Ruby 初級者向けレッスン 第 1 回

かずひこ@ネットワーク応用通信研究所

平成 17 年 7 月 11 日

メニュー

- Apache ログで遊ぼう
  - Apache ログってどんなもの？
  - 正規表現で分解しよう
  - クラスを作ろう
  - イテレータで整理しよう

Apache ログ

- ウェブサーバの接続記録
  - 時刻、IP アドレス、リクエスト
  - ステータス、ブラウザなどなど
- ログの例 (実際は一行)

```
10.136.161.73 - - [26/Jun/2005:05:48:37 +0900]
"GET /20040414.html HTTP/1.0" 200 2240 "-"
"DoCoMo/1.0/P505iS/c20/TB/W20H10"
```

前から順に、IP アドレス、(略)、ユーザ、時刻、
メソッド、URI、プロトコル、ステータス、リンク元、
ブラウザ

正規表現

- 文字列のパターンを記述するもの
- 文字列の処理には必ず必要

正規表現の書き方

- /パターン/ と書けば正規表現 (Regexp) オブジェクトを作れる
- スラッシュを含むパターンを記述する場合は「%r」記法を使うのが便利

```
/^\/home\/[^\/]*\/public_html/
  ↓
%r!^/home/[^/]*/public_html!
%r(^/home/[^/]*/public_html)
```

など

正規表現の使い方

- =~ 演算子で文字列と比較する

```
/正規表現/ =~ "文字列"
```

- マッチした場合はマッチした位置のインデックスが、マッチしなかった場合は nil がそれぞれ返る
- 条件式の真偽値に使うのが一般的

```
if /正規表現/ =~ 文字列
  # 文字列が正規表現にマッチした場合の処理
end
```

正規表現の使い方 (2)

- String#scan でマッチした部分を取り出す

```
"文字列".scan(/正規表現/)
#=> マッチした部分の配列
```

通常の文字によるマッチング

- 特殊な記号を使わない文字列からなるパターンの場合、その文字列を含む文字列とマッチする

```
/ABC/ =~ "ABC"      #=> 0 (真)
/ABC/ =~ "AABCC"    #=> 1 (真)
/ABC/ =~ "AABBCC"   #=> nil (偽)
"ABC".scan(/AB/)    #=> ["AB"]
"ABCABC".scan(/AB/) #=> ["AB", "AB"]
```

行頭と行末とのマッチング

- **「」が行頭に「$」が行末にマッチする**

```
/^ABC$/ =~ "ABC"       #=> 0
/^ABC$/ =~ "ABCDEF"    #=> nil
/^ABC$/ =~ "123ABC"    #=> nil

/^ABC/ =~ "ABC"        #=> 0
/^ABC/ =~ "ABCDEF"     #=> 0
/^ABC/ =~ "123ABC"     #=> nil

/ABC$/ =~ "ABC"        #=> 0
/ABC$/ =~ "ABCDEF"     #=> nil
/ABC$/ =~ "123ABC"     #=> 3
```

ある範囲の文字

- **「複数の文字のうちどれか 1 文字に」にマッチする 1 文字分のパターン**
  - /[Ruby]/ ... **「R」「u」「b」「y」のいずれか 1 文字**
  - /[0-9]/ ... **「0」から「9」のいずれかの 1 文字**
  - /[a-z]/ ... **「a」から「z」のいずれか**
  - /[a-zA-Z]/ ... **「a」から「z」あるいは「A」から「Z」のいずれか 1 文字**

```
"a0b1c2d3".scan(/[0-9]/) #=> ["0", "1", "2", "3"]
```

- **「」で範囲外を指定する**
  - [ABC] ... **「A」「B」「C」以外の 1 文字**
  - [0-9] ... **「0」〜「9」以外の 1 文字**


```
"a0b1c2d3".scan(/[^0-9]/) #=> ["a", "b", "c", "d"]
```

- **例**
  - /p[lr]ay/ ... **「play」か「pray」を含む文字列にマッチ**
  - /data[0-9][0-9]/ ... **「data00」「data99」などを含む文字列にマッチ**

任意の文字

- /./ ... **何か 1 文字にマッチ**

```
"abcd".scan(/./) #=> ["a", "b", "c", "d"]
"abcd".scan(/../) #=> ["ab", "cd"]
"abcd".scan(/.b./) #=> ["abc"]
```

