ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στην Επιστήμη και Τεχνολογία των Υπηρεσιών

Ενότητα 12: Document Object Model (DOM) - 1

Χρήστος Νικολάου
Τμήμα Επιστήμης Υπολογιστών

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται στην άδεια χρήσης **Creative Commons** και ειδικότερα

Αναφορά – Μη εμπορική Χρήση – Όχι Παράγωγο Έργο v. 3.0

(Attribution – Non Commercial – Non-derivatives)

- Εξαιρείται από την ως άνω άδεια υλικό που περιλαμβάνεται στις διαφάνειες του μαθήματος, και υπόκειται σε άλλου τύπου άδεια χρήσης. Η άδεια χρήσης στην οποία υπόκειται το υλικό αυτό αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

XML
Document Object Model (DOM)
605.444 / 635.444

David Silberberg

Lecture 18

Introduction to DOM

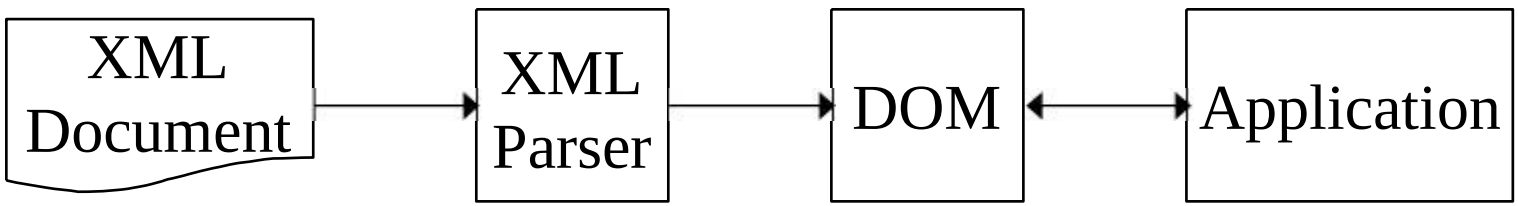
- Origins in the World Wide Web Consortium (W3C)
  - DOM is a standard, like XML itself
  - SAX is different
    - Public-domain software
    - Developed through XML-dev mailing list
- Not designed specifically for Java
  - Represents content across all programming languages and tools
  - Bindings exist for
    - JavaScript
    - Java
    - CORBA
  - Cross-platform and cross-language specification

Introduction (continued)

- DOM is organized by levels, not versions
  - Each level gets more specific for different representations
    - XML
    - HTML
    - Cascading Stylesheets
  - Level 1 an accepted recommendation
    - http://www.w3c.org/TR/REC-DOM-Level-1/
    - Details the navigation of content within an XML-like document
  - Level 2
    - http://www.w3c.org/TR/REC-DOM-Level-2/
    - Supplies modules and options for specific content models
    - Specifically address representations unique to XML, HTML, CSS, etc.
  - Level 3
    - http://www.w3.org/TR/DOM-Level-3-Core/
    - Recommendation 07-April-2004

What is the DOM?

- The Document Object Model (DOM) is an abstract representation of an XML document
- A structural representation of a document
- Process
  - Parse an XML document
  - Create a representative model
  - Allow application(s) to interact with and manipulate the model



Why Not SAX?

- SAX is sequential
  - SAX acquires and loses the data as the parser does
  - DOM provides an entire document model at once
  - Access to DOM can be random
    - Can access multiple elements simultaneously
    - Can access elements out of order
- SAX has no concept of siblings
  - It is a depth-first parser
  - Does not inherently keep track of parents, children, or siblings
  - DOM provides the entire hierarchy at once

Getting a DOM Parser

- Apache Xerces: http://xml.apache.org
- IBM XML4J: http://alphaworks.ibm.com/tech/xml4j
- Aelfred: http://www.microstar.com/aelfred.html
- James Clark's XP: http://www.jclark.com/xml/xp
- Datachannel DXP:
  http://www.datachannel.com/products/xjparser.html
- OpenXML: http://www.openxml.org
- Oracle XML Parser: http://technet.oracle.com/tech/xml
- Sun Microsystems Project X: http:// java.sun.com/products/xml
- Tim Bray's Lark and Larval: http://www.textuality.com/Lark
- W3C Web site: http://www.w3c.org

- Microsoft MSXML parser does not conform to W3C standards

Getting Started with DOM

- DOM does not specify structure of hierarchy
- DOM specifies interfaces
  - Enables multi-language binding
  - Enables freedom of representation or DOM vendors
- Classes to import
  - import org.w3c.dom.*;
- Most parsers and XSLT packages already contain DOM

Importing DOM Parser

```
import org.apache.xerces.parsers.DOMParser;    // vendor DOM parser


public class DOMParserDemo {

    pubic void performDemo(String uri) {
        System.out.println("Parsing XML File: " + uri + "\n");
        DOMParser parser = new DOMParser();
        try {
            // parser.parse(uri);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
```

Importing DOM Parser (2)

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

        String uri = args[0];

        if (args.length != 1) {
            System.out.println("Usage: java DOMParser [URI]");
            System.exit(0);
        }
        DOMParserDemo pDemo = new DOMParserDemo();
        pDemo.performDemo(uri);
    }
}
```

Lack of DOM Portability

- SAX provides a SAXParserFactory(*SAXObject*) class
  - Import the class name
  - Factory instantiates a call Class.forName(*SAXObject*)
- DOM does not provide this portability
  - Vendor differences prevent this
  - Some DOM **parse()** routines return an **org.w3c.dom.Document** object
  - Some DOM **parse()** routines provide a **getDocument()** method
  - Sometimes different parameters are required for the parse
- However, routines that access and manipulate the DOM tree are standard

Performing the Parse and Getting the Document

- SAX uses callbacks
  - Programmer creates methods to deal with events
  - Events are called during the parse
  - Callbacks are like "hooks" into the data
- DOM creates a tree structure first
  - Program looses control until the parse is complete
  - Program gets a "handle" on the document
  - Programmer writes routines to access and manipulate the DOM Document tree

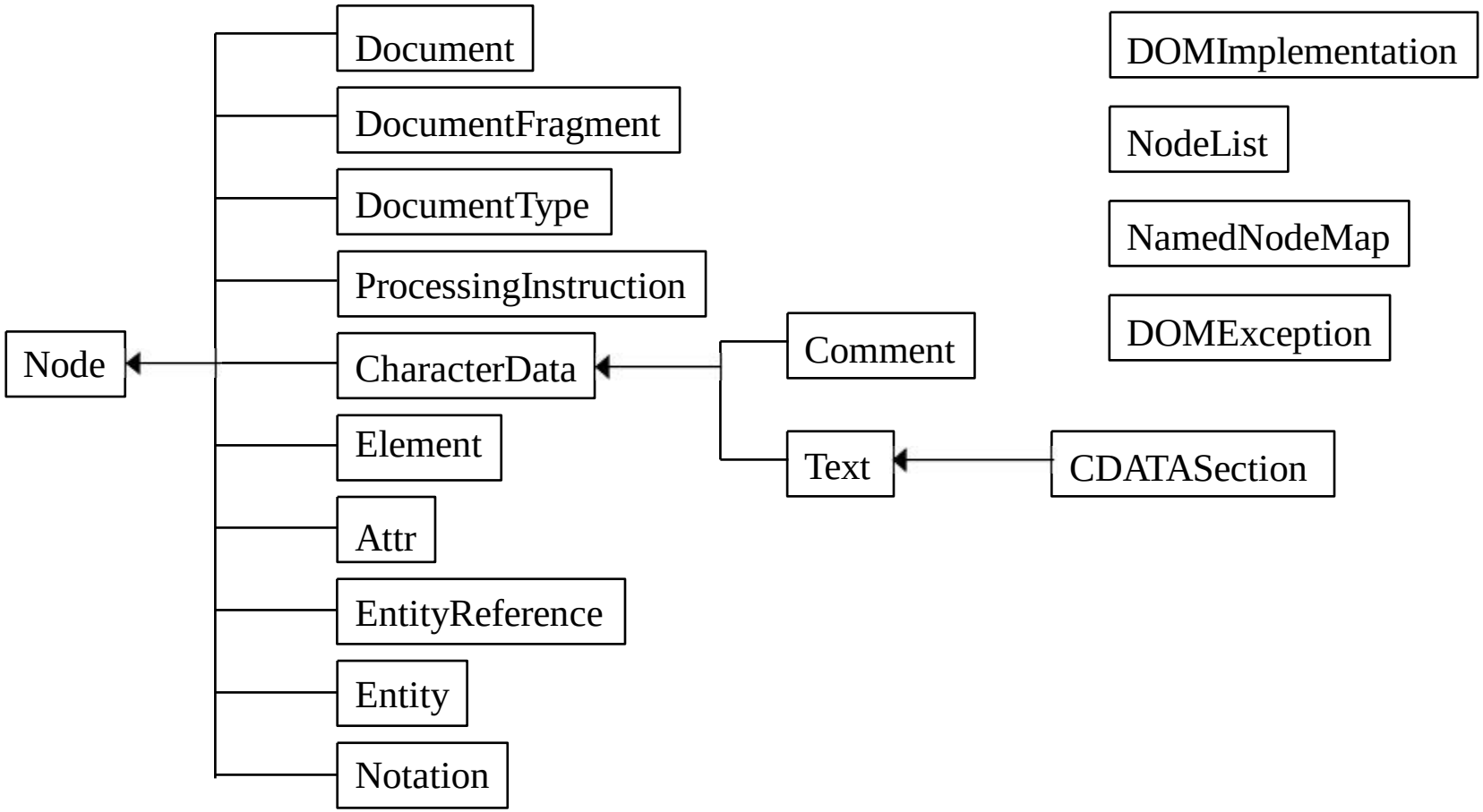
Example (Xerces Xalan)

```
import org.apache.xerces.parsers.DOMParser;          // vendor DOM parser
import org.w3c.dom.Document;                         // Document class


public class DOMParserDemo {

    pubic void performDemo(String uri) {
        System.out.println("Parsing XML File: " + uri + "\n");
        DOMParser parser = new DOMParser();
        try {
            parser.parse(uri);
            Document doc = parser.getDocument();
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
```

Accessing the DOM Tree

- A DOM document is a tree
- To print out a document backwards (sort of), you need to traverse the tree
- Each element of the tree is a Node
- Depending on the type of node, different methods are available
  - Different nodes are subclasses of the Node class
  - Methods are inherited
- Find children and recursively process them

Core DOM Data Type Inheritance Hierarchy



Setup for Print Reverse

```
import org.apache.xerces.parsers.DOMParser;          // vendor DOM parser
import org.w3c.dom.Document;                         // Document class
import org.w3c.dom.Node;                             // Node class
import java.io.File;
import java.io.FileWriter;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import org.w3c.dom.DocumentType;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.NodeList;
```

Setup for Print Reverse (2)

```
public class DOMParserDemo {

    pubic void writeReverse(String uri, String fileName) {
        System.out.println("Parsing XML File: " + uri + "\n");
        System.out.println("Writing reverse XML File: " + fileName + "\n");
        DOMParser parser = new DOMParser();
        try {
            parser.parse(uri);
            Document doc = parser.getDocument();
            printReverse(doc, fileName);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }
```

Setup for Print Reverse (3)

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

    String uri = args[0];
    String fileName = args[1];

    if (args.length != 2) {
        System.out.println("Usage: java DOMParser [URI] [Outfile]");
        System.exit(0);
    }
    DOMParserDemo pDemo = new DOMParserDemo();
    pDemo.writeReverse(uri, fileName);
}
```

Setup for Print Reverse (4)

```
int indent = -4;          // The number of characters to indent the output


public void printReverse(Document doc, String fileName)  {
      try{
         FileWriter fw = new FileWriter(fileName);
         BufferedWriter bw = new BufferedWriter(fw);
         printReverseNodes(doc, bw);
      } catch (IllegalArgumentException iae) {
         System.out.println("Illegal argument : " + iae.getMessage());
      } catch (IOException ioe) {
         System.out.println("IO exception: " + ioe.getMessage());
      }
}
```

Setup for Print Reverse (5)

```
public void printReverseNodes(Node node, Writer writer)  {
    int iter;
    indent +=4;
    switch (node.getNodeType()) {
    case Node.DOCUMENT_NODE:
        break;

    case Node.ELEMENT_NODE:
        break;

    case Node.TEXT_NODE:
        break;

    case Node.CDATA_SECTION_NODE:
        break;
```

Setup for Print Reverse (6)

```
            case Node.COMMENT_NODE:
                break;

            case Node.PROCESSING_INSTRUCTION_NODE:
                break;

            case Node.ENTITY_REFERENCE_NODE:
                break;

            case Node.DOCUMENT_TYPE_NODE:
                break;

            indent = indent - 4;
        }
    }
```

Printing Out the Document Level Information

- Not everything should be reversed
  - Just the body of the document
  - Therefore, just print the document and processing information first
- Cast the document to a node
- Call the **printReverseNodes()** routine recursively

```
case Node.DOCUMENT_NODE:
    writer.write("<?xml version=\"1.0\"?>\n");
    Document doc = (Document)node;
    indent = -4;  // start at the front
    printReverseNodes(doc.getDocumentElement(), writer);
    break;
```

Printing Elements

- Most common task
- Elements have names, attributes, and children
  - getNodeName()
  - getAttributes()
  - getChildNodes()
- Children are represented as a node list
- Need to iterate through list of children to process children
- Need to iterate through list of attributes to process attributes

Printing Elements (2)

```
case Node.ELEMENT_NODE:
    String eltName = node.getNodeName();
    writeIndent(writer);

    // Write element
    writer.write("<" + eltName);
    // recurse on attributes - reverse their orders as well
    NamedNodeMap attributes = node.getAttributes();
    for (iter=attributes.getLength()-1; iter>=0; iter--) {
        Node attribute = attributes.item(iter);
        writer.write(" " +  attribute.getNodeName() +
                     "=\"" + attribute.getNodeValue() + "\"");
    }
    writer.write(">\n");
```

Printing Elements (3)

```
// Recurse on children elements
NodeList children = node.getChildNodes();

if (children != null) {
      // Write a line break
      if (children.item(0) != null &&
         children.item(children.getLength()-1).getNodeType() ==
         Node.ELEMENT_NODE) {
         writer.write("\n");
      }
      // Recurse in reverse on the children
      for (iter=children.getLength()-1; iter>=0; iter--) {
          printReverseNodes(children.item(iter), writer) ;
      }
```

Printing Elements (4)

```
        // Write another line break at the end
        if (children.item(0) != null &&
            children.item(0).getNodeType() ==
            Node.ELEMENT_NODE) {
            writer.write("\n");
        }
}

//Finish up the element
writeIndent(writer);
writer.write("</" + eltName + ">\n");
break;
```

Writing Data and Comments

```
case Node.TEXT_NODE:
    writer.write(node.getNodeValue());
    break;

case Node.CDATA_SECTION_NODE:
    writer.write("<![CDATA[" + node.getNodeValue() + "]]>");
    break;

case Node.COMMENT_NODE:
    writer.write("\n");
    writeIndent(writer);
    writer.write("<!-- " + node.getNodeValue() + " -->\n");
    break;
```

Processing Instructions

```
// Processing instructions are divided into names and values.
// The name is the processing instruction name.
// The value is a string which contains the list of data values.
// If you want to process these (e.g., reverse them), you need to
// parse and process the value string.

case Node.PROCESSING_INSTRUCTION_NODE:
    writer.write("\n");
    writeIndent(writer);
    writer.write("<?" + node.getNodeName() + " " +
                 node.getNodeValue() + "?>\n");
    break;
```

DOCTYPES

```
case Node.DOCUMENT_TYPE_NODE:
    DocumentType docType = (DocumentType)node;
    writer.write("\n");
    writeIndent(writer);
    writer.write("<!DOCTYPE " + docType.getName());
    if (docType.getPublicId() != null) {
        writer.write("PUBLIC \"" + docType.getPublicId() +
            "\"");
    } else {
        writer.write("SYSTEM ");
    }
    writer.write("\""+ docType.getSystemId() + "\">");
    writer.write("\n");
    break;
```

Entity Reference Nodes

```
        // Entity references are like #DEFINEs in C or C++
        // DOM parsers do not always pass them through -- they may
        // be preprocessed

        case Node.ENTITY_REFERENCE_NODE:
            writer.write("\n");
            writeIndent(writer);
            writer.write("&" + node.getNodeName() + ";");
            break;
    }
```

Write Indent

```
public void writeIndent(Writer writer)  {
        StringBuffer sb = new StringBuffer();
        for (int j = 0; j < indent; j++)
                sb.append(" ");
        writer.write(sb.toString());
}
```

Printing Processing Instructions

- We want to reverse the elements of the document
- The call to **doc.getDocumentElement()** actually retrieves processing the document root element
- However, the PI's and DOCTYPE's are at the same level as the root object.
- Therefore, they will not be printed out
- Furthermore, one doesn't want these reversed
- PI's and DOCTYPE's need to remain at the top of the document

Ensure that the Document Element is Last

```
case Node.DOCUMENT_NODE:
    writer.write("<?xml version=\"1.0\"?>\n");
    // recurse on highest children
    NodeList nodes = node.getChildNodes();
    if (nodes != null) {
        for (iter = 0; iter < nodes.getLength()-1; iter++) {
            indent = -4;
            printReverseNodes(nodes.item(iter), writer);
        }
    }
    Document doc = (Document)nodes.item(nodes.getLength()-1);
    indent = -4;
    printReverseNodes(doc.getDocumentElement(), writer);
    break;
```

SAX Exceptions

- DOM is a standard interface
- Nothing standardizes the internal structure of the DOM tree
- Often, DOM parsers use SAX
  - Parses document
  - Places results in DOM tree
- This means that SAX exceptions may be thrown
  - Need to catch SAX exceptions
  - Need to import **org.xml.sax.SAXException** in code

SAX Exception Code

```
pubic void writeReverse(String uri, String fileName) {
        System.out.println("Parsing XML File: " + uri + "\n");
        System.out.println("Writing reverse XML File: " + fileName + "\n");
        DOMParser parser = new DOMParser();
        try {
                parser.parse(uri);
                Document doc = parser.getDocument();
                printReverse(doc, fileName);
        } catch (Exception ex) {
                System.out.println(ex.getMessage());
        } catch (SAXException saxe) {
                System.out.println(saxe.getMessage());
        }
}
```

Τέλος Ενότητας