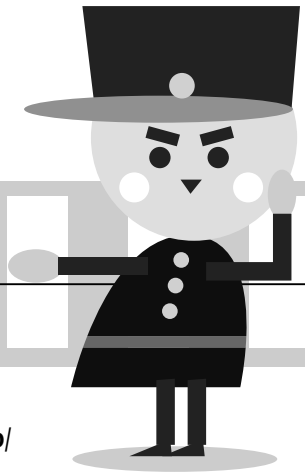
第3章

PHP編



pregとereg系関数を学ぶ

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

はじめに

現在PHPではpreg系関数，ereg系関数，mb_ereg系関数という，それぞれ実装の異なる3種類の正規表現ライブラリが利用できます．本章ではPerl 5.005互換の正規表現を実装するPCREライブラリをベースとしたpreg系関数を中心に，IEEE 1003.2ベースのereg系関数との違いやその利用方法を解説します．

PHPの方言

PHPで利用できる正規表現にはPerlバージョン5.005の正規表現の機能を再現したPCRE（Perl-compatible regular expression library）ベースのpreg系関数，grepなどで採用されているIEEE 1003.2（POSIX.2）準拠のライブラリ関数を使用したereg系関数，マルチバイト文字による文字クラスなどをサポートしたmb_ereg系関数といった実装の異なる3種類のインタフェースが利用できます．

PHPらしくそれぞれ別々の関数を利用することになりますが，正規表現を利用したマッチング・置換・メタ文字による文字の指定・オプションによる動作の設定・キャプチャなどなど，第1章で解説した一般的な正規表現の利用方法は変わりません．しかし利用できるメタ文字や機能や制限事項についてはそれぞれ異なっているので注意が必要です．特にマルチバイト対応のmb_ereg系関数は現在実験的に導入されている状態ですので，将来的にAPIや機能が変更・削除される可能性もありこちらも利用に際して注意が必要です．

現状では特に理由がない限り性能面，機能面，仕様の安定性，異なるプラットフォームでの互換性などから見てpreg系関数を利用するのが無難な状態ですので，本章ではpreg系関数を中心に解説します．

マッチング

変数$textに格納した文字列にpreg系関数でマッチングを行うには，リスト1のように記述します．同様にereg系関数を利用する場合はリスト2のように記述します．

preg系・ereg系関数ともに第1引数の文字列で正規表現を指定する点は同じですが，preg系関数では正規表現を / などのデリミタでくくった文字列で指定する必要があります．

▼リスト1　preg系関数でのマッチング

```
<?php
$text = 'Hello World!';
if (preg_match("/Hello/", $text)) {
    // $textはHelloを含んでいる
} else {
    // $textはHelloを含んでいない
}
?>
```

▼リスト2　ereg系関数でのマッチング

```
<?php
$text = 'Hello World!';
if (ereg("Hello", $text)) {
    // $textはHelloを含んでいる
} else {
    // $textはHelloを含んでいない
}
?>
```

▼リスト3　preg_replace関数での置換

```
<?php
$text = 'www.example.com';
$replaced = preg_replace('/\.com/', ".jp", $text);

echo $replaced, "\n";  // "www.example.jp"
?>
```

▼リスト4　ereg_replace関数での置換

```
<?php
$text = 'www.example.com';
$replaced = ereg_replace('\.com', ".jp", $text);

echo $replaced, "\n";  // "www.example.jp"
?>
```

▼表1　preg系関数（PCREベース）

戻り値	関数
int	preg_match(string regex, strig src [, array matches [, int flags]])
int	preg_match_all(string regex, string src, array matches [, int flags])
array	preg_grep(string regex, array src)
mixed	preg_replace(mixed regex, mixed replacement, mixed src [, int limit])
mixed	preg_replace_callback(mixed regex, mixed callback, mixed src [, int limit])
array	preg_split(string regex, string src [, int limit [, int flags]])
array	preg_grep(string regex, array inputs)
string	preg_quote(string src [, string delimiter])

▼表2　ereg系関数（IEEE 1003.2ベース）

戻り値	関数
bool	ereg(string regex, string src [, array capture])
string	ereg_replace(string regex, string replacement, string src)
array	split(string regex, string src [, int limit])
bool	eregi(string regex, string src [, array capture])
string	eregi_replace(string regex, string replacement, string src)
array	spliti(string regex, string src [, int limit])

...置換

preg系関数で置換を行う場合はリスト3のようにpreg_replace関数を使用します．preg_replace関数の第1引数に正規表現，第2引数に置換後の文字列，第3引数に対象の文字列を指定します．置換結果はpreg_replace関数の戻り値で取得できます．

