BASI DI DATI II – 2 modulo
Parte VI: XML programming

Prof. Riccardo Torlone

Università Roma Tre

Outline

- How XML may be manipulated from general-purpose programming languages
- How streaming may be useful for handling large documents

Goal

- You want to read/write data from/to XML files, and you don't want to write an XML parser.
- Applications:
  - □ processing an XML-tagged document
  - □ saving configs, prefs, parameters, etc. as XML files
  - □ sharing results with outside users in portable format
  - □ alternative to serialization for persistent store
  - □ …

General Purpose XML Programming

- Needed for:
  - complex XML domain-specific applications
  - implementing new generic XML tools

- Primitives:
  - parsing XML documents into XML trees
  - navigating through XML trees
  - manipulating XML trees
  - serializing XML trees as XML documents

The DOM API

- Document Object Model: a W3C proposal
- A language neutral API for manipulating XML trees
- Written in OMG Interface Definition Language
- A language binding translates these interfaces into native syntax (e.g., Java)

JAXP

- Java API for XML Processing
- Java implementation of DOM
  - All JAXP packages are included standard in JDK 1.4+
- It also includes a SAX implementation (see later)

DOM Interfaces

- The interface `Node` has a number of derived interfaces:
  - `Document`
  - `Element`
  - `Attr`
  - `Entity`
  - `ProcessingInstruction`
  - `CharacterData`
- `CharacterData` has two derived interfaces:
  - `Text`
  - `Comment`

DOM primitives

- Navigation:
  - `getParentNode`
  - `getNextSibling`
  - `getFirstChild`
  - `getChildNodes` (returns a `NodeList` interface)
  - ...
- Access to nodes:
  - `getNodeType`
  - `getNodeName`
  - `getNodeValue`
  - `...`

A JAXP example

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

public class first_level {
  public static void main(String args[]) {
    try {
      DocumentBuilderFactory factory =
         DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = factory.newDocumentBuilder();
      Document document = builder.parse(“file.xml");
      Element root = document.getDocumentElement();
      Node n = root.getFirstChild();
         while (n != null) {
            System.out.println(n.getNodeType());
            System.out.println(n.getNodeName());
            System.out.println(n.getNodeValue());
            n= n.getNextSibling();
         }
      }
    catch (Exception e) { e.printStackTrace(System.out); }
   }
}
```

JDOM

- DOM can be awkward for Java programmers
  - Language-neutral: does not use Java features
    - Example: `getChildNodes()` returns a `NodeList`, which is not a `List`. (`NodeList.iterator()` is not defined.)
- JDOM looks like a good alternative:
  - open source project, Apache license
  - builds on top of JAXP, integrates with SAX and DOM
  - similar to DOM model
  - API designed to be easy for Java programmers
  - exploits power of Java language: collections, method overloading
  - rumored to become integrated in future JDKs

The JDOM Framework

- It integrates DOM data structures into the Java language
  - the `Java.util.List` interface is used to represent collections of elements and attributes
  - `Iterator` objects are used to traverse XML node collections
- An implementation of generic XML trees in Java
- Nodes are represented as classes and interfaces

JDOM Classes and Interfaces

- The abstract class `Content` has subclasses:
  - □ `Comment`
  - □ `DocType`
  - □ `Element`
  - □ `EntityRef`
  - □ `ProcessingInstruction`
  - □ `Text`
- Other classes are `Attribute` and `Document`
- The `Parent` interface describes `Document` and `Element`

A Simple Example

```
int xmlHeight(Element e) {
  java.util.List contents = e.getContent();
  java.util.Iterator i = contents.iterator();
  int max = 0;
  while (i.hasNext()) {
    Object c = i.next();
    int h;
    if (c instanceof Element)
      h = xmlHeight((Element)c);
    else
      h = 1;
    if (h > max)
      max = h;
  }
  return max+1;
}
```

Another Example

