No easy way to format exception messages per thread/fiber scheduler context.

In the new error highlighting gem, formatting exception messages appears to be per-process which is insufficiently nuanced for existing use cases.

As in:

```ruby
class TerminalColorFormatter
  def message_for(spot)
    # How do we know the output format here? Maybe it's being written to a log file?
    "..."
  end
end

ErrorHighlight.formatter = TerminalColorFormatter.new
```

But we won't know until the time we actually write the error message whether terminal codes are suitable or available. Or an error message might be formatted for both the terminal and a log file, which have different formatting requirements. There are many consumers of error messages an some of them produce text, or HTML, or JSON, etc.

Because of this design we are effectively forcing everyone to parse the default text output if they want to do any kind of formatting, which will ossify the format and make it impossible in practice for anyone to use anything but the default ErrorHighlight.format. For what is otherwise a really fantastic idea, this implementation concerns me greatly.

I would like us to consider introducing sufficient metadata on the exception object so that complete formatting can be implemented by an output layer (e.g. logger, terminal wrapper, etc). This allows the output layer to intelligently format the output in a suitable way, or capture the metadata to allow for processing elsewhere.

In addition, to simplify this general usage, we might like to introduce Exception#formatted_message.

In order to handle default formatting requirements, we need to provide a hook for formatting uncaught exceptions. This would be excellent for many different use cases (e.g. HoneyBadger type systems), and I suggest we think about the best interface. Probably a thread-local with some default global implementation makes sense... maybe even something similar to at_exit { ... $! ... }.

Related issues:
Related to Ruby master - Feature #18296: Custom exception formatting should o... Open
I think that we need to clarify the issue first. An error message might be formatted for both the terminal and a log file, which have different formatting requirements.

I completely agree with this. For the terminal, we may want to see colors by escape sequences, and detailed explanation like did_you_mean and error_highlight. For a log file, a simple one-line message is often enough. This issue is also pointed by byroot (Jean Boussier) in https://github.com/ruby/error_highlight/pull/10.

The subject of this ticket proposes to make the configuration per-thread, but I don't think that it is a good solution against the issue. I think we want to use the different formats even in a simple thread application. How about focusing on only this issue of tty and a log file? Mixing other ideas like "per-thread" and "metadata" would complicate things.

My naive idea is to add to the interpreter a new method Exception#detailed_information that returns an additional text to help users understand the error, to let did_you_mean and error_highlight define Exception#detailed_information instead of overriding Exception#message, and to let the interpreter show Exception#message and Exception#detailed_information in turn when an uncaught exception occurs. For a log file, using only Exception#message would be enough.

Just a quick mention that error_highlight is a CRuby-internal gem. So any new API here should not depend on error_highlight or ErrorHighlight.

Eregon (Benoit Daloze) wrote in #note-5:

Yes, of course. I'm unsure why ioquatix (Samuel Williams) mentioned only error_highlight, but this issue is never only about error_highlight, but also about did_you_mean, at least.

Also, gem writers may use the new API to show supplemental information about an error that their gem raises. Here is a trivial (and uninteresting) example.

```ruby
class ConnectionError < StandardError
  def message
    "failed to connect #{@host}"
  end

  def detailed_information
    "try \"ping #{@host}\""
  end
end

test.rb:1:in `<main>': failed to connect 192.168.0.1 (ConnectionError)
  try "ping 192.168.0.1"
    from /home/mame/local/lib/ruby/gems/3.0.0/gems/irb-1.3.8.pre.8/exe/irb:11:in `<top (required)>'
    from /home/mame/local/bin/irb:23:in `load'
    from /home/mame/local/bin/irb:23:in `<main>'
```

I didn't think about other cases, I was mostly concerned about the global hook for formatting messages. When I asked about it mame (Yusuke Endoh) said it's not possible to have context-aware formatting. Yes, this probably applies to did_you_mean gem too. I don't think I proposed anywhere about depending on a specific gem as a hard requirement.