同様にereg系関数で置換を行う場合はリスト4のようにereg_replace関数を使用します．

リスト1とリスト2，リスト3とリスト4をご覧いただくとpreg系関数とereg系関数の利用方法は，正規表現の指定方法以外は同じに見えます．しかし表1と表2で示すとおりpreg系にはマッチした文字列を引数に，任意の関数を実行して置換できるpreg_replace_callback関数が存在したり，ereg系関数では正規表現のオプションの指定はなく別々の関数で動作を切り替えるといった差異があることがわかります．各関数の詳細についてはPHPマニュアルの「Perl互換の正規表現関数」および「正規表現（regex）関数（POSIX拡張サポート）」を参照してください[注1]．

...メタ文字

preg系関数は第1章表1に挙げた一般的な正規表現のほとんどのに加え，Perlバージョン5.005で実装されていた正規表現の多くを利用できます．ただし\p{Prop}といったUnicodeプロパティ・スクリプト・ブロックなどのUnicode関連の機能は利用できません．

注1）http://www.php.net/manual/ja/ref.pcre.php，http://www.php.net/manual/ja/ref.regex.php

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！

もちろん現在の正規表現で一般的になりつつあるグループ化のみの括弧(?:...)や，(?=...)や(?!...)といった「先読み」や，(?<=...)と(?>=...)の「後読み」はサポートしています．リスト5は否定の先読みを利用して，「Message: 」に続く文字列がhoge以外の英数字の場合にマッチします[注2]．

▼表3　ereg系関数のメタ文字

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
[...]	...のいずれかの文字にマッチ
[^...]	...以外の文字にマッチ
[[:alpha:]]	a〜z，A〜Zのアルファベットにマッチ
[[:digit:]]	0〜9の数字にマッチ
[[:alnum:]]	0〜9，a〜z，A〜Zの英数字にマッチ
[[:xdigit:]]	0〜9，a〜z，A〜Fの16進数で使用する英数字にマッチ
[[:lower:]]	小文字のアルファベット
[[:upper:]]	大文字のアルファベット
[[:punct:]]	記号にマッチ（!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~）
[[:graph:]]	英数字と記号にマッチ（alnumとpunct）
[[:print:]]	表示可能な文字にマッチ（graphと\x20（スペース））
[[:space:]]	空白文字にマッチ（\t\n\v\f\rと\x20（スペース））
[[:blank:]]	スペースとタブ文字にマッチ
[[:cntrl:]]	制御コードにマッチ
[[:<:]]	ワード境界の先頭
[[:>:]]	ワード境界の末尾
(...)	グループ化とキャプチャ
|	or条件
?	0もしくは1つ
*	0以上の繰り返し
+	1以上の繰り返し
{n}	n個の繰り返し
{n,}	n個以上の繰り返し
{n,m}	n個以上m個以下の繰り返し

またpreg系関数のメタ文字\sは，\t，\n，\v，\f，\r，\x20（スペース）にマッチしますが，\vをサポートしないPerlとは\sがマッチする対象が異なる場合があります．他にも細かな部分で差異や制限または拡張されている箇所もあるので，詳しくはPHPマニュアルの「Perl互換の正規表現関数」の「パターン構文」を参照してください[注3]．

ereg系関数はさらに事情が異なり第1章表1で示した\dや\sといったメタ文字は利用できず，[[:digit:]]や[[:space:]]といったPOSIX文字クラスと，[[:<:]]などの拡張文字クラスのみが利用できます[注4]（表3）．またグループ化のみの括弧や前後読みなどのモダンな機能は利用できません．

...オプション

preg系関数でのオプションは，正規表現文字列の中で/regex/iや/regex/xのように指定します．利用できるオプションと役割は表4のとおりです．またpreg_match関数・reg_match_all関数・reg_split関数では，第4引数で固有のオプションを指定して戻り値の形式を変更することができます．この戻り値を変更するオプションについてはPHPマニュアルの「Perl互換の正規表現関数」を参照してください．

前記したとおりereg系関数には正規表現のオプションはなく，大文字と小文字を区別する場合には関数eregやereg_replace，大文字と小文字を無視する場合には関数eregi，eregi_replaceというふうに関数を使い分けます．

▼リスト5　preg系関数で否定の先読み

```
<?php
$text = 'Message: UREEEEE';

// hoge以外で始まる英数字の文字列にマッチ
if (preg_match('/^Message: (?!hoge)\w+$/', $text)) {
    echo 'OK', "\n";
}
?>
```

