Object lifetime analysis with Ruby 2.1

Koichi Sasada

<ko1@heroku.inc>
Who am I?

- Koichi Sasada a.k.a. ko1
- From Japan
- 笹田 (family name) 耕一 (given name) in Kanji character
Who am I?

• CRuby/MRI committer
  • Virtual machine (YARV) from Ruby 1.9
  • YARV development since 2004/1/1
  • Recently, improving GC performance

• Matz team at Heroku, Inc.
  • Full-time CRuby developer
  • Working in Japan

• Director of Ruby Association
• Foundation to encourage Ruby dev. and communities
  • Chairman is Matz, Located at Matsue-city, Shimane, Japan

• Activities
  • Ruby programmer certification program
  • Maintenance of Ruby (Cruby) interpreter
    • Now, it is for Ruby 1.9.3
    • Ruby 2.0.0 in the future?
  • Events, especially RubyWorld Conference
  • Ruby Prize
  • Grant project. We have selected 3 proposals in 2013
    • Win32Utils Support, Conductor, Smalruby - smalruby-editor
    • We will make this grant 2014!!
  • Donation for Ruby developments and communities
• Heroku, Inc.  http://www.heroku.com

You should know about Heroku!!
This presentation is supported by

[Heroku logo]
• Heroku, Inc.  [http://www.heroku.com](http://www.heroku.com)

• Heroku supports OSSs / Ruby development
  • Many talents for Ruby, and also other languages
  • Heroku employs 3 Ruby interpreter core developers
    • Matz
    • Nobu
    • Ko1 (me)

• We name our group “Matz team”
Matz team in Heroku in Japan

Object lifetime analysis with Ruby 2.1, RubyConf.tw 2014, K.Sasada from Heroku, Inc.
Mission of Matz team

• **Improve quality of next version of CRuby**
  • Matz decides a spec finally
  • Nobu fixed huge number of bugs
  • Ko1 improves the performance

Current target is Ruby 2.2!!
Now, Ruby 2.1 is old version for us.
Ruby 2.1
Current stable

http://www.flickr.com/photos/loginesta/5266114104
Ruby 2.1

• **Ruby 2.1.0** was released at **2013/12/25**
  • New features
  • Performance improvements

• **Ruby 2.1.1** was released at **2014/02/24**
  • Includes many bug fixes found after 2.1.0 release
  • Introduce a new GC tuning parameter to change generational GC behavior (introduce it later)
Ruby 2.1 the biggest change

Version policy

• Change the versioning policy
  • Drop “patch level” in the version
  • Teeny represents patch level
    • Release new teeny versions about every 3 month
    • Teeny upgrades keep compatibility
  • Minor upgrades can break backward compatibility
    • We make an effort to keep compatibility
      (recently. Remember Ruby 1.9 😊)
Ruby 2.1 New syntax

• New syntaxes
  • Required keyword parameter
  • Rational number literal
  • Complex number literal
  • `def` returns symbol of method name

http://www.flickr.com/photos/rooreynolds/4133549889
Ruby 2.1 Runtime new features

• String#scrub
• Process.clock_gettime
• Binding#local_variable_get/set
• Bignum now uses GMP (if available)
• Extending ObjectSpace
Performance improvements

• Optimize “string literal”.freeze
• Sophisticated inline method cache
• Introducing Generational GC: RGenGC
RGenGC
Performance evaluation (RDoc)

About x15 speedup!

* Disabled lazy sweep to measure correctly.

Object lifetime analysis with Ruby 2.1, RubyConf.tw 2014, K.Sasada from Heroku, Inc.
RGenGC
Performance evaluation (RDoc)

* 12% improvements compare with w/ and w/o RGenGC
* Disabled lazy sweep to measure correctly.
Ruby 2.2
Next version

http://www.flickr.com/photos/adafruit/8483990604
Schedule of Ruby 2.2

• Not published officially
• Schedule draft is available by Naruse-san
Ruby 2.2 schedule

We are here!

