情報学科CSコース
情報システム（3年後期）
第12回
（田島担当分第4回）

田島 敬史

2012 年 12 月 26 日

XML (eXtensible Markup Language)

- SGML (standard generalized markup language):
  汎用のマークアップ言語の規格
- HTML = SGML の一インスタンス
  - HTML の規格で定められたタグを使い，論理/表示構造を記述
  - WYSIWYG と論理記述の中間？
- XML = SGML − 余計な(複雑過ぎる)ところ
  - ユーザがタグを定義し，論理構造を記述
  - スタイルシートにより，表示を制御
    CSS(Cascading Style Sheets), XSL
  - 構造化「文書」だけでなく，データ交換にも有用.

XML (eXtensible Markup Language)

パラダイム・シフト

HTML → XML

- 人手による記述からプログラムによる生成へ
- 人間による閲覧からプログラムによる利用へ
  例：Google や Amazon の API

⇓

- 人間向きの「文書」から機械向きの「データ」へ

XML

XML データの例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE univ SYSTEM "univ.dtd">
<univ>
  <dept>
    <name>CIS</name>
    <staff staffId="253">
      <name>Smith</name>
      <email>smith@cis.univ.edu</email></staff>
    <student advisor="253"><name>Mich&eacute;l</name></stud
  </dept>
  <dept>
    <name>BIO</name>
          :
  </dept>
</univ>
```

XML

XML データの基本構成要素

- XML 宣言: `<?xml version=1.0 encoding="UTF-8" ...?>`
  - 必ずファイルの先頭
  - `encoding` — 文字コード. UTF-8, UTF-16, ISO2022-JP, EUC-JP, SHIFT_JIS など. XML の default は Unicode.
  - `standalone` — (略)
- コメント: `<!-- ...-->`
- 要素: <要素名> ...</要素名>
  - 必ず一つの document root を根とする木構造とする
  - 同じ名前の子要素が複数あってもよい
  - 空の要素は開始/終了タグをまとめて <要素名/> のように書く

XML

- 属性: <要素名 ... 属性名="属性値" ...>
  - 一要素に同名の属性は一つ
  - 順序には意味はない
  - 属性値は文書の「character data」を構成しない.
  - 属性と属性値 $\longleftrightarrow$ 子要素とそのテキスト値

```
<person name="John Doe" age="24" email="john@abc.com">
                              ↕
<person>
    <name>John Doe</name>
    <age>24</age>
    <email>john@abc.com</email>
</person>
```


XML

- ID 属性と IDREF, IDREFS 属性
  - DTD(後述) の中で,
    - person 要素の personID 属性を ID 型の属性
    - computer 要素の compID 属性を ID 型の属性
    - person 要素の pc 属性を IDREFS 型の属性
    - computer 要素の onwer 属性を IDREF 型の属性

    と定義

```
<person name="John Doe" personID="p295" pc="hal1 hal2 ">
        ⋮
<computer compID="hal1" owner="p295">
<computer compID="hal2" owner="p295">
```


XML

- PCDATA:
  - 文書の「character data」を構成. ($\leftrightarrow$属性値)
- CDATA: `<![CDATA[ ... ]]>`
  - TEX の `\verb`
- 実体参照: &実体名;
  - DTD 中で定義(後述) した定数を参照.
  - `&`, `<`, `>` を表す `&amp;` `&lt;` `&gt;` は常に定義されている

Namespace (名前空間)

同じ要素名が別の意味で使われていると問題が生じる

- 別の場所，組織で作成されたデータを合成して使いたい場合
  例：XHTMLの中に，数式記述用の規格MathMLの記述を入れたい．
- 任意の XML データに対し，タグを付加するような応用
  例：XMLデータに自分用の注釈を書き込めるシステム

Namespace (名前空間)

- namespace: URI を使って人が使わない「namespace 名」を決め，要素名，属性名の prefix とする
  `<univ xmlns:abc="http://xyz.net/">` ← prefix abcを定義
  `<abc:annotation>`あいう`</abc:annotation>`
- 「デフォルトの namespace 名」も定義可能
  `<univ xmlns="http://aiueo.net/">` ← デフォルトを定義
  `<annotation>`あいう`</annotation>` ← デフォルトが使われる
  (ただし属性名にはデフォルトは適用されない)

参考：XMLの「規格」について

- W3C (www.w3.org) が規格化の作業を行なっている．
  - Note — メモ
  - Working Draft — 標準化案の草案
  - Proposed Recommendation — 標準化案
  - Recommendation — 標準化された規格
- その他の W3C の規格:
  - XPath — XML データ中の部分データの指定方式．
    XSL, XQuery などの中で使用される．
  - XML Schema — DTD に変わるスキーマ記述方式の規格
  - XSL, XSLT — スタイルシートとそのための変換記述言語
  - XQuery — 検索言語

XMLデータの処理の枠組

基本的には「大規模木構造データ」処理一般に通じる話

プログラム API = 既存言語のためのインターフェイス

⇕

検索・操作言語 (XPath, XQuery, ...) = 専用言語

XMLのためのプログラム API

DOM

- XML データをパーズしてメモリ中に木構造を生成
- `Document`, `Element`, `Attribute`, `Text` などが木のノードとなる
- `getParentNode()`, `getNextSibling()`, `appendChild()` などのインターフェイスで木のノード間をナビゲートしながら操作
- 複雑な処理を行なうには容易
- 大きな XML をパーズすると，大量にメモリを食う

XMLのためのプログラム API

DOM: プログラム例