注2）(?!(?:hoge){1,64}$)\w+$のようにして，hogeが65回以上繰り返される場合は「よほどhogeを入力したかった」と見てマッチさせることもできます．いや，ホントどうでもいいんですが．

注3）http://www.php.net/manual/ja/pcre.pattern.syntax.php

注4）第1章では説明していませんが，.NETを除くPerl，Java，PHPのpreg系関数はともに，POSIX文字クラスと[[:^alpha:]]などの否定文字クラスが利用できます（ただしJavaの場合は\p{Alpha}や\P{Alpha}と指定します）．

...キャプチャ

マッチした文字列の一部分を取り出したり置換結果に埋め込みたい場合はキャプチャを使用します．マッチングの際のキャプチャ結果はリスト6のようにpreg系・ereg系関数ともに，第3引数で指定したarray型変数にキャプチャ結果を格納し参照することができます．

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$array[1]，$array[2]，$array[3]…と取得したいインデックス番号を与えて個別にアクセスできます．

関数preg_replaceやereg_replaceでの文字列置換の際にキャプチャを利用する場合は，リスト7のようにキャプチャ結果を埋め込みたい位置にキャプチャ用括弧の出現順に応じて$1，$2，$3…もしくは\1，\2，\3…の文字を指定します．リスト7を実行すると図1のように出力します．

preg系関数では置換時のキャプチャ結果の参照に$1と\1の両方の形式で指定できますが，ereg系関数では\1の形式だけがサポートされています．またpreg系・ereg系関数ともに置換文字列を"（ダブルクォート）で与える場合は，\\2 (url: \\1)のように\（バックスラッシュ）でエスケープする必要があります．

図1●リスト7の実行結果

```
% php ex7.php
preg: Here... (url: http://www.example.jp/)
ereg: Here... (url: http://www.example.jp/)
```

▼表4　preg系関数のオプション

オプション	役割
//i	大文字と小文字を無視する
//m	行単位でマッチングする
//s	.（ドット）を改行文字にもマッチさせる
//x	正規表現中の空白やコメントを許可する
//e	preg_replace関数の置換結果をPHPの式として評価し，その結果で置換する
//A	先頭行からのみにマッチするアンカーマッチモードにする（\Aでも可）
//D	$（ダラー）を文字列の末尾のみにマッチさせる
//S	与えられた正規表現をより詳細に最適化します
//U	+や*などの欲張りな量指定子を，非欲張りな量指定子に変更する
//X	\（バックスラッシュ）と文字の組み合わせがメタ文字として存在しない場合エラーにする
//u	UTF-8の文字列としてマッチする

▼リスト6　preg系関数，ereg系関数でのキャプチャ

```
<?php
$text = 'Name: Regular Expressions';
if (preg_match('/^(\w+):\s+(.+)$/', $text, $capture)) {
    $key   = $capture[1];
    $value = $capture[2];
    echo "preg: key   = $key\n";
    echo "preg: value = $value\n";
}

if (ereg('^([[:alnum:]_]+):[[:space:]]+(.+)$', $text, $capture)) {
    $key   = $capture[1];
    $value = $capture[2];
    echo "ereg: key   = $key\n";
    echo "ereg: value = $value\n";
}
?>
```

▼リスト7　preg系関数，ereg系関数でのキャプチャ結果による置換

```
<?php
$text = '<a href="http://www.example.jp/">Here...</a>';

$replaced = preg_replace('|<a href="(.+?)">(.+?)</a>|', '$2 (url: $1)', $text);
echo "preg: $replaced\n";

$replaced = ereg_replace('<a href="([^"]+)">([^<]+)</a>', '\2 (url: \1)', $text);
echo "ereg: $replaced\n";
?>
```

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！

利用例

それではWebアプリケーションに応用できるような例題を使って，PHPの正規表現をより具体的に見ていきましょう．ここからはサポートしているメタ文字の差異や柔軟性からpcre系関数のみを取り上げます．

...データのチェック

正規表現で郵便番号をチェックしてみましょう．現在利用されている郵便番号は数字7桁で表され，3桁目と4桁目の間に－（ハイフン）を入れます．また，平成10年1月以前の旧郵便番号には5桁のものと3桁のものがあります．現在利用されている7桁の郵便番号を入力させる場合が多いと思いますが，この例では第2章と同様に

- 154
- 350-11
- 350-1106
- 35011
- 3501106

といったパターンにマッチする正規表現

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

を用意します（リスト8）．

ereg系関数を使用する場合は

```
^[0-9]{3}(-?([0-9]{2}|[0-9]{4}))?$
```

といった正規表現を使用することになります．

...ひらがな・カタカナ・漢字

preg系とereg系関数ではマルチバイト文字による文字クラスの指定やUnicodeプロパティなどがサポートされていないため，文字境界や文字コードブロックを意識しながら正規表現を使用する必要があります．

