Ruby trunk - Feature #8377

Deprecate :: for method calls in 2.1

05/07/2013 07:49 PM - charliesome (Charlie Somerville)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:	Next Major	

Description

=begin
(({::})) is usually a constant lookup operator, but it can also be used to call methods. This can confusing to people learning Ruby.

I propose deprecating (({::})) as a method call operator in Ruby 2.1, then removing it in 2.2 (or whichever version comes after 2.1).

As part of the deprecation, Ruby's parser should emit a warning whenever (({::})) is used as a method call operator. This warning should be emitted even if (({-w})) is not enabled.
=end

History

#1 - 05/08/2013 03:35 AM - Hanmac (Hans Mackowiak)

i am against that, what about methods that are defined in Kernel, but you want when you are inside an BasicObject?
you need something like:
Kernel::Array([])
or
Kernel.Array([])
what does make more sense for you?

#2 - 05/08/2013 05:03 AM - Eregon (Benoit Daloze)

Hanmac (Hans Mackowiak) wrote:

> i am against that, what about methods that are defined in Kernel, but you want when you are inside an BasicObject?
> you need something like:
> Kernel::Array([])
> or
> Kernel.Array([])
> what does make more sense for you?

You need :: before Kernel, unless you define const_get
::Kernel::Array(3)
And in that case I much prefer
::Kernel.Array(3)
even if it does not look as nice because at least is is clear :: is constant resolution.

This is a weird case due to the upper case first letter, what do you think about:
::Kernel::puts "Hello"
versus
::Kernel.puts "Hello"

#3 - 05/08/2013 08:53 AM - Anonymous

On Wednesday, 8 May 2013 at 4:35 AM, Hanmac (Hans Mackowiak) wrote:

> i am against that, what about methods that are defined in Kernel, but you want when you are inside an BasicObject?
> you need something like:
> Kernel::Array([])
> or
> Kernel.Array([])
> what does make more sense for you?

I actually prefer the '::Kernel.Array' example here (although I can see both sides of the argument), just because it makes it perfectly obvious that 'Array' is actually a method call and not something else.

On Wednesday, 8 May 2013 at 6:03 AM, Eregon (Benoit Daloze) wrote:

> This is a weird case due to the upper case first letter, what do you think about:
> ::Kernel::puts "Hello"
> versus
> ::Kernel.puts "Hello"

I definitely think '::Kernel.puts' is better here.

#4 - 05/08/2013 08:56 AM - henry.maddocks (Henry Maddocks)

charliesome (Charlie Somerville) wrote:

> (({::})) is usually a constant lookup operator, but it can also be used to call methods.

Is it? I thought it was the scope resolution operator.

charliesome (Charlie Somerville) wrote:

> This can confusing to people learning Ruby.

It depends what language you're coming from.

#5 - 05/08/2013 09:57 AM - Anonymous

+1

#6 - 05/08/2013 11:55 AM - phluid61 (Matthew Kerwin)

=begin
henry.maddocks (Henry Maddocks) wrote:

> charliesome (Charlie Somerville) wrote:
>
> > (({::})) is usually a constant lookup operator, but it can also be
> > used to call methods.
>
> Is it? I thought it was the scope resolution operator.

While I know better, my gut instinct is always to agree; and that, by contrast, (({.})) is for the receiver of a method. For example:

```
::Kernel::Array(3)
#=> in global namespace,
#     in nested namespace 'Kernel',
#       invoke method 'Array' with no receiver
# perlish: @::Kernel::Array(3)

::Kernel.Array(3)
#=> in global namespace,
#     in nested namespace 'Kernel',
#       invoke method 'Array' with 'Kernel' as receiver
# perlish: @::Kernel->Array(3), or @::Kernel::Array(::Kernel,3)
```

However I know that's not the case; and in fact (({::})) means different things if the right-hand parameter thingy is a constant or function. I.e. (({obj.foo})) ~ (({obj::foo()})) ~ (({obj::foo})), but (({obj.FOO})) ~ (({obj::FOO()})) ≠ (({obj::FOO})), irrespective of ((|obj|)) being a Class or not.

The whole thing would be made much more clear if (({::FOO()})) and (({::foo})) were removed, and (({::})) was always only used to resolve constants.

> charliesome (Charlie Somerville) wrote:
>
> > This can confusing to people learning Ruby.
>
> It depends what language you're coming from.

Unless you're coming from Ruby, I'm pretty sure it's confusing for everyone.

=end

#7 - 05/08/2013 06:23 PM - yorickpeterse (Yorick Peterse)

I'm in favour of deprecating :: for method calls as well. I'm all for "multiple roads to Rome" but when it comes to the method calling syntax I really want it to be consistent.

Actually getting rid of the operator is going to be interesting though since I believe it's used in various places of MRI's core/std library as well as third-party projects. This is probably something that shouldn't be removed until at least 2.2.

