Guía del usuario - Contenidos

1. [Acerca de esta guía](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#about_this_guide)
2. [¿Cuál es la Velocity?](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#what_is_velocity)
3. [¿Qué puede hacer Velocity por mi?](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#what_can_velocity_do_for_me)
   1. [En el ejemplo de la tienda de barro](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#the_mud_store_example)
4. [Plantillas de Velocity Idioma (VTL): Una introducción](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#velocity_template_language_vtl:_an_introduction)
5. [Hola Mundo Velocity!](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#hello_velocity_world)
6. [Comentarios](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#comments)
7. [Referencias](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#references)
   1. [Variables](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#variables)
   2. [Propiedades](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#properties)
   3. [Métodos](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#methods)
   4. [Reglas consulta de propiedad](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#propertylookuprules)
8. [Referencia de la notación formal](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#formalreferencenotation)
9. [Referencia de la notación Quiet](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#quietreferencenotation)
10. [Obtener literal](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#getting_literal)
    1. [Moneda](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#currency)
    2. [Escapar válidas referencias VTL](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#escapingvalidvtlreferences)
11. [Caso Sustitución](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#case_substitution)
12. [Directivas](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#directives)
    1. [Establecer](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#set)
    2. [Literales](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#stringliterals)
    3. [Instrucciones if-else](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#conditionals)
       1. [Operadores relacionales y lógicos](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#relationalandlogicaloperators)
    4. [Foreach Loops](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#loops)
    5. [Incluir](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#include)
    6. [Analizar](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#parse)
    7. [Deténgase](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#stop)
    8. [Velocimacros](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#velocimacros)
13. [Escapar directivas VTL](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#escaping_vtl_directives)
14. [VTL: Asuntos de formato](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#vtl:_formatting_issues)
15. [Otras Características y Miscelánea](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#other_features_and_miscellany)
    1. [Matemáticas](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#math)
    2. [Operador Gama](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#range_operator)
    3. [Temas avanzados: Escapar y!](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#advanced_issues:_escaping_and_)
    4. [Velocimacro Miscelánea](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#velocimacro_miscellany)
    5. [Concatenación de cadenas](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#string_concatenation)
16. [Feedback](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#feedback)

Acerca de esta guía

La Guía del usuario de Velocity está diseñado para ayudar a los diseñadores y proveedores de contenidos para familiarizarse con la velocidad y la sintaxis de su simple pero potente lenguaje de script, la plantilla Idioma Velocity (VTL). Muchos de los ejemplos de esta guía de acuerdo con el uso de Velocity para incrustar contenido dinámico en páginas web, pero todos los ejemplos de VTL son igualmente aplicables a otras páginas y plantillas.

Gracias por elegir la velocidad!

¿Cuál es la velocidad?

Velocity es un motor de plantillas basado en Java. Permite a los diseñadores de páginas web con los métodos de referencia definidos en el código Java. Los diseñadores web pueden trabajar en paralelo con los programadores de Java para desarrollar sitios web según el Modelo-Vista-Controlador (MVC) el modelo, lo que significa que los diseñadores de páginas web pueden centrarse exclusivamente en la creación de un sitio bien diseñado, y los programadores pueden centrarse exclusivamente en la escritura de la parte superior primer nivel de código. Velocity separa el código Java de las páginas web, por lo que el sitio web más fácil de mantener en el largo plazo y ofrecer una alternativa viable a [Java Server Pages](http://java.sun.com/products/jsp/) (JSP) o [PHP](http://www.php.net/) .

La velocidad se puede utilizar para generar páginas web, SQL, PostScript y otro de salida a partir de plantillas. Se puede utilizar ya sea como una utilidad independiente para la generación de código fuente y los informes, o como un componente integrado de otros sistemas. Cuando haya terminado, Velocity proveerá servicios de plantilla para el [Turbine](http://java.apache.org/turbine/) framework de aplicaciones web. Velocity + Turbine ofrecerá un servicio de plantilla que va a permitir que las aplicaciones web se desarrollan de acuerdo a un modelo MVC cierto.

¿Qué puede hacer Velocity por mi?

**El barro tienda Ejemplo**

Suponga que usted es un diseñador de página para una tienda en línea que se especializa en la venta de barro. Digamos que es "La tienda Mud Online".Negocio está prosperando. Los clientes hacen pedidos para diversos tipos y cantidades de lodo. Ellos inicie sesión en su sitio utilizando su nombre de usuario y contraseña, que les permite ver sus órdenes y comprar más barro. En este momento, Terracota Barro está en venta, que es muy popular. Una minoría de sus clientes regularmente compra Lodo Rojo Brillante, que también está en venta, aunque no tan popular y por lo general relegado al margen de su página web. La información sobre cada cliente se realiza un seguimiento en la base de datos, así que un día surge la pregunta, ¿Por qué no usar Velocity para apuntar ofertas especiales de barro para los clientes que están más interesados ​​en este tipo de barro?

La velocidad hace que sea fácil de personalizar las páginas web a sus visitantes en línea. Como un diseñador de sitios web en la sala de barro, que desea que la página web que el cliente verá después de acceder a su sitio.

Usted cumple con los ingenieros de software en su empresa, y todo el mundo ha aceptado que *$ cliente* tendrá la información relativa al cliente actualmente registrado, que *$ mudsOnSpecial* será todo el tipo de barro a la venta en la actualidad. El *$ flogger* objeto contiene métodos que ayudan a la promoción. Para la tarea en cuestión, vamos a ocuparnos sólo con estas tres referencias. Recuerde, usted no tiene que preocuparse acerca de cómo los ingenieros de software de extraer la información necesaria de la base de datos, sólo tiene que saber cómo funciona. Esto le permite seguir con su trabajo, y permite a los ingenieros de software de seguir adelante con ellos.

Puedes insertar el siguiente enunciado VTL en la página web:

<HTML>

<BODY>

Hola $ customer.Name!

<table>

# Foreach ($ lodo en $ mudsOnSpecial)

# If ($ customer.hasPurchased ($ lodo))

<tr>

<td>

$ Flogger.getPromo ($ lodo)

</ Td>

</ Tr>

# End

# End

</ Table>

Los detalles exactos del *foreach* declaración se describirán en mayor detalle en breve, lo que importa es el impacto que este pequeño script puede tener en su sitio web. Cuando un cliente con una inclinación de Lodo Rojo Brillante se conecta, y Lodo Rojo Brillante está en venta, eso es lo que el cliente verá, de manera destacada. Si otro cliente con un largo historial de compras de Lodo Terracota inicia sesión, el anuncio de una venta de barro terracota será frente y al centro.La flexibilidad de Velocity es enorme y limitado solamente por su creatividad.

Documentado en la Referencia VTL son los muchos otros elementos de velocidad, que en conjunto le dan el poder y la flexibilidad que necesita para hacer su sitio web una web *presencia* . A medida que se familiarice con estos elementos, usted comenzará a liberar el poder de la velocidad.

Plantillas de Velocity Idioma (VTL): Una introducción

El Lenguaje de Plantillas Velocity (VTL) tiene por objeto proporcionar la más fácil, más simple y más limpia de incorporar contenido dinámico en una página web. Incluso un desarrollador de páginas web con poca o ninguna experiencia en programación pronto será capaz de utilizar VTL para incorporar contenido dinámico en un sitio web.

VTL utiliza *referencias* para incrustar contenido dinámico en un sitio web, y una variable es un tipo de referencia. Las variables son un tipo de referencia que puede referirse a algo definido en el código de Java, o puede obtener su valor de un VTL *declaración* en la propia página web. He aquí un ejemplo de un enunciado VTL que podrían incluirse en un documento HTML:

# Set ($ a = "Velocity")

Esta declaración VTL, como todos los enunciados VTL comienza con el *#* de caracteres y contiene una directiva: *set* . Cuando un visitante solicita en línea a su página web, el Motor de Plantillas de Velocity buscará a través de su página Web para encontrar todo *#* caracteres, luego determinar que marcan el comienzo de enunciados VTL y cuáles de los *#* personajes que nada tienen que ver con VTL.