特別なパターン

- \s ... **改行、空白、タブにマッチ** (\S **で空白以外**)
- \d ... **数字** (0-9) **にマッチ** (\D **で数字以外**)
- \w ... [0-9A-Za-z_] **と同じ範囲の文字にマッチ** (\W **でそれ以外**)
- \A ... **文字列の先頭にマッチ**
- \z ... **文字列の末尾にマッチ**

繰り返し

- /R?/ ... 0 **個か** 1 **個の「R」にマッチ**
- /R*/ ... **「R」の** 0 **個以上の並び (「」「R」「RR」など) にマッチ**
- /R+/ ... **「R」の** 1 **個以上の並び (「R」「RR」「RRR」など) にマッチ**
- /[0-9]+/ ... **「0」から「9」の** 1 **個以上の並びにマッチ**
- /\d+/ ... **同上**

パターンのグループ化

- **パターンを括弧でくくることで、パターンの繰り返しなどを記述できる**
  - /(ABC)+/ ... **「ABC」の 1 回以上の繰り返し**
  - /(\d\d)+/ ... **二桁の数字の繰り返し**
  - /[0-9]{1,3}(,[0-9}{3})*$/ ... **カンマ区切りの数字**

選択

- /Ruby|Sapphire/ ... **「Ruby」か「Sapphire」にマッチ**

```
"Ruby Sardonix Sapphire".scan(/Ruby|Sapphire/) #=> ["Ruby", "Sapphire"]
```

正規表現のオプション

- /.../i
  - **アルファベットの大文字と小文字を区別しない**
- /.../m
  - **改行を普通の文字として扱う**

後方参照

- **キャプチャとも言う**
- () **で括った部分にマッチしたものを参照する機能**
- **括弧の位置で先頭から順に「**$1**」、「**$2**」、... という変数で取り出す**

```
/(.)(.)(.)/ =~ "abc"
p $1 #=> "a"
p $2 #=> "b"
p $3 #=> "c"
```


「$数字」以外の変数

- **「**$‘**」 ... マッチした部分より前の文字列**
- **「**$&**」 ... マッチした部分の文字列**
- **「**$’**」 ... マッチした部分より後ろの文字列**

```
/C./ =~ "ABCDEF"
p $‘   #=> "AB"
p $&   #=> "CD"
p $’   #=> "EF"
```

例えば...

- /(\S+) - - \[(.+)\]/ **で以下をマッチ**

```
10.136.161.73 - - [26/Jun/2005:05:48:37 +0900] ...

p $1 #=> "10.136.161.73"
p $2 #=> "26/Jun/2005:05:48:37 +0900"
p $’ #=> " ..."
```

正規表現を使うメソッド

- String#sub, String#gsub

  マッチした部分を置換

```
"abracatabra".sub(/a/, "A") #=> "Abracatabra"
"abracatabra".gsub(/a/, "A") #=> "AbrAcAtAbrA"
```

- String#scan

  マッチした部分を取り出す

```
"abracatabra".scan(/.a/) #=> ["ra", "ca", "ta", "ra"]
```

- String#split

  マッチした部分で分割

```
"this is a pen".split(/\s/) #=> ["this", "is", "a", "pen"]
```

- String#slice

  マッチした最初の部分を返す

```
"abracatabra".slice(/.a/) #=> "ra"
```

- Regexp#match

  マッチして MatchData **オブジェクトを返す**

```
re = /(\d+):(\d+)/
md = re.match("Time: 12:34 AM")
p md.class #=> MatchData
p md[0] #=> "12:34" (==$&)
p md[1] #=> "12" (==$1)
p md[2] #=> "34" (==$2)
p md.pre_match #=> "Time: " (==$`)
p md.post_match #=> " AM" (==$')
```

クラスとインスタンス

- **オブジェクトの種類、型**
  - 123 **というオブジェクトのクラスは** Fixnum
  - "123" **というオブジェクトのクラスは** String
- **オブジェクトのことを、クラスと対比させる時にとくにインスタンスと呼ぶ**
  - 123 **は** Fixnum **クラスのインスタンス**
  - "123" **は** String **クラスのインスタンス**
- **クラスからオブジェクトを作る**

```
ary = Array.new
str = String.new("Hello, world.")
```

- **オブジェクトは自分にできること (メソッド) を知っている**

```
str = "Hello, world."
p str.length          #=> 13
p str.upcase          #=> "HELLO, WORLD."
```