```
static void doubleSugar(Document d)
  throws DataConversionException {
  Namespace rcp =
    Namespace.getNamespace("http://www.uniroma3.it/recipes");
  Filter f = new ElementFilter("ingredient",rcp);
  java.util.Iterator i = d.getDescendants(f);
  while (i.hasNext()) {
    Element e = (Element)i.next();
    if (e.getAttributeValue("name").equals("sugar")) {
      double amount = e.getAttribute("amount").getDoubleValue();
      e.setAttribute("amount",new Double(2*amount).toString());
    }
  }
}
```

A Final Example (1/3)

- Modify all elements like

```
<ingredient name="butter" amount="0.25" unit="cup"/>
```

into a more elaborate version:

```
<ingredient name="butter">
  <ingredient name="cream" unit="cup" amount="0.5" />
  <preparation>
    Churn until the cream turns to butter.
  </preparation>
</ingredient>
```

A Final Example (2/3)

```
void makeButter(Element e) throws DataConversionException {
  Namespace rcp =
    Namespace.getNamespace("http://www.uniroma3.it/recipes");
  java.util.ListIterator i = e.getChildren().listIterator();
  while (i.hasNext()) {
    Element c = (Element)i.next();
    if (c.getName().equals("ingredient") &&
        c.getAttributeValue("name").equals("butter")) {
      Element butter = new Element("ingredient",rcp);
      butter.setAttribute("name","butter");
```

A Final Example (3/3)

```
      Element cream = new Element("ingredient",rcp);
      cream.setAttribute("name","cream");
      cream.setAttribute("unit",c.getAttributeValue("unit"));
      double amount = c.getAttribute("amount").getDoubleValue();
      cream.setAttribute("amount",new Double(2*amount).toString());
      butter.addContent(cream);
      Element churn = new Element("preparation",rcp);
      churn.addContent("Churn until the cream turns to butter.");
      butter.addContent(churn);
      i.set((Element)butter);
    } else {
      makeButter(c);
    }
  }
}
```

Parsing and Serializing

```
public class ChangeDescription {
  public static void main(String[] args) {
    try {
      SAXBuilder b = new SAXBuilder();
      Document d = b.build(new File("recipes.xml"));
      Namespace rcp = Namespace.getNamespace(
             "http://www.uniroma3.it/recipes");
      d.getRootElement().getChild("description",rcp)
                        .setText("Cool recipes!");
      XMLOutputter outputter = new XMLOutputter();
      outputter.output(d,System.out);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Validation (DTD)

```
public class ValidateDTD {
  public static void main(String[] args) {
    try {
      SAXBuilder b = new SAXBuilder();
      b.setValidation(true);
      String msg = "No errors!";
      try {
        Document d = b.build(new File(args[0]));
      } catch (JDOMParseException e ) {
        msg = e.getMessage();
      }
      System.out.println(msg);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Validation (XML Schema)

```
public class ValidateXMLSchema {
  public static void main(String[] args) {
    try {
      SAXBuilder b = new SAXBuilder();
      b.setValidation(true);
      b.setProperty(
        "http://java.sun.com/xml/jaxp/properties/schemaLanguage",
        "http://www.w3.org/2001/XMLSchema");
      String msg = "No errors!";
      try {
        Document d = b.build(new File(args[0]));
      } catch (JDOMParseException e ) {
        msg = e.getMessage();
      }
      System.out.println(msg);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

XPath Evaluation

```
void doubleSugar(Document d) throws JDOMException {
  XPath p = XPath.newInstance("//rcp:ingredient[@name='sugar']");
  p.addNamespace("rcp","http://www.uniroma3.it/recipes");
  java.util.Iterator i = p.selectNodes(d).iterator();
  while (i.hasNext()) {
    Element e = (Element)i.next();
    double amount = e.getAttribute("amount").getDoubleValue();
    e.setAttribute("amount",new Double(2*amount).toString());
  }
}
```

