Velocity User Guide

저자: Apache Jakarta Project
번역: 박현준(blackmire@empal.com)
검토: 안영회
배포: Bizcom Java Article Center

이 글은 다음과 같은 독자님들에게 권장합니다.
[Intermediate level java developer]
기본적인 자바 프로그램 작성이 가능하며, 고품질의 자바 프로그램 제작을 원하는 분

Bizcom Java Article Center는 지식공유를 위한 비영리 조직입니다.
문의사항은 아래로 연락 바랍니다.
1기 대표 – 안영회(ahnyounghoe@empal.com)

이 글은 Apache Jakarta Project의 Velocity User Guide를 우리나라 개발자들이 쉽게 접할 수 있도록 번역한 내용입니다. 원문은 아래의 URL에서 보실 수 있습니다.

http://jakarta.apache.org/velocity/user-guide.html

Velocity란?

Velocity는 자바기반 Template 엔진이다.
Velocity를 사용함으로써 웹 페이지 디자이너가 자바코드로 작성된 method를 참조할 수 있다.
또한 MVC모델을 적용시킨 웹사이트를 개발하기 위해 웹 디자이너와 자바 프로그래머가 병렬적으로 작업할 수 있다.
Velocity는 자바코드를 웹 페이지로부터 분리시킴으로써 유지보수가 용이하게 하며 JSP나 PHP의 실용적인 대안을 제시한다.
Velocity는 template으로부터 웹 페이지뿐만 아니라 SQL, PostScript 또는 다른 결과물을 생성하는데 사용될 수 있다. 또한 소스코드나 보고서 생성을 위한 단독 유틸리티, 또는 다른 시스템에 통합된 컴포넌트로서 사용될 수 있다. MVC 모델을 적용하여 웹 어플리케이션을 개발할 때, Velocity는 Turbine 웹 어플리케이션 프레임워크를 위한 template 서비스를 제공할 것이다.

Velocity Template Language(VTL)

VTL은 웹 페이지에 동적 컨텐츠를 넣기 위한 가장 쉽고, 간단하고, 명확한 방법을 제공하기 위한 수단이다. 프로그래밍 경험이 거의 없는 웹 페이지 개발자도 웹 사이트에 동적인 컨텐츠를 넣기 위해 VTL을 사용할 수 있다.

VTL은 웹 사이트에 동적 컨텐츠를 삽입하기 위해 reference를 사용하며 변수는 reference 타입 중 하나가 된다.

reference중 하나의 타입을 갖는 변수는 자바코드로 정의된 것을 참조할 수 있으며 variable로부터 VTL 문장을 통해 값을 얻을 수 있다.

```
VTL 문장 예
    #set ( $a = “ Velocity ” )
```

모든 VTL문장처럼 위 VTL문장은 #문자로 시작하며 set과 같은 지시자를 포함하고 있다. 온라인 방분자가 웹 페이지를 요청하면, Velocity Templating Engine은 모든 # 문자를 찾기위해 요청된 웹페이지 전체를 검색하고 # 문자가 VTL문장과 관련 있는지 없는지를 결정하게 된다.

문자 다음에 set과 같은 지시자가 오게 된다. 지시자는 ()를 사용하며 () 안에는 변수에 값을 할당하는 식이 들어가게 된다. () 안에서 변수는 등식의 왼쪽에 오게 되며 할당될 값은 오른쪽에 오게 된다. 위 예제에서 변수는 $a이며, 값은 ‘ Velocity ’ 이다.

VTL에서 변수는 항상 $로 시작하며 변수에 할당되는 값은 “ 로 열고 닫아줘야 한다. Velocity에서 변수에 할당되는 데이터의 타입에 대한 혼동이 없다. 왜냐하면 오직 string(텍스트 기반 정보)만 변수에 할

당 되기 때문이다.
#으로 시작하는 지시자는 어떠한 행위를 나태내기 위해 사용되며, $로 시작하는 reference 즉, variable은 어떤 값을 얻기 위해 사용된다.

Comment

VTL에서 comment의 사용법은 다음과 같다.

>> 단일 문장에 대한 comment : ##을 이용한다.
ex) ## This is a single line comment.

>> 여러 문장에 대한 comment : #*으로 시작하여 *#로 끝난다.
ex) #*
Thus begins a multi-line comment. Online visitors won't
see this text because the Velocity Templating Engine will ignore it.
*#

>> 페이지 작성자나 버전과 관련된 comment : #** 으로 시작하여 *# 으로 끝난다.

Reference

VTL에서 사용하는 Reference는 세가지 타입이 있다. 각각의 타입은 다음과 같다.

