XML プログラミング入門
「正規表現と XML 文書の操作」

師 茂樹*

2004 年 8 月 16 日

1　テキスト処理とモデル

現実世界のある存在や現象について漏らすことなく「厳密に」説明しようとすると、無限の言葉を連ねても不可能ではないかと思われる。古典テクストについても、写本であれば一文字一文字の位置、大きさ、一点一角、インクや墨の濃度やかすれ具合、紙の材質、寸法、厚み、保存状態、各部分の日焼け具合、虫食い... などなど、数え上げたらきりがないほどの情報が含まれており、かつそれらが複雑に絡み合っている。さらにテクストを読む段階でも、テクストの作者がそれまでに経験したであろう様々なことが、多かれ少なかれそのテクストに反映されていると考えれば、それらもテクストを「厳密に」読むためには考慮に入れなければなるまい。

しかし、本当にこれらすべてを扱おうとすると、文字を一文字読むことすら不可能である。実際には、これらの情報は古典学者の頭の中で、問題意識や経験、能力、慣習などによって自動的に取捨選択されているので、このようなことはあまり問題にならない。例えば、内容を読むことに重点を置いている研究者は、テクストの読解に活字化されたものを使うことが多い。

モデル化とは、これと同じように、現実世界の存在や現象、問題に含まれる無限とも言える情報の中から、その存在や現象を記述するのに適当であると思われるものや、問題解決に必要なものを抽出（抽象化）し、ある一定の構造の中に落とし込む作業のことである。抽出されたモデルを分析したり操作することで、対象を完璧に記述しきる（そのようなことができれば、の話であるが）ことはできないものの、うまくモデル化することができれば有意な結果を（できれば効率的に）得ることができるのではないか、という方法である。

モデル化、という方法は、物理学などの理系の分野では日常的に用いられている方法であるが、従来の人文学でも用いられてきた手法である。ただし、従来の古典学における情報の取捨選択と、人文学の情報処理におけるモデル化との決定的な違いは、前者が無意識的なのに対して、後者においてはコンピュータを間に挟むことによってモデル化について常に自覚的でなければならない、ということである。別の言葉で言えば、古典学の情報処理とは「モデルを通じてテクストを読む」ということになるだろう。

逆に言えば、プレーンテキストや XML などもまた、テキストに対する見方＝モデルが背景にある。すなわち、我々がこれまで何気なく行ってきたプレーンテキストや XML を使った研究が、これらのモデルによって我々のテキストの見方を少なからず規制していた可能性がある。

さて、「モデルを通じて古典を読む」ためには、

* 花園大学専任講師、s-moro@hanazono.ac.jp

1. 既存のモデルの理解
2. 新たなモデルの考案（と実践）

ことが必要となる。すなわち、XML などの既存の形式の背景にあるモデルを理解するだけでなく、もし既存のモデルに限界があればそれを克服するために新たなモデルを創り出す必要がある。

モデルについて理解したり、新たに創り出したりするためには、モデル化の対象と、モデルを実現する計算機についての深い知識と洞察が必要になる。写本を扱う場合であれば、前者は書誌学や文字学、文献学などの知識ということになろうし、後者は離散数学や形式言語理論、符号化理論などの計算機科学のことである。

2 正規表現

2.1 正規表現とは何か

正規表現（regular expression）とは、文字列のパターンを表記する方法である。例えば、“あああ” と “いいい” と “ううう” は全く異なる文字列であるが、“同じ文字を三回繰り返している” というパターンは同じである。このようなパターンは、正規表現であれば「`.{3}`」という表記で表すことができる。

正規表現は、その名前からもわかるように正規文法に基づいている。ただし厳密に言えば、多くの処理系で正規文法ではカバーできないがあれば便利な表現を拡張して実装しているので、正規文法内に収まる正規表現処理系はほとんどない。

2.2 grep で使うことができる正規表現

正規表現には処理系によって様々な「方言」が存在する。以下、一例として、grep で使うことができる正規表現について概観する。grep の正規表現は様々な処理系の中の最小公倍数的なものであるが、ここであげられているものがすべて他の処理系で使用可能というわけではない。

