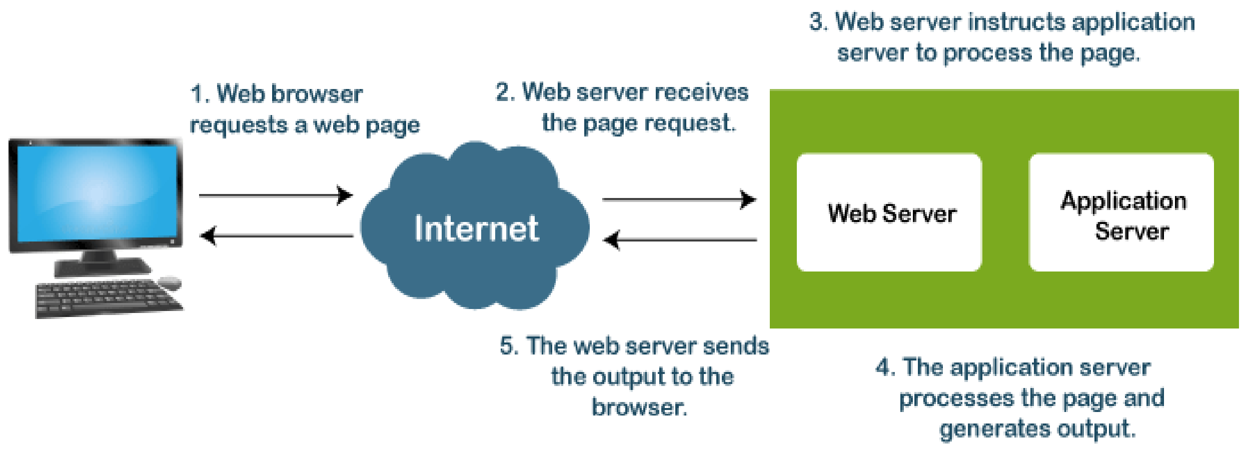
UNIT –IV (Servlets)

Web Application

- A web-application is an application program that is usually stored on a remote server, and users can access it through the use of Software known as web-browser.
- A web application are generally coded using the languages supported by almost every web-browsers such as HTML, JavaScript because these are the languages that rely on the web browsers to render the program executable.


1. Web browser requests a web page
2. Web server receives the page request.
3. Web server instructs application server to process the page.
Internet
Web Server
Application Server
4. The application server processes the page and generates output.
5. The web server sends the output to the browser.

Flow of the Web Application

Some of the web applications are entirely static due to which they not required any processing on the server at all while, on the other hand, some web applications are dynamic and require server-side processing.

To operate a web-application, we usually required a web server (or we can say some space on the web-server for our programs/application's code) to manage the clients' upcoming requests and required an application server.

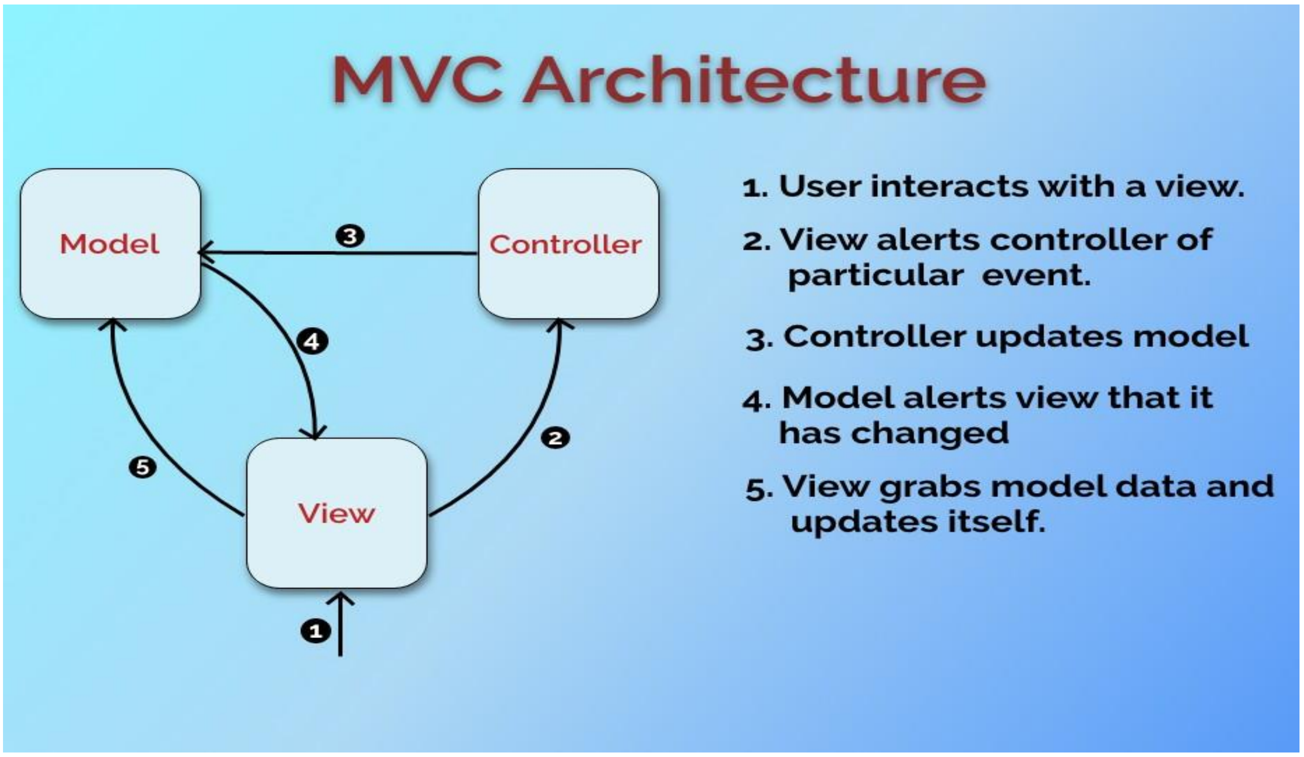
The application server performs the task that requested by the clients, which also may need a database to store the information sometimes.

Application server technologies range from ASP.NET, ASP, and ColdFusion to PHP and JSP.

Web Server
A web server is software and hardware that uses HTTP (Hypertext Transfer Protocol) and other protocols to respond to client requests made over the World Wide Web.
The main job of a web server is to display website content through storing, processing and delivering webpages to users.
Application Server
An application server enables a server to generate a dynamic, customized response to a client request.
The application server collaborates with the web server to return a dynamic, customized response to a client request.

MVC Architecture

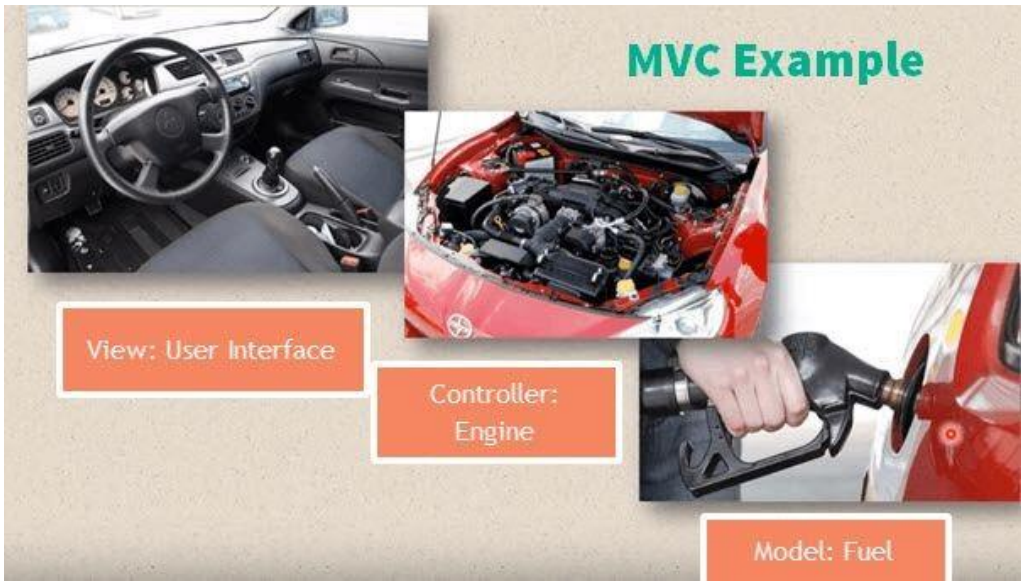
- MVC separates the business logic and presentation layer from each other. It was traditionally used for desktop graphical user interfaces (GUIs).
- Nowadays, MVC architecture in web technology has become popular for designing web applications as well as mobile apps.
- The MVC is an architectural pattern that separates an application into
    1) Model, 2) View and 3) Controller
- – Model: It includes all the data and its related logic
- – View: Present data to the user or handles user interaction
- – Controller: An interface between Model and View components
- – MVC architecture was first discussed in 1979 by T Reenskaug
- **Some popular MVC frameworks are Rails, Zend Framework, CodeIgniter, Laravel, Fuel PHP, etc.**



1. User interacts with a view.
2. View alerts controller of particular event.
3. Controller updates model
4. Model alerts view that it has changed
5. View grabs model data and updates itself.

