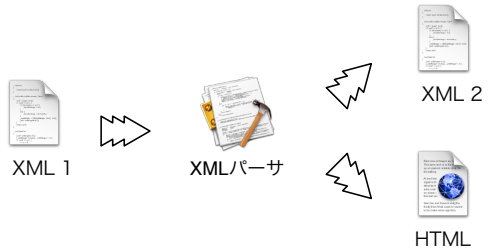
知識プログラミング方法論
第１０回 JavaによるXML文書の操作（DOM）

ライフスタイルデザイン研究センター 金井秀明

XMLパーサ

- XMLパーサ：XML文書を処理するためのモジュール
  - XML文書の検証
  - ソフトウェアがXML文書にアクセスする手段の提供.



XML文書を処理手順

Step 1: XML文書を読み込む or 新規作成する

Step 2: XML文書のデータを利用する

DOMやSAXによる操作

Step 3: XML文書を書き出す

代表的なXMLパーサ

- 木構造に基づくAPI: DOM(Document Object Model)
  - XML文書をメモリー上にツリー構造で管理し，プログラムからアクセスできるようにしたAPI
- イベント駆動に基づくAPI: SAX (Simple API for XML)
  - XML文書を要素ごとに順々に読み込み処理をする.

実装されたパーサ

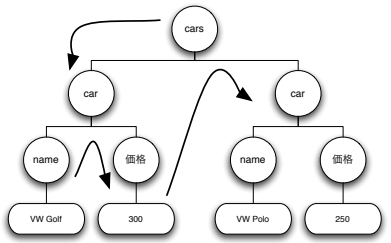
- JAXP
  - 標準XML API
  - Java API for XML Processing.　javax.xmlをimportする.
- Apache Xerces
  - 様々な言語用に実装されている.
  - 独自のXerces Native API(XNI)も提供している.
  - Java用はXerces-J(Apache Java XML Parser)

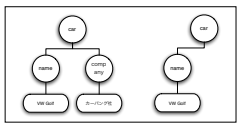
DOMの仕組み

XML文書

```
<cars>
  <car>
    <name>
      乗用車
    </name>
  </car>
  ...
</cars>
```

メモリ上に展開





ノードへの操作

SAXの仕組み

parsing

XML文書

```
<cars>
  <car>
    <name>
      乗用車
    </name>
  </car>
  ...
</cars>
```

event-driven

Handler

```
startDocument()
startElement()
startElement()
startElement()
characters()
endElement()
endElement()
...
endDocument()
```


JavaによるDOM操作

- JAXP: Java API for XML Processing
  - J2SE v1.4以降には標準で含まれている.
- 基本インタフェース
  - Document, Node, NodeList, Element, Text, NamedNodeMap, Attr

DOMに読み込む#1

- import宣言をする.

```
import java.io.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.transform.dom.*;
import org.w3c.dom.*;
```


DOMに読み込む#2

- DOMの準備
  - DocoumentBuilder(XML文書をDOMで扱うためのobject)をつくるためのファクトリと呼ばれるオブジェクトの生成する.
  - DocumentBuilderをファクトリに作る.

```
DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
```


DOMに読み込む#3

- XML文書を読み込む
  - 既存のファイルを読み込む

```
Document doc
      = db.parse(new FileInputStream("ファイル名"));
```


XML文書を新規作成する場合

- 文書を新規作成する

```
Document doc = db.newDocument();
```

- 作成した文書にルート要素を追加する

```
Element root = db.createElement("要素名");
doc.appendChild(root);
```

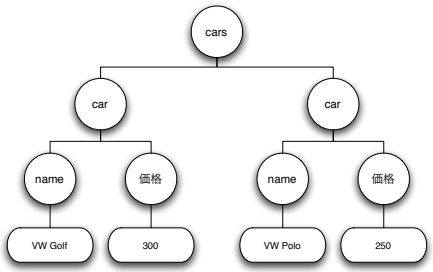
文書を書き出す

```
TransformerFactory tff
    = TransformerFactory.newInstance();
Transformer tf
    = tff.newTransformer();
tf.setOutputProperty(OutputKeys.ENCODING, "Shift_JIS");
tf.transform(new DOMSource(doc),
                new StreamResult("出力ファイル名"));
```


XML文書への操作

- 木構造をたどる操作
- データ（要素, テキスト, 属性）の追加，削除，その取り出し操作

```
<cars>
<car>
  <name>VW Golf</name>
  <価格>300</価格>
</car>
<car>
  <name>VW Polo</name>
  <価格>250</価格>
</car>
</cars>
```



