



Document Object Model (DOM)

Asst. Prof. Dr. Kanda Runapongsa Saikaew
(krunapon@kku.ac.th)
Dept. of Computer Engineering
Khon Kaen University



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



Document Object Model

- Define how XML and HTML documents are represented as objects in programs
- W3C Standard
- Defined in IDL; this language independent
- HTML as well as XML
- Writing as well as reading
- DOM focuses more on the document, SAX focuses more on the parser

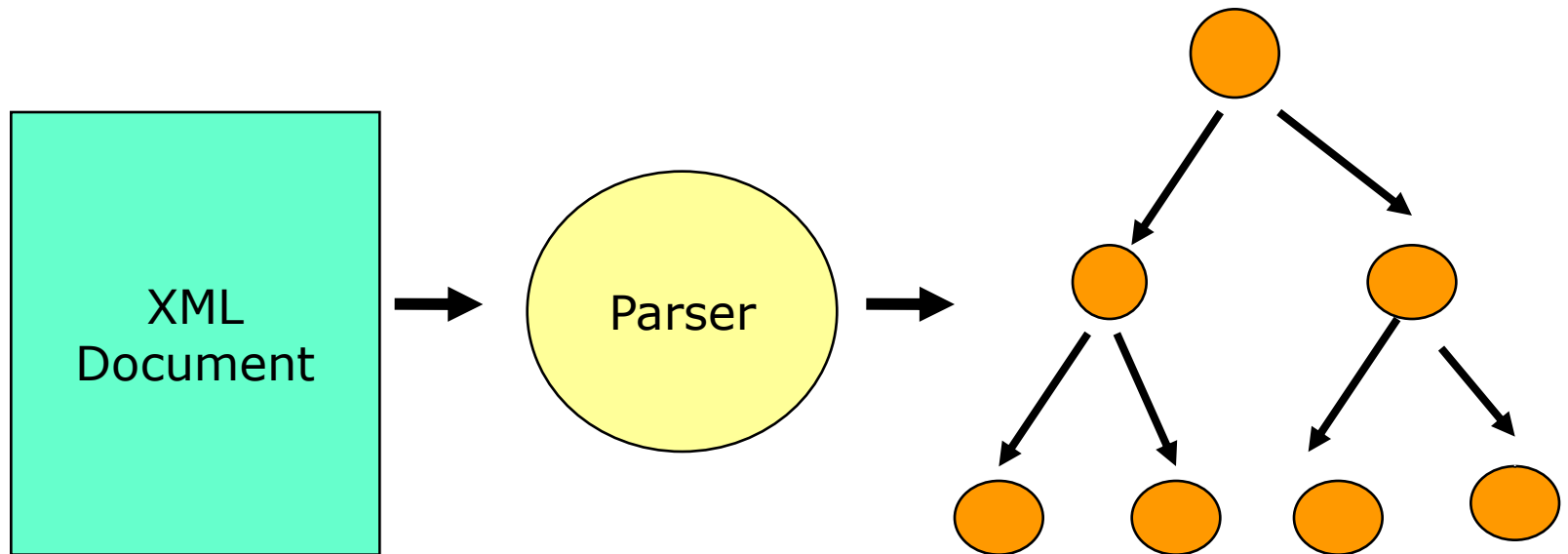


DOM Characteristics

- ❑ 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 type)
 - Parent node contains child nodes



Tree

- According to DOM, an XML document is a tree made up of nodes of several types
- The tree has a single root node, and all nodes in this tree except for root have a single parent node
- Each node has a list of child nodes
 - How to call a node that has the empty list of children?
 - A leaf node



Tree and Nodes

- There can also be nodes that are not part of the tree structure
 - Each attribute node belongs to one element node but is not considered to be a child of that element
- A full DOM document is composed of
 - A tree of nodes
 - Various nodes that are somehow associated with other nodes but are not themselves part of the tree



Tree Node Types (1/2)

- DOM divides nodes into twelve types, seven of which can potentially be part of a DOM tree
 - Document nodes
 - Element nodes
 - Text nodes
 - Attribute nodes



Tree Node Types (2/2)

- Types of a node in DOM
 - Processing instruction nodes
 - Comment nodes
 - Document type nodes
 - Document fragment nodes
 - Notation nodes
 - CDATA section nodes
 - Entity nodes
 - Entity reference nodes



Tree Node Types Example

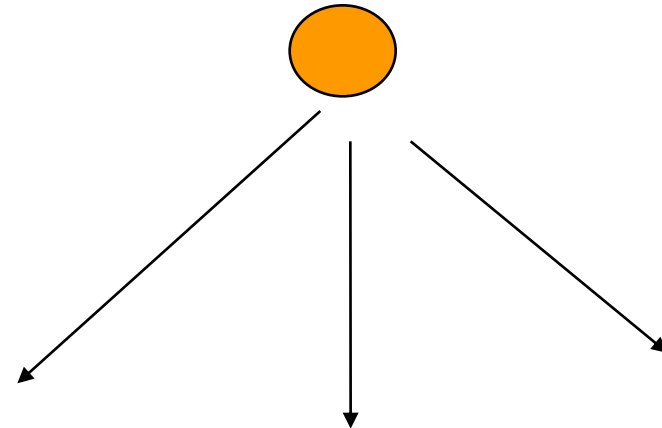
Node Type	Example
Document type	<code><!DOCTYPE nation SYSTEM "nation.dtd"></code>
Processing instruction	<code><?xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/Transform?></code>
Element	<code><nation id="th">...</nation></code> <code><name>Thailand</name></code>
Attribute	<code>id="th"</code>
Text	<code><name>Thailand</name></code>



Tree View of the XML Document

- **documentElement** is the root node of the tree
- **documentElement** has the type as Document node

documentElement





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



Tree Nodes (1/2)

- Besides its tree connections, each node has a local name, a namespace URI, and a prefix

- Example:

```
<html xmlns:h="http://www.w3.org/tr/html">
```

```
<h:booktitle/>
```

```
</html>
```

- Local name: booktitle
- Namespace URI: <http://www.w3.org/tr/html>
- Prefix: h
- Qualified name: h:booktitle



Tree Nodes (2/2)

- Though for several kinds of nodes, local name, namespace URI, and prefix may be null
 - For instance, these information of a comment are always be null

`<!-- comment -->`

- Comment node does not have its name, its namespace URI, and its prefix



Tree Node Names

- For an element or attribute, the node name is the prefixed name
 - Example: `<h:booktitle>`
- For the notation or entities, the node name is the name of the thing
 - `<!ENTITY xml "Extensible Markup Language">`
- For nodes without names such as text nodes, the node name is the value from the following list matching the node type
 - `#comment`



Tree Node Values

- Each node has a string value
- For text-ish things like text nodes and comments, this tends to be the text of the node
 - `<name>Thailand</name>`
- For attributes, it's normalized value of the attribute
 - `<nation id="th">...</nation>`
- For everything else, including elements and documents, the value is null



Example XML Document

```
<?xml version="1.0"?>
<nation id="th">
  <name>Thailand</name>
  <location>Southeast
Asia</location>
</nation>
```

- Document node
 - Element node “nation”
 - Text node
 - Element node “name”
 - Text node “Thailand”
 - Text node
 - Element “location”
 - Text node “Southeast Asia”
 - Text node
 - * Attribute node “id”
 - Text node “th”



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

- Most frequently used interface methods
 - `public short getNodeType()`
 - `public String getNodeName()`
 - `public String getNodeValue()`
 - `public Node getParentNode()`
 - `public Node getPreviousSibling()`
 - `public Node getNextSibling()`
 - `public NamedNodeMap getAttributes()`
 - `public NodeList getChildNodes()`



Node Interface (1/3)

```
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 (2/3)

```
public String getNodeName();  
public String getNodeValue() throws DOMException;  
public void setNodeValue(String nodeValue) throws  
DOMException;  
public short getNode Type ();  
public Node getParentNode();  
public NodeList getChildNodes();  
public Node getFirstChild();  
public Node getLastChild();  
public Node getPreviousSibling();  
public Node getNextSibling();  
public NamedNodeMap getAttributes();  
public Document getOwnerDocument( ) ;  
public Node insertBefore (Node newChild , Node refChild)  
throws DOMException;
```



Node Interface (3/3)

```
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 be identified **by name**
- Return type of `getAttributes()` method of Node interface
- Most frequently used methods
 - `public Node item(int index);`
 - `public int getLength();`
 - `public Node getNamedItem(String name);`
 - `public Node removeNamedItem(String name)` throws `DOMException`;



Document Node (1/2)

- Root node of DOM tree
- Represents entire document
- Child node types
 - One element node
 - Optional document type node
 - Optional instruction nodes
 - Optional comment nodes



Document Node (2/2)

- If the document contains any comments or processing instructions before or after the root element, then these will also be child nodes of the document node

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

```
<!DOCTYPE nation SYSTEM "nation.dtd">
```

```
<nation>...</nation>
```

- The document node has the document type node as the first child and has the `<nation>` element as the second child



Document Interface Purpose

- Contains factory methods for **creating other nodes**
 - Elements, text nodes, comments, processing instructions, etc.
- Method to **get the root element node** of the input XML document
 - Element `getDocumentElement()`



Document Interface

```
public interface Document extends Node {
    Attr createAttribute (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) }
```



Document Node Example

```
// Create an instance of a DOM parser and
// get the Document Object
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder parser =
factory.newDocumentBuilder();

// xmlDoc is the root node of DOM tree
Document xmlDoc = parser.parse(filePath);

// obtain the root element of the document
Element rootElement =
xmlDoc.getDocumentElement();
```



Element Node

- Represents an element
 - Includes start tag, end tag, content
- Child node types
 - Element nodes
 - Attribute nodes
 - Processing instruction 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

```
if (node.getNodeType() == Node.ELEMENT_NODE) {
    NodeList nodeList = node.getChildNodes();
    for (int i = 0; i < nodeList.getLength(); i++) {
        Node childNode = nodeList.item(i);
        if (childNode.hasAttributes()) {
            NamedNodeMap attrMap =
                node.getAttributes();
            for (int j = 0; j < attrMap.getLength(); j++) {
                Attr attribute = (Attr) attrMap.item(j);
                System.out.println(attribute.getName()
                    + "=" + attribute.getValue());
            }
        }
    }
}
```



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
- Use `getNodeValue()` to return text



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



Attr Interface

```
public interface Attr extends Node {  
    String getName();  
    Element getOwnerElement();  
    boolean getSpecified();  
    String getValue() ;  
    void setValue(String);  
}
```



JAXP

- 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)



DOM Programming Procedures

- Create a parser object
- Read or set properties
- Parse XML document and get Document object
- Perform operations
 - Traversing DOM
 - Manipulating DOM
 - Creating a new DOM
 - Writing out DOM



Create a Parser Object (Sample)

```
import javax.xml.parsers.*;
import java.io.File;
import org.w3c.dom.*;
public class JAXPChecker {
    ...
    String documentName = argv[0];
    DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder =
        factory.newDocumentBuilder();
        Document doc = builder.parse( new
        File(documentName));
        ...
    }
}
```



Read or set builder properties

□ For reading/setting

- is/set Validating: validate the XML content during parse
- is/set Coalescing: convert CDATA nodes to Text nodes and append it to the adjacent (if any) text node
- is/set ExpandEntityReferences: expand entity reference nodes
- is/set NamespaceAware: provide support for XML namespaces



Read/set builder properties (Sample)

```
import javax.xml.parsers.*;  
import java.io.File;  
import org.w3c.dom.*;  
public class JAXPChecker {
```

```
    ...
```

```
        DocumentBuilderFactory factory =  
        DocumentBuilderFactory.newInstance();  
        factory.setValidating(true);  
        factory.setNamespaceAware(true);  
        System.out.println("coalescing or not " +  
            factory.isCoalescing());
```



Validating (Sample 1/2)

```
// Create a SchemaFactory capable of
// understanding W3C XML Schemas (WXS)
SchemaFactory schemaFactory =
    SchemaFactory.newInstance(
        XMLConstants.W3C_XML_SCHEMA_NS_URI);

// Set the error handler to receive any
// error during Schema compilation
    schemaFactory.setErrorHandler(new
        MyErrorHandler());
```



Validating (Sample 2/2)

```
// Load a WXS schema, represented by a Schema instance
```

```
Source schemaSource =
```

```
    new StreamSource(new File(xsdName));
```

```
Schema schema =
```

```
schemaFactory.newSchema(schemaSource);
```

```
// Validate the XML file that refers to the schema instance
```

```
Validator validator = schema.newValidator();
```

```
validator.validate(new
```

```
    StreamSource(documentName));
```



Traversing DOM

- After we get document node, then we traverse DOM tree using methods
 - getChildNodes()
 - getFirstChild()
 - getLastChild()
 - getParentNode()
 - getNextSibling()
 - getPreviousSibling()



Traversing DOM Example

```
// note use of recursion
public void followNode(Node node) throws IOException {

    printer.writeNode(node);
    if (node.hasChildNodes()) {
        String name = node.getNodeName();
        int numChildren = node.getChildNodes().getLength();
        System.out.println("node " + name + " has " + numChildren
            + " children");
        Node firstChild = node.getFirstChild();
        followNode(firstChild);
    }
    Node nextNode = node.getNextSibling();
    if (nextNode != null) followNode(nextNode);

}

...
// Read the entire document into memory
Node document = parser.parse(args[0]);
// Start processing from the root node of DOM tree
followNode(document);
```



Create an XML Document

- Create a new document node
- Use the document node to
 - Create element nodes
 - Create attribute nodes
 - Create text nodes
 - Link the relationships between those nodes
- Transform from DOM to an XML file



Create an XML Document (1/2)

```
// Create a new Document node
Document doc = builder.newDocument();
// Create various nodes
Element root = doc.createElement("profile");
    Element nameE =
doc.createElement("name");
    Text nameT = doc.createTextNode("Kanda
Saikaew");
// Link relationships between created nodes
nameE.appendChild(nameT);
root.appendChild(nameE);
doc.appendChild(root);
```



Create an XML Document (2/2)

```
OutputStream os = new  
FileOutputStream(xmlProfile);
```

```
TransformerFactory tf =  
TransformerFactory.newInstance();
```

```
Transformer trans = tf.newTransformer();  
trans.transform(new
```

```
DOMSource(doc),  
new StreamResult(os));
```



Create an XML Doc with Namespace

```
// default namespace
```

```
Element root = doc.createElementNS("http://www.kku.ac.th",  
    "profile");
```

```
root.setAttribute("xmlns:xsi",  
    "http://www.w3.org/2001/XMLSchema-instance");
```

```
root.setAttribute("xsi:schemaLocation",  
    "http://www.kku.ac.th profile.xsd");
```

```
root.setAttribute("xmlns", "http://www.kku.ac.th");
```

```
// namespace with prefix
```

```
Element nameE =
```

```
doc.createElementNS("http://gear.kku.ac.th", "name");
```

```
nameE.setAttribute("xmlns:coe", "http://gear.kku.ac.th");
```

```
nameE.setPrefix("coe");
```



Benefits of DOM

- Provides random-access manipulation of an XML file
 - Traverse from a node to its parent and its child very easily
- A DOM can be created from scratch or an existing file can be edited in memory
- Easy to develop a DOM parser
 - The developer does not need to design the model and the data structure for the model



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



Summary

□ DOM Characteristics

- DOM models XML document as a tree and keeps the tree in memory

□ DOM node tree and node types

- The root node of DOM tree is the document node which contains the root node of the input XML document

□ DOM Programming

- Traverse the document
- Manipulate the document
- Create the new document



References

- “XML in a Nutshell” written by Elliotte Rusty Harold & W.Scott Means, O’Reilly
- Sang Shin “DOM (Document Object Model)”,
<http://www.javapassion.com/xml>
- K. Yue, “An Introduction to DOM Parser (with JAXP)”,
<http://sce.uhcl.edu/yue/courses/xml/notes/xmlparser/IntroDOM.asp>