- variables
- properties
- methods

Variables

variable은 $으로 시작하며 $다음은 VTL에서 정의한 identifier로 구성된다. VTL identifier는 반드시 영어 소문자 또는 대문자로 시작되며 그 다음에는 아래에 정의된 나머지 identifier가 올 수 있다.

VTL에서 정의한 identifier는 다음과 같다.
- 알파벳 (a…z,A….Z)
- 숫자 (0….9)
- 하이픈 (“ - “)
- 밑줄 (“ _ ”)

Properties

properties의 표기법은 다음과 같다.

```
$+VTL identifier+.+VTL identifier
    예) $customer.Address
```

위 예제는 두 가지 의미를 가지고 있을 수 있다. 첫 번째는 customer라는 hashtable에서 Address라는 키를 가지고 있는 값을 리턴하는 경우다. 두 번째는 customer객체의 getAddress()라는 method를 호출 하는 경우다. 다음에 설명할 methods reference 에서 언급하겠지만 method를 호출할 경우 위 예제처럼 단축된 표기법을 사용하기도 한다. 각각에 대한 처리는 페이지 요청시 Velocity에 의해 판단된다.

Methods

methods는 자바코드로 정의한 것이다. VTL에서 methods 표기법은 다음과 같다.

```
$+VTL Identifier+VTL Method Body
```

VTL Method Body 는 “(“로 시작하여 다음에 VTL Identifier로 구성되며 “)”로 끝나게 된다. 추가적으로 “(“와 “)”사이에 parameter list가 올 수 있다.

다음은 VTL에서 method reference 예이다.

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

처음 두줄의 예제는 Properties 부분에서 말했듯이 $customer.getAddress, $purchase.getTotal과 같은 의미를 가진다. VTL에서는 이처럼 VTL Methods를 표기하기 위해 Properties 표기법을 사용할 수 있는데 Properties로 표기가 가능하다면 이를 더 선호한다. Properties와 Methods의 차이점은 parameter list의 명시여부에 있다.

Formal Reference Notation

위에서 설명한 예제들은 모두 단축표기법(shorthand notation)으로 references를 나타내었다. 하지만 다음과 같이 formal notation을 사용할 수 있다.

```
${customer.Address}
```

보통 reference를 표기하기 위해 단축표기법(shorthand notation)을 사용하지만 정확한 처리를 위해 formal notation를 사용해야 하는 경우도 있다. 다음 예제를 보자.

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

약간 혼동스러울 수 있지만 위 예제에서 reference로 사용된 것은 vice이다. 하지만 Velocity는 $vice로 처리하지 않고 $vicemaniac로 처리할 것이다. 이러한 문제를 해결하기 위해서는 formal notation를 써야 한다. 위 예제를 formal notation으로 처리한 것이 다음 예제이다.

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

formal notation을 사용함으로써 어떤 것이 reference로 사용된 것인지 정확하게 구분할 수가 있다.

Quite Reference Notation

Velocity는 정의되지 않은 reference는 문자로 인식하고 그 형태를 그대로 출력하게 된다. 다음 예제는 VTL template의 일부분이다.

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

보통 처음 form이 로딩될 때 variable reference인 $email 은 값을 가지고 있지 않다. 일반적으로 값이 없는 경우 공백으로 출력되기를 원하지만 Velocity는 $email을 그대로 출력하게 된다. 이러한 문제를 해결하기 위해 quite notation을 사용한다. 위 예제를 quite notation으로 표기한 결과는 다음과 같다.

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

quite notation을 사용함으로써 Velocity는 정의되지 않은 reference 값을 공백으로 처리하게 된다.

quite notation은 formal notation과 같이 사용할 수 있다.

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

Getting literal

VTL에서 #과 $과 같은 특별한 문자를 사용한다. 그래서 자신의 template에 이와 같은 문자를 사용하고자 할 때 약간의 주의가 필요하다. 여기서는 자신의 template에서 $ 문자를 표현하는 법에 대해 다룬다.

Currency

보통 달러를 표기할 때 $문자를 사용한다.

```
$2.50
```

VTL에서는 identifiers는 반드시 대/소문자로 시작되어야 되기 때문에 통화 표기시 $를 사용하는데 아무런 문제가 없다.

Escaping Valid VTL Reference

template에서 VTL special 문자를 핸들링 하기 위해 ‘ \ ’를 사용한다.

```
#set { $email = “ foo ” }
    $email
```

위 예제는 email이라는 variable reference에 foo라는 값을 넣고 출력하는 예제이다.
Velocity가 template에서 $email이라는 reference를 만날 경우 Context에서 일치하는 값을 찾게 된다. 위 예에서는 foo가 그 값이 되며 따라서 foo가 출력된다.