I remembered one significant concern: application monitors like Sentry. I want to see error_highlight's information in Sentry's error reports. This is one of the important motivations of error_highlight. I have no idea how such an application monitor service captures the error log, but if they uses Exception#message, my proposal to separate #message and #detailed_information will not work unless they supports the new API.

I understand your concern. Well, I think Sentry should update their implementation if they want extended information. If you believe they can't do it, can you ask them why?

There are other APM services besides Sentry. Now I wonder if we should introduce a new API for a log file, say #oneline_message, that returns a simple one-line message, instead of changing #message.
You think it's an advantage to change the default exception message to include additional formatted details. But I'm not so sure about that. Another way of looking at it is because there are other APMs consuming this information, such a change may be unexpected/incompatible.

Let's take Honeybadger as an example. They deduplicate errors by class and message. I don't know their exact algorithm, but including extra information in the Exception#message may cause them to incorrectly deduplicate errors. Changes in the loaded files might change did_you_mean output.

Another example I can think of would be the performance impact. By default, did_you_mean implementation or source code highlighting might be computationally expensive (loading source code, etc). In a production environment this might not be desirable, or may not even be useful.

I think what's more important is an interface that makes sense. Ruby has no mechanism for formatting or catching top level exceptions, but this IS a critical feature for APMs, not only APMs but a lot of applications would like to provide better formatting for exceptions which bubble all the way up, either in a thread or globally.

I think having a top level hook for exceptions makes total sense.

e.g.

```ruby
Exception.unhandled do |exception|
  $stderr.puts exception.formatted_message
end

# or

Exception.unhandled do |exception|
  $stdout.puts Terminal.format(exception)
end
```

Threads should defer to the main thread if otherwise unset.

With this in place, any kind of formatting becomes really trivial especially if we define hooks for it. APMs can either opt in, or we can even make it the default, e.g.

```ruby
Exception.unhandled do |exception|
  $stdout.puts exception.full_message(highlight: $stdout.tty?)
end
```

For APMs, they can opt into the same formatting interface if they so desire. I think the assumption is highlight: true gives xterm256 control sequences, so even web applications can consume this and convert to HTML relatively easily.

Gems like did_you_mean or error_highlighter should extend full_message or we should have some internal interface with extension points, for example:

```ruby
class Exception
  # The supplied message.
  def message
    end

  def format_summary(output)
    output.print(:title, self.class.name, ":")
    output.print(:message, self.message)
  end

  def format_backtrace(output)
    self.backtrace.each do |line|
      output.print(:backtrace, line)
    end
  end

  def format(output)
    format_summary(output)
    format_backtrace(output)
  end
end
```

My preference is to inject an output wrapper which can handle styling in a simple but generic way. It's better than trying to use control sequences because it naturally supports any kind of output format (text, log, json, html, etc).

#12 - 09/29/2021 12:43 AM - ioquatix (Samuel Williams)
Whoops, accidentally overwrote title :p

#13 - 09/29/2021 05:35 AM - mame (Yusuke Endoh)
- Backport deleted (2.6: UNKNOWN, 2.7: UNKNOWN, 3.0: UNKNOWN)
- Tracker changed from Bug to Feature

I'm moving this ticket to the Feature tracker.

ioquatix (Samuel Williams) I'd like you to sort out the problem statement, and put it into a concrete proposal. I just don't know what to do.

byroot (Jean Boussier) I assume you are aware of this issue. Do you have any opinion?

#14 - 09/29/2021 10:10 AM - Eregon (Benoit Daloze)
ioquatix (Samuel Williams) wrote in #note-11:

I think having a top level hook for exceptions makes total sense.

Maybe, but that won't be of any use for e.g. web applications which need to catch and format the exception before it kills the thread. The current default to print Exception#full_message for exceptions reaching the top of a thread seems already good enough to me.

For formatting maybe full_message is already good enough? If the usage needs no colors it can say highlight: false, if it wants some it can indeed relatively easily parse them.

I think it's already clear that Exception#message itself should not use ANSI escape sequences (for color/bold/etc).

#15 - 11/10/2021 11:50 AM - Eregon (Benoit Daloze)
- Related to Feature #18296: Custom exception formatting should override 'Exception#full_message'. added