Ruby - Feature #15778

Expose an API to pry-open the stack frames in Ruby

04/20/2019 02:27 AM - gsamokovarov (Genadi Samokovarov)

Status:	Assigned
Priority:	Normal
Assignee:	ko1 (Koichi Sasada)
Target version:	

Description

Hello,

I'm the maintainer of the web-console (https://github.com/rails/web-console/) gem, where one of our features is to jump between the frames in which an error occurred. To accomplish this, I currently use the Debug Inspector CRuby API. I think we should expose this functionality in Rubyland, so tools like web-console don't need to resort to C code for this. This also makes it quite harder for me to support different implementations like JRuby or TruffleRuby as everyone is having a different way to create Ruby Binding objects that represent the frames.

Here the API ideas:

Add Thread::Backtrace::Location#binding method that can create a binding for a specific caller of the current frame. We can reuse the existing Kernel.caller_locations method to generate the array of Thread::Backtrace::Location objects. We can optionally have the Kernel.caller_locations(debug: true) argument if we cannot generate the bindings lazily on the VM that can instruct the VM to do the slower operation.

- Thread::Backtrace::Location#binding returns Binding|nil. Nil result may mean that the current location is a C frame or a JITted/optimized frame and we cannot debug it.

We can also expose the DebugInspector API directly, as done in the https://github.com/banister/debug_inspector gem, but for tools like web-console, we'd need to map the bindings with the backtrace, as we cannot generate Bindings for every frame (C frames) and this needs to be done in application code, so I think the Thread::Backtrace::Location#binding is the better API for Ruby-land.

Such API can help us eventually write most of our debuggers in Ruby as right now we don't have a way to do Post-Mortem debugging without native code or even start our debuggers without monkey-patching Binding.

I have presented this idea in a RubyKaigi's 2019 talk called "Writing Debuggers in Plain Ruby", you can check-out the slides for more context: http://kaigi-debuggers-in-ruby.herokuapp.com.

Related issues:

Related to Ruby - Bug #18780: Incorrect binding receiver for C API rb_eval_st...	**Closed**

History

#1 - 04/20/2019 10:21 AM - chrisseaton (Chris Seaton)

Is your idea that all exception backtraces always come with the bindings attached? Or just when you call Kernel#caller_locations you then get the bindings attached?

#2 - 04/20/2019 02:36 PM - Eregon (Benoit Daloze)

- Description updated

- Assignee set to ko1 (Koichi Sasada)

#3 - 04/20/2019 02:43 PM - Eregon (Benoit Daloze)

I discussed with @gsamokovarov at RubyKaigi and I think this is a good idea and Thread::Backtrace::Location#binding is a natural fit for it.

chrisseaton (Chris Seaton) wrote:

> Is your idea that all exception backtraces always come with the bindings attached? Or just when you call Kernel#caller_locations you then get the bindings attached?

I think Thread::Backtrace::Location are only created by Kernel#caller_locations, Thread#backtrace_locations and Exception#backtrace_locations.

The Binding should only be set if debug: true (or e.g., binding: true) is passed, as it incurs additional overhead.

For Exception#backtrace_locations this is however problematic as that backtrace is captured when raising the exception, not when calling Exception#backtrace_locations. Therefore, I think Exception#backtrace_locations should never provide bindings.

#4 - 04/20/2019 03:12 PM - chrisseaton (Chris Seaton)

So this could be used to implement Binding.of_caller as caller_locations(1, 1, debug: true).first.binding?

#5 - 04/20/2019 03:49 PM - gsamokovarov (Genadi Samokovarov)

Yeah, it could be used to implement Binding.of_caller, but if we have the proposed API, we may not need actual Binding.of_caller as the tools can use Kernel.caller_locations(debug: true). Our caller can be a C Frame in the case of CRuby and we may not be able to create the Ruby Binding for it, so IMO, the API should have a good signal for that. Returning a nil Thread::Backtrace::Location#binding can mean: "I cannot debug that frame". This can mean different things based on implementations: C Frames, optimized frames, etc.

IMO, the better way to get the frames is with Kernel.caller_locations(debug: true) and not Binding.of_caller(2) as in the binding of caller case, an application may need to map bindings to traces, like we currently do in tools similar to rails/web-console and this would still need custom code to do on the tool side. If the API itself solves that problem, that would be great!.

#6 - 04/24/2019 07:46 AM - ko1 (Koichi Sasada)

Now we are not publishing Ruby API because we shouldn't use this kind of API on application code.
For example, if people rely on Binding.of_caller, we can't use delegation code freely.
I understand debugger like tools require the API, so this is why we prepare debug_inspector C-API (and debug_inspector gem).
This is current situation.

#7 - 04/26/2019 01:55 PM - Eregon (Benoit Daloze)

> This is current situation.

Thanks for the summary.

> For example, if people rely on Binding.of_caller, we can't use delegation code freely.

I think it's fair enough for usages of Binding.of_caller to have to care about this.

