Ruby master - Feature #7434
Allow caller_locations and backtrace_locations to receive negative params
11/26/2012 06:59 AM - saffron (Sam Saffron)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>ko1 (Koichi Sasada)</td>
</tr>
<tr>
<td>Target version:</td>
<td>2.6</td>
</tr>
</tbody>
</table>

**Description**
Further to [http://bugs.ruby-lang.org/issues/7051](http://bugs.ruby-lang.org/issues/7051)

It would be nice if caller_locations and backtrace_locations has some parity with range apis.

pry(main)> [1,2,3][0..-2]
=> [1, 2]

Similarly:

caller_locations(0,-2) # should strip the bottom frame

This is actually quite important for diagnostics of Rails and other frameworks where the app code starts at a very deep frame.

At the moment you are forced to materialize all frames just to discard.

**Associated revisions**

Revision 3d47e7b2 - 11/29/2012 12:12 AM - ko1 (Koichi Sasada)
- vm_backtrace.c (vm_backtrace_to_ary): support range argument like Array[]. [ruby-core:50092] [ruby-trunk - Feature #7434] Test and document is not available. Please help us.

[git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37957 b2dd03c8-39d4-4d8f-98ff-823fe69b080e]

Revision 37957 - 11/29/2012 12:12 AM - ko1 (Koichi Sasada)
- vm_backtrace.c (vm_backtrace_to_ary): support range argument like Array[]. [ruby-core:50092] [ruby-trunk - Feature #7434] Test and document is not available. Please help us.

Revision 37957 - 11/29/2012 12:12 AM - ko1 (Koichi Sasada)
- vm_backtrace.c (vm_backtrace_to_ary): support range argument like Array[]. [ruby-core:50092] [ruby-trunk - Feature #7434] Test and document is not available. Please help us.

Revision 37957 - 11/29/2012 12:12 AM - ko1 (Koichi Sasada)
- vm_backtrace.c (vm_backtrace_to_ary): support range argument like Array[]. [ruby-core:50092] [ruby-trunk - Feature #7434] Test and document is not available. Please help us.

Revision 37957 - 11/29/2012 12:12 AM - ko1 (Koichi Sasada)
- vm_backtrace.c (vm_backtrace_to_ary): support range argument like Array[]. [ruby-core:50092] [ruby-trunk - Feature #7434] Test and document is not available. Please help us.

Revision 37957 - 11/29/2012 12:12 AM - ko1 (Koichi Sasada)
- vm_backtrace.c (vm_backtrace_to_ary): support range argument like Array[]. [ruby-core:50092] [ruby-trunk - Feature #7434] Test and document is not available. Please help us.
document is not available. Please help us.

Revision e4be7f97 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38664 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

Revision 38664 - 12/31/2012 04:03 AM - zzak (Zachary Scott)

- test/ruby/test_backtrace.rb: Add test for r37957 [Feature #7434]

History

#1 - 11/26/2012 07:08 AM - sam.saffron (Sam Saffron)
ouch, this was meant to be pointed at 2.0 can't figure out how to change

#2 - 11/26/2012 07:49 AM - zzak (Zachary Scott)
- Category set to core
- Status changed from Open to Assigned
- Assignee set to ko1 (Koichi Sasada)
- Target version set to 2.0.0

#3 - 11/26/2012 07:54 AM - zzak (Zachary Scott)
- Target version changed from 2.0.0 to 2.6

#4 - 11/26/2012 08:59 AM - ko1 (Koichi Sasada)
(2012/11/26 6:59), sam.saffron (Sam Saffron) wrote:

caller_locations(0,-2) # should strip the bottom frame

2nd parameter specify 'how many frames'.
I feel 1st parameter is nice to accept negative index.
I don't have positive and negative one.
Could you show the concrete example and performance issue?

