正規表現について

作成日: 2016/01/21
作成者: 西村

正規表現？

正規表現(Regular Expression。Regex)、というと難しいもののように感じますが、
正規表現、というのは「文字のパターンを表したもの」です。

(例)

A1500033

これはソエルで使用している見積書の番号です。

この番号は、下記のルールで付けられています。

	固定	年度	固定	通番	(枝番)
ルール	"A"	数字 2 桁	0 を 2 桁	数字 3 桁	数字 2 桁
例	A	15	00	033	01

※枝番はある時のみ

こういったルール(パターン)の部分をプログラムで確認したり、必要な部分を抽出したりするために使うのが正規表現です。

正規表現を使うと、下記のような利点があります。

1. ユーザーに入力された文字が正しい形式かどうかを 1 文字 1 文字調べる長いプログラムを書かなくても、
   プログラム 1 行でチェックできる
2. 文字のパターンをチェックしつつ、必要な部分のみ抽出して次の行以降で使用することができる
   (例えば上の見積書番号であれば、形式が正しいかチェックしつつ通番部分だけ取得して次の処理をするなど)
3. 文字のパターンをチェックしつつ、必要な部分のみ置換することができる
   (例えば上の見積書番号であれば、年度部分のみ削除した「A00033」のような形にしたい、となったときにも簡単に
   対応できる)

このルールをそのまま正規表現として書くと、下記のようになります。

```
^A[0-9]{2}00[0-9]{3}([0-9]{2})?$
```

…ちょっと難しく見えるかもしれませんが、下記のような対応になっています。

	最初	固定	年度	固定	通番	(枝番)	最後
ルール		"A"	数字 2 桁	0 を 2 桁	数字 3 桁	数字 2 桁	
例		A	15	00	033	なし	
正規表現	^	A	[0-9]{2}	00	[0-9]{3}	([0-9]{2})?	$

なんとなく意味がわかりますか？

ここで出てきた文字を整理すると、下記のようになります。

記号	意味
^	「最初である」ということを示す記号です。 これがない場合は「A1500033」も「CBA1500033」も OK、ということになります。 ※　巻末に補足があります。一通り読み進めて十分に理解したら見てみてください。
$	「最後である」ということを示す記号です。 これがない場合は「A1500033」も「A1500033XXXX」も OK、ということになります。 ※　巻末に補足があります。一通り読み進めて十分に理解したら見てみてください。
[0-9]	「半角の 0 から 9 までのいずれか」ということを表します。 []の中は「いずれか」という意味、「-」は「～」という意味です。 ※　巻末に補足があります。一通り読み進めて十分に理解したら見てみてください。
{数字}	○回繰り返す、ということを表します。たとえば「3 回繰り返す」なら「{3}」です。 [0-9]{2}は、「0 から 9 までのいずれかの文字を 2 回繰り返す」という意味です。
(ルール)?	かっこ内のルールがあってもなくてもよい、という意味です。 ※　巻末に補足があります。一通り読み進めて十分に理解したら見てみてください。

※正規表現には、他にも特殊な記号(特殊文字)があります。巻末に記載しておきます。

正規表現チェッカー（ http://www.rider-n.sakura.ne.jp/regexp/regexp.php ）を使って、「パターン文字列」に正規表現を、「対象文字列」の欄に「A1500033」や「A3400999」などを入れてみてください。パターンに一致すると、「マッチした文字列」に赤い文字で表示されます。

ざっくりとしたパターンで考えると、下記のようにもできます。

^A.+$

これは下記のような対応になっています。

	最初	固定	年度	固定	通番	(枝番)	最後
ルール		"A"	文字が 1 文字以上				
例		A	1500033 150003300				
正規表現	^	A	.+				$

なんとなく意味がわかりますか？

ここで出てきた文字を整理すると、下記のようになります。

記号	意味
.	ドットは、「任意の 1 文字」という意味です。
+	「1 回以上」という意味です。「.+」は「1 文字以上」という意味になります。

これを例えば PHP でチェックするプログラムを書くとすると、下記のようになります。

(正規表現を使わない時の例)

```php
<?php
// 入力された文字とする
$input = "A1500033";

// 1文字ずつチェックする
$len = strlen($input);
$ok = true;
for ($i = 0; $i < $len; $i++) {
    $c = $input[$i];
    $n = $i + 1;
    // 1文字目がA
    if ($n == 1 && $c == "A") {}
    // 2,3,6,7,8,9,10文字目が0～9
    else if (in_array($n, array(2,3,6,7,8,9,10), true) && ('0' <= $c && $c <= '9'))
{}
    // 4,5文字目が0
    else if (in_array($n, array(4,5), true) && ($c == '0')) {}
    // その他
    else {
        $ok = false;
        break;
    }
}
// 長さチェック
$ok = ($ok && $len == 8 || $len == 10);
// 結果
echo "結果: " . ($ok ? "○" : "×") . PHP_EOL;
```

