Ruby master - Feature #14914

Add BasicObject#instance_exec_with_block

07/16/2018 04:07 PM - jeremyevans0 (Jeremy Evans)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Currently, you cannot use instance_exec with a block that requires a block argument. If you have a block that requires a block argument and you want to get the equivalent of instance_exec, you have to do something like:

```
singleton_class.define_method(:temporary_method_name, &block)
send(:temporary_method_name, *args, &block_arg)
singleton_class.remove_method(:temporary_method_name)
```

That approach doesn't work for frozen objects, making it impossible to get the equivalent of instance_exec for a frozen object if you have a block that requires a block argument.

The attached patch implements instance_exec_with_block, which is the same as instance_exec, except the last argument must be a Proc which is passed as the block argument instead of the regular argument to the block.

```
1.instance_exec_with_block(42, 2, proc{|x| self * x}) do |a, b, &blk|
  (self + a).instance_exec(b, &blk)
end
# => 86
```

This method cannot be implemented in a gem because it requires changes to vm.c and vm_eval.c to implement. There wasn't a function allowing you to provide both a cref and a block argument when evaluating a block (hence the addition of vm_yield_with_cref_and_block in the patch).

There are alternative APIs that could support this feature. One approach would be adding support for :args and :block_arg keyword arguments to instance_eval (if no regular arguments are given).

History

#1 - 07/18/2018 06:23 AM - matz (Yukihiro Matsumoto)

Real-world use-case, please?
Any problem with the following code?

```
blk = proc{|x| self * x}
1.instance_exec(42, 2) do |a, b|
  (self + a).instance_exec(b, &blk)
end
# => 86
```

Matz.

#2 - 07/18/2018 02:41 PM - jeremyevans0 (Jeremy Evans)

matz (Yukihiro Matsumoto) wrote:

> Real-world use-case, please?

This is needed when you don't control the block you want to instance_exec, because it comes from another library or the user. If such a block requires a block argument, you can't use instance_exec. Currently in this situation, you need to define a method on the receiver to get similar behavior.

One case where this could be used is in Sequel, where currently we define multiple methods for all associations where the default implementation for each method can be modified by the user using options with proc values (where the procs can potentially accept block arguments). We can't use instance_exec to execute the procs directly, because that can't handle block arguments. We could potentially use instance_exec_with_block to avoid defining methods in cases where methods are not needed and only created to work around this limitation in instance_exec.

> Any problem with the following code?

```
blk = proc{|x| self * x}
1.instance_exec(42, 2) do |a, b|
  (self + a).instance_exec(b, &blk)
end
# => 86
```

This requires that you control the block and can modify it to use the closed over variable. You can't use this approach if you don't control the block being instance_execed.

#3 - 08/09/2018 08:07 AM - ioquatix (Samuel Williams)

What about some kind of currying? i.e. binding the block to the block you want to instance exec.

#4 - 08/09/2018 02:30 PM - jeremyevans0 (Jeremy Evans)

ioquatix (Samuel Williams) wrote:

> What about some kind of currying? i.e. binding the block to the block you want to instance exec.

I'm not sure what you mean, could you elaborate with example code?

Files

0001-Add-BasicObject-instance_exec_with_block.patch	6.04 KB	07/16/2018	jeremyevans0 (Jeremy Evans)