
Ruby と XML

吉田正人

ドコモ・システムズ(株)

`yoshidam@yoshidam.net`
`yoshidam@docomo-sys.co.jp`

目次

- XML プログラミングの概要
- XMLParser ライブラリの実装について
- DOM ライブラリの実装について
- XMLParser ライブラリの使い方
- DOM ライブラリの使い方
- もう少し実用的なプログラムの例
- 今後について
- 付録

XML プログラミングの概要 (1)

XML プログラミングのための API

■ イベント形式

- SAX (Simple API for XML)
- PYX (ESIS)

■ オブジェクト形式

- DOM (Document Object Model)

XML プログラミングの概要 (1-1)

SAX (<http://www.megginson.com/SAX/>):

- イベント (要素開始, 終了, 文字データ, 処理命令など) に対するハンドラを定義する
- ほとんどの Java の XML パーサで対応している

SAX API の一部 (Java)

```
public interface DocumentHandler {
    public abstract void startDocument();
    public abstract void endDocument();
    public abstract void startElement(String name, AttributeList atts);
    public abstract void endElement(String name);
    public abstract void characters(char ch[], int start, int length);
    public abstract void ignorableWhitespace(char ch[], int start, int
length);
    public abstract void processingInstruction(String target, String data);
}
```

XML プログラミングの概要 (1-2)

PYX (<http://www.xml.com/pub/a/2000/03/15/feature/index.html>):

- イベントを行指向の記法で出力する
- SGML のプログラミングでよく使われていた

XML ファイル

```
<test attr1="test">
<hode>
  ほげ
</hode>
</test>
```

PYX 記法

```
(test
Aattr1 test
-\n
(hoge
-\n ほげ\n
)hode
-\n
)test
```

XML プログラミングの概要 (1-3)

DOM (<http://www.w3.org/TR/REC-DOM-Level-1>):

- IDL で定義されたオブジェクトモデル
- W3C 標準

DOM のインターフェイスの例 (OMG IDL)

```
interface Node {
  readonly attribute DOMString      nodeName;
  attribute          DOMString      nodeValue;
  readonly attribute unsigned short  nodeType;
  readonly attribute Node            parentNode;
  readonly attribute NodeList       childNodes;
  readonly attribute Node            firstChild;
  readonly attribute Node            lastChild;
  readonly attribute Node            previousSibling;
  readonly attribute Node            nextSibling;
  Node insertBefore(in Node newChild,
                   in Node refChild)
    raises(DOMException);
  Node appendChild(in Node newChild)
    raises(DOMException);
```

XML プログラミングの概要 (2)

Ruby の利点

- お手軽
 - コンパイル不要
 - 変数宣言不要
 - メモリ管理不要
- 強力な文字列処理能力
 - Perl と同等の正規表現
 - 日本語, UTF-8 の処理が可能
- 読みやすい, 書きやすい

XMLParser ライブラリの実装について (1)

expat とは

- SP の作者である James Clark 氏が作った C 言語による高速 XML パーサライブラリ (<http://www.jclark.com/xml/expat.html>)
- 現在はバージョン 2.0 が expat プロジェクト (<http://sourceforge.net/projects/expat/>) によって開発中

expat の特徴

- 高速
- validating は行わない
- イベント方式の API
- アプリケーションやライブラリへの組込みが容易 (Mozilla, Perl, Python, Tcl, PHP3 / PHP4, w3c-libwww, Apache)

XMLParser ライブラリの実装について (1-1)

expat API の例 (C 言語)

```
XML_Parser XMLPARSEAPI
XML_ParserCreate(const XML_Char *encoding);

int XMLPARSEAPI
XML_Parse(XML_Parser parser, const char *s, int len, int isFinal);

void XMLPARSEAPI
XML_SetElementHandler(XML_Parser parser,
                      XML_StartElementHandler start,
                      XML_EndElementHandler end);

void XMLPARSEAPI
XML_SetCharacterDataHandler(XML_Parser parser,
                             XML_CharacterDataHandler handler);

void XMLPARSEAPI
XML_SetCommentHandler(XML_Parser parser,
                      XML_CommentHandler handler);
```

XMLParser ライブラリの実装について (2)

基本方針

- できるだけ簡略に記述できること
- expat の高速さを生かすこと
 - イベントハンドラ無しの場合は expat そのもののスピードになるように
- expat のすべての機能が扱えること
 - ただし , Ruby では不要な機能は除く

XMLParser ライブラリの実装について (3)

Parser API について

■ イベントハンドラ・インタフェース

- XML::Parser クラスにメソッド定義することによりイベントハンドラを定義する
- メソッド定義しなかったイベントは処理しない
- イベント駆動プログラム

