University of Glasgow

Processing XML

Web Application Development 2

PROCESSING XML
DOM PARSING
SAX PARSING
SAX VS. DOM

XML Structure

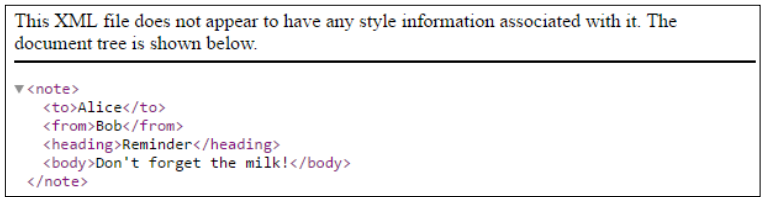
- XML has a tightly controlled structure; documents must follow a number of rules to be *well-formed*
- Case sensitive <**start_end**>tags</**start_end**>
- Obeys a **hierarchical structure**
- All documents have a **root** element
- Might also be DTD/Schema rules to validate against

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
<food>
   <name>Belgian Waffles</name>
   <price>$5.95</price>
   <description>
      Two of our famous Belgian Waffles with plenty of real maple syrup
   </description>
   <calories>650</calories>
</food>
…
</breakfast_menu>
```

Displaying XML

- Valid XML can be displayed in a browser:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
   <to>Alice</to>
   <from>Bob</from>
   <heading>Reminder</heading>
   <body>Don't forget the milk!</body>
</note>
```



Displaying XML

- Browser might give an error in the case of invalid XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Alice</to>
  <from>Bob</Ffrom>
  <heading>Reminder</heading>
  <body>Don't forget the milk!</body>
</note>
```

Programming and XML

There are two main ways of using XML in a program:

- **DOM (The Document Object Model)**
  - builds an **in-memory hierarchical model** of the XML elements
  - appropriate if you need the **whole document** or need to **move about it freely**
- **SAX (The Simple API for XML)**
  - provides an **event driven parser** for XML
  - appropriate for using **parts of the data** in the order they appear in the file, or if there are memory constraints

Definition: Document Object Model

- W3C standardised for accessing documents
  *"The W3C **Document Object Model** is a **platform** and **language-neutral interface** that allows programs and scripts to **dynamically access** and **update** the **content**, **structure**, and **style** of a **document**."*
- DOM is separated in three main parts
  - Core DOM: standard model for any structured doc
  - HTML DOM: standard model for HTML docs
  - XML DOM: standard model for XML docs

XML DOM

- Is a standard object model and programming interface for XML
- It defines **objects** and **properties** of all XML elements along with the **methods** to access them
  - **It is the standard for getting, changing, adding, and deleting XML elements**
- **DOM defines everything in an XML document as a node**
- Yes, everything is a node

XML DOM Nodes

- The XML document is a document node
- Every XML element within the document is an element node
- The root element of the document is the root node
- Even the text of XML elements is a node

