XML Parsing

José M. Vidal

Thu Mar 18 14:10:36 EST 2004

This talk provides a quick overview of XML parsing using Java

- IBM developerWorks, Validating XML[1] (PDF)
- SAX Quickstart[2]
- IBM developerWorks Understanding DOM[3] (PDF)
- Elliot Rusty Harold, Processing XML with Java[4], 2002. *(Optional)*

1 Parsing

- **parse**[5] vt. To determine the syntactic structure of a sentence or other utterance (close to the standard English meaning). "That was the one I saw you." "I can't parse that.". *Jargon File*
- That is, how to turn an XML document into a data structure you can use.
- XML was designed to be easy to parse, so many XML parsers exist. It is unlikely that your favorite language will not already have one.
- All parsers implement one of the two well-known API's: SAX and DOM.

2 Simple API for XML

- The SAX API (Javadoc)[6] is defined independent of any implementation.
- A "de facto" standard.
- It is simple, quick, and can handle documents of any size.
- You use it by defining a number of methods which will get called back when the parser reaches the appropriate place in the file.
- The available methods include `startElement`, `startDocument`, etc.

2.1 SAX Sample Application

```
import java.io.FileReader;

import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;
```

```
/** Our handler must extend DefaultHandler */
public class MySAXApp extends DefaultHandler
{

  public static void main (String args[])
    throws Exception
  {
    //Create an instance of the XML parser.
    XMLReader xr = XMLReaderFactory.createXMLReader();

    //Create an instance of our handles (see below)
    MySAXApp handler = new MySAXApp();
    xr.setContentHandler(handler);
    xr.setErrorHandler(handler); //does double duty.

    // Parse each *file* provided on the
    // command line.
    for (int i = 0; i < args.length; i++) {
      FileReader r = new FileReader(args[i]);
      xr.parse(new InputSource(r));
    }
  }

  public MySAXApp ()
  {
    super();
  }


  ////////////////////////////////////////////////////////////////////
  // Event handlers.
  ////////////////////////////////////////////////////////////////////


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


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

  /**
   * Receive notification at the beginning of an element.
   *
   * @param uri The Namespace URI, or the empty string if the element
     has no Namespace URI or if Namespace processing is not being
     performed.
   * @param name The local name (without prefix), or the empty string
     if Namespace processing is not being performed.
   * @param qName The qualified name (with prefix), or the empty
```

```
    string if qualified names are not available.[namespaceprefix:name]
   * @param atts The attributes attached to the element. If there are
    no attributes, it shall be an empty Attributes object.
   */
  public void startElement (String uri, String name,
                            String qName, Attributes atts)
  {
    if ("".equals (uri))
      System.out.println("Start element: " + qName);
    else
      System.out.println("Start element: {" + uri + "}" + name);
  }


  public void endElement (String uri, String name, String qName)
  {
    if ("".equals (uri))
      System.out.println("End element: " + qName);
    else
      System.out.println("End element:   {" + uri + "}" + name);
  }

  /**
   * Receive notification of character data.
   *
   * @param ch[] The characters from the XML document.
   * @param start The start position in the array.
   * @param length The number of characters to read from the array
   */
  public void characters (char ch[], int start, int length)
  {
    System.out.print("Characters:    \"");
    for (int i = start; i < start + length; i++) {
      switch (ch[i]) {
      case '
':
        System.out.print("

");
        break;
      case '"':
        System.out.print("
\"");
        break;
      case '\n':
        System.out.print("
textbackslashn");
        break;
      case '\r':
        System.out.print("
textbackslashr");
        break;
      case '\t':
        System.out.print("
```

```
textbackslasht");
        break;
      default:
        System.out.print(ch[i]);
        break;
      }
    }
    System.out.print("\"\n");
  }

}
```

