

# Velocity 用户手册

## 关于这个用户手册

Velocity 用户手册是帮助页面设计者和内容提供者认识 Velocity 和其简单而功能强大的脚本语言——Velocity 模板语言（VTL）。在手册上的许多例子，都是用 Velocity 插入动态的内容到网页上，但是所有的 VTL 例子都能应用到其他的页面和模板中。

感谢使用 Velocity!

## Velocity 是什么？

Velocity 是一个基于 java 的模板引擎（template engine）。它允许任何人仅仅简单的使用模板语言（template language）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，而由 java 程序开发人员关注业务逻辑编码。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 的能力远不止 web 站点开发这个领域，例如，它可以从模板（template）产生 SQL 和 PostScript、XML，它也可以被当作一个独立工具来产生源代码和报告，或者作为其他系统的集成组件使用。Velocity 也可以为 Turbine web 开发架构提供模板服务（template service）。Velocity+Turbine 提供一个模板服务的方式允许一个 web 应用以一个真正的 MVC 模型进行开发。

## Velocity 能为我们作什么？

Mud 商店例子

假设你是一家专门出售 Mud 的在线商店的页面设计人员，让我们暂且称它为“在线 MUD 商店”。你们的业务很旺，客户下了各种类型和数量的 mud 订单。他们都是通过输入用户名和密码后才登陆到你的网站，登陆后就允许他们查看订单并购买更多的 mud。现在，一种非常流行的 mud 正在打折销售。另外有一些客户规律性的购买另外一种也在打折但是不是很流行的 Bright Red Mud，由于购买的人并不多所以它被安置在页面的边缘。所有用户的信息都是被跟踪并存放于数据库中的，所以某天有一个问题可能会冒出来：为什么不使用 velocity 来使用户更好的浏览他们感兴趣的商品呢？

Velocity 使得 web 页面的客户化工作非常容易。作为一个 web site 的设计人员，你希望每个用户登陆时都拥有自己的页面。

你会见了一些公司内的软件工程师，你发现他们每个人都同意客户应该拥有具有个性化的信息。那让我们把软件工程师应该做的事情发在一边，看一看你应该做些什么吧。

你可能在页面内嵌套如下的 VTL 声明：

```
<HTML>
<BODY>
Hello $customer.Name!
```

```

<table>
#foreach( $mud in $mudsOnSpecial )
  #if ( $customer.hasPurchased($mud) )
    <tr>
      <td>
        $flogger.getPromo( $mud )
      </td>
    </tr>
  #end
#end
</table>

```

foreach 的详细用法不久就会进行深入描述。重要的是，这短小的脚本能在你的网站上出现。当一个对 Bright Red Mud 很感兴趣的顾客登陆的时候，同时 Bright Red Mud 在热卖中，这时顾客就能显著地看到。假如一个玩 Terracotta Mud 很久的顾客登陆，Terracotta Mud 的售卖信息就会出现在前面中间。Velocity 的适用性是很巨大的，限制的只是你的创造性。

VTL Reference 含有许多其他 Velocity 元素，这些元素能够共同帮助你，使你的网站更加好。当你越来越熟悉那些原理，你开始释放 Velocity 的能力。

## Velocity 模板语言(VTL):说明

VTL 意味着提供最简单、最容易并且最整洁的方式合并页面动态内容。VTL 使用 references 来在 web site 内嵌套动态内容，一个变量就是一种类型的 reference。变量是某种类型的 reference，它可以指向 java 代码中的定义，或者从当前页面内定义的 VTL statement 得到值。下面是一个 VTL statement 的例子，它可以被嵌套到 HTML 代码中：

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

和所有的 VTL statement 一样，这个 statement 以 # 字符开始并且包含一个 directive: set。当一个在线用户请求你的页面时，Velocity 模板引擎将查询整个页面以便发现所有 # 字符，然后确定哪些是 VTL statement，哪些不需要 VTL 做任何事情。

# 字符后紧跟一个 directive: set 时，这个 set directive 使用一个表达式（使用括号封闭）——一个方程式分配一个值给变量。变量被列在左边，而它的值被列在右边，最后他们之间使用 = 号分割。

在上面的例子中，变量是 \$a，而它的值是 Velocity。和其他的 references 一样以 \$ 字符开始，而值总是以双引号封闭。Velocity 中仅有 String 可以被赋值给变量。

使用 \$ 字符开始的 references 用于得到什么；使用 # 字符开始的 directives 用于做点什么。在上面的例子中，#set 是分配一个值给变量。变量 \$a 在模板中输出 "Velocity"。

## Hello Velocity World!

一旦某个变量被分配了一个值，那么你就可以在 HTML 文件的任何地方引用它。在下面的例子中，一个值被分配给 \$foo 变量，并在其后被引用。

```
<html>
```

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

上面的实现结果是在页面上打印“Hello Velocity World!”

