Ruby - Feature #6895

TracePoint API

08/20/2012 04:10 PM - ko1 (Koichi Sasada)

Status:	Closed	
Priority:	Normal	
Assignee:	ko1 (Koichi Sasada)	
Target version:	2.0.0	

Description

=begin
= Abstract

Let's introduce TracePoint API that can replace set_trace_func().

= Background

See discussions on [Feature #6649] http://bugs.ruby-lang.org/issues/show/6649.

Problems with set_trace_func:

- Invoke trace funcs on all events (C level API can filter events)
- With this spec, we can't add new trace event (compatibility issue)
- Slow because it generates binding object each trace func

= Proposal

Introduce TracePoint API.

```
trace = TracePoint.trace(event1, event2, ...) do |tp|
# tp has methods like:
#   event: event name represented by Symbol like :line.
#          "c-call", "c-return" is :c_call, :c_return (sub('-', '_'))
#   file, line: return filename and line number
#   klass, id: return class or id (depends on a context. Same as set_trace_func)
#   binding: return (generated) binding object
#   self: new method.  same as tp.binding.eval('self')
... # trace func
# tp is TracePoint object.
# In fact, tp is same object as `trace'.
# We don't need any object creation on trace func.
end
... # Proc are called   1.600000   0.000000   1.600000 (  1.601750)
0.280000   0.000000   0.280000 (  0.287466)
0.750000   0.000000   0.750000 (  0.741344)
1.400000   0.020000   1.420000 (  1.420574)
trace.untrace # stop tracing
...
trace.retrace # restart tracing
```

`eventN' parameter for TracePoint.trace() is set of symbols. You can specify events what you want to trace. If you don't specify any events on it, then all events are activate (similar to set_trace_func).

= Implementation

See https://github.com/ko1/ruby/compare/tracepoint.

Try https://github.com/ko1/ruby/tracepoint.

= Evaluation

TracePoint API doesn't make temporary object on default. It is faster than current set_trace_func.

```
require 'benchmark'
```

```
MAX = 1_000_000
$c1 = $c2 = 0

Benchmark.bm{|x|
x.report{
set_trace_func(lambda{|args| $c1+=1})
MAX.times{
a = 1
}
set_trace_func(nil)
}
x.report{
trace = TracePoint.trace(%i{line call return c_call c_return}){|tp| $c2+=1}
MAX.times{
a = 1
}
trace.disable
}
x.report{
trace = TracePoint.trace(%i{line call return c_call c_return}){|tp| $c2+=1; tp.event; tp.self; tp.method_id}
MAX.times{
a = 1
}
trace.disable
}
x.report{
trace = TracePoint.trace(%i{line call return c_call c_return}){|tp| $c2+=1; tp.event; tp.self; tp.method_id; tp.binding}
MAX.times{
a = 1
}
trace.disable
}
}
END
#=>
user     system      total        real
1.140000   0.000000   1.140000 (  1.145847)
0.200000   0.000000   0.200000 (  0.194970)
0.380000   0.000000   0.380000 (  0.385857)
1.250000   0.000000   1.250000 (  1.251083)
```

= Problems

- TracePoint.trace(...) is good interface?
- Now, noway to specify Thread specific hooks (like Thread#set_trace_func)

= Next Step

I also want to introduce block enter/leave events on TracePoint.

=end

Associated revisions

Revision d28e07d57bc15a4baf08f8f4fb64d27cf927aac7 - 08/22/2012 05:12 AM - ko1 (Koichi Sasada)

- vm_trace.c: support TracePoint. [ruby-trunk - Feature #6895]
- test/ruby/test_settracefunc.rb: add tests for above.
- proc.c (rb_binding_new_with_cfp): add an internal function.
- vm.c (rb_vm_control_frame_id_and_class): add an internal function.
- vm_trace.c: add rb_add_event_hook2() and rb_thread_add_event_hook2().
  Give us the good name for them!