2013/12
Ruby 2.1.0

2014/12/25
Ruby 2.2.0

Rubyconf.PH
3/28, 29

RDRC
6/26, 27

Rubyconf.tw
4/25, 26

RubyKaigi
9/18, 19, 20

RubyConfi
11/17, 18, 19

Events are important for
EDD (Event Driven Development) Developers

Object lifetime analysis with Ruby 2.1,
RubyConf.tw 2014, K.Sasada from Heroku, Inc.
Ruby 2.2 (rough) schedule

- **2013/12**
  - Ruby 2.1.0

- **2014/12/25**
  - Ruby 2.2.0

- **Sep/2014**
  - Preview 1
  - Big feature freeze

- **Nov/2014**
  - Preview 2
  - Feature freeze

- **Dec/2014**
  - Critical Bug fix only

- **2014/12/25**
  - Release candidate

We are here!
2.2 big features (planned)

• New syntax: not available now
• New method: not available now
• Internal
  • GC
    • Symbol GC (merged recently)
    • 2age promotion strategy for RGenGC
    • Incremental GC to reduce major GC pause time
• VM
  • More sophisticated method cache
Symbol GC

• Symbols remain forever → Security issue
  • “n.times{|i| i.to_s.to_sym}”
    creates “n” symbols and they are never collected
• Symbol GC: Collect dynamically created symbols
Object lifetime
Ruby’s object/memory management

• “Object.new” allocate a new object
  • “foo” (string literal) also allocate a new object
  • Everything are objects in Ruby!
• We don’t need to “de-allocate” objects manually

Why we don’t need to clean up objects?
Garbage collection
The automatic memory management

Fig. 109. — A Garbage Collector.
http://www.flickr.com/photos/cicasassy/6817999189/
Garbage collection
The automatic memory management
Garbage collection
The automatic memory management
Object lifetime

• Human’s lifetime
Object lifetime

# Object creation
obj = Object.new

# Collected by GC
# when nobody refers it

Birth --> Death
Object lifetime

Birth \rightarrow \text{GC surviving count} \rightarrow \text{Age} \rightarrow \text{Age at the dying} \rightarrow \text{Lifetime} \rightarrow \text{Death}

Object lifetime analysis with Ruby 2.1, RubyConf.tw 2014, K.Sasada from Heroku, Inc. 31
Why lifetime is important?

• Old objects slow down major (full) GC
  • RGenGC have reduced GC overhead comes from old objects
  • However, major (full) GC needs a time
• Long life (old) objects can be unexpected memory leak

Reducing old objects makes Ruby program faster 😊
Questions

• How to measure object lifetime?
• How to find old objects?
Allocation tracer gem
Allocation tracer

• Trace object allocations
  • Two features
    1. Show statistics data by line
    2. Show lifetime statistics table
  • Using newobj/freeobj internal hooks introduced from Ruby 2.1.0.
  • So that this gem only supports Ruby 2.1 and later
Allocation Tracer

(1) Show statistics data by line

- Show statistics data by each lines
- Collect following data
  - Allocation count
  - Old object count
  - Total object lifetime (average lifetime by dividing allocation count)
  - Min/Max lifetime
  - Total amount of memory usage (average memory usage by dividing allocation count)
Allocation Tracer
(1) Show statistics data by line

