Ruby - Feature #17881

Add a Module#const_added callback

05/22/2021 11:43 AM - byroot (Jean Boussier)

Status:	Closed	
Priority:	Normal	
Assignee:	byroot (Jean Boussier)	
Target version:		

Description

Use case

Autoloaders like zeitwerk need a callback when a new class or module is registered in the constant table.

Currently this is implemented with TracePoint's :class event. It works, but it is a bit unfortunate to have to use an API intended for debugging to implement production features. It doesn't feel "conceptually clean".

It also doesn't play well with MJIT, even though it's more of an MJIT limitation.

Additionally this usage of TracePoint cause some incompatibilities with some debuggers like byebug (even though others don't have this issue).

Proposal

I believe that if Ruby was to call Module#const_added when a constant is registered, Zeitwerk could get rid of TracePoint.

For now I implemented it as: const_added(const_name) for similarity with method_added. But maybe it could make sense to have the signature be const_added(const_name, const_value).

Also since method_removed exists, maybe const_removed would need to be added for consistency.

Links

Patch: https://github.com/ruby/ruby/pull/4521
Zeitwerk side discussion: https://github.com/fxn/zeitwerk/issues/135

cc @k0kubun (Takashi Kokubun)

Associated revisions

Revision 8d05047d72d0a4b97f57b23bddbca639375bbd03 - 01/14/2022 10:30 AM - byroot (Jean Boussier)

Add a Module#const_added callback

[Feature #17881]

Works similarly to method_added but for constants.

```
Foo::BAR = 42 # call Foo.const_added(:FOO)
class Foo::Baz; end # call Foo.const_added(:Baz)
Foo.autoload(:Something, "path") # call Foo.const_added(:Something)
```

Revision 8d05047d72d0a4b97f57b23bddbca639375bbd03 - 01/14/2022 10:30 AM - byroot (Jean Boussier)

Add a Module#const_added callback

[Feature #17881]

Works similarly to method_added but for constants.

```
Foo::BAR = 42 # call Foo.const_added(:FOO)
class Foo::Baz; end # call Foo.const_added(:Baz)
Foo.autoload(:Something, "path") # call Foo.const_added(:Something)
```

Revision 8d05047d - 01/14/2022 10:30 AM - byroot (Jean Boussier)

Add a Module#const_added callback

[Feature #17881]

Works similarly to method_added but for constants.

```
Foo::BAR = 42 # call Foo.const_added(:FOO)
class Foo::Baz; end # call Foo.const_added(:Baz)
Foo.autoload(:Something, "path") # call Foo.const_added(:Something)
```

History

#1 - 05/22/2021 12:00 PM - Eregon (Benoit Daloze)

This would trigger for all constants, which might affect startup quite a bit by having all those extra calls (I already see it as a problem for method_added, but maybe we can optimize for the common case that it's not redefined).
Maybe we should have a callback just for modules/classes bodies, exactly like the :class TracePoint?

I think a JIT doesn't need to give up anything for the :class TracePoint.
In fact, it even seems reasonable to always check if there is an active :class TracePoint when entering a module body, module bodies are after all rarely executed more than once, and the check should be fairly cheap (especially compared to creating a new Module).

#2 - 05/22/2021 12:09 PM - byroot (Jean Boussier)

> This would trigger for all constants

Indeed, as it seemed the most consistent API to me. But I'm open to an alternative that would only trigger for class / module. Not sure how it could be named though.

> which might affect startup quite a bit by having all those extra calls

I haven't measured, but I doubt it would be slower than the current TracePoint.new(:class) callback.

> I already see it as a problem for method_added

We have that hook defined in our app because of sorbet, and the overhead really isn't that bad.

> I think a JIT doesn't need to give up anything for the :class TracePoint.

Of course, it's only a current MJIT limitation. I only mentioned it because @k0kubun's blog post is what prompted me to look at this again. So it's definitely not the main reason to do this, it could just happen to be a nice side effect. A JIT that strip tracing code, and simply de-optimize when tracing is enabled doesn't seem that far fetched to me.

