Server Side Technology

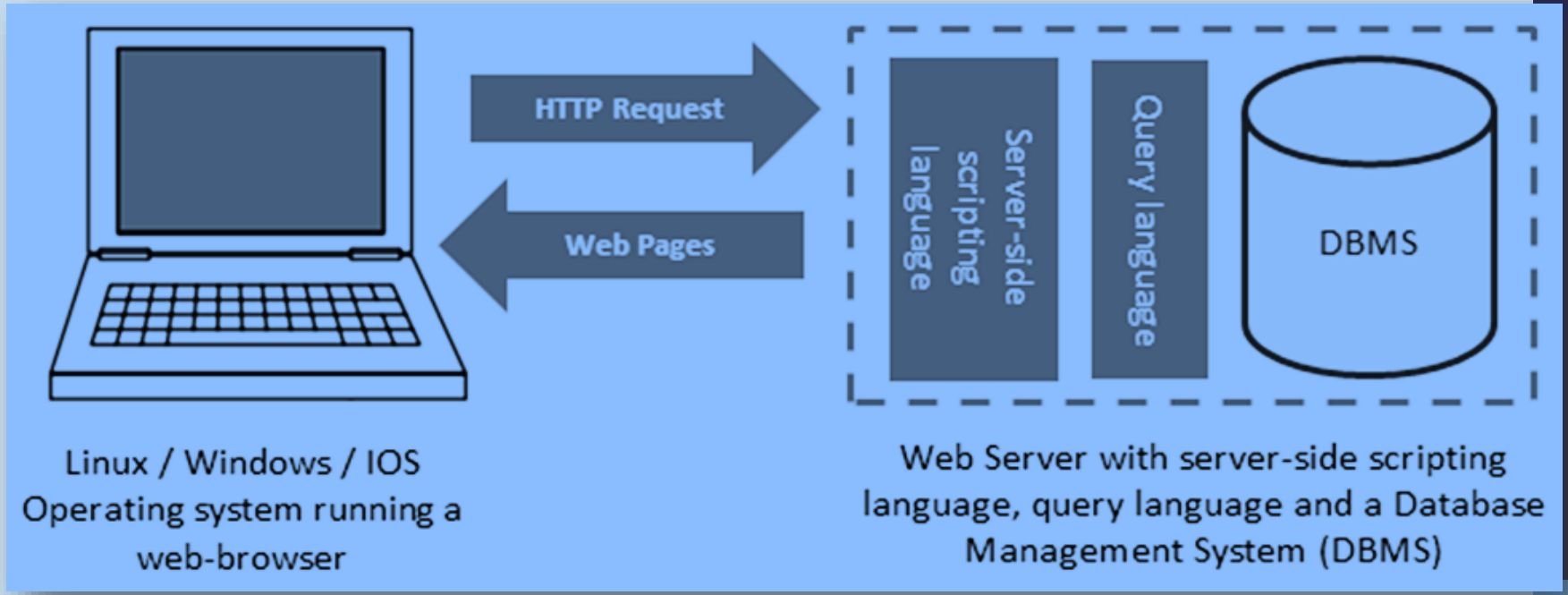
Introduction

- The term 'server-side technologies' can encompass a range of software solutions, mainly:
  - Server-side scripting languages;
  - Database Management Systems (DBMS);
  - Web server software such as Apache;
  - and many other technologies depending upon the application being built.
- The essential combination of technologies required to build a service is known as a 'software-solution stack‘.

Introduction

- Stack must include four components:
  - An operating system;
  - A web-server instance;
  - A database management system and
  - A server-side scripting language.
- The components of the stack are interchangeable.
- LAMP (An acronym for Linux, Apache, MySQL and PHP).
- WAMP, MAMP.

Introduction



Client-Server relationship in WWW

- Let’s imagine how the Facebook log-in process works:
  - The Facebook login URL is entered in a web browser address bar
  - Web browser initiates a HTTP request from the client machine to Facebook’s web server.
  - Facebook Web Server responds with a HTTP response containing a static webpage of a log-in form.
  - Log-in credentials are entered and the 'Submit' button is clicked calling (probably) some JavaScript validation.

Client-Server relationship in WWW

- Let's imagine how the Facebook log-in process works:
  - The credentials are passed to a scripting language which establishes a secure connection to the Facebook server-side Database Management System using a query language.
  - Once a secure connection has been established a query is called which checks if the user-credentials match a user in the database.
  - A series of server-side processes occur to compile the user profile and display it in the browser.

Type of Server Side Technologies

- Server side web technology is used to develop dynamic web resource programs that having the capability to generate dynamic web pages.
- Server side web technologies are two types.
  - Process Based
    - Since processes based scheduling takes more time
    - Performance will be degraded when multiple requests are given simultaneously.
    - This makes the process based web applications as non-scalable web applications.
    - e.g. CGI (Common Gateway Interface)

Type of Server Side Technologies

- Server side web technology is used to develop dynamic web resource programs that having the capability to generate dynamic web pages.
- Server side web technologies are two types.
  - Thread Based
    - Since, scheduling on threads takes less time.
    - Gives good performance in all situations.
    - This makes thread based web applications as “scalable applications.”
    - e.g. JSP (Java Server Pages), Servlet.

