Ruby trunk - Feature #14145

Proposal: Better Method#inspect

11/30/2017 10:42 AM - zverok (Victor Shepelev)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

The idea: When investigating (in example scripts, debugger or console) the library you are unfamiliar with, Ruby's reflection is very useful mechanism to understand "what it can": classes, modules, their constants, methods and so on.
I propose to expose a bit more information Ruby has internally in Method#inspect:

```
# before:
some_interesting_object.method(:foo) # => #<Method Klass#foo>
# after:
some_interesting_object.method(:foo)
# => #<Method Klass#foo(first_arg, *other_args, keyword_arg:)>
```

Dead-naive implementation:

```
class Method
  def signature
    recv = case receiver
    when Module
      "#{receiver.name}."
    else
      "#{receiver.class}#"
    end
    parameters.map.with_index { |(type, name), i|
      case type
      when :req then "#{name || "param#{i+1}"}"
      when :opt then "#{name || "param#{i+1}"} = <default>"
      when :keyreq then "#{name || "kw#{i+1}"}:"
      when :key then "#{name || "kwparam#{i+1}"}: <default>"
      when :rest then "*#{name || "rest"}"
      when :keyrest then "**#{name || "kwrest"}"
      end
    }.join(', ').prepend("#{recv}#{name}(") << ")"
  end

  def inspect
    "#<#{self.class.name} #{signature}>"
  end
end
```

This works "sub-optimal" for methods implemented in C, yet pretty decently for Ruby-implemented methods:

```
# C method, default param names
[1,2,3].method(:at)
# => #<Method Array#at(param1)>

# Ruby method, proper param names
CGI.method(:escape)
# => #<Method CGI.escape(string)>
Addressable::URI.method(:parse)
# => #<Method Addressable::URI.parse(uri)>
Addressable::URI.method(:join)
 => #<Method Addressable::URI.join(*uris)>

# We can't extract default values, but at least we can say they are there
Addressable::URI.method(:heuristic_parse)
# => #<Method Addressable::URI.heuristic_parse(uri, hints = <default>)>
```

If the proposal is accepted, I am ready to implement it properly in C (for all callable objects: Method, UnboundMethod, Proc)

Related issues:

Related to Ruby trunk - Feature #14111: ArgumentError□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ **Open**

History

#1 - 11/30/2017 11:55 AM - zverok (Victor Shepelev)

- Description updated

#2 - 11/30/2017 12:23 PM - shevegen (Robert A. Heiler)

I have nothing at all against better diagnostics, introspection etc...

However had, in that particular case that you showed above, I would personally not want it, even though I normally am always in favour of introspection. But to each their own - I have no problem if others can use it. I should also note that my contra this is only mild, so I ultimately would not have any real big objection to it either.

My proposal would be to allow this to be customized or customizable, so that ruby hackers can decide on their own what they prefer. (e. g. to use your new variant, or to default to the old variant; It's perfectly fine if we can use what we prefer to have IMO.)

One could use symbols to toggle the behaviour such as via:

:simple
:expanded

Where the latter may refer to the display that you described and :simple may be the default. Although I assume that your proposal would mean to also change the default to your variant, rather than retain the old behaviour. :)

But I guess in most ways, it really depends on the personal preferences of the ruby hacker. A good example how to solve this is one of my favourite gems of all times, the did-you-mean gem. The toggle situation is to ... either use the gem - or to uninstall it. :)

That way people can decide which variant they prefer.

Anyway, I don't want to make your suggestion too complex, since it really is simple - you propose a different default display purpose, which is an ok suggestion to make. In general it would be nice if ruby could allow for more customization and fine tuning of its internal behaviour or parts; see also Martin's proposal to have unicode-related stuff be easier debuggable; see also other suggestions to make threads more easily debuggable (by default) and so on and so forth. So I guess we can say that many people would like to have more control, better messages, different messages etc... - I guess the ruby core team has some things to ponder about towards the path of ruby 3.x with all of that. :)

PS: Perhaps for testing purposes, the above could be enabled via a configure switch for post ruby 2.5.x, so that people could test it, and give early feedback, and then also provide more feedback here. The reason I mention this is because ruby 2.5.x changed a few things in regards to exceptions and messages, such as underlining the error, changing the stack trace display, etc... compared to ruby 2.4.x and I think people may need some days or a few weeks to adjust to it. Even adjusting to the did-you-mean gem took me a while. :)

