第1章

正規表現のいろはにほへと

まずは基本を押さえよう

Shibuya Perl Mongers
小山 浩之　OYAMA Hiroyuki　oyama@cpan.org　http://module.jp/

はじめに

現在Webアプリケーションの実装で利用されているプログラミング言語のほとんどに正規表現が組み込まれています．正規表現を使うことで，ユーザからの入力を検査したり，文字列の中から必要な情報を取り出したり，別の文字列に整形したりといった煩雑な処理をコンパクトかつシンプルに表現することができます．

本特別企画では正規表現について解説していきます．まず本章では正規表現の概要を説明し，その表記法や簡単な使い方を説明します．第2章以降ではJava，PHP，.NETにおける正規表現の利用方法と注意すべきポイントと，具体的な利用例を解説していきます．

正規表現とは？

Webアプリケーションのプログラミングの多くは文字列の操作です．ユーザからの入力や，データベースから取り出した値，データファイルの内容，最終的に出力するHTMLなどなど，さまざまな場面で文字列を相手にプログラミングする必要があります．私たちは文字列に対して

- 比較
- 抽出
- 置換
- 追加
- 削除
- 分割

といった処理を頻繁に行っています．プログラミング言語のプリミティブな機能だけでこれらの処理を行うと非常に多くの手数を要し，予期しないバグを作り込む原因になってしまいます．しかしプログラミング言語が正規表現をサポートしていれば事情が変わってきます．正規表現によって文字列に対するさまざまな操作をシンプルに記述できるようになり，高度で複雑な処理でさえも高い抽象度でシンプルに記述することができます．正規表現は文字列の処理において，効率的で柔軟で非常に強力で重要なツールなのです．

たとえばAmazon.co.jpの商品紹介ページのHTMLから売上ランキングを抜き出したいとき，正規表現内蔵言語の代表格であるPerlを使うと，**リスト1**のようにたった2行のコードで記述することができます．

リスト1は商品紹介ページのHTMLを保持した変数$htmlから正規表現でランキングを抽出し，その値を変数$rankingにコピーしています．

もし正規表現を使わない場合は，

❶\nAmazon.co.jp売上ランキング： \n という文字列を検索
❷それに続く数字もしくは，（カンマ）の文字列を記憶
❸それに続く \nを発見したら終了

と複数のステップを踏む必要があるでしょう．さらに言語によっては一時変数や処理ステップを記憶する変数を用意するといった，処理の本質とは遠い煩雑な手

▼リスト1　Perlで Amazon.co.jp売上ランキングを抽出

```
$html =~ m|\n<b>Amazon.co.jp売上ランキング： </b>\n([\d,]+)\n|;
$ranking = $1;
```

続きが必要になる場合もあります．このように正規表現は文字列処理にかかる面倒な処理を高度に抽象化して確実に表現する，文字列処理に特化した言語と言えます．

プログラミング言語から正規表現を利用する方法は，その言語によってさまざまな手段が用意されています．Javaの場合は最近の実装では標準のクラスライブラリとして利用できますし，サードパーティ製の正規表現パッケージを別途利用することもできます．それとは異なりPerlの場合はプログラミング言語に組み込まれた演算子でシームレスに正規表現を利用することができます．他にも関数として組み込まれている場合などもあり，プログラミング言語によって正規表現を利用する手続きは異なります．

また，モダンなプログラミング言語に用意されている正規表現の多くは「Perlバージョン5互換」を謳(うた)っており，その実装や利用できる機能はある程度似ています．以前のように「ここでPerlの正規表現が使えたらどんなに楽か…」といった不自由な思いをすることは少なくなってきていますが，それでも細かいところで挙動の違いがあったり独自の機能拡張が含まれている場合があったり，利用できない機能があったりするのが実状ですので，それぞれのプログラミング言語に実装されている正規表現の特徴を捉えておく必要があります．

プログラミング言語「正規表現」

それでは正規表現の実装のデファクトスタンダードであるPerlを例に見ながら，正規表現の基本的な記法や利用方法を説明します．ここでは正規表現の一般的な記法を理解してもらうことを目的としているため，最新のPerlバージョン5.8系で実装されている正規表現の拡張機能についてはあえて触れませんのでご了承ください．

