Ruby master - Feature #15778

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

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

Status:	Open
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.

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)

- Assignee set to ko1 (Koichi Sasada)

- Description updated

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

I discussed with gsamokovarov (Genadi Samokovarov) 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.