Ruby master - Feature #17122
Add category to Warning#warn
08/13/2020 08:42 PM - eileencodes (Eileen Uchitelle)

Status: Closed
Priority: Normal
Assignee: eileencodes (Eileen Uchitelle)
Target version:

Description
Deprecation warnings and other warnings in Ruby have a category (:deprecated, etc) but those categories aren't exposed or accessible. In the most recent Ruby 2.7 upgrade at GitHub we monkey patched Warning#warn to be able to turn warnings into exceptions. However, there was no way to tell which warnings were deprecations and which were other types of warnings.

I want to expose the category on the Warning module so that I'm able to monkey patch Warning#warn and treat deprecation warnings differently from other warnings without using a regex the strings.

Here's an example program demonstrating what I'd like to get from Ruby by implementing this feature:

```ruby
module Warning
  def self.warn(msg, category: nil)
    if category == :deprecated
      raise msg
    else
      super
    end
  end
end

def ivar
  Object.new.instance_variable_get(:ivar)
end

# Doesn't raise, but warns with verbose set
ivar

# Raises an error
Object.new.tainted?
```

The PR I worked on with @tenderlove is here: https://github.com/ruby/ruby/pull/3418

It moves the Warning module to be written in Ruby, updates rb_warning_s_warn to pass kwargs, and adds a category to Warning#warn.

Related issues:
- Related to Ruby master - Feature #11588: Implement structured warnings
- Related to Ruby master - Feature #12026: Support warning processor
- Related to Ruby master - Feature #12299: Add Warning module for customized wa...
- Related to Ruby master - Bug #17387: About Warning.warn compatibility in Ruby...

History

#1 - 08/13/2020 09:06 PM - jeremyevans0 (Jeremy Evans)
I don't think we should add a category keyword if the only usage is for deprecation warnings. If we are going to do this, I think:

- All warning messages emitted by the core and the stdlib should have a category added
  - This requires we agree on category symbols
  - This requires new C-API warning functions added that accept a category
- Kernel#warn should accept a category keyword and pass it to Warning.warn

Having Warning.warn accept any keywords will break existing versions of the warning gem, but I can update that if this is accepted.

I don't think the benefits of adding this outweigh the cost, but I'm not strongly opposed to it.

09/23/2021
#2 - 08/13/2020 09:23 PM - tenderlovenmaking (Aaron Patterson)

jeremyevans0 (Jeremy Evans) wrote in #note-1:

I don't think we should add a category keyword if the only usage is for deprecation warnings. If we are going to do this, I think:

- All warning messages emitted by the core and the stdlib should have a category added
  - This requires we agree on category symbols

I think we have category symbols already:

- https://github.com/ruby/ruby/blob/66efe373116d510c05d57964b5f5fa47f1c5e565c/error.c#L153-L168
- https://github.com/ruby/ruby/blob/66efe373116d510c05d57964b5f5fa47f1c5e565c/test/ruby/test_module.rb#L31-L32

But Warning.warn doesn't know what category the message is associated with.

- This requires new C-API warning functions added that accept a category

I think this might depend on how many categories we have. Right now we have specific C functions for the deprecated type:

https://github.com/ruby/ruby/blob/66efe373116d510c05d57964b5f5fa47f1c5e565c/error.c#L377-L405

- Kernel#warn should accept a category keyword and pass it to Warning.warn

Having Warning.warn accept any keywords will break existing versions of the warning gem, but I can update that if this is accepted.

I don't think the benefits of adding this outweigh the cost, but I'm not strongly opposed to it.

I think the benefit is that developers can tell which warnings they should really heed without resorting to string parsing and specific warning investigation. If you have the category, it's easy to test your app with Ruby version N and know what will break in version N+1 without resorting to string parsing.

Maybe we could agree that the format of "will be removed" messages is always the same, but that seems like a brittle contract.

#3 - 08/14/2020 02:39 AM - shyouhei (Shyouhei Urabe)

I guess we want to revisit #11588.

#4 - 08/14/2020 02:39 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #11588: Implement structured warnings added

#5 - 08/14/2020 02:39 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #12026: Support warning processor added

#6 - 08/14/2020 02:39 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #12299: Add Warning module for customized warning handling added

#7 - 08/18/2020 07:04 PM - eileenloves (Eileen Uchitelle)

I don't think we should add a category keyword if the only usage is for deprecation warnings. If we are going to do this, I think:

- All warning messages emitted by the core and the stdlib should have a category added
  - This requires we agree on category symbols
  - This requires new C-API warning functions added that accept a category
- Kernel#warn should accept a category keyword and pass it to Warning.warn

As Aaron mentioned, categories already have warnings. My use-case is for deprecations, but that doesn't mean that others won't want to change behavior for other types of warnings. I could see applications raising everything but :experimental or only raising for :deprecated.

