# Ruby1.9 の仕様 — Array 関連の新しい機能

小波秀雄

2008/02/16

## 1　配列まわりの定石

従来の Ruby 1.8 で使われてきた定石ともいえる手法をまずおさらいしておきましょう。

### 1.1　join — 整形した文字列を出力

配列の要素を適当な文字列で区切って出力するのには `join` が便利です。irb で次のように入力して結果をみてください。

```
irb(main):001:0> ar = ["1","a","X"]
=> ["1", "a", "X"]
irb(main):002:0> ar.join(",")
=> "1,a,X"
irb(main):003:0>
```

### 1.2　イテレータは配列使いの強力な道具

いかにも Ruby らしい道具，イテレータを簡単に紹介しておきます。配列を扱う Array クラスやハッシュを扱う Hash クラス，範囲を扱う Range クラスは，**要素を順番に数え上げることができる**，つまり **Enumerable** であるという性質を持っています。このようなクラスによく用いられるメソッドは `each` です。

たとえば次のソースを見てください。

```
ar = ["Apple","Mango","Banana","Melon"]
ar.each do |item|
  puts item
end
```

実行すると，配列 `ar` の要素がひとつずつ取り出されては変数 `item` に代入されていることが分かります。このように Enumerable なオブジェクトから要素を順番にひとつずつ取り出すようなメソッドを**イテレータ (Iterator)** と呼んでいます。イテレータにはいろいろな種類がありますが，初心者はとりあえず `each` を知っておくだけでも便利です。

### ちょっとおことわり— ブロックの囲み方

Ruby では，繰り返し処理のブロックを囲むのに { ... } を使うか，あるいは do ... end を使うかは，プログラマの好みに任されていて，短い処理では { ... } が使われることが多いようです。したがって上のプログラムは下のように書いてもかまいません。

```
ar = ["Apple","Mango","Banana","Melon"]
ar.each{ |item|
  puts item
}
```

上のブロックを ar.each{|item| puts item} のように一気に書く人もよくいます。もちろん do ... end でも書けるのですが，短く書きたい人はこちらを好むようです。

以下では，説明がしやすいように do ... end を使って，行も分ける書法をとることにします。

## 2 情報源

Ruby 1.9.0 **の** NEWS **というドキュメント**: ソースのアーカイブを解凍すると doc/NEWS というファイルがあり，“Changes for 1.9” として変更が一覧されています。

Ruby 1.9.0 **の** C **ソース**: ソースのアーカイブを解凍して C のソースを覗いてみましょう。ソースを読むのは大変！だけどコメントにメソッドの動作が記述されています (array.c, enumerator.c *etc.*)。

```
/*
 *  call-seq:
 *     array.shuffle -> an_array
 *
 *  Returns a new array with elements of this array shuffled.
 *
 *     a = [ 1, 2, 3 ]           #=> [1, 2, 3]
 *     a.shuffle                 #=> [2, 3, 1]
 */
```

yhara **さんの** Hiki: **Ruby1.9.0-0 で増減したメソッド一覧**が整理してまとめてあります。

http://mono.kmc.gr.jp/~yhara/w/?Ruby1900MethodsBuiltin#l0

# 3　Array クラスの新しいメソッド

## 3.1　choice — ランダムに要素を取り出す

配列の要素から 1 個をランダムに選んで返します。次の例はサイコロを 10 回振る操作のシミュレーションです。

```
dice = [1,2,3,4,5,6]
10.times do
   puts dice.choice
end
```

じゃんけんも簡単！

```
3.times do
   puts ["ぐう","ちょき","ぱあ"].choice
end
```

## 3.2　shuffle — 要素をランダムに混ぜる

トランプのカードを切るように，配列の要素をシャッフルした配列を返します。シャッフルに使われる乱数は srand(seed) で「種まき」することができます。

```
ar = [1,2,3,4,5,6,7,8,9,10]
# srand(1)  # これをコメントアウトすると，毎回同じシャッフルが起きる
5.times do
   p ar.shuffle
end
```

