ーXMLを利用するサーバサイドJavaー

岐阜経済大学　経営学部 経営情報学科　井戸　伸彦

来歴：

0．0版　2004年8月8日

スライドの構成

はじめに

（1）

はじめに

- 本スライドでは、Ｊａｖａサーブレットにて ＸＭＬを利用する次の２つの方法について、初歩的な説明を行います。
  - ＸＳＬＴ、Ｘａｌａｎを用いたサーブレット
  - ＤＯＭ、Ｘｅｒｃｅを用いたＪＳＰ
- 本スライドでは、次のスライドは学習済みであるとを前提としています。
  (1)「月に吠える　ーeclipseを用いたＪａｖａアプリケーションの作成ー」
  (2)「ただ一疋の青い猫のかげ　ーeclipseを用いたＪａｖａサーブレットの作成ー」
  (3)「されど我らが日々　ーJavaサーブレット入門ー」
  (4)「ーＸＭＬ入門とＸＳＬＴー」

  特に、(4) のスライドでの実習で作成したＸＭＬ文書、ＸＳＬＴスタイルシートは、本スライドの実習でそのまま使用します。
- 直感的な説明を行い、若干不正確な言い回しを含んでいます。
- 実習は次の環境で行うことを想定しています。
  - Linux PC(実際に授業で使ったのは、Fedora Core2)
  - j2sdk、tomcatインストール済み

  eclipse (Ｌｏｍｂｏｚ) インストール済み
- オンラインでソースは公開しています。

(1) Javaサーブレットでの XML利用

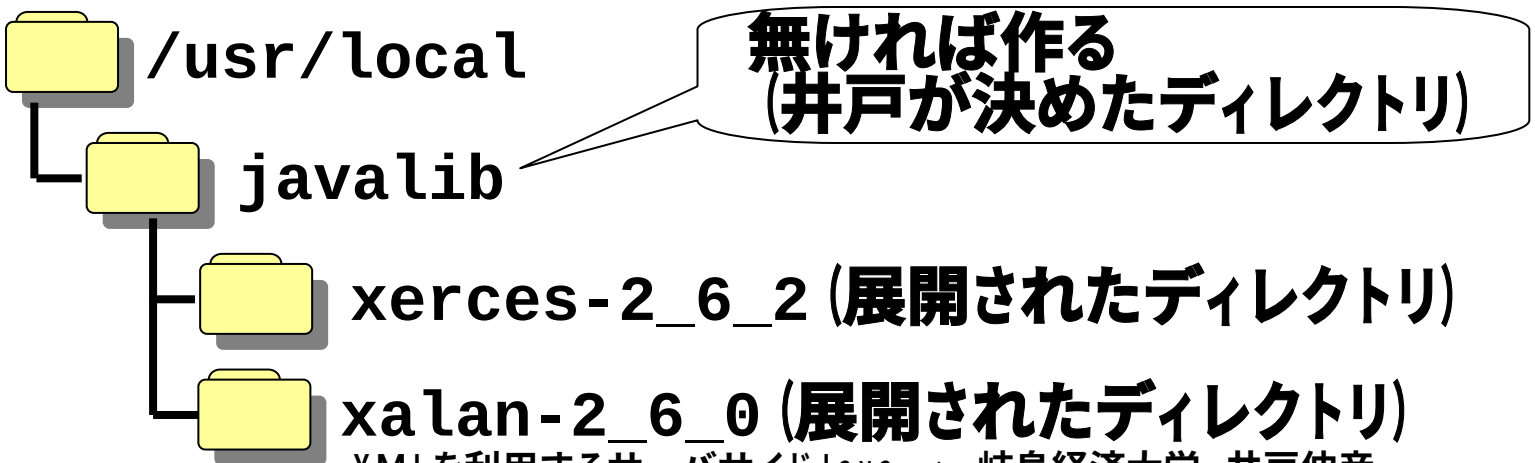
- 本スライドでは、次の2つの形でサーブレットでXMLを利用する方法を学びます。
- XSLT利用のサーブレット (スライド(2))
  - XSLTプロセッサXalanを利用して、複数のXSLTスタイルシートから1つを動的に選んでXML文書に適用してページを返すサーブレット。
- DOM利用のJSP (スライド(3)～(6))
  - XML文書をDOMを利用して読み取り、内容を表示するJSP
  - ブラウザからの登録により、DOMを利用してXML文書を更新するJSP
- 上記の2つは、サーバサイドプログラミングであること以外は共通点の少ないアプリケーションであるため、本スライドでも、上記のスライド対応に内容が分かれています。

(1．1．1) Xerces,Xalanのインストール

■入手するファイルは2つ。
- Xerces : Xerces-J-bin.2.6.2.tar.gz (2004.8.11での最新版)
- Xalan　: xalan-j_2_6_0-bin.tar.gz (同上)

■例えば次のサイトからダウンロードする。
- Xerces : http://nagoya.apache.org/mirror/xml/xerces-j/
- Xalan　: http://nagoya.apache.org/mirror/xml/xalan-j/

■展開(# tar zxvf xxxxx.tar.gz)し、スーパーユーザとなって、次のディレクトリに置く。



(1．1．2) クラス･パスの設定

■次のクラス･パスを、各自のアカウント･ルートにあるファイル“.bashrc”に書き加える。

(きちんと編集する自信の無い人は、井戸のサイトからコピー＆ペーストしてください。)

```
XERCES_HOME=/usr/local/javalib/xerces-2_6_2
CLASSPATH=$XERCES_HOME/xmlParserAPIs.jar:$CLASSPATH
CLASSPATH=$XERCES_HOME/xercesImpl.jar:$CLASSPATH
CLASSPATH=$XERCES_HOME/xercesSamples.jar:$CLASSPATH
CLASSPATH=$XERCES_HOME/resolver.jar:$CLASSPATH
CLASSPATH=$XERCES_HOME/xml-apis.jar:$CLASSPATH

XALAN_BIN=/usr/local/javalib/xalan-j_2_6_0/bin
CLASSPATH=$XALAN_BIB/xalan.jar:$CLASSPATH
```

