UNIT – I

Overview

This section provides information about web technologies that relate to the interface between web servers and their clients. This information includes markup languages, programming interfaces and languages, and standards for document identification and display. The term "**Web 2.0**" (2004–present) is commonly associated with web applications that facilitate interactive information sharing, interoperability, user-centered design and collaboration on the World Wide Web. Examples of Web 2.0 include web-based communities, hosted services, web applications, social-networking sites, video-sharing sites, wikis, blogs, mashups, and folksonomies. A Web 2.0 site allows its users to interact with other users or to change website content, in contrast to non-interactive websites where users are limited to the passive viewing of information that is provided to them.

Objectives:

- Explain about the principles of internet
- Explain a basic web concepts
- Explain how does client sever model work
- what is marup language and how it is embedded in web

1. INTRODUCTION

1.1. INTERNET PRINCIPLES

This page reviews the central concepts of software on the internet. It briefly explains:

- **TCP/IP**
- **UDP**
- **IP Addresses**
- **domain names**
- **the domain name system**
- **ports**
- **sockets**
- **URL's**

1.2. 1. TCP/IP

The Internet is the network that connects computers all over the world. It works according to a set of agreed protocols. **TCP (Transmission Control Protocol)** and **IP(Internet Protocol)** are the most commonly-used protocols for using the Internet. (But there are others at lower levels.) The combination is simply known as **TCP/IP.**

The Internet is a **packet switching** system. Any message is broken into packets that are transmitted independently across the interment (sometime by different routes). These packets are called **datagrams.** The route chosen for each datagram depends on the traffic at any point in time. Each datagram has a header of between 20 and 60 bytes, followed by the payload of up to 65,515 bytes of data. The header consists of, amongst other data:

1. the version number of the protocol in use
2. IP address of sender
3. IP address of destination

TCP breaks down a message into packets. At the destination, it re-assembles packets into messages. It attaches a checksum to each packet. If the checksum doesn't match the computed checksum at the destination, the packet is re-transmitted. Thus TCP ensures reliable transmission of information. In summary, TCP:

1. provides re-transmission of lost data
2. delivery of data in the correct order

1.1.2. UDP

Most applications use TCP. However, an example of a situation in which it is desirable to use a lower-level protocol is the case of audio **streaming.** If you want to download a sound file, it can take some time, even though it may be compressed. You have to wait (maybe some time) for the complete file to download before it can be played. An alternative is to listen to the sound as it is being downloaded - streaming. One of the most popular technologies is called RealAudio.

RealAudi does not use TCP because of its overhead. The sound file is sent in IP packets using the **UDP (User Datagram Protocol)** instead of TCP. UDP is an unreliable protocol:

- it doesn't guarantee that a packet will arrive

- it doesn't guarantee that packets are in the right order

UDP doesn't re-send a packet if it is missing or there is some other error, and it doesn't assemble packets into the correct order. But it is faster than TCP. In this application, losing a few bits of data is better than waiting for the re-transmission of some missing data. The application's major mission is to keep playing the sound without interruption. (In contrast, the main goal of a file transfer program is to transmit the data accurately.)

The same mechanism is used with video streaming.

UDP is a protocol at the same level as TCP, above the level of IP.

1.1.3. Domain name

The **domain name** is the user-friendly equivalent of an IP address. Used because the numbers in an IP address are hard to remember and use. Also known as a host name. Example:

shu.ac.uk

Such a name starts with the most local part of the name and is followed by the most general. The whole name space is a tree, whose root has no name. The first level in the tree has com, org, edu, UK, etc.

The parts of a domain name don't correspond to the parts of an IP address. Indeed

domain names don't always have 4 parts - they can have 2, 5 or whatever.

All applications that use an address should work whether an IP address or a domain name is used. In fact, a domain name is converted to an IP address before it is used.

Exercise: Compare and contrast IP addresses with domain names.

1.1.4. Domain Name System

A program, say a Web browser, that has a domain address usually needs to convert it into an IP address before making contact with the server. The domain name system (DNS) provides a mapping between IP addresses and domain names. All this information cannot be all in one place and so it is a distributed database.

1.1.5. Clients, Servers and Peers

A network application usually involves a client and a server. Each is a process (an independently running program) running on a (different) computer.

A server runs on a host and provides some particular service, e.g. e-mail, access to local Web pages. Thus a Web server is a server. A commonly-used web server program is called Apache.

A client runs on a host but needs to connect with a sever on another host to accomplish its task. Usually, different clients are used for different tasks, e.g. Web browsing and e-mail. Thus a Web browser is a client.

Some programs are not structured as clients and servers. For example a game, played across the internet by two or more players is a peer to peer relationship. Other examples: chat, internet phone, shared whiteboard.

1.1.6. Port Numbers

To identify a host machine, an IP address or a domain name is needed. To identify a particular server on a host, a port number is used. A port is like a logical connection to a machine. Port numbers can take values from 1 to 65,535. It has no correspondence with the physical connections, of which there might be just one. Each type of service has, by convention, a standard port number. Thus 80 usually means Web serving and 21 means file transfer. If the default port number is used, it can be omitted in the URL (see below). For each port supplying a service there is a server program waiting for any requests. Thus a web server program listens on port 80 for any incoming requests. All these server programs run together in parallel on the host machine.

When a packet of information is received by a host, the port number is examined and the packet sent to the program responsible for that port. Thus the different types of request are distinguished and dispatched to the relevant program.

The following table lists the common services, together with their normal port numbers. These conventional port numbers are sometimes not used for a variety of reasons. One example is when a host provides (say) multiple web servers, so only one can be on port 80. Another reason is where the server program has not been assigned the privilege to use port 80.

1.1.7. Sockets

A socket is the software mechanism for one program to connect to another. A pair of programs opens a socket connection between themselves. This then acts like a

telephone connection - they can converse in both directions for as long as the connection is open. (In fact, data can flow in both directions at the same time.) More than one socket can use any particular port. The network software ensures that data is routed to or from the correct socket.

When a server (on a particular port number) gets an initial request, it often spawns a separate thread to deal with the client. This is because different clients may well run different speeds. Having one thread per client means that the different speeds can be accommodated. The new thread creates a (software) socket to use as the connection to the client. Thus one port may be associated with many sockets.

1.1.8. Streams

Accessing information across the Internet is accomplished using streams. A stream is a serial collection of data, such as can be sent to a printer, a display or a serial file. Similarly a stream is like data input from a keyboard or input from a serial file. Thus reading or writing to another program across a network is just like reading or writing to a serial file.

1.1.9. URL

A URL (Uniform Resource Locator) is:

- a unique identifier for any resource on the Internet
- typed into a Web browser
- used as a hyperlink within a HTML document
- quoted as a reference to a source

A URL has the structure:

protocol: //hostname[:port]/[pathname]/filename#section

The host name is the name of the server that provides the service. This can either be a domain name or an IP address.

The port number is only needed when the server does not use the default port number. For example, 80 is the default port number for HTTP.

1.2. BASIC WEB CONCEPTS

This section describes the essential concepts for working with PowerDynamo (Storing, locating, and transmitting information on the Web)

How information is located: the URL

To move from one page of a document to another page, or to another document on the same or another Web site, the user clicks a hyperlink (usually just called a link) in the document shown in their Web client. Documents and locations within documents are identified by an address, defined as a Uniform Resource Locator, or **URL**. The following URL illustrates the general form:

http://www.sybase.com/productsl

or

http://www.sybase.com/inc/corpinfo/mkcreate.html

URLs contain information about which server the document is on, and may also specify a particular document available to that server, and even a position within the document. In addition, a URL may carry other information from a Web client to a Web server, including the values entered into fields in an HTML form.

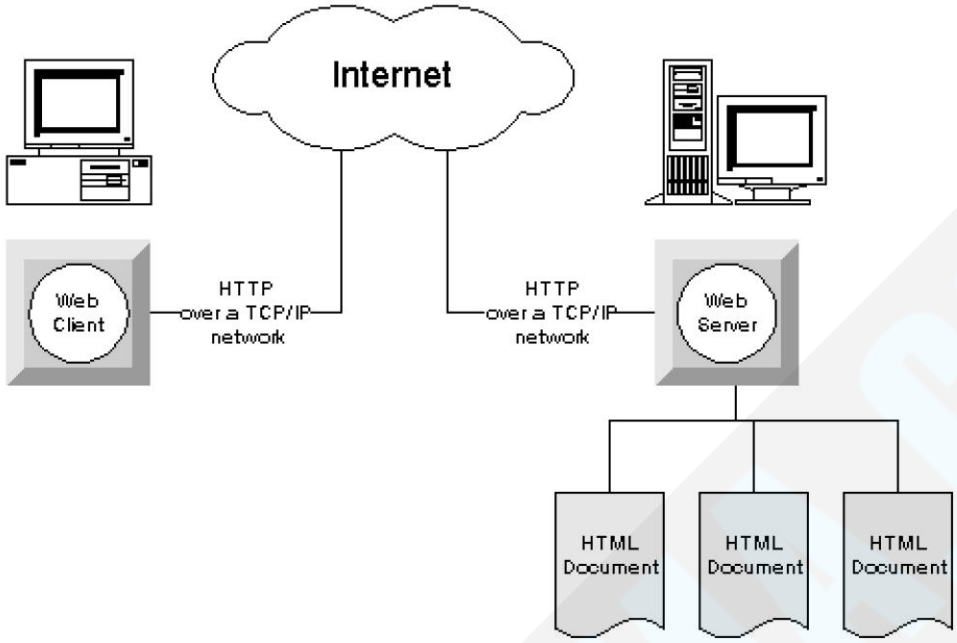
For more information about URLs and addresses on the Web, see the material on the World Wide Web Consortium pages, at the following address:

http://www.w3.org/pub/WWW/Addressing/

When a user clicks a link on a document on their Web client, the URL is sent to the server of the indicated Web site. The Web server locates the document, and sends the HTML to the Web client across the network.

