Ruby 2.3.0 の新機能について

Kazuhiro NISHIYAMA

第70回 Ruby関西 勉強会
2016-01-09

自己紹介

- twitter や github では @znz

Ruby 関連

- Ruby (CRuby) のコミッター
- Ruby 関係でいろいろ
  - るびま (Rubyist Magazine) とか
    - http://magazine.rubyist.net/
  - るりま (Ruby リファレンスマニュアル) とか
    - http://docs.ruby-lang.org/ja/
  - Ruby 関連のイベントでの発表

Ruby 2.3.0

- 2015-12-25 リリース
- 2.3 系列最初のリリース

新機能を知るには?

- www.ruby-lang.org のリリースアナウンス
- 配布物の NEWS ファイル
- それらを元に紹介を書いているブログ記事

Frozen String Literal プラグマ (1)

- Ruby 2.1 から `"str".freeze` が最適化
- `freeze` 追加の pull request が社会問題化
- 新しいマジックコメントとコマンドラインオプション追加

Frozen String Literal プラグマ (2)

- 結果: 新しいマジックコメント追加の pull request 増加か(?)
- 要経過観察

Frozen String Literal プラグマ (3)

```
#!/usr/bin/env ruby
# -*- coding: utf-8 -*-
# frozen_string_literal: true
p "".frozen? # => true
```

shebang や coding マジックコメントがあれば、その後

Frozen String Literal プラグマ (4)

_ でも - でも良い

```
# frozen_string_literal: true
```

```
# frozen-string-literal: true
```

Frozen String Literal プラグマ (5)

大文字でも良い

```
# FROZEN-STRING-LITERAL: true
```

しかし typo していると効かないのでしっかり動作確認しましょう

Frozen String Literal プラグマ (6)

コマンドラインオプションでデフォルトの挙動変更

```
ruby --enable=frozen-string-literal foo.rb
```

Frozen String Literal プラグマ (7)

影響を受けなくする (今までの挙動にする) には

```
# frozen_string_literal: false
```

標準添付ライブラリには追加済み

Frozen String Literal プラグマ (8)

```
# frozen-string-literal: true
"str" << "ing"
# ~> foo.rb:2:in `<main>': can't modify frozen String (RuntimeError)
```

Frozen String Literal プラグマ (9)

```
# frozen-string-literal: true
"str" << "ing"
# ~> foo.rb:2:in `<main>': can't modify frozen String,
     created at foo.rb:2 (RuntimeError)
```

ruby --debug=frozen-string-literal foo.rb で実行すると文字列の生成場所がわかる (ruby -d foo.rb でも有効になる)

Frozen String Literal プラグマ (10)

個人的にはおすすめは

- 速度が必要なファイルだけ `frozen-string-literal: true`
- 破壊的変更が必要なリテラルは `"str".dup` で mutable にする
- 互換性が必要なら、引き続き `"str".freeze`

String#+@, String#-@ (1)

- +'foo' は変更可能な String を返す
- -'foo' は freeze された String を返す
  - 覚え方: 水は 0 度以下で凍るから

String#+@, String#-@ (2)

演算子の優先順位に注意が必要

```
+"str" << "ing" # => "string"
```

```
+"str".concat("ing") # ~> RuntimeError
```

```
(+"str").concat("ing") # => "string"
```

String#+@, String#-@ (3)

- "str".freeze は最適化される
- -"str" は最適化されない
- 個人的感想としては、広く使われるかどうかはよく分からない

safe navigation operator (1)

- セーフナビゲーション演算子
- 日本語の定訳は (まだ) ない
- lonely operator (ぼっち演算子) とも呼ばれている

safe navigation operator (2)

- C#, Groovy, Swift などの ?. に似た機能
- ?. は採用できなかったので &.
- 開発版の途中までは .? だった
- Active Support の try! に相当

Active Support

r52232 で修正されているように ActiveSupport ではなく Active Support が正式名称

> Railsのコンポーネント名は語の間にスペースを1つ置く表記を正式なものとする（例: "Active Support"）。

http://railsguides.jp/api_documentation_guidelines.html#%E8%AA%9E%E8%AA%BF

safe navigation operator (3)

```
obj&.foo
```

obj && obj.foo 相当

safe navigation operator (4)

```
obj.try!(:foo, bar())
```

bar() が常に呼ばれる

```
obj&.foo(bar())
```

bar() が呼ばれるかは obj が nil かどうかによる

safe navigation operator (5)

```
obj&.attr += 1
```

属性の代入形式も使える

safe navigation operator (6)

```
obj&.foo.bar
```

obj が nil でも bar が呼ばれる

#11816 に議論あり

safe navigation operator (7)

```
obj&.foo&.bar
```

```
obj && (foo = obj.foo) &&
foo.bar に相当
```

dig (1)

```
params[:order] && params[:order][:shipping_info] &&
  params[:order][:shipping_info][:country]
# or
params[:order][:shipping_info][:country] rescue nil
# or
params.fetch(:order, {}).fetch(:shipping_info, {}).
  fetch(:country, nil)
```

```
params.dig(:order, :shipping_info, :country)
```

dig (2)