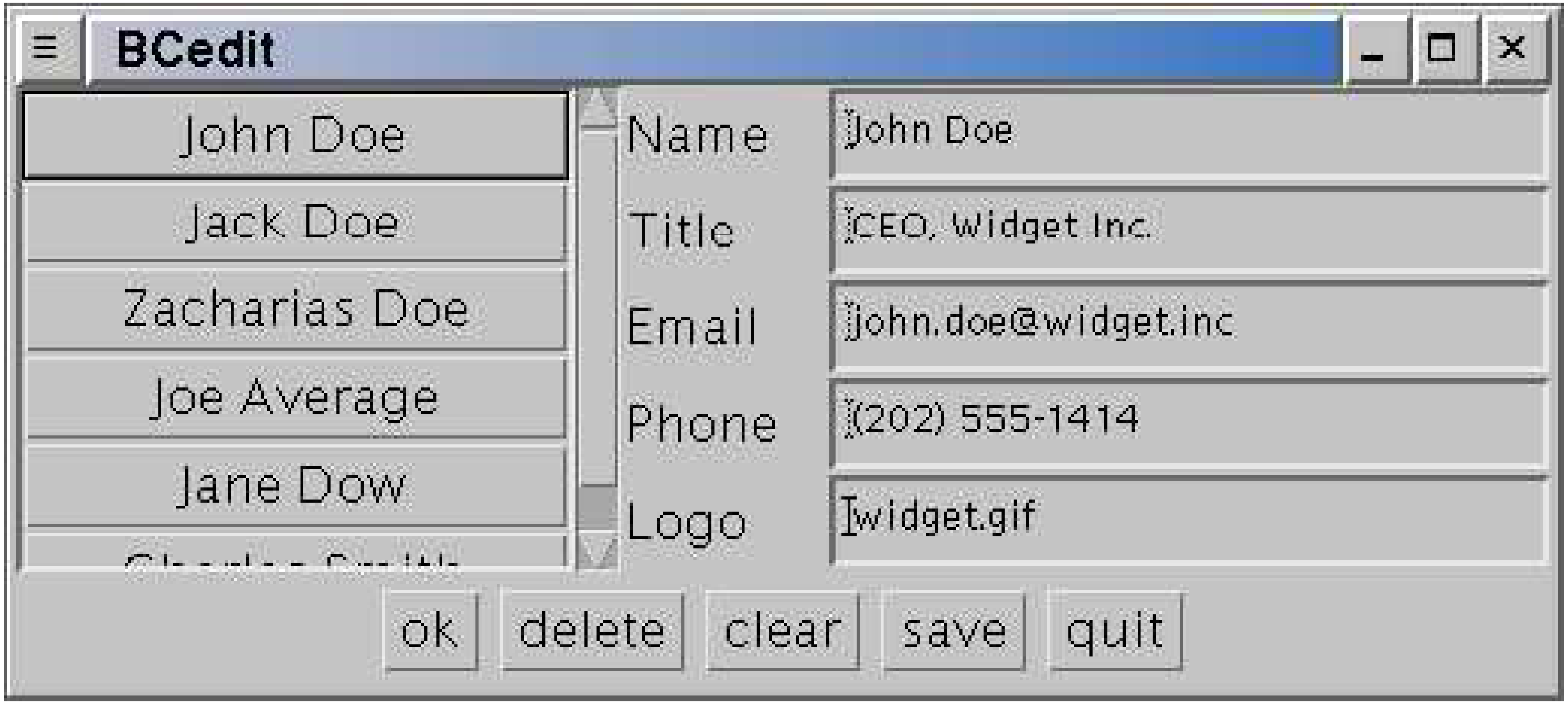
XSLT Transformation

```
public class ApplyXSLT {
  public static void main(String[] args) {
    try {
      SAXBuilder b = new SAXBuilder();
      Document d = b.build(new File(args[0]));
      XSLTransformer t = new XSLTransformer(args[1]);
      Document h = t.transform(d);
      XMLOutputter outputter = new XMLOutputter();
      outputter.output(h,System.out);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Business Cards