利用方法

正規表現は文字列に対して，先述したように比較・抽出・置換・追加・削除・分割といった処理を行うことができます．

これらの処理を大ざっぱに一般化すると

- マッチング
- 置換
- 分割

という3種類の処理に分類できます．正規表現の実装の多くで，この3種類の処理のためのインタフェースが用意されています．Perlの場合にはマッチングを行うm//演算子，置換を行うs///演算子，分割を行うsplit演算子が用意されています[注1]．

マッチング

特定の変数に保持している文字列にマッチングを行う場合，=~または!~のバインディング演算子を使用します．たとえば変数$textに格納した文字列にマッチングを行う場合はリスト2-❶のように，

変数名 バインディング演算子 正規表現（m//演算子）

と記述します．この際=~はマッチする場合に真，マッチしない場合は偽の値を返します．!~はその逆でマッチする場合に偽，マッチしない場合に真を返します．

m//演算子は正規表現をくくるためにデリミタ/（スラッシュ）を必要としますが，このデリミタは任意の文字に変更することができます．たとえばHTMLタグを含む文字列を正規表現で表す場合，デリミタが/のままだとリスト3のように\（バックスラッシュ）でいちいちエスケープする必要があります．この場合デ

▼リスト2　Perlでのマッチング

```
#!/usr/bin/perl
use strict;

my $text = "Hello World!";
if ($text =~ m/Hello/) { …❶
    # $textはHelloを含んでいる.
}
else {
    # $textはHelloを含んでいない.
}
```

注1）もちろんマッチングの結果として文字列を分割することもできます．さらにPerlにはこの他にも正規表現オブジェクトを生成するqr//演算子や，tr///演算子など，関連する演算子が用意されているのですがここでは割愛します．

Java, PHP, .NETではじめる正規表現

文字列処理はこれでバッチリ！

▼表1　一般的なメタ文字や文字クラス

メタ文字	役割
^	文字列の先頭を示す
$	文字列の末尾を示す
.	任意の1文字にマッチ
\t	タブ文字にマッチ
\n	改行文字にマッチ
\r	キャリッジリターンにマッチ
\s	スペース・タブ・改行・キャリッジリターンなどの空白文字にマッチ
\w	a～z，A～Z，0～9の英数字と_（アンダースコア）にマッチ
\d	0～9の数字にマッチ
[...]	...のいずれかの文字にマッチ（例：[abc]はa，b，cのいずれかにマッチ）
\p{Prop}	Propで指定したUnicodeプロパティ・スクリプト・ブロックにマッチ
\S	\s以外の文字にマッチ
\W	\w以外の文字にマッチ
\D	\d以外の文字にマッチ
[^...]	...以外の文字にマッチ（例：[^abc]はa，b，c以外の文字にマッチ）
\P{Prop}	Propで指定したUnicodeプロパティ・スクリプト・ブロック以外の文字にマッチ
(...)	グループ化とキャプチャ
(?:...)	グループ化のみ
|	or条件（例：正規表現(Perl|Ruby)はPerlとRubyにマッチ）
?	0もしくは1つ（例：A?pacheはApacheとpatchのいずれかにマッチ）
*	0以上の繰り返し（例：\d*は1や12345や，長さ0の空の文字などにマッチ）
+	1以上の繰り返し（例：\d+は1や12345などにマッチ）
{n}	n個の繰り返し（例：\d{3}は3桁の数字にマッチ）
{n,}	n個以上の繰り返し（例：\d{2,}は2桁以上の数字にマッチ）
{n,m}	n個以上m個以下の繰り返し

▼リスト3　デフォルトのデリミタ

```
#!/usr/bin/perl
use strict;

my $text =~ m/<head><title>Hello World!<\/title><\/head>/;
```

▼リスト4　デリミタを|に変更

```
#!/usr/bin/perl
use strict;

my $text =~ m|<head><title>Hello World!</title></head>|;
```

