Ruby - Feature #18296

Custom exception formatting should override `Exception#full_message`.

11/10/2021 11:49 AM - ioquatix (Samuel Williams)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

After discussing with @Eregon (Benoit Daloze), we came to the conclusion that the current implementation of did_you_mean and error_highlight could avoid many issues by using Exception#full_message.

We propose to introduce a more nuanced interface:

```
class Exception
  def full_message(highlight: bool, order: [:top or :bottom], **options)
    # ...
  end
end

module DidYouMean
  module Formatter
    def full_message(highlight:, did_you_mean: true, **options)
      buffer = super(highlight: highlight, **options).dup
      buffer << "extra stuff"
    end
  end
end
Exception.prepend DidYouMean::Formatter

module ErrorHighlight
  module Formatter
    def full_message(highlight:, error_highlight: true, **options)
      # same as above
    end
  end
end
Exception.prepend ErrorHighlight::Formatter
```

Related issues:

Related to Ruby - Bug #18170: Exception#inspect should not include newlines	**Closed**
Related to Ruby - Feature #18194: No easy way to format exception messages pe...	**Closed**
Related to Ruby - Feature #18370: Call Exception#full_message to print except...	**Closed**
Related to Ruby - Feature #18438: Add `Exception#additional_message` to show ...	**Closed**

History

#1 - 11/10/2021 11:50 AM - ioquatix (Samuel Williams)

- Description updated

#2 - 11/10/2021 11:50 AM - Eregon (Benoit Daloze)

- Related to Bug #18170: Exception#inspect should not include newlines added

- Related to Feature #18194: No easy way to format exception messages per thread/fiber scheduler context. added

#3 - 11/10/2021 11:51 AM - Eregon (Benoit Daloze)

- Description updated

#4 - 11/10/2021 11:57 AM - Eregon (Benoit Daloze)

I think this is a clean and simple proposal/fix, which enables users to choose whether they want that extra information on a per-call to #full_message-basis.

As a side effect it also enables DidYouMean and ErrorHighlight to e.g. look at the value of the highlight: keyword argument value and decide whether to use ANSI escape sequences.

cc @mame (Yusuke Endoh) @yuki24 (Yuki Nishijima) I think this is the best way to integrate ErrorHighlight and DidYouMean cleanly.
Overriding message was always a shortcut/hack (because it doesn't let the user choose what they want, only globally via --disable which is not good enough), and I believe this is the way to fix it.

#5 - 11/10/2021 11:58 AM - ioquatix (Samuel Williams)

Also given how this works, I believe we should remove ErrorHighlight.formatter from the public interface.
https://github.com/ruby/error_highlight/issues/15

#6 - 11/11/2021 05:49 AM - mame (Yusuke Endoh)

TBH I'm unsure if I could understand what you propose. It would be very helpful if you could create a proof-of-concept patch.

I have one concern. My minimum expectation of error_highlight and did_you_mean is that the ruby interpreter should print their hints by default. However, at the present time, the ruby interpreter uses #message instead of #full_message to show the error information of uncaught exception.

```
$ ruby -e 'class Foo < Exception; def message = "message"; def full_message = "full_message"; end; raise Foo.n
ew'
-e:1:in `<main>': message (Foo)
```

Does this proposal include that the ruby interpreter should use #full_message to show the error information? This is an incompatibility, is it acceptable?

There is an ecosystem to handle uncaught exception, including application monitoring services (Newrelic, DataDog, ScoutAPM, ...) and Rails' exception handling system for development mode. They also should change their code base to use #full_message (or more dedicated display depending on their context). I think we need to coordinate them if we introduce this incompatibility. cc @ivoanjo

#7 - 11/11/2021 11:07 AM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote in #note-6:

> Does this proposal include that the ruby interpreter should use #full_message to show the error information? This is an incompatibility, is it acceptable?

Yes, let's fix that.
I don't think there is much if any compatibility issue here.
The output of the uncaught exception handler is already the same as the default Exception#full_message AFAIK, let's actually call it.
TruffleRuby already calls exc.full_message for the uncaught exception handler.

If the custom exc.full_message raises an exception, then it's best to report that exception *and* the original exception using the default Exception#full_message (written in C).
This is the current TruffleRuby output for that case and I think it's clear:

```
$ ruby -e 'class Foo < Exception; def full_message(**); raise "bar"; end; end; raise Foo, "message"'
Error while formatting Ruby exception:
-e:1:in `full_message': bar (RuntimeError)
 from <internal:core> core/truffle/exception_operations.rb:183:in `get_formatted_backtrace'
Original Ruby exception:
-e:1:in `<main>': message (Foo)
```

> There is an ecosystem to handle uncaught exception, including application monitoring services (Newrelic, DataDog, ScoutAPM, ...) and Rails' exception handling system for development mode.
> They also should change their code base to use #full_message (or more dedicated display depending on their context). I think we need to coordinate them if we introduce this incompatibility. cc @ivoanjo

It's a fair point.
I think for them it can make sense in some cases to not include the backtrace in it or to query it separately for convenience.
Thus I propose adding a keyword argument backtrace: to full_message so exc.full_message(backtrace: false) is like message but with all the extra decorations from did_you_mean/error_highlight/etc (and not include the backtrace or causes in that case).