El *#* carácter es seguido por una Directiva, *set* . El *conjunto* Directiva utiliza una expresión (entre paréntesis) - una ecuación que asigna un *valor* a una *variable de* . La variable aparece en el lado izquierdo y su valor en el lado derecho, y los dos están separados por un *=* carácter.

En el ejemplo anterior, la variable es *un $* y el valor es *de velocidad* . Esta variable, al igual que todas las referencias, comienza con el *$* carácter. Los valores de cadena siempre se escriben entre comillas, comillas simples o dobles. Las comillas simples se asegurará de que el valor de cotización será asignado a la referencia de inmediato. Las comillas dobles le permiten usar referencias y directivas de velocidad para interpolar, como "Hola $ nombre", en la que *$ name*será reemplazado por el valor actual antes de esa cadena literal se asigna a la parte izquierda del *=*

La siguiente regla puede ser útil para comprender mejor cómo funciona Velocity: **Referencias empiezan con *$* y se utilizan para conseguir algo. Las directivas comienzan con *#* y se utilizan para hacer algo.**

En el ejemplo anterior *# set* se utiliza para asignar un valor a una variable. La variable, *un $* , a continuación, se puede utilizar en la plantilla a la salida "velocidad".

Hola Mundo Velocity!

Una vez que el valor se ha asignado a una variable, puede hacer referencia a la variable en cualquier parte del documento HTML. En el siguiente ejemplo, se asigna un valor a *$ foo* y hace referencia más adelante.

<html>

<body>

# Set ($ foo = "Velocity")

Hola $ foo mundo!

</ Body>

<html>

El resultado es una página web que imprime "Hola Mundo Velocity".

Para hacer las declaraciones que contienen directivas VTL más legibles, te animamos a comenzar cada enunciado VTL en una nueva línea, aunque no están obligados a hacerlo. El *conjunto* Directiva será revisado en mayor detalle más adelante.

Comentarios

Comentarios permite a texto descriptivo para ser incluido que no se coloca en la salida del motor de plantillas. Los comentarios son una forma útil de recordarse a sí mismo y explicar a los demás lo que sus enunciados VTL están haciendo, o cualquier otro propósito que encuentra útiles. A continuación se muestra un ejemplo de un comentario en VTL.

# # Este es un comentario de línea.

Una sola línea de comentario comienza con *# #* y termina al final de la línea. Si usted va a escribir unas pocas líneas de comentario, no hay necesidad de tener numerosos comentarios de una sola línea. Comentarios de varias líneas que comienzan con *# \** y terminan con *\* #* , están disponibles para manejar esta situación.

Este es el texto que se encuentra fuera del comentario de varias líneas.

Visitantes en línea pueden ver.

# \*

Así comienza un comentario de varias líneas. Visitantes en línea no se

ver este texto porque el Motor de Plantillas de Velocity

ignorarlo.

\* #

Aquí está el texto fuera del comentario de varias líneas, es visible.

Aquí hay algunos ejemplos para aclarar cómo funcionan una sola línea y comentarios de varias líneas:

Este texto es visible. # # Este texto no lo es.

Este texto es visible.

Este texto es visible. # \* Este texto, como parte de una multilínea

comentario, no es visible. Este texto no es visible, sino que también es

parte del comentario de varias líneas. Esto todavía no texto

visibles. \* # Este texto se encuentra fuera de la observación, por lo que es visible.

# # Este texto no es visible.

Hay un tercer tipo de comentario, el bloque de comentario VTL, que puede ser utilizado para almacenar información tal como el autor de un documento y la información de versión:

# \*\*

Este es un comentario de bloque y VTL

puede ser utilizado para almacenar dicha información

como el autor de documentos y control de versiones

información:

@ Author

@ Version 5

\* #

Referencias

Hay tres tipos de referencias en el VTL: variables, propiedades y métodos. Como diseñador con el VTL, usted y sus ingenieros deben llegar a un acuerdo sobre los nombres específicos de referencias para que pueda utilizarlas correctamente en sus plantillas.

Todo lo que viene hacia y desde una referencia se trata como un objeto String. Si hay un objeto que representa *$ foo* (como un objeto Integer), entonces Velocity llamará su . toString () método para resolver el objeto en una cadena.

**Variables**   
  
La notación abreviada de una variable consiste en un personaje principal "$" seguido de un VTL *identificador* . Un identificador VTL debe comenzar con un carácter alfabético (a .. z o A .. Z). El resto de los personajes se limitan a los siguientes tipos de caracteres:

* Alfabética (a .. z, A .. Z)
* numérico (0 .. 9)
* guión ("-")
* guión bajo ("\_")

Estos son algunos ejemplos de referencias a variables válidos en el VTL:

$ Foo

$ Mudslinger

$ Barro eslinga

$ Mud\_slinger

$ MudSlinger1

Cuando las referencias VTL una variable, como por ejemplo *$ foo* , la variable puede obtener su valor desde un *conjunto de* directiva en la plantilla, o desde el código de Java. Por ejemplo, si la variable de Java *$ foo* tiene el valor *bar* en el momento en que se pidió a la plantilla, *bar* reemplaza todas las instancias de *$ foo* en la página web. Alternativamente, si incluyo la declaración

# Set ($ foo = "bar")

La salida será el mismo para todas las instancias de *$ foo* que siguen a esta directiva.

**Propiedades**   
  
El segundo sabor de referencias VTL son las propiedades y las propiedades tienen un formato distintivo. La notación abreviada consta de un líder *$* carácter seguido de un identificador VTL, seguido por un carácter de punto (".") y otro identificador VTL. Estos son ejemplos de referencias de propiedades válidas en el VTL:

$ Cliente.Direccion

$ Compra.Total

Tome el primer ejemplo, *$ cliente.Direccion* . Puede tener dos significados. Puede significar, Busque en la tabla hash identificado como *cliente* y devolver el valor asociado a la clave de *Dirección* . Pero *$ cliente.Direccion* también puede referirse a un procedimiento (referencias que se refieren a los métodos serán discutidos en la próxima sección); *$ cliente.Direccion* podría ser una forma abreviada de escribir *$ cliente.getDireccion ()* . Cuando se solicita la página, Velocity determinará cuál de estas dos posibilidades tiene sentido, y luego devolver el valor apropiado.

**Métodos**   
  
Un método es definido en el código Java y es capaz de hacer algo útil, como el funcionamiento de un cálculo o llegar a una decisión. Los métodos son referencias que constan de un carácter de "$" seguido de un identificador VTL, seguido de un VTL *Método cuerpo* . Un Método Cuerpo VTL consiste en un identificador VTL seguido por un carácter de paréntesis de apertura ("("), seguido de una lista de parámetros opcional, seguido por el carácter de paréntesis de cierre (")"). Estos son ejemplos de referencias método válido en la VTL:

$ Cliente.getDireccion ()

$ Compra.getTotal ()

$ Page.setTitle ("Mi Página")

$ Person.setAttributes (["extraño", "raro", "Emocionado"])

Los dos primeros ejemplos - *$ cliente.getDireccion ()* y *$ compra.getTotal ()* - pueden parecer similares a los utilizados en la sección Propiedades anterior, *$ cliente.Direccion* y *$ compra.Total* . Si has pensado que estos ejemplos deben estar relacionados algunos de alguna manera, estás en lo correcto!

Propiedades VTL se pueden utilizar como una notación abreviada para los métodos de VTL. La propiedad *$ cliente.Direccion* tiene exactamente el mismo efecto que utilizar el método *$ cliente.getDireccion ()* . En general, es preferible utilizar una propiedad cuando esté disponible. La principal diferencia entre propiedades y métodos es que se puede especificar una lista de parámetros a un método.

La notación abreviada se puede utilizar para los siguientes Métodos

$ Sun.getPlanets ()

$ Annelid.getDirt ()

$ Album.getPhoto ()

Podríamos esperar que estos métodos para devolver los nombres de los planetas pertenecientes al sol, alimentar a nuestra lombriz de tierra, o conseguir una fotografía de un álbum. Sólo la notación larga funciona para los siguientes métodos.

