Ruby
正規表現

Release:2016/10(var2.0.0)

正規表現（パターンマッチ）

Ruby では Perl とほぼ互換性のある正規表現を使用して文字列のパターンマッチを行うことができます。

正規表現を使う

Ruby では正規表現を扱う場合 Regexp クラスを使用します。スクリプト中で、

```
r = Regexp.new("ruby")
```

のように明示的に Regexp のインスタンスを生成することもできますし、

```
/ruby/
```

のようにリテラルで記述することもできます。

正規表現を条件式で使う

if 文などの条件式に正規表現を使用することが出来ます。以下の例は、str の先頭が abc の場合、条件式が真となります。=~は Regexp クラスのメソッドでマッチした場合にそのインデックス(位置)を返却します。

```
if /^abc/ =~ str
  puts "match"
end
```

Perl 風に書く

=~ は String クラスのメソッドでもあるので以下のようにも書けます。

```
if str =~ /^abc/
  puts "match"
end
```

文字にマッチさせる

正規表現では文字そのものを表現するリテラルと、特別な意味を持つメタ文字を使用し特定のパターンにマッチさせることができます。

```
if /abc/ =~ str
  puts "match"
end
```

この例では str 中に abc という文字列が含まれている場合、match という文字列を標準出力へ出力します。このような単純な条件でなく行頭が#で始まるか？ 電話番号が含まれるか？など複雑な形式を表現する場合に、メタ文字を使います。

Ruby で使用できる主要なメタ文字の一覧を以下に示します。

^

　　行頭にマッチします。

$

　　行末にマッチします。

.

　　改行を除く任意の 1 文字にマッチします。

[...]

　　[]内のいずれか 1 文字にマッチします。

[^...]

　　[]内に含まれない 1 文字にマッチします。

*

　　直前の表現の 0 回以上の繰り返しにマッチします。

+

　　直前の表現の 1 回以上の繰り返しにマッチします。

?

直前の表現の 0 回または 1 回の繰り返しにマッチします。

{n}

直前の表現 n 回の繰り返しにマッチします。

{n,}

直前に表現に n 回以上の繰り返しにマッチします。

{n,m}

直前に表現に n 回以上の繰り返しにマッチします。

a|b

a または b にマッチします。

(...)

正規表現をグループ化する。マッチしたテキストは記憶されます。

¥w

英数字と_にマッチします。

¥W

英数字と_以外にマッチします。

¥s

空白文字(スペース、タブ)にマッチします。

¥S

空白文字(スペース、タブ)以外にマッチします。

¥d

数字にマッチします。

¥D

数字以外にマッチします。

繰り返し文字とマッチさせる

メタ文字の*、+、?などを使用して繰り返し文字とマッチさせることができます。*は直前の正規表現の 0 回以上の繰り返しとマッチします。以下の例では、直前の文字 a が 0 回以上繰り返しの後に c という文字が存在する場合にマッチします。

```
/a*c/ =~ "abc" #=> マッチする
/a*c/ =~ "ac" #=> マッチする
/a*c/ =~ "aaaaaac" #=> マッチする
/a*c/ =~ "c" #=> マッチする
/a*c/ =~ "abcccccc" #=> マッチする
/a*c/ =~ "abd" #=> マッチしない
```

+は直前の正規表現の 1 回以上の繰り返しとマッチします(最低でも 1 回以上の繰り返しが必要)。以下の例では、直前の文字 a が 1 回以上繰り返した後に c という文字が存在する場合にマッチします。

```
/a+c/ =~ "abc" #=> マッチしない
/a+c/ =~ "ac" #=> マッチする
/a+c/ =~ "aaaaaac" #=> マッチする
/a+c/ =~ "c" #=> マッチしない
/a+c/ =~ "abccccc" #=> マッチしない
/a+c/ =~ "abd" #=> マッチしない
```

?は直前の正規表現の 0 回、または 1 回の繰り返しとマッチします。以下の例では、直前の文字 a が 0 回または 1 回繰り返した後に c という文字が存在する場合にマッチします。

```
/a?c/ =~ "abc" #=> マッチする
/a?c/ =~ "ac" #=> マッチする
/a?c/ =~ "aaaaaac" #=> マッチする
/a?c/ =~ "c" #=> マッチする
/a?c/ =~ "abccccc" #=> マッチする
/a?c/ =~ "abd" #=> マッチしない
```