Web Server

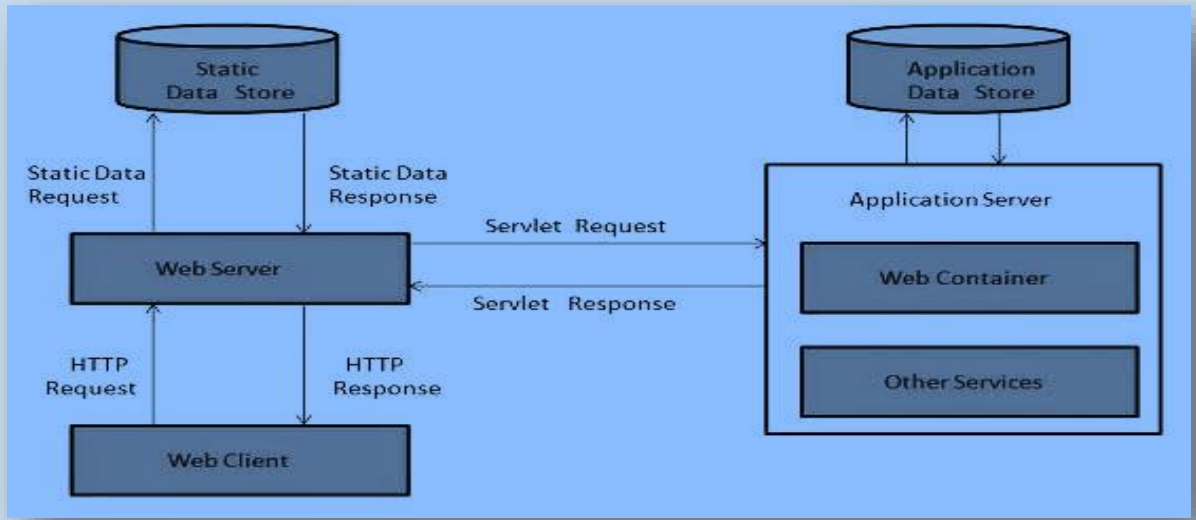
- Web server is a computer where the web content is stored.
- A program that uses HTTP protocol for serving files that create web pages for users.
- A web server is never disconnected from the Internet.
- Each of the web servers has a unique static IP address.

Accessing Web Server

- To request documents from web servers, users must know the hostnames on which the web server software resides.
- Users can request web documents from local web servers (i.e., ones residing on users' machines) or remote web servers (i.e., ones residing on different machines).
- Local web servers can be accessed through the name localhost—a hostname that references the local machine and normally translates to the IP address 127.0.0.1 (known as the loopback address).
- A remote web server referenced by a fully qualified hostname or an IP address can also serve documents.

Web Server Working

- Web server respond to the client request in either of the following two ways:
  - Sending the file to the client associated with the requested URL.
  - Generating response by invoking a script and communicating with database.



