Ruby trunk - Feature #6609

Toplevel as self extended module

06/19/2012 09:58 PM - trans (Thomas Sawyer)

Status:	Closed	
Priority:	Normal	
Assignee:	nobu (Nobuyoshi Nakada)	
Target version:	2.0.0	

Description

As promised sometime back here is my proposal that Toplevel object become a self-extended module instead of the current partial Object class proxy.

I have written about it in a blog post: http://trans.github.com/2012/06/17/kill-the-proxy-and-save-toplevel.html

In summary the basic idea is to have a special toplevel namespace that is self-extended, e.g.

module Toplevel
extend self
end

in which all toplevel code is evaluated.

Definitions at the toplevel would no longer inject into Object class. This frees up the toplevel to be used for general purpose DSL "batch" scripting. What I mean by that is that one can create a DSL, load it in to toplevel and then evaluate scripts based on it simply by load/require and without fret that the code loaded in will infect Object if it defines it's own methods.

Conceptually the idea of self-extended module is much simpler than current proxy object --there is really nothing special to understand about it since it is just a module like any other module.

With regard to backward compatibility, the only programs that would be effected are any that defined a toplevel method fully expecting it to add a method to Object. But those will be very rare since it is generally considered bad form to do that. (And of course the simple fix is to wrap the method in the proper class Object private ... end code.

Associated revisions

Revision 499b5a91 - 11/01/2012 11:24 PM - nobu (Nobuyoshi Nakada)

proc.c: main.define_method

- proc.c (top_define_method): new method, main.define_method. [ruby-core:45715] [Feature #6609]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37417 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 37417 - 11/01/2012 11:24 PM - nobu (Nobuyoshi Nakada)

proc.c: main.define_method

- proc.c (top_define_method): new method, main.define_method. [ruby-core:45715] [Feature #6609]

Revision 37417 - 11/01/2012 11:24 PM - nobu (Nobuyoshi Nakada)

proc.c: main.define_method

- proc.c (top_define_method): new method, main.define_method. [ruby-core:45715] [Feature #6609]

Revision 37417 - 11/01/2012 11:24 PM - nobu (Nobuyoshi Nakada)

proc.c: main.define_method

- proc.c (top_define_method): new method, main.define_method. [ruby-core:45715] [Feature #6609]

Revision 37417 - 11/01/2012 11:24 PM - nobu (Nobuyoshi Nakada)

proc.c: main.define_method

- proc.c (top_define_method): new method, main.define_method. [ruby-core:45715] [Feature #6609]

Revision 37417 - 11/01/2012 11:24 PM - nobu (Nobuyoshi Nakada)

proc.c: main.define_method

- proc.c (top_define_method): new method, main.define_method. [ruby-core:45715] [Feature #6609]

Revision 37417 - 11/01/2012 11:24 PM - nobu (Nobuyoshi Nakada)

proc.c: main.define_method

- proc.c (top_define_method): new method, main.define_method. [ruby-core:45715] [Feature #6609]

History

#1 - 06/20/2012 05:57 AM - drbrain (Eric Hodel)

trans (Thomas Sawyer) wrote:

> As promised sometime back here is my proposal that Toplevel object become a self-extended module instead of the current partial Object class proxy.

What is a "partial Object class proxy"? It does not seem to exist. The top-level object (rb_vm_top_self) is an instance of Object where to_s is overridden to return "main" (see Init_top_self in vm.c).

> I have written about it in a blog post: http://trans.github.com/2012/06/17/kill-the-proxy-and-save-toplevel.html

This post has the misconception that the top-level object is somehow special.

