CommonRuby - Feature #7895

Exception#backtrace_locations to go with Thread#backtrace_locations and Kernel#caller_locations

02/21/2013 06:02 AM - headius (Charles Nutter)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:	Ruby 2.1.0	

Description

Thread#backtrace_locations and Kernel#caller_locations were added in Ruby 2.0.0, but no equivalent method was added to Exception to get backtrace locations. The String format of Exception#backtrace elements makes it difficult to do any inspection of the actual files and lines it reports, so having backtrace_locations would be very useful.

In #7435 ko1 pointed out that Exception defines set_backtrace, which takes an array (presumably of Strings in the same format as Exception#backtrace) and that this would need to be addressed in order to support backtrace_locations. I propose that if you set_backtrace, you are already breaking the ability to get structured Location elements, so set_backtrace should cause backtrace_locations to return nil or an empty array (probably an empty array, so it can always be expected to be non-nil).

We could consider also adding set_backtrace_locations as a way to set Location elements into the exception.

An example script where backtrace_locations would be very useful is here (avoiding the need to regexp match to get file and line number): https://gist.github.com/headius/4999244

Related issues:

Related to Ruby trunk - Feature #8960: Add Exception#backtrace_locations	**Assigned**

History

#1 - 02/21/2013 06:53 AM - ko1 (Koichi Sasada)

(2013/02/21 6:02), headius (Charles Nutter) wrote:

> I propose that if you set_backtrace, you are already breaking the ability to get structured Location elements, so set_backtrace should cause backtrace_locations to return nil or an empty array (probably an empty array, so it can always be expected to be non-nil).
>
> We could consider also adding set_backtrace_locations as a way to set Location elements into the exception.

I want to agree with you. But I feel there are worries.

(a) Exception#set_backtrace are used by frameworks

Maybe Exception#set_backtrace are used by frameworks such as Rails. So
that your proposal "if Exception#set_backtrace was called, disable
Exception#backtrace_locations" seems bad idea because 99% of people
works on Rails (*1) and 99% (*1) of people can't use
Exception#backtrace_locations.

*1: joking.

(b) We can't use Exception#backtrace_locations in library

If you write a library using Exception#backtrace_locations, this library
can't use with this library with programs using Exception#set_backtrace.
As I pointed out by (a), we can't mix with frameworks.

Other options:

(1) Make Exception#set_backtrace parse each backtrace lines.
Problem:
(1-p1) Parsing is difficult. Path format is OS dependent.
(1-p2) How to parse not well-formed line?

(2) Deprecate set_backtrace and replace it with set_backtrace_locations
Problem:
(2-p1) Compatibility!!!

(1) with heuristics parsing technique is pragmatic?

--
// SASADA Koichi at atdot dot net

#2 - 02/21/2013 08:53 AM - headius (Charles Nutter)

On Thu, Feb 21, 2013 at 8:36 AM, SASADA Koichi ko1@atdot.net wrote:

> I want to agree with you. But I feel there are worries.

How about #backtrace/#set_backtrace and
#backtrace_locations/#set_backtrace_locations are completely
separate representations of the backtrace? That preserves backward
compatibility while providing a path forward to the more structured
backtrace form. I think it's reasonable to expect that the
unstructured backtrace form doesn't update or break the structured
form.

- Charlie

#3 - 02/21/2013 03:53 PM - ko1 (Koichi Sasada)

(2013/02/21 8:50), Charles Oliver Nutter wrote:

> On Thu, Feb 21, 2013 at 8:36 AM, SASADA Koichi ko1@atdot.net wrote:
>
>> I want to agree with you. But I feel there are worries.
>
> How about #backtrace/#set_backtrace and
> #backtrace_locations/#set_backtrace_locations are completely
> *separate* representations of the backtrace? That preserves backward
> compatibility while providing a path forward to the more structured
> backtrace form. I think it's reasonable to expect that the
> unstructured backtrace form doesn't update or break the structured
> form.

I think it is reasonable.

(3) Separate Exception#bactrace and Exception#backtrace_locations
Problems:
(3-p1) which should output as an error backtrace when interpreter dies.
(3-p2) may introduce confusion?

Other comments and ideas are welcome.

--
// SASADA Koichi at atdot dot net

#4 - 02/21/2013 08:53 PM - headius (Charles Nutter)

On Thu, Feb 21, 2013 at 5:47 PM, SASADA Koichi ko1@atdot.net wrote:

> I think it is reasonable.
>
> (3) Separate Exception#bactrace and Exception#backtrace_locations
> Problems:
> (3-p1) which should output as an error backtrace when interpreter dies.

