

Ruby trunk - Feature #15289

Accept "target" keyword on `TracePoint#enable`

11/07/2018 08:51 AM - ko1 (Koichi Sasada)

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

Description

Abstract

To enable TracePoint for specific location, I propose new keyword argument target: to TracePoint#enable for specific events (line, call, return, class, end).

Usage:

```
line_trace = TracePoint.new(:line){|tp|
  p tp
}

def foo
  p 1
end

line_trace.enable(target: method(:foo)) do
  foo
end
```

In this case, a line trace is enable only on the method foo.

target: keyword accepts Method, UnboundMethod, Proc and RubyVM::InstructionSequence (ISeq for short) . All of objects can pull ISeq objects.

Further more, I propose target_lines: keyword too, which specify effective line(s).

These features can improve "break point" feature and so on.

Background

If we want to insert break point in Ruby's source code, we can use TracePoint (line event, for example). However, it enables all locations. It hurt performance.

This proposal overcomes this performance penalty.

Implementation

Now basic design are completed in my mind :p

Discussion

line or lines

For breakpoint purpose, we only need to specify a line. Specifying lines is needed?

no events on lines

It is not clear that if line is specified (for example: 10) and there are no line event (for example, empty line).

Possible options:

- (1) raise an exception
- (2) adjust before/after effective event
- (3) ignore

I prefer (1).

no events in Proc

Similar to last topic, if we specify call event on Proc object, there are no call event. What happens?

recursive or not

If method foo refers other blocks, I think we need to enable recursively.

how to get File target?

Sometimes we want to specify breakpoint to the location specified by "file:line". How to get the lines?

https://github.com/ko1/iseq_collector provides the feature to collect all of ISeqs. debugger can collect all of them and debugger can filter with path name.

Also [Feature #15287] will help to hook not loaded locations.

enable w/ keywords or other method name?

I have no strong opinion, but TracePoint#enable_on(target, lines: ...) is another idea?

reference

Related: <https://github.com/ruby/ruby/pull/2000>

This implementation is based on trace_insn.

Associated revisions

Revision 96990203 - 11/26/2018 06:16 PM - ko1 (Koichi Sasada)

Support targetting TracePoint [Feature #15289]

- vm_trace.c (rb_tracepoint_enable_for_target): support targetting TracePoint. [Feature #15289]

Tragetting TracePoint is only enabled on specified method, proc and so on, example: tp.enable(target: code).

code should be consisted of InstructionSeuqnece (iseq) (RubyVM::InstructionSeuqnece.of(code) should not return nil)
If code is a tree of iseq, TracePoint is enabled on all of iseqs in a tree.

Enabled tragetting TracePoints can not enabled again with and without target.

- vm_core.h (rb_iseq_t): introduce rb_iseq_t::local_hooks to store local hooks.
rb_iseq_t::aux::trace_events is renamed to global_trace_events to contrast with local_hooks.
- vm_core.h (rb_hook_list_t): add rb_hook_list_t::running to represent how many Threads/Fibers are used this list.
If this field is 0, nobody using this hooks and we can delete it.

This is why we can remove code from cont.c.

- vm_core.h (rb_vm_t): because of above change, we can eliminate rb_vm_t::trace_running field.
Also renamed from rb_vm_t::event_hooks to global_hooks.

- `vm_core.h`, `vm.c` (`ruby_vm_event_enabled_global_flags`): renamed from ``ruby_vm_event_enabled_flags`.
- `vm_core.h`, `vm.c` (`ruby_vm_event_local_num`): added to count enabled targetting `TracePoints`.
- `vm_core.h`, `vm_trace.c` (`rb_exec_event_hooks`): accepts hook list.
- `vm_core.h` (`rb_vm_global_hooks`): added for convinience.
- `method.h` (`rb_method_bmethod_t`): added to maintain `Proc` and `rb_hook_list_t` for `bmethod` (defined by `define_method`).
- `prelude.rb` (`TracePoint#enable`): extract a keyword parameter (because it is easy than writing in C). It calls `TracePoint#__enable` internal method written in C.
- `vm_inshelper.c` (`vm_trace`): check also `iseq->local_hooks`.
- `vm.c` (`invoke_bmethod`): check `def->body.bmethod.hooks`.
- `vm.c` (`hook_before_rewind`): check `iseq->local_hooks` and `def->body.bmethod.hooks` before rewind by exception.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@66003 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 66003 - 11/26/2018 06:16 PM - ko1 (Koichi Sasada)

Support targetting `TracePoint` [Feature #15289]

- `vm_trace.c` (`rb_tracepoint_enable_for_target`): support targetting `TracePoint`. [Feature #15289]

Tragetting `TracePoint` is only enabled on specified method, `proc` and so on, example: `tp.enable(target: code)`.

code should be consisted of `InstructionSeuqnece` (`iseq`) (`RubyVM::InstructionSeuqnece.of(code)` should not return nil)
If code is a tree of `iseq`, `TracePoint` is enabled on all of `iseqs` in a tree.