Web Server Architecture

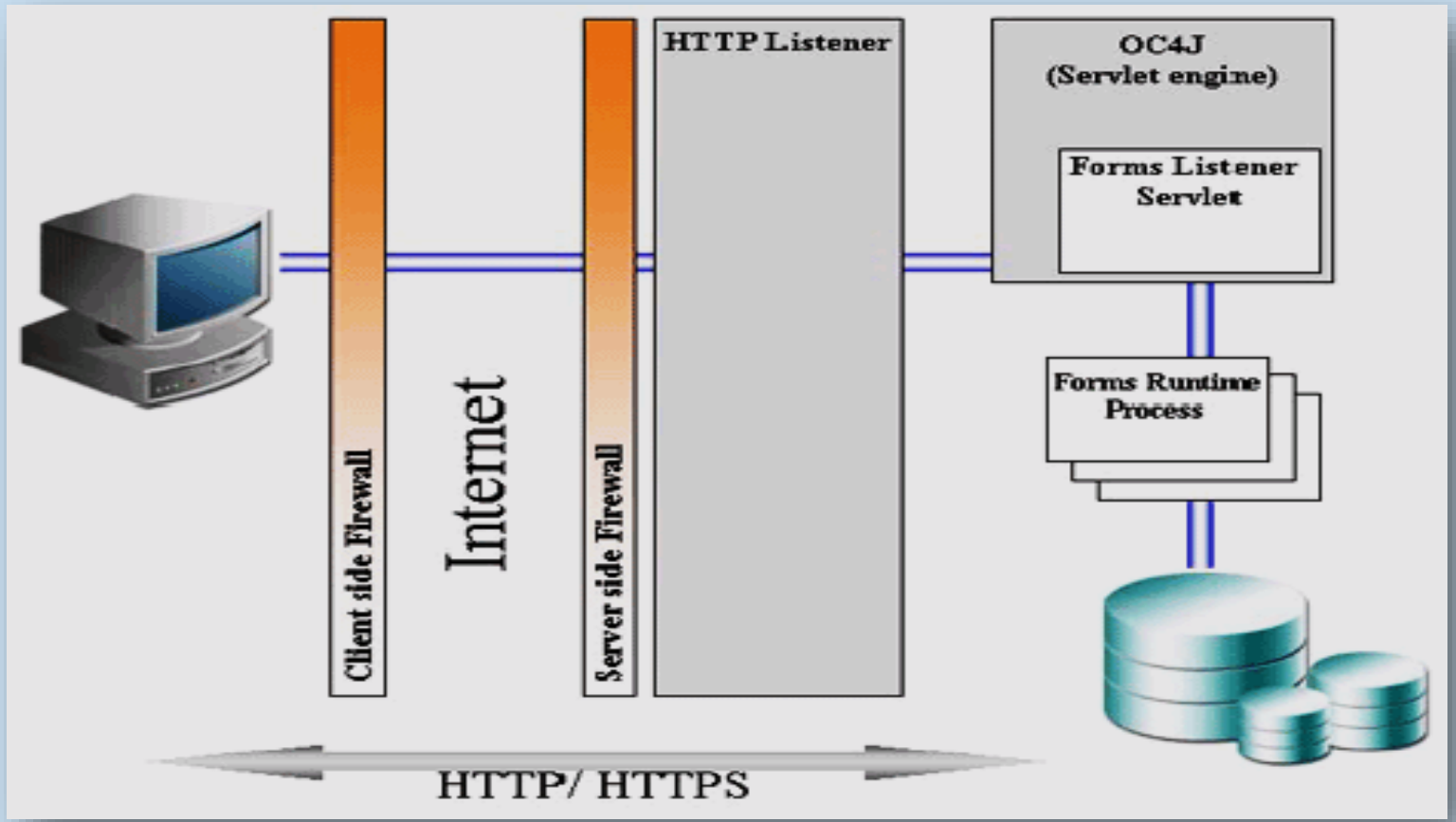
- Web Server includes a number of software modules:
  - Content Engines
    - Web Server content engines are designed for manipulating customer data.
    - HTTP Handling: Accept and responds to HTTTP requests.
    - Content Handling: manage your server's content. E.g. create and store web pages.
    - Search: search webpage and files according to users request.

Web Server Architecture

- Web Server includes a number of software modules:
  - Server Extensions
    - Enable you to extend or replace the function of the server for better performance.
  - Runtime Environments
  - Application Services

Web Server Architecture

- Web Server includes a number of software modules:



Scripting in Web

Client-Side Scripting versus Server-Side Scripting

- Client-side scripts
  - Validate user input
  - Reduce requests needed to be passed to server
  - Access browser
  - Enhance Web pages with DHTML, ActiveX controls, and applets

Client-Side Scripting versus Server-Side Scripting

- Server-side scripts
  - Executed on server
  - Generate custom response for clients
  - Wide range of programmatic capabilities
  - Access to server-side software that extends server functionality

Purpose of Server-Side Scripting

- Insertion of continuously changing (Dynamic) content into a web page, for example - weather or stock quotes.
- Authentication, authorization and session tracking and handling cookies and keeping information about the session and/or the user is best handled by server-side scripting.
- Template-driven page generation.
- Personalization and customization of content based on authentication - serving of content based on the content of the page (e.g. ads) or the browsing behavior of the user.

Purpose of Server-Side Scripting

- Dynamic image generation, e.g. page counters, human-readable characters for security, maps, overlays etc.
- Dynamic generation of CSS and Javascript.
- Generating and reading HTTP headers - server-side scripting can best generate cache control and other complex headers.
- Handling POST form input - accepting the input of a form and writing it to storage (file system, database, session etc.).
- Device mapping - generating different types of content (HTML, XML) based on the user agent that sent the HTTP request.

Purpose of Server-Side Scripting

- Retrieval of data in response to query string parameters and insertion into a web page.
- The data can be retrieved from a database, file system or other forms of storage.
- Communication with other programs, libraries and APIs - e.g. sending out e-mail.

Introduction to JSP

Overview

- Java Server Pages (JSP) is a technology for developing web pages that support dynamic content.
- A Java Server Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application.
- Java Servlet is underlying technology of Java Server Pages.
- Servlets are Java classes that extend the functionality of a Web server by dynamically generating Web pages.

Overview

- Developers insert java code in HTML pages by making use of special JSP tags.
- JSP tags can be used for a variety of purposes, such as
  - Retrieving information from a database.
  - Registering user preferences.
  - Passing control between pages.
  - Sharing information between requests, pages etc.

WHY USE JSP?

