UNIT-1
Fundamentals

Introduction to the World Wide Web

The "Web", short for "World Wide Web" (which gives us the acronym www), is the name for one of the ways that the Internet lets people browse documents connected by hypertext links.

The concept of the Web was perfected at CERN (Centre EuropÃ©en de Recherche NuclÃ©aire) in 1991 by a group of researchers which included Tim-Berners Lee, the creator of the hyperlink, who is today considered the father of the Web.

The principle of the Web is based on using hyperlinks to navigate between documents (called "web pages") with a program called a browser. A web page is a simple text file written in a markup language (called HTML) that encodes the layout of the document, graphical elements, and links to other documents, all with the help of tags.

Besides the links which connect formatted documents to one another, the web uses the HTTP protocol to link documents hosted on distant computers (called web servers, as opposed to the client represented by the broswer). On the Internet, documents are identified with a unique address, called a URL, which can be used to locate any resource on the Internet, no matter which server may be hosting it.

What is a website?

A website (also called an Internet site or a home page in the case of a personal site) is a group of HTML files that are stored on a hosting computer which is permanently connected to the Internet (a web server).

A website is normally built around a central page, called a "welcome page", which offers links to a group of other pages hosted on the same server, and sometimes "external" links, which lead to pages hosted by another server.

A URL looks something like this:

http://en.kioskea.net/www/www-intro.php3
Let's take a closer look at this address:

http:// indicates that we want browse the web using the HTTP protocol, the default protocol for browsing the Web. There are other protocols for other uses of the Internet.
www.commentcamarche.net corresponds to the address of the server that hosts the web pages. By convention, web servers have a name that begins with www., to make it clear that they are dedicated web servers and to make memorising the address easier. This second part of the address is called the domain name. A website can be hosted on several servers, each belonging to the same name: www.commentcamarche.net www2.commentcamarche.net, intranet.commentcamarche.net, etc.
/www/www-intro.php3 indicates where the document is located on the machine. In this case, it is the file www-intro.php3 situÃ© located in the directory www.

Web client and web Server

A Client and a Server are two parts of a connection. In a web environment, these are two distinct machines, A Client is any machine that requests information, the Server is who the client makes the request too.
So a Web Server is basically a PC that is designed to accept requests from remote computers and send on the information requested.
A Web client is actually your browser. It is the browser on your PC/Mac that makes the requests to the remote server. A PC/Mac that uses a web (Client) browser is referred to as a Client Machine.

Web Resources

Web resources are any software artifacts that the web application requires for proper rendering, including images, script files, and any user-created component libraries. Resources must be collected in a standard location, which can be one of the following.

A resource packaged in the web application root must be in a subdirectory of a resources directory at the web application root: resources/resource-identifier.

A resource packaged in the web application's classpath must be in a subdirectory of the META-INF/resources directory within a web application: META-INF/resources/resource-identifier. You can use this file structure to package resources in a JAR file bundled in the web application. See Chapter 53, Duke's Forest Case Study Example for an application that uses this mechanism.

The JavaServer Faces runtime will look for the resources in the preceding listed locations, in that order.

Resource identifiers are unique strings that conform to the following format:

[locale-prefix/][library-name/][library-version/]resource-name[/resource-version]
Elements of the resource identifier in brackets ([]) are optional, indicating that only a resource-name, which is usually a file name, is a required element. For example, the most common way to specify a style sheet, image, or script is to use the library and name attributes, as in the following tag from the guessnumber example:

<h:outputStylesheet library="css" name="default.css"/>
This tag specifies that the default.css style sheet is in the directory web/resources/css.

You can also specify the location of an image using the following syntax, also from the guessnumber example:

<h:graphicImage value="#{resource['images:wave.med.gif']}"/>
This tag specifies that the image named wave.med.gif is in the directory web/resources/images.

Resources can be considered as a library location. Any artifact, such as a composite component or a template that is stored in the resources directory, becomes accessible to the other application components, which can use it to create a resource instance.

URL

URL stands for Uniform Resource Locator. A URL is nothing more than the address of a given unique resource on the Web. In theory, each valid URL points to a unique resource. Such resources can be an HTML page, a CSS document, an image, etc. In practice, there are some exceptions, the most common being a URL pointing to a resource that no longer exists or that has moved. As the resource represented by the URL and the URL itself are handled by the Web server, it is up to the owner of the web server to carefully manage that resource and its associated URL.

The Anatomy of a URL

Although each URL is a single string of numbers, letters, and special characters, each URL has four distinct components:

Protocol — always present
Hostname — always present
Path or Stem — always present…but sometimes is, basically, null
Parameters — optional (but this is where some of the real fun can happen)
Below is a fictitious URL with each of these components identified:

When a URL is "executed," a couple of things happen:

The magic of the internet occurs (browser mechanics, DNS resolution, etc.) to actually get that request routed all the way through the interwebtubes to a web server somewhere; the mechanics of this are beyond the scope of this post.
That web server interprets the request (the URL plus some other information that invisibly — and equally magically — comes along with it) and figures out what information needs to get sent back to the requestor

Message Format

When a client and a web service communicate they exchange messages. A request message is sent from the client to the web service. The web service responds with a response message. This is just like in ordinary HTTP, where a web browser sends an HTTP request to a web server, and the web server replies with an HTTP response.

web service message format available in these forms

SOAP
REST + XML
REST + JSON
XML RPC

Non-persistent and persistent connections

Non-persistent

HTTP/1.0

server parses request, responds, and closes TCP connection

2 RTTs to fetch each object

Each object transfer suffers from slow start

Persistent

default for HTTP/1.1

on same TCP connection: server, parses request, responds, parses new request,..

Client sends requests for all referenced objects as soon as it receives base HTML.

Fewer RTTs and less slow start.

Web Caching

A Web cache is a temporary storage place for files requested from the Internet. After an original request for data has been successfully fulfilled, and that data has been stored in the cache, further requests for those files (a Web page complete with images, for example) results in the information being returned from the cache rather than the original location.

We'll start with a simple analogy. This analogy is not a perfect match but it will give you the basic idea. Each morning before Joe goes to get the daily paper he asks his roommate, Bill, if he has already purchased one. If Bill already has the paper there is no reason for Joe to walk all the way to the store and spend money on the exact same information. He'll just read the paper that's already there. If a copy of the daily paper was already retrieved by Bill and is on hand, Joe saves his money and his time by not making a trip to the store.

Web caching enhances Web browsing in much the same way. When a user visits a site such as www.ala.org, Web caching (if in use and available) will retrieve the page from the ala.org W eb server and store a copy of that page on your local are network (LAN). The next time a user requests www.ala.org the W eb cache delivers the locally cached copy of the page. The user will experience a very fast download because the request did not have to traverse the entire Internet - the files all came from a local source. Also, the bandwidth that would normally be used to download the Web site is not required and is free for other information retrieval or delivery.

Caching is useful for any library. Faster response to users' requests and saved bandwidth are never a bad thing. Caching really makes sense for libraries that feel they must purchase more bandwidth to keep up with increased usage. In such cases a cache server or cache appliance could very likely lower the demand on the existing bandwidth, thus make a costly bandwidth upgrade unnecessary. Suppose an upgrade from a 256K data circuit to a full T1 will increase your monthly Internet bill by $500 . A $3,000 investment in caching (and caching solutions often can be implemented for much less than that) will start to show a return after six months.