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@36773 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision d28e07d5 - 08/22/2012 05:12 AM - ko1 (Koichi Sasada)

- vm_trace.c: support TracePoint. [ruby-trunk - Feature #6895]
- test/ruby/test_settracefunc.rb: add tests for above.
- proc.c (rb_binding_new_with_cfp): add an internal function.
- vm.c (rb_vm_control_frame_id_and_class): add an internal function.
- vm_trace.c: add rb_add_event_hook2() and rb_thread_add_event_hook2().
  Give us the good name for them!

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@36773 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 89c889d4e72eb39bb70734d2360b9607c1099bdb - 11/29/2012 05:52 AM - ko1 (Koichi Sasada)

- vm_trace.c (rb_tracepoint_attr_defined_class):
  rename TracePoint#klass to TracePoint#defined_class.
  [ruby-core:50187] Re: [ruby-trunk - Feature #6895]
- include/ruby/debug.h: ditto.
- test/ruby/test_settracefunc.rb: ditto.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37972 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 89c889d4 - 11/29/2012 05:52 AM - ko1 (Koichi Sasada)

- vm_trace.c (rb_tracepoint_attr_defined_class):
  rename TracePoint#klass to TracePoint#defined_class.
  [ruby-core:50187] Re: [ruby-trunk - Feature #6895]
- include/ruby/debug.h: ditto.
- test/ruby/test_settracefunc.rb: ditto.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@37972 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision f2fee8446648c12b80f2d7d7a16fe6d527971838 - 11/30/2012 09:25 AM - zzak (zzak _)

- vm_trace.c: Documentation for TracePoint API
  [ruby-core:47243] [Feature #6895]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38045 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision f2fee844 - 11/30/2012 09:25 AM - zzak (zzak _)

- vm_trace.c: Documentation for TracePoint API
  [ruby-core:47243] [Feature #6895]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38045 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 08/21/2012 06:11 AM - trans (Thomas Sawyer)

=begin
Looks great. Really nice that there is performance gain too.

Can we use TracePoint.new to get instance that is not automatically active? eg.

tracer = TracePoint.new{ |tp| ... }
tracer.trace

Is same as:

tc = TracePoint.trace{ |tp| ... }

Also, I assume that if no events are given as arguments, it includes all events?

=end

#2 - 08/21/2012 11:23 AM - ko1 (Koichi Sasada)

(2012/08/21 6:11), trans (Thomas Sawyer) wrote:

> Can we use TracePoint.new to get instance that is not automatically active? eg.
>
> tracer = TracePoint.new{ |tp| ... }
> tracer.trace

> Is same as:
>
> tc = TracePoint.trace{ |tp| ... }

I understand your proposal. I use "Thread.new (Thread.start)" analogy (and current set_trace_func analogy). I think two different behavior is not good. Which one do you like?

> Also, I assume that if no events are given as arguments, it includes all events?
> Yes.
>
> > `eventN' parameter for TracePoint.trace() is set of symbols. You can specify events what you want to trace. If you don't specify any events on it, then all events are activate (similar to set_trace_func).

But there is an issue. Now "all" means same events that set_trace_func supports. But if I add other events like "block invoke", then what mean the "all"?

--
// SASADA Koichi at atdot dot net

#3 - 08/21/2012 01:29 PM - Anonymous

Looks good so far. What I'd ask though is that for return events one be able to get the return value and for exception events one be able to get the exception message.

Thanks.

On Mon, Aug 20, 2012 at 10:03 PM, SASADA Koichi ko1@atdot.net wrote:

> (2012/08/21 6:11), trans (Thomas Sawyer) wrote:
>
> > Can we use TracePoint.new to get instance that is not automatically active? eg.
> >
> > tracer = TracePoint.new{ |tp| ... }
> > tracer.trace
> >
> > Is same as:
> >
> > tc = TracePoint.trace{ |tp| ... }
>
> I understand your proposal. I use "Thread.new (Thread.start)" analogy (and current set_trace_func analogy). I think two different behavior is not good. Which one do you like?
>
> > Also, I assume that if no events are given as arguments, it includes all events?
> > Yes.
> >
> > > `eventN' parameter for TracePoint.trace() is set of symbols. You can specify events what you want to trace. If you don't specify any events on it, then all events are activate (similar to set_trace_func).
>
> But there is an issue. Now "all" means same events that set_trace_func supports. But if I add other events like "block invoke", then what mean the "all"?
>
> --
> // SASADA Koichi at atdot dot net

#4 - 08/21/2012 01:59 PM - ko1 (Koichi Sasada)

(2012/08/21 13:25), Rocky Bernstein wrote:

> Looks good so far.

