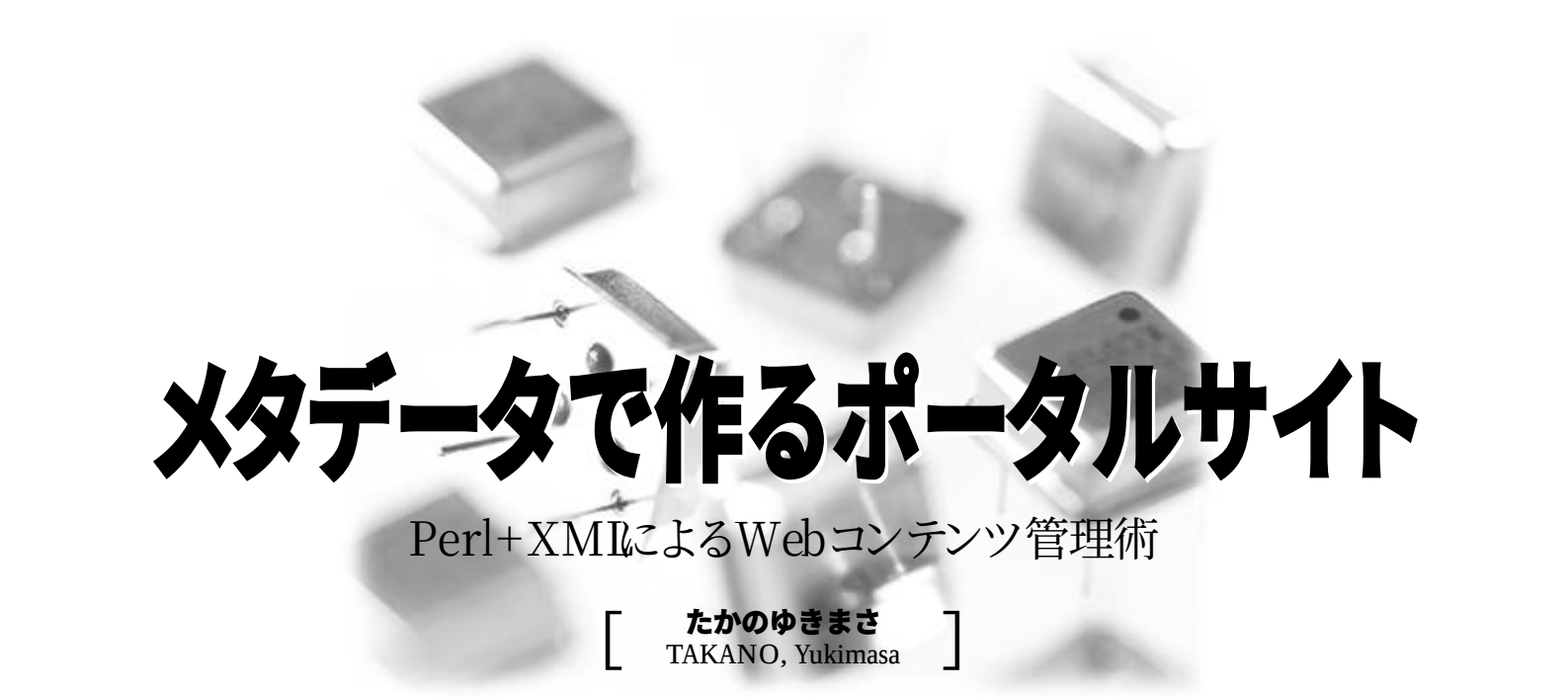

メタデータで作るポータルサイト

Perl+XMLによるWebコンテンツ管理術

[たかのゆきまさ
TAKANO, Yukimasa]

PerlでXMLを扱ってみよう
Webサーバー上でもっとも多く使われているプログラミング言語、
Perlを使ったXMLアプリケーションを解説する。
豊富なリソースとドキュメントを足がかりに、XMLがさらに身近に。

はじめに

XMLが産声を上げた日を、1998年2月10日（XML 1.0 勧告）とするならば、それからすでに2年以上の日々が経過している。この間、XMLは確かに一部の間でもてはやされ、着実に利用されるようになってきたが、その一方でXMLを疑問視する声も聞かれるようになった。否定的な意見としては、次の3通りに分類されるのではないかと思う。

1.仕様が氾濫していること

XMLには基本仕様のXML 1.0のほか、関連仕様として（勧告済みのものだけでも）DOM Level 1、ネームスペース、RDF、XSLT、XPath 1.0とあり、これらのどこまで追従すればよいのか見極めがしづらい。

2.規格化（DTDの書き起こし）に多大な労力を必要とすること

DTDを書き起こして規格化するということは、言い換えれば文書を紋切り型で一定の書式におさめるということである。あまりにもガチガチに項目を決めてしまうと、何度も規格を変更するハメになる。かといって「その他1／その他2」のような、ルーズな項目の切り方をしていては、既存の（XML化されていない）文書同様、機械的に処理することが難しくなり、XML化する意義そのものを疑われかねない。