#3 - 05/22/2021 12:13 PM - byroot (Jean Boussier)

- Description updated

#4 - 05/22/2021 12:16 PM - Eregon (Benoit Daloze)

- Description updated

> A JIT that strip tracing code, and simply de-optimize when tracing is enabled doesn't seem that far fetched to me.

Yes, and I think it makes a lot of sense to deoptimize/recompile for e.g., :line and :call events (those have a huge perf impact).
But for :class I don't see how avoiding the check at the cost of deoptimization/recompilation would ever be worth it.

#5 - 05/22/2021 12:18 PM - Eregon (Benoit Daloze)

Ugh, Redmine doesn't handle posting a comment concurrently and now incorrectly reverted the description changes even though I didn't touch the description.
@byroot (Jean Boussier) Could you re-apply your changes?

#6 - 05/22/2021 12:20 PM - byroot (Jean Boussier)

- Description updated

#7 - 05/22/2021 12:36 PM - Eregon (Benoit Daloze)

One potential overhead of the :class TracePoint is that it allows accessing the binding through TracePoint#binding:

```
$ ruby -e 'TracePoint.new(:class) { |tp| p tp; p tp.self; p tp.binding.local_variables }.enable; a=1; class C;
 b=2; end'
#<TracePoint:class@-e:1>
C
[:b]
```

I'm not sure how much it matters in practice, probably not a lot.

If we add some more direct callback instead we won't have that issue, but we'd need to lookup the callback method every time vs just checking a global flag "any class TracePoint enabled".

#8 - 05/22/2021 12:39 PM - byroot (Jean Boussier)

Honestly I shouldn't have mentioned current performance at all in the first place, it's really not the main reason we'd like this new API.

However you do have a point that this new API should be at least as performant, otherwise it will be a difficult value proposition.

#9 - 05/22/2021 01:31 PM - fxn (Xavier Noria)

Thanks a lot for working on this Jean.

If we add this, Zeitwerk would be using a somewhat more "normal" API. TracePoint is correct, but seems a bit off to me conceptually (I do not know if I should revise that perception, maybe should I?)

However, the comments by Eregon are key. The JIT is WIP and it is going to be revised, there are JITs that work with TP just fine, current Ruby has no issue with a TP on the :class event, etc. It would be a patch in light of proposing an API that makes sense and would have at least this use case, and would make Zeitwerk a tad less hacky (again, modulus technically it is correct).

There's also something that makes me think. The constants API, in general, does not distinguish constants defined for real, from constants that have only an autoload configured that has not been triggered. For example, if you autoload :Foo, "file", and "file" is not evaluated, the constants API for Object tells you Foo is a constant in Object.

So, to be coherent with this, perhaps const_added should be triggered in autoload calls. But that would not help Zeitwerk, because what we need is to grab the class/module object at the top of its class/module body. And, we do not need to be called on reopenings (something the :class event does).

In 2018, I even played with the idea of decorating some low-level method in Module that allowed me to intercept new class/module objects being created, but that does not seem reachable from Ruby, so took the TP as a necessary evil. It is the only technique I know to support explicit namespaces.

#10 - 05/22/2021 05:01 PM - fxn (Xavier Noria)

Let me also add something for context. @byroot (Jean Boussier) knows it and probably @Eregon (Benoit Daloze) too, just for anybody else following.

Something the TP on the :class event allows you to do is to enable only if you need it. Allows you to be precise.

If the TP is not needed ever, it is not even enabled. If at some point you needed it, but no longer, Zeitwerk disables it. The tracer is managed by this module where you can see the fine logic in register/deregister.

A TP on the :class event is only needed if some of the loaders in the Ruby process saw both a file "foo.rb" and a directory "foo", and Foo has not been yet loaded. As soon as Foo is loaded, if there are no more constants in a similar situation, the tracer is disabled.

In particular, the tracer becomes normally disabled in a Rails application in production mode because Zeitwerk eager loads not only the application, but any other gem managed by Zeitwerk. That is what Zeitwerk::Loader.eager_load_all does here.

