Ruby - Feature #16245

Add interfaces to count and measure size all IMEMO objects

10/07/2019 10:02 PM - sam.saffron (Sam Saffron)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Koichi introduced an experimental gem: https://github.com/ko1/iseq_collector

It allows:

ObjectSpace.each_iseq{|iseq| ...}
ObjectSpace.count_iseq #=> Integer
ObjectSpace.memsize_of_all_iseq (should not generate RubyVM::InstructionSequence wrappers for IMEMOs)

Since the wrapper object RubyVM::InstructionSequence is lazily allocated, ObjectSpace.each_object does not find these IMEMOs unless they have been wrapped. This design is good and conserves memory.

count_iseq and memsize_of_all_iseq are very powerful metrics most large Ruby deployments can use to automatically detect method leaks introduced via meta programming. These issues are invisible now short of walking a heap dump.

Can we add the new interface into 2.7?

History

#1 - 10/07/2019 10:07 PM - sam.saffron (Sam Saffron)

An alternative design could be to add 1 extra object to the heap

RubyVM::NonMaterializedInstructionSequences (sizeof all IMEMOS that are not wrapped)

Then each_object could include it and we could use that to measure size of all IMEMOs that are not materialized yet. Advantage here is that naive memsize_of all objects will return all the memory.

#2 - 10/07/2019 10:52 PM - methodmissing (Lourens Naudé)

sam.saffron (Sam Saffron) wrote:

> Koichi introduced an experimental gem: https://github.com/ko1/iseq_collector
>
> It allows:
>
> ObjectSpace.each_iseq{|iseq| ...}
> ObjectSpace.count_iseq #=> Integer
> ObjectSpace.memsize_of_all_iseq (should not generate RubyVM::InstructionSequence wrappers for IMEMOs)
>
> Since the wrapper object RubyVM::InstructionSequence is lazily allocated, ObjectSpace.each_object does not find these IMEMOs unless they have been wrapped. This design is good and conserves memory.
>
> count_iseq and memsize_of_all_iseq are very powerful metrics most large Ruby deployments can use to automatically detect method leaks introduced via meta programming. These issues are invisible now short of walking a heap dump.
>
> Can we add the new interface into 2.7?

I worked on imemo_memsize some time ago to correctly reflect the type sizes in https://github.com/ruby/ruby/commit/90c4bd2d2bd10b19c2b09834396553742bc7e8a4 which makes heap dumps more accurate. I understand the API proposal, but also I believe the intention was for these objects to be internal and not necessarily to be exposed through API. However I do suspect for large Rails applications their combined footprint can add up, especially for the types that can allocate heap memory too:

- imemo_ment (method entries)
- imemo_iseq (as per your description above)
- imemo_env (bindings)
- imemo_tmpbuf (tried to support these on the transient heap but found them to be almost never used much in practice as it appears to be a fallback for ALLOCA under some circumstances.
- imemo_ast

I have not had any free time to investigate further, but I think this is an interesting storage class to explore further and I'd be interesting in helping, whichever way the proposal goes.

#3 - 10/08/2019 05:55 PM - shevegen (Robert A. Heiler)

Personally I love introspection so I am all in favour of giving ruby people lots of tools to play with internal. I also liked oldschool evi.rb. :)

I guess this is for koichi to comment e. g. how stable he considers the gem/code; and possibly also whether the API is wanted in the first place. And perhaps also whether the name iseq is already an official name or not, ruby-internal wise (I really don't know, just pointing that out).

As for the name **NonMaterializedInstructionSequences** - I think that name is too long and complicated. Ideally accessing should be simple, whenever possible, in my opinion. I am not even sure what a "non-materialized instruction sequence" is - is that ruby's version of a monoid-endofunctor monad?

IMO, simpler names would be better. Although I guess if the functionality is what matters, then I guess we may agree that the functionality can be useful.

methodmissing wrote:

> I understand the API proposal, but also I believe the intention was for these objects to be internal and not necessarily to be exposed through API.

Yeah, I think I have read similar discussions in the past, also comments made by matz, koichi and shyouhei, in a different context. Which I guess makes sense too - less exposure may mean less problems. I am also neutral about the proposal really, don't mind either way - guess it may be for sam to reason in favour of it. :-)

Even then, though, I love introspection in general. Ruby is like a closed box initially, just like on xmas (and the xmas release), and you get the tools to poke inside and try to find out how it works! \o/

Perhaps if it may help the discussion (not that I contribute much to it), there could be a discussion for potential problems in this regard, e. g. pitfalls, problems etc... or it may remain a separate gem, and it may be evaluated how useful it may be to integrate it into ruby directly. That discussion has also happened with other code elements / gems in the past, e. g. martin duerst pointed this out a few times before. But as said, I am really neutral either way here.