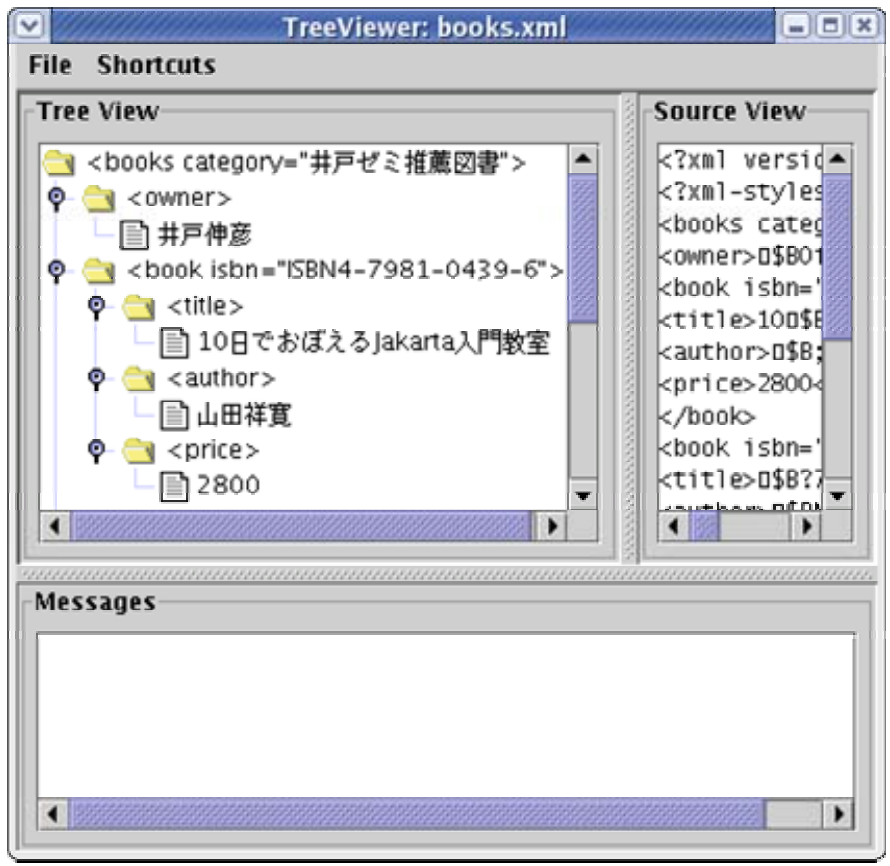
■同ファイルを実行する (# source .bashrc)。

(1．1．3) ui.TreeViewer

■XMLファイル(下記例では、“books.xml”)を次のとおり、TreeViewerで開く。

- $ java ui.TreeViewer books.xml

■右のように、ＸＭＬ文書の木構造が表示されるのを確認する。



（1．2）eclipseでのビルドパスの追加

■ビルド･パスに外部ＪＡＲの追加を行う。

■操作方法については、次の資料を参照。

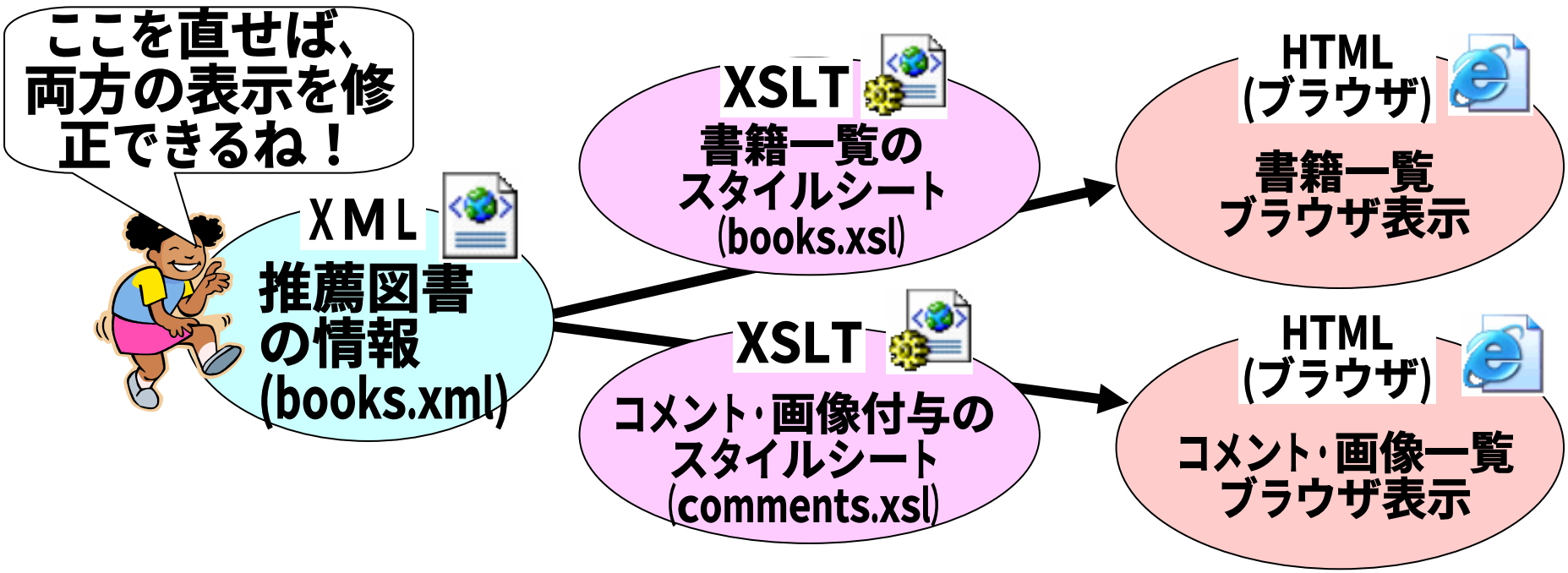
- 「ただ一疋の青い猫のかげ　ーeclipseを用いたＪａｖａサーブレットの作成ー」　スライド(2.3.2)～(2.3.4)

■追加を行うのは、次のファイル。

- Xerces
  - /usr/local/javalib/xerces-2_6_2直下のすべてのＪＡＲファイル。
  - /usr/local/javaliv/xalan-2_6_0/bin直下のすべてのＪＲファイル。

（2）XSLT利用のサーブレット

■ＸＳＬＴのスライドにて、次のような実習をスタンドアロンLinux PCにて行いました。



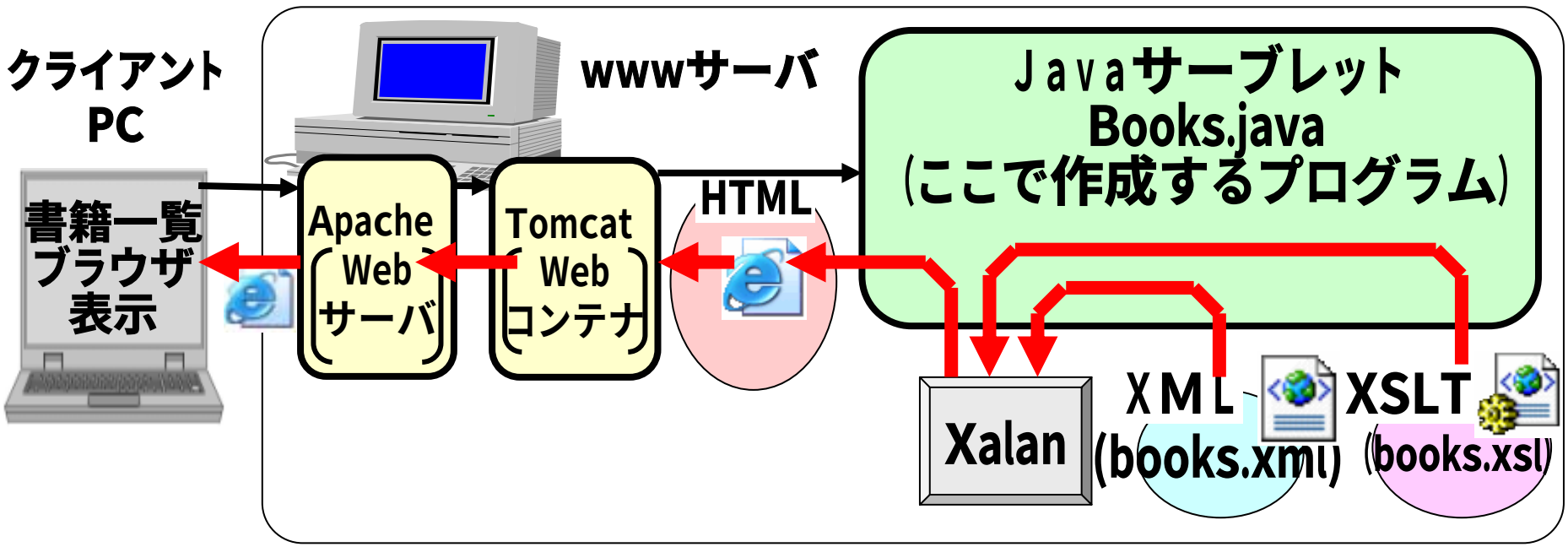
- 上記の処理では、「XML+スタイルシート　⇒　HTML」の変換を、ブラウザ (Mozilla) が行っていました。