(正規表現を使う時)

```php
<?php
// 入力された文字とする
$input = "A1500033";
// パターン(PHPは先頭と終了に正規表現を示す文字(「/」や「#」)が必要)
$pattern = "#^A[0-9]{2}00[0-9]{3}([0-9]{2})?$#";
// チェック
$ok = preg_match($pattern, $input);
// 結果
echo "結果: " . ($ok ? "○" : "×") . PHP_EOL;
```

1 行(preg_match())だけで形式のチェックができていることがわかると思います。

C#や VB.NET で正規表現を使う場合は下記のようになります。

```
namespace ConsoleApplication3
{
    class Program
    {
        static void Main(string[] args)
        {
            // 入力された文字とする
            var input = "A1500033";
            // パターン
            var pattern = @"^A[0-9]{2}00([0-9]{3})([0-9]{2})?$";
            // チェック
            var ok = Regex.IsMatch( pattern, input );
            // 結果
            Console.WriteLine("結果: " + (ok ? "○" : "×"));
        }
    }
}
```

正規表現を使ってチェックするためのメソッドは下記になります。

言語	メソッド、関数など	補足
PHP	preg_replace(正規表現, 文字列)	正規表現は"#…#"のような形で「#」や「/」で囲う
JavaScript	正規表現.test(文字列) 文字列.match(正規表現)	正規表現は/…/の形で「/」で囲う ※引用符では囲わない
C#、VB.NET	Regex.IsMatch(文字列, 正規表現)	正規表現は@"…"の形で文字列の前に「@」を付ける
Java	文字列.matches(正規表現)	正規表現内の「¥」を認識させるには「¥¥」、「¥¥」を認識させるには「¥¥¥¥」とする

文字のパターンから部分を抽出

丸かっこ(「(」と「)」)で囲んだものは、「グループ」と呼ばれます。丸かっこでグループ化すると、
その部分を後から抽出することが出来ます。

例えば、先程の見積書番号の通番だけ取りたい、となった場合、

```
^A[0-9]{2}00[0-9]{3}([0-9]{2})?$
```

まず下記のように正規表現のパターンをグループ化します。

```
^A[0-9]{2}00([0-9]{3})([0-9]{2})?$
```

PHP であれば、下記のようにします。

```php
<?php
// 入力された文字とする
$input = "A1500033";
// パターン(PHPは先頭と終了に正規表現を示す文字(「/」や「#」)が必要)
$pattern = "#^A[0-9]{2}00([0-9]{3})([0-9]{2})?$#";
// チェック ($mは、結果が入る)
$ok = preg_match($pattern, $input, $m);
// 結果
echo "結果: " . ($ok ? "○" : "×") . PHP_EOL;
if ($ok) {
    echo "通番: " . $m[1]; // 033が出力される
}
```

※ $m[0]が全体、$m[1]が 1 番目のグループ(キャプチャ結果)、$m[2]が 2 番目のグループ(キャプチャ結果)…になります。

必要な部分のみ置換

正規表現を使うと、パターンにもとづいて必要な部分のみ置換することも出来ます。

例えば、先程の見積書番号の年度だけ削除したい(「A1500033」→「A00033」)、となった場合、

```
^A[0-9]{2}00[0-9]{3}([0-9]{2})?$
```

まず下記のように正規表現のパターンをグループ化します。

```
^A[0-9]{2}00([0-9]{3})([0-9]{2})?$
```

置換後の文字列は下記とします。

```
A00$1$2
```

※ $1、$2 はグループ番号です。

PHP であれば、下記のようにします。

```php
<?php
// 入力された文字とする
$input = "A1500033";
// パターン(PHPは先頭と終了に正規表現を示す文字(「/」や「#」)が必要)
$pattern = "#^A[0-9]{2}00([0-9]{3})([0-9]{2})?$#";
// 置換 ($resultに、置換結果が入る)
$result = preg_replace($pattern, "A00$1$2", $input);
echo "結果: " . $result . PHP_EOL;
```