#3 - 11/30/2017 02:39 PM - Hanmac (Hans Mackowiak)

zverok (Victor Shepelev) some changes for your code:
use Method#owner to get the class/module where the method is defined
that can make a difference for Singleton methods on a non-module object

also use with_index(1) instead of the i+1 in your code

Thema:

i think this change would be okay, but i am indifferent about the C-level funtions

for my bindings i do overload the methods depending on the type and amount of parameters
no need to show more of the code but this is the documenation block

```
/*
 * call-seq:
 *   contains?(x, y) -> bool
 *   contains?(point) -> bool
 *   contains?(rect) -> bool
 *
 * return true if the point or the rect is (not strictly) inside this rect
 * ===Arguments
 * * x and y are Integer
 * * pos is a WX::Point
 * * rect is a WX::Rect
 * ===Return value
 * bool
 *
 */
```

as you guys can see there are different cases,
because how ruby defines methods and use rb_scan_args,

my function returns parameters also just [:rest]

for most Ruby C-Api functions, there might be a way to make parameters return something useful,
but that's probably not the case for all functions

#4 - 11/30/2017 03:41 PM - zverok (Victor Shepelev)

shevegen (Robert A. Heiler)

> However had, in that particular case that you showed above, I would personally not want it.

Just curious: why? I mean, I'd understood "Don't think I need it", but to disgust it so much to propose having options for it -- why?.. Is it too verbose and therefore somehow painfully to read or what?.. If it would be introduced, how your life'll become worse?

#5 - 12/04/2017 07:54 AM - ko1 (Koichi Sasada)

I don't have strong opinions on your proposal, but source location information seems also helpful. Which is important or both?

#6 - 12/04/2017 08:15 AM - Hanmac (Hans Mackowiak)

ko1 (Koichi Sasada) wrote:

> I don't have strong opinions on your proposal, but source location information seems also helpful. Which is important or both?

i didn't looked at the ruby source code for now, but might it be possible to implement source location for C-Api functions?
i don't care much about the line, but the so file would be cool

#7 - 12/04/2017 11:35 AM - zverok (Victor Shepelev)

ko1 (Koichi Sasada) wrote:

> I don't have strong opinions on your proposal, but source location information seems also helpful. Which is important or both?

I believe that source location is less necesary. I mean, what I had in mind was like "I am trying something in IRB/Debugger, and want to quickly experiment". This flow seems somehow natural to me:

```
obj = SomeNewLib.call_how_readme_shows
obj.class # => SomeClass
obj.methods # => [:foo, :bar]
obj.method(:bar) # => <Method #bar(limit, **options)>
# Oh!
obj.bar(3)
```

That's simple, same-context flow, which allows quick "pokin around".

Source location is NOT "same context" (you ask it to switch to the editor or fetch source some way), so for me it doesn't need to be "on the fingertips" (read: in #inspect output).

#8 - 10/27/2018 11:56 PM - guilhermereiscampos (Guilherme Reis Campos)

zverok (Victor Shepelev) wrote:

```
# We can't extract default values, but at least we can say they are there
Addressable::URI.method(:heuristic_parse)
# => #<Method Addressable::URI.heuristic_parse(uri, hints = <default>)>
```

I wonder why is not possible to extract the default values?

#9 - 10/28/2018 06:47 AM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote:

> I don't have strong opinions on your proposal, but source location information seems also helpful. Which is important or both?

I find source location (FILE:LINE) very valuable for debugging, e.g. to know if a method was overriden and to quickly check the source in my editor.

FWIW, TruffleRuby already shows the source location in Method#inspect.

```
$ truffleruby -e 'p 1.method :times'
#<Method: Integer(Integer)#times .../truffleruby/src/main/ruby/core/integer.rb:89>
```

#10 - 10/28/2018 09:10 AM - jwmittag (Jörg W Mittag)

guilhermereiscampos (Guilherme Reis Campos) wrote:

> zverok (Victor Shepelev) wrote:
>
> ```
> # We can't extract default values, but at least we can say they are there
> Addressable::URI.method(:heuristic_parse)
> # => #<Method Addressable::URI.heuristic_parse(uri, hints = <default>)>
> ```
>
> I wonder why is not possible to extract the default values?