만일 $email이 정의가 되어 있지 않다면 출력 결과는 $email이 될 것이다.(Quite Reference Notation 부분 참고)
$email라는 reference가 정의 되어 있으며, $email이라는 문자를 출력하고자 할 경우 여러 방법이 있지만 가장 간단한 방법은 ‘ \ ’를 사용하는 것이다.

```
## The following line defines $email in this template:
#set( $email = "foo" )
    $email --------------- (1)
    \$email--------------- (2)
    \\$email-------------- (3)
    \\\$email------------  (4)
```

위 예제의 출력결과는 다음과 같다.

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

\ 문자는 왼쪽부터 문자를 대입시킨다. (1)은 바로 $email의 값인 foo를 출력하게 되며 (2)에서 \문자에 $가 대입되어 Velocity가 reference인 $email로 인식하지 않기 때문에 $email이 그대로 출력되게 된다. (3)은 \문자에 두 번째 \가 대입되기 때문에 \foo가 출력되며 (4)에서는 첫 번째 \문자에 두 번째 \가 대입되고 세 번째 \문자에 $가 대입되기 때문에 \$email이 출력된다.

$email이 정의되지 않은 경우를 살펴 보자.

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

위 예에서 reference $email은 정의되지 않은 경우이다. Velocity는 정의되지 않은 reference는 문자로 취급하기 때문에 결과값은 다음과 같다.

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

Velocity가 정의되지 않은 reference를 어떻게 다루는지 다음 예를 통해 좀더 살펴보자.

```
#set { $foo = “ gibbous ” }
$moon = $foo
```

위 예에서 $foo는 정의된 reference이며 $moon은 정의되지 않은 reference이다. 위 예제의 출력결과는 ‘ $moon = gibbous ’ 가 된다. 이 예제를 통해 정의되지 않은 reference는 Velocity가 문자 그대로 처리한다는 것을 알 수 있다.

Case Substitution

VTL은 reference를 사용하는데 있어 여러 방식을 제공한다. 따라서 template에 reference를 사용할 때 효과적인 방식을 선택할 수 있다.

```
$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}
```

위의 예제는 같은 reference를 사용하는 여러 방법을 보여주고 있다. Velocity는 Sun에서 정의한 Bean Specification을 기반으로 설계되었기 때문에 대소문자를 구분한다. 하지만 Java Bean과는 조금 다르다. 예를 들어 getFoo()라는 method가 template에서 $bar.foo에 의해 참조될 경우 비록 getFoo()라는 method를 참조하더라도 Velocity는 먼저 getfoo()를 찾게 된다. getfoo()를 찾지 못할 경우 Velocity는 getFoo()를 찾게 된다. $bar.Foo로 참조 되었을 경우도 마찬가지로 getFoo()를 먼저 찾은다음 없을 경우 getfoo()를 찾게 된다.

참고) template에서 객체의 instance 변수는 참조 할 수 없다. 예를 들어 $foo.Name이라는 property reference는 Foo라는 객체의 getName()이라는 method를 참조하는 것이지 Foo 객체의 Name이라는 instance 변수를 참조하는 것이 아니다.

Directive

#set

#set 지시자는 reference에 값을 setting하기 위해 사용된다. 주로 variable reference나 property reference에 값이 할당된다..

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

“()” 안에서 왼쪽에는 variable reference나 property reference가, 오른쪽에는 다음과 같은 타입 중 하나가 올 수 있다.

- Variable reference
- String literal
- Property reference
- Method reference
- Number literal
- ArrayList

위에서 열거한 타입을 적용시킨 예는 다음과 같다.

```
#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
```

위 예제 마지막에 ArrayList타입을 적용시킨 부분에서 '[]' 로 둘러싼 값들은 ArrayList에 정의되어 있는 method를 통해 접근할 수 있다. 예를 들어 첫 번째 값인 'Not' 을 얻기 위해서는 $monkey.Say.get(0) 과 같이 ArrayList에 정의되어 있는 get() method를 사용하면 된다.

오른쪽 부분에는 아래와 같이 수식이 올 수도 있다.

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

오른쪽에 온 property나 method reference의 값이 null일 경우 그때는 왼쪽에 선언된 변수에 null 값이 할당 되지 않는다. 따라서, null 할당을 통해 context에 생성된 reference를 제거하는 것은 불가능하다.

```
#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
```

$query.criteria("name") 가 bill이라는 스트링 값을 리턴하고 $query.criteria("address") 가 null을 리턴할 경우 위 예제의 결과는 다음과 같다.

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

다른 지시자와는 달리 #set 지시자는 #end 지시자를 갖지 않는다.

