Ruby 2.4 Internals

Koichi Sasada
ko1@cookpad.com
Ruby 2.4.0 Released
Posted by naruse on 25 Dec 2016

We are pleased to announce the release of Ruby 2.4.0.

Ruby 2.4.0 is the first stable release of the Ruby 2.4 series. It introduces many new features, for example:

- **Introduce hash table improvement (by Vladimir Makarov)**
  Improve the internal structure of hash table (st_table) by introducing open addressing and an inclusion order array. This improvement has been discussed with many people, especially with Yura Sokolov.

- **Binding#irb: Start a REPL session similar to binding.pry**
  While you are debugging, you may often use `p` to see the value of variables. With ` pry you can use `binding.pry` in your application to launch a REPL and run any Ruby code. r56624 introduces `binding.irb` which behaves like that with irb.
Note:
Ruby 2.4.0 has several bugs.

Note2:
No x.y.0 doesn’t have several bugs.
Ruby 2.4.0 released

Ruby 2.4.0 is the first stable release of the Ruby 2.4 series. It introduces many new features, for example:

**Introduce hash table improvement (by Vladimir Makarov)**

Improve the internal structure of hash table (st_table) by introducing open addressing and an inclusion order array. This improvement has been discussed with many people, especially with Yura Sokolov.

**Binding#irb: Start a REPL session similar to binding.pry**

While you are debugging, you may often use `pp` to see the value of variables. With `pry` you can use `binding.pry` in your application to launch a REPL and run any Ruby code. r56624 introduces `binding.irb` which behaves like that with `irb`.
New features written in a release announcement

• Introduce hash table improvement (by Vladimir Makarov)
• Binding#irb: Start a REPL session similar to binding.pry
• Unify Fixnum and Bignum into Integer
• String supports Unicode case mappings
• Performance improvements
  • Array#max, Array#min
  • Regexp#match?
  • speed up instance variable access
• Debugging
  • Thread#report_on_exception and Thread.report_on_exception
  • Thread deadlock detection now shows threads with their backtrace and dependency
• Other notable changes since 2.3
  • Support OpenSSL 1.1.0 (drop support for 0.9.7 or prior)
  • ext/tk is now removed from stdlib Feature #8539
  • XMLRPC is now removed from stdlib Feature #12160
Ruby 2.4 Released: Faster Hashes, Unified Integers and Better Rounding

December 25, 2016 by Jonan Scheffler

The Ruby maintainers continued their annual tradition by gifting us a new Ruby version to celebrate the holiday: Ruby 2.4 is now available and you can try it out on Heroku.

Ruby 2.4 brings some impressive new features and performance improvements to the table, here are a few of the big ones:

Continue reading »
Any other topics?
People should really upgrade to Ruby 2.4. We're seeing GC time cut in half for the same throughput of allocations
@schneems likely #10212 in redmine. But could be hash restructuring. Two possibilities. (not sure if I care though! ;))
Bug #10212

MRI is not for lambda calculus

ko1 (Koichi Sasada) が2年以上前に追加。約1ヶ月前に更新。

ステータス: Closed
優先度: Normal
担当者: ko1 (Koichi Sasada)
対象バージョン: -
ruby -v:
  ruby 2.2.0dev (2014-08-21)
  trunk 47241) [x86_64-linux]
[ruby-core:64838]
benchmark/bm_app_lc_fizzbuzz.rb
アンダースタンディング コンピュテーション
——単純な機械から不可能なプログラムまで

Tom Stuart 著、笹田 耕一 監訳、笹井 崇司 訳
2014年09月 発行
336ページ
ISBN978-4-87311-697-6
フォーマット Print PDF
原書: Understanding Computation

オライリー・ジャパンで書籍を購入:定価3,456円
Ebook Storeで電子版を購入:価格2,765円
Bug #10212 MRI is not for lambda calculus

JRuby 26 sec

mruby 27 sec

MRI 114 sec
Feature #12628
change block/env structs
Ruby 2.4 Internals
Change block/env structs

Koichi Sasada
ko1@cookpad.com
Issues

1. we need to clear `rb_control_frame_t::block_iseq` for every frame setup. It consumes space (a VALUE for each frame) and initializing time.

2. There are several block passing ways by `ISeq(iter{...})`, `Proc(iter(&pr))`, `Symbol(iter(:sym))`. However, they are not optimized (for Symbol blocks, there is only ad-hoc check code).

3. Env (and Proc, Binding) objects are not WB-protected ([Bug #10212]).
Patch

• [https://github.com/ruby/ruby/compare/trunk...ko1:block_code](https://github.com/ruby/ruby/compare/trunk...ko1:block_code)

• “Showing with 1,863 additions and 1,070 deletions.”
Approaches

• For (1), (2)
  • Introduce Block Handler (BH)
  • Using BH

• For (3)
  • Introduce Write Barriers (WB) for Env objects
WB protected or unprotected?