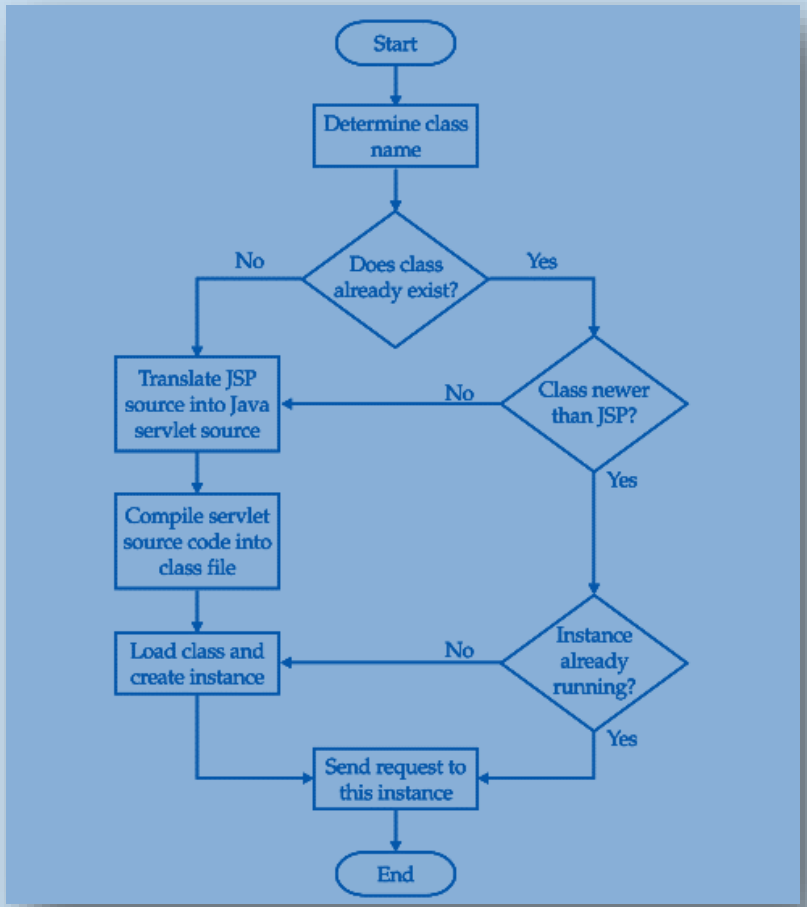
- JSP Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate files.
- Better scalability than CGI scripts because they are persistent in memory and multithreaded.
- Java Server Pages are built on top of the Java Servlets API, so like Servlets; JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
- JSP pages can be used in combination with servlets.

JSP ARCHITECTURE

- The web server needs a JSP engine i.e. container to process JSP pages.
- The JSP container is responsible for intercepting requests for JSP pages.
- A JSP container works with the Web server to provide the runtime environment and other services a JSP needs.
- Apache server is most commonly used which has built-in JSP container to support JSP pages development.

JSP WORKING

- JSP exists in three forms:
  - JSP Source Code
  - JAVA Source Code
  - Compiled JAVA Class



JSP LIFE CYCLE

- A JSP life cycle can be defined as the entire process from its creation till the destruction.
- The following are the phases followed by a JSP
  - Compilation
  - Initialization
  - Execution
  - Cleanup

JSP LIFE CYCLE

- JSP Compilation
  - The compilation process involves three steps:
    - Parsing the JSP.
    - Turning the JSP into a servlet.
    - Compiling the servlet.
- JSP Initialization
  - When a container loads a JSP it invokes the jspInit() method before servicing any requests.