String Literals

#set 지시자를 사용할 때, string 문자는 " 로 감싸줘야 한다. 물론 일반적인 string 문자를 사용할 경우 ' 로 감싸줘도 상관 없지만 variable이나 property reference에 reference의 값을 할당하고자 할 때 ' 로 감쌀 경우 Velocity가 처리하지 못한다. 다음 예제를 보자.

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

위 예제의 결과는 다음과 같다.

```
bar
$foo
```

Velocity에서 처리되지 않은 문자를 보여주기 위해 ' 를 사용하는 것이 초기 설정 값이다. 하지만 velocity.properties 파일 내의 stringliterals.interpolate 값을 false로 지정함으로써 초기 설정을 변경할 수 있다.

Conditionals

If / ElseIf / Else

#if 지시자는 조건에 따라 text를 웹 페이지에 포함시키고자 할 때 사용된다.

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

variable $foo가 true일 경우는 두 가지 상황으로 나누어 볼 수 있다. 첫 번째 상황은 $foo가 true 값을 가진 boolean인 경우이며, 두 번재 상황은 $foo 값이 null이 아닌 경우이다. Velocity context에는 오직 Object만 저장된다. 따라서 우리가 'boolean' 이라고 말할 때 그것은 Boolean 이라는 class를 의미한다. 이것은 method의 리턴값이 boolean인 경우에도 해당된다.

#if와 #end 사이의 내용은 조건이 참인 경우에만 출력된다. 위 예제에서 가령 $foo가 참인 경우 "Velocity!" 출력되며, $foo가 null이거나 false인 경우에는 아무것도 출력되지 않는다.

#elseif나 #else는 #if와 함께 사용한다. Velocity Templating Engine은 처음 조건이 참인 경우 #if~#end 실행을 중단한다.

Velocity에서 숫자 비교는 Integer객체에 제한된다. 그 밖에 다른 경우는 false가 리턴된다. 유일한 예외는 ‘ ==(동등비교) ’ 이다. 동등비교에서 양쪽 모두 같은 class가 되어야 한다.

Relational and Logical Operators (관계 및 논리 연산자)

Velocity는 variable간 관계를 결정하기 위해 동등연산자를 사용한다. 다음은 동등 연산자를 사용한 예제이다.

```
#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
```

Velocity에서 AND,OR,NOT 과 같은 논리 연산자도 사용할 수 있다.

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

자바에서처럼 위 예제에서는 $foo와 $bar가 모두 true 이어야만 #if() 문이 true가 된다. 만일 $foo가 false이면 $bar를 검증하지 않고 바로 fase를 리턴한다.

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

위 예제는 OR 연산자의 예이다. AND와는 달리 $foo나 $bar중 하나만 true면 #if()는 true를 리턴하게 된다. 만일 $foo가 true 라면 $bar를 검증하지 않고 바로 true를 리턴하게 된다. 반대로 $foo가 false를 리턴한다면 AND와는 달리 $bar를 검증한 후 true/false 리턴 여부를 결정하게 된다.

```
##logical NOT
#if( !$foo )
```

```
  <strong>NOT that</strong>
#end
```

위 예제는 NOT 연산자의 예이다. $foo 가 true면 !$foo는 false를 리턴하게 된다. 반대로 $foo가 false면 !$foo는 true를 리턴하게 된다.

Loop

Foreach Loop

#foreach는 반복적인 수행이 필요할 때 사용한다.

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

위 예제는 $allProducts에 있는 모든 값들을 출력하기 위한 것이다. $allProducts에 있는 값들은 반복될 때마다 $product 변수에 저장되게 된다.

$allProducts 변수 값은 Vector, Hashtable, Arrary가 될 수 있다. $allProducts에 저장된 값이 Hashtable 객체라고 가정해 보자. 이때 Hashtable에 있는 객체뿐만 아니라 Hashtable의 키 값까지 추출하고자 한다면 다음과 같이 하면 된다.

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

위 예제에서 Hashtable에 정의되어 있는 keySet() method를 통해 키 값을 얻은 후 얻어진 키 값과 Hashtable의 get() method를 통해 키 값에 대응하는 값을 얻어오고 있다.

Velocity는 반복횟수(loop counter)를 얻기 위한 쉬운 방법을 제공한다. 아래는 그 예이다.

