Fast Code for Ruby

@284km

はいさい！

今日話すこと

- Ruby で速いコードを書く
- benchmark_driver がすごく良い
- Ruby の楽しみ方をひとつ

速いコードを書く

速いコードを書く

- 測定
  - ベンチマーカー

ベンチマーカー

benchmark.rb (標準添付ライブラリ)

evanphx/benchmark-ips

k0kubun/benchmark_driver

速いコード

- 同じ結果が得られる複数の書き方
- 計測して、速いほうが速い
- 前提: 今回のベンチマークは僕の MBP で実行したものです

benchmark_driver

e.g.

```ruby
require 'benchmark_driver'

Benchmark.driver do |x|
  x.rbenv '2.4.3', '2.5.0'
  x.prelude %{
    class Okinawa
      def naha; end
      def method_missing(_method,*args); naha; end
    end

    def fastest
      o = Okinawa.new; o.naha
    end

    def slow
      o = Okinawa.new; o.send(:naha)
    end

    def slowest
      o = Okinawa.new; o.sapporo
    end
  }

  x.report "call", %{ fastest }
  x.report "send", %{ slow }
  x.report "method_missing", %{ slowest }
end
```

- 複数の処理系
- 出力形式 (.md 可)

```
Warming up --------------------------------------
                call     5.463M i/s
                send     4.445M i/s
      method_missing     3.035M i/s
Calculating -------------------------------------
                           2.4.3        2.5.0
                call       4.866M       6.521M i/s -     16.389M times in 3.368235s 2.513330s
                send       3.963M       4.157M i/s -     13.334M times in 3.364348s 3.207583s
      method_missing       3.989M       3.074M i/s -      9.106M times in 2.282851s 2.962082s

Comparison:
                                call
                2.5.0:   6520699.2 i/s
                2.4.3:   4865654.9 i/s - 1.34x  slower

                                send
                2.5.0:   4156965.9 i/s
                2.4.3:   3963268.1 i/s - 1.05x  slower

                    method_missing
                2.4.3:   3988784.2 i/s
                2.5.0:   3074121.5 i/s - 1.30x  slower
```

Hash#keys.each
Hash#each_key

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    MYSELF = {
      'id' => '@284km',
      'info' => {
        'language' => 'japanese / ruby',
        'from' => 'Tokyo',
        'name' => 'Kazuma Furuhashi',
        'nickname' => '秒速さん / byousokusan',
        'work' => 'https://www.feedforce.jp/',
        'community' => 'asakusarb, shibuyarb, omotesandorb, shinjukurb',
        'organize' => 'railsdm 2018 3/24, 3/25'
      },
    }
    def slow
      MYSELF.keys.each(&:to_sym)
    end

    def fast
      MYSELF.each_key(&:to_sym)
    end
  }
  x.report 'Hash#keys.each', %{ slow }
  x.report 'Hash#each_key',  %{ fast }
end
```

Hash#keys.each
Hash#each_key

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    MYSELF = {
      'id' => '@284km',
      'info' => {
        'language' => 'japanese / ruby',
        'from' => 'Tokyo',
        'name' => 'Kazuma Furuhashi',
        'nickname' => '秒速さん / byousokusan',
        'work' => 'https://www.feedforce.jp/',
        'community' => 'asakusarb, shibuyarb, omotesandorb, shinjukurb',
        'organize' => 'railsdm 2018 3/24, 3/25'
      }
    }
    de
      
    en

    de
      
    en
  }
  x.re
  x.re
end
```

```
Warming up --------------------------------------
      Hash#keys.each   754.914k i/s
       Hash#each_key     1.257M i/s
Calculating -------------------------------------
      Hash#keys.each     1.020M i/s -      2.265M times in 2.220187s (980.33ns/i)
       Hash#each_key     1.511M i/s -      3.772M times in 2.495286s (661.60ns/i)

Comparison:
       Hash#each_key:   1511492.9 i/s
      Hash#keys.each:   1020067.7 i/s - 1.48x  slower
```