为了使包含 VTL directives 的 statement 更具有可读性，我们鼓励你在新行开始每个 VTL statement，尽管你不是必须这么做。Set 的用法将在后面详细描述。

## 注释

注释是那些描述文本不出现在模板引擎输出里面。注释一个主要用处是提醒自己和解释出现在 VTL 中的声明，或是其他用途。下面是一个在 VTL 中的注释例子。

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

单行注释以##开始，结束在这行的结尾。如果你要写几行注释，这没有必要写几个单行注释。

多行注释，以\*\*开始\*\*结束，可以解决这个问题。

```
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.
**
```

```
Here is text outside the multi-line comment; it is visible.
```

这里有几个例子关于单行和多行注释如何工作。

```
This text is visible. ## This text is not.
This text is visible.
This text is visible. ** This text, as part of a multi-line comment,
is not visible. This text is not visible; it is also part of the
multi-line comment. This text still not visible. *# This text is outside
the comment, so it is visible.
## This text is not visible.
```

这里有第三种类型的注释，VTL 注释块，可能用于存放文档的作者名称和版本信息。

```
***
This is a VTL comment block and
may be used to store such information
as the document author and versioning
information:
@author
```

```
@version 5
*#
```

## References

在 VTL 中有三种类型的 references: 变量(variables)、属性(properties)、方法(methods)。作为一个使用 VTL 的页面设计者, 你和你的工程师必须就 references 的名称达成共识, 以便你可以在你的 template 中使用它们。

所有的 reference 被作为一个 String 对象处理。如果有一个对象\$foo 是一个 Integer 对象, 那么 Velocity 将调用它的 toString() 方法将这个对象转型为 String 类型。

## 变量

非正式变量是由 “\$” 开头, 接着是 VTL 标识符。VLT 标识符必须以字母 (a..z, A..Z) 开头。剩下的部分限于以下几种:

- 字母 (a..z, A..Z)
- 数字 (0..9)
- 连字符 (“-”)
- 下划线 (“\_”)

这里是几个在 VTL 中有效的变量 reference。

```
$foo
$mudSlinger
$mud-slinger
$mud_slinger
$mudSlinger1
```

当 VLT 定义一个变量, 例如\$foo, 变量能或者通过模板中的 set 方法, 或者通过 Java 代码获得值。

例如, Java 变量 \$foo 的值是 bar, 在这个模板被请求时, 在网页上 bar 会替代所有 \$foo。假如包括了下面的声明:

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

按照这样的设置, 输出就会跟之前的一样。

## 属性

第二种有趣的 VTL reference 就是属性, 而且属性有一种与众不同的格式。非正式变量是由 “\$” 开头, 接着是 VTL 标识符, 再接着就是字符 (“.”) 和其他的 VLT 标识符。这里是一些在 VLT 中有效属性的定义:

```
$customer.Address
$purchase.Total
```

在第一个例子中\$customer.Address 有两种含义。它可以表示: 查找 hashtable 对象 customer 中以 Address 为关键字的值; 也可以表示调用 customer 对象的 getAddress() 方法。当你的页面被请求时, Velocity 将确定以上两种方式选用那种, 然后返回适当的值。

## 方法

一个方法就是被定义在 java 中的一段代码, 并且它有完成某些有用工作的

能力，例如一个执行计算和判断条件是否成立、满足等。方法是一个由\$开始并跟随 VTL 标识符组成的 References，一般还包括一个 VTL 方法体。一个 VTL 方法体包括一个 VTL 标识接着一个左括号(“(”)，接着是参数列表，再接着是右括号(“)”)。这里是一些在 VTL 中有效的方法定义：

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

前两个例子\$customer.getAddress()和\$purchase.getTotal()看起来挺像上面的属性\$customer.Address 和 \$purchase.Total。如果你觉得他们之间有某种联系的话，那你是正确的。

VTL 属性可以作为 VTL 方法的缩写。\$customer.Address 属性和使用\$customer.getAddress()方法具有相同的效果。如果可能的话使用属性的方式是比较合理的。属性和方法的不同点在于你能够给一个方法指定一个参数列表。

非正式定义能够用下面的方法：

```
$sun.getPlanets()  
$annelid.getDirt()  
$album.getPhoto()
```

