Ruby - Feature #18231

`RubyVM.keep_script_lines`

09/30/2021 08:16 AM - ko1 (Koichi Sasada)

Status:	Closed
Priority:	Normal
Assignee:	ko1 (Koichi Sasada)
Target version:	3.1

Description

This ticket proposes the method RubyVM.keep_script_lines to manage the flag to keep the source code.

Now, SCRIPT_LINES__ = {} constant is defined, the hash will stores the compiled script with a path name ({path => src}). However, eval source code doesn't stores in it (maybe because it can be bigger and bigger).

Proposal

This proposal to add script lines to the ISeq and AST.

```
RubyVM::keep_script_lines = true

eval("def foo = nil\ndef bar = nil")
pp RubyVM::InstructionSequence.of(method(:foo)).script_lines
#=> ["def foo = nil\n", "def bar = nil"]
```

In this case, methods foo and bar are defined by eval() and ISeq of foo can returns script lines (all script for eval).
ISeq#script_lines returns compiled script lines.

When ISeq is GCed, then the source code will be also free'ed.

Discussion

- This feature will be used by debugger (to show the source code) or REPL support (irb and so on) with error_highlight.
- Of course memory usage will be increased.
- We can introduce new status only_eval in future which will help REPL systems.
- We can implement ISeq#source method like AST#source method (similar to toSource in JavaScript), but this ticket doesn't contain it.

Implementation

https://github.com/ruby/ruby/pull/4913

For the Ractor support script lines object should be immutable (not implemented yet).

Related issues:

Related to Ruby - Feature #17930: Add column information into error backtrace	**Closed**
Related to Ruby - Feature #18159: Integrate functionality of syntax_suggest g...	**Closed**
Related to Ruby - Feature #6012: Proc#source_location also return the column	**Closed**

History

#1 - 09/30/2021 09:13 AM - Eregon (Benoit Daloze)

This new API must be implement-able for other Ruby implementations.
Hence, it must not be on RubyVM, and it should take a Method/UnboundMethod or filename.

#2 - 09/30/2021 12:00 PM - ko1 (Koichi Sasada)

> This new API must be implement-able for other Ruby implementations.

It is ISeq/AST related API so it should be internal.

If we found another good place to implement it, we can implement them later.

#3 - 09/30/2021 08:03 PM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote in #note-2:

> It is ISeq/AST related API so it should be internal.

I don't see how it is ISeq or AST-related.

The new script_lines just returns the source code as an array of strings for a given method.
So, why not simply {Method,UnboundMethod}#script_lines and e.g. Method.keep_script_lines = true?

#4 - 10/01/2021 01:07 AM - ko1 (Koichi Sasada)

for debugger/error_highlight we need source code other than methods so Method is not enough.
I agree Method#script_lines can be considered but it's not enough.

#5 - 10/02/2021 10:36 AM - Eregon (Benoit Daloze)

I am strongly against this proposal as it currently stands.

This is yet another hack added under RubyVM.
I think adding new APIs under RubyVM is careless and lazy, because:

- It hurts the portability of the Ruby language, this code won't work on other Ruby implementations
- Because it's under RubyVM CRuby devs feel they can add anything because it's a "special module where things are experimental", even if the API is hacky and unreliable
- Such API will typically be used by gems at some points(e.g. because no other proper public API provides the functionality), but that's bad because it's typically hacky and unreliable, i.e., it's likely to break the usages whenever it evolves (e.g., RubyVM exposing the internal order or AST children which obviously will evolve when the syntax evolves)

I think every time someone thinks to add a new API under RubyVM they should instead think what a proper API would look like, and propose that instead.
Take the time to design a new API, don't rush a hack under RubyVM.

So, for our concrete use cases here:

For error_highlight I already mentioned from the start it was a bad idea to read code from disk and re-parse (because that's unreliable). And now it's clear this also doesn't work for eval'd code.
What we should do is design a proper API.
What does error_highlight need? Is it just the code location (i.e. col+line+length info) of the whole call and of the receiver? Maybe we can add such info in Exception#backtrace_locations and as NameError#receiver_code_location then.
Is it more complicated than that and it needs more info from the AST? How about preserving the needed metadata when parsing (so as some bytecode metadata for CRuby), and then expose it cleanly like NameError#highlight_code_location.
Since error_highlight needs to show the code, we also need a way to access the code for a loaded source. Those code_location could be objects and have that functionality (e.g., via to_s or some other method).
It's not hard, but both of these designs are much more reliable than the current error_highlight, they are portable and not hacky.

For the debugger/debug gem, could you describe what is needed?

RubyVM::keep_script_lines = true feels hacky.
Either CRuby is OK to preserve the source for every source loaded (until the whole file/source is GC'd), or it's not.
Also returning a String of the code location's code seems better than already splitting in lines for script_lines.
I would think having code_location on exceptions and methods is also enough for the debugger.

#6 - 10/02/2021 02:46 PM - ko1 (Koichi Sasada)

> I would think having code_location on exceptions and methods is also enough for the debugger.

How to get the proper source code?

#7 - 10/02/2021 02:51 PM - ko1 (Koichi Sasada)

I agree RubyVM namespace is not portable, and not well-considered API.

So I think:

- Ruby users should not use them if they can not catch up future changes and portability-limitation.
- If there are other better API, they should be provided.

#8 - 10/03/2021 08:31 AM - duerst (Martin Dürst)

Eregon (Benoit Daloze) wrote in #note-5:

> I am strongly against this proposal as it currently stands.
>
> For error_highlight I already mentioned from the start it was a bad idea to read code from disk and re-parse (because that's unreliable). And now it's clear this also doesn't work for eval'd code.

