XML を採用した
分散コンピューティングプロトコルの構築

奥井 康弘（(株)日本ユニテック）

村田 真（富士ゼロックス情報システム(株)）

1999 年 12 月 16 日

Internet Week 99 パシフィコ横浜

（社）日本ネットワークインフォメーションセンター編

目次

1 概要

このチュートリアルでは、XML の概要、XML を操作するための API、XML データの送受信について説明します。

XML の利用分野はデータ交換と文書処理の 2 つです。特に、データ交換分野での利用に対する関心が高まっています。データ交換分野での XML の役割は、任意のデータの表現方法を提供することにあります。ここでは、データ交換分野での利用を中心に説明していきます。

さらに XML 文書を操作するための API では、W3C（World Wide Web Consortium）で制定された DOM（Document Object Model)や、xml-dev メーリングリストで検討された SAX(Simple API for XML)を中心に、メリットとデメリットを説明します。このほか、XML DTD や XML スキーマとプログラミング言語との対応関係も含め推奨できる利用方法も述べます。HTTP などで XML データを送受信する際の推奨と将来動向にも言及します。

2　XML の基礎

ここでは、XML （Extensible Markup Language）の文法的な基礎と XML を制御するために用意されている仕組みを説明します。

2.1　XML とは

XML は正式名称を Extensible Markup Language と言います。Extensible には「タグセットの自由な設定が可能」、Markup には「定義されたタグでデータを記述」、Language には「タグ定義のメタ言語」というそれぞれの意味があります。拡張可能マークアップ言語と意訳できます。

XML のほかに、タグを定義する機能を持った汎用マークアップのメタ言語には、1986 年に ISO 国際規格として制定された SGML（Standard Generalized Markup Language）があります。XML はその SGML を Web 上で利用するために規格内容をスリムにしたものと言えます。

もともと、SGML はアプリケーションの独自形式によるデータ交換の弊害への反省をもとに誰にでも理解可能なテキストでの記述を目指して規格化されました。しかし、規格上の制約のためにマシンに対して大きな負荷が掛かったため HTML のように普及しませんでした。XML は SGML の規格をスリムにすることで Web 上での利用の制約がなくなりました。現在、SGML の目的であった文書の互換性を基盤に、文書処理に留まらずデータの表現形式、利用手段として注目されています。

XML のデータの保存が独自形式でないということは、結果的に資産性にも寄与します。技術進歩によってデータを処理するソフトは変わりますが、データは一時点での技術に依存せずに保全できるためです。XML では、データの互換性のほか、階層構造（木構造）のデータを表現できる特徴があります。この特徴とリンク機能の組み合わせによって、1 つのデータを様々な形で共有し再利用することが可能になります。

以上のような XML の機能を簡単にまとめますと、(1)用途に合わせたタグ定義が可能、(2)階層型の任意のデータ構造が表現可能、ということになります。

データ交換分野での XML の役割は、任意のデータの表現方法を提供することにあります。XML をデータ交換分野で利用するためには次の 3 つの段階の作業が必要です。

(1)　表現したいデータ構造について分析
(2)　データの設計書となる DTD(Document Type Definition) の開発
(3)　DTD に合わせたアプリケーションのインプリメンテーション、XML アプリケーションの開発

さらに、アプリケーションのインプリメンテーションの効率化のため、DTD の拡張仕様に相当する XML スキーマが提唱され現在、議論されています。

XML のデータ交換分野での活用イメージは、3-Tier モデルが典型的な例といえます。

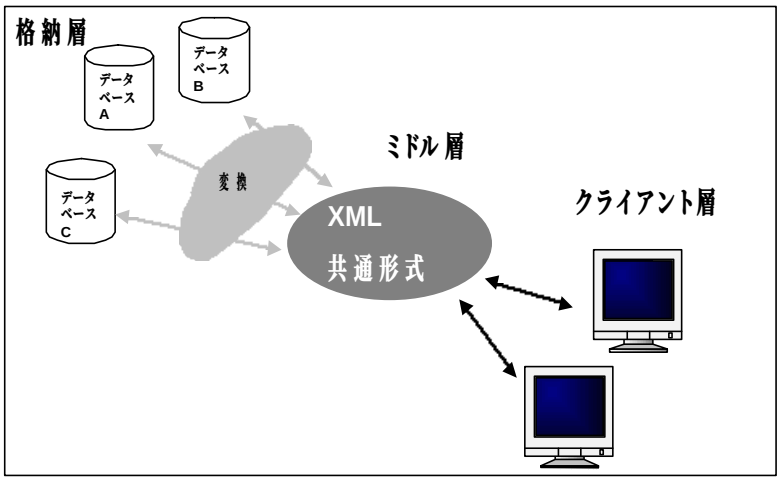


図 1：3-Tier モデル

現時点での XML の活用例は以下のようなものがあります。

E コマースでは、サプライチェーン取引の標準のための非営利組織である RosettaNet が、パートナー間のインターフェースプロセスを定義するために採用しています。また、マイクロソフトが提唱する電子商取引技術の枠組みである BizTalk でも XML を採用しています。さらに、請求書・領収書発行などのインターネット取引全体のプロセスをサポートする標準取引プロトコルである IOTP（Internet Open Trading Protocol）では、記述言語として採用しています。