Examples of MVC architecture in Real world :

View= You Waiter= Controller Cook= Model Refrigerator= Data



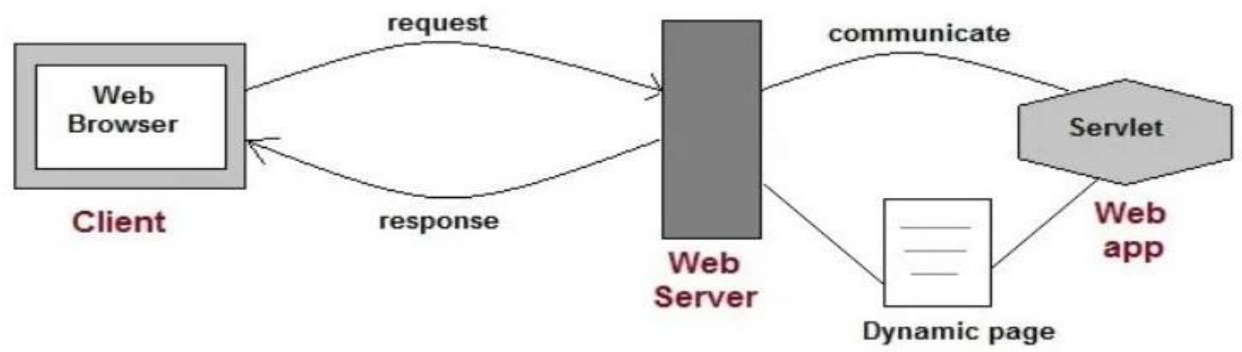
View= User interface : (Gear lever, panels, steering wheel, brake, etc.)
Controller- Mechanism (Engine) Model- Storage (Petrol or Diesel tank)

Describing Servlets

- Servlet is a Java Technology for server-side programming.
- It is a Java module that runs inside a Java-enabled web server and services the requests obtained from the web server.
- Servlets are the Java programs that run on the Java-enabled web server or application server.
- They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.
- Working with Servlets is an important step in the process of Application development and delivery through the Internet.
- A Web application is sometimes called a Web app, Thus, it is an application that is accessed using a Web browser over the network such as the Internet or an Intranet.

Working With Servlets

Working with Servlets is an important step in the process of Application development and delivery through the Internet.

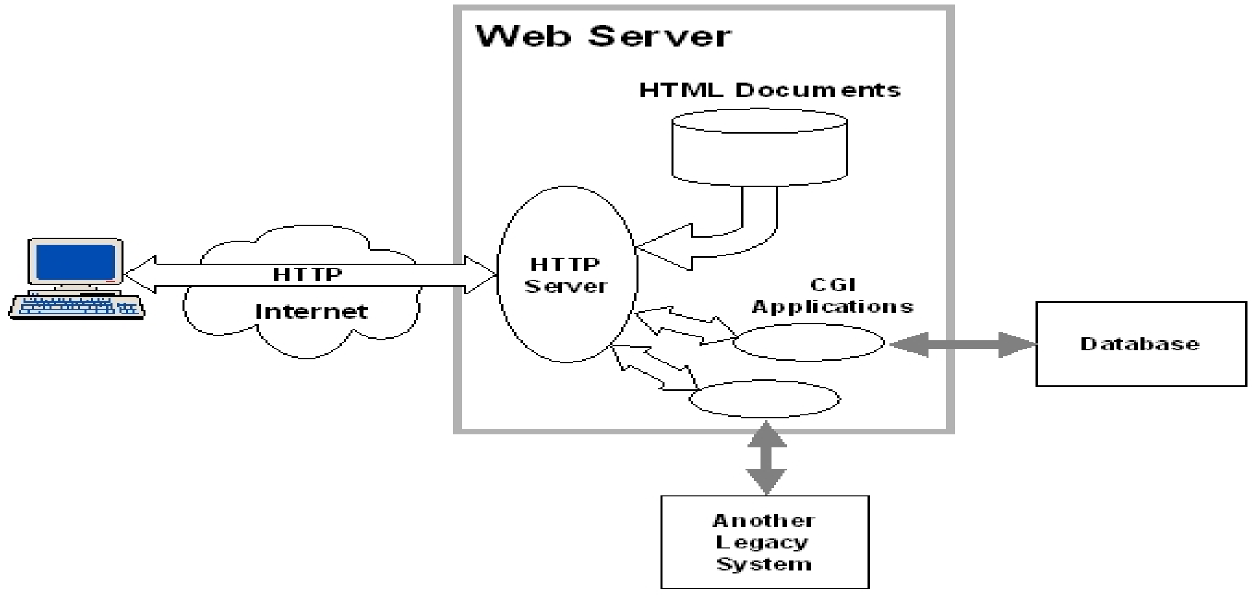


- Servlets are not tied to a specific client-server protocol, but are most commonly used with HTTP.
- Execution of servlet consists of 4 steps:
  - The client sends request to server
  - The server alerts appropriate servlet
  - The servlet process the request and generates the output(if any), and sends back to the Webserver
  - The Webserver sends back response to the client

Advantages Over Applets :

- Applets are downloaded from the server and executes in clients browser
  - Uses JRE
  - Problems with browsers (not compatible)
- Servlets run on the server machine and usually generates HTML code.
  - So, even older version of browser can easily display

CGI :



Why Servlet?

Servlets provide a component-based, platform-independent method for building Webbased applications. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically. Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI).

Differences between Servlets and CGI :

Servlet	**CGI(Common Gateway Interface)**
Servlets are portable and efficient.	CGI is not portable
In Servlets, sharing data is possible.	In CGI, sharing data is not possible.

Servlets can directly communicate with the webserver.	CGI applications cannot directly communicate with the webserver(Some more interface is required).
Servlets are less expensive than CGI.	CGI is more expensive than Servlets.
Servlets can handle the cookies.	CGI cannot handle the cookies.

Servlet Alternatives :

- CGI
- Proprietary API's
  - NETSCAPE NSAPI's
  - Microsoft ISAPI's
  - Orelly's WSAPI's
    - These are not free, community is less, memory leak
- ASP
- Server Side JS

Advantages of Servlets :

- **Better performance:** because it creates a thread for each request, not process.
- **Portability:** because it uses Java language.
- **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
- **Secure:** because it uses java language (Java Security Manager component).

Applications of Servlet :

- 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 talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- 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.

Apache Tomcat Web Server :

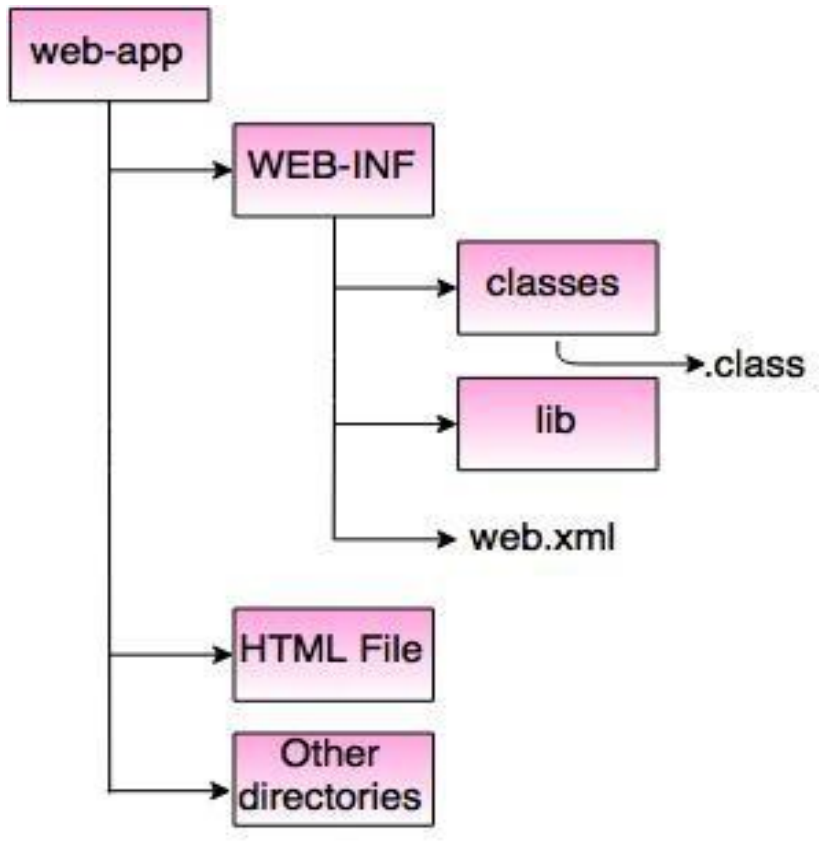


Fig: Directory Structure of Servlet Application

Exploring Features of Servlets :

- Servlet – A Request – Response Model
- Servlets are based on the programming model that accepts requests and generates responses accordingly.
- Developer can extends GenericServlet or HttpServlet class to create servlet.