XML文書の操作#1

インタフェース名	操作内容
Document	ノードを作る
Node or NodeList	ノードを操作する
Element or Text	要素や要素の内容を操作する
Attr	属性を操作する

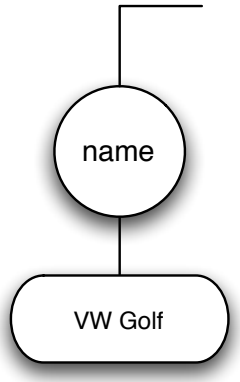
XML文書の操作#2

操作内容	メソッド
ノードを作る	createElement
ノードを操作する	getFirstChild（最初の子ノードへ移動）， getNextSibling（次の子ノードへ移動）， appendChild（子ノードの追加）， removeChild（子ノードの削除）
要素や要素の内容を操作する	getElementsByTagName（要素名を指定してノードリストを取り出す）など
属性を操作する	getName（属性名の取り出し）， getValue（属性値の取り出し）など

ノードの表現(DOM)

- ノードを以下の３つの情報で表現する．
  - ノードの種類，ノード名，ノード値
- それらの情報の取得のメソッドは，
  - getNodeType(),
  - getNodeName()
  - getNodeValue()

ノードの情報(DOM)



ノード種類: Node.ELEMENT_NODE
ノード名: name
ノード値: null

ノード種類: Node.TEXT_NODE
ノード名: #text
ノード値: VW Golf

XML文書をたどる(DOM)

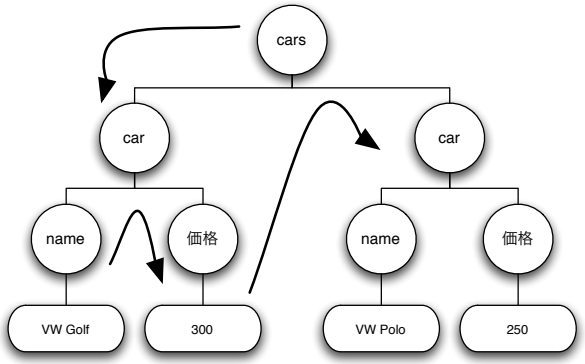
メソッド	機能
Element getDocumentElement()	文書のルート要素を得る
Node getFirstChild()	ノードの「最初の子」を得る
Node getNextSibling()	ノードの「次の兄弟」を得る
String getNodeType()	ノードの「ノード種類」を得る
short getNodeName()	ノードの「ノード名」を得る
String getNodeValue()	ノードの「ノード値」を得る

XML文書をたどる#1

- 出発点を決める．（通常，ルート要素にする）

```
Element root =
doc.getDocumentElement()
```

- たどる



XML文書をたどる#2

- ノードの子ノードをたどる（１階層分）
  - 最初の子ノードから始めて，兄弟のノードがある間

```
for (Node ch=root.getFirstChild();
      ch !=null;
      ch=ch.getNextSibling()) {
    処理内容1
  …
}
```

- 多層をたどるには，１階層分の処理を，再帰的に呼ぶことで実装できる．

例：一階層分#1

- 教科書３２５ページ Sample3.java

```
Element  root = doc.getDocumentElement();

walk(root);

public static void walk(Node n) {
  for (Node ch=root.getFirstChild();
       ch !=null;
       ch=ch.getNextSibling()) {
      System.out.println(ch.getNodeName());
      }
}
```


例：空白処理

- 教科書３２８ページ Sample4.java

```
public static void walk(Node n) {
  for (Node ch=root.getFirstChild();
       ch !=null;
       ch=ch.getNextSibling()) {
      if (ch.getNodeType() == Node.ELEMENT_NODE) {
        System.out.println(ch.getNodeName());
      }
      }
}
```


例：再帰処理

- 教科書３３０ページ Sample5.java

```
public static void walk(Node n) {
  for (Node ch=root.getFirstChild();
       ch !=null;
       ch=ch.getNextSibling()) {
      if (ch.getNodeType() == Node.ELEMENT_NODE) {
        System.out.println(ch.getNodeName());
        walk(ch);
      }
      }
}
```

例：テキスト処理

- 教科書３３３ページ Sample6.java

```
public static void walk(Node n) {
  for (Node ch=root.getFirstChild(); ch !=null;
       ch=ch.getNextSibling()) {
     if (ch.getNodeType() == Node.ELEMENT_NODE) {
        System.out.println(ch.getNodeName());
       walk(ch);
     }
     else if  (ch.getNodeType() == Node.TEXT_NODE
          && ch.getNodeValue().trim().length() !=  0) {
        System.out.println(ch.getNodeValue());
     }
     …
```


