Ruby master - Feature #17000

2.7.2 turns off deprecation warnings by default

06/30/2020 11:52 AM - mame (Yusuke Endoh)

Status:	Closed
Priority:	Normal
Assignee:	nagachika (Tomoyuki Chikanaga)
Target version:	

Description

Matz has decided to disable deprecation warnings for 3.0 keyword separation by default because many users feel them noisy and painful rather than useful. See https://discuss.rubyonrails.org/t/new-2-7-3-0-keyword-argument-pain-point/74980/47 .

https://github.com/ruby/ruby/pull/3273 is a pull request for the change. It is essentially one-line change in error.c, though it has many changes for tests that checks if the warning is appropriately emitted.

Note that this changeset disables *all* deprecation warnings by default. The reason why I stop not only keyword-related deprecation warnings but all other ones is because, if we disable only keyword-related deprecation warnings, and if keep other deprecation on by default, it becomes ambiguous what Warning[:deprecated] should return. We considered a new API like Warning[:keyword_separation_deprecated] = true / false, but we want to minimize the change because it goes to 2.7 branch. Fortunately, there are not so many other warnings disabled together; a notable one is Capturing the given block using Kernel#proc is deprecated; use&blockinstead, but other warnings are minor, as far as I see. It is somewhat unfortunate, but matz has already agreed with this direction.

Matz also said in the forum, "we will move on to the new keyword argument behavior in Ruby3.0 as planned". This violates the traditional convention that "gradually" makes deprecation warnings noisy: deprecated behavior is (1) warned only when VERBOSE is enabled, (2) always warned, and (3) removed. However, matz made this decision due to special circumstances of 3.0 keyword arguments; delaying the change will be also painful.

Note that, currently, the deprecation convention itself is not changed, so this patch is *not* going to master branch. We can discuss the deprecation policy change in another ticket, if needed. Anyway, let this ticket focus on ruby_2_7 branch change. Please do not discuss the convention change in this ticket.

The final decision is up to 2.7 branch maintainer, nagachika (Tomoyuki Chikanaga) san, but I hope this change is accepted.

cc/ jeremyevans0 (Jeremy Evans)Eregon (Benoit Daloze)nobu (Nobuyoshi Nakada)nagachika (Tomoyuki Chikanaga)

Related issues:

Related to Ruby master - Feature #16345: Don't emit deprecation warnings by d...	**Closed**

Associated revisions

Revision df3f52a6 - 09/29/2020 01:43 PM - nagachika (Tomoyuki Chikanaga)

merge revision(s) 996af2ce086249e904b2ce95ab2fcd1de7d757be: [Backport #16345] [Backport #17000]

```
    Disable deprecation warning by the default [Feature #16345]

    And `-w` option turns it on.
```

History

#1 - 06/30/2020 04:29 PM - marcandre (Marc-Andre Lafortune)

I understand (and also had to feel) the pain points. I can't imagine what this would have been with duplication warnings https://bugs.ruby-lang.org/issues/16289...

IIUC the proposal is basically to have "-W:no-deprecated" by default, right?

I fear this is only postponing the issue to having even more drastic breakage when 3.0 comes out. It has been 6 months that Ruby 2.7 is out with warnings by default.

The plan that these go from "opt-in warnings" to flat out errors in Ruby 3.0 in 6 months can only seem to me to be a mistake. I said it before (https://bugs.ruby-lang.org/issues/14183#change-73946), delegation is the major issue. The path should have involved a no warning by default release, and then a warning by default.

If we turn these deprecation warnings off, then we must have a 2.8 with warning turned on and a 3.0 in 1.5 years. Or we tick with deprecation warnings with 3.0 in 6 months.

I have the impression that the main annoyance is the difficulty in filtering between the end-user's code and dependencies. IMO a gem that could be loaded that would filter the warnings according to their originating call would be more useful and create less pain when 3.0 actually comes out.

In short: I'm am not in favor of the proposal as it stands.

#2 - 07/01/2020 12:34 PM - nagachika (Tomoyuki Chikanaga)

I understand the pain point discussion of warning for the 3.0 keyword arguments, and I agree to turn off the warning for keyword arguments in ruby_2_7 branch.

