Ruby 2.3 のてざわり

新機能と使いどころ

Kunihiko Ito

ESM

富山合同勉強会2016

2016-01-30

はじめ

まして

p self

p self

- 名前: 伊藤 邦彦
- 出身: 富山
- 在住: 東京
- 所属: ESM アジャイル事業部
- 仕事: [Rails，neo4j]

@kunitoo

@kunitoo

From Java To Ruby

変わったこと

- IDE を使わなくなった
- REPLで試しながら書くようになった

プログラムを書くときにしていること

- rails console を立ちあげる

1. console に書いて試していく
2. ファイルに書き移していく
3. 複雑になったらテストを書く

今日伝えたいこと

Ruby おもしろい

Ruby を触ってみたい

今日の内容

昨年の12月25日にリリースされた
Ruby 2.3 の新機能とその使いどこ
ろを紹介します

提　供

情報化技術を通じて社会と共生する

esm

永和システムマネジメント

Ruby 2.3 新機能

すぐに使える

- safe navigation operator
- SQUIGGLY HEREDOC

使う機会が増えそう

- #dig
- Enumerable#grep_v

その他

- the did_you_mean gem
- NameError#receiver
- Hash#to_proc

すぐに使える

- safe navigation operator
- SQUIGGLY HEREDOC

safe navigation operator

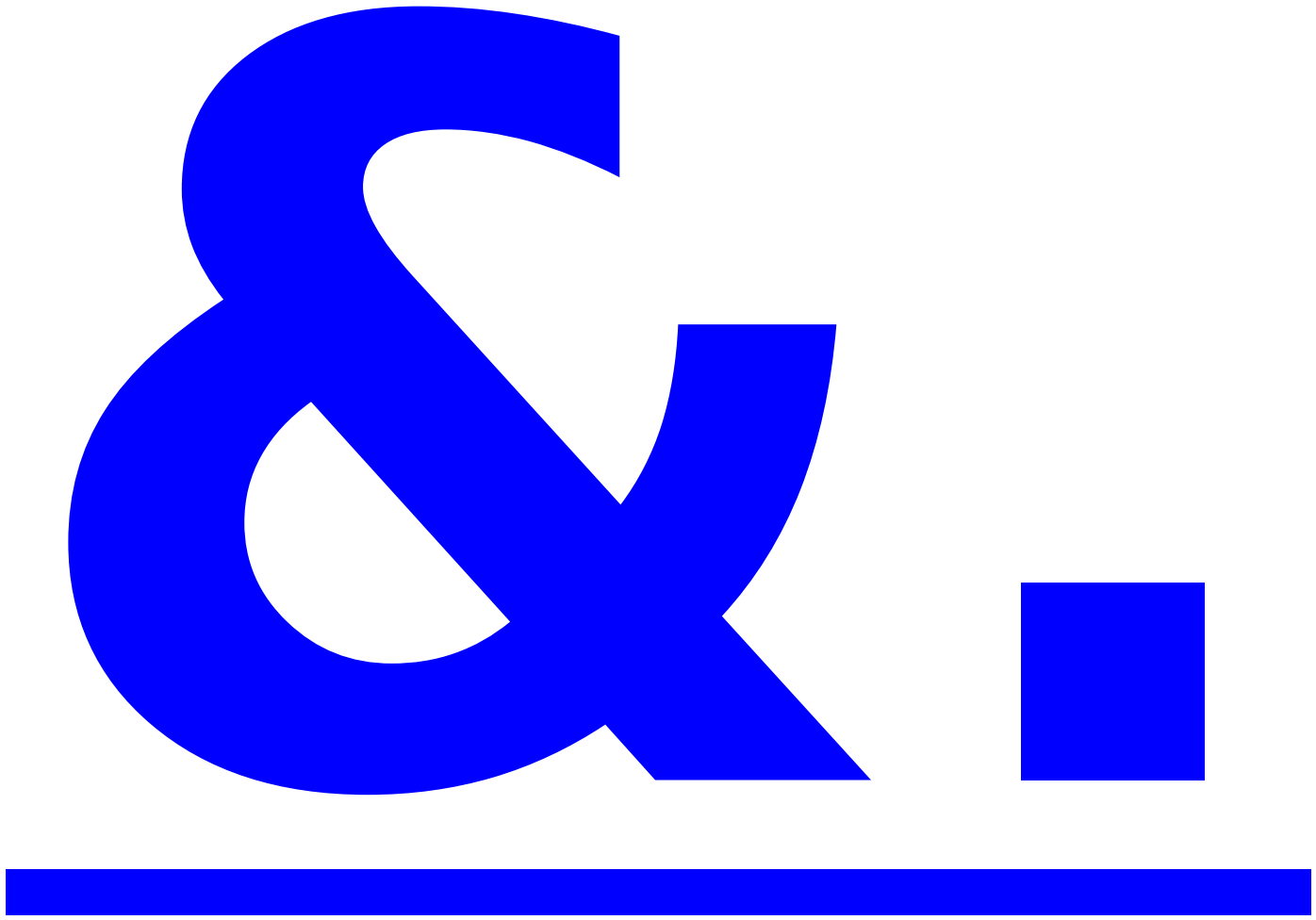
- 別名
  - lonely operator
  - ぼっち演算子
- Active Support の try! と同様の挙動

safe navigation operator

```
obj = nil

obj.hoge #=> NoMethodError:
# undefined method `hoge' for nil:NilClass

obj&.hoge #=> nil
```