  – The service() method in GenericServlet class, doXXX() methods in HttpServlet classes are responsible for handling requests and responses.

  public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException { }

Stream related classes for servlet :

```
ServletInputStream i=req.getInputStream();
ServletOutputStream i=req.getOutputStream();
```

Exploring Features of Servlets :

- Servlet and Environment State
  - Servlets are similar to other java objects and runs within the server environment
  - ServletConfig – is used to obtain configuration information from web.xml file (initialization paramerts of Only one servlet can be stored in servletconfig object)
  - ServletContext – is used to obtain configuration information from web.xml file – parameter information related to entire Web Appliation (servlets, JSP, HTML…)

```
<web-app>
<init-param>
<param-name>Country</param-name>
<param-value>India</param-value>
</init-param>
<init-param>
```

<param-name>Age</param-name> <param-value>24</param-value>
</init-param> </web-app>

To retrieve parameters through servlet we use the following methods :

```
getServletConfig().getInitParameter("Country");
getServletConfig().getInitParameter("Age");
```

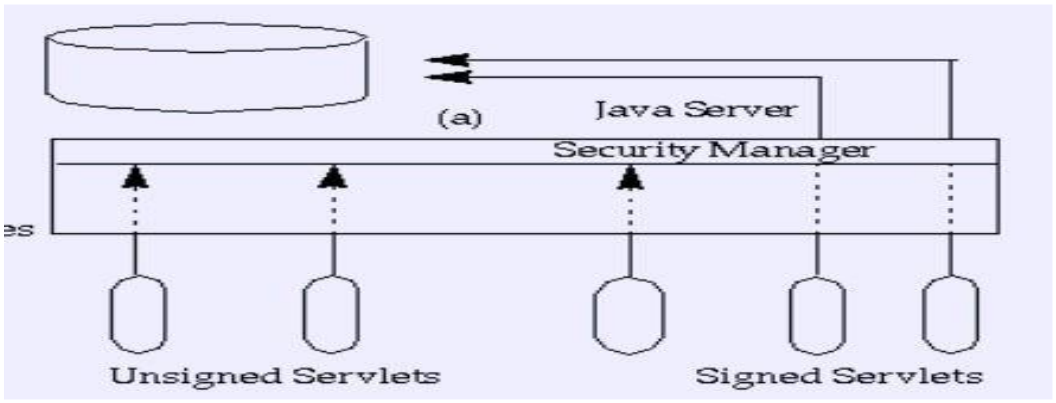
Exploring Features of Servlets :

- Usage Modes
  - Servlets can be used in various modes:
    - Servlets can be chained together into filter chains (recording all incoming requests, logs the IP addresses of the computers from which the requests originate, conversion, data compression, encryption and decryption, input validation etc.) by the servers.
    - Servlets can support protocols such as HTTP.
    - Servlets are replacement of CGI.
    - Servlets are used to generate dynamic HTML content

- **Possible sources of Servlets**
  - When a Client request for a Servlet, the server maps the request of the client and loads the servlet.
  - The Server administrator can specify the mapping of client request to server in the following ways:
    - Mapping of client request to particular servlet.
    - Mapping client requests to the servlets found in the administrative directory(shared by multiple servers)
    - Invokes other servlets

- **Primary Servlet Methods:**
  - init()
  - service()
  - destroy()
- **Security Features**
  - Servlets access information about their client using SSL(Secure Socket Layer)

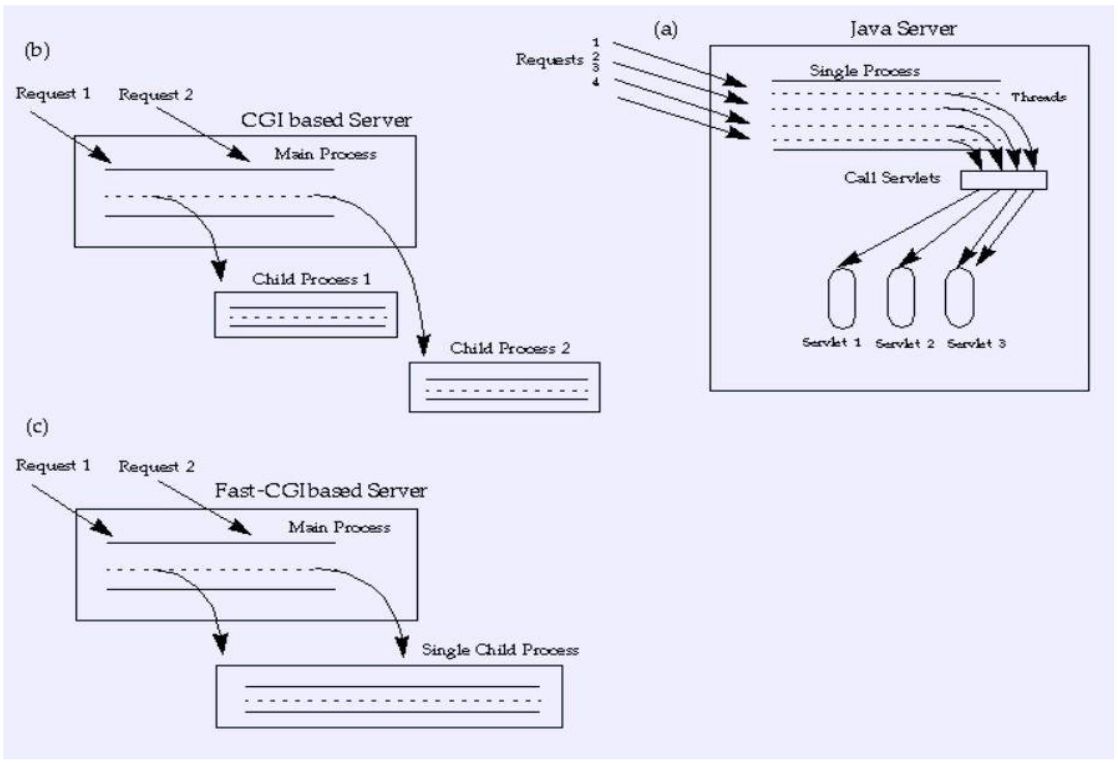
- Managed by Java Server Security Manager
    - Activities of servlets are monitored by the manager (also called as signed servlets).
    - Activities of native code extensions in case of (unsigned servlets) can not be monitored.



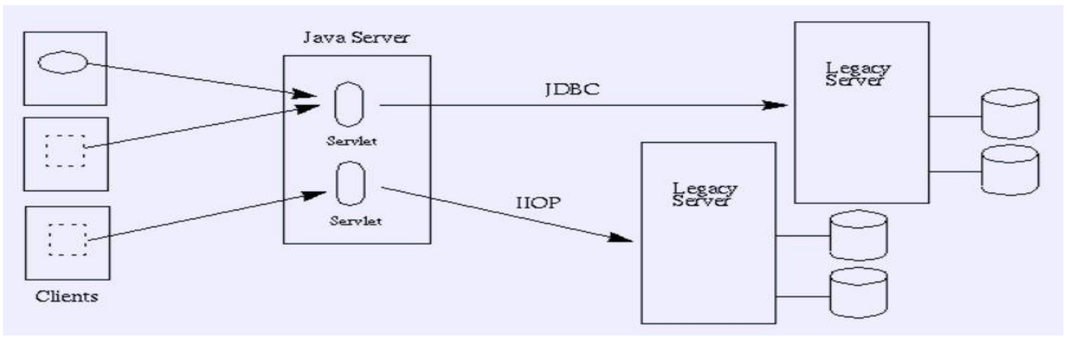
- **HTTP – Specific Servlets**
  - Servlets uses HTTP protocol
    - Servlets which are being used with the HTTP protocol may support any HTTP method, including GET, POST, HEAD, and more.
    - They can get access to parameters which were passed through standard HTML forms, including the HTTP method and the URL which identifies the destination of the request:

» String method = request.getMethod (); // e.g. POST
» String uri = request.getRequestURI ();
» String name = request.getParameter ("name");
» String phone = request.getParameter ("phone");

- **Performance Features**
  - In CGI, for every request one process will be created
  - But in Servlets, one thread will be created
  - In Fast- CGI, child process wont be terminated after request completion but wait for another incoming connection.



