分散システム特論レポート

eXtensible Markup Language(XML) 1.0

情報工学専攻 修士 1 年 岡村研究室 冷水 明

平成 14 年 2 月 6 日

1 概要

XML (eXtensible Markup Language) は、W3C により規定されたメタマークアップ言語であり、SGML (Standard Generalized Markup Language) の制限したサブセットとなっている。SGML は規格が大規模となり、インターネット上での情報交換には向かない。また、従来インターネットで使われている HTML は拡張性がないという問題がある。そのため、XML はインターネット向きかつ拡張性のあるメタ言語として開発された。XML は、記憶レイアウトと論理構造の制約条件記述機構を提供する。

2 XML 文書

W3C 勧告「XML1.0」によって定義されるデータオブジェクトを XML 文書と呼ぶ。XML 文書は実体 (entity) と呼ばれる記憶単位からなり、XML 文書は文書実体と呼ばれる実体から始まる。実体は構文解析されるデータとされないデータから構成される。そして、構文解析されるデータは文字の集合であり、その一部はマークアップを構成する。マークアップを構成しないデータは文字データである。マークアップとは、開始タグや終了タグのように文書やデータの構造を明示するために使用する特別な文字列記号である。

また、XML 文書を単に文書と呼ぶこともある。2 章では、XML の基本的な構成要素について説明する。

2.1 整形式 XML

整形式 (Well-formd) XML とは、その構文が W3C 勧告 XML1.0 の定義にしたがった XML 文書であり、以下で示す document 生成規則に適合する必要がある。

document 生成規則

- 一つ以上の要素を含む
- ルート要素が一つだけ存在する
- ルートは他の要素の内容に含まれない
- 要素は完全な入れ子構造をなす。

要素とは、XML 文書において基本となる情報単位であり、開始タグと終了タグで区切られる部分を指す。

全ての XML 文書は整形式となる。整形式でない XML 文書は、XML 文書としては誤りである。また、整形式の XML 文書では、その XML 文書から参照される全ての解析対象実体も整形式となる必要がある。

2.2 文字及び文字データとマークアップ

文字はテキストの最小単位であり、テキストは文字データとマークアップから構成される。そして、マークアップでない全てのテキストは文字データとなる。また、XML 文書で使用できる文字は Unicord の文字集合と規定されているが、文字の符号化機構は実体ごとに異なっても良い。

2.3 共通の構文構成子

XML1.0 では、XML 文書内で広く使用する幾つかの記号も定義している。以下でその定義について簡潔に説明する。

- 空白
  一つ以上のスペース文字、復帰、改行、タブ
- 名前トークン
  名前に使用される文字列。最初に数字や-,_などの幾つかの文字を使用することができない。XML や xml の名前文字列は予約済みである。
- リテラル
  引用符 (“) で囲まれた文字列

2.4 コメント

<!--で始まり、-->で終わる部分をコメントと呼ぶ。この 2 つの記号で囲まれた内部は、2 つの記号とともに

無視される。コメント文は、他のマークアップの外ならどこに現れても良い。

2.5 CDATA セクション

<[CDATA[で始まり、]]>で終わる部分を CDATA セクションと呼ぶ。CDATA セクションでは、囲まれた部分の文字をそのままの形で扱い、マークアップとしては解釈されない。

2.6 処理命令セクション

<?で始まり、?>で終わる部分を処理命令セクションと呼ぶ。処理命令セクションは
<?ターゲット名 データ?>
の形式をとり、ターゲット名で指定される特定のアプリケーションへの命令を文書に組み込むために使用される。

2.7 XML 宣言

XML 宣言は XML 文書の先頭に付き、その文書が XML 文書であることを明確に示すための文字列である。XML 文書は XML 宣言ではじめることが望ましい。XML 宣言では、使用する XML のバージョンや。その文書内で使用する文字コードの種類を宣言する。また、XML 宣言にはスタンドアロン文書宣言と呼ばれる宣言も含まれる。スタンドアロン文書宣言では、その文書実体の外部に参照すべきマークアップ言語があるかどうかを宣言する。スタンドアロン文書は外部の実体を参照しなくても全ての文書が得られる XML 文書となる。以下に XML 宣言の例を示す。

```
<?xml version="1.0" encording="UTF-8"?>
<?xml version="1.0" standalone='yes'?>
```