```
<cardlist xmlns="http://businesscard.org"
          xmlns:xhtml="http://www.w3.org/1999/xhtml">
  <title>
    <xhtml:h1>My Collection of Business Cards</xhtml:h1>
    containing people from <xhtml:em>Widget Inc.</xhtml:em>
  </title>
  <card>
    <name>John Doe</name>
    <title>CEO, Widget Inc.</title>
    <email>john.doe@widget.com</email>
    <phone>(202) 555-1414</phone>
  </card>
  <card>
    <name>Joe Smith</name>
    <title>Assistant</title>
    <email>thrall@widget.com</email>
  </card>
</cardlist>
```

Business Card Editor



John Doe
Jack Doe
Zacharias Doe
Joe Average
Jane Dow
Charles Smith

Name: John Doe
Title: CEO, Widget Inc.
Email: john.doe@widget.inc
Phone: (202) 555-1414
Logo: widget.gif

Class Representation

```
class Card {
  public String name,title,email,phone,logo;

  public Card(String name, String title, String email,
              String phone, String logo) {
    this.name=name;
    this.title=title;
    this.email=email;
    this.phone=phone;
    this.logo=logo;
  }
}
```

From JDOM to Classes

```
Vector doc2vector(Document d) {
   Vector v = new Vector();
   Iterator i = d.getRootElement().getChildren().iterator();
   while (i.hasNext()) {
     Element e = (Element)i.next();
     String phone = e.getChildText("phone",b);
     if (phone==null) phone="";
     Element logo = e.getChild("logo",b);
     String uri;
     if (logo==null) uri="";
     else uri=logo.getAttributeValue("uri");
     Card c = new Card(e.getChildText("name",b),
                       e.getChildText("title",b),
                       e.getChildText("email",b),
                       phone, uri);
     v.add(c);
   }
   return v;
}
```

From Classes to JDOM (1/2)

```
Document vector2doc() {
   Element cardlist = new Element("cardlist");
   for (int i=0; i<cardvector.size(); i++) {
     Card c = (Card)cardvector.elementAt(i);
     if (c!=null) {
       Element card = new Element("card",b);
       Element name = new Element("name",b);
       name.addContent(c.name); card.addContent(name);
       Element title = new Element("title",b);
       title.addContent(c.title); card.addContent(title);
       Element email = new Element("email",b);
       email.addContent(c.email); card.addContent(email);
```

From Classes to JDOM (2/2)

```
      if (!c.phone.equals("")) {
        Element phone = new Element("phone",b);
        phone.addContent(c.phone);
        card.addContent(phone);
      }
      if (!c.logo.equals("")) {
        Element logo = new Element("logo",b);
        logo.setAttribute("uri",c.logo);
        card.addContent(logo);
      }
      cardlist.addContent(card);
    }
  }
  return new Document(cardlist);
}
```

A Little Bit of Code

```
void addCards() {
   cardpanel.removeAll();
   for (int i=0; i<cardvector.size(); i++) {
     Card c = (Card)cardvector.elementAt(i);
     if (c!=null) {
       Button b = new Button(c.name);
       b.setActionCommand(String.valueOf(i));
       b.addActionListener(this);
       cardpanel.add(b);
     }
   }
   this.pack();
}
```

The Main Application

```
public BCedit(String cardfile) {
   super("BCedit");
   this.cardfile=cardfile;
   try {
     cardvector = doc2vector(
        new SAXBuilder().build(new File(cardfile)));
   } catch (Exception e) { e.printStackTrace(); }
   // initialize the user interface
   ...
 }
```

XML Data Binding

- The methods `doc2vector` and `vector2doc` are tedious to write

- XML data binding provides tools to:
  - □ map schemas to class declarations
  - □ automatically generate unmarshalling code
  - □ automatically generate marshalling code
  - □ automatically generate validation code

Binding Compilers

- Which schemas are supported?
- Fixed or customizable binding?
- Does roundtripping preserve information?
- What is the support for validation?
- Are the generated classes implemented by some generic framework?

The JAXB Framework

- It supports most of XML Schema
- The binding is customizable (annotations)
- Roundtripping is almost complete
- Validation is supported during unmarshalling or on demand
- JAXB only specifies the interfaces to the generated classes

Business Card Schema (1/3)

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org"
        elementFormDefault="qualified">

 <element name="cardlist" type="b:cardlist_type"/>
 <element name="card" type="b:card_type"/>
 <element name="name" type="string"/>
 <element name="email" type="string"/>
 <element name="phone" type="string"/>
 <element name="logo" type="b:logo_type"/>

 <attribute name="uri" type="anyURI"/>
```