我们期待那些方法返回属于太阳系的行星的名称，喂养我们的蚯蚓，或者从相册里面取出一张照片。只有长符号为下面的方法服务。

```
$sun.getPlanet( ["Earth", "Mars", "Neptune"] )  
## Can't pass a parameter list with $sun.Planets  
  
$sisyphus.pushRock()  
## Velocity assumes I mean $sisyphus.getRock()  
  
$book.setTitle( "Homage to Catalonia" )  
## Can't pass a parameter list
```

## 正式 reference 标记

非正式 references 用于上述的例子中。但是同样有正式的 references，如下面所示：

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

在几乎所有场合你都可以使用非正式 references，但是在某些场合，只能使用正式 reference 才能正确处理。

设想你创建一个句子：\$vice 作为句子的名词。目标是为了使某些人选择不同的词，产生下面两种结果之一：“Jack is a pyromaniac.” 或者 “Jack is a kleptomaniac.”。使用非正式定义不太适合用于这种情况。看一下下面的例子：

```
Jack is a $vicemaniac.
```

本来变量是\$vice 现在却变成了\$vicemaniac，这样 Velocity 就不知道您到底要什么了。所以，应该使用正式格式书写

```
Jack is a ${vice}maniac.
```

现在 Velocity 就知道 reference 是\$vice，而不是\$vicemaniac。正式定义经常用于模板中 references 与文本连接在一起的情况。

## Quiet reference notation

当 Velocity 遇到没有定义的 reference，通常它会直接输出 reference。例如：假如下面的 reference 出现在一个 VTL 模板中：

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

当 form 最初加载的时候，变量\$email 没有值，但你想出现一个空白的文本框设定值为“\$email”。

使用 quiet reference notation 可以使 Velocity 正常显示，你需要用\$!email，代替\$email。所以上面的例子，会改成下面：

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

这样文本框的初始值就不会是 email 而是空值了。

正式和 quiet 格式的 reference notation 也可一同使用，像下面这样：

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

## Getting literal

Velocity 使用特殊字符\$和#来帮助它工作，所以如果要在 template 里使用这些特殊字符要格外小心。本节将讨论\$字符。

货币字符

这是没有问题的："I bought a 4 lb. sack of potatoes at the farmer's market for only \$2.50!"，VTL 中使用\$2.5 这样的货币标识是没有问题得的，VTL 不会将它错认为是一个 reference，因为 VTL 中的 reference 总是以一个大写或者小写的字母开始。

## Escaping valid VTL reference

某些情况使用 Velocity 可能会觉得很烦恼。逃避特殊符的有效方法，就是使用反斜杠(“\”)。

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

假如 Velocity 在你的模板中遇到 \$email，它会搜索上下文，得到相应的值。这里的输出是 foo，因为 \$email 被定义了。假如 \$email 没有被定义，输出会是 \$email。

设想 \$email 被定义了(例如，它的值是 foo)，而且你想输出 \$email。这里有几种方法能达到目的，但是最简单的是使用逃避符。

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

将显示为：

```
foo
$email
```

```
\foo
\${email}
```

注意 Velocity 处理定义了的 references 与没有定义的不一样。这里 set \$foo 的值为 gibbous。

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

输出会是: \$moon = gibbous , \$moon 按照字面上输出因为它没有定义, gibbous 替代 \$foo 输出。

避开 VTL 的 directives 还有其它方法, 在 Directives 那章节会更详细描述。

## Case substitution

现在你已经对 reference 比较熟悉了, 你可以将他们高效的应用于你的 template 了。Velocity 利用了很多 java 规范以方便设计人员的使用。例如:

```
$foo

$foo.getBar()
## is the same as
$foo.Bar

$data.getUser("jon")
## is the same as
$data.User("jon")

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

那些例子说明了同样的 references 用法。Velocity 利用 Java 的 introspection 和 bean features 解决 reference 的名称对于对象和对象方法的问题。它可以出入你的模板中和求出 references 的值。

Velocity 是模仿 Sun 微系统中的 Bean 规范定义的, 因而它是很灵活的。然而, 它的开发者已经很努力地捕捉和纠正可能出现的错误。当方法 getFoo() 在模板中被 \$bar.foo 调用时, Velocity 首先尝试 \$getfoo。如果失败, 它会继续尝试 \$getFoo。同样地, 当模板查询 \$bar.Foo, Velocity 会先尝试 \$getFoo(), 然后再尝试 \$getfoo()。

但是, 注意 VTL 中不会将 reference 解释为对象的实例变量。例如: \$foo.Name 将被解释为 Foo 对象的 getName() 方法, 而不是 Foo 对象的 Name 实例变量。

## Directives

Reference 允许设计者使用动态的内容, 而 directive 使得你可以应用 java 代码来控制你的显示逻辑, 从而达到你所期望的显示效果。