XML 宣言は<?xml と?>で囲まれる。version の部分で XML のバージョンを指定し、encording の部分で使用している符号化方式を指定する。そして、standalone の部分がスタンドアロン宣言であり、yes ならば、その文書はスタンドアロン文書となる。

2.8 文書型宣言

文書型宣言では、XML 文書が満たすべき制約条件を定義した文書型定義 (DTD, Data Type Definition) を宣言する。DTD にはマークアップ宣言と呼ばれる XML 文書が満たすべき制約を宣言するための記法が含まれる。文書型宣言では、宣言の中に定義を直接書いても良いが、一般的には別ファイルとして用意された DTD を参照する。文書型宣言は、<!DOCTYPE と>で囲まれ、最初の要素の前にでてこなければならない。以下に文書型宣言の例を示す

```
<!DOCTYPE sample SYSTEM "sample.dtd">
```

この文書型宣言では、DTD がかかれたファイルとしてとして sample.dtd を参照する。

2.9 妥当な XML

妥当な (Valid) XML とは、文書型宣言を持ち、DTD で宣言された制約条件を満たす文書のことである。これに対し、整形式 XML は構文だけが XML の仕様に適合したものであり、制約条件を満たす必要はない。

3 XML 論理構造

XML 論理構造とは、要素によって構成される XML 文書の構造のことである。XML では、種々の宣言により文書構造に制約を与えることができる。まず、最初に XML 文書が満たすべき制約として以下のものがある。

- XML 文書は一つ以上の要素を含む
- 要素の境界は開始タグと終了タグで区切られる
- 要素は幾つかの属性を持つことができる
- 要素は完全な入れ子構造を持つ

つまり、XML 文書は一つの要素から始まる木構造となる。3 章では、XML 文書の構造に制約を与えるための各宣言とその構成要素について説明する。

3.1 開始、終了タグと空要素タグ

要素の開始を示すマークアップを開始タグ、終了を示すマークアップを終了タグと呼ぶ。そして、開始タグと終了タグの間のテキストを要素の内容と呼ぶ。開始タグは、以下の形式となる。

```
<Name Attribute="Value">
```

Attribute は属性の指定となる。属性の指定はなくても良い。属性については後述する。
次に、終了タグは、以下のような形式となる。

```
</Name>
```

終了タグは対応する開始タグと対になって現れ、終了タグの Name は、対応する開始タグと同じとなる。

また、要素の内容が空であるものを空要素と呼ぶ。空要素を表すために空要素タグが用いられる。空要素タ

グは以下のような形式となる。

```
<Name Attribute="Value"/>
```

空要素は開始タグと終了タグを用いて

```
<Name></Name>
```

と表現してもよい。

3.2 要素型宣言

要素型宣言は、XML 文書の要素の構造に制約を与えるための宣言である。要素型宣言は<!ELEMENT と>で囲まれる。そして、要素型宣言は以下のような形式となる。

```
<!ELEMENT Name contentspec>
 contentspec ::=
    'EMPTY' | 'ANY' | Mixed | children
```

contentspec の部分の記述によって Name の要素に制約が与えられる。contentspec として取りうる記述には以下のものがある。

- EMPTY
  内容は空である
- ANY
  内容は何でも良い
- childeren(要素内容)
  子要素だけを含む (文字データを含まない)
- Mixed(混合内容)
  子要素に混在して文字データが含まれる

3.2.1 要素内容

ある型の要素が子要素だけを必ず含み、それらの間には空白だけしか現れないとき、その要素は要素内容型をもつという。要素内容モデルは、子要素の型及び子要素の出現順序を制御する簡単な文法となる。この文法は , 内容素子から構成され、内容素子は、名前、内容素子の選択リスト又は内容素子の列リストから構成される。以下に、要素内容モデルの例を示す。

```
<!ELEMENT book (head, (title | author)* )>
```

この例では、book 要素の内容として、head 要素が現れた後、title と author が何度か現れることを、定義している。

3.2.2 混合内容

ある要素型の要素内に、子要素に混在して文字データが含まれる可能性があるとき、その要素型は , 混合内容をもつという。 この場合、子要素の型についての制約が存在してもよいが、子要素の順序又は出現回数についての制約は存在しない。以下に、混合内容モデルの例を示す。

```
<!ELEMENT p (#PCDATA|a|b)*>
```