Thanks!

> What I'd ask though is that for return events one be
> able to get the return value and for exception events one be able to get
> the exception message.

Okay. I'll try it.

--
// SASADA Koichi at atdot dot net

#5 - 08/21/2012 03:23 PM - ko1 (Koichi Sasada)

(2012/08/20 16:10), ko1 (Koichi Sasada) wrote:

> Issue #6895 has been reported by ko1 (Koichi Sasada).
>
> ---
>
> Feature #6895: TracePoint API
> https://bugs.ruby-lang.org/issues/6895
>
> =begin
> = Abstract
>
> Let's introduce TracePoint API that can be replaced with set_trace_func().

I asked matz ana he said "commit it and try".

Please check it.

--
// SASADA Koichi at atdot dot net

#6 - 08/22/2012 02:12 PM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r36773.
Koichi, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- vm_trace.c: support TracePoint. [ruby-trunk - Feature #6895]
- test/ruby/test_settracefunc.rb: add tests for above.
- proc.c (rb_binding_new_with_cfp): add an internal function.
- vm.c (rb_vm_control_frame_id_and_class): add an internal function.
- vm_trace.c: add rb_add_event_hook2() and rb_thread_add_event_hook2().
  Give us the good name for them!

#7 - 08/22/2012 02:53 PM - ko1 (Koichi Sasada)

(2012/08/21 6:11), trans (Thomas Sawyer) wrote:

> Can we use TracePoint.new to get instance that is not automatically active? eg.
>
> tracer = TracePoint.new{ |tp| ... }
> tracer.trace

Another idea:

tracer = TracePoint.new(events...){...}
set_trace_func(tracer) # activate

--
// SASADA Koichi at atdot dot net

#8 - 08/23/2012 12:59 AM - Anonymous

Two things. OO methods (in contrast to global methods from Kernel) are generally the way Ruby does things, right? Second, to reduce confusion and keep compatibility set_trace_func() should be retired.

That said, these are small points in the larger issue of having something that makes writing debuggers, profilers and tracers easier and provide more usefulness. So it is those aspects, personally, I care more about than this.

On Wed, Aug 22, 2012 at 1:35 AM, SASADA Koichi ko1@atdot.net wrote:

> (2012/08/21 6:11), trans (Thomas Sawyer) wrote:
>
> > Can we use TracePoint.new to get instance that is not automatically active? eg.
> >
> > tracer = TracePoint.new{ |tp| ... }
> > tracer.trace
>
> Another idea:
>
> tracer = TracePoint.new(events...){...}
> set_trace_func(tracer) # activate
>
> --
> // SASADA Koichi at atdot dot net

#9 - 08/23/2012 01:22 AM - trans (Thomas Sawyer)

> I understand your proposal. I use "Thread.new (Thread.start)" analogy (and current set_trace_func analogy). I think two different behavior is not good. Which one do you like?

Hmm... Well if we can only have one it would have to be the first b/c it gives most flexibility. But I don't understand why we can't have both when the second (TracePoint.trace) would just be a convenience shortcut to the first, i.e.

```
def TracePoint.trace(*events, &block)
TracePoint.new(*events, &block).trace
end
```

> > Also, I assume that if no events are given as arguments, it includes all events?
> > Yes.
>
> `eventN' parameter for TracePoint.trace() is set of symbols. You can specify events what you want to trace. If you don't specify any events on it, then all events are activate (similar to set_trace_func).

> But there is an issue. Now "all" means same events that set_trace_func supports. But if I add other events like "block invoke", then what mean the "all"?