2.2 Application Output

- Given the following XML file as input

```
<?xml version="1.0"?>

<poem xmlns="http://www.megginson.com/ns/exp/poetry">
  <title>Roses are Red</title>
  <l>Roses are red,</l>
  <l>Violets are blue;</l>
  <l>Sugar is sweet,</l>
  <l>And I love you.</l>
</poem>
```

- Using the command-line argument (replace `com.example.xml` with the implementation you are using):
- `java -Dorg.xml.sax.driver=com.example.xml.SAXDriver MySAXApp roses.xml`
- It will print out:

```
Start document
Start element: {http://www.megginson.com/ns/exp/poetry}poem
Characters:    "\n"
Start element: {http://www.megginson.com/ns/exp/poetry}title
Characters:    "Roses are Red"
End element:   {http://www.megginson.com/ns/exp/poetry}title
Characters:    "\n"
Start element: {http://www.megginson.com/ns/exp/poetry}l
Characters:    "Roses are red,"
End element:   {http://www.megginson.com/ns/exp/poetry}l
Characters:    "\n"
Start element: {http://www.megginson.com/ns/exp/poetry}l
Characters:    "Violets are blue;"
End element:   {http://www.megginson.com/ns/exp/poetry}l
Characters:    "\n"
Start element: {http://www.megginson.com/ns/exp/poetry}l
Characters:    "Sugar is sweet,"
End element:   {http://www.megginson.com/ns/exp/poetry}l
Characters:    "\n"
```

```
Start element: {http://www.megginson.com/ns/exp/poetry}l
Characters:    "And I love you."
End element:   {http://www.megginson.com/ns/exp/poetry}l
Characters:    "\n"
End element:   {http://www.megginson.com/ns/exp/poetry}poem
End document
```

3 Document Object Model

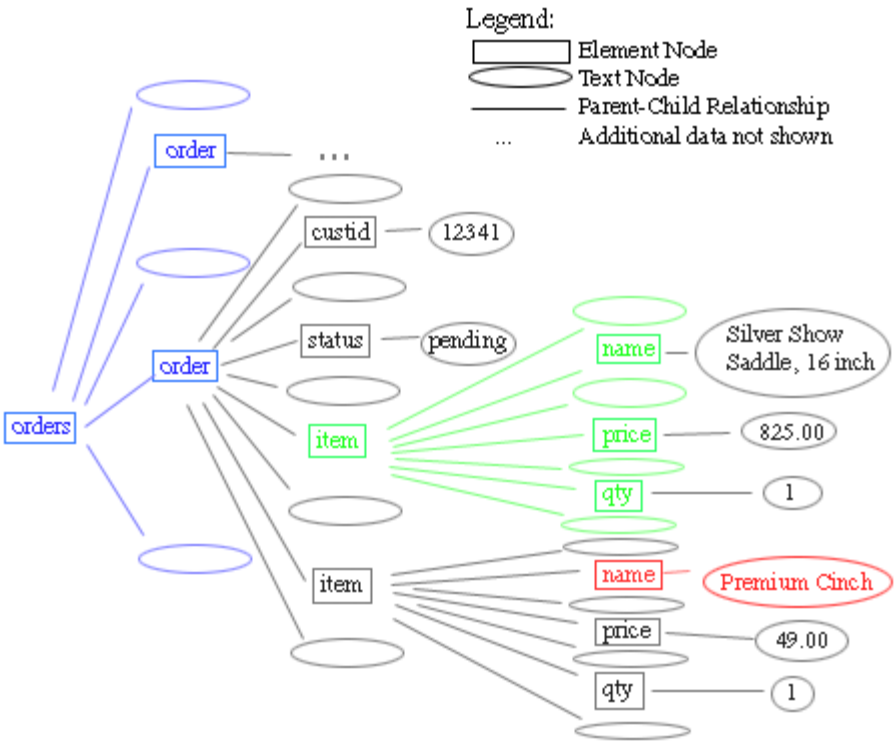
- DOM[7] is a way to describe the XML elements and the relationships between them.
- The DOM recommendation is currently at Level 2[8], with Level 3 a proposed recommendation.
- DOM allows programmers to access and change the structure of a document (XML, HTML).
- DOM XML parsers usually must read the whole XML document into memory, creating a data structure that you can access.

