Ruby master - Feature #16832

Use #name rather than #inspect to build "uninitialized constant" error messages

05/06/2020 11:38 AM - byroot (Jean Boussier)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

While debugging a bug in Rails (https://github.com/rails/rails/pull/37632#issuecomment-623387954) I noticed NameError calls inspect on the const_get receiver to build its error message.

The problem is that some libraries such as Active Record have been redefining inspect for years to provide human readable information, e.g.:

```
>> Shipit::Stack.inspect
=> "Shipit::Stack (call 'Shipit::Stack.connection' to establish a connection)"
>> Shipit::Stack.connection; nil
>> Shipit::Stack.inspect
=> "Shipit::Stack(id: integer, environment: string, ...)"
```

Which makes for fairly unhelpful error messages:

```
>> Shipit::Stack.const_get(:Foo)
Traceback (most recent call last):
        2: from (irb):4
        1: from (irb):4:in `const_get'
NameError (uninitialized constant #<Class:0x00007fc8cadf2dd0>::Foo)
```

So perhaps it's Active Record that is at fault here, but from my understanding since the goal is to display the constant path that was requested, name is much more likely to return a relevant constant name.

Proposed patch: https://github.com/ruby/ruby/pull/3080

Associated revisions

Revision 7d5da30c - 05/26/2020 06:09 AM - nobu (Nobuyoshi Nakada)

Test for [Feature #16832]

History

#1 - 05/06/2020 01:01 PM - shevegen (Robert A. Heiler)

Nobu recommended creating an issue request here, so I won't comment on that in itself - but I would like to point out that ruby users may wonder why/if/how to use #name rather than #inspect, if applicable, so that should be considered as well. Some may already struggle with #to_s versus #to_str distinction; we should be considering this as well.

#2 - 05/06/2020 10:42 PM - Eregon (Benoit Daloze)

From your example, why don't we see uninitialized constant Shipit::Stack(id: integer, environment: string, ...)::Foo?

I feel calling name here is a hack, if people return a useless String for inspect that's the bug, inspect is meant to be helpful and human-readable.
Many exceptions use inspect on the receiver including NoMethodError, I don't think we should make that protocol more complicated.

#3 - 05/07/2020 06:57 PM - byroot (Jean Boussier)

> why don't we see uninitialized constant Shipit::Stack(id: integer, environment: string, ...)::Foo?

Because inspect is called with rb_protect and fail in this context, so Ruby fallback to rb_obj_as_string.

> I feel calling name here is a hack

That's fair, I'm personally on the fence and as I said in my ticket, perhaps this shouldn't change.

But my point here is that here Ruby is looking to provide the class path, in that context it makes sense to look at the name. You have similar checks for T_CLASS/T_MODULE in other parts of ruby, such as rb_profile_frame_classpath.

So IMHO it's worth considering.

#4 - 05/07/2020 07:03 PM - byroot (Jean Boussier)

> Because inspect is called with rb_protect and fail in this context, so Ruby fallback to rb_obj_as_string.

Actually, I'm really unsure about that one. It doesn't appear to fail. I'll try to dig the root cause.

#5 - 05/07/2020 07:20 PM - byroot (Jean Boussier)

Ok, so after digging into it, it's not caused by an error, but by the length on the string returned by #inspect:

```
module OK
  class << self
    def inspect
      "a" * 65
    end
  end
end

module BAD
  class << self
    def inspect
      "a" * 66
    end
  end
end

begin
  OK::A
rescue NameError => error
  p error.message
end

begin
  BAD::B
rescue NameError => error
  p error.message
end

"uninitialized constant aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa::A"
"uninitialized constant #<Module:0x00007fdc0c8561d8>::B"
```

It's actually in the code:

```
if (NIL_P(d) || RSTRING_LEN(d) > 65) {
    d = rb_any_to_s(obj);
}
```

#6 - 05/07/2020 07:30 PM - byroot (Jean Boussier)

I tried to look for the reason for this 65 size limit, but it was present in the initial commit dated from 1998, so I have no idea why this limitation is here.

#7 - 05/07/2020 07:40 PM - byroot (Jean Boussier)

Actually I found the older history. It was introduced in https://github.com/ruby/ruby/commit/554b989ba1623b9f6a0b76f00824c83a23fbcbc1 version 0.99.4-961224 (December 1996)

And I think the changelog is

```
* eval.c (f_missing): □□□□□□□□□□□□□□□□□□□□□□□□
  □□□□□□□□
```

Which according to Google Translate means:

> Buffer if the string representation of the object is too long

> Was overwritten

I'm not sure this limitation still make sense.

#8 - 05/07/2020 07:54 PM - Eregon (Benoit Daloze)

Right, name_err_mesg_to_str does this.
And TruffleRuby actually replicated that logic but I didn't think about it for this issue:
https://github.com/oracle/truffleruby/blob/e8270af20bc4fae74be7f78d3d474e271768b943/src/main/ruby/truffleruby/core/truffle/exception_operations.rb#L18

In the case we exceed the limit, I think we could call rb_class_name() for Module/Class, instead of rb_any_to_s().

