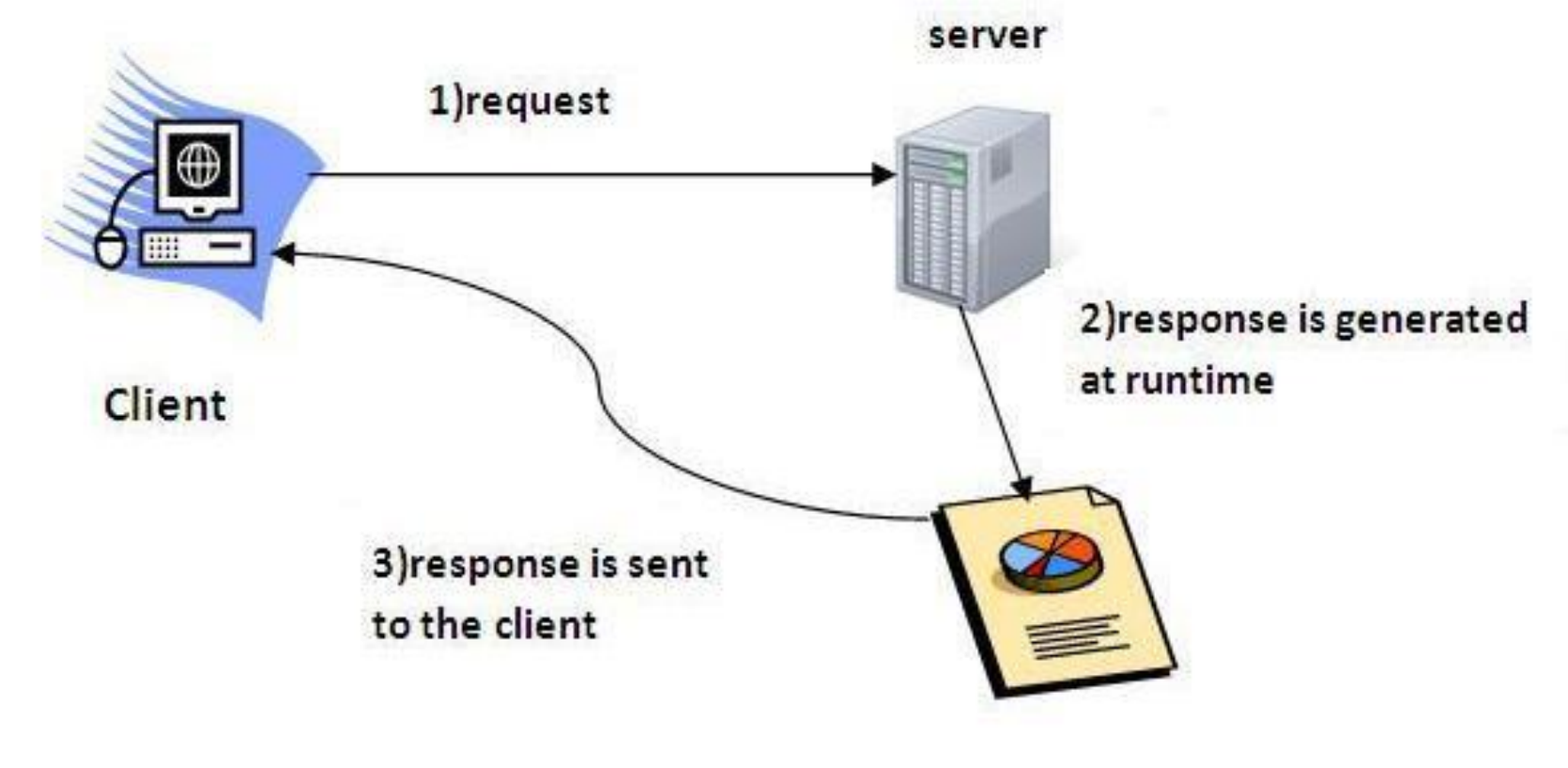
SERVLET

What is Servlet?

- Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Servlet can be described in many ways, depending on the context.
- Servlet is a technology i.e. used to create web application.

- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

Figure

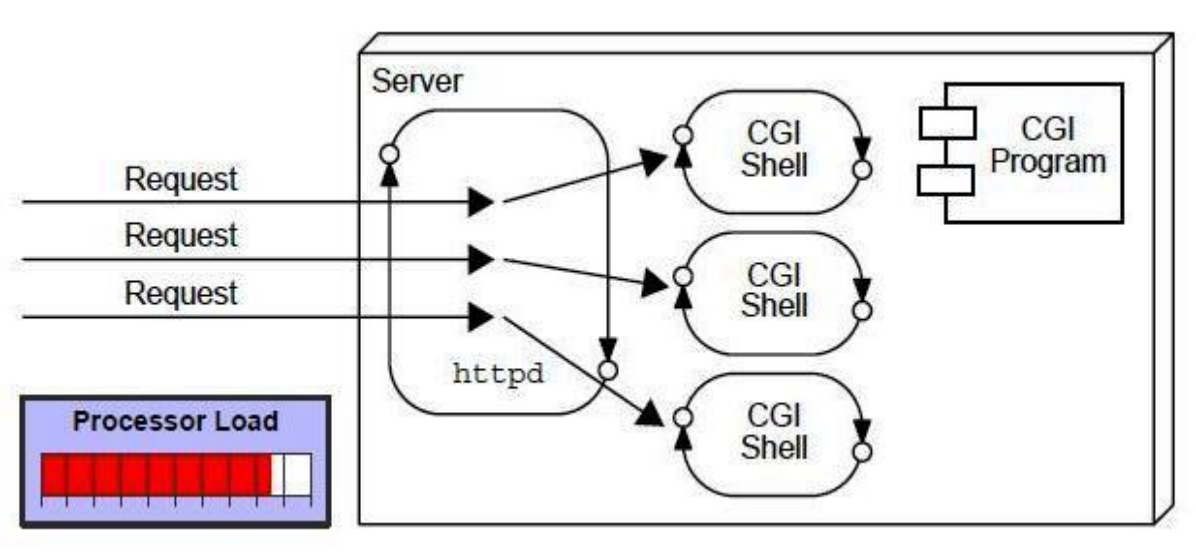

server
1)request
Client
2)response is generated
at runtime
3)response is sent
to the client

What is web application?

- A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

CGI(Common Gateway Interface)

- CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Disadvantages of CGI

- If number of clients increases, it takes more time for sending response.
- For each request, it starts a process and Web server is limited to start processes.
- It uses platform dependent language e.g. C, C++, perl.

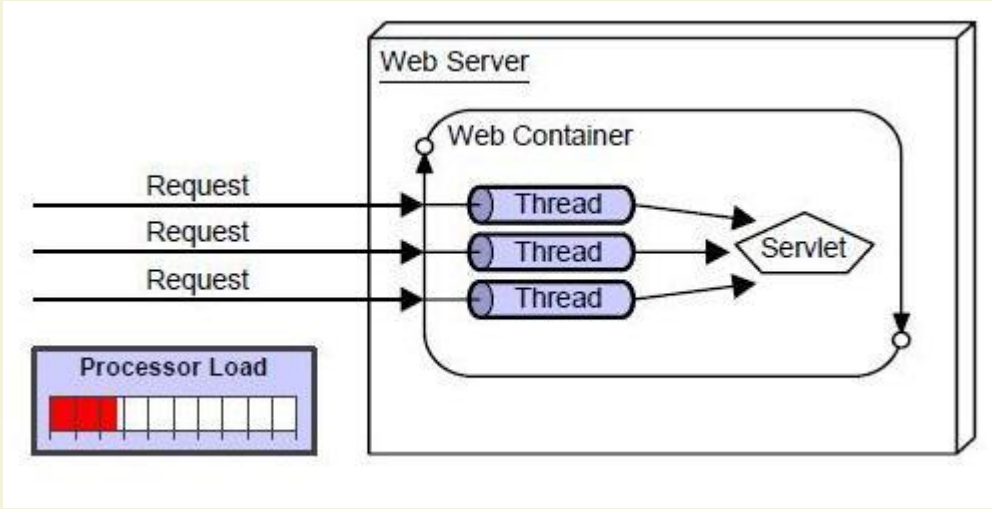
Http Request Methods & Difference

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked**	Post request **cannot be** bookmarked
4)Get request is **idempotent**. It means second request will be ignored until response of first request is delivered.	Post request is **non-idempotent**
5) Get request is **more efficient** and used more	Post request is **less efficient** and used less