#8 - 11/11/2021 11:12 AM - Eregon (Benoit Daloze)

- Description updated

#9 - 11/11/2021 11:38 AM - Eregon (Benoit Daloze)

While initially thinking about this issue I kind of forgot full_message includes the backtrace (the name doesn't make that super obvious).

One thing to address is how did_you_mean/error_highlight can insert additional output after the original message, but before the backtrace (for order: :top the default).

super(highlight: highlight, **options) + "\nextra" wouldn't work as that would put it after the end of the backtrace.
Also the backtrace can be before the message with order: :bottom.

I think one possibility is:

```
module DidYouMean
  module Formatter
    def full_message(highlight: Exception.to_tty?, order: :top, backtrace: true, did_you_mean: true, **options
)
      super if backtrace == :only || !did_you_mean
      message = super(highlight: highlight, **options, backtrace: false)
      backtrace = super(highlight: highlight, **options, backtrace: :only)
      if order == :top
        "#{message}\nextra stuff\n#{backtrace}"
      else
        "#{backtrace}#{message}\nextra stuff\n"
      end
    end
  end
end

Exception.prepend DidYouMean::Formatter
```

If we want to avoid the if order == :top, we'd need a new method that's basically the same as full_message but always without the backtrace, and that method would be called from Exception#full_message (needs at least the highlight: keyword, best to forward all keywords).
(then we don't need the backtrace: kwarg for full_message of course)
Like:

```
module DidYouMean
  module Formatter
    def message_with_extras(highlight: Exception.to_tty?, did_you_mean: true, **options)
      super unless did_you_mean
      message = super(highlight: highlight, **options)
      "#{message}\nextra stuff\n"
    end
  end
end

Exception.prepend DidYouMean::Formatter
# And Exception#full_message would call #message_with_extras` and not #message.
# The default Exception#message_with_extras just calls #message.
```

I'm not sure which is better, WDYT?

Do we want to also have "message with extras" for causes' messages in the full_message output?
If so the #message_with_extras approach seems better, because that's quite difficult to make it work with full_message.

Of course #message_with_extras is just a name, maybe we can find a better one.

(We could also add kwargs to Exception#message but that would be incompatible with all definitions of message which don't accept any argument, hence doesn't seem possible.)

#10 - 11/11/2021 11:39 AM - Eregon (Benoit Daloze)

- Description updated

#11 - 11/11/2021 11:39 AM - Eregon (Benoit Daloze)

- Description updated

#12 - 11/11/2021 11:39 AM - Eregon (Benoit Daloze)

- Description updated

#13 - 11/11/2021 02:06 PM - mame (Yusuke Endoh)

Eregon (Benoit Daloze) wrote in #note-7:

> mame (Yusuke Endoh) wrote in #note-6:
>
> > Does this proposal include that the ruby interpreter should use #full_message to show the error information? This is an incompatibility, is it acceptable?
>
> Yes, let's fix that.

Thanks. I think that this is the most important part of this proposal. It should be clearly explained in the description. Actually, I don't think that the implementation of error_highlight and did_you_mean is so important.

I have no strong opinion, but if the change is accepted, I will be happy to change the implementation of error_highlight.

Also, the motivation of this proposal is not clear to me. The implementation of error_highlight and did_you_mean is indeed dirty, but I don't see how it matters to end users. I guess it may matter when a user calls Exception#message, but anyway, it should also be clearly explained the description.

@Eregon (Benoit Daloze) changed the code snippet in the description, which made the proposal clear to me. However, in regard to the backtrace, eregon's proof-of-concept looks much more difficult than the code in the original description. I believe that it would be good to implement it and to confirm that the proposal actually works before the discussion.

Because this ticket started with unclear description, and the discussion so far is also unclear. How about closing this ticket and creating a new ticket with a clear explanation of the motivation, the proposal, and a proof-of-concept patch if any?

#14 - 11/14/2021 12:40 PM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote in #note-13:

> Also, the motivation of this proposal is not clear to me.

Some important motivation is explained in https://bugs.ruby-lang.org/issues/18296#note-4:

- Callers can choose whether they want did_you_mean/error_highlight (individually) in the exception message or not, and they have a way to get the original exception message (just Exception#message would be the original message).
- It makes it possible to choose more dynamically than global --disable-did_you_mean/error_highlight. Given e.g. error_highlight has a non-trivial performance impact, it seems useful to be able only get this output in some situations where the caller might know whether it is useful.
- did_you_mean/error_highlight can find out about options given to full_message such as highlight:, or even introduce their own additional options (keyword arguments).
  For instance error_highlight could use highlight: to decide whether to use ANSI sequences to colorize the failing call, or to use ^^^^ on the next line (I know @ioquatix (Samuel Williams) cares about this flexibility).
  This can be useful e.g. to have different output for Exception#inspect (don't want ANSI escape sequences, and probably not extra information) and printing the exception to a terminal (all information and ANSI escape sequences). Some test harness might choose different trade-offs, and this lets them choose.
- It's a cleaner integration than overriding #message, it can allow more gems to customize exceptions and there is no problem if some exception subclasses override #message.

