Re-introduction to Ruby

Shugo Maeda

2009-07-19

Self introduction

- 前田 修吾 (Shugo Maeda)
  - Ruby committer
  - NaCl
  - Ruby Association
  - RubyWorld Conference

Ruby committer

- 最近commitしてませんが…
  - no commit in these days...

NaCl

- ネットワーク応用通信研究所
- まつもとさんの会社
  - a company that Matz works for
- 受託開発やってます!
  - entrusted development

Ruby Association

- Rubyの普及と発展のための組織
  - an organization for development and popularization of Ruby
- 認定制度 (Certification)
  - プログラマ (for programmers)
  - システムインテグレータ (for SIers)

RubyWorld Conference

期間 (Period)

- 2009-09-7 (Mon) - 2009-09-08 (Tue)

会場 (Venue)

- 島根県松江市 くにびきメッセ
  - Matsue, Shimane, Japan

入場料 (Admission Fee)

- Free (registration required!)

Program

- http://www.rubyworld-conf.org/ja/program/

1日目: 国際会議場

10:00-10:55	オープニングセレモニー 記念講演: 経済産業省商務情報政策局
11:00-12:00	基調講演: Tim Bray (Sun Microsystems)
13:00-15:00	国際標準化 講演1: 筧捷彦 (早稲田大学) 講演2: まつもとゆきひろ パネルディスカッション
15:15-16:00	Bruce Tate
16:15-17:00	正村勉 (日立ソフトウェアエンジニアリング)
17:15-18:00	吉川正晃 (富士通四国システムズ)

1日目: 501会議室

13:00-15:00　Rubyビジネス・コモンズ　ワークショップ
15:15-16:00　竹内郁雄（東京大学）
16:15-17:00　久野靖（筑波大学）
17:15-18:00　パネルディスカッション
モデレータ：野田哲夫（島根大学）
パネリスト：竹内郁雄（東京大学）
久野靖（筑波大学）
原元司（松江高専）
最首英裕（イーシー・ワン）

2日目: 国際会議場

10:00-11:00	基調講演: まつもとゆきひろ
11:15-12:00	Jeremy Kemper (37signals)
13:00-13:30	笹田耕一 (東京大学)
13:30-14:00	頃末和義 (マイクロソフト)
14:00-14:30	Charles Oliver Nutter (Sun Microsystems) Thomas Enebo (Sun Microsystems)
14:30-15:00	Evan Phoenix (Engine Yard)
15:15-16:00	Stephen Kung (Shanghai on Rails)

2日目: 501会議室

11:15-12:00　森正弥（楽天）
13:00-15:00　開催地リレートーク
　　松江市
　　島根県
　　井上浩（ネットワーク応用通信研究所）
　　室脇俊二（テクノプロジェクト）
　　坂田真一（バブ日立ソフト）
　　田辺勉（小松電機産業）
15:15-16:00　橋本健太（クックパッド）
16:00-16:30　クロージングセレモニー

Theme

- Re-introduction to Ruby

Background

- 以前は実装は一つのみ
  - only one implementation in the past
- 仕様と実装を分ける必要はなかった
  - no need to separate spec from it
- ところが最近は多数の実装が
  - several implementations in these days

Specification

- RubySpec
- Drafting of the standard

RubySpec

RubySpec is a project to write a complete, executable specification for the Ruby programming language.

「RubySpec - The Standard You Trust」
http://rubyspec.org/

Drafting of the standard

- IPA事業
  - Rubyの国際標準化に関する調査
- 文書化された仕様
  - Written specification
- RubyWorld Conferenceで発表予定
  - Presentation in RubyWorld

What is Ruby?

- 改めて考えよう
  - Let's take anothor look

Hello world

```
print "Hello, world!¥n"
```

Description

```
print "Hello, world!¥n"
```

- メソッド呼び出し
  - method call
- 括弧の省略
  - parenthesis omission

Python 2 version

```
print "Hello, world!¥n"
```

- Ruby版との違いは?
  - What is different from Ruby?