Proxy

A proxy or proxy server is basically another computer which serves as a hub through which internet requests are processed. By connecting through one of these servers, your computer sends your requests to the proxy server which then processes your request and returns what you were wanting. In this way it serves as an intermediary between your home machine and the rest of the computers on the internet. Proxies are used for a number of reasons such as to filter web content, to go around restrictions such as parental blocks, to screen downloads and uploads and to provide anonymity when surfing the internet.

Why Use a Proxy?

If you are wanting to surf the web anonymously then proxies can provide you with a means to hide your home IP address from the rest of the world. By connecting to the internet through proxies, the home IP address of your machine will not be shown but rather the IP of the proxy server will be shown. This can provide you with more privacy then if you were simply connecting directly to the internet. There are number of proxies that can provide you with service. You can find a list of these simply by typing "Proxy List" into any search engine. There are some proxies which are free and some which charge money, the choice is up to you but we have found that the paid proxies are more reliable, faster and more secure.

Java and Net

The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols:

TCP: TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

UDP: UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

Networking Classes and interfaces

Java is a premier language for network programming. java.net package encapsulate large number of classes and interface that provides an easy-to use means to access network resources. Here are some important classes and interfaces of java.net package.

Some Important Classes

CLASSES
CacheRequest CookieHandler
CookieManager Datagrampacket
Inet Address ServerSocket
Socket DatagramSocket
Proxy URL

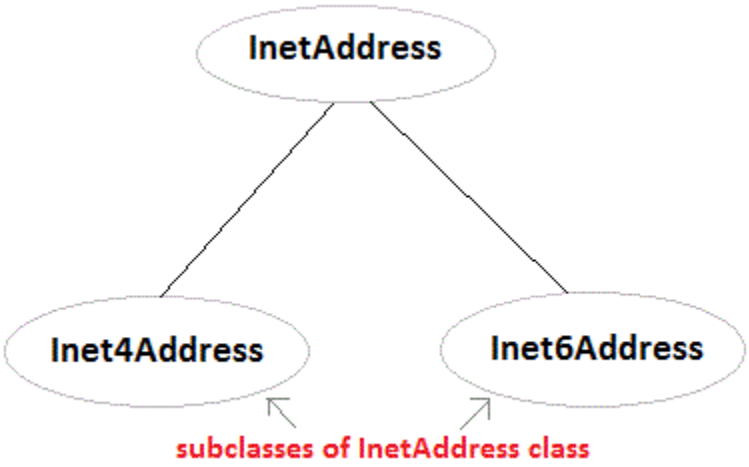
URLConnection
Some Important Interfaces

INTERFACES
CookiePolicy CookieStore
FileNameMap SocketOption
InetAddress ServerSocket
SocketImplFactory ProtocolFamily
InetAddress

Inet Address encapsulates both numerical IP address and the domain name for that address. Inet address can handle both IPv4 and Ipv6 addresses. Inet Address class has no visible constructor. To create an inet Address object, you have to use Factory methods.

Three commonly used Inet Address factory methods are.

static InetAddress getLocalHost() throws UnknownHostException
static InetAddress getByName (String hostname) throws UnknownHostException
static InetAddress[ ] getAllByName (String hostname) throws UnknownHostException



Lookingup internet address

Devices connected to the Internet are called nodes. Nodes that are computers are called hosts. Each node or host is identified by at least one unique 32-bit number called an Internet address, an IP address, or a host address, depending on who you talk to. This takes up exactly four bytes of memory. An IP address is normally written as four unsigned bytes, each ranging from to 255, with the most significant byte first. Bytes are separated by periods for the convenience of human eyes. For example, the address for hermes.oit.unc.edu is 152.2.21.1. This is called the dotted quad format.

IP addresses are great for computers, but they are a problem for humans, who have a hard time remembering long numbers. In the 1950s, it was discovered that most people could remember about seven digits per number; some can remember as many as nine, while others remember as few as five. This is why phone numbers are broken into three- and four-digit pieces with three-digit area codes.[13] Obviously an IP address, which can have as many as 12 decimal digits, is beyond the capacity of most humans to remember. I can remember about

two IP addresses, and then only if I use both daily and the second is a simple permutation of the first.

Client/Server Programs

The Server (typically)
An applications runs on a large computer (large meaning either fast or large storage space or both). The server application is usually written in such a way that it does not provide any method to interact with a user. Instead, it waits for other programs to connect (i.e. other programs take the place of a user) and interacts with these programs. Typically, a server application has control over large amounts of data, and can access that data fast and efficiently. It can also handle requests by many clients (more or less) simultaneously.
The Client (typically)
An application that runs on a personal computer. It has an extensive and appealing user interface, but it has no data that it can manipulate or present to the user. The client application 'asks' what a user wants, then connects to the server application to obtain that data. Once the data has been obtained, it is presented to the user in a nice format. A client usually gets the data from one server at a time, and interacts with one user only.
Example:
A client-server database program would work as follows: The database is stored on the computer where the server application is running. The server application can access each part of the data very fast, but a user can not interact with the server program. A person sits on another, comparatively small computer and starts the client program. It presents a convenient screen image, and the user enters, say, search criteria to search for a particular item in the database. Once the criteria has been entered in the client program, it connects to the server program and presents the request. The server program searches for the appropriate data in the database and delivers it (in a machine-suitable, often compressed) format. The client then 'decodes' the information and displays it nicely on the screen.

SOCKET PROGRAMMING

The two key classes from the java.net package used in creation of server and client programs are:

1. ServerSocket

2. Socket

A server program creates a specifi c type of socket that is used to listen for client requests (server socket), In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the fi le concept: developers have to open a socket, perform I/O, and close it. Figure 13.5 illustrates key steps involved in creating socket-based server and client programs.

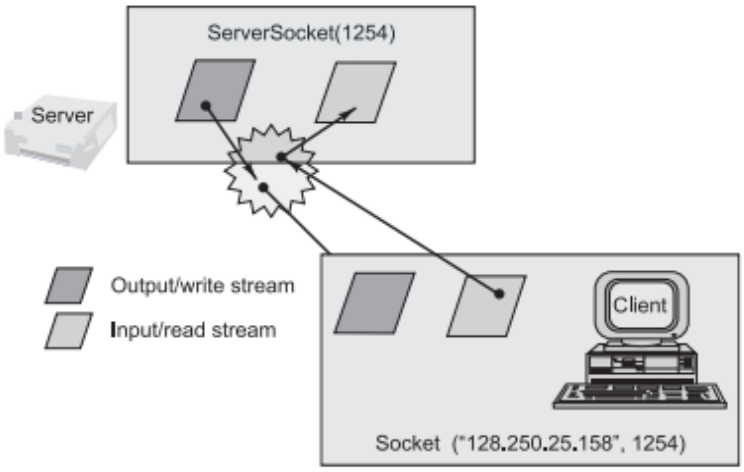


Fig. 13.5 Socket-based client and server programming

A simple Server Program in Java .The steps for creating a simple server program are:

1. Open the Server Socket:

ServerSocket server = new ServerSocket(PORT);

2. Wait for the Client Request:

Socket client = server.accept();

3.Create I/O streams for communicating to the client

DataInputStream is = new DataInputStream(client.getInputStream());

DataOutputStream os = new DataOutputStream(client.getOutputStream());

4. Perform communication with client Receive from client:

String line = is.readLine(); Send to client: os.writeBytes("Hello\n");

5. Close socket: client.close();

An example program illustrating creation of a server socket, waiting for client request, and then responding to a client that requested for connection by greeting it is given below:

Program

```
// SimpleServer.java: A simple server program.
import java.net.*;
import java.io.*;
```

```
public class SimpleServer {
 public static void main(String args[]) throws IOException {
 // Register service on port 1254
 ServerSocket s = new ServerSocket(1254);
 Socket s1=s.accept(); // Wait and accept a connection
 // Get a communication stream associated with the socket
 OutputStream s1out = s1.getOutputStream();
 DataOutputStream dos = new DataOutputStream (s1out);
 // Send a string!
 dos.writeUTF("Hi there");
 // Close the connection, but not the server socket
 dos.close();
 s1out.close();
 s1.close();
 }
}
```

A simple Client Program in Java The steps for creating a simple client program are:

1. Create a Socket Object: Socket client = new Socket(server, port_id);

2. Create I/O streams for communicating with the server.

is = new DataInputStream(client.getInputStream());

os = new DataOutputStream(client.getOutputStream());

3. Perform I/O or communication with the server: Receive data from the server:

String line = is.readLine(); Send data to the server:

os.writeBytes("Hello\n");

4. Close the socket when done: client.close();

An example program illustrating establishment of connection to a server and then reading a message sent by the server and displaying it on the console is given below

Program

// SimpleClient.java: A simple client program.

```
import java.net.*;

import java.io.*;

public class SimpleClient {

 public static void main(String args[]) throws IOException {

 // Open your connection to a server, at port 1254

 Socket s1 = new Socket("localhost",1254);

 // Get an input file handle from the socket and read the input

 InputStream s1In = s1.getInputStream();

 DataInputStream dis = new DataInputStream(s1In);

 String st = new String (dis.readUTF());

 System.out.println(st);

 // When done, just close the connection and exit

 dis.close();

 s1In.close();

 s1.close();

 }

}
```

Running Socket Programs Compile both server and client programs and then deploy server program

code on a machine which is going to act as a server and client program, which is going to act as a client.

If required, both client and server programs can run on the same machine. To illustrate execution of server and client programs, let us assume that a machine called mundroo.csse.unimelb.edu.au on which we want to run a server program as indicated below:

[raj@mundroo] java SimpleServer

The client program can run on any computer in the network (LAN, WAN, or Internet) as long as there

is no fi rewall between them that blocks communication. Let us say we want to run our client program on a machine called gridbus.csse.unimelb.edu.au as follows:

[raj@gridbus] java SimpleClient

The client program is just establishing a connection with the server and then waits for a message. On

receiving a response message, it prints the same to the console. The output in this case is:

Hi there

which is sent by the server program in response to a client connection request.

It should be noted that once the server program execution is started, it is not possible for any other server program to run on the same port until the fi rst program which is successful using it is terminated. Port numbers are a mutually exclusive resource. They cannot be shared among different processes at the same time

e-mail client

An application that runs on a personal computer or workstation and enables you to send, receive and organize e-mail. It's called a *client* because e-mail systems are based on a client-server architecture . Mail is sent from many clients to a central server, which re-routes the mail to its intended destination.

POP3 Programs

POP3, which is an abbreviation for Post Office Protocol 3, is the third version of a widespread method of receiving email. Much like the physical version of a post office clerk, POP3 receives and holds email for an individual until they pick it up. And, much as the post office does not make copies of the mail it receives, in previous versions of POP3, when an individual downloaded email from the server into their email program, there were no more copies of the email on the server; POP automatically deleted them.

POP3 makes it easy for anyone to check their email from any computer in the world, provided they have configured their email program properly to work with the protocol

REMOTE METHOD INVOCATION

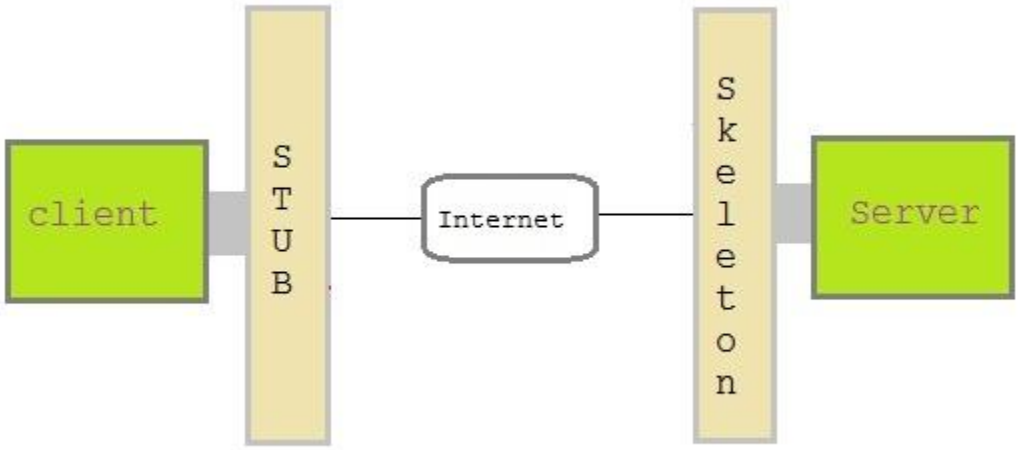
Remote method invocation(RMI) allow a java object to invoke method on an object running on another machine. RMI provide remote communication between java program. RMI is used for building distributed application.

Concept of RMI application

A RMI application can be divided into two part,**Client** program and **Server** program. A **Server** program creates some remote object, make their references available for the client to invoke method on it. A **Client**program make request for remote objects on server and invoke method on them. **Stub** and **Skeleton** are two important object used for communication with remote object.

Stub and Skeleton

Stub act as a gateway for Client program. It resides on Client side and communicate with **Skeleton** object. It establish the connection between remote object and transmit request to it.



Skeleton object resides on server program. It is responsible for passing request from **Stub** to remote object.