```
import javax.xml.parsers.*;
import org.w3c.dom.*;

DocumentBuilderFactory factory =
                    DocumentBuilderFactory.newInstance();
DocumentBuilder parser = factory.newDocumentBuilder();

Document d = parser.parse(uri);
Node root = d.getDocumentElement();
Node n = root.getFirstChild();
NodeList anchors = d.getElementsByTagName("a");
                    ⋮
```


XMLのためのプログラム API

SAX

- イベント駆動型(event driven)
- パーザが XML を先頭からパーズしていき，`startDocument`, `startElement`, `endElement` などのイベントを発生するので，そのイベントに応じて処理を行なうハンドラをあらかじめ作成してパーザに登録する.
- メモリの使用量が少ない
- ストリーム処理が可能
- 複雑な処理のプログラムは困難

XMLのためのプログラム API

SAX: プログラム例

```
import javax.xml.parsers.*;

SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();

MyHandler h = new MyHandler();

parser.parse(uri, h);
```

XMLのためのプログラム API

さらに，ハンドラを定義する．

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;

class MyHandler extends DefaultHandler {
  private int counter;

  public void startDocument() { counter = 0; }

  public void startElement(String ns, String local, ...){
    counter++;
  }
```

⋮

XMLのためのプログラム API

ちょっと変わったところで...

Relaxer

- RELAX 文法を読み込んで XML データをマップする Java クラス定義を自動生成
  - 要素 $\longrightarrow$ オブジェクト
  - 文字列型の子要素 $\longrightarrow$ 文字列型のインスタンス変数
  - さらに子要素を持つような子要素 $\longrightarrow$ オブジェクトを参照するインスタンス変数
  - * とともに定義されている子要素 $\longrightarrow$ List を参照するインスタンス変数

⋮

- XML データから 自動生成したクラスのオブジェクトを自動生成

XML のための検索・操作言語

概観

- 狭義の検索=データから必要な部分のみを抜き出す
- 広義の検索=必要な部分を必要な構造に再構成して呈示≒操作
- スタイルシート中で行なわれる「変換」も，一種の「検索」
- 狭義の検索言語: XPath (XQuery や XSLT の中で，XML文書中の一部分を指定するためにも使用される)
- 広義の検索(操作，変換)言語: XQuery, XSLT, ...

XPath

XPath 1.0 の data model

- XML =「syntax」 or 「format」 $\longleftrightarrow$ semantics
- 以下の7つをノードとする一つの木構造とみなす
  - root (document root を子に持つ)
  - element (element name をラベルとして持つ)
  - attribute (attribute name に@ を付けたものがラベル)
  - PCDATA (必ず葉)
  - namespace
  - comment
  - processing instruction

