UNIT III

Host Objects:-

JavaScript supports three types of objects: native, host, and user-defined. Native objects are objects supplied by the JavaScript language. String, Boolean, Math, and Number are examples of native objects. Host objects are JavaScript objects that provide special access to the host environment. They are provided by the browser for the purpose of interaction with the loaded document. In a browser environment,

1. window
2. document

objects are host objects. Several other browser host objects are informal, *de facto* standards. They are: alert, prompt, confirm.

DOM

• The Document Object Model (DOM) is an API that allows programs to interact with HTML (or XML) documents

• The primary function of the Document Object Model is to view, access, and change the structure of an HTML document separate from the content contained within it.

• The DOM will provide you with methods and properties to retrieve, modify, update, and delete parts of the document you are working on. The properties of the Document Object Model are used to describe the web page or document and the methods of the Document Object Model are used for working with parts of the web page.

In DOM, HTML document is represented in tree like structure. It constructs a hierarchical tree structure for a HTML document to traverse and to manipulate the document.

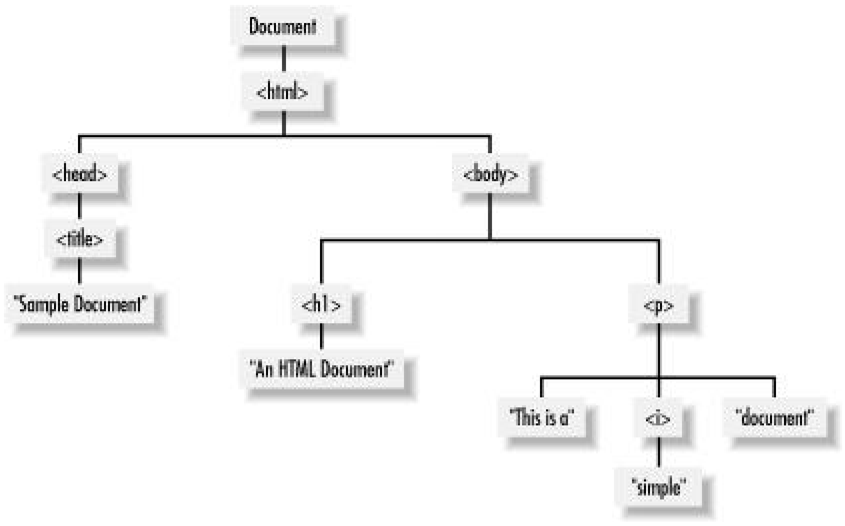
• For example,

```
<html>
<head>
<title>Sample Document</title>
</head>
<body>
<h1>An HTML Document</h1>
<p>This is a <i>simple</i> document.
```

</body>

</html>

The DOM representation of this document is as follows:



The node directly above a node is the *parent* of that node. The nodes one level directly below another node are the *children* of that node. Nodes at the same level, and with the same parent, are *siblings*. The set of nodes any number of levels below another node are the *descendants* of that node.

Types of nodes :-

• There are many types of nodes in the DOM document tree that specifies what kind of node it is.
Every Object in the DOM document tree has properties and methods defined by the Node host
object.
The following table lists the non method properties of Node object.

Traversing a Document: Counting the number of Tags

The DOM represents an HTML document as a tree of Node objects. With any tree structure, one of the
most common things to do is traverse the tree, examining each node of the tree in turn. The following program shows one way to do this.

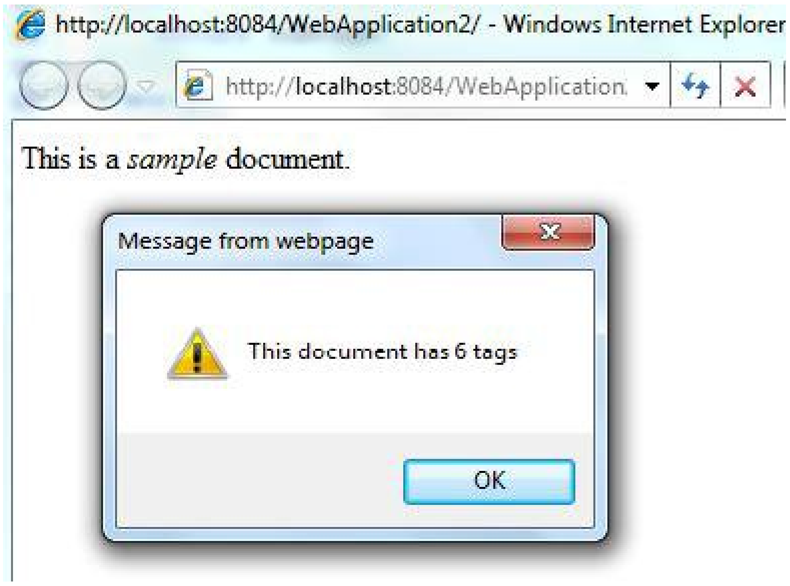
<html>

<head>

<script>

```
function countTags(n)
{ // n is a Node
var numtags = 0; // Initialize the tag counter
if (n.nodeType == 1 ) // Check if n is an Element
numtags++; // Increment the counter if so
var children = n.childNodes; // Now get all children of n
for(var i=0; i < children.length; i++)
{ // Loop through the children
numtags += countTags(children[i]); // Recurse on each one
}
return numtags; // Return the total number of tags
}
</script>
</head>
<body onload="alert('This document has ' + countTags(document) + ' tags')">
This is a <i>sample</i> document.
</body>
</html>
```

Output


http://localhost:8084/WebApplication2/ - Windows Internet Explorer
http://localhost:8084/WebApplication
Message from webpage
OK

This is a *sample* document.

This document has 6 tags

Server-side Programming: Servlet

The combination of

 HTML

 JavaScript

 DOM

is sometimes referred to as Dynamic HTML (DHTML). Web pages that include scripting are often called dynamic pages. Similarly, web server response can be static or dynamic

 Static: **HTML document is retrieved** from the file system by the server and the same returned to the client.

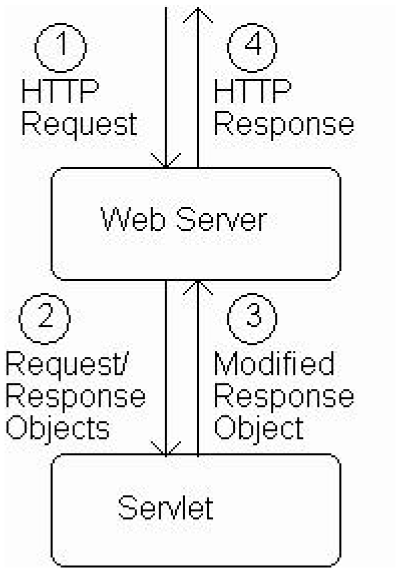
 Dynamic: In server, a **HTML document is generated** by a program in response to an

HTTP request

Java servlets are one technology for producing dynamic server responses. Servlet is a class instantiated by the server to produce a dynamic response.

Servlet Overview

The following figure illustrates the servlet program working principle.



1.When server starts it instantiates servlets

2. Server receives HTTP request, determines need for dynamic response

3. Server selects the appropriate servlet to generate the response, creates request/response objects, and passes them to a method on the servlet instance

4. Servlet adds information to response object via method calls

5. Server generates HTTP response based on information stored in response object

Types of Servlet

- Generic Servlet
- HttpServlet

Servlets vs. Java Applications

- Servlets do not have a main() method
- Entry point to servlet code is via call to a method doGet() /doPost()
- Servlet interaction with end user is indirect via request/response object APIs
- Primary servlet output is typically HTML

Running Servlets

1. Compile servlet (make sure that JWSDP libraries are on path)
2. Copy .class file to **shared/classes** directory
3. (Re)start the Tomcat web server
4. If the class is named ServletHello, browse to

What are Servlets?

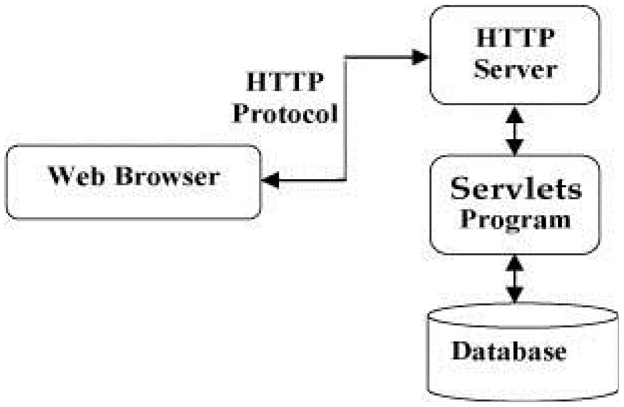
Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface

(CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.



Servlets Tasks:

Servlets perform the following major tasks:

1. Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
2. Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
3. Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
4. Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
5. Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.
Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard Part of the Java's enterprise edition, an expanded version of the Java class library that supports large-Scale development projects.
These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.
Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

Servlets - Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following
are the paths followed by a servlet

- The servlet is initialized by calling the **init ()** method.
- The servlet calls **service()** method to process a client's request.

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
public void init() throws ServletException
{
// Initialization code...
}
```

The service() method :

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

```
public void service(ServletRequest request,
ServletResponse response)
throws ServletException, IOException
{
}
```

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here are the signature of these two methods.

The doGet() Method

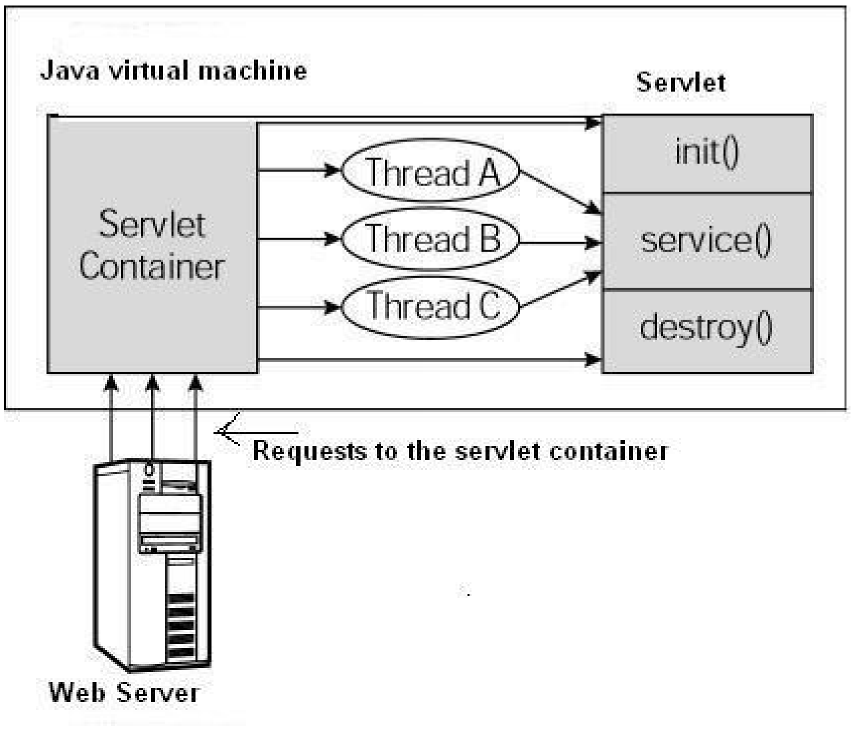
Architecture Diagram:

The following figure depicts a typical servlet life-cycle scenario.

 First the HTTP requests coming to the server are delegated to the servlet container.

 The servlet container loads the servlet before invoking the service() method.

 Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



Structure of a servlet program

import java.io.*;

import javax.servlet.*;

```
import javax.servlet.http.*;
public class NewServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html"); // content type of the response
PrintWriter out = response.getWriter(); // used to create a response as a Html doc
try {
out.println("<html>");
---------------------
---------------------
out.println("</html>");
}catch(Exception e){}
}
}
}
```

GET vs. POST method for forms:

GET:

 It is used to process the query string which is part of URL

 If the length of query string is limited it may be used.

 It is recommended when parameter data is not stored but used only to request information.

POST:

 It is used to process the query string as well as to store the data on server.

 If the Query string is sent as body of HTTP request, the post method will be used to retrieve.

 If the length of query string is unlimited, it can be used

 It is recommended if parameter data is intended to cause the server to update stored data

 Most browsers will warn you if they are about to resubmit POST data to avoid duplicate updates