Because they can be arbitrary Ruby code and Ruby is a Turing-complete language, which means that figuring out statically what the default value is, is equivalent to solving the Halting Problem. Additionally, Ruby is also capable of I/O and other side-effects, which means that the value can depend on external entities not visible to the documentation generator.

What would Method#inspect show for these:

```
def foo(a = while true do; end) end

def bar(a = Time.now) end

def baz(a = if rand < 0.5 then 23 else 'fourty-two' end) end

def qux(a = File.read('somefile.txt')) end
```

And what about this:

```
def crazy(a = FileUtils.rm_rf('/')) end
```

Would you expect Method#inspect to evaluate the default argument in order to be able to display its value?

Also, Method#inspect returns a String, but that loses all information about the type and structure of the default value:

```
class Foo
  def inspect; '42' end
end

def foo(a = Foo.new) end

def bar(a = 42) end

method(:foo).inspect
#=> #<Method: Object#foo(a = 42)>
```

```
method(:bar).inspect
#=> #<Method: Object#bar(a = 42)>
```

We could copy the source text used to define the default argument into the output of Method#inspect, but that would require the source code to be available at runtime, which is a massive memory overhead (one String for every optional parameter in the entire system, and there is no upper bound on the size of that String, since a default argument can be any arbitrarily large Ruby program). Plus, there is the additional complication that not all methods in a Ruby system even *have* (Ruby) source code.

#11 - 10/29/2018 12:43 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #14111: ArgumentError□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ added

#12 - 11/06/2018 11:36 PM - baweaver (Brandon Weaver)

It may be possible to procify the arguments to self-contain them without really disturbing their integrity. method(:name) already returns a proc, so would it be wholly unexpected that the arguments may do the same?

Thinking out loud about potential implementations as per your idea:

```
def foo(a = while true do; end) end

def bar(a = Time.now) end

def baz(a = if rand < 0.5 then 23 else 'fourty-two' end) end

def qux(a = File.read('somefile.txt')) end

method(:foo).default_values
=> { a: -> { while true do; end } }

method(:bar).default_values
=> { a: -> { Time.now } }

method(:qux).default_values
=> { a: -> { File.read('somefile.txt') } }

method(:baz).default_values
=> { a: -> { if rand < 0.5 then 23 else 'fourty-two' end } }
```

This works well because of non-overlapping argument names (as in you can't have argument a and keyword a)

Amusingly this could almost be used as a setter as well, as is the way with Hashes default values:

```
hash = {}

# value-based, same issue as with stated above, like the common `[]` issue
hash.default = 0

hash.default_proc = -> h, k { h[k] = [] }
```

So this is not unusual behavior as per Ruby standard implementation.

Consider a potential method implementation:

```
method(:foo).default_values[:a]
=>  -> { while true do; end }

method(:foo).default_values[:a] = -> { while false do; end }
```

This could also be named default_proc but would not be as clear. I think this could yield some very interesting code down the road.

#13 - 11/07/2018 07:06 AM - nobu (Nobuyoshi Nakada)

Ruby's default argument is not so simple as you think.

Where foo is defined as

```
def foo(a, b = a.bar) end
```

from where should method(:foo).default_values[:b] refer a?

#14 - 11/07/2018 08:35 AM - baweaver (Brandon Weaver)

Good point. I can't think of a good solution to that that would not involve substantial work.

nobu (Nobuyoshi Nakada) wrote:

> Ruby's default argument is not so simple as you think.
>
> Where foo is defined as
>
> ```
> def foo(a, b = a.bar) end
> ```
>
> from where should method(:foo).default_values[:b] refer a?

#15 - 01/10/2019 05:02 AM - matz (Yukihiro Matsumoto)

I agree with the concept of showing more information by Method#inspect. Patch welcome.

Matz.

#16 - 01/11/2019 01:49 AM - ko1 (Koichi Sasada)

Eregon (Benoit Daloze) wrote:

> ko1 (Koichi Sasada) wrote:
>
> > I don't have strong opinions on your proposal, but source location information seems also helpful. Which is important or both?
>
> I find source location (FILE:LINE) very valuable for debugging, e.g. to know if a method was overriden and to quickly check the source in my editor.

