Ruby trunk - Feature #12299

Add Warning module for customized warning handling

04/18/2016 10:59 PM - jeremyevans0 (Jeremy Evans)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

This is another attempt to provide the same type of support for customized warning handling as #12026. matz had a couple of issues with #12026. First, he did not like the introduction of a new global variable. Second, he wanted the ability to control warning handling on a per-gem and/or per-class basis.

The approach in this patch adds a Warning module instead of using a global variable, and allows for per-gem customization by using separate processing for separate gems based on the path prefix in the warning message. I don't think per-class warnings are possible without major changes.

This changes all warnings raised call to Warning.warn, which by default does the same thing it does currently (rb_write_error_str). You can override Warning.warn to change the behavior.

To provide a common API that can be used to modify warnings, add a lib/warning.rb file, which offers three methods: Warning.ignore, Warning.process, and Warning.clear.

Warning.ignore takes a regexp and optionally a path prefix, and ignores any warning that matches the regular expression if it starts with the path prefix.

Warning.process takes an optional path prefix and a block, and if the string starts with the path prefix, it calls the block with the warning string instead of performing the default behavior. You can call Warning.process multiple times and it will be operate intelligently, choosing the longest path prefix that the string starts with.

Warning.clear just clears the current ignored warnings and warning processors.

By using path prefixes, it's fairly easy for a gem to set that warnings should be ignored inside the gem's directory.

Note that path prefixes will not correctly handle warnings raised by Kernel#warn, unless the warning message given to Kernel#warn starts with the filename where the warning is used.

Note that the current implementations of Warning.ignore, Warning.process, and Warning.clear in the patch are not thread safe, but that can be changed without much difficulty.

Examples from the RDoc in the patch:

```
# Ignore all uninitialized instance variable warnings
Warning.ignore(/instance variable @\w+ not initialized/)

# Ignore all uninitialized instance variable warnings in current file
Warning.ignore(/instance variable @\w+ not initialized/, __FILE__)

# Write warning to LOGGER at level warning
Warning.process do |warning|
  LOGGER.warning(warning)
end

# Write warnings in the current file to LOGGER at level error
Warning.process(__FILE__) do |warning|
  LOGGER.error(warning)
end
```

Associated revisions

Revision 4e60f998 - 09/27/2016 09:19 AM - shyouhei (Shyouhei Urabe)

- error.c: This makes all warnings raised call Warning.warn, which by default does the same thing it does currently (rb_write_error_str). You can override Warning.warn to change the behavior. [ruby-core:75016] [Feature #12299]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@56269 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 56269 - 09/27/2016 09:19 AM - shyouhei (Shyouhei Urabe)

- error.c: This makes all warnings raised call Warning.warn, which by default does the same thing it does currently (rb_write_error_str). You can override Warning.warn to change the behavior. [ruby-core:75016] [Feature #12299]

Revision 56269 - 09/27/2016 09:19 AM - shyouhei (Shyouhei Urabe)

- error.c: This makes all warnings raised call Warning.warn, which by default does the same thing it does currently (rb_write_error_str). You can override Warning.warn to change the behavior. [ruby-core:75016] [Feature #12299]

Revision 56269 - 09/27/2016 09:19 AM - shyouhei (Shyouhei Urabe)

- error.c: This makes all warnings raised call Warning.warn, which by default does the same thing it does currently (rb_write_error_str). You can override Warning.warn to change the behavior. [ruby-core:75016] [Feature #12299]

Revision 56269 - 09/27/2016 09:19 AM - shyouhei (Shyouhei Urabe)

- error.c: This makes all warnings raised call Warning.warn, which by default does the same thing it does currently (rb_write_error_str). You can override Warning.warn to change the behavior. [ruby-core:75016] [Feature #12299]

History

#1 - 06/13/2016 08:19 AM - matz (Yukihiro Matsumoto)

I like the basic idea. I can be fundamental of higher level abstraction.
My only concern is the name. Warning may be too generic and cause conflict.

Matz.