Python 2 version

```
print "Hello, world!¥n"
```

- Ruby版との違いは?
  - What is different from Ruby?
- printが紫色
  - print is purple

print in Python 2

- 紫色 = 予約語
  - purple = reserved word
- print文
  - print statement
- だから括弧がいらない
  - so no parenthesis

Python 3 version

```
print(”Hello, world!¥n”)
```

print in Python 3

- 関数呼び出し
  - function call
- だから括弧が必要
  - so parenthesis needed
- よりシンプルな言語仕様に
  - simpler than Python 2

Ruby 0.49 version

```
print(”Hello world¥n”)
```

print in Ruby 0.49

- 当時もメソッド呼び出し
  - method call
- ただし、括弧は省略できなかった
  - but can't omit parenthesis

Parenthesis omission

```
Fri Aug 26 10:46:30 1994  Yukihiro Matsumoto  (matz@ix-02)

        * spec: 整理された文法にしたがって書き直した.

        * parse.y: ここ数日で混乱していた文法を整理した. 括弧を省略したメ
          ソッド呼び出しができるようになったこと, modifierが付けられるよう
          になったこと, returnにリストが渡せるようになったことが主な変更点
          である.
```

Parenthesis omission

- Perl 5のリリース(1994/10/17)よりも早い!
  - introduced before the release of Perl 5
- Perl 4ではユーザ定義関数は括弧が省略できない
  - can't omit parenthesis for user defined functions in Perl 4

Pros

- 書きやすい
  - easy to write
- 読みやすい
  - easy to read
- 拡張性
  - extensible

Easy to write

```
p x
```

- タイプ数が少ない
  - less typing

Easy to read

```
class Foo
  include Bar
  attr_reader :baz, :quux
end
```

- 宣言的
  - declarative

Extensible

- Ruby 0.49では、includeは予約語
  - include is a reserved word in 0.49
- 括弧の省略によりメソッド化
  - reimplemented as a method thanks to parenthesis omission

Importance of syntax

- 拡張性の高い、もっとシンプルな言語
  - simpler extensible languages
    - Lisp, Smalltalk, JavaScript...
- Rubyは構文を重視
  - syntax is important in Ruby

Importance of syntax

- includeがメソッドとして実装されていることをユーザは意識しない
  - Users are not aware of that include is implemented as a method

Cons

- 文法の複雑化
  - complex syntax

Example 1

```
p (1 + 1)
p (1 + 1).to_s
p(1 + 1).to_s
p (1, 2)
```

Result

```
p (1 + 1) #=> 2
p (1 + 1).to_s #=> "2"
p(1 + 1).to_s #=> 2
p (1, 2) #=> syntax error (warn in 1.8)
```