■今回は、上記のＸＳＬＴ利用をサーバサイドで行います。

(2．1) Xalan (ザラン) の役割

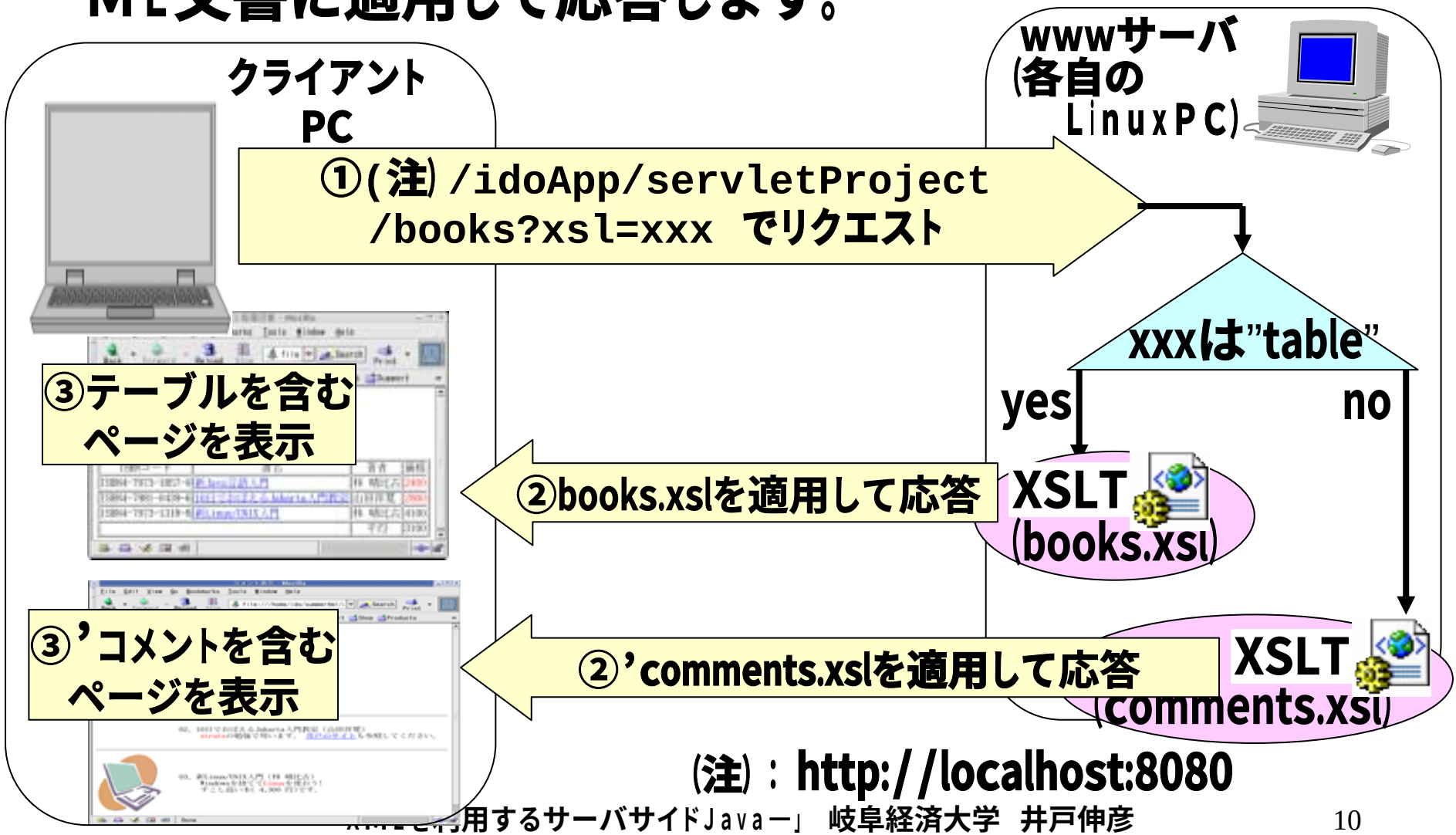
■前スライドでの処理では、ブラウザが行っていた、「XML+スタイルシート ⇒ HTML」の変換を、今回サーバサイドで行うのが、Xalanです。このようなソフトを、“ＸＳＬＴプロセッサ”と呼びます。

■今回はサーブレット (サーバサイド) でXalanを用いますが、そのような必然性はありません。すなわち、スタンドアロンＰＣ上のＪａｖａアプリケーションでXalanを使用することも、当然可能です。



(2．2) 作成するプログラムの動作

■リクエスト･パラメータにより、異なるスタイルシートをＸＭＬ文書に適用して応答します。



(2．3) ファイルの配置

■次のようにファイルを配置します。操作はeclipse上で行います。

スライド「ーXML入門とXSLTー」の実習で作成したファイルを用います。

（2．4．1）Books.java（1／2）

■Xalanに関連した処理（import以外）は、次のスライドにあります。

```
 1:import java.io.IOException;
 2:import javax.servlet.ServletException;
 3:import javax.servlet.http.*;
 4:import javax.xml.transform.*;
 5:import javax.xml.transform.stream.*;
 6:public class Books extends HttpServlet {
 7:   protected void doGet(
 8:     HttpServletRequest request,
 9:     HttpServletResponse response)
10:     throws ServletException, IOException {
11:       response.setContentType
("text/html;charset=iso-2022-jp");
```

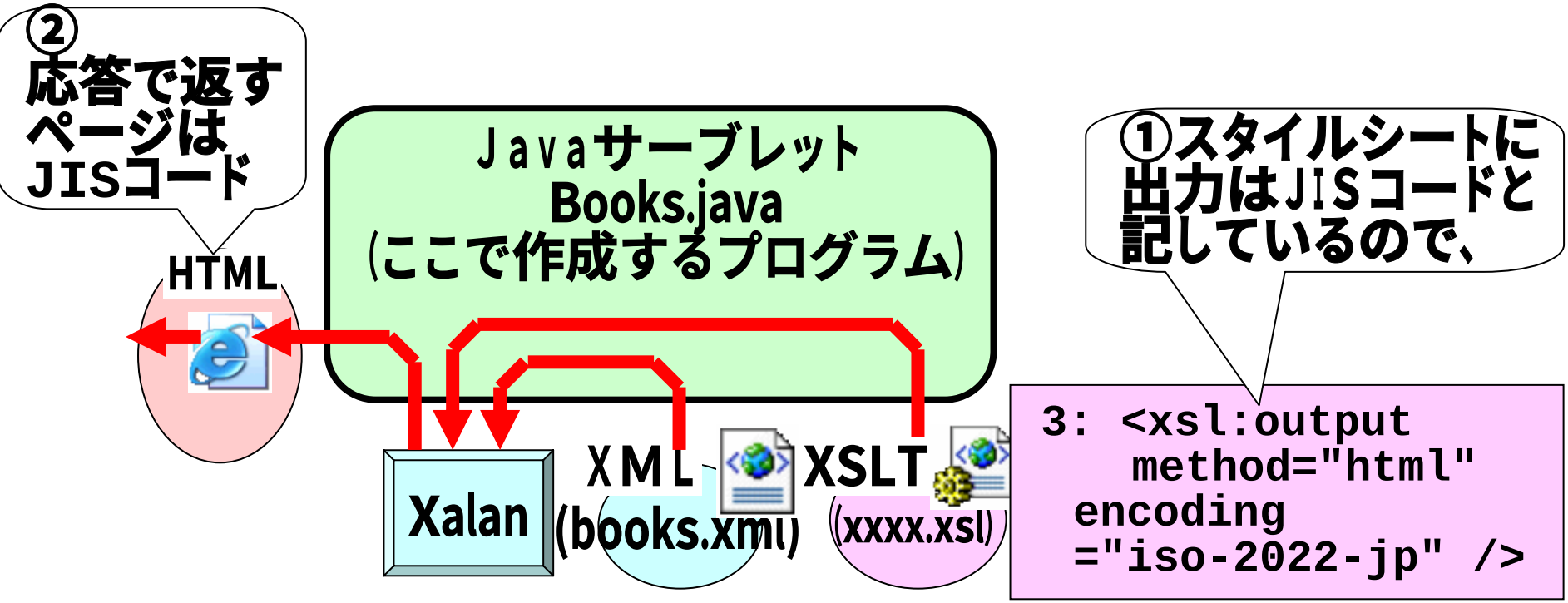
(2．4．2) Books.java (1／2)

```
12:    String xmlFile, xslFile;
13:    xmlFile=getServletContext().getRealPath
("/books.xml");
14:    if(request.getParameter("xsl")==null ||
15:       request.getParameter("xsl").equals("table")){
16:      xslFile=getServletContext().getRealPath
("/books.xsl");
17:    }else{
18:      xslFile=getServletContext().getRealPath
("/comments.xsl");
19:    }
20:    TransformerFactory tff = TransformerFactory
.newInstance();
21:    Transformer tf;
22:    try {
23:      tf = tff.newTransformer
(new StreamSource(xslFile));
24       tf.transform(new StreamSource(xmlFile)
,new StreamResult(response.getWriter()));
25:    } catch (Exception e) {
26:    }
27:  }
28:}
```