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

```
#end
</table>
```

위 예에서 반복횟수를 나타내는 reference는 $velocidyCount이다. 반복횟수를 나타내는 이 같은 reference 이름은 기본적으로 velocity.properties 파일에 설정되어 있으며 기본값으로 $velocityCount이다. 초기 값은 1부터 시작하지만 이 값 역시 velocity.properties 파일에서 수정이 가능하다. 다음은 velocity.properties 파일의 loop counter 속성을 지정하는 부분이다.

```
# 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 지시자는 원하는 위치에 local 파일을 삽입시키기 위해 사용된다. 삽입될 파일의 내용은 template engine을 거쳐 보여지지 않는다. 보안을 위해 삽입될 파일은 TEMPLATE_ROOT 밑에 위치 시켜야 한다. 사용법은 다음과 같다.

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

#include 지시자가 가리키는 파일은 " 로 감싸줘야 한다. 하나이상의 파일을 삽입하기 위해서는 ' , ' 로 구분해서 삽입할 파일명을 나열한다.

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

파일을 include하고자 할 때 꼭 파일명을 사용할 필요는 없다. 파일명 대신 variable를 사용할 수도 있다. 물론 이때 variable의 값은 include할 파일명이 되어야 한다. 특정 조건이나 기준에 따라 include 할 파일을 다르게 해야 할 경우 이러한 variable을 이용해 파일을 include하는 경우가 많다.

```
>> variable를 이용한 파일 include
    #include( "greetings.txt", $seasonalstock )
```

Parse

#parse 지시자는 VTL이 포함된 파일을 include하고자 할 때 사용한다. #parse를 통해 VTL이 포함된 파일을 include할 경우 Velocity는 먼저 VTL를 파싱한 후 명시된 template을 보여주게 된다.

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

위 예제는 me.vm이라는 template 파일을 include한 경우이다. #include 지시자와 마찬가지로 직접 template 명을 사용할 수도 있지만 variable를 사용해서 include할 수 있다. #parse가 참조하는 template 파일(include하고자 하는 template 파일)은 반드시 TEMPLATE_ROOT 밑에 위치해야 한다. #parse는 #include와는 달리 argument로 여러 개의 template 파일이 올 수 없다.

VTL template은 #parse 문장을 가진 template을 참조 할 수 있다. 보통 이 것을 중첩이라고 하는데 Velocity에서는 기본적으로 최대 10 단계까지 중첩을 허용한다. 이러한 단계는 velocity.properties 파일내의 parse_directive.maxdepth 부분에서 임의로 조절할 수 있다. velocity.properties 파일에 parse_directive.maxdepth 항목이 없으면 Velocity는 10으로 셋팅한다. 또한 VTL template에서는 재귀호출이 가능하다. 다음 예를 보자.

```
>> Myself.vm
   Before parse directives.
   #parse( " Myself.vm")
   After parse directive.
```

위 예제는 Myself.vm이라는 template에서 자기 자신을 재귀 호출하고 있다. 가령 velocity.properties 파일내의 parse_directive.maxdepth의 값이 3으로 셋팅되어 있을 경우 결과는 다음과 같이 출력될 것이다.

```
Before parse directives.
Before parse directives.
Before parse directives.
After parse directive.
After parse directive.
After parse directive.
```

Stop

#stop 지시자는 template engine의 실행을 멈추고자 할 때 사용한다. 보통 #stop 지시자는 디버깅 목적으로 유용하게 사용된다. Velocity가 #stop 지시자를 만나게 되면 #stop이후 VTL의 실행을 멈추게 된다.

Velocimacros

#macro 지시자는 VTL template에서 반복적인 부분을 정의하고자 할 때 사용한다. 이러한 Velocimacro는 코드량을 줄이고, 화면 출력시 발생할 수 있는 에러를 최소화하기 위한 것이다. 여기서는 Velocimacro를 ‘VM’으로 칭하도록 한다.

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

위 예에서 정의한 Velocimacro는 d이며, 이것은 다른 지시자와 유사한 방식으로 호출될 수 있다. d를 호출하기 위해서는 다음과 같이 하면 된다.

```
>> macro d 호출
    #d()
```

template이 호출되면 Velocity는 #d() 대신 ‘<tr><td></td></tr>’를 출력하게 된다.

Velocimacro는 인자값을 가질 수 있으며 위에서 본 것처럼 아무런 인자값도 가지고 있지 않을 수 있다. 하지만 Velocimacro가 호출 되었을 때 정의한 macro와 인자값의 수가 일치해야 한다. 다음은 color와 array를 인자값으로 가지고 있는 VM의 예이다.

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

VM의 body안에는 VTL template에 들어갈 수 있는 어떤 것도 들어갈 수가 있다. 위에서 정의한 tablerows라는 VM의 경우 foreach 지시자가 들어가 있다.

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

위 예는 앞에서 정의한 tablerows라는 VM을 사용한 예이다. 위 코드를 실행 했을 경우 다음과 같은 코드가 생성된다.