また、マスコミでは、新聞を始めとするニュース発信業での XML ベースのデータ形式として NITF（News Industry Text Format）があります。Dow Jones 社が Wall Street Journal Interactive Edition などの Web サイトで活用しています。このほか、Oracle データベースに蓄積された XML 文書を HTML 文書に変換してデータ配信を行っている PC World Online の例や、XML を採用して Web 上の求人欄を運用しているワシントンポスト紙の例があります。ワシントンポスト紙の求人欄では雇用主 150 社求人職種 2 万 5000 職のデータが XML 化されています。さらに出版業では、Open eBook が XML を採用した例などがあります。

2.2 XML の文法的な基礎

XML を利用するためには、表現したいデータ構造の分析と、DTD あるいは XML スキーマの開発、XML に対応するアプリケーションの開発が必要になります。ここでは、その前提となる XML の文法的な基礎を説

明します。XML の基本は、データ型式として、開始タグと終了タグが必須です。どんな開始タグ、終了タグが使えるかを規定するには、DTD（Document Type Definition）を利用することになります。最近では DTD の拡張として XML スキーマも登場しました。

まず、タグ付けの基本から説明します。タグ付けの基本は、以下のように要素（Element）を開始タグと終了タグで囲むことにあります。

<要素名>.... 要素内容.... </要素名>

要素では上位要素、下位要素を定義でき、この上位、下位関係を利用することで要素の階層構造を表現できます。この階層構造では系統図などの木構造も表現できます。階層構造をタグを用いて記述するときは以下のようになります。

<上位要素>
<下位要素>.... 要素内容.... </下位要素>
<下位要素>.... 要素内容.... </下位要素>
</上位要素>

以下のような例は間違いです。ルート要素は 1 つということが決まっていますが、次の例だとルート要素が 1 つに定まらないからです。

<sec>
<title>SGML</title>
SGML は 1986 年....
</sec>
<sec>
<title>XML</title>
XML は 1998 年....
</sec>

このほか、空要素、属性も記述できます。空要素とは、その要素の下位に文字列や要素がないことを示す要素です。XML になって初めて採用されました。SGML には存在しない概念です。空要素は

<要素名/>

と記述します。

一方、属性は要素を補うために設定するものです。複数の指定ができます。主に、文書処理としての XML で利用します。タグ付き文書の本体部分である XML インスタンスの開始タグの中で、以下のように記述します。

<要素名　属性名="属性値">

ただし、データ交換で XML を使う場合、データ構造はすべて要素によって記述することを推奨します。

次に、文書やデータの内容を要素に分ける例を示します。

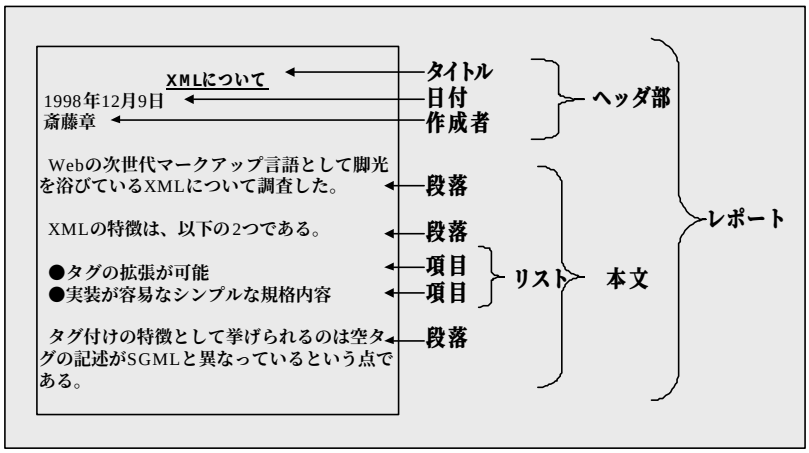


図 2：データ内容の要素分け

図 2 中では、データ内容を以下のとおりに要素を分けています。

「レポート」はルート要素
「ヘッダ部」、「本文」はルート要素の下位要素
「タイトル」、「日付」、「作成者」は「ヘッダ部」の下位要素
「段落」、「リスト」は「本文」の下位要素
「項目」は「リスト」の下位要素