$ Sun.getPlanet (["tierra", "Mars", "Neptuno"])

# # No se puede aprobar una lista de parámetros con $ sun.Planets

$ Sisyphus.pushRock ()

# # Velocity asume que quiero decir $ sisyphus.getRock ()

$ Book.setTitle ("Homenaje a Cataluña")

# # No se puede aprobar una lista de parámetros

**Reglas consulta de propiedad**   
  
como se mencionó anteriormente, las propiedades a menudo se refieren a los métodos del objeto padre. La velocidad es muy inteligente cuando averiguar qué método corresponde a una propiedad requerida. Se ensaya diferentes alternativas basadas en varias convenciones de nombres establecidas. La secuencia exacta de búsqueda depende de si o no el nombre de la propiedad comienza con una letra mayúscula. Para los nombres en minúscula, como *$ cliente.Direccion* , la secuencia es

1. GetAddress ()
2. getAddress ()
3. get ("dirección")
4. ISAddress ()

Para los nombres de propiedad en mayúsculas como *$ cliente.Direccion* , es un poco diferente:

1. getAddress ()
2. GetAddress ()
3. get ("Dirección")
4. ISAddress ()

**Formal notación de la referencia**   
  
notación de taquigrafía para referencias se utilizó para los ejemplos mencionados anteriormente, pero también hay una notación formal para referencias, que se demuestra a continuación:

$ {Mudslinger}

$ {Cliente.Direccion}

$ {Compra.getTotal ()}

En casi todos los casos van a utilizar la notación abreviada para referencias, pero en algunos casos se requiere la notación formal para un procesamiento correcto.

Supongamos que esté construyendo una sentencia sobre la marcha donde *$ vice* iba a ser utilizado como base de la palabra en el nombre de una oración. El objetivo es permitir a alguien para elegir la palabra base y producir uno de los dos resultados siguientes: "Jack es un pirómano." o "Jack es un cleptómano.".Usando la notación abreviada sería inadecuada para esta tarea. Consideremos el siguiente ejemplo:

Jack es un $ viciomano.

No hay ambigüedad aquí, y Velocity asume que *$ viciomano* , no *$ vicio* , es el identificador que se refiere a su uso. Al no encontrar el valor de *$ viciomano* , devolverá *$ viciomano* . Utilizando la notación formal puede resolver este problema.

Jack es un $ {} vice maníaco.

Ahora Velocity sabe que *$ vicio* , no *$ viciomano* , es la referencia. Notación formal suele ser útil cuando las referencias están directamente adyacente al texto en una plantilla.

**Referencia de la notación Quiet**   
  
Cuando Velocity encuentra una referencia indefinida, su comportamiento normal es para la salida de la imagen de la referencia. Por ejemplo, supongamos que la siguiente referencia aparece como parte de una plantilla VTL.

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

Cuando inicialmente se carga el formulario, la variable de referencia *$ correo electrónico* no tiene un valor, pero se prefiere un campo de texto en blanco para uno con un valor de "$ email". Con la tranquilidad referencia de la notación evita el comportamiento normal de la velocidad, en lugar de utilizar *$ email* en el VTL usted utilizaría *$ email!* . Así que el ejemplo anterior puede parecer similar al siguiente:

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

Ahora, cuando el formulario se carga inicialmente y *$ email* todavía no tiene ningún valor, una cadena vacía se emitirá en lugar de "$ email".

Referencia de la notación formal y tranquilo se puede utilizar en conjunto, como se demuestra a continuación.

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

Obtener literal

VTL utiliza caracteres especiales, como *$* y *#* , para hacer su trabajo, por lo que algunos cuidados añadido se debe tomar en el uso de estos caracteres en sus plantillas. Esta sección trata de escapar de la *$* personaje.

**Moneda**   
  
No hay ningún problema por escrito "Compré un saco £ 4 de patatas en el mercado del agricultor por sólo $ 2.50!" Como se ha mencionado, un identificador VTL siempre comienza con una letra en mayúsculas o minúsculas, por lo que $ 2.50 no sería un error para una referencia.

**Escapar válidos VTL Referencias**   
  
que puede ocurrir que existe el potencial para la velocidad a confundirse. *Escapar* caracteres especiales es la mejor manera de manejar los caracteres especiales de VTL en sus plantillas, y esto se puede hacer usando la barra invertida ( *\* ) carácter.

# Set ($ email = "foo")

$ Email

Si Velocity encuentra una referencia en la plantilla VTL a *$ email* , buscará el contexto de un valor correspondiente. Aquí la salida será *foo* , porque *$ correo electrónico* se define. Si *$ email* no se ha definido, la salida será *$ email* .

Supongamos que *$ email* está definido (por ejemplo, si tiene el valor *foo* ), y que desea emitir *$ email* . Hay algunas maneras de hacer esto, pero la más sencilla es utilizar el carácter de escape.

# # La siguiente línea define $ correo en esta plantilla:

# Set ($ email = "foo")

$ Email

\ $ Email

\ \ $ Correo electrónico

\ \ \ $ Correo electrónico

renderiza como

foo

$ Email

\ Foo

\ $ Email

Tenga en cuenta que el *\* carácter se unen a la *$* desde la izquierda. El bind-de-izquierda causas regla *\ \ \ $ correo electrónico* para hacer como *\ \ $ correo electrónico* . Comparar estos ejemplos para aquellos en los que *$ correo electrónico* no está definido.

$ Email

\ $ Email

\ \ $ Correo electrónico

\ \ \ $ Correo electrónico

renderiza como

$ Email

\ $ Email

\ \ $ Correo electrónico

\ \ \ $ Correo electrónico

Aviso Velocity maneja referencias que se definen de manera diferente de los que no se han definido. Aquí está una directiva set que le da *$ foo* el valor*menguante* .

# Set ($ foo = "menguante")

$ Luna = $ foo

La salida será: *$ luna menguante =* - donde *$ luna* se emite como un literal, ya que no está definido y *creciente* se emite en lugar de *$ foo* .

También es posible para escapar de directivas VTL; esto se describe con más detalle en la sección de directivas.

Caso Sustitución

Ahora que usted está familiarizado con las referencias, se puede empezar a aplicar de manera efectiva en sus plantillas. Referencias Velocity aprovechan algunos principios Java que diseñadores de plantillas encontrarán fácil de usar. Por ejemplo:

$ Foo

$ Foo.getBar ()

# # Es el mismo que

$ Foo.bar

$ Data.setUser ("Jon")

# # Es el mismo que

# Set ($ data.User = "jon")

$ Data.getRequest (). GetServerName ()

# # Es el mismo que

$ Data.Request.ServerName

# # Es el mismo que

$ {Data.Request.ServerName}

Estos ejemplos ilustran usos alternativos de las mismas referencias. La velocidad se aprovecha de la introspección de Java y características de frijol para resolver los nombres de referencia a ambos objetos en el contexto, así como los métodos de objetos. Es posible integrar y evaluar referencias casi cualquier parte de su plantilla.

Velocity, que se basa en las especificaciones de frijol definidas por Sun Microsystems, es sensible, sin embargo, sus desarrolladores han esforzó en detectar y corregir los errores del usuario siempre que sea posible. Cuando el método *getFoo ()* se refiere en una plantilla por $ bar.foo , Velocity intentará primero $ getFoo . Si esto falla, se intentará $ getFoo . Del mismo modo, cuando una plantilla se refiere a $ bar.Foo , Velocity intentará *$ getFoo ()* primero y luego tratar *getFoo ()* .

Nota: *. Las referencias a las variables de instancia en una plantilla no se resuelven* sólo referencias a los equivalentes de atributo de métodos de captador / JavaBean se resuelven (es decir, $ foo.Name no resolver a la de clase Foo getNombre () método de instancia, pero no a un público Nombre de variable de instancia de Foo).

Directivas

Referencias permiten a los diseñadores de plantillas para generar contenido dinámico para sitios web, mientras que *las directivas* - fácil de usar elementos de script que se pueden utilizar para manipular creativamente la salida del código de Java - permiten a los diseñadores web tienen verdaderamente responsable de la apariencia y el contenido de la web sitio.

