6

XML

6.4 DOM

The XML 'Alphabet Soup'

✓ **XML**	Extensible Markup Language	Defines XML documents
✓ **XSL**	Extensible Stylesheet Language	Language for expressing stylesheets; consists of XSLT and XSL-FO
✓ **XSLT**	XSL Transformations	Language for transforming XML documents
✓ **XSL-FO**	XSL Formatting Objects	Language to describe precise layout of text on a page
✓ **Data Island**	XML data embedded in a HTML page	
✓ **Data Binding**	Automatic population of HTML elements from XML data	
✓ **Namespace**	A collection of names, identified by a URI reference, which are used in XML documents	

The XML 'Alphabet Soup'

✓

DTD	Document Type Definition	Non-XML schema
DOM	Document Object Model	API to read, create and edit XML documents; creates in-memory object model
SAX	Simple API for XML	API to parse XML documents; event-driven
XML Schema (XSD)	XML Schema Definition	an XML based alternative to DTD
XPath	XML Path Language	A language for addressing parts of an XML document, designed to be used by both XSLT and XPointer
XPointer	XML Pointer Language	Supports addressing into the internal structures of XML documents
XLink	XML Linking Language	Describes links between XML documents
XQuery	XML Query Language (draft)	Flexible mechanism for querying XML data as if it were a database

The XML 'Alphabet Soup'

SOAP	Simple Object Access Protocol	A simple XML based protocol to let applications exchange information over HTTP
WSDL	Web Services Description Language	An XML-based language for describing Web services and how to access them
WAP	Wireless Application Protocol	The leading standard for information services on wireless terminals like digital mobile phones
WML	Wireless Markup Language	WAP uses the mark-up language WML (not HTML)

SAX and DOM

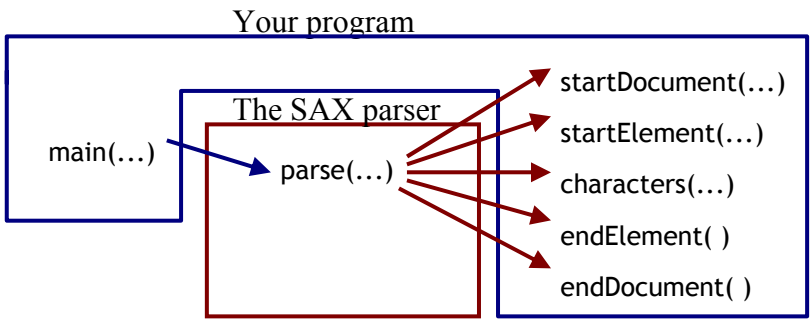
- SAX and DOM are standards for XML parsers - program APIs to read and interpret XML files
  - DOM is a W3C standard
  - SAX is an ad-hoc (but very popular) standard
- There are various implementations available
- Java implementations are provided in JAXP (Java API for XML Processing)
- JAXP is included as a package in Java 1.4
  - JAXP is available separately for Java 1.3
- Unlike many XML technologies, SAX and DOM are relatively easy

Difference between SAX and DOM

- DOM reads the entire XML document into memory and stores it as a tree data structure
- SAX reads the XML document and sends an event for each element that it encounters
- Consequences:
  - DOM provides “random access” into the XML document
  - SAX provides only sequential access to the XML document
  - DOM is slow and requires huge amounts of memory, so it cannot be used for large XML documents
  - SAX is fast and requires very little memory, so it can be used for huge documents (or large numbers of documents)
    - This makes SAX much more popular for web sites
  - Some DOM implementations have methods for changing the XML document in memory; SAX implementations do not

SAX Callbacks

- SAX works through callbacks: you call the parser, it calls methods that you supply



What is the DOM?