But the proposed changesets in the pull request https://github.com/ruby/ruby/pull/3273 turns off the whole deprecated warnings. I worried about a kind of discontinuity of the some deprecation warnings. For example, IO#lines (and IO#charse, IO#bytes) has been deprecated for the long time, and displayed warning message without any options in 2.6.x and 2.7.1. The warning for IO#lines be disabled by default if the proposed changesets are applied into ruby_2_7.
On the other hand, the warning categories feature (https://bugs.ruby-lang.org/issues/16420) was introduced at 2.7.0. Some long standing deprecated warnings are categorized as "deprecated" class and can be controlled by -W:no-deprecated option or Warning[:deprecated]= method after 2.7.0.

I propose to introduce the dependent category for "keyword arguments" to controll warnings for the 3.0 keyword arguments and backport it into the ruby_2_7 branch.

I think introducing the new warning category has less impacto for compatibility than changing the deprecated warning default behavior. I did a rough search in GitHub and didn't find any code contains Warning[:depcated] or option -W:depracated. Maybe my query for the search was wrong and overlook for the real usecase. Please tell me if you find the real application/libraries already using warning categories feature.

#3 - 07/01/2020 02:47 PM - mame (Yusuke Endoh)

nagachika-san, thank you for your reply!

nagachika (Tomoyuki Chikanaga) wrote in #note-2:

> I worried about a kind of discontinuity of the some deprecation warnings.

I agree that it is best to disable only keyword-related warnings. If you who is 2.7 branch maintainer are okay for an unnecessarily big new feature towards 2.7, I'm happy to create a patch.

> I propose to introduce the dependent category for "keyword arguments" to controll warnings for the 3.0 keyword arguments and backport it into the ruby_2_7 branch.

Let me confirm: what do you mean "dependent"?

Let's call the new category ":keyword_deprecated". Warning[:deprecated] is true by default, and Warning[:keyword_deprecated] is false by default. I think that the following behavior is expected:

Warning[:deprecated]	Warning[:keyword_deprecated]	IO#lines deprecation	keyword deprecation
true (default)	false (default)	printed	not printed
true	true	printed	printed
false	false	not printed	not printed
false	true	not printed	printed

But this new category is not "dependent" on :deprecated category. Rather, they are independent. Is this okay? Or is your expectation different?

> I think introducing the new warning category has less impacto for compatibility than changing the deprecated warning default behavior.

Adding a new category means that we need to change not only 2.7 branch but also master branch. In master, we need to accept (and just ignore) Warning[:keyword_deprecated] = true. We need to get approval from matz.

#4 - 07/01/2020 03:51 PM - jeremyevans0 (Jeremy Evans)

An alternative approach to adding Warning[:keyword_deprecated] that would be easier to implement in 2.7 and would not require changes to master would be only printing keyword argument warnings in verbose mode. This should be a one-line change here:
https://github.com/ruby/ruby/blob/ruby_2_7/vm_args.c#L600

This also makes it easier to turn on keyword warnings (-w vs -W:keyword_deprecated). The downside is it also turns on other verbose warnings, which the user may not be interested in.

#5 - 07/01/2020 05:36 PM - byroot (Jean Boussier)

> This also makes it easier to turn on keyword warnings (-w vs -W:keyword_deprecated). The downside is it also turns on other verbose warnings, which the user may not be interested in.

Yes, that would be a big problem for us. We hook into Warning.warn to ensure our test suite pass with no warnings, but we never run with -w.

So as users who spent lots of effort into clearing all our keyword deprecation warnings, we'd like a way to continue to enable them without also enabling the entire verbose mode.

#6 - 07/02/2020 01:00 AM - nagachika (Tomoyuki Chikanaga)

mame (Yusuke Endoh) wrote in #note-3:

> Let me confirm: what do you mean "dependent"?

I'm sorry, I just misspelled it. I mean "new *deprecated* category for keyword arguments".

> Let's call the new category ":keyword_deprecated". Warning[:deprecated] is true by default, and Warning[:keyword_deprecated] is false by default. I think that the following behavior is expected:

Thank you for clarifying. The table you show is exactly what I expected.

> Adding a new category means that we need to change not only 2.7 branch but also master branch. In master, we need to accept (and just ignore) Warning[:keyword_deprecated] = true. We need to get approval from matz.