#2 - 06/13/2016 08:26 AM - ko1 (Koichi Sasada)

- If C extension shows warning, we can't stop with filen names because the file is caller's file (maybe not a problem because we should specify caller's file).
- Now, passed |message| is a string like ": ". Should we pass separated objects like |file, line, message|?
- Do we need other information like backtrace?
- Should we filter also with line? (over spec?)

#3 - 06/13/2016 02:51 PM - jeremyevans0 (Jeremy Evans)

Yukihiro Matsumoto wrote:

> I like the basic idea. I can be fundamental of higher level abstraction.
> My only concern is the name. Warning may be too generic and cause conflict.

How about using RubyVM::Warning?

Koichi Sasada wrote:

> If C extension shows warning, we can't stop with filen names because the file is caller's file (maybe not a problem because we should specify caller's file).

I don't think this is a problem. If the user is calling a C extension that raises a warning, the problem is in the caller's file, or they need to fix the C extension.

> Now, passed |message| is a string like ": ". Should we pass separated objects like |file, line, message|?

I don't think so. Because the data originates as a string, you could only get that information from splitting the string, which the user could easily do themselves.

> Do we need other information like backtrace?

I think caller would provide this information.

> Should we filter also with line? (over spec?)

I think the current patch allows this, since it just works on string prefixes and warning strings start with file and then line. Filtering by line would be fragile, as even non-code changes like modifying comments can modify line numbers.

#4 - 06/20/2016 05:52 PM - jeremyevans0 (Jeremy Evans)

- File 0001-Add-Exception-Warning-module-for-customized-warning-.patch added

Yukihiro Matsumoto wrote:

> I like the basic idea. I can be fundamental of higher level abstraction.
> My only concern is the name. Warning may be too generic and cause conflict.

I originally thought about using RubyVM::Warning for this, but rb_cRubyVM is defined in vm.c without extern, so it cannot be used from error.c. Additionally, RubyVM is really for things specific to the CRuby virtual machine, and this is not related.

I've attached an updated patch that moves Warning under Exception (e.g. Exception::Warning), which hopefully should avoid conflicts with existing code.

Additionally, this adds thread safety via MonitorMixin for the previously thread-unsafe code in lib/warnings.rb.

Hopefully this patch is now acceptable for merging.

#5 - 06/22/2016 04:13 PM - djberg96 (Daniel Berger)

Wrapping regular expressions feels very clumsy to me.

I still prefer the approach I put forward in https://bugs.ruby-lang.org/issues/11588

#6 - 07/19/2016 06:51 AM - jeremyevans0 (Jeremy Evans)

In the notes from the developer meeting earlier today, it says "no progress this month", but that is incorrect as a new patch was submitted on June 20th addressing the naming issues raised in June's developer meeting. Hopefully this patch can be discussed at next month's developer meeting.

#7 - 07/20/2016 01:40 AM - shyouhei (Shyouhei Urabe)

Sorry for the miss (my fault). As far as I read the proposed patch, it seems the "wrapping regular expression" part that Daniel points is separated to a library and does not take effect until one explicitly require 'warning'. Isn't this enough? Users can ignore the library and implement other approaches like structured warnings on top of the basic part.

#8 - 08/09/2016 05:44 AM - matz (Yukihiro Matsumoto)

I accept the mechanism.
I prefer Warning to Exception::Warning.

Besides that, the basic should be included in the core (i.e. the patch to error.c), but its customization (i.e. lib/warning.rb) should be a Gem at first.

Matz.

#9 - 08/09/2016 03:38 PM - jeremyevans0 (Jeremy Evans)

- File 0001-Add-Warning-module-for-customized-warning-handling.patch added

Here is a patch that uses Warning instead of Exception::Warning, and only includes the error.c change (and a test for Warning.warn). I'll work on a gem shortly for lib/warning.rb.

#10 - 08/09/2016 04:41 PM - jeremyevans0 (Jeremy Evans)

I've added a gem for lib/warning.rb (https://rubygems.org/gems/warning).

#11 - 09/07/2016 02:12 PM - jeremyevans0 (Jeremy Evans)

Matz accepted this mechanism about a month ago, and shortly thereafter I added a patch with his requested change. Can this be patch be applied before 2.4.0-preview2?