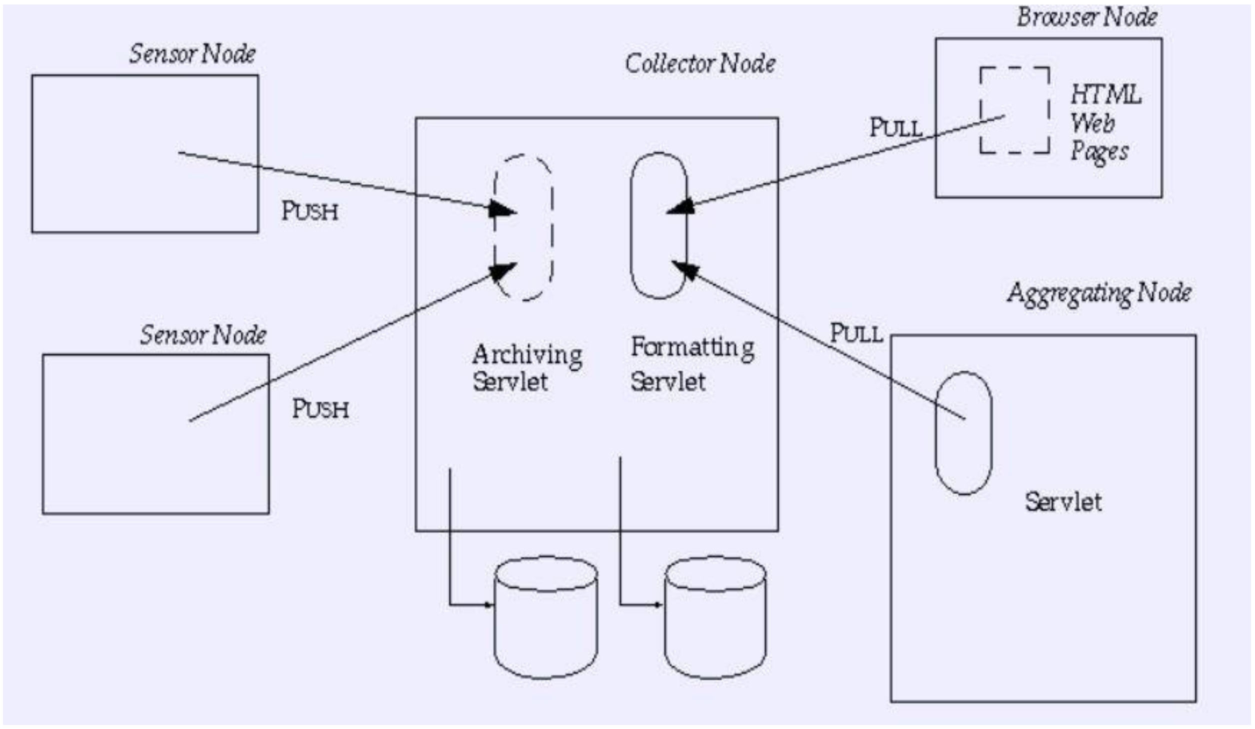
- **3-tier Applications**
  - Using Java Servlet we can opt for 3- tier architecture.
    - The first tier – Web browser
    - Second tier – Servlets – For Business logic
    - Third – Database connectivity



IIOP – Internet Interop – ORB(Object Request

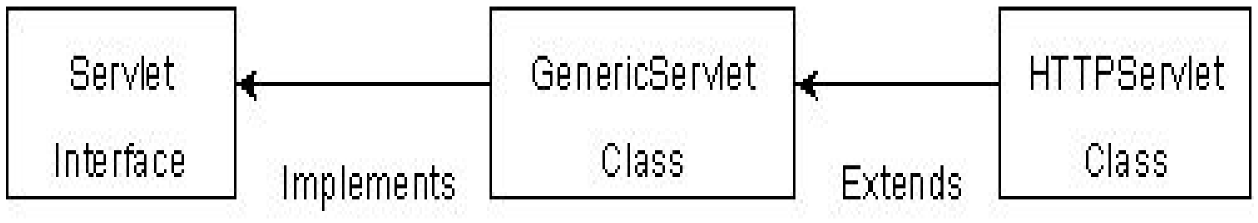
- **Web Publishing System**
  - Many organizations have large collections of data to manage and publish.
  - Servlets easily supports for pushing, pulling data and manages data.

Ex. Weather Forecast]



Exploring Servlet API

- The *javax.servlet* and *javax.servlet.http* packages provide interfaces and classes for writing servlets.
- All servlets must implement the Servlet interface, which defines life-cycle methods.
  - When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API.
  - TheHttpServlet class which extends from GenericServlet , provides methods, such as doGet and doPost, for handling HTTP-specific services.
  -



javax.servlet package

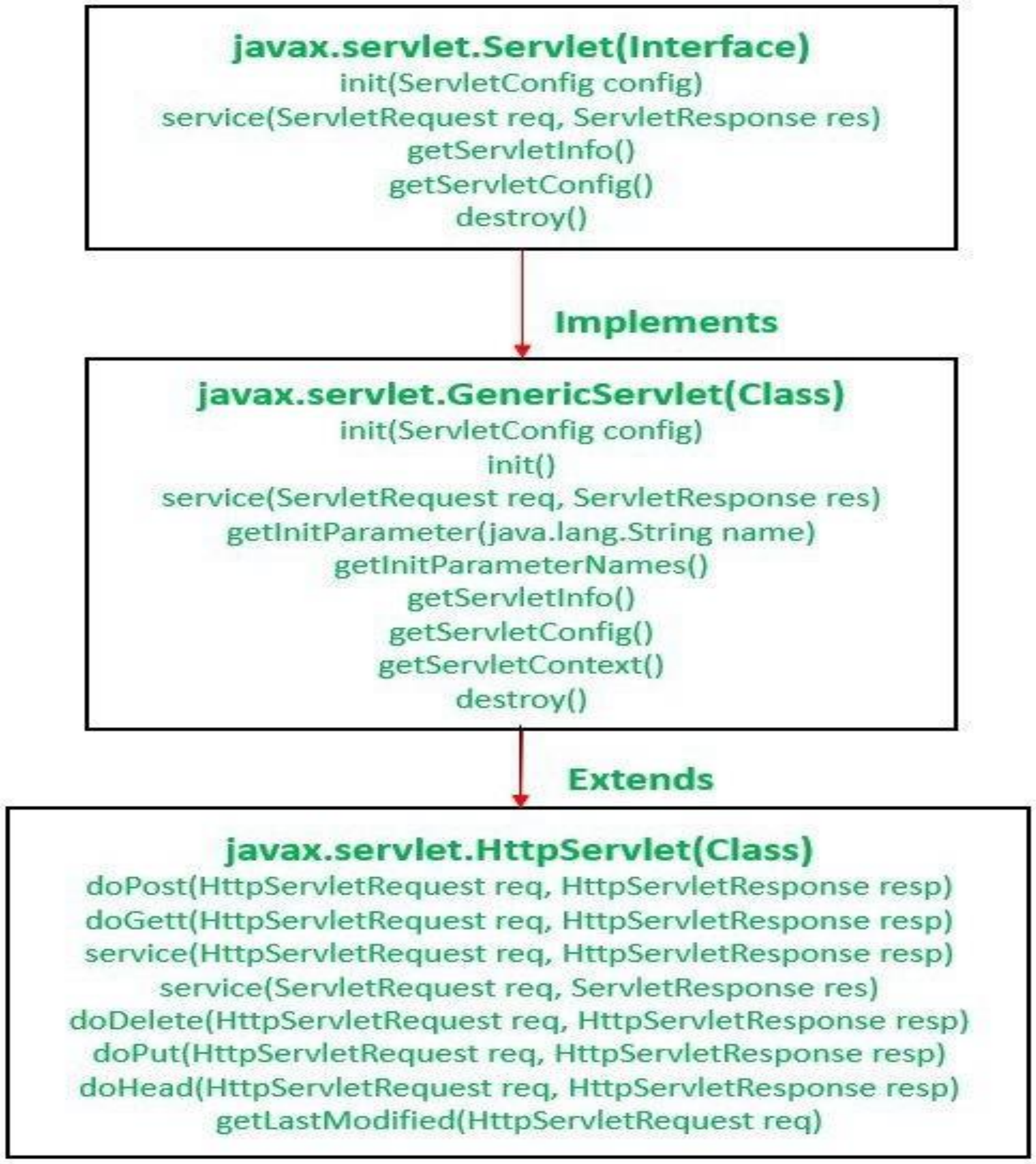
- This package provides the number of interfaces and classes to support Generic servlet which is protocol independent.

Interfaces

- ■ **Servlet**
- ■ **ServletConfig**
- ■ **ServletContext**
- ■ **ServletRequest**
- ■ **ServletResponse**
- ■ **RequestDispatcher**
- ■ **FilterChain**
- ■ **FilterConfig**

Classes

- ■ **GenericServlet**
- ■ **ServletInputStream**
- ■ **ServletOutputStream**
- ■ **ServletException**
- ■ **UnavailableException**


javax.servlet.Servlet(Interface)
init(ServletConfig config)
service(ServletRequest req, ServletResponse res)
getServletInfo()
getServletConfig()
destroy()
Implements
javax.servlet.GenericServlet(Class)
init(ServletConfig config)
init()
service(ServletRequest req, ServletResponse res)
getInitParameter(java.lang.String name)
getInitParameterNames()
getServletInfo()
getServletConfig()
getServletContext()
destroy()
Extends
javax.servlet.HttpServlet(Class)
doPost(HttpServletRequest req, HttpServletResponse resp)
doGett(HttpServletRequest req, HttpServletResponse resp)
service(HttpServletRequest req, HttpServletResponse resp)
service(ServletRequest req, ServletResponse res)
doDelete(HttpServletRequest req, HttpServletResponse resp)
doPut(HttpServletRequest req, HttpServletResponse resp)
doHead(HttpServletRequest req, HttpServletResponse resp)
getLastModified(HttpServletRequest req)

Classes available in javax.servlet package:

GenericServlet	To define a generic and protocol-independent servlet.
ServletInputStream	This class provides an input stream to read binary data from a client request.
ServletOutputStream	This class provides an output stream for sending binary data to the client.
ServletException	Problem with user request or problem to send response
UnavailableException	Requested resource is not available

Interfaces available in javax.servlet package:

Interface Name	**Description**
Servlet	This is the main interface that defines the methods in which all the servlets must implement. To implement this interface, write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.
ServletConfig	It defines an object created by a servlet container at the time of servlet instantiation and to pass information to the servlet during initialization.

