Ruby master - Feature #10038

Extend ObjectSpace.dump to expose buffer addresses for String and Array

07/15/2014 07:15 AM - ko1 (Koichi Sasada)

Status:	Assigned
Priority:	Normal
Assignee:	tmm1 (Aman Gupta)
Target version:	

Description

ObjectSpace.dump() expose internal information in JSON.
How about to expose buffer addresses for String and Array?

```
Index: ext/objspace/objspace_dump.c
===================================================================
--- ext/objspace/objspace_dump.c    (revision 46821)
+++ ext/objspace/objspace_dump.c    (working copy)
@@ -178,12 +178,16 @@ dump_object(VALUE obj, struct dump_confi
        dump_append(dc, ", \"broken\":true");
    if (FL_TEST(obj, RSTRING_FSTR))
        dump_append(dc, ", \"fstring\":true");
-   if (STR_SHARED_P(obj))
+
+   if (STR_SHARED_P(obj)) {
        dump_append(dc, ", \"shared\":true");
+   }
    else {
        dump_append(dc, ", \"bytesize\":%ld", RSTRING_LEN(obj));
-       if (!STR_EMBED_P(obj) && !STR_SHARED_P(obj) && (long)rb_str_capacity(obj) != RSTRING_LEN(o
bj))
+       if (!STR_EMBED_P(obj) && !STR_SHARED_P(obj) && (long)rb_str_capacity(obj) != RSTRING_LEN(o
bj)) {
        dump_append(dc, ", \"capacity\":%ld", rb_str_capacity(obj));
+       dump_append(dc, ", \"ptr\":\"%p\"", RSTRING_PTR(obj));
+       }

        if (is_ascii_string(obj)) {
        dump_append(dc, ", \"value\":");
@@ -205,8 +209,14 @@ dump_object(VALUE obj, struct dump_confi
    dump_append(dc, ", \"length\":%ld", RARRAY_LEN(obj));
    if (RARRAY_LEN(obj) > 0 && FL_TEST(obj, ELTS_SHARED))
        dump_append(dc, ", \"shared\":true");
-   if (RARRAY_LEN(obj) > 0 && FL_TEST(obj, RARRAY_EMBED_FLAG))
+   if (RARRAY_LEN(obj) > 0) {
+       if (FL_TEST(obj, RARRAY_EMBED_FLAG)) {
        dump_append(dc, ", \"embedded\":true");
+       }
+       else {
+       dump_append(dc, ", \"ptr\":\"%p\"", RARRAY_PTR(obj));
+       }
+   }
    break;

      case T_CLASS:
```

With this hack, we can know the real memory address of them and cooperate with other native tools.

BTW, ObjectSpace.dump() should support T_SYMBOL.

History

#1 - 07/15/2014 11:01 AM - shevegen (Robert A. Heiler)

+1

#2 - 07/17/2014 05:04 PM - headius (Charles Nutter)

-1 from me, mostly because exposing actual memory addresses will *further* limit what Ruby can do with object references. In the presence of RGenGC, some objects are already being relocated in memory, so those objects would either have to become "shady" or not expose their addresses. As MRI's GC improves further, this will become more and more common.

And another (perhaps minor) reason: other implementations that already have fully generational GCs won't be able to support this feature at all.

If a feature like this must be added, put it in an MRI-specific namespace like RubyVM. And I'd still recommend not adding another feature that limits Ruby's (and MRI's) evolution.

#3 - 07/17/2014 05:21 PM - normalperson (Eric Wong)

headius@headius.com wrote:

> If a feature like this must be added, put it in an MRI-specific
> namespace like RubyVM. And I'd still recommend not adding another
> feature that limits Ruby's (and MRI's) evolution.

The objspace extension is already implementation-specific and documented
as having no compatibility guarantees (even with future/past MRI
versions).

I'm only mildly in favor of it, I'm not sure I would use or need this.

#4 - 07/17/2014 06:22 PM - headius (Charles Nutter)

Eric Wong wrote:

> The objspace extension is already implementation-specific and documented
> as having no compatibility guarantees (even with future/past MRI
> versions).

I suppose that's fair. We haven't had anyone request features from 'objspace' in JRuby.

I would still prefer an MRI-specific namespace for features only MRI can implement. JRuby and Rubinius have both been a good citizens, keeping our impl-specific APIs hidden. MRI should do the same.