その要素に従って以下のとおりのタグを定義します。

レポートを report
ヘッダ部を header
本文を body
タイトルを title
日付を date
作成者を author
段落を p
リストを list
項目を item

このタグ付けに従って記述した例は、以下のとおりです。

```
<report>
<header>
<title>XMLについて</title>
<date>1998年12月9日</date>
<author>斎藤章</author>
</header>
<body>
<p>Webの次世代マークアップ言語として脚光を浴びているXMLについて調査した。</p>
<p>XMLの特徴は、以下の2つである。</p>
<list>
<item>タグの拡張が可能</item>
<item>実装が容易なシンプルな規格内容</item>
</list>
<p>タグ付けの特徴として挙げられるのは空タグの記述がSGMLと異なっているという点である。</p>
</body>
</report>
```

図３：タグ付けの例

XML 文書には、検証済み XML 文書（Valid XML Document）とウェルフォームド XML 文書（Well-formed XML Document）の 2 つの形式があります。

(1)検証済み XML 文書

検証済み XML 文書は、XML 宣言と DTD(Document Type Definition)と XML インスタンスで構成されます。XML 宣言はそのファイルが XML 文書であることを示します。XML 宣言は

<?xml version="1.0" encoding="文字コード指定"?>

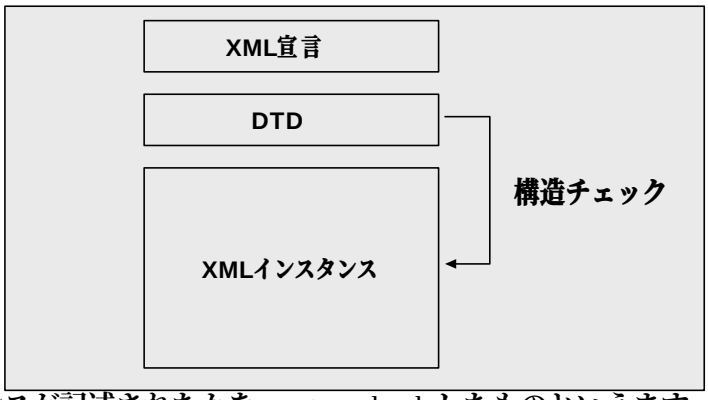
と記述します。

encoding 指定はオプションです。xml version1.0 規格では UTF-8 と UTF-16 は必須でサポートされています。Shift_JIS や EUC-JP、ISO-2022-JP は encoding 指定によって文字コードを指定しないと利用できません。

XML インスタンスはタグ付き文書の本体部分です。XML インスタンスは要素を 1 つ以上含むこと、階層構造が正しく記述されていること、ルート要素がただ 1 つであること、の条件を満たしている必要があります。

DTD はタグの要素とタグの要素同士の上位、下位の関係を定義し、XML インスタンスを syntax check します。syntax check とは、XML インスタンスのタグ要素とタグ要素同士の順番や位置関係、階層構造（木構造）が、DTD であらかじめ定義したと同一であるかの検証のことです。つ

まり、検証済み XML 文書は、DTD で定義された構造通りに XML イン



スタンスが記述されたかを syntax check したものといえます。

図 4：検証済み XML 文書

(2)ウェルフォームド XML 文書

これに対し、ウェルフォームド XML 文書は、XML 宣言と XML インスタンスで構成されます。形式が既に整っているという意味でこのような言い方をします。XML の基盤となった SGML が HTML のように Web での利用が広まらなかったのは、クライアントマシンにかかる負荷が大きかったことにありました。このため、XML では、負荷の原因であった DTD を省略可能にし、XML インスタンスを syntax check することによって生じる負荷の軽減を狙いました。

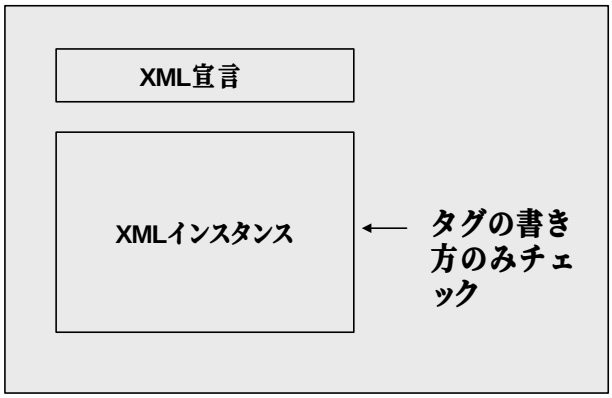


図 5：ウェルフォームド XML 文書

2.3 DTD の働き

DTD の働きについてもう少し詳しく説明します。

2.3.1 DTD の構成

DTD では、要素について、要素名として指定できる文字列、出現順序、出現回数を定義します。また属性、ENTITY、記法についても定義できます。ENTITY とは、ある文書の一部分を他のファイルで定義したり、置換文字列の扱いを可能にしたりする機能です。ENTITY は実体ともいいます。記法の定義とは、アプリケーションに記法を通知するための仕組みです。ENTITY と記法は主に XML を文書処理で利用する際に用います。データ処理で利用する際にはほとんど用いません。