2.2.1 文字の選択

正規表現においては、ある文字（メタ文字以外）を指定すれば、その文字そのものを表す。

通常の文字では表せない文字列のパターンは、メタ文字を用いる。メタ文字に使われている文字を通常の文字として指定したい場合には、メタ文字の前に「`\`」を付ける。例えばピリオドを指定したい場合には「`\.`」とする。

メタ文字	内容
`.`	任意の一文字
`[`*char-list*`]`	[...] 内のどれか一文字
`[:`*classname*`:]`	文字クラス *classname* に属する任意の一文字
`\w`	文字と数字とアンダースコア
`\W`	文字と数字とアンダースコア以外
regexp`|`*regexp*	正規表現の選択

`\w` は少しわかりにくいが、英語などのアルファベットを使う言語における単語で使われる文字のことである。

多くの正規表現処理系では、下のようなクラス指定が可能である。

クラス指定	内容
[:alnum:]	アルファベットと数字
[:alpha:]	アルファベット
[:upper:]	アルファベットの大文字
[:lower:]	アルファベットの小文字
[:digit:]	数字
[:xdigit:]	16 進数の数字
[:punct:]	句読点類
[:space:]	スペース、タブ、改ページ
[:blank:]	空白文字（スペース、タブなど）
[:cntrl:]	制御文字
[:graph:]	印字可能かつ表示可能な文字
[:print:]	印字可能な文字

この中、例えば [:alpha:] は、[a-zA-Z] と等価である。

正規表現がアメリカ生まれのためか、先ほどの\w と言い、ここで挙げられているクラスと言い、英語に代表されるアルファベット系言語に便利なメタ文字ばかりであり、漢字文献を扱うものにとって有用そうなものはほとんど見られない。

Perl 5.8 では Unicode の文字プロパティに基づくメタ文字が利用可能となっており、漢字やデーヴァーナーガリーなどアジアの古典で使われる文字を処理しやすいメタ文字が使えるようになってきた。例えば、\p{Han}というメタ文字は、Unicode に漢字として登録されている任意の漢字 1 文字を表す。文字プロパティについては [6]、文字プロパティを用いたメタ文字については man perlunicode を参照のこと。

2.2.2 繰り返し

次にあげるものは、パターンの繰り返しを指定するためのメタ文字である。

メタ文字	内容
*	0 回以上の繰り返し
+	1 回以上の繰り返し
?	0、1 回の繰り返し
$\{n\}$	n 回の繰り返し
$\{n,m\}$	n 回以上 m 回以下の繰り返し
$\{n,\}$	n 回以上の繰り返し
(*regexp*)	正規表現のグループ化
\1 \2 \3 \4 \5 \6 \7 \8 \9	後方参照

*、+、?、$\{n\}$、$\{n,m\}$、$\{n,\}$ は、メタ文字などで書かれたパターンの後ろに続けて書く（(*regexp*) の後に書くこともできる)。例えば「あ {2,4}」という正規表現は、「ああ」「あああ」「あああああ」にマッチする。

後方参照[*1]とは、それ以前にマッチしたパターンを指示するメタ文字である。`\1` は、それ以前の 1 番目の (*regexp*) を指す。例えば、「`(.).\1.`」という正規表現は、「無念無想」「アイアイ」などのパターンにマッチする。

2.2.3 位置

次のメタ文字は、文中の位置を示すメタ文字である。

メタ文字	内容
`^`	行の先頭
`$`	行の末尾
`\b`	単語の先頭または末尾
`\B`	単語の途中
`\<`	単語の先頭
`\>`	単語の末尾

`\b` 等は、英語などの分かち書き言語を前提としたメタ文字である。

2.3 正規表現の問題点

漢字文献について言えば、従来ある正規表現でも、例えば、

```
(\p{Han})[^\1]\1[^\1]
```

という具合に後方参照を用いた正規表現を用いれば、漢語表現でよく用いられる「無念無想」というようなパターンを抽出することが可能である。しかしながら、漢文中に見出される表現上のパターンはこのような単純なものだけではない。例えば「一日千秋」という成語であれば、

任意の漢数字 . 任意の漢数字 .