Example 2

```
p {"a"=>1}
p {}
```

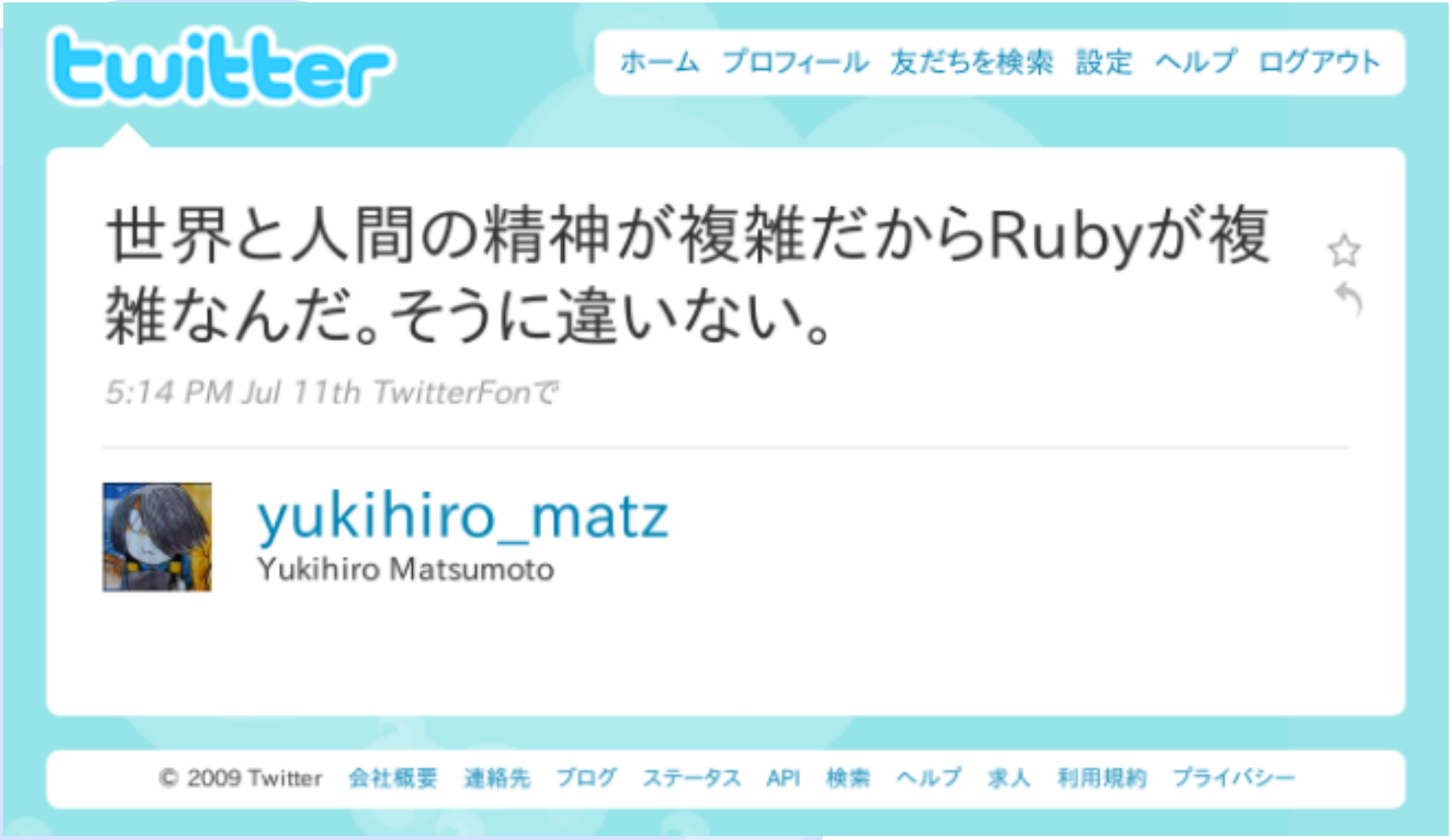
Result

```
p {"a"=>1} #=> syntax error
p {} #=> no output
```

Simple Syntax?

- プログラムはシンプルに見える
  - programs look simple
- 文法自体は複雑
  - syntax itself is complex
- 構文解析が大変
  - hard to parse

RT @yukihiro_matz:



RT @yukihiro_matz:

- because the world and the human spirit are complex, Ruby is complex
  - Larry Wall?

Why so complex?

- specがないから
  - because there is no spec
- でもそれがよかったんじゃない?
  - but it might work well, mightn't it?

Summary (Hello world)

- Pythonはシンプルに
  - Python got simpler
- Rubyは複雑に
  - Python got more complex

ChangeLog again

```
Fri Aug 26 10:46:30 1994  Yukihiro Matsumoto  (matz@ix-02)

        * spec: 整理された文法にしたがって書き直した.

        * parse.y: ここ数日で混乱していた文法を整理した. 括弧を省略したメ
          ソッド呼び出しができるようになったこと, modifierが付けられるよう
          になったこと, returnにリストが渡せるようになったことが主な変更点
          である.
```

spec

- 昔はspecがあった!
  - in the past, there was a file named spec in the source archive

Ruby 0.49 spec (1)

** イテレータ

イテレータとは制御構造(特にループ)の抽象化のために用いられるメソッドの一種である. イテレータの呼び出しは以下の構文で行なわれる.

```
do
  文1
using 変数
  文2
end [ do ]
```

Ruby 0.49 spec (2)

- イテレータの形式が違う
- endの後に対応する予約語を書ける
- クラス/ローカル変数が同じ識別子
- private methodは@ではじまる
- raiseはない(failのみ)

Ruby 0.49 spec (3)

- beginはprotect
- !や!=はメソッド呼び出し!
- ::は2要素の配列(後にCons)を生成
- 他にも%FOOでクラス変数とか色々

Example (1/2)

```
class greeting
  def !=(array)
    do array.each using x
      @say_hello(x)
    end
    fail("error")
  end def

  def @say_hello(x)
    print("Hello, ", x, "!¥n")
  end def
end class
```

Example (2/2)

```
protect
  g = greeting.new()
  g != ("Matz" :: "Larry")
  print("success¥n")
resque
  print("failed¥n")
end protect
```

- resque is a typo by Matz

Impression

- こんなのRubyじゃない!
  - It's not Ruby!