数字だけ・アルファベットだけとマッチさせる

文字クラスを使用すると数字だけ、アルファベットだけ、A か B か Z のどれかなど、複数文字中の任意の 1 文字を表現することができます。文字クラスを指定する場合、複数の文字を[]で括ります。

```
/[ABZ]/ =~ "A" #=> マッチする
/[ABZ]/ =~ "Z" #=> マッチする
/[ABZ]/ =~ "Q" #=> マッチしない
```

[]内で-を使用すると文字の範囲を示します。[A-Z]とすれば A から Z まで、[0-9]とすれば 0 から 9 までの数字を表現することができます。

```
/[0-9]/ =~ "5" #=> マッチする
/[0-9]/ =~ "A" #=> マッチしない
/[A-Z]/ =~ "A" #=> マッチする
/[A-Z]/ =~ "5" #=> マッチしない
/[A-Z]/ =~ "a" #=> マッチしない
```

[]内で^を指定するとそれ「以外」を表現することができます。

```
/[^0-9]/ =~ "A" #=> マッチする
/[^0-9]/ =~ "5" #=> マッチしない
```

改行コードを含む文字列にマッチさせる

メタ文字の . に改行コードをマッチさせるには、正規表現式の末尾に m 修飾子を付けて複数行モードにします。

```
/abc.*xyz/m  =~ multiLines    #=> abc と xyz の間に改行があってもマッチする
```

n 番めのマッチを見つける

正規表現のグルーピングを使用し、正規表現内の n 番めのグループにマッチした文字列という処理を記述することができます。正規表現内で()で括ら

れた文字列がグループになり、n 番目の要素を参照する場合、$n と記述します。

```
/1(.)3(.)5(.)/ =~ "123456" #=> $1="2", $2="4", $3= "6"

/([a-z]+),([¥d¥-]+),(¥S+)/ =~ "take,045-111-2234,Yokohama"
  #=> $1="take", $2="045-111-2234", $3="Yokohama"
```

正規表現を使って文字列を置き換える

String#sub メソッド、String#gsub メソッドは文字列の置換を行いますが、置換される文字列の指定に正規表現を使用することができます。

```
str = "Copyright 2001 by TAKEUCHI Hitoshi.".sub(/[A-Za-z]*right/,
"Copyleft")
  #=> "Copyleft 2001 by TAKEUCHI Hitoshi."
```

パターンで区切られたレコードを読む

String#split メソッドは文字列を指定したパターンにより分割し、結果を配列へ格納することができます。そのため、CSV ファイルなど特定のパターンにより区切られたレコードを解析する際に便利です。

```
"001,TAKEUCHI Hitoshi, Yokohama".split(/¥s*,¥s*/) #=> ["001",
"TAKEUCHI Hitoshi", "Yokohama"]
```

マッチした文字列を全て抜き出して配列へ格納する

String#scan メソッドは文字列中で任意のパターンにマッチするものを全て抜き出し、配列へ格納することができます。

```
p "hoge:045-111-2222 boke:045-222-2222".scan(/[¥d¥-]+/)
 #=> ["045-111-2222", "045-222-2222"]
```

正規表現の中にグルーピングの括弧を使うと，それぞれにマッチした文字列の配列を要素とする配列が得られます。

```
p "hoge:045-111-2222 boke:045-222-2222".scan(/(¥S+):([¥d¥-]+)/)

#=> [["hoge", "045-111-2222"], ["boke", "045-222-2222"]]
```

正規表現にコメントを付ける

正規表現の生成時のオプションに x を指定すると、正規表現中の空白文字（スペース、タブ、改行など）を無視し、また、#によるコメントを付けることができます。

```
r = /(¥S+):        # 名前
     ([¥d¥-]+)     # 電話番号
    /x

p "hoge:045-111-2222 boke:045-222-2222".scan(r)
  #=> [["hoge", "045-111-2222"], ["boke", "045-222-2222"]]
```

正規表現内で String 型変数を使う

正規表現内で、String 型変数を使用する際には、#{変数名}の様に、変数を#{}で括ってやる必要が有ります。

```
regEx = "^a"              # 先頭のaにマッチする、正規表現を表すString型変数
p "abc".scan(/regEx/)     #=>[]  正規表現 "regEx" とはマッチしない
p "abc".scan(/#{regEx}/) #=>["a"] 正規表現 "^a" とマッチ
```