```
#set
```

#set 标志是用于对一个 reference 赋值。值会赋给一个变量或者一个属性，而且赋值会在括号里出现：

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

左边 (LHS) 一定是一个变量或者一个属性。右边 (RHS) 可以是下面中的一个类型：

- 变量
- 字符串
- 属性
- 方法
- 数字
- 数组

这些例子显示上述的每一种类型：

```
#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
```

注意：最后一个例子的取值方法为：\$monkey.Say.get(0)。

RHS 也可以是一个简单的算术表达式：

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

如果你的 RHS 是一个 null，VTL 的处理将比较特殊：它将指向一个已经存在的 reference，这对初学者来讲可能是比较费解的。例如：

```
#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”) 返回一个 “bill”，而 \$query.criteria(“address”) 返回的是 null，则显示的结果如下：

```
The result of the first query is bill  
  
The result of the second query is bill
```

这容易使新手糊涂：创建一个 #foreach 循环，企图想通过一个属性或者一个方法 #set 一个 reference，然后马上就用 #if 测试。例如：

```
#set( $criteria = ["name", "address"] )  
  
#foreach( $criterion in $criteria )  
  
    #set( $result = $query.criteria($criterion) )
```



```

    #if( $result )
        Query was successful
    #end

#end

```

在上面的例子中，程序将不能智能的根据`$result` 的值决定查询是否成功。在`$result` 被`#set` 后（added to the context），它不能被设置回 `null`（removed from the context）。打印的结果将显示两次查询结果都成功了，但是实际上有一个查询是失败的。

为了解决以上问题我们可以通过预先定义的方式：

```

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

#foreach( $criterion in $criteria )

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

    #if( $result )
        Query was successful
    #end

#end

```

不像其他 Velocity 指示符号，`#set` 没有一个`#end` 结束。

## String Literals

当你使用`#set` directive, String literal 封闭在一对双引号内。象下面：

```

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

```

上面这段代码的输出结果为：

```
www/index.vm
```

但是，当 string literal 被封装在单引号内时，它将不被解析：

```

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

```

输出为：

```
bar $foo
```

上面这个特性可以通过修改 `velocity.properties` 文件的 `stringliterals.interpolate = false` 的值来改变上面的特性是否有效。

## 条件语句

## If / ElseIf / Else

当一个 web 页面被生成时使用 Velocity 的 #if directive, 如果条件成立的话可以在页面内嵌入文字。例如:

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

上例中的条件语句将在以下两种条件下成立: (i)\$foo 是一个 boolean 型的变量, 且它的值为 true; (ii)\$foo 变量的值不为 null。这里需要注意一点: Velocity context 仅仅能够包含对象, 所以当我们说“boolean”时实际上代表的是一个 Boolean 对象。即便某个方法返回的是一个 boolean 值, Velocity 也会利用内省机制将它转换为一个 Boolean 的相同值。

如果条件成立, 那么 #if 和 #end 之间的内容将被显示。在这个例子中, 如果 \$foo 的值为 true, 输出为 “Velocity!”。相反地, 如果 \$foo 是一个 null 值, 或者是一个 false 值, 表达式值为 false, 没有输出。

#elseif 和 #else 元素可以同 #if 一同使用。注意: Velocity 模板引擎遇到一个为 true 值的表达式就会停止。在下面的例子, 假设 \$foo=15, \$bar=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
```

在这个例子中, \$foo 比 10 大, 所以在开始的两个比较中都失败。接着 \$bar 跟 6 比较是真的, 所以输出为 Go South。

注意这里的 Velocity 的数字是作为 Integer 来比较的——其他类型的对象将使得条件为 false, 但是与 java 不同, 它使用 “==” 来比较两个值, 而且 velocity 要求等号两边的值类型相同。

## 关系、逻辑运算符

Velocity 中使用等号操作符判断两个变量的关系。这里有个简单例子, 关于等于号的使用:

```
#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
```

Velocity 有 AND、OR 和 NOT 逻辑运算符。想得到更多信息, 请看 VTL Reference Guide。下面的例子是说明 AND、OR 和 NOT 逻辑运算符的用法:

```
## logical AND

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

只有当\$foo 和 \$bar 都为 true, #if 才会得到 true 值。如果\$foo=false , 表达式的值为 false, \$bar 就不会求值。如果 \$foo 的值为 true , Velocity 模板引擎会检查 \$bar 的值, 如果\$bar=true, 整个表达式的值为 true, 输出为 “This AND that”。如果\$bar=false, 整个表达式的值为 false, 没有输出。