Such an arbitrary limit seems weird, maybe we should extend it to at least 100?
Or just not limit at all and if people write a too large inspect then they'll just see the issue (my preference).

I guess maybe it's done that way so it avoids the printed error message to be longer than 80 chars in the terminal?
A more intuitive way to achieve that could be to print something like "#{obj.inspect[0, 65]} ..."

#9 - 05/07/2020 08:59 PM - byroot (Jean Boussier)

> In the case we exceed the limit, I think we could call rb_class_name() for Module/Class

IMO we should all that in priority over inspect as it's more likely to return an useful class path.

> Such an arbitrary limit seems weird, maybe we should extend it to at least 100?

I actually submitted a PR to remove it entirely: https://github.com/ruby/ruby/pull/3090

> I guess maybe it's done that way so it avoids the printed error message to be longer than 80 chars in the terminal?

No, according to the changelog it was to prevent a buffer overflow. But I don't think it's an issue anymore, I tested with 4000 chars and it doesn't crash.

#10 - 05/12/2020 03:29 AM - ko1 (Koichi Sasada)

https://github.com/ruby/ruby/pull/3090 was merged. Can we close this ticket? (using #name instead of #inspect).

#11 - 05/12/2020 07:01 AM - byroot (Jean Boussier)

https://github.com/ruby/ruby/pull/3090 is simply a weird limitation I discovered while studying that behavior.

I still think we should try to display the actual class path.

#12 - 05/12/2020 09:39 AM - Eregon (Benoit Daloze)

Nice work on that PR :)

I think whoever overrides #inspect should make sure it's easy to identify which object it is.
In other words, I don't think we should special-case here for Module & Class (also #name can be nil, and calling #name on arbitrary objects could easily be worse than #inspect).

#13 - 05/12/2020 09:53 AM - byroot (Jean Boussier)

> (also #name can be nil, and calling #name on arbitrary objects could easily be worse than #inspect).

So the issue title is no longer relevant. Since then I dug more into this issue, and I think the behavior should be similar to rb_profile_frame_classpath, in short try to use rb_class2name for T_MODULE and T_CLASS, see this PR for instance: https://github.com/ruby/ruby/pull/3084

I'll try to better explain my reasoning:

- #inspect semantic is to describe an object
- uninitialized constant error message tells you about a missing constant, constants are not objects, they are references to objects.
- So that message should try to tell you the name of the missing constant (AKA class path) rather than describe the object that was supposed to hold the constant.

The fact is, that message builder basically does "#{receiver.inspect}::#{missing_constant_name}", that joining :: indicates that it assumes receiver.inspect returned a class path.

#14 - 05/12/2020 03:33 PM - Dan0042 (Daniel DeLorme)

+1 for using #name and falling back to #inspect

Rails is correct in extending inspect to return more useful human-readable information, and a NameError should tell you about the problem name (including namespace), not about extra information regarding the module/class to which the namespace resolves. I feel that depending on #inspect to return the name here is a hack.

Ideally the NameError would contain the fully-qualified constant name actually used in the code:

```
class A; end
B = A
B::X #=> NameError (uninitialized constant A::X)
     # ideally would say constant B::X
```

but that's getting a bit out of scope for this ticket. #name is the next-best choice.

#15 - 05/12/2020 04:04 PM - Eregon (Benoit Daloze)

Right, I agree it makes sense for missing constants NameError's, since the user expect a "constant path" including the missing constant.

But it doesn't seem right for other NameError such as:

```
$ ruby -e foobar
Traceback (most recent call last):
-e:1:in `<main>': undefined local variable or method `foobar' for main:Object (NameError)
```

#16 - 05/12/2020 06:26 PM - byroot (Jean Boussier)

> But it doesn't seem right for other NameError

Eregon (Benoit Daloze) Absolutely, but my ticket is only about uninitialized constant. If my description or my PR made you think otherwise, it's a misunderstanding and / or a bug in my PR.

#17 - 05/14/2020 08:42 AM - matz (Yukihiro Matsumoto)

Sounds OK. Let's see how it works.

Matz.

#18 - 05/26/2020 06:09 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset git|7d5da30c9e9c572f6ef0aaccc1ca0e724966e2ee.

Test for [Feature #16832]

#19 - 05/29/2020 10:13 AM - Eregon (Benoit Daloze)

FWIW it also applies to missing methods and other NameError, which sounds nice but would deserve a spec in spec/ruby/core/exception/name_error_spec.rb:

```
$ ruby -e 'c=Class.new { def self.name; "MyClass"; end }; c.foo'
master:
-e:1:in `<main>': undefined method `foo' for MyClass:Class (NoMethodError)
2.7.1:
-e:1:in `<main>': undefined method `foo' for #<Class:0x000055dcc0cda620> (NoMethodError)
```

#20 - 05/29/2020 03:27 PM - byroot (Jean Boussier)

Eregon (Benoit Daloze) good catch, I'll submit a PR with extra specs early next week and tag you.