練習：教科書３３９ページ

- Sample.xmlのすべてのノードをたどり，要素とテキストの「ノード名」，「ノードの種類」，「ノード値」を表示する.

```
<?xml version="1.0" encoding="Shift_JIS" ?>
<cars>
 <car>
  <name>乗用車</name>
  <price>150</price>
 </car>
 <car>
  <name>トラック</name>
  <price>500</price>
 </car>
 <car>
  <name>オープンカー</name>
  <price>200</price>
 </car>
</cars>
```


26

練習：教科書３３９ページ

- Sample.xmlから次のようにテキストデータだけを取り出して画面に表示するコードを作成せよ

```
乗用車
150
トラック
500
オープンカー
200
```

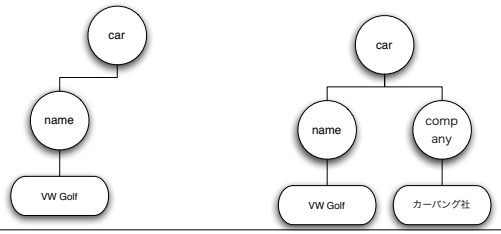

27

XML文書の操作#1

- データ（要素•テキスト•属性）の
  - 追加
  - 削除
  - 属性の取り出し
- どこに，何を，どうする.

XML文書の操作#2

- 要素の追加（教科書３４８１ページ，Sample1.java）
  - 対象の文書　　getOwnerDocument()
  - 追加要素　　createElement("追加要素名")
  - 追加する．　　追加先要素.appendChild(追加要素)



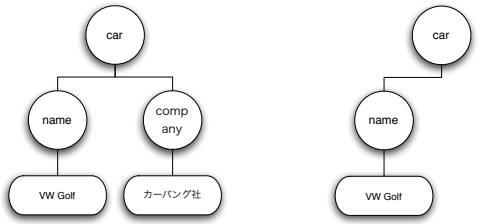
XML文書の操作#3

- テキストの追加（教科書３５２ページ，Sample2.java）
  - 対象の文書　　getOwnerDocument()
  - 追加テキスト　　createTextNode("テキストの内容")
  - 追加する．　　追加先要素.appendChild(追加テキスト)
- 属性の追加（教科書３５５ページ，Sample3.java）
  - 属性，属性値を追加する．

    追加先要素.setAttribute(属性名,属性値)

XML文書の操作#4

- 要素の削除（教科書３５８ページ，Sample4.java）
  - 削除要素の親要素を決め．削除要素.getParentNode()
  - 削除する．　　親要素.removeChild(削除要素)



XML文書の操作#5

- 属性の削除（教科書３６０ページ，Sample5.java）
  - 属性リストを得る．そのリストから属性を削除する．

    NamedNodeMap 属性リスト = 対象要素.getAttributes()
    属性リスト.removeNamedItem(属性名)

  - 対象要素から属性を削除する．

    ((Element)対象要素).removeAttribute(属性名)

XML文書の操作#6

- 要素の取り出し
  - 要素名を指定して，要素を取り出す.

```
Docoument doc;
NodeList lst = doc.getElementsByTagName(要素名)
```

  - 例：教科書３６４ページ，Sample6.java
    - 取り出した要素nameだけのXML文書作成

XML文書の操作#7

- 属性の取り出し（教科書３６９ページ，Sample8.java）
  - 属性リストを得る．そのリストから指定した属性名の属性を得る.

```
NamedNodeMap 属性リスト = 対象要素.getAttributes()
Node 属性 = 属性リスト.getNamedItem(属性名)
```

(country Lang)

DOMに読み込む#2

- DOMをファイルへの書き出し
  - Transfomerメソッドをつかって，XMLのソースツリーを結果ツリーに変換する.

```
TransformerFactory tff = TransformerFactory.newInstance();
Transformer tf = tff.newTransformer();
tf.setOutputProperty(OutputKeys.ENCODING,"Shift_JIS");
tf.transform(new DOMSource(doc), new
StreamResult("result.xml"));
```


XML文書の新規作成

XML文書の新規作成

```
Document doc = db.newDocument();
```

ルート要素の追加

```
Element root = doc.createElement("cars");
doc.appendChild(root);
```

練習：教科書３８８ページ

- 価格データを削除する
- 「やさしい」という文字列が入っている書名リストを作る.

```
<?Xml version="1.0" encoding="Shift_JIS" ?>
<books>
 <book>
  <title>入門SQL</title>
  <price>2400</price>
 </book>
 <book>
  <title>やさしいC++</title>
  <price>2500</price>
 </book>
 <book>
  <title>やさしいJava</title>
  <price>2600</price>
 </book>
</books>
```


37

まとめ

- DOMについて
- JavaによるDOMの利用
- ．．．