Yorick

#8 - 05/08/2013 11:42 PM - dgutov (Dmitry Gutov)

+1

"::" method call syntax has no advantages over ".", and it should be relatively simple to search through any codebase and convert the instances of using the former into the latter automatically.

#9 - 05/09/2013 04:13 AM - bozhidar (Bozhidar Batsov)

+1

Apart from having no advantages over ".", "::" for method calls is very rarely used even anyways. It has been effectively deprecated by the Ruby community and it now has to be deprecated by Ruby itself :-)

#10 - 05/09/2013 04:13 AM - rubiii (Daniel Harrington)

I'm also very much in favour of this change for simplicity.

When I learned about Nokogiri, I had no idea how this was supposed to work: Nokogiri::XML('')
Simply changing the code to use a '.' instead of '::' makes it clear, that we're sending the '.XML' message.
So why not encourage people to write code that is easier to read?!

#11 - 05/09/2013 05:35 PM - henry.maddocks (Henry Maddocks)

phluid61 (Matthew Kerwin) wrote:

> =begin
> henry.maddocks (Henry Maddocks) wrote:
>
> > charliesome (Charlie Somerville) wrote:
> > It depends what language you're coming from.
>
> Unless you're coming from Ruby, I'm pretty sure it's confusing for everyone.
>
> =end

C++ and PHP also use the scope resolution operator.

As for the rest of what you said, I don't understand.

#12 - 05/09/2013 05:54 PM - henry.maddocks (Henry Maddocks)

rubiii (Daniel Harrington) wrote:

> I'm also very much in favour of this change for simplicity.
>
> When I learned about Nokogiri, I had no idea how this was supposed to work: Nokogiri::XML('')
> Simply changing the code to use a '.' instead of '::' makes it clear, that we're sending the '.XML' message.
> So why not encourage people to write code that is easier to read?!

But '.' and '::' mean different things.
'::' means you are calling the method that is defined inside the Nokogiri module/namespace. Changing it to use a dot means that you are sending a message to the object Nokogiri.

#13 - 05/09/2013 05:59 PM - yorickpeterse (Yorick Peterse)

Although in PHP you can also abuse the :: for both static and non static method calls it's actually not used for namespaces (unlike Ruby), instead it uses backslashes for that. Although this means that there's still two ways to call a method (syntax wise) you at least don't end up using a syntax feature meant for something completely unrelated to

method calls.

PHP is also a terrible language so I don't think it's worth comparing to
at all.

Yorick

#14 - 05/09/2013 06:52 PM - charliesome (Charlie Somerville)

henry.maddocks (Henry Maddocks) wrote:

> But '.' and '::' mean different things.
> '::' means you are calling the method that is defined inside the Nokogiri module/namespace. Changing it to use a dot means that you are sending a message to the object Nokogiri.

This isn't correct. Nokogiri.XML() and Nokogiri::XML() are equivalent.

#15 - 05/09/2013 10:13 PM - matz (Yukihiro Matsumoto)

- we haven't reached consensus to remove double colons for method calls from the language.
- even if we do, 2.2 is not the right time to remove, maybe 3.0.
- we have convention of Array class and Array(obj) conversion method, why not Foo::Bar and Foo::Bar(obj)?

Matz.

#16 - 05/09/2013 11:11 PM - dgutov (Dmitry Gutov)

> we haven't reached consensus to remove double colons for method calls from the language.

That's what this issue is about, isn't it?

> even if we do, 2.2 is not the right time to remove, maybe 3.0.

I agree. But deprecating it would already be beneficial.

> we have convention of Array class and Array(obj) conversion method, why not Foo::Bar and Foo::Bar(obj)?

Foo.Bar(obj) is much better. Writing it another way makes it confusing for a human reader, like it's somehow a special, callable class.

#17 - 05/10/2013 12:50 AM - headius (Charles Nutter)

More cases of ambiguity:

obj::FOO is always a constant lookup. obj::FOO() is always a method call.

obj::foo is always a method call. There's no way to omit parens for a :: call if the method name is capitalized.

I support removal as well. It doesn't add anything and it confuses more often than not.