3.処理系実装の負担

現在普及しているXML処理系の多くはJavaで実装されている。もともとJavaでシステム構築を業務としていればすんなり移行できるが、そうではない場合、独自のXMLパーサ（あるいはその類似品）を書き起こすハメになり、実装するための作業的負担が大きい。

このような理由で、XML化を見送っている現場は少なくないのではないだろうか。

しかし、XMLの導入には利点がある。ここで改めて触れるまでもなく、XMLは「人間にも読め（human-readable）」、なおかつ「機械処理がしやすい（machine-understandable）」データ記述手法だ。内部実装までこれに縛られる必要はまったくないが、データ出力・データ交換・データの保存に際しては、バイトコードや独自フォーマットのバイナリファイルよりは、はるかに流用性が高い。

筆者はこの利害のジレンマから、なんとかXML導入の糸口をつかもうと、Perlや外部コマンドを利用したアプローチを模索した。この際留意したことは、

1.処理系の構築に時間をかけないこと
2.処理系の内部実装に凝らないこと
3.妥協点を見いだすこと

の3点である。つまり、プログラムの流用性よりも、文書の流

用性を優先させた。

メインプラットフォームとしてPerlを選択したのは、筆者が主に業務で利用している言語だからにすぎない。少なくとも、プログラミングをしたことのない人にとっては、比較的取り組みやすいアプローチだとは思うが、「XMLを扱うにはPerlが良い」と思われてしまうことのないよう、あらかじめお断りしておく。

この記事は、筆者が模索の末に得た、現時点における手法を紹介するものである。標準的なアプローチとは異なる部分もあるかもしれないが、あくまで一例として読み進めていただきたい。

PerlによるXML処理のアプローチ

PerlでのXML処理系を調べるために、まず筆者はCPANのXMLディレクトリを参照した。

http://www.cpan.org/modules/by-module/XML/
http://www.ring.gr.jp/pub/lang/perl/CPAN/modules/by-module/XML/（国内ミラー）

ディレクトリには多くのXML関連モジュールが存在している。さらに、

http://www.cpan.org/modules/by-module/XML/perl-xml-modules.html

に、XML関連のモジュールについてのリストがある。

モジュールの中には、特定のXML文書に特化した処理を提供するものもあるようだが、筆者の場合、

・XMLの派生規格（DOMなど）について理解が足りないこと
・とにかくXML文書を処理できればよいこと

という2点から、XML::Parserモジュールを直接利用することにした。本稿執筆時点のバージョンは2.28である。

調べてみると、XML::ParserモジュールはXML文書がDTDにしたがっているかを構文チェック（validation）する機能を備えていないらしい。そこで、構文チェック機能を補助するコマンドツールを探したところ、rxpというものがあった。

http://www.cogsci.ed.ac.uk/~richard/rxp.html

rxpは、もともとはC言語による構文解析機能（parser）を提供するライブラリと、テキスト処理のためのコマンドツールを備えたものらしい。が、-Vオプションをつけることで構文チェックを行なってくれる。また、-sオプションをつけることで、出力を抑止できる。これを構文チェックのみに利用することにした。rxpの本稿執筆時点でのバージョンは1.2pre1である。

PerlのXMLパーサ、XML::Parserを使う

PerlのXMLパーサ“XML::Parser”モジュールは、James Clarkの書いた、C版のexpatをベースにしている。基本的な使い方は、リスト1のようなスクリプトを用いる。

リスト1：XML::Parserを使う

```
use XML::Parser;

$parser = new XML::Parser(Handlers => {
                          Start => \&handle_start,
                          End   => ￥&handle_end,
                          Char  => ￥&handle_char});
$parser->parsefile('Sample1.xml');

sub handle_start
{
   # タグ開始時の処理を記述 ...
}

sub handle_end
{
   # タグ終了時の処理を記述 ...
}

sub handle_char
{
   # 文字列取得時の処理を記述 ...
}
```

最初のnew XML::Parserでは、新規にXML::Parserオブジェクトを生成し、関数handle_start、handle_end、handle_charをハンドラとして登録している。ハンドラとは特定のイベントが発生したときに呼ばれる処理のことである。この場合は、タグの開始・終了や文字列に遭遇したときに、独自の処理をさせるためにハンドラを定義している。

サンプルとして簡単なXMLドキュメントSample1.xmlを構文解析してみよう。内容はリスト2のとおりである。

このドキュメントをparsefile()メソッドから呼び出した場合、ハンドラとなっている関数はリスト3のようなタイミングで呼び出される。

グレーで示した部分が、解析中に遭遇した「タグの開始・タグの終了・テキスト」という「イベント」に相当する。XML文書中の部品ごとに、バラバラに関数が呼び出されているのがおわかりいただけるだろうか。