DTD を記述するには、(1)DOCTYPE 宣言、(2)要素型宣言、(3)属性リスト宣言の宣言文を組み合わせます。

(1)DOCTYPE 宣言

DOCTYPE 宣言では DTD の指定を行います。DOCTYPE 宣言には DOCTYPE 宣言文内に直接記述する内部サブセットの方式と外部ファイルを参照する外部サブセットの方式があります。

内部サブセットの方式は、

```
<!DOCTYPE ルート要素名 [
    要素宣言
    属性宣言
    . .
 ]>
```

と記述します。

外部サブセットの方式は、

```
<!DOCTYPE ルート要素名 SYSTEM “ ファイルの URL">
```

と記述します。

両者を併用することもできます。

(2)要素型宣言

要素型宣言は、

```
<!ELEMENT 要素名 要素内容>
```

と記述します。

要素内容では下位となる要素の種類、出現順序、出現回数を記述します。出現順序は要素を並べた順番です。その際、並べた要素のうちどれか 1 つが出現することになります。出現回数は並べた要素の後に記号が何も付かない場合は、その要素は必ず 1 回出現するという意味です。「+」のときは、その要素は必ず 1 回以上出現します。「*」のときは、その要素は必ず 0 回以上出現します。「?」のときは、その要素は必ず 0 回あるいは 1 回出現します。DTD では上記以外のたとえば 3 回出現するという

指定はできません。そのように指定したい場合はアプリケーション側で要素が 3 回出現するなどの制限を加えなければなりません。出現順序や出現回数は丸括弧で括ってグループ化しそれらを組み合わせることで複雑な階層構造を記述できるのです。

(3)属性リスト宣言

属性リスト宣言は、

<!ATTLIST 要素名 属性名 属性値の候補 ”デフォルト値”>

と記述します。属性名は要素に付ける属性の名称です。

ただ DTD をデータ処理で利用する場合には、属性の使い方には（特に属性値の候補でデフォルト値を指定する際に）注意が必要です。属性値の候補でデフォルト値を指定した DTD を記述し、これを外部サブセットとして格納し URL で参照する場合、parser によっては外部サブセットを参照しないで処理するものがあるため、属性値の候補でデフォルト値が全く無視され、アプリケーションによって全く異なった処理結果となるためです。それを避けるために、DTD をデータ処理で利用する場合には、デフォルト値の指定を行わないことを推奨します。

2.3.2 DTD の記述例

次に DTD の記述例について説明します。以下に簡単なマニュアル用 DTD の記述例を示します。

```
<!ELEMENT doc          (chapter*)          >
<!ELEMENT chapter                (title,para*)        >
<!ELEMENT title                  (#PCDATA)      >
<!ELEMENT para                   (#PCDATA)      >
<!ATTLIST chapter id   ID   #REQUIRED         >
```

まず、1 行目です。<!ELEMENT の右隣に doc と記述して、doc がタグとして表現できることを宣言します。さらにその右隣に(chapter*)と記述し、doc の下位要素として chapter という要素を 0 回以上繰り返して使うと宣言します。0 回以上の繰り返しは chapter の右隣の「*」で指定しています。

2 行目の「<!ELEMENT chapter　(title,para*)>」は chapter という要素の構造を記述しています。chapter の下位要素では title という要素の次に para という要素が出現し、しかも、para については 0 回以上繰り返して使うというわけです。title と para を区切っているように見える「,」は左側の要素の次に右側の要素が出現するという要素の出現順番を決めるための記号です。

3 行目と 4 行目とでは、title と para のそれぞれの要素に右隣に(#PCDATA)と記述しています。#PCDATA がそれぞれの要素の下位にあるということを示しています。#PCDATA は文字データがあるという意味で、それがデータ型であることを示しています。PCDATA はパーストキャラクタデータの略です。3 行目は title という要素の下位に任意の

