

XML

XML -- Hype and Reality

Extensible Markup Language

Just a textual data format standard.

XML -- portable data

Java -- portable code

XML is just a way of describing a bunch of bindings in a textual form.

It's a simple thing, but by being standard, it makes many boring incompatibilities go away -- creating real value.

Trying to make things work together more easily -- a data exchange format.

XML helps with the basic problem of structure and parsing. To be compatible, two applications still need to agree on the **meaning** of the data, but at least the parsing is taken care of.

DTD -- meta description of a class of XML documents.

Formal description of the allowed structure for a class of XML documents

The parser or other tool can formally check that a document meets the DTD structure definition. In theory, just regular code does not need to worry about structure errors -- it's handled by the parser/DTD system.

<http://www.xml.org/>

<http://www.w3.org/XML>

<http://java.sun.com/xml>

XML Tags

Tags -- meta content in the text. Like HTML tags

here is some text <red>with this</red> marked as red

Tags are case sensitive <foo> and <FOO> may be treated differently

Between the tags there may be raw text and/or more tags

Tags with nothing in between them, <tag></tag>, can be written as <tag />

Tag attributes

Stores a binding inside of a tag

May use single quote (') or double quote (")

<dot x="72" y="13" />

<node foo="bar" pi='3.14'>

Special Characters

A few characters have meta-meaning in XML, and so must always be encoded.

Note that the encodings end with a semi-colon (;)

< encode as: <

> encode as: >

& encode as: &

" encode as: "

' encode as: '

1. XML Text Strategy

XML can be used like HTML -- large blocks of text with tags sprinkled around to mark sections of text.

```
<foo>And here is some <b>text</b></foo>
```

The resulting structure is somewhat free-form

2. XML Tree Strategy

Tree shaped

Can write XML without free text between tags, except the innermost (leaf) tags. In that case, the structure is like a tree.

e.g.

```
<person>
  <name>Hans Gruber</name>
  <id>123456</id>
  <username>hans</username>
</person>
```

In my experience, this is the most common use of XML in programming projects -- save out the internal state as an XML tree

Tags vs. Attributes

Suppose we want to have a "dot" that stores an x and a y

How should each x,y be stored in its <dot> ...

1. Attribute Method

```
<dot x="27" y="13">
```

2. Tag Method

```
<dot>
  <x>27</x>
  <y>13</y>
</dot>
```

Tags vs. Attributes style

There is **not** wide agreement about exactly when to use tags vs. attributes.

I prefer the "attribute" way where possible, since it seems simpler. It works best when the number of children is fixed, and where the data itself is short.

```
<dot x='6' y='13' />
```

If a node can have an arbitrary number of children, then tags are the best way

```
<parent> <child>..</child> <child>..</child> <child>..</child>
</parent>
```

The tag method is also appropriate if the data is lengthy...

```
<description>How did our constructed suburban landscape
come to be so unpleasant, and what to do about it.
The Geography of Nowhere is a landmark work in growth
of the New Urbanism movement.</description>
```

Dots XML Example

The "Dots" XML format -- a set of (x,y) points

Root node : "dots" -- parent of dot nodes

Child nodes : "dot" -- each with "x" and "y" attributes

```
<?xml version="1.0" encoding="UTF-8"?>
<dots>
  <dot x="72" y="101" />
  <dot x="170" y="164" />
  <dot x="184" y="158" />
  <dot x="194" y="146" />
  <dot x="191" y="133" />
  <dot x="164" y="84" />
  <dot x="119" y="89" />
</dots>
```

Java XML

XML in Java

<http://java.sun.com/xml/>

SAX

The SAX interface will show you, one at a time, the nodes of the XML document in a single, linear pass of the document.. It does not build the whole tree in memory, so it's faster.

DOM

Tree of nodes -- slower

Can iterate over the tree to look at the nodes, traverse siblings, etc.

Can edit the tree: add/remove nodes

Jar Files

jaxp.jar and crimson.jar -- add XML features of Java 1.3 and earlier

XML is part of the default install with Java 1.4

SAX Strategy

Subclass off DefaultHandler, and implement the "notification" methods.

startElement()

endElement()

characters() -- char data between tags

Notifications are sent to mark the start of a node, end of a node, characters, etc.