Demo
Allocation Tracer
(2) Object lifetime statistics

• Show object lifetime statistics table for each type

<table>
<thead>
<tr>
<th>Age</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_OBJECT</td>
<td>7534</td>
<td>435532</td>
<td>6695</td>
<td>12916</td>
<td>2841</td>
<td>2298</td>
<td>2570</td>
<td>4222</td>
<td>3256</td>
<td>2659</td>
<td>4839</td>
</tr>
<tr>
<td>T_CLASS</td>
<td>0</td>
<td>90</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>T_STRING</td>
<td>116307</td>
<td>17706739</td>
<td>24244</td>
<td>97981</td>
<td>9461</td>
<td>5470</td>
<td>4825</td>
<td>7243</td>
<td>6533</td>
<td>5581</td>
<td>15826</td>
</tr>
<tr>
<td>T_REGEXP</td>
<td>0</td>
<td>41192</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>T_ARRAY</td>
<td>32874</td>
<td>4490228</td>
<td>5878</td>
<td>9262</td>
<td>4863</td>
<td>2567</td>
<td>2485</td>
<td>5697</td>
<td>3752</td>
<td>3033</td>
<td>5672</td>
</tr>
<tr>
<td>T_HASH</td>
<td>872</td>
<td>89158</td>
<td>113</td>
<td>323</td>
<td>90</td>
<td>85</td>
<td>93</td>
<td>138</td>
<td>161</td>
<td>101</td>
<td>183</td>
</tr>
<tr>
<td>T_STRUCT</td>
<td>1</td>
<td>8507</td>
<td>247</td>
<td>197</td>
<td>62</td>
<td>55</td>
<td>59</td>
<td>9</td>
<td>42</td>
<td>67</td>
<td>127</td>
</tr>
<tr>
<td>T_FILE</td>
<td>1</td>
<td>9330</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T_DATA</td>
<td>8510</td>
<td>578913</td>
<td>146</td>
<td>69</td>
<td>96</td>
<td>53</td>
<td>77</td>
<td>54</td>
<td>65</td>
<td>90</td>
<td>62</td>
</tr>
<tr>
<td>T.Match</td>
<td>7</td>
<td>829992</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T_RATION</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T_NODE</td>
<td>9333</td>
<td>649080</td>
<td>19438</td>
<td>4948</td>
<td>2665</td>
<td>7</td>
<td>17</td>
<td>5</td>
<td>9</td>
<td>26</td>
<td>135</td>
</tr>
<tr>
<td>T_ICLASS</td>
<td>0</td>
<td>90</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T_MODUL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T_FLOAT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Allocation Tracer
(2) Object lifetime statistics

• Show object lifetime table for each type
Allocation Tracer
(2) Object lifetime statistics

• Show object lifetime table for each type

Age/percentage

Object lifetime analysis with Ruby 2.1, RubyConf.tw 2014, K.Sasada from Heroku, Inc.
Allocation Tracer
(2) Object lifetime statistics

Most of allocated objects are **String**
Allocation Tracer demo
(2) Object lifetime statistics

Most of String die at first GC
Allocation Tracer example

• Using “by line” statistics on RDoc program
## Allocation Tracer example

<table>
<thead>
<tr>
<th>path</th>
<th>line</th>
<th>count</th>
<th>total_memsize</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/rdoc/markup/parser.rb</td>
<td>327</td>
<td>128,097</td>
<td>498,086,366</td>
</tr>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/rdoc/parser/c.rb</td>
<td>644</td>
<td>3,132</td>
<td>111,437,037</td>
</tr>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/time.rb</td>
<td>325</td>
<td>1,146,900</td>
<td>85,714,539</td>
</tr>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/rdoc/token_stream.rb</td>
<td>59</td>
<td>1,267,168</td>
<td>81,137,600</td>
</tr>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/rdoc/store.rb</td>
<td>892</td>
<td>23,038</td>
<td>65,504,272</td>
</tr>
</tbody>
</table>
@input.byteslice(0, byte_offset).length

• @input is String
• @input.byteslice(0, byte_offset) makes new String
• Return only length of sliced string
  → Created string object is completely temporary
Introduce String#byteslice_length

@input.byteslice_length(0, byte_offset)

• No temporary String object
After introducing new method...