という正規表現が書ければ（四角で囲んだ部分が 1 文字分のメタ文字を表す。以下同様）、ヒットさせることができるであろう[*2]。すなわち、文字そのものではなく文字の属性によるパターンマッチである。このような用途に対しては、文字クラス `[ … ]` を用いることで（煩瑣になろうが）比較的容易に実現されるであろうし、Perl 5.8 で実装された Unicode のプロパティを指定できるメタ文字は、そのような考え方を Unicode の定義する範囲でサポートしたものと言ってよい。Unicode には残念ながら「漢数字」というプロパティはないが、Perl 5.8 であれば簡単なスクリプトを追加することで容易に拡張が可能である 。

もっとも、これでもまだ柔軟性に欠き、古典テキストの分析には不十分である。例えば「春眠暁を覚えず」で有名な孟浩然の漢詩は、

春眠不覺曉
處處聞啼鳥
夜來風雨聲

[*1] backreference の訳語についてはゆれがあるが、ここでは「後方参照」とする。[1] 日本語訳 p. 19 の訳注参照。

[*2] 厳密に言えば、この正規表現では「一二三四」などの意図しない文字列にもヒットしてしまうので、実際には Perl 的に書けば「`\p{漢数字}\P{漢数字}\p{漢数字}\P{漢数字}`」などとしなければならないだろう。

花落知多少

であるが、各詩句の最後の文字（暁、鳥、聲、少）の韻のパターンを調べると「○○●○」となっている。このような韻を踏むという作法は、言うまでもなく作詩する上でのもっとも基本的なルールである。これと同形式の韻文を、改行も句読点もないまったくの白文から抽出しようとする場合、

```
.{4}(.).{4}一番目の丸括弧の文字と同じ韻の文字.{4}一番目の丸括弧の文字と異なる韻の文字
.{4}一番目の丸括弧の文字と同じ韻の文字
```

のように、文字の属性を参照可能な後方参照メタ文字が必要となってくる。同様に、

```
.(.).一番目の丸括弧の文字と同じ意味の文字
```

という正規表現で「何因何縁」などのような定型表現を抽出したりする用途などが考えられる。これまで気づくことのなかったパターンを発見することができれば、テクスト解釈の新たな可能性を開けてくることから、このような後方参照は漢字文献の分析に資するところが多いように思われる。

しかし、現在ある正規表現における後方参照は文字コードに依存したものであるため、このようなパターンマッチを実現することは困難である。原因は正規表現ではなく、たったひとつの数字だけで形・音・義を持つ文字を表そうとする文字コードの貧弱さにある。

これを克服するには、文字コードからの脱却が必要である。CHISE プロジェクト[*3]を参照のこと。また、CHISE プロジェクトに基づく正規表現の拡張については、[3] を参照のこと。

3 XML 文書の操作

3.1 XML の背景にあるモデル

XML は拡張文脈自由文法（Context Free Grammer）に基づいているため、処理もまたその制限の中にある。

そのほか、XML については様々な問題点が指摘されている。ここでは、[4]、[5] をあげるにとどめる。

3.2 操作の種類

XML 文書に対しては、現在、以下のような操作方法が主流である。

1. （リッチな）プレーンテキストとしての操作（エディタによる編集、grep による検索など）
2. プログラミング言語による操作
   (a) 自力（正規表現による置換など）
   (b) 標準 API 等の利用
3. DBMS による操作
   (a) XML ネイティブ DBMS[*4]

[*3] http://www.kanji.zinbun.kyoto-u.ac.jp/projects/chise/
[*4] DBMS: Database Management System の略。データベースを管理するためのシステム。

(b) RDBMS[*5]

このうち、1 と 2a については、正規表現を含む多くの検索・置換システムのほとんどが、テキストを行単位で処理するため、XML の要素の単位で処理することは意外と面倒である。例えば、タグを、

```
<p class="hoge"
>ほげほげ</p>
```

のように書くことは珍しくないため、普通のテキストエディタの置換機能などでは処理することは不可能に近い。Perl などのスクリプト言語で、正規表現を用いて処理をしようという場合には、すべての行をつないでから処理をするなどの工夫が必要である。

3.3 標準的 API を利用した操作

プログラミング言語による操作は、W3C[*6]などが提案、提供する API[*7]や、XML 操作言語を利用する方が、開発も早く効率的である。

現在、以下のような API や言語がよく使われている。

- DOM[*8]
- SAX[*9]
- XSL[*10]
  - XSLT[*11]
  - XPath[*12]
  - XSL-FO[*13]