Directivas siempre comienzan con un # . Como referencia, el nombre de la directiva puede ir precedida por un { y } símbolo. Esto es útil con las directivas que están inmediatamente seguido de un texto. Por ejemplo, el siguiente produce un error:

# If ($ a == 1) bastante cierto modo elseno #! # End

En tal caso, utilizar los soportes para separar # else del resto de la línea.

# If ($ a == 1) muy cierto # {else} No puede ser! # End

**# Set**

El *# set* directiva se utiliza para ajustar el valor de referencia. Un valor puede ser asignada a una referencia variable o una referencia de la propiedad, y esto se produce en soportes, como se ha demostrado:

# Set ($ primate = "mono")

# Set ($ customer.Behavior = $ primate)

El lado izquierdo (LHS) de la asignación debe ser una referencia variable o una referencia de la propiedad. El lado derecho (RHS) puede ser uno de los siguientes tipos:

* Referencia Variable
* Cadena literal
* Referencia del inmueble
* Método de referencia
* Número literal
* ArrayList
* Mapa

Estos ejemplos demuestran cada uno de los tipos mencionados anteriormente:

# Set ($ mono = $ cuenta) Referencia # # variables

# Set ($ monkey.Friend = "monica") # # string literal

# Set ($ monkey.Blame = $ whitehouse.Leak) # # Referencia del inmueble

# Set ($ monkey.Plan = $ spindoctor.weave ($ web)) # # Referencia método

# Set ($ monkey.Number = 123) # # Número literal

# Set ($ monkey.Say = ["No", $ mi, "culpa"]) # # ArrayList

# Set ($ monkey.Map = {"banana": "bueno", "carne asada": "malo"}) # # Map

NOTA: Para el ejemplo ArrayList los elementos definidos con el [..] operador son accesibles usando los métodos definidos en la clase ArrayList. Así, por ejemplo, se puede acceder al primer elemento anterior mediante $ monkey.Say.get (0).

Del mismo modo, en el ejemplo del mapa, los elementos definidos en el {} operador son accesibles usando los métodos definidos en la clase Map. Así, por ejemplo, se puede acceder al primer elemento anterior mediante $ monkey.Map.get ("banana") para devolver una cadena "bueno", o incluso $ monkey.Map.banana para devolver el mismo valor.

Los RHS también puede ser una expresión aritmética simple:

# Set ($ value = $ foo + 1)

# Set ($ value = $ bar - 1)

# Set ($ value = $ foo \* $ bar)

# Set ($ value = $ foo / $ bar)

Si el lado derecho es una propiedad o un método de referencia que se evalúa como *NULL* , será **que no** se asignará al primer miembro. Dependiendo de cómo esté configurado de velocidad, que por lo general no es posible eliminar una referencia existente a partir del contexto a través de este mecanismo. (Tenga en cuenta que esto puede ser permitido por el cambio de una de las propiedades de configuración de velocidad). Esto puede ser confuso para los recién llegados a la velocidad. Por ejemplo:

# Set ($ result = $ consulta.criterio ("nombre"))

El resultado de la primera consulta es $ resultado

# Set ($ result = $ consulta.criterio ("Dirección"))

El resultado de la segunda consulta es $ resultado

Si *$ consulta.criterio ("name")* devuelve la cadena "proyecto de ley", y *$ consulta.criterio ("dirección")* devuelve *nulo* , lo anterior VTL rendirá como las siguientes:

El resultado de la primera consulta es proyecto de ley

El resultado de la segunda consulta es proyecto de ley

Esto tiende a confundir a los recién llegados que construyen *# foreach* bucles que intentan *# coloca* una referencia a través de una propiedad o un método de referencia, a continuación, pruebe de inmediato que la referencia con un *# Si* directiva. Por ejemplo:

# Set ($ criterios = ["Nombre", "Dirección"])

# Foreach ($ criterios de $ criterios)

# Set ($ result = $ consulta.criterio ($ criterios))

# If ($ resultado)

Consulta fue exitosa

# End

# End

En el ejemplo anterior, no sería prudente confiar en la evaluación de *$ resultado* para determinar si una consulta se ha realizado correctamente. Después de *$ resultado* ha sido *# set* (añadido al contexto), no se puede establecer de nuevo a *nulo* (eliminado por el contexto). Los detalles del *caso #* y *# foreach* directivas se tratan más adelante en este documento.

Una solución a esto sería pre-set *$ resultado* a *false* . Entonces, si el *$ consulta.criterio ()* llamada falla, se puede comprobar.

# Set ($ criterios = ["Nombre", "Dirección"])

# Foreach ($ criterios de $ criterios)

# Set ($ resultado = false)

# Set ($ result = $ consulta.criterio ($ criterios))

# If ($ resultado)

Consulta fue exitosa

# End

# End

A diferencia de las otras directivas de Velocity, el *# set* Directiva no tiene un *# end* comunicado.

**Literales**

Cuando se usa el *# set* Directiva, literales de cadena que se incluyen entre comillas dobles serán analizadas y prestados, como se muestra:

# Set ($ directoryRoot = "www")

# Set ($ templateName = "index.vm")

# Set ($ template = "$ directoryRoot / $ templateName")

$ Template

La salida será

www / index.vm

Sin embargo, cuando la cadena literal se encierra en comillas simples, no va a ser analizada:

# Set ($ foo = "bar")

$ Foo

# Set ($ blargh = '$ foo')

$ Blargh

Esto se representa como:

bar

$ Foo

De forma predeterminada, esta característica de utilizar comillas simples para representar texto no analizada está disponible en Velocity. Este defecto se puede cambiar editando velocity.properties tal que stringliterals.interpolate = false .

Alternativamente, el *# literal* elemento script permite al diseñador de la plantilla a utilizar fácilmente grandes trozos de contenido no interpretado en clave VTL.Esto puede ser especialmente útil en lugar de [escapar](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#EscapingVTLDirectives) múltiples directivas.

# Literal ()

# Foreach ($ woogie boogie en $)

nada va a pasar a $ woogie

# End

# End

Se representa como:

# Foreach ($ woogie boogie en $)

nada va a pasar a $ woogie

# End

Condicionales

**If / ElseIf / Else**

El *# Si* directiva Velocity permite texto a incluir cuando se genera la página web, en el condicional que si la declaración es verdadera. Por ejemplo:

# If ($ foo)

Velocity <strong> </ strong>

# End

La variable *$ foo* se evalúa para determinar si es cierto, lo que va a pasar en una de dos circunstancias: (i) *$ foo* es un valor lógico (verdadero / falso), que tiene un valor verdadero, o (ii) el valor no es nulo . Recuerde que el contexto Velocity solamente contiene objetos, por lo que cuando se dice 'booleano', que se representa como un valor booleano (la clase). Esto es cierto incluso para los métodos que devuelven booleano - la infraestructura de introspección devolverá un Boolean del mismo valor lógico.

El contenido entre el *# si* y los *# end* declaraciones se convierten en la salida si la evaluación es cierto. En este caso, si *$ foo* es verdadero, la salida será: "Velocity". Por el contrario, si *$ foo* tiene un valor nulo, o si es un falso booleano, la sentencia se evalúa como falsa, y no hay salida.

Un *# elseif* o *# else* elemento se puede utilizar con un *# si* elemento. Tenga en cuenta que el Motor de Plantillas de Velocity parará en la primera expresión que se encuentra para ser verdad. En el siguiente ejemplo, supongamos que *$ foo* tiene un valor de 15 y *$ bar* tiene un valor de 6.

# If ($ foo <10)

<strong> Go Norte </ strong>

# Elseif ($ foo == 10)

<strong> Go Medio </ strong>

# Elseif ($ bar == 6)

Ir <strong> Sur </ strong>

# Else

<strong> Go West </ strong>

# End

En este ejemplo, *$ foo* es mayor que 10, por lo que las dos primeras comparaciones fallan. Siguiente *$ bar* es comparado a 6, lo cual es cierto, por lo que la salida es **Vaya al sur** .

**Operadores relacionales y lógicos**

Velocity usa el operador equivalente para determinar las relaciones entre las variables. He aquí un ejemplo simple para ilustrar cómo se utiliza el operador equivalente.

# Set ($ foo = "ácido desoxirribonucleico")

# Set ($ bar = "ácido ribonucleico")

# If ($ foo == $ bar)

En este caso está claro que no son equivalentes. Así que ...