- The Document Object Model (DOM) provides a standard programming interface to a wide variety of applications. The XML DOM is designed to be used with any programming language and any operating system.
  - It is fully described in the W3C DOM specification
    - http://www.w3.org/DOM/
- With the XML DOM, a programmer can create an XML document, navigate its structure, and add, modify, or delete its elements
- DOM provides generic access to DOM-compliant documents: add, edit, delete, manipulate
- DOM is language-independent
- The DOM is based on a tree view of your document. Nodes! Nodes! Nodes!
- DOM useful for CSS, HTML, XML
- DOM + client-side scripting + HTML = DHTML

DOM components

- `Document` top-level view of the document, with access to all nodes (including root element)
  - `createElement` method - creates an element node
  - `createAttribute` method - creates an attribute node
  - `createComment` method - creates a comment node
  - `getDocumentElement` method - returns root element
  - `appendChild` method - appends a child node
  - `getChildNodes` method - returns child nodes

DOM components II

- `Node` represents a node - "A node is a reference to an element, its attributes, or text from the document."
  - `cloneNode` method - duplicates a node
  - `getNodeName` method - returns the node name
  - `getNodeName` method - returns the node's name
  - `getNodeType` method - returns the node's type
  - `getNodeValue` method - returns the node's value
  - `getParentNode` method - returns the node's parent's name
  - `hasChildNodes` method - true if has child nodes
  - `insertBefore` method - stuffs child in before specified child
  - `removeChild` method - removes the child node
  - `replaceChild` method - replaces one child with another
  - `setNodeValue` method - sets node's value

DOM components III

- `attribute` represents an attribute node -
  - `getAttribute` method - gets attribute!
  - `getTagName` method - gets element's name
  - `removeAttribute` method - deletes it
  - `setAttribute` method - sets att's value

Parsing the DOM

- To read and update - create and manipulate - an XML document, you need an XML parser.
- The Microsoft XMLDOM parser features a programming model that:
  - Supports JavaScript, VBScript, Perl, VB, Java, C++ and more
  - A COM component that comes with Microsoft Internet Explorer 5.0
  - Supports W3C XML 1.0 and XML DOM
  - Supports DTD and validation

Creating an XML document object

- JavaScript:
  - `var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")`
- VBScript:
  - `set xmlDoc = CreateObject("Microsoft.XMLDOM")`
- .asp:
  - `set xmlDoc = Server.CreateObject("Microsoft.XMLDOM")`

Adding in a new element

- ```
  var link = document.createElement('a');
  link.setAttribute('href', 'mypage.htm');
  ```
  ```

locating a slot in the document

- by location:
  - `document.childNodes[1].childNodes[0]`
  - Find the main document element (HTML), and find its second child (BODY), then look for its first child (DIV)
- by ID:
  - `document.getElementById('myDiv').appendChild(txt);`

Hiding an element

- ```
  document.childNodes[1].childNodes[1].childNodes[0]
  .style.display = "none";
  ```
  ```

Loading an XML document object into the parser

- ```
  <script language="JavaScript">

  var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
  xmlDoc.async="false"
  xmlDoc.load("note.xml")

  // ....... processing the document goes here

  </script>
  ```

Manually loading XML into the parser

- ```
  <script language="JavaScript">
  // load up variable var with some xml
  var text="<note>"
  text=text+"<to>John</to><from>Robert</from>"
  text=text+"<heading>Reminder</heading>"
  text=text+"<body>Don't forget your homework!</body>"
  text=text+"</note>"  // now create the DO

  var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
  xmlDoc.async="false"
  xmlDoc.loadXML(text)
  // ..... process the document
  </script>
  ```

parseError object

- document.write(xmlDoc.parseError.*property*)
  - errorCode: Returns a long integer error code
  - reason: Returns a string explaining the reason for the error
  - line: Returns a long integer representing the line number for the error
  - linePos: Returns a long integer representing the line position for the error
  - srcText: Returns a string containing the line that caused the error
  - url: Returns the url pointing the loaded document
  - filePos: Returns a long integer file position of the error

Traversing nodes

```
set xmlDoc=CreateObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")

for each x in xmlDoc.documentElement.childNodes
      document.write(x.nodename)
      document.write(": ")
      document.write(x.text)
next
```

Calling XML nodes by name

```
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")

document.write(xmlDoc.getElementsByTagName("from").item(0).text)
```

References

- W3School DOM Tutorial
  - http://www.w3schools.com/dom/default.asp
- MSXML 4.0 SDK

Reading List

- W3School DOM Tutorial
  - http://www.w3schools.com/dom/default.asp