文字列があると宣言したことになります。同様に 4 行目も para という要素の下位に任意の文字列があると宣言したことになります。

ここまでの 1 行目～4 行目で doc の階層構造を記述しています。これを木構造として図式化すると図 4 のようになります。

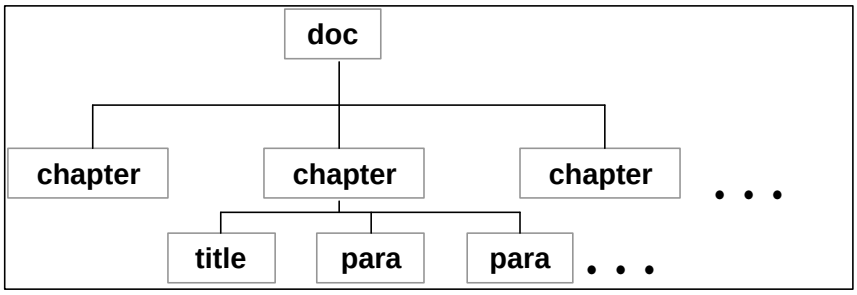


図 4：doc の木構造

5 行目の!ATTLIST は属性を定義するものです。2 行目で chapter を要素として使うことを宣言しましたが、その chapter に対し、id という名前の属性を定義したものです。id の右隣の ID はその属性がデータ型だと指定しています。ID は文書中に任意に決めるラベル名です。さらに、#REQUIRED によって、必須で省略できないものであることを指定しています。

DTD では以上のように要素、階層構造、属性を定義します。

2.4 XML プロセッサ

次に XML プロセッサについて説明します。

XML プロセッサは XML インスタンスのなかのタグの記述などをチェックします。前に、XML 文書には検証済み XML 文書（Valid XML Document）とウェルフォームド XML 文書（Well-formed XML Document）の 2 つの形式があると説明しました。その違いは DTD によって定義された要素の出現順番に従って階層構造が正しいかを、XML インスタンスを syntax check（検証）するかしないかにありました。

このため、XML プロセッサもそれらの 2 つの形式にそれぞれ対応する検証 XML プロセッサと非検証 XML プロセッサが用意されています。

検証 XML プロセッサは、XML インスタンスの中のタグの記述のほか、DTD によって定義された要素の出現順番に従って階層構造が正しいかをチェックします。このとき、DTD の外部サブセットを含め、すべての外部ファイルを参照します。

これに対し、非検証 XML プロセッサはタグの記述だけチェックします。DTD によって syntax check（検証）は行いませんが、属性値のデフォルト、置換文字列、外部ファイル指定などは DTD を参照します。ただし、DTD で指定された外部サブセットは参照しても参照しなくてもよ

いことになっています。

先に、DTD をデータ処理で利用する場合には、属性のデフォルト値の指定を行わないことを推奨しました。その際に、実装では外部サブセットを参照しないで処理する parser と外部サブセットを参照して処理する parser が存在しているためと説明しました。その理由は、非検証 XML プロセッサの仕様が外部サブセットは参照しても参照しなくてもよいことになっているため、異なる実装が登場してしまったことにあります。

3 XML の API

ここでは XML の API について説明します。XML の API には 2 つの有力なモデルがあります。1 つは DOM（Document Object Model）です。これは、W3C(World Wide Web Consortium)で設定され、既にレベル 1 が勧告されています。レベル 2 も近々勧告される予定です。もう 1 つは SAX（Simple API for XML）です。SAX は xml-dev というメーリングリストで有志が決めた API です。技術的な完成度が高いため、デファクトスタンダードになっています。

3.1 DOM

DOM（Document Object Model）は XML の要素間の関係を定義するインタフェースセットです。階層構造で記述されたデータに適用する関数をアプリケーションへ提供するものです。DOM によるデータへのアクセスは、parser によってメモリ上に作成された XML の階層構造に対し、firstChild や appendChild などの関数を適用する形で行われます。

DOM の利点は、階層構造のすべてがメモリ上に作成されるため、階層のどの部分に対しても処理できるほか、XML 文書の作成・変更もメモリ上でできるということです。XML 文書を最初から DOM の API で生成することも可能です。

これに対し欠点はメモリを消費することです。ただプロトコルの上でデータを交換するだけでの XML の利用なら、キロバイト単位のファイルサイズになるはずなのでこれが欠点になるとは言えません。

ただし、DOM の場合だと XML の要素は常に Element というクラスのインスタンスになります。このため、オブジェクト指向言語のプログラマが自分でデータフォーマットを決め、そのデータフォーマットを解釈するプログラムを作成し、メモリ上にデータを展開する際にはオブジェクト指向言語の恩恵を受けることができません。メモリ上に作られるオブジェクトに対し、たとえば Year というクラスのインスタンスを作ったりすることはできないからです。もともと、オブジェクト指向の場合はクラスごとにメソッドを定義できることに意味がありました。しかし、DOM の場合だと要素がすべて Element というクラスのインスタンスとなります。このためメソッドの追加が自分では一切できません。オブジェクト指向言語のプログラマにとっては、オブジェクトを使うメリットが薄れてしまうわけです。