速いコード

- 条件に依存する場合がある
- 必ずその書き方が良いとは言えないケースがある
- (e.g. gsub / tr)

gsub
tr

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    WORD = 'I-wanted-to-eat-syuri-soba-but-closed-at-14'
    def slow
      WORD.gsub('-', ' ')
    end

    def fast
      WORD.tr('-', ' ')
    end
  }
  x.report 'String#gsub', %{ slow }
  x.report 'String#tr', %{ fast }
end
```

gsub
tr

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    WORD = 'I-wanted-to-eat-syuri-soba-but-closed-at-14'
    def slow
      WORD.gsub('-', ' ')
    end

    def fast
      WORD.tr('-', ' ')
    end
  }
  x.repo
  x.repo
end
```

```
Comparison:
            String#tr:   662593.9 i/s
          String#gsub:   156502.2 i/s - 4.23x  slower
```

gsub
tr

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.report 'WORD = 'wa----------i'
    WORD = 'I-wanted-to-eat-syuri-soba-but-closed-at-14'
    def slow
      WORD.gsub('-', ' ')
    end

    def fast
      WORD.tr('-', ' ')
    end
  }
  x.repo
  x.repo
end
```

```
Comparison:
            String#tr:   662593.9 i/s
          String#gsub:   156502.2 i/s - 4.23x  slower
```

gsub
tr

```
require 'benchmark_driver'
Benchmark.driver do |x|
  x.
    WORD = 'wa----------i'
    WORD = 'I-wanted-to-eat-syuri-soba-but-closed-at-14'
    def slow
      WORD.gsub('-', ' ')
    end

    def
      W
    end
  }
  x.repo
  x.repo
end
```

```
Comparison:
            String#tr:   1557474.2 i/s
          String#gsub:    165049.1 i/s - 9.44x  slower
```

```
Comparison:
            String#tr:    662593.9 i/s
          String#gsub:    156502.2 i/s - 4.23x  slower
```

gsub
tr

```
require 'benchmark_driver'
Ben
  x
    WORD = 'I-wanted-to-eat-syuri-soba-but-closed-at-14'
    def slow
      WORD.gsub('-', ' ')
    end

    def
      WO
    end
  }
  x.repo
  x.repo
end
```

```
WORD = 'sugoku-nagai-length-thousands-sssssssssssssssssssssssssssssssssssssssssssssss'
```

```
WORD = 'wa----------------------------------------i'
```

```
Comparison:
          String#tr:   1557474.2 i/s
        String#gsub:    165049.1 i/s - 9.44x  slower
```

```
Comparison:
          String#tr:    662593.9 i/s
        String#gsub:    156502.2 i/s - 4.23x  slower
```

gsub
tr

```
require 'benchmark_driver'
Ben
  x
  WORD = 'sugoku-nagai-length-thousands-sssssssssssssssssssssssssssssssssssssssssssssss'
  WORD = 'wa----------------------------------------------------i'
  WORD = 'I wanted to eat syuni soba but closed at 14'
  def
    WO
  end

  def
    WO
  end
end
x.repo
x.repo
end
```

```
Comparison:
          String#gsub:   208960.5 i/s
           String#tr:    67908.6 i/s - 3.08x  slower
```

```
Comparison:
           String#tr:  1557474.2 i/s
          String#gsub:   165049.1 i/s - 9.44x  slower
```

```
Comparison:
           String#tr:   662593.9 i/s
          String#gsub:   156502.2 i/s - 4.23x  slower
```

速いコード

- どういう時に有効かを知るとよさそう
- 実装を覗くと分かることがある

Bang method

- 破壊的 / 非破壊的 なメソッドの例
- この実装は見やすいと思います

Hash#merge
Hash#merge!

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    ENUM = (1..100)
    def slow
      ENUM.inject({}) do |h, e|
        h.merge(e => e)
      end
    end

    def fast
      ENUM.inject({}) do |h, e|
        h.merge!(e => e)
      end
    end
  }
  x.report 'Hash#merge', %{ slow }
  x.report 'Hash#merge!', %{ fast }
end
```

💎 ここではサンプルコードはそれほど重要ではなく、

Hash#merge
Hash#merge!

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    ENUM = (1..100)
    def slow
      ENUM.inject({}) do |h, e|
        h.merge(e => e)
      end
    end

    def fast
      ENUM.inject({}) do |h, e|
        h.merge!(e => e)
      end
    end
  }
  x.r
  x.r
end
```

```
Comparison:
          Hash#merge!:    14275.9 i/s
           Hash#merge:     5251.8 i/s - 2.72x  slower
```

💎 ここではサンプルコードはそれほど重要ではなく、

💎 速度差の理由を追ってみる

Hash#merge
Hash#merge!