I agree it would be best to reopen a ticket with everything needed in the description.
I think mostly we need a better name than message_with_extras in https://bugs.ruby-lang.org/issues/18296#note-9.
Maybe #decorated_message?

Also if you have any feedback on https://bugs.ruby-lang.org/issues/18296#note-9 that would be useful to make a better proposal.

#15 - 11/15/2021 04:19 AM - yuki24 (Yuki Nishijima)

While I believe this seems like a good direction, the conversation seems a bit scattered. Could one of you summarize the proposal?

#16 - 11/17/2021 05:44 PM - jeremyevans0 (Jeremy Evans)

- Tracker changed from Bug to Feature

- Backport deleted (2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN)

#17 - 11/18/2021 07:34 AM - matz (Yukihiro Matsumoto)

I understand the motivation. But as @yuki24 (Yuki Nishijima) stated, I also have a hard time understanding the issue. Could you summarize the current and latest proposal?

Matz.

#18 - 11/23/2021 07:51 PM - Dan0042 (Daniel DeLorme)

From what I understand, one motivation of this proposal is to have the "did you mean" be part of #full_message but not #message or #inspect. So if we have putz rescue p($!) then

```
==current==
#<NameError: undefined local variable or method `putz' for main:Object
Did you mean?  putc
               puts>

==with #18170==
#<NameError:"undefined local variable or method `putz' for main:Object\nDid you mean?  putc\n               pu
ts">
```

```
==desired==
#<NameError: undefined local variable or method `putz' for main:Object>
```

Did I understand correctly?

#19 - 11/30/2021 02:36 PM - Eregon (Benoit Daloze)

- Related to Feature #18370: Call Exception#full_message to print exceptions reaching the top-level added

#20 - 12/15/2021 06:15 PM - Dan0042 (Daniel DeLorme)

I'm interested in this feature because I'm uncomfortable with the current situation of sticking everything into Exception#message and losing access to the original message.

In #18194 I think everyone has agreed that

> an error message might be formatted for both the terminal and a log file, which have different formatting requirements.

(and also html formatting via Rack::ShowExceptions where all whitespace in the message is collapsed.)
Overall I think this full_message idea is really the best way to achieve these different formatting requirements. And in particular this:

> we'd need a new method that's basically the same as full_message but always without the backtrace

is basically the same thing as Exception#detailed_information that @mame (Yusuke Endoh) suggested in #18194#note-4

So if I try to summarize all this, there's a need for three different kinds of "message": (names are only for illustration)

1. original_message => The original string provided to Exception.new; there really should be *some* way to get the unmodified message
2. detailed_message(**opts) => The original_message plus additional info (did_you_mean and error_highlight), optionally with terminal highlighting; opts allows to override the global on/off defaults for highlighting and did_you_mean and error_highlight. Alternatively it could be detailed_information(**opts), just the info without the original_message.
3. full_message(**opts) => Same as above, but with backtrace as well. Using this for the top-level formatter would improve maintainability.

There's a bit of an issue concerning what Exception#message should be.
a) Ideally it would be the unmodified original_message. So detailed_ or full_message must be used to get error_highlight information.
b) @mame (Yusuke Endoh) really wants error_highlight's information in Sentry's error reports, which only uses message at the moment. So it can be either a no-opts version of detailed_message, or else detailed_message could actually be called just message.

It seems to me that since error_highlight is only for NameError, and NameError doesn't happen so often in production, and Sentry is for gathering error reports in production... the value of putting error_highlight info in Exception#message for the sake of Sentry is somewhat limited. IMHO, not enough to be worth changing the semantics of Exception#message so much. I feel that option a) above is a better, cleaner choice in the long run.

Respectfully.

#21 - 12/15/2021 08:49 PM - ioquatix (Samuel Williams)

@Dan0042 I basically agree with your assessment and support (a) and in addition I'd prefer if we don't introduce inconsistent formatting for quoted messages, i.e. #<NameError:"..."> should definitely have a space between the : and ", i.e. #<NameError: "...">. Otherwise it just feels like one step forward and one step backwards w.r.t. consistency.

#22 - 01/01/2022 11:36 AM - Eregon (Benoit Daloze)

- Related to Feature #18438: Add `Exception#additional_message` to show additional error information added

#23 - 09/23/2022 07:42 AM - ioquatix (Samuel Williams)

- Status changed from Open to Closed

I'm satisfied that this has been fully addressed and the design is much better. Thanks everyone.