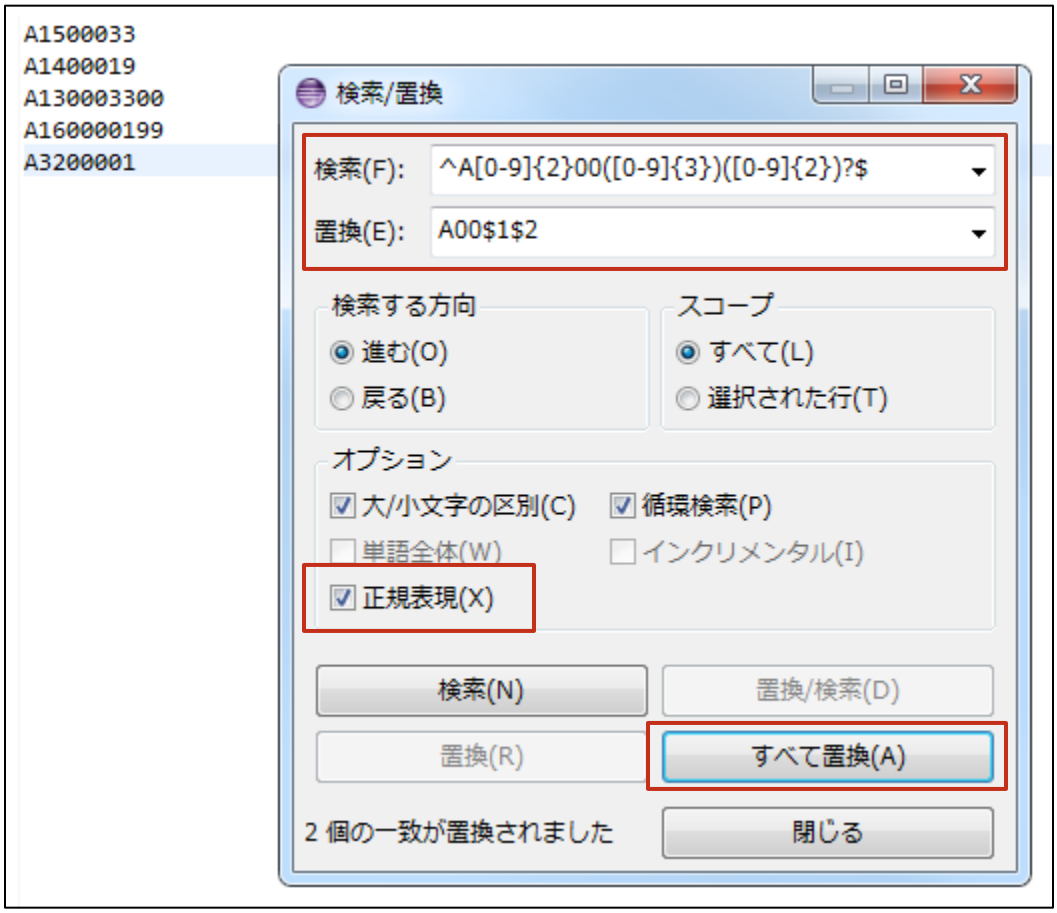
※　$m[0]が全体、$m[1]が 1 番目のグループ、$m[2]が 2 番目のグループ…になります。

「15」の部分が「13」「14」「32」「99」でも置換ができるので、単純な置換よりも手間が省けます。

テキストエディタなどでの正規表現置換

テキストエディタや開発環境(Eclipse, Visual Studio, サクラエディタなど)でも正規表現を使って置換ができます。

(Eclipse の例。Ctrl ＋ F で出ます。もしよければ実際にやってみてください)

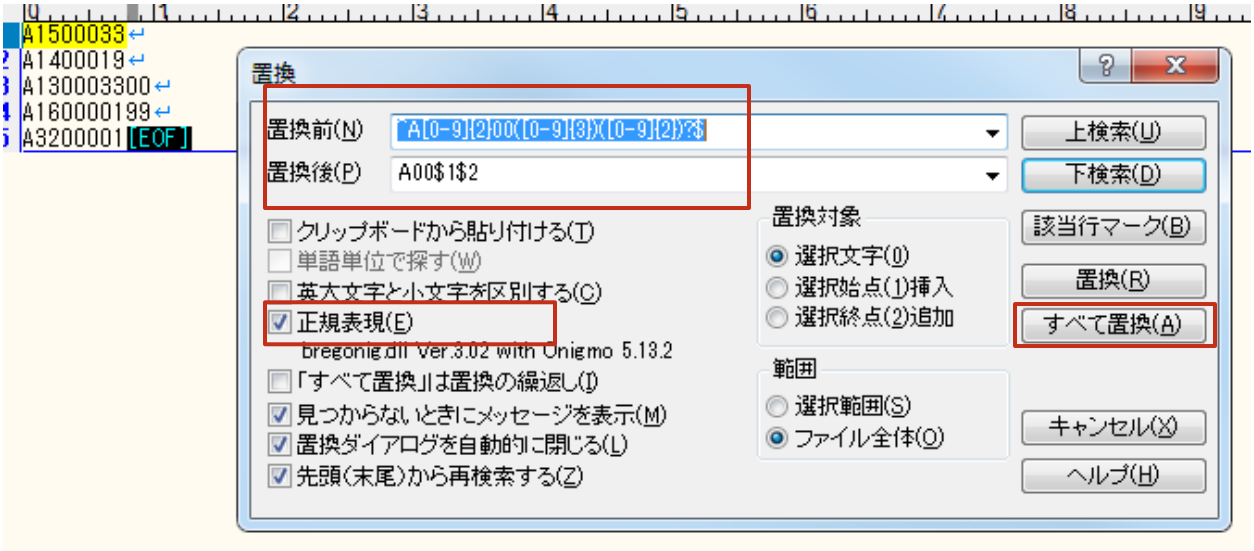


※　実際にはここまでの厳密な正規表現ではなく、「A..([0-9]+)」を「A$1」に置換、のようなざっくりとしたパターンで簡単にやるほうがいいと思います。(「.」は正規表現では任意の 1 文字、という意味です)