Of course you cannot call define_method at top-level. self is not a Module subclass, it's an Object instance:

$ ruby -e 'p self.class.ancestors'
[Object, Kernel, BasicObject]

Of course methods defined at the top-level affect every other object. That's exactly how it works everywhere else:

$ cat t.rb
class C
def initialize name
@name = name
end

```
def make
  def name
    @name
  end
end
```

end

c1 = C.new :c1
c2 = C.new :c2

c2.name rescue puts "c2.name not defined"

c1.make

p c1.name
p c2.name

$ ruby t.rb
c2.name not defined
:c1
:c2

> In summary the basic idea is to have a special toplevel namespace that is self-extended, e.g.
>
> module Toplevel
> extend self
> end
>
> in which all toplevel code is evaluated.
>
> Definitions at the toplevel would no longer inject into Object class. This frees up the toplevel to be used for general purpose DSL "batch" scripting. What I mean by that is that one can create a DSL, load it in to toplevel and then evaluate scripts based on it simply by load/require and

> without fret that the code loaded in will infect Object if it defines it's own methods.
>
> Conceptually the idea of self-extended module is much simpler than current proxy object --there is really nothing special to understand about it since it is just a module like any other module.

Since you are adding more things it seems more complicated. There are no extra things in the current top-level object. It's so simple it can be literally described as "the top-level object".

> With regard to backward compatibility, the only programs that would be effected are any that defined a toplevel method fully expecting it to add a method to Object. But those will be very rare since it is generally considered bad form to do that. (And of course the simple fix is to wrap the method in the proper class Object private ... end code.

Such a change would break this:

https://github.com/xml4r/libxml-ruby/blob/REL_1_1_3/ext/libxml/extconf.rb#L5-12

While this is a thing you should probably not write it does exist in the wild.

#2 - 06/21/2012 07:42 PM - trans (Thomas Sawyer)

> What is a "partial Object class proxy"? It does not seem to exist. The top-level object (rb_vm_top_self) is an instance of Object where to_s is overridden to return "main" (see Init_top_self in vm.c).
> This post has the misconception that the top-level object is somehow special.

It is "special" in that is delegates certain calls to the Object class. e.g.

```
def foo; end
Object.private_instance_methods  #=> [:foo]
```

Hence it is a "proxy" for Object class. But it is not a complete proxy, e.g. it does not delegate "define_method", so it is "partial".

> Since you are adding more things it seems more complicated. There are no extra things in the current top-level object. It's so simple it can be literally described as "the top-level object".

To the contrary, top-level Object is not a generic instance of Object. What I am proposing is both simpler and more useful.

> Such a change would break this:

Yes, but such cases are rare and easily fixed. And you explain why that is so...

> While this is a thing you should probably not write it does exist in the wild.

Strike out "probably", and you have one of the reasons for this proposal.

#3 - 06/22/2012 03:20 AM - drbrain (Eric Hodel)

trans (Thomas Sawyer) wrote:

> > What is a "partial Object class proxy"? It does not seem to exist. The top-level object (rb_vm_top_self) is an instance of Object where to_s is overridden to return "main" (see Init_top_self in vm.c).
> > This post has the misconception that the top-level object is somehow special.
>
> It is "special" in that is delegates certain calls to the Object class. e.g.
>
> ```
> def foo; end
> Object.private_instance_methods  #=> [:foo]
> ```

There are no method calls on the top-level object in this example. Nothing is delegated.

> Hence it is a "proxy" for Object class. But it is not a complete proxy, e.g. it does not delegate "define_method", so it is "partial".

You can't call define_method on an Object instance because there is no such method on an Object instance.

Again, define_method only exists on Module and its subclasses.

> Since you are adding more things it seems more complicated. There are no extra things in the current top-level object. It's so simple it can be literally described as "the top-level object".

> To the contrary, top-level Object is not a generic instance of Object.

I cannot find this in the ruby source code. Can you point me to a function that modifies rb_vm_top_self() in this way?

I can only find where it is a generic instance of Object (see my previous comment).

> What I am proposing is both simpler and more useful.

> Such a change would break this:

> Yes, but such cases are rare and easily fixed. And you explain why that is so...

> While this is a thing you should probably not write it does exist in the wild.