- **使えるメソッドはクラスによって決まる**

クラスを作る

- Hello **というクラスを作る**

```
class Hello
  def initialize(myname="Ruby")
    @name = myname
  end

  def hello
    puts "Hello, I am #{@name}."
  end
end
```

- Hello **クラスを使う**

```
bob = Hello.new("Bob")
ruby = Hello.new

bob.hello    #=> Hello, I am Bob.
ruby.hello   #=> Hello, I am Ruby.
```

class 文

- **クラスを定義するにはクラス文を使う**

```
class クラス名
  クラスの定義
end
```

- **クラス名は大文字から始める**

initialize メソッド

- **クラスが初期化される時** (new **でインスタンスが作られる時**) **に呼ばれる**
- **初期化処理をここに記述する**
- initialize **には** new **の引数がそのまま渡される**

```
def initialize(myname="Ruby")
  @name = myname  # インスタンス変数の初期化
end
```

インスタンス変数とインスタンスメソッド

- 「@」で始まる変数をインスタンス変数という
- class 文の中に定義したメソッドはインスタンスメソッドになる
- インスタンス変数はインスタンスメソッドから参照できる

```
def hello
  puts "Hello, I am #{@name}."
end
```

- インスタンスメソッドの中では「self」という変数でインスタンス自身を参照できる

```
def hello2
  self.hello   # hello メソッドの呼び出し
end
```

- レシーバを省略すると、self に対するメソッド呼び出しを行う

```
def hello2
  hello        # self.hello と同じ
end
```

アクセスメソッド

- オブジェクトの外部からインスタンス変数を参照したり、代入したりするメソッドを簡単に作成する

```
class Hello
  attr_reader :name
  ...
end

alice = Hello("Alice").new
p alice.name           #=> "Alice"
```

アクセスメソッドの定義

- attr_reader :name
  - @name を参照できるようになる。以下と同じ

```
def name
  return @name
end
```

- attr_writer :name
  - @name を代入できるようになる。以下と同じ

```
def name=( val )
  @name = val
end
```

- attr_accessor :name
  - @name を参照および代入できるようになる。以下と同じ

```
attr_reader :name
attr_writer :name
```

さあ作ろう

- 今回は読むだけでいいので attr_reader

```
# apache_log.rb
class ApacheLog
  attr_reader :remote_host, :remote_user, ...

  def initialize( str )
    # 正規表現で @remote_host などを求める
  end
end

# 余力のある人は時刻を Time クラスにしよう
```

- こんな感じで使えます

```
require 'apache_log'
log_txt = File.read( 'log1.txt' ) # 1 行だけのログ
log = ApacheLog.new( log_txt )
p log.remote_host # IP アドレスの表示
p log.referer # リンク元の表示
```

イテレータ

- 配列の要素などを繰り返し処理をするもの
- 集めたり、選んだり、見つけたり、いろいろ

イテレータいろいろ

- each
  - イテレータの基本
  - 順にブロックを実行する

```
[1,2,3].each {|i| puts i*2 }
2
4
6
#=> [1, 2, 3] # 返り値は元の配列
```

- collect / map
  - 順にブロックを実行してその値を配列にする

```
[1,2,3].collect {|i| i*2 }
#=> [2, 4, 6]
```

- select / find_all
  - 順にブロックを実行して真のものだけ集める

```
[1,2,3].select {|i| i % 2 == 1 } # 奇数
#=> [1,3]
```

- detect / find
  - 順にブロックを実行して最初に真になったものを返す

```
[1,2,3].detect {|i| i % 2 == 1 } # 奇数
#=> 1
```


前準備

- ログを読み込んで処理するプログラムの雛型

```
require 'apache_log'
logs = []
File.readlines('log2.txt').each do |line|
  logs << ApacheLog.new( line )
end

# logs に対していろいろ操作
```

いろいろやってみよう

- '/index.rdf' へのアクセスは何回？
- 最初の Mac ユーザのアクセスは？
- Google 経由のアクセスは何回？
- 何曜日のアクセスが一番多い？
- ブラウザごとのアクセス数ランキングは？
- などなど

今後の情報源

公式 Web サイト
: http://www.ruby-lang.org/

リファレンスマニュアル
: http://www.ruby-lang.org/ja/man/

日本 Ruby の会
: http://jp.rubyist.net/

Rubyist Magazine
: http://jp.rubyist.net/magazine/