```
<ysearchresponse xmlns="https://www.google.com/" >
…
      <resultset_web count="10" start="1" ....>
            <result>
                  <title>University of Glasgow</title>
                  <abstract>The University of Glasgow, Scotland, UK. The University of
Glasgow is a major research-led university operating in….
                  </abstract>
                  <url>https://www.gla.ac.uk/</url>
            </result>
      ….
```

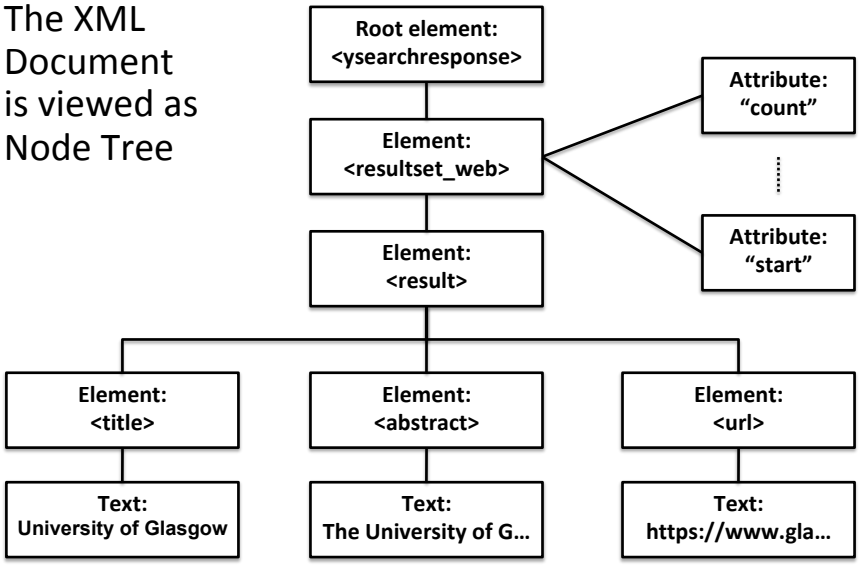
Root Node

Text Node

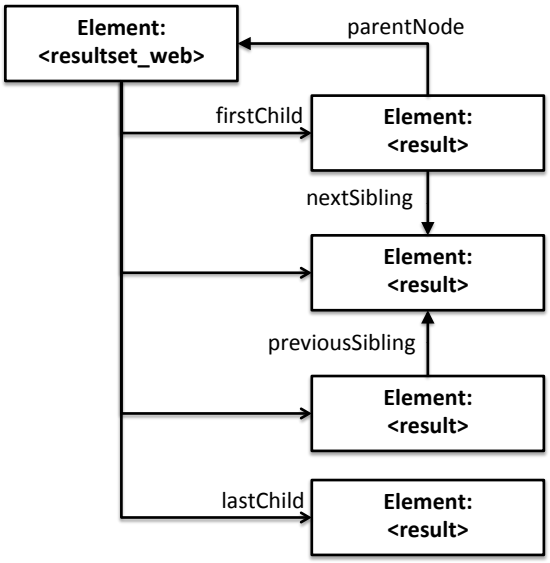
Element Node

XML DOM Node Tree

The XML Document is viewed as Node Tree



Parents, Children, Siblings



- <result> nodes are childNodes of <resultset_web>
- And siblings to each other
- An advantage of a tree structure is that is can be traversed without knowing the exact structure and without knowing the type of data it houses

Working with DOM

- In whichever language or environment you are working, the technique is basically the same:
  1. load the XML document object
  2. locate the root element or some other element that is of interest
     - either traverse the tree
     - or search for the desired element
  3. for the given element
     - extract the attributes and their values
     - extract the element data
     - and/or add/modify/remove elements or attributes
  4. Go to step 2 and repeat until all processing is done

Example of DOM Parsing

```
<html>
  <body>

    <p id="demo"></p>

    <script>
        var text, parser, xmlDoc;

        text = "<bookstore><book>" +
        "<title>Everyday Italian</title>" +
        "<author>Giada De Laurentiis</author>" +
        "<year>2005</year>" +
        "</book></bookstore>";
        parser = new DOMParser();

        xmlDoc = parser.parseFromString(text,"text/xml");

        document.getElementById("demo").innerHTML
           = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
    </script>
  </body>
</html>
```

- Points to note here:
  - Creation of DOMParser object
  - Parsing of XML into xmlDoc object
  - Traversal of DOM node tree
- In Ajax examples, didn't need to parse first; obtained DOM via xhttp.responseXML

Processing XML example

cd_catalog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
  </CD>
…
```

DOM Parsing using AJAX

(partial code from XMLDemo1 – these examples are on Moodle. Seen this one before)

```
var xhttp, xmlDoc, txt, x, i;
xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        xmlDoc = this.responseXML;
        txt = "";
        x = xmlDoc.getElementsByTagName("ARTIST");
        for (i = 0; i < x.length; i++) {
            txt = txt + x[i].childNodes[0].nodeValue + "<br>";
        }
        document.getElementById("demo").innerHTML = txt;
    }
};
xhttp.open("GET", "cd_catalog.xml", true);
xhttp.send();
```

DOM Parsing using AJAX – 2

(partial code from XMLDemo2)

```
...
<table id="demo"></table>
...

function myFunction(xhttp) {
    var i;
    var xmlDoc = xhttp.responseXML;
    var table="<tr><th>Artist</th><th>Title</th></tr>";
    var x = xmlDoc.getElementsByTagName("CD");
    for (i = 0; i <x.length; i++) {
        table += "<tr><td>" +
              x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue
              + "</td><td>" +
              x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue
              + "</td></tr>";
    }
    document.getElementById("demo").innerHTML = table;
}
```

DOM Parsing using AJAX – 3

(partial code from XMLDemo3)

```
...
<table id="cd-table"></table>
...

    xmlDoc = xmlhttp.responseXML;
    x = xmlDoc.getElementsByTagName("CD");
    table="<tr><th>Artist</th><th>Title</th></tr>";
    for (i = 0; i <x.length; i++) {
        table += "<tr onclick='displayCD(" + i + ")'><td>";
        table += x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue;
        table += "</td><td>";
        table +=  x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue;
        table += "</td></tr>";
    }
    document.getElementById("demo").innerHTML = table;
```

DOM Parsing using AJAX – 3

(more partial code from XMLDemo3)

```
function displayCD(i) {
    document.getElementById("showCD").innerHTML =
    "Artist: " +
    x[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "<br>Title: " +
    x[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "<br>Year: " +
    x[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue +
    "<br>Company: " +
    x[i].getElementsByTagName("COMPANY")[0].childNodes[0].nodeValue +
    "<br>Price: &#163;" +
    x[i].getElementsByTagName("PRICE")[0].childNodes[0].nodeValue;
}
```

Traversing the XML DOM Tree (1)

```
<html>
  <body>
     <p id="demo"></p>
     <script>
        var xhttp = new XMLHttpRequest();
        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                 myFunction(this);
            }
        };
        xhttp.open("GET", "XML_example.xml", true);
        xhttp.send();

        function myFunction(xhttp) {
            var xmlDoc = xhttp.responseXML;
            document.getElementById("demo").innerHTML = traverse(xmlDoc);
        }
```

Traversing the XML DOM Tree (2)

```
function traverse(x) {
    var txt = "";
    if (x.nodeType == 3) { /*Node.TEXT_NODE*/
         if (x.nodeValue.trim().length>0) {
              txt += "Text node: "+x.nodeValue+"<br>";
         }
    } else if (x.nodeType == 8) { /*Node.COMMENT_NODE*/
         txt += "Comment node: "+x.nodeValue+"<br>";
    } else {
         if (x.nodeType == 1) { /*Node.ELEMENT_NODE*/
              txt += "Element node: "+x.nodeName+"<br>";
              var attr = x.attributes; /* Look at attributes */
              for (var i = 0; i<attr.length; i++) {
                   txt += "Attribute node: name="+attr[i].nodeName+",
                                value="+attr[i].nodeValue+"<br>";
              }
         } else if (x.nodeType == 9) { /*Node.DOCUMENT_NODE*/
              txt += "Document node: "+x.nodeName+"<br>";
         }
```

Traversing the XML DOM Tree (3)

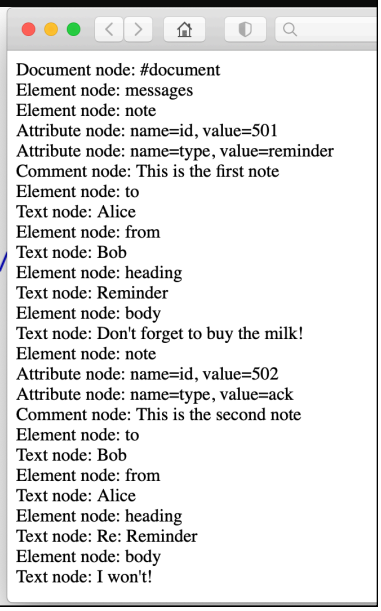
```
                var kids = x.childNodes;
                for (var i = 0; i < kids.length; i++) {
                    txt += traverse(kids[i]);
                }
            }
            return txt;
        }
    </script>
</body>
</html>
```

XML notes

```
<?xml version="1.0" encoding="UTF-8"?>
<messages>
  <note id="501" type="reminder">
    <!-- This is the first note -->
    <to>Alice</to>
    <from>Bob</from>
    <heading>Reminder</heading>
    <body>Don't forget to buy the milk!</body>
  </note>
  <note id="502" type="ack">
    <!-- This is the second note -->
    <to>Bob</to>
    <from>Alice</from>
    <heading>Re: Reminder</heading>
    <body>I won't!</body>
  </note>
</messages>
```

XML notes

```xml
<?xml version="1.0" encoding="UTF-8"?>
<messages>
  <note id="501" type="reminder">
    <!-- This is the first note -->
    <to>Alice</to>
    <from>Bob</from>
    <heading>Reminder</heading>
    <body>Don't forget to buy the milk!</
  </note>
  <note id="502" type="ack">
    <!-- This is the second note -->
    <to>Bob</to>
    <from>Alice</from>
    <heading>Re: Reminder</heading>
    <body>I won't!</body>
  </note>
</messages>
```