So, I find a callback to be neat from an implementation standpoint (if one associates TracePoint to debugging use cases), but you wouldn't normally be able to be so precise. Though you could probably at least bail out first thing in the block body via a maintained flag or something.

TP on the :class event, ironically, seems a tad off for me, but it happens to be perfect to solve this problem. There is no performance penalty that I know of, and in the case of JITs, on paper it should be fine (it is in a JIT @byroot (Jean Boussier) checked).

So... no idea, it's core's call :).

#11 - 05/22/2021 05:16 PM - byroot (Jean Boussier)

> If the TP is not needed ever, it is not even enabled. If at some point you needed it, but no longer, Zeitwerk disables it.

True, but what worries me would be gems using Zeitwerk and not fully eagerloading, there's even an explicit API for doing just that highligthed in the readme: https://github.com/fxn/zeitwerk#eager-loading

```
db_adapters = "#{__dir__}/my_gem/db_adapters"
loader.do_not_eager_load(db_adapters)
loader.setup
loader.eager_load # won't eager load the database adapters
```

If I'm not mistaken, such usage could cause the TracePoint to stay active at runtime, which isn't a huge deal, but will still trigger every time a singleton_class or anonymous class is created, which isn't a good practice at runtime, but that I nonetheless see happen.

On the other hand, assigning constants is extremely rare past the boot phase, so I'm much less worried about it.

But again all this is hard to quantify and a bit "handwavy". I believe the cleanliness of the API alone should be the main argument, as well as not interfering with other TracePoint usages.

#12 - 05/22/2021 05:29 PM - fxn (Xavier Noria)

> If I'm not mistaken, such usage could cause the TracePoint to stay active at runtime, which isn't a huge deal, but will still trigger every time a singleton_class or anonymous class is created, which isn't a good practice at runtime, but that I nonetheless see happen.

Yes, that is right. Is in that sense that I said *normally*. You'd need someone to opt out from eager loading a subtree, and to have explicit namespaces in there in a level visited by the tree walker (which only descends on demand). It is a possibility, but I'd believe should be rare. Rare enough not to be your main design driver.

> But again all this is hard to quantify and a bit "handwavy". I believe the cleanliness of the API alone should be the main argument

Agree, and I believe it is core's call to decide if that is cleaner, perhaps Zeitwerk's usage of TP is already "clean" in their opinion. I do not know :).

#13 - 05/23/2021 05:36 AM - fxn (Xavier Noria)

Another gotcha I see is that we cannot assume const_added is exclusive to Zeitwerk. Users could have their own. So we would need to prepend our own, to make sure it is not rewritten. And on reload, we would need to keep track of which non-reloadable class/module already has the module prepended, not to prepend again. And would need to document that if you prepend, make sure to call super in case there's ours.

What I had in mind in the conversation in the Zeitwerk ticket was not const_added. It was a callback that gets called when a new class/module is created and after it gets its name set if defined with the class/module keywords. Analogous to what we use today. That is, some sort of global chainable callback "on_new_module(&block)", hence all these remarks :).

So, as I see it, these are some things to think about:

1. If there's an autoload for Foo, Object.constants has Foo, Object.const_defined?(Foo) is true, etc. but you did not get a const_added call for it. That does not seem coherent. On the other hand, if we make it coherent, it is not usable by Zeitwerk. This is an important one, semantically.

2. Zeitwerk cannot assume it controls const_added, needs cooperation from the user (this is doable via documentation).

3. There is no performance issue to address that I know. I believe this patch should be discussed on a generic Ruby API context, independently of Zeitwerk.

4. What would really work for Zeitwerk if we want a "cleaner" API is a chainable on_new_module(&block).

I am not exactly proposing on_new_module(&block) because the :class event of TP already does that (plus reopening, it is less precise), and because does no seem like existing Ruby hooks. I am mentioning this to highlight the difference between what I thought Zeitwerk could benefit from, from what we are discussing.