- JSON をパースした結果など用
- `Array#dig`, `Hash#dig`, `Struct#dig`, `OpenStruct#dig`

did_you_mean gem (1)

- did_you_mean gem がバンドルされた
- NameError と NoMethodError の発生時に修正候補表示

did_you_mean gem (2)

```
$ ruby -e '"Yuki".starts_with?("Y")'
-e:1:in `<main>': undefined method `starts_with?'
for "Yuki":String (NoMethodError)
Did you mean?  start_with?
```

did_you_mean gem (3)

無効にするには

```
$ ruby --disable=did_you_mean -e '"Yuki".starts_with?("Y")'
-e:1:in `<main>': undefined method `starts_with?'
for "Yuki":String (NoMethodError)
```

```
$ ruby --disable-did_you_mean -e '"Yuki".starts_with?("Y")'
-e:1:in `<main>': undefined method `starts_with?'
for "Yuki":String (NoMethodError)
```

did_you_mean gem (4)

内部的にサポートが増えて速度的にもほぼ問題なくなっている

- Feature #10881 で NoMethodError#receiever が追加された
- Feature #11777 で NameError#local_variables が追加されて TracePoint を使わなくなった

RubyVM::InstructionSequence (1)

- experimental feature
- RubyVM::InstructionSequence#to_binary と .load_from_binary
- コンパイル済みバイナリを読み書き

RubyVM::InstructionSequence (2)

- 使用例として yomikomu gem
- https://github.com/ko1/yomikomu
- 詳細は http://atdot.net/~ko1/diary/201512.html#d13

さまざまなパフォーマンス改善 (1)

https://www.ruby-lang.org/ja/news/2015/12/25/ruby-2-3-0-released/ より

- method entry データ構造の再検討
- 新しい table data 構造
- `Proc#call` 最適化

さまざまなパフォーマンス改善 (2)

- オブジェクトアロケーションとメソッド呼び出しのコードにおけるマシンコードレベルでの最適化
- よりスマートな instance variable データ構造

さまざまなパフォーマンス改善 (3)

- *_nonblock における exception: false キーワード引数のサポート

*_nonblock に exception: false キーワード引数

- IO::WaitReadable/
  IO::WaitWritable

を発生する代わりに

- :wait_readable/:wait_writable

を返す

NEWS

- ここまでが www.ruby-lang.org で取り上げられていた新機能 (とその関連の話)
- ここからは NEWS からその他の新機能を抜粋

indented here document (1)

<<- の代わりに <<~

```
expected_result = <<~SQUIGGLY_HEREDOC
  This would contain specially formatted text.

  That might span many lines
SQUIGGLY_HEREDOC
```

indented here document (2)

一番浅いインデントが削られる

```
p <<~END # => "  foo\nbar\n baz\n"
    foo
 bar
  baz
  END
```

indented here document (3)

2.3.0 では '' の場合にバグあり

```
p <<~'END' # => "foo\nbar\nbaz\n"
    foo
  bar
   baz
   END
```

Array#bsearch_index

- Array#bsearch (要素を返す) のインデックスを返すバージョン

Comparable#== no longer rescues exceptions

```
$ cat /tmp/a.rb
class C;
  include Comparable
  def <=>(o)
    raise 'stop!'
  end
end
p C.new == C.new
$ ruby -v -W0 /tmp/a.rb
ruby 2.2.4p230 (2015-12-16 revision 53155) [x86_64-darwin14]
false
% ruby -v -W0 /tmp/a.rb
ruby 2.3.0p0 (2015-12-25 revision 53290) [x86_64-darwin14]
/tmp/a.rb:4:in `<=>': stop! (RuntimeError)
        from /tmp/a.rb:7:in `=='
        from /tmp/a.rb:7:in `<main>'
```

new Encoding::IBM037 (alias ebcdic-cp-us; dummy)

- EBCDIC は「主にIBM系のメインフレームやオフィスコンピュータなどで使用されている」(Wikipedia より)
- ASCII とは非互換なので dummy

grep_v (1)

- Enumerable#grep_v,
  Enumerator::Lazy#grep_v
- inverse version of #grep

grep_v (2)

```
%w(aaa bbb ccc).grep(/b/) #=> ["bbb"]
%w(aaa bbb ccc).grep_v(/b/) #=> ["aaa", "ccc"]
```

Enumerable#chunk_while (1)