@ko1 (Koichi Sasada) The debug_inspector gem just makes the Bindings of the stack available to Ruby code, so if somebody wants to use them in application code they already can (but agreed it's very rarely good to do so).
My opinion is it's not valuable to "hide" such capabilities by moving them to C extensions, because they are still easy to access if one wants them (i.e., it's easy to write a C ext or add debug_inspector as dependency).

The fact that binding_of_caller is used in the wild shows that it's not because a C-API is needed that it's not or less used.
https://github.com/banister/binding_of_caller/network/dependents

I think rather we should just document these methods are meant for debugging and might slow down execution significantly, and therefore should not be used in application code.
Maybe a good way too to indicate that further than documentation is having a clear namespace, such as e.g., ::DebugInspector or ::Debugging.
Then it's fairly clear this is only meant for debugging.

https://github.com/banister/debug_inspector#usage clearly shows having Thread::Backtrace::Location#binding is a natural fit.
Is there any use-case for frame_iseq? That's obviously going to be MRI-specific, isn't it?
Can frame_class be derived from the Binding? Is it like Module.nesting.first?

I think we should really aim to have portable debugging APIs if we want Ruby tooling to improve.
And therefore, they must be defined in Ruby (it doesn't make much sense for JRuby to implement a C API).

@ko1 (Koichi Sasada) What do you think of the new API caller_locations(debug: true) + Thread::Backtrace::Location#binding, doesn't it make perfect sense?
We would of course document that the :debug keyword should only be used for debugging, and Thread::Backtrace::Location#binding would raise e.g., an ArgumentError if :debug is not given.

#8 - 04/29/2019 04:22 PM - gsamokovarov (Genadi Samokovarov)

As a case study, we may look at Python. They have such an API for 20+ years and I don't think anyone explicitly complained it makes Python slow or dangerous to use. The API is sys._getframe (https://docs.python.org/3/library/sys.html#sys._getframe). There are also traces of such API by having linked lists to previous frames in tracebacks as well.

#9 - 05/15/2019 09:10 AM - Eregon (Benoit Daloze)

One thing we can do in any case for TruffleRuby is implementing the debug_inspector C API.

However, that doesn't let JRuby implement it, and hiding APIs in C doesn't seem a good way to communicate "should only be used for debugging".

As the debug_inspector README example shows, the API would be much more natural under Thread::Backtrace::Location, and easier to use for debuggers.
The current C-API feels suboptimal/hard-to-use by manually merging Kernel#backtrace_locations and passing indices to the debug inspector.
It's also inefficient, by walking N + N*(N+1)/2 frames instead of just N with the proposed Ruby API.

#10 - 07/14/2019 06:05 AM - ko1 (Koichi Sasada)

> ko1 (Koichi Sasada) What do you think of the new API caller_locations(debug: true) + Thread::Backtrace::Location#binding, doesn't it make perfect sense?

I heard that one advantage that current debug_inspector gem has is we can declare the usage of this API in Gemfile. It means that we can avoid some kind of vulnerable feature in production explicitly.

#11 - 07/31/2019 09:37 AM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote:

> I heard that one advantage that current debug_inspector gem has is we can declare the usage of this API in Gemfile. It means that we can avoid some kind of vulnerable feature in production explicitly.

I see. Although the rb_debug_inspector_open() C API is still there (and could be called, e.g., via Fiddle I imagine).

Proc#binding in Ruby also gives access to local variables potentially far from the current method.
Similarly, TracePoint#binding already allows to read values from all over the program.
That's why I think it's necessary to assume that the attacker cannot call arbitrary methods.

So I think having the caller_locations(debug: true) API is safe enough.
If the attacker can call caller_locations and pass it the extra :debug argument, then I think anyway all is already lost, they might as well #eval, #system, #exit, etc.

I'd like to introduce this new keyword argument for caller_locations, it's one of the main features missing, and hiding it in the C API is not much of a safety, but it makes it inconvenient to use, inefficient (see above) and not portable (e.g., cannot be implemented on JRuby as a C API).

#12 - 08/29/2019 06:24 AM - duerst (Martin Dürst)

During today's meeting, it was mentioned that production deployments may not want to include this functionality.

To do this, one solution would be to make this functionality available as a bundled gem.

#13 - 06/12/2022 11:26 AM - Eregon (Benoit Daloze)

- Related to Bug #18780: Incorrect binding receiver for C API rb_eval_string() added

#14 - 12/30/2022 04:38 PM - st0012 (Stan Lo)

I think such feature has a great potential for production use too, especially for error monitoring. For example, both Rollbar and Sentry tries to support local variables capturing with TracePoint:

- Rollbar's implementation (captures all frames' data)
- Sentry's implementation (captures only the current frame's data)

But as we all know, using TracePoint in production is usually risky. So this feature in both services remained "experimental" and is disabled by default, even though it'd be super helpful to users. I think in a sense it proves @Eregon's point:

> hiding it in the C API is not much of a safety, but it makes it inconvenient to use, inefficient (see above) and not portable (e.g., cannot be implemented on JRuby as a C API).