"All" should mean all. If you add other events then those should be included too. Why would you think not to include your new events?

#10 - 08/23/2012 01:23 AM - trans (Thomas Sawyer)

```
=begin
Techinically that should be:

def TracePoint.trace(*events, &block)
t = TracePoint.new(*events, &block)
t.trace
t
end
=end
```

#11 - 08/23/2012 05:23 AM - drbrain (Eric Hodel)

On Aug 20, 2012, at 00:10, ko1 (Koichi Sasada) redmine@ruby-lang.org wrote:

> trace.untrace # stop tracing
> ...
> trace.retrace # restart tracing

I think start/stop or on/off (like tracer.rb) are preferable to having "trace" twice on the same line.

#12 - 08/23/2012 12:23 PM - ko1 (Koichi Sasada)

(2012/08/23 1:22), trans (Thomas Sawyer) wrote:

> > I understand your proposal. I use "Thread.new (Thread.start)" analogy
> >
> > > (and current set_trace_func analogy). I think two different behavior is
> > > not good. Which one do you like?
> > > Hmm... Well if we can only have one it would have to be the first b/c it gives most flexibility. But I don't understand why we can't have
> > > both when the second (TracePoint.trace) would just be a convenience shortcut to the first, i.e.
>
> def TracePoint.trace(*events, &block)
> TracePoint.new(*events, &block).trace
> end

In some cases, flexibility doesn't improve usability.

I understand that TracePoint.new API improves flexibility. But it is
more difficult to retrieve it. For example, users need to consider when
the trace activate.

And I doubt that the flexibility helps users. Any use-case?

> > > Also, I assume that if no events are given as arguments, it includes all events?
> > > Yes.
> >
> > > `eventN' parameter for TracePoint.trace() is set of symbols. You can specify events what you want to trace. If you don't
> > > specify any events on it, then all events are activate (similar to set_trace_func).
>
> > But there is an issue. Now "all" means same events that set_trace_func
> > supports. But if I add other events like "block invoke", then what mean
> > the "all"?
> > "All" should mean all. If you add other events then those should be included too. Why would you think not to include your new events?

Compatibility. But it can be avoid if we add about it in a document.

--
// SASADA Koichi at atdot dot net

#13 - 08/23/2012 12:29 PM - ko1 (Koichi Sasada)

(2012/08/23 5:04), Eric Hodel wrote:

> On Aug 20, 2012, at 00:10, ko1 (Koichi Sasada) redmine@ruby-lang.org wrote:
>
> > trace.untrace # stop tracing
> > ...
> > trace.retrace # restart tracing
>
> I think start/stop or on/off (like tracer.rb) are preferable to having "trace" twice on the same line.

I use "untrace/retrace" because tracer is active after
TracePoint.trace{}. I want to emphasize trace *again* by "untrace".

I agree "on/off" naming if we accept TracePoint.new API. like:

trace = TracePoint.new(...){...} # not activated
...
trace.on
...
trace.off

or

trace.activate{
...
}

--
// SASADA Koichi at atdot dot net

#14 - 08/23/2012 12:53 PM - ko1 (Koichi Sasada)

(2012/08/23 12:26), SASADA Koichi wrote:

> I use "untrace/retrace" because tracer is active after
> TracePoint.trace{}. I want to emphasize trace *again* by "untrace".

Oops. I use "retrace" to emphasize *again*.

I also think we need reconsidering "TracePoint" name.

Now, TracePoint object have two functionality.

(1) Trace control

"start and stop" tracing. #trace, #untrace.

(2) Trace status snapshot

Get current status. #event, #line, #file, #biding

You can see this mixture by the following code.

tracer = TracePoint.trace(){
p tracer.binding # (2)
}
...
tracer.untrace # (1)

BTW tracer.binding out from tracer block causes an exception.

Maybe the name "TracePoint" by trans is for (2).

--
// SASADA Koichi at atdot dot net

#15 - 08/25/2012 04:53 AM - trans (Thomas Sawyer)

On Wed, Aug 22, 2012 at 11:18 PM, SASADA Koichi ko1@atdot.net wrote:

> I understand that TracePoint.new API improves flexibility. But it is
> more difficult to retrieve it. For example, users need to consider when
> the trace activate.
>
> And I doubt that the flexibility helps users. Any use-case?