--
// SASADA Koichi at atdot dot net

#5 - 11/27/2012 07:35 AM - sam.saffron (Sam Saffron)
This is a "typical" caller trace for a rails app:

```
app/controllers/list_controller.rb:8:in block (2 levels) in <class:ListController>:
actionpack (3.2.9) lib/action_controller/metal/implicit_render.rb:4:in send_action
actionpack (3.2.9) lib/action_controller/metal/rendering.rb:10:in process_action
actionpack (3.2.9) lib/action_controller/metal/rescue.rb:29:in `process_action'
activerecord (3.2.9) lib/active_record/callbacks.rb:47:in `run_callbacks'
activerecord (3.2.9) lib/active_record/instrumentation.rb:29:in `in process_action'
```

This is a "typical" caller trace for a rails app:
This means that there are 60 or so frames below my application. If I just want to gather a trace for my application code, I really want:

caller_locations(0, -60) ... I work around by doing caller_locations[0..-60] however the VM is going to do tons of work to grab all these objects just for me to discard them.

#6 - 11/27/2012 07:37 AM - sam.saffron (Sam Saffron)

This means, I don't really know how many frames I want to take, I just know how many frames I want to discard.

#7 - 11/27/2012 08:53 AM - ko1 (Koichi Sasada)

This means that there are 60 or so frames below my application. If I just want to gather a trace for my application code, I really want:

caller_locations(0, -60) ... I work around by doing caller_locations[0..-60] however the VM is going to do tons of work to grab all these objects just for me to discard them.

I know that Rails (and framework) has big backtrace array.

My question is: use-case of negative number for performance. Your answer only shows "backtrace is big".

By the way, current implementation grabs all of backtrace information and cuts n'th elements fromlevel' (n' is 2nd argument andlevel' is 1st argument). This is implementation limitation. We need to seek all of backtrace to make correct backtrace. If you want to know "why it is needed?", please read source code. Short answer is "There are some frames which does not have location information".

--
// SASADA Koichi at atdot dot net

#8 - 11/27/2012 03:52 PM - sam.saffron (Sam Saffron)

My question is: use-case of negative number for performance. Your answer only shows "backtrace is big".

Apologies, well for MiniProfiler (https://github.com/SamSaffron/MiniProfiler/tree/master/Ruby) we gather stack traces quite aggressively, every time a query is execute a backtrace is materialized.

A classic place of where I would like to cut down on backtrace overhead is the sampling profiler see: https://github.com/SamSaffron/MiniProfiler/blob/master/Ruby/lib/mini_profile/profiler.rb#L289

Another use case would be a user option for :application_trace_only, that would figure out how many frames to skip in the first SQL interception and store that on the Thread so following calls to grab stack traces only include the "application" section and the framework stuff is not gathered.

The seek is still cheaper than materializing all the strings for the filenames and so on just to discard.

#9 - 11/27/2012 05:23 PM - ko1 (Koichi Sasada)

This means that there are 60 or so frames below my application. If I just want to gather a trace for my application code, I really want:

caller_locations(0, -60) ... I work around by doing caller_locations[0..-60] however the VM is going to do tons of work to grab all these objects just for me to discard them.
Issue #7434 has been updated by sam.saffron (Sam Saffron).

My question is: use-case of negative number for performance. Your answer only shows "backtrace is big".

Apologies, well for MiniProfiler (https://github.com/SamSaffron/MiniProfiler/tree/master/Ruby) we gather stack traces quite aggressively, every time a query is execute a backtrace is materialized.

A classic place of where I would like to cut down on backtrace overhead is the sampling profiler see: https://github.com/SamSaffron/MiniProfiler/blob/master/Ruby/lib/mini_profiler/profiler.rb#L289

I think you need to grab all of stack each time.
I'm not sure the negative number is good for this reason.

Another use case would be a user option for :application_trace_only, that would figure out how many frames to skip in the first SQL interception and store that on the Thread so following calls to grab stack traces only include the "application" section and the framework stuff is not gathered.

You need to grab all of stack trace to check which is skipped frame or not. Please correct my if my understanding is wrong.

I also am not sure why negative number is needed for this purpose.