(2．5) 応答のコンテンツ形式の設定

■レスポンスの漢字コードが、JISコード (iso-2022-jp) であることを設定します。

```
11: response.setContentType
                ("text/html;charset=iso-2022-jp");
```



(2．6) ファイルの絶対パスの取得

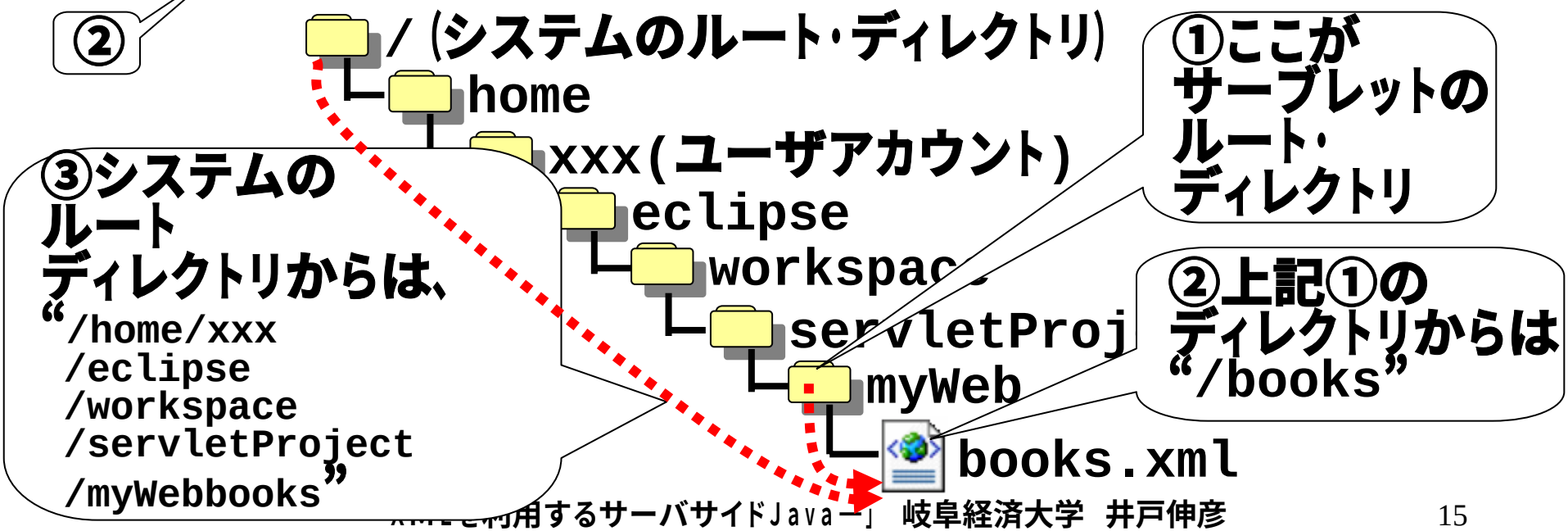
■これからスライド(2.3)に記したファイル“books.xml”を読み出すわけですが、その際には絶対パスが必要です。

■サーブレットのルート･ディレクトリからのファイル名から、絶対パスを求めておきます。

```
12:     String xmlFile, xslFile;
13:     xmlFile=getServletContext().getRealPath
("/books.xml");
```

③

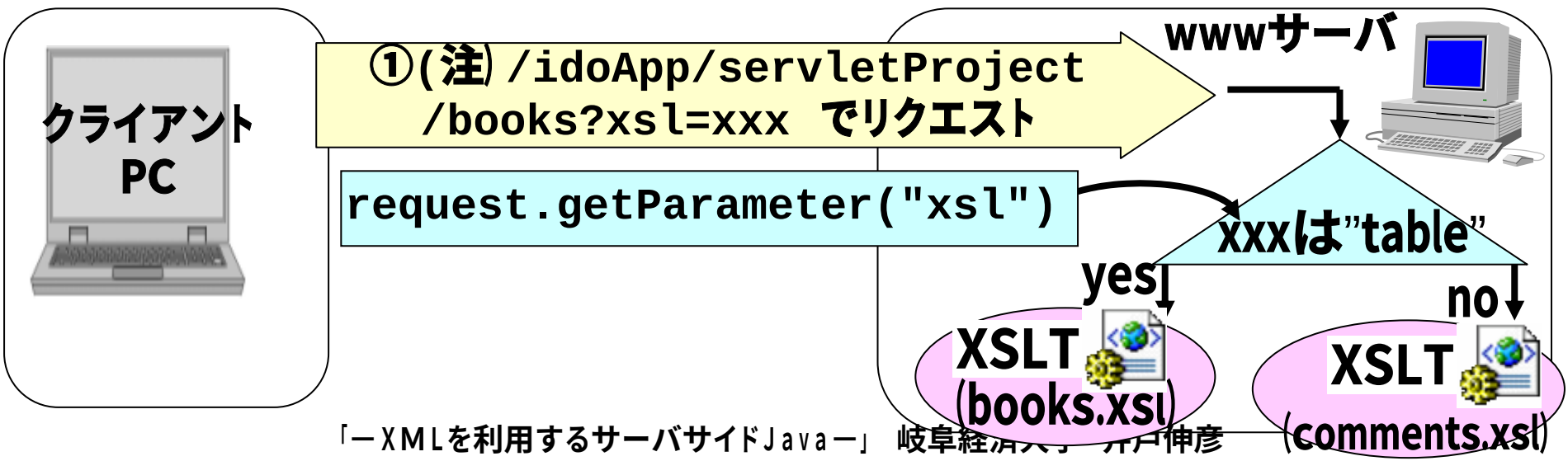
②



(2．7) リクエスト･パラメータにより分岐

■スタイル･シートについても絶対パスを求めますが、リクエストパラメタにより、“books.xsl”と“comments.xsl”とを読み分けます。

```
14:     if(request.getParameter("xsl")==null ||
15:        request.getParameter("xsl").equals("table")){
16:       xslFile=getServletContext().getRealPath
          ("/books.xsl");
17:     }else{
18:       xslFile=getServletContext().getRealPath
          ("/comments.xsl");
19:     }
```