(その他の例)

検索	置換	結果
(¥r¥n)+	¥r¥n	連続改行を 1 回ずつのみにできる
"(.+)"	$1	二重引用符で囲われた文字列の囲いを外せる
¥r¥n	","	改行区切りの文字を "a","b","c" などの形でプログラムの配列などに含めたい場合に途中まで整形できる

(サクラエディタの例。Ctrl + R で出ます)



Visual Studio でも正規表現が使えますが、Visual Studio 2012 以前の正規表現はグループ化のかっこが「{}」になっていたり独自の正規表現になってしまっているのですごく使いづらいです…

参考 よくある正規表現の例

	正規表現
拡張子が画像ファイルか調べる (jpg, png, gif)	¥.(jpe?g|png|gif)$ ※ test1.png、test2.jpg, test3.jpeg, test4.gif などが一致する ※「.」は特殊文字なので、普通の「.」として扱うには「¥」を付ける
数値 (int)	^(0|[1-9][0-9]*)$ 0, 1, 10, 1234 などが一致する
改行	¥r¥n
空白の繰り返し	¥s+
メールアドレス	メールアドレスは厳密に確認しようとすると正規表現ではチェックできないくらい複雑なので下記くらいのルーズなチェックになることが多いです。 ^.+@.+¥..+$ ※ a@a.a などが一致する

参考 特殊な文字の一覧

(開始、終了の記号)

記号	意味
^	「最初である」ということを示す記号です。 ※　言語やモードによって「行の最初」という意味になるので注意が必要な場合があります。 PHP、Ruby、Perl 等の場合は代わりに「¥A」を使うと確実に「文字列の最初」という意味になり安全です。
$	「最後である」ということを示す記号です。 ※　言語やモードによって「行の最後」という意味になるので注意が必要な場合があります。 PHP、Ruby、Perl 等の場合は代わりに「¥z」を使うと確実に「文字列の最後」という意味になり安全です。

(繰り返しや回数を表す記号)

記号	意味
+	「1 回以上」という意味です。「.+」は「1 文字以上」という意味になります。
*	「0 回以上」という意味です。「.*」は「0 文字以上」という意味になります。
{数字}	○回繰り返す、ということを表します。たとえば「3 回繰り返す」なら「{3}」です。 [0-9]{2}は、「0 から 9 までのいずれかの文字を 2 回繰り返す」という意味です。
? (ルール)?	直前にあるルールがあってもなくてもよい、という意味です。 かっこを付けるとグループ化がされます。グループ化されないようにするには「(?:)」という形にします。

(文字の範囲などを表す記号)

記号	意味
[0-9]	「半角の 0 から 9 までのいずれか」ということを表します。 []の中は「いずれか」という意味、「-」は「～」という意味です。 ※　「¥d」でも「0-9」という意味になりますが、¥d だと全角数字も一致する実装があるため注意してください。
[文字-文字]	「コード順での文字から文字までのいずれか」ということを表します。 よく使うのは、 [a-z] (英小文字)や [A-Z] (英大文字)、 [a-zA-Z0-9] (半角英数字)です。
[文字]	「いずれかの文字」を表します。 例えば [abc] だと a、b、c いずれかという意味です。
[^文字]	「いずれかの文字ではない」(否定)を表します。 例えば [^abc] だと a、b、c のどれでもないという意味です。 [^0-9] だと「数字以外」です。
(ルール)	かっこ内のルールを 1 つのグループとします。 例えば abc+ だと「a が 1 回、b が 1 回のあと c が 1 回以上」(abcccc など)になりますが、 (abc)+ だと「"abc"が 1 回以上」(abcabcabc など)という意味に変わります。 キャプチャしたくない場合は (?:abc)+ のように (?:ルール) とします。

(特殊な文字)

記号	意味
¥r	キャリッジリターン(CR)です。
¥n	ラインフィード(LF)です。¥r¥n だと CRLF になります。
¥t	タブ(水平タブ)文字です。
¥s	空白です。実装によって全角を含んだりすることもあります。
¥	特殊文字のエスケープです。「¥.」などとします。