# Else

Ellos no son equivalentes y esta será la salida.

# End

Tenga en cuenta que la semántica de *==* son ligeramente diferentes de Java donde *==* sólo puede ser utilizado para probar la igualdad de objeto. En el operador de velocidad equivalente puede ser utilizado para comparar directamente números, cadenas u objetos. Cuando los objetos son de diferentes clases, las representaciones de cadena se obtienen llamando toString () de cada objeto y luego comparar.

Velocity tiene operadores lógicos AND, OR y NOT también. Para más información, por favor vea la [Guía de Referencia VTL](http://velocity.apache.org/engine/releases/velocity-1.5/vtl-reference-guide.html) A continuación se presentan ejemplos que muestran el uso de los operadores lógicos AND, OR y NOT.

# # Y lógico

# If ($ foo && $ bar)

<strong> esto y que </ strong>

# End

El *# if ()* Directiva sólo se evalúa a cierto si tanto *$ foo* y *$ bar* son verdaderas. Si *$ foo* es falso, la expresión se evalúa como falsa, *$ bar* no será evaluada. Si *$ foo* es verdadero, el Motor de Plantillas de Velocity se compruebe el valor de *$ bar* , si *$ bar* es verdad, entonces toda la expresión es verdadera y **esto y que** se convierte en la salida. Si *$ bar* es falso, entonces no habrá salida como toda la expresión es falsa.

OR lógico operadores funcionan de la misma manera, excepto sólo una de las referencias necesita evaluarse como true para que toda la expresión para ser considerado verdadero. Consideremos el siguiente ejemplo.

# # OR lógico

# If ($ foo | | $ bar)

<strong> tal o cual </ strong>

# End

Si *$ foo* es verdadero, el Motor de Plantillas de Velocity no tiene necesidad de mirar a *$ bar* ; si *$ bar* es verdadero o falso, la expresión será verdadera, y **tal o cual** será la salida. Si *$ foo* es falso, sin embargo, *$ bar* tiene que ser revisado. En este caso, si *$ bar* es también falso, la expresión se evalúa como falso y no hay salida. Por otro lado, si *$ bar* es verdad, entonces toda la expresión es verdadera, y la salida es **tal o cual**

Con NO operadores lógicos, sólo hay un argumento:

# # NO lógico

# If ($ foo)

<strong> que NO </ strong>

# End

Aquí, el si *$ foo* es verdadero, entonces *! $ foo* se evalúa como falsa, y no hay salida. Si *$ foo* es falso, entonces *! $ foo* se evalúa como cierta y **que no** se emitirá.Tenga cuidado de no confundir esto con la *calma referencia! $ foo* , que es algo completamente diferente.

Hay versiones de texto de todos los operadores lógicos, como *eq* , *ne* , *y* , *o* , *no* , *gt* , *ge* , *lt* , y *le* .

Una nota más útil. Si desea incluir el texto inmediatamente después de un *# else* Directiva tendrá que usar llaves que rodean la directiva para diferenciarlo del texto siguiente. (Cualquier directiva puede ser delimitado por llaves, aunque esto es más útil para *# else* ).

# If ($ foo == $ bar), es verdad! # {Else} no lo es! # End </ li>

Loops

**Foreach Loop**

El *# foreach* elemento permite looping. Por ejemplo:

Pasado

# Foreach ($ producto $ todosProductos)

<li> $ product </ li>

# End

</ Ul>

Este *# foreach* loop hace que el *todosProductos $* list (el objeto) para ser puesto en loop para todos los productos (objetivos) en la lista. Cada iteración del bucle, el valor de *$ todosProductos* se coloca en el *producto de $* variable.

El contenido de la *$ todosProductos* variable es un vector, un Hashtable o una matriz. El valor asignado al *producto $* variable es un objeto de Java y puede hacer referencia a partir de una variable como tal. Por ejemplo, si *$ product* era realmente una clase de producto en Java, su nombre podría ser recuperada haciendo referencia a la *$ product.Name* método (por ejemplo: *$ Product.getName ()* ).

Digamos que *$ todosProductos* es una Hashtable. Si quieres recuperar los valores fundamentales para la Hashtable, así como los objetos dentro de la Hashtable, puede utilizar código como el siguiente:

Pasado

# Foreach ($ key en $ allProducts.keySet ())

<li> Leyenda: $ key -> Valor: $ allProducts.get ($ key) </ li>

# End

</ Ul>

Velocity proporciona una manera fácil de conseguir el contador del bucle de modo que usted puede hacer algo como lo siguiente:

<table>

# Foreach ($ cliente en $ customerList)

$ <tr> velocidad cuenta </ ​​td> <td> $ customer.Name </ td> </ tr>

# End

</ Table>

El nombre predeterminado del contador de bucle variable de referencia, que se especifica en el archivo velocity.properties, es de $ velocityCount. Por defecto, el contador comienza en 1, pero esto puede ser ajustado a 0 o 1 en el velocity.properties archivo. Esto es lo que la sección de propiedades contador del bucle del velocity.properties parece archivo:

# Nombre predeterminado del contador del bucle

# Referencia variable.

directive.foreach.counter.name = velocityCount

# Valor predeterminado de inicio del bucle

# Contador de referencia variable.

directive.foreach.counter.initial.value = 1

Es posible establecer un número máximo de veces que un circuito puede ser ejecutado. Por defecto, no hay máximo (indicado por un valor de 0 o menos), pero esto puede estar configurado para un número arbitrario en el velocity.properties archivo. Esto es útil como prueba de fallos.

# El número máximo permitido de ciclos.

directive.foreach.maxloops = -1

Incluir

El *# include* elemento script permite al diseñador de plantillas importar un archivo local, que a su vez se inserta en la ubicación en la que *# include* se define Directiva. El contenido del archivo no se procesan a través del motor de plantillas. Por razones de seguridad, el archivo que se incluye sólo puede estar bajo TEMPLATE\_ROOT.

# Include ("uno.txt")

El archivo en el que el *# include* Directiva se refiere está encerrada entre comillas. Si se incluye más de un archivo, deben estar separados por comas.

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

El archivo que se está incluido no necesita ser referenciado por su nombre, de hecho, a menudo es preferible utilizar una variable en lugar de un nombre de archivo. Esto podría ser útil para orientar la producción de acuerdo a criterios determinados, cuando se presente la solicitud de la página. He aquí un ejemplo que muestra tanto un nombre de archivo y una variable.

# Include ("greetings.txt", $ seasonalstock)

Analizar

El *# parse* elemento script permite al diseñador de plantillas importar un archivo local que contiene VTL. Velocity analizará el VTL y hacer que la plantilla especificada.

# Parse ("me.vm")

Al igual que el *# include* Directiva, *# parse* puede tomar una variable en lugar de una plantilla. Cualquier plantilla a la que *# parse* refiere deben incluirse en TEMPLATE\_ROOT. A diferencia de la *# include* Directiva, *# parse* sólo tomará un solo argumento.

Plantillas de VTL pueden tener *# analizar* los datos referentes a las plantillas que a su vez han *# Analizar* declaraciones. Por ajustado a 10 por defecto, el*directive.parse.max.depth* línea del velocity.properties permite a los usuarios personalizar el número máximo de *# parse* referencias que pueden producirse a partir de una única plantilla. (Nota: Si el *directive.parse.max.depth* propiedad está ausente de la velocity.properties archivo, la velocidad se establece este valor predeterminado a 10.) recursividad se permite, por ejemplo, si la plantilla hacerfoo.vm contiene las siguientes líneas:

Cuenta atrás.

# Set ($ count = 8)

# Parse ("parsefoo.vm")

Todo hecho con hacerfoo.vm!

Sería referencia a la plantilla parsefoo.vm , que podría contener la siguiente VTL:

$ Cuenta

# Set ($ cuenta = $ count - 1)

# If ($ count> 0)

# Parse ("parsefoo.vm")

# Else

Todo hecho con parsefoo.vm!