■ イテレータ・インターフェイス

- XMLParser#parse メソッドをイテレータとして使用する
- ループ処理のように記述できる
- イベント駆動プログラムに慣れていなくても使える
- 基本イベント以外はダミーメソッドの定義が必要

■ SAX インターフェイス

XMLParser ライブラリの実装について (3-1)

イベントハンドラ・インターフェイスの実装

```
static void
myStartElementHandler(void *recv,
                      const XML_Char *name, const XML_Char **atts)
{
    VALUE attrhash;

    attrhash = rb_hash_new();
    while (*atts) {
        const char* key = *atts++;
        const char* val = *atts++;
        rb_hash_aset(attrhash, rb_str_new2((char*)key), rb_str_new2((char*)val));
    }
    rb_funcall((VALUE)recv, id_startElementHandler, 2,
              rb_str_new2((char*)name), attrhash);
}
```

XMLParser ライブラリの実装について (3-2)

イベントハンドラ・インターフェイスの実装 (続き)

```
id_startElementHandler = rb_intern("startElement");
id_endElementHandler = rb_intern("endElement");

if (rb_method_boundp(CLASS_OF(obj), id_startElementHandler, 0))
    start = myStartElementHandler;
if (rb_method_boundp(CLASS_OF(obj), id_endElementHandler, 0))
    end = myEndElementHandler;
if (start || end)
    XML_SetElementHandler(parser->parser, start, end);
```

XMLParser ライブラリの実装について (3-3)

イベントハンドラのメソッド名

メソッド名	イベント
startElement	element start tag
endElement	element end tag
character	character data
processingInstruction	processing instruction
unparsedEntityDecl	unparsed entity declaration
notationDecl	notation declaration
externalEntityRef	external entity reference
comment	comment
startCdata	CDATA section start
endCdata	CDATA section end
startNamespaceDecl	Namespace declaration start
endNamespaceDecl	Namespace declaration end
startDoctypeDecl	DOCTYPE declaration start
endDoctypeDecl	DOCTYPE declaration end
notStandalone	document is not standalone
default	other data
defaultExpand	same as default
unknownEncoding	unknown character encoding

XMLParser ライブラリの実装について (3-4)

イテレータ・インターフェイスの実装

```
static void
iterStartElementHandler(void *recv,
                        const XML_Char *name, const XML_Char **atts)
{
    VALUE attrhash;

    attrhash = rb_hash_new();
    while (*atts) {
        const char* key = *atts++;
        const char* val = *atts++;
        rb_hash_aset(attrhash, rb_str_new2((char*)key), rb_str_new2((char*)val));
    }

    rb_yield(rb_ary_new3(3, INT2FIX(XML_START_ELEM),
                        rb_str_new2((char*)name), attrhash));
}
```

XMLParser ライブラリの実装について (3-5)

イテレータ・インターフェイスの実装 (続き)

```
parser->iterator = rb_iterator_p();
if (parser->iterator) {
    XML_SetElementHandler(parser->parser,
                        iterStartElementHandler, iterEndElementHandler);
} ...
```

XMLParser ライブラリの実装について (3-6)

イテレータブロックへの引数

第一引数 (イベントタイプ)	第二引数	第三引数
START_ELEM	element name	hash of attributes
END_ELEM	element name	nil
CDATA	nil	string
PI	PI name	string
UNPARSED_ENTITY_DECL	entity name	array
NOTATION_DECL	notation name	array
EXTERNAL_ENTITY_REF	entity names	array
COMMENT	nil	string
START_CDATA	nil	nil
END_CDATA	nil	nil
START_NAMESPACE_DECL	prefix	URI
END_NAMESPACE_DECL	prefix	nil
START_DOCTYPE_DECL	doctype name	nil
END_DOCTYPE_DECL	nil	nil
DEFAULT	nil	string

XMLParser ライブラリの実装について (4)

文字エンコーディング対応

■ パース前にエンコーディング変換

- ISO-2022-JP はこの方法でのみ対応可能

■ XML::Encoding クラス

- expat の UnknownEncoding イベントでパーサにコンバータを登録
- Ruby を使ったエンコーディングコンバータも可能

■ Perl エンコーディングマップの流用

- Perl の XML::Parser モジュールの機能を流用
- Perl の XML::Encoding モジュールを利用し、マッピングテーブル作成
- EUC-JP , Shift_JIS 等が利用可能

XMLParser ライブラリの実装について (4-1)

パース前にエンコーディング変換

```
jis = open("sample.xml").read
utf8 = Uconv.euctou8(NKF.nkf("-Je", jis))
XML::Parser.new("UTF-8").parse(utf8)
```