逻辑 OR 的工作方式一样, 除了只要有一个值为 true , 整个表达式的值就为 true。考虑一下下面的例子。

```
## logical OR

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

如果\$foo=true, Velocity 模板引擎就没有必要查找 \$bar, 无论 \$bar 是 true 还是 false , 表达式的值为 true , 输出为 “This OR That”。如果\$foo=false, \$bar 的值就一定要检查, 在这个例子中, 如果 \$bar 同样是 false, 表达式的值为 false , 没有输出。从另外一个角度看, 如果\$bar 的值为 true , 整个表达式的值为 true , 输出为 “This OR That”。

关于逻辑 NOT , 只有一个疑问:

```
##logical NOT

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

如果 \$foo=true, !\$foo 的值为 false, 没有输出。如果\$foo=false, !\$foo 的值为 true, 输出为 “NOT that”。注意不要跟 quiet reference !\$foo 混为一谈, 那是完全不一样的。

## 循环

### Foreach 循环

#foreach 用于循环。例子:

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

每次循环\$allProducts 中的一个值都会赋给\$product 变量。

\$allProducts 可以是一个 Vector、Hashtable 或者 Array。分配给\$product 的值是一个 java 对象, 并且可以通过变量被引用。例如: 如果\$product 是一个 java 的 Product 类, 并且这个产品的名字可以通过调用他的 getName() 方法得到。

现在我们假设`$allProducts` 是一个 Hashtable，如果你希望得到它的 key 应该像下面这样：

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

Velocity 还特别提供了得到循环次数的方法，以便你可以像下面这样做：

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

`$velocityCount` 变量的名字是 Velocity 默认的名字，你也可以通过修改 `velocity.properties` 文件来改变它。默认情况下，计数从“1”开始，但是你可以在 `velocity.properties` 设置它是从“1”还是从“0”开始。下面就是文件中的配置：

```
# 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
```

## include

`#include` script element 允许模板设计者引入本地文件。被引入文件的内容将不会通过模板引擎被 render。为了安全的原因，被引入的本地文件只能在 `TEMPLATE_ROOT` 目录下。

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

`#include` 引用的文件用引号括起来。

如果您需要引入多个文件，可以用逗号分隔就行：

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

在括号内可以是文件名，但是更多的时候是使用变量的。这用于根据页面提交的需求而输出。这里有一个例子同时有文件名和变量。

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

## parse

`#parse` script element 允许模板设计者一个包含 VTL 的本地文件。Velocity 将解析其中的 VTL 并 render 模板。

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

就像 `#include`，`#parse` 接受一个变量而不是一个模板。任何由 `#parse` 指向的模板都必须包含在 `TEMPLATE_ROOT` 目录下。与 `#include` 不同的是，`#parse` 只能指定单个对象。

你可以通过修改 velocity.properties 文件的 parse\_directive.maxdepth 的值来控制一个 template 可以包含的最多 #parse 的个数——默认值是 10。#parse 是可以递归调用的，例如：如果 dofoo.vm 包含如下行：

```
Count down.
#set( $count = 8 )
#parse( "parsefoo.vm" )
All done with dofoo.vm!
```

那么在 parsefoo.vm 模板中，你可以包含如下 VTL：

```
$count
#set( $count = $count - 1 )
#if( $count > 0 )
    #parse( "parsefoo.vm" )
#else
    All done with parsefoo.vm!
#end
```

在显示“Count down”后，Velocity 通过 parsefoo.vm，从 8 往下数。当计数到了 0，它就会显示“All done with parsefoo.vm!”。在这时，Velocity 会返回到 dofoo.vm，输出信息：“All done with dofoo.vm!”。

## 停止

#stop script element 允许模板设计者停止执行模板引擎并返回。把它应用于 debug 是很有帮助的。

```
#stop
```

## Velocimacros

#macro script element 允许模板设计者定义一段可重用的 VTL template。Velocimacros 广泛用于简单和复杂的行列。Velocimacros 的出现是为了减少编码和极小化排版错误，对 Velocimacros 的概念提供一个介绍。

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

在上面的例子中 Velocimacro 被定义为 d，然后你就可以在任何 VTL directive 中以如下方式调用它：

```
#d()
```

当你的 template 被调用时，Velocity 将用<tr><td></td></tr>替换为#d()。

每个 Velocimacro 可以拥有任意数量的参数——甚至 0 个参数，虽然定义时可以随意设置参数数量，但是调用这个 Velocimacro 时必须指定正确的参数。下面是一个拥有两个参数的 Velocimacro，一个参数是 color 另一个参数是 array：