```c
static VALUE
rb_hash_merge(VALUE hash1, VALUE hash2)
{
    return rb_hash_update(rb_hash_dup(hash1), hash2);
}
```

```c
static VALUE
rb_hash_update(VALUE hash1, VALUE hash2)
{
    rb_hash_modify(hash1);
    hash2 = to_hash(hash2);
    if (rb_block_given_p()) {
        rb_hash_foreach(hash2, rb_hash_update_block_i, hash1);
    }
    else {
        rb_hash_foreach(hash2, rb_hash_update_i, hash1);
    }
    return hash1;
}
```

dup したものを渡して結果を返しているのでその分遅い

速さ

と、読みやすさ

each_with_index
while

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    ARRAY = [*1..100]
    def fast_
      index = 0
      while index < ARRAY.size
        ARRAY[index] + index
        index += 1
      end
      ARRAY
    end

    def slow
      ARRAY.each_with_index do |number, index|
        number + index
      end
    end
  }
  x.report "While Loop", %{ fast }
  x.report "each_with_index", %{ slow }
end
```

each_with_index
while

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    ARRAY = [*1..100]
    def fast_
      index = 0
      while index < ARRAY.size
        ARRAY[index] + index
```

```
Comparison:
          While Loop:   191258.8 i/s
    each_with_index:   102315.0 i/s - 1.87x  slower
```

```ruby
      ARRAY.each_with_index do |number, index|
        number + index
      end
    end
  }
  x.report "While Loop", %{ fast }
  x.report "each_with_index", %{ slow }
end
```

読みやすさ大事

- 読みやすさとは...(後で話題にします
が yield / Proc#call もそうですよね)

- 仕事で書くコードでは、チームの事情などによって何が読みやすいかは変わりそうではある（僕の感想

Rubocop::Cop::Performance

- パフォーマンスが改善されそうなところを指摘してくれる
- 静的解析ツールに頼ることも有効
- この間ひとつ指摘されました

String#match
String#match?

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    # Because the methods avoid creating a `MatchData` object or saving backref.
    # So, when `MatchData` is not used, use `match?` instead of `match`.
    def fast
      "aaacolorzzz".match?(/color/)
    end

    def slow
      "aaacolorzzz".match(/color/)
    end

    def slow2
      "aaacolorzzz" =~ /color/
    end
  }
  x.report('match?', %{ fast })
  x.report('match', %{ slow })
  x.report('=~', %{ slow2 })
end
```

String#match
String#match?

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    # Because the methods avoid creating a `MatchData` object or saving backref.
    # So, when `MatchData` is not used, use `match?` instead of `match`.
    def fast
      "aaacolorzzz".match?(/color/)
    end

    def slow
      "aaacolorzzz".match(/color/)
    end

    d
      
    e
  }
  x.r
  x.r
  x.report('=~', %{ slow2 })
end
```

```
Comparison:
            match?:  6628814.6 i/s
                =~:   912646.2 i/s - 7.26x  slower
             match:   812622.0 i/s - 8.16x  slower
```

String#match
String#match?

```c
static VALUE
rb_str_match_m(int argc, VALUE *argv, VALUE str)
{
    VALUE re, result;
    if (argc < 1)
	rb_check_arity(argc, 1, 2);
    re = argv[0];
    argv[0] = str;
    result = rb_funcallv(get_pat(re), rb_intern("match"), argc, argv);
    if (!NIL_P(result) && rb_block_given_p()) {
	return rb_yield(result);
    }
    return result;
}
```

```c
static VALUE
rb_str_match_m_p(int argc, VALUE *argv, VALUE str)
{
    VALUE re;
    rb_check_arity(argc, 1, 2);
    re = get_pat(argv[0]);
    return rb_reg_match_p(re, str, argc > 1 ? NUM2LONG(argv[1]) : 0);
}
```

String#match
String#match?

```c
static VALUE
rb_str_match_m(int argc, VALUE *argv, VALUE str)
{
    VALUE re, result;
    if (argc < 1)
	rb_check_arity(argc, 1, 2);
    re = argv[0];
    argv[0] = str;
    result = rb_funcallv(get_pat(re), rb_intern("mat
    if (!NIL_P(result) && rb_block_given_p()) {
	return rb_yield(result);
    }
    return result;
}
```

```c
static VALUE
rb_str_match_m_p(int argc, VALUE *argv, VALUE str)
{
    VALUE re;
    rb_check_arity(argc, 1, 2);
    re = get_pat(argv[0]);
    return rb_reg_match_p(re, str, argc > 1 ? NUM2LONG(argv[1]) : 0);
}
```

💎 MatchData オブジェクト
を返す（マッチした場合）

String#match
String#match?