また、DOM を利用する場合の注意点として要素内容における空白の問題もあります。たとえば、

```
<ol>
  <li>test</li>
</ol>
```

の場合です。DOM ではそれを「
改行コードスペースコード
test
改行コード
改行コード」という文字列と認識していますので、firstChild 関数を適用して先頭要素を求めても、期待される
test
という結果は得られず、改行コードとスペースコードを先頭要素として導き出してしまいます。

逆に、要素 ol に要素
test
を追加するため、appendChild 関数を適用しても、

```
<ol>
  <li>test</li>
</ol>
```

というファイル出力の結果は得られず、

```
<ol><li>test</li></ol>
```

となる場合があります。この場合だと、createTextNode 関数を使って空白テキストノードを生成したうえで appendChild 関数を適用しなければなりません。

DOM は以下のプログラムに実装されています。

まず、Java ベースだと、IBM XML Parser for Java、Sun Java Project X があります。これに対して C++ベースだと MS IE5.0、IBM XML4C です。IE5.0 付属の parser は信頼性の面で課題があります。これらのプログラムのほかに、perl ベースの XML::DOM があります。

3.2 SAX

次に SAX（Simple API for XML）について説明します。SAX は要素を読み込み、関数での処理を終えたら読んだ部分を消去していきます。DOM と異なりメモリ上に階層構造を作成しないわけです。イベントベースの XML パージングインタフェースといえます。

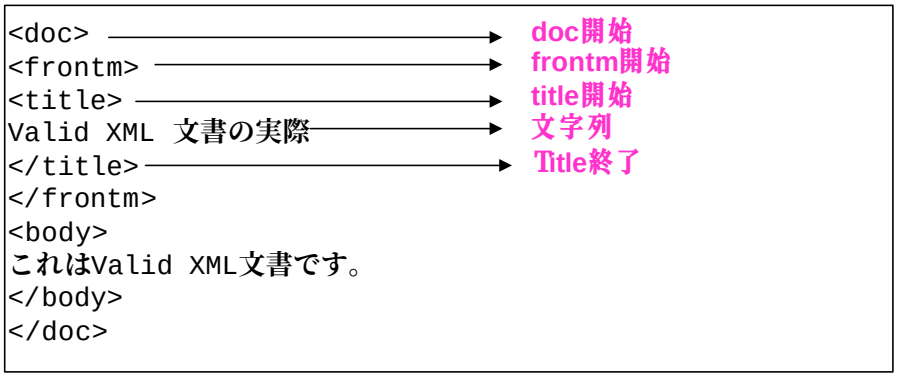


図 6：SAX によるイベント通知

図 6 の例だと、SAX が最初に<doc>を読むと、doc を開始したというイ

ベント通知があります。このときに関数が適用されます。関数処理が終わると<doc>の情報はメモリ上には残っていません。次に<frontm>という開始タグを読むと startElement 関数が呼ばれ、処理を行います。それが終わると<frontm>の情報はやはりメモリ上には残りません。SAX による関数の適用は、この行程を繰り返します。

このため、SAX の利点は、どんなに文書容量が大きくてもメモリ効率が良いということになります。

ただし、後の処理に影響を与えるような処理を実行する場合は、プログラマが自分で変数を確保して、それを記録に残しておくようなプログラムを作成しておく以外に方法がありません。

SAX では、プログラマがあらかじめ自分で定義した関数を parser が呼び込むという仕組みです。

イベントハンドラは以下のとおりです。

開始タグ（および空要素タグ）：

```
public abstract void startElement (String name,
AttributeList atts)
  throws SAXException;
```

終了タグ（および空要素タグ）：

```
public abstract void endElement (String name)
  throws SAXException;
```

文字データ：

```
public abstract void characters (char ch[], int start, int
length)
  throws SAXException;
```

無視できる文字データ：

```
public abstract void ignorableWhitespace (char ch[], int start,
int length)
  throws SAXException;
```

次に SAX を用いる際に注意すべき点は以下の 4 点です。

(1)文字内容
(2)要素内容における空白
(3)DTD が存在する状態で検証を行う parser を用いた場合
(4)DTD が存在しないかまたは検証を行わない parser を用いた場合

(1) 文字内容

文字内容では、たとえば、

```
<para>
開始タグは&lt;と&gt;で囲まれます。
</para>
```

という場合だと、関数 startElement が<para>に対して適用された後、関数 endElement が<li>に対して適用されるまで、関数 character は何回適用されるかが問題になります。これらは利用した parser によって異なります。まとめて処理する parser では 1 回、各行ごとに処理する parser では 3 回、「<」以前・「<」・「と」・「>」・「>」以降という処理を行う parser では、5 回ということになります。