Advantage of Servlet

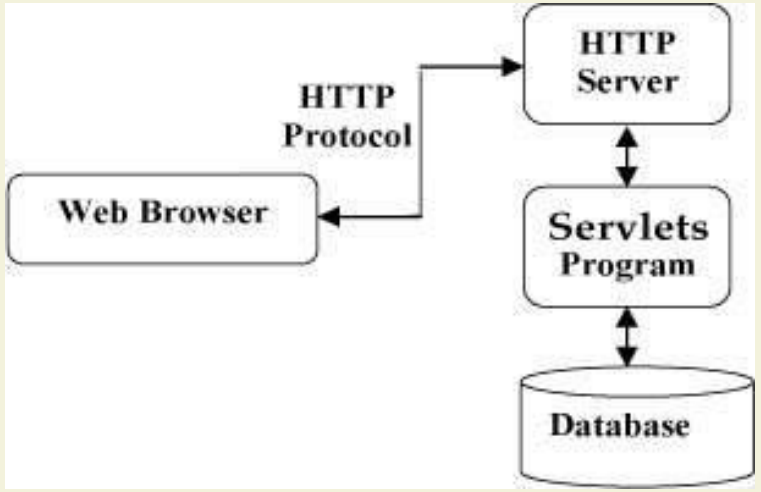
- There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:
- **better performance:** because it creates a thread for each request not process.
- **Portability:** because it uses java language.
- **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
- **Secure:** because it uses java language..



Servlets Architecture:

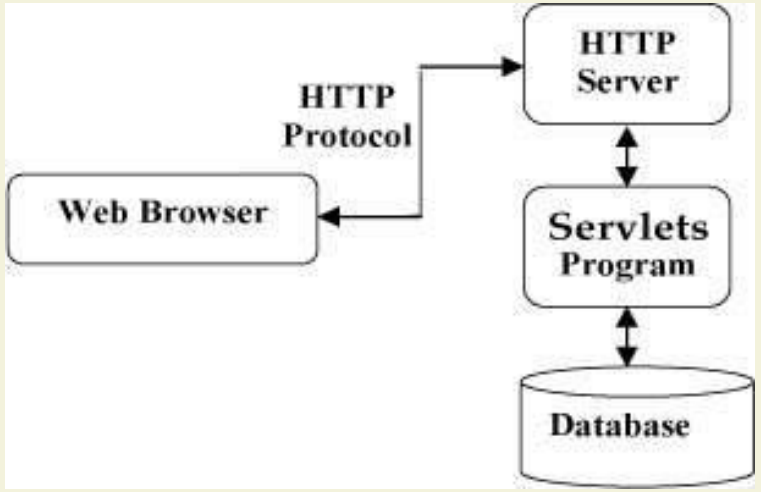
Servlets perform the following major tasks:

- *Read the explicit data sent by the clients (browsers).* This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require taking to a database, invoking a Web service, or computing the response directly.



Servlets Architecture:

- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.



Servlets Packages

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

Servlets - Life Cycle

- A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet
- The servlet is initialized by calling the **init ()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

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

Servlets - Life Cycle

- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.
- The init method definition looks like this:
- public void init() throws ServletException
- {
- // Initialization code...
- }

The service() method :

- The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

The service() method :

- Here is the signature of this method:
- public void service(ServletRequest request,
- ServletResponse response)
- throws ServletException, IOException{
- }
- The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.
- The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.
- public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException, IOException {
- // Servlet code
- }
- **The doPost() Method**
- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.
- public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
- // Servlet code
- }

The destroy() method :

- The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:
- public void destroy() {
- // Finalization code...
- }

EXAMPLE

```
// Import required java libraries
 import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
        public class HelloWorld extends
HttpServlet {
        private String message;
        public void init() throws
ServletException  {
// Do required initialization
        message = "Hello World";  }
        public void doGet(HttpServletRequest
request,                HttpServletResponse
response)          throws
        ServletException, IOException  {
// Set response content type
        response.setContentType("text/html");
// Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message +
"</h1>");  }      public void destroy()
        {
// do nothing.
} }
```

SERVLETS - FORM DATA

- GET method: The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows:
- http://www.test.com/hello?key1=value1&key2=value2
- The GET method is the defualt method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limtation: only 1024 characters can be in a request string. This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using doGet method.

Reading Form Data using Servlet:

- Servlets handles form data parsing automatically using the following methods depending on the situation:
- **getParameter():** You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.