The change I'm proposing isn't introducing categories, it's adding a way to interact with existing categories. Applications can already turn off deprecations completely with Warning[:deprecated] = false, but there's no way to interact the other way around. What we've implemented in this PR is being able to change application behavior of warnings in specific categories. All we're doing is exposing the category kwarg and making sure that deprecation warnings get tagged "deprecated".

I like this implementation because it's lightweight and utilizes existing behavior.

I guess we want to revisit #11588.
For the use cases outlined here I think that structured warnings is unnecessary and more complex than what we're looking to implement. One of the things we get out of this change is the ability to have Ruby 2.7 behave like Ruby 2.8 by raising on deprecation warnings instead of silently piping them to a log file.

We run GitHub deprecation warning free. We fixed over 11k kwargs warnings in our application and want to deprecation warning free until we're running on Ruby 2.8+. The best way for us to accomplish this is being able to raise exceptions if Ruby 2.7 sees a deprecation warning. We basically want Ruby 2.8 behavior in Ruby 2.7 - anything that will break in 2.8 we want to mimic being broken in 2.7 (and continue using this pattern for future N+1 Ruby versions). I think structured warnings are more than we need to improve the interaction with warnings in Ruby.

Note: Aaron and I fixed the tests in the PR. We ended up undoing the change that turned Warning into a Ruby module so now this PR only implements this feature.

#8 - 08/19/2020 04:05 AM - mame (Yusuke Endoh)
Hi Eileen,

First of all, thank you not only for contributing to Ruby but also for working hard on keyword argument separation for GitHub code base!

I think that this feature is reasonable, but I concern its compatibility.

```ruby
module Warning
  def self.warn(msg)
    puts msg
  end
end

Object.new.tainted?
 #=> 2.7: "t.rb:9: warning: Object#tainted? is deprecated and will be removed in Ruby 3.2.\n"
 #=> master with your patch: wrong number of arguments (given 2, expected 1) (ArgumentError)
```

2.7 works great, but master with your PR passes extra argument, which causes an error.
Jeremy says that he can update gem ("warning" gem), but I guess that this kind of code is used in other places.
Do you think if such a code is less often so that we can ignore?

In addition: your PR changed only rb_warn_deprecated_to_remove but not rb_warn_deprecated. Is there any reason?

#9 - 08/19/2020 08:27 AM - byroot (Jean Boussier)
I'd like to second this feature request. We used the same workflow in Shopify, and still have a bunch of regexps today to raise on deprecations that would break on 2.8. Having a simple category symbol would make all this much easier.

I don't have an opinion on the Warning.warn interface change, for us it wouldn't be a big deal as very few code out there do define this. Maybe it could be acceptable to do an arity check?

#10 - 08/19/2020 04:31 PM - mame (Yusuke Endoh)
I've added this ticket to the next dev-meeting agenda.

byroot (Jean Boussier) Yes, an arity check would work.

My another idea is to call Warning.warn_deprecated or Warning.warn_experimental before the general Warning.warn. I'm not sure if it is good enough, though.

#11 - 08/19/2020 05:52 PM - jeremyevans0 (Jeremy Evans)
mame (Yusuke Endoh) wrote in #note-10:

My another idea is to call Warning.warn_deprecated or Warning.warn_experimental before the general Warning.warn.

I prefer this approach unless we decide to add categories for all warnings.

#12 - 08/26/2020 06:35 AM - ko1 (Koichi Sasada)
If we can accept the compatibility issue, I support it.
However, I'm not sure the "category" is a last attribute, so to make stable API, accept **kw is one idea (but it increases runtime overhead).

#13 - 08/28/2020 02:54 AM - akr (Akira Tanaka)
I feel the original proposal, adding a category keyword argument, is natural.

Although it is incompatible, arity check (or parameter check), checking the arity of Warning.warn and don't give actual arguments more than the arity, can preserve compatibility.
This fact raises a question. Ruby has optional argument that formal argument which corresponding actual argument may not be exist. But Ruby don't have "ignorable" argument that actual argument which corresponding formal argument may not be exist.

Ignorable arguments is useful for compatibility of extension of callbacks. The ary check is a hack to implement it.