この例では、要素 p はその内容として、文字データ、要素 a、要素 b のどれかを何度かとる。

3.3 属性リスト宣言

属性は要素に対して負荷的な情報を付け加えるために使用され、名前と値の対で表現される。属性リスト宣言では、要素に関連づけられる属性を宣言する。属性の指定は開始タグ又は空要素タグ内だけに現れる。また、属性リスト宣言では、要素に適用する属性の集合を規定すると共に、属性の型制約やデフォルトの値を規定している。属性リスト宣言は以下のような形式となる。

```
<!ATTLIST Name AttDef* >
  AttDef ::= Name AttType DefaultDecl
```

AttType が属性値の型となる。属性リスト内で使用できる属性の型は以下のものがある。

- 文字列型
  任意のリテラル文字をとる
- トークン化型
  ID、IDREF、ENTITY、NMTOKEN など、ある制約をもった型
- 列挙型
  宣言した幾つかの値の内、1 つをとる

また、DefaultDecl の値において、属性の指定が必須かどうかについての情報も同時に与える。指定できる値には以下のものがある。

- #REQUIRED : 属性が必須
- #IMPLIED : デフォルト値なし
- AttValue : デフォルトの属性値

もし、同じ要素に対して複数の属性リストが存在する場合、その内容はマージされる。また、同じ属性に対し複数の定義がある場合は、最初の宣言が有効となる。以下に、属性リスト宣言の例を示す。

```
<!ATTLIST book title CDATA #REQUIRED>
```

上の例では、book は属性 title をとり、その属性値は任意の文字データで、要素が現れる際には必ず必要となる。

3.4 条件つきセクション

条件つきセクションでは、制御キーワードを指定することで、特定の部分を DTD の論理構造に含めるかどうかを制御できる。その構文は以下のようになる。
<![条件キーワード [DTD の宣言群]]>
条件キーワードとして IGNORE と INCLUDE のどちらかを指定できる。IGNORE だと DTD の宣言群は無視され、INCLUDE だと宣言群を処理する。

4 XML 物理構造

XML 文書は一つ以上の記憶単位 (実体) から構成される。物理構造とは、複数の実体と実体参照により作られる構造のことである。最初に満たすべき基本的な物理構造として、各 XML 文書は文書実体を一つ持つことがある。文書実体とは、実体の成す木構造のルートであり、処理開始の対象である。4 章では、XML の物理構造について説明する。

4.1 文字参照と実体参照

文字参照は、キーボードから直接入力できない文字など、特定の文字を参照するために使用される。文字参照の区切り子としては、&#と; が使われ、&#文字コード; の形式となる。

実体参照は、名前の付いた実体の内容を参照する。DTD で規定された定義済みの実体を XML 文中で参照する。その区切り子としては、&及び、; を用いる。また、実体参照の中でも、DTD 内部だけで有効な実体であるパラメタ実体を参照する場合をパラメタ実体参照と呼び、その区切り子として、%及び、; を用いる。実体参照では、実体宣言で定義された実体のみを参照できるが、例外として、マークアップに使用する記号に対応する実体は、定義しなくても参照することができる。この実体を定義済み実体呼ぶ。

4.2 実体宣言

実体宣言では、実体参照で参照される実体を定義する。定義される実体の種類として、内部実体と外部実体がある。内部実体は、実体の内容を宣言内で直接与える。それに対し、外部実体は、内部実体でない実体であって、外部にある実体を参照する。以下に実体宣言の例を示す。

```
<!ENTITY hello "Hello world.">
```

この例では、Hello world を&hello; という文字列で参照することができるようになる。

5 XML プロセッサと実装

XML 文書の基本処理を行なうものを XML プロセッサと呼ぶ。XML プロセッサは、XML 文書を読み込み、その内容へのアクセス機構を提供する。さらに、必要に応じて文書の正しさもチェックする。XML プロセッサは、他のモジュールのための動作を前提とし、アプリケーションに依存しない共通ＸＭＬ処理を行う。