3.1 DOM Structure

- A DOM `Document` is a hierarchy (tree) of nodes
- The following XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ORDERS SYSTEM "orders.dtd">
<orders>
 <order>
  <customerid limit="1000">12341</customerid>
  <status>pending</status>
  <item instock="Y" itemid="SA15">
   <name>Silver Show Saddle, 16 inch</name>
   <price>825.00</price>
   <qty>1</qty>
  </item>
  <item instock="N" itemid="C49">
   <name>Premium Cinch</name>
   <price>49.00</price>
   <qty>1</qty>
  </item>
 </order>
 <order>
  <customerid limit="150">251222</customerid>
  <status>pending</status>
  <item instock="Y" itemid="WB78">
   <name>Winter Blanket (78 inch)</name>
   <price>20</price>
   <qty>10</qty>
  </item>
 </order>
</orders>
```

- Is represented by a tree like:



- Notice all the text nodes. They are there because there is whitespace in between all tags.

3.2 Node Types

- **Elements**: Usually contain other elements.
- **Attributes**
- **Text**
- **Document**: The root node.
- **Comment**
- **Processing Instructions**: start with <?

3.3 DOM Parser

- Our goal is to get a `Document` object that represents our XML file. Since `Document` is an interface we must use a factory.

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import org.w3c.dom.Document;

public class OrderProcessor {
  public static void main (String args[]) {
    File docFile = new File("orders.xml");
```

```
    Document doc = null;
    try {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();

      //A document builder builds specific documents.
      DocumentBuilder db = dbf.newDocumentBuilder();

      //This call parses the file and creates the Document in memory.
      doc = db.parse(docFile);
    } catch (Exception e) {
      System.out.print("Problem parsing the file.");
    }
  }
}
```

3.4 Traversing the Document

- In order to traverse the Document we first need to get the root element.
- We do this with a `doc.getDocumentElement()` call.

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import org.w3c.dom.*;

public class OrderProcessor {
  public static void main (String args[]) {

    //some code here

    doc = db.parse(docFile);

    //STEP 1:  Get the root element
    Element root = doc.getDocumentElement();
    System.out.println("The root element is "+root.getNodeName());

    //STEP 2:  Get the children
    NodeList children = root.getChildNodes();
    System.out.println("There are "+children.getLength()
                           +" nodes in this document.");

    //STEP 3:  Step through the children
    for (Node child = root.getFirstChild();
         child != null;
         child = child.getNextSibling())
    {
      //The node name is either the tag name or #text
      System.out.println(child.getNodeName()+" = "+child.getNodeValue());
    }

    //STEP 4:  Recurse this functionality
    stepThrough(root);
```

```
  }

  private static void stepThrough (Node start)
  {
    System.out.println(start.getNodeName()+" = "+start.getNodeValue());

    //if its a node, then print the attributes.
    if (start.getNodeType() == start.ELEMENT_NODE)
    {
      NamedNodeMap startAttr = start.getAttributes();
      for (int i = 0;
           i < startAttr.getLength();
           i++) {
        Node attr = startAttr.item(i);
        System.out.println("  Attribute:  "+ attr.getNodeName()
                           +" = "+attr.getNodeValue());
      }
    }


    for (Node child = start.getFirstChild();
         child != null;
         child = child.getNextSibling())
    {
      stepThrough(child);
    }
  }

}
```

3.5 Modifying a Document

- With DOM we can change the `Document` in any way we want.

