Ruby master - Feature #17143

Improve support for warning categories

09/03/2020 12:04 AM - jeremyevans0 (Jeremy Evans)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Support was recently added for Warning.warn to accept a category keyword. However, the initial implementation was limited to having rb_warn_deprecated and rb_warn_deprecated_to_remove use the :deprecated value for the category keyword.

It doesn't make sense to me to have a category keyword if it is only used for deprecation, so I propose we extend the support so that Kernel#warn accepts a category keyword (for Ruby-level warnings) and rb_category_warn and rb_category_warning functions be added to the C-API (for C-level warnings). I also propose that we change existing rb_warn and rb_warning calls to rb_category_warn and rb_category_warning, so that all warnings issued by core Ruby are issued with an appropriate category.

I have implemented support for this in a pull request: https://github.com/ruby/ruby/pull/3508

Related issues:

Related to Ruby master - Feature #17055: Allow suppressing uninitialized inst...	**Closed**

History

#1 - 09/23/2020 04:57 AM - ko1 (Koichi Sasada)

Let me clear the proposal. Your PR seems to allow any categories (and any class of object). Is it intentional?

#2 - 09/23/2020 05:35 AM - jeremyevans0 (Jeremy Evans)

ko1 (Koichi Sasada) wrote in #note-1:

> Let me clear the proposal. Your PR seems to allow any categories (and any class of object). Is it intentional?

Allowing arbitrary categories is intentional. We probably should restrict the category to be a symbol. The C-API already does this (rb_category_warn{,ing} accept const char * and convert to symbol), so it would only need to be made for Kernel#warn (raising TypeError if :category is not a Symbol?). That's a simple change, and I'll try to make it tomorrow, before the developer meeting.

I'm fine with changing the implementation to limit the categories to specific categories and not allow arbitrary categories, if that is what is decided at the developer meeting. That reduces flexibility, but also reduces the chance of a typo resulting in a category being missed. Allowing arbitrary categories has the advantage that users can define their own categories of warnings and filter on them.

#3 - 09/23/2020 05:44 PM - ko1 (Koichi Sasada)

My concern is, arbitrary categories make filtering difficult on Warning.warn method.

Another concern is, "what is the category?" problem.
You added "file" category, but some warning should be "deprecated".
These two categories can be combine (file and deprecated).
Allow multiple categories?

#4 - 09/23/2020 06:01 PM - jeremyevans0 (Jeremy Evans)

ko1 (Koichi Sasada) wrote in #note-3:

> My concern is, arbitrary categories make filtering difficult on Warning.warn method.

Agreed. It's definitely a tradeoff. As I said, I'm fine disallowing arbitrary categories and will make the necessary changes if that is what is decided at the developer meeting.

> Another concern is, "what is the category?" problem.
> You added "file" category, but some warning should be "deprecated".
> These two categories can be combine (file and deprecated).
> Allow multiple categories?

I don't think we should allow multiple categories in a single warning. I'm fine changing any categories I used or combining any of the categories I

used into an existing category (e.g. file -> deprecated), and will make the necessary changes if that is what is decided at the developer meeting.

#5 - 09/25/2020 04:58 AM - matz (Yukihiro Matsumoto)

Adding category to the warning seems a good idea. But I have the following concerns:

- Category should be specified only by symbols
- Category symbols must be among predefined set (to avoid typos and confusions)

Matz.

#6 - 09/25/2020 05:04 AM - shyouhei (Shyouhei Urabe)

- I'm not against adding the category keyword. However,
- I guess it is quite hard for us to +1 the "Add categories for all core warnings" commit at once.
  - It might contain OK changes, but might also contain questionable ones.
  - Do you really need to do this in a big changeset like this? Can you split it into each categories, and let us consider case-by-case?

#7 - 09/25/2020 07:43 PM - jeremyevans0 (Jeremy Evans)

matz (Yukihiro Matsumoto) wrote in #note-5:

> Adding category to the warning seems a good idea. But I have the following concerns:
>
> - Category should be specified only by symbols

OK, this change has already been made.

> - Category symbols must be among predefined set (to avoid typos and confusions)

I'll make this change. I'll start with only :deprecated being allowed, since that is already used. Future commits can add support for other symbols.

shyouhei (Shyouhei Urabe) wrote in #note-6:

> - I'm not against adding the category keyword. However,
> - I guess it is quite hard for us to +1 the "Add categories for all core warnings" commit at once.
>   - It might contain OK changes, but might also contain questionable ones.
>   - Do you really need to do this in a big changeset like this? Can you split it into each categories, and let us consider case-by-case?

I can split the commit into separate commits per category. I'll focus on the categories I think are most important first and submit separate pull requests for those.

#8 - 09/25/2020 10:06 PM - jeremyevans0 (Jeremy Evans)

I've updated my pull request to only allow specific categories, and limit the currently allowed categories to :deprecated. Assuming it passes CI, I'll merge it, and then work on pull requests for additional categories.

