UNIT-III: XML(extensible markup language)

CONTENTS

XML

The markup language developed to add structural and formatting information to data and which was designed to be simple enough to be included in any application that language is Standard Generalized Markup Language and was adopted as standard by International Organization for Standardization(ISO).

Markup is nothing but instructions, which are often called as tags. There are many languages which shows how the data is displayed but no one describes what the data is.This is the point at which XML enters. XML is a subset of SGML. XML is used to describe the structure of a document not the way that is presented. XML is the recommendation of World Wide Consortium (W3C). The structure of basic XML is

shown below which resembles HTML. The first line is the processing instruction which tells applications how to handle the XML. It is also serves as version declaration and says that the file is XML.

Example Sample XML program

```
<?xml version="1.0"?>
<college>
<studdetail>
        <regno>05j0a1260</regno>
        <name>
        <firstname>karthik></firstname>
        <lastname>btech</lastname>
        </name>
        <country name="india"/>
        <branch>csit</branch>
</studdetail>
</college>
```

Valid an Well Formed XML

XML documents may be either valid or well formed. A well formed document is one which follows all of the rules of XML. Tags are matched and do not overlap, empty elements are ended properly, and the document contains an XML declaration. A valid XML document has its own DTD. XML should also conforms the rules set out in the DTD. There are many XML parsers that checks the document and its DTD

XML elements

XML documents are composed of three things i.e., elements, control information, and entities. Most of the markup in an XML document is element markup. Elements are surrounded by tags much as they are in HTML. Each document has a single root element which contains al of the other markup.

Nesting tags: Even the simplest XML document has nested tags. Unlike HTML these must be properly nested and closed in the reverse of the order in which they were opened. Each XML tag has to have a closing tag, again unlike HTML.

Case Sensitive: XML is case sensitive and you must use lower case for your markup.

Empty tags: Some tags are empty, they don't have content. Where the content of the tag is missing, it appears as <content/>

Attributes: Sometimes it is important that elements have information associated with them without that information becoming a separate element.

Control Information:

There are three types of control information

- Comments
- processing instructions

- document type declaration.

Comments: XML comments are exactly same as HTML. They take the form as
<!- -comment text - ->

Processing Instructions: Processing Instructions are used to control applications. One of the processing instructions is **<?xml version=”1.0”>**

Document Type Declarations: Each XML document has an associated DTD. The DTD is usually present in separate file, so that it can be used by many files. The statement that includes DTD in XML file is **<?DOCTYPE cust SYSTEM “customer.dtd”>**

Entities

Entities are used to create small pieces of data which you want to use repeatedly throughout your schema.

Example A Complete XML program

```
<?xml version=”1.0”?>
<!DOCTYPE stud S?YSTEM “student.dtd”>
<college>
<studdetail>
<regno>05j0a1260</regno>
<name>
<firstname>feroz></firstname>
<lastname>btech</lastname>
</name>
<country name="india"/>
<branch>csit</branch>
</studdetail>
</college>
```

Document Type Definition

Document type definition have been successfully used in SGML applications for many year. DTD are document centric. They are well understood. There are plenty of tools that support DTD.

```
<!ELEMENT college(studetail+)>
<!ELEMENT studetail(regno, name+, country, branch)>
<!ELEMENT regno(#PCDATA)>
<!ELEMENT name(firstname, lastname)>
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT lastname(#PCDATA)>
<!ELEMENT country(#PCDATA)>
<!ATTLIST country name CDATA #REQUIRED>
<!ELEMENT branch(#PCDATA)>
```

XML Schema

W3C developed a technology called XML schema which they accepted as a recommendation. XML schema is itself an XML application which means when you use it your only need a single grammar and can use your normal XML editor to create it.

Example XML Schema for XML document shown in Example

```
<?xml version =”1.0” ?>
<xsd:schema xmlns =” http://.........”>
<xsd:element name = “college”>
<xsd:complexType>
<xsd:sequence>
```

```
<xsd:element name = “studetail”>
<xsd:complexType>
<xsd:sequence>
<xsd:element name = “regno” type = “xsd:string”/>
<xsd:element name = “name”>
<xsd:complexType>
<xsd:sequence>
<xsd:element name = “firstname” type=”xsd:string”/>
<xsd:element name= “lastname” type = “xsd:string”/>
<xsd:element name = “country”>
<xsd:complexType>
<xsd:attribute name = “India” type= “xsd:string”/>
</xsd:complexType>
</xsd:element>
xsd:element name = “branch” type = “xsd:string”/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Document Object Model

XML parsers can handle documents in any way that their developers choose. There are two models commonly used for parsers i.e., SAX and DOM. SAX parsers are used when dealing with streams of data. This type of parsers are usually used with java.

- SAX-based parsers run quickly.
- DOM is and application program interface (API) for XML documents.

The DOM API specifies the logical structure of XML documents and the ways in which they can be accessed and manipulated. The DOM API is just a specification. DOM-complaint applications include all of the functionality needed to handle XML documents. They can build static documents, navigate and search through them, add new elements, delete elements, and modify the content of existing elements. The views XML document as trees. The DOM exposes the whole of the document to applications. It is also scriptable so applications can manipulate the individual nodes.

Presenting XML

XML documents are presented using Extensible Stylesheet which expresses stylesheets. XSL stylesheet are not the same as HTML cascading stylesheets. They create a style for a specific XML element, with XSL a template is created. XSL basically transforms one data structure to another i.e., XML to HTML.

Example Here is the XSL file for the XML document of Example

This line must be included in the XML document which reference stylesheet <?xml:stylesheet type = “text/xsl” href = “student.xsl”?.
Here goes the XSL file

```
<xsl:stylesheet smlns:xsl =”uri:xsl”.
<xsl:template match=”/”>
<html>
<body>
<h1> Student Database </h1.
<xsl:for-each select = “college”>
<xsl:for-each select = “studetail”>
<xsl:value-of select = “regno”/>
<xsl:for-each select = “name”>
<xsl:value-of select = “firstname”/>
<xsl:value-of select = “lastname”/>
</xsl:for-each>
<xsl:value-of select=”country/@name” />
<xsl:value-of select = “branch”/>
</xsl:for-each>
</xsl:for-each>
</body>
</xsl:template>
</xsl:stylesheet>
```

Evolution of the XML Parsing

The combination of Java and XML has been one of the most attracting things which had happened in the field of software development in the 21st century. It has been mainly for two reasons - Java, arguably the most widely used programming language and XML, almost unarguably the best mechanism of data description and transfer.

Since these two were different technologies and hence it initially required a developer to have a sound understanding of both of these before he can make the best use of the combination. Since then there have been a paradigm shift towards Java a few interesting technologies getting evolved to make this happen. Some of them are:-

SAX - Simple API for XML Parsing

It was the first to come on the scene and interestingly it was developed in the XML-Dev maling list. Evidently the people who developed this were XML gurus and it is quite visible in the usage of this API. You got to have a fair understanding of XML, but at least Java developers got something to combine the two worlds - Java and XML in a structured way. It instantly became a hit for the obvious reasons.

Since this API does require to load the entire XML doc and also because it offers only a sequential processing of the doc hence it is quite fast. Another reason of it being faster is that it does not allow modification of the underlying XML data.

DOM - Document Object Model

The Java binding for DOM provided a tree-based representation of the XML documents - allowing random access and modification of the underlying XML data. Not very difficult to deduce that it would be slower as compared to SAX.
The event-based callback methodology was replaced by an object-oriented in-memory representation of the XML documents. Though, it differs from one implementation to another if the entire document or a part of it would be

kept in the memory at a particular instant, but the Java developers are kept out of all the hassle and they get the entire tree readily available whenever they wish.

JAXP - Java API for XML Parsing

The creators and designers of Java realized that the Java developers should not be XML gurus to use the XML in Java applications. The first step towards making this possible was the evolution of JAXP, which made it easier to obtain either a DOM Document or a SAX-compliant parser via a factory class. This reduced the dependence of Java developers over the numerous vendors supplying the parsers of either type. Additionally, JAXP made sure that an interchange between the parsers required minimal code changes.

Differences between DOM and SAX

SAX v/s DOM

Main differences between SAX and DOM, which are the two most popular APIs for processing XML documents in Java, are:-

- **Read v/s Read/Write:** SAX can be used only for reading XML documents and not for the manipulation of the underlying XML data whereas DOM can be used for both read and write of the data in an XML document.
- **Sequential Access v/s Random Access:** SAX can be used only for a sequential processing of an XML document whereas DOM can be used for a random processing of XML docs. So what to do if you want a random access to the underlying XML data while using SAX? You got to store and manage that information so that you can retrieve it when you need.
- **Call back v/s Tree:** SAX uses call back mechanism and uses event-streams to read chunks of XML data into the memory in a sequential manner whereas DOM uses a tree representation of the underlying XML document and facilitates random access/manipulation of the underlying XML data.
- **XML-Dev mailing list v/s W3C:** SAX was developed by the XML-Dev mailing list whereas DOM was developed by W3C (World Wide Web Consortium).
- **Information Set:** SAX doesn't retain all the info of the underlying XML document such as comments whereas DOM retains almost all the info. New versions of SAX are trying to extend their coverage of information.