```
public class OrderProcessor {

  /** Change the value of node elemName to elemValue */
  private static void changeOrder (Node start,
                                   String elemName,
                                   String elemValue)
  {
    if (start.getNodeName().equals(elemName)) {
      start.getFirstChild().setNodeValue(elemValue);
    }

    for (Node child = start.getFirstChild();
         child != null;
         child = child.getNextSibling())
    {
      changeOrder(child, elemName, elemValue);
    }
  }
```

```
public static void main (String args[]) {

  // Change text value of node named status to processing.
  changeOrder(root, "status", "processing");

  //Get a list of nodes that have a status element.
  NodeList orders = root.getElementsByTagName("status");

  for (int orderNum = 0;
       orderNum < orders.getLength();
       orderNum++)
  {
    System.out.println(orders.item(groupNum).getFirstChild().getNodeValue());

    Element thisOrder = (Element)orders.item(orderNum);

    //Remove the limit attribute from customer.
    Element customer = (Element)thisOrder.getElementsByTagName("cusomertid").item(0);
    customer.removeAttribute("limit");

    NodeList orderItems = thisOrder.getElementsByTagName("item");
    double total = 0;
    for (int itemNum = 0;
         itemNum < orderItems.getLength();
         itemNum++) {

      // Total up cost for each item and
      // add to the order total

      //Get this item as an Element
      Element thisOrderItem = (Element)orderItems.item(itemNum);

      //Remove a node.
      //Remove anything with <item instock="N">
      if (thisOrderItem.getAttributeNode("instock").getNodeValue().equals("N")) {
        Node deadNode = thisOrderItem.getParentNode().removeChild(thisOrderItem);

        continue;

        //Alternatively, we could have replaced this item with a backorderd element.
        // <item itemid="123">
        //   <backordered></backordered>
        // </item>
        Element backElement = doc.createElement("backordered");

        //<backordered itemid="">
        backElement.setAttributeNode(doc.createAttribute("itemid"));

        //<backordered itemid="123">
        String itemIdString = thisOrderItem.getAttributeNode("itemid").getNodeValue();
        backElement.setAttribute("itemid", itemIdString);

        Node deadNode = thisOrderItem.getParentNode()
```

```
            .replaceChild(backElement, thisOrderItem);
        }

        //Get pricing information for this Item
        String thisPrice = thisOrderItem.getElementsByTagName("price").item(0)
          .getFirstChild().getNodeValue();
        double thisPriceDbl = new Double(thisPrice).doubleValue();

        //Get quantity information for this Item
        String thisQty = thisOrderItem.getElementsByTagName("qty").item(0)
          .getFirstChild().getNodeValue();
        double thisQtyDbl = new Double(thisQty).doubleValue();

        double thisItemTotal = thisPriceDbl*thisQtyDbl;
        total = total + thisItemTotal;
      }
      String totalString = new Double(total).toString();

      //1234.34
      Node totalNode = doc.createTextNode(totalString);

      //<total></total>
      Element totalElement = doc.createElement("total");

      //<total>1234.34</total>
      totalElement.appendChild(totalNode);

      //Add that element before anyone else.
      thisOrder.insertBefore(totalElement, thisOrder.getFirstChild());

    }
  }
}
```

3.6 Outputting a Document

- Going from a a `Document` to its XML representation only requires calling `toString()` on it.
- A **normalized** XML document has no extraneous whitespace.
- In order to normalize your document call `normalize()` on it.
- To be complete your document will need the <?xml....> and <!DOCTYPE...> lines, these can be added:

```
try
{
  File newFile = new File("processedOrders.xml");
  FileWriter newFileStream = new FileWriter(newFile);
  newFileStream.write("<?xml version=\"1.0\"?>");
  newFileStream.write("<!DOCTYPE "+doc.getDoctype().getName()+" ");
  if (doc.getDoctype().getSystemId() != null)
  {
    newFileStream.write(" SYSTEM ");
```