#9 - 09/30/2020 03:12 PM - jeremyevans0 (Jeremy Evans)

I submitted the first pull request to add the :deprecated category to additional warnings, and to add warning categories for :uninitialized_ivar and :redefine: https://github.com/ruby/ruby/pull/3601

matz (Yukihiro Matsumoto) Do you want to approve all warning categories before they are merged, or are you comfortable delegating approval of warning categories to another developer (such as shyouhei (Shyouhei Urabe))? We can revert any warning category you are not comfortable with, as a separate commit will be used for each category.

#10 - 10/26/2020 07:50 AM - matz (Yukihiro Matsumoto)

I am OK with warning categories. But the proposed categories are too fine-grained, I think. Here's the Python warning categories:

https://docs.python.org/3/library/warnings.html#warning-categories

Matz.

#11 - 10/26/2020 03:59 PM - jeremyevans0 (Jeremy Evans)

matz (Yukihiro Matsumoto) wrote in #note-10:

> I am OK with warning categories. But the proposed categories are too fine-grained, I think. Here's the Python warning categories:
>
> https://docs.python.org/3/library/warnings.html#warning-categories

Here are the Python 3 warning categories, along with the possible categories for Ruby:

Warning: Base category in Python. I assume this would be a nil category in Ruby, and used for warnings that are not assigned a category?

UserWarning: Should we use :user for this category and make Kernel#warn default to this category if a category isn't given?

DeprecationWarning: We already have this category, :deprecated.

SyntaxWarning: The :compile warning category in my original proposal covered this. Do we want to use :syntax instead?

RuntimeWarning: This could potentially cover uninitialized instance/global variable access, method redefinition, ignored blocks, unknown argv flags, and many other things. Do we want to use a :runtime category for all of these?

FutureWarning, PendingDeprecationWarning: Additional types of deprecation warnings in Python. Do we want to have multiple deprecation warning categories in Ruby?

ImportWarning: The :require category in my original proposal covered this.

UnicodeWarning, BytesWarning: This is related to unicode/bytes split in Python that Ruby does not have. The :encoding category in my original proposal is probably the closest similar thing in Ruby.

ResourceWarning: The :range category in my original proposal is probably the closest similar thing in Ruby.

The reason for using :uninitialized_ivar and :redefine fine grained categories is that they are probably the most useful to Rubyists. Most other warnings you would want to fix and make the warning go away, but there are valid performance reasons for not initializing instance variables, and valid thread-safety reasons for not undefining methods before redefining them.

I'm happy to implement whatever warning categories are decided upon, after the decision has been made.

#12 - 11/20/2020 04:53 AM - mame (Yusuke Endoh)

- Related to Feature #17055: Allow suppressing uninitialized instance variable and method redefined verbose mode warnings added

#13 - 11/20/2020 05:07 AM - naruse (Yui NARUSE)

In regard to #17055, Instead of new feature to suppress warning, how about assuming @foo = nil is that. instance_variable_get and so on should also be mimicked.

#14 - 11/20/2020 05:28 AM - jeremyevans0 (Jeremy Evans)

naruse (Yui NARUSE) wrote in #note-13:

> In regard to #17055, Instead of new feature to suppress warning, how about assuming @foo = nil is that.

@foo = nil sets the instance variable, with the negative performance implications of doing so. See https://bugs.ruby-lang.org/issues/17055#note-13. This is a realistic benchmark (not a microbenchmark) that shows it is 100-150% faster to not set instance variables to nil when using Sequel with PostgreSQL to return rows from the database.

The whole point of suppressing instance variable warnings is to allow the highest possible performance without warnings. Similarly, the whole point of suppressing method redefinition warnings is to allow atomic method replacement without alias hacks.

> instance_variable_get and so on should also be mimicked.

I'm not sure I understand this point, could you explain?

#15 - 11/20/2020 05:31 AM - shyouhei (Shyouhei Urabe)

Persuaded at today's developer meeting. I now think that :redefine shall be handled in other ways.

The point is, it is the author's intention, not the end user's, that they wants to redefine a method without warnings. Asking everyone else to set RUBYOPT='-W:no-redefine' sounds just a wrong way to solve the problem. The intention must be clearly declared by the author into their code. We should have a dedicated method something like Class#suppress_redefinition_of_this_symbol(sym).

Deprecation is a different story (people did not intend to use a deprecated feature; core devs make them deprecated). I can understand the use of warning mechanism there.

#16 - 11/20/2020 01:05 PM - nobu (Nobuyoshi Nakada)

Regarding redefine, though I'm not sure what case you're thinking, guess it's better to declare that method to be redefined.

For instance, the case defer the "real" method definition till called, such as Binding#irb and Kernel#pp in prelude.rb,

```
class Module
  def overridable(name)
    alias_method name, name
    name
  end
end

class Binding
  overridable def irb
    require 'irb'
    irb
  end
end

module Kernel
  private overridable def pp(*objs)
    require 'pp'
    pp(*objs)
  end
end
```