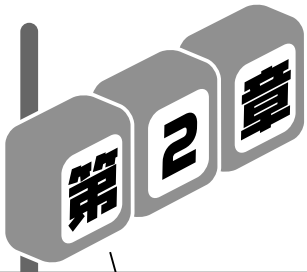
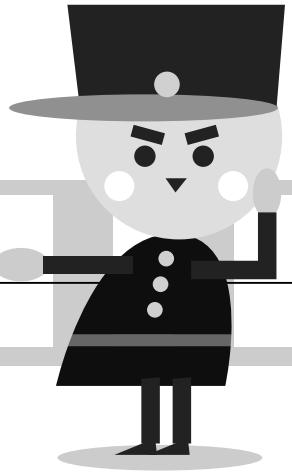


Java編



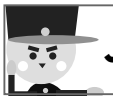
J2SEの正規表現パッケージを学ぶ

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

はじめに

J2SE（Java 2 Platform, Standard Edition） 1.4が登場するまで，Javaにおける正規表現はJakarta ORO，IBMのcom.ibm.regex，com.stevesoft.pat，gnu.regexpなどサードパーティが提供するパッケージに頼る必要がありました．性能や機能面で優秀なパッケージもあれば，バグが多くメンテナンスも活発でないパッケージもあるなど，利用者から見るとどれを選択すれば良いのか非常にわかりにくい状況が続いていました．しかしJ2SE 1.4から標準装備されたjava.util.regexパッケージは，Perlライクでモダンな機能と良好な処理性能，Unicodeの幅広いサポート，整備されたドキュメントなど，機能・性能・サポート・将来性などの面から安心して利用できるパッケージになっています．

本章ではJ2SE 1.4から標準装備されたjava.util.regexパッケージの利用方法を説明します．



Javaの方言

Javaにおける正規表現の操作に必要なオブジェクトは，コンパイルした正規表現を表すjava.util.regex.Patternと，Patternオブジェクトが持つ正規表現の適用結果を表すjava.util.regex.Matcher，さらに不正な正規表現を使用した際に発生する例外java.util.regex.PatternSyntaxExceptionの合計3つだけです．正規表現の操作には通常PatternとMatcherの2つのオブジェクトを利用しますが，Stringオブジェクトに追加されたreplaceAllメソッドやreplaceFirstメソッドなどでも正規表現を利用した置換処理が利用できます．

...マッチング

Stringオブジェクトtextで指し示す文字列を対象にマッチングを行うには，リスト1のように記述します．

正規表現の文字列でPatternオブジェクトを生成し，その正規表現と対象となる文字列をmatcherメソッドで関連付けMatcherオブジェクトを生成します．生成したMatcherオブジェクトのfindメソッドなどで最終的にマッチング処理を実行します．

...置換

置換を行う場合はリスト2のようにMatcherオブジェクトのreplaceFirstメソッドかreplaceAllメソッドを使用します．replaceFirstメソッドが最初にマッチした箇所の置換結果を返すのに対し，replaceAllメソッドはマッチするすべての箇所を置換した結果を返します．リスト2を実行すると文字列「www.example.com」

▼リスト1　Javaでのマッチング

```
public class Ex1 {
    public static void main(String[] args) {
        String text = "Hello World!";
        java.util.regex.Pattern p =
                java.util.regex.Pattern.compile("Hello");
        java.util.regex.Matcher m = p.matcher(text);

        if (m.find()) {
            // textはHelloを含んでいる
        } else {
            // textはHelloを含んでいない
        }
    }
}
```

▼表1 java.util.regex.Patternクラス

戻り値	メソッド
static Pattern	compile(String regex)
static Pattern1	compile(String regex, int flags)
Matcher	matcher(CharSequence input)
static boolean	matches(String regex, CharSequence input)
String[]	split(CharSequence input)
String[]	split(CharSequence input, int limit)
int	flags()
String	pattern()