リミタを/以外の文字，たとえばリスト4のように|や#などの文字に変更すれば，/をエスケープする必要はなくなります[注2]．

……置換

変数に保持している文字列を正規表現で置換したい場合はs///演算子を使用します．たとえば変数$textに格納したIISという文字列をApacheに置換する場合は，リスト5のように記述します．

マッチする部分が複数ある場合は最初にマッチした箇所で置換を行って処理を終了します．マッチするすべての箇所を置換したい場合はリスト6のようにgオプションを指定することでグローバルに置換するように動作が変更されます．マッチングと置換で使用できるオプションについては後記します．

……メタ文字

マッチングと置換の例を見てきましたが，ここまでの正規表現は単純なもので，そのパワーはまったく現れていません．正規表現には特定の文字種に一致させたり，文字や文字列の出現回数を指定したり，複数の条件を指定したりするためのメタ文字が用意されています．使用できるメタ文字は正規表現の実装によって異なりますが，一般的なものとして表1に示すようなメタ文字があります．

これらのメタ文字を組み合わせることで，たとえば

```
m/^\d{3}-?(\d{2}|\d{4})$/
```

は

❶先頭から（^）
❷続いて数字3文字（^\d{3}）
❸続いて省略可能な-（-?）
❹続いて2または4文字の数字（(\d{2}|\d{4})）
❺続いて末尾（$）

に一致する日本の郵便番号の文字列を表します．

文字の繰り返しを指定する*や+などの量指定子は正規表現の実装によって動作が異なるので注意が必要です．

たとえばPerlでは，

注2）他にもデリミタを(と)や{と}といった，対応する括弧で指定することもできます．

続きを読む

といった文字列からHTMLタグだけを取り除き「続きを読む」だけを取り出したい場合，リスト7のような正規表現ではHTMLタグ以外の文字列まで取り除いてしまいます．リスト7を実行すると変数$textには文字列「.」（句点）だけが格納されます．これは<.+>という正規表現が「<に続く任意の文字1文字以上に『できるだけ多く』マッチした後に>」という文字列を示しているためです．例に挙げた文字列とこの正規表現を照らし合わせると図1のような範囲でマッチしてしまっています．

期待する動作は図2のような「<に続く任意の文字1文字以上に『最小限に』マッチしたあとに>」という文字列です．

図2のようにマッチさせる場合は*や+といった「欲張り」な量指定子ではなく，リスト8のように*?や+?の「非欲張り」な量指定子を使用します．リスト8を実行すると変数$textは文字列「続きを読む.」が格納されます．

また欲張りな量指定子を使用した場合でもリスト9のようにマッチする文字を明示的に指定し，目的を達成することもできます．リスト9は「<に続く『>以外の文字列』にマッチした後に>」という正規表現になります．

●Unicodeプロパティ

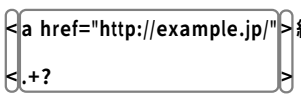
PerlやJavaや.NETなどUnicode対応の正規表現エンジンでは，Unicodeプロパティやスクリプトもしくはブロックを\p{Prop}というメタ文字で指定できます．これを使用することにより，表2のような文字の特性（プロパティ）や，言語別の書記体系（スクリプト）や，文字コード上の範囲（ブロック）を正規表現で指定することができます．

たとえばUnicodeプロパティを用いて一般的な日本語の文字列にマッチする正規表現を記述すると，リスト10のように複数のプロパティとスクリプトの組み合わせになります．ここで使用しているグルーピング用括弧(?:...)については後記します．

これらUnicodeプロパティを用いたマッチングは非常に便利ですが，正規表現の実装によっては基本的なプロパティのみのサポートでスクリプトやブロックをサポートしないもの，プロパティとブロックのみをサポートするものなど，環境によっては利用できないものがあるので注意が必要です．また，ブロックの指定方法はPerlやJavaでは\p{InLatin1

図1●欲張りな量指定子でマッチする範囲

図2●非欲張りな量指定子でマッチする範囲



▼リスト5　置換

```
#!/usr/bin/perl
use strict;

my $text = "IISのWebサーバを管理する.";
$text =~ s/IIS/Apache/;  # 「ApacheのWebサーバを管理する.」に置換
```