The below figure illustrates, information is stored at **Web sites**. Access to the information is managed by a **Web server** for the site. Users access the information using **Web clients**, which are also called **browsers**.

Figure 2-1: Accessing information on the Web



Information on the Web is stored in documents, using a language called **HTML** (HyperText Markup Language). Web clients must interpret HTML to be able to display the documents to a user. The protocol that governs the exchange of information between the Web server and Web client is named **HTTP** (HyperText Transfer Protocol).

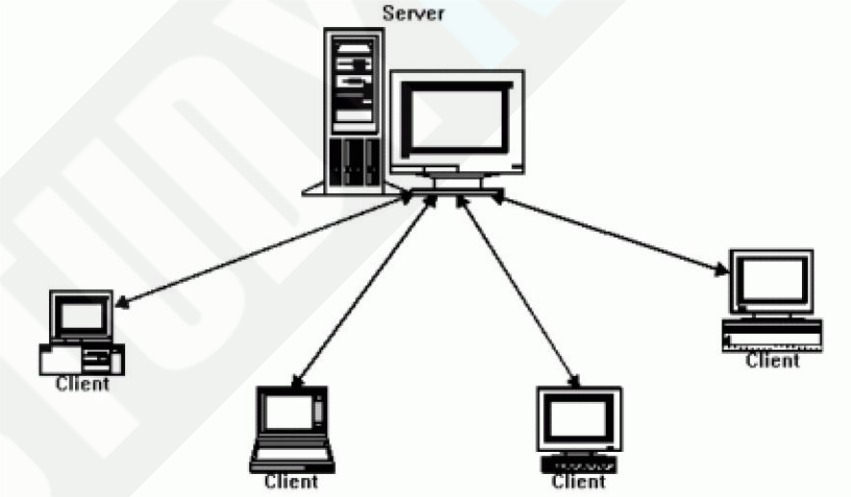
External data in HTML documents

HTML documents can include graphics or other types of data by referencing an external file (for example, a GIF or JPEG file for a graphic). Not all these external formats are supported by all Web clients. When the document contains such data, the Web client can send a request to the Web server to provide the relevant graphic. If the Web client does not support the format, it does not request the information from the server.

1.3. CLIENT / SERVER MODEL

1.3.1. Client server

Client/server describes the relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfils the request. Although programs within a single computer can use the client/server idea, it is a more important idea in a network. In a network, the client/server model provides a convenient way to interconnect programs that are distributed efficiently across different locations. Computer transactions using the client/server model are very common. For example, to check your bank account from your computer, a client program in your computer forwards your request to a server program at the bank. That program might in turn forward the request to its own client program that sends a request to a database server at another bank computer to retrieve your account balance. The balance is returned back to the bank data client, which in turn serves it back to the client in your personal computer, which displays the information for you.



Client server model diagram

Source: http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html -

The communications method in computing includes Local procedure calls and Remote procedure calls.

1.3.2. Remote Procedure Call:

This is a protocol that one program can use to request the services from other located in other machine in a network without having to understand the network details. Usually when a program using RPC are compiled into an executable program, a stub is included which acts as the representative of remote procedure code. When the program is run, and the procedure issue the stub receives a request and forwards it to the client runtime in the local computer by the daemons. The client runtime program knows the address of the remote computer and server application. It then sends the request across the network .The server also have a runtime program and stub that interface with remote procedure. The result is returned the same way.

1.3.3. Local Procedure Call

A local procedure call (LPC) is an interprocess communication facility for high-speed message passing

In Windows NT, client-subsystem communication happens in a fashion similar to that in the MACH operating system. Each subsystem contains a client-side DLL that links with the client executable. The DLL contains stub functions for the subsystem's API. Whenever a client process–an application using the subsystem interface–makes an API call, the corresponding stub function in the DLL passes on the call to the subsystem process. The subsystem process, after the necessary processing, returns the results to the client DLL. The stub function in the DLL waits for the subsystem to return the results and, in turn, passes the results to the caller. The client process simply resembles calling a normal procedure in its own code. In the case of RPC, the client actually calls a procedure sitting in some remote server over the network–hence the name remote

procedure call. In Windows NT, the server runs on the same machine; hence the mechanism is called as a local procedure call.

LPC is designed to allow three methods of exchanging messages:

- Message that is shorter than 256 bytes can be sent by calling LPC with a buffer containing the message. That is a small message. This message is then copied from the address space of the sending process into system address space, and then to the address space of the receiving process.

- If a client and a server want to exchange more than 256 bytes of data, they can choose to use a shared section to which both are mapped. The sender places message data in the shared section and then sends a small message to the receiver with pointers to where the data is to be found in the shared section.

- When a server wants to read or write larger amount of data than will fit in a shared section, data can be directly read from or written to a client's address space. The LPC component supplies two functions that a server can use to accomplish this. A message sent by the first method is used to synchronize the message passing.

There are three types of LPC. The first type sends small messages up to 304 bytes. The second type sends larger messages. The third type of LPC is called as Quick LPC and used by the Win32 subsystem in Windows NT 3.51.

The first two types of LPC use port objects for communication. Ports resemble the sockets or named pipes in Unix. A port is a bi-directional communication channel between two processes. However, unlike sockets, the data passed through ports is not

UNIT-II

Overview

Client-side scripting generally refers to the class of computer programs on the web that are executed client- side, by the user's web browser, instead of server- side (on the web server) This type of computer programming is an important part of the Dynamic HTML (DHTML) concept, enabling web pages to be scripted; that is, to have different and changing content depending on user input, environmental conditions (such as the time of day), or other variables.

Web authors write client-side scripts in languages such as JavaScript (Client-side JavaScript) and VBScript.

- JavaScript is considered to be one of the most famous scripting languages of all time. Please don't confuse JavaScript with Java; we shall discuss the difference amongst them later. JavaScript, by definition, is a Scripting Language of the World Wide Web. The main usage of JavaScript is to add various Web functionalities, Web form validations, browser detections, creation of cookies and so on. JavaScript, along with VBScript, is one of the most popular scripting languages and that is why it is supported by almost all web browsers available today like Firefox, Opera or the most famous Internet Explorer.

- JavaScript was originally designed in order to add interactivity to the Web Pages or HTML pages. Previously, the interactivity was achieved through the use of forms made in JavaScript but later JavaScript was used to add more dynamic and interactive features. JavaScript, as mentioned before, is a scripting language which falls under the category of light-weight programming languages. Using JavaScript is quite easy as the web programmer can easily embed the JavaScript code into the HTML pages. Another interesting fact about JavaScript is that it doesn't require the user to purchase any license. Another advantage of using JavaScript is that it is already interpreted, which is the reason it is also called as an interpreted language. Interpreted language means that the JavaScript execute simply without any preliminary compilation; which is another reason why it is called as light-weight programming language.

Objectives

- **Explain about client side scripting**
- **What is java script**
- **Why scripting languages are used**
- **Differentiate between javascript and vbscript**

Introduction

Client-side scripts are often embedded within an HTML document (hence known as an "embedded script"), but they may also be contained in a separate file, which is referenced by the document (or documents) that use it (hence known as an "external script"). Upon request, the necessary files are sent to the user's computer by the web server (or servers) on which they reside. The user's web browser executes the script, then displays the document, including any visible output from the script. Client-side scripts may also contain instructions for the browser to follow if the user interacts with the document in a certain way, e.g., clicks a certain button. These instructions can be followed without further communication with the server, though they may require such communication.

By viewing the file that contains the script, users may be able to see its source code. Many web authors learn how to write client-side scripts partly by examining the source code for other authors' scripts.

In contrast, server-side scripts, written in languages such as Perl, PHP, and server-side VBScript, are executed by the web server when the user requests a document. They produce output in a format understandable by web browsers (usually HTML), which is then sent to the user's computer. The user cannot see the script's source code (unless the author publishes the code separately), and may not even be aware that a script was executed. The documents produced by server-side scripts may, of course, contain client-side scripts.

Client -side scripts have greater access to the information and functions available on the user's browser, whereas server-side scripts have greater access to the information and functions available on the server. Server-side scripts require that their language's interpreter be installed on the server, and produce the same output regardless of the client's browser, operating system, or other system details. Client-side scripts do not require additional software on the server (making them popular with authors who lack administrative access to their servers); however, they do require that the user's web browser understands the scripting language in which they are written.

It is therefore impractical for an author to write scripts in a language that is not supported by the web browsers used by a majority of his or her audience.

Due to security restrictions, client -side scripts may not be allowed to access the user's computer beyond the web browser application. Techniques like ActiveX controls can be used to sidestep this restriction.

Unfortunately, even languages that are supported by a wide variety of browsers may not be implemented in precisely the same way across all browsers and operating systems. Authors are well -advised to review the behavior of their client-side scripts on a variety of platforms before they put them into use.