Employees.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Employees>
    <Employee id="1">
        <age>29</age>
        <name>Pankaj</name>
        <gender>Male</gender>
        <role>Java Developer</role>
    </Employee>
    <Employee id="2">
        <age>35</age>
        <name>Lisa</name>
        <gender>Female</gender>
        <role>CEO</role>
    </Employee>
    <Employee id="3">
        <age>40</age>
        <name>Tom</name>
        <gender>Male</gender>
        <role>Manager</role>
    </Employee>
…
</Employees>
```

DOM Parsing using Java (1)

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

public class XMLParserDOM {

  public static void main(String argv[]) {
    try {

        File fXmlFile = new File("employees.xml");
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(fXmlFile);

        System.out.println("Root element: " + doc.getDocumentElement(). getNodeName());
        System.out.println("-----------------------");
```

Key imports

Opening and parsing the file

DOM Parsing using Java (2)

```
    NodeList nList = doc.getElementsByTagName("Employee");

    for (int temp = 0; temp < nList.getLength(); temp++) {
       Node nNode = nList.item(temp);
       System.out.println("\nCurrent Element :" + nNode.getNodeName());
       if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                // in fact all items in the list must satisfy this

           Element eElement = (Element) nNode;
           System.out.println("Employee id : " + eElement.getAttribute("id"));
           System.out.println("Name : " + eElement.getElementsByTagName("name")
                   .item(0).getTextContent());
           System.out.println("Age : " + eElement.getElementsByTagName("age")
                   .item(0).getTextContent());
           System.out.println("Gender : " + eElement.getElementsByTagName("gender")
                   .item(0).getTextContent());
           System.out.println("Role : " + eElement.getElementsByTagName("role")
                   .item(0).getTextContent());

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

Extracting information from the DOM tree

DOM Parsing using Java (3)

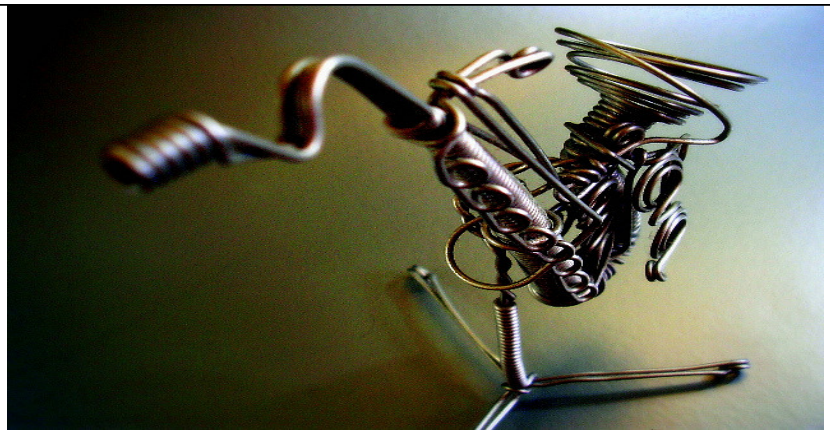
Output:

```
Root element: Employees
----------------------

Current Element :Employee
Employee id : 1
Name : Pankaj
Age : 29
Gender : Male
Role : Java Developer

Current Element :Employee
Employee id : 2
Name : Lisa
Age : 35
Gender : Female
Role : CEO

Current Element :Employee
Employee id : 3
...
```



SAX PARSING

SAX: Simple API for XML

- SAX is a sequential access parser API for XML
- It is not an alternative to DOM
  - there is no default object model
  - but another mechanism for reading XML
- It is a **stream parser** which is **event-driven**
  - **Parsing is unidirectional, i.e., there is no going back**
  - **Callback methods** are **triggered** by **events** when parsing
- Is oriented towards **state independent processing**
  - An alternative to SAX is **StAX** which is oriented to state-dependent processing

Event Handling in SAX

- Events are available for the following XML features:
  - XML text nodes
  - XML element nodes
  - XML comments
  - XML processing instructions (often used in XPath and XQuery)
- Events are triggered when:
  - Open or close element tags are encountered
  - Data (#PCDATA and CDATA) sections are encountered
  - Processing instructions, comments, etc. are encountered

Working with SAX

- The three steps to using SAX in your programs are:
  1. **Creating a custom object model**
     - like ResultSet and Result
  2. **Creating a SAX parser**
  3. **Creating a DocumentHandler** to turn the XML document into instances of your custom object model
     - ContentHandler: implements the main SAX interface for handling document events
     - DTDHandler: for handling DTD events
     - EntityResolver: for resolving external entities
     - ErrorHandler: for reporting errors and warning
     - DefaultHandler: for everything else

SAX Handler Methods

- *startDocument*
  – performs any work required before parsing
- *endDocument*
  – performs any work required at the end of the parsing – e.g., reporting analytical results
- *startElement*(name, attributes)
  – perform any work required when the start tag of an element of that name is encountered
- *endElement*(name)
  – perform any work required when the end tag of an element of that name is encountered
- *characters*(ch)
  – perform any work required when a text node is encountered

Employee.java