What is Ruby?

- 変わり続ける言語
  - Ever-changing language

Chanegs in Ruby 1.9

- Ruby 1.8の問題の解決
  - Solve problems in Ruby 1.8

Problems in Ruby 1.8

- 暗黙のレキシカルスコープ
  - implicit lexical scope
- 多重代入
  - multiple assignment

Implicit lexical scope

- Rubyには変数宣言がない
  - no variable declaration
- 最初の代入が宣言を兼ねる
  - declared by first assignments
- ブロックローカル変数
  - block local variable

Why block local variable?

```
Subject: [ruby-list:8474] Re: scope of local variables
From: matz netlab.co.jp (Yukihiro Matsumoto)
Date: Tue, 23 Jun 1998 18:16:53 +0900

まつもと ゆきひろです
(snip)
ブロック内変数は
Block local variables are necessary for:

  Thread固有変数
  Thread local variable
  手続きオブジェクト固有変数
  Proc object local variable

などのために必要なものです.
```

Example

```
m, r, max = Mutex.new, 0, 100
(1..max).map{
  Thread.new{
    i=0
    while i<max*max
      i+=1
      m.synchronize{
        r += 1
      }
    end
  }
}.each{|e| e.join}
```


Problem 1

```
i = 0
[1, 2, 3].each do |i|
  ...
end
p i #=> 3
```


Workaround 1

- 変数名が衝突しないように気を付ける
  - do not use same name

Problem 2

```
1.times do
  hash = {:a => 1, :b => 2}
end
p hash
```


Problem 2

```
1.times do
  hash = {:a => 1, :b => 2}
end
p hash #=> 494778040
```


Workaround 2

```
Subject: [ruby-list:8474] Re: scope of local variables
From: matz netlab.co.jp (Yukihiro Matsumoto)
Date: Tue, 23 Jun 1998 18:16:53 +0900

まつもと ゆきひろです
(snip)
ですから, ローカル変数のワナに引っかからないためには
So, you should obey the following rule to avoid the trap
of local variable:

   「普通の」ローカル変数は先頭で代入(宣言)しておく
  assign (declare) local variables at the beginning of methods

という「ローカル変数のオキテ」に従うことを強くお勧めします.
```


Workaround 2

```
Subject: [ruby-list:8474] Re: scope of local variables
From: matz netlab.co.jp (Yukihiro Matsumoto)
Date: Tue, 23 Jun 1998 18:16:53 +0900

まつもと ゆきひろです
(snip)
ですから, ローカル変数のワナに引っかからないためには
So, you should obey the following rule to avoid the trap
of local variables:

   「普通の」ローカル変数は先頭で代入(宣言)しておく
  assign (declare) local variables at the beginning of methods

という「ローカル変数のオキテ」に従うことを強くお勧めします.
```

- えー、宣言いらないんじゃないのー?
  - Oh, should we declare local variables?

Larry Wall criticized

As for specifics, I must say that the example of Ruby is the main reason I decided against implicit lexical scoping for Perl6

http://interviews.slashdot.org/article.pl?sid=02/09/06/1343222&mode=thread&tid=145

Matz agreed

また、Rubyのローカル変数のスコープの点に気がついた彼はやっぱりとても鋭い人だと思います(古今最高のルネッサンス人かどうかは別として)。私自身もこれはRubyの最大の欠点だと思っていて、機会があれば直したいと思っているので。

http://slashdot.jp/developers/article.pl?sid=03/03/14/0258247

Multiple assignment

```
x = 1
y = 2
x, y = y, x
p [x, y] #=> [2, 1]
```


Problem

- 謎の挙動
  - enigmatic behavior

Enigmatic behavior

```
class Foo
  def to_ary; [1, 2, 3]; end
end
a, *b = Foo.new
p a #=> 1
p b #=> [2, 3]
*c = Foo.new
p c #=> [#<Foo:0xb7d088f8>]
    #   not [1, 2, 3]
```

- Is it a bug?