# End

Después de "cuenta atrás". Se muestra, Velocity pasa por parsefoo.vm , a contar a partir del 8. Cuando la cuenta llegue a 0, se mostrará el "Todo hecho con parsefoo.vm!" mensaje. En este punto, la velocidad volverá a hacerfoo.vm y salida del "Todo hecho con hacerfoo.vm!" mensaje.

Deténgase

El *# parada* elemento de secuencia de comandos permite que el diseñador de la plantilla para detener la ejecución del motor de la plantilla y de regreso. Esto es útil para propósitos de depuración.

# Stop

Velocimacros

El *# macro* elemento script permite a los diseñadores de plantillas para definir un segmento repetido de una plantilla VTL. Velocimacros son muy útiles en una amplia gama de escenarios simples y complejos. Este Velocimacro, creado con el único propósito de salvar a las pulsaciones de teclado y minimizar los errores tipográficos, ofrece una introducción al concepto de Velocimacros.

# Macro (d)

<tr> <td> </ td> </ tr>

# End

El Velocimacro está definido en este ejemplo es *d* , y que puede ser llamado de una manera análoga a cualquier otra directiva VTL:

# D ()

Cuando esta plantilla se llama, la velocidad reemplazaría *# d ()* con una fila que contiene una única, celda de datos vacía.

A Velocimacro podría tomar cualquier número de argumentos - incluso argumentos cero, como se demostró en el primer ejemplo, es una opción - pero cuando se invoca el Velocimacro, debe ser llamado con el mismo número de argumentos con los que se ha definido. Muchos Velocimacros más complicado que el que se ha definido anteriormente. Aquí está una Velocimacro que toma dos argumentos, un color y una gran variedad.

# Macro (filastabla $ color $ algunalista)

# Foreach ($ algo en $ algunalista)

<tr> <td bgcolor=$color> $ algo </ td> </ tr>

# End

# End

El Velocimacro está definido en este ejemplo, *filastabla* , toma dos argumentos. El primer argumento toma el lugar de *$ color* , y el segundo argumento toma el lugar de *$ algunalista* .

Cualquier cosa que se puede poner en una plantilla VTL puede entrar en el cuerpo de un Velocimacro. El *filastabla* Velocimacro es un *foreach* comunicado. Hay dos *# end* declaraciones en la definición del *# filastabla* Velocimacro, el primero pertenece a la *# foreach* , la segunda termina la definición Velocimacro.

# Set ($ GREATLAKES = ["Superior", "Michigan", "Huron", "Erie", "Ontario"])

# Set ($ color = "blue")

<table>

# Filastabla ($ color $ GREATLAKES)

</ Table>

Note que *$ GREATLAKES* toma el lugar de *$ algunalista* . Cuando el *# filastabla* Velocimacro se llama en esta situación, se genera el siguiente resultado:

<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 pueden definirse *en línea* en una plantilla de velocidad, lo que significa que no está disponible a otras plantillas de Velocity en el mismo sitio web.Definición de un Velocimacro de tal manera que puede ser compartida por todas las plantillas tiene ventajas evidentes: se reduce la necesidad de redefinir el Velocimacro en numerosas plantillas, el ahorro de trabajo y reducir la posibilidad de error, y se asegura de que un único cambio en una macro disponible para más de una plantilla.

Fueron los *# filastabla ($ color $ lista)* Velocimacro definido en una biblioteca de plantillas Velocimacros, esta macro se puede utilizar en cualquiera de las plantillas regulares. Podría ser utilizado muchas veces y para muchos propósitos diferentes. En la plantilla vegetal.vm dedicado a todas las cosas hongos, la *# filastabla* Velocimacro podría invocarse para listar las partes de una seta típica:

# Set ($ partes = ["volva", "tronco", "anillo", "agallas", "píleo"])

# Set ($ cellbgcol = "# CC00FF")

<table>

# Filastabla ($ $ cellbgcol partes)

</ Table>

En el cumplimiento de una solicitud de vegetal.vm , Velocity encontraría el *# filastabla* Velocimacro en la biblioteca de plantillas (definida en elvelocity.properties archivo) y generar la salida siguiente:

<table>

<tr> <td bgcolor="#CC00FF"> volva </ td> </ tr>

<tr> <td bgcolor="#CC00FF"> estípite </ td> </ tr>

<tr> <td bgcolor="#CC00FF"> anillo </ td> </ tr>

<tr> <td bgcolor="#CC00FF"> branquias </ td> </ tr>

<tr> <td bgcolor="#CC00FF"> píleo </ td> </ tr>

</ Table>

**Argumentos Velocimacro**

Velocimacros pueden tomar como argumentos de cualquiera de los siguientes elementos de VTL:

* Referencia: cualquier cosa que empieza con '$'
* Cadena literal: algo así como "$ foo" o "hola"
* Número literal: 1, 2, etc
* IntegerRange: [1 0,2] o [$ foo .. $ Bar]
* ObjectArray: ["a", "b", "c"]
* valor booleano verdadero
* valor booleano false

Al pasar referencias como argumentos de Velocimacros, tenga en cuenta que se pasan referencias "por su nombre". Esto significa que su valor es 'generado' en cada uso dentro de la Velocimacro. Esta función le permite pasar referencias con llamadas a métodos y tienen el método llamado de cada uso. Por ejemplo, cuando se llama a la siguiente como se muestra Velocimacro

# Macro (callme $ a)

$ A $ a $ a

# End

# Callme ($ foo.bar ())

resultados en la barra de método () de la referencia $ foo ser llamado 3 veces.

A primera vista, esta característica parece sorprendente, pero cuando se tiene en cuenta la motivación original de Velocimacros - para eliminar la duplicación cut'n'paste de uso común VTL - tiene sentido. Te permite hacer cosas como objetos con estado pase, como un objeto que genera colores en una secuencia repetitiva para colorear filas de la tabla, en el Velocimacro.

Si usted necesita para eludir esta función, siempre puede obtener el valor del método como una nueva referencia y pasar a lo siguiente:

# Set ($ mivalor = $ foo.bar ())

# Callme ($ mivalor)

**Velocimacro Propiedades**

Varias líneas de la velocity.properties archivo permiten la aplicación flexible de los Velocimacros. Tenga en cuenta que estos también están documentadas en la [Guía del Desarrollador](http://velocity.apache.org/engine/releases/velocity-1.5/developer-guide.html) .

velocimacro.library - Una lista separada por comas de todas las bibliotecas de plantillas Velocimacro. Por defecto, la velocidad se ve una sola biblioteca:*VM\_global\_library.vm* . La ruta de plantilla configurada se utiliza para encontrar las librerías Velocimacro.

velocimacro.permissions.allow.inline - Esta propiedad, que cuenta con los posibles valores de verdadero o falso, determina si Velocimacros pueden ser definidos en las plantillas habituales. El valor por defecto, true, permite a los diseñadores de plantillas para definir Velocimacros en las propias plantillas.

velocimacro.permissions.allow.inline.to.replace.global - Con valores posibles de verdadero o falso, esta propiedad permite al usuario especificar si una línea definida Velocimacro en una plantilla puede sustituir a una plantilla definida a nivel mundial, uno que se ha cargado en el arranque a través de lavelocimacro.library propiedad. El valor predeterminado, falsa , impide Velocimacros en línea se define en una plantilla de la sustitución de los definidos en las bibliotecas de plantillas cargados durante el arranque.

velocimacro.permissions.allow.inline.local.scope - Esta propiedad, con posibles valores de verdadero o falso, por defecto a false, los controles si Velocimacros definidos inline son 'visibles' sólo en la plantilla de definición. En otras palabras, con esta propiedad establecida en true, una plantilla puede definir máquinas virtuales en línea que se pueden utilizar solamente por la definición de la plantilla. Usted puede utilizar este para los trucos de VM de lujo - si una máquina virtual mundial llama a otro VM global, con un alcance en línea, una plantilla puede definir una implementación privada del segundo VM que será llamada por el primer VM cuando se invoca por esa plantilla. Todas las otras plantillas no se ven afectados.

velocimacro.context.localscope - Esta propiedad cuenta los posibles valores true o ​​false, y el predeterminado es falso. Cuando es true, cualquier modificación en el contexto a través de # set () dentro de una Velocimacro se consideran "local" a la Velocimacro, y no afectarán permanentemente el contexto.

velocimacro.library.autoreload - Esta propiedad controla Velocimacro biblioteca de carga automática. El valor predeterminado es false . Cuando se establece en verdadera la fuente de la biblioteca Velocimacro una Velocimacro invocado será verificada por los cambios y vuelve a cargar si fuera necesario.Esto le permite cambiar y probar bibliotecas Velocimacro sin tener que reiniciar la aplicación o el contenedor de servlets, como usted puede con plantillas regulares. Este modo sólo funciona cuando el almacenamiento en caché está *fuera* de los cargadores de los recursos (por ejemplofile.resource.loader.cache = false ). Esta característica está diseñada para el desarrollo, no para la producción.

