨论如何消除这种语义歧义问题.

1. Currency(货币标志)

如美元 $2.50!这样的写法出现到模板中, VTL 处理时是不会出错，会正确的输出$2.50!这个你想要的结果。为什么呢？一个合法的 VTL 标示符是以一个字母开头的，我们前面已说过.

2. Escaping Valid VTL References(封装有效的引用)

如下示，如果没有#set($email = "foo")这一行且 java 代码中 Context 对象中没有放放 email 对象，将直接输出$email.

```
#set( $email = "foo" )
$email
```

如果 email 已定义了 (比如它的值是 *foo*),而这里你却想输出 *$email*. 这样一个字符串,就需要使用转义字符”\”.

```
## The following line defines $email in this template:
#set( $email = "foo" )
$email
\$email
\\$email
\\\$email
```

上面的模板在 web 页面上的输出将是:

```
foo
$email
\foo
\$email
```

但如果 email 并没有定义，我们这样写：.

```
$email
\$email
\\$email
\\\$email
```

输出就原封不动了：

```
$email
\$email
\\$email
\\\$email
```

注意：当已定义变量和未定义变量一起输出时，会输出字面意思,如下便，$moon 是未定义的：

```
#set( $foo = "gibbous" )
$moon = $foo
```

输出到 web 页面中将是

$moon = gibbou

11.Case Substitution(可选的格式)

至此，你对 velocity 的 refenerce 已比较熟悉了，你可以在你的模板中开始应用这些功能. 但你还可以知道的是 Velocity references 从 java 语法中汲取了一些优点以便模板设计者更容易使用 VTL.比如:

```
$foo

$foo.getBar()
## 等同于
$foo.Bar

$data.setUser("jon")
##等同于
#set( $data.User = "jon" )

$data.getRequest().getServerName()
##等同于
$data.Request.ServerName
## is the same as
${data.Request.ServerName}
```

这里示例了你可选的一些引用方式. VTL 汲取了 java 语法和 java bean 的一些简洁语法以解析 java 代码中 Context 中的对象和这些对象的命令及属性---这样，一个 java 对象的所有功能都可以展示到视图中了.

Velocity 也借见了 java Bean 的规范(Bean specifications defined by Sun Microsystems), 是大小写敏感的; 但 Velocity 会尽可能的帮你修正错误. 当命令 *getFoo()* 通过指令 $bar.foo 在模板中引用时,Velocity 的搜索规则我们在前面已讲了，你还记得是什么吗？.

注意:*模板中引用的必须是通过 java Bean 中的* getter/setter 实现的，而直接的 java 对象的数据域是不能直接引用的,如$foo.Name 会解析到 class Foo's getName() 的实例方法,但不会解析到 Foos 类的 public Name 这个实例变量.

12.Directives(指令符号)

模板设计者使用“引用“生成动态内容, 指令(*directives*) - 简单的说就是设计者在模板中操作 java 对象—让视图设计者全面控制输出内容的格式.

指令总是以 #开头后面紧跟具体的指令符. 就像引用一样（指令的一种）,可以将指令理解为”表示这里是一个什么东东).如下例生成一个出错提示:

```
#if($a==1)true enough#elseno way!#end
```

这个例子中应使用括号将 else 分开.

```
#if($a==1)true enough#{else}no way!#end
```

1.#set 指令

#set 用来给一个引用赋值.值可以被赋给变量引用或属性引用, 但要将它们放入括号中，如下示:

```
#set( $primate = "monkey" )
#set( $customer.Behavior = $primate )
```

“左操作数被赋值“是引用操作的一个规则.=号右侧可能是以下类型之一:

- Variable reference 变量引用
- String literal 字符串
- Property reference 属性引用
- Method reference 命令引用
- Number literal 数字
- ArrayList 数组
- Map 映射

下面是对上述类型设置的示例:

```
#set( $monkey = $bill ) ## variable reference
#set( $monkey.Friend = "monica" ) ## string literal
#set( $monkey.Blame = $whitehouse.Leak ) ## property reference
#set( $monkey.Plan = $spindoctor.weave($web) ) ## method reference
```

```
#set( $monkey.Number = 123 ) ##number literal
#set( $monkey.Say = ["Not", $my, "fault"] ) ## ArrayList
#set( $monkey.Map = {"banana" : "good", "roast beef" : "bad"}) ## Map
```

注意：在 ArrayList 类型引用的例子中，其原素定义在数组 [..]中，因此，你可以使表 $monkey.Say.get(0)访问第一个元素.