▼リスト2 Javaでの置換

```
import java.util.regex.*;

public class Ex2 {
    public static void main(String[] args) {
        String text = "www.example.com";
        Pattern p = Pattern.compile("\\.com");
        Matcher m = p.matcher(text);

        String replaced = m.replaceFirst(".jp");
        System.out.println(replaced);
    }
}
```

を対象に，正規表現\.comに一致する箇所を.jpに置換した結果が返され，「www.example.jp」と出力されます．

またPattern，Matcherオブジェクトを介さずにStringオブジェクトのreplaceAllメソッドやreplaceFirstメソッドでも正規表現を使用した置換を行えます．

PatternおよびMatcherクラスのメソッドを表1，表2それぞれに示します．各メソッドの詳細についてはJavadocを参照してください[注1]．

メタ文字

java.util.regexは第1章 表1に挙げた現在一般的な正規表現のほとんどの機能を利用できます．java.util.regexではさらに*+や++や?+といった強欲な量指定子や，文字クラスの集合演算などが利用できます（表3）．

正規表現はStringオブジェクトで与えるため，ソースコード中で\w+\sなどのメタ文字を記述する場合は，\\w+\\sといったふうに\（バックスラッシュ）

▼表2 java.util.regex.Matcherクラス

戻り値	メソッド
boolean	find()
boolean	find(int start)
String	group(int group)
String	group()
int	groupCount()
String	replaceFirst(String replacement)
String	replaceAll(String replacement)
Matcher	appendReplacement(StringBuffer sb, String replacement)
StringBuffer	appendTail(StringBuffer sb)
int	start()
int	start(int group)
int	end()
int	end(int group)
boolean	lookingAt()
boolean	matches()
Pattern	pattern()
Matcher	reset()
Matcher	reset(CharSequence input)

▼表3 java.util.regexのメタ文字[注2]

メタ文字	役割
\p{Prop}	Propで指定したUnicodeプロパティとブロック（スクリプトは未サポート）
\P{Prop}	Propで指定したUnicodeプロパティとブロック以外
?+	0もしくは1文字（強欲な量指定子）
*+	0文字以上の繰り返し（強欲な量指定子）
++	1文字以上の繰り返し（強欲な量指定子）
{n}+	n個の繰り返し（強欲な量指定子）
{n,}+	n個以上の繰り返し（強欲な量指定子）
{n,m}+	n個以上m個以下の繰り返し（強欲な量指定子）

を\でエスケープした上で指定する必要があります．

java.util.regexの特徴的な機能として強欲な量指定子が挙げられます．通常のマッチング処理ではさまざまなマッチの可能性に備え，検索経路の状態（ステート）を記憶しています．強欲な量指定子はこのマッチ中の状態を生成せず，マッチした範囲を強欲に保持し

注1）http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/package-summary.html
注2）この他の一般的なメタ文字については第1章 表1とjava.util.regex.Patternのドキュメントを参照してください．

特別企画 1

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！

て解放しません．たとえば文字列12345に正規表現^(\d+)\d$を使用するとキャプチャ結果1234を得ることができますが，強欲な量指定子^(\d++)\d$の場合は12345すべての文字列を\d++が押さえるために最終的にマッチしなくなります．

Unicodeに関係する要素として\w，\d，\sおよび\W，\D，\SはASCII文字だけにマッチし，Unicodeにおけるほかの英数字や空白などにはマッチしない点に注意が必要です．Unicode上の英数字にマッチさせたい場合は\p{L}，Unicode上の数字にマッチさせたい場合は\p{Nd}，空白文字にマッチさせたい場合は\p{Z}を使用します．ちなみに，プロパティ名は\p{Lu}といった短縮系の名前だけをサポートし，\p{Lowercase_Letter}といった長い名前はサポートしていない点にも注意が必要です．

java.util.regexではUnicodeプロパティとブロックをほぼ完全にサポートしていますが，Unicodeスクリプトはサポートしていません．そのため日本語のひらがな・カタカナ・漢字を指すUnicodeスクリプト\p{Hiragana}・\p{Katakana}・\p{Han}は使用できないので，代わりにひらがな・カタカナ・漢字のコードブロックを指す\p{InHiragana}・\p{InKatakana}・\p{InCJKUnifiedIdeographs}などのUnicodeブロックを使用します．

