Ruby master - Feature #7051

Extend caller_locations API to include klass and bindings. Allow caller_locations as a method hanging off Thread.

09/23/2012 03:43 PM - sam.saffron (Sam Saffron)

Status: Closed
Priority: Normal
Assignee: ko1 (Koichi Sasada)
Target version: 2.6

Description
The new caller_locations api allows one to get label, base_label, path, lineno, absolute_path.

I feel this API should be extended some what and cleaned up a bit.

1. I think there should be complete parity with caller and backtrace APIs.

As it stands caller_locations is a fine replacement for caller, however the same code responsible for caller is also responsible for Thread#backtrace. As it stands there is no equivalent Thread#backtrace_locations API that allows one to grab OO backtraces for a thread.

For sampling profilers a common technique is to spin a thread that grabs stack traces from a profiled thread.

I played around with some of this here: https://github.com/SamSaffron/stacktrace/blob/master/test/test_stacktrace.rb

1. I think there should be a way to get the class the method belongs to where possible.

At the moment you need to use something like ruby2ruby to determine the class of a frame in a backtrace. Getting the class is actually very important for auto-instrumentation techniques. A general approach is to grab stack traces, determine locations that seemed to be called a lot, and then instrument them. Trouble is, without a class it is very hard to pick a point to instrument.

I got this working with my stacktrace gem so I think it is feasible.

1. Ability to grab bindings up the call chain

For some advanced diagnostic techniques it is very handy to grab bindings up the chain, that way you can print out local vars in a calling method and so on. It would be handy if this api exposed bindings.

1. Naming and longer term deprecation of caller

The name caller_locations to me feels a bit unnatural, as an API it is superior in all ways to the current caller, yet is far harder to type or remember. I am not really sure what it should be named but caller feels a bit wrong. Also it feels very tied to MRI returning RubyVM::Backtrace::Location , Location seems to me in the wrong namespace. Is JRuby and Rubinius going to be expected to add this namespace? Is this going to be spec?

Has any though been given to longer term deprecation of "caller".

1. What about exceptions?

Exception has no equivalent of caller_locations, for exceptions it makes much more sense to cut down on allocations and store a thinner stack trace object, in particular for the massively deep stacks frameworks like rails have. Then materialize the strings if needed on iteration through Exception#backtrace

History

#1 - 09/23/2012 03:46 PM - sam.saffron (Sam Saffron)
also, lineno seems a bit of an odd name, should it be line_number?

#2 - 09/25/2012 04:02 PM - ko1 (Koichi Sasada)
Hi,

Thank you for your interest about this API.
Summery: I want to discuss about 1 and 4. Other points should be discuss on other ticket.

1. I think there should be complete parity with caller and backtrace APIs. Any idea?

I agree with that. I missed Thread#backtrace completely.

1. I think there should be a way to get the class the method belongs to where possible.

At the moment you need to use something like ruby2ruby to determine the class of a frame in a backtrace. Getting the class is actually very important for auto-instrumentation techniques. A general approach is to grab stack traces, determine locations that seemed to be called a lot, and then instrument them. Trouble is, without a class it is very hard to pick a point to instrument.

I'm not sure what is your example. But I feel caller() is not for this purpose.

1. Ability to grab bindings up the call chain

I need to reject this proposal with the following reasons.

(1) Performance issue. Making bindings for all frames become huge overhead.
(2) caller() is not for this purpose. I know often such requests are proposed. However, it is too powerful and need more discussion. I'll introduce C-level API to achieve this reason for debugger.

1. Naming and longer term deprecation of caller

Yes. We need consider this issue with first issue you suggested. Any idea?

1. What about exceptions?

Exception has no equivalent of caller_locations, for exceptions it makes much more sense to cut down on allocations and store a thinner stack trace object, in particular for the massively deep stacks frameworks like rails have. Then materialize the strings if needed on iteration through Exception#backtrace

I can't understand your point. I think the current trunk implement what you said.

also, lineno seems a bit of an odd name, should it be line_number?

I prefer 'line_no'. However, there are 'IO#lineno' already (pointed out by Kohei [ruby-dev:45686]).

#3 - 09/25/2012 04:53 PM - ko1 (Koichi Sasada)
(2012/09/25 16:02), ko1 (Koichi Sasada) wrote:

Summery: I want to discuss about 1 and 4. Other points should be discuss on other ticket.

Ah, Summer was gone.

--
// SASADA Koichi at atdot dot net

#4 - 10/25/2012 04:22 PM - ko1 (Koichi Sasada)
Ping (I need feed back).

Current idea:

add 'Thread#backtrace_locations(...)' (correspond to Thread#backtrace)

Good name is welcome.

#5 - 10/27/2012 07:43 AM - ko1 (Koichi Sasada)
- Assignee set to ko1 (Koichi Sasada)

#6 - 11/01/2012 05:43 AM - headius (Charles Nutter)
I concur with ko1's objection to having the bindings available. It would introduce massive performance overhead and kill many optimizations we implementers do currently or hope to do in the future.
Having the classes available is also rather limiting; in many cases, JRuby optimizes a given call so that there's no Ruby-related frame at all. This is a key part of optimizing Ruby: eliminating frames and call overhead. Where would you store the class if you want to optimize the frame away? Answer: you can't optimize the frame away.

If there's a real demand for this feature as a diagnostic tool, I could see it being something you have to turn on like tracing.

#7 11/20/2012 10:10 AM - sam.saffron (Sam Saffron)

Apologies it has taken me so long to respond here.

I concur with ko1’s objection to having the bindings available. It would introduce massive performance overhead and kill many optimizations we implementers do currently or hope to do in the future.

