Ruby trunk - Feature #1844

Immediates Should Not Respond to :dup

07/31/2009 04:15 AM - runpaint (Run Paint Run Run)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	2.0.0	

Description

=begin
Immediate can't be dup'd but they :respond_to?(:dup). This leads to ugly logic for determining whether a value can be dup'd. I suggest that immediates return false for :respond_to?(:dup). This is consistent with methods that would raise a NotImplementedError returning false for #respond_to?.
=end

History

#1 - 07/31/2009 06:16 AM - shyouhei (Shyouhei Urabe)

=begin
This is a bit radical opinion, I admit.

From my point of view, "determining" whether an object has a method is a wrong idea.
Instead you should just call that method. Which is:

```
begin
obj.method
rescue
...
end
```

instead of:

```
if obj.respond_to? :method then
obj.method
else
...
end
```

Why you should avoid respond_to? is that it is nothing special; just a method which can be overridden. So respond_to? is sometimes not realiable for various reasons. If you want to make sure, the safest way is to see if it actually quacks like a duck.
=end

#2 - 07/31/2009 08:56 AM - hongli (Hongli Lai)

=begin
I disagree.

- Exceptions are expensive.
- Rescuing a specific exception requires a multi-line statement, while respond_to? allows one to write "bar.foo if bar.respond_to?(:foo)"
- How do you know the exception was raised by the method itself and not by a method called by the method?

You say respond_to? is not always reliable. I think it should be reliable.
=end

#3 - 07/31/2009 09:01 AM - bitsweat (Jeremy Daer)

=begin
I agree that simply calling the method is ideal. However, rescuing an exception for such a common case may be very expensive in an inner loop.

Rails introduced Object#duplicable? for this reason:
http://github.com/rails/rails/blob/2c2ca833a531d825d9b46e501b564a52a8a69358/activesupport/lib/active_support/core_ext/object/duplicable.rb

So you say:

```
if obj.duplicable?
obj.dup
else
...
```

end

I would prefer that dup simply returned self for immediate objects, though. It's almost always what I'd prefer.
=end

#4 - 07/31/2009 09:07 AM - bitsweat (Jeremy Daer)

=begin
Hongli, respond_to? may not be reliable in the case of proxy objects. Also, calling respond_to? adds an implicit API requirement that simply calling the method does not. For example: http://github.com/rails/rails/commit/78af2710695973bbd747738d175fb3b1f488df6c
=end

#5 - 07/31/2009 09:46 AM - shyouhei (Shyouhei Urabe)

=begin
Well, I was about to write the case of Rake::TaskArguments :p I agree that respond_to? should be reliable, but doing wrong here is fairly easy, especially when you write your method_missing to tweak method dispatches.

Anyway when getting back to :dup story, I'm not against to make it more "reliable". But there seems to be several menu here:

- respond_to?(:dup) to be false and calling dup to raise exception. Runpaint's original suggestion.
- respond_to?(:dup) to be true and dup to return self. Jeremy's idea.

Maybe we need some more discussion for it?
=end

#6 - 07/31/2009 10:03 AM - dblack (David Black)

=begin
Hi --

On Fri, 31 Jul 2009, Hongli Lai wrote:

> Issue #1844 has been updated by Hongli Lai.
>
> I disagree.
>
> - Exceptions are expensive.
> - Rescuing a specific exception requires a multi-line statement, while respond_to? allows one to write "bar.foo if bar.respond_to?(:foo)"

But you probably wouldn't do:

```
x = y.dup if y.respond_to?(:dup)
```

because if it didn't, you'd end up with nil.

> - How do you know the exception was raised by the method itself and not by a method called by the method?

That's always a risk with exceptions, though, isn't it?

> You say respond_to? is not always reliable. I think it should be reliable.

I think that the exceptions and their messages can be useful in cases
like:

```
(0.0..10.0).each {} # TypeError: can't iterate from float
```

as opposed to having float ranges not respond to #each, and having to
ask each range whether it responds to #each. I tend to see the
immediate.dup thing as similar to that.

David

--
David A. Black / Ruby Power and Light, LLC / http://www.rubypal.com
Q: What's the best way to get a really solid knowledge of Ruby?
A: Come to our Ruby training in Edison, New Jersey, September 14-17!
Instructors: David A. Black and Erik Kastner
More info and registration: http://rubyurl.com/vmzN

=end

#7 - 07/31/2009 10:10 AM - dblack (David Black)

=begin
Hi --

On Fri, 31 Jul 2009, Shyouhei Urabe wrote:

> Issue #1844 has been updated by Shyouhei Urabe.
>
> Anyway when getting back to :dup story, I'm not against to make it
> more "reliable". But there seems to be several menu here:
>
> - respond_to?(:dup) to be false and calling dup to raise exception. Runpaint's original suggestion.
> - respond_to?(:dup) to be true and dup to return self. Jeremy's idea.

My own difficulty with Jeremy's idea is that 1 isn't a duplicate of 1.
In general, if x.dup returns x, it's not returning a duplicate of x,
so the method name becomes problematic.

