DOM (Document Object Model)

Sang Shin
JPassion.com
“Learn with Passion!”

Topics

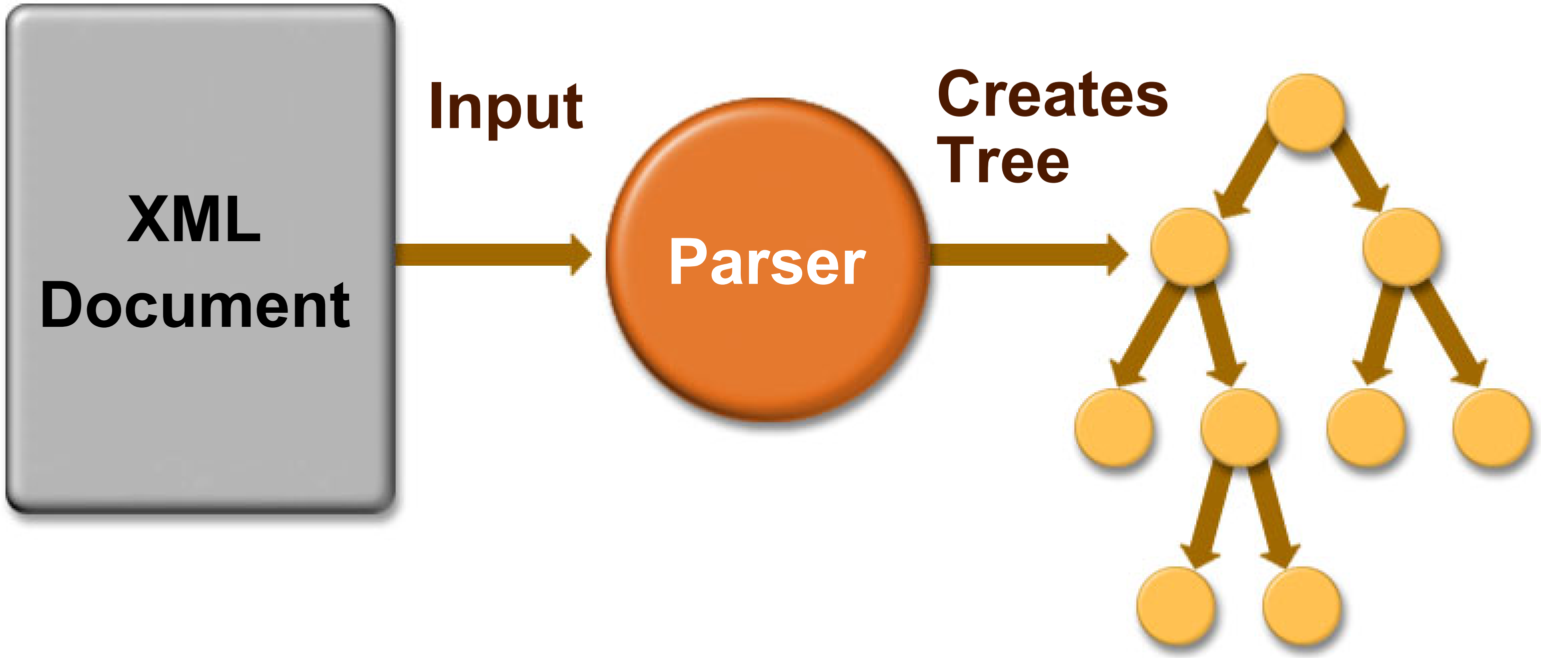
- DOM Characteristics
- DOM Node Tree and Node Types
- DOM & Java Interfaces
- DOM Operations
  - > Traversing DOM
  - > Manipulating DOM
  - > Creating a new DOM
  - > Writing out (Serializing) DOM
- Benefits and Drawbacks of DOM
- DOM Support in JAXP 1.1

DOM Characteristics

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 a large XML document
- Use it when for walking and modifying the tree

DOM Operational Model



DOM Node Tree and Node Types

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
<?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 & Java Interfaces

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 convenience 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
  - Text 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;
}
```

DOM Operations

4. Perform DOM operations

- Traversing DOM
- Manipulating DOM
  - Appending nodes
  - Removing nodes
  - Modifying nodes
- Generating a new DOM
- Serializing DOM

Traversing 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

Manipulating DOM

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);
```

Benefits and Drawbacks of DOM

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

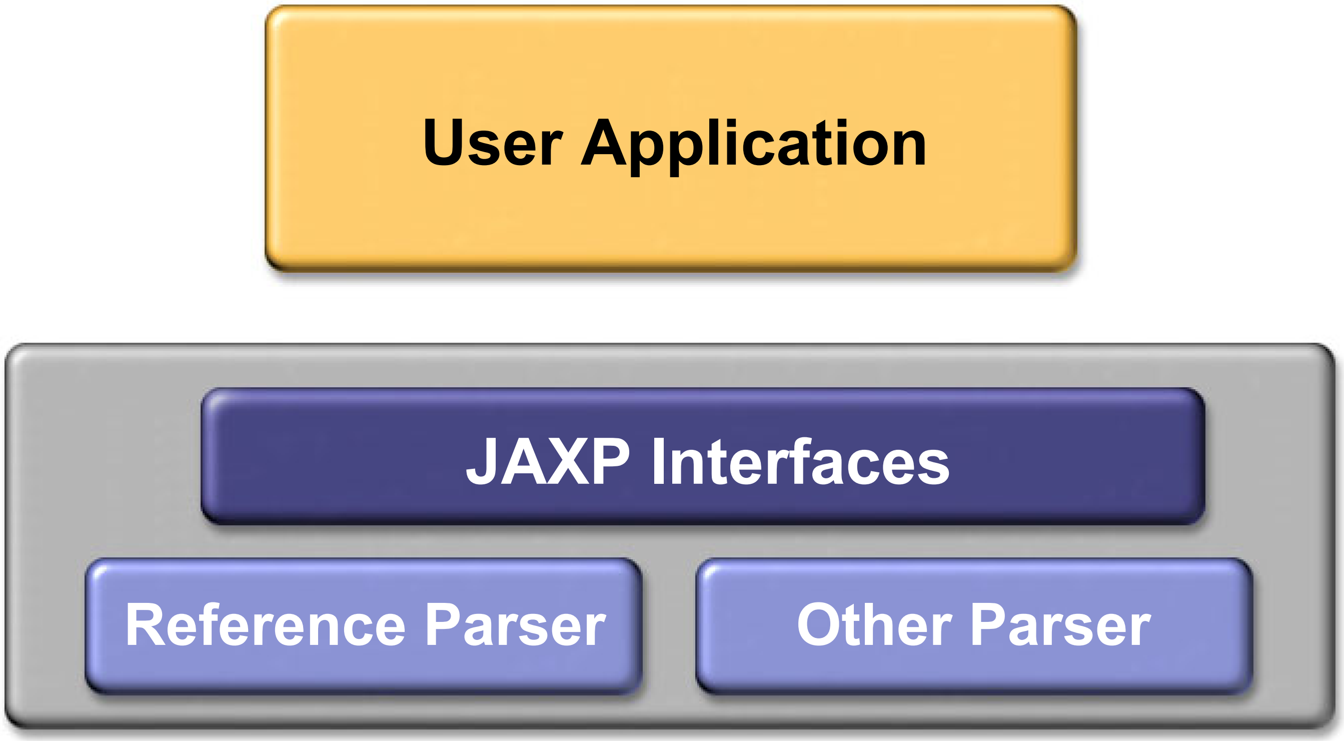
- Large documents could be problematic since the entire document is read into memory
- Performance could suffer using DOM with large document and/or limited memory availability
- No standard support for reading documents or writing DOM models out to files (addressed in the Level 3 specification)

DOM Support in JAXP 1.1

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

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

DocumentBuilderFactory factory =
     DocumentBuilderFactory.newInstance();
factory.setValidating(true);
DocumentBuilder builder =
     factory.newDocumentBuilder();

// can also parse InputStreams, Files, and
// SAX input sources
Document doc =
     builder.parse("http://foo.com/bar.xml");
```

Learn with Passion!
JPassion.com