

Ruby trunk - Feature #8481

Module#using

06/03/2013 10:58 PM - shugo (Shugo Maeda)

Status:	Closed
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	2.1.0
Description	
<p>As I said at RubyKaigi2013, refinements in Ruby 2.0 have a problem that it's impossible to use incompatible refinements in the same file.</p> <p>And at RubyKaigi, I've proposed the new option using: of instance_eval, by which you can activate refinements in the specified module only in the given block.</p> <p>See my presentation slides for details: http://shugo.net/tmp/refining_refinements.pdf</p> <p>However, Matz doesn't like that idea for the following two reasons:</p> <ol style="list-style-type: none">1. It's difficult for users to expect what refinements are activated in a block.2. It's difficult for Ruby implementors to implement it efficiently. <p>So, I propose Module#using instead of the using: option of instance_eval.</p> <p>Module#using had once been introduced into trunk, but it was removed from Ruby 2.0.</p> <p>I'd like to make it simpler, as follows.</p> <ol style="list-style-type: none">1. Module#using activates refinements in a given module only in the current class or module definition.2. So Module#using is private and the receiver of Module#using should be self.3. The refinements never be activated in class or module definitions reopened later.4. The refinements never be inherited to subclasses. <p>That is, Module#using works lexically.</p> <p>EXAMPLE 1</p> <pre>class Foo using Ref1 # Refinements in Ref1 are activated only in the current definition of Foo. end class Bar using Ref2 # Refinements in Ref2 are activated only in the current definition of Bar. end</pre> <p>EXAMPLE 2</p> <pre>class Foo using Ref1 # Refinements in Ref1 are activated only in the current definition of Foo. end class Foo # Refinements in Ref1 are not activated here. end</pre> <p>EXAMPLE 3</p> <pre>class Foo using Ref1 # Refinements in Ref1 are activated only in the current definition of Foo. end class Bar < Foo # Refinements in Ref1 are not activated here. end</pre> <p>Any thoughts?</p>	

Associated revisions

Revision 17f1cdaa - 06/12/2013 02:33 PM - shugo (Shugo Maeda)

- eval.c (mod_using): new method Module#using, which activates refinements of the specified module only in the current class or module definition. [ruby-core:55273] [Feature #8481]
- test/ruby/test_refinement.rb: related test.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@41261 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 41261 - 06/12/2013 02:33 PM - shugo (Shugo Maeda)

- eval.c (mod_using): new method Module#using, which activates refinements of the specified module only in the current class or module definition. [ruby-core:55273] [Feature #8481]
- test/ruby/test_refinement.rb: related test.

Revision 41261 - 06/12/2013 02:33 PM - shugo (Shugo Maeda)

- eval.c (mod_using): new method Module#using, which activates refinements of the specified module only in the current class or module definition. [ruby-core:55273] [Feature #8481]
- test/ruby/test_refinement.rb: related test.

Revision 41261 - 06/12/2013 02:33 PM - shugo (Shugo Maeda)

- eval.c (mod_using): new method Module#using, which activates refinements of the specified module only in the current class or module definition. [ruby-core:55273] [Feature #8481]
- test/ruby/test_refinement.rb: related test.

Revision 41261 - 06/12/2013 02:33 PM - shugo (Shugo Maeda)

- eval.c (mod_using): new method Module#using, which activates refinements of the specified module only in the current class or module definition. [ruby-core:55273] [Feature #8481]
- test/ruby/test_refinement.rb: related test.

Revision 41261 - 06/12/2013 02:33 PM - shugo (Shugo Maeda)

- eval.c (mod_using): new method Module#using, which activates refinements of the specified module only in the current class or

module definition. [ruby-core:55273] [Feature #8481]

- test/ruby/test_refinement.rb: related test.