ServletContext	It defines a set of methods that a servlet uses to communicate with its servlet container. The information related to the web application available in web.xml file is stored in ServletContext object created by container.
Filter	To perform filtering tasks on either the request to a resource, or on the response from a resource, or both.
RequestDispatcher	It defines an object to dispatch the request and response to any other resource, means it receives requests from the client and sends them to a servlet/HTML file/JSP file on the server.
ServletRequest	It defines an object that is created by servlet container to pass client request information to a servlet.
ServletResponse	It defines an object created by servlet container to assist a servlet in sending a response to the client.

Methods in Servlet Interface

- Declares life cycle methods for a servlet
  - **public void init(ServletConfig config)**
  - **public void service(ServletRequest req,ServletResponse res)**
  - **public void destroy()**
- public ServletConfig getServletConfig()
- public String getServletInfo()

Servlet Config and Servlet context in Detail :

The Servlet Container

Servlet container, also known as **Servlet engine** is an integrated set of objects that provide a run time environment for Java Servlet components.
In simple words, it is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

ServletConfig:

Signature: public interface ServletConfig

ServletConfig is implemented by the servlet container to initialize a single servlet using init(). That is, you can pass initialization parameters to the servlet using the web.xml deployment descriptor. For understanding, this is similar to a constructor in a java class.

Example code:

```
<servlet>
<servlet-name>ServletConfigTest</servlet-name>
<servlet-class>com.javapapers.ServletConfigTest</servlet-class>
<init-param>
<param-name>sum</param-name>
<param-value>14 </param-value>
</init-param>
</servlet>
```

ServletContext:

Signature: public interface ServletContext

ServletContext is implemented by the servlet container for all servlet to communicate with its servlet container, for example, to get the MIME type of a file, to get dispatch requests, or to write to a log file. That is to get detail about its execution environment. It is applicable only within a single Java Virtual Machine. If a web application is distributed between multiple JVM this will not work. For understanding, this is like a application global variable mechanism for a single web application deployed in only one JVM.

The ServletContext object is contained within the ServletConfig object. That is, the ServletContext can be accessed using the ServletConfig object within a servlet. You can specify param-value pairs for ServletContext object in <context-param> tags in web.xml file.

Example code:

```
<web-app>
 <context-param>
<param-name>globalVariable</param-name>
<param-value>gv </param-value>
</context-param>
<web-app>
```

Sample code for web.xml to identify servletconfig using init-param tag :

```
<web-app>

<servlet>
        <servlet-name>redteam</servlet-name>
        <servlet-class>mysite.server.TeamServlet</servlet-class>
        <init-param>
            <param-name>teamColor</param-name>
            <param-value>red</param-value>
        </init-param>
        <init-param>
            <param-name>bgColor</param-name>
            <param-value>#CC0000</param-value>
        </init-param>
    </servlet>

    <servlet>
        <servlet-name>blueteam</servlet-name>
        <servlet-class>mysite.server.TeamServlet</servlet-class>
        <init-param>
            <param-name>teamColor</param-name>
            <param-value>blue</param-value>
        </init-param>
        <init-param>
            <param-name>bgColor</param-name>
            <param-value>#0000CC</param-value>
        </init-param>
    </servlet>

  <servlet-mapping>
        <servlet-name>redteam</servlet-name>
        <url-pattern>/red/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>blueteam</servlet-name>
        <url-pattern>/blue/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Differences between Servelt config and servlet context

ServletConfig

1.ServletConfig available in javax.servlet.*; package
2.ServletConfig object is one per servlet class
3.destroyed once the servlet execution is completed.
4.We should give request explicitly, in order to create ServletConfig object for the first time
5.Object of ServletConfig will be created during initialization process of the servlet
6.This Config object is public to a particular servlet only

ServletContext

1.ServletContext available in javax.servlet.*; package
2.ServletContext object is global to entire web application
3.Object of ServletContext will be created at the time of web application deployment
4.Scope: As long as web application is executing, ServletContext object will be available, and 5.it will be destroyed once the application is removed from the server.
6.ServletContext object will be available even before giving the first request

Deployment Descriptor :

web. xml file is also called Deployment descriptor defines mappings between URL paths and the servlets that handle requests with those paths. The web server uses this configuration to identify the servlet to handle a given request and call the class method that corresponds to the request method.

Servlet Mapping   (Example web.xml)

```
<servlet>
  <servlet-name>watermelon</servlet-name>
  <servlet-class>myservlets.watermelon</servlet-class>
</servlet>

<servlet>
  <servlet-name>garden</servlet-name>
  <servlet-class>myservlets.garden</servlet-class>
</servlet>

<servlet>
  <servlet-name>list</servlet-name>
  <servlet-class>myservlets.list</servlet-class>
</servlet>

<servlet>
  <servlet-name>kiwi</servlet-name>
  <servlet-class>myservlets.kiwi</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>watermelon</servlet-name>
  <url-pattern>/fruit/summer/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>garden</servlet-name>
  <url-pattern>/seeds/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>kiwi</servlet-name>
  <url-pattern>*.abc</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>list</servlet-name>
  <url-pattern>/seedlist</url-pattern>
</servlet-mapping>
```

url-patterns and Servlet Invocation using above web.xml

URL	**Servlet Invoked**
`http://host:port/mywebapp/fruit/summer/index.html`	`watermelon`
`http://host:port/mywebapp/fruit/summer/index.abc`	`watermelon`
`http://host:port/mywebapp/seedlist`	`list`
`http://host:port/mywebapp/seedlist/index.html`	The default servlet, if configured, or an HTTP 404 File Not Found error message. If the mapping for the `list` servlet had been `/seedlist*`, the `list` servlet would be invoked.

`http://host:port/mywebapp/seedlist/pear.abc`	`kiwi` If the mapping for the `list` servlet had been `/seedlist*`, the `list` servlet would be invoked.
`http://host:port/mywebapp/seeds`	`garden`
`http://host:port/mywebapp/seeds/index.html`	`garden`
`http://host:port/mywebapp/index.abc`	`kiwi`

Sample code for Servlet :

HelloWorldServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloWorldServlet extends HttpServlet

{
public void service(HttpServletRequest req,
                    HttpServletResponse res)
      throws IOException
  {
    // Must set the content type first
    res.setContentType("text/html");
    // Now obtain a PrintWriter to insert HTML into
    PrintWriter out = res.getWriter();

    out.println("<html><head><title>" +
             "Hello World!</title></head>");
    out.println("<body><h1>Hello World!</h1></body></html>");
  }
}
```

Methods in ServletConfig Interface

- Allows servlets to get initialization parameters
  - **String getInitParameter(String name)**
  - **Enumeration getInitParameterNames()**
  - **ServletContext getServletContext()**
  - **String getServletClass()**

Methods in ServletRequest Interface

- Used to read data from a client request
  - **String getParameter(String name)**
  - **String[] getParameterValues(String name)**
  - **Enumeration getParameterNames()**

Methods in ServletResponse Interface

- Used to write data to a client response.

– void setContentType(String type)

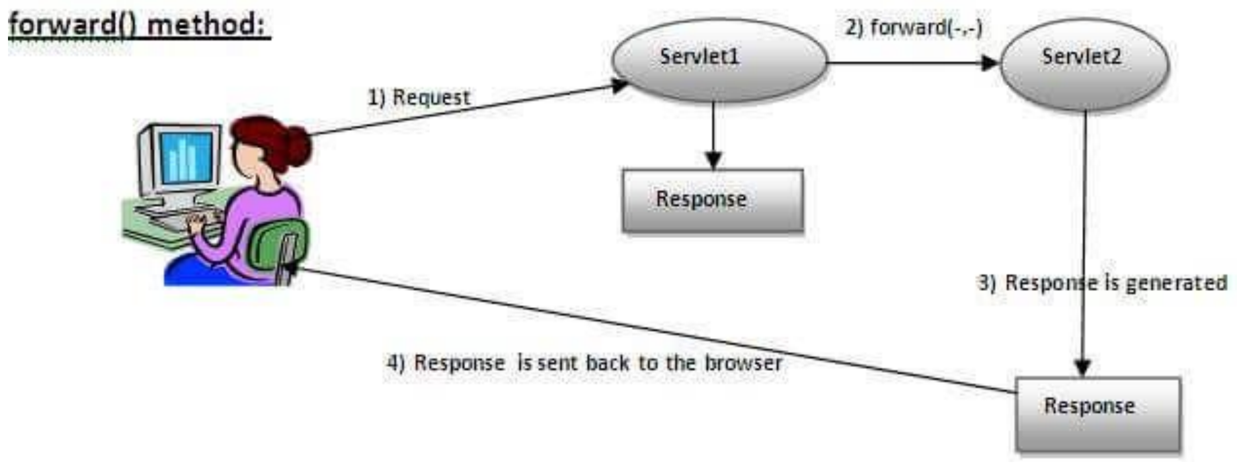
– ServletOutputStream getOutputStream()

– PrintWriter getWriter()

Methods in RequestDispatcher interface

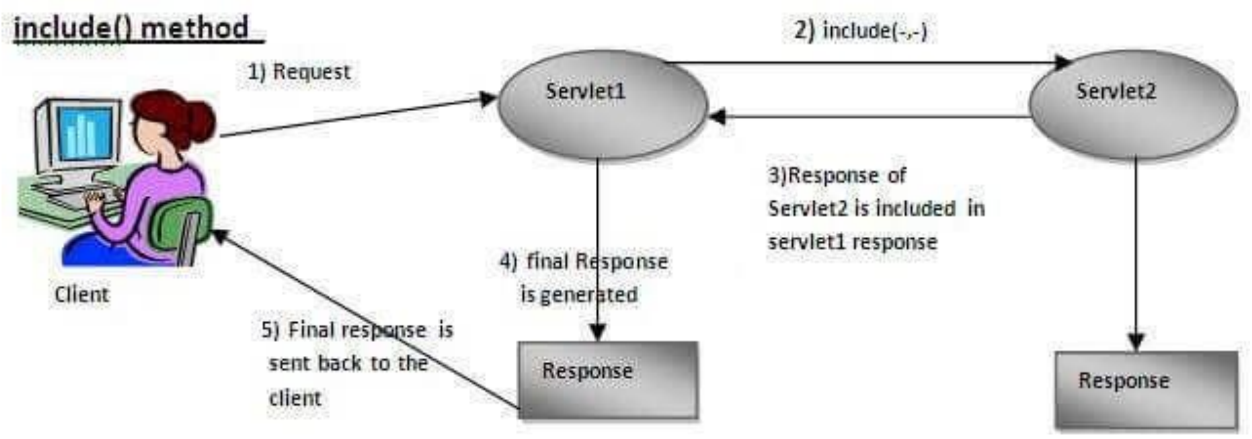
- **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**
  - Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
- **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:**
  - Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

Use of forward() :



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user

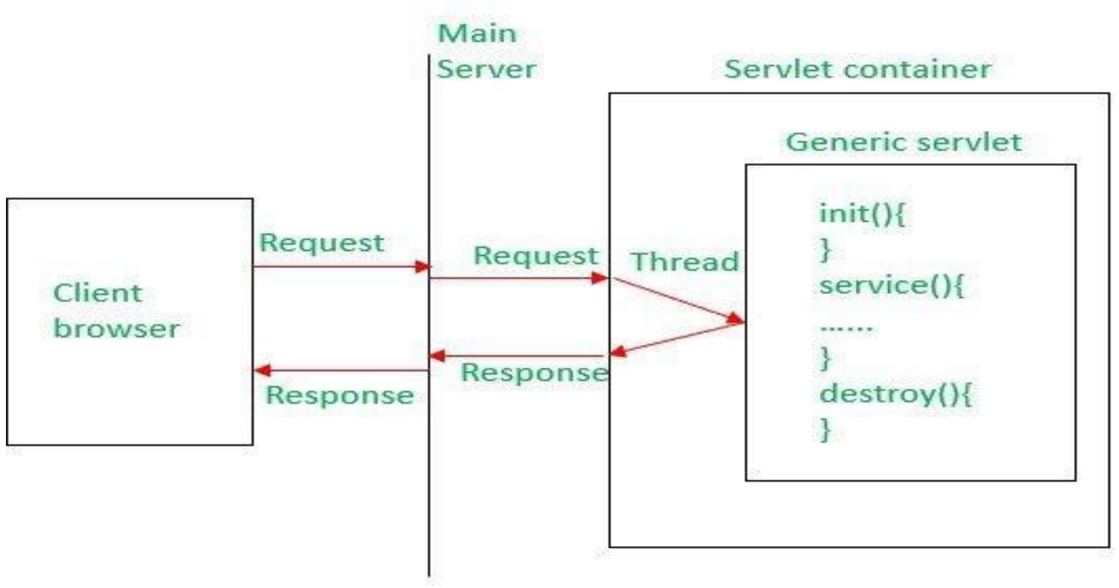
Use of include() :


include() method
1) Request
Servlet1
2) include(-,-)
Servlet2
3)Response of Servlet2 is included in servlet1 response
Client
4) final Response is generated
5) Final response is sent back to the client
Response
Response

As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

Methods in GenericServlet

- Implements the **Servlet** and **ServletConfig** interfaces
    - **void init(ServletConfig config)**
    - **void service(ServletRequest req, ServletResponse res)**
    - **void destroy()**
    - **String getInitParameter(String name)**
    - **Enumeration getInitParameterNames()**
- ServletContext getServletContext()
- String getServletName()
- String getServletInfo()



Methods in ServletInputStream and ServletOutputStream :

- **ServletInputStream** – provides an input stream for reading requests from a client

– public int readLine(byte b[], int offset, int length)

- **ServletOutputStream** –provides an output stream for writing responses to a client

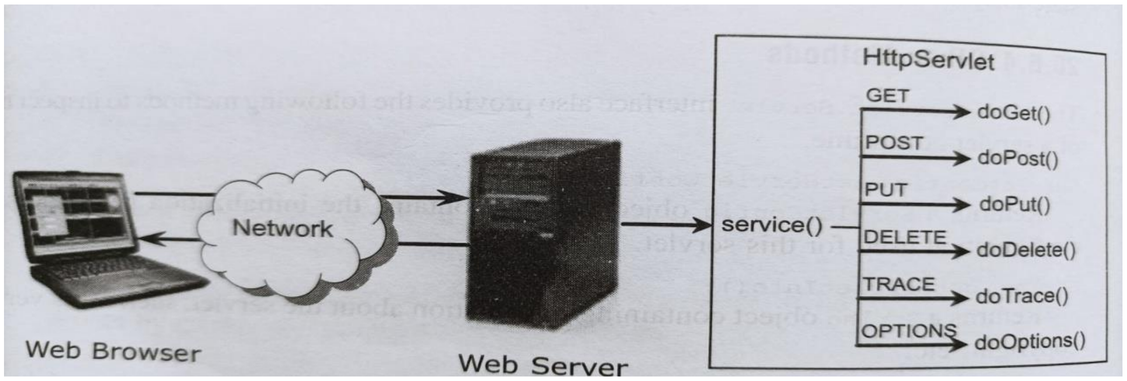
Various methods of ServletOutputStream :

–	public void	print(boolean)	public void println(boolean)
–	public void	print(char)	public void println(char)
–	public void	print(double)	public void println(double)
–	public void	print(float)	public void println(float)
–	public void	print(int)	public void println(int)
–	public void	print(long)	public void println(long)
–	public void	print(String)	**public void println(String)**

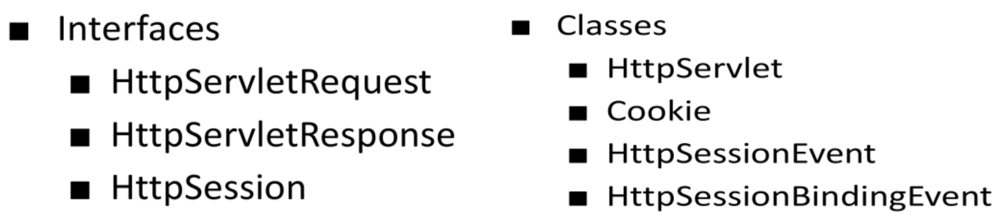
javax.servlet.http :

- This package provides the number of interfaces and classes to support HTTP servlet which is HTTP protocol dependent.
- The two most common *HTTP request types* are ***get*** and ***post***.
- A **get** request *gets* (or *retrieves*) information from a server. Common uses of **get** requests are to retrieve an HTML document or an image.
- A **post** request *posts* (or *sends*) data to a server. Common uses of **post** requests are to send information to a server—such as authentication information, data from a form, information that the server uses to search the Internet or query to a database, etc.
- Class **HttpServlet** defines methods ***doGet*** and ***doPost*** to respond to **get** and **post** requests from a client, respectively.
- These methods are called by the **service** method, which is called when a request arrives at the server.
- Method **service()** first determines the request type, then calls the appropriate method for handling such a request.

Diagram showing various methods of HttpServlet :



Interfaces and Classes



- Interfaces
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
- Classes
  - HttpServlet
  - Cookie
  - HttpSessionEvent
  - HttpSessionBindingEvent

Methods in HttpServlet

- ✓ void doGet(HttpServletRequest req,   HttpServletResponse resp)
- ✓ void doPost(HttpServletRequest req, HttpServletResponse resp)
- ✓ void doPut(HttpServletRequest req, HttpServletResponse resp)
- ✓ void doDelete(HttpServletRequest req, HttpServletResponse resp
- ✓ void doOptions(HttpServletRequest req, HttpServletResponse resp)
- ✓ void doTrace(HttpServletRequest req, HttpServletResponse resp)
- ✓ void doHead(HttpServletRequest req, HttpServletResponse resp)
- ✓ void service(HttpServletRequest req, HttpServletResponse resp)

HTTP Methods :

- GET
- POST – Create or adds the resource at server side
- HEAD – Server Sends only header, without Response body
- PUT – Update/replace the resource at Server Side
- DELETE
- TRACE – To trace the route/debugging
- OPTIONS – for Communication Purpose – What are the options available such as GET,POST,HEAD, PUT, DELETE…..