If there can be a bundled gem to power usage like:

```
caller_locations(1, 1, debug: true).first.binding
```

it'd level up our developer's debugging experience significantly in both development and production.

Reasons to have a bundled gem then just use binding_of_caller

1. From a business' perspective, it's easier to convince customers adding a language-official library than a community library.
2. We can be confident that it'd work well with Ruby's latest development, especially with YJIT.
3. It'd allow better cross-platform support.

(As a side note, Honeybdger supports it via binding_of_caller, but also disables it by default).

#15 - 01/20/2023 10:49 PM - st0012 (Stan Lo)

Outside of error reporting, having this or similar API will also:

- Allow IRB to display richer exception backtrace. This is helpful when used as binding.irb.
- Allow IRB to implement a bt command that's similar to debug's.
  - Whether to do this is another discussion and depends on other factors, but this API will enable that possibility.
- Allow Rails to print backtrace with local variables in log or error pages.

All of the above can be toggled by checking the gem's presence.

#16 - 01/22/2023 10:16 AM - gsamokovarov (Genadi Samokovarov)

What if we settle for an API and implement and test it first in the debug gem? The debug gem does need/has similar functionality for post-mortem debugging. That way, gems like web-console can depend on the debug standard gem which I would expect more people to have, instead of using custom C extension. Other tooling can get debug APIs from the debug gem as well. It seems like a good place to start because we can test the APIs in real project and get real feedback.

#17 - 01/24/2023 02:35 PM - Eregon (Benoit Daloze)

gsamokovarov (Genadi Samokovarov) wrote in #note-16:

> What if we settle for an API and implement and test it first in the debug gem? The debug gem does need/has similar functionality for post-mortem debugging. That way, gems like web-console can depend on the debug standard gem which I would expect more people to have, instead of using custom C extension. Other tooling can get debug APIs from the debug gem as well. It seems like a good place to start because we can test the APIs in real project and get real feedback.

One concern I have about that is currently the debug gem only supports CRuby, and uses some CRuby-private APIs under RubyVM (I filed https://github.com/ruby/debug/issues/887).
I think that should be fixed at some point as I don't think the usages of RubyVM are critical in the debug gem, but that work is not done and it's hard to be sure until then.

I would advise to use https://github.com/banister/debug_inspector instead of the debug gem if you need this functionality.
That's a gem literally just to expose that API.
It should work fine on TruffleRuby since TruffleRuby support the debug_inspector C API, I'll make a PR there.
Not sure about JRuby, but JRuby devs could probably add support in that gem too.

#18 - 01/25/2023 03:54 PM - st0012 (Stan Lo)

> What if we settle for an API and implement and test it first in the debug gem?

The debug gem already has a method DEBUGGER__.capture_frames that achieves something similar. But I agree with @Eregon (Benoit Daloze) on not using the debug gem for this:

- Pulling the debug gem in means this the API won't be used and shaped for production use.
- The debug gem causes side-effects upon required (e.g. new threads and TracePoints activated in the background). In some cases it changes applications' behaviours and breaks tests for example. You can see more related discussions in this issue.
  - It's probably not a huge risk for most apps, but this is something we should avoid when possible.
- Even without all these issues, it complicates debug gem's role and could complicate its maintenance. I know the proposal is just to experiment it in debug first, but the reality is that we can't make gems move away from a dependency easily.

#19 - 01/30/2023 09:44 PM - st0012 (Stan Lo)

I had a chance to chat with multiple people about this proposal, so I want to share what I found here:

- From chatting with @maximecb (Maxime Chevalier-Boisvert), my understanding is that such feature requires Ruby to keep all the frames on the stack in an accessible format, which will limit the optimisation YJIT could do. This is fine for development, but using it in production with YJIT "might" not be ideal.
- Rails would be happy to use such feature to enrich its exception backtrace in log, which will be helpful in development.
  - In production, in addition to the potential issue with YJIT, more things like data scrubbing needs to be considered.

#20 - 02/05/2023 12:28 PM - Eregon (Benoit Daloze)

st0012 (Stan Lo) wrote in #note-19:

> my understanding is that such feature requires Ruby to keep all the frames on the stack in an accessible format, which will limit the optimisation YJIT could do.

I don't think so since rb_debug_inspector_open() already exposes that, having it as a Ruby method doesn't really change much from an optimization POV AFAIK.
But yes, it's very expensive to get the bindings of all frames, especially with a deep stack e.g. in Rails apps (e.g., it causes deoptimization of the stack

frames).

A thought: maybe the new Thread.each_caller_location could have a debug: true/binding: true keyword argument.
That would make it possible to only get the binding for the first N frames, which is much faster than for all frames.
And it would also encourage to not hold onto such bindings for long (which would otherwise be a sort of leak), good.

#21 - 04/03/2024 03:50 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned