Lecture 0 (2 hours)

DOM

(Document Object Model)

Sang Shin
Java™ Technology Evangelist
sang.shin@sun.com

Topics

- Features and Characteristics
- DOM node tree and node types
- Java interfaces
- DOM Programming
  - Traversing DOM
  - Manipulating DOM
  - Creating a new DOM
  - Writing out (Serializing) DOM

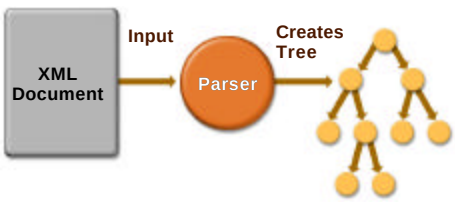
Historical Background

- DOM is a standard defined by the W3C, just like XML
- DOM was not designed specifically for Java technology (unlike SAX)
- DOM is cross-platform and cross-language
  - Uses OMG's IDL to define interfaces
  - IDL to language binding

DOM Characteristics

- Access XML document as a tree structure
- Composed of mostly element nodes and text nodes
- Can "walk" the tree back and forth
- Larger memory requirements
  - Fairly heavyweight to load and store
- Use it when for walking and modifying the tree

DOM in Action



DOM Tree and Nodes

- XML document is represented as a tree
- A tree is made of nodes
- There are 12 different node types
- Nodes may contain other nodes (depending on node types)
  - parent node contains child nodes

Node Types

- Document node
- Document Fragment node
- Element node
- Attribute node
- Text node
- Comment node
- Processing instruction node
- Document type node
- Entity node
- Entity reference node
- CDATA section node
- Notation node

DOM Tree Hierarchy

- A document node contains
  - one element node (root element node)
  - one or more processing instruction nodes
- An element node may contain
  - other element nodes
  - one or more text nodes
  - one or more attribute nodes
- An attribute node contain
  - a text node

Example XML Document

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

 <person born="1912">
  <name>
   <first_name>Alan</first_name>
   <last_name>Turing</last_name>
  </name>
  <profession>computer scientist</profession>
 </person>

</people>
```

DOM Tree Example

- XML Document node
  - element node “people”
    - element node “person”
      - element node “name”
        - » element node “first_name”
          - * text node “Alan”
        - » element node “last_name”
          - * text node “Turing”
      - element node “profession”
        - » text node “computer scientist”
      - attribute node “born”
        - » text node “1912”

DOM and Java Programming

- How do you represent each node type in Java programs?
  - Java interface type
- How do you encapsulate common characteristics of among node types?
  - Java interface hierarchy

Java Interface Hierarchy

- Node interface (super interface of)
  - Document interface
  - DocumentFragment interface
  - DocumentType interface
  - ProcessingInstruction interface
  - CharacterData interface
    - Comment
    - Text
      - CDATASection
  - Element interface
  - Attr interface
  - EntityReference interface
  - Entity interface
  - Notation interface

Other Inferfaces for DOM

- NodeList
- NamedNodeMap
- DOMImplementation

Node Interface

- Primary data type in DOM
- Represents a single node in a DOM tree
- Every node is Node interface type
  - Since every node is a Node interface type, every node can be processed in the same way (polymorphism)
- Specialized interfaces contain additional or more convenient methods

Methods in Node Interface

- Useful Node interface methods
  - public short getNodeType()
  - public String getNodeName()
  - public String getNodeValue()
  - public NamedNodeMap getAttributes();
  - public NodeList getChildNodes()
- Not all methods would make sense to all node types, however
  - GetNodeValue() for Element node
  - GetAttributes() for Comment node

Node Interface - Node Types

```
public interface Node {

  // NodeTypes
  public static final short ELEMENT_NODE              = 1;
  public static final short ATTRIBUTE_NODE            = 2;
  public static final short TEXT_NODE                 = 3;
  public static final short CDATA_SECTION_NODE        = 4;
  public static final short ENTITY_REFERENCE_NODE     = 5;
  public static final short ENTITY_NODE               = 6;
  public static final short PROCESSING_INSTRUCTION_NODE = 7;
  public static final short COMMENT_NODE              = 8;
  public static final short DOCUMENT_NODE             = 9;
  public static final short DOCUMENT_TYPE_NODE        = 10;
  public static final short DOCUMENT_FRAGMENT_NODE    = 11;
  public static final short NOTATION_NODE             = 12;
```

Node Interface

```
public String      getNodeName();
public String      getNodeValue() throws DOMException;
public void        setNodeValue(String nodeValue) throws DOMException;
public short       getNodeType();
public Node        getParentNode();
public NodeList    getChildNodes();
public Node        getFirstChild();
public Node        getLastChild();
public Node        getPreviousSibling();
public Node        getNextSibling();
public NamedNodeMap getAttributes();
public Document    getOwnerDocument();
```

Node Interface

```
public Node        insertBefore(Node newChild, Node refChild)
                                throws DOMException;
public Node        replaceChild(Node newChild, Node oldChild)
                                throws DOMException;
public Node        removeChild(Node oldChild) throws DOMException;
public Node        appendChild(Node newChild) throws DOMException;

public boolean     hasChildNodes();
public Node        cloneNode(boolean deep);
public void        normalize();
public boolean     supports(String feature, String version);
public String      getNamespaceURI();
public String      getPrefix();
public void        setPrefix(String prefix) throws DOMException;
public String      getLocalName();
}
```

NodeList Interface

- Represents a collection of nodes
- Return type of *getChildNodes*() method of *Node* interface

```
public interface NodeList {
    public Node item(int index);
    public int  getLength();
}
```

NamedNodeMap Interface

- Represents a collection of nodes each of which can identified by name
- Return type of *getAttributes*() method of Node interface

NamedNodeMap Interface

```
Public interface NameNodeMap{
  public Node getNamedItem (String name);
  public Node setNamedItem(Node arg) throws DOMException;
  public Node removeNamedItem(String name)
                                        throws DOMException;
  public Node item(int index);
  public int getLength();
  public Node getNamedItemNS(String namespaceURI,
                String localName);
  public Node setNamedItemNS(Node arg) throws DOMException;
  public Node removeNamedItemNS(String namespaceURI,
                String localName) throws DOMException;
}
```

Document Node

- Root node
- Represents entire document
- Child node types
  - One Element node
  - Optional document type node
  - Processing instruction nodes
  - Comment nodes

Document Interface

- Contains factory methods for creating other nodes
  - elements, text nodes, comments, processing instructions, etc
- Method to get root element node

Document Interface

```
public interface Document extends Node {
  Attr createAttrbute (String name)
  Attr createAttributeNS(String namespaceURI , String qName)
  CDATASection createCDATASection (String data)
  Comment createComment(String data)
  DocumentFragment createDocumentFragment ()
  Element createElement(String tagName)
  Element createElementNS(String namespaceURI , String qName)
  EntityReference createEntityReference(String name)
  ProcessingInstruction createProcessingInstruction (String target, String data)
  Text createTextNode(String data)
  DocumentType getDocType()
  Element getDocumentElement ()
  Element getElementById(String elementId )
  NodeList getElementsByTagName(String tagName)
  NodeList getElementsByTagNameNS(String namespaceURI , String localName )
  DOMImplementation getImplementation()
  Node importNode (Node importNode , boolean deep)
}
```

Example

```
case Node.DOCUMENT_NODE:
  System.out.println("<xml version=\"1.0\">\n");
  Document document = (Document)node;
  processNode(document.getDocumentElement());
  break;
```

Element Node

- Represents an element
  - Includes starting tag, ending tag, content
- Child node types
  - Element nodes
  - Attribute nodes
  - ProcessingInstruction nodes
  - Comment nodes
  - Text nodes
  - CDATASection nodes
  - EntityReference nodes

Element Interface

```
public interface Element extends Node {
  public String   getTagName();
  public String   getAttribute(String name);
  public void     setAttribute(String name, String value) throws DOMException;
  public void     removeAttribute(String name) throws DOMException;
  public Attr     getAttributeNode(String name);
  public Attr     setAttributeNode(Attr newAttr) throws DOMException;
  public Attr     removeAttributeNode(Attr oldAttr) throws DOMException;
  public NodeList getElementsByTagName(String name);
  public String   getAttributeNS(String namespaceURI, String localName);
  public void     setAttributeNS(String namespaceURI, String qualifiedName,
    String value) throws DOMException;
  public void     removeAttributeNS(String namespaceURI, String localName)
    throws DOMException;
  public Attr     getAttributeNodeNS(String namespaceURI, String localName);
  public Attr     setAttributeNodeNS(Attr newAttr) throws DOMException;
  public NodeList getElementsByTagNameNS(String namespaceURI, String
    localName);
}
```

Element Node

- Using methods of Node interface
  - Element name
    - getNodeName()
  - Attribute names and values
    - NamedNodeMap getAttributes()
  - Child elements
    - NodeList getChildNodes()

Element Node Example

```
case Node.ELEMENT_NODE:
  String name = node.getNodeName();
  System.out.print("<" + name);

  NameNodeMap atts = node.getAttributes();
  for(int i = 0; i < atts.getLength(); i++){
      Node n = atts.item(i);
      System.out.print(" " + n.getNodeName() + "=\"" + n.getNodeValue()) + "\"");
  }
  System.out.println(">");

  // recurse on each child
  NodeList children = node.getChildNodes();
  if (children != null){
    for(int i = 0; i < children.getLength(); i++){
        processNode(children.item(i));
    }
  }
  System.out.println("</" + name + ">");
  break;
```

CharacterData Interface

- Represents things that are text
- Super interface of
  - Text interface
  - Comment interface

CharacterData Interface

```
public interface CharacterData extends Node {
   public String  getData()              throws DOMException;
   public void    setData(String data) throws DOMException;
   public int     getLength();
   public String  substringData(int offset, int count)
                                         throws DOMException;
   public void    appendData(String arg)
                                         throws DOMException;
   public void    insertData(int offset, String arg)
                                         throws DOMException;
   public void    deleteData(int offset, int count)
                                         throws DOMException;
   public void    replaceData(int offset, int count, String arg)
                                         throws DOMException;
}
```

Text Node

- Represents textual content of an element or attribute
- Contains only pure text, no markup
- A text node for each contiguous run of pure text
  - Element with no sub-elements
    - text is represented in a single Text object
  - Element with sub-elements
    - many Text objects

Text Node

- Node interface methods
  - *getNodeValue*() returns text
    - Same as *getData*() method of *CharacterData* interface

Text Interface

```
public interface Text extends CharacterData {
    public Text splitText(int offset)
                    throws DOMException;
}
```

Comment Node

- Represents a comment
- Child node types
  - None
- Node interface methods
  - *getNodeValue*() returns text

Comment Interface

```
public interface Comment extends CharacterData {
}
```

CDATA Section Node

- Represents a CDATA section
- Child node types
  - None
- Node interface methods
  - *getNodeValue*() returns text

CDATASection Interface

```
public interface CDATASection extends Text {
}
```

Code Example

```
case Node.TEXT_NODE:
case Node.CDATA_SECTION_NODE:
case Node.COMMENT_NODE:
    System.out.println(node.getNodeValue());
    break;
} // end switch statement
```

Attribute Node

- Represents an attribute
- Child node types
  - Text nodes
  - Entity reference nodes
- Node interface methods
  - *getNodeName*() returns name of attribute
  - *getNodeValue*() returns value of attribute

Attribute Interface

```
public interface Attr extends Node {

  public String   getName();
  public boolean  getSpecified();
  public String   getValue();
  public void     setValue(String value)
                          throws DOMException;
  public Element  getOwnerElement();

}
```

Document Type Node

- Represents document type declaration
- Child node types
  - None

DocumentType Interface

```
public interface DocumentType extends Node {

  public String      getName();
  public NamedNodeMap getEntities();
  public NamedNodeMap getNotations();
  public String      getPublicId();
  public String      getSystemId();
  public String      getInternalSubset();

}
```

Processing Instruction Node

- Represents a processing instruction
  `<?xml-stylesheet href="XSL\JavaXML.html.xsl" type="text/xsl"?>`
- Child node types
  - None
- Node interface methods
  - *getNodeName*() returns target
    - same as *getTarget*() of ProcessingInstruction interface
  - *getNodeValue*() returns the rest
    - same as *getData*() of ProcessingInstruction interface

Code Example

```
case Node.PROCESSING_INSTRUCTION_NODE:
        System.out.println("<?" + node.getNodeName() +
                           " " + node.getNodeValue() +
                           "?>");
        break;
```

Processing Instruction Interface

```
public interface ProcessingInstruction extends Node {

    public String  getTarget();
    public String  getData();
    public void    setData(String data)
                           throws DOMException;

}
```

ENTITY Node

- Represents an actual entity (not an entity reference)
- Child node types
  - Element nodes
  - ProcessingInstruction nodes
  - Comment nodes
  - Text nodes
  - CDATASection nodes
  - EntityReference nodes

ENTITY Interface

```
public interface Entity extends Node {

  public String  getPublicId();
  public String  getSystemId();
  public String  getNotationName();

}
```

DOM Programming Procedures

- Create a parser object
- Set Features and Read Properties
- Parse XML document and get Document object
- Perform operations
  - Traversing DOM
  - Manipulating DOM
  - Creating a new DOM
  - Writing out DOM

Creating a Parser Object

- Not standardized yet
  - DOM specification does not specify how to create parser object
  - Different API is used depending on implementation
  - Portability of your code gets affected
  - Reason to use JAXP

Code Example (Xerces)

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException ;
import java.io.IOException ;
        ...
String xmlFile  = "file://xerces-1_3_0/data/personal.xml";

DOMParser parser = new DOMParser();

try {
    parser.parse(xmlFile);
} catch (SAXException se) {
    se.printStackTrace();
} catch (IOException ioe) {
    ioe.printStackTrace();
}

Document document = parser.getDocument();
```

Set Features and Read Properties

- Standardized for both SAX 2 and DOM level 2
- Features
  - General features
  - DOM specific features
  - SAX specific features

Set Features

- General Features
  - http://xml.org/sax/features/validation
  - http://xml.org/sax/features/namespaces
  - http://apache.org/xml/features/validation/schema
  - http://apache.org/xml/features/allow-java-encodings
  - http://apache.org/xml/features/continue-after-fatal-error
  - http://apache.org/xml/features/nonvalidating/load-dtd-grammar
  - http://apache.org/xml/features/nonvalidating/load-external-dtd

Set Features

- DOM Features
  - http://apache.org/xml/features/dom/defer-node-expansion
    - Only when http://apache.org/xml/properties/dom/document-class-name is set to default
  - http://apache.org/xml/features/dom/create-entity-ref-nodes
  - http://apache.org/xml/features/dom/include-ignorable-whitespace

Code Example

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException ;
import java.io.IOException ;
                ...
String xmlFile = "file:///xerces-1_3_0/data/personal.xml";

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

try {
    parser.parse(xmlFile);
 } catch (SAXException se) {
    se.printStackTrace();
 } catch (IOException ioe) {
    ioe.printStackTrace();
 }

Document document = parser.getDocumen();
```

Read Properties

- http://apache.org/xml/properties/dom/current-element-node
- http://apache.org/xml/properties/dom/document-class-name
  - org.apache.xerces.dom.DocumentImpl

Parse XML Document & Get Document object

- Before any DOM operation can occur, XML document needs to be parsed and *Document* object needs to be created
- Not standardized by W3C
- Reason for using JAXP

Code Example (Xerces)

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException ;
import java.io.IOException ;
                ...
String xmlFile = "file:///xerces-1_3_0/data/personal.xml";

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

try {
    parser.parse(xmlFile);
 } catch (SAXException se) {
    se.printStackTrace();
 } catch (IOException ioe) {
    ioe.printStackTrace();
 }

Document document = parser.getDocumen();
```

Perform DOM operations

- Traversing DOM
- Manipulating DOM
  - Appending nodes
  - Removing nodes
  - Modifying nodes
- Generating a new DOM
- Serializing DOMb
- Q: subclassing DOM?

Traversing DOM Tree

- http://www.w3.org/TR/DOM-Level-2-Traversal-Range/traversal.html
- org.w3c.dom.traversal.*
- Interfaces
  - DocumentTraversal
  - NodeIterator
  - NodeFilter
  - TreeWalker

DocumentTraversal Interface

- Contains methods that create iterators and tree-walkers to traverse a node and its children in document order
- In DOMs which support the Traversal feature, *DocumentTraversal* will be implemented by the same objects that implement the *Document* interface.
  - document.isSupported("Traversal", "2.0")

DocumentTraversal Interface

```
public interface DocumentTraversal{
    public NodeIterator createNodeIterator(
                                Node root,
                                int whatToShow,
                                NodeFilter filter,
                                boolean entityReferenceExpansion)
                                        throws DOMException;
    public TreeWalker createTreeWalker(
                                Node root,
                                int whatToShow,
                                NodeFilter filter,
                                boolean entityReferenceExpansion)
                                        throws DOMException;
}
```

Code Example

```
// Assuming DOM implementation supports
// the traversal
NodeIterator  iterator =
            ((DocumentTraversal)document).
            createNodeIterator(
                        document,
                        NodeFilter.SHOW_ALL,
                        new NameNodeFilter(),
                        true);
```

NodeIterator Interface

- Used to step through a set of nodes, i.e. the set of nodes in a *NodeList*
  - Document subtree
  - Result of a query
- Implementation instance of this interface gets returned from *createNodeIterator*() method of *DocumentTraversal* interface

NodeIterator Interface

```
public interface NodeIterator{
    public Node getRoot();
    public int getWhatToShow();
    public NodeFilter getFilter();
    public boolean getExpandEntityReferences();
    public Node nextNode()
                throws DOMException;
    public Node previousNode()
                throws DOMException;
    public void detach();
}
```

NodeFilter Interface

- Used to filter out a single node
- Used with NodeIterator and TreeWalker
  - Applied in determining next node
- You have to implement it
- Also contains WhatToShow flags

NodeFilter Interface

```
public interface NodeFilter {
    // Return values of acceptNode() method
    public static final short FILTER_ACCEPT         = 1;
    public static final short FILTER_REJECT         = 2;
    public static final short FILTER_SKIP           = 3;
    // WhatToShow flags
    public static final int SHOW_ALL                = 0xFFFFFFFF;
    public public static finalint SHOW_ELEMENT          = 0x00000001;
    public static final int SHOW_ATTRIBUTE          = 0x00000002;
    public static final int SHOW_TEXT               = 0x00000004;
    public static final int SHOW_CDATA_SECTION      = 0x00000008;
    public static final int SHOW_ENTITY_REFERENCE   = 0x00000010;
    public static final int SHOW_ENTITY             = 0x00000020;
    public static final int SHOW_PROCESSING_INSTRUCTION = 0x00000040;
    public static final int SHOW_COMMENT            = 0x00000080;
    public static final int SHOW_DOCUMENT           = 0x00000100;
    public static final int SHOW_DOCUMENT_TYPE      = 0x00000200;
    public static final int SHOW_DOCUMENT_FRAGMENT  = 0x00000400;
    public static final int SHOW_NOTATION           = 0x00000800;
    public short acceptNode(Node n);
}
```

TreeWalker Interface

- Used to navigate logical view of a tree (or subtree)
- Works with a logical view of a tree
  - Tree or subtree where nodes are filtered out via *whatToShow* flags and *filter* (if any)
  - Different from non-filtered tree
    - Tree structure
    - Parent and sibling relationships

TreeWalker Interface

```
public interface TreeWalker{
    public Node getRoot();
    public int getWhatToShow();
    public NodeFilter getFilter();
    public getExpandedEntityReferences();
    public Node getCurrentNode();
    public void setCurrentNode (Node currentNode)
                        throws DOMException;
    public Node parentNode();
    public Node firstChild();
    public Node lastChild();
    public Node previousSibling();
    public Node nextSibling();
    public Node previousNode();
    public Node nextNode();
}
```

Example

```
treeWalker = ((DocumentTraversal)document).
                createTreeWalker(
                        document,
                        NodeFilter.SHOW_ALL,
                        new NameNodeFilter(),
                        expand);
```

Manipulating DOM

- Create a new node
- Add a child node
- Remove a child node
- Change value of a node
- Normalizing text node

Create a New Node and Add a Child Node

```
Node node = jtree.getNode(treeNode);
Node textNode = document.createTextNode(text);
try {
    node.appendChild(textNode);
} catch (DOMException dome) {
    setMessage("DOMException:"+dome.code+", "+dome);
    return;
}
```

Remove a Child Node

```
Node parent = node.getParentNode();
parent.removeChild(node);
```

Generating A New DOM

- Generate a new DOM from scratch
- Not a DOM standard yet
  - Use Apache Xerces implementation
  - *org.apache.xerces.dom.DocumentImpl*
    - *WMLDocumentImpl*
    - *HTMLDocumentImpl*

Example

```
try {
        // Generate a new DOM tree
        Document doc= new DocumentImpl ();
        Element root = doc.createElement("person");     // Create Root Element
        Element item =doc.createElement("name");      // Create element
        item. appendChild( doc.createTextNode("Jeff") );
        root.appendChild( item );              // atach element to Root element
        item = doc.createElement("age");              // Create another Element
        item. appendChild( doc.createTextNode("28" ) );
        root.appendChild( item );          // Attach Element to previous element down tree
        item = doc.createElement("height");
        item. appendChild( doc.createTextNode("1.80" ) );
        root.appendChild( item );                  // Attach another Element - grandaugther
        doc.appendChild( root );                    // Add Root to Document

} catch ( Exception ex ) {
        ex.printStackTrace();
 }
```

Serializing DOM

- Not DOM standard
  - org.apache.xml.serialize.* package
- Serialize DOM into a string
  - Used to create XML document
- Interfaces
  - Serializer
- Classes
  - XMLSerializer
  - OutputFormat

Serializer Interface

- Interface for a DOM serializer implementation
  - *BaseMarkupSerializer*
    - *HTMLSerializer*
    - *TextSerializer*
    - *XMLSerializer*
- Contains factory method for SAX and DOM serializer
- Contains static method for serializing DOM

OutputFormat Class

- Specifies an output format to control the serializer
- Defaults for
  - encoding
  - indentation
  - line separator
- Default encoding is UTF-8

Example

```
try {

    // Generate a new DOM tree
    Document doc= new DocumentImpl ();
    Element root = doc.createElement("person");     // Create Root Element

    //  Output DOM to a String using Serializer
    OutputFormat    format  = new OutputFormat( doc );  //Serialize DOM
    StringWriter  stringOut= new StringWriter();      //Writer will be a String
    XMLSerializer    serial = new XMLSerializer( stringOut format );
    serial. asDOMSerializer();                        // As a DOM Serializer

    serial.serialize( doc.getDocumentElement() );

    //Spit out DOM as a String
    System.out.println( "STRXML = " +stringOut.toString() );
  } catch ( Exception ex ) {
    ex.printStackTrace();
  }
```

Normalizing Text Nodes

- Text nodes include newline characters
- Normalizing reduce different line endings to a single newline
- Normalizing reduces multiple white space characters to one white space character
- Normalizing ensures that **Text** Nodes have no adjacent **Text** nodes
- Useful to compare XML documents