GET Method Example Using Form:

- Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this imput.
- <html><body>
- <form action="HelloForm" method="GET">
- First Name: <input type="text" name="first_name">
- Last Name: <input type="text" name="last_name" />
- <input type="submit" value="Submit" />
- </form></body></html>
- Keep this HTML in a file Hello.htm and put it in <Tomcat-installation-directory>/webapps/ROOT directory. When you would access http://localhost:8080/Hello.htm, here is the actual output of the above form.
- Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

GET Method Example Using Form:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
                  String title = "Using GET Method to Read Form Data";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
              "<html>\n" +
              "<head><title>" + title + "</title></head>\n" +
              "<body bgcolor=\"#f0f0f0\">\n" +
              "<h1 align=\"center\">" + title + "</h1>\n" +
              "<ul>\n" +
              "  <li><b>First Name</b>: "
              + request.getParameter("first_name") + "\n" +
              "  <li><b>Last Name</b>: "
              + request.getParameter("last_name") + "\n" +
              "</ul>\n" +
              "</body></html>");
  }}
```

POST Method Example Using Form:

- <html>
- <body>
- <form action="HelloForm" method="POST">
- First Name: <input type="text" name="first_name">
-
- Last Name: <input type="text" name="last_name" />
- <input type="submit" value="Submit" />
- </form>
- </body>
- </html>

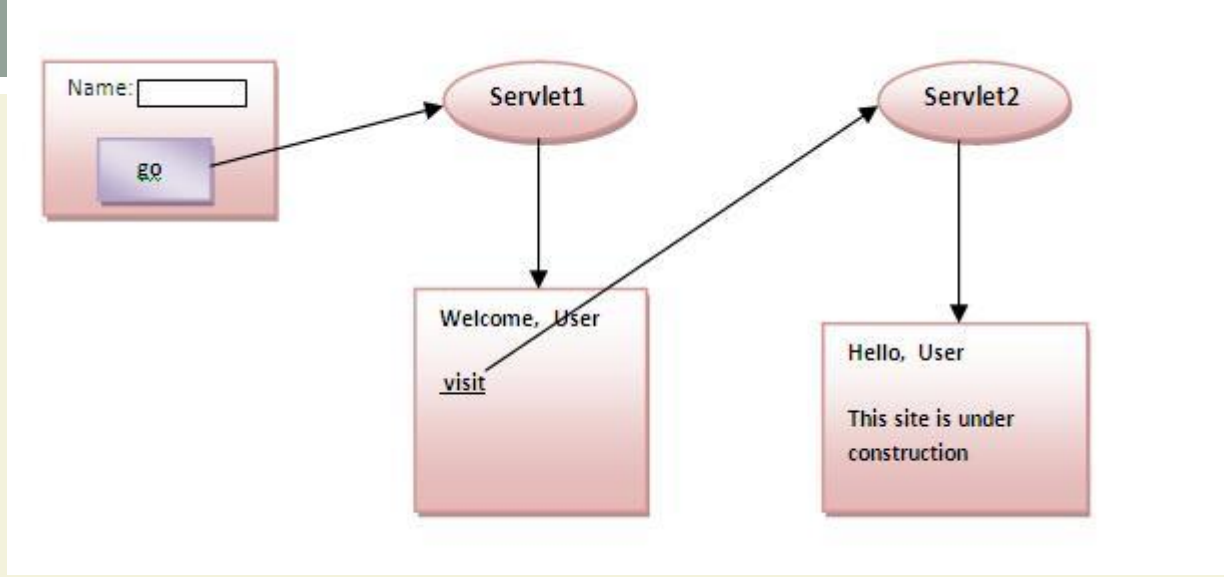
POST Method Example Using Form:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm extends HttpServlet {
  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
                  String title = "Using GET Method to
Read Form Data";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                "<html>\n" +
                "<head><title>" + title +
"</title></head>\n" +
                "<body bgcolor=\"#f0f0f0\">\n" +
                "<h1 align=\"center\">" + title + "</h1>\n"
+
                "<ul>\n" +
                "  <li><b>First Name</b>: "
                + request.getParameter("first_name") +
"\n" +
                "  <li><b>Last Name</b>: "
                + request.getParameter("last_name") +
"\n" +
                "</ul>\n" +
                "</body></html>");  }
  // Method to handle POST method request.
   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
      throws ServletException, IOException {
     doGet(request, response);
}}
```

Session Tracking

- To support the software that needs keep track of the state, Java Servlet technology provides an API for managing sessions and allows several mechanisms for implementing sessions. Session tracking is a great thing. Every user can be associated with a javax.servlet.http.HttpSessionobject that servlets can use to store or retrieve information about that user. Any set of arbitry can be saved by the Java objects in a session object. For example, a user s session object provides a convenient location for a servlet to store the user shopping cart contents
- Methods to Track the Session
- There are four types of techniques used in servlet to handle the session which are as follows:
- 1.URL Rewritting
- 2.Hidden Form Fieds
- 3.Http Session
- 4.Secure Socket Layer(SSL)

URL REWRITING



- In url rewriting, we append a token or identifier to the url of the next servlet or the next resource. We can send parameter name/value pairs using the following format:
- url?name1=value1&name2=value2&??
- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand (&). When the user clicks the hyperlink, the parameter name/value pairs will passed to the server. From a servlet, we can getParameter() method to obtain a parameter value.
- Advantage of URL ReWriting
- -It will always work whether cookie is disabled or not (browser dependent)
- -Extra form submission is not required on each pages.

Contd..

- **Disadvantage of URL Rewrinting**
- -Generate more network traffic.
- -It will work only with links.
- -It can send Only textual information.
- -Less secure because query string in session id displace on address bar.
- Example of URL ReWriting
- Intex.html
- <form action="servlet1">
- Name:<input type="text" name="UserName"/></br>
- <input type="submit" value="go"/>
- </form>

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
        try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
         String n=request.getParameter("userName");
        out.print("Welcome "+n);
         HttpSession session=request.getSession();
        session.setAttribute("uname",n);
         out.print("<a href='servlet2'>visit</a>");
        out.close();
              }catch(Exception e){System.out.println(e);}
    }
}
```

SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
        try{

response.setContentType("text/html");
        PrintWriter out = response.getWriter();
  //getting value from the query string
        String n=request.getParameter("uname");
        out.print("Hello "+n);
        out.close();
}
                                catch(Exception e){System.out.println(e);}
    }
}
```

contd

- Hidden Form Fieds
- HTML forms
  have an entry that looks like following: <input type="hidden" name="session"value="...">. This means that, when the form is submitted, the specified name and value are includedin GET or POST data. This can be used to store information about the session. However, it has themajor disadvantage that it only works if every page is dynamically generated, since the whole pointis that each session has the unique identifier.
- **Http Session**
- The HttpSession interface is implemented by the services to provide an association between anHTTP client and HTTP server. This association, or session, persists over multiple connection and/or requests during a given time period. Sessions are used to maintain the state and user identity acrossmultiple page requests.A session can be maintained either by using the cookies or by URL rewriting. To expose whether theclient supports cookies, HttpSession defines the isCookieSupportDetermined method an d anisUsingCookies method.

Servlets - Cookies Handling

- Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.
- There are three steps involved in identifying returning users:
- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

Setting cookies with servlet involves three steps:

- (1) Creating a Cookie object: You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.
- Cookie cookie = new Cookie("key","value");
- Keep in mind, neither the name nor the value should contain white space or any of the following characters:
- [ ] ( ) = , " / ? @ : ;

Contd..

- (2) Setting the maximum age: You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

- cookie.setMaxAge(60*60*24);
- (3) Sending the Cookie into the HTTP response headers: You use response.addCookie to add cookies in the HTTP response header as follows:

- response.addCookie(cookie);

Reading Cookies with Servlet:

- Let's see the simple code to get all the cookies.

- Cookie ck[]=request.getCookies();
- for(int i=0;i<ck.length;i++){
- out.print("
"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
- }

Delete Cookies with Servlet:

- To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps:
- Read an already exsiting cookie and store it in Cookie object.
- Set cookie age as zero using **setMaxAge()** method to delete an existing cookie.
- Add this cookie back into response header.

Data Storage

- Almost all web application (servlets or related dynamic web server software) store and retrieve data.
- Typical web application uses a database management system (DBMS)
- Another option is to use the file system.
- One common problem in data storage is concurrency
- **Parameter Data:**
- The request object (which implement HttpServletRequest) provies information from the HTTP request to the servlet
- One type of information is parameter data, which is information from the query string portion of the HTTP request.
- http://www.example.com/servlet/PrintThis?arg=aString
- Parameter data is the web analog of arguments in a method call
- System.out.println(“aString”);

JSP

What is JavaServer Pages?

- JavaServer Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.
- A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.
- Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

Why Use JSP?

- JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.
- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer 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 that handle the business logic, the model supported by Java servlet template engines.
- Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Advantages of JSP:

- Following is the list of other advantages of using JSP over other technologies:
- **vs. Active Server Pages (ASP):** The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.
- **vs. Pure Servlets:** It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
- **vs. Server-Side Includes (SSI):** SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- **vs. JavaScript:** JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.
- **vs. Static HTML:** Regular HTML, of course, cannot contain dynamic information.