(2．8．2) HTMLへの変換の実行

■ (XML＋XSLT)⇒HTMLの変換のプログラムは、このように行うものであると思っておいてください (次スライドに、直感的なイメージを記しています)。

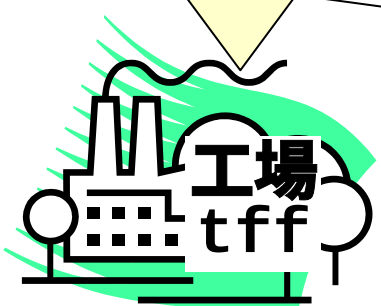
```
20:    TransformerFactory tff = TransformerFactory
                                    .newInstance();
21:    Transformer tf;
22:    try {
23:      tf = tff.newTransformer
                     (new StreamSource(xslFile));
24       tf.transform(new StreamSource(xmlFile)
            ,new StreamResult(response.getWriter()));
25:    } catch (Exception e) {
26:    }
```

■変換においては、例外 (TransformerConfigurationException、TransformerException、IOException) が発生する恐れがあるので、try～catch～で囲みます。

(2.8.2) XML＋XSLT⇒HTML変換のイメージ

```
20: TransformerFactory
     tff = TransformerFactory
     .newInstance();
```

①まず、変換機工場をつくり、



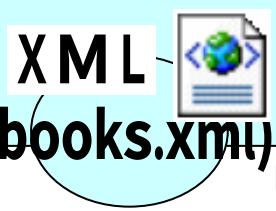

```
21: Transformer tf;
23: tf = tff.newTransformer
  (new StreamSource(xslFile));
```

②スタイルシートを与えて、
変換機つくり、


```
24 tf.transform(

new
StreamSource(
xmlFile)
```

③XML文書を与えて、
変換する。出力先は、
クライアントへのページ


```
,new StreamResult
(response.getWriter()));
```



(2．9) 課題

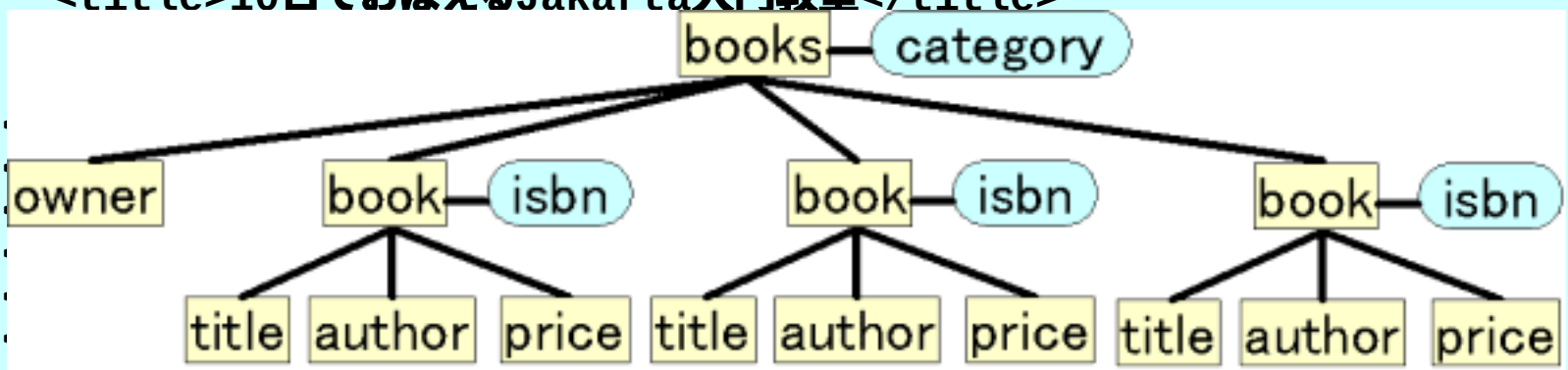
■スライド「ーＸＭＬ入門とＸＳＬＴー」の実習で作成したＸＭＬ文書とスタイルシートとを用いて、サーブレットを作成してください。

(3．1．1) DOM(*)とは何か？　－1－

(*)：Document Object Model

■スライド「－XML入門とXSLT－」では、XML文書が木構造を持っていることを学びました。

```
<?xml version="1.0" encoding="iso-2022-jp" ?>
<?xml-stylesheet type="text/xsl" href="books.xsl" ?>
<books category="井戸ゼミ推薦図書">
  <owner>井戸伸彦</owner>
  <book isbn="ISBN4-7981-0439-6">
    <title>10日でおぼえるJakarta入門教室</title>
```



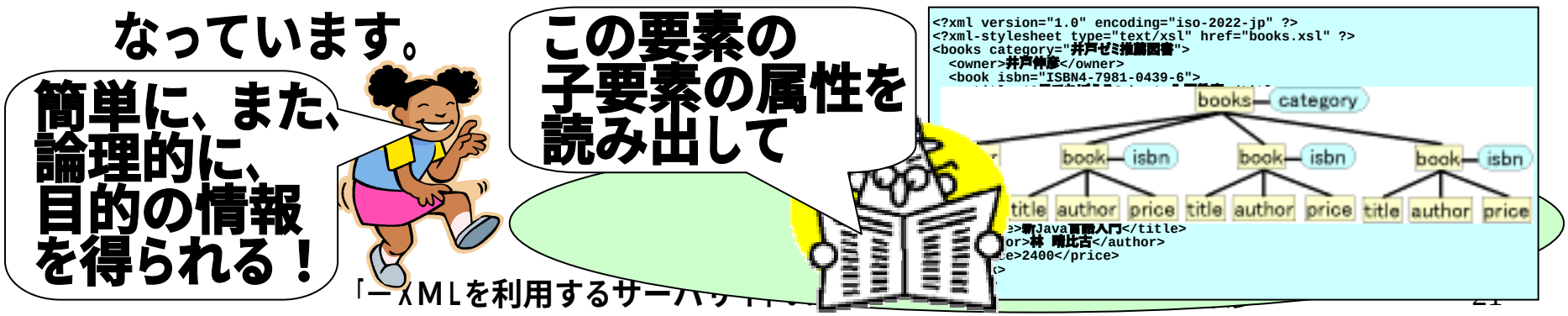

```
  <book isbn="ISBN4-7973-1857-0">
    <title>新Java言語入門</title>
    <author>林 晴比古</author>
    <price>2400</price>
  </book>
</books>
```

(3．1．2) XML文書の読み書きの仕方

■木構造を持つXML文書を、プログラムで読み書きすることを考えます。

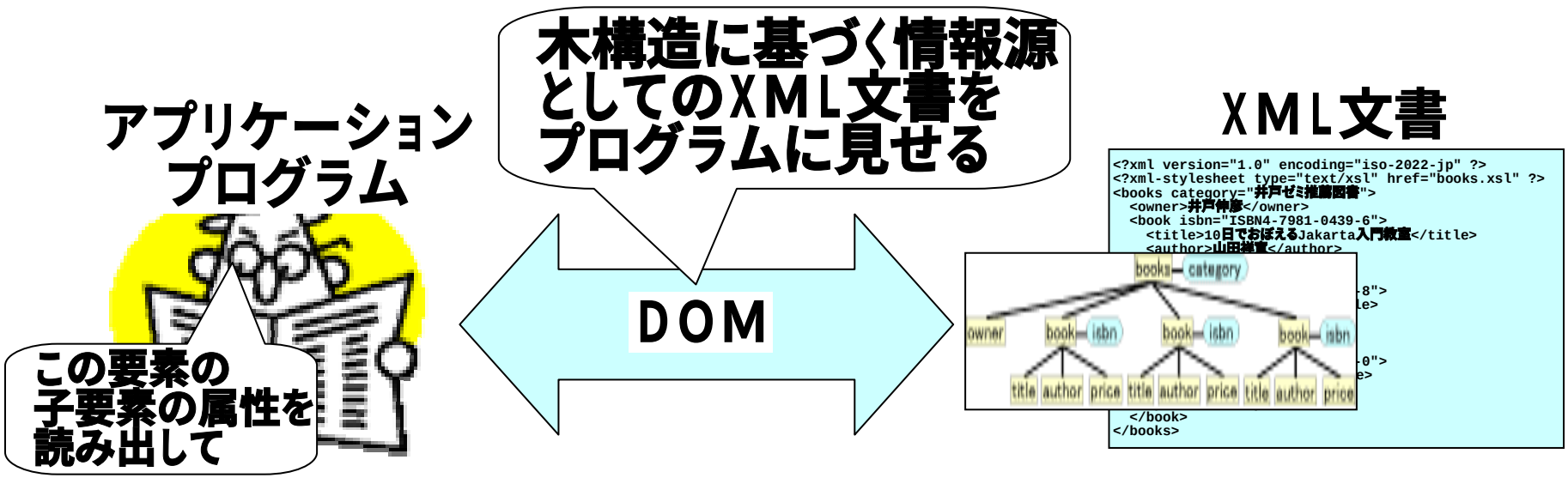
■物理的に読み書きをするやり方 (ex.「3行の5文字めから読み出す」) は、あまり利口なやり方とは言えません。