You are right. I didn't consider about master branch.
In this point, Jeremy's proposal could inspires us.

jeremyevans0 (Jeremy Evans) wrote in #note-4:

> An alternative approach to adding Warning[:keyword_deprecated] that would be easier to implement in 2.7 and would not require changes to master would be only printing keyword argument warnings in verbose mode. This should be a one-line change here:
> https://github.com/ruby/ruby/blob/ruby_2_7/vm_args.c#L600

I understand your proposal is enabling all warnings in "deprecated" category (keywords and others) only if $VERBOSE=true and Warning[:deprecated]=true, is that right?
I rather like to suppress only keyword arguments warning without $VERBOSE=true and don't suppress other deprecated category warning.
I show the another proposal in the following table.

Warning[:deprecated]	$VERBOSE	other deprecation	keyword deprecation
true (default)	false (default)	printed	not printed
true	true	printed	printed
false	false	not printed	not printed
false	true	not printed	not printed

I know we have to create more than one line patch for this change, but the implementation bug can be fixed in the next teeny releases.
I'd like to concentrate to consider the good specification which has less impact for users while reduce verbosity for keyword args warning.

#7 - 07/02/2020 01:15 AM - jeremyevans0 (Jeremy Evans)

nagachika (Tomoyuki Chikanaga) wrote in #note-6:

> jeremyevans0 (Jeremy Evans) wrote in #note-4:
>
> > An alternative approach to adding Warning[:keyword_deprecated] that would be easier to implement in 2.7 and would not require changes to master would be only printing keyword argument warnings in verbose mode. This should be a one-line change here:
> > https://github.com/ruby/ruby/blob/ruby_2_7/vm_args.c#L600
>
> I understand your proposal is enabling all warnings in "deprecated" category (keywords and others) only if $VERBOSE=true and Warning[:deprecated]=true, is that right?

No, my proposal only affects keyword arguments. rb_warn_check is only called by the three functions that issue deprecation warnings for keyword arguments (all three callers are directly below in vm_args.c).

> I rather like to suppress only keyword arguments warning without $VERBOSE=true and don't suppress other deprecated category warning.

I believe modifying the line I referenced should be sufficient to implement the proposal in your table. Well, it may spill into a second line as that line already has 78 characters, but that should be the only location needed. I can prepare a patch if you would like.

#8 - 07/04/2020 11:34 AM - nagachika (Tomoyuki Chikanaga)

jeremyevans0 (Jeremy Evans) wrote in #note-7:

> No, my proposal only affects keyword arguments. rb_warn_check is only called by the three functions that issue deprecation warnings for keyword arguments (all three callers are directly below in vm_args.c).

Oh, I misunderstood it.
I'm willing to try confirming behaviors if you kindly prepare the patch.

#9 - 07/04/2020 05:05 PM - jeremyevans0 (Jeremy Evans)

- File keyword-warnings-verbose-mode.patch added

nagachika (Tomoyuki Chikanaga) wrote in #note-8:

> jeremyevans0 (Jeremy Evans) wrote in #note-7:
>
> > No, my proposal only affects keyword arguments. rb_warn_check is only called by the three functions that issue deprecation warnings for keyword arguments (all three callers are directly below in vm_args.c).
>
> Oh, I misunderstood it.
> I'm willing to try confirming behaviors if you kindly prepare the patch.

Attached is the patch. It's only a 1 line change to vm_args.c, but it includes all test changes.

#10 - 07/09/2020 07:36 AM - mame (Yusuke Endoh)

How difficult it is to just disable some warnings!

I think that byroot is right here. Unfortunately, the practice of $VERBOSE=true is not so popular, so it will make many libraries "too verbose". Many unrelated warnings will makes it difficult to fix code to be ready for the Ruby 3 keyword argument change.

A simpler idea: How about printing keyword-related warnings only when an environment variable RUBY3_KEYWORD_WARNING set to 1? It allows user to enable the warnings with no change of the program nor command-line argument. Note that it is also programmable to enable the warnings by ENV["RUBY3_KEYWORD_WARNING"]="1".

#11 - 07/09/2020 07:37 AM - mame (Yusuke Endoh)

- Subject changed from 2.7.2 turns off deprecation warnings by deafult to 2.7.2 turns off deprecation warnings by default

#12 - 07/10/2020 01:58 PM - Dan0042 (Daniel DeLorme)

It seems to me the big issue is that developers have little power over warnings that come from gems. If the gem that produces warnings has not been patched yet, you have to wait for an update and suffer the warnings in the meantime.