Business Card Schema (2/3)

```
<complexType name="cardlist_type">
   <sequence>
     <element name="title" type="b:cardlist_title_type"/>
     <element ref="b:card" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <complexType name="cardlist_title_type" mixed="true">
   <sequence>
     <any namespace="http://www.w3.org/1999/xhtml"
           minOccurs="0" maxOccurs="unbounded"
           processContents="lax"/>
   </sequence>
 </complexType>
```

Business Card Schema (3/3)

```
  <complexType name="card_type">
    <sequence>
      <element ref="b:name"/>
      <element name="title" type="string"/>
      <element ref="b:email"/>
      <element ref="b:phone" minOccurs="0"/>
      <element ref="b:logo" minOccurs="0"/>
    </sequence>
  </complexType>

  <complexType name="logo_type">
    <attribute ref="b:uri" use="required"/>
  </complexType>
</schema>
```

The `org.businesscard` Package

- The binding compiler generates a number of classes and interfaces:
  - `Cardlist, CardlistType, CardlistImpl, CardlistTypeImpl`
  - `Card, CardType, CardImpl, CardTypeImpl`
  - `...`
  - `Logo, LogoType`
  - `LogoImpl, LogoTypeImpl`
  - `ObjectFactory`

The CardType Interface

```
public interface CardType {
    java.lang.String getEmail();
    void setEmail(java.lang.String value);
    org.businesscard.LogoType getLogo();
    void setLogo(org.businesscard.LogoType value);
    java.lang.String getTitle();
    void setTitle(java.lang.String value);
    java.lang.String getName();
    void setName(java.lang.String value);
    java.lang.String getPhone();
    void setPhone(java.lang.String value);
}
```

A Little Bit of Code

```
void addCards() {
   cardpanel.removeAll();
   Iterator i = cardlist.iterator();
   int j = 0;
   while (i.hasNext()) {
     Card c = (Card)i.next();
     Button b = new Button(c.getName());
     b.setActionCommand(String.valueOf(j++));
     b.addActionListener(this);
     cardpanel.add(b);
   }
   this.pack();
 }
```

The Main Application

```
public BCedit(String cardfile) {
   super("BCedit");
   this.cardfile=cardfile;
   try {
     jc = JAXBContext.newInstance("org.businesscard");
     Unmarshaller u = jc.createUnmarshaller();
     cl = (Cardlist)u.unmarshal(
             new FileInputStream(cardfile)
          );
   } catch (Exception e) { e.printStackTrace(); }
   // initialize the user interface
   ...
}
```

Streaming XML

- JDOM and JAXB keeps the entire XML tree in memory
- Huge documents can only be streamed:
  - movies on the Internet
  - Unix file commands using pipes
- What is streaming for XML documents?

- The SAX framework has the answer...

Parsing Events

- View the XML document as a stream of events:
    - the document starts
    - a start tag is encountered
    - an end tag is encountered
    - a namespace declaration is seen
    - some whitespace is seen
    - character data is encountered
    - the document ends
- The SAX tool observes these events
- It reacts by calling corresponding methods specified by the programmer

Tracing All Events (1/4)

```
public class Trace extends DefaultHandler {
  int indent = 0;

  void printIndent() {
    for (int i=0; i<indent; i++) System.out.print("-");
  }

  public void startDocument() {
    System.out.println("start document");
  }

  public void endDocument() {
    System.out.println("end document");
  }
```

Tracing All Events (2/4)

```
public void startElement(String uri, String localName,
                         String qName, Attributes atts) {
  printIndent();
  System.out.println("start element: " + qName);
  indent++;
}

public void endElement(String uri, String localName,
                       String qName) {
  indent--;
  printIndent();
  System.out.println("end element: " + qName);
}
```