■木構造に基づき、論理的に読み書きをするやり方 (ex.「その要素の子要素の属性を読み出す」) が、本来のXMLの目的にかなっています。



(3．1．3) DOMの役割

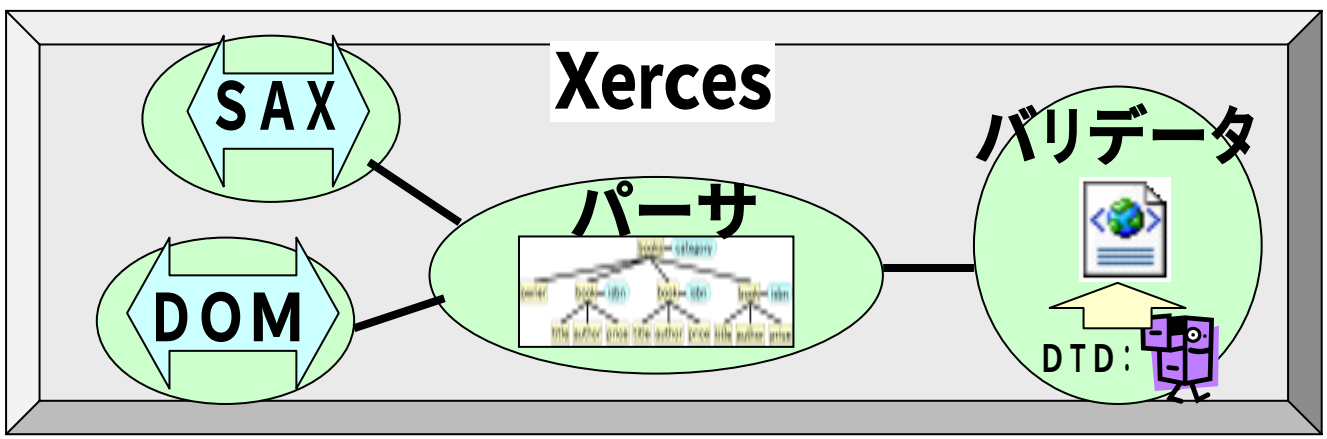
■DOMは、XML文書を木構造に基づく情報源として扱うための、プログラムへのインターフェースを指します。



■DOMは具体的なソフトウェアを指す訳ではありません。DOMの機能を実現した(実装した)ソフトウェアが、次に説明するXercesです。

（3．2．1）Xerces(ザーシズ)

- Xercesは、Tomcatも提供しているＡｐａｃｈｅプロジェクトが開発したオープンソースのソフトウェアです。
- 前述のとおりＤＯＭの実装であるだけでなく、パーサ(parser)、バリデータ(validator)など、ＸＭＬに関連する複数の機能を実装しています。
- ＤＯＭとして機能するためには、パーサの機能が必要なことは理解できるかと思います。Xercesの持つ機能の中で一番基本となる機能がパーサとなるため、“XercesはXMLパーサである”というような言い方もします。



(3．2．2) サーバサイド固有ではない

■Xalan同様、今回はサーブレット(サーバサイド)でXercesを用いますが、そのような必然性はありません。すなわち、スタンドアロンＰＣ上の　Ｊａｖａアプリケーションで Xercesを使用することも、当然可能です。

(3．3．1) 作成するプログラム

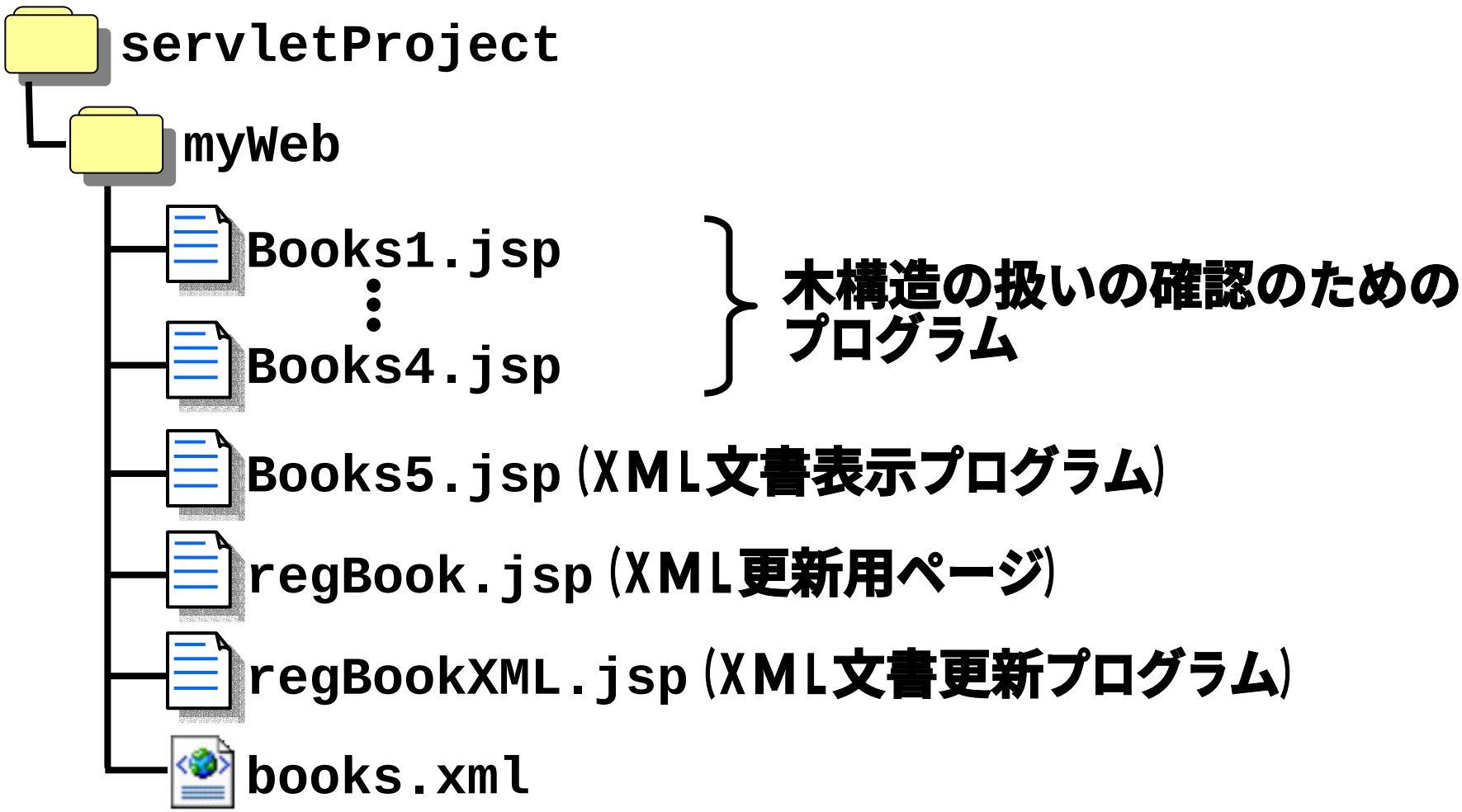
■DOMでの木構造の扱いを確認するためのプログラム
- 本の一覧をまとめたＸＭＬ文書、“books.xml”を例にして、木構造を表示してみます。