(2)要素内容における空白

要素内容における空白では、たとえば、

```
<ol>
  <li>test</li>
</ol>
```

を、SAX も DOM と同様に「
改行コードスペースコード
test
改行コード
改行コード」という文字列と認識しています。このため、startElement 関数が<ol>に対して適用された後、直ちに startElement 関数が<li>に対して適用されるわけではありません。

(3)DTD が存在する状態で検証を行う parser を用いた場合

この場合では、

```
public abstract void ignorableWhitespace (char ch[], int start,
int length)
  throws SAXException;
```

が何回か呼び出されます。この場合、改行コードとスペースコードに対してまとめて 1 回の処理、または、改行コードで 1 回・スペースコードで 1 回の処理、のどちらかです。どちらになるかは、処理系プログラムに依存します。

(4)DTD が存在しないかまたは検証を行わない parser を用いた場合

この場合では、

```
public abstract void characters (char ch[], int start, int
length)
  throws SAXException;
```

が何回か呼び出されます。この場合も、改行コードとスペースコードに

対してまとめて 1 回の処理、または、改行コードで 1 回・スペースコードで 1 回の処理、のどちらかになります。この場合も処理系プログラムに依存するのでどちらだとも言えません。

SAX は以下のプログラムに実装されています。

まず、Java ベースだと、IBM XML Parser for Java、Sun Java Project X があります。これに対して C++ベースだと IBM XML4C、expat です。IE5.0 付属の parser は信頼性の面で課題があります。これらのプログラ Perl ベースでは XML::Paser があります。

4　ネットワークにおける XML データ通信

4.1　データの送受信での XML の利点

データ送受信での XML の利点は、きわめて簡単に扱えることです。データフォーマットを自分で設計して paser を自分で開発するには、言語の国際化の問題も考慮しなければならないだけに、大変な労力が必要です。これに対して、XML では、完成度の高い XML paser が普及しています。しかも、言語の国際化でも十分な対応がなされており、日本語環境での利用も可能です。

また、XML が提供する階層構造によって、アプリケーションが利用するほとんどのデータ構造を記述できます。さらに、XML はテキストとして扱えるため、不具合が生じたときに、テキストエディタを使って内容を確認できるというメリットもあります。

4.2　現時点で採用可能な方式

ここでは、プロトコル、XML の形式、符号化方式、送信側での XML データ作成、送信側でのプログラミング言語、受信側での XML データの解読、受信側でのプログラミング言語について説明します。送信側での XML データ作成と受信側での XML データの解読は、ともに DOM を用いることを推奨します。

(1)プロトコル

プロトコルは HTTP が最も容易です。HTTP で POST、GET を使います。受信側では CGI、Servlet も利用できます。

(2)XML の形式

次に、XML の形式は、ウェルフォームド XML 文書（Well-formed XML Document：DTD を持たない整形式 XML 文書）を採用すれば処理結果が同じであることが保証されます。

検証済み XML 文書（Valid XML Document：DTD を持つ妥当な XML 文書）は、DTD が外部サブセットとして格納され URL で参照するような場合に、アプリケーションによっては全く異なった処理結果を招く可能性があります。それは、parser によっては外部サブセットを参照しないで処理するものがあるためです。これを回避するには、DTD を記述する際に、デフォルト、実体（ENTITY）、記法を使用しないことです。

(3)符号化方式

文字コードでは、Unicode と JIS X 0208 との変換の揺れは必ず問題に

なります。
JIS X 0208 の

¯ ―＼～ ‖ －￥¢ £ ¬

という文字が Unicode でどのように表示されるかは処理系のアプリケーションに依存しています。マイクロソフト、IBM、サン・マイクロシステムズなど各社で採用している符号化方式変換テーブルは異なります。このため、比較演算を行った場合、演算結果は一致しません。推奨できるのは UTF-8 か UTF-16 の利用です。UTF-8 と UTF-16 は Windows95/98 のエディタでは扱えませんが、WindowsNT の場合だと標準装備のメモ帳でも扱えます。

(4)送信側での XML データ作成

送信側で XML データ作成する方法は 2 つあります。write 文を繰り返して XML 文書を作成する方法と DOM で XML 文書をメモリ上に生成してから書き出す方法です。

write 文を繰り返して XML 文書を作成方法は、覚えなければならないことが少ないだけに容易です。その反面、XML 文法エラーが発生したり、データ処理時に予想外の結果を招いたりする可能性も高いと言えます。

これに対し、DOM で XML 文書をメモリ上に生成してから書き出す方法は、文法エラーがほとんど発生しないというメリットがあります。その反面、DOM を覚える必要があります。また、ファイルに書き出すときの API には業界標準がないため、独自仕様に従わなければならないという欠点もあります。