5.1 XML の基本処理

XML 文書を処理する際の基本的な処理の流れについて説明する。まず、XML 文書を処理する場合、作成した XML 文書を XML プロセッサに渡す。次に、XML プロセッサが XML 文書を解析し、文書構造のモデルを作成する。そして、その文書構造をたどってアプリケーションが処理を行なう。XNL 文書の文書構造生成・操作方法としては、DOM (Document Object Model) と SAX (Simple API for XML) の 2 種類がある。DOM は XML 文書から DOM ツリーと呼ばれる木構造を生成し、アプリケーションはその木構造をたどって、処理を行なう。それに対し、SAX では、XML 文書を逐次解析しながら、結果をアプリケーションに渡す。

5.2 実装例

XML プロセッサの実装例として、Xerces とその実行例を紹介する。Xerces は、Apache XML Project の一部として開発されている XML パーサである。XML パーサとは、XML プロセッサの中でも、特に論理構造をチェックする部分をさすが、現在は XML プロセッサと同じ意味で使われることが多い。Xerces には、Java と C++のバージョンがあり、それぞれに DOM パーサと SAX パーサが存在する。今回は Java バージョンの DOM パーサを使用し、本のタイトルや著者、出版年を書いた XML 文書を解析して、タイトルと著者の一覧を表示するサンプルプログラムを作成した。サンプルの DTD 及び XML 文書、サンプルプログラムとその実行結果は、レポート後部に付録として添付する。

付録 A. サンプルDTD

本に関するデータを書くための DTD この DTD により本のタイトル、著者、発行年などの構文を規定している。

```
<!ELEMENT books (book)+>
<!ELEMENT book (title,year,author)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT author (name,others?)>
<!ATTLIST author country CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)><!ELEMENT others (#PCDATA)>
```

付録 B. サンプルXML

上記の DTD に基づいて書いた XML 文書。3 冊の本についてのデータが書かれている。

```
<?xml version="1.0"?>
<books>
  <book>
    <title>Mastering TCP/IP</title>
    <year>1999</year>
    <author country="USA">
      <name>Dave Kosiue</name>
    </author>
    <name>hiyami</name>
  </book>
  <book>
    <title>UNIX Network Programing</title>
    <year>1999</year>
    <author country="USA">
      <name>W.Richard Stevens</name>
    </author>
  </book>
  <book>
    <title>Harry Potter and the Sorcerer's Stone</title>
    <year>2001</year>
    <author country="England">
      <name>J.K. Rowling</name>
    </author>
  </book>
</books>
```

付録 C. サンプルプログラム

XML 文書をパースし、タイトルと著者の要素だけをとりだし、タイトルと著者の一覧を表示するプログラム。

```
//DOM パーサーのパッケージをインポート
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Attr;

class DOMSample{
  public static void main(String argv[]){
    try{
      DOMParser xml = new DOMParser();
      xml.parse(argv[0]); //XML 文書の解析
      Document xdoc = xml.getDocument();

      Element rootEl = xdoc.getDocumentElement();// ルート要素を取得
      System.out.println(rootEl.getNodeName());// ルートノードの名前を取得

      NodeList book = xdoc.getElementsByTagName("book");
      int num = book.getLength();
      System.out.println("全部で" + num + "冊あります。");
      System.out.println("タイトル一覧");

      NodeList titleList = xdoc.getElementsByTagName("title");
        //title という名前の要素のリストを取得
      NodeList nameList = xdoc.getElementsByTagName("name");
        //name という名前の要素のリストを取得
      int i=0;
      Node title; Node name;
      //タイトルと著者の一覧を表示
      for(i=0; i<num; i++){
        title = titleList.item(i);
        name = nameList.item(i);
        System.out.print(i+1);
        System.out.print(". " + title.getFirstChild().getNodeValue());
        System.out.println("\t" + name.getFirstChild().getNodeValue());
      }
    }
    catch(Exception e){
      System.out.println(e + ":例外発生");
    }
  }
}
```

付録 D. 実行結果

サンプルプログラムを実行した結果。

```
[akira@artemis sample]$ ./DOMSample sample.xml
books
全部で 3 冊あります。
タイトル一覧
1. Mastering TCP/IP Dave Kosiue
2. UNIX Network Programing  W.Richard Stevens
3. Harry Potter and the Sorcerer's Stone    J.K. Rowling
```

サンプルの XML と比較すると、title 要素と name 要素の内容を正しく取り出すことに成功していることが分かる。