I never really intended for this to be something that is on always, just a best-effort, optional thing. (if tracing is enabled you avoid optimisations, if not, you provide a best effort stack to the caller). Same goes for classes, if you can not provide them, do not, its a best effort API for diagnostics.

My point here is that the current API is very limiting. It comes with no option to provide granularity of data collected in the backtrace.

For example: there is no way to peel off a portion of the backtrace.

Often when profiling rails I would like a backtrace that is limited to the frames in my application. To achieve this I would do something like the following:

```ruby
before entering my code

current_depth = caller_locations.length

Problem, allocating a bunch of strings for filenames etc, I do not need

This is very common as Rails starts off with a VERY deep frame after a ton of Rack stuff etc.

in my code

partial_trace = caller_locations(0-current_depth) # Problem, negative depth is not supported - so I am forced to grab a full trace and discard my stuff

Ideally the API should allow you to opt-in to information and pass that through to vm_backtrace_to_ary and vm_backtrace_frame_ary etc. It should also allow for negative params.

A better API would be:

current_depth = caller_locations(0,-1, BacktraceInfo::None)

app_frames = caller_locations(0,-current_depth, BacktraceInfo::Label) # if I only happen to want labels

These changes make it much more efficient to gather stack traces at the correct granularity.

Then you set BacktraceInfo::Default to be Label & Path & LineNo, which allow you to add the fancy best effort Klass & Bindings

Overall this makes it much faster to profile stuff that has deep stack traces, you can cut down on allocations a lot.

As to my exception stuff.

def boom
  raise "boom"
end

begin
  boom
rescue => e
  p e.backtrace
end

["t.rb:2:in 'boom'", "t.rb:6:in'"]

Shouldn't the backtrace be stored in RubyVM::Backtrace::Location objects and then optionally grabbed using backtrace_locations or backtrace depending on how you feel?
I would go as far as saying you should allow for a user flag

```
Exception.backtrace_location_info = BacktraceInfo::Label
```

and possibly

```
Exception.disable_backtraces = true
```

That allows you to gather less information in exceptions causing exceptions to be much faster.

As to naming

backtrace_locations is ok, though I personally prefer Stacktrace and Stackframe , #stacktrace

---

#8 - 11/24/2012 11:19 AM - name (Yusuke Endoh)
- Status changed from Open to Assigned
- Target version changed from 2.0.0 to 2.6

#9 - 11/24/2012 12:10 PM - ko1 (Koichi Sasada)
- Status changed from Assigned to Closed

I added Thread#backtrace_locations.

I close this issue. Please re-open new tickets for separated one.

#10 - 11/26/2012 07:09 AM - sam.saffron (Sam Saffron)
Addeed #7436#7435 and #7434

#11 - 11/27/2012 09:23 AM - ko1 (Koichi Sasada)
(2012/09/23 15:43), sam.saffron (Sam Saffron) wrote:

> Also it feels very tied to MRI returning RubyVM::Backtrace::Location , Location seems to me in the wrong namespace. Is JRuby and Rubinius going to be expected to add this namespace? Is this going to be spec?

I want to discuss about this issue.
I agree that it is not CRuby specific feature
and RubyVM namespace is not good.

Background:

(0) Now, caller_locations returns array of ::RubyVM::Backtrace::Location objects.

(1) I don't care about class name of objects what caller_locations returns.

(2) ::Backtrace::Location seems good name.
But I'm not sure that ::Backtrace is common class or module name.
I'm afraid naming conflict.

Ideas:

(a) Don't care about conflict.
Rename to ::Backtrace::Location.

(b) Don't touch any more.
CRuby returns `::RubyVM::Backtrace::Location'.
But make it not spec.
Spec is "This object should respond to several methods which ::RubyVM::Backtrace::Location has".

(c) Other names.
Such as `::BacktraceLocation'.
Ah, this is good name because it is long
and maybe no conflict with existing code (I googled it), isn't it?
I think (c) seems good now.

---

03/19/2022
Thanks,
Koichi

--
// SASADA Koichi at atdot dot net

#12 - 11/29/2012 04:53 AM - ko1 (Koichi Sasada)
(2012/11/27 9:02), SASADA Koichi wrote:

(2012/09/23 15:43), sam.saffron (Sam Saffron) wrote:

   Also it feels very tied to MRI returning RubyVM::Backtrace::Location, Location seems to me in the wrong namespace. Is JRuby and Rubinius going to be expected to add this namespace? Is this going to be spec?

I want to discuss about this issue.
I agree that it is not CRuby specific feature
and RubyVM namespace is not good.

Background:

(0) Now, caller_locations returns array of
::RubyVM::Backtrace::Location objects.

(1) I don't care about class name of objects
what caller_locations returns.

(2) ::Backtrace::Location seems good name.
But I'm not sure that ::Backtrace is common class or module name.
I'm afraid naming conflict.

Ideas:

(a) Don't care about conflict.
   Rename to ::Backtrace::Location.

(b) Don't touch any more.
   CRuby returns ::RubyVM::Backtrace::Location'.
   But make it not spec.
   Spec is "This object should respond to several
   methods which ::RubyVM::Backtrace::Location has".

(c) Other names.
   Such as '::BacktraceLocation'.
   Ah, this is good name because it is long
   and maybe no conflict with existing code (I googled it), isn't it?

   I think (c) seems good now.

(d) ::Thread::Backtrace and ::Thread::Backtrace::Location

   I decided to use this name.
   ● Backtrace belong to each threads.
   ● More common than ::RubyVM
   ● I believe existing programs may not use ::Thread namespace

--
// SASADA Koichi at atdot dot net