Revision 41261 - 06/12/2013 02:33 PM - shugo (Shugo Maeda)

- eval.c (mod_using): new method Module#using, which activates refinements of the specified module only in the current class or module definition. [ruby-core:55273] [Feature #8481]
- test/ruby/test_refinement.rb: related test.

History

#1 - 06/04/2013 12:05 AM - Anonymous

Then, if it works lexically, it is not a function of Module, but of ... of ... Kernel? If it modified the module logically, it would be Module's concern. But like this, it is a concern of the interpreter, imao. Something like "private". I don't even know if "private" keyword is method :-)

#2 - 06/04/2013 08:23 AM - matz (Yukihiro Matsumoto)

As I already told you, I can accept (or even encourage) this change. I am interested in how implementers of other Ruby.

Matz.

#3 - 06/04/2013 10:12 AM - shugo (Shugo Maeda)

boris_stitnicky (Boris Stitnicky) wrote:

Then, if it works lexically, it is not a function of Module, but of ... of ... Kernel? If it modified the module logically, it would be Module's concern. But like this, it is a concern of the interpreter, imao. Something like "private". I don't even know if "private" keyword is method :-)

private is not a keyword. private at toplevel is the method main.private and private at module level is the method Module#private. Hence, it's reasonable to provide module-level using as Module#using.

#4 - 06/10/2013 09:17 PM - headius (Charles Nutter)

I do not currently have a problem with Module level #using, provided it is still lexical. The same mechanism for file-level #using would work here: if a "using" call is present in the code, we'll treat all calls in that scope as potentially refined (different call logic). Scopes without a #using call will not do refined dispatch.

A few other clarifications I think are important:

- Since this is lexical, you can't **send** or `send :using` from any scope and expect it to work. It might work by accident on some implementations, but this is non-spec. Should it be specified that it explicitly does *not* work?
- I assume we still do not have "magic" `#send` and `#instance_method` and friends, correct?

Regarding timeline for implementation in JRuby...

Because of the invasive nature of the changes required, we are not comfortable implementing it in the JRuby 1.7.x codebase. Therefore, we will probably not be able to start in earnest on a final implementation until after we branch 1.7. However, I would like to get a prototype implementation based on current 2.0 and upcoming 2.1 refinement features so we can work through issues sooner rather than later. I hope to do that this month...but hope is a complicated thing.

#5 - 06/10/2013 11:11 PM - matz (Yukihiro Matsumoto)

- It's OK that `send(:using)` do not work
- we will not provide magic send in the near future (no plan at all)

Matz.

#6 - 06/11/2013 02:21 PM - shugo (Shugo Maeda)

headius (Charles Nutter) wrote:

A few other clarifications I think are important:

- Since this is lexical, you can't **send** or `send :using` from any scope and expect it to work. It might work by accident on some implementations, but this is non-spec. Should it be specified that it explicitly does *not* work?
- I assume we still do not have "magic" `#send` and `#instance_method` and friends, correct?

Agreed. Reflection API should not care refinements in Ruby 2.1.

Because of the invasive nature of the changes required, we are not comfortable implementing it in the JRuby 1.7.x codebase. Therefore, we will probably not be able to start in earnest on a final implementation until after we branch 1.7. However, I would like to get a prototype implementation based on current 2.0 and upcoming 2.1 refinement features so we can work through issues sooner rather than later. I hope to do that this month...but hope is a complicated thing.

I see. Thanks for your effort for refinements.

#7 - 06/12/2013 11:34 PM - shugo (Shugo Maeda)

- Status changed from *Open* to *Closed*

- % Done changed from 0 to 100

This issue was solved with changeset [r41261](#).

Shugo, thank you for reporting this issue.

Your contribution to Ruby is greatly appreciated.

May Ruby be with you.

-
- `eval.c (mod_using)`: new method `Module#using`, which activates refinements of the specified module only in the current class or module definition. [ruby-core:55273] [Feature [#8481](#)]
 - `test/ruby/test_refinement.rb`: related test.