类似的,引用 Map 的例子中，原素定义在 { } 中，其键和值间以:隔成一对，使用 $monkey.Map.get("bannana") 在上例中将返回 'good'，($monkey.Map.banana 也会有同样效果).

下面是一般的计算表达式:

```
#set( $value = $foo + 1 )
#set( $value = $bar - 1 )
#set( $value = $foo * $bar )
#set( $value = $foo / $bar )
```

但注意：如果右边的操作数是一个属性或命令的引用而返回 null,那么赋值将不会成功，且在随后的 VTL 中也不能再取出使用. 如下例:

```
#set( $result = $query.criteria("name") )
The result of the first query is $result

#set( $result = $query.criteria("address") )
The result of the second query is $result
```

如果 *$query.criteria("name")* 返回的是字符串 "bill"，但 *$query.criteria("address")* 返回 null，上面的 TVL 输出结果将是:

```
The result of the first query is bill

The result of the second query is bill
```

这对与初学者的理解有些麻烦,比如在 *#foreach* loops 中， 你使用 *#set* 给一个属性或命令赋值时，如下例示:

```
#set( $criteria = ["name", "address"] )

#foreach( $criterion in $criteria )

    #set( $result = $query.criteria($criterion) )

    #if( $result )
        Query was successful
    #end
```

```
#end
```

在上例中，就不能依赖 if($result)来决定查询是否成功. *$result* 一但被 *#set* 为 *null* (context 会同样)，它将不能被赋 *其它值* (不能从 context 中取出).

一个解决办法是，每次都将*$result* 设为 *false*. 如果 *$query.criteria()* 调用成功，就可以检测到.

```
#set( $criteria = ["name", "address"] )

#foreach( $criterion in $criteria )

    #set( $result = false )
    #set( $result = $query.criteria($criterion) )

    #if( $result )
        Query was successful
    #end

#end
```

注意： *#set* 不需要使用 *#end* 来声明结尾.

2. Literals (语义解析)

使用*#set* 指令时,变量如果用 “” 引起会被解析，如:

```
#set( $directoryRoot = "www" )
#set( $templateName = "index.vm" )
#set( $template = "$directoryRoot/$templateName" )
$template
```

输出的将是:

```
www/index.vm
```

但当用单引号引起来时，就不会被解析：:

```
#set( $foo = "bar" )
$foo
#set( $blargh = '$foo' )
$blargh
```

输出后会是:

```
  bar
  $foo
```

默认情况下，不会解析单引号中的变量，当然，这可以通过改变 Velocity 的配置参数来改变：

velocity.properties such that stringliterals.interpolate=false.

另外，指令 *#literal* 元素可以用来输出字面意思，如下示.

```
#literal()
#foreach ($woogie in $boogie)
  nothing will happen to $woogie
#end
#end
```

会输出：：

```
#foreach ($woogie in $boogie)
  nothing will happen to $woogie
#end
```

3. Conditionals(条件判断)

1. If / ElseIf / Else

***#if* 指令用来根据条件在页面中输出内容，如下简单的例子:**

```
#if( $foo )
    <strong>Velocity!</strong>
#end
```

根据变量 *$foo* 计算后是否为 true 决定输出，这会有两种情况: (i) *$foo* 的是值是一个 boolean (true/false)型并有一个 true value，或 (ii) 它是一个非 null 值. 要记者，Velocity context 中只能包含 Objects，因此当我们讲 'boolean'时，它就是一个 Boolean (the class).