In other words, I'm not sure about having dup be a no-op. On the other
hand, I would very much *not* like to start seeing #respond_to? every
time dup is used.

David

--
David A. Black / Ruby Power and Light, LLC / http://www.rubypal.com
Q: What's the best way to get a really solid knowledge of Ruby?
A: Come to our Ruby training in Edison, New Jersey, September 14-17!
Instructors: David A. Black and Erik Kastner
More info and registration: http://rubyurl.com/vmzN

=end

#8 - 07/31/2009 10:19 AM - tmat (Tomas Matousek)

=begin
What about adding Kernel#dup? method that is an alias of Kernel#dup: a class that cannot be duplicated would implement dup that throws an exception and dup? that returns nil or self?

Tomas

-----Original Message-----
From: David A. Black [mailto:dblack@rubypal.com]
Sent: Thursday, July 30, 2009 6:10 PM
To: ruby-core@ruby-lang.org
Subject: [ruby-core:24635] Re: [Bug #1844] Immediates Should Not Respond to :dup

Hi --

On Fri, 31 Jul 2009, Shyouhei Urabe wrote:

> Issue #1844 has been updated by Shyouhei Urabe.
>
> Anyway when getting back to :dup story, I'm not against to make it
> more "reliable". But there seems to be several menu here:
>
> - respond_to?(:dup) to be false and calling dup to raise exception. Runpaint's original suggestion.
> - respond_to?(:dup) to be true and dup to return self. Jeremy's idea.

My own difficulty with Jeremy's idea is that 1 isn't a duplicate of 1.
In general, if x.dup returns x, it's not returning a duplicate of x, so the method name becomes problematic.

In other words, I'm not sure about having dup be a no-op. On the other hand, I would very much *not* like to start seeing #respond_to? every time dup is used.

David

--
David A. Black / Ruby Power and Light, LLC / http://www.rubypal.com
Q: What's the best way to get a really solid knowledge of Ruby?
A: Come to our Ruby training in Edison, New Jersey, September 14-17!
Instructors: David A. Black and Erik Kastner

More info and registration: http://rubyurl.com/vmzN

=end

#9 - 07/31/2009 05:07 PM - godfat (Lin Jen-Shin)

=begin
There's try_dup in extlib [0], which returns self in various class,
i.e. NilClass, Symbol, TrueClass, FalseClass, Numeric, and Module.
I am not sure, but I think this is used to prevent from side-effect.

I am wondering, if what we want is not duplicate something,
but ensure there's no side-effect, why not use another method?
For example, dup_or_freeze [1] or so...

My two cents.

[0] http://github.com/datamapper/extlib/
[1] I know that immediates aree not frozen in the beginning.
=end

#10 - 08/02/2009 03:10 PM - runpaint (Run Paint Run Run)

=begin

> > - Exceptions are expensive.
> > - Rescuing a specific exception requires a multi-line statement, while respond_to? allows one to write "bar.foo if bar.respond_to?(:foo)"
>
> But you probably wouldn't do:
>
> x = y.dup if y.respond_to?(:dup)
>
> because if it didn't, you'd end up with nil.

The reason that I noticed this behaviour was because I had an Array that contained Fixnums, Floats, and Symbols, along with other objects. I wanted to #dup those that could be duplicated so I could process them safely. My instictual way to code this was ary.select{|e| e.respond_to?(:dup)}... Which failed.

That Rails and other libraries implement their own approaches to this problem suggests that it needs fixing in the core.

> > You say respond_to? is not always reliable. I think it should be reliable.
>
> I think that the exceptions and their messages can be useful in cases
> like:
>
> (0.0..10.0).each {} # TypeError: can't iterate from float
>
> as opposed to having float ranges not respond to #each, and having to
> ask each range whether it responds to #each. I tend to see the
> immediate.dup thing as similar to that.

I disagree. This isn't about a specific invocation of an object, as is the case for Range objects with Float values; it's whole classes that never allow #dup to be called claiming that it can be, then raising when it is. These are two rather distinct cases.

Either Jeremy's suggestion or fixing #respond_to? is fine by me.
=end

#11 - 08/03/2009 03:01 AM - dblack (David Black)

=begin
Hi --

On Sun, 2 Aug 2009, Run Paint Run Run wrote:

> I wrote:
>
> > I think that the exceptions and their messages can be useful in cases
> > like:
> >
> > (0.0..10.0).each {} # TypeError: can't iterate from float

> > as opposed to having float ranges not respond to #each, and having to
> > ask each range whether it responds to #each. I tend to see the
> > immediate.dup thing as similar to that.
>
> I disagree. This isn't about a specific invocation of an object, as
> is the case for Range objects with Float values; it's whole classes
> that never allow #dup to be called claiming that it can be, then
> raising when it is. These are two rather distinct cases.

Yes and no. There could be a FloatRange class that doesn't mix in
Enumerable, or something like that. It doesn't really matter how it's
implemented; the point is that you can't do:

```
ranges.each {|r| r.map ... }
```

with an array of ranges, without risking an exception, just as you
can't dup each of an array of arbitrary objects without that risk.