- Enumerable#slice_when の仲間

Enumerable#chunk_while (2)

```
# 1ずつ増加する部分配列ごとに分ける。
[1,2,4,9,10,11,12,15,16,19,20,21]
.chunk_while{|i, j| i + 1 == j}.to_a
# => [[1, 2], [4], [9, 10, 11, 12], [15, 16], [19, 20, 21]]

# ソート済の配列を近い値(差が6以内)の部分配列ごとに分ける。
[3, 11, 14, 25, 28, 29, 29, 41, 55, 57]
.chunk_while{|i, j| j - i <= 6}.to_a
# => [[3], [11, 14], [25, 28, 29, 29], [41], [55, 57]]

# 増加のみの部分配列ごとに分ける。
[0, 9, 2, 2, 3, 2, 7, 5, 9, 5]
.chunk_while{|i, j| i <= j}.to_a
# => [[0, 9], [2, 2, 3], [2, 7], [5, 9], [5]]

# 隣り合う偶数同士、奇数同士の部分配列ごとに分ける。
[7, 5, 9, 2, 0, 7, 9, 4, 2, 0]
.chunk_while{|i, j| i.even? == j.even?}.to_a
# => [[7, 5, 9], [2, 0], [7, 9], [4, 2, 0]]
```

Hash#fetch_values

キーが存在しない時の挙動が
Hash#fetch と同様の
Hash#values_at

```
h = { "cat" => "feline", "dog" => "canine", "cow" => "bovine" }

h.fetch_values("cow", "cat")                  #=> ["bovine", "feline"]
h.fetch_values("cow", "bird")                 # raises KeyError
h.fetch_values("cow", "bird") { |k| k.upcase } #=> ["bovine", "BIRD"]
```

Hash#<=, Hash#<, Hash#>=, Hash#> (1)

包含関係を調べられる

```
h1 = {a:1, b:2}
h2 = {a:1, b:2, c:3}
p h1 <= h2 # => true
p h2 <= h1 # => false
```

Hash#<=, Hash#<, Hash#>=, Hash#> (2)

全順序ではないので Hash#<=> は定義されていない

```
h1 = {a:1}
h2 = {a:2}
p h1 <= h2 # => false
p h2 <= h1 # => false
```

Hash#to_proc

Hash を map などでそのまま使えるようになる

```
h = {
  1 => 10,
  2 => 20,
  3 => 30,
}

[1, 2, 3].map(&h) # => [10, 20, 30]
```

Module#deprecate_constant

deprecated だが互換性のために残している定数を参照した時に警告を表示する機能

```
% cat foo.rb
module Foo
  BAR = :bar
  deprecate_constant :BAR
end
Foo::BAR
% ruby foo.rb
foo.rb:5: warning: constant Foo::BAR is deprecated
```

Numeric#positive?, Numeric#negative?

正の数か負の数かを調べるメソッド

```
bunch_of_numbers.select(&:positive?)
```

Queue#close

close すると無限の nil を push したかのような動作をする

```
# 複数スレッドを同時に開始する例
synq = Queue.new
10.times{
  Thread.new{
    synq.pop #=> nil from closed Queue.
    # do something
  }
}

# do something initialization
synq.close
```

Regexp/String

- Unicode のバージョンが 7.0.0 から 8.0.0 に上がった

String.new に encoding キーワード引数

frozen-string-literal を考慮すると

```
str = "\xA4\xA2".dup.force_encoding('euc-jp')
```

となるのが

```
str = String.new("\xA4\xA2", encoding: 'euc-jp')
```

とかける

Thread#name, Thread#name=

- Thread の名前の取得・設定を行うAPIが追加された
- デバッグなどで識別したい時に便利

inspect での例外抑制

- inspect で Encoding::CompatibilityError が発生する代わりに自動でエスケープされるようになった
- Feature #11801 rb_inspect shouldn't raise error even if calling inspect returns non compatible strings

close 済み IO の IO#close の例外抑制

```
f.close if !f.closed?
```

と書いていたのが

```
f.close
```

だけでよくなった

Object#timeout が deprecated

- Object#timeout が呼ばれた時に deprecated だと警告が出るようになった

Net::FTP

- passive mode がデフォルトになった
- `Net::FTP.default_passive=` で設定可能

Net::HTTP

- Net::HTTP#open_timeout のデフォルトが nil から 60 (秒) に変わった

Net::Telnet

- net-telnet gem として外だしされた
- https://github.com/ruby/net-telnet
- bundled gem として配布には含まれる

rake

- rake も ruby のレポジトリに直接含まれるのはやめて bundled gem になった

$SAFE

- $SAFE=2 と $SAFE=3 が obsolete になった
- $SAFE を 2 以上に設定しようとすると ArgumentError

CGI.escapeHTML 高速化

- CGI.escapeHTML が C 拡張で高速化された
- https://github.com/ruby/ruby/pull/1164
- 最大約7倍
- ワーストケースでも約3倍

まとめ

- いろんな新機能が追加された
- いろいろと高速化された
- 紹介しきれなかったものは NEWS 参照
- https://github.com/ruby/ruby/blob/v2_3_0/NEWS