- Ruby 2.1.0 introduced Generational GC
  - Only newer objects
  - GenGC requires “Write barriers” (WB), but MRI allows WB unprotected objects
    (See my past presentations for details)

- WB protected objects: GenGC → Fast
- WB unprotected objects: Not GenGC → Slow
RubyVM::Env objects

- Env objects represent captured local variables
- Each Proc or Binding has at least one Env object
- Proc object "$pr" consists of 3 Env objects

```
a = 1
1.times{|b|
  1.times{|c|
    $pr = Proc.new{
      # you can access a, b, c
    }
  }
}
```

Diagram:
- Proc $pr
- Env c=0
- Env b=0
- Env a=1
RubyVM::Env objects were WB-unprotected

• They were WB unprotected because:
  • Difficulty of implementation
  • Performance issue
Performance issue
Assignment performance

• Ruby 2.3 or before
  *(ep - idx) = val;

• Naïve implementation

```c
#define VM_EP_IN_HEAP_P(th, ep) ¥
 !((th)->stack <= (ep) && ¥
 (ep) < ((th)->stack + (th)->stack_size)))

if (VM_EP_IN_HEAP_P(ep)) {
    RB_OBJ_WRITE(VM_ENV_EP_ENVVAL(ep),
                ep-idx, val);
}
else *(ep - idx) = val;
```
Ideas

1. Lightweight escape detection
2. Skip WB except really required timing
Idea
Lightweight escape detection

• Move `cfp->flags` to `ep[0]`

• Introduce a `VM_ENV_FLAG_ESCAPE` flag to represent escaped Env.

• // Before

```c
#define VM_EP_IN_HEAP_P(th, ep)     (!((th)->stack <= (ep) && (ep) < ((th)->stack + (th)->stack_size)))
```

• // After

```c
#define VM_EP_IN_HEAP_P(ep)        ((ep[0] & VM_ENV_FLAG_ESCAPE)
```
Idea
Skip WB except really required timing

1. At initializing Env objects, VM_ENV_FLAG_WB_REQUIRED is true.
2. At first local variable assignment, VM_ENV_FLAG_WB_REQUIRED is true, we remember this Env object forcibly. **And turn off this flag.**
3. At next local variable assignment, VM_ENV_FLAG_WB_REQUIRED is false, so we can ignore WB protection.
4. **At GC marking for this Env object, we turn on VM_ENV_FLAG_WB_REQUIRED** and goto (2).

Very danger technique because it depends on GC implementation
Naïve code

#define VM_EP_IN_HEAP_P(th, ep)   (!((th)->stack <= (ep) && (ep) < (th)->stack + (th)->stack_size))

vm_env_write(const VALUE *ep, int index, VALUE v) {
    if (VM_EP_IN_HEAP_P(ep)) {
        RB_OBJ_WRITE(VM_ENV_EP_ENVVAL(ep), ep-idx, val);
    }
    else {
        *(ep - idx) = val;
    }
}
Final code

vm_env_write(const VALUE *ep, int index, VALUE v) {
    VALUE flags = ep[VM_ENV_DATA_INDEX_FLAGS];
    if (LIKELY((flags & VM_ENV_FLAG_WB_REQUIRED) == 0)) {
        *(ep - idx) = val; /* mostly used */
    } else {
        /* remember env value forcibly */
        vm_env_write_slowpath(ep, index, v);
    }
}
Benchmark result

<table>
<thead>
<tr>
<th></th>
<th>trunk</th>
<th>modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>app_lc_fizzbuzz</td>
<td>58.277</td>
<td>41.729 (sec)</td>
</tr>
<tr>
<td>vm1_simplereturn*</td>
<td>0.660</td>
<td>0.638 (sec)</td>
</tr>
<tr>
<td>vm1_yield*</td>
<td>0.738</td>
<td>0.650 (sec)</td>
</tr>
</tbody>
</table>
Bug #10212 MRI is not for lambda calculus
lc_fizzbuzz with MRI versions
Bug #10212 MRI is not for lambda calculus

lc_fizzbuzz with MRI, JRuby, mruby

<table>
<thead>
<tr>
<th></th>
<th>Execution time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>trunk</td>
<td>36.976</td>
</tr>
<tr>
<td>jruby</td>
<td>18.706</td>
</tr>
<tr>
<td>mruby</td>
<td>40.185</td>
</tr>
</tbody>
</table>
Summary

• Ruby 2.4.0 has many improvements
• Now Proc (Env) objects are WB protected and we have more faster GC (marking)
• My ideas allow to protect Env objects without big performance impact
Thank you for your attention

Koichi Sasada
<ko1@cookpad.com>