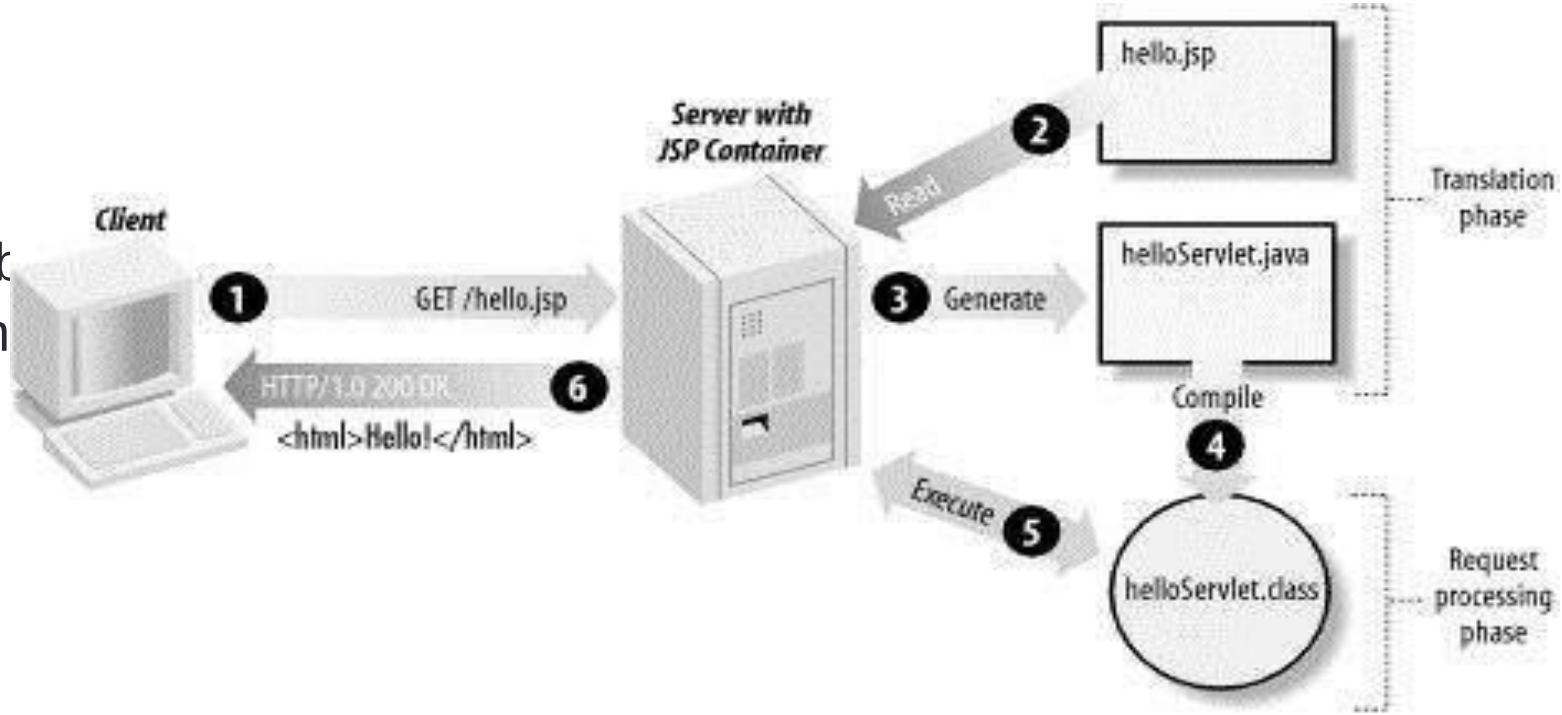
JSP - Architecture

- JSP Processing:
- The following steps explain how the web server creates the web page using JSP:
- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of .html.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

Contd..

- All the above mentioned steps can b

shown below in the following diagram



JSP Processing

- Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.
- So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet

JSP - Syntax

- The Scriptlet:
- A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- Following is the syntax of Scriptlet:
- <% code fragment %>
- You can write XML equivalent of the above syntax as follows:
- <jsp:scriptlet>
- code fragment
- </jsp:scriptlet>
- Any text, HTML tags, or JSP elements you write must be outside the scriptlet. Following is the simple and first example for JSP:

```
<html>
<head><title>Hello World</title></head>
<body>
Hello World!<br/>
<%
out.println("Your IP address is " + request.getRemoteAddr());
%>
</body>
</html>
```

JSP Declarations:

- A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

- Following is the syntax of JSP Declarations:

- <%! declaration; [declaration;]+ ... %>
- You can write XML equivalent of the above syntax as follows:

- <jsp:declaration>
- code fragment
- </jsp:declaration>
- Following is the simple example for JSP Declarations:

- <%! int i = 0; %>
- <%! int a, b, c; %>
- <%! Circle a = new Circle(2.0); %>

JSP Expression:

- A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.
- Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.
- The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.
- Following is the syntax of JSP Expression:
- <%= expression %>
- You can write XML equivalent of the above syntax as follows:
- <jsp:expression>
- expression
- </jsp:expression>
- Following is the simple example for JSP Expression:
- <html>
- <head><title>A Comment Test</title></head>
- <body>
-
- Today's date: <%= (new java.util.Date()).toLocaleString()%>
-
- </body>
- </html>
- This would generate following result:
- Today's date: 11-Sep-2010 21:24:25

JSP Comments:

- JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.
- Following is the syntax of JSP comments:
- <%-- This is JSP comment --%>
- Following is the simple example for JSP Comments:
- <html>
- <head><title>A Comment Test</title></head>
- <body>
- <h2>A Test of Comments</h2>
- <%-- This comment will not be visible in the page source --%>
- </body>
- </html>
- This would generate following result:
- A Test of Comments

Control-Flow Statements:

- JSP provides full power of Java to be embedded in your web application. You can use all the APIs and building blocks of Java in your JSP programming including decision making statements, loops etc.
- Decision-Making Statements:
- The if...else block starts out like an ordinary Scriptlet, but the Scriptlet is closed at each line with HTML text included between Scriptlet tags.

- <%! int day = 3; %>
- <html>
- <head><title>IF...ELSE Example</title></head>
- <body>

```
<% if (day == 1 | day == 7) { %>
    <p> Today is weekend</p>
<% } else { %>
    <p> Today is not weekend</p>
<% } %>
```

- </body>
- </html>
- This would produce following result:

- Today is not weekend

SWITCH...CASE

- Now look at the following switch...case block which has been written a bit differentlty using out.println() and inside Scriptletas:
- <%! int day = 3; %>
- <html>
- <head><title>SWITCH...CASE Example</title></head>
- <body>
- <%
- switch(day) {
- case 0:
- out.println("It\'s Sunday.");
- break;
- case 1:
- out.println("It\'s Monday.");
- break;
- case 2:
- out.println("It\'s Tuesday.");
- break;
- case 3:
- out.println("It\'s Wednesday.");
- break;
- case 4:
- out.println("It\'s Thursday.");
- break;
- case 5:
- out.println("It\'s Friday.");
- break;
- default:
- out.println("It's Saturday.");
- }
- %>
- </body>
- </html>
- This would produce following result:
- It's Wednesday.

Loop Statements:

- You can also use three basic types of looping blocks in Java: for, while,and do…while blocks in your JSP programming.

- Let us look at the following for loop example:
- <%! int fontSize; %>
- <html>
- <head><title>FOR LOOP Example</title></head>
- <body>
- <%for (fontSize = 1; fontSize <= 3; fontSize++){ %>
- <font color="green" size="<%= fontSize %>">
- JSP Tutorial
-
- <%}%>
- </body>
- </html>
- This would produce following result:
- JSP Tutorial
-
- JSP Tutorial
-
- JSP Tutorial

JSP - Database Access

- To start with basic concept, let us create a simple table and create few records in that table as follows:
- **Create Table**
- To create the Employees table in EMP database, use the following steps:

- **Step 1:**
- Open a Command Prompt and change to the installation directory as follows:

- C:\>
- C:\>cd Program Files\MySQL\bin
- C:\Program Files\MySQL\bin>

Steps

- **Step 2:**
- Login to database as follows
- C:\Program Files\MySQL\bin>mysql -u root -p
- Enter password: ********
- mysql>
- **Step 3:**
- Create the table Employee in TEST database as follows:

- mysql> use TEST;

- mysql> create table Employees
- (
- id int not null,
- age int not null,
- first varchar (255),
- last varchar (255)
-);
- Query OK, 0 rows affected (0.08 sec)
- mysql>

Create Data Records

- Finally you create few records in Employee table as follows:
- mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
- Query OK, 1 row affected (0.05 sec)
-
- mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
- Query OK, 1 row affected (0.00 sec)
-
- mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
- Query OK, 1 row affected (0.00 sec)
-
- mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
- Query OK, 1 row affected (0.00 sec)
-
- mysql>

SELECT Operation:

- Following example shows how we can execute SQL SELECT statement using JTSL in JSP programming:
- <%@ page import="java.io.*,java.util.*,java.sql.*"%>
- <%@ page import="javax.servlet.http.*,javax.servlet.*" %>
- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
- <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
- <html>
- <head><title>SELECT Operation</title></head>
- <body>
- <sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
- url="jdbc:mysql://localhost/TEST"
- user="root" password="pass123"/>
- <sql:query dataSource="${snapshot}" var="result">
- SELECT * from Employees;
- </sql:query>
- <table border="1" width="100%">
- <tr>
- <th>Emp ID</th>
- <th>First Name</th>
- <th>Last Name</th>
- <th>Age</th>
- </tr>
- <c:forEach var="row" items="${result.rows}">
- <tr>
- <td><c:out value="${row.id}"/></td>
- <td><c:out value="${row.first}"/></td>
- <td><c:out value="${row.last}"/></td>
- <td><c:out value="${row.age}"/></td>
- </tr>
- </c:forEach>
- </table>
- </body>
- </html>

Now try to access above JSP, which should display the following result:

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18
101	Mahnaz	Fatma	25
102	Zaid	Khan	30
103	Sumit	Mittal	28

INSERT Operation:

- Following example shows how we can execute SQL INSERT statement using JTSL in JSP programming:
- <%@ page import="java.io.*,java.util.*,java.sql.*"%>
- <%@ page import="javax.servlet.http.*,javax.servlet.*" %>
- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
- <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
- <html>
- <head><title>JINSERT Operation</title></head>
- <body>
- <sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
- url="jdbc:mysql://localhost/TEST"
- user="root" password="pass123"/>
- <sql:update dataSource="${snapshot}" var="result">
- INSERT INTO Employees VALUES (104, 2, 'Nuha', 'Ali');
- </sql:update>
- <sql:query dataSource="${snapshot}" var="result">
- SELECT * from Employees;
- </sql:query>
- <table border="1" width="100%">
- <tr>
- <th>Emp ID</th>
- <th>First Name</th>
- <th>Last Name</th>
- <th>Age</th>
- </tr>
- <c:forEach var="row" items="${result.rows}">
- <tr>
- <td><c:out value="${row.id}"/></td>
- <td><c:out value="${row.first}"/></td>
- <td><c:out value="${row.last}"/></td>
- <td><c:out value="${row.age}"/></td>
- </tr>
- </c:forEach>
- </table>
- </body>
- </html>

DELETE Operation:

- Following example shows how we can execute SQL DELETE statement using JTSL in JSP programming:
- <%@ page import="java.io.*,java.util.*,java.sql.*"%>
- <%@ page import="javax.servlet.http.*,javax.servlet.*" %>
- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
- <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
- <html>
- <head><title>DELETE Operation</title></head>
- <body>
- <sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
- url="jdbc:mysql://localhost/TEST"
- user="root" password="pass123"/>
- <c:set var="empId" value="103"/>
- <sql:update dataSource="${snapshot}" var="count">
- DELETE FROM Employees WHERE Id = ?
- <sql:param value="${empId}" />
- </sql:update>
- <sql:query dataSource="${snapshot}" var="result">
- SELECT * from Employees;
- </sql:query>
- <table border="1" width="100%">
- <tr>
- <th>Emp ID</th>
- <th>First Name</th>
- <th>Last Name</th>
- <th>Age</th>
- </tr>
- <c:forEach var="row" items="${result.rows}">
- <tr>
- <td><c:out value="${row.id}"/></td>
- <td><c:out value="${row.first}"/></td>
- <td><c:out value="${row.last}"/></td>
- <td><c:out value="${row.age}"/></td>
- </tr>
- </c:forEach>
- </table>
- </body>
- </html>

UPDATE Operation:

- Following example shows how we can execute SQL UPDATE statement using JTSL in JSP programming:
- <%@ page import="java.io.*,java.util.*,java.sql.*"%>
- <%@ page import="javax.servlet.http.*,javax.servlet.*" %>
- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
- <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
- <html>
- <head><title>DELETE Operation</title></head>
- <body>
- <sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
- url="jdbc:mysql://localhost/TEST"
- user="root" password="pass123"/>
- <c:set var="empId" value="102"/>
- <sql:update dataSource="${snapshot}" var="count">
- UPDATE Employees SET last = 'Ali'
- <sql:param value="${empId}" />
- </sql:update>
- <sql:query dataSource="${snapshot}" var="result">
- SELECT * from Employees;
- </sql:query>
- <table border="1" width="100%">
- <tr>
- <th>Emp ID</th>
- <th>First Name</th>
- <th>Last Name</th>
- <th>Age</th>
- </tr>
- <c:forEach var="row" items="${result.rows}">
- <tr>
- <td><c:out value="${row.id}"/></td>
- <td><c:out value="${row.first}"/></td>
- <td><c:out value="${row.last}"/></td>
- <td><c:out value="${row.age}"/></td>
- </tr>
- </c:forEach>
- </table>
- </body>
- </html>