The dead_end gem (that we will try to integrate into Ruby early next year, see #18159) also needs code, and as far as I understand, it also reads it from disk. Can you say exactly what the problem is? Is it that the file may get changed? We should try to find a solution that works not only for error_highlight, but also for dead_end and others, and of course preferably for all Ruby implementations.

#9 - 10/03/2021 09:53 AM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote in #note-6:

> I would think having code_location on exceptions and methods is also enough for the debugger.
>
> How to get the proper source code?

code_location would be an object, so e.g., method(:foo).code_location.code or method(:foo).code_location.to_s => String of the code spanning that code_location.

#10 - 10/03/2021 09:59 AM - Eregon (Benoit Daloze)

duerst (Martin Dürst) wrote in #note-8:

> Can you say exactly what the problem is? Is it that the file may get changed?

Yes, it might get changed, which might then cause confusing errors when parsing it again.
The file could also have been removed in between.
Of course that approach cannot work for eval (there is no file).
And finally it might also break when .rb files are packaged (e.g., ruby-packer, in some archive, or some virtual filesystem) since then there is also no real file.

eval is the most visible issue, because the value of __FILE__ in eval is not reffering to a real file, or if it is that file typically it has different contents (the whole file and not just the String eval'd).

#11 - 10/04/2021 03:05 AM - matz (Yukihiro Matsumoto)

I don't want to bother progressing here, so I accept RubyVM.keep_script_lines with the indication of implementation dependency (and possible fragility).

When we got a better place, let's move on then.

Matz.

#12 - 10/04/2021 09:38 AM - Eregon (Benoit Daloze)

We already have something similar to "code locations" objects: Thread::Backtrace::Location.
We could add #{first,last}_{column,line} and #code to Thread::Backtrace::Location, how about that?
And then add a way to get a Thread::Backtrace::Location from a Method.

In Truffle terminology each code is backed by a Source object and each method/AST node has a SourceSection, equivalent to these "code locations".

If we make this new API dependent on RubyVM, we give up needlessly on having both error_highlight and the debug gems working on non-CRuby.
I'd say that's a shame, because there is no need.

#13 - 10/04/2021 12:29 PM - ko1 (Koichi Sasada)

Eregon (Benoit Daloze) wrote in #note-12:

> We already have something similar to "code locations" objects: Thread::Backtrace::Location.
> We could add #{first,last}_{column,line} and #code to Thread::Backtrace::Location, how about that?
> And then add a way to get a Thread::Backtrace::Location from a Method.

Do you talking about toSource method in JS?

> We can implement ISeq#source method like AST#source method (similar to toSource in JavaScript), but this ticket doesn't contain it.

#14 - 10/04/2021 12:38 PM - Eregon (Benoit Daloze)

ko1 (Koichi Sasada) wrote in #note-13:

> Do you talking about toSource method in JS?

Yeah it's JS Function's toSource, but since it's an object it would also provide more information like line & column info.

> for debugger/error_highlight we need source code other than methods so Method is not enough.

For what else do you need to get the source? Ruby::AbstractSyntaxTree::Node?
How do you get the Ruby::AbstractSyntaxTree::Node if it's eval'd code?

Is it really needed on RubyVM::InstructionSequence?
That's probably the worst for portability as some Ruby implementations don't have bytecode or don't want to expose it.

#15 - 10/04/2021 03:02 PM - ko1 (Koichi Sasada)

> For what else do you need to get the source? Ruby::AbstractSyntaxTree::Node?
> How do you get the Ruby::AbstractSyntaxTree::Node if it's eval'd code?

This is why we propose this feature.

> Is it really needed on RubyVM::InstructionSequence?
> That's probably the worst for portability as some Ruby implementations don't have bytecode or don't want to expose it.

I agree that.

#16 - 10/04/2021 04:51 PM - Eregon (Benoit Daloze)

I clarified a bit with @ko1 (Koichi Sasada).
First, script_lines returns all lines, not the lines of a specific method (I missed that in the description).
The debugger use case is to show the code around a breakpoint.

Adding something like caller_locations(0)[0].script_lines would work too, and it would be portable.
@matz (Yukihiro Matsumoto) Could you consider if Thread::Backtrace::Location#script_lines is acceptable?
This is easy to implement in other Ruby implementations, while going via ISeq is very hard, and would make the debug gem unusable on non-CRuby which would be unfortunate.

Also, do we need an explicit RubyVM.keep_script_lines = true, or would it be fine to always save the source?
FWIW TruffleRuby always keeps the source code (until no code in that loaded file can run anymore).

#17 - 10/04/2021 05:20 PM - ko1 (Koichi Sasada)

Some comments:

- For debugger, Thread::Backtrace::Location#script_lines is enough.
- If we introduce this feature outside of RubyVM, people can use it casually and I'm not sure we can support it well because this feature (keeping all loaded sources). For example, if I install rails with gem install rails and [master]$ du -sh .../lib/ruby/gems/3.1.0/gems/ returns 72MB.

#18 - 12/14/2021 04:58 PM - ko1 (Koichi Sasada)

- Status changed from Open to Closed

c7550537f11dcf6450a9d3df3af3fa1f4fe05b15

#19 - 06/17/2022 02:02 PM - Eregon (Benoit Daloze)

- Related to Feature #17930: Add column information into error backtrace added

#20 - 06/17/2022 02:02 PM - Eregon (Benoit Daloze)

- Related to Feature #18159: Integrate functionality of syntax_suggest gem into Ruby added

#21 - 09/27/2024 03:21 PM - Eregon (Benoit Daloze)

- Related to Feature #6012: Proc#source_location also return the column added