Good question! I was wondering about this too. Does #backtrace remain
the canonical stack trace source, or do we move to
#backtrace_locations? Perhaps there's a heuristic necessary here to
support forward migration. To be honest, the ability to replace the
stack trace has always bothered me a bit, since it means a program
could potentially hide where an error occurs. I feel like if the
exception bubbles all the way out, you should get a proper trace. I'm
probably missing the use case where you want to provide a customized
trace, though.

> (3-p2) may introduce confusion?

I think this is unavoidable, to some extent. We would like to migrate
people toward using the structured backtrace, but it is not
backward-compatible with the unstructured backtrace. So...we make a
decision: prioritize the structured version or prioritize the
unstructured version? I tend toward the ideal future and would
emphasize the structured version as much as possible...meaning that
interpreter death uses backtrace_locations, unless set_backtrace has
been used to provide the old-style trace. As I say above, there's
probably a reasonable heuristic possible here based on what we want to
encourage people to use.

Thinking through this a bit...

- Base all backtrace output on the backtrace_locations content unless otherwise indicated.
- If you specify a backtrace at construction time, inspect whether it is string content (old style) or Location content (new style).
- Any modification of the string-based #backtrace overrides #backtrace_locations for printing purposes. If #backtrace has not been set, either via constructor or via #set_backtrace, then we generate it based on #backtrace_locations.

Basically, #backtrace_locations is the primary source of backtrace
data, unless the user provides an old-style #backtrace or uses
#set_backtrace to do something custom.

Of course #set_backtrace_locations should enforce that the array
provided is actually Location objects.

- Charlie

#5 - 03/09/2013 05:08 PM - headius (Charles Nutter)

What's next here? I'm about to implement caller_locations in JRuby, so I'm close to being able to prototype this feature request too.

#6 - 03/11/2013 10:53 AM - ko1 (Koichi Sasada)

(2013/03/09 17:08), headius (Charles Nutter) wrote:

> What's next here? I'm about to implement caller_locations in JRuby, so I'm close to being able to prototype this feature request too.

Now, only two attendees (you and I).
I want to get other opinions, especially users of #set_backtrace().

--
// SASADA Koichi at atdot dot net

#7 - 03/12/2013 05:03 AM - headius (Charles Nutter)

For now I have implemented it in JRuby.

Here's the commit: https://github.com/jruby/jruby/commit/281ead709e1e855b552a42e86e48f8f91ca8ebb5

It works simply enough; Exception#backtrace_locations always reflects the original exception trace and (at least for now) it can't be modified.

```
irb(main):004:0> def foo; raise; end
=> nil
irb(main):005:0> def bar; 1.times { foo }; end
=> nil
irb(main):006:0> ex = begin; bar; rescue; $!; end
=> RuntimeError
irb(main):007:0> ex.backtrace_locations.each {|loc| puts "method: #{loc.label}, line: #{loc.lineno}"}
method: foo, line: 4
method: bar, line: 5
method: times, line: 271
method: bar, line: 5
method: evaluate, line: 6
```

#8 - 03/13/2013 12:57 AM - kallistec (Daniel DeLeo)

Hi, I work on Opscode's Chef, a server configuration framework. I have an interest in both parts of this topic. In Chef, we use set_backtrace but we also parse backtrace lines to find relevant code in our error messages.

When we use set_backtrace, it is because we are catching an exception and then re-raising as either the same exception class or as a different exception class. Our goal here is to add context to exceptions that would otherwise be very difficult for the end user to understand.

We somewhat recently rewrote our error messaging to be more user friendly. To do this, we need to process backtraces and filter out framework code (leaving only user code) and then use regular expressions to find the relevant line numbers. Having structured backtrace data would be much better than what we're doing now.

For our use case, we need a way to set backtrace_locations from one exception to another. We currently only use set_backtrace immediately after creating an exception, so it would be fine if we could pass the needed data into the constructor, as long as there is a reasonable way to make the method signatures in current ruby compatible via monkey patching or #respond_to?

#9 - 03/16/2013 02:12 AM - headius (Charles Nutter)

kallistec (Daniel DeLeo) wrote:

> Hi, I work on Opscode's Chef, a server configuration framework. I have an interest in both parts of this topic. In Chef, we use set_backtrace but we also parse backtrace lines to find relevant code in our error messages.
>
> When we use set_backtrace, it is because we are catching an exception and then re-raising as either the same exception class or as a different exception class. Our goal here is to add context to exceptions that would otherwise be very difficult for the end user to understand.

So, as long as you'd be happy with Thread::Backtrace::Location's elements, it should just be a matter of creating a new Array of Location objects and passing it to an hypothetical set_backtrace_locations (or hey, let's get crazy and call it backtrace_locations= perhaps?)