```
<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>
```

VM은 Velocity template 안에 필요할 때마다 정의할 수가 있다. 하지만 그럴 경우 같은 웹 사이트내의 다른 template에서는 정의한 VM을 사용할 수가 없게 된다. 모든 template에서 사용할 수 있는 VM을 정의하게 되면 여러 template에서 VM을 재 정의할 필요가 없으며, 작업량과 에러를 줄일 수 있다. 그리고 한번 VM을 수정함으로써 여러 template에서 수정된 내용을 반영할 수도 있다.

위에서 정의한 #tablerows라는 VM을 Velocimacros template library 안에 정의하게 되면 일반적인 모든 template에서 사용할 수 있다. 예를 들어 모든 균류를 나타내기 위한 mushroom.vm이라는 template에서 #tablerows VM은 대표적인 버섯의 일부에 대한 목록을 위해 호출될 수 있다.

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

mushroom.vm이 호출되면, Velocity는 template library(velocity.properties에 정의 된)에서 #tablerows를 찾게된다. 그리고 나서 다음과 같은 코드를 생성한다.

```
<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

VM에서 argument는 아래 나와있는 VTL 요소 중 하나가 될 수 있다.

- Reference : '$'로 시작하는 모든 것

- String literal : "$foo" 와 'hello'처럼 따옴표로 싸인 것
- Number literal : 1, 2 등
- IntegerRange : [1..2] 나 [$foo .. $bar]
- ObjectArray : ["a", "b", "c"]
- boolean 값 true
- boolean 값 false

VM에서 인자로 reference를 넘겨줄 때, reference 이름으로 넘어가게 된다. 그리고 실제로 VM 내부에서 사용될 때 그 값을 얻게 된다. 이러한 mechanism을 통해 method 호출 형식을 가진 reference를 인자값으로 넘겨줄 수 있으며, 실제 사용될 때 그 method를 호출하게 된다. 아래 예를 보자.

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

위 예제에서 callme라는 매크로 인자값으로 ‘$foo.bar()’라는 method reference를 넘겨주고 있다. 인자값이 넘어갈 때 실제 method가 리턴하는 값이 넘어가는 것이 아니라 이름만 넘어간 다음에 실제 VM 내부에서 사용될 때 method를 호출하여 그 리턴값을 사용하게 된다. 결과적으로 위 코드에서 foo.bar()는 세 번 호출되게 된다.

이렇게 하지 않으려면 method로부터 얻은 값을 새로운 reference에 넣고 그 reference를 넘겨주면 된다.

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

Velocimacro Properties

velocimacro.library

이 속성은 어플리케이션에서 사용할 Velocimacro template library 파일을 정의하기 위한 것이다. default 값은 VM_global_library.vm이라는 library 파일이며 여러 개의 Velocimacro template library 파일을 명시하고자 할 때는 comma(,)로 각 library 파일을 구분하면 된다.

velocimacro.permissions.allow.inline

이 속성은 VM을 필요할 때 마다 template 파일 내에 정의할 수 있는지 여부를 결정하기 위한 것이다. true나 false값을 가질 수 있으며 true값인 경우 필요할 때 마다 template에 VM을 정

의할 수 있다. false일 경우 velocimacro.library 속성에서 정의한 library파일 내에서만 VM을 정의할 수 있다.

velocimacro.permission.allow.inline.to.replace.global

이 속성은 필요에 의해 일반적인 template 파일 내에 정의한 VM이 Velocimacro template library 파일에 정의한 동일한 이름을 가진 Velocimacro를 대체할 수 있는지 여부를 결정하기 위한 것이다. true값일 경우 일반 template파일내에 정의한 VM이 Velocimacro template library에 정의한 동일한 이름을 가진 VM을 대신하게 된다. default 값은 false이다.

velocimacro.permissions.allow.inline.local.scope

이 속성은 보통 Velocimacro template library 파일 내에 정의한 VM의 scope를 결정하기 위해 사용된다. true나 false값을 가질 수 있으며, true일 경우 정의된 VM은 VM을 정의한 Velocimacro template library 내에서만 사용이 가능하게 된다. default값은 false이다.

velocimacro.context.localscope

true나 false값을 가질 수 있으며, true로 설정 되었을 때 VM내에서 #set() 지시자를 통해 context내에 값을 저장할 경우 VM 내에서만 참조가 가능하다.

velocimacro.library.autoreload

이 속성은 Velocimacro template library 파일의 자동 로딩 기능을 위한 것이다. true 값일 경우 VM이 호출되었을 때 해당 VM library파일의 변경여부를 체크한 후 필요에 따라 자동으로 reloading하게 된다. 보통 개발단계에서 true로 설정한 후 개발이 완료되면 false로 설정하는 것이 일반적이다. Velocimacro template library 파일의 자동로딩은 resource loader cache와 관련된 속성 값이 false일 경우에만 정상적으로 작동하게 된다(e.g. file.resource.loader.cache = false).