Unicodeブロックを指定する場合はブロック名の接頭子としてInを付加し，下線や空白を含まない形式でブロックを指定します．たとえばUnicodeで0000から007Fのブロックをさす名前としてBasic Latinが定義されていますが，正規表現でこのブロックを指定する場合は\p{InBasicLatin}と記述します[注3]．

UnicodeブロックはUnicode文字マップ内の特定の範囲（ブロック）を表すものであるのに対し，Unicodeスクリプトは言語別の書記体系（スクリプト）をグループ化したものです．そのためUnicodeスクリプトがサポートされていなくとも，Unicodeブロックや Unicodeプロパティの組み合わせで期待する文字を指定することができますが，Unicodeブロックはあくまでも文字マップ上の特定の範囲を指すものですので，実際に文字が割り当てられていない領域を含む場合がある点に注意が必要です．

...オプション

正規表現のオプションはPattern.compileメソッドの第2引数か，正規表現中の(?mode)で指定します．java.util.regexで使用できるオプションを表4に示します．

java.util.regexではUnicodeのサポートに際し，行末を示す終端子として

- LF（U+000A）
- CR（U+000D）
- CRLF（U+000D U+000A）
- NEL（U+0085）
- LS（U+2028）
- PS（U+2029

の6種類をサポートしています．これらの文字は行の先頭を指す^（ハット），行の末尾を指す$や，.（ドット）や\Zに作用します．これらのメタ文字が6種類の行終端子のどれに作用するかは表4で挙げたオプションUNIX_LINES，DOTALL，MULTILINEの組み合わせで変化するので注意が必要です．

▼表4 java.util.regexのオプション

オプション	(?mode)	役割
Pattern.CASE_INSENSITIVE	i	大文字と小文字を無視する
Pattern.MULTILINE	m	行単位でマッチングする
Pattern.DOTALL	s	.（ドット）を改行文字にもマッチさせる
Pattern.UNIX_LINES	d	^（ハット）を\nだけに反応させる（.の動作も変化する）
Pattern.COMMENTS	x	正規表現中の空白やコメントを許可する
Pattern.CANON_EQ	（なし）	通常は同じ文字として扱われながら実際にはエンコーディングの異なる文字を，同じ文字としてマッチングする
Pattern.UNICODE_CASE	u	Unicodeにおける大文字・小文字を無視する

注3）その他のブロック名とその範囲はhttp://www.unicode.org/Public/UNIDATA/Blocks.txtを参照してください．

…キャプチャ

マッチした文字の一部分を取り出したり検索結果に埋め込みたい場合は，正規表現のキャプチャを使用します．リスト3のように正規表現中でキャプチャ用括弧（と）で指定した範囲にマッチした文字列は，Matcher.group(int num)メソッドで参照できます．

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m.group(1)，m.group(2)，m.group(3)…と取得したいインデックス番号を与えることで個別にアクセスできます．またgroupメソッドで指定できる値の最大値はMatcher.groupCount( )メソッドで得ることができます．

replaseFirstメソッドやreplaceAllメソッド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4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…などの文字を指定します．リスト4を実行すると「Here... (url: http://www.example.jp)」と出力されます．

java.util.regexによる基本的な正規表現の利用手順は以上のとおりです．この他にもMatcherオブジェクトには，置換結果をStringBufferオブジェクトに詰め込んでいくappendReplacementメソッドなど，パフォーマンス的に有効なメソッドがいくつか用意されています．

利用例

それではWebアプリケーション開発に際して頻繁に発生する課題を例に，java.util.regexのより具体的な利用例を見ていきましょう．

…データのチェック

正規表現によるデータのチェックの例として，郵便番号のチェックを見てみましょう．現在利用されている郵便番号は7桁の数字で，3桁目と4桁目の間にハイフンを入れて表記します[注4]．また，旧郵便番号には3桁のものと5桁のものがあります．通常は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，ここでは