&. の使いどころ

ユーザーがログインしていないとき

```
current_user = nil
current_user&.name #=> nil
current_user.try!(:name) #=> nil

currnet_user = User.find_by(name: 'kunitoo')
current_user&.name #=> 'kunitoo'
current_user.try!(:name) #=> 'kunitoo'
```

SQUIGGLY HEREDOC

ヒアドキュメント内のインデントを取り除く `<<~` リテラルです

Active Support の strip_heredoc と同様の動きをします

SQUIGGLY HEREDOC

```
# 通常のヒアドキュメント
<<-HEREDOC
    hoge

    fuga
HEREDOC
#=> "    hoge\n\n    fuga\n"
# SQUIGGLY HEREDOC
<<~SQUIGGLY_HEREDOC
    hoge

    fuga
SQUIGGLY_HEREDOC #=> "hoge\n\nfuga\n"
```

SQUIGGLY HEREDOC

```
# strip_heredoc
<<-HEREDOC.strip_heredoc
    hoge

    fuga
HEREDOC
#=> "hoge\n\nfuga\n"
# SQUIGGLY HEREDOC
<<~SQUIGGLY_HEREDOC
    hoge

    fuga
SQUIGGLY_HEREDOC #=> "hoge\n\nfuga\n"
```

SQUIGGLY_HEREDOC の使いどころ

簡易なメッセージやQueryのテンプレートとして使う

SQUIGGLY_HEREDOC の使いどころ

```
def calc_billin
  query = <<~SQL
     INSERT INTO bills (name, total)
     SELECT name, sum(amount) AS total
     FROM
       orders
     JOIN
       ...
     WHERE ...
  SQL
  ActiveRecord::Base.connection.execute(query)
end
```

使う機会が増えそう

- #dig
- Enumerable#grep_v

#dig

- 追加されたクラス
  - Array
  - Hash
  - Struct
  - OpenStruct
- 深い階層にある値を取得することができる

#dig

```ruby
a = [[1, 2], [3, 4]]

a.dig(0, 1) #=> 2
a.dig(1, 2) #=> nil

h = {foo: {bar: 1}}

h.dig(:foo, :bar) #=> 1
```

#dig

dig メソッドを持つオジェクトであれば、交ざっていても使えます

```
user = {
  user: {
    address: [
      {name: '富山市', ruby: 'とやまし'},
      {name: '呉羽町', ruby: 'くれはまち'}
    ]
  }
}
user.dig(:user, :address, 1, :name) #=> "呉羽町"
user.dig(:user, :address, 2, :name) #=> nil
```

#dig の使いどころ

- JSON の値の取得
- Hash の値に Array がある場合
- request paramter

#dig の使いどころ

```ruby
(0..1).map {|index|
  user.dig(:user, :address, index, :name)
} #=> ["呉羽町", nil]
```

Enumerable#grep_v

Enumerable#grep のマッチの条件を逆にして、pattern === item が成立 しない要素を全て含んだ配列を返します

```
(1..10).grep_v 2..5 # => [1, 6, 7, 8, 9, 10]
```

正規表現クイズ

Bob, John, Jahn の中から Jo から始まる名前以外を抽出するには?

1. ^[^Jo]
2. ^[^J][^o]
3. ^(?!Jo)

こたえ

3. ^(?!Jo)

```
names.grep /^[^Jo]/ #=> ['Bob']
names.grep /^[^J][^o]/ #=> []
names.grep /^(?!Jo)/ #=> ["Bob", "Jahn"]
```

Enumerable#grep_v の使いどころ

Enumerable#grep では逆の条件が書きづらいときや型情報を使うとき

```
['Bob', 'John', 'Jahn'].grep_v /^Jo/ #=> ["Bob", "Jahn"]
[1, '1', 1.0].grep_v String #=> [1, 1.0]
```

the did_you_mean gem

did_you_mean gem がバンドルされるようになりました。

NameError と NoMethodError の発生時、デバッグを容易にするため、正しい名前と思われる候補を合わせて表示します。

the did_you_mean gem の使いどころ

irb や rails console でお世話になります。自信のないスペルでも調べなくてもよくなります。

the did_you_mean gem の使いどころ

```
'hello'.revarse
# => NoMethodError: undefined method `revarse' for "hello":String
# Did you mean?  reverse
#                reverse!

'str'.encodeing
# => NoMethodError: undefined method `encodeing' for "str":String
# Did you mean?  encoding
#                encode
#                encode!
```

NameError#receiver

NameError が発生した時のレシーバオブジェクトを返します

```
begin
  'abc'.foo
rescue => e
  p e.receiver
end #=> 'abc'
```

NameError#receiver の使いどころ

```
module DidYouMean
  class VariableNameChecker
    ... snip ...
    def initialize(exception)
      @name       = exception.name.to_s.tr("@", "")
      @lvar_names = exception.local_variables
      receiver    = exception.receiver

      @method_names = receiver.methods + receiver.private_methods
      @ivar_names   = receiver.instance_variables
      @cvar_names   = receiver.class.class_variables
      @cvar_names  += receiver.class_variables if receiver.kind_of?(Module)
    end
  end
end
```

NameError#receiver の使いどころ

デバッグするときに呼び出し元のオブジェクトそのものを取得できる

Hash#to_proc

self に対応する Proc オブジェクトを返します。

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

Hash#to_proc の使いどころ

おもいつきません...

その他

- frozen string literal
- String#+@, String#-@

まとめ

- Rails に存在した &. や HEREDOC はすぐに使えそう
- #dig や grep_v は意識していれば、使えるところがありそう
- Hash#to_proc はだれか使いどころおしえてください

Enjoy Ruby