Creating a Simple RMI application involves following steps

- Define a remote interface.
- Implementing remote interface.
- create and start remote application
- create and start client application

Define a remote interface

A remote interface specifies the methods that can be invoked remotely by a client. Clients program communicate to remote interfaces, not to classes implementing it. To be a remote interface, a interface must extend the **Remote** interface of **java.rmi** package.

```
import java.rmi.*;

public interface AddServerInterface extends Remote

{

public int sum(int a,int b);

}
```

Implementation of remote interface

For implementation of remote interface, a class must either extend **UnicastRemoteObject** or use exportObject() method of **UnicastRemoteObject** class.

```
import java.rmi.*;
import java.rmi.server.*;
public class Adder extends UnicastRemoteObject implements AddServerInterface
{
Adder()throws RemoteException{
super();
}
public int sum(int a,int b)
{
return a+b;
}
}
```

Create AddServer and host rmi service

You need to create a server application and host rmi service **Adder** in it. This is done using rebind()method of **java.rmi.Naming** class. rebind() method take two arguments, first represent the name of the object reference and second argument is reference to instance of **Adder**

```
import java.rmi.*;
import java.rmi.registry.*;
public class AddServer{
public static void main(String args[]){
try{
AddServerInterface addService=new Adder();
Naming.rebind("AddService",addService);
//addService object is hosted with name AddService.

}catch(Exception e){System.out.println(e);}
}
}
```

Create client application

Client application contains a java program that invokes the lookup() method of the **Naming** class. This method accepts one argument, the **rmi** URL and returns a reference to an object of type**AddServerInterface**. All remote method invocation is done on this object.

```
import java.rmi.*;
public class Client{
public static void main(String args[]){
try{
AddServerInterface st=(AddServerInterface)Naming.lookup("rmi://"+args[0]+"/AddService
");
System.out.println(st.sum(25,8));
}catch(Exception e){System.out.println(e);}
}
}
```

Steps to run this RMI application

Save all the above java file into a directory and name it as "rmi"

compile all the java files

- javac *.java
- compile all the java files

```
javac *.java
```

- Start RMI registry

```
start rmiregistry
```

- Run Server file

```
java AddServer
```

- Run Client file in another command prompt abd pass local host port number at run time

```
java Client 127.0.0.1
33
```

Unit-2
HTML

HTML and its Flavors

HTML (Hypertext Markup Language) is the language of the web - every website out there is written in some kind of HTML. Because of the rapid evolution of the web, though, HTML grew quickly in a very unplanned way, which can lead to problems if you're not sure what kind or version of HTML you're using.
Here's a quick history of HTML's flavors so far.
The first version of HTML was created by the web's inventor, Tim Berners-Lee, and was loosely based on an existing standard called SGML (Standardised General Markup Language). This very first version didn't have an img tag, which meant that no graphics at all could appear on web pages. Berners-Lee informally extended the language, but didn't standardize it.

As the web grew, the lack of standardization started to make it difficult for web browsers to interact - one web browser might have a new tag that others didn't support, meaning that people would see pages completely differently depending on which browser they used. In 1995, HTML was formalized as a standard named HTML 2, which was the version that the first mass-market web browsers were based on.

As they extended the standard further, an HTML 3 was introduced in 1997 to keep up-to-date. HTML 4 was introduced later that year as an effort to clean up the standard, making it clear that some tags should no longer be used. Apart from a few minor fixes in 1999, this is the version of HTML that is still in use today.

DHTML

Parallel to this development, though, other languages were being developed that could be included in HTML documents: languages like Javascript (for interactive pages) and CSS (for styling). DHTML (Dynamic HTML) was the name given to the combination of HTML and these technologies. To put it simply, HTML is for web pages while DHTML is for 'web applications'. As people start to do more and more things on the web that they used to do with separate programs, DHTML techniques are becoming ever-more popular.

XHTML

Sometimes considered 'next-generation HTML', XHTML is a stricter version of HTML that makes it follow XML standards. XML (eXtensible Markup Language) is a standard for HTML-like languages that is being used for more and more purposes, including configuration and sharing data.

Stripped of the technical talk, XHTML can basically be thought of as a stricter version of HTML. Where HTML is often messy and hard to test, XHTML is strictly standardised and can be run through automatic 'validators' that will point out any errors you've made. This improves cross-browser compatibility and makes web pages much easier to maintain, since it mostly forces information on the style of the page to be separated from the actual text of the page.

XHTML exists in a few different versions: there is a 'transitional' version, which lets you keep using some old practices from HTML4, and there is a 'strict' version, which is the one you need to use to get most of XHTML's benefits.

Web Page Structure

So now we know some basic facts about web pages:

web pages are text files with an ".html" suffix and HTML commands / tags
HTML commands are written in angle brackets < >
Most HTML tags have a beginning and ending command; the ending has a backslash
Most HTML tags have attributes which give specific information
Attributes are not added to ending commands
In HTML, the "Enter" key will not make a new line in the web page
In HTML, more than one space will not be visible unless you use the special character
"Enter"s and spaces will be visible if you put them inbetween <PRE> and </PRE> tags

EXAMPLE

```
<HTML>

<HEAD>
<TITLE>
Welcome to My Web Page!
</TITLE>
</HEAD>

<BODY>
<BR><BR><BR><BR>
<CENTER>
How do you like <B>my web page</B> so far?
<HR>
<P><P>
Are you <EM>having fun</EM> yet?
</CENTER>
</BODY>

</HTML>
```

The Web Model

The web is a very general concept -- one universal space of information. The concepts it requires such as identifiers and information resources (documents) are as general and abstract as possible. However, there have been some design decisions made which define some interfaces, and effectively define modules or agents which are independent. These agents are independent in many ways

There is knowledge they have individually but do not share
There is knowledge their designers had individually but did not share
This is basic modularity. The interfaces are defined by the data formats and protocols, and the important features to understand about the design I have ranted about in the linked articles in this series. This modularity, ability for different parts of the system, shows up when different specs are independent, such that you could change one without having to change the other.

Advantages of CSS:

1) Increased website reach to different technologies used for accessing the Internet
2) Improved web page download times
3) Bandwidth savings
4) Cross-browser compatibility
5) Higher search engine rankings
6) Easier website management
7) Web page print friendly

Adding CSS:

Following are the four methods to add CSS to HTML documents.

1. Inline Style Sheets
2. Embedded Style Sheets
3. External Style Sheets
4. Imported Style Sheets

Inline Style Sheets

Inline Style Sheets are included with HTML element i.e. they are placed inline with the element. To add inline CSS, we have to declare style attribute which can contain any CSS property.

Syntax:

```
<Tagname STYLE = " Declaration1 ; Declaration2 "> …. </Tagname>
```

Embedded Style Sheets

Embedded Style Sheets are used to apply same appearance to all occurrence of a specific element. These are defined in <head> element by using the **<style>** element.

Syntax

```
<head> <title> …. </title>
<style type =”text/css”>
…….CSS Rules/Styles….
</head>
```

External Style Sheets

External Style Sheets are the separate **.css** files that contain the CSS rules. These files can be linked to any HTML documents using <link> tag with rel attribute.

Syntax:

```
<head> <link rel= "stylesheet"  type="text/css" href= "url of css file">
</head>
```

In order to create external css and link it to HTML document, follow the following steps:

- First of all create a CSS file and define all CSS rules for several HTML elements. Let's name this file as external.css.
- Now create HTML document

Imported Style Sheets

Imported Style Sheets allow us to import style rules from other style sheets. To import CSS rules we have to use @import before all the rules in a style sheet.

Syntax:

```
<head><title> Title Information </title>
<style type="text/css">
@import URL (cssfilepath)
… CSS rules…
</style>
</head>
</style>
```

Browser compatibility:

By using CSS for layout and valid markup (HTML or XHTML), website will work well at different screen sizes such as 640x480, 800x600, 1024x768, and 1152x864. We won't see any horizontal scrolling that can get with tabular layouts.The site will work well on all browsers such as Internet Explorer, Firefox, Opera, Maxthon, Safari, Netscape, Konqueror, JAWS screen reader, and on PDAs, and screen sizes ranging from 600px to 1600px in width.

CSS and page layout:

We can use CSS to specify various visual effects that change the layout of your document.

For example, in HTML you can use the element to create structure.

Example

In sample document, the numbered paragraphs under the second heading do not have a container of their own. Style sheet cannot draw a border around those paragraphs, because there is no element to specify in the selector.

To fix this structural problem, you can add a tag around the paragraphs. This tag is unique, so it can be identified by an id attribute:

```
<h3>Numbered paragraphs</h3>
<div id="numbered">
  <p>Lorem ipsum</p>
  <p>Dolor sit</p>
  <p>Amet consectetuer</p>
  <p>Magna aliquam</p>
  <p>Autem veleum</p>
</div>
```

Now stylesheet can use one rule to specify borders around both lists:

```
ul, #numbered {

border: 1em solid #69b;

padding-right:1em;

}
```

Selectors:

CSS has its own terminology to describe the CSS language. Previously in this tutorial, you created a line in your stylesheet like this:

```
strong {

  color: red;

}
```

In CSS terminology, this entire line is a *rule*. This rule starts with strong, which is a *selector*. It selects which elements in the DOM the rule applies to.
In addition to tag names, we can use attribute values in selectors. This allows rules to be more specific.Two attributes have special status for CSS. They are class and id.

Class selectors

Use the class attribute in an element to assign the element to a named class. It is up to you what name you choose for the class. Multiple elements in a document can have the same class value.
In stylesheet, type a full stop (period) before the class name when you use it in a selector.

ID selectors

Use the id attribute in an element to assign an ID to the element. It is up to you what name you choose for the ID. The ID name must be unique in the document.
In style sheet, type a number sign (hash) before the ID when you use it in a selector.

UNIT-3

JAVA SCRIPT

Introduction:

JavaScript is most commonly used as a client side scripting language. This means that JavaScript code is written into an HTML page. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do something with it.

JavaScript is *not* the same as Java.

JavaScript is a client side, interpreted, object oriented, high level scripting language, while Java is a client side, , object oriented high level language.

Although the names are much alike, JavaScript is primarily a scripting language for use within HTML pages, while Java is a real programming language that does quite different things from JavaScript. In addition Java is much harder to learn. It was developed by Sun for use in pretty much anything that needs some computing power.

JavaScript was developed by Brendan Eich, then working at Netscape, as a client side scripting languageOriginally the language was called Live Script, but when it was about to be released Java had become immensely popular so Netscape changed the name of its scripting language to "JavaScript".

Java and JavaScript both descend from C and C++, but the languages. Both are object orientedand they share some syntax, but the differences are more important than the similarities.. JavaScript is not a programming language. it is a scripting language because it uses the browser to do the work.

Uses of JavaScript

- Use it to add multimedia elements

With JavaScript you can show, hide, change, resize images, and create image rollovers. You can create scrolling text across the status bar.

- Create pages dynamically

Based on the user's choices, the date, or other external data, JavaScript can produce pages that are customized to the user.

- Interact with the user

It can do some processing of forms and can validate user input when the user submits the form.

Variables :

Programmers use variables to store values. A variable can hold several types of data. In JavaScript you don't have to declare a variable's data type before using it. Any variable can hold any JavaScript data type, including:

• String data

• Numbers

• Boolean values (T/F)

Variable Names:

There are rules and conventions in naming variables in any programming language. It is good practice to use descriptive names for variables. The following are the JavaScript rules:

• The variable name must start with a letter or an underscore.

Eg: firstName or _myName

• You can use numbers in a variable name, but not as the first character. name01 or tuition$
• You can't use space to separate characters.

Eg: userName not user Name

• Capitalize the first letter of every word except the first

Eg: salesTax or userFirstName Variables

• To declare variables, use the keyword var and the variable name:

Eg: var userName 7

• To assign values to variables, add an equal sign and the value:

Eg: var userName = "Smith" var price = 100

Literals

Literals are used to represent values in JavaScript. These are fixed values, not variables, that you *literally* provide in your script. The following are the types of literals:

- Array literals
- Boolean literals
- Floating-point literals
- Integers
- Object literals
- String literals

JavaScript : Array literals

In Javascript an array literal is a list of expressions, each of which represents an array element, enclosed in a pair of square brackets ' [ ] ' . When an array is created using an array literal, it is initialized with the specified values as its elements, and its length is set to the number of arguments specified. If no value is supplied it creates an empty array with zerolength.

Creating an empty array :
var fruits= [ ];

Creating an array with four elements:
var fruits = ["Orange", "Apple", "Banana", "Mango"]

Comma in array literals

There is no need to specify all elements in an array literal. If we put two commas in a row at any position in an array then an unspecified element will be created in that place.

The following example creates the fruits array :

fruits = ["Orange", , "Mango"]

This array has one empty element in the middle and two elements with values. (fruits[0] is "Orange", fruits[1] is set to undefined, and fruits[2] is "Mango").

If you include a single comma at the end of the elements, the comma is ignored. In the following example, the length of the array is three. There is no fruits[2].

fruits = ["Orange", "Mango",]

In the following example, the length of the array is four, and fruits[0] and fruits[2] are undefined.

fruits = [, 'Apple', , 'Orange'];

JavaScript : Integers literals

Description

An **integer** must have at least one digit (0-9).
- No comma or blanks are allowed within an integer.
- It does not contain any fractional part.
- It can be either positive or negative, if no sign precedes it is assumed to be positive.

In JavaScript, integers can be expressed in three different bases.

1. Decimal (base 10)
Decimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and there will be no leading zeros.

Example : 123, -20, 12345

2. Hexadecimal (base 16)
Hexadecimal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and letters A, B, C, D, E, F or a, b, c, d, e, f. A leading 0x or 0X indicates the number is hexadecimal.

Example : 7b, -14, 3039

3. Octal (base 8)
Octal numbers can be made with the digits 0, 1, 2, 3, 4, 5, 6, 7. A leading 0 indicates the number is octal.

Example : 173, -24, 30071

JavaScript : Floating number literals

Description

A floating number has the following parts.

- A decimal integer.
- A decimal point ('.').
- A fraction.
- An exponent.

The exponent part is an "e" or "E" followed by an integer, which can be signed (preceded by "+" or "-").

Example of some floating numbers :

- 8.2935
- -14.72
- 12.4e3 [Equivalent to 12.4×10^3]
- 4E-3 [Equivalent to 4×10^{-3} => .004]

JavaScript : Boolean literals

The Boolean type has two literal values :

- true
- false

JavaScript : Object literals

Description

An object literal is zero or more pairs of comma separated list of property names and associated values, enclosed by a pair of curly braces.

In JavaScript an object literal is declared as follows:

1. An object literal without properties:

```
var userObject = {}
```

2. An object literal with a few properties :

```
var student = {
First-name : "Suresy",
Last-name : "Rayy",
Roll-No : 12
};
```

Syntax Rules

Object literals maintain the following syntax rules:

- There is a colon (:) between property name and value.
- A comma separates each property name/value from the next.
- There will be no comma after the last property name/value pair.

JavaScript : String literals

JavaScript has its own way to deal with string literals. A string literal is zero or more characters, either enclosed in single quotation (') marks or double **quotation** (") marks. You can also use + operator to join strings. The following are the examples of string literals :

- string1 = "w3resource.com"
- string1 = 'w3resource.com'
- string1 = "1000"
- string1 = "google" + ".com"

In addition to ordinary characters, you can include special characters in strings, as shown in the following table.

List of special characters used in JavaScript string :

Character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\'	Single quote
\"	Double quote
\\	Backslash character (\)
\XXX	The character with the Latin-1 encoding specified by up to three octal digits XXX between 0 and 377. For example, \100 is the octal sequence for the @ symbol.
\xXX	The character with the Latin-1 encoding specified by the two hexadecimal digits XX between 00 and FF. For example, \x40 is the hexadecimal sequence for the @ symbol.
\uXXXX	The Unicode character specified by the four hexadecimal digits XXXX. For example, \u0040 is the Unicode sequence for the @ symbol.

string1 = "First line. \n Second line."

Operators:

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators
- JavaScript supports the following arithmetic operators –

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Eg:var x= 5;
var y= 2;
var z = x * y;

Comparision and Logical Operators:

Operator	Description
==	equal to
===	equal value and equal type

!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Assignment Operators:

The **assignment** operator (=) assigns a value to a variable.

Operator	**Example**	**Same As**
=	x = y	x = y
+=	x += y	x = x + y

-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Conditional (or ternary) Operators (? :):

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No	Operator and Description
1	**? : (Conditional)** If Condition is true? Then value X : Otherwise value Y

Conditional statements:

In JavaScript we have the following conditional statements:

- Use **if** to specify a block of code to be executed, if a specified condition is true
- Use **else** to specify a block of code to be executed, if the same condition is false
- Use **else if** to specify a new condition to test, if the first condition is false
- Use **switch** to specify many alternative blocks of code to be executed

The if Statement

Use the **if** statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax:

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

Eg:

```
if (hour < 18) {
   greeting = "Good day";
}
```

The else Statement

Use the **else** statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
   block of code to be executed if the condition is true
} else {
   block of code to be executed if the condition is false
}
```

Eg:

```
if (hour < 18) {
   greeting = "Good day";
} else {
   greeting = "Good evening";
}
```

The else if Statement

Use the **else if** statement to specify a new condition if the first condition is false.

Eg:

```
if (time < 10) {
   greeting = "Good morning";
} else if (time < 20) {
   greeting = "Good day";
} else {
   greeting = "Good evening";
}
```

Switch Statement

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

Syntax:

```
switch(expression) {
   case n:
      code block
      break;
   case n:
      code block
      break;
   default:
```

```
    default code block
}
```

Eg:
```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
    break;
}
```

Control structure:

JavaScript supports different kinds of loops:

- **for** - loops through a block of code a number of times
- **for/in** - loops through the properties of an object
- **while** - loops through a block of code while a specified condition is true
- **do/while** - also loops through a block of code while a specified condition is true
- **The For Loop**
- The for loop has the following syntax:
- for(*statement1*;*statement2*;*statement3*){
  code block to be executed
  }
- **Statement 1** is executed before the loop (the code block) starts.
- **Statement 2** defines the condition for running the loop (the code block).
- **Statement 3** is executed each time after the loop (the code block) has been executed.

The For/In Loop

The JavaScript for/in statement loops through the properties of an object:

Example:
```
var person = {fname:"John", lname:"Doe", age:25};
var text = "";
var x;
```

```
for (x in person) {
    text += person[x];
}
```

The While Loop

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

Syntax:

```
while (condition) {
    code block to be executed
}
```

Eg:

```
while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

```
do {
    code block to be executed
}
while (condition);
```

Eg:

```
do {
    text += "The number is " + i;
    i++;
}
while (i < 10);
```

Arrays:

JavaScript arrays are used to store multiple values in a single variable. An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Syntax:

var *array-name* = [*item1*, *item2*, ...];

Example:

```
var cars = ["Saab", "Volvo", "BMW"];
<p id="demo"></p>
<script>
```

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
</script>
```

The first line (in the script) creates an array named cars.The second line "finds" the element with id="demo", and "displays" the array in the "innerHTML" of it.

Using the JavaScript Keyword new

The following example also creates an Array, and assigns values to it:

Example:

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

This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

This statement modifies the first element in cars:

```
cars[0] = "Opel";
```

Functions:

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

Syntax:

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

Function **parameters** are the **names** listed in the function definition.

Function **arguments** are the real **values** received by the function when it is invoked.

Inside the function, the arguments behave as local variables.

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

When JavaScript reaches a **return statement**, the function will stop executing.

- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":

Example:

Calculate the product of two numbers, and return the result:

```
var x = myFunction(4, 3);        // Function is called, return value will end up in x

function myFunction(a, b) {
    return a * b;                // Function returns the product of a and b
}
```

Objects:

In JavaScript, almost "everything" is an object. In JavaScript, all values, except primitive values, are objects.

Objects are variables too. But objects can contain many values.

The values are written as **name : value** pairs (name and value separated by a colon).

Example:
```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Objects are variables too. But objects can contain many values.

The values are written as **name : value** pairs (name and value separated by a colon).

Example:
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Methods:

Methods are **actions** that can be performed on objects.

Object properties can be both primitive values, other objects, and functions.

An **object method** is an object property containing a **function definition**.

Creating a JavaScript Object:

Eg:
```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Predefined objects:

Some Predefined Objects (with methods and properties)

1) window object

methods.

1. alert (str): displays an alert D.B. containing the string argument as text.
2. confirm(str): displays a confirm D.B. containing the string argument as text. you need either OK(confirm) or cancel. returns a boolean value.
3. prompt (str,str) displays a prompt D.B. containing the first argument string as an instruction on what to enter. The second argument is the default to be returned in case the user does not enter a string.

properties.

1. closed contains a boolean value corresponding to whether or not the window is closed.
2. document an object itself
3. history an object itself

2) Image[] : a property of the document object. This array is automatically created to index all of images that have been marked up with the HTML IMG tags. A new image can be constructed with the Image() constructor to preload it.

properties:

1. src contains the image's source URL
2. border contains the image's border setting in pixels
3. height contains the image's height in pixels
4. width contains the image's width in pixels
5. name can be set with the NAME attribute of the HTML IMG tag.can be declared when a new image is constructed with Image() constructor.

3) Date New Date objects can be created with Date() constructor as in: var d=new Date();

methods

1. getDate() returns the day of the month (1-31)
2. getDay() returns the day of the week (1-7)
3. getHours() returns the hours (0-23)
4. getMonth (1-12)
5. getSeconds() (0-59)
6. getYear() returns the year.

4) String This object is created automatically for each string. It is not a property of any object.

methods

1. charAt(ind) returns the character in the string at given ind
2. charCodeAt(ind) returns the ASCII number for the character at specified index of the string.

3. indexOf(char) returns the first index of the given character
   returns -1 if no such character
4. lastIndexOf(char)
   returns the last index at which the char is found.
   -1 is returned if not found.
5. substring(ind)
   returns a substring starting from the given index.
6. toLowerCase()
7. toUpperCase()

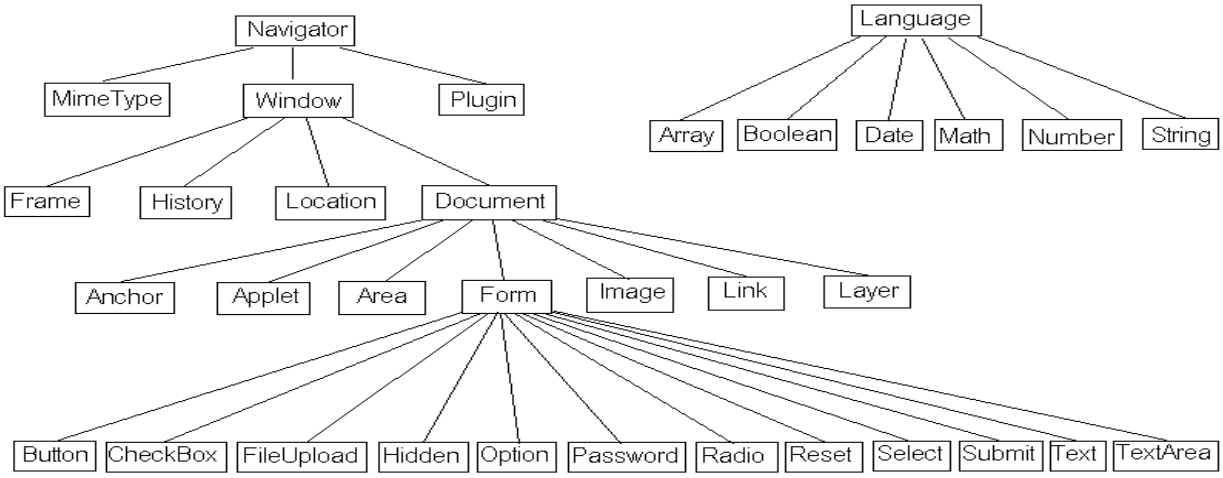
Object hierarchy:

Many JavaScript objects are contained within each other. JavaScript objects have a container to contained object relationship rather than a class and subclass relationship. Properties are not inherited from one type of object to another. There are two main types of JavaScript objects.

- Language Objects - Objects provided by the language and are not dependent on other objects.
- Navigator - Objects provided by the client browser. These objects are all sub objects to the navigator object.

Beyond that, objects are those created by the programmer.

JavaScript Object Hierarchy



Accessing objects:

We can access object properties in two ways:

objectName.propertyName

or

objectName["propertyName"]

```
Eg: var person = {
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue"
};
person.lastName;   (or)
person["lastName"];
```

We access an object method with the following syntax:

objectName.methodName()

Eg: name = person.fullName();

Events & Event handlers:

HTML events are **"things"** that happen to HTML elements.

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

- An HTML web page has finished loading
- An HTML input field was changed
- An HTML button was clicked

Often, when events happen, you may want to do something. JavaScript lets us execute code when events are detected.

An event handler executes a segment of a code based on certain events occurring within the application, such as onLoad, onClick. JavaScript event handers can be divided into two parts: interactive event handlers and non-interactive event handlers. An interactive event handler is the one that depends on the user interactivity with the form or the document. For example, onMouseOver is an interactive event handler because it depends on the users action with the mouse. On the other hand non-interactive event handler would be onLoad, because this event handler would automatically execute JavaScript code without the user's interactivity. Here are all the event handlers available in JavaScript:

Event Handler	Used In
onAbort	image
onBlur	select, text, text area

onChange	select, text, textarea
onClick	button, checkbox, radio, link, reset, submit, area
onError	image
onFocus	select, text, testarea
onLoad	windows, image
onMouseOver	link, area
onMouseOut	link, area
onSelect	text, textarea
onSubmit	form
onUnload	window

HTML allows event handler attributes, **with JavaScript code**, to be added to HTML elements.

Example:
<button onclick='getElementById("demo").innerHTML=Date()'>The time is?</button>

In the example above, the JavaScript code changes the content of the element with id="demo".

Multiple windows and Frames:

Most of the client-side JavaScript examples we've seen so far have involved only a single window or frame. In the real world, JavaScript applications often involve multiple windows or frames. frames within a window are represented by Window objects; JavaScript makes little distinction between windows and frames. In the most interesting applications, there is JavaScript code that runs independently in each of several windows.

Relationships Between Frames

The open( ) method of the Window object returns a new Window object representing the newly created window.This new window has an opener property that refers back to the original window. In this way, the two windows can refer to each other, and each can read properties and invoke methods of the other. The same thing is possible with frames. Any frame in a window can refer to any other frame through the use of the frames, parent, and top properties of the Window object.

Every window has a frames property. This property refers to an array of Window objects, each of which represents a frame contained within the window. (If a window does not have any frames, the frames[] array is empty and frames.length is zero.) Thus, a window (or frame) can refer to its first subframe as frames[0], its second subframe as frames[1], and so on. Similarly, JavaScript code running in a window can refer to the third subframe of its second frame like this:

frames[1].frames[2]

Every window also has a parent property, which refers to the Window object in which it is contained. Thus, the first frame within a window might refer to its sibling frame (the second frame within the window) like this:

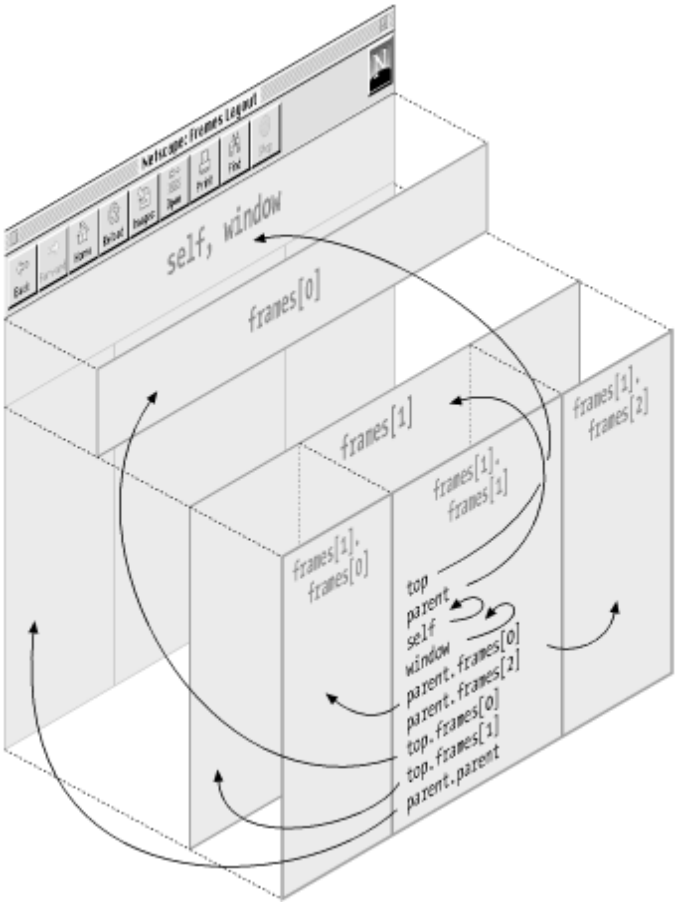
parent.frames[1]

If a window is a top-level window and not a frame, parent simply refers to the window itself:

parent == self; // For any top-level window

If a frame is contained within another frame that is contained within a top-level window, that frame can refer to the top-level window as parent.parent. The top property is a general-case shortcut, however: no matter how deeply a frame is nested, its top property refers to the top-level containing window. If a Window object represents a top-level window, top simply refers to that window itself. For frames that are direct children of a top-level window, the top property is the same as the parent property.

Frames are typically created with <frameset> and <frame> tags. In HTML 4, however, as implemented in IE 4 and later and Netscape 6 and later, the <iframe> tag can also be used to create an "inline frame" within a document. As far as JavaScript is concerned, frames created with <iframe> are the same as frames created with <frameset> and <frame>.



This fig. illustrates these relationships between frames and shows how code running in any one frame can refer to any other frame through the use of the frames, parent, and top properties. The figure shows a browser window that contains two frames, one on top of the other. The second frame (the larger one on the bottom) itself contains three subframes, side by side.

Form object and Element:

The Form object represents an HTML <form> element.We can access a <form> element by using getElementById():

var x = document.getElementById("myForm");

We can create a <form> element by using the document.createElement() method:

var x = document.createElement("FORM");

Form Object Collections

Collection	Description
elements	Returns a collection of all elements in a form

Form Object Properties

Property	Description
acceptCharset	Sets or returns the value of the accept-charset attribute in a form
action	Sets or returns the value of the action attribute in a form
autocomplete	Sets or returns the value of the autocomplete attribute in a form
encoding	Alias of enctype
enctype	Sets or returns the value of the enctype attribute in a form
length	Returns the number of elements in a form
method	Sets or returns the value of the method attribute in a form
name	Sets or returns the value of the name attribute in a form
noValidate	Sets or returns whether the form-data should be validated or not, on submission

target	Sets or returns the value of the target attribute in a form

Form Object Methods

Method	Description
reset()	Resets a form
submit()	Submits a form

Finding HTML Elements

Often, with JavaScript, you want to manipulate HTML elements.

To do so, you have to find the elements first. There are a couple of ways to do this:

- Finding HTML elements by id
- Finding HTML elements by tag name
- Finding HTML elements by class name
- Finding HTML elements by CSS selectors
- Finding HTML elements by HTML object collections

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with id="intro":

Example
```
var myElement = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in myElement).

If the element is not found, myElement will contain null.

Finding HTML Elements by Tag Name

This example finds all <p> elements:

Example
```
var x = document.getElementsByTagName("p");
```

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that matches a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

This example returns a list of all <p> elements with class="intro".

Example
```
var x = document.querySelectorAll("p.intro");
```

This example finds the element with id="main", and then finds all <p> elements inside "main":

Example
```
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
```

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use getElementsByClassName().

This example returns a list of all elements with class="intro".

Example
```
var x = document.getElementsByClassName("intro");
```

Finding HTML Elements by HTML Object Collections

This example finds the form element with id="frm1", in the forms collection, and displays all element values:

Example
```
var x = document.forms["frm1"];
var text = "";
var i;
for (i = 0; i < x.length; i++) {
    text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

Data entry and Validation:

Data validation is the process of ensuring that computer input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct input to a computer application.

Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server.

Client side validation is performed by a web browser, before input is sent to a web server.

Tables and Forms:

Tables are defined with the **<table>** tag.Tables are divided into **table rows** with the **<tr>** tag.

Table rows are divided into **table data** with the **<td>** tag.A table row can also be divided into **table headings** with the **<th>** tag.

If you do not specify a border for the table, it will be displayed without borders.A border can be added using the border attribute:

Table headings are defined with the **<th>** tag. To add a caption to a table, use the **<caption>** tag. To make a cell span more than one row, use the **rowspan** attribute. To make a cell span more than one column, use the **colspan** attribute.

Eg:
```
<table style="width:100%">
  <caption>Monthly savings</caption>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
  </tr>
  <tr>
    <td>February</td>
    <td>$50</td>
  </tr>
</table>
```

Forms:

HTML forms are used to collect user input.

The **<form>** element defines an HTML form:

```
<form>
.
form elements
.
</form>
```

HTML forms contain **form elements**.

Form elements are different types of input elements, checkboxes, radio buttons, submit buttons, and more.

The **<input>** element is the most important **form element**.

The <input> element has many variations, depending on the **type** attribute.

Here are the types used in this chapter:

Type	Description
text	Defines normal text input
radio	Defines radio button input (for selecting one of many choices)
submit	Defines a submit button (for submitting the form)

- **<input type="text">** defines a one-line input field for **text input**
- **<input type="radio">** defines a **radio button**.

  Radio buttons let a user select ONE of a limited number of choices

- **<input type="submit">** defines a button for **submitting** a form to a **form-handler**.
- The **action attribute** defines the action to be performed when the form is submitted.
- The **method attribute** specifies the HTTP method (**GET** or **POST**) to be used when submitting the forms.

Eg:
```
<form action="action_page.php" method="get">
  First name:<br>
    <input type="text" value="Mickey">
    <br>
   Last name:<br>
    <input type="text" name="lastname" value="Mouse">
    <br><br>
    <input type="submit" value="Submit">
</form>
```

DHTML with JavaScript:

Dynamic HTML (DHTML) combines HTML and a scripting language that runs on the Client's browser to bring special effects to otherwise static pages. The scripting language that we will be using is JavaScript as most browsers support it. The scripting language can be used to alter HTML data shown (or present but hidden) on the current page by manipulating the properties for the HTML tags involved. Basically some script function is called to execute the required effect when events like 'MouseOver', 'MouseOut', 'Click', etc. occur. The major examples of DHTML are as follows :

- Using Layers : All the data to ever be displayed is loaded into the existing pages in HTML format but is hidden by using the visibility property of HTML layer elements like LAYER, DIV, etc. Now when an event like Mouse Over occurs a JavaScript function is called to show the layer. The layer is then hidden on Mouse Out event.

- Using Image Rollovers and Swaps : If you have some knowledge of web designing then you should know that when creating Image rollovers or image Swaps you usually take some script and include it to the content of your HTML pages. If you look carefully at the HTML code for the images you will find that they call functions on MouseOver and MouseOut events.

- Using Dynamic Forms : While surfing you must have noticed how some forms seem to have special functions like (a) validation of fields (b) text box character limits (c) dynamic displayed lists depending on your selection (d) redirection to other pages (e) manipulating your keyboard input or disabling some keys or browser buttons, etc. Well all this is done through calling some corresponding script functions.

- Using Cascading Style Sheets (CSS) : Yes, you might be surprised to know that CSS is part of DHTML. A CSS file is just a JavaScript file with the extension .css which is necessary for it to be recognized by the server and tools like Macromedia's Dreamweaver to contain styles.