DOM と SAX は、XML のツリー構造に対するプログラミングが可能な API であり、スタンダードな位置を占めている。DOM は、XML 文書全体のツリー構造をメモリ上に構築するので、文書のどの位置にでも自由にアクセスできる利点があるが、メモリ効率や処理速度の面で若干不利である。一方 SAX は、XML 文書を先頭から順に読み込んでいきながら処理するため、DOM のような自由度はないものの、省メモリかつ高速に処理できる。

XSL は「スタイルシート」とあることからもわかるように、画面表示や印刷などを意識した言語であるが、特に XSLT は、一般的な XML 文書の変形に用いられるようになっている。

[*5] Relational Database Management System の略。Microsoft Access や PostgreSQL など、関係データベース（関係＝表に基づくデータベース）を管理するためのシステム。

[*6] World Wide Web Consortium; http://www.w3.org/

[*7] Application Program Interface; あるプラットフォーム向けのソフトウェアを開発する際に使用できる命令や関数の集合のこと。また、それらを利用するためのプログラム上の手続きを定めた規約の集合。個々のソフトウェアの開発者がソフトウェアの持つすべての機能をプログラミングするのは困難で無駄が多いため、多くのソフトウェアが共通して利用する機能は、OS やミドルウェアなどの形でまとめて提供されている。個々の開発者は規約に従ってその機能を「呼び出す」だけで、自分でプログラミングすることなくその機能を利用したソフトウェアを作成することができる。（IT 用語辞典、http://e-words.jp/w/API.html）

[*8] Document Object Model; http://www.w3.org/DOM/

[*9] Simple API for XML; http://www.saxproject.org/

[*10] Extensible Stylesheet Language; http://www.w3.org/Style/XSL/

[*11] XSL Transformations; http://www.w3.org/TR/xslt

[*12] XML Path Language; http://www.w3.org/TR/xpath

[*13] XSL Formatting Objects; http://www.w3.org/TR/xsl/

これらは、様々なプログラム言語、スクリプト言語のためのインターフェースが開発、公開されている。

3.4 Perl による DOM プログラミング

3.4.1 特定の名前の要素の内容を取り出す

■XML 文書（0251.xml）

```
<?xml version="1.0"?>
<sutra>
<title>般若波羅蜜多心經</title>
<translator>唐三藏法師<person type="chinese">玄奘</person>譯</translator>
<body>
<paragraph><person type="bodhisattva">觀自在菩薩</person>。行深般若波羅蜜多時。照見五蘊皆空。度一切苦厄。<person type="indian">舍利子</person>。色不異空。空不異色。色即是空。空即是色。受想行識亦復如是。<person type="indian">舍利子</person>。是諸法空相。不生不滅。不垢不淨不增不減。是故空中。無色。無受想行識。無眼耳鼻舌身意。無色聲香味觸法。無眼界。乃至無意識界。無無明。亦無無明盡。乃至無老死。亦無老死盡。無苦集滅道。無智亦無得。以無所得故。菩提薩&#x57F5;。依般若波羅蜜多故。心無&#x7F63;礙。無&#x7F63;礙故。無有恐怖。遠離顛倒夢想。究竟涅槃。三世諸佛。依般若波羅蜜多故。得阿耨多羅三藐三菩提。故知般若波羅蜜多。是大神咒。是大明咒是無上咒。是無等等咒。能除一切苦。眞實不虛故。說般若波羅蜜多咒。即說咒曰</paragraph>
<dharani>揭帝揭帝　般羅揭帝　般羅僧揭帝菩提僧莎訶</dharani>
</body>
<title>般若波羅蜜多心經</title>
</sutra>
```

■スクリプト例

```
use XML::DOM;

$parser = new XML::DOM::Parser;
$doc = $parser->parsefile ("0251.xml");

$nodes = $doc->getElementsByTagName("person");
$n = $nodes->getLength;
for $i (0 .. $n - 1) {
    $node = $nodes->item($i);
    $value = $node->getFirstChild()->getData();
    $href = $node->getAttributeNode("type");
    print $href->getValue, "\t", $value, "\n";
}
```

```
$doc->dispose;
```