### shuffle! で破壊的に変更

shuffle! メソッドはそれが作用した配列を破壊的に変更します。irb で試してみましょう。

```
irb(main):065:0> ar = ["A","B","C","D"]
=> ["A", "B", "C", "D"]
irb(main):066:0> ar.shuffle
=> ["B", "D", "A", "C"]
irb(main):067:0> ar
=> ["A", "B", "C", "D"]
irb(main):068:0> ar.shuffle!
=> ["D", "A", "B", "C"]
irb(main):069:0> ar
=> ["D", "A", "B", "C"]
```

### 3.3 permutation — 順列の生成

配列の要素を並べて得られる，指定された長さの順列をすべて生成します。

```
ar = ["A","B","C","D"]
ar.permutation(2) do |x|
  p x
  puts x.join(" ")
end
```

得られる順列の総数は，配列の長さ（要素数）を $n$, 並べる数を $m$ としたときに，${}_n\mathrm{P}_m = m!/(n-m)!$ で与えられます。これは簡単に巨大な数になることに留意しておかないといけません。

### 3.4 combination — 組み合わせの生成

配列から，引数で指定された数の要素を取り出して作られる，すべての組み合わせを生成します。

```
ar = ["A","B","C","D"]
ar.combination(3) do |x|
  p x
end
```

実行結果は次のようになります。

```
["A", "B", "C"]
["A", "B", "D"]
["A", "C", "D"]
["B", "C", "D"]
```

組み合わせの総数は 配列の要素の数を $n$, 取り出す要素の数を $m$ として，${}_n\mathrm{C}_m = 4$ となっていることが分かります。$n$ が大きく，かつ $m$ がそれほど小さくない場合には，この値は非常に大きくなるので，使うときにはどれくらいの組み合わせができるかをあらかじめ予想した方がよいでしょう。

たとえば 52 枚のトランプから 7 枚選ぶ組み合わせをすべて取り出そうなどというのは，やめたほうがいいと思います[*1]。

### 3.5 product — 2 つの配列から取り出した要素の組み合わせを生成

2 つの配列 ar1, ar2 があったとき，次の演算によって，ar1 と ar2 からそれぞれ 1 個ずつ要素を取り出してできる，すべての組み合わせの配列を要素とする配列を生成します。

```
ar1.product(ar2)
```

[*1] $\frac{52!}{7!45!} = 133784560$ にもなります。

たとえば次のようになります。

```
[1,2,3].product([4,5])     # => [[1,4],[1,5],[2,4],[2,5],[3,4],[3,5]]
```

次のような応用を考えてみました。結果はどうなるでしょうか？

```
familynames = ["山田","川田","森田","畑田"]
firstnames = ["花子","史恵","加奈"]
p familynames.product(firstnames)
familynames.product(firstnames).each do |fullname|
  puts fullname.join(" ")
end
```

# 4 Enumerable::Enumerator クラスの新設

## 4.1 Array クラスのインスタンスに each を作用させると？

Enumerable::Enumerator とは何か，まず irb で実験してみましょう。

**Ruby 1.8.6**

```
>> ar = ["A","B","C","D"]
=> ["A", "B", "C", "D"]
>> ar.each
LocalJumpError: no block given
from (irb):3:in ‘each’
```

**Ruby 1.9.0**

```
irb(main):113:0> ar
=> ["A", "B", "C", "D"]
irb(main):114:0> ar.each
=> #<Enumerable::Enumerator:0x3826b0>
```

配列に each を作用させただけでは，Ruby 1.8 ではブロックが与えられていないという例外が発生します。

しかし，Ruby 1.9 では Enumerable::Enumerator という見なれないクラスのオブジェクトが生成していることが分かります。このクラスは 1.8 ですでに添付ライブラリとして登場していますが，1.9 では仕様を拡張して組み込みクラスになりました。

なお，1..10 のような Range オブジェクト，{"a" => 12, "b" => 20} のような Hash オブジェクトについても Array クラスと同様の処理が可能ですし，10.times の形も Enumerator クラスに属します。

## 4.2 複数の配列を同時に回す処理

このことによって，次のような新しい記述が可能になっています。

```
ax = [1,3,5,7,9].each
ay = [2,4,6,8,10].each
loop do
  x = ax.next
  y = ay.next
  puts "#{x} x #{y} -> #{x*y}"
end
```