```c
static VALUE
rb_str_match_m(int argc, VALUE *argv, VALUE str)
{
    VALUE re, result;
    if (argc < 1)
	rb_check_arity(argc, 1, 2);
    re = argv[0];
    argv[0] = str;
    result = rb_funcallv(get_pat(re), rb_intern("mat
    if (!NIL_P(result) && rb_block_given_p()) {
	return rb_yield(result);
    }
    return result;
}
```

```c
static VALUE
rb_str_match_m_p(int argc, VALUE *argv, VALUE str)
{
    VALUE re;
    rb_check_arity(argc, 1, 2);
    re = get_pat(argv[0]);
    return rb_reg_match_p(re, str, argc > 1 ? NUM2LONG(argv[1]) : 0);
}
```

- MatchData オブジェクトを返す（マッチした場合）

- rb_reg_match_p() は、マッチしたら true しなければ false を返す。

Ruby 2.5
使っていますか？

- 💎 バージョンアップと共に改善される部分は知っておくと便利

unpack1
unpack.first

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    PACKED = "ABC"
    def fast
      PACKED.unpack1("C*")
    end

    def slow
      PACKED.unpack("C*").first
    end
  }
  x.report 'unpack1', %{ fast }
  x.report 'unpack.first', %{ slow }
end
```

💎 String#unpack1 は 2.4 で入った

unpack1
unpack.first

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    PACKED = "ABC"
    def fast
      PACKED.unpack1("C*")
    end

    def slow
      PACKED.unpack("C*").first
    end
  }
  x.report 'unpack1', %{ fast }
  x.report 'unpack.first', %{ slow }
end
```

```
Comparison:
             unpack1:  7327454.1 i/s
       unpack.first:  3336299.1 i/s - 2.20x  slower
```

💎 String#unpack1 は 2.4 で入った

yield
Proc#call

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    def slow(&block)
      block.call
    end

    def slow2(&block)
      yield
    end

    def slow3(&block)
    end

    def fast
      yield
    end
  }
  x.report 'block.call', %{ slow { 1 + 1 } }
  x.report 'block + yield', %{ slow2 { 1 + 1 } }
  x.report 'block argument', %{ slow3 { 1 + 1 } }
  x.report 'yield', %{ fast { 1 + 1 } }
end
```

💎 2.5 で Proc#call が速くなった

yield
Proc#call

```ruby
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    def slow(&block)
      block.call
    end
    def slow2(&block)
      yield
    end
    def slow3(&block)
      block
    end
    def fast
      yield
    end
  }
  x.report 'block.call', %{ slow { 1 + 1 } }
  x.report 'block + yield', %{ slow2 { 1 + 1 } }
  x.report 'block argument', %{ slow3 { 1 + 1 } }
  x.report 'yield', %{ fast { 1 + 1 } }
end
```

```
Comparison:
      block argument:  20788489.4 i/s
               yield:  17384226.1 i/s - 1.20x  slower
        block + yield:  12459415.0 i/s - 1.67x  slower
           block.call:   2616398.1 i/s - 7.95x  slower
```

yield
Proc#call

```
require 'benchmark_driver'
Benchmark.driver do |x|
  x.prelude %{
    def slow(&bl
      block.call
    end

    Comparis
          bl

          b

    end
  }
  x.report 'bloc
  x.report 'bloc
  x.report 'bloc
  x.report 'yiel
end
```

```
Comparison:
                    block.call
          2.5.0:   2665729.7 i/s
          2.4.3:   2663290.9 i/s - 1.00x  slower

        block + yield
          2.5.0:  12969553.8 i/s
          2.4.3:   2959699.7 i/s - 4.38x  slower

      block argument
          2.5.0:  16662638.4 i/s
          2.4.3:   3326284.3 i/s - 5.01x  slower

                   yield
          2.5.0:  18704171.2 i/s
          2.4.3:  18062692.4 i/s - 1.04x  slower
```

最新の Ruby

- 知ってると仕事が楽になったりする
- 知ると、追うのが楽しくなるループ
- JIT を有効にすると、結果の変わる
  ベンチマークもありそう (2.6.0)

まとめ

にふぇーでーびる

- benchmark_driver がすごく良い
- Fast Code という Ruby の楽しみ方
  - Ruby の実装を覗いてみる
    - 入り口としてのベンチマークも良い

feedforce