The advantage comes if you want to setup a tracepoint prior to
activating it, say when something triggers a callback for example. If
we can't pre-build the tracepoint, then we will have to store the
events and procedure separately before applying it and then cache the
result -- it just makes it a little less straight-forward to
implement. A super simplistic example:

class MyPoint
def initialize(*events, &block)

@trace = TracePoint(*events, &block)
end
def trigger!
@trace.on unless @trace.on?
end
end

vs.

class MyPoint
def initialize(*events, &block)*
@events = events
@block = block
end
def trigger!
@trace ||=TracePoint.trace(@events, &@block)
@trace.on unless @trace.on?
end
end

And what about creating a TracePoint and passing it as an argument
(e.g. IOC pattern)? Without new we have to create it and turn it off
real quick first?

#16 - 11/01/2012 12:48 PM - ko1 (Koichi Sasada)

- Status changed from Closed to Assigned

- Assignee set to ko1 (Koichi Sasada)

- Target version set to 2.0.0

I need to consider about:

- TracePoint enable/disable API [ruby-core:47305]
- Add interface to get returned/raised value [ruby-core:47259]
- trace block enter/leave interface

Sorry for my late response.
This ticket should be open.

#17 - 11/20/2012 08:12 PM - ko1 (Koichi Sasada)

Finally, I changed TracePoint API to

- TracePoint.new
- TracePoint.trace
- TracePoint#enable
- TracePoint#disable
- TracePoint#enabled?

I believe you can imagine what happen. Please review it.

Remaining task is writing documents. Anyone can help us?

#18 - 11/20/2012 08:23 PM - ko1 (Koichi Sasada)

(2012/08/21 13:53), SASADA Koichi wrote:

> What I'd ask though is that for return events one be
> able to get the return value and for exception events one be able to get
> the exception message.
> Okay. I'll try it.

Sorry for my laziness, I made two methods:

TracePoint#return_value
TracePoint#raised_exception

at r37752.

Could you try and review it?

And any other requests for TracePoint?

--
// SASADA Koichi at atdot dot net

#19 - 11/24/2012 11:23 AM - mame (Yusuke Endoh)

Ko1, may I close this ticket?

--
Yusuke Endoh mame@tsg.ne.jp

#20 - 11/24/2012 12:16 PM - ko1 (Koichi Sasada)

- Status changed from Assigned to Closed

Sure.

I close it.

#21 - 11/24/2012 12:17 PM - zzak (zzak _)

I want to help with documentation, but I will open a separate ticket for review.

#22 - 11/26/2012 04:27 AM - trans (Thomas Sawyer)

I made a comparison of the API with the pure Ruby TracePoint gem I had written and have a few points:

- Do all traces have a defined event type now? I see that I had #event? and #eventless? methods in my API b/c sometimes event is nil, I think.
- Is there a way to get the callee, i.e. the current method name (if any)?
- I had defined #=== so one could use case statements matching against event types. Might be handy.
- Is #klass method necessary since one can call self.class? But maybe self.class is much less efficient? Also, "klass" is an ugly name. Would be nice if there were a better alias, but I can't think of a good name.
- Lastly, is there a way to get prior binding(s)? It's been a long time, but when I first worked on this I recall a need to get access to the previous binding. Unfortunately I can't recall why now, but I remember it being a significant enough problem that I made sure to add support for it.

#23 - 11/26/2012 05:53 AM - ko1 (Koichi Sasada)

Hi,

(2012/11/26 4:27), trans (Thomas Sawyer) wrote:

> I made a comparison of the API with the pure Ruby TracePoint gem I had
> written and have a few points:

I don't copy TracePoint gem.

> - Do all traces have a defined event type now? I see that I had #event? and #eventless? methods in my API b/c sometimes event is nil, I think.

I can't get points.
What is "defined event"?