Some specifications I proposed before:

- If an exception has had no modifications applied, #backtrace and #backtrace_locations will be the same content (in different form, obviously).
- If an exception has had #set_backtrace_locations (#backtrace_locations=) called, #backtrace and #backtrace_locations will reflect the new content.
- If an exception has had #set_backtrace called, #backtrace will reflect the new content but #backtrace_locations will not (due to the impossibility of guaranteeing #set_backtrace lines will parse into Location objects).

So basically #backtrace_locations is the future path for dealing with exception data and #backtrace (probably deprecated in future) only piggy-backs off it.

Seem reasonable?

#10 - 03/17/2013 05:53 AM - Anonymous

On Sat, Mar 16, 2013 at 02:12:03AM +0900, headius (Charles Nutter) wrote:

> Issue #7895 has been updated by headius (Charles Nutter).
>
> kallistec (Daniel DeLeo) wrote:
>
>> Hi, I work on Opscode's Chef, a server configuration framework. I have an interest in both parts of this topic. In Chef, we use set_backtrace but we also parse backtrace lines to find relevant code in our error messages.
>>
>> When we use set_backtrace, it is because we are catching an exception and then re-raising as either the same exception class or as a different exception class. Our goal here is to add context to exceptions that would otherwise be very difficult for the end user to understand.

Rails uses set_backtrace, and is basically in the same boat.

> So, as long as you'd be happy with Thread::Backtrace::Location's elements, it should just be a matter of creating a new Array of Location objects and passing it to an hypothetical set_backtrace_locations (or hey, let's get crazy and call it backtrace_locations= perhaps?)
>
> Some specifications I proposed before:
>
> - If an exception has had no modifications applied, #backtrace and #backtrace_locations will be the same content (in different form, obviously).
> - If an exception has had #set_backtrace_locations (#backtrace_locations=) called, #backtrace and #backtrace_locations will reflect the new content.
> - If an exception has had #set_backtrace called, #backtrace will reflect the new content but #backtrace_locations will not (due to the impossibility of guaranteeing #set_backtrace lines will parse into Location objects).
>
> So basically #backtrace_locations is the future path for dealing with exception data and #backtrace (probably deprecated in future) only piggy-backs off it.
>
> Seem reasonable?

This is something we could work with. :-)

--
Aaron Patterson
http://tenderlovemaking.com/

#11 - 04/17/2013 02:54 AM - headius (Charles Nutter)

What's the next step here? Do we need a patch to proceed? It seems like this feature is universally liked.

#12 - 04/17/2013 04:59 AM - enebo (Thomas Enebo)

Has anyone talked about .backtrace returning a subclass of String (or singleton) which duck-types to the same methods as Location? It is arguably hacky and would not work for === but it would still allow them to be used in a compatible way. Or maybe .backtrace_locations provides this special string. If set_backtrace is used it will just be dressed up with this methods it .backtrace (or .backtrace_locations) is called.

#13 - 04/17/2013 07:32 AM - headius (Charles Nutter)

I think that was tossed around on IRC or somewhere. It would certainly work, but it feels a little too hacky to me.

You'd also never be able to rely on the String objects from #backtrace actually being Location-like, since set_backtrace would clear them. I still feel like this is the best plan:

- backtrace_locations is the master copy of backtrace data
- set_backtrace_locations can be used to modify it (optional...I have no strong opinion)
- backtrace generates based on backtrace_locations unless explicitly set with set_backtrace
- setting backtrace to nil reverts it to using backtrace_locations (maybe?)

#14 - 04/24/2013 06:22 AM - headius (Charles Nutter)

I should mention that Exception#backtrace_locations is implemented in JRuby on master (1.7.4) in ruby 2.0 mode. I had not intended to add it before MRI, but it's there. We may remove it before release, but this would give folks a change to play with it on a standard build (jruby master).

#15 - 04/28/2013 07:59 AM - ko1 (Koichi Sasada)

(2013/04/17 2:54), headius (Charles Nutter) wrote:

> What's the next step here? Do we need a patch to proceed? It seems like this feature is universally liked.

I don't have any objection about it now.

Patches are welcome! :)

Make another issue for CRuby/trunk?

--

// SASADA Koichi at atdot dot net

#16 - 09/27/2013 08:12 PM - headius (Charles Nutter)

I have filed https://bugs.ruby-lang.org/issues/8960 to get this feature added to MRI. It is already available in JRuby.

#17 - 10/01/2013 09:39 AM - headius (Charles Nutter)

- Target version set to Ruby 2.1.0