#14 - 05/23/2021 05:52 AM - fxn (Xavier Noria)

Oh, and while Zeitwerk can disable the TracePoint, it wouldn't be able to disable on_new_module(&block), at most have a flag.

My personal take is: As far as Zeitwerk is concerned, I'd study some API if core believes we should not use a TP on the :class event.

Otherwise, personally, **I would do nothing**.

Then, the conversation is legit as Ruby API per se, if you like. But in that case, I believe you should get const_added called on autoload calls for consistency with existing semantics.

#15 - 05/23/2021 07:08 AM - fxn (Xavier Noria)

> If I'm not mistaken, such usage could cause the TracePoint to stay active at runtime, which isn't a huge deal, but will still trigger every time a singleton_class or anonymous class is created, which isn't a good practice at runtime, but that I nonetheless see happen.

For singleton classes, in the (rare?) eventual case that the tracer remains enabled we return immediately. The cost of that has to be absolutely negligible in practice.

Regarding anonymous classes and modules, remember TP does NOT trigger :class on Class.new or Module.new, which is perfect for our use case, since Zeitwerk is only interested in classes/modules with names.

#16 - 05/23/2021 10:38 AM - byroot (Jean Boussier)

> What I had in mind in the conversation in the Zeitwerk ticket was not const_added. It was a callback that gets called when a new class/module is created and after it gets its name set if defined with the class/module keywords.

I see, seems like I missed that.

I suppose the "ideal" API for an autoloader like Zeitwerk would be something like Module#module_defined(name, module):

- Would trigger on classic module Foo::Bar first opening.
- Would trigger on Foo::Bar = Class.new. (This one being a current limitation with TracePoint if I'm not mistaken).
- Would **not** trigger on Foo.autoload(:Bar, "bar").
- Would not trigger for non Module or Class constants.

#17 - 05/23/2021 12:10 PM - fxn (Xavier Noria)

> Would trigger on Foo::Bar = Class.new. (This one being a current limitation with TracePoint if I'm not mistaken)

Correct. Nowadays, you cannot define an explicit namespace with a constant assignment like that (docs), because Class.new does not trigger the :class event (test coverage, since a certain optimization is based on this).

#18 - 07/18/2021 09:19 AM - fxn (Xavier Noria)

Followup here.

Before I started working on Zeitwerk, I benchmarked whether a TP enabled on the :class event had a measurable impact. Richard Schneeman and Sam Saffron ran benchmarks too. Tests said it did not, that validated the technique for explicit namespaces and gave me green light to go ahead.

@k0kubun (Takashi Kokubun) recently ran a new synthetic benchmark with similar results and concluded:

> ... so, in summary, Zeitwerk doesn't have to change for VM, and thus we don't need new API for it. While both MJIT and YJIT have a challenge to optimize TP-enabled programs, :class events could be a special case which is easier to optimize at least than :line. I'll work on it.

I am told YJIT works fine with Zeitwerk (right @byroot (Jean Boussier)?), and MJIT will work on it.

So, as far as Zeitwerk is concerned, I believe we are fine and we do not need new APIs.

This ticket could be pursued, if you like, for the sake of the Ruby language itself.

#19 - 08/13/2021 06:35 AM - k0kubun (Takashi Kokubun)

- Status changed from Open to Feedback

FYI, I changed the MJIT implementation in ac4d53bd461ff386cd45fdd484ffb6b628a251ad to not disable JIT-ed code when a TracePoint for class events gets enabled. So, as long as Zeitwerk only uses TracePoints for class events, we no longer need this API for MJIT w/ Zeitwerk either.

#20 - 12/03/2021 11:02 PM - brasic (Carl Brasic)

I came across this issue when researching why zeitwerk-enabled applications can fail to autoload constants inside ruby/debug breakpoints (see issues below). It does seem that an alternative to the :class tracepoint like the one proposed here could be useful to reduce conflicts between zeitwerk and the various debuggers that cannot avoid the use of tracepoint.