Enabled tragetting `TracePoints` can not enabled again with and without target.

- `vm_core.h` (`rb_iseq_t`): introduce `rb_iseq_t::local_hooks` to store local hooks.
`rb_iseq_t::aux::trace_events` is renamed to `global_trace_events` to contrast with `local_hooks`.
- `vm_core.h` (`rb_hook_list_t`): add `rb_hook_list_t::running` to represent how many `Threads/Fibers` are used this list.
If this field is 0, nobody using this hooks and we can delete it.

This is why we can remove code from `cont.c`.

- `vm_core.h` (`rb_vm_t`): because of above change, we can eliminate `rb_vm_t::trace_running` field.
Also renamed from `rb_vm_t::event_hooks` to `global_hooks`.
- `vm_core.h`, `vm.c` (`ruby_vm_event_enabled_global_flags`): renamed from ``ruby_vm_event_enabled_flags`.
- `vm_core.h`, `vm.c` (`ruby_vm_event_local_num`): added to count enabled targetting `TracePoints`.
- `vm_core.h`, `vm_trace.c` (`rb_exec_event_hooks`): accepts hook list.
- `vm_core.h` (`rb_vm_global_hooks`): added for convinience.
- `method.h` (`rb_method_bmethod_t`): added to maintain `Proc` and `rb_hook_list_t` for `bmethod` (defined by `define_method`).

- prelude.rb (TracePoint#enable): extract a keyword parameter (because it is easy than writing in C).
It calls TracePoint#__enable internal method written in C.
- vm_inshelper.c (vm_trace): check also iseq->local_hooks.
- vm.c (invoke_bmethod): check def->body.bmethod.hooks.
- vm.c (hook_before_rewind): check iseq->local_hooks and def->body.bmethod.hooks before rewind by exception.

Revision 66003 - 11/26/2018 06:16 PM - ko1 (Koichi Sasada)

Support targetting TracePoint [Feature #15289]

- vm_trace.c (rb_tracepoint_enable_for_target): support targetting TracePoint. [Feature #15289]

Tragetting TracePoint is only enabled on specified method, proc and so on, example: tp.enable(target: code).

code should be consisted of InstructionSeuqnece (iseq) (RubyVM::InstructionSeuqnece.of(code) should not return nil)
If code is a tree of iseq, TracePoint is enabled on all of iseqs in a tree.

Enabled tragetting TracePoints can not enabled again with and without target.

- vm_core.h (rb_iseq_t): introduce rb_iseq_t::local_hooks to store local hooks.
rb_iseq_t::aux::trace_events is renamed to global_trace_events to contrast with local_hooks.
- vm_core.h (rb_hook_list_t): add rb_hook_list_t::running to represent how many Threads/Fibers are used this list.
If this field is 0, nobody using this hooks and we can delete it.

This is why we can remove code from cont.c.

- vm_core.h (rb_vm_t): because of above change, we can eliminate rb_vm_t::trace_running field.
Also renamed from rb_vm_t::event_hooks to global_hooks.
- vm_core.h, vm.c (ruby_vm_event_enabled_global_flags): renamed from `ruby_vm_event_enabled_flags`.
- vm_core.h, vm.c (ruby_vm_event_local_num): added to count enabled targetting TracePoints.
- vm_core.h, vm_trace.c (rb_exec_event_hooks): accepts hook list.
- vm_core.h (rb_vm_global_hooks): added for convinience.
- method.h (rb_method_bmethod_t): added to maintain Proc and rb_hook_list_t for bmethod (defined by define_method).
- prelude.rb (TracePoint#enable): extract a keyword parameter (because it is easy than writing in C).
It calls TracePoint#__enable internal method written in C.
- vm_inshelper.c (vm_trace): check also iseq->local_hooks.
- vm.c (invoke_bmethod): check def->body.bmethod.hooks.
- vm.c (hook_before_rewind): check iseq->local_hooks and def->body.bmethod.hooks before rewind by exception.