Ah, I understand your purpose. I can't understand how to use negative number.

But I understand now.

[top of stack frame]
caller
foo4
foo3
foo2
foo1
f/w3 # f/w = framework
f/w2
f/w1
main
[bottom of stack frame]

In this case, framework consumes 3 frames and you only want to know stack frame only "all of frame size - framework frame size (=3)".

I missed the point that framework knows how many frames consumed (in this case: 3).

2nd argument is "how many frames from top".

So your suggestion.
(2nd argument) `n` means:
if n > 0
how many frames from top of frame
elsif n <= 0
how many frames should eliminate from bottom of frame
end

I feel it makes sense.

The last point is API design.
"negative number" to this purpose is feasible?
Any other example for other methods?

--
// SASADA Koichi at atdot dot net

#10 - 11/27/2012 06:35 PM - sam.saffron (Sam Saffron)

The last point is API design.
"negative number" to this purpose is feasible?
I like negative numbers cause it has some parity with the range APIs, eg: `[1,2,3][0..-2]` # skip last

Any other example for other methods?

I can not think of a use case for for the first param being negative, even though for arrays `[1,2,3][-2..-1]` makes sense, I can not think of a case where one would do `caller_locations(-10, -1)`, the proposal is all about making the second param more flexible.

#11 - 11/27/2012 06:59 PM - ko1 (Koichi Sasada)

(2012/11/27 18:35), sam.saffron (Sam Saffron) wrote:

The last point is API design.
"negative number" to this purpose is feasible?
I like negative numbers cause it has some parity with the range APIs, eg: `[1,2,3][0..-2]` # skip last

I agree that it is similar to the negative end of range. However, range points position (index), not a size.

I think similar, but different concept cause confusion.

Ideas:

(1) Accept your idea.
Define new concept.
(Do not refer Array#[])

(2) Introduce 3rd argument to specify eliminate
I feel it is bad.

(3) Consider argument

(3-1) Accept Range object on 1st parameter
caller(2..-2) #=> [foo2, ..., t/w2] on [ruby-core:50208] example.

(3-2) ... no idea

(3-1) seems good? It is same concept of Array#[]. Easy to explain.

Any other example for other methods?
I can not think of a use case for for the first param being negative, even though for arrays 1,2,3 makes sense, I can not think of a case where one would do `caller_locations(-10, -1)`, the proposal is all about making the second param more flexible.

I want to know other similar methods we can mimic.

--
// SASADA Koichi at atdot dot net

#12 - 11/28/2012 06:08 AM - sam.saffron (Sam Saffron)

I think either 1) or 3-1), works ok.

Though there is an issue with accepting ranges: caller(1..-6, 8) becomes a bit weird cause you are defining the length in two spots.

I can not find any other place with a similar api, anyone else?

#13 - 11/29/2012 04:53 AM - ko1 (Koichi Sasada)

(2012/11/28 6:08), sam.saffron (Sam Saffron) wrote:

Though there is an issue with accepting ranges: caller(1..-6, 8) becomes a bit weird cause you are defining the length in two spots.

It should be error.

See:

```
[[0..-1, 2]
 #=>
```
t.rb:1:in []': can't convert Range into Integer (TypeError)
from t.rb:1

--
// SASADA Koichi at atdot dot net

#14 - 11/29/2012 09:12 AM - ko1 (Koichi Sasada)
- Status changed from Assigned to Closed
- % Done changed from 0 to 100

This issue was solved with changeset r37957.
Sam, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- vm_backtrace.c (vm_backtrace_to_ary): support range argument like Array[]. [ruby-core:50092] [ruby-trunk - Feature #7434] Test and document is not available. Please help us.

#15 - 11/29/2012 09:23 AM - ko1 (Koichi Sasada)
(2012/11/28 6:08), sam.saffron (Sam Saffron) wrote:
I think either 1) or 3-1), works ok.
I committed (3-1).
Could you check it?
Also I'm very happy if you write tests for it.