Client Side Image Maps

How to add an image map to your page
Image maps aren't as bad as they seem, at least if you use a client side image map using HTML rather than a CGI program. I may add a section on using a CGI-driven image map later on, but for now, lets look at one that just uses HTML and an image!

To begin, let's look at a simple image map I created. If you click the left half of the image, you will be taken to a tables tutorial. If you click the right side, you'll get a tutorial on frames. Both the links are on one single image, so we just call it an image map. Try it out:

So, how do we create one of these? First, you need to have an image you want to use. Just pick one you like, or make your own. Next, you need to know the width and height of the image you are using. If you are not sure, open the image in your image editing program. You should have some option that will tell you the width and height of the image.

Now you need to put the image on the page. To do this, you use the image tag, but with a new attribute: usemap.

The **usemap="#mymap"** command tells the browser to use a map on the page, which is named "mymap". Notice how it uses the "#" symbol in front of the map name. Also notice that we defined the width and height of the image. This needs to be done so we can use coordinates later on when we define the map. Speaking of that, lets see how to define

the map. For this map, we would place the following code somewhere on the page. I just put it right after the <img> tag to make it easy to find.

<map name="mymap" id="mymap"> <area shape="rect" coords="0,0,99,40" href="table1.htm" alt="Tables" /> <area shape="rect" coords="100,0,200,40" href="frame1.htm" alt="Frames" /> <area shape="default" href="http://www.pageresource.com" alt="Home" /> </map>

Now you can see where the usemap="#mymap" from the <img> tag comes from. The name of the map is "mymap". Now, let's look at what all of this means:

- **<map name="mymap" id="mymap">**
  This defines your image map section, and gives the map a name. This map is named "mymap" In XHTML, the id attribute is required rather than name. If you are using XHTML transitional, both the name and id can be used.
- **<area shape="rect" coords="0,0,99,40" href="table1.htm" alt="Tables" />**
  The area tag defines an area of the image that will be used as a link. The shape attribute tells the browser what shape the area will be. To keep it simple, I only used "rect", which stands for rectangle. The coords attribute is where we define the edges of each area. Since it is a rectangle, we will use two sets of coordinates. The first set defines where to start the rectangle, where the top-left edge of the rectangle will be. Since this rectangle starts at the top-left edge of the image, the coordinates are (0 pixels, 0 pixels). The second two numbers define where to end the rectangle. This will be the lower-right edge of the rectangle. Remember that the total image size was 200x40. We want the lower-right edge of this rectangle to be halfway accross the image and at the bottom of the image. Going accross, half of 200 is 100, but we use 99 here because 100 can only be used once. We will use it in the second rectangle here. Of course, 40 pixels takes us to the bottom of the image. So the lower-right corner of this rectangle will be 99 pixels accross the image, and 40 pixels (all the way) down the image. And now the easy part: The href attribute is used to tell the browser where to go when someone clicks someplace on that rectangle. Put the URL of the page you want to go to in there, and the first rectangle is set up! The alt attribute allows you to define alternate text for that area.
- **<area shape="rect" coords="100,0,200,40" href="frame1.htm" alt="Frames" />**
  Basically the same as the previous area tag, but it is for our second rectangle. We start where the other one left off, but back at the top of the image. Since the right edge of the last rectangle was at 99 pixels accross, we start this one at 100 pixels accross. And since this will be the upper-left of the second rectangle, we start it at 0 pixels down the image (the top!). We end this rectangle where the image ends, so the lower-right coordinate here is pretty nice- (200, 40), the size of the image!
- **<area shape="default" href="http://www.pageresource.com" alt="Home">**
  The default is not really a new shape, it just covers anything that may have been

left out. We didn't leave out anything in this map, but if we had, this would be the URL someone would go to if they clicked on any area we did not define earlier.

- **</map>**
  This ends the map section!

Now, you can use other shapes besides rectangles, but those are alot tougher to code by hand. I would suggest using a shareware image map creation program if you use other shapes in your map.

So, if you need a simple image map separated into rectangular sections, just do what we did above. You can have as many rectangular areas as you want, just add the extra area tags with coordiantes and urls. Sounds like fun, don't you think?

JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML / XHTML

If you want to study these subjects first, find the tutorials on our Home page.

What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

Are Java and JavaScript the same?

NO!

Java and JavaScript are two completely different languages in both concept and design!

Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What can a JavaScript do?

- **JavaScript gives HTML designers a programming tool -** HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page -** A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

---

The Real Name is ECMAScript

JavaScript's official name is ECMAScript.

ECMAScript is developed and maintained by the ECMA organization.

ECMA-262 is the official JavaScript standard.

The language was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996.

The development of ECMA-262 started in 1996, and the first edition of was adopted by the ECMA General Assembly in June 1997.

The standard was approved as an international ISO (ISO/IEC 16262) standard in 1998.

The development of the standard is still in progress.

The HTML <script> tag is used to insert a JavaScript into an HTML page.

JavaScript

JavaScript is a scripting language used to enable programmatic access to objects within other applications. It is primarily used in the form of client-side JavaScript for the development of dynamic websites. JavaScript is a dialect of the ECMAScript standard and is characterized as a dynamic, weakly typed, prototype-based language with first-class functions. JavaScript was influenced by many languages and was designed to look like Java, but to be easier for non-programmers to work with.

JavaScript-specific

JavaScript is officially managed by Mozilla, and new language features are added periodically. However, only some non-Mozilla "JavaScript" engines support these new features:

- conditional `catch` clauses
- property getter and setter functions
- iterator protocol adopted from Python
- shallow generators/coroutines also adopted from Python
- array comprehensions and generator expressions also adopted from Python
- proper block scope via new `let` keyword
- array and object destructuring (limited form of pattern matching)
- concise function expressions (`function(args) expr`)
- E4X

Use in web pages

The primary use of JavaScript is to write functions that are embedded in or included from HTML pages and interact with the Document Object Model (DOM) of the page. Some simple examples of this usage are:

- Opening or popping up a new window with programmatic control over the size, position, and attributes of the new window (i.e. whether the menus, toolbars, etc. are visible).
- Validation of web form input values to make sure that they will be accepted before they are submitted to the server.
- Changing images as the mouse cursor moves over them: This effect is often used to draw the user's attention to important links displayed as graphical elements.

Because JavaScript code can run locally in a user's browser (rather than on a remote server) it can respond to user actions quickly, making an application feel more responsive. Furthermore, JavaScript code can detect user actions which HTML alone cannot, such as individual keystrokes. Applications such as Gmail take advantage of this: much of the user-interface logic is written in JavaScript, and JavaScript dispatches requests for information (such as the content of an e-mail message) to the server. The wider trend of Ajax programming similarly exploits this strength.

A JavaScript engine (also known as JavaScript interpreter or JavaScript implementation) is an interpreter that interprets JavaScript source code and executes the script accordingly. The first ever JavaScript engine was created by Brendan Eich at Netscape Communications Corporation, for the Netscape Navigator web browser. The engine, code-named SpiderMonkey, is implemented in C. It has since been updated (in JavaScript 1.5) to conform to ECMA-262 Edition 3. The Rhino engine, created primarily by Norris Boyd (formerly of Netscape; now at Google) is a JavaScript implementation in Java. Rhino, like SpiderMonkey, is ECMA-262 Edition 3 compliant.

JavaScript and Java

A common misconception is that JavaScript is similar or closely related to Java; this is not so. Both have a C-like syntax, are object-oriented, are typically sandboxed and are widely used in client-side Web applications, but the similarities end there. Java has static typing; JavaScript's typing is dynamic (meaning a variable can hold an object of any type and cannot be restricted). Java is loaded from compiled bytecode; JavaScript is loaded as human-readable code. C is their last common ancestor language.

Nonetheless, JavaScript was designed with Java's syntax and standard library in mind. In particular, all Java keywords are reserved in JavaScript, JavaScript's standard library follows Java's naming conventions, and JavaScript's Math and Date classes are based on those from Java 1.0.

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true
- **if...else statement** - use this statement to execute some code if the condition is true and another code if the condition is false

- **if...else if....else statement** - use this statement to select one of many blocks of code to be executed
- **switch statement** - use this statement to select one of many blocks of code to be executed

If Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
```

Example

```
<script type="text/javascript">
//Write a "Good morning" greeting
if //the time is less than 10

var d=new Date();
var time=d.getHours();

if (time<10)
  {
  document.write("<b>Good morning</b>");
  }
</script>
```

If...else Statement

Use the if....else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)
  {
  code to be executed if condition is true
  }
else
```

```
  {
  code to be executed if condition is not true
  }
```

Example

```
<script type="text/javascript">
//If the time is less than 10, you will get a "Good morning" greeting.
//Otherwise you will get a "Good day" greeting.

var d = new Date();
var time = d.getHours();

if (time < 10)
  {
  document.write("Good morning!");
  }
else
  {
  document.write("Good day!");
  }
</script>
```

If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)
  {
  code to be executed if condition1 is true
  }
else if (condition2)
  {
  code to be executed if condition2 is true
  }
else
  {
  code to be executed if condition1 and condition2 are not true
  }
```

Example

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
```

```
if (time<10)
  {
  document.write("<b>Good morning</b>");
  }
else if (time>10 && time<16)
  {
  document.write("<b>Good day</b>");
  }
else
  {
  document.write("<b>Hello World!</b>");
  }
</script>
```

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

This is how it works: First we have a single expression n (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically.

Example

```
<script type="text/javascript">
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.

var d=new Date();
```

```
theDay=d.getDay();
switch (theDay)
{
case 5:
  document.write("Finally
  Friday"); break;
case 6:
  document.write("Super Saturday");
  break;
case 0:
  document.write("Sleepy Sunday");
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
```

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
alert("sometext");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
alert("I am an alert box!");
}
</script>
</head>
<body>

<input type="button" onclick="show_alert()" value="Show alert box" />

</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

confirm("sometext");

Example

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
  {
  document.write("You pressed OK!");
  }
else
  {
  document.write("You pressed Cancel!");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_confirm()" value="Show confirm box" />

</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
prompt("sometext","defaultvalue");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
  {
  document.write("Hello " + name + "! How are you today?");
  }
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()" value="Show prompt box" />

</body>
</html>
```

JavaScript Functions

To keep the browser from executing a script when the page loads, you can put your script into a function.

A function contains code that will be executed by an event or by a call to the function.

You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

How to Define a Function

Syntax

```
function functionname(var1,var2,...,varX)
{
some code
}
```

The parameters var1, var2, etc. are variables or values passed into the function. The { and the } defines the start and end of the function.

Note: A function with no parameters must include the parentheses () after the function name.

Note: Do not forget about the importance of capitals in JavaScript! The word function must be written in lowercase letters, otherwise a JavaScript error occurs! Also note that you must call a function with the exact same capitals as in the function name.

Example

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello World!");
}
</script>
</head>

<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" />
</form>
</body>
</html>
```

If the line: alert("Hello world!!") in the example above had not been put within a function, it would have been executed as soon as the line was loaded. Now, the script is not executed before a user hits the input button. The function displaymessage() will be executed if the input button is clicked.

The return Statement

The return statement is used to specify the value that is returned from the function.

So, functions that are going to return a value must use the return statement.

The example below returns the product of two numbers (a and b):

Example

```
<html>
<head>
<script type="text/javascript">
function product (a,b)
{
return a*b;
}
</script>
</head>

<body>
<script type="text/javascript">
document.write(product(4,3));
</script>

</body>
</html>
```

Loops execute a block of code a specified number of times, or while a specified condition is true.

JavaScript Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript, there are two different kinds of loops:

- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

The for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (var=startvalue;var<=endvalue;var=var+increment)
{
code to be executed
}
```

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs.

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

The while Loop

The while loop loops through a block of code while a specified condition is true.

Syntax

```
while (var<=endvalue)
  {
  code to be executed
  }
```

Note: The <= could be any comparing statement.

Example

The example below defines a loop that starts with i=0. The loop will continue to run as long as **i** is less than, or equal to 5. **i** will increase by 1 each time the loop runs:

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
  {
  document.write("The number is " + i);
  document.write("<br />");
  i++;
  }
</script>
</body>
</html>
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Syntax

```
do
  {
  code to be executed
  }
while (var<=endvalue);
```

Example

The example below uses a do...while loop. The do ...while loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested:

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
  {
```

```
 document.write("The number is " + i);
 document.write("<br />");
 i++;
 }
while (i<=5);
</script>
</body>
</html>
```

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
 {
 if (i==3)
   {
   break;
   }
 document.write("The number is " + i);
 document.write("<br />");
 }
</script>
</body>
</html>
```

The continue Statement

The continue statement will break the current loop and continue with the next value.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
 {
 if (i==3)
```

```
  {
  continue;
  }
 document.write("The number is " + i);
 document.write("<br />");
 }
</script>
</body>
</html>
```

JavaScript For...In Statement

The for...in statement loops through the elements of an array or through the properties of an object.

Syntax

```
for (variable in object)
 {
 code to be executed
 }
```

Note: The code in the body of the for...in loop is executed once for each element/property.

Note: The variable argument can be a named variable, an array element, or a property of an object.

Example

Use the for...in statement to loop through an array:

```
<html>
<body>

<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars = "Volvo"; mycars
= "BMW";

for (x in mycars)
 {
 document.write(mycars[x] + "<br />");
 }
```

```
</script>

</body>
</html>
```

JavaScript is an Object Oriented Programming (OOP) language.

Object Oriented Programming

JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types.

However, creating your own objects will be explained later, in the Advanced JavaScript section. We will start by looking at the built-in JavaScript objects, and how they are used. The next pages will explain each built-in JavaScript object in detail.

Note that an object is just a special kind of data. An object has properties and methods.

Properties

Properties are the values associated with an object.

In the following example we are using the length property of the String object to return the number of characters in a string:

```
<script type="text/javascript">
var txt="Hello World!";
document.write(txt.length);
</script>
```

Methods

Methods are the actions that can be performed on objects.

In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

```
<script type="text/javascript">
var str="Hello world!";
document.write(str.toUpperCase());
</script>
```

What is an Array?

An array is a special variable, which can hold more than one value, at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own ID so that it can be easily accessed.

Create an Array

The following code creates an Array object called myCars:

```
var myCars=new Array();
```

There are two ways of adding values to an array (you can add as many values as you need to define as many variables you require).

1:

```
var myCars=new Array();
myCars[0]="Saab";
myCars="Volvo";
myCars="BMW";
```

You could also pass an integer argument to control the array's size:

```
var myCars=new Array(3);
myCars[0]="Saab";
myCars="Volvo";
myCars="BMW";
```

2:

var myCars=new Array("Saab","Volvo","BMW");

Note: If you specify numbers or true/false values inside the array then the type of variables will be numeric or Boolean instead of string.

Access an Array

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0.

The following code line:

document.write(myCars[0]);

Modify Values in an Array

To modify a value in an existing array, just add a new value to the array with a specified index number:

myCars[0]="Opel";

Now, the following code line:

document.write(myCars[0]);

Example

```
<html>
<head>
<script language="JavaScript">

function temp(form)
{
var f = parseFloat(form.DegF.value,
10); var T = 0;
T = (f - 62.0) * 8.0 / 7.0;
form.DegC.value = T;
}
// done hiding from old browsers --
> </script>
</head>
<body>
<FORM>
```

```
<h2>Fahrenheit to Celsius Converter</h2>
Enter a temperature in degrees F:
<INPUT NAME="DegF" VALUE="0" MAXLENGTH="25"
SIZE=25> <p>
Click button to calculate the temperature
in degrees T:
<INPUT NAME="calc" VALUE="Calculate"
TYPE=BUTTON onClick=temp(this.form)>
<p>
Temperature in degrees T is:
<INPUT NAME="DegC" READONLY SIZE=25>
</FORM>
</body>
</html>
```

Review

Summary

- JavaScript was originally designed in order to add interactivity to the Web Pages or HTML pages.
- JavaScript is quite easy as the web programmer can easily embed the JavaScript code into the HTML pages.
- Iinteresting fact about JavaScript is that it doesn't require the user to purchase any license.
- The advantage of using JavaScript is that it is already interpreted, which is the reason it is also called as an interpreted language. Interpreted language means that the JavaScript execute simply without any preliminary compilation; which is another reason why it is called as light-weight programming language.
    - conditional `catch` clauses
    - property getter and setter functions
    - iterator protocol adopted from Python
    - shallow generators/coroutines also adopted from Python
    - array comprehensions `and` generator expressions also adopted from Python

- proper block scope via new `let` keyword
- array and object destructuring (limited form of pattern matching)
- concise function expressions (`function(args) expr`)
- E4X

Key terms ::

- **ECMA**
- **Java Script**
- **href**
- **https,http**
- **PERL,PHP,CGI,VB Script**

Key Terms Quiz:

Complete each statement by writing one of the terms listed under key terms in each blank

1. Client-side scripts are often embedded within an ----------.
2. Server-side scripts are written in ---------- languages.
3. JavaScript was designed to add interactivity to---------- pages.
4. A scripting language is a ---------- programming language.
5. JavaScript is an ---------- language.
6. JavaScript can read and write ---------- elements
7. ---------- is the official JavaScript standard.
8. JavaScript can be used to create ---------- .
9. ECMAScript is developed and maintained by ---------- .
10. DOM stands for---------- .

Multiple choices

1. A prompt box is often used if you want the user to input a value.

a.after entering a page b. before entering a page

c select the page d.none of the above

2. JavaScript is officially managed by

a. Hot java b.Internet Explore

c Mozilla d.Netscape

3. A JavaScript engine is ---------- JavaScript source code and executes the script accordingly

a. an interpreter that interprets b. a compiler that compiles

c both compiler and interpreter that compiles and interprets d. none of the above.

4. Internet Explorer has ----------available for java script

a. Three debuggers b. Four debuggers

c only one debugger d. none of the above

5. Properties are ----------associated with ---------

a. constant , a class b. the values ,a class

c the values, an object. d. constant, an object

Review questions

In your own words briefly answer the following questions

1. What is client side scripting?

2 Can you make ac omparision between Clinetside scripting and server side scripting?

3. What is java scitping?

4. What is client side image map?

5. What can java script do?

6. How many features are supported by java script engine? What are they?

7. How many conditional statements are there in java script?

8. Why is confirm box used in jscript?

9. How do you define a function?

10. What is the syntax for "for loop "in java script?

Lesson lab::

Complete the following exercise

1. Develop static pages (using only HTML) of an online Book store. The pages should resemble: www.amazon.com
The website should consist of the following pages.
Home page, Registration and user Login, User profile page, Books catalog, Shopping cart, Payment By credit card, order confirmation.

2. Validate the registration, user login, user profile and payment by credit card pages using JavaScript**.**

UNIT III

Overview

The Java programming language and environment is designed to solve a number of problems in modern programming practice. Java started as a part of a larger project to develop advanced software for consumer electronics. These devices are small, reliable, portable, distributed, real-time embedded systems. When we started the project we intended to use C++, but encountered a number of problems. Initially these were just compiler technology problems, but as time passed more problems emerged that were best solved by changing the language.

Java: A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, dynamic language.

Objectives::

- Describe how java has become more popular on internet.
- Explain JVM functions.
- Explain why applet is used in internet.
- Describe how server side programmings are used.

Java Programming

Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture.

The original and reference implementation Java compilers, virtual machines, and class libraries were developed by Sun from 1995. As of May 2007, in compliance with the

specifications of the Java Community Process, Sun made available most of their Java technologies as free software under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java and GNU Classpath.

History

James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later renamed as Java, from a list of random words. Gosling aimed to implement a virtual machine and a language that had a familiar C/C++ style of notation.

Sun released the first public implementation as Java 1.0 in 1995. It promised "Write Once, Run anywhere" (WORA), providing no-cost run-times on popular platforms. Fairly secure and featuring configurable security, it allowed network- and file-access restrictions. Major web browsers soon incorporated the ability to run Java applets within web pages, and Java quickly became popular. With the advent of Java 2 (released initially as J2SE 1.2 in December 1998), new versions had multiple configurations built for different types of platforms. For example, J2EE targeted enterprise applications and the greatly stripped-down version J2ME for mobile applications. J2SE designated the Standard Edition. In 2006, for marketing purposes, Sun renamed new J2 versions as Java EE, Java ME, and Java SE, respectively.

In 1997, Sun Microsystems approached the ISO/IEC JTC1 standards body and later the Ecma International to formalize Java, but it soon withdrew from the process. Java remains a de facto standard, controlled through the Java Community Process. At one time, Sun made most of its Java implementations available without charge, despite their proprietary software status. Sun generated revenue from Java through the selling of licenses for specialized products such as the Java Enterprise System. Sun distinguishes between its Software Development Kit (SDK) and Runtime Environment (JRE) (a subset of the SDK); the primary distinction involves the JRE's lack of the compiler, utility programs, and header files.

On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL). On 8 May 2007 Sun finished the process, making all of Java's core code available under free software / open-source distribution terms, aside from a small portion of code to which Sun did not hold the copyright.

Philosophy

Primary goals

There were five primary goals in the creation of the Java language:

1. It should be "simple, object oriented, and familiar".
2. It should be "robust and secure".
3. It should be "architecture neutral and portable".
4. It should execute with "high performance".
5. It should be "interpreted, threaded, and dynamic".

Java Platform

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any supported hardware/operating-system platform. One should be able to write a program once, compile it once, and run it anywhere.

This is achieved by compiling the Java language code, not to machine code but to Java bytecode – instructions analogous to machine code but intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

Standardized libraries provide a generic way to access host specific features such as graphics, threading and networking. In some JVM versions, bytecode can be compiled to native code, either before or during program execution, resulting in faster execution.

A major benefit of using bytecode is porting. However, the overhead of interpretation means that interpreted programs almost always run more slowly than programs compiled to native executables would, and Java suffered a reputation for poor performance. This gap has been narrowed by a number of optimization techniques introduced in the more recent JVM implementations.

One such technique, known as just -in-time (JIT) compilation, translates Java bytecode into native code the first time that code is executed, then caches it. This results in a program that starts and executes faster than pure interpreted code can, at the cost of introducing occasional compilation overhead during execution. More sophisticated VMs also use dynamic recompilation, in which the VM analyzes the behavior of the running program and selectively recompiles and optimizes parts of the program. Dynamic recompilation can achieve optimizations superior to static compilation because the dynamic compiler can base optimizations on knowledge about the runtime environment and the set of loaded classes, and can identify hot spots - parts of the program, often inner loops, that take up the most execution time. JIT compilation and dynamic recompilation allow Java programs to approach the speed of native code without losing portability.

Another technique, commonly known as static compilation, or ahead-of-time (AOT) compilation, is to compile directly into native code like a more traditional compiler. Static Java compilers translate the Java source or bytecode to native object code. This achieves good performance compared to interpretation, at the expense of portability; the output of these compilers can only be run on a single architecture. AOT could give Java

something close to native performance, yet it is still not portable since there are no compiler directives, and all the pointers are indirect with no way to micro manage garbage collection.

Java's performance has improved substantially since the early versions, and performance of JIT compilers relative to native compilers has in some tests been shown to be quite similar. The performance of the compilers does not necessarily indicate the performance of the compiled code; only careful testing can reveal the true performance issues in any system.

One of the unique advantages of the concept of a runtime engine is that even the most serious errors (exceptions) in a Java program should not 'crash' the system under any circumstances, provided the JVM itself is properly implemented. Moreover, in runtime engine environments such as Java there exist tools that attach to the runtime engine and every time that an exception of interest occurs they record debugging information that existed in memory at the time the exception was thrown (stack and heap values). These Automated Exception Handling tools provide 'root-cause' information for exceptions in Java programs that run in production, testing or development environments. Such precise debugging is much more difficult to implement without the run-time support that the JVM offers.

Implementations

Sun Microsystems officially licenses the Java Standard Edition platform for Microsoft Windows, Linux, Mac OS X, and Solaris. Through a network of third-party vendors and licensees, alternative Java environments are available for these and other platforms.

Sun's trademark license for usage of the Java brand insists that all implementations be "compatible". This resulted in a legal dispute with Microsoft after Sun claimed that the Microsoft implementation did not support RMI or JNI and had added platform-specific features of their own. Sun sued in 1997, and in 2001 won a settlement of $20 million as well as a court order enforcing the terms of the license from Sun. As a result, Microsoft no longer ships Java with Windows, and in recent versions of Windows, Internet Explorer cannot support Java applets without a third-party plugin. Sun, and others, have made available free Java run-time systems for those and other versions of Windows.

Platform-independent Java is essential to the Java EE strategy, and an even more rigorous validation is required to certify an implementation. This environment enables portable server-side applications, such as Web services, servlets, and Enterprise JavaBeans, as well as with embedded systems based on OSGi, using Embedded Java environments. Through the new GlassFish project, Sun is working to create a fully functional, unified open-source implementation of the Java EE technologies.

Sun also distributes a superset of the JRE called the Java Development Kit (commonly known as the JDK), which includes development tools such as the Java compiler, Javadoc, Jar and debugger.

Automatic memory management

Java uses an automatic garbage collector to manage memory in the object lifecycle. The programmer determines when objects are created, and the Java runtime is responsible for recovering the memory once objects are no longer in use. Once no references to an object remain, the unreachable object becomes eligible to be freed automatically by the garbage collector. Something similar to a memory leak may still occur if a programmer's code holds a reference to an object that is no longer needed, typically when objects that are no longer needed are stored in containers that are still in use. If methods for a nonexistent object are called, a "null pointer exception" is thrown.

One of the ideas behind Java's automatic memory management model is that programmers be spared the burden of having to perform manual memory management. In some languages memory for the creation of objects is implicitly allocated on the stack, or explicitly allocated and deallocated from the heap. Either way the responsibility of managing memory resides with the programmer. If the program does not deallocate an object, a memory leak occurs. If the program attempts to access or deallocate memory that has already been deallocated, the result is undefined and difficult to predict, and the program is likely to become unstable and/or crash. This can be partially remedied by the use of smart pointers, but these add overhead and complexity.

Garbage collection may happen at any time. Ideally, it will occur when a program is idle. It is guaranteed to be triggered if there is insufficient free memory on the heap to allocate a new object; this can cause a program to stall momentarily. Where performance or response time is important, explicit memory management and object pools are often used.

Java does not support C/C++ style pointer arithmetic, where object addresses and unsigned integers (usually long integers) can be used interchangeably. This allows the garbage collector to relocate referenced objects, and ensures type safety and security.

As in C++ and some other object-oriented languages, variables of Java's primitive types are not objects. Values of primitive types are either stored directly in fields (for objects) or on the stack (for methods) rather than on the heap, as commonly true for objects (but see Escape analysis). This was a conscious decision by Java's designers for performance reasons. Because of this, Java was not considered to be a pure object-oriented programming language. However, as of Java 5.0, autoboxing enables programmers to proceed as if primitive types are instances of their wrapper classes.

Syntax

The syntax of Java is largely derived from C++. Unlike C++, which combines the syntax for structured, generic, and object-oriented programming, Java was built almost exclusively as an object oriented language. All code is written inside a class and everything is an object, with the exception of the intrinsic data types (ordinal and real numbers, boolean values, and characters), which are not classes for performance reasons.

UNIT IV
DATABASE - ASP - XML

Database:

A **database** is an organized collection of data. The data is typically organized to model relevant aspects of reality (for example, the availability of rooms in hotels), in a way that supports processes requiring this information (for example, finding a hotel with vacancies).

The term *database* is correctly applied to the data and their supporting data structures, and not to the database management system (DBMS). The database data collection with DBMS is called a database system.

Relational database model Overview :

The **relational model** for database management is a database model based on first-order predicate logic, first formulated and proposed in 1969 by Edgar F. Codd.[1][2] In the relational model of a database, all data is represented in terms of tuples, grouped into relations. A database organized in terms of the relational model is a relational database.

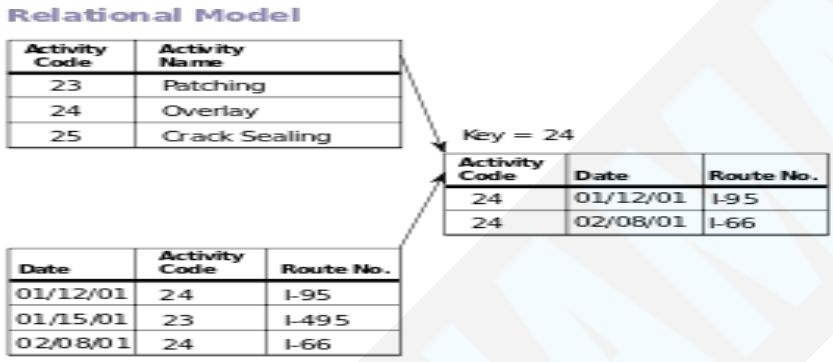
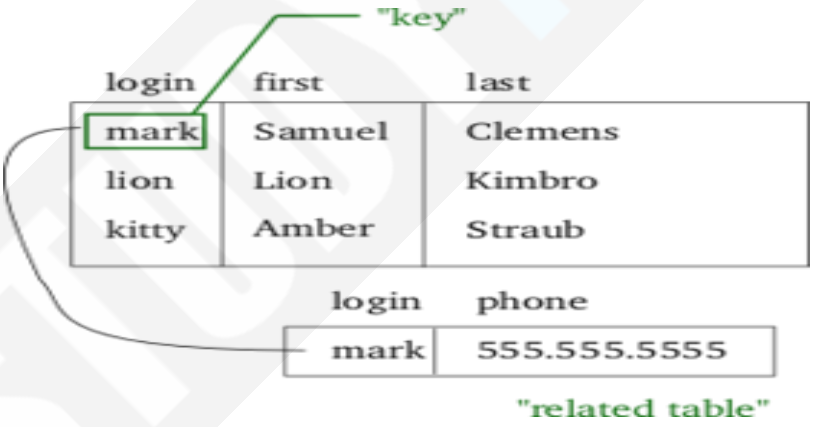


Diagram of an example database according to the Relational model.[3]



In the relational model, related records are linked together with a "key".

SQL:

SQLstands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database.

SQL SELECT Statement

The most commonly used SQL command is SELECT statement. The SQL SELECT statement is used to query or retrieve data from a table in the database.

Syntax of SQL SELECT Statement:

SELECT *column_list* FROM *table-name*
[WHERE Clause] [GROUP BY clause] [HAVING clause] [ORDER BY clause];

- *table-name* is the name of the table from which the information is retrieved.
- *column_list* includes one or more columns from which data is retrieved.
- The code within the brackets is optional.

SQL WHERE Clause

The WHERE Clause is used when you want to retrieve specific information from a table excluding other irrelevant data.

Syntax of SQL WHERE Clause:

WHERE {column or expression} comparison-operator value

Syntax for a WHERE clause with Select statement is:

SELECT *column_list* FROM *table-name* WHERE *condition*;

- *column or expression* - Is the column of a table or a expression
- *comparison-operator* - operators like = < > etc.
- *value* - Any user value or a column name for comparison

SQL UPDATE Statement

The UPDATE Statement is used to modify the existing rows in a table.

The Syntax for SQL UPDATE Command is:

UPDATE table_name
SET column_name1 = value1,
column_name2 = value2, ...
[WHERE condition]

- table_name - the table name which has to be updated.
- column_name1, column_name2.. - the columns that gets changed.
- value1, value2... - are the new values.

SQL Delete Statement

The DELETE Statement is used to delete rows from a table.

The Syntax of a SQL DELETE statement is:

DELETE FROM table_name [WHERE condition];

- table_name -- the table name which has to be updated.

ASP :

- ASP stands for **A**ctive **S**erver **P**ages
- ASP is a Microsoft Technology
- ASP is a program that runs inside **IIS**
- IIS stands for **I**nternet **I**nformation **S**ervices
- IIS comes as a free component with **Windows 2000**

- IIS is also a part of the **Windows NT 4.0 Option Pack**
- The Option Pack can be **downloaded** from Microsoft
- **PWS** is a smaller - but fully functional - version of IIS
- PWS can be found on your **Windows 95/98 CD**

ASP Compatibility

- To run IIS you must have Windows NT 4.0 or later
- To run PWS you must have Windows 95 or later
- ChiliASP is a technology that runs ASP without Windows OS
- InstantASP is another technology that runs ASP without Windows

ASP File

- An ASP file is just the same as an HTML file
- An ASP file can contain text, HTML, XML, and scripts
- Scripts in an ASP file are executed on the server
- An ASP file has the file extension ".asp"
- Dynamically edit, change, or add any content of a Web page
- Respond to user queries or data submitted from HTML forms

Objects :

The Request object

Allows data to be read that was sent by the client browser: Form, Querystring, and HTTP Cookie. It also provides information on the server, the client browser, and retrieve HTTP Cookie stored on the visitor's machine. Can retrieve data from a form using both methods HTTP:

Request.Form reads data sent by POST.

Request.QueryString reads data sent by GET.

```
<%
Response.Write("Welcome " & Request.QueryString("name") & "!") 'this script is vulnerable to XSS, the input has not been encoded (see below)
%>
```

The Response object

Can send information to the client, such as the writing of the text on a page or HTTP Cookie.

```
<%
If (Len(Request.QueryString("name")) > 0) Then
    Response.Cookies("name") = Request.QueryString("name")
End If

Response.Write("Welcome " & Response.Cookies("name") & "!") 'this script is vulnerable to XSS, the input has not been encoded (see below)
%>
```

The Server object

Allows connections to databases (ADO), filesystem, and use of components installed on the server.

```
<%
If (Len(Request.QueryString("name")) > 0) Then
    Response.Cookies("name") = Request.QueryString("name")
End If

Response.Write("Welcome " & Server.HTMLEncode(Response.Cookies("name")) & "!") 'this
script is NOT vulnerable to XSS, the input has been encoded using HTML Encoding
%>
<%
Dim oAdoCon, oAdoRec, oAdoStm, oCdoCon, oCdoMsg, oSciDic, oSciFsm, oMswAdr

Set oAdoCon = Server.CreateObject("ADODB.Connection")
Set oAdoRec = Server.CreateObject("ADODB.Recordset")
Set oAdoStm = Server.CreateObject("ADODB.Stream")
Set oCdoCon = Server.CreateObject("CDO.Configuration")
Set oCdoMsg = Server.CreateObject("CDO.Message")
Set oSciDic = Server.CreateObject("Scripting.Dictionary")
Set oSciFsm = Server.CreateObject("Scripting.FileSystemObject")
Set oMswAdr = Server.CreateObject("MSWC.AdRotator")
%>
```

The Application object

Stores global variables.

```
<%
Application("name") = "My ASP Application"
Response.Write("Welcome to " & Application("name") & "!")
%>
```

The Session object

Stores variables accessible only to a single visitor.

```
<%
If (Len(Request.QueryString("name")) > 0) Then
    Session("name") = Request.QueryString("name")
End If

Response.Write("Welcome " & Server.HTMLEncode(Session("name")) & "!") 'this script is
NOT vulnerable to XSS, the input has been encoded using HTML Encoding
%>
```

The Error object

Allows for the management of errors.

```
<%
On Error Resume Next
 Dim oError
Set oError = Server.GetLastError()

Response.Write("ASPCode: " & oError.ASPCode & "<br />")
Response.Write("ASPDescription: " & oError.ASPDescription & "<br />")
Response.Write("Category: " & oError.Category & "<br />")
Response.Write("Column: " & oError.Column & "<br />")
Response.Write("Description: " & oError.Description & "<br />")
```

```
Response.Write("File: " & oError.File & "<br />")
Response.Write("Line: " & oError.Line & "<br />")
Response.Write("Number: " & oError.Number & "<br />")
Response.Write("Source: " & oError.Source & "<br />")

If (Err.Number <> 0) Then
    Err.Clear
End If
%>
```

File system objects :

The FileSystemObject object is used to access the file system on a server.

This object can manipulate files, folders, and directory paths. It is also possible to retrieve file system information with this object.

The following code creates a text file (c:\test.txt) and then writes some text to the file:

```
<%
dim fs,fname
set fs=Server.CreateObject("Scripting.FileSystemObject")
set fname=fs.CreateTextFile("c:\test.txt",true)
fname.WriteLine("Hello World!")
fname.Close
set fname=nothing
set fs=nothing
%>
```

The FileSystemObject object's properties and methods are described below:

Methods

Method	**Description**
BuildPath	Appends a name to an existing path
CopyFile	Copies one or more files from one location to another
CopyFolder	Copies one or more folders from one location to another
CreateFolder	Creates a new folder
CreateTextFile	Creates a text file and returns a TextStream object that can be used to read from, or write to the file
DeleteFile	Deletes one or more specified files
DeleteFolder	Deletes one or more specified folders
DriveExists	Checks if a specified drive exists
FileExists	Checks if a specified file exists
FolderExists	Checks if a specified folder exists

GetAbsolutePathName	Returns the complete path from the root of the drive for the specified path
GetBaseName	Returns the base name of a specified file or folder
GetDrive	Returns a Drive object corresponding to the drive in a specified path
GetDriveName	Returns the drive name of a specified path
GetExtensionName	Returns the file extension name for the last component in a specified path
GetFile	Returns a File object for a specified path
GetFileName	Returns the file name or folder name for the last component in a specified path
GetFolder	Returns a Folder object for a specified path
GetParentFolderName	Returns the name of the parent folder of the last component in a specified path
GetSpecialFolder	Returns the path to some of Windows' special folders
GetTempName	Returns a randomly generated temporary file or folder

Session tracking and cookies

Session:

A Session refers to all the request that a single client makes to a server. A session is specific to the user and for each user a new session is created to track all the request from that user. Every user has a separate session and separate session variable is associated with that session. In case of web applications the default time-out value for session variable is 20 minutes, which can be changed as per the requirement.

Session ID:

A session ID is an unique identification string usually a long, random and alpha-numeric string, that is transmitted between the client and the server. Session IDs are usually stored in the cookies, URLs (in case url rewriting) and hidden fields of Web pages.

Session Tracking:

HTTP is stateless protocol and it does not maintain the client state. But there exist a mechanism called "Session Tracking" which helps the servers to maintain the state to track the series of requests from the same user across some period of time.

Types of Session Tracking

Mechanism for Session Tracking are:

a) Cookies
b) URL rewriting
c) Hidden form fields
d) SSL Sessions

Cookie

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With ASP, you can both create and retrieve cookie values.

ADO

- ADO is a Microsoft technology
- ADO stands for **A**ctiveX **D**ata **O**bjects
- ADO is a Microsoft Active-X component
- ADO is automatically installed with Microsoft IIS
- ADO is a programming interface to access data in a database

Accessing a Database from an ASP Page

The common way to access a database from inside an ASP page is to:

1. Create an ADO connectio
2. Create an ADO connection to a database
3. Open the database connection
4. Create an ADO recordset
5. Open the recordset
6. Extract the data you need from the recordset
7. Close the recordset
8. Close the connection

Server side active - X components:
A server-side software module constructed as an ActiveX component that is stored on a Windows client/server system or a Windows Web site. On a Web site, ActiveX Server Components can be called from Active Server Pages.
Web resources :
XML :

- XML stands for EXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML is a W3C Recommendation

The Difference Between XML and HTML

XML is not a replacement for HTML.

XML and HTML were designed with different goals:

- XML was designed to transport and store data, with focus on what data is
- HTML was designed to display data, with focus on how data looks

HTML is about displaying information, while XML is about carrying information.

Structure in data

Name spaces:

XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

This XML carries information about a table (a piece of furniture):

```
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

When using prefixes in XML, a so-called **namespace** for the prefix must be defined.

The namespace is defined by the **xmlns attribute** in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

In the example above, the xmlns attribute in the <table> tag give the h: and f: prefixes a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

DTD:

The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document.

A DTD defines the document structure with a list of legal elements and attributes.

DTD Newspaper Example

```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

]>
```

DOM methods:

XML DOM Properties

These are some typical DOM properties:

- x.nodeName - the name of x
- x.nodeValue - the value of x
- x.parentNode - the parent node of x
- x.childNodes - the child nodes of x
- x.attributes - the attributes nodes of x

Note: In the list above, x is a node object.

XML DOM Methods

- x.getElementsByTagName(*name*) - get all elements with a specified tag name
- x.appendChild(*node*) - insert a child node to x
- x.removeChild(*node*) - remove a child node from x

Note: In the list above, x is a node object.

Example

The JavaScript code to get the text from the first <title> element in books.xml:

txt=xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue

After the execution of the statement, txt will hold the value "Everyday Italian"

Explained:

- **xmlDoc** - the XML DOM object created by the parser.
- **getElementsByTagName("title")[0]** - the first <title> element
- **childNodes[0]** - the first child of the <title> element (the text node)
- **nodeValue** - the value of the node (the text itself)

UNIT V SERVLETS AND JSP

Introduction

The Java Servlet framework is a means of writing web server code in Java without the programmer having to deal with the HTTP protocol directly. Servlets are therefore a convenient means of generating dynamic web pages. Moreover, many web hosting companies nowadays provide cheap hosting plans that allow you to run servlets. They are able to do so even on fairly cheap plans because Java servlets are generally more efficient than some other ways of providing dynamic content, such as using perl scripts[1]. To write, compile and run a Java servlet, you will need the following (in addition to your usual JDK and Java development tools):

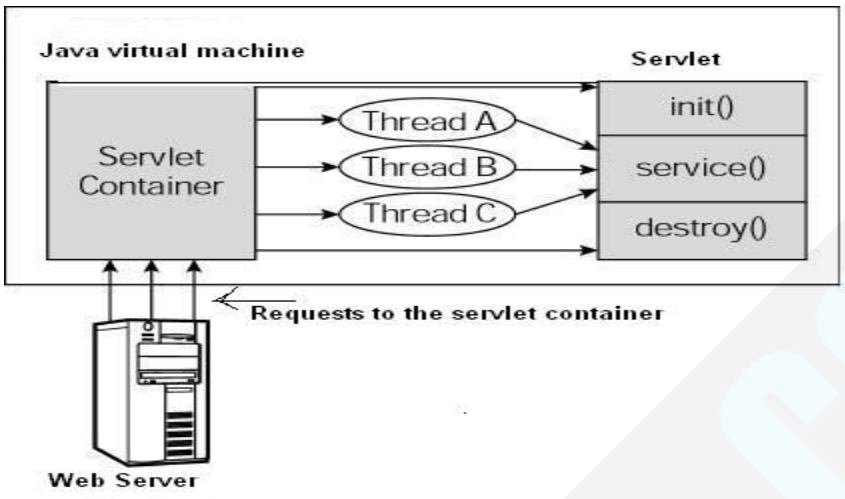
- an edition of the JDK that includes Servlet framework classes to compile against (and possibly test), typically Sun's **Java Enterprise Edition**;
- a server and installed web server software that supports Java servlets– often called a **servlet runner**– (provided ready-to-use on many shared hosting plans), such as Tomcat or Resin.

Servlet overview architecture

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.



Handling HTTP request :Get and post request

Handling GET and POST Requests

To handle HTTP requests in a servlet, extend the HttpServlet class and override the servlet methods that handle the HTTP requests that your servlet supports. This lesson illustrates the handling of GET and POST requests. The methods that handle these requests are doGet and doPost.

Handling GET requests

Handling GET requests involves overriding the doGet method. The following example shows the BookDetailServlet doing this. The methods discussed in the Requests and Responses section are shown in **bold**.

```
public class BookDetailServlet extends HttpServlet {

  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
    throws ServletException, IOException
  {
    ...
        // set content-type header before accessing the Writer
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

        // then write the response
    out.println("<html>" +
                "<head><title>Book Description</title></head>" +
                ...);

    //Get the identifier of the book to display
    String bookId = request.getParameter("bookId");
    if (bookId != null) {
      // and the information about the book and print it
      ...
    }
    out.println("</body></html>");
```

```
        out.close();
    }
    ...
}
```

The servlet extends the HttpServlet class and overrides the doGet method. Within the doGet method, the getParameter method gets the servlet's expected argument. To respond to the client, the example doGet method uses a Writer from the HttpServletResponse object to return text data to the client. Before accessing the writer, the example sets the content-type header. At the end of the doGet method, after the response has been sent, the Writer is closed.

Handling POST Requests

Handling POST requests involves overriding the doPost method. The following example shows the ReceiptServlet doing this. Again, the methods discussed in the Requests and Responses section are shown in **bold**.

```
 public class ReceiptServlet extends HttpServlet {


    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                throws ServletException, IOException
    {
        ...
        // set content type header before accessing the Writer
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // then write the response
        out.println("<html>" +
                    "<head><title> Receipt </title>" +
                    ...);

        out.println("<h3>Thank you for purchasing your books from us " +
                    request.getParameter("cardname") +
                    ...);
        out.close();
    }
    ...
}
```

The servlet extends the HttpServlet class and overrides the doPost method.

Within the doPost method, the getParameter method gets the servlet's expected argument.

To respond to the client, the example doPost method uses a Writer from the HttpServletResponse object to return text data to the client. Before accessing the writer the example sets the content-type header. At the end of the doPost method, after the response has been set, the Writer is closed.

Multi–tier applications:

Introduction

Multitier or 'N-tier' architecture is a software system segregated into separate sections (referred to as tiers or layers). The default set value of 'N' is three. It's an architectural methodology incorporated as a part of web based **software application development**.

The need

There are many reasons of using such layers to construct an application. Here they go:

- **Scalability**: A layered structure provides an optimal amount of scalability to the system. Any of the tiers can be upgraded or interchanged independently without affecting the functioning of others.
- **An ease in implementing changes**: A software system is subject to changes. This change can come in the form of a functionality change, a functionality enhancement, a completely new module, or even a hardware infrastructure enhancement, and so on. In any case, changes can occur during the initial development or after the first version is complete. By designing a system architecture as multitiered from the very beginning, one can minimize the impact on the whole system when implementing any change to any layer.
- **Maintenance ease**: If the source code is organized into multiple tiers, debugging and application maintenance as a whole becomes easier. As a developer it is easy for you to locate specific sections where an error is occurring or a change is needed and do the needful. A well-structured architecture brings an easy solution to defects and bugs and allows easy upgrades or enhancements, which otherwise would be very much time consuming.

The description of individual layers

A typical multitiered application consists of:

- *Entity Tier (your website application)*
- *Business Tier (the brain of your application)*
- *Data Tier (the data processing section of your application)*

Entity Tier: This is the presentation logic layer, something that the users view. This includes simple control and user input validation, is similar to a web application. It is also known as 'thin client.' It has a reference of the business layer. This layer has no knowledge about the system working, database, or any information.

Business Tier: This is the 'brain' of your entire application. It provides the business processes logic and the data access. It will act as a bridge from the presentation to the data access so that the information can be processed in some cases and, in other cases, can simply be the conduit to ensure the smooth flow of information. This layer will have the most information about the activity and the processing that needs to take place and will have a heavy implementation of the business objects.

Data Tier: This is the only layer of your application that knows how to interact with the database servers and retrieve the required information. Without any knowledge as to who is asking or whom is it giving the information to, all what this layer does is that it gives information back to the request object.

Data retrieval process is most likely in the form of selecting, querying, inserting, updating, or deleting information to and from the database.

JSP: JavaServer Pages

JavaServer Pages (JSP) technology is the Java platform technology for delivering dynamic content to web clients in a portable, secure and well-defined way. The JavaServer Pages specification extends the Java Servlet API to provide web application developers with a robust framework for creating dynamic web content on the server using HTML, and XML templates, and Java code, which is secure, fast, and independent of server platforms. JSP has been built on top of the Servlet API and utilizes Servlet semantics. JSP has become the preferred request handler and response mechanism.

.

Overview :

The JavaServer Pages 1.2 specification provides web developers with a framework to build applications containing dynamic web content such as HTML, DHTML, XHTML and XML. A JSP page is a text based document containing static HTML and dynamic actions which describe how to process a response to the client in a more powerful and flexible manner. Most of a JSP file is plain HTML but it also has, interspersed with it, special JSP tags.

There are many JSP tags such as:

- JSP directive denoted by <%@,

  2. scriplets indicated by <% ... %> tags and
- directive includes the contents of the file sample.html in the response at that point.

To process a JSP file, we need a JSP engine that can be connected with a web server or can be accommodated inside a web server. Firstly when a web browser seeks a JSP file through an URL from the web server, the web server recognizes the .jsp file extension in the URL requested by the browser and understands that the requested resource is a JavaServer Page. Then the web server passes the request to the JSP engine. The JSP page is then translated into a Java class, which is then compiled into a servlet.

.**Objects :**

Implicit objects in jsp are the objects that are created by the container automatically and the container makes them available to the developers, the developer do not need to create them explicitly. Since these objects are created automatically by the container and are accessed using standard variables; hence, they are called implicit objects. The implicit objects are parsed by the container and inserted into the generated servlet code. There are nine implicit objects. Here is the list of all the implicit objects:

Object	Class
application	javax.servlet.ServletContext
config	javax.servlet.ServletConfig
exception	java.lang.Throwable
out	javax.servlet.jsp.JspWriter
page	java.lang.Object
PageContext	javax.servlet.jsp.PageContext

- **Application:** These objects has an application scope. These objects are available at the widest context level, that allows to share the same information between the JSP page's servlet and any Web components with in the same application.
- **Config:** These object has a page scope and is an instance of javax.servlet.ServletConfig class. Config object allows to pass the initialization data to a JSP page's servlet. Parameters of this objects can be set in the deployment descriptor (web.xml) inside the element <jsp-file>. The method getInitParameter() is used to access the initialization parameters.

- **Exception:** This object has a page scope and is an instance of java.lang.Throwable class. This object allows the exception data to be accessed only by designated JSP "error pages."
- **Out:** This object allows us to access the servlet's output stream and has a page scope. Out object is an instance of javax.servlet.jsp.JspWriter class. It provides the output stream that enable access to the servlet's output stream.
- **Page:** This object has a page scope and is an instance of the JSP page's servlet class that processes the current request. Page object represents the current page that is used to call the methods defined by the translated servlet class. First type cast the servlet before accessing any method of the servlet through the page.
- **Pagecontext:** PageContext has a page scope. Pagecontext is the context for the JSP page itself that provides a single API to manage the various scoped attributes. This API is extensively used if we are implementing JSP custom tag handlers. PageContext also provides access to several page attributes like including some static or dynamic resource.
- **Request:** Request object has a request scope that is used to access the HTTP request data, and also provides a context to associate the request-specific data. Request object implements javax.servlet.ServletRequest interface. It uses the getParameter() method to access the request parameter. The container passes this object to the _jspService() method.
- **Response:** This object has a page scope that allows direct access to the **HTTPServletResponse** class object. Response object is an instance of the classes that implements the javax.servlet.ServletResponse class. Container generates to this object and passes to the _jspService() method as a parameter.
- **Session:** Session object has a session scope that is an instance of javax.servlet.http.HttpSession class. Perhaps it is the most commonly used object to manage the state contexts. This object persist information across multiple user connection

Scripting:

JSP Scripting Elements :Scriptlet, expression, declaration
There are three Scripting Elements

1. Expressions of the form <%= expression %> that are evaluated and inserted into the output,

A JSP expression is used to insert Java values directly into the output. It has the following form:
<%= Java expression %>
The Java expression is evaluated, converted to a string, and inserted in the page
Current time: <%= new java.util.Date() %>
predefined variables that you can use
request, the HttpServletRequest;
response, the HttpServletResponse;
session, the HttpSession associated with the request (if any); and
out, the PrintWriter (a buffered version of type JspWriter) used to send output to the client.
Example :Your hostname: <%= request.getRemoteHost() %>

2. Scriptlets of the form <% code %> that are inserted into the servlet's service method, and

If you want to do something more complex than insert a simple expression, JSP scriptlets insert code into the service() method of the generated servlet.
Scriptlets have the following form:
<% Java Code %> Scriptlets have access to the same automatically defined variables as

expressions
<%
String name = request.getParameter("name");
out.println("name: " + name);
%>

3. Declarations of the form <%! code %> that are inserted into the body of the servlet class, outside of any existing methods.
A JSP *declaration* define methods or fields that get inserted into the main body of the servlet class (outside of the service method processing the request). It has the following form:

<%! Java Code **%>**

or
<%!

public int getValue(){

return 78;
}
%>
The method we can use in Expression also.
<%=getValue()%>
Standard actions:

Standard Actions are actions that are defined in JSP itself, it need not be declared with a "taglib" directive. All the Standard Actions in JSP has a default prefix "jsp".

The following is the list of Standard Actions in JSP.

Action Element	Description
<jsp:setProperty>	Sets the JavaBeans property value
<jsp:include>	Includes the response from the jsp page while processing.
<jsp:forward>	Forwards the processing of a request to ajsp page
<jsp:param>	Adds a parameter value to a request handed off using the <jsp:include or <jsp:forward>
<jsp:plugin>	Generates a HTML elements appropriate to a browser to execute an applet using Java Plugin.
<jsp:attribute>	Sets the value of the action element based on the body of this element.
<jsp:body>	Sets action element body based on the body of this element.
<jsp:element>	Dynamically generates XML element.
<jsp:text>	Encapsulates template text, needed in JSP pages written in XML.

Example:

```
<html>
<body>
<jsp:forward page="index.jsp"/>
</body>
<html>
```

In the above exampe we use the Standard Tag "<jsp:forward page" to redirect to the "index.jsp" page.

– Directives:

JSP directives provide directions and instructions to the container, telling it how to handle certain aspects of JSP processing.

A JSP directive affects the overall structure of the servlet class. It usually has the following form:

<%@ directive attribute="value" %>

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

There are three types of directive tag:

Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page

The page Directive:

The **page** directive is used to provide instructions to the container that pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of page directive:

<%@ page attribute="value" %>

You can write XML equivalent of the above syntax as follows:

<jsp:directive.page attribute="value" />

Attributes:

Following is the list of attributes associated with page directive:

Attribute	Purpose
buffer	Specifies a buffering model for the output stream.
autoFlush	Controls the behavior of the servlet output buffer.
contentType	Defines the character encoding scheme.

errorPage	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
import	Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
info	Defines a string that can be accessed with the servlet's getServletInfo() method.
isThreadSafe	Defines the threading model for the generated servlet.
language	Defines the programming language used in the JSP page.
session	Specifies whether or not the JSP page participates in HTTP sessions
isELIgnored	Specifies whether or not EL expression within the JSP page will be ignored.
isScriptingEnabled	Determines if scripting elements are allowed for use.

The include Directive:

The **include** directive is used to includes a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code *include* directives anywhere in your JSP page.

The general usage form of this directive is as follows:

<%@ include file="relative url" >

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write XML equivalent of the above syntax as follows:

<jsp:directive.include file="relative url" />

The taglib Directive:

The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.

The taglib directive follows the following syntax:

<%@ taglib uri="uri" prefix="prefixOfTag" >

Where the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.