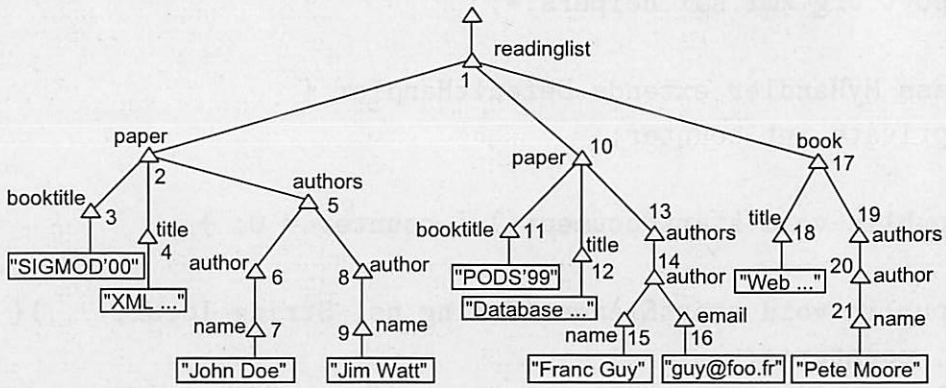
XPath

例:
```
<?xml version="1.0" encoding="UTF-8"?>
<readinglist>
  <paper>
    <booktitle>SIGMOD'00</boottitle>
    <title>XML...</title>
    <authors>
      <author><name>John Doe</name></author>
      <author><name>Jim Watt</name></author>
    </authors>
  </paper>
    :
</readinglist>
```


XPath

XPath 1.0 の data model

例:



XPath

基本構成

- URI や XML 属性中に埋め込める記法を採用
- 基本的な記法以外に多くの略記法が定められている
- 基本となるデータ構造 = ノードの集合(or シーケンス)
- 基本操作 =
  - ノードからノード集合への関数
  - ノード集合へのフィルタ操作
- 基本制御構造 = 集合上での繰り返しと結果の集合和

$f(\{e_1, e_2, \ldots, e_n\}) = g(e_i) \cup g(e_i) \cup \ldots \cup g(e_n)$

$f : \{node\} \rightarrow \{node\}$

$g : \ node \ \rightarrow \{node\}$

XPath

XPath の評価の基本的な流れ

例: (略記法での記述例)

```
/readlinglist/paper[authors//email]
```

$/ \rightarrow \{0\} \text{—readinglist}\rightarrow \{1\} \text{—paper}\rightarrow \{2, 10\} \text{---}$

$\text{---} \begin{cases} \{②, 10\} \text{—authors}\rightarrow \{5\} \text{—//email}\rightarrow \{\} = \text{false} \\ \{2, ⑩\} \text{—authors}\rightarrow \{13\} \text{—//email}\rightarrow \{16\} = \text{true} \end{cases} \rightarrow \{10\}$

XPath

様々な XPath 式の例(略記法による)

- /*/title　　「*」は「任意の名前」
- /book//title/text()　　「text()」は「任意のテキストノード」
- /book//author/text()
- /book//author//text()
- //title
- /book/chapter[title/text()='Introduction']
- /book[.//title='XML']　　「.」は「現在の要素」
- /book/chapter[2]/section[3]
  [ ]中の値が数字の場合は集合中でのその要素の順番と比較
- /book/chapter[figure[10]][3]　　[ ]中の値がノード集合の場合は$\neq \emptyset$

XQuery

概観

- データモデルは XPath と同様
- FLWOR 構文と呼ばれる SQL(関係DB用言語)のような構文
  - For, Let, Where, Orderby, Return
- FLWOR 構文は入れ子可
  → 出力したい階層構造に応じた入れ子で使用
- 基本となるデータ型: ノード(or 原子型)のタプルのシーケンス
- 基本となる制御構造: FOR 文によるシーケンスの各タプルに対する繰り返しと結果の連結

XQuery

FLWOR 構文

```
for $p in document("bib.xml")//paper
let $a := $p//author[1]/name/text()
where $a = "John Doe" or $a = "Franc Guy"
order by $p/year ascending, $p/title ascending
return ⟨copyright⟩
         $p/title
         ⟨owner⟩$p//author⟨/owner⟩
       ⟨/copyright⟩
```

Doe または Guy が筆頭著者である論文の著作権を持つ人物について，論文タイトルと著者名リストの組を year の昇順(辞書順)，同じ year のものに関しては，title の昇順に出力

XQuery

XQuery の評価の基本的な流れ

- for は変数にシーケンスの各要素を束縛し，束縛のシーケンスを生成
- let は変数にシーケンス全体を束縛
- 複数の for, let 全体で，束縛のタプル(のシーケンス)を生成
- その際，for が複数ある場合は直積を取る
- 各タプルに対して where を評価し，true となるもののみを選択
- order by によって，タプルのシーケンスをソート
- シーケンス中の各タプルに対して return を評価
- 各 return の結果(シーケンス)を連結

XQuery

- for, let によって生成される束縛の組のシーケンス

$$\langle \$p{:=}\ 2,\ \$t{:=}7\rangle,\ \langle \$p{:=}\ 10,\ \$t{:=}15\rangle$$