Example

```
Node node = jtree.getNode(treeNode);
node.normalize();
```

Error Handling

- DOMException
  - For many DOM methods
- SAXException
  - For parse() method

Code Example

```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.SAXException ;
import java.io.IOException ;
                ...
String xmlFile  = "file:///xerces-1_3_0/data/personal. xml";

DOMParser parser = new DOMParser();

try {
    parser.parse(xmlFile);
 } catch (SAXException se) {
    se.printStackTrace();
 } catch (IOException ioe) {
    ioe.printStackTrace();
 }

Document document = parser. getDocument();
```

Error Handlers (Xerces)

- Same error handler scheme work for both SAX and DOM on Xerces implementation
  - Both SAXParser and DOMParser are subclasses of *org.apache.xerces.framework.XMLParser*
  - Both SAX and DOM code share the same methods including error handlers

Code Example (Xerces)

```
DOMParser parser = new DOMParser();
parser.setErrorHandler(new myErrorHandler());

// Error handler class
public class myErrorHandler implements ErrorHandler{
    public void warning(SAXParseException ex) {
        System.err.println("[Warning] "+ getLocationString(ex)+": "+ ex.getMessage());
    }

    public void error(SAXParseException ex) {
        System.err.println("[Error] "+ getLocationString(ex)+": "+ ex.getMessage());
    }

    public void fatalError(SAXParseException ex) throws SAXException {
        System.err.println("[Fatal Error] "+ getLocationString(ex)+": "+ ex.getMessage());
        throw ex;
    }
}
```

Demo & Code Review

- DOMWriter (Apache Xerces)
- DOMCount (Apache Xerces)
- DOMGenerate (Apache Xerces)
- IteratorView (Aparche Xerces)
- TreeWalkerView (Apache Xerces)
  - NamedNodeFilter (Apache Xerces)

Benefits of DOM

- Provides random-access manipulation of an XML file
- A DOM can be created from scratch or an existing file can be edited in memory

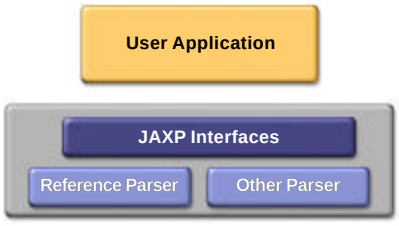
Drawbacks of DOM

- Large documents could be problematic since the entire document is read into memory
- Performance could suffer using DOM with large document and/or limited memory availability
- No standard support for reading documents or writing DOM models out to files (addressed in the Level 3 specification)

JAXP 1.1

- A thin and lightweight Java API for parsing and transforming XML documents
- Allows for pluggable parsers and transformers
- Allows parsing of XML document using:
  - Event-driven (SAX 2.0)
  - Tree based (DOM Level 2)

JAXP: Pluggable Framework for Parsers and Transformers



JAXP 1.1

- JAXP 1.1 implements the DOM interfaces
- DOM does not specify how a tree is created in memory nor how its content is obtained
- DOM does specify how this tree can be navigated and manipulated
- The DOM parser must be instantiated explicitly using vendor-specific routines

JAXP/DOM Code Example

```
01 import javax.xml.parsers.*;
02 import org.w3c.dom.*;
03
04 DocumentBuilderFactory factory =
05     DocumentBuilderFactory.newInstance();
06 factory.setValidating(true);
07 DocumentBuilder builder =
08     factory.newDocumentBuilder();
09
10 // can also parse InputStreams, Files, and
11 // SAX input sources
12 Document doc =
13     builder.parse("http://foo.com/bar.xml");
```

Summary

- DOM Characteristics
- DOM node tree and node types
- Java interface hierarchy
- Java interfaces
- DOM Programming

References

- Apache Xerces 1.3
- "XML in a Nutshell" written by Elliotte Rusty Harold & W. Scott Means, O'Reilly, Jan. 2001 (First Edition), Chapter "DOM"