XMLParser ライブラリの実装について (4-2)

XML::Encodingクラス

```
class EUCHandler
  def map(i)
    return i if i < 128
    return -1 if i < 160 or i == 255
    return -2
  end
  def convert(s)
    Uconv.euctou2(s)
  end
end

def unknownEncoding(name)
  return EUCHandler.new if name =~ /^euc-jp$/i
  nil
end
```

XMLParser ライブラリの実装について (4-3)

Perlエンコーディングマップの流用

```
big5.enc
euc-kr.enc
iso-8859-2.enc
iso-8859-3.enc
iso-8859-4.enc
iso-8859-5.enc
iso-8859-7.enc
iso-8859-8.enc
iso-8859-9.enc
windows-1250.enc
x-euc-jp-jisx0221.enc
x-euc-jp-unicode.enc
x-sjis-cp932.enc
x-sjis-jdk117.enc
x-sjis-jisx0221.enc
x-sjis-unicode.enc
```

Shift_JIS と EUC-JP が使えないのは不便なので...

```
shift_jis.enc (x-sjis-cp932.enc と同じ)
euc-jp.enc    (x-euc-jp-unicode.enc と同じ)
```

DOM ライブラリの実装について

■ XML::DOM クラス

- DOM level1 Core API にほぼ準拠
- 一部未実装 (Document#implementationなど)
- 文字列型非互換 (wstringではなく、多バイト文字列)
- Ruby 向けのメソッド、イテレータなどを含む
- XPointer (WD) 対応 福嶋正機氏による

■ Visotor モジュール

- XML::DOM クラスで Visotor パターンを使うためのモジュール
- Perl 用 XML::Grove::Visitor とほぼ同等

XMLParser ライブラリの使い方 (1)

イベントハンドラの使用例

要素開始タグの数を数えて、各要素の使用回数を調べる。

```
require 'xmlparser'

class NewParser < XML::Parser
  attr :elemNames

  def initialize
    @elemNames = {}
  end

  def startElement(name, attrs)
    if @elemNames[name].nil?
      @elemNames[name] = 1
    else
      @elemNames[name] += 1
    end
  end
end
```

XMLParser ライブラリの使い方 (2)

イベントハンドラの使用例(続き)

```
parser = NewParser.new

begin
  parser.parse($<.read)
rescue XML::ParserError
  print "Error: #{$!} in line #{parser.line}\n"
end

p parser.elemNames
```

XMLParser ライブラリの使い方 (3)

イテレータの使用例

```
require 'xmlparser'
parser = XML::Parser.new
elemNames = {}
begin
  parser.parse($<.read) do |type, name, data|
    case (type)
    when XMLParser::START_ELEM
      if elemNames[name].nil?
        elemNames[name] = 1
      else
        elemNames[name] += 1
      end
    end
  end
end
## print result
p elemNames
rescue XML::ParserError
  print "Error: #{!} in line #{parser.line}\n"
end
```

DOM ライブラリの使い方 (1)

DOM と Visitor の使用例

```
require 'xmlltreebuilder'
require 'xmlltreevisitor'

class Counter < XML::DOM::Visitor
  attr :elemNames
  def initialize
    @elemNames = {}
  end
  def visit_Element(elem)
    name = elem.nodeName
    if @elemNames[name].nil?
      @elemNames[name] = 1
    else
      @elemNames[name] += 1
    end
    super
  end
end
```

DOM ライブラリの使い方 (2)

DOM と Visitor の使用例(続き)

```
parser = XML::DOM::Builder.new

begin
  tree = parser.parse($<.read)
rescue XML::ParserError
  print "Error: #{$_} in line #{parser.line}\n"
end

tree.documentElement.accept(c = Counter.new)
p c.elemNames
```

もう少し実用的なプログラムの例 (1)

DocBook (もどき) を HTML に変換する。

```
<article lang="ja">
  <articleinfo>
    <title>テスト文書</title>
    <abstract><para>DocBook から HTML に変換するテストです。</para></abstract>
  </articleinfo>
  <sect1 label="section1">
    <title>セクション1</title>
    <itemizedlist>
      <listitem>
        <para><ulink url="hoge.html">ほげ</ulink></para>
      </listitem>
    </itemizedlist>
  </sect1>
  <sect1 label="section2">
    <title>セクション2</title>
    <sect2>
      <title>サブセクション2-1</title>
      <para>サブセクション2-1 です。</para>
    </sect2>
  </sect1>
</article>
```

もう少し実用的なプログラムの例 (2)