So has anyone considered silencing warnings when they originate from gems? I mean, the warning already includes that information:

```
/path/to/foo.rb:2: warning: Using the last argument as keyword parameters is deprecated; maybe ** should be ad
ded to the call
/path/to/foo.rb:1: warning: The called method `foo' is defined here
```

It should be pretty simple to check if the filename in the first warning starts with Gem::RUBYGEMS_DIR and only display the 2 warnings if explicitly opted in (via $VERBOSE or Warning[:gems] or such)

#13 - 07/10/2020 02:47 PM - jeremyevans0 (Jeremy Evans)

Dan0042 (Daniel DeLorme) wrote in #note-12:

> So has anyone considered silencing warnings when they originate from gems? I mean, the warning already includes that information:
>
> ```
> /path/to/foo.rb:2: warning: Using the last argument as keyword parameters is deprecated; maybe ** should b
> e added to the call
> /path/to/foo.rb:1: warning: The called method `foo' is defined here
> ```
>
> It should be pretty simple to check if the filename in the first warning starts with Gem::RUBYGEMS_DIR and only display the 2 warnings if explicitly opted in (via $VERBOSE or Warning[:gems] or such)

This is simple to do with the warning gem:

```
require 'warning'
Gem.path.each do |path|
  Warning.ignore(:keyword_separation, path)
end
```

#14 - 07/16/2020 02:33 AM - akr (Akira Tanaka)

- Related to Feature #16345: Don't emit deprecation warnings by default. added

#15 - 07/20/2020 03:57 AM - akr (Akira Tanaka)

nagachika (Tomoyuki Chikanaga) wrote in #note-2:

> But the proposed changesets in the pull request https://github.com/ruby/ruby/pull/3273 turns off the whole deprecated warnings. I worried about a kind of discontinuity of the some deprecation warnings. For example, IO#lines (and IO#charse, IO#bytes) has been deprecated for the long time, and displayed warning message without any options in 2.6.x and 2.7.1. The warning for IO#lines be disabled by default if the proposed changesets are applied into ruby_2_7.

How about enabling non-keyword-argument related warnings regardless of Warning[:deprecated] as Ruby 2.6 for compatibility?

If we do it, we can change the default of Warning[:deprecated] to false to
disable keyword-argument related warnings by default.

Of course, this degrades the feature of Warning[:deprecated] for
non-keyword-argument related warnings but
it's impact is minimal because it is introduced recently.

I don't like keyword-argument specific category for Warning because
we need to maintain trunk.

Anyway, we should change Warning[:deprecated] to false by default on trunk
at same time or earlier than Ruby 2.7, I think.

#16 - 07/20/2020 01:01 PM - nagachika (Tomoyuki Chikanaga)

According to https://bugs.ruby-lang.org/issues/16345#note-46, Matz changed his mind and proposed to turn off the deprecated warnings by default in 2.8/3.0. If that policy will be taken, I'd like to adopt mame-san's first proposal; turn off Warning[:deprecated] by default. In my opinion, the main downside of this approach was the deprecation warnings will be turned off only in 2.7, but it can be interpreted as the backporting the feature changes in 2.8/3.0 if the deprecation warnings will be turned off by default in 2.8/3.0.
If the 3.0 doesn't change the policy for deprecated warning or the discussion in #16345 doesn't reached to conclusion in the following few weeks, I will accept akr's suggestion. I feel this will be best way to mitigate compatibility impact.

#17 - 08/11/2020 12:23 AM - inopinatus (Joshua GOODALL)

I think deprecation warnings are necessary if, and only if, a feature or behaviour is going away in the lifespan of the minor (2.x) versions.

I also believe you can make any breaking change without prior warning in a major version (i.e. 3.0). For deprecation this is against the SemVer rules, but I think they are wrong on this point.

However I also think that minor-version deprecation warnings should be opt-out, rather than opt-in, otherwise they fail to serve their own purpose.

#18 - 09/29/2020 01:43 PM - nagachika (Tomoyuki Chikanaga)

- Status changed from Open to Closed

Applied in changeset git|df3f52a6331f1a47af9933b77311a8650727d8d1.

---

merge revision(s) 996af2ce086249e904b2ce95ab2fcd1de7d757be: [Backport #16345] [Backport #17000]

```
    Disable deprecation warning by the default [Feature #16345]

    And `-w` option turns it on.
```

Files

keyword-warnings-verbose-mode.patch	161 KB	07/04/2020	jeremyevans0 (Jeremy Evans)