■サーバサイドプログラムとして、ＸＭＬ文書を表示するプログラム。
- スタイルシート“books.xsl”で作成したようなページを表示します。

■サーバサイドプログラムとして、ＸＭＬ文書を更新するプログラム。
- 本を登録し、books.xmlを更新するプログラムを作成します。

(3．3．2) ファイル構成

■今回はすべてＪＳＰで作成します。myWeb配下にファイルを置きます。

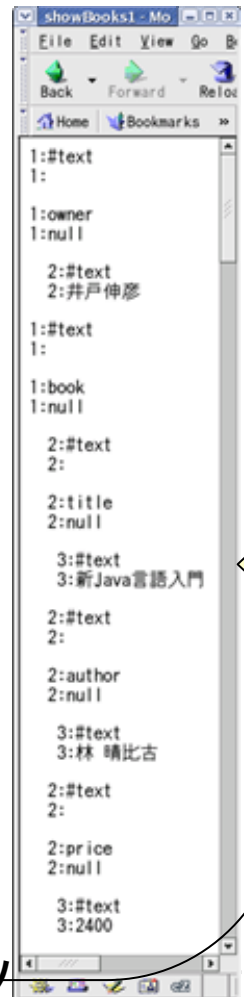


(4) 最初のDOMプログラム

クライアント
PC

```
1:#text
1:

1:owner
1:null

  2:#text
  2:井戸伸彦

1:#text
1:

1:book
1:null

  2:#text
  2:
```

```
2:title
2:null

  3:#text
  3:新Java言語入門

2:#text
2:

2:author
2:null

  3:#text
  3:林　晴比古

2:#text
2:

2:price
2:null

  3:#text
  3:2400
```



①(注)/idoApp
/servletProject
/showBooks1.jsp
でリクエスト

②応答

WWW
サーバ
(各自の
LinuxPC)

(4．1．1) showBooks1.jsp (1／2)

```
<%@ page language="java" pageEncoding="ISO-2022-JP"
         import="java.io.*,org.apache.xerces.parsers.*,
                 org.xml.sax.*,org.w3c.dom.*"%>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0
transitional//en">
<html>
<head>
<title>showBooks1</title>
</head>
<body bgcolor="#FFFFFF">
<% InputStreamReader Isr=new InputStreamReader(
      new FileInputStream(application.getRealPath(
                           "/books.xml")),"iso-2022-jp");
    BufferedReader br=new BufferedReader(Isr,10);
    InputSource src = new InputSource(br);
    DOMParser parser = new DOMParser();
    parser.parse(src);
    Document document = parser.getDocument();
    Element rootElement = document.getDocumentElement();
    NodeList nodeList = rootElement.getChildNodes();
%>
```

(4.1.1) showBooks1.jsp (2／2)

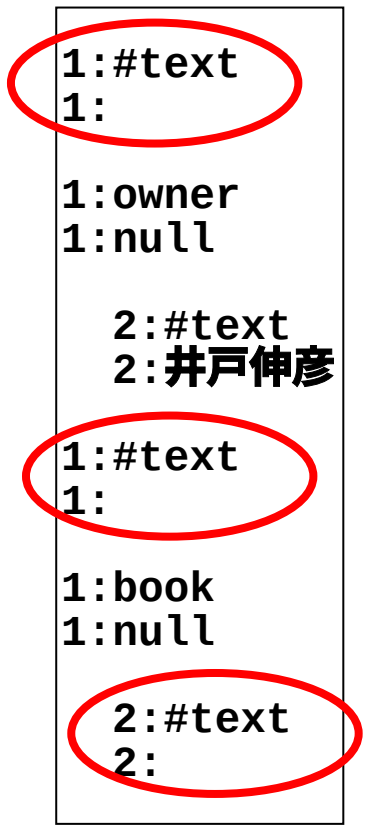
```
20:<% for(int i=0;i<nodeList.getLength();i++){
21:      Node node = nodeList.item(i);
22: %>
23:<pre>1:<%= node.getNodeName()%>
24:1:<%= node.getNodeValue()%></pre>
25:<%   NodeList nodeList2 = node.getChildNodes();
26:      for(int j=0;j<nodeList2.getLength();j++){
27:        Node node2 = nodeList2.item(j);
28:%>
29:<pre>  2:<%= node2.getNodeName() %>
30:  2:<%= node2.getNodeValue() %></pre>
31:<%      NodeList nodeList3 = node2.getChildNodes();
32:        for(int k=0;k<nodeList3.getLength();k++){
33:          Node node3 = nodeList3.item(k);
34:%>
35:<pre>   3:<%= node3.getNodeName() %>
36:   3:<%= node3.getNodeValue() %></pre>
37:<%      }
38:      }
39:   }
40:%>
41:</body>
42:</html>
```

(4．2) 何が表示されているか？

■スライド(1.1.3)に示したui.TreeViewerのように、ＸＭＬ文書の木構造を表示する (ただし、テキストのみの表示)。

■何か変でないか？

- 意味のない、空白の要素 (ノード) が表示されている (右図) 。
  ⇒ これは、books.xml中の改行やタブ。
  (DTDが無いと、改行等をテキストと扱うしかない)



(4．3．1)インポート

■最初に、ページ･ディレクティブにて、必要なパッケージをインポートしています。

```
1:<%@ page language="java" pageEncoding="ISO-2022-JP"
2:        import="java.io.*,org.apache.xerces.parsers.*,
3:                org.xml.sax.*,org.w3c.dom.*"%>
```