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

Velocimacro 在这个例子的定义，tablerows，有两个元素。第一个是替换

`$color` ,第二个是替换`$someslist`。

任何东西都可以通过 `Velocimacro` 加入到 VTL 模板中。`tablerows Velocimacro` 是一个 `foreach` 标识。 这里有两个`#end` 标识在`#tablerows Velocimacro` 定义里面。第一个是属于 `#foreach` ,第二个是属于 `Velocimacro` 定义的。

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

注意到`$greatlakes` 替代 `$someslist`。当`#tablerows Velocimacro` 被调用时,就会产生下面的输出:

```
<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>
```

`Velocimacros` 可以在 `Velocity` 模板内实现行内定义(`inline`),也就意味着同一个 web site 内的其他 `Velocity` 模板不可以获得 `Velocimacros` 的定义。定义一个可以被所有模板共享的 `Velocimacro` 显然是有很多好处的:它减少了在一大堆模板中重复定义的数量、节省了工作时间、减少了出错的几率、保证了单点修改。

上面定义的`#tablerows( $color $list )Velocimacro` 被定义在一个 `Velocimacros` 模板库(在 `velocity.properties` 中定义)里,所以这个 `macro` 可以在任何规范的模板中被调用。它可以被多次应用并且可以应用于不同的目的。例如下面的调用:

```
#set( $parts = ["volva","stipe","annulus","gills","pileus"] )
#set( $cellbgcol = "#CC00FF" )
<table>
#tablerows( $cellbgcol $parts )
</table>
```

当为 `mushroom.vm` 实现一个请求时,`Velocity` 会在模板库中找到`#tablerows Velocimacro`(定义在 `velocity.properties` 文件),产生以下输出:

```
<table>
  <tr><td bgcolor="#CC00FF">volva</td></tr>
  <tr><td bgcolor="#CC00FF">stipe</td></tr>
  <tr><td bgcolor="#CC00FF">annulus</td></tr>
  <tr><td bgcolor="#CC00FF">gills</td></tr>
  <tr><td bgcolor="#CC00FF">pileus</td></tr>
</table>
```

### Velocimacro arguments

`Velocimacro` 可以使用以下任何元素作为参数:

- Reference: 任何以\$开头的 reference
- String literal:
- Number literal:
- IntegerRange: [1...3]或者[\$foo...\$bar]
- 对象数组: [“a”, “b”, “c”]
- boolean 值: true、false

当将一个 reference 作为参数传递给 Velocimacro 时, 请注意 reference 作为参数时是以名字的形式传递的。这就意味着参数的值在每次 Velocimacro 内执行时才会被产生。这个特性使得你可以将一个方法调用作为参数传递给 Velocimacro, 而每次 Velocimacro 执行时都是通过这个方法调用产生不同的值来执行的。例如:

```
#macro( callme $a )
    $a $a $a
#end
#callme( $foo.bar() )
```

执行的结果是: reference \$foo 的 bar() 方法被执行了三次。

在第一次扫描中, 出现惊人的作用。但是当你考虑到 Velocimacro 的最原始动机: 消除剪切、粘贴、复制, 简单地使用 VTL 使它更灵活。它允许你传递对象到 Velocimacro, 例如一个产生重复序列颜色, 着色表格的行的对象。

如果你不需要这样的特性可以通过以下方法:

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

## Velocimacro properties

Velocity.properties 文件中的某几行能够使 Velocimacros 的实现更加灵活。注意更多的内容可以看 Developer Guide。

Velocity.properties 文件中的 velocimacro.library: 一个以逗号分隔的模板库列表。默认情况下, velocity 查找唯一的一个库: VM\_global\_library.vm。你可以通过配置这个属性来指定自己的模板库。

Velocity.properties 文件中的 velocimacro.permissions.allow.inline 属性: 有两个可选的值 true 或者 false, 通过它可以确定 Velocimacros 是否可以被定义在 regular template 内。默认值是 true——允许设计者在他们自己的模板中定义 Velocimacros。

Velocity.properties 文件中的 velocimacro.permissions.allow.inline.replace.global 属性有两个可选值 true 和 false, 这个属性允许使用者确定 inline 的 Velocimacro 定义是否可以替代全局 Velocimacro 定义(比如在 velocimacro.library 属性中指定的文件内定义的 Velocimacro)。默认情况下, 此值为 false。这样就阻止本地 Velocimacro 定义覆盖全局定义。

Velocity.properties 文件中的 velocimacro.permissions.allow.inline.local.scale 属性也是有 true 和 false 两个可选值, 默认是 false。它的作用是用于确定你 inline 定义的 Velocimacros 是否仅仅在被定义的 template 内可见。换句话说, 如果这个属性设置为 true, 一个 inline 定义的 Velocimacros 只能在定义它的 template 内使用。你可以使用此设置实现一个奇妙的 VM 窍门: a template can define a private

implementation of the second VM that will be called by the first VM when invoked by that template. All other templates are unaffected.