Revision 72e60a04 - 11/26/2018 08:16 PM - ko1 (Koichi Sasada)

TracePoint#enable(target_line:) is supported. [Feature #15289]

- vm_trace.c: TracePoint#enable(target_line:) is supported. This option enables a hook only at specified target_line. target_line should be combination with target and :line event.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@66008 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 66008 - 11/26/2018 08:16 PM - ko1 (Koichi Sasada)

TracePoint#enable(target_line:) is supported. [Feature #15289]

- vm_trace.c: TracePoint#enable(target_line:) is supported. This option enables a hook only at specified target_line. target_line should be combination with target and :line event.

Revision 66008 - 11/26/2018 08:16 PM - ko1 (Koichi Sasada)

TracePoint#enable(target_line:) is supported. [Feature #15289]

- vm_trace.c: TracePoint#enable(target_line:) is supported. This option enables a hook only at specified target_line. target_line should be combination with target and :line event.

Revision d46cd60f - 11/29/2018 08:39 PM - tenderlove

Use a shared array for the duparray instruction

In this example code:

```
def foo
  [1, 2, 3, 4]
end
```

The array literal uses a duparray instruction. Before this patch, rb_ary_resurrect would malloc and memcpy a new array buffer. This patch changes rb_ary_resurrect to use ary_make_partial so that the new array object shares the underlying buffer with the array stored in the instruction sequences.

Before this patch, the new array object is not shared:

```
$ ruby -r objspace -e'p ObjectSpace.dump([1, 2, 3, 4])'
{"address\":"0x7fa2718372d0", "type\":"ARRAY", "class\":"0x7fa26f8b0010", "length\:4, "memsize\:72, "flags\":{"wb_protected\:true}}\n"
```

After this patch:

```
$ ./ruby -r objspace -e'p ObjectSpace.dump([1, 2, 3, 4])'
{"address\":"0x7f9a76883638", "type\":"ARRAY", "class\":"0x7f9a758af900", "length\:4, "shared\:true, "references\":[\n0x7f9a768837c8"], "memsize\:40, "flags\":{"wb_protected\:true}}\n"
```

[Feature #15289] [ruby-core:90097]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@66095 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 66095 - 11/29/2018 08:39 PM - tenderlovmaking (Aaron Patterson)

Use a shared array for the duparray instruction

In this example code:

```
def foo
  [1, 2, 3, 4]
end
```

The array literal uses a duparray instruction. Before this patch, rb_ary_resurrect would malloc and memcpy a new array buffer. This patch changes rb_ary_resurrect to use ary_make_partial so that the new array object shares the underlying buffer with the array stored in the instruction sequences.

Before this patch, the new array object is not shared:

```
$ ruby -r objspace -e'p ObjectSpace.dump([1, 2, 3, 4])'
{"address\":"0x7fa2718372d0", "type\":"ARRAY", "class\":"0x7fa26f8b0010", "length\:4, "memsize\:72, "flags\":{"wb_protected\:true}}\n"
```

After this patch:

```
$ ./ruby -r objspace -e'p ObjectSpace.dump([1, 2, 3, 4])'
{"address":"0x7f9a76883638", "type":"ARRAY", "class":"0x7f9a758af900", "length":4, "shared":true, "references":["0x7f9a768837c8"], "memsize":40, "flags":{"wb_protected":true}}\n"
```

[Feature #15289] [ruby-core:90097]

Revision 66095 - 11/29/2018 08:39 PM - tenderlove

Use a shared array for the duparray instruction

In this example code:

```
def foo
  [1, 2, 3, 4]
end
```

The array literal uses a duparray instruction. Before this patch, `rb_ary_resurrect` would `malloc` and `memcpy` a new array buffer. This patch changes `rb_ary_resurrect` to use `ary_make_partial` so that the new array object shares the underlying buffer with the array stored in the instruction sequences.