--
// SASADA Koichi at atdot dot net

#16 - 11/30/2012 10:23 AM - sam.saffron (Sam Saffron)
ko1 (Koichi Sasada) will try to write some tests next week.

On first go I got a segfault:

(pry):1: [BUG] backtrace_collect: unreachable
ruby 2.0.0dev (2012-11-30) [i686-linux]

-- Control frame information -------------------------------

c:0024 p:---- s:0006 e:000095 CFUNC  caller_locations

c:0023 p:0009 s:0091 e:000090 EVAL   (pry):1 [FINISH]
c:0022 p:---- s:0089 e:000088 CFUNC  eval

c:0021 p:0089 s:0083 e:000062 METHOD /home/sam/.rvm/gems/ruby-head/gems/pry-0.9.10/lib/pry/pry_instance.rb:275

c:0020 p:0037 s:0076 e:000075 METHOD /home/sam/.rvm/gems/ruby-head/gems/pry-0.9.10/lib/pry/pry_instance.rb:251

c:0019 p:0012 s:0071 e:000070 BLOCK /home/sam/.rvm/gems/ruby-head/gems/pry-0.9.10/lib/pry/pry_instance.rb:231 [FINISH]
c:0018 p:---- s:0069 e:000068 CFUNC  loop

c:0016 p:---- s:0064 e:000063 CFUNC  catch

c:0014 p:---- s:0058 e:000057 CFUNC  catch

c:0013 p:0051 s:0054 e:000053 METHOD /home/sam/.rvm/gems/ruby-head/gems/pry-0.9.10/lib/pry_instance.rb:228

c:0012 p:0019 s:0048 e:000047 METHOD /home/sam/.rvm/gems/ruby-head/gems/pry-0.9.10/lib/pry_class.rb:154

c:0010 p:---- s:0035 e:000034 CFUNC  call

c:0008 p:---- s:0028 e:000027 CFUNC  each


c:0006 p:0030 s:0020 e:000019 TOP /home/sam/.rvm/gems/ruby-head/gems/pry-0.9.10/bin/pry.rb:16 [FINISH]
c:0005 p:---- s:0018 e:000017 CFUNC  load

c:0004 p:0099 s:0014 e:0001a8 EVAL   /home/sam/.rvm/gems/ruby-head/bin/pry:19 [FINISH]
c:0003 p:---- s:0011 e:000010 CFUNC  eval

c:0002 p:0077 s:0005 e:0019a8 EVAL   /home/sam/.rvm/gems/ruby-head/bin/ruby_noexec_wrapper:14 [FINISH]
c:0001 p:0000 s:0002 e:000574 TOP [FINISH]

/home/sam/.rvm/gems/ruby-head/bin/ruby_noexec_wrapper:14:in <main>
/home/sam/.rvm/gems/ruby-head/bin/ruby_noexec_wrapper:14:in eval
/home/sam/.rvm/gems/ruby-head/bin/pry:19:in <main>
/home/sam/.rvm/gems/ruby-head/bin/pry:19:in loaded

03/18/2022
#17 - 11/30/2012 01:53 PM - ko1 (Koichi Sasada)

(2012/11/30 10:25), sam.saffron (Sam Saffron) wrote:

ko1 (Koichi Sasada) will try to write some tests next week.

Thank you so much.

On first go I got a segfault:

Cool!
Reproduce code?

--
// SASADA Koichi at atdot dot net

#18 - 12/03/2012 11:00 AM - sam.saffron (Sam Saffron)

Getting it on a clean rvm install of ruby-head with
test.rb
caller_location(0,-1)

I hope my compile is correct, other stuff seems to work fine

#19 - 12/31/2012 01:04 PM - zzak (Zachary Scott)
- Status changed from Closed to Assigned

I have committed a test for range argument in r38664, please check it.

#20 - 01/31/2017 09:33 AM - ko1 (Koichi Sasada)
- Status changed from Assigned to Closed

Maybe ok.