- 154
- 350-11
- 350-1106

の3パターン，およびハイフンを省略した郵便番号文字列を許可する以下の正規表現を用意します（リスト5）．

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

もし7桁の郵便番号のみを許可する場合は以下の正規表現を使用します．

```
^\\d{3}-?\\d{4}$
```

5桁と7桁の郵便番号を許可してハイフンの省略を許可しない場合は以下の正規表現を使用します．

▼リスト3　キャプチャの例

```
import java.util.regex.*;

public class Ex3 {
    public static void main(String[] args) {
        String text = "Name: Regular Expressions";
        Pattern p = Pattern.compile("^(\\w+):\\s+(.+)$");
        Matcher m = p.matcher(text);

        if (m.find()) {
            String key   = m.group(1);  // key   = "Name";
            String value = m.group(2);  // value = "Regular Expressions";
            System.out.println("key   = " + key);
            System.out.println("value = " + value);
        }
    }
}
```

▼リスト4　キャプチャ結果による置換

```
import java.util.regex.*;

public class Ex4 {
    public static void main(String[] args) {
        String text = "<a href=\"http://www.example.jp/\">Here...</a>";
        String replaced = text.replaceFirst(
                "<a href=\"(.+?)\">(.+?)</a>", "$2 (url: $1)");
        System.out.println(replaced);
    }
}
```

注4）現在は算用数字のみが利用されていますが，郵便番号制マニュアルには「当分の間，算用数字のみを使用します．」と記載されているので，ライフサイクルの長いプログラムでは注意が必要かもしれません．http://www.post.japanpost.jp/zipcode/zipmanual/

特別企画 1

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！

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

...ひらがな・カタカナ・漢字

java.util.regexではUnicodeプロパティとUnicodeブロックがサポートされています．そのため日本語に関係するひらがなやカタカナもしくは漢字などのマッチングを行う場合は，Unicodeブロックを使用します．

つまりひらがなの文字列にマッチさせる場合は`\\p{InHiragana}+`，カタカナの場合は`\\p{InKatakana}+`，漢字の場合は`\\p{InCJKUnifiedIdeographs}+`を使用します．

リスト6は文字列から英数字・ひらがな・カタカナ・漢字の各文字種ごとにブロックを抽出するサンプルです．

リスト6を実行すると

```
% java Ex6
漢字
カナ
混
じりの
文章
を
input
します
```

と分割した文字列を逐次出力します．日本語を含む文字列の簡単なインデックスを作る場合や，文章中の漢字とひらがなの含有率を調べる場合などに便利です[注5]．

▼リスト5　郵便番号

```
import java.util.regex.*;

public class Ex5 {
    public static void main(String[] args) {
        String text = args[0];
        String pcode_reg = "^\\d{3}(?:-?(?:\\d{2}|\\d{4}))?$";

        if (Pattern.matches(pcode_reg, text)) {
            System.out.println(text + " is postal code");
        } else {
            System.out.println(text + " is not postal code");
        }
    }
}
```

▼リスト6　文字種ごとの抽出

```
import java.util.regex.*;

public class Ex6 {
    public static void main(String[] args) {
        String text = "漢字カナ混じりの文章をinputします";
        Pattern p = Pattern.compile(
                "(\\p{InBasicLatin}+|" +
                " \\p{InHiragana}+|" +
                " \\p{InKatakana}+|" +
                " \\p{InCJKUnifiedIdeographs}+)",
                Pattern.COMMENTS);
        Matcher m = p.matcher(text);

        while (m.find()) {
            String chunk = m.group(1);
            System.out.println(chunk);
        }
    }
}
```

...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をハイパーリンクとしてマークアップする場面がよく現れます．正規表現の置換を利用すれば，この類いの面倒な処理も簡素に記述できます．リスト7はURLを単純化した正規表現を使って，URLをaタグでマークアップします．

