User guide

Created for the sole purpose: save keystroke, minimize typographic errors

When the Velocimacro is invoked, it must be called with **the same number of arguments** with which it was defined

Velocimacros can be defined *inline* in a Velocity template, *inline* means **UNAVAILABLE** to other Velocity templates

When passing reference as arguments to Velocimacros, please note that references are passed ‘by name’.

|  |
| --- |
| **This means that their values is ‘generated’ at each use inside the Velocimacro** |

|  |
| --- |
| #macro( callme $a )  $a $a $a  #end  ## Pay attention here: **$foo.bar() will be called 3 times**(NOT once when passed into callme)  #callme( $foo.bar() ) |

At first glance, this feature appears surprising, but when you take into consideration the original motivation behind Velocimacros (eliminate copy/paste duplication of commonly used VTL), it makes sense.

It allows you to do things like pass stateful objects, such as an object that generate colors in a repeating sequence for coloring table rows (colorRender.getItsColor()), into the Velocimacro

Velocimacro.permission.allow.inline=true, determine whether Velocimacros can be defined in regular templates (NOT only in velocimacro.library=VM\_global\_library.vm)（默认true：**可以在所有页面定义macro**）

Velocimacro.permissions.allow.inline.local.scope=false, whether Velocimacros defined inline are ‘visible’ only to the defining template. true inline Velocimacros only usable by the defining template（默认false：**普通页面中macro在所有 页面中使用**）

A hack：如果设置为true，则template自己的定义会覆盖global的定义，因为search Velocimacro时，template’s namespace before global namespace

velocimacro.context.localscope=false（默认false：macro中进行的 #set**会改变context**），如果不想改变context，则使用 #set($macro.foo=”bar”), $macro.myMarcoLocalVariable

|  |
| --- |
| WARNING: The velocimacro.context.localscope feature is deprecated and will be removed in Velocity 2.0. Instead, please use the $macro scope to store references that must be local to your macros (e.g. #set( $macro.foo = 'bar' ) and $macro.foo). This $macro namespace is automatically created and destroyed for you at the beginning and end of the macro rendering. |

velocimacro.arguments.strict = false（默认false：传入参数个数不对时，**不会报错**）

velocimacro.provide.scope.control = false（默认false：在macro调用时**还是global的**namespace）

MacroEvaluationStrategy

在macro中，是如何给传入的参数赋值的

1.7: macro中#set的reference按着macro 中#set时的名字，全部进入到context中去

仅仅就是按着名字来#set的；主调中的reference不会有任何变化

1.7

Macro**定义中 #set 操作reference时**，将以此时名字作为key的key-value放入context

主调中传入的reference不会受到影响（除非主调时传入的reference名字和macro定义中使用同样的名字）

macro定义中，#set操作reference时，绝对不会把Reference对应的值，或者直接就传入String literal作为key，从而在context中存入key-value pair的

macro定义中，#set操作reference时，

1.3.1中，macro定义中，#set操作reference时，绝对不会把Reference对应的值，或者直接就传入String literal作为key，从而在context中存入key-value pair的

macro定义中，#set操作reference时，macro定义所使用的reference名称，被设置了后，在macro范围内，context中存在这些key-value pairs；但是，出了macro的定义，这些就不存在了，而是，传入时的引用指向的值本身改变了；

如果不是作为参数引用传入的，而是直接在macro中#set一个值，则，此值会放入context（for both 1.7 and 1.3.1）

In 1.3.1，macro定义中使用的reference名字不会作为key放入context的；但是，会改变传入的reference

1.7中，macro定义中的也会放入，而且不会改变传入的reference

It’s a mix between Call-by-macro-expansion, Call-by-sharing, call by value and other behaviors

在**非严格(也就是我们一般用的)**模式下

1. Call-by-macro-expansion internally
2. Call-by-sharing external effects
3. 传入的是non-lvalue，却硬要给其赋值，那么就会自动生成global variables

