Ruby における正規表現

1. Regexp オブジェクトの生成
2. 正規表現の構文
3. パターンマッチング
4. 置換

正規表現

パターンを文字列にマッチさせる. 意味不明だなぁ

- UNIX コマンド: egrep, awk, sed
- Ruby は、正規表現をかなり拡張している.

正規表現の応用例

- ログファイル内のログイン失敗を見つける
- 電子メールの差出人を全て取り出す.
- ファイル中のIPアドレスを全部取り出す.

Regexp オブジェクトの生成

3種類

```
re = Regexp.new('^¥s*[a-z]')
re = /^¥s*[a-z]/
re = %r|^¥s*[a-z]|
```

オプション

i：正規表現はマッチ時に大文字小文字の区別を行わない

m：複数行モード。正規表現 "." が改行にもマッチするようになる

x：正規表現中の空白(改行も含む)を無視する

o：一番最初に正規表現の評価が行われた時に一度だけ式展開（#{}）を行う

パターンの構文

文字は基本的には自分自身とマッチする. 例外は:

., |, (,), [,], {, }, +, ¥, ^, $, *, ?

エスケープには ¥ を用いる, i.e. ¥| がマッチするのは |

何にでもマッチするメタキャラクタは . です.

アンカーは先頭や末尾にマッチする

^ 行頭 ←―――― 文字列先頭または改行の次

$ 行末

¥A 文字列先頭。改行の有無には影響されない

¥Z 文字列末尾。文字列が改行で終っていればその改行の直前

Regexp エスケープ

特殊なコード

- ¥t タブ
- ¥n ニューライン
- etc.

語境界

- /red/ は “red”, “bred”, “reddened” とマッチ
- /¥bred¥b/ がマッチするのは “red” だけ
- ¥b 文字クラス指定の外では語境界 (¥w と ¥W の間)
- ¥B 非語境界
- /¥brub¥B/ は “ruby” とマッチ. “rub” にマッチせず

文字=非記号

非文字=記号

参考:「正規表現」 http://www.ruby-lang.org/ja/man/html/_C0B5B5ACC9BDB8BD.html

文字クラス

[ ] で指定するもの

いずれかの一文字にマッチ

- [aeiou] は任意の母音とマッチ
- [0123456789] は任意の数字とマッチ
- 特殊文字も、[]内では普通の文字

さらに

- [A-Z] はすべての大文字
- [A-Za-z0-9] は、アルファニューメリック
- [^A-Z] は大文字以外の文字

先頭以外にある `^' はその文字そのものとマッチします。また、先頭、末尾 にある `-' は、その文字そのものとマッチします。

特殊文字

Rubyでは	POSIX Classes
¥d は [0-9]	[:alnum:] は [A-Za-z0-9]
¥D は [^0-9]	[:alpha:] は [A-Za-z]
¥s は [¥t¥r¥n¥f]	[:digit:] は [0-9]
¥S は [^ ¥t¥r¥n¥f]	[:xdigit:] は [0-9A-Fa-f]
¥w は [A-Za-z0-9_]	[:lower:] は [a-z]
¥W は [^A-Za-z0-9_]	[:upper:] は [A-Z]
	[:space:] は [¥t¥r¥n¥f]

選択

垂直線で選択を表す。どちらか一方

- *pattern1|pattern2*

優先順位

空白の個数も重要

- red|blue は “red” または “blue” の一方とマッチ
- red ball|blue sky は “red ball” または “blue sky” とマッチ. “red blue sky” または “red ball sky” とマッチではない

括弧を用いれば纏めることができる

- red (ball|blue) sky

繰り返し

繰返しは、"欲張り." とにかくできるだけたくさんマッチする.