リスト9はEUC-JPの文字列から，英数字・ひらがな・カタカナ・漢字の各文字種ごとのブロックを抽出するサンプルです．

リスト9を実行すると

```
% php ex9.php
漢字
カナ
混
じりの
文章
を
input
します
```

▼リスト8　preg系関数での郵便番号のチェック

```
<?php
$text = $argv[1];

if (preg_match('/^\d{3}(?:-?(?:\d{2}|\d{4}))?$/', $text)) {
    echo "$text is postal code\n";
} else {
    echo "$text is not postal code\n";
}
?>
```

▼リスト9　preg系関数での文字種ごとの抽出

```
<?php
$text = '漢字カナ混じりの文章をinputします';  // EUC-JP
$pattern = '/(
    [\x21-\x7E]+                    | # Latin-1
    (?:\xA4[\xA1-\xF3])+            | # Hiragana
    (?:\xA5[\xA1-\xF6])+            | # Katakana
    (?:[\xB0-\xF4][\x00-\xFF])+     # Kanji
)/x';

if (preg_match_all($pattern, $text, $capture)) {
    echo join("\n", $capture[1]), "\n";
}
?>
```

▼リスト10　preg系関数でのURLのマークアップ

```
<?php
$text = 'PHPのマニュアルはhttp://www.php.net/manual/で参照できます．';

$replaced = preg_replace('|(https?://[-_.!~*\'()a-zA-Z0-9;/?:@&=+$,%#]+)|',
    '<a href="$1">$1</a>', $text);
echo $replaced, "\n";
?>
```

図2●リスト10の実行結果

```
% php ex10.php
PHPのマニュアルは<a href="http://www.php.net/manual/">http://www.php.net/manual/</a>で参照できます．
```

と分割した文字列を出力します．

テキスト中のURLをハイパーリンクに置換する

WeblogやWebメールなどの実装では，入力された文章に含まれるURLをマークアップする場合がよくあります．この場合はpreg_replaceなど正規表現の置換を使えばシンプルな記述で処理できます．リスト10はURLを簡単に表現した正規表現を使って，キャプチャしたURLをaタグでマークアップします．

リスト10を実行すると変数$textに含まれるURL「http://www.php.net/manual/」をaタグでマークアップして出力します[注5]（図2）．

XML文書を解析する

PHPにはGNOMEのlibxmlを使用してXMLのDOMにアクセスする関数や，expatライブラリを使用してXML文書を解析する関数がすでに用意されていますが，必要な情報を取り出すだけであれば正規表現だけでも簡単に記述できます．リスト11はRSSファイルを解析し，新しい上位5件のタイトルとURLをリストアップします．

まとめ

現在PHPで利用できる3種類の正規表現インタフェースのうちPCREライブラリベースのpreg系関数と，IEEE 1003.2準拠の拡張正規表現ライブラリベースのereg系関数の2つのインタフェースの利用方法を見てきました．また，今回は正規表現のさわりの解説を目指したため，あえてマルチバイト対応のmb_ereg系関数は取り扱いませんでした．

近日リリースされるPHPバージョン5.xではmb_ereg系関数の実装にOnigurumaライブラリが利用される予定です．現行のPHPバージョン4.3.xのmb_ereg系関数で利用されているmbregexライブラリやpreg系関数で利用されているPCREライブラリとの差異や，その機能をまとめた「PHPにおける正規表現」のドキュメントがいつかリリースされることを密かに期待しています[注6]．

▼リスト11　preg系関数でのRSSの解析

```
<?php
$rss = file_get_contents('php://stdin');
$pattern = '!
    (?:<item>|<item\s.+?>)
      .*?<title>(.*?)</title>
      .*?<link>(.*?)</link>
      .*?</item>
!sx';

if (preg_match_all($pattern, $rss, $capture)) {
    for ($i = 0; $i < count($capture[0]) && $i < 5; $i++) {
        $title = $capture[1][$i];
        $link  = $capture[2][$i];
        echo "$title ($link)\n";
    }
}
?>
```

注5）ここで使用しているURLの正規表現は簡易的なものです．より正確なURLの正規表現は大崎博基さんのWebサイトの「Perlメモ」→「http URLの正規表現」を参照してください．この他にもShift_JISの文字列にマッチする正規表現など有用な例と考察がたくさん掲載されています．http://www.din.or.jp/~ohzaki/

注6）あと，似た機能を提供するAPIがたくさんある，という状況が整理されると個人的にうれしいのは秘密です．