1. Lazy evaluation:用到时才执行传入的参数

2. Call-by-macro-expansion:每次用到时，**都会执行一次**传入的参数（也就是说，传入的内容会被执行多次）。这等同于C语法中macro

3. Call-by-sharing: **不是**“传引用”，在macro定义中，给参数#set新值，等macro执行完毕后，参数的值**并没有**改变；但是，如果传入的是一个map，那么，修改map中的值，当macro执行完毕后，map中的值确实被改变了。这等同于Java语法中的方法调用。Warning：这是 1.7中引入的特性。在1.7之前，就是true Call-by-macro-expansion

4. error-free non-lvalue assignment: macro有个参数$thirdArg，而我们传入” thirdConstantString”

在macro定义中，我们有#set($thirdArg = "Set thridArg In macro")

那么，在执行过此句后，在macro体中使用时，当使用$thirdArg时，我们用到的是 "Set thridArg In macro"，而不是传入的” thirdConstantString”（the new variable will shadow the formal parameter）

这近似于 C语法中的方法调用

转义--转义 valid VTL Reference（即语法上，是一个有效的VTL reference，注意：此valid VTL reference可能定义在context中，也可能没有哦）

$2.50 would not be mistaken for a reference(因为VTL identifier必须是, upper/lower case的字母打头。注意：中杠、下岗都是不行的，当然，他们可以放在非最前)

email**已经定义在**context中了

\$email，就是把它转义了，会被render成 \$email

如果我们想在前面加个backslash

\\$email --> \foo

\\\$email --> \$email

\是从左开始算起的，我们可以称其为“bind-from-left”rule，即---------------------------->

\\\$email --> \$email（前面两个\的后果是得到一个string literal \，后面的 \$email 导致render时直接输出$email(**而不是**使用context中的内容替换)）

在强调一次，我们上面讲的这些，有个大前提：**email已经定义在context中了**

如果 email**没有定义**在context中呢？

那就无所谓 “转不转义” 啦，直接原样输出，例如：

$email

\$email

\\$email

\\\$email

renders as

$email

\$email

\\$email

\\\$email

结论：转义时，如果reference根本就没有定义，那么，就原样输出了（无所谓“转不转义”了）

转义--转义 invalid VTL Reference对于invalid VTL References（指语法上不能作为reference，**而不是指**“未定义的reference”）如何转义

${my:invalid:non:reference}

注意：上面**不是**一个合法的 VTL reference

我们这么写：\${my:invalid:non:reference}, 这种转义是**不对**的

我们应该这么写：

#set($D = ‘$’)

${D}{my:invalid:non:reference}

这样才是**正确的**转义方法

转义--转义VTL Directives转义

Similar to valid VTL references

\#include( "a.txt" )--> render as #include( "a.txt" )（不会执行include命令了，直接当做string literal输入了）

单个directive，但是多个script elements（例如：if-else-end）

\#if( $jazz )

Vyacheslav Ganelin

\#end

1. if—end被转义了，
2. $jazz依然会被render

#if( true )

Vyacheslav Ganelin

#end

在继续看：

|  |
| --- |
| \\#if( $jazz )  Vyacheslav Ganelin  \\#end |

会得到下面这个

|  |
| --- |
| \ Vyacheslav Ganelin  \ |

一个很好的知识点，注意：

如果#if以newline结尾（whitespace不属于这种情况啊，记住了哦），则，输出中会忽略掉newline。所以，#if下一行的输出就和#if前的 \ 跑到一块去了。

2012-02-16，继续看Velocity（既然咱不搞JSP，就好好看Velocity吧）

Properties:

* Regular Notation: $customer.Address
* Formal Notation: ${purchase.Total}

注意：大括号必须**扩到最后**，**不是**点号前面

Method:

* Regular Notation: $customer.getAddress()
* Formal Notation: ${purchase.getTotal("My Home Page")}

注意：和Properties一样，大括号也是**扩到最后**

|  |
| --- |
| #set的定义：#[{]set[}]($ref = [", ']arg[ ", ' ] )  #set($**!**myVariable = "HelloMarvin")//无效  注意：**不**要**!** |