<table>
<thead>
<tr>
<th>path</th>
<th>line</th>
<th>count</th>
<th>total_memsize</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/rdoc/parser/c.rb</td>
<td>644</td>
<td>4,369</td>
<td>93,019,334</td>
</tr>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/rdoc/token_stream.rb</td>
<td>59</td>
<td>1,386,966</td>
<td>88,834,304</td>
</tr>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/time.rb</td>
<td>325</td>
<td>1,137,612</td>
<td>84,959,225</td>
</tr>
<tr>
<td>/home/ko1/src/ruby/trunk/lib/rdoc/ruby_lex.rb</td>
<td>162</td>
<td>944,833</td>
<td>60,724,300</td>
</tr>
</tbody>
</table>

And total consuming memory reduced from about 140MB to 120MB
Another GC tip

GC Tracer
gc_tracer gem

• Helper gem to analyze GC behavior for tuning

http://www.flickr.com/photos/nasa_goddard/5188180370
How to use tuning parameters?

1. Profile your application
2. Try GC parameters (environment variables)
Profile memory management
GC.stat (MRI specific)

• “GC.stat” returns statistics information about GC
  • Counts
    • :count=>2, # GC count
    • :minor_gc_count=>2, # minor GC count
    • :major_gc_count=>0, # major GC count
  • Current slot information
    • :heap_live_slot=>6836, #=> # of live objects
    • :heap_free_slot=>519, #=> # of freed objects
    • :heap_final_slot=>0, #=> # of waiting finalizer objects
    • total_slots = heap_live_slot + heap_free_slot + heap_final_slot
  • Statistics
    • :total_allocated_object=>7674, # total allocated objects
    • :total_freed_object=>838, # total freed objects
    • Current living objects = total_allocated_object - total_freed_object
Profile memory management

GC.latest_gc_info (MRI specific)

• “GC.latest_gc_info” returns details of latest GC
  • :gc_by=>:newobj  # why GC invoked?
    • newobj: no slots available
    • malloc: malloc_increase > malloc_limit
  • :major_by=>nil  # why major GC invoked?
  • :have_finalizer=>false  # have finalizer?
  • :immediate_sweep=>false  # immediate sweep?
Profile memory management “gc_tracer” gem (MRI 2.1.0 later!!)

• GC::Tracer.start_logging(filename)
  • Save all GC.stat/GC.latest_gc_info results at every GC events into specified file
• GC events:
  • Start
  • End marking
  • End sweeping
Profile memory management
“gc_tracer” gem

• Run your application with gc_tracer
• Plot with Excel!

http://www.flickr.com/photos/microsoftsweden/5394685465
Profile memory management
“gc_tracer” gem

Object lifetime analysis with Ruby 2.1,
RubyConf.tw 2014, K.Sasada from Heroku, Inc.
Profile memory management
“gc_tracer” gem

Object lifetime analysis with Ruby 2.1,
RubyConf.tw 2014, K.Sasada from Heroku, Inc.
Profile memory management
“gc_tracer” gem

Object lifetime analysis with Ruby 2.1,
RubyConf.tw 2014, K.Sasada from Heroku, Inc.
Profile memory management “gc_tracer” gem

ruby 2.2 dev/RUBY_GC_HEAP_OLDOBJECT_FACTOR=2.0 (default)

Object lifetime analysis with Ruby 2.1,
RubyConf.tw 2014, K.Sasada from Heroku, Inc.
Profile memory management
“gc_tracer” gem

Ruby 2.2dev w/ RUBY_GC_HEAP_OLDOBJECT_FACTOR=1.3

Object lifetime analysis with Ruby 2.1,
RubyConf.tw 2014, K.Sasada from Heroku, Inc.
For more information

Please refer my slide at RubyConf PH 2014
Summary of this talk

• Ruby 2.1 was released and 2.2 will be released this year
• Object lifetime analysis is important for program tuning
• Allocation Tracer gem helps how applications allocate objects
• Another GC Tips
  • GC Tracer gem helps GC behavior analysis for GC tuning
Thank you for your attention
Q&A?

Koichi Sasada
<ko1@heroku.com>