在 *#if* 和 *#end* 的内容是否会输出，由$foo 是否为 true 决定. 这里，如果 *$foo* is true, 输出将是: "Velocity!". 如果*$foo* 为 null 或 false,将不会有任何输出.

#elseif 或 *#else* 可以 *#if* 和组合使用. 如果第一个表达式为 true,将会不计算以后的流程，如下例假设 t *$foo* 是 15 and *$bar* 产 6.

```
#if( $foo < 10 )
    <strong>Go North</strong>
#elseif( $foo == 10 )
    <strong>Go East</strong>
#elseif( $bar == 6 )
    <strong>Go South</strong>
#else
    <strong>Go West</strong>
#end
```

输出将会是

Go South.

2.Relational and Logical Operators(关系和逻辑运算)

Velocity 使用==来做比较,如下例.

```
#set ($foo = "deoxyribonucleic acid")
#set ($bar = "ribonucleic acid")
```

```
#if ($foo == $bar)
   In this case it's clear they aren't equivalent. So...
#else
   They are not equivalent and this will be the output.
#end
```

注意：== 计算与 java 中的 == 计算有些不同：不能用来测试对象是否相等(指向同一块内存). Velocity 中是否相等仅直接的用来比较 numbers, strings 的值, or objects 的 toString()结果是否相等. 如果是不同的对象,会调用它们的 toString()命令结果来比较.

Velocity 也使用 AND, OR and NOT 执行逻辑运算.详细说明请参看《VTL 参考中文版》，如下是一些简单示例：

```
## logical AND

#if( $foo && $bar )
    <strong> This AND that</strong>
#end
```

仅当 *$foo $bar* 和都为 true 时，*#if()*才会输出中间内容.

OR 运算例子

```
## logical OR


#if( $foo || $bar )
      <strong>This OR That</strong>
#end
```

$foo 或$bar 只要有一个为 true 就可以输出。

NOT 运算则只有一个操作参数或表达式：

```
##logical NOT


#if( !$foo )
   <strong>NOT that</strong>
#end
```

考虑下，下面的例子有几种输出结果？.

```
#if( $foo == $bar)it's true!#{else}it's not!#end</li>
```

4. Loops(循环)

Foreach Loop

#foreach 用来创建循环. For example:

```
<ul>
#foreach( $product in $allProducts )
```

```
    <li>$product</li>
#end
</ul>
```

#foreach 会生成包含$products 中对象的一个列表. 每一次循环都会将列表中的一个对象赋与变量 *$product* .

$allProducts 或以是一个 Vector, a Hashtable or an Array 类型的容器. 指定给变量 *$product* 是一个引用到其中一个 java 对象的引用. For example, if *$product* 确实是一个 java 代码中的 Product class i,它可以这样的方法访问*$product.Name* method (ie: *$Product.getName()*).

我们假设 *$allProducts* 是一个 Hashtable.看看取出其中的东东多么简单:

```
<ul>
#foreach( $key in $allProducts.keySet() )
    <li>Key: $key -> Value: $allProducts.get($key)</li>
#end
</ul>
```

通过引用变量$velocityCount 可以访问到 Velocity 提供的计数器:

```
<table>
#foreach( $customer in $customerList )
    <tr><td>$velocityCount</td><td>$customer.Name</td></tr>
#end
</table>
```

$velocityCount 默认的计数器引用，你可以在配置 velocity.properties 中改成你喜欢的:

```
# Default name of the loop counter
# variable reference.
directive.foreach.counter.name = velocityCount

# Default starting value of the loop
# counter variable reference.
directive.foreach.counter.initial.value = 1
```

当然，你还可以设置其它参数，具体见《Velocity Java 开发指南中文版》中讲解.

```
# The maximum allowed number of loops.
directive.foreach.maxloops = -1
```

3. Include(引入)

#include 脚本元素让模板设计者可以在模板中引入一个本地文件，这个被引入的文件将不会经过 Velocity 的解析. 安全起见，可以引放的文件只是是配置参数 TEMPLATE_ROOT 所定义目录下的，默认为当前目录下.

```
#include( "one.txt" )
```

如果需要引入多个文件，可以像下面这样.

```
#include( "one.gif","two.txt","three.htm" )
```

当然，还可用一个变量名来代替文件名引入.

```
#include( "greetings.txt", $seasonalstock )
```

6. Parse(解析模板)

#parse 元素指示可以引入一个包含 TVL 的本地文件，这个文件将被 Veloict engine 解析输出。.

```
#parse( "me.vm" )
```

与 *#include* 指令不同，*#parse* 可以从引入的模板中得到变量引用.但#parse 指令只能接受一个参数.