#18 - 05/10/2013 01:22 AM - jeremyevans0 (Jeremy Evans)

headius (Charles Nutter) wrote:

> More cases of ambiguity:
>
> obj::FOO is always a constant lookup. obj::FOO() is always a method call.
>
> obj::foo is always a method call. There's no way to omit parens for a :: call if the method name is capitalized.

obj::FOO 1 is a method call. You don't even need parens for a capitalized method that takes no arguments if you get creative: obj::FOO *[] :)

#19 - 05/10/2013 01:33 AM - alexeymuranov (Alexey Muranov)

This is related and has not yet been rejected: #6806

#20 - 05/12/2013 12:12 AM - jballanc (Joshua Ballanco)

=begin
I'm not sure how I feel about this. I understand the potential for confusion, but at the same time there's a certain congruity of being able to inherit from classes or methods that create classes in the same way. For example:

```
module Foo
  class Bar
    def say
      "Hello from Bar!"
    end
  end

  def self.Bar(greeting)
    klass = Class.new
    klass.class_eval <<-END
      def say
        "#{greeting} from parameterized Bar!"
      end
    END
    klass
  end
end

class Baz < Foo::Bar
  def shout
    say.upcase
  end
end

class Quux < Foo::Bar("Howdy")
  def shout
    say.upcase
  end
end

puts Baz.new.shout
puts Quux.new.shout
```

If we remove the ability to call methods with (({::})), then the class definition lines don't match as nicely:

```
class Baz < Foo::Bar
...
class Quux < Foo.Bar("Howdy")
...
```

Though I'd be interested to hear Mr. Evans opinion, since I think Sequel is where I've seen this used to the greatest effect...
=end

#21 - 05/12/2013 02:11 AM - jeremyevans0 (Jeremy Evans)

jballanc (Joshua Ballanco) wrote:

> If we remove the ability to call methods with (({::})), then the class definition lines don't match as nicely:
>
> ```
> class Baz < Foo::Bar
> ...
> class Quux < Foo.Bar("Howdy")
> ...
> ```
>
> Though I'd be interested to hear Mr. Evans opinion, since I think Sequel is where I've seen this used to the greatest effect...

I'm against removing it, since I think there are places where the syntax looks nicer with :: (constructors such as Sequel::Model() and Nokogiri::XML()). Having only one way to do something for its own sake is not the ruby way, and I think the loss of backwards compatibility and nicer syntax outweighs the reduced confusion. I don't train new ruby programmers, though, so maybe I underestimate the confusion this causes.

#22 - 05/14/2013 04:23 AM - judofyr (Magnus Holm)

On Sat, May 11, 2013 at 7:11 PM, jeremyevans0 (Jeremy Evans) <
merch-redmine@jeremyevans.net> wrote:

> Issue #8377 has been updated by jeremyevans0 (Jeremy Evans).
>
> jballanc (Joshua Ballanco) wrote:
>
> > If we remove the ability to call methods with (({::})), then the class

> > definition lines don't match as nicely:
> >
> > ```
> > class Baz < Foo::Bar
> > ...
> > class Quux < Foo.Bar("Howdy")
> > ...
> > ```
> >
> > Though I'd be interested to hear Mr. Evans opinion, since I think Sequel is where I've seen this used to the greatest effect...
>
> I'm against removing it, since I think there are places where the syntax looks nicer with :: (constructors such as Sequel::Model() and Nokogiri::XML()). Having only one way to do something for its own sake is not the ruby way, and I think the loss of backwards compatibility and nicer syntax outweighs the reduced confusion. I don't train new ruby programmers, though, so maybe I underestimate the confusion this causes.

Can't you use use #[] for this?

class Quuz < Foo::Bar["Howdy"]
end

class User < Sequel::Model[:User]
end

doc = Nokogiri::XML[data]

#23 - 05/14/2013 07:24 AM - jeremyevans (Jeremy Evans)

On 05/14 04:01, Magnus Holm wrote:

> On Sat, May 11, 2013 at 7:11 PM, jeremyevans0 (Jeremy Evans) <
>
> > I'm against removing it, since I think there are places where the syntax looks nicer with :: (constructors such as Sequel::Model() and Nokogiri::XML()). Having only one way to do something for its own sake is not the ruby way, and I think the loss of backwards compatibility and nicer syntax outweighs the reduced confusion. I don't train new ruby programmers, though, so maybe I underestimate the confusion this causes.
>
> Can't you use use #[] for this?
>
> class Quuz < Foo::Bar["Howdy"]
> end
>
> class User < Sequel::Model[:User]
> end
>
> doc = Nokogiri::XML[data]

In the case of Sequel::Model, no, since Sequel::Model.[] is already defined and does something different than creating a subclass.

Usage of [] for constructors seems fairly uncommon (Hash.[] is the exception that comes to mind), so from a syntax level it is more likely to be confusing. Using capitalized methods for constructors is much more common in ruby (e.g. Kernel::{String,Array,Integer,Float,...}).

Jeremy

#24 - 06/02/2013 04:12 PM - zzak (Zachary Scott)

- Target version changed from 2.1.0 to Next Major

See [ruby-core:54883]

#25 - 08/11/2013 01:46 PM - tenderlovemaking (Aaron Patterson)

- Status changed from Open to Rejected

Hi,

Matz rejected this here:

https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20130809#Deprecate-for-method-calls-8377

So I'm closing this. :-)