Differences between GET and POST :

GET	**POST**
Data is visible to everyone in the URL	Data is not displayed in the URL. Data is set as part of message header.
GET is less secure compared to POST because data is appended as part of the URL	POST is a little safer than GET because added with header part.
Only ASCII characters allowed	No restrictions. Binary data is also allowed
When sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Can be bookmarked	Cannot be bookmarked

Can be cached	Not cached
Harmless – read Operations	Maybe – Update, Modify operations

Methods in HttpServletResponse

- void addHeader(String name, String value)
- boolean containsHeader(String name)
- void setHeader(String name, String value)
- void setStatus(int sc)
- void sendError(int sc)
- void sendError(int sc, String msg)
- void sendRedirect(String location)
- void addCookie(Cookie cookie)
- String encodeURL(String url)
- String encodeRedirectURL(String url)
- setContentType(String)

Methods in HttpServletRequest

- String getContextPath()
- String getHeader(String name)
- Enumeration getHeaderNames()
- String getMethod()
- String getQueryString()

- String getRemoteUser()
- String getRequestURI()
- StringBuffer getRequestURL()
- String getServletPath()
- Cookie[] getCookies()
- HttpSession getSession()
- HttpSession getSession(boolean create)
- String getRequestedSessionId()
- boolean isRequestedSessionIdFromCookie()
- boolean isRequestedSessionIdFromUrl()
- boolean isRequestedSessionIdFromURL()
- boolean isRequestedSessionIdValid()

HttpServletRequest Interface

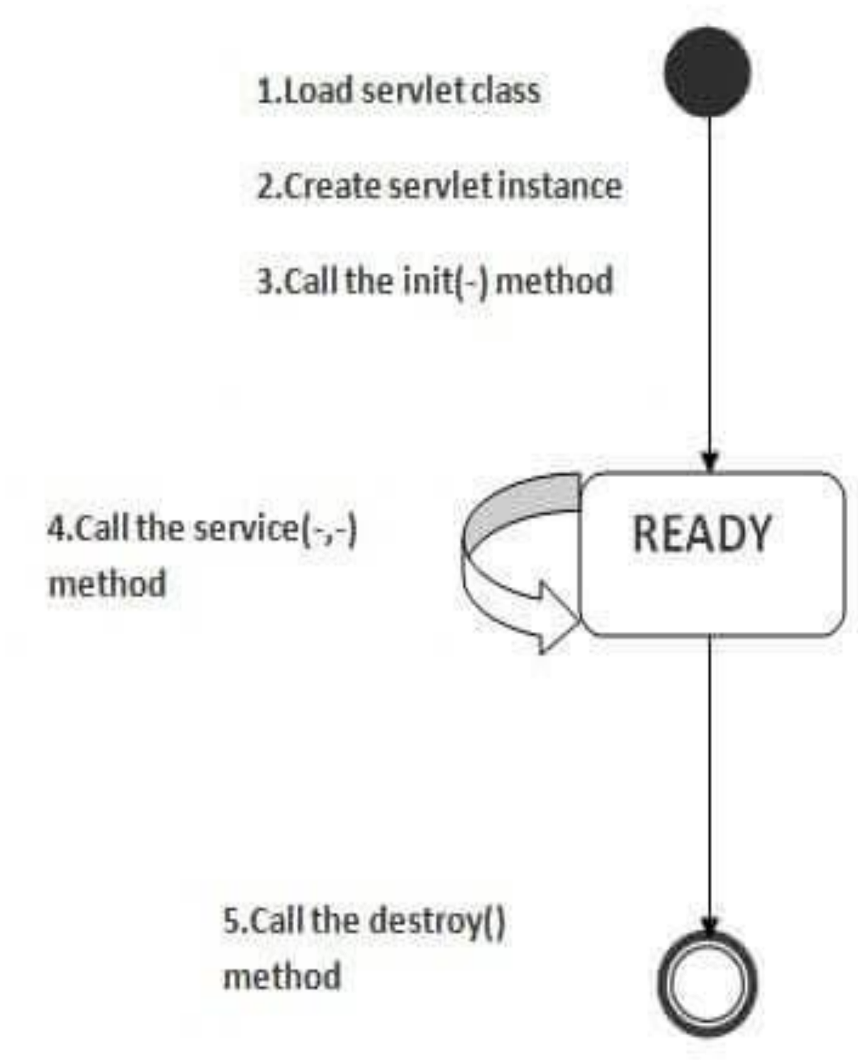
–The role of form Data

- GET – Appended as part of the URL
- POST – As part of the Header

–Form data and parameters

- String getParameter(String name)
- String[] getParameterValues(String name)
- Enumeration getParameterNames()

Life Cycle of a Servlet :


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

- **Servlet life cycle steps**:
  - Load Servlet Class.
  - Create Servlet instance.
  - Call init() method.
  - Call service() method.

- Call destoy() method.

1. Load Servlet Class:
Web container loads the servlet when the first request is received.
This step is executed only once at the time of first request.

2. Create Servlet instance:

- After loading the servlet class web container creates the servlet instance.
- Only one instance is created for a servlet and all concurrent requests are executed on the same servlet instance.

3. Call init() method:

- After creating the servlet instance web container calls the servlet’s init method.
- This method is used to initialize the servlet before processing first request.
- It is called only once by the web container.

Prototype for init() Method :

- public void init(ServletConfig config)throws ServletException

4. Call service() method:

- After initialization process web container calls service method.
- Service method is called for every request.
- For every request servlet creates a separate thread.

Prototype for service() method :

```
public void service(ServletRequest req,ServletResponse res) throws

              ServletException, IOException
```

5. Call destoy() method:

- This method is called by web container before removing the servlet instance.
- Destroy method asks servlet to releases all the resources associated with it.

– It is called only once by the web container when all threads of the servlet have exited or in a timeout case.

Prototype for destroy() Method : public void destroy()

Other Methods:

– The javax.servlet.Servlet interface provides the following methods to inspect the properties of a servlet at run time.

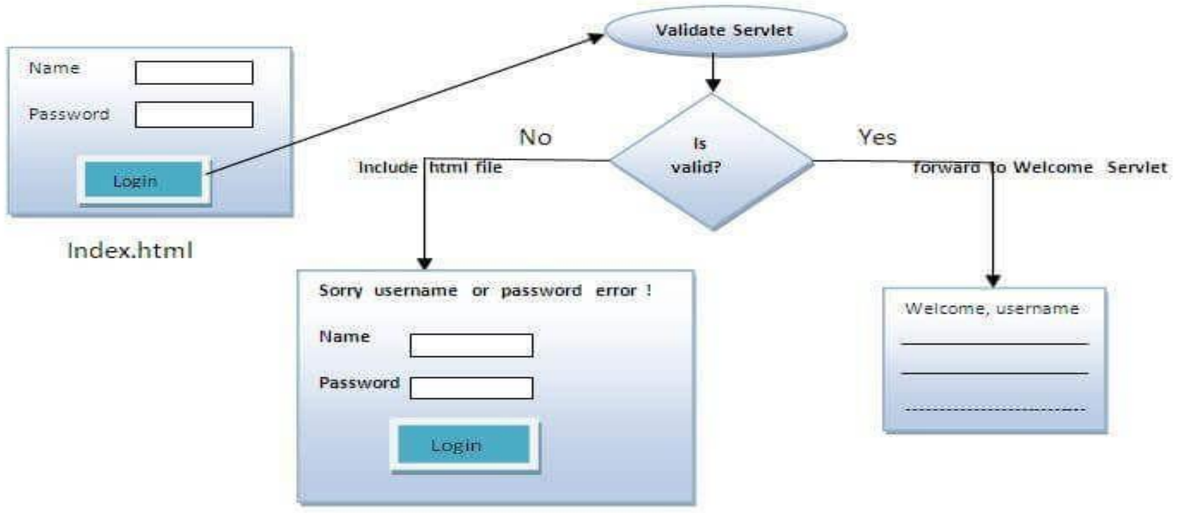
– ServletConfig getServletConfig();

Returns a ServletConfig object, which contains initial parameters, and some other additional information.

– String getServletInfo();

Returns a string object containing info about servlet such as version, author and so on.

Simple Example



Headers :

- HTTP headers are sent by client to the server about the
- software, language and other information the client is using
- HTTP request header can be accessed by server using various methods:
  - getHeader(String) – Returns the specified header value
  - getHeaders(java.lang.String name)Returns all the values of the specified request header as an Enumeration of String objects. (Servers add multiple Cookies)
  - getHeaderNames() – Returns all the headers as enemeration
- File Uploads
  - Two Primary MIME types
    - application/x-www-form-urlencoded – getParameter()
    - multipart/form-data – user has decide –by using
      - » getInputStream()
      - » getReader()

HTTPServletResponse Interface:

- HTTPServletResponse object helps
  - To provide response by obtaining stream
  - To specify the content type
  - To add the header
  - To redirect the HTTP request to another URI

- To send information to the client, we are using the functionality of HTTPServletResponse object by obtaining output stream with either getWriter() or getOutputStream() methods.
- Use only one method to obtain output stream, otherwise exception will come,
- PrintWrite pw=response.getWriter()
- pw.print() ,pw.println() pw.write()

Response Header

- HttpServletResponse object is used to manipulate the HTTP headers of a response.
- This information is used to inform the client about the information about the content such as MIME type, language, protocol version.
- HttpServletResponse object includes the following methods to manipulate HTTP response headers:
  - addHeader(String name, String value)
  - containsHeader(String named) – Returns a boolean value that indicates whether named header is already set or not.
  - setHeader(String name, String value) – sets the name and value as header. The previous value is overwritten.
  - addDateHeader(String name, Date) – adds a response header having name and Date values
  - setDateHeader(String name, Date) – set and overwritten the date value.
  - addIntHeader(String,int) – ads an integer value