このような手法をコールバック方式と呼ぶ。これはもともとC言語による手法であり、「必要な処理だけをコーディングすればよく、開発効率が上がる」という利点がある。

しかし上記の例のように、「関数が呼び出されるタイミングが煩雑になりやすい」という欠点がある。Perlを主言語にしている筆者にはわかりづらい。

幸いなことに、XML::Parserモジュールでは「基本的なコールバック形式のパーサ」以外の実装も持っており、“ Style ” 指定によって切り替えることができる。

以下はXML::ParserのPerlドキュメント（perldoc）から抜

リスト2： Sample1.xml

```
<?xml version="1.0" encoding="EUC-JP"?>
<!DOCTYPE sample1 [
<!ELEMENT sample1 (header, text)>
<!ATTLIST sample1
      version        CDATA   #REQUIRED
>

<!ELEMENT header (#PCDATA)>
<!ELEMENT text (#PCDATA)>
]>

<sample1 version="1.00">

<header>これがヘッダです</header>

<text>
この部分がテキストになります。
ここは2行目です。
</text>

</sample1>
```

リスト3： リスト1をparsefile()メソッドから呼び出したときに呼び出される関数とそのタイミング

```
$parser->parsefile('Sample1.xml');

<sample1>                          → handle_start();

（スペース・改行）                  → handle_char();

<header>                           → handle_start();

「これがヘッダです」                → handle_char();

</header>                          → handle_end();

（スペース・改行）                  → handle_char();

<text>                             → handle_start();

「この部分がテキストになります。
   ここは2行目です。」              → handle_char();

</text>                            → handle_end();

（スペース・改行）                  → handle_char();

</sample1>                         → handle_end();

（$parser->parsefile('Sample1.xml')メソッド終了）
```

粋したものである。筆者もすべてを試したわけではないことをここにお断りしておく。詳細はXML::Parserのドキュメントを参照して欲しい。

◆スタイル：Debug

これはXMLドキュメントをアウトライン形式で表示（出力）する。解析結果としては、とくに返り値をもたない。

◆スタイル：Subs

要素が開始するたびに、その要素名を関数名に持つ関数（sub）が呼ばれる。また、要素が終了するたびに、要素名にアンダースコアをつけた関数が呼ばれる。

たとえば、

```
<TEXT>これはテキストです。</TEXT>
```

というドキュメントであれば、<TEXT>タグの開始時に

```
sub TEXT {}
```

という関数を呼び、終了時(</TEXT>)に

```
sub _TEXT {}
```

という関数を呼ぶ。返り値はとくになし。

◆スタイル：Tree

ドキュメントの解析結果を、ツリー形式で返す。各ノードごとに、タグと内容（content）のペアで返す。ノードがテキストの場合、ダミーのタグ名として"0"を埋め、実際の文字列とのペアとなる。タグが要素（element）の場合、内容は配列への参照になり、配列の先頭には属性(attributes)がハッシュ変数の参照として乗せられる。

◆スタイル：Objects

Treeとよく似ているが、各要素ごとにハッシュオブジェクトが生成される。

◆スタイル：Stream

これは、コメントや宣言などを取り除きながらドキュメントを出力するために使われる。

ドキュメントを構文解析しながら、StartDocumentStartTagEndTagText,PIEndDocumentといった関数が適宜呼び出される。それぞれの関数の第一パラメータには、解析中のドキュメントを持つExpatインスタンスが渡される。

さて、この中でわかりづらいのがTreeとObjectsだ。

XML::Parserのドキュメントにはもう少し詳しく書いてあるのだが、それでもハッシュ変数や配列変数の「参照」を扱ったことのない人にはわかりづらい。なにより、この返されたツリー形式の変数のかたまりから、どうやって目的の要素や属性を取り出したらいいのか、途方にくれてしまうだろう。

このふたつのスタイルに対する理解を深めるため、筆者はReferenceTree.pmというPerlモジュールを作成した。TreeXML.plは、これを使うためのサンプルプログラムで、以下のようにコマンド入力する。

```
$ ./TreeXML.pl Sample1.xml ...①
$ ./TreeXML.pl http://snap.shot.cx/sample/SiteRSS.rdf ...②
```

①は、ローカルファイルのXMLドキュメントを構文解析してツリー表示する。②は、リモートのウェブサーバー上のXMLドキュメントを取得し、構文解析してツリー表示する。

TreeXML.plには、表1のPerlモジュールが必要となる。CPANのサイトなどを参考に、適宜インストールして欲しい。

EasyXMLモジュールは、XMLドキュメントを容易に扱うために筆者が書いたPerlモジュールで、内部的にはXML::Parserモジュール、及びLWPモジュールを呼び出している。これについては次節で触れる。

先に挙げたXML文書Sample1.xmlを、TreeXML.plによって構文解析してみよう。リスト4のとおり表示されただろうか。

これは、XML::Parserモジュールを、スタイル「Tree」として呼び出したときの結果をツリー表示したものだ。

表1：TreeXML.plが必要とするモジュールのリスト

モジュール	備考
XML::Parser	
LWP	libwww-perl5
Jcode	
EasyXML	本記事に添付
ReferenceTree	本記事に添付