VTL templates 被*#parse* 的模板中还可以再包含#parse 声明，默认的深度为 10，这是由配置参数 *directive.parse.max.depth* 在文件 velocity.properties 中决定的，你可以修改它以适合项目要求

7. Stop

#stop 指令用来指示在模板的某处，engine 停止解析，这一般用来调用。用法很简单.

```
#stop
```

8. Velocimacros(宏调用)

#macro 指令让模板设计者可以将些重复、相关的的脚本版断定义为一个功能块.无论在什么情况下. 出于单一意图设计的 Velocimacro 都会最大程序的减少模板编写中可以的出错,还是看个例子来理解一下 Velocimacros 的概念.

```
#macro( d )
<tr><td></td></tr>
#end
```

这样就定义了一个名为 d 的宏，它可以在其它的模板中像下面那样直接引用:

```
#d()
```

Velocimacro 可以接收 0 到任意多的传入参数.如上个例是 0 个参数,但当它被调用时，也必须传入同样多的参数. 这里定义了一个有两个参数的宏.

```
#macro( tablerows $color $somelist )
#foreach( $something in $somelist )
    <tr><td bgcolor=$color>$something</td></tr>
#end
#end
```

然后，我们在页面中来使用：

```
#set( $greatlakes = ["Superior","Michigan","Huron","Erie","Ontario"] )
#set( $color = "blue" )
<table>
    #tablerows( $color $greatlakes )
</table>
```

注意变量 *$greatlakes* 取代了宏中变量 *$somelist* 的输出，最终的输出如下:

```
<table>
    <tr><td bgcolor="blue">Superior</td></tr>
    <tr><td bgcolor="blue">Michigan</td></tr>
    <tr><td bgcolor="blue">Huron</td></tr>
    <tr><td bgcolor="blue">Erie</td></tr>
    <tr><td bgcolor="blue">Ontario</td></tr>
</table>
```

宏一般被定义在模板中，那么站点上的其它模板中又如何调用呢？如能定义一个可以更大范围内共想的宏就太好了

如果将宏#tablerows($color $list) 定义到一个模板库中(Velocimacros template library)，其它模板就都可以访问它了.

Velocimacro Arguments（宏的参数）

Velocimacros 可以从 TVL 中接受以下参数 ：

- 引用类型 : 所有以'$'开头的
- String literal : something like "$foo" or 'hello'
- Number literal : 1, 2 etc
- IntegerRange : [1..2] or [$foo .. $bar]
- ObjectArray : ["a", "b", "c"]
- boolean value true
- boolean value false

当传一个引用参数给宏时, 引用是通过名字传入的('pass by name').

```
#macro( callme $a )
    $a $a $a
#end

#callme( $foo.bar() )
```

上例中命令 bar() 被调用了 3 times.

最后要说的是,这个特性有些难以学习,但当你精心组织规划你的宏库时， 消除在 VTL 中重复功能的脚本时 - 你可以像使用一个对象或组件一样使用宏, 比如一个宏对象生成多个表格的重复色彩.