#4 - 10/08/2019 07:28 PM - Eregon (Benoit Daloze)

Do you think it would be possible for this new API to not rely on whether there is bytecode/iseqs?
That way, it could be implemented on other Ruby implementations.

Is the main purpose to be able to estimate memory used by loaded Ruby code (methods)?
A count of reachable methods' internal representations (iseq, AST, etc) would be a metric that is likely easy to provide for all Ruby implementations.

#5 - 10/09/2019 10:49 AM - sam.saffron (Sam Saffron)

To be honest I think the best spot for this is RubyVM.stat

perhaps:

```
RubyVM.stat
{
   :global_method_state=>143,
   :global_constant_state=>1369,
   :class_serial=>8768,
   :imemo_ment_count,
   :imemo_iseq_count,
   :imemo_env_count,
   :imemo_tmpbuf_count,
   :imemo_ast_count,
   :imemo_ment_size,
   :imemo_iseq_size,
   :imemo_env_size,
   :imemo_tmpbuf_size,
   :imemo_ast_size
}
```

Since RubyVM.stat(:class_serial) is already supported as an efficient way to grab a single metric this interface fits nicely. It does not expand the signature surface of Ruby and is something that would be very simple to add for 2.7.

Additionally for extra bonus points:

RubyVM.stat(:total_allocated_bytes): all the bytes Ruby xmalloc and family allocated since process start
RubyVM.stat(:total_freed_bytes): all the bytes freed

This comprehensive set of changes would make introspection of "why is my Ruby size XYZ?" really easy and provide some extremely powerful metrics for graphing.

Thoughts?

#6 - 10/09/2019 11:00 AM - methodmissing (Lourens Naudé)

I like this API more, however RubyVM has been under discussion in https://bugs.ruby-lang.org/issues/15752 regarding implementation specific exposure of experimental API and / or insights. These 2 issues are in a way strongly coupled.

sam.saffron (Sam Saffron) wrote:

> To be honest I think the best spot for this is RubyVM.stat
>
> perhaps:
>
> ```
> RubyVM.stat
> {
>    :global_method_state=>143,
>    :global_constant_state=>1369,
>    :class_serial=>8768,
>    :imemo_ment_count,
>    :imemo_iseq_count,
>    :imemo_env_count,
>    :imemo_tmpbuf_count,
>    :imemo_ast_count,
>    :imemo_ment_size,
>    :imemo_iseq_size,
>    :imemo_env_size,
>    :imemo_tmpbuf_size,
>    :imemo_ast_size
> }
> ```
>
> Since RubyVM.stat(:class_serial) is already supported as an efficient way to grab a single metric this interface fits nicely. It does not expand the signature surface of Ruby and is something that would be very simple to add for 2.7.
>
> Additionally for extra bonus points:
>
> RubyVM.stat(:total_allocated_bytes): all the bytes Ruby xmalloc and family allocated since process start
> RubyVM.stat(:total_freed_bytes): all the bytes freed
>
> This comprehensive set of changes would make introspection of "why is my Ruby size XYZ?" really easy and provide some extremely powerful metrics for graphing.
>
> Thoughts?

#7 - 10/16/2019 05:52 AM - ko1 (Koichi Sasada)

Already we have:

```
require 'objspace'
pp ObjectSpace.count_imemo_objects
#=>
{:imemo_env=>42,
 :imemo_cref=>177,
 :imemo_ment=>3662,
 :imemo_iseq=>1194,
 :imemo_tmpbuf=>117,
 :imemo_ast=>22,
 :imemo_svar=>40,
 :imemo_throw_data=>55,
 :imemo_ifunc=>35,
 :imemo_memo=>32,
 :imemo_parser_strterm=>118}
```

There is no size version. do you want to introduce it?

#8 - 10/17/2019 09:16 PM - sam.saffron (Sam Saffron)

Yes!

ObjectSpace.memsize_of_imemo_objects sounds perfect to me.

I also support adding ObjectSpace.each_iseq which seems the simplest way to get iteration working.

I get the concern about not wanting to pollute MRI with MRI specific logic directly in ObjectSpace, but given we already have ObjectSpace.count_imemo_objects it feels a bit too late for this case.