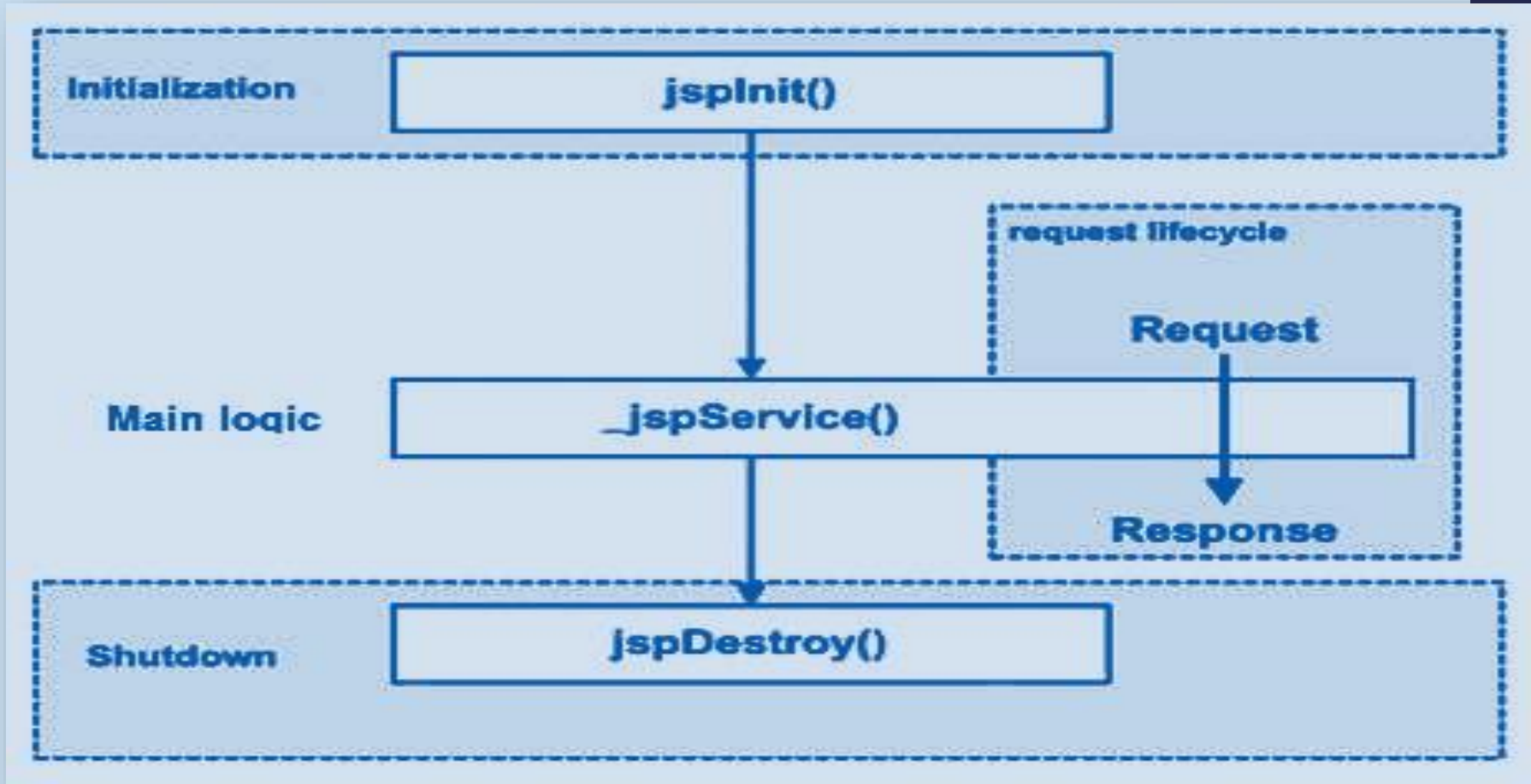
JSP LIFE CYCLE

- JSP Execution
  - This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
  - Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.
  - The _jspService() method of a JSP is invoked once per a request and is responsible for
    - generating the response for that request
    - responsible for generating responses to all of the HTTP methods i.e. GET, POST, DELETE etc.

JSP LIFE CYCLE

- JSP Cleanup
  - The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
  - The jspDestroy() method is used.

JSP LIFE CYCLE



The JSP Development Model

- A JSP page exists in three forms:
  - The .jsp source file containing HTML statements and JSP elements.
  - The Java source code (.java) for a servlet program.
  - The compiled Java class (.class).
- The JSP developer writes a .jsp source file and stores it somewhere in the document file system of a Web server.

The JSP Development Model

- When the .jsp URL is invoked for the first time, the JSP container reads the .jsp file, parses its contents, and generates the source code for an equivalent Java servlet.
- It then compiles the servlet and creates a .class file.
- Finally, the JSP container loads the servlet class and uses it to service the HTTP request.

Components of a JSP Page

- A .jsp file can contain:
  - JSP elements
  - Fixed template data (HTML)
  - Any combination of the two
- JSP elements are instructions to the JSP container about what code to generate and how it should operate.
- These elements have specific start and end tags that identify them to the JSP compiler.
- Template data is everything else that is not recognized by the JSP container.

JSP Elements

- Three types of JSP elements exist:
  - Directives
  - Scripting elements:
    - expressions
    - scriptlets
    - declarations
  - Actions

JSP Directives

- The directive elements are used to do page related operation during page translation time.
- JSP supports 3 types of directive elements:
  - Page directive
  - Include directive
  - Taglib directive
- Syntax of directive elements:

```
<%@ directive attribute="value" %>
```

JSP Directives : page Directive

- The page directive defines attributes that apply to an entire JSP page.
- Syntax of page directive:

```
<%@ page attribute="value" %>
```

- Examples:
  - <%@ page import="java.util.*" %>
  - <%@ page session="true" %>

JSP Directives : page Directive

buffer	Specifies a buffering model for the output stream.	<%@ page buffer = "none" %> <%@ page buffer = "8kb" %>
autoFlush	Controls the behavior of the servlet output buffer.	<%@ page autoFlush = "false" %> <%@ page autoFlush = "true" %>
contentType	Defines the character encoding scheme.	<%@ page contentType = "text/xml" %> <%@ page contentType = "text/html" %>
errorPage	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.	<%@ page errorPage = "MyErrorPage.jsp" %>

JSP Directives : page Directive

isErrorPage	Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.	<%@ page isErrorPage = "true" %>
extends	Specifies a superclass that the generated servlet must extend.	<%@ page extends = "somePackage.SomeClass" %>
import	Specifies a list of packages or classes for use in the JSP.	<%@ page import = "java.sql.*,java.util.*" %>
language	Defines the programming language used in the JSP page.	<%@ page language = "java" %>
session	whether or not the JSP page participates in HTTP sessions.	<%@ page session = "true" %>

JSP Directives : include Directive

- The include directive merges the contents of another file at translation time into the .jsp source input stream.
- Syntax of include directive:

  <%@ include file="fileName" %>

- Examples:
  - <%@ include file="/header.html" %>
  - <%@ include file="/doc/legal/disclaimer.html" %>

JSP Expressions

- JSP provides a simple means for accessing the value of a Java variable or other expression and merging that value with the HTML in the page.
- The Syntax is:

  <%= expression %>

  here exppression is any valid Java expression
- Example:
  - <%= new java.util.Date() %>

JSP Scriptlets

- A scriptlet is a set of one or more Java language statements intended to be used to process an HTTP request.
- The syntax is:

```
<%
        statement1;
        statement12;
        ...
%>
```

- A JSP page may contain any number of scriptlets.

JSP Declarations

- Like scriptlets, declarations contain Java language statements, but with one big difference:

  Scriptlet code becomes part of the _jspService() method, whereas declaration code is incorporated into the generated source file outside the _jspService() method.

- Declaration sections can be used to declare class or instance variables, methods, or inner classes.

- They have no access to the implicit objects of JSP.

JSP Declarations

- The syntax is:

```
<%!
        statement1;
        statement12;
        ...
%>
```

JSP Actions

- Actions are high-level JSP elements that create, modify, or use other objects.
- JSP actions use the construct in XML syntax to control the behavior of the JSP engine.
- We can dynamically insert a file, reuse the beans components, forward user to another page, etc. through JSP Actions.
- Directives are used during translation phase while actions are used during request processing phase.
- Unlike Directives Actions are re-evaluated each time the page is accessed.

JSP Actions

- The syntax is:

<tagname attr="value" attr="value" ... > ..... </tagname>

or, if the action has no body

<tagname attr="value" attr="value" ... />

- Example
  - <jsp:include> Action

<jsp:include page="page URL“ />

JSP Implicit Objects

- Implicit objects are created by the jsp container that are available to all the jsp pages.

Object	Type	Uses
out	JspWriter	writing content to the client.
request	HttpServletRequest	get the data on a JSP page
response	HttpServletResponse	dealing with the response
config	ServletConfig	getting configuration information
application	ServletContext	maintain useful data across whole JSP application.

JSP Implicit Objects

- Implicit objects are created by the jsp container that are available to all the jsp pages.

Object	Type	Uses
session	HttpSession	storing the user's data to make it available on other JSP pages till the user session is active.
pageContext	PageContext	accessing page, request, application and session attributes.
page	Object	a reference to the current Servlet instance like this keyword
exception	Throwable	exception handling for displaying the error messages.

JSP out Implicit Object

- For writing any data to the buffer, JSP provides an implicit object named out.

```
<html>
        <body>
                <%
                        out.print(“Hello JSP”);

                 %>
        </body>
</html>
```

JSP request Implicit Object

- The JSP request is an implicit object of type HttpServletRequest and created for each jsp request by the jsp container.
- It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

```
<form action="userinfo.jsp">
        Enter User Name: <input type="text" name="uname" /> <br>
        Enter Password: <input type="text" name="pass" /> <br>
        <input type="submit" value="Submit Details"/>
</form>
```

JSP request Implicit Object

```
<%@ page import = " java.util.* " %>
<html>
<body>
        <%
                String username=request.getParameter("uname");
                String password=request.getParameter("pass");
                out.print("Name: "+username+" Password: "+password);
        %>
</body>
</html>
```

JSP request Implicit Object

Method	Uses	Example
getParameter(String name)	get the value of a request's parameter	String Uid= request.getParameter("user-id");
getParameterNames()	returns enumeration of all the parameter names	Enumeration e= request.getParameterNames();
getParameterValues(String name)	returns the array of parameter values.	String[] allchoices = request.getParameterValues("choice");
getAttribute(String name)	get the attribute value.	request.getAttribute("admin")
getAttributeNames()	get the attribute names associated to the current session.	Enumerator e = request.getAttributeNames();

JSP request Implicit Object

Method	Uses	Example
setAttribute(String,Object)	assigns an object's value to the attribute.	request.setAttribute("password", str)
removeAttribute(String)	attribute can be removed	request.removeAttribute("userid")
getCookies()	returns an array of cookie objects	request.getCookies()
getHeader(String name)	get the header information	
getQueryString()	getting the query string associated to the JSP page URL.	
getRequestURI()	returns the URL of current JSP page	
getMethod()	returns HTTP request method.	

JSP response Implicit Object

- The JSP response is an implicit object of type HttpServletResponse and created for each jsp request by the jsp container.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

```
<form action="checkdetails.jsp">
        UserId:     <input type="text" name="id" /> <br><br>
        Password: <input type="text" name="pass" /> <br><br>
        <input type="submit" value="Sign In!!"/>
</form>
```

JSP response Implicit Object

```
<%
        String uid=request.getParameter("id");
        String password=request.getParameter("pass");
        session.setAttribute("session-uid", uid);
        if(uid.equals("admin") && password.equals("1234"))
        {
                response.sendRedirect("success.jsp");
        }
        else
        {
                response.sendRedirect("failed.jsp");
        }
 %>
```

JSP response Implicit Object

```
<%
        String data=(String)session.getAttribute("session-uid");
        out.println("Welcome "+ data+"!!");
%>
```

```
<%
        String data2=(String)session.getAttribute("session-uid");
        out.println("Hi "+ data2+". Id/Password are wrong. Please try Again.");
%>
```

JSP response Implicit Object

Method	Uses	Example
setContentType(String type)	tells browser, the type of response data by setting up the MIME type and character encoding.	response.setContentType("application/pdf");
sendRedirect(String address)	It redirects the control to a new JSP page.	response.sendRedirect("http://google.com");
addHeader(String name, String value)	adds a header to the response, basically it includes a header name and it's value.	response.addHeader("Site", "Google.com");
setHeader(String name, String value)	overrides the current value of header with the new value.	response.setHeader("Site", "yahoo.com");
boolean containsHeader(String name)	It basically checks the whether the header is present in the response or not.	response.containsHeader("Site");

JSP response Implicit Object

Method	Uses	Example
addCookie(Cookie cookie)	This method adds a cookie to the response.	response.addCookie(Cookie Siteinfo);
sendError(int status_code, String message)	send error response with a code and an error message.	response.sendError(404, "Page not found error");
boolean isCommitted()	It checks whether the Http Response has been sent to the client, if yes then it returns true else it gives false.	response.isCommited()
setStatus(int statuscode)	set the HTTP status to a given value.	response.setStatus(404);

JSP session Implicit Object

- Session is most frequently used implicit object in JSP.
- The main usage of it to gain access to all the user's data till the user session is active.
- The Java developer can use this object to set, get or remove attribute or to get session information.

```
<form action="session.jsp">
            <input type="text" name="inputname"> <br/>
            <input type="submit" value="click here!!"> <br/>
 </form>
```

JSP session Implicit Object

```
<%
        String uname=request.getParameter("inputname");
        out.print("Welcome "+ uname);
        session.setAttribute("sessname",uname);
%>
<a href="output.jsp">Check Output Page Here </a>
```

```
<body>
<%
        String name=(String)session.getAttribute("sessname");
        out.print("Hello User: You have entered the name: "+name);
%>
</body>
```

JSP session Implicit Object

Method	Uses
setAttribute(String, object)	save an object in session by assigning a unique string to the object.
getAttribute(String name)	The object stored by setAttribute method is fetched from session using getAttribute method.
removeAttribute(String name)	The objects which are stored in session can be removed from session using this method.
getAttributeNames()	It returns all the objects stored in session. Basically, it results in an enumeration of objects.
getCreationTime()	This method returns the session creation time, the time when session got initiated (became active).
getId()	returns the unique string identifier of session (session ID).

JSP session Implicit Object

Method	Uses
isNew()	check whether the session is new. It returns Boolean value (true or false).
invalidate()	It kills a session and breaks the association of session with all the stored objects.
getMaxInactiveInterval()	Returns session's maximum inactivate time interval in seconds.
getLastAccessedTime()	used to know the last accessed time of a session.

Session Tracking

Session Tracking

- Web server doesn't remember clients from one request to the next, the only way to maintain a session is for clients to keep track of it.
- It can be accomplished in two basic ways:
  - Client remember all session-related data and send it back to the server as needed.
    - Cookies
    - Hidden Fields
    - URL Rewriting
  - Server maintain all the data, assign an identifier to it, and have the client remember the identifier.
    - Session Object

Hidden Form Fields

- HTML forms support input elements with a type of HIDDEN.
- Hidden fields are passed along with other form parameters in the HTTP request sent to the Web server, but they don't have any visual representation.
- A web server can send a hidden HTML form field along with a unique session ID as follows:

```
<input type="hidden" name="sessionid" value="5432“ />
```

- The problem with hidden fields is they can only be used in HTML forms.
- If the user clicks a hyperlink and leaves the page, the hidden fields are lost.

URL Rewriting

- A URL can have parameters appended to it that are sent along with the request to the Web server.

  http://server/MyPage.jsp?name1=value1&name2=value2&...

- Some extra data on the end of each URL can be appended to identify the session, and the server associates that session identifier with data it has stored about that session.
- This technique is guaranteed to work in all browser environments and security settings.
- The technique tends to degrade performance if large amounts of data are stored.
- URLs aren’t secure, being visible in the browser address window and in Web server logs.

Cookies

- Cookie is a key value pair of information, sent by the server to the browser and then browser sends back this identifier to the server with every request there on.
- There are two types of cookies:
  - **Session cookies** - are temporary cookies and are deleted as soon as user closes the browser. The next time user visits the same website, server will treat it as a new client as cookies are already deleted.
  - **Persistent cookies** - remains on hard drive until we delete them or they expire.

Cookies

- If cookie is associated with the client request, server will associate it with corresponding user session otherwise, it will create a new unique cookie and send back with response.
- Browser provides a way to disable the cookies and in that case server will not be able to identify the user.

Session Object

- User Session starts when a user opens a browser and sends the first request to server.
- Session object is available in all the request (in entire user session) so attributes stored in Http session will be available in any servlet or in a jsp.
- When session is created, server generates a unique ID and attach that ID with the session.
- Server sends this Id to the client with first response.
- Browser sends back this ID with every request of that user to server with which server identifies the user.

JSP Cookies

- JSP provides a class Cookie in javax.servlet.http package.
- Cookie class provides a two argument constructor (name and value of cookie).

Cookie(String name , String value)

- Commonly used methods available in Cookie class.
  - public void setMaxAge(int expiry)
    - sets maximum age (in seconds).
  - public int getMaxAge()
    - returns the maximum age of the cookie.

JSP Cookies

- JSP provides a class Cookie in javax.servlet.http package.
- Cookie class provides a two argument constructor (name and value of cookie).

Cookie(String name , String value)

- Commonly used methods available in Cookie class.
  - public String getName()
    - returns the name of the cookie.
  - public String getValue()
    - returns the value of the cookie.

JSP Cookies

- JSP provides a class Cookie in javax.servlet.http package.
- Cookie class provides a two argument constructor (name and value of cookie).

Cookie(String name , String value)

- Commonly used methods available in Cookie class.
  - public void setComment(String comment)
    - add a comment that describes a cookie's purpose.
  - public String getComment()
    - returns the comment describing the purpose

JSP Cookies

- HttpServletResponse class provides a method addCookie() to add the cookie in response and it is sent to the browser.

- HttpServletRequest class provides a method getCookies() to read the cookie sent by client along with request.

JSP Cookies Handling

Sending Cookies to Client

- Sending cookie to client is a three step process.

1. Create a cookie object using two argument constructor.

2. You can call setMaxAge() method on cookie object created in step if you want to make the cookie persistent.

3. Add the cookie using addCookie() method provided by HttpServletResponse object.

JSP Cookies Handling

Reading Cookies

- Reading Cookies sent by client in request is two step process.

1. Get an array of Cookies using getCookies() method of HttpServletRequest.

2. Run a loop and call getName() , getValue() and getMaxAge() to see the details of cookie is sent by client.

Remember Username Password Example

- Cookies can be used to achieve remember username and password features. Lets create a small example:

1. Create RememberMe.jsp which will have username and password fields with a Remember Me check box. This jsp will check for cookies and if found , it will set the value of fields with the cookie values.
2. Create displayHomePage.jsp which will display username and password. Also if the user has checked the “Remember Me” check box in RememberMe.jsp page, this jsp will add username and password as cookie.

RememberMe.jsp

```
<%
        Cookie[] cookies = request.getCookies();
        String username="";
        String password = "";
        if(cookies!=null){
                for(int i=0;i<cookies.length;i++){
                        Cookie cookie = cookies[i];
                        if(cookie.getName().equals("username-cookie"))
                        {
                                                username= cookie.getValue();
                         }
                       else if(cookie.getName().equals("password-cookie"))
                        {
                                                password= cookie.getValue();
                        }
                }
        }
 %>
    <form name="logonform" action="displayHomePage.jsp" method="POST">
    Username: <input type="text" name="username" value ="<%= username %>"/>
    <br/>
    Password: <input type="password" name="password" value="<%= password %>"/>
    <br/>
    Remember Me  <input type="checkbox" name="rememberMe" value ="true"/>
    <input type="submit" value="Submit"/>
    </form>
```

displayHomePage.jsp

```
<html>
<head>
<title>Display Details</title>
</head>
<body>
  <%
          String username=request.getParameter("username");
          String password=request.getParameter("password");
          String message="Username is : "+ username + "<br/>
                                Password is :" + password ;
          String rememberMe=  request.getParameter("rememberMe");
          if(rememberMe!=null)
          {
                    Cookie usernameCookie = new Cookie("username-cookie", username);
                    Cookie passwordCookie = new Cookie("password-cookie", password);
                    usernameCookie.setMaxAge(24*60*60);
                    passwordCookie.setMaxAge(24*60*60);
                    response.addCookie(usernameCookie);
                    response.addCookie(passwordCookie);
          }
  %>
  <%= message %>

</body>
</html>
```