Ruby master - Feature #13483

TracePoint#enable with block for thread-local trace

04/19/2017 07:38 AM - ko1 (Koichi Sasada)

Status:	Rejected
Priority:	Normal
Assignee:	ko1 (Koichi Sasada)
Target version:	2.5

Description

Summary

TracePoint#enable with block should enable thread-local trace.

Current behavior

TracePoint#enable enables TracePoint for all of threads, even if it called with do...end blcok.

```
t1 = Thread.new{
  loop{
    x = 1
  }
}
th = nil
trace = TracePoint.new(:line){|tp|
  if th != Thread.current
    p th = Thread.current
  end
}

trace.enable do
  loop{
    a = 1
    b = 2
  }
end
```

This program shows both main thread and thread t1 hooked by line events.

Problem

However, usually trace.enable do ... end imply the programmer want to enable hooks only for this block, not for other threads.
For example, Ruby's test for TracePoint skips hooks on other threads.
https://github.com/ruby/ruby/blob/trunk/test/ruby/test_settracefunc.rb#L620

Proposal

TracePoint#enable with block should enable thread-local trace.
I believe proposed behavior is easy to use.

Consideration

(1) It breaks backward compatibility. Is it acceptable?
(2) What happen on created threads? Should inherit thread-local hooks or ignore them?

I want to ask users of TracePoint.

Thanks,
Koichi

History

#1 - 04/28/2017 01:45 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

#2 - 05/19/2017 07:30 AM - nobu (Nobuyoshi Nakada)

- Description updated

#3 - 05/19/2017 07:31 AM - ko1 (Koichi Sasada)

Matz approved it.

#4 - 05/26/2017 05:32 AM - ko1 (Koichi Sasada)

After consideration, I found several problems.

There is a implicit expectation that we can emulate block accept call with begin/ensure like:

```
open do
  ...
end
#=> same as:
begin
  open
  yield
ensure
  close
end
```

However, this proposal breaks this expectation. So I reject this ticket.

I try to consider to introduce how to filter the probes, like:

```
trace = TracePoint.new(:line, thread: Thread.current){ ... }
trace.enable #=> only enable on the current thread.

trace = TracePoint.new(:line, file: __FILE__){ ... }
trace.enable #=> only enable on this file.

trace = TracePoint.new(:line, method: method(:foo)){ ... }
trace.enable #=> only enable on `foo` method.
```

Thanks,
Koichi

#5 - 05/26/2017 05:32 AM - ko1 (Koichi Sasada)

- Status changed from Assigned to Rejected

#6 - 05/27/2017 09:49 AM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote:

> However, this proposal breaks this expectation.

Could you explain it?

Is it because trace.enable { code } does not behave like
begin; trace.enable; code; ensure; trace.disable; end ?

If so, I think this problem could be avoided by just changing the name to imply "thread-local",
such as trace.enable_for_current_thread { code } or
trace.enable_in_block { code }.

#7 - 05/30/2017 06:41 AM - ko1 (Koichi Sasada)

On 2017/05/27 18:49, eregontp@gmail.com wrote:

> > However, this proposal breaks this expectation.
> > Could you explain it?
>
> Is it because trace.enable { code } does not behave like

> begin; trace.enable; code; ensure; trace.disable; end ?

Yes.

> If so, I think this problem could be avoided by just changing the name to imply "thread-local",
> such as trace.enable_for_current_thread { code } or
> trace.enable_in_block { code }.

Yes.

This is what

> I try to consider to introduce how to filter the probes, like:

Considerations about introducing "thread-lcoal" enable:

(1) POSITIVE: because it may be common use case to enable.
(2) NEGATIVE:
(2-1) because enable_xxx seems verbose.
(2-2) because we will want to introduce similar method to
limit file name or method name, like enable_file do ... end
(this is why I proposed keyword arg)

--
// SASADA Koichi at atdot dot net

#8 - 12/03/2018 09:33 PM - ioquatix (Samuel Williams)

ko1 (Koichi Sasada) Was there an update to this proposal?