Maybe not a bug

```
*x = [1]
# x should be [[1]] ([1] if to_ary called)

# Why?
y = *[1]
# y is 1 in 1.8
*x = [1]
y = *x
# y should be [1]
```


Changes in Ruby 1.9

- Block
- Multiple assignment

Block
~~ブロックパラメータのスコープ~~

- Scope of block parameters

- ブロックローカル変数宣言

- Block local variable declaration

- メソッド仮引数との整合性

- Consistency with method parameters

- ->

Scope of block parameters

```
i = 0
[1, 2, 3].each do |i|
  ...
end
p i #=> 0
```


Block local var declaration

```
x = 0
[1, 2, 3].each do |i; x|
  x = 1
end
p x #=> 0
```


Consistency with method parameters

```
lambda {|x, y = 2, &b| ... }.call(1) {
  ...
}
lambda {|x| p x}.call(1, 2)
#=> ArgumentError
Proc.new {|x| p x}.call(1, 2)
#=> 1
```


->

```
->(x = 1 | 2) { p x }.()
lambda {|x = (1 | 2)| p x }.()
```


Multiple assignment

- 多重代入はシンプルに
  - multiple assignment got simpler
- 詳細は田中哲さんの日記参照
  - see akr's diary for details
  - 2007-08-17
- Happy end?

New problem

- 以下で、yが配列でなかった場合、どのように振る舞うべきか
  - if y is not an array, how should the following code behave?

```
x = *y
```


Conversion to array

- in Ruby 1.8
  - call to_ary or to_a
- in Ruby 1.9
  - call only to_a
  - → Matz said, should call only to_ary

Problem with Range

```
x = *1..5
p x
# [1, 2, 3, 4, 5] → [1..10]?
```


Possible solutions

- Rangeにto_aryを定義 → NG
  - define Range#to_ary → NG
- Ruby 1.8の挙動に戻す?
  - reverted to the behavior of Ruby 1.8
- to_splat?

Summary (Changes in 1.9)

- いくつかの問題が解決された
  - some problems are solved
- でも…
  - But...

Scope problem again

```
1.times do
  hash = {:a => 1, :b => 2}
end
p hash #=> 494778040
```

- 1.9でも未解決
  - not solved in 1.9

Proposal 1

- 暗黙のレキシカルスコーピングの廃止
  - obsolete implicit lexical scoping

Obsolete implicit scoping

```
1.times do
  hash = {:a => 1,  :b => 2}
end
p hash #=> {:a => 1,  :b => 2}
```


Probem

- 互換性がない
  - Compatibility

Solution

- デフォルトでは暗黙のスコーピングを有効に
  - implicit scoping is enabled by default
- コマンドラインオプションで無効化
  - disable by a command line option

Problem of command line option

- 両方の挙動が共存できない
  - Both behavior can't coexist

Solution

- magic comment

magic comment

- コンパイラへの指示
  - directive for compiler
- ファイル単位
  - per file

Example

```
# -*- encoding: utf-8 -*-
print "こんにちは¥n"
```


implicit-scope

```
# -*- implicit-scope: off -*-
1.times do
  hash = {:a => 1, :b => 2}
end
p hash #=> {:a => 1, :b => 2}
```

- デフォルトはon (1.8と同じ)
  - default is on (same as 1.8)

Explicit scoping

```
1.times do |;x|
  x = 1
end
p x #=> error
```

- 何か汚い
  - it looks dirty

Proposal 2

- let

let

```
let do |x|
  x = 1
end
p x #=> Error
```

- letはブロックを1回呼ぶだけ
  - let just calls the block once

Initial value

```
let do |x, y = 2|
  ...
end
```

- 初期値も指定できる!
  - You can specify the initial value!

Demo

- 実装してみました
  - implemented them

Patch

- http://shugo.net/RubyKaigi/2009/no-implicit-lexical-scope.diff

Conclusion

- Rubyは変わる
  - Ruby is an ever-changing language
- Rubyを変える
  - You may also be able to change it

References

- matzを説得する方法
  - http://www.a-k-r.org/pub/howto-persuade-matz-rubykaigi2008.pdf
- strftimeの%Nは採用されました
  - 説得はしてないかも

Thank you

- ご質問は?
- Any questions?

100