こんな感じに...

```
<HTML lang="ja">
  <HEAD><TITLE>テスト文書</TITLE></HEAD>
  <BODY>
    <H1>テスト文書</H1>
    <P>DocBook から HTML に変換するテストです。</P>
    <DIV class="section"><A name="section1"></A>
      <H2>1 セクション1</H2>
      <UL>
        <LI><P><A href="hoge.html">ほげ</A></P></LI>
      </UL>
    </DIV>
    <HR>
    <DIV class="section"><A name="section2"></A>
      <H2>2 セクション2</H2>
      <DIV class="subsection">
        <H3>サブセクション2-1</H3>
        <P>サブセクション2-1 です。</P>
      </DIV>
    </DIV>
    <HR>
  </BODY>
</HTML>
```

もう少し実用的なプログラムの例 (3)

要素名の置き換え

<article>	<HTML>
<emphasis>	
<itemizedlist>	
<listitem>	
<para>	<P>
<sect1 label="label">	<DIV class="section">
<sect2>	<DIV class="subsection">
<articleinfo><title>	<H1>
<sect1><title>	<H2>
<sect2><title>	<H3>
<ulink url="url">	

もう少し実用的なプログラムの例 (3)

目次も作ってみる

```
<sect1 label="label1">
  <title>Title1</title>
</sect1>
<sect1 label="label2">
  <title>Title2</title>
</sect1>
```

```
<OL>
  <LI><A href="#label1">Title1</A></LI>
  <LI><A href="#label2">Title2</A></LI>
</OL>
```

もう少し実用的なプログラムの例 (4)

プログラムの設計

- 1-パスでは目次作成が困難
- DOM ツリーから DOM ツリーへの変換ではかなり遅い

したがって...

1. イベントハンドラで要素名を置き換えつつ HTML DOM ツリー作成
2. HTML DOM ツリーを検索して目次を作成
3. HTML ファイル出力 (EUC-JP に変換できない文字は文字参照に変換)

もう少し実用的なプログラムの例 (5)

1. イベントハンドラで要素名を置き換えつつ HTML DOM ツリー作成

```
class DocBook2HTML < XML::Parser
  def initialize
    @current = []
    @ret = HTML::HTMLDocument.new
  end

  def startElement(name, attrs)
    case name
    when "article"
      html = @ret.createHTMLElement('HTML')
      html.appendChild(_createHTMLHeader(@ret))
      body = @ret.createHTMLElement('BODY')
      html.appendChild(body)
      @ret.appendChild(html)
      @current.push(body)
    when "emphasis"
      em = @ret.createHTMLElement("EM")
      @current[-1].appendChild(em)
      @current.push(em)
    end
  end
end
```

もう少し実用的なプログラムの例 (6)

1. イベントハンドラで要素名を置き換えつつ HTML DOM ツリー作成 (続き)

```
def endElement(name)
  case name
  when "article"
    @current[-1].appendChild(_createHTMLFooter(@ret))
  when "emphasis"
    @current.pop
  end
end

def character(data)
  @current[-1].appendChild(@ret.createTextNode(data))
end
```

もう少し実用的なプログラムの例 (7)

2. HTML DOM ツリーを検索して目次を作成

```
def _createIndex(indexNode)
  ol = @ret.createHTMLElement("OL")
  indexNode.appendChild(ol)

  section = 1
  @ret.getNodesByXPath('/HTML/BODY/DIV[@class="section"]').each do |n|
    label = n.getElementsByTagName("A")[0].attributes['name'].nodeValue
    text = n.getElementsByTagName("H2")[0].childNodes[0].nodeValue
    li = @ret.createHTMLElement("LI")
    a = @ret.createHTMLElement("A", {'href'=>'#' + label}, text)
    li.appendChild(a)
    ol.appendChild(li)

    section += 1
  end
end
```

もう少し実用的なプログラムの例 (8)

3. HTML ファイル出力

```
def getHTML
  case @charset
  when /ISO-8859-1/i
    Uconv.u8tolatin1(@ret.to_s)
  when /EUC-JP/i
    Uconv.u8toeuc(@ret.to_s)
  end
end

class <<Uconv
  def unknown_unicode_handler(u)
    return '&#' + u.to_s + ';'
  end

  def u8tolatin1(str)
    ret = ''
    str.unpack('U*').each do |c|
      if c < 256; ret << c; else ret << '&#' + c.to_s + ';'; end
    end
    ret
  end
end
```

今後について

将来の予定

- expat 2.0 対応
- ドキュメント整備
 - プログラミングガイド, チュートリアルなど
- 最新標準に準拠
 - XPath
 - DOM Level2 (Core , Traversal)
 - SAX2

付録

配布場所 (この資料やサンプルプログラムの全体も置く予定...)

- http://www.yoshidam.net/Ruby_ja.html