如果你想利用这个特性，你只需要像下面那样简单的编码传一个值给它来调用 :

```
#set( $myval = $foo.bar() )
#callme( $myval )
```

Velocimacro Properties(宏的属性)

配置文件 velocity.properties 中有多行相关配置，具体请见《Velocity Java 开发指南中文版》.

velocimacro.library - 用来指定全局的宏库，多个可以,号分开.

velocimacro.permissions.allow.inline - 默认为 true,可以让宏定义在一个正规的模板文件中.

velocimacro.permissions.allow.inline.to.replace.global - 用来指定模板内定义的宏的功能是否要以替换全局库，默认为 false.

velocimacro.permissions.allow.inline.local.scope - 模板中定义的宏的使用范围是否只是本模板可用.

velocimacro.context.localscope - 如果为 true,宏通过#set 赋值时.宏中将保持一个，且不会由于 context 中的数据被修改而变化,同样，宏中的修改也不会改变 context 中的。

velocimacro.library.autoreload - 是否自动重新载入，用于调试环境,默认 false,如为 true,需要取掉 chcheing:. file.resource.loader.cache = false).

一些细节：

宏必须在模板中使用#macro()指令前定义.

尽量不要直接在模板中使用#parse() 包含 #macro() 指令.因为 #parse() 动作在运行时执行,时会有一个在 VM 中查找元素的过程.

13.注掉 VTL Directives

嗯，你学下英语吧，一看就懂☺

VTL directives can be escaped with the backslash character ("\") in a manner similar to valid VTL references.

```
## #include( "a.txt" ) renders as <contents of a.txt>
#include( "a.txt" )

## \#include( "a.txt" ) renders as #include( "a.txt" )
\#include( "a.txt" )

## \\#include ( "a.txt" ) renders as \<contents of a.txt>
\\#include ( "a.txt" )
```

Extra care should be taken when escaping VTL directives that contain multiple script elements in a single directive (such as in an if-else-end statements). Here is a typical VTL if-statement:

```
#if( $jazz )
    Vyacheslav Ganelin
#end
```

If *$jazz* is true, the output is

```
Vyacheslav Ganelin
```

If *$jazz* is false, there is no output. Escaping script elements alters the output. Consider the following case:

```
\#if( $jazz )
    Vyacheslav Ganelin
\#end
```

Whether *$jazz* is true or false, the output will be

```
 #if($ jazz )
     Vyacheslav Ganelin
 #end
```

In fact, because all script elements are escaped, *$jazz* is never evaluated for its boolean value. Suppose backslashes precede script elements that are legitimately escaped:

```
\\#if( $jazz )
```

```
   Vyacheslav Ganelin
\\#end
```

In this case, if *$jazz* is true, the output is

```
\ Vyacheslav Ganelin
\
```

To understand this, note that the #if(arg) when ended by a newline (return) will omit the newline from the output. Therefore, the body of the #if() block follows the first '\', rendered from the '\\' preceding the #if(). The last \ is on a different line than the text because there is a newline after 'Ganelin', so the final \\, preceding the #end is part of the body of the block.

If *$jazz* is false, the output is

```
\
```

Note that things start to break if script elements are not properly escaped.

```
\\\#if( $jazz )
    Vyacheslave Ganelin
\\#end
```

Here the *#if* is escaped, but there is an *#end* remaining; having too many endings will cause a parsing error.

14.VTL: 一般使用的格式

尽量是这样：

```
#set( $imperial = ["Munetaka","Koreyasu","Hisakira","Morikune"] )
#foreach( $shogun in $imperial )
    $shogun
#end
```

而不要这样写，虽然它可以运行:

```
Send #set($foo=["$10 and ","a pie"])#foreach($a in $foo)$a#end please.
```

Velocity 会自动跳过空格，上面的可以这样写：:

```
Send me
#set( $foo = ["$10 and ","a pie"] )
#foreach( $a in $foo )
$a
```

```
#end
please.
```

or as

```
Send me
#set($foo      = ["$10 and ","a pie"])
               #foreach          ($a in $foo )$a
        #end please.
```

In each case the output will be the same.