+1. I'm using #source_location like p method(:foo).source_location sometimes to know the definition. If "Method#inspect" shows this information, it will help *me*.

Also, Proc#inspect shows source location. p proc{} #=> #<Proc:0x00000298afcdb258@t.rb:1>

```
class C
  def m
  end
end

p C.new.method(:m) #=> #<Method: C#m@/home/ko1/src/ruby/trunk/test.rb:3 >
```

patch:

```
Index: proc.c
===================================================================
--- proc.c  (□□□□□ 66764)
+++ proc.c  (□□□□□)
@@ -2681,6 +2681,16 @@
     if (data->me->def->type == VM_METHOD_TYPE_NOTIMPLEMENTED) {
         rb_str_buf_cat2(str, " (not-implemented)");
     }
+
+    {
+        VALUE loc = rb_method_location(method);
+        if (!NIL_P(loc)) {
+            VALUE loc_str = rb_sprintf("@%"PRIsVALUE":%"PRIsVALUE" ",
+                                       RARRAY_AREF(loc, 0), RARRAY_AREF(loc, 1));
+            rb_str_buf_append(str, loc_str);
+        }
+    }
+
     rb_str_buf_cat2(str, ">");

     return str;
```

#17 - 03/11/2019 06:30 AM - ko1 (Koichi Sasada)

Discussion at dev-meeting https://bugs.ruby-lang.org/issues/15614:

- eregon and ko1 want to show source_location.
  - with parameters? or without parameters?
  - ko1: I think parameters are too long to show with source_location
  - matz: source_location is longer than parameters.

    - several people are positive to show both.
- znz: Proc#inspect shows source_lcoation, so I want to know it on Method, too.

other discussion:

- how about to show rdoc entry?
    - github url?
    - show source code?
    - extend by clicking is cool, like on Jupyter environment.
    - extensible with gem?
- really useful to know parameters?

#18 - 03/11/2019 03:20 PM - zverok (Victor Shepelev)

I'd like to address this line from the dev log:

> ko1: I'm against to show the name of parameters. we only need line no. doesn't it?
> matz: ask the original author.

I believe that I've already provided a use-case of my point of view, let me just repeat it:

> I mean, what I had in mind was like "I am trying something in IRB/Debugger, and want to quickly experiment". This flow seems somehow natural to me:

```
obj = SomeNewLib.call_how_readme_shows
obj.class # => SomeClass
obj.methods # => [:foo, :bar]
obj.method(:bar) # => <Method #bar(limit, **options)>
# Oh!
obj.bar(3)
```

That's simple, **same-context flow**, which allows quick "pokin around".

The source location is NOT "same context" (you ask it to switch to the editor or fetch source some way), so for me, it doesn't need to be "on the fingertips" (read: in #inspect output).

ko1 (Koichi Sasada) doesn't it sound reasonable to you?

Generally speaking I believe that #inspect's design goal is to provide enough, but not too much, information to understand "what is this object and how it can be used in the current context". I believe that <Method #bar(limit, **options)> follows this design goal -- "I know what I can do next with this object".

Next, I don't think "Method should have source location, because Proc has": for proc, it is typically the **only** way to tell one proc from another -- "it is a proc from that file", that's why "always visible" source location is important. For method, we have other means of identification and telling one from another (name, owner, receiver), and adding source location to it will make #inspect's output just less readable.

#19 - 03/13/2019 08:21 PM - Eregon (Benoit Daloze)

My understanding is #inspect shows useful information for debugging, so then the source location is very useful, and gives me more than the parameters (e.g., I can find the code, documentation, etc from a file and line).

I'm not against showing parameters too, and it's encouraging it can easily be done from the information already existing for Method#parameters (as shown in the issue description).
So I think having both is nice.

#20 - 03/14/2019 06:45 AM - duerst (Martin Dürst)

Eregon (Benoit Daloze) wrote:

> My understanding is #inspect shows useful information for debugging, so then the source location is very useful, and gives me more than the parameters (e.g., I can find the code, documentation, etc from a file and line).

Yes indeed. A flexible console can even be set so that it can automatically parse the source location information and open the relevant file at the relevant position. Very convenient.

> I'm not against showing parameters too, and it's encouraging it can easily be done from the information already existing for Method#parameters (as shown in the issue description).
> So I think having both is nice.

Agreed.