- where の評価結果が true となるもの

$$\langle \$p{:=}\ 2,\ \$t{:=}7\rangle,\ \langle \$p{:=}\ 10,\ \$t{:=}15\rangle$$

- それらに対する return 節の評価結果

```
⟨copyright⟩
  ⟨title⟩XML...⟨/title⟩
  ⟨owner⟩
    ⟨author⟩John Doe⟨/author⟩
    ⟨author⟩Jim Watt⟨/author⟩
  ⟨/owner⟩
⟨/copyright⟩
```

```
⟨copyright⟩
  ⟨title⟩Database...⟨/title⟩
  ⟨owner⟩
    ⟨author⟩Franc Guy⟨/author⟩
  ⟨/owner⟩
⟨/copyright⟩
```


XQuery

FOR と LET の違い

- for $p in document("bib.xml")//paper
  return ⟨result⟩$p⟨/result⟩

  $\Longrightarrow$ ⟨result⟩⟨paper⟩...⟨/paper⟩⟨/result⟩
  ⟨result⟩⟨paper⟩...⟨/paper⟩⟨/result⟩
  ⋮

- let $p := document("bib.xml")//paper
  return ⟨result⟩$p⟨/result⟩

  $\Longrightarrow$ ⟨result⟩⟨paper⟩...⟨/paper⟩
  ⟨paper⟩...⟨/paper⟩
  ⋮
  ⟨/result⟩

XQuery

FLWOR 構文の入れ子

```
for $a in distinct-values(document("bib.xml")//author/name)
return ⟨author⟩
         $a
         {for $p in
            document("bib.xml")//paper[.//author/name=$a]
          return $p/title}
         ⟨/author⟩
```

論文毎に分類されていたデータを著者毎に分類する階層に再構成

出力の階層構造に対応した FLWOR の入れ子構造を用いている

XQuery

複数の for による直積

```
for $p in document("bib.xml")//paper
for $aa in $p//author/name
let $a := $p//author[1]/name
where $a = "John Doe" or $a = "Franc Guy"
return ⟨copyright⟩
         $p/title
         ⟨owner⟩$aa⟨/owner⟩
       ⟨/copyright⟩
```

Doe または Guy が筆頭著者である論文の著作権を持つ人物について，そのタイトルと著者名一人一人の組を出力

XSLT

概要

- 「スタイルシート」(特にXSL)のための構造変換を記述する言語
- 構造再帰に近い
- 構造再帰
  - 再帰的なデータ構造に対して，その構造に沿って再帰的に処理を行う手法
  - データ全体を走査しながら，各部の構造に応じた処理を行う形になる

⇕

XQuery: パターンにマッチする部分を抜き出し，その部分に対して操作

XSLT

構造再帰

再帰的に定義される型とその操作の定義

構造の各パターンに対する処理規則の集合によって定義される

- 型 t のリスト: $L(t) = null \mid t.L(t)$

$$h(f,g,u)(l) = \begin{cases} u & \text{if } l = null \\ f(g(e), h(f,g,u)(l')) & \text{if } l = e.l' \end{cases}$$

- 型 t の集合: $S(t) = \emptyset \mid \{t\} \mid S(t) \cup S(t)$

$$h(f,g,u)(s) = \begin{cases} u & \text{if } s = \emptyset \\ g(e) & \text{if } s = \{e\} \\ f(h(f,g,u)(s_1), h(f,g,u)(s_2)) & \text{if } s_1 \cup s_2 \end{cases}$$

(ただし f は結合則を満たす関数)

XSLT

基本構造

スタイルシート = テンプレート規則の集合

```
⟨xsl:template match="パターン"⟩     ← このパターンにマッチする部分を
    テンプレート                    ← このテンプレートで変換せよ
⟨/xsl:template⟩
```

あるいは，

```
⟨xsl:template match="パターン"⟩
    テンプレート
    ⟨xsl:apply-templates/⟩          ← 部分構造への再帰的な適用
    テンプレート
⟨/xsl:template⟩
```


XSLT

例:

```
⟨xsl:template match="/"⟩
  ⟨html⟩⟨xsl:apply-templates/⟩⟨/html⟩
⟨/xsl:template⟩

⟨xsl:template match="paper"⟩
  ⟨h1⟩⟨xsl:value-of select="title"/⟩⟨/h1⟩
  ⟨xsl:apply-templates/⟩
⟨/xsl:template⟩

⟨xsl:template match="author"⟩
  ⟨b⟩⟨xsl:value-of select="name"/⟩⟨/b⟩
⟨/xsl:template⟩
```