15.Other Features and Miscellany(其它特性和细节)

1. 数学计算

如下是四则运算的例子:

```
#set( $foo = $bar + 3 )
#set( $foo = $bar - 4 )
#set( $foo = $bar * 6 )
#set( $foo = $bar / 2 )
```

2. Range Operator

The range operator can be used in conjunction with *#set* and *#foreach* statements. Useful for its ability to produce an object array containing integers, the range operator has the following construction:

```
[n..m]
```

Both *n* and *m* must either be or produce integers. Whether *m* is greater than or less than *n* will not matter; in this case the range will simply count down. Examples showing the use of the range operator as provided below:

```
First example:
#foreach( $foo in [1..5] )
$foo
#end

Second example:
#foreach( $bar in [2..-2] )
$bar
#end
```

```
Third example:
#set( $arr = [0..1] )
#foreach( $i in $arr )
$i
#end

Fourth example:
[1..3]
```

Produces the following output:

```
First example:
1 2 3 4 5

Second example:
2 1 0 -1 -2

Third example:
0 1

Fourth example:
[1..3]
```

Note that the range operator only produces the array when used in conjunction with *#set* and *#foreach* directives, as demonstrated in the fourth example.

Web page designers concerned with making tables a standard size, but where some will not have enough data to fill the table, will find the range operator particularly useful.

3.Advanced Issues: Escaping and !

When a reference is silenced with the *!* character and the *!* character preceded by an \ escape character, the reference is handled in a special way. Note the differences between regular escaping, and the special case where \ precedes *!* follows it:

```
#set( $foo = "bar" )
$\!foo
$\!{foo}
$\\!foo
$\\\!foo
```

This renders as:

```
$!foo
$!{foo}
```

```
$\!foo
$\\!foo
```

Contrast this with regular escaping, where \ precedes *$*:

```
\$foo
\$!foo
\$!{foo}
\\$!{foo}
```

This renders as:

```
$foo
$!foo
$!{foo}
\bar
```

4.Velocimacro Miscellany(关于宏的一些问题)

这是一些简短的问题总结,也许你先要有这样一个概念:. 'Velocimacro' 就像一个 'VM' 。

可否用一个指示符做为另外一个指示符运算的参数?

如 : #center(#bold("hello"))

No. 指示符不是有效的参数但你可以这样实现你想要的:

```
#set($stuff = "#bold('hello')" )
#center( $stuff )
```

或者:

```
#center( "#bold( 'hello' )" )
```

上面这个例子中, 参数是在调用的宏中生成的.不是调用者传入的. 看看下面的例子 :

```
#macro( inner $foo )
  inner : $foo
#end

#macro( outer $foo )
   #set($bar = "outerlala")
   outer : $foo
#end
```

```
#set($bar = 'calltimelala')
#outer( "#inner($bar)" )
```

输出将是:

```
Outer : inner : outerlala
```

因为 "#inner($bar)" 是发生在 #outer()内部的!

可否通过 #parse()来注册一个宏 ?

宏必须在模板使用前定义好.前面已有一个关于此问题的建议，#parse()是运行时执进的，JVM 查找对象的顺序不一定会全按我们预计的执行。

什么是宏的自动重新装载?

这是由配置参数决定的, 为方例开发者，在生产环境中则不需要 :

velocimacro.library.autoreload

默认的是 false.当设为 true 中，需要设定 chcheing 参数;

<type>.resource.loader.cache = false

(具体配置请见开发指南，如下是一个配置的例子)

```
    file.resource.loader.path = templates
    file.resource.loader.cache = false
    velocimacro.library.autoreload = true
```

注意：Don't keep this on in production.

5.String Concatenation(连结字符串)

很简单，看例子就是 :

```
      #set( $size = "Big" )
      #set( $name = "Ben" )

     The clock is $size$name.
```

上面的输出将是

'The clock is BigBen'.

或者：

```
#set( $size = "Big" )
#set( $name = "Ben" )

#set($clock = "$size$name" )

The clock is $clock.
```

它们都是同样的输出，最后一个例子如下，：

```
#set( $size = "Big" )
#set( $name = "Ben" )

#set($clock = "${size}Tall$name" )

The clock is $clock.
```

输出将是

'The clock is BigTallBen'.

16.Feedback

如果您有什么问题或建议，请联系 Velocity developers list. Thanks!