Escaping VTL Directives

VTL 지시자도 VTL reference와 비슷한 방식으로 “\”을 사용하여 directive를 escape 할 수 있다. 다음은 \를 이용하여 #include 지시자를 escape하는 예이다.

```
## #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" )
```

if-else-end 문장같이 단일 지시자 내에 여러 스크립트 요소를 포함하고 있는 VTL 지시자의경우 좀더 주의가 필요하다.

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

위 예에서 $jazz가 false인 경우 어떤 값도 출력되지 않는다.

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

위 예는 “\”을 사용하여 스크립트 요소를 제거한 경우다. 따라서 $jazz 가 true나 false에 상관없이 ‘Vyacheslav Ganelin’ 가 출력된다. 실제로 모든 스크립트 요소가 제거되었기 때문에 $jazz는 그 값이 boolean값인지 검사되지 않는다.

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

위와 같은 경우 $jazz가 true라면 다음과 같은 결과를 출력하게 된다.

```
\Vyacheslav Ganelin
\
```

$jazz가 false일 경우는 아무것도 출력되지 않는다.

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

위 예제에서 #if는 escape 되었지만 #end는 escape되지 않은 경우이다. 이런 경우 parsing 에러가 발생하게 된다.

VTL : Formatting Issues

VTL 문장 작성시 새줄문자(\n)나 공백은 무시가 된다.

아래 예제는 VTL 문장 작성시 새 줄이나 공백을 사용한 경우다.

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

다음 예제는 새 줄이나 공백을 사용하지 않고 이어서 작성한 경우다.

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

위 예는 다음과 같이 작성해도 같은 결과를 얻을 수 있다.

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

또는 다음과 같이 작성해도 된다.

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

Other Features and Miscellany

Math

Velocity에 set 지시자를 이용해서 template에서 사용할 수 있는 수학관련 함수를 작성할 수 있다. 아래는 덧셈,뺄셈,곱셈,나눗셈에 대한 각각의 예제이다

```
    #set( $foo = $bar + 3 )
```

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

나눗셈의 경우 결과는 항상 integer값이 된다. 나머지는 %를 사용하여 구할 수 있다.

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

Velocity에서 수식을 수행할 때 오직 integer값만 가능하다. integer가 아닌 값으로 수행할 때 로그에 기록되며 null을 리턴하게 된다.

Range Operator

범위를 나타내는 연산자는 #set이나 #foreach와 같은 지시지와 함께 사용된다. 범위 연산자는 integer를 값으로 가지는 객체배열을 생성할 때 유용하며 구문은 다음과 같다.

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

n과 m은 integer값이 되어야 한다. m이 n보다 크거나 작을 수 있다. 다음은 범위를 나타내는 연산자 예제이다.

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

위 예의 결과는 다음과 같다.

```
First example:
    1 2 3 4 5

Second example:
    2 1 0 -1 -2

Third example:
    0 1

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

네 번째 예에서 보는 것처럼 범위 연산자는 #set과 #foreach 지시자와 같이 사용할 때만 배열을 생성하게 된다.

Advanced Issues : Escaping and !

Escaping Valid VTL References 부분에서 '\'를 '$' 문자 앞에 둠으로써 reference를 escape하는 일반적인 방법을 설명했다. 여기서는 '$' 앞이 아닌 '!' 앞에 '\'를 두는 특별한 경우 reference가 어떻게 처리 되는지 알아본다. 아래 예제를 보자.

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

위 예제는 일반적인 reference escape 방법과는 다르게 ! 앞에 '\' 문자가 위치하고 있다. 위 예제의 결과는 다음과 같이 처리된다.

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

그럼 일반적인 reference escape방법을 사용했을 경우 어떻게 처리되는지 비교해 보자.

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

위 결과는 다음과 같이 처리된다.

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

위에서 보여준 예제를 통해 '!' 앞에 '\'를 두었을 경우 모든 경우에 reference가 escape된 반면 '$' 앞에 '\'를 두었을 경우 마지막 경우만 제외하고 reference가 escape 된 것을 알 수 가 있다.

Velocimacro Miscellany

이 부분은 Velocimacro와 관련된 주제에 관한 FAQ의 일부이다.

- VM의 인자값으로 지시자나 다른 VM을 사용할 수 있는가?

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

  사용할 수 없다. 지시자는 지시자의 인자값으로 유효하지 않다. VM역시 특별한 목적을 위한 지시자로 볼 수 있기 때문에 사용할 수 없다. 한가지 방법이 있는데 그것은 지시자로 표현할 내용을 " 로 묶어주는 것이다. 아래 예를 보자.

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

  위 예제를 다음과 같이 한 줄로도 작성이 가능하다.

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

  여기서 알아둬야 할 것은 bold라는 VM에 대한 처리 결과가 center라는 VM에 넘어가는 것이 아니라 #bold('hello') 자체가 넘어가게 된다는 것이다. 그리고 넘겨진 인자값은 VM내부에서 처리되게 된다. 다음 예를 보자.