```
    newFileStream.write(doc.getDoctype().getSystemId());
  }
  if (doc.getDoctype().getPublicId() != null)
  {
    newFileStream.write(" PUBLIC ");
    newFileStream.write(doc.getDoctype().getPublicId());
  }
  newFileStream.write(">");

  newFileStream.write(newRoot.toString());

  newFileStream.close();

} catch (IOException e) {
  System.out.println("Can't write new file.");
}
```

4 Validating

- DTDs and XML Schema describe the valid tags, attributes, data types, etc. that can be present in a given XML document.
- **Validation** is the process ascertaining whether a given document obeys its DTDs or Schemas.
- Is should not be confused with "**well-formed** XML document": one that is legal XML.
- Valid documents are always well-formed, but well-formed documents may not be valid.
- Validation is not required when working with XML data. In fact, it is often omitted altogether (although, that is probably a bad idea).

4.1 Doctype

- The DOCTYPE tells the validating parser where to find the DTD or schema it should use for that document.
- `<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">`
- The DOCTYPE declaration consists of several parts
  - `<!DOCTYPE` Indicates to the processor that this is a `DOCTYPE` declaration.
  - `html` Indicates the name of the root element for the document.
  - `PUBLIC` A DOCTYPE can designate a publicly recognized DTD, potentially saving the processor a trip to the server to retrieve it.
  - `"-//W3C//DTD HTML 4.01 Transitional//EN"`: The actual public identifier for the Transitional XHTML DTD.
- For custom DTDs, developers typically use a SYSTEM identifier, such as:
- `<!DOCTYPE memories SYSTEM "/home/jmvidal/memories.dtd">`

4.2 Xerces Validation

- To validate we simply run the parser with validation turned on. This is true for most parsers.

1. Create a parser.
2. Turn on validation.
3. Set the error handler.
4. Parse the document.

4.3 Error Handler

- We catch the errors by implementing an error handler and passing it to the parser.
- The error handler must extend `DefaultHandler`.

```
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXParseException;

/** A simple error handlier which just prints out
    the errors. */
public class ErrorChecker extends DefaultHandler
{

  public ErrorChecker() {
  }

  public void error (SAXParseException e) {
    System.out.println("Parsing error:  "+e.getMessage());
  }

  public void warning (SAXParseException e) {
    System.out.println("Parsing problem:  "+e.getMessage());
  }

  public void fatalError (SAXParseException e) {
    System.out.println("Parsing error:  "+e.getMessage());
    System.out.println("Cannot continue.");
    System.exit(1);
  }
}
```

4.4 Xerces Validator

- The code to run the validating parser looks like:

```
import org.apache.xerces.parsers.DOMParser;
import java.io.File;
import org.w3c.dom.Document;

public class SchemaTest {
  public static void main (String args[]) {
```

```
    File docFile = new File("memory.xml");

    try {

      DOMParser parser = new DOMParser();
      parser.setFeature("http://xml.org/sax/features/validation", true);

      //Here we specificy the schema location,
      //but we could have used the ones specified in the document.
      parser.setProperty(
                "http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation",
                    "memory.xsd");

      ErrorChecker errors = new ErrorChecker();
      parser.setErrorHandler(errors);

      parser.parse("memory.xml");
    } catch (Exception e) {
      System.out.print("Problem parsing the file.");
    }
  }
}
```

Notes

[1] http://www-106.ibm.com/developerworks/xml/edu/x-dw-xvalid-i.html
[2] http://www.saxproject.org/?selected=quickstart
[3] http://www-106.ibm.com/developerworks/xml/edu/x-dw-xudom-i.html
[4] http://www.ibiblio.org/xml/books/xmljava/
[5] http://jargon.watson-net.com/jargon.asp?w=parse
[6] http://www.saxproject.org/apidoc/overview-summary.html
[7] http://www.w3c.org/DOM/
[8] http://www.w3c.org/DOM/DOMTR#dom2

This talk is available at `http://jmvidal.cse.sc.edu/talks/xmlparsing`