> Strike out "probably", and you have one of the reasons for this proposal.

That people have done bad things is not a compelling reason.

#4 - 06/22/2012 09:31 AM - trans (Thomas Sawyer)

> There are no method calls on the top-level object in this example. Nothing is delegated.

I've been able to find the code for three methods which delegate:

```
eval.c:      rb_define_singleton_method(rb_vm_top_self(), "include", top_include, -1);
vm_method.c: rb_define_singleton_method(rb_vm_top_self(), "public", top_public, -1);
vm_method.c: rb_define_singleton_method(rb_vm_top_self(), "private", top_private, -1);
```

I understand what you are saying about def now, b/c of how Ruby handles def in other defs. Actually, I always thought that was little odd. I know others have suggested that maybe inner defs should be local to the method itself. Not that it's a big deal --I think that's very minor thing. But now I see how one oddity became the result of another.

> Again, define_method only exists on Module and its subclasses.

If one is able to use def in a context, does in not stand to reason that one could also use define_method? Since #define_method is, after all, the means given us to create methods dynamically (without resorting to eval).

> That people have done bad things is not a compelling reason.

That is but *one* (good) reason. In fact, have you considered the other merits I mentioned? Arguing against this solely on the basis that the top-level is what the top-level is, pretty much misses the point to *improve* upon what it is.

P.S. In the course of working on this I may have come across an issue with irb. Looks like it is making top-level methods public methods of Object instead of private.

#5 - 06/22/2012 10:03 AM - drbrain (Eric Hodel)

trans (Thomas Sawyer) wrote:

> There are no method calls on the top-level object in this example. Nothing is delegated.

> I've been able to find the code for three methods which delegate:
>
> ```
> eval.c:      rb_define_singleton_method(rb_vm_top_self(), "include", top_include, -1);
> vm_method.c: rb_define_singleton_method(rb_vm_top_self(), "public", top_public, -1);
> vm_method.c: rb_define_singleton_method(rb_vm_top_self(), "private", top_private, -1);
> ```

These aren't delegated, they are singleton methods which add behavior to make top-self more useful.

rb_define_singleton_method() is the same as:

o = Object.new
def o.include(*) end

> I understand what you are saying about def now, b/c of how Ruby handles def in other defs. Actually, I always thought that was little odd. I know others have suggested that maybe inner defs should be local to the method itself. Not that it's a big deal --I think that's very minor thing. But now I see how one oddity became the result of another.
>
> > Again, define_method only exists on Module and its subclasses.
>
> If one is able to use def in a context, does in not stand to reason that one could also use define_method? Since #define_method is, after all, the means given us to create methods dynamically (without resorting to eval).

No, due to what you said above ("how Ruby handles def in other defs"). Also:

ruby -e 'class C; def make; define_method(:foo) { } end; end; C.new.make'

> > That people have done bad things is not a compelling reason.
>
> That is but *one* (good) reason.

The current behavior is bad for some things and good for others.

Also, since the top-self is one of the first objects new rubyists interact with, changing the behavior of top-self may break introductory tutorials on ruby. Many, many people use it so we must be very careful when changing it.

> In fact, have you considered the other merits I mentioned? Arguing against this solely on the basis that the top-level is what the top-level is, pretty much misses the point to *improve* upon what it is.

It is easy to create a clean-room DSL evaluation context now:

m = Module.new do
extend SomeDSL
end

m.instance_eval File.read 'dsl_file.rb'

So the improvement is a convenience for a very rare use at the expense of a change every rubyist needs to learn about.

#6 - 06/22/2012 11:40 AM - trans (Thomas Sawyer)

=begin

> These aren't delegated, they are singleton methods which add behavior to make top-self more useful.

static VALUE
top_include(int argc, VALUE *argv, VALUE self)
{
rb_thread_t *th = GET_THREAD();

```
rb_secure(4);
if (th->top_wrapper) {
rb_warning
    ("main#include in the wrapped load is effective only in wrapper module");
return rb_mod_include(argc, argv, th->top_wrapper);
}
return rb_mod_include(argc, argv, rb_cObject);
```

}

Also, #define_method would make top level more useful too.