Enumerator を使わない従来の処理で 2 つの配列を回す方法は，zip を使う[*2]，transpose を使う[*3] などがありますが，このやり方の方がずっとスマートです。

[*2] ar1.zip(ar2).each{|v1,v2| puts v1*v2 }
[*3] [ar1,ar2].transpose.each{|v1,v2| puts v1*v2}

## 4.3 文字列に対するイテレータが変更された

String オブジェクト (文字列) は従来 Enumerable なオブジェクトとして，each メソッドで処理できていました。しかし，その内容は改行コードで各行を切り出すものでした。irb で試してみましょう。

```
irb(main):155:0> 'ABC'.each{|c| p c}
NoMethodError: undefined method `each' for "ABC":String
```

しかし，Ruby 1.9 から文字列は Enumerable でなくなり，その代わりに，each_byte, each_byte, each_line メソッドで Enumerator のオブジェクトを生成できるように機能が拡張されました。

```
irb(main):160:0> str = "AB\nCD"
=> "AB\nCDE"
irb(main):161:0> str.each_byte {|v| p v}
65
66
10
67
68
irb(main):162:0> str.each_char {|v| p v}
"A"
"B"
"\n"
"C"
"D"
irb(main):163:0> str.each_line {|v| p v} # 1.8 の each に相当
"AB\n"
"CD"
```

each_char でちょっといたずらしてみましょう。実は文字コードの処理も改善されていて，従来 jcode ライブラリのお世話になったのが不要になっています。

```
#! /usr/local/bin/ruby -Ku  # -K の後ろは s，e など文字コードに合わせて変えてね。
e1 = "こんにちは".each_char
e2 = "こんばんは".each_char
loop do
  print   e1.next, e2.next
end
puts
```

出力は **ここんんにばちんはは** となりました。

### 4.4 Array クラスの permutation, combination は enumerator のオブジェクトを返す

3.3, 3.4 で取り上げられている permutation, combination を使ったプログラムでは， each メソッドがないのにイテレータを使った繰り返しと同じ処理ができています。それはこれらのメソッドが Enumerable::Enumerator クラスのオブジェクトを返すからです。

Enumerable::Enumerator クラスのオブジェクトは to_a メソッドで配列に変換することができます。to_a の 'a' は Array の 'a' です。次のようにして irb で確かめてみましょう。

```
irb(main):3:0> ar = ["A", "B", "C", "D"]
=> ["A", "B", "C", "D"]
irb(main):4:0> ar.combination(2)
=> #<Enumerable::Enumerator:0x1a6fe4>
irb(main):5:0> ar.combination(2).to_a
=> [["A", "B"], ["A", "C"], ["A", "D"], ["B", "C"], ["B", "D"], ["C", "D"]]
```

このように，Array クラスのオブジェクトは each メソッドで Enumerator のオブジェクトに変換されますから，結局これらの関係は，下の図のようになっているわけです。

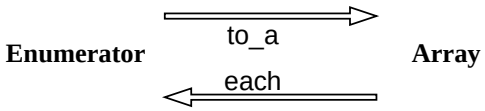


図 1　Enumerator と Array のクラスオブジェクトの変換

## おまけ： Cygwin へのインストール

コンパイルされた Ruby 1.9.0 のバイナリの Cygwin 版はコンパイル時のテストでエラーが出るためか，現時点では公開されていないようです。しかしそれでもたいていのことはできます[*4]。

とりあえず Cygwin に gcc がインストールされていれば，次のようにしてコンパイルしてインストールできるようです。Ruby の言語仕様の拡張を目の当たりに楽しめるチャンスですからやってみてください。

```
./configure --prefix=$HOME/ruby1.9
make
make test
make intall
```

ここでは，ふだん使っている Ruby 1.8.x が消されてしまうのは困るので，インストール先を configure の --prefix オプションで指定しています。ここでは /usr/local/bin/ に 1.8.x があるものとして，1.9 は $HOME/ruby1.9/bin/ にパスを通して実行します。

[*4] もともと 1.9 はまだ開発途中でバグがあって当たり前なわけですから，何かあったら自分で責任をとりましょう。