(5)送信側でのプログラミング言語

送信側でのプログラミング言語は、write 文を使うのならどのようなプログラミング言語でも構いません。DOM を使う場合は、DOM が使える言語である必要があります。Java、C++、perl、 JavaScript、VBScript など多くの言語が DOM をサポートしています。

(6)受信側での XML データの解読

受信側での XML データの解読には、SAX か他のイベントベース API を用いる方法と DOM を用いる方法があります。送信側での XML データ作成と受信側での XML データの解読は、ともに DOM を用いることを推奨します。覚えなければならないことが 1 つで済むからです。

SAX や他のイベントベース API は文書容量が大きくてもメモリ効率が良い反面、階層構造が必要な場合は、プログラマが自分で変数を確保して、それを記録に残しておくプログラムを作成しなければなりません。

DOM は、階層構造がメモリ上に全部作成されるので階層のどの部分に対しても処理できる反面、メモリを消費します。ただ、処理対象はプロトコルの上でデータ交換のための XML データですから、キロバイト単位のファイルサイズになるはずで欠点にはなりません。

(7)受信側でのプログラミング言語

DOM を使う場合は、DOM が使える言語である必要があります。Java、C++、perl、 JavaScript、VBScript など多くの言語が DOM をサポートしています。このうち、Servlet を用いる際には、Java を推奨します。それ以外だと、perl も有力です。C++は推奨できません。

4.3 将来の方式

4.3.1 XML スキーマ

現状では、プログラミングで採用するインタフェースは SAX と DOM の 2 つの選択肢があります。どちらも実用に耐えるものです。しかし、SAX と DOM のそれぞれに欠点もあります。SAX はデータがメモリに作られないというものです。DOM は、メモリに作られるデータは、つねに Element クラスと Text クラスのインスタンスになってしまうため、XML を使うとオブジェクト指向のメリットが活かせないというものです。

このため、アプリケーション固有のクラス定義に従っているデータを、XML 文書からメモリ上に自動的に生成することを目標に掲げた取り組みが始まっています。XML スキーマからクラス定義に従っているデータをメモリ上に自動的に生成するという議論です。

XML スキーマは DTD の拡張仕様に相当します。

もともと DTD には、以下のようなデータ型しか用意されていません。

#PCDATA (要素)	CDATA (属性)
ID (属性)	IDREF (属性)
IDREFS (属性)	ENTITY (属性)
ENTITIES (属性)	NMTOKEN (属性)
NMTOKENS (属性)	NOTATION (属性)

これらのデータ型だけでは、データベース開発者が期待するようなデータ処理はできません。integer、string、boolean などが一切ありません。このため、DTD ではたとえば age に表記できる数字は 60 までなどという記述ができません。このため、そのためのプログラムが別途作成する必要が生じてきます。

これに対し、XML スキーマのデータ型は、以下のように豊富なデータ型が用意されています。

string	boolean
real	timeInstant
timeDuration	recurringInstant
binary	uri
language	NMTOKEN
NMTOKENS	Name
NCName	ID

IDREF	IDREFS
ENTITY	ENTITIES
NOTATION	decimal
integer	non-negative-integer
positive-integer	non-positive-integer
negative-integer	date
time	

XML スキーマからプログラミング言語のクラス定義を生成することを目指すマッピングでの取り組みでは、XML Data binding facility for Java(Sun)、XML/Value RFP(OMG)から提案が出されています。また、XML スキーマとプログラミング言語のクラス定義を関連付ける取り組みとして Coins の活動があります。

ただ、XML Data binding facility for Java でも XML スキーマからクラス定義に従っているデータをメモリ上に自動的に生成するという目標だけが決まっており、それ以外のことは何も決まっていません。

XML スキーマは現在、W3C で議論されています。マイクロソフトが XML-Data Reduced という提案をかなり前から出しています。このほかにも、SOX、DDML などの提案があります。

W3C の XML スキーマワーキンググループでの議論では、様々なデータモデルに対する要求に応えようとして、仕様書をどんどん肥大化させています。その一方で実装はまだありませんので、今後、どのようになるかは現時点で見通しが立ちません。

4.3.2 新たなプロトコル

XML のデータ送受信のプロトコルは現状では、HTTP が最も確実に利用できます。そのほかにマイクロソフトから HTTP ベースの SOAP という提案がなされています。SOAP は HTTP のヘッダ部分を M-POST などで拡張した仕様です。ヘッダだけを見ることで、たとえば XML を使ったリモートプロシジャーコールを行うとしていることなどが分かるという仕組みです。現在、Internet Draft (Informational)となっており、今後 IETF で審議される予定です。