> No, due to what you said above ("how Ruby handles def in other defs"). Also:
>
> ruby -e 'class C; def make; define_method(:foo) { } end; end; C.new.make'

No? This only serves to demonstrate the oddity. def and define_method aren't resulting in the same thing here. How is that what one would expect in the least? It's not, and this is something that one has to commit to memorization b/c it is not intuitive. But this proposal is also not about def. I understand it explains why top-level ends up a method of Object class, but this proposal seeks to end that regardless of def's behavior.

> The current behavior is bad for some things and good for others.

What is it good for?

> Also, since the top-self is one of the first objects new rubyists interact with, changing the behavior of top-self
> may break introductory tutorials on ruby. Many, many people use it so we must be very careful when changing it.

Don't think it will break any such tutorials. This proposal should be compatible it every respect that counts. If a tutorial is teaching new rubyists to define top-level methods so they can be called upon any object, that's not a good tutorial.

But of course I agree, we have to be careful about such changes. I have considered this idea for many years and have asked many times over those years if it had any show-stopper issues that would make it not feasible. No one's come forward with any, so I think it's worth more serious consideration at this time.

> It is easy to create a clean-room DSL evaluation context now:

Yes, I have done this a number of times. It's actually not quite that simple or convenient. The ability to load/require into toplevel makes repurpose of top level for DSL very attractive. Using instance eval in subcontext also presents constant lookup issues.

> So the improvement is a convenience for a very rare use at the expense of a change every rubyist needs to learn about.

I think it would be used more than you might think. I have at least two programs I can think of off the top of my head where I could use this. Whereas the current behavior is of no beneficial use at all and seems to me really only serves to create bad code.

I realize this proposal entails a bit of reorientation with the idea of what the top level object *is*, especially for someone like yourself who is familiar with the underlying implementation. But I think if you step back and think about what how we conceive of the top level already -- as a "namespace", then this proposal really starts to take shape.

Rather then something every rubyist needs to learn, it would bring Ruby up to what most Rubyists would generally expect.
=end

#7 - 06/22/2012 05:23 PM - now (Nikolai Weibull)

On Fri, Jun 22, 2012 at 3:03 AM, drbrain (Eric Hodel)
drbrain@segment7.net wrote:

> It is easy to create a clean-room DSL evaluation context now:
>
> ```
> m = Module.new do
>   extend SomeDSL
> end
>
> m.instance_eval File.read 'dsl_file.rb'
> ```

Well, load does quite a bit more than File.read.

Am I correct in understanding that what is being discussed could have
been solved in a way that I've previously proposed, where load's
optional second argument can instead be a module that should be used
as the top-level namespace instead of the anonymous one you get if
it's true?

#8 - 06/23/2012 12:42 AM - trans (Thomas Sawyer)

@nikolai You are correct in part. I'm a big-time supporter of your previous proposal as well. The two don't completely overlap though. Using a self-extended module as top-level object also gives the top-level all the capabilities you would expect from any namespace.

For example, try defining a #method_added callback on top-level.

Or try simply seeing if a constant is defined:

```
Foo = 10
const_defined?(:Foo)
NoMethodError: undefined method `const_defined?' for main:Object
```

Or removing a method:

```
def x; end
remove_method(:x)
NoMethodError: undefined method `remove_method' for main:Object
```

etc.

#9 - 07/01/2012 08:20 AM - trans (Thomas Sawyer)

- File 6609.pdf added

Here is slide for this feature.

#10 - 07/02/2012 02:52 AM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

Received, thank you!

This proposal seems to require careful case study.
So I guess it is difficult for matz to accept it without sufficient thought.
But it will be valuable to know his impression.