#5 - 07/21/2014 01:11 PM - ko1 (Koichi Sasada)

(2014/07/17 23:52), headius@headius.com wrote:

> I would still prefer an MRI-specific namespace for features only MRI can implement. JRuby and Rubinius have both been a good citizens, keeping our impl-specific APIs hidden. MRI should do the same.

I'm neutral about it. There are many MRI specific features and
behaviors. I want to wait about discussion.

(Ruby 3 can do it?)

--
// SASADA Koichi at atdot dot net

#6 - 07/25/2014 06:13 PM - enebo (Thomas Enebo)

ko1, when you say native tools, do you mean things like valgrind? Can you elaborate a bit (curious since I am doing some stuff with Java Heap dumps atm)?

I am neutral on this because it feels like an extension to help with your implementation and does not seem like 'Ruby'. I would like there to be some convention in MRI source perhaps (or even docs) indicating that something is an impl-specific method or API. I don't mind that each impl has extra useful stuff, but I would like a standard for indicating that it is impl-specific.

#7 - 07/25/2014 06:30 PM - enebo (Thomas Enebo)

Oh! I should also note I know dump_all is marked as experimental and implementation-specific in the documents so I am not saying you are not trying to mark things. I am just hoping we all can agree on a standard for this.

Also, I am interested in supporting dump_all in JRuby and it would be great if we could get to a common subset of what both impls can dump.

Being experimental and also that JRuby does not implement this atm makes this last paragraph a desire at this point.

#8 - 07/25/2014 07:46 PM - ko1 (Koichi Sasada)

Thomas Enebo wrote:

> ko1, when you say native tools, do you mean things like valgrind? Can you elaborate a bit (curious since I am doing some stuff with Java Heap dumps atm)?

I wrote a tool to detect which memory part is shared with parent and child processes like that.
(Make big Array and detect how many memory pages are shared or not, with Linux specific feature)

```
require 'objspace'
require 'json'

PAGES = 1024 * 100 #* 1024
ary = Array.new(4096/8 * PAGES){|e| nil}

class SharedState
  PAGE_SIZE = 4096

  def initialize
    @compare_process = Process.pid
  end

  def each_page ptr, size
    page = first_page = (ptr/PAGE_SIZE) * PAGE_SIZE
    last_page = ((ptr+size)/PAGE_SIZE) * PAGE_SIZE

    while page <= last_page
      yield page/PAGE_SIZE
      page += PAGE_SIZE
    end
  end

  def open_proc_files
    open("/proc/#{@compare_process}/pagemap"){|f1|
      open("/proc/#{Process.pid}/pagemap"){|f2|
        yield f1, f2
      }
    }
  end

  def get_pfn f, page_index
    f.seek(page_index * 8)
    d = f.read(8).unpack('Q').first
    d & ~(0xff << 55)
  end

  def make_shared_ary ptr, size
    ary = []
    open_proc_files{|f1, f2|
      each_page(ptr, size){|page_index|
        pfn1 = get_pfn f1, page_index
        pfn2 = get_pfn f2, page_index
        ary << (pfn1 == pfn2)
      }
    }
    p true: ary.count(true), false: ary.count(false)
    ary
  end

  def shared? obj
    info = JSON.load(ObjectSpace.dump(obj))
    case info['type']
    when 'ARRAY'
      ptr = info['ptr'].to_i(16)
      size = info['memsize']
      return unless ptr
      make_shared_ary(ptr, size)
    else
      raise 'unsupported'
    end
  end
end

ss = SharedState.new
```

```
pid = fork{
  p
  ss.shared?(ary)

  # ary[0] = true
  (PAGES/4).times{|i|
    ary[4096/8 * i * 4] = true
  }

  p
  ss.shared?(ary)
}

p Process.waitpid(pid)
```

I can make it by gem. But it is easy to modify this ext compare with making gem for me :)

> I am neutral on this because it feels like an extension to help with your implementation and does not seem like 'Ruby'. I would like there to be some convention in MRI source perhaps (or even docs) indicating that something is an impl-specific method or API. I don't mind that each impl has extra useful stuff, but I would like a standard for indicating that it is impl-specific.

I'm also neutral, it is only just idea.

I propose this idea because this feature is already "implementation specific (MRI and versions)", and for example, its provides implementation/version specific information such as WB-protected.

BTW, above tool is only a toy to know can I detect such shared pages.
And I could do. I had achieved my goal.

I will make more general tools to analyse to make ruby interepter more CoW frinedly,
Yes, like valgrind (or with valgrind).

#9 - 08/06/2014 04:10 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned

#10 - 01/05/2018 09:01 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)