Velocity.properties 文件中的 velocimacro.context.localscope 属性有 true 和 false 两个可选值，默认值为 false。当设置为 true 时，任何在 Velocimacro 内通过 #set() 对 context 的修改被认为是针对此 velocimacro 的本地设置，而不会永久地影响内容。

Velocity.properties 文件中的 velocimacro.library.autoreload 属性控制 Velocimacro 库的自动加载。默认是 false。当设置为 true 时，对于一个 Velocimacro 的调用将自动检查原始库是否发生了变化，如果变化将重新加载它。这个属性使得你可以不用重新启动 servlet 容器而达到重新加载的效果，就像你使用 regular 模板一样。这个属性可以使用的前提就是 resource loader 缓存是 off 状态 (file.resource.loader.cache = false)。注意这个属性实际上是针对开发而非产品的。

### Velocimacro Trivia

Velocimacro 必须被定义在他们被使用之前。也就是说，你的 #macro() 声明应该出现在使用 Velocimacros 之前。

特别要注意的是，如果你试图 #parse() 一个包含 #macro() 的模板。因为 #parse() 发生在运行期，但是解析器在 parsetiem 决定一个看似 VM 元素的元素是否是一个 VM 元素，这样 #parse() 一组 VM 声明将不按照预期的样子工作。为了得到预期的结果，只需要你简单的使用 velocimacro.library 使得 Velocity 在启动时加载你的 VMs。

## Escaping VTL directives

VTL directives can be escaped with “\” 号，使用方式跟 VTL 的 reference 使用逃逸符的格式差不多。

```
## #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" )
```

在对一个 directive 内包含多个 script 元素的 VTL directives 使用逃逸符时要特别小心（比如在一个 if-else-end statement 内）。下面是 VTL 的 if-statement 的典型应用：

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

如果 \$jazz 是 true，输出将是：

```
Vyacheslav Ganelin
```

如果 \$jazz 是 false，将没有输出。使用逃逸符将改变输出。考虑一下下面的情况：

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



```
Vyacheslav Ganelin
\#end
```

现在无论\$ jazz 是 true 还是 false, 输出结果都是:

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

事实上, 由于你使用了逃逸符, \$jazz 根本就没有被解析为 boolean 型值。在逃逸符前使用逃逸符是合法的, 例如:

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

以上程序的显示结果为:

```
\ Vyacheslav Ganelin
\
```

为了方便理解, 注意到 #if(arg) 当新一行的结束(return)在输出中会省略新行。因此, #if() 块是接着第一个 ‘\’, 显示来自在 #if() 前面的 ‘\\’。最后的 ‘\’ 在跟文本不同的行, 因为那里有新的一行在 ‘Ganelin’ 后面, 所以最后 ‘\\’ 在 #end 的前面。

但是如果 \$jazz 为 false, 那么将没有输出。注意: 事情也有例外, 如果 script elements 没有完全的逃避。

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

在这里 #if 已经逃避了, 但是 #end 没有, 有太多的 #end 的产生解析错误。

## VTL: Formatting issues

尽管在此用户手册中 VTL 通常都开始一个新行, 如下所示:

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

但是像下面这种写法也是可以的:

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

上面的代码可以被改写为:

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

或者

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

```
#foreach      ($a in $foo )$a
#end please.
```

每一种的输出结果将一样。

## 其他特性和杂项

### math

在模板中可以使用 Velocity 内建的算术函数，如：加、减、乘、除。下面的等式分别是：加、减、乘、除：

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

当执行除法时将返回一个 Integer 类型的结果。而余数你可以使用%来得到：

```
#set( $foo = $bar % 5 )
```

在 Velocity 内使用数学计算公式时，只能使用像 -n, -2, -1, 0, 1, 2, n 这样的整数，而不能使用其它类型数据。当一个非整型的对象被使用时它将被 logged 并且将以 null 作为输出结果。

### Range Operator

Range operator 可以被用于与 #set 和 #foreach statement 联合使用。对于处理一个整型数组它是很有用的，Range operator 具有以下构造形式：

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

m 和 n 都必须是整型，而 m 是否大于 n 则无关紧要。例子：

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]
```

产生以下输出：

First example:

```
1 2 3 4 5
```

Second example:

```
2 1 0 -1 -2
```

Third example:

```
0 1
```

Fourth example:

```
[1..3]
```

注意：range operator 只在#set 和#foreach 中有效。

网页设计者使用标准尺寸制作表格，但是某时候会没有足够的数据填入表格，就会发现 range 非常有用。

### Advanced Issue: Escaping and!

当一个 reference 被 “!” 分隔时，并且在它之前有逃逸符时，reference 将以特殊的方式处理。注意这种方式与标准的逃逸方式是不同的。对照如下：

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

```
$!foo
```

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

```
$\\!foo
```

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

这将显示为：

```
$!foo
```

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

```
$\\!foo
```

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

\ 在 \$ 前面跟逃避规则相比较；

```
\$foo
```

```
\$!foo
```

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

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

这将显示为：

```
\$foo
```

```
\$!foo
```

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

```
\bar
```

### Velocimacro 杂记

这章节是一个小型涉及 Velocimacros 的常见问题解答。这章节会随时间的变化而变化，所以它的作用有时是检验新的信息。

注意：在这章节‘Velocimacro’简称为‘VM’。

1. 我能用一个标识或者另外的 VM 作为一个 VM 的参数吗？

例如：#center ( #bold( “hello” ) )

答：不可以。一个 directive 的参数使用另外一个 directive 是不合法的。

但是，还是有些事情你可以作的。最简单的方式就是使用双引号，所以你可以这样做：

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

```
#center( $stuff )
```

上面的格式也可以缩写为一行：

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

请注意在下面的例子中参数被 evaluated 在 Velocimacro 内部，而不是在 calling level。例子：

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

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

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

输出结果为：

```
Outer : inner : outerlala
```

因为#inner(\$bar)的赋值发生在#outer()，所以\$bar 的值在#outer()赋给的。

记住 Velocity 的特性：参数的传递是 By Name 的。例如：

```
#macro( foo $color )
    <tr bgcolor=$color><td>Hi</td></tr>
    <tr bgcolor=$color><td>There</td></tr>
#end

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

以上代码将导致 rowColor() 方法两次调用，而不是一次。为了避免这种现象的出现，我们可以按照下面的方式执行：

```
#set($color = $bar.rowColor())
#foo( $color )
```

2. 我能通过#parse() 登记 velocimacros 吗？

答：目前，Velocimacros 必须在第一次被模板调用前被定义。这就意味着你的 #macro() 声明应该出现在使用 Velocimacros 之前。

如果你试图#parse() 一个包含#macro() directive 的模板，这一点是需要牢记的。因为#parse() 发生在运行期，但是解析器在 parsetiem 决定一个看似 VM 元素的元素是否是一个 VM 元素，这样#parse() 一组 VM 声明将不按照预期的样子工作。为了得到预期的结果，只需要你简单的使用 velocimacro.library 使得 Velocity 在启动时加载你的 VMs。

3. 什么是 velocimacro 自动加载？

答：velocimacro.library.autoreload 是专门为开发而非产品使用的一个属性。此属性的默认值是 false。

当<type>.resource.loader.cache = false 连同一起设为 true () (<type> 是你用的资源器, 例如 'file'), 当你编译它们的时候 Velocity 引擎会自动重载那些在你的 velocimacro 库文件中的修改, 所以你不用 servlet 引擎(或者应用程序)或者其他窍门来使你的 velocimacros 重载。

这里有一个简单的属性配置。

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

不要在你的作品中出现上述的配置

## 字符串串联

开发人员最常问的问题是我如何做字符拼接? 在 java 中是使用 “+” 号来完成的。

在 VTL 里要想实现同样的功能你只需要将需要联合的 reference 放到一起就行了。例如:

```
#set( $size = "Big" )
#set( $name = "Ben" )
The clock is $size$name.
```

输出结果将是: The clock is BigBen.。更有趣的情况是:

```
#set( $size = "Big" )
#set( $name = "Ben" )
#set($clock = "$size$name" )
The clock is $clock.
```

上例也会得到同样的结果。最后一个例子, 当你希望混合固定字段到你的 reference 时, 你需要使用标准格式:

```
#set( $size = "Big" )
#set( $name = "Ben" )
#set($clock = "${size}Tall$name" )
The clock is $clock.
```

输出结果是: The clock is BigTallBen.。使用这种格式主要是为了使得 \$size 不被解释为 \$sizeTall。

## 反馈

如果你在手册中遇到任何错误或者有其他关于 Velocity 用户手册的反馈, 请 email 到 Velocity user list。谢谢!