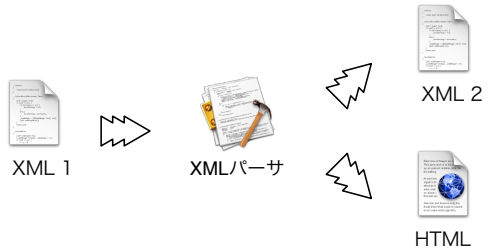
知識プログラミング方法論
第１１回 JavaによるXML文書の操作（SAX）

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.



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)

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

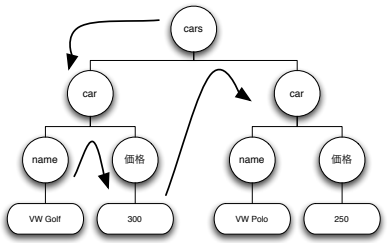
Step 3: XML文書を書き出す

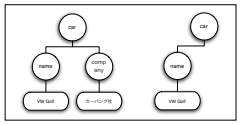
DOMの仕組み

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()
```


準備：SAXの利用

- import宣言
- SAXパーサを作るためのオブジェクトの生成
- SAXパーサ（XML文書をSAXで扱うためのオブジェクト）の生成

SAXの準備#1

- import宣言
- SAXパーサの生成
- ハンドラの作成
- XML文書の読み込み

SAXの準備#2

- import宣言

```
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.sax.helpers.*;
```


SAXの準備#3

- SAXパーサの生成
  - SAXパーサを作るためのオブジェクトの生成

```
SAXParserFactory spf =
SAXParserFactory.newInstance();
```

  - SAXパーサ（XML文書をSAXで扱うためのオブジェクト）の生成

```
SAXParser sp = spf.newSAXParser();
```


SAXの準備#4

- ハンドラの作成
  - この本では，DefaultHandlerクラスを継承したSampleHandlerクラスを作って，そこでSAXの処理を扱っている．

```
SampleHandler1 sh = SampleHandler1();
```

  - mainメソッドでhandlerを扱う場合には，

```
public class クラス名 extends
DefaultHandler {
…
DefaultHandler sh = クラス名();
```

SAXの準備#5

- XML文書の読み込み

```
sp.parse(new FileInputStream("Sample.xml"), sh);
```


ハンドラのメソッド

メソッド	受け取る内容
void startDocument()	文章開始
void endDocument()	文書終了
void startElement(String namespaceURI, String localname, String qName, Attributes atts)	要素の開始
void endElement(String namespaceURI, String localname, String qName, Attributes atts)	要素の終了
void characters(char[] ch, int start, int length)	文字データ

ハンドラの記述

教科書３９５ページ，Sample1.java

```
class SmapleHandler1 extends DefualtHandler {
  public void startDocument() {
    System.out.println("XML文書が処理開始");
  }

  public void endDocument() {
    System.out.println("XML文書が終了しました.");
  }
}
```

startDocument()とendDocument()は，

DefualtHandlerのメソッドオーバライド

startElement,endElement

```
class SmapleHandler2 extends DefualtHandler {
  public void startDocument() {
    System.out.println("XML文書が開始しました.");
  }
  public void endDocument() {
    System.out.println("XML文書が終了しました.");
  }
  public void startElement(String namespaceURI, String localName,
     String qName, Attributes attrs) {
     System.out.println(qName+ "が開始しました.");
  }
  public void endElement(String namespaceURI, String localName,
String qName, Attributes attrs) {
    System.out.println(qName+ "が終了しました.");
  }
}
```

教科書401ページ，Sample2.java

characters

```
class SmapleHandler2 extends DefualtHandler {
  public void startDocument() {
    System.out.println("XML文書が開始しました.");
  }
  …

  public void characters(char[] ch, int start, int
length) {
    String str = new String(ch, start, length);
    if (str.trim().length() != 0) {
      System.out.println(str);
    }
  }
}
```

教科書405ページ，Sample2_2.java

characters(char[] ch, int start, int length)

- char[] ch：characters()が起動された時点での読み込まれているXML文書の「すべての文字データ」
- int start：今回characterの起動対象となった「文字データ」のchar[] ch配列でのオフセット値
- int length：その「文字データ」 の長さ

補足：javaのメソッド#1

- String(char[] ch, int start, int length)
  - 文字配列chのうち，オフセット値startから長さlengthの部分を文字列として生成する.
  - 例

```
char a[]={'D', 'O', 'M'};
String s1 = new String(a, 0, 2);
System.out.println(a);
System.out.println(s1);
```

```
DOM
DO
```

test1.java

補足：javaのメソッド#2

- trim()
  - 対象の文字列の前後の空白を削除する
  - 例

```
char b[]={' ','S', 'A', 'X', ' '};
String s2 = new String(b).trim();
System.out.println(b);
System.out.println(s2);
```

```
 SAX
SAX
```

test1.java

要素の取り出し

- ターゲットの要素の開始部を特定 (startElement)
  - ターゲット要素処理中かを判断するフラグ isPrint
- その要素の文字データ部分の抽出 (characters)
- ターゲットの要素の終了部を特定 (endElement)

教科書４０８ページ，Sample3.java

属性の取り出し

- 要素がある場合には，startElementの引数Attributesに属性リストがセットされる．そのリストに対して，以下のメソッドがある．

教科書４１１ページ，Sample4.java

メソッド	取り出す内容
int getLength()	属性の数
String getQName(int n)	n番目の属性名
String getValue(int n)	n番目の属性値を

属性の取り出しの例

```
public void startElmenet(String namespaceURI, String
localName, String qName, Attributes attrs) {
  for(int i=0; i<attrs.getLength(); i++) {
        System.out.println("要素 "+qName);
        System.out.println("属性 "+attrs.getQName(i));
        System.out.println("属性値 "+attrs.getValue(i));
        }
        }
```

Sample4_2.java, Sample4_3.java

練習：教科書４２６ページ

- SAXを使って，書籍のタイトルと価格を画面に表示するコードを記述せよ．

```
<?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>
```

```
入門SQL
2400
やさしいC++
2500
やさしいJava
2600
```

画面出力

練習：教科書４２７ページ

- タイトルだけを取り出したresult.xmlを作成してください.

```
<?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>
```

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

result.xml

DOMの長所

- 長所
  - XML文書の構造を踏まえた，細かい操作ができる.
  - XML文書の各要素にランダムアクセスでき，効率がいい.

DOMの短所

- 短所
  - XML文書全体をメモリ上に展開するので，メモリを大量に食う.
  - XML文書をメモリ上に展開できないと，処理ができない.

SAX

- XML文書を順番に読み込み
- 出現した要素の種類に応じて，イベント（処理）が行われる.
- だたし，XML文書の編集機能はない.

SAXの長所

- 長所
  - メモリの使用量が少ない.
  - XML文書を順番に処理するので，解析速度が速い.

SAXの短所

- 短所
  - XML文書を先頭から読み込むので，複雑な処理がやりにくい.
  - XML文書の編集ができない.

DOM v.s. SAX

- DOMは，XML文書をツリー構造に展開し，それを操作する.
- SAXはXML文書の要素を，イベントドリブン（イベント駆動）で，順次処理していく.

まとめ

- SAXについて
- JavaによるSAXの利用
- ・・・