> - Is there a way to get the callee, i.e. the current method name (if any)?

caller or caller_locations is not enough?

> - I had defined #=== so one could use case statements matching against event types. Might be handy.

```
case tp.event
when :call, :c_call
...
end
```

is not enough?

- Is #klass method necessary since one can call self.class? But maybe self.class is much less efficient? Also, "klass" is an ugly name. Would be nice if there were a better alias, but I can't think of a good name.

I agree with

- it is ugly
- but no good name

.

- Lastly, is there a way to get prior binding(s)? It's been a long time, but when I first worked on this I recall a need to get access to the previous binding. Unfortunately I can't recall why now, but I remember it being a significant enough problem that I made sure to add support for it.

No. It is too powerful. It should be discuss at another place.

--
// SASADA Koichi at atdot dot net

#24 - 11/26/2012 06:43 AM - trans (Thomas Sawyer)

> I can't get points.
> What is "defined event"?

Undefined event, where tp.event #=> nil. I would assume there is always an event type, but just checking this is so.

> caller or caller_locations is not enough?

Wouldn't caller_locations return the location in the trace block itself? And caller has to be parsed. It would be nice to have method to get calling method name. Should I write new issue?

> No. It is too powerful. It should be discuss at another place.

Ok.

> I don't copy TracePoint gem.

Considering your wrote C API and gem is pure Ruby, that would be hard to do. But API is necessarily close to same by definition. But you sound defensive. I never said you copied. Is it really too much to ask to recognize that I had done previous work in this regard?

#25 - 11/27/2012 08:53 AM - ko1 (Koichi Sasada)

I renamed methods:

- TracePoint#file -> TracePoint#path
- TracePoint#line -> TracePoint#lineno
  to make consistent to RubyVM::Backtrace::Location.

Make sense?

--
// SASADA Koichi at atdot dot net

#26 - 11/27/2012 08:53 AM - ko1 (Koichi Sasada)

(2012/11/26 5:47), SASADA Koichi wrote:

- Is #klass method necessary since one can call self.class? But maybe self.class is much less efficient?

klass' and self.class' is different.

`klass' is several meaning:

- on call/return event: method defined class.

class C0
def m

```
end
end

class C1 < C0
end

TracePoint.trace(:call){|tp| p [tp.klass, tp.self.class]}
C1.new.m #=> [C0, C1]
```

- on class/end event: nil (I wonder this behavior!!)
- on line, raise event: class of current method.
  ...

The best solution seems prepare methods for each event.

such as:
defined_class # it is for call, return event.
current_class # it is for line, raise event.

...

I think only `defined_class' is enough.

Any comments?

--
// SASADA Koichi at atdot dot net

#27 - 11/30/2012 01:22 AM - ko1 (Koichi Sasada)

- Status changed from Closed to Assigned

I didn't receive your comment from ruby-core.

trans (Thomas Sawyer) wrote:

> > I can't get points.
> > What is "defined event"?
>
> Undefined event, where tp.event #=> nil. I would assume there is always an event type, but just checking this is so.

tp.event has always symbol or raise an exception.

> > caller or caller_locations is not enough?
>
> Wouldn't caller_locations return the location in the trace block itself? And caller has to be parsed. It would be nice to have method to get calling method name. Should I write new issue?

I understand your point.

Which method name do you like?
tp#caller_location?

> > I don't copy TracePoint gem.
>
> Considering your wrote C API and gem is pure Ruby, that would be hard to do. But API is necessarily close to same by definition. But you sound defensive. I never said you copied. Is it really too much to ask to recognize that I had done previous work in this regard?

Sorry if you feel bad. I only want to say new TracePoint is independent from your gem and should not depend on it.
Comments are welcome, of course.

#28 - 11/30/2012 04:33 PM - zzak (zzak _)

- Description updated

#29 - 11/30/2012 06:25 PM - zzak (zzak _)

- Status changed from Assigned to Closed

This issue was solved with changeset r38045.
Koichi, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- vm_trace.c: Documentation for TracePoint API
  [ruby-core:47243] [Feature #6895]