

# RubyのXML対応

吉田正人

INSエンジニアリング(株)

`yoshidam@yoshidam.net`

`yoshidam@inse.co.jp`

# 目次

---

---

- Rubyの紹介
- Expatの紹介
- 設計と開発
- RubyによるXMLプログラミング
- XMLParserモジュールの今後

# Rubyの紹介

---

---

Rubyとは

- まつもとゆきひろ氏によって設計されたシンプルかつ強力なオブジェクト指向スクリプト言語

# Rubyの紹介

## Rubyの特徴

- インタプリタ
  - コンパイル不要
- 型宣言が不要
  - 変数には型が無い
- 書きやすく読みやすい文法
  - Perlのような暗号プログラムになりにくい
- システムプログラミング, ネットワークプログラミング
  - Perlと同等
- ポータブル
  - Unix, Windows, Macintosh, BeOS, MS-DOS, X68000 など

# Rubyの紹介

## Rubyの特徴(続き)

- 全てがオブジェクト
  - Integer, Stringなどの基本型もオブジェクト
- メモリ管理が不要
  - GC
- さまざまな拡張ライブラリ
  - DBM, Kconv, Tk, PostgreSQL など
- イテレータ
- 例外処理機能
- Cによる拡張が容易
  - Perl XSよりはるかに簡単

# Rubyの紹介

## Rubyの特徴(続き)

- 組み込みの安全チェック
- スレッド機能
  - カーネルスレッドではない (MS-DOSでも使える)
- テキスト処理機能
  - Perlと同等の正規表現など
- 日本語処理(多バイト文字列処理)
  - EUC-JP, Shift\_JIS, UTF-8対応
  - 変数, メソッド名, リテラル, 正規表現, 文字列処理

# Rubyの紹介

## なぜRubyでXML?

- 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
- Webアプリケーション
  - cgi-lib, mod\_ruby

# Expatの紹介

---

---

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

# Expatの紹介

## Expatの特徴

- 高速
- validatingは行わない
- イベント方式のAPI
- アプリケーションへの組み込みが容易
  - Perl
  - Python
  - Tcl
  - PHP3
  - Mozilla

# Expatの紹介

## Expatの特徴(続き)

- **さまざまな文字エンコーディングに対応可能**
  - 標準ではUS-ASCII, UTF-8, UTF-16, ISO-8859-1に対応
  - 拡張可能 (以下の条件を満たすエンコーディングのみ)
    - ▶ \$@\'{}~以外のASCII文字はASCIIと同じ
    - ▶ 1文字が4バイトを越えない
    - ▶ すべての文字が0xFFFF以下のUnicode値を持つ
    - ▶ 一つのUnicode文字はただ一通りのバイト列に変換される
  - **ただし, 出力はUTF-8のみ**

# 設計と開発

## 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

# 設計と開発

## 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
- やや遅い(不要なイベントも捕捉してしまうため)
- 基本イベント以外はダミーメソッドの定義が必要

# 設計と開発

## 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

# 設計と開発

## XMLツリーAPI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対応

# 設計と開発

## XPointerサポート --- 福嶋正機氏による

### □ サポートする機能

- root, origin, html, child, descendant, ancestor, preceding, following, psibling, fsibling, attr
- Element typeによる選択
- Instance number('all'を含む)による選択
- 属性による選択

### □ 不完全サポートの機能

- id (ID属性未対応のため属性名による選択)

### □ 未サポートの機能

- span, string

# 設計と開発

## その他

### □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

### □Uconvモジュール

- EUC-JPとUTF-16, UTF-8, UCS-4の相互変換モジュール
- XML用に特化していない
- Kconvモジュール(Ruby標準添付)との組合せでShift\_JIS, ISO-2022-JPにも対応可能

# RubyによるXMLプログラミング

## イベントハンドラの使用例

```
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
```

# RubyによるXMLプログラミング

## 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
```

# RubyによるXMLプログラミング

## 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
  ## print result
  p elemNames
rescue XML::ParserError
  print "Error: #{!} in line #{parser.line}\n"
end
```

# RubyによるXMLプログラミング

## DOMとVisitorの使用例

```
require 'xmldb'
require 'xmldbvisitor'

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
  end
end
```

# RubyによるXMLプログラミング

## 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
```

# XMLParserモジュールの今後

## 現在の問題点について

### □ドキュメント不足

- 用意されているのは XML::ParserクラスのAPIリファレンスのみ
- XML::DOMクラスにドキュメントが無い
  - ▶DOM準拠/非準拠が不明確
  - ▶XPointer拡張などの使い方が明確ではない

### □APIの問題

- 機能拡張を繰り返したため、複雑化している
- SAX(Simple API for XML)のような標準的APIと互換性がない

# XMLParserモジュールの今後

## 現在の問題点について(続き)

- 不完全なUnicode対応
  - wstringとして処理できない
  - EUC-JP/Shift\_JISへの変換(どの変換表を採用するか)
- 速度(DOM インタフェイス)
  - DOMのCライブラリ化?
  - XML::Grove?

# XMLParserモジュールの今後

## 将来の予定

### □ドキュメント整備

- XML::DOMマニュアル
- プログラミングガイド, チュートリアルなど

### □API見直し

- Ruby2以降(?)
- SAX対応

### □Expat改造

- 属性がデフォルトかどうか
- DTDの情報取得(?)

# XMLParserモジュールの今後

## 将来の予定(続き)

- Unicode対応
  - よりよいUTF-8対応 (jcode.rbのUTF-8化?)
  - WStringクラス(?)
  - Uconv再設計 (XML対応, 変換表カスタマイズ)

# 付録

## □ Rubyに関する情報

- ホームページ

  - ▷ <http://www.netlab.co.jp/ruby/jp/>

- メーリングリスト

  - ▷ ruby-list

  - ▷ ruby-dev (Rubyのバグ, 仕様など)

## □ Expatに関する情報

- ホームページ

  - ▷ <http://www.jclark.com/xml/expat.html>

- メーリングリスト

  - ▷ xml-dev

## □ XMLParserモジュールに関する情報

- ホームページ

  - ▷ [http://www.yoshidam.net/Ruby\\_ja.html](http://www.yoshidam.net/Ruby_ja.html)