▼リスト6　グローバルに置換

```
#!/usr/bin/perl
use strict;

my $text = '!NAME!さんこんにちは.'.
           '今日は!NAME!さん限定でお得な情報をおしらせします.';
$text =~ s/!NAME!/正規表現/g;
```

▼リスト7　欲張りな量指定子

```
#!/usr/bin/perl
use strict;

my $text = '<a href="http://example.jp/">続きを読む</a>.';
$text =~ s/<.+>//g;
```

▼リスト8　非欲張りな量指定子

```
#!/usr/bin/perl
use strict;

my $text = '<a href="http://example.jp/">続きを読む</a>.';
$text =~ s/<.+?>//g;
```

▼リスト9　否定文字クラスによるマッチ

```
#!/usr/bin/perl
use strict;

my $text = '<a href="http://example.jp/">続きを読む</a>.';
$text =~ s/<[^>]+>//g;
```

▼リスト10　一般的な日本語にマッチする正規表現

```
m/(?:(?:\p{Hiragana}|\p{Katakana}|\p{Han}|\p{Latin}|\p{Common})(?:\p{Inherited}|\p{Me}|\p{Mn})?)+/
```

Java, PHP, .NETではじめる正規表現

文字列処理はこれでバッチリ！

Supplement}など接頭子Inを使用するのに対し，.NETでは\p{IsLatin1Supplement}など接頭子Isを使用するといった差異がある点にも注意が必要です．利用できるUnicodeプロパティ・スクリプト・ブロックの種類とその指定方法についてはそれぞれの環境のドキュメントを参照してください[注3]．

...オプション

正規表現の動作はオプションによって変更することができます．一般的なオプションを表3に示します．

▼表2　一般的なUnicodeプロパティの指定

プロパティ	性質
\p{L}	表示可能な文字
\p{M}	他の文字と組み合わせて使用する文字
\p{Z}	区切り文字
\p{S}	記号
\p{N}	数字
\p{P}	句読点
\p{C}	L，M，Z，S，N，P以外のすべての文字
\p{Ll}	小文字
\p{Lu}	大文字
\p{Lo}	ヘブライ語や日本語などの大文字と小文字を持たない文字
\p{Sm}	数学記号
\p{Sc}	通貨記号
\p{Cc}	ASCIIとLatin-1の制御文字
スクリプト	**対象**
\p{Katakana}	カタカナ
\p{Hiragana}	ひらがな
\p{Han}	漢字
\p{Latin}	英数字
\p{Common}	スペースや句読点など

▼表3　正規表現オプション

オプション	用途
i	大文字と小文字を無視する
g	文字列全体の複数箇所でマッチングする
m	行単位でマッチングする
s	改行を含む文字列全体でマッチングする
x	正規表現中の空白を無視し，コメントの記述を許可する

正規表現のオプション指定方法は実装やAPIによって異なるので注意が必要です．

通常，正規表現m/abc/は文字列abcにだけマッチして文字列ABCにはマッチしませんが，m/abc/iとオプションiを付与した場合はABCやAbcのように大文字と小文字の違いを無視してマッチします．

オプションmを付与すると行単位でマッチングを行うようになります．^と$は通常「文字列全体の先頭と末尾」を表すメタ文字ですが，オプションmを付与すると「各行の行頭と行末」に役割が変化します．

これに対しオプションsは改行を含む文字列を1本の文字列として扱いたい場合に使用します．オプションsの場合は任意の文字を示すメタ文字．（ピリオド）が示す文字が，「改行文字\nを含む任意の文字」に拡張されます．

...キャプチャ

マッチした文字の一部分を取り出したり，置換結果に埋め込みたい場合には正規表現のキャプチャを使用することができます．リスト11を実行するとキャプチャ用括弧（と）で指定した範囲にマッチした文字列を取り出すことができます．Perlの場合キャプチャ結果は特殊変数$1で参照できます．

キャプチャ用括弧が複数ある場合は，括弧の出現順に応じて$1，$2，$3，$4…と対応した番号の変数に結果がセットされます．リスト12はaタグから参照先のURLとマークアップされた文字を抜き出して整形する例です．実行するとマッチした文字列は図3のようにキャプチャされて，変数$textは「Here (url:

▼リスト11　キャプチャの例

```
#!/usr/bin/perl
use strict;

my $text = 'Name: hiroyuki oyama';
if ($text =~ m/^Name:\s+(.+)$/) {
    my $value = $1;   # 「hiroyuki oyama」を取り出す
}
```

▼リスト12　複数のキャプチャと置換

```
#!/usr/bin/perl
use strict;

my $text = '<a href="http://example.jp/">Here</a>'
$text =~ s|<a href="(.+?)">(.+?)</a>|$2 (url: $1)|;
```

注3）Perlの場合はperldoc perlunicodeもしくはhttp://www.perldoc.com/perl5.8.2/pod/perlunicode.htmlを参照してください．

図3●複数のキャプチャ

http://example.jp/)」に置換されます．

キャプチャ用括弧はグルーピングにも使用されます．もしキャプチャを必要とせずに単にグルーピングしたい場合はリスト13のように(?:...)でグルーピングします．リスト13はaタグとimgタグの参照先URLを取り出す例で，この正規表現は

❶<
❷ ❶に続く文字列a hrefもしくはimg src（キャプチャしない）
❸ ❷に続く文字列="
❹ ❸に続く1文字以上で最小限の任意の文字（キャプチャする）
❺ ❹に続く"

に繰り返しマッチします．❷の括弧(?:...)はキャプチャせずグルーピングのみを行い，❹のキャプチャ用括弧内にマッチした文字列が特殊変数$1にセットされます．リスト13を実行すると「http://example.jp/」と「http://img.example.jp/path/to/file.png」が出力されます．

...分割

文字列を特定の文字や文字列で分割したい場合にも正規表現が利用できます．Perlの場合はsplit演算子を使用し，リスト14のように文字列の境界を正規表現で指定して分割結果の文字列を配列として取得することができます．リスト14はpasswordファイルの1エントリを格納した変数$textを，デリミタ:で分割し，結果を配列@userに格納します．このコードを実行すると

```
useradd -c 'System Administrator' -d /var/root -g 0 -s bin/sh -u 0 root（実際は1行）
```

という文字列を出力します．

split演算子は第1引数に文字列の分割位置を指定する正規表現，第2引数に分割対象の文字列を指定し使用します．また，分割結果がn番目まで必要な場合は，リスト15のように第3引数に分割し取り出したい要素数を指定します．リスト15は変数$textから:で分割した3つの要素を取り出し配列@userに格納しています．

▼リスト13　キャプチャしない括弧

```
#!/usr/bin/perl
use strict;

my $text = "<a href=\"http://example.jp/\">Read more...</a>\n".
           "<img src=\"http://img.example.jp/path/to/file.png\">\n";
while ($text =~ m/<(?:a href|img src)="(.+?)"/g) {
    my $url = $1;
    print $url, "\n";
}
```

▼リスト14　文字列の分割

```
#!/usr/bin/perl
use strict;

my $text = "root:*:0:0:System Administrator:/var/root:bin/sh";
my @user = split m/:/, $text;
printf "useradd -c '%s' -d %s -g %d -s %s -u %d %s\n",
    $user[4], $user[5], $user[3], $user[6], $user[2], $user[0];
```

▼リスト15　n番目の要素まで分割

```
#!/usr/bin/perl
use strict;

my $text = "root:*:0:0:System Administrator:/var/root:bin/sh";
my @user = split m/:/, $text, 3;
printf 'User %sのUIDは %dです. ', $user[0], $user[2];
```

まとめ

正規表現の実装のデファクトスタンダードであるPerlを例に，正規表現の基本的な利用方法を説明しました．正規表現の実装と利用方法は個々のツールや環境によって差異があります．それでもPHPやRubyやPython，Java や.NETといった現在利用できるモダンな環境であれば，APIや微妙な挙動や独自の拡張といった差異はあるものの，基本となる利用方法や考え方はすべて同じものです．故に正規表現を文字列表現や文字列操作における基本ツールとして身につけておけば，たとえ環境が変わっても同じように使いこなすことができるでしょう．Wb