I feel arity check is not so problematic here. It just complement a feature Ruby lacks.

#14 - 08/31/2020 02:04 AM - akr (Akira Tanaka)
akr (Akira Tanaka) wrote in #note-13:

But Ruby don't have "ignorable" argument that actual argument which corresponding formal argument may not be exist.

Maeda-sensei (@maeda) told me that Common Lisp has allow-other-keys which suppress keyword argument checking at function call.

http://www.lispworks.com/documentation/HyperSpec/Body/03_dada.htm
http://www.lispworks.com/documentation/HyperSpec/Body/03_daf.htm

% clisp
   I i i i i i           ooooo o ooooooo ooooo ooooo
  I I I I I I           8 8 8 8 8 8 8 8
 I  \\ '+=' / I           8 8 8 8 8 8 8 8
 \  `-+-' /             8 8 8 ooooo Soooo
  '___:___':            8 8 8 8 8 8 8 8
    |                  8 o 8 8 8 o 8 8
  ------+++++++       ooooo Soooooo ooo8ooo ooooo 8

Welcome to GNU CLISP 2.49.92 (2018-02-18) <http://clisp.org/>  
Copyright (c) Bruno Haible, Michael Stoll 1992-1993  
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997  
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998  
Copyright (c) Bruno Haible, Sam Steingold 1999-2000  
Copyright (c) Sam Steingold, Bruno Haible 2001-2018  
Type :h and hit Enter for context help.  

[1]> (defun f (x &key y) (+ x y))  
F  
[2]> (f 1 :y 2)  
3  
[3]> (f 1 :y 2 :z 3 :allow-other-keys t)  
3  
[4]> (f 1 :y 2 :z 3)  
*** - F: illegal keyword/value pair :Z, 3 in argument list.  
The allowed keywords are (:Y) 
The following restarts are available:  
ABORT :R1 Abort main loop  
Break 1 [5]>  

#15 - 08/31/2020 06:06 AM - matz (Yukihiro Matsumoto)  
OK. Accepted. But we might need to add arity check before calling warning callback for compatibility's sake.  
Matz.  

#16 - 08/31/2020 03:22 PM - eileencodes (Eileen Uchitelle)  

OK. Accepted. But we might need to add arity check before calling warning callback for compatibility's sake.

Great, very exciting! I will work on implementing an arity check / fixing the incompatibility and fix the failing tests on my PR.
I've addressed the backwards compatibility on the PR ([https://github.com/ruby/ruby/pull/3418](https://github.com/ruby/ruby/pull/3418)) and added an entry to NEWS.md. Let me know if there are other changes <3

#18 - 09/02/2020 12:16 AM - hsbt (Hiroshi SHIBATA)
- Status changed from Open to Closed

Merged at [https://bugs.ruby-lang.org/projects/ruby-master/repository/git/revisions/6e8ec9ab6da228ade70fe7d0dd16540d8f859f00](https://bugs.ruby-lang.org/projects/ruby-master/repository/git/revisions/6e8ec9ab6da228ade70fe7d0dd16540d8f859f00)

#19 - 09/02/2020 06:36 AM - mame (Yusuke Endoh)
- Assignee set to eileencodes (Eileen Uchitelle)
- Status changed from Closed to Assigned

Hi eileencodes (Eileen Uchitelle)

mame (Yusuke Endoh) wrote in #note-8:

In addition: your PR changed only rb_warn_deprecated_to_remove but not rb_warn_deprecated. Is there any reason?

Could you add the same logic not only to rb_warn_deprecated_to_remove but also to rb_warn_deprecated? Thanks.

#20 - 09/02/2020 12:17 PM - eileencodes (Eileen Uchitelle)

Could you add the same logic not only to rb_warn_deprecated_to_remove but also to rb_warn_deprecated? Thanks.

Yup, I will send a PR!

#21 - 09/02/2020 01:40 PM - eileencodes (Eileen Uchitelle)
PR for rb_warn_deprecated opened here: [https://github.com/ruby/ruby/pull/3505](https://github.com/ruby/ruby/pull/3505)

#22 - 09/03/2020 03:30 AM - mame (Yusuke Endoh)
- Status changed from Assigned to Closed

[https://github.com/ruby/ruby/pull/3505](https://github.com/ruby/ruby/pull/3505) has been merged. Thank you Eileen!

#23 - 12/12/2020 12:47 PM - Eregon (Benoit Daloze)
- Related to Bug #17387: About Warning.warn compatibility in Ruby 3.0.0 added