```
(1)
    #macro( inner $foo )
        inner : $foo
    #end

(2)
    #macro( outer $foo )
    #set($bar = "outerlala")
        outer : $foo
    #end

(3)
     #set($bar = 'calltimelala')
     #outer( "#inner($bar)" )
```

위 예제 결과는 다음과 같이 처리된다.

```
Outer : inner : outerlala
```

(3)부분에서 outer라는 VM을 사용하고 있으며 인자값으로 "#inner($bar)"가 넘겨진다. 이 때 inner VM에 대한 처리를 하는 (1)번이 호출되는 게 아니라 "#inner($bar)" 이 그대로 넘겨져 (2)를 수행하게 된다. (2)에서 선언된 bar라는 variable reference 값을 인자값으로 넘어온 "#inner($bar)" 부분의 $bar 부분에 셋팅하게 되며 그 후 (1) 호출되어 inner VM이 실행되게 된다. 만일 (3)에서 inner VM이 처리된 값이 인자값으로 넘어갔다면 다음과 같이 처리되었을 것이다.

```
Outer : inner : calltimelala
```

다음 예를 보자.

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

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

위 예에서 rowColor()는 VM내에서 반복적으로 호출된다. 이러한 것을 피하려면 아래처럼 VM외

부에서 method를 호출한 후 결과값을 VM 인자값으로 넘겨주면 된다.

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

- #parse()를 통해 Velocimacro를 등록할 수 있는가?

현재 VM은 template내에서 처음 사용되기 전에 반드시 정의 되어야 한다. 이것은 VM을 사용하기 전에 VM을 선언해야 된다는 것을 의미한다. 만일 #macro() 지시자가 포함된 template을 #parse()를 사용하여 parsing하고자 한다면 이 것은 중요하다. 왜냐하면 #parse()는 runtime시 동작하지만 VM을 사용하는 element가 있는 경우 parser는 parsetime에 #macro()를 찾게 되기 때문이다. 따라서 이 경우 제대로 작동하지 않는다. 이러한 것을 해결하기 위해서 Velocity가 처음 시동시 VM을 로드할 수 있도록 velocimacro.library의 기능을 사용하면 된다.

- Velocimacro 자동 로딩기능은 무엇인가?

자동로딩 기능은 개발 중에 사용하는 속성이다. 따라서 개발이 완료 되어 배포할 경우 자동로딩기능을 사용하지 않도록 설정 값을 바꿔줘야 한다. 이 속성은 velocity.properties 파일 내에서 지정할 수 있다. 다음은 자동로딩 기능 속성을 지정하는 부분이다.

```
velocimacro.library.autoreload
```

초기 값은 false이다. 이 속성 값을 true로 설정하면 자동로딩 기능을 사용할 수 있다. 하지만 이 기능도 다음과 같이 resource loader에 대한 속성이 false로 지정되어야만 제대로 작동한다.

```
>> resource loader에 대한 속성값 설정
    <type>.resource.loader.cache = false
```

위 속성 역시 velocity.properties 파일 내에서 설정할 수 있다. 여기서 <type>은 ‘file’과 같이 application에서 사용하는 resource loader의 이름이 된다.

String Concatenation

VTL에서 reference를 결합하려면 다음과 같이 결합할 reference를 붙여서 사용하면 된다.

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

```
The clock is $size$name.
```

위 예제의 결과는 ‘The clock is BigBen’으로 처리 될 것이다. 결합한 String을 method 인자 값으로 넘겨주거나 새로운 reference에 할당하고자 할 때도 같은 방식으로 하면 된다. 다음은 새로운 reference의 값으로 결합한 string을 할당하는 경우이다.

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

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

The clock is $clock.
```

위 예제 결과 역시 ‘The clock is BigBen’으로 처리 된다. 만일 reference와 일반문자를 결합할 때는 ‘formal notation’ 사용하면 된다. 아래 예제를 보자.

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

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

The clock is $clock.
```

위 예제는 두 개의 reference와 일반문자 ‘Tall’을 결합하는 예이다. clock이라는 reference variable에 값을 할당할 때 $sizeTall$name과 같이 한다면 Velocity는 첫 번째 reference를 $sizeTall로 인식하여 처리할 것이다. 따라서 이러한 것을 방지하기위해 formal notation을 이용하여 size라는 reference variable를 표기하고 있다. 예제의 결과는 ‘The clock is BigTallBen’으로 처리 될 것이다.