- `re*` 0 回以上
- `re+` 1回以上
- `re?` 0 または1回
- `re{n}` ぴったり n 回
- `re{n,}` n回以上
- `re{n,m}` 最低n回、最大m回

さらに

後方参照

正規表現 ¥1 ¥2 ... ¥n は、後方参照です。*n* 番目の括弧(正規 表現 ( ) グルーピング)にマッチした文字列にマッチします

`/([Rr])uby&¥1ails/` は次のどちらにもマッチ

- Ruby & Rails
- ruby & rails

`/(¥w+) ¥1/` は語の1回繰り返しにマッチ

? は最短一致 — 復習: メタキャラクタ

`<.*>` は欲張り. “<ruby>perl>” にマッチ

`<.*?>` は最短一致. “<ruby>” にマッチ

? は量指定子(quantifier)と名づけられている

マッチング

パターンマッチの演算子は =~

```
re = /[Rr]uby|[Pp]ython/
re =~ "Ruby is better than PHP."
```

マッチのあと、その詳細を取り出すことができる:

data = Regexp.last_match
data.string: 比較した文字列
data.to_s: マッチした部分文字列
data.pre_match: マッチした部分の前の文字列
data.post_match: マッチした部分の後の文字列
data[1]: マッチした第一の () の中味
data[2]: マッチした第二の () の中味
data.captures: () でマッチしたものすべて

Regexp (正規表現クラス) のメソッド

パターンマッチ様々

スライス

復習: 1回以上 / 正規表現の]ではありません

```
“ruby123”[/¥d+/]                 # 123
“ruby123”[/([a-z]+)(¥d+)/,1]     # ruby
“ruby123”[/([a-z]+)(¥d+)/,2]     # 123
r = “ruby123”
r.slice(/¥d+/)                   # 123
r.slice!(/¥d+/)                  # 123, r = “ruby”
```

スプリット

```
s = “one, two, three”
s.split                          # [“one,”, “two,”, “three”]
s.split(‘, ‘)                    # [“one, “two”, “three”]
s.split(/¥s*,¥s*/)               # [“one”,”two”,”three”]
```

置換

置換機能が用意されている

- `sub(re, str)`：`re` に最初にマッチした部分が `str` に置き換わった文字列が値
- `sub!(re, str)`：破壊的置換
- `gsub(re, str)`：`re` にマッチするすべての部分が `str` に置き換わった文字列が値
- `gsub!(re, str)`：破壊的置換

置換の例

Rubyスタイルのコメントを除去

```
line.sub!(/#.*$/, "")
```

数字以外を除去

```
line.gsub!(/¥D/, "")
```

ある単語の置換

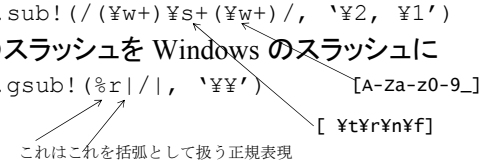
```
line.gsub!(/¥brails¥b/, 'Rails')
```

単語の入替: 例 “John Smith” を “Smith, John” に

```
name.sub!(/(¥w+)¥s+(¥w+)/, '¥2, ¥1')
```

UNIX のスラッシュを Windows のスラッシュに

```
path.gsub!(%r|/|, '¥¥')
```



例

- 次の例を実際に、Ruby で試してみてください。

```
a = Regexp.new('^¥s*[a-z]')
b = /^¥s*[a-z]/
c = %r{^¥s*[a-z]}
a==b
b==c
a==c

string = "spam and eggs"
string =~ /a/
string =~ /spam/
string =~ /foobar/
/a/ =~ string
```

```
irb(main):001:0> a = Regexp.new('^¥s*[a-z]')
=> /^¥s*[a-z]/
irb(main):002:0> b = /^¥s*[a-z]/
=> /^¥s*[a-z]/
irb(main):003:0> c = %r{^¥s*[a-z]}
=> /^¥s*[a-z]/
irb(main):004:0> a==b
=> true
irb(main):005:0> b==c
=> true
irb(main):006:0> a==c
=> true
```

```
def show_regexp(a, re)
  if a =~ re
    "#{$`}<<#{$&}>>#{$'}"
  else
    "no match"
  end
end
```