--
Yusuke Endoh mame@tsg.ne.jp

#11 - 10/27/2012 07:11 AM - ko1 (Koichi Sasada)

- Assignee changed from matz (Yukihiro Matsumoto) to mame (Yusuke Endoh)

Could you tell us the result of dev-meeting about it and judge this ticket?

#12 - 10/27/2012 11:39 AM - mame (Yusuke Endoh)

- Assignee changed from mame (Yusuke Endoh) to nobu (Nobuyoshi Nakada)

Sorry for my very late reply.

We thought your slide included multiple requests.
Some were accepted, and others rejected.

- Matz rejected "module Main". He said that "Toplevel pollution" (you are saying) is actually designed.
- Matz accepted "define_method in Toplevel". We need a patch. Nobu, could you?
- Matz rejected "const_defined? in Toplevel". He said he couldn't understand why it is needed.
- If you want another method, please propose them individually.

--
Yusuke Endoh mame@tsg.ne.jp

#13 - 10/28/2012 06:39 AM - trans (Thomas Sawyer)

> We thought your slide included multiple requests.

Basically there were only two. It might seem like more b/c it is comprehensive in scope, covering many details in one go, i.e. Toplevel as a self extended module would provide #define_method, #const_defined?, #instance_methods, etc. The only request separate from this is for a non-polluting toplevel.

> Matz rejected "module Main". He said that "Toplevel pollution" (you are saying) is actually designed.

I would like to understand this design then. In my experience it has only served to limit what I can achieve, and gained me no additional benefit.

> - Matz accepted "define_method in Toplevel". We need a patch. Nobu, could you?
> - Matz rejected "const_defined? in Toplevel". He said he couldn't understand why it is needed.
> - If you want another method, please propose them individually.

That really misses the point. Oh well.

#14 - 10/28/2012 06:39 AM - trans (Thomas Sawyer)

> We thought your slide included multiple requests.

Basically there were only two. It might seem like more b/c it is comprehensive in scope, covering many details in one go, i.e. Toplevel as a self extended module would provide #define_method, #const_defined?, #instance_methods, etc. The only request separate from this is for a non-polluting toplevel.

> Matz rejected "module Main". He said that "Toplevel pollution" (you are saying) is actually designed.

I would like to understand this design then. In my experience it has only served to limit what I can achieve, and gained me no additional benefit.

> - Matz accepted "define_method in Toplevel". We need a patch. Nobu, could you?
> - Matz rejected "const_defined? in Toplevel". He said he couldn't understand why it is needed.
> - If you want another method, please propose them individually.

That really misses the point. Oh well.

#15 - 10/30/2012 03:59 AM - nobu (Nobuyoshi Nakada)

Hi,

At Sat, 27 Oct 2012 11:39:04 +0900,
mame (Yusuke Endoh) wrote in [ruby-core:48433]:

> - Matz accepted "define_method in Toplevel". We need a patch. Nobu, could you?

Sorry, I haven't followed this topic. To where should the defined method belong, Object?

--
Nobu Nakada

#16 - 10/30/2012 08:23 PM - mame (Yusuke Endoh)

2012/10/30, Nobuyoshi Nakada nobu@ruby-lang.org:

> At Sat, 27 Oct 2012 11:39:04 +0900,
> mame (Yusuke Endoh) wrote in [ruby-core:48433]:
>
> > - Matz accepted "define_method in Toplevel". We need a patch. Nobu, could you?
>
> Sorry, I haven't followed this topic. To where should the defined method belong, Object?

Or Kernel? Left to you.

--

Yusuke Endoh mame@tsg.ne.jp

#17 - 11/02/2012 02:48 PM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r37417.
Thomas, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

---

proc.c: main.define_method

- proc.c (top_define_method): new method, main.define_method. [ruby-core:45715] [Feature #6609]

Files

6609.pdf	77 KB	07/01/2012	trans (Thomas Sawyer)