Take action at the time of the notification, or...

Store state in ivars to be used in a later notification -- e.g. the "flip" boolean

Simplest to assume that that XML document is correct and well formed

SAX Dot Example

```
// XMLDotReader.java

import java.io.*;
import java.util.*;

import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;

/*
This is a simple class that can read state out of an XML file
using a SAX state-machine parser.

In this case, we support data as below --
simple <dot "1" y="2"> nodes, possibly nested
inside a <flip> tag which swaps x and y.

<?xml version="1.0" encoding="UTF-8"?>

<dots>
  <dot x="1" y="2" />
  <dot x="3" y="4" />
  <flip>
    <dot x="6" y="5" />
    <dot x="8" y="7" />
  </flip>
  <dot x="9" y="10" />
</dots>

*/

public class XMLDotReader extends DefaultHandler
{
    // State we keep track of -- like a state machine,
    // where startElement() etc. keep getting called
    private int x;
    private int y;
    private boolean flip;

    public XMLDotReader() {
        clear();
    }

    public void clear() {
        x = -1;
        y = -1;
        flip = false;
    }

    /*
SAX notifications -- sent to the parser to signal elements
of the document as they are found in a linear top-down, left-right
*/
}
```

```

    reading of the document.
*/

public void startDocument ()
    throws SAXException
{
    //System.out.println("startDocument");
}

public void endDocument ()
    throws SAXException
{
    //System.out.println("startDocument");
}

/*
    Called to mark a starting <tag>
    -Look at the tag name+attributes -- decide to process this node,
      or set state to be used in later calls to startElement(), characters(), etc.
    -The qName (qualified name) is the full name of the tag
    -For <tag x="1" y="2">, use atts.getValue("x") to retrieve value
      (null if no such value)
*/
public void startElement (String namespaceURI, String localName,
    String qName, Attributes atts)
    throws SAXException
{
    //System.out.println("start element:" + qName);
    if (qName.equals("dot")) {
        x = Integer.parseInt(atts.getValue("x"));
        y = Integer.parseInt(atts.getValue("y"));

        if (flip) {
            int temp = x; x = y; y = temp;
        }

        // do something with our x,y state (could wait for endElement)
        System.out.println(x + ", " + y);
    }

    /*
        In this case, the x/y values are used immediately, but in a more
        complex application, they would be stored in the ivars here for use
        in a later call, as the 'flip' boolean demonstrates.
    */

    else if (qName.equals("flip")) {
        flip = true;
    }
}

```

```

/*
    Called at the </tag>
*/
public void endElement(java.lang.String uri,
                      java.lang.String localName,
                      java.lang.String qName)
    throws SAXException
{
    //System.out.println("end element:" + qName);

    // </flip> -> flip mode off
    // (or could do something more complex to
    // get the right answer for nested flips)
    if (qName.equals("flip")) {
        flip = false;
    }
}

/*
    Called for characters between nodes -- the chars start
    at the given offset in the array.
    Use trim() to get a string with whitespace removed from its ends.
*/
public void characters (char buf [], int offset, int len)
    throws SAXException
{
    //String s = new String(buf, offset, len);
    //s = s.trim();
    //if (!s.equals("")) {
    //    System.out.println("characters:" + s);
    //}
}

/**
    Read the XML in the given input stream
*/
public void read(InputStream stream) {
    try {
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser saxParser = factory.newSAXParser();
        clear();
        saxParser.parse(stream, this);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

```
/*
Process the xml file from the command line.
*/
public static void main (String argv []) {
    if (argv.length != 1) {
        System.err.println ("Usage: cmd filename");
        System.exit (1);
    }

    try {
        XMLDotReader xr = new XMLDotReader();
        InputStream in = new BufferedInputStream(
            new FileInputStream(new File(argv[0])));
        xr.read(in);
    } catch (Exception e) {
        e.printStackTrace ();
    }
}
}
```