> Either Jeremy's suggestion or fixing #respond_to? is fine by me.

It's not really fixing #respond_to? (which is correctly reporting that
these objects can resolve the message "dup"). It would be a matter of
undef'ing #dup for certain classes. I have mixed emotions about it. On
the one hand, if (say) nil can't be duped, there's no really
compelling reason for it to respond to #dup. On the other hand,
there's a pretty common pattern of methods defined in Object that
exist mainly to be overridden (to_s, inspect, ==, ===, etc.). Usually
they're not overridden to raise an exception, but there's no
underlying reason why they shouldn't be.

David

--
David A. Black / Ruby Power and Light, LLC / http://www.rubypal.com
Q: What's the best way to get a really solid knowledge of Ruby?
A: Come to our Ruby training in Edison, New Jersey, September 14-17!
Instructors: David A. Black and Erik Kastner
More info and registration: http://rubyurl.com/vmzN

=end

#12 - 04/21/2010 09:43 PM - mame (Yusuke Endoh)

- Assignee set to matz (Yukihiro Matsumoto)

- Target version set to 2.0.0

=begin
Hi,

This is apparently not a bug but feature request.
I move this to Feature tracker.

--
Yusuke Endoh mame@tsg.ne.jp
=end

#13 - 04/21/2010 11:41 PM - RickDeNatale (Rick DeNatale)

=begin
On Thu, Jul 30, 2009 at 9:10 PM, David A. Black dblack@rubypal.com wrote:

> Hi --
>
> On Fri, 31 Jul 2009, Shyouhei Urabe wrote:
>
> > Issue #1844 has been updated by Shyouhei Urabe.
> >
> > Anyway when getting back to :dup story, I'm not against to make it
> > more "reliable". But there seems to be several menu here:
> >
> > - respond_to?(:dup) to be false and calling dup to raise exception. Runpaint's original suggestion.

> > - respond_to?(:dup) to be true and dup to return self. Jeremy's idea.
>
> My own difficulty with Jeremy's idea is that 1 isn't a duplicate of 1.
> In general, if x.dup returns x, it's not returning a duplicate of x,
> so the method name becomes problematic.

I agree with Jeremy.

The reason I dup an object is so that if I can change the state of the
duplicate without affecting the original object.

Since immediate objects are immutable, I can't change their state so
having dup return the original is not an issue.

Having to code around the possibility that dup will throw an error
seems worse than just having immediates just handle them in what seems
a reasonable fashion.

As a reference point, in Smalltalk the SmallInteger copy method
returns the receiver, this is a direct analogy to Fixnum#dup

--
Rick DeNatale

Blog: http://talklikeaduck.denhaven2.com/
Github: http://github.com/rubyredrick
Twitter: RickDeNatale (Rick DeNatale)
WWR: http://www.workingwithrails.com/person/9021-rick-denatale
LinkedIn: http://www.linkedin.com/in/rickdenatale

=end

#14 - 04/22/2010 05:11 AM - spatulasnout (B Kelly)

=begin

Rick DeNatale wrote:

> On Thu, Jul 30, 2009 at 9:10 PM, David A. Black dblack@rubypal.com wrote:
>
> > My own difficulty with Jeremy's idea is that 1 isn't a duplicate of 1.
> > In general, if x.dup returns x, it's not returning a duplicate of x,
> > so the method name becomes problematic.
>
> I agree with Jeremy.
>
> The reason I dup an object is so that if I can change the state of the
> duplicate without affecting the original object.

Agreed. I don't recall ever having dup'd an object to satisfy
an abstract desire to "create a duplicate". Rather, I use #dup
as a means to an end, and in my experience it's an end that would
be best served by having immediate objects simply return their
immutable selves.

I'm having trouble even inventing a contrived example where
having #dup succeed for immediates would lead to an undesirable
or unexpected result, in terms of how we would then put the
dup'd values to use.

I don't find it useful to adhere extremely rigidly to the
dictionary definition of the word 'dup', as i don't use #dup for
the end purpose of making duplicates, but to ensure that potential
modifications to the dup'd object won't modify the original.

Having #dup succeed for immediate objects would match my intent,
and if pressed about adherence to the definition of the word 'dup',
I'd want to call this a case of DWIMNWIS. (Do What I Mean Not What
I Say.)

Regards,

Bill

=end

#15 - 09/14/2010 04:25 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

=begin

=end

#16 - 02/14/2012 10:16 PM - mame (Yusuke Endoh)

- Status changed from Assigned to Rejected

I'm rejecting this feature ticket because no progress has been made for a long time. See [ruby-core:42391].

My personal opinion.

I might miss somthing, but I cannot understand the motivation well. When we duplicate an object, we then usually modify the object, don't we? To modify them, we must eventually check the type. I wonder why this feature is needed.

It makes no sense (to me) to compare a method that raises NotImplementedError with Immediate.dup. They are different.
In an extreme case, Kernel#raise always raise an exception.
Do you think respond_to?(:raise) should return false?

--
Yusuke Endoh mame@tsg.ne.jp