#12 - 09/27/2016 09:19 AM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Closed

Applied in changeset r56269.

- error.c: This makes all warnings raised call Warning.warn, which by default does the same thing it does currently (rb_write_error_str). You can override Warning.warn to change the behavior. [ruby-core:75016] [Feature #12299]

#13 - 09/27/2016 10:25 AM - shyouhei (Shyouhei Urabe)

Sorry, I was too busy to merge this before preview2.

#14 - 04/23/2017 08:22 PM - Eregon (Benoit Daloze)

This feature is great, thank you for introducing it!

A couple things I noticed while using it for ruby/spec:
https://github.com/ruby/mspec/commit/954054c03da1c90c9653f238c78c888644760ab8

- Parser warnings are emitted at parse time. This is expected but makes them harder to filter as for instance changing $VERBOSE around does not have effect, nor defining Warning.warn before running a method. One workaround is to use #eval which then delays the warnings until execution of the eval expression.
- Kernel#warn does not use Warning.warn yet, so one needs to entirely redefine Kernel#warn which is not so trivial (see commit).
- Warning #extend itself. This is a common pattern in Ruby but I have never seen it in core modules before. It confused me while looking in a REPL:

```
> Warning.methods(false)
=> []
> Warning.method :warn
=> #<Method: Module(Warning)#warn>
```

On the upside it allows to call super in Warning.warn, but that's mostly $stderr.write(message).
Is there any reason to make Warning#warn include-able?
I think it would be less surprising to just have the singleton Warning.warn method.

#15 - 04/24/2017 12:38 AM - jeremyevans0 (Jeremy Evans)

Eregon (Benoit Daloze) wrote:

> This feature is great, thank you for introducing it!
>
> A couple things I noticed while using it for ruby/spec:
> https://github.com/ruby/mspec/commit/954054c03da1c90c9653f238c78c888644760ab8
>
> - Parser warnings are emitted at parse time. This is expected but makes them harder to filter as for instance changing $VERBOSE around does not have effect, nor defining Warning.warn before running a method. One workaround is to use #eval which then delays the warnings until execution of the eval expression.

I think this is expected. If you want to filter/process warnings, it's best to setup Warning.warn close to process initialization, before loading other files.

> - Kernel#warn does not use Warning.warn yet, so one needs to entirely redefine Kernel#warn which is not so trivial (see commit).

This was fixed after 2.4 was released (r57370). In trunk, Kernel#warn calls Warning.warn.

> - Warning #extend itself. This is a common pattern in Ruby but I have never seen it in core modules before. It confused me while looking in a REPL:
>
> ```
> > Warning.methods(false)
> => []
> > Warning.method :warn
> => #<Method: Module(Warning)#warn>
> ```
>
> On the upside it allows to call super in Warning.warn, but that's mostly $stderr.write(message).
> Is there any reason to make Warning#warn include-able?

Precisely because it makes it easier to define Warning.warn manually and call super to get default behavior. Without this, if you wanted to override and call super, you would have to use Module#prepend, which is more work. Warning.warn was added specifically to make it easy to override the default handling of warnings.

> I think it would be less surprising to just have the singleton Warning.warn method.

Other than using Warning.methods(false), I doubt anyone would notice. In any case, it can't be changed now without breaking backwards compatibility, as existing libraries already rely on the current behavior.

#16 - 04/24/2017 06:12 PM - Eregon (Benoit Daloze)

> Precisely because it makes it easier to define Warning.warn manually and call super to get default behavior.

I see, this sounds worth documenting in the rdoc of Warning#warn then :)

Files

0001-Add-Warning-module-for-customized-warning-handling.patch	10.1 KB	04/18/2016	jeremyevans0 (Jeremy Evans)
0001-Add-Exception-Warning-module-for-customized-warning-.patch	10.7 KB	06/20/2016	jeremyevans0 (Jeremy Evans)
0001-Add-Warning-module-for-customized-warning-handling.patch	4.7 KB	08/09/2016	jeremyevans0 (Jeremy Evans)