MacroEvaluationStrategy

Note Velocity 2.0 now always uses the call by sharing (aka Java) calling convention.

From Wikipedia: In computer science, an evaluation strategy is a set of (usually deterministic) rules for evaluating expressions in a programming language. An evaluation strategy defines when and in what order the arguments to a function are evaluated, when they are substituted into the function, and what form that substitution takes.

Looking at the various standard evaluation strategies (see http://www.knowledgerush.com/kr/encyclopedia/Call-by-something/ and http://en.wikipedia.org/wiki/Evaluation_strategy), none of them applies to Velocity. It's a mix between call by macro expansion, call by sharing, call by value and other behaviors.

In non-strict mode, Velocity macros work mostly as call-by-macro expansion internally, with call-by-sharing external effects, but affected by automatic assignment of global variables when formal parameters can't be assigned (using a non-lvalue argument as an lvalue). We could call this *relaxed call by lazy sharing*. This means that:

- **Lazy evaluation**: actual parameters are not computed (reduced in lambda-calculus semantics) until they are actually used. For example, calling a macro with `#someMacro($aList.add('a value'))` will not execute the `add` call until the formal parameter of the macro is actually used, which means that the call might not get executed at all.
- **Call by macro expansion:** the actual parameters are computed each time the formal parameter is used. This means that in the previous example, the `add` call will be executed each time the formal parameter is used, possibly adding the value to the list several times. *This is the behavior of C macros.*
- **Call by sharing:** the method arguments are not passed by reference, meaning that trying to assign a new value to a formal parameter will not change the value of the actual parameter after the macro terminates execution, but changing the internal data of a passed object will work. *This is the behavior of Java method calling.* **Warning: the call-by-sharing behavior is new in Velocity 1.7. The previous behavior was closer to true call-by-macro expansion.**
- **Relaxed**, or **error-free non-lvalue assignment:** trying to assign a value to a formal parameter which is bound to a non-lvalue, for example calling `#someMacro('a')` where `someMacro` tries to assign a new value to that parameter, will create a new variable which shadows the formal parameter. *This is closer to the behavior of C method calling, but holds true only when assigning values to non-lvalue arguments.*

Examples

Call by sharing example (new behavior introduced in 1.7b1, and shortly only in 1.6.1):

```
#macro(callBySharing $x $map)
  #set($x = 'a')
  $map.put('x', 'a')
#end
#set($y = 'y')
#set($map = {})
#callBySharing($y $map)
$y -> 'y' (but is 'a' in 1.6.2, 1.6.0 and before)
$map.x -> 'a'
```

Java-like behavior. See https://issues.apache.org/jira/browse/VELOCITY-681

Call by name/macro expansion example (and call by need counter-example)

```
#macro(callByMacro1 $p)
  not using
#end
#macro(callByMacro2 $p)
  using: $p
  using again: $p
#end
#set($x = [])
#callByMacro1($x.add('t'))
$x -> [], the add call was not executed
#callByMacro2($x.add('t'))
$x -> [t,t], the add call was executed twice
```

This is a classic call by name example.

Call by value? example (and call by name or expansion counter-example)

```
#macro(callByValueSwap $a $b)
  $a $b becomes ##
  #set($tmp = $a)
  #set($a = $b)
  #set($b = $tmp)
  $a $b
#end
#callByValueSwap('a', 'b') ->
a b becomes b a
```

In a true call-by-name (or macro-expansion) implementation, $a would always be `'a'`. What actually happens is that `#set($a = $b)` creates the global variable $a which shadows the formal parameter. This is a bit strange, since formal parameters should never be shadowed by external parameters. Since $a is not an lvalue, a call-by-name behavior would result in an exception.

Call by macro expansion example (and call by value or sharing counter-example)

```
#macro(changeMap $map)
  Before: $map.someKey
  #set($map.someKey = 'new value')
  After: $map.someKey
#end
#changeMap({'someKey' : 'old value'}) -> Before: old value, After: old value
```

If this was true call-by-sharing (or call-by-reference), then $map would be a pointer to a real map which would be changed by the first set. See https://issues.apache.org/jira/browse/VELOCITY-684

Call by macro expansion example (exposes name capture, call by name counter-example)

```
#macro(nameCaptureSwap $a $b)
  $a $b becomes ##
  #set($tmp = $a)
  #set($a = $b)
  #set($b = $tmp)
  $a $b
#end
#set($x = 'a')
#set($tmp = 'b')
#nameCaptureSwap($x $tmp) ->
a b becomes a a
```

This is the classic name capture example, which only happens with call by macro expansion.

Mixing different types of actual and formal parameters will expose mixed behaviors.

Future

The current (1.7 beta 1) behavior is complex, non-uniform and surprising in several (not so common) cases, and might be revisited for 2.0. The most likely candidate is a true *call by sharing* behavior, familiar to Java developers.