$がつくとグローバル変数
そしてこれは「特殊変数」

```
show_regexp("spam and eggs", /a/)
show_regexp("spam and eggs", /spam/)
show_regexp("spam and eggs", /foobar/)
```

```
irb(main):196:0> show_regexp("spam and eggs", /a/)
=> "sp<<a>>m and eggs"
irb(main):197:0> show_regexp("spam and eggs", /spam/)
=> "<<spam>> and eggs"
irb(main):198:0> show_regexp("spam and eggs", /foobar/)
=> "no match"
```

```
show_regexp("spam and eggs", /eggs/)
show_regexp("!@%&-_=+", /%&/)
show_regexp("yes | no", /¥|/)
show_regexp("yes (no)", /¥(no¥)/)
show_regexp("are you sure?", /e¥?/)

show_regexp("this is¥nthe time", /^the/)
show_regexp("this is¥nthe time", /is$/)
show_regexp("this is¥nthe time", /¥Athis/)
show_regexp("this is¥nthe time", /¥Athe/)
show_regexp("this is¥nthe time", /¥bis/)
show_regexp("this is¥nthe time", /¥Bis/)
```

```
show_regexp("Price $12.", /[aeiou]/)
show_regexp("Price $12.", /[¥s]/)
show_regexp("Price $12.", /[¥d]/)
show_regexp("Price $12.", /[¥daeiou]/)
a = 'See [Design Patterns-page 123]'
show_regexp(a, /[A-F]/)
show_regexp(a, /[A-Fa-f]/)
show_regexp(a, /[0-9]/)
show_regexp(a, /[0-9][0-9]/)
show_regexp(a, /[0-9][0-9][0-9]/)
show_regexp(a, /¥s/)
show_regexp(a, /¥d¥d¥d/)

show_regexp("aaaaa", /a/)
show_regexp("aaaaa", /a*/)
show_regexp("aaaaa", /a+/)
show_regexp("aaaaa", /a?/)
show_regexp("aaaaa", /a{0,5}/)
show_regexp("aaaaa", /a{0,10}/)
show_regexp("aaaaa", /a{5,10}/)
show_regexp("aaaaa", /a{6,10}/)
show_regexp("aaaaa", /a{6,}/)
show_regexp("aaaaa", /a{3,}/)
show_regexp("aaaaa", /a{3}/)
```

```
a = "The moon is made of cheese"
show_regexp(a, /¥w+/)
show_regexp(a, /¥s.*¥s/)
show_regexp(a, /¥s.*?¥s/)
show_regexp(a, /[aeiou]{2,99}/)
show_regexp(a, /mo?o/)

a = "red ball blue sky"
show_regexp(a, /d|e/)
show_regexp(a, /al|lu/)
show_regexp(a, /red ball|angry sky/)

show_regexp("banana", /an*/)
show_regexp("banana", /(an)*/)
show_regexp("banana", /(an)+/)
show_regexp("banana", /(an)?/)
show_regexp(a, /red (ball|angry) sky/)
a = "the red angry sky"
show_regexp(a, /red (ball|angry) sky/)

show_regexp("Hello world", /(¥w)¥1/)
show_regexp("Hello to the the world", /(¥w+ )¥1/)
```

```
"11:33pm" =~ /(¥d¥d):(¥d¥d)../
puts "The hour is $1, number of minutes is $2"
"11:33pm" =~ /(¥d¥d):(¥d¥d)../
puts "The hour is #$1, number of minutes is #$2"
"8:49am" =~ /(¥d¥d):(¥d¥d)../
puts "The hour is #$1, number of minutes is #$2"
"8:49am" =~ /(¥d?¥d):(¥d¥d)../
puts "The hour is #$1, number of minutes is #$2"

a = "the quick brown fox"
a.sub(/[aeiou]/, '*')
a.gsub(/[aeiou]/, '*')
a.sub(/¥s¥S+/, '')
a.gsub(/¥s¥S+/, '')
name = "Smith, John"
name.sub(/(¥w+), (¥w+)/, '¥2 ¥1')
a.sub(/^./) {|match| match.upcase}
a.gsub(/[aeiou]/) {|match| match.upcase}
vars = Hash.new
vars["name"] = "John Smith"
vars["here"] = "NKU"
```