リスト7を実行すると文中に含まれる「http://www.example.jp/path/to/file.html」というURLを，

```
<a href="http://www.example.jp/path/to/file.html">http://www.example.jp/paht/to/file.html</a>
```

（実際は1行）

▼リスト7　URLのマークアップ

```
import java.util.regex.*;

public class Ex7 {
    public static void main(String[] args) {
        String text = "httpやhttpsのスキームを持つ"
                + "http://www.example.jp/path/to/file.htmlといったURLを"
                + "マークアップします．";

        String replaced = text.replaceAll(
                "(https?://[-_.!~*'()a-zA-Z0-9;/?:@&=+$,%#]+)",
                "<a href=\"$1\">$1</a>");
        System.out.println(replaced);
    }
}
```

注5）同様の処理をPerlのUnicodeスクリプトで記述した例と，この他のUnicodeプロパティなどの概要についてはhttp://module.jp/blog/regex_unicode_prop.htmlを参照してください．

に置換します[注6].

...HTMLタグを取り除く

クロスサイトスクリプティング防止や，表示上の一貫性を維持する目的などで，HTMLの文書からタグを取り除いたテキストだけを取り出したい場合があります．ここでは単純に < と > でくくられた要素を削除するだけですので String.indexOf( )や String.substring( )メソッドなどでも処理できますが，正規表現を使えばリスト8のように処理の内容がより明確になります．

...XML文書を解析する

通常JavaにおけるXMLの処理はjavax.xml.parsersやサードパーティが供給するXML用のパッケージを使用しますが，必要な情報を抽出するだけであれば正規表現でも簡単に記述できます．リスト9はWebサイトのコンテンツの一覧と内容の配信に使用されるRSS（RDF Site Summary）ファイルを解析し，新しい記事上位5件のタイトルとURLをリストアップします．

まとめ

今までは安心して採用できる正規表現パッケージがJavaにはなかったため，文字列処理での手数の多さやまどろっこしさからJavaを採用しないプログラマも多かったことと思います．しかしバージョン1.4から標準採用されたjava.util.regexによって，これらの不満は解消されつつあります．java.util.regexはかなりの部分で最新のPerlに迫る（一部は追い越した）機能と完成度の正規表現パッケージです．これからはどの正規表現パッケージを選べば良いのかを悩む必要はなく，特に理由がない限り，標準添付され，必要十分な機能と性能を備え，ドキュメントが整備され，将来的にもメンテナンスされ続けるであろうjava.util.regexを選べばよいのです．

今まで正規表現を使ったことがない方も，今まで正規表現が使えないためにJavaを避けていた方も，ぜひ一度java.util.regexのキモチヨサを体感してみてください．

▼リスト8　HTMLタグを取り除く

```
import java.util.regex.*;

public class Ex8 {
    public static void main(String[] args) {
        String text = "<p>今週の<a href=\"/to/file\">特選品</a>！</p>";

        String replaced = text.replaceAll("<.+?>", ""); // "<[^>]+>"でも可
        System.out.println(replaced);
    }
}
```

▼リスト9　RSSの解析

```
import java.io.*;
import java.util.regex.*;

public class Ex9 {
    public static void main(String[] args) {
        BufferedReader r = new BufferedReader(new InputStreamReader(System.in));
        StringBuffer rss = new StringBuffer("");
        String line;

        try {
            while ((line = r.readLine()) != null) {
                rss.append(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        Pattern p = Pattern.compile(
                "(?:<item>|<item\\s.+?>)" +
                "    .*?<title>(.*?)</title>" +
                "    .*?<link>(.*?)</link>" +
                ".*?</item>", Pattern.COMMENTS);
        Matcher m = p.matcher(rss);
        for (int i = 0; m.find() && i < 5; i++) {
            System.out.println(m.group(1) + "(" + m.group(2) + ")");
        }
    }
}
```

注6）URLのより正確な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．他にも有用な正規表現の例と考察がたくさん掲載されています．http://www.din.or.jp/~ohzaki/