```
public class Employee {
    private int id;
    private String name;
…

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getGender() {
        return gender;
    }
…

@Override
    public String toString() {
        return "Employee: ID="+this.id+" Name=" + this.name + " Age=" + this.age +
             " Gender=" + this.gender + " Role=" + this.role;
    }
}
```

XMLParserSAX.java

```
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;

public class XMLParserSAX {

    public static void main(String[] args) {
        SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
        try {
            SAXParser saxParser = saxParserFactory.newSAXParser();
            MyHandler handler = new MyHandler();
            saxParser.parse(new File("employees.xml"), handler);
        } catch (ParserConfigurationException | SAXException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

MyHandler.java (1)

```
public class MyHandler extends DefaultHandler {

    //List to hold Employees object
    private ArrayList<Employee> empList = null;
    private Employee emp = null;

    boolean bAge = false;
    boolean bName = false;
    boolean bGender = false;
    boolean bRole = false;

    @Override
    public void startDocument() {
       empList = new ArrayList<>();
    }
```

MyHandler.java (2)

```
@Override
public void startElement(String uri, String localName, String qName, Attributes attributes)
        throws SAXException {
    if (qName.equalsIgnoreCase("Employee")) {
    //initialize Employee object and set id attribute
        emp = new Employee();
    String id = attributes.getValue("id");
        emp.setId(Integer.parseInt(id));
    } else if (qName.equalsIgnoreCase("name")) {
        //set boolean values for fields, will be used in setting Employee variables
        bName = true;
    } else if (qName.equalsIgnoreCase("age")) {
        bAge = true;
    } else if (qName.equalsIgnoreCase("gender")) {
        bGender = true;
    } else if (qName.equalsIgnoreCase("role")) {
        bRole = true;
    }
}
```

MyHandler.java (3)

```
@Override
public void characters(char ch[], int start, int length) throws SAXException {
    if (bAge) {
        //age element, set Employee age
        emp.setAge(Integer.parseInt(new String(ch, start, length)));
        bAge = false;
    } else if (bName) {
        emp.setName(new String(ch, start, length));
        bName = false;
    } else if (bRole) {
        emp.setRole(new String(ch, start, length));
        bRole = false;
    } else if (bGender) {
        emp.setGender(new String(ch, start, length));
        bGender = false;
    }
}
```

MyHandler.java (4)

```
@Override
public void endElement(String uri, String localName, String qName) throws SAXException {
    if (qName.equalsIgnoreCase("Employee")) {
        //add Employee object to list
        empList.add(emp);
    }
}

@Override
public void endDocument() {
    for(Employee emp : empList)
        System.out.println(emp);
    }
}
```

Output:

```
java XMLParserSAX
Employee: ID=1 Name=Pankaj Age=29 Gender=Male Role=Java Developer
Employee: ID=2 Name=Lisa Age=35 Gender=Female Role=CEO
Employee: ID=3 Name=Tom Age=40 Gender=Male Role=Manager
Employee: ID=4 Name=Meghna Age=25 Gender=Female Role=Manager
```

DOM versus SAX

- Uses more memory ☹
- Tends to be slower ☹
- Can handle parsing that requires access to the entire document ☺
  - (if it fits in memory)
- Easier to program ☺
- Can process files larger than main memory through disk caching ☹
  - but this is even slower!

- Uses less memory ☺
- Tends to be faster ☺
- Can process files that are larger than main memory ☺
- Requires more programmer effort ☹
- Cannot handle all parsing tasks directly, i.e., if all XML is required for validation ☹
  - would need multiple parses