Difference between forward() and sendRedirect() method :

Header Field	**Description**
Age	+ve number, seconds, represents estimated time since last response
Content-length	Length of the content which is sent to client
Content-type	MIME Type
Date	Represents date and time
Location	Specifies the location of new resource – sendRedirect()
Retry-after	Server is busy, retry after some time later
Server	Provides description about the server
Refresh	Value in seconds

Response Redirection

- The sendRedirect() method

of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

- It accepts relative as well as absolute URL.
- It works at client side because it uses the URL bar of the browser to make another request.

public void sendRedirect(String URL)throws IOExcep tion

Ex: response.sendRedirect("www.pvpsit.ac.in");

Difference between forward() and sendRedirect() method :

forward() method	**sendRedirect() method**
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2") .forward(request,response);	Example: response.sendRedirect("servlet2")

Auto-Refresh/Wait Pages :

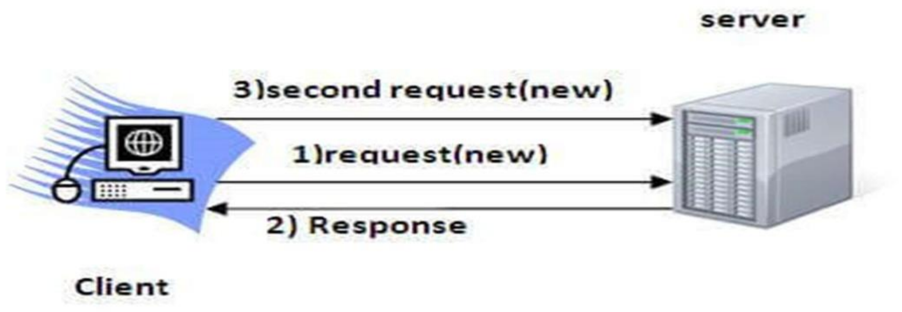
- The other useful response header is to send a wait page or a page that
- automatically refreshes to a new page after a specified amount of time is elapsed.
  - Wait:
    - response.setHeader("Refresh",10);

– 10 – value(10) in seconds

- Refresh
  - response.setHeader(“refresh”, “10;URL=“pvpsit.ac.in”);

Session Tracking :

- **Session** simply means a particular interval of time.
- **Session Tracking** is a way to maintain state (data) of an user. It is also known as session management in servlet.
- **Http protocol** is a stateless so we need to maintain state using session tracking techniques.
- Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to articular user.

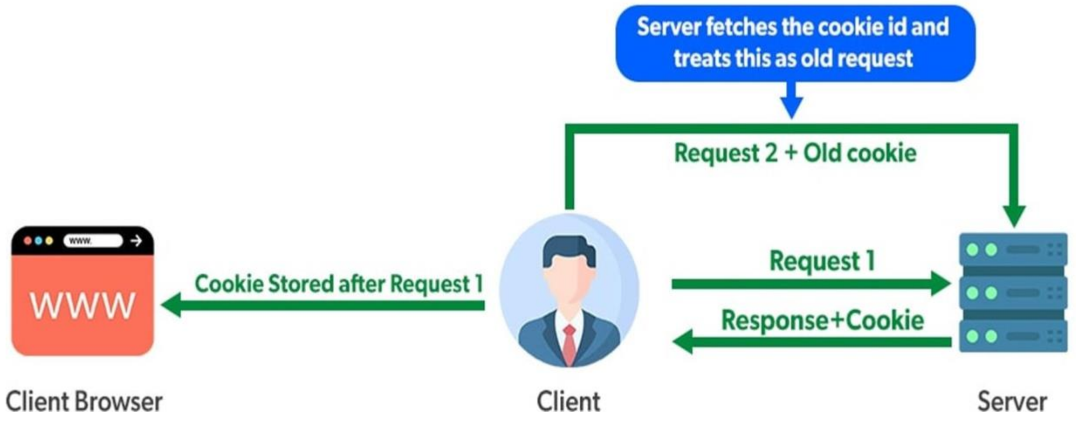


Session Tracking Techniques:

- There are four techniques used in Session tracking:
  - Cookies
  - Hidden Form Field
  - URL Rewriting
  - HttpSession

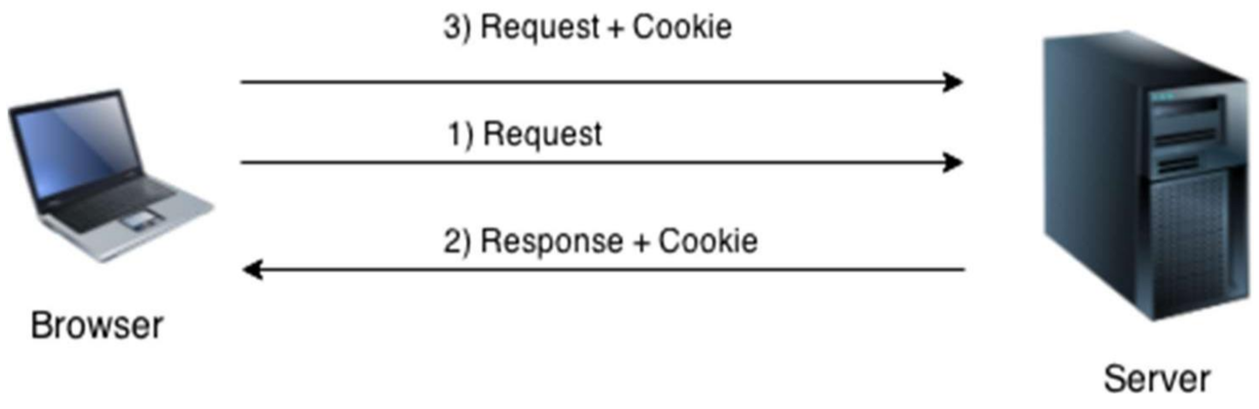
Cookies :

A cookie is a small piece of information that is persisted between the multiple client requests


Server fetches the cookie id and treats this as old request
Request 2 + Old cookie
Cookie Stored after Request 1
Request 1
Response+Cookie
Client Browser
Client
Server

How Cookie works:

- By default, each request is considered as a new request.
- In cookies technique, we add cookie with response from the servlet.
- So cookie is stored in the cache of the browser.
- After that if request is sent by the user, cookie is added with request by default.
- Thus, we recognize the user as the old user.



Types of Cookie :

- There are 2 types of cookies in servlets.
  - Non-persistent cookie
    - It is valid for single session only. It is removed each time when user closes the browser.
  - Persistent cookie
    - It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.
    - Ex: gmail uses persistent cookie for login
- **Advantage of Cookies**
  - Simplest technique of maintaining the state.
  - Cookies are maintained at client side.
- **Disadvantage of Cookies**
  - It will not work if cookie is disabled from the browser.
  - Only textual information can be set in Cookie object.

Cookie Class:

- javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name,String value)	Constructs a cookie with a specified name and value.

Useful Methods of Cookie class:

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

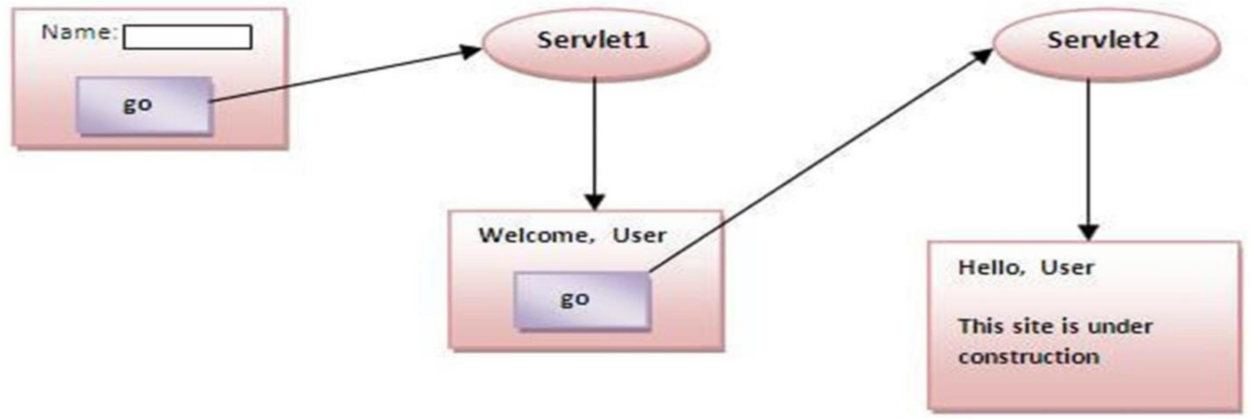
Other methods required for using Cookies :

public void addCookie(Cookie ck): method of HttpServletResponse interface is used to add cookie in response object.
public Cookie[] getCookies(): method of HttpServletRequest interface is used to return all the cookies from the browser.

- How to create Cookie?
  - Cookie ck=new Cookie("admin","pvpsit");
  - response.addCookie(ck);

How to get Cookies?

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++)
{
 out.println( ck[i].getName()+" "+ck[i].getValue())
}
```