Tracing All Events (3/4)

```
public void ignorableWhitespace(char[] ch, int start, int length)
 {
 printIndent();
  System.out.println("whitespace, length " + length);
}

public void processingInstruction(String target, String data) {
  printIndent();
  System.out.println("processing instruction: " + target);
}

public void characters(char[] ch, int start, int length){
  printIndent();
  System.out.println("character data, length " + length);
}
```

Tracing All Events (4/4)

```
 public static void main(String[] args) {
    try {
      Trace tracer = new Trace();
      XMLReader reader = XMLReaderFactory.createXMLReader();
      reader.setContentHandler(tracer);
      reader.parse(args[0]);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Output for the Recipe Collection

```
start document
start element: rcp:collection
-character data, length 3
-start element: rcp:description
--character data, length 44
-end element: rcp:description
-character data, length 3
-start element: rcp:recipe
--character data, length 5
--start element: rcp:title
---character data, length 42
...
--start element: rcp:nutrition
--end element: rcp:nutrition
--character data, length 3
-end element: rcp:recipe
-character data, length 1
end element: rcp:collection
end document
```

A Simple Streaming Example (1/2)

```
public class Height extends DefaultHandler {
  int h = -1;
  int max = 0;

  public void startElement(String uri, String localName,
                           String qName, Attributes atts) {
    h++; if (h > max) max = h;
  }

  public void endElement(String uri, String localName,
                         String qName) {
    h--;
  }

  public void characters(char[] ch, int start, int length){
    if (h+1 > max) max = h+1;
  }
```

A Simple Streaming Example (2/2)

```
  public static void main(String[] args) {
    try {
      Height handler = new Height();
      XMLReader reader = XMLReaderFactory.createXMLReader();
      reader.setContentHandler(handler);
      reader.parse(args[0]);
      System.out.println(handler.max);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Comments on the Example

- This version is less intuitive (stack-like style)
- The JDOM version:

  `java.lang.OutOfMemoryError`

  on 18MB document
- The SAX version handles 1.2GB in 51 seconds

SAX May Emulate JDOM (1/2)

```
public void startElement(String uri, String localName,
                           String qName, Attributes atts) {
  if (localName.equals("card")) card = new Element("card",b);
  else if (localName.equals("name"))
    field = new Element("name",b);
  else if (localName.equals("title"))
    field = new Element("title",b);
  else if (localName.equals("email"))
    field = new Element("email",b);
  else if (localName.equals("phone"))
    field = new Element("phone",b);
  else if (localName.equals("logo")) {
    field = new Element("logo",b);
    field.setAttribute("uri",atts.getValue("","uri"));
    }
  }
```

SAX May Emulate JDOM (2/2)

```
  public void endElement(String uri, String localName,
                         String qName) {
    if (localName.equals("card")) contents.add(card);
    else if (localName.equals("cardlist")) {
      Element cardlist = new Element("cardlist",b);
      cardlist.setContent(contents);
      doc = new Document(cardlist);
    } else {
      card.addContent(field);
      field = null;
    }
  }

  public void characters(char[] ch, int start, int length) {
    if (field!=null)
      field.addContent(new String(ch,start,length));
}
```

SAX vs. DOM

SAX

- Java-specific
- interprets XML as a stream of events
- you supply event-handling callbacks
- SAX parser invokes your event-handlers as it parses
- doesn't build data model in memory
- serial access
- very fast, lightweight
- good choice when
  - no data model is needed, or
  - natural structure for data model is list, matrix, etc.

DOM

- W3C standard for representing structured documents
- platform and language neutral (not Java-specific!)
- interprets XML as a tree of nodes
- builds data model in memory
- enables random access to data
- good for interactive apps
- more CPU- and memory-intensive
- good choice when
  - data model has natural tree structure

Essential Online Resources

- http://www.jdom.org/
- http://java.sun.com/xml/jaxp/
- http://java.sun.com/xml/jaxb/
- http://www.saxproject.org/