Before this patch, the new array object is not shared:

```
$ ruby -r objspace -e'p ObjectSpace.dump([1, 2, 3, 4])'
{"address":"0x7fa2718372d0", "type":"ARRAY", "class":"0x7fa26f8b0010", "length":4, "memsize":72, "flags":{"wb_protected":true}}\n"
```

After this patch:

```
$ ./ruby -r objspace -e'p ObjectSpace.dump([1, 2, 3, 4])'
{"address":"0x7f9a76883638", "type":"ARRAY", "class":"0x7f9a758af900", "length":4, "shared":true, "references":["0x7f9a768837c8"], "memsize":40, "flags":{"wb_protected":true}}\n"
```

[Feature #15289] [ruby-core:90097]

History

#1 - 11/08/2018 02:24 PM - shevegen (Robert A. Heiler)

Sounds nice. Please do not forget documentation when/if approved/added. :)

I have no particular opinion on the API/name(s) as such, but I would recommend to try to keep it simple and "logical" if possible. A Hash would probably make the most sense, and the names for the parameters should ideally be the same, e. g. if "lines:" is used, to also keep the same names whenever possible (both for TracePoint, but also perhaps elsewhere in ruby if that is necessary; a bit like how recently `:exception` was added to `Kernel#system` and other methods there, like in the news entry for the upcoming 2.6.x xmas-release.

That is not necessarily an endorsement of the name "lines:" as such, mind you; just more a general comment.

(I think the name "lines:" may not always be ideal, e. g. in the cases where a ruby user may only want to add one breakpoint to a single line - but this is mostly a detail, IMO. I am sure any fitting name can be derived when we have some examples of how this is used. Ideally a short name if possible.)

#2 - 11/22/2018 08:04 AM - ko1 (Koichi Sasada)

I got Matz's approval.
I'll implement it and commit soon before ruby 2.6 rc1.

Now, I finished most of features, except handling at exception.

#3 - 11/26/2018 06:16 PM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

Applied in changeset [trunk|r66003](#).

Support targeting TracePoint [Feature [#15289](#)]

- `vm_trace.c (rb_tracepoint_enable_for_target)`: support targetting TracePoint. [Feature [#15289](#)]

Tragetting TracePoint is only enabled on specified method, proc and so on, example: `tp.enable(target: code)`.

code should be consisted of InstructionSeuqnece (iseq) (RubyVM::InstructionSeuqnece.of(code) should not return nil)
If code is a tree of iseq, TracePoint is enabled on all of iseqs in a tree.

Enabled tragetting TracePoints can not enabled again with and without target.

- `vm_core.h (rb_iseq_t)`: introduce `rb_iseq_t::local_hooks` to store local hooks.
`rb_iseq_t::aux::trace_events` is renamed to `global_trace_events` to contrast with `local_hooks`.
- `vm_core.h (rb_hook_list_t)`: add `rb_hook_list_t::running` to represent how many Threads/Fibers are used this list.
If this field is 0, nobody using this hooks and we can delete it.

This is why we can remove code from `cont.c`.

- `vm_core.h (rb_vm_t)`: because of above change, we can eliminate `rb_vm_t::trace_running` field.
Also renamed from `rb_vm_t::event_hooks` to `global_hooks`.
- `vm_core.h, vm.c (ruby_vm_event_enabled_global_flags)`: renamed from ``ruby_vm_event_enabled_flags`.
- `vm_core.h, vm.c (ruby_vm_event_local_num)`: added to count enabled targetting TracePoints.
- `vm_core.h, vm_trace.c (rb_exec_event_hooks)`: accepts hook list.
- `vm_core.h (rb_vm_global_hooks)`: added for convinience.
- `method.h (rb_method_bmethod_t)`: added to maintain Proc and `rb_hook_list_t` for `bmethod` (defined by `define_method`).
- `prelude.rb (TracePoint#enable)`: extracet a keyword parameter (because it is easy than writing in C).
It calls `TracePoint#__enable` internal method written in C.
- `vm_inshelper.c (vm_trace)`: check also `iseq->local_hooks`.
- `vm.c (invoke_bmethod)`: check `def->body.bmethod.hooks`.
- `vm.c (hook_before_rewind)`: check `iseq->local_hooks` and `def->body.bmethod.hooks` before rewind by exception.

#4 - 11/28/2018 01:56 AM - k0kubun (Takashi Kokubun)

- Status changed from Closed to Assigned

I reopen this ticket as a reminder to fix MinGW test failure by this. Please close this ticket once AppVeyor becomes green with [r66062](#) reverted.

#5 - 11/29/2018 11:36 AM - k0kubun (Takashi Kokubun)

- Status changed from Assigned to Closed

The above issue was fixed in [r66087](#). Thank you.