- 名前から、内容が想像がつくものがありますね。

(4．3．2)ＸＭＬファイルの読み出し

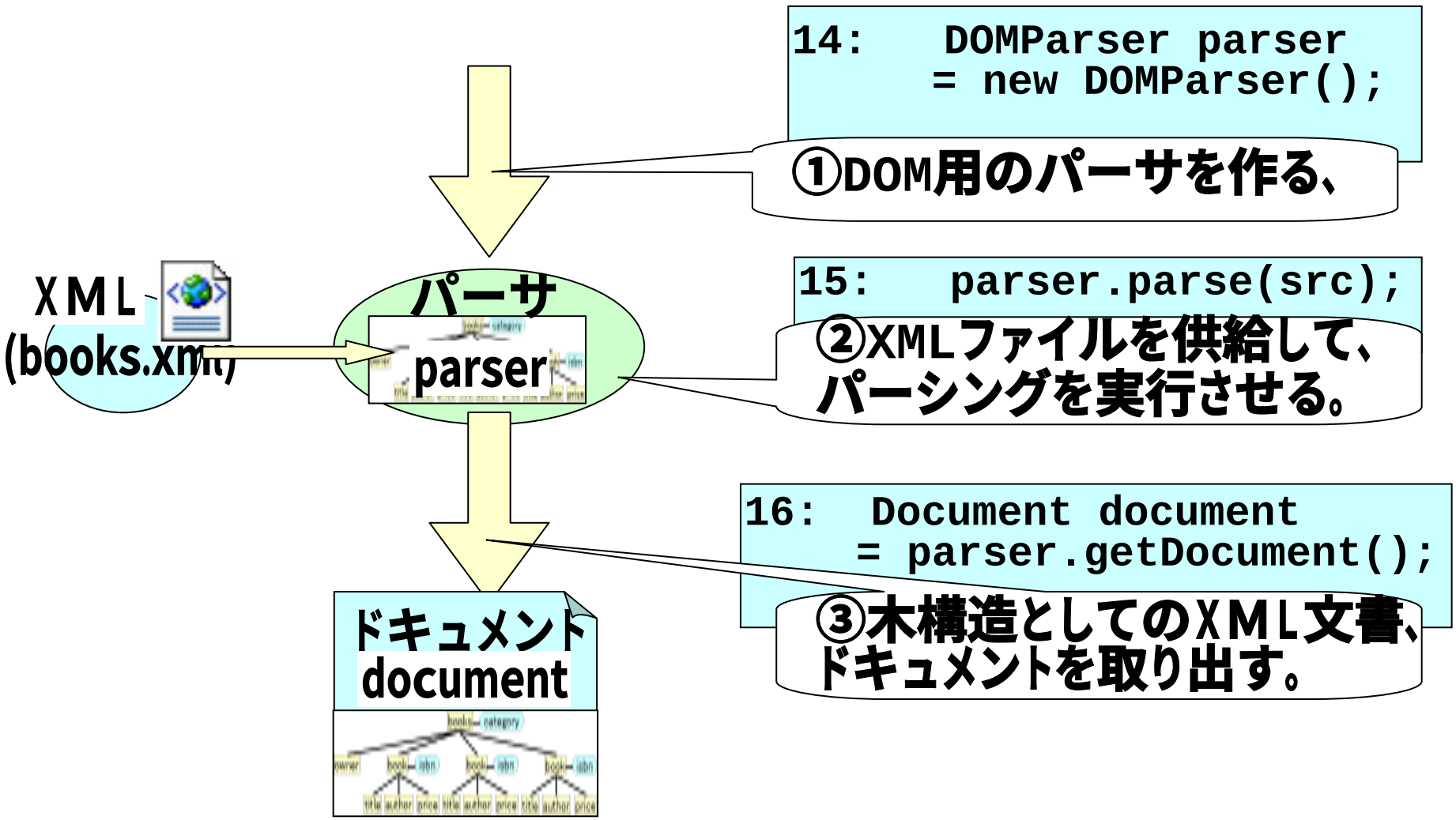
■ＸＭＬファイル“books.xml”を読み出す準備をします。

```
10:<% InputStreamReader Isr=new InputStreamReader(
11:      new FileInputStream(application.getRealPath(
                                  "/books.xml")),"iso-2022-jp");
12:    BufferedReader br=new BufferedReader(Isr,10);
13:    InputSource src = new InputSource(br);
```

- 詳細については、省略します。
- インスタンス“src”からＸＭＬファイルが読み出される準備が出来ました。

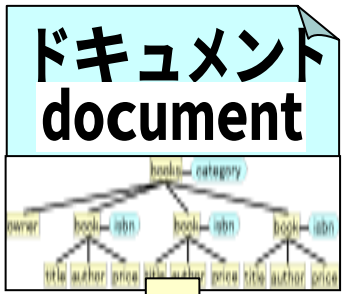
(4．3．3) 木構造としてのXML文書を取り出す

■次の手順で木構造としてのXML文書を取り出します。



(4．3．4) 木構造としてXML文書を操作する

■ルート要素を取り出し、その子要素のリストを取り出します。




```
17: Element rootElement
    = document.getDocumentElement();
```

①ルーと要素を取り出し、

books

```
18: NodeList nodeList
    = rootElement.getChildNodes();
```

②その子要素のリストを取り出す

owner　book　book　book

子要素のリスト　nodeList

(4．3．5)順次処理

■子要素のリストに①順次、次の処理を行います。

- ②子要素を取り出す。
- ③子要素の名前を表示する。
- ④要素の値を表示する。
- 子要素のさらに子要素のリストを作り、3段まで、同様の処理をする。

■次のプログラムの部分は、2段目の部分になります。

```
26:      for(int j=0;j<nodeList2.getLength();j++){
27:        Node node2 = nodeList2.item(j);
28:%>
29:<pre>   2:<%= node2.getNodeName() %>
30:   2:<%= node2.getNodeValue() %></pre>
```

①

②

③

④

(4．4) 変更版

■showBook1.jspでは、次の問題があります。
- スライド(4.2)に示したとおり、改行等による要素（ノード）を表示している。
- 3世代までしか子要素を表示しない。

■showBook2.jspでは、次のように変更しています。
- メソッド“isIgnorable”(無視できるか)にて、空白文字しか含まない要素を判定し、表示を止めている。
- リカーシブ・コール（再帰呼び出し）により、任意世代の子要素を表示する。

■showBook3.jspでは、メソッド“isIgnorable”のみ実装しています。

■課題：showBook2.jspの動作を理解して、表示方法をリファインしてください。

(4．5)属性の表示

■showBook4.jspでは、showBook2.jspに次のラインを追加して、属性を表示しています。

```
25: NamedNodeMap attributes = node.getAttributes();
26: if(attributes != null){
27: for(int i=0;i<attributes.getLength();i++){
28:       Attr attr = (Attr)attributes.item(i);
29:       buf.append(space+space.length()+":attribute“
+i+":“+attr.getName()+"¥n");
30:       buf.append(space+space.length()+":attribute“
+i+":value:"+attr.getValue());
31:    }
32: }
```

- 属性の取り出しは、要素の取り出しと似ています。

(5) XML文書の表示

```
<%@ page language="java" pageEncoding="UTF-8"
         import="java.io.*,org.apache.xerces.parsers.*,
                 org.xml.sax.*,org.w3c.dom.*"%>
<!DOCTYPE HTML PUBLIC "-//w3c//dtd html 4.0 transitional//e
<html>
<head>
<title>showBooks5</title>
</head>
<body bgcolor="#FFFFFF">
<%
  InputStreamReader Isr=new InputStreamReader(
    new FileInputStream(application.getRealPath("/books.xml
  BufferedReader br=new BufferedReader(Isr,10);
  InputSource src = new InputSource(br);
  DOMParser parser = new DOMParser();
  parser.parse(src);
  Document document = parser.getDocument();
  Element rootElement = document.getDocumentElement();
  NodeList nodeList = rootElement.getChildNodes();
%>
```

```
<table border="1">
<tr><th>ISBN</th><th>title</th><th>author</th><th>price</th
<%
  LP:for(int i=0;i<nodeList.getLength();i++){
    Node node = nodeList.item(i);
    if(!node.getNodeName().equals("book")){
      continue LP;
    }
    NamedNodeMap attributes = node.getAttributes();
    Attr attr = (Attr)attributes.item(0);
    String isbn = attr.getValue();
%>
      <tr><td><%= isbn%></td>

          // 次のスライド

      </tr>
<%
            }
%>
</table>
</body>
</html>
```

```
<%
   NodeList nodeList2 = node.getChildNodes();
   for(int j=0;j<nodeList2.getLength();j++){
     Node node2 = nodeList2.item(j);
     if(node2.getNodeName().equals("title")){
%>
      <td> <%= node2.getFirstChild().getNodeValue() %></td>
<%
     }else if(node2.getNodeName().equals("author")){
%>
      <td> <%= node2.getFirstChild().getNodeValue() %></td>
<%
     }else if(node2.getNodeName().equals("price")){
%>
      <td> <%= node2.getFirstChild().getNodeValue() %></td>
<%
     }
   }
%>
```