■結果

```
chinese 玄奘
bodhisattva 觀自在菩薩
indian 舍利子
indian 舍利子
```

3.4.2 特定の値の属性を持つ要素の内容を取り出す

■XML 文書（scenario.xml）

```
<?xml version="1.0"?>
<scenario>
<dialogue who="太郎冠者">「まだ、ZOILIA の土を踏むには、一週間以上かかりましょう。私は、もう、船が飽き飽きしました。」</dialogue>
<dialogue who="次郎冠者">「ゾイリア---ですか。」</dialogue>
<dialogue who="太郎冠者">「さよう、ゾイリア共和国です。」</dialogue>
<dialogue who="次郎冠者">「ゾイリアと云う国がありますか。」</dialogue>
</scenario>
```

■スクリプト例

```
use utf8;
use Encode;
use XML::DOM;

$parser = new XML::DOM::Parser;
$doc = $parser->parsefile ("scenario.xml");

$root = $doc->getDocumentElement();

$nodes = $doc->getElementsByTagName("dialogue");
$n = $nodes->getLength;
for $i (0 .. $n - 1) {
    $node = $nodes->item($i);
    $who = decode('utf-8', $node->getAttributeNode("who")->getValue);
    if ($who eq "太郎冠者") {
        print $node->toString, "\n";
    }
}
```

```
$doc->dispose;
```

■結果

```
<dialogue who="太郎冠者">「まだ、ZOILIA の土を踏むには、一週間以上かかりましょう。私は、もう、船が飽き飽きしました。」</dialogue>
<dialogue who="太郎冠者">「さよう、ゾイリア共和国です。」</dialogue>
```

3.5 DBMS による操作

DBMS を用いて XML を操作する方法がある。上記の方法と比べて、スケーラビリティ、処理速度、堅牢性、複数ユーザへの対応などの面において有利な点が多い。ここではいくつかの製品名を紹介するにとどめる[*14]。

1. XML ネイティブ DBMS
   - NeoCore XMS
   - Tamino
   - Sonic XIS
   - EsTerra XSS
   - Apache Xindice
2. RDBMS
   - Microsoft SQL Server 2000
   - DB2 Universal Database
   - Sybase Adaptive Server Enterprise
   - Oracle XML DB

XML ネイティブ DBMS は、半構造的なデータを扱える XML のツリー構造をそのまま扱える利点があるが、まだ新しい技術であるため、スケーラビリティ、処理速度、堅牢性、複数ユーザへの対応などの面では RDBMS に一日の長がある。

一方、RDBMS による XML の操作は、上記の利点のほかにも既存の RDB との連携がとりやすいなどの利点があるが、データ構造の制限が XML ネイティブ DBMS よりもきつい場合が多く、文書型の XML を扱うには不利な場合がある[*15]。

参考文献

[1] Jeffrey E. F. Friedl. *Mastering Regular Expressions*, 2nd Edition. O'Reilly. 2002. 田和勝訳『詳説正規表現　第 2 版』（オライリー・ジャパン、2003）

[*14] 以下にあげたものは [7] を参照した。

[*15] この問題は、XML 文書を定義するスキーマ言語の設計思想の対立にも見て取ることができる。[2] 参照。

[2] 川俣晶「XML における「ボヘミアンと貴族の階級闘争」を読み解く」(http://www.atmarkit.co.jp/fxml/tanpatsu/24bohem/01.html、2003)
[3] 師茂樹「Perl/CHISE による正規表現の拡張の試み —文字素性による後方参照の実装実験と課題—」(『Linux Conference 抄録集』1、2003、http://lc.linux.or.jp/paper/lc2003/)
[4] Allen Renear, Elli Mylonas, David Durand. "Refining our Notion of What Text Really Is: The Problem of Overlapping Hierarchies." (Final version, January 6, 1993. http://www.stg.brown.edu/resources/stg/monographs/ohco.html)
[5] 豊島正之「XML の骨抜き利用法　アジア・アフリカ言語文化研究所データベースの例」(http://www.classics.jp/Contents/Assets/notice/xmt-main.pdf, 2001)
[6] The Unicode Consortium. *The Unicode Standard, Version 4.0.* Addison-Wesley. 2003.
[7] 山田祥寛「XML データベース製品カタログ 2003」(http://www.atmarkit.co.jp/fxml/indexes/index_dev.html#dbcata)