1. https://github.com/ruby/debug/issues/408
2. https://github.com/rails/rails/issues/43691

#21 - 12/03/2021 11:13 PM - byroot (Jean Boussier)

I thought debug didn't have this tracepoint issue, hence why I didn't bother to push this more. But if it does, then yes I do think we need to add non-tracepoint API for Zeitwerk.

What's too bad is that's it's a bit too late for 3.1 :/

#22 - 12/03/2021 11:19 PM - byroot (Jean Boussier)

I added the ticket back to the next developer meeting (Dec 9th) agenda.

#23 - 12/04/2021 03:40 PM - Eregon (Benoit Daloze)

To clarify my earlier concern, I'm against adding a hook triggered on every constant defined, because of the involved startup cost of doing so.
A hook just for class Foo/module Foo would be more reasonable from a startup perf POV, but still it feels like we're just working around the real problem with TracePoint.

#24 - 12/04/2021 08:48 PM - cvss (Kirill Vechera)

Alternative solution can be implemented with two hooks for "opening" and "closing" class/module definition i.e. Module::on_open. One can get the existing list of constants, methods, class variables etc on the opening, and calculate difference on the closing - added, removed, changed components.

```
module M # #on_open called
 A = 'a'
 def b; end
end # #on_close called
```

Setting a constant out of a module definition:

```
M::C = 'c' # #on_open called before assignment, #on_close called after assignment
```

It still cause a slight performance overhead for the nested classes and modules due to invoking the hooks for each nesting level, i.e. defining a nested module O in the existing module M would call both the hooks twice for M and for M::O. And this approach will also ease some other meta-programming methods and will allow auto-decorating of nested classes and modules.

#25 - 12/09/2021 07:59 AM - matz (Yukihiro Matsumoto)

Accepted. Although I think it's too late for the 3.1 release.

Matz.

#26 - 12/09/2021 08:01 AM - byroot (Jean Boussier)

Thank you.

> Although I think it's too late for the 3.1 release.

Assuming https://bugs.ruby-lang.org/issues/15912 does make it through for 3.1, then it's no big deal.

#27 - 12/14/2021 11:37 AM - Eregon (Benoit Daloze)

I highly doubt const_added is a good API for zeitwerk, i.e. if someone defines const_added and doesn't call super it will break zeitwerk.
TracePoint.new(:class) has no such problems, TracePoint.new(:class) seems already the best API according to @fxn in https://bugs.ruby-lang.org/issues/17881#note-18

So, I think we should not add this until we have a good use case.

#28 - 12/14/2021 11:42 AM - byroot (Jean Boussier)

> So, I think we should not add this until we have a good use case.

Yeah, this is no longer for Zeitwerk, since TracePoint reentrancy was made possible by @ko1 (Koichi Sasada).

We still want it though for https://github.com/Shopify/tapioca, cc @ufuk (Ufuk Kayserilioglu)

From our conversation:

> Yeah, all consts would be very handy for my use case indeed. I can already get class/module definitions/reopens with TracePoint, but cannot catch other constant definitions. We end up having to query const_source_location for those.

The super issue is totally acceptable, that's already a common issue with inherited and such, and it's very easily caught by static checkers.

#29 - 01/14/2022 02:39 AM - mame (Yusuke Endoh)

- Status changed from Feedback to Assigned

- Assignee set to byroot (Jean Boussier)

@byroot (Jean Boussier) Do you merge your pull request to the HEAD? This proposal is already approved by matz.

#30 - 01/14/2022 08:21 AM - byroot (Jean Boussier)

@mame (Yusuke Endoh) I will today, thank you for the notice.

#31 - 01/14/2022 10:30 AM - byroot (Jean Boussier)

- Status changed from Assigned to Closed

Applied in changeset git|8d05047d72d0a4b97f57b23bddbca639375bbd03.

Add a Module#const_added callback

[Feature #17881]

Works similarly to method_added but for constants.

```
Foo::BAR = 42 # call Foo.const_added(:FOO)
class Foo::Baz; end # call Foo.const_added(:Baz)
Foo.autoload(:Something, "path") # call Foo.const_added(:Something)
```