**Velocimacro Trivia**

Actualmente, Velocimacros deben ser definidos antes de que se utilizan por primera vez en una plantilla. Esto significa que sus declaraciones de # macro () deben venir antes de utilizar los Velocimacros.

Esto es importante de recordar si intenta hacerle # parse () de una plantilla que contiene directivas # macro () directivas. Debido a que el # parse () ocurre en tiempo de ejecución, y el parser decide si un elemento VM de futuro en la plantilla es una máquina virtual en parsetime, # parse ()-ción de un conjunto de declaraciones de VM no funcionará como se espera. Para evitar esto, utilice el velocimacro.library facilidad de tener Velocity cargar sus máquinas virtuales en el inicio.

Escapar directivas VTL

Directivas de VTL se pueden escapar con la barra invertida ("\") de una manera similar a las referencias VTL válidas.

# # # Include ("a.txt") se representa como <contents de a.txt>

# Include ("a.txt")

# # \ # Include ("a.txt") se representa como # include ("a.txt")

\ # Include ("a.txt")

# # \ \ # Include ("a.txt") se representa como \ <contents de a.txt>

\ \ # Include ("a.txt")

Un cuidado especial se debe tomar cuando se escapan directivas de VTL que contienen varios elementos de script en una misma directiva (por ejemplo, en una declaraciones if-else-end). Aquí está una VTL típica sentencia if:

# If ($ jazz)

Vyacheslav Ganelin

# End

Si *$ jazz* es verdadero, la salida es

Vyacheslav Ganelin

Si *$ jazz* es falso, no hay salida. Escapar elementos script altera la salida. Consideremos el siguiente caso:

\ # If ($ jazz)

Vyacheslav Ganelin

\ # End

Si *$ jazz* es verdadero o falso, la salida será

# If ($ jazz)

Vyacheslav Ganelin

# End

De hecho, ya que todos los elementos del guión se escaparon, *$ jazz* nunca es evaluado por su valor booleano. Supongamos barras invertidas preceden elementos de script que se escaparon legítimamente:

\ \ # If ($ jazz)

Vyacheslav Ganelin

\ \ # End

En este caso, si *$ jazz* es verdadero, la salida es

\ Vyacheslav Ganelin

\

Para entender esto, tenga en cuenta que el # if (arg) al terminar un (retorno) de nueva línea se omita la nueva línea de la salida. Por lo tanto, el cuerpo de la# si () bloque sigue a la primera '\', representa a partir de la '\ \' anterior a la # si () . El último \ está en una línea diferente a la del texto, porque hay una nueva línea después de "Ganelin ', por lo que la final \ \, que precede a la # end es parte del cuerpo de la manzana.

Si *$ jazz* es falso, la salida es

\

Tenga en cuenta que las cosas empiezan a romperse si los elementos de script no son escapados correctamente.

\ \ \ # If ($ jazz)

Vyacheslav Ganeli

\ \ # End

Aquí el *# si* se escapó, pero hay un *# end* restante, tener demasiadas terminaciones causarán un error de análisis.

VTL: Asuntos de formato

Aunque VTL en esta guía del usuario es a menudo aparece con saltos de línea y espacios en blanco, la VTL se muestra a continuación

# Set ($ imperial = ["Munetaka", "Koreyasu", "Hisakira", "Morikune"])

# Foreach ($ shogun en $ imperial)

$ Shogun

# End

es igualmente válida como el siguiente código que Geir Magnusson Jr. envió a la lista de correo de usuarios de Velocity para ilustrar un punto completamente ajenos:

Envíe # set ($ foo = ["$ 10 y", "a pie"]) # foreach ($ a en $ foo) $ a # end favor.

El comportamiento de la velocidad es engullir el exceso de espacio en blanco. La directiva anterior se puede escribir como:

Envíenme

# Set ($ foo = ["$ 10 y", "a pie"])

# Foreach ($ a en $ foo)

$ A

# End

por favor.

o como

Envíenme

# Set ($ foo = ["$ 10 y", "a pie"])

# Foreach ($ a en $ foo) $ a

# End por favor.

En cada caso, la salida será la misma.

Otras Características y Miscelánea

**Matemáticas**

Velocity tiene un puñado de funciones incorporadas matemáticos que se pueden utilizar en las plantillas con el *conjunto* Directiva. Las ecuaciones siguientes son ejemplos de suma, resta, multiplicación y división, respectivamente:

# Set ($ foo = $ bar + 3)

# Set ($ foo = $ bar - 4)

# Set ($ foo = $ bar \* 6)

# Set ($ foo = $ bar / 2)

Cuando una operación de división se lleva a cabo entre dos números enteros, el resultado será un número entero. Cualquier resto puede ser obtenida usando el módulo ( *%* operador).

# Set ($ foo = $ bar 5%)

**Operador Gama**

El operador de rango se puede utilizar en combinación con *# establecidos* y *# foreach* declaraciones. Útil para su capacidad de producir una matriz de objetos que contiene enteros, el operador de rango tiene la siguiente construcción:

[N .. m]

Tanto *n* y *m* debe ser o bien producir o números enteros. Si *m* es mayor que o menor que *n* no importa, en este caso el rango será simplemente contar hacia abajo. Ejemplos que muestran el uso del operador de rango que se indica a continuación:

Primer ejemplo:

# Foreach ($ foo en [1 .. 5])

$ Foo

# End

Segundo ejemplo:

# Foreach ($ bar en [2 .. -2])

$ Bar

# End

Tercer ejemplo:

# Set ($ arr = [0 .. 1])

# Foreach ($ i en $ arr)

$ I

# End

Cuarto ejemplo:

[1 .. 3]

Produce la siguiente salida:

Primer ejemplo:

1 2 3 4 5

Segundo ejemplo:

2 1 0 -1 -2

Tercer ejemplo:

0 1

Cuarto ejemplo:

[1 .. 3]

Tenga en cuenta que el operador no coincidir con la matriz produce cuando se utiliza junto con *# establecidos* y *# foreach* directivas, como se demuestra en el cuarto ejemplo.

Diseñadores de páginas web que ver con lo que las tablas de tamaño estándar, pero en algunos no se tienen suficientes datos para llenar la tabla, se encuentra el operador de rango particularmente útil.

**Temas avanzados: Escapar y!**

Cuando una referencia es silenciado con el *!* carácter y la *!* carácter precedido por un *\* carácter de escape, la referencia se maneja de una manera especial.Tenga en cuenta las diferencias entre regular de escape, y el caso especial en el que *\* precede *!* sigue:

# Set ($ foo = "bar")

$ \! Foo

$ \ {Foo}

$ \ \! Foo

$ \ \ \! Foo

Esto se representa como:

! $ Foo

$ {Foo}

$ \! Foo

$ \ \! Foo

Esto contrasta con escapar regular, donde *\* precede *$* :

\ $ Foo

\ $! Foo

\ $ {Foo}

\ \ $ {Foo}

Esto se representa como:

$ Foo

! $ Foo

$ {Foo}

\ Bar

**Velocimacro Miscelánea**

Esta sección es un mini-FAQ sobre temas relacionados con Velocimacros. En esta sección se va a cambiar con el tiempo, por lo que es digno de la comprobación para la nueva información de vez en cuando.

Nota: En esta sección, 'Velocimacro' será comúnmente abreviado como "VM".

**¿Puedo utilizar una directiva u otro VM como argumento para una máquina virtual?**

Ejemplo: centro # (# negrita ("hola"))

No. Una directiva no es un argumento válido para una directiva y, a efectos prácticos, un VM es una directiva.

*Sin embargo ...* , hay cosas que usted puede hacer. Una solución fácil es tomar ventaja del hecho de que "Comillas dobles" (") hace que su contenido. Así que se podría hacer algo como

# Set ($ stuff = "# negrita ('hola')")

Centro # ($ stuff)

Usted puede ahorrar un paso ...

Centro # ("# negrita ('hola')")

Tenga en cuenta que en este último ejemplo se evalúa la arg *dentro* de la máquina virtual, no en el nivel que llama. En otras palabras, el argumento de la VM se pasa en su totalidad y evaluado dentro de la máquina virtual se pasa a. Esto le permite hacer cosas como:

# Macro (interior $ foo)

interior: $ foo

# End

# Macro (exterior $ foo)

# Set ($ bar = "outerlala")

externa: $ foo

# End

# Set ($ bar = 'calltimelala')

# Externo ("# interno ($ bar)")

Cuando la salida es

Exterior: interior: outerlala

debido a que la evaluación de la "# interno ($ bar)" ocurre dentro de # externo (), por lo que el valor de $ bar establecido dentro de # externo () es la que se utiliza.

Esta es una característica intencional y celosamente vigilado - args se pasan 'por su nombre' en las máquinas virtuales, por lo que puede entregar VM cosas como referencias con estado como

# Macro (foo $ color)

<tr bgcolor=$color> <td> Hola </ td> </ tr>

<tr bgcolor=$color> <td> There </ td> </ tr>

# End

# Foo ($ bar.rowColor ())

Y han rowColor () llamado en repetidas ocasiones, en lugar de una sola vez. Para evitar que, invoque el método fuera de la VM, y pasar el valor al VM.

# Set ($ color = $ bar.rowColor ())

# Foo ($ color)

**¿Puedo registrar Velocimacros vía # parse ()?**

Actualmente, Velocimacros deben ser definidos antes de que se utilizan por primera vez en una plantilla. Esto significa que sus declaraciones de # macro () deben venir antes de utilizar los Velocimacros.

Esto es importante de recordar si intenta hacerle # parse () de una plantilla que contiene directivas # macro () directivas. Debido a que el # parse () ocurre en tiempo de ejecución, y el parser decide si un elemento VM de futuro en la plantilla es una máquina virtual en parsetime, # parse ()-ción de un conjunto de declaraciones de VM no funcionará como se espera. Para evitar esto, utilice el velocimacro.library facilidad de tener Velocity cargar sus máquinas virtuales en el inicio.

**¿Qué es Velocimacro Autoreloading?**

Hay una propiedad, destinado a ser utilizado en *el desarrollo* , no la producción:

velocimacro.library.autoreload

que por defecto es false. Cuando se establece en true *, junto con*

<tipo>. resource.loader.cache = false

(Donde <tipo> es el nombre del gestor de recursos que está utilizando, como 'archivo') y luego el motor de Velocity recargará automáticamente los cambios en los archivos de la biblioteca Velocimacro cuando se hacen, por lo que no tiene que volcar el servlet motor (o aplicación) o hacer otros trucos para tener sus Velocimacros recargados.

Esto es lo que un simple conjunto de propiedades de configuración se vería así.

file.resource.loader.path = plantillas

file.resource.loader.cache = false

velocimacro.library.autoreload = true

No tener esto en la producción.

**Concatenación de cadenas**

Una pregunta común que los desarrolladores se hacen es *¿Cómo lo hago concatenación de cadenas? ¿Hay algún análogo al operador '+' en Java?* .

Para concatenación de referencias en VTL, usted apenas tiene que 'ponerlos juntos'. El contexto en el que desea ponerlos juntos no importa, así que vamos a ilustrar con algunos ejemplos.

En el 'schmoo' regular de una plantilla (al mezclar adentro con contenido regular):

# Set ($ size = "Big")

# Set ($ nombre = "Ben")

El reloj es de $ size $ name.

y la salida va a hacer como "El reloj es BigBen. Para los casos más interesantes, como por ejemplo cuando se desea concatenar cadenas para pasar a un método, o para establecer una nueva referencia, acaba de hacer

# Set ($ size = "Big")

# Set ($ nombre = "Ben")

# Set ($ reloj = "$ size $ name")

El reloj es de $ reloj.

Lo que resultará en la misma salida. Como último ejemplo, cuando se quiere mezclar cadenas 'estáticas' con sus referencias, es posible que tenga que utilizar "referencias formales":

# Set ($ size = "Big")

# Set ($ nombre = "Ben")

# Set ($ reloj = "$ {tamaño} Tall $ name")

El reloj es de $ reloj.

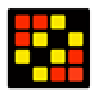
Ahora la salida es "El reloj es BigTallBen '. Se necesita la notación formal para que el parser sabe que significas para utilizar la referencia '$ size' versus '$ sizeTall "que lo haría si el' {} 'no estaban allí.

Feedback

Si encuentra algún error en este manual o tiene comentarios relacionados con la Guía del usuario de Velocity, por favor escriba a la [lista de desarrolladores de velocidad](mailto:dev@velocity.apache.org) . ¡Gracias!

# Velocimacros disponibles en todas las plantillas .vm

* [Programación](http://www.liferaytips.com/taxonomy/term/2)
* [Velocity](http://www.liferaytips.com/taxonomy/term/3)
* [Macros](http://www.liferaytips.com/taxonomy/term/4)

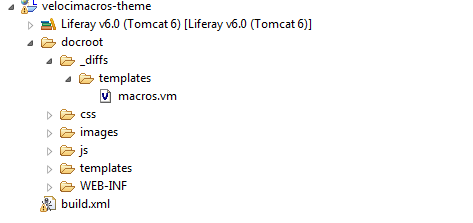


Enviado por **alukard13** en Vie, 03/30/2012 - 12:11

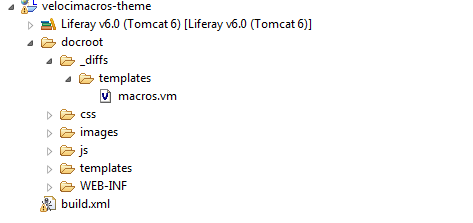
Una velocimacro es un elemento muy útil para no tener que repetir enemil veces un código dentro de una plantilla vm. Para el que todavía no sepa lo que es le recomiendo que se documente  aquí  ([FAQ de Velocity de apache](http://velocity.apache.org/engine/releases/velocity-1.5/user-guide.html#velocimacros)).

En esta pequeña entrada explicaré cómo incluir macros de velocity en un tema de Liferay para que estén disponibles en todas las plantillas vm  del ámbito journal article.

Para ello, lo primero que haremos  es crear (si no la teníamos ya creada)  una carpeta templates dentro de tema tal y cómo se ve en la imagen.



Dentro de ella crearemos el fichero vm que contendrá nuestras macros que para nuestro ejemplo se llamará macros.vm



Abrimos el fichero macros.vm y creamos una macro sencilla que nos permitirá recuperar videos de youtube pasándole ciertos parámetros.

#macro (getVideoYoutube $heightVideo $widthVideo $idVideo)

<iframe class="youtube-player" type="text/html" width="$widthVideo" height="$heightVideo" src="<http://www.youtube.com/embed/>$idVideo?wmode=opaque" frameborder="0">

</iframe>

#end

Ahora sólo tenemos que guardar nuestro fichero y referenciarlo para que se cargue con el tema. Para añadir la referencia a este fichero hay que hacerlo en otro de los ficheros clásicos de los temas de Liferay, el fichero portal\_normal.vm. Este fichero también debería encontrarse en nuestra carpeta \_diffs -> templates (si es que lo hemos modificado). Abrimos el fichero portal\_normal.vm e incluimos esta línea:

#parse ("$full\_templates\_path/macros.vm")

Recordad incluirlo después de #parse ($init).

Una vez hecho esto tendréis disponible esta macro en cualquier plantilla de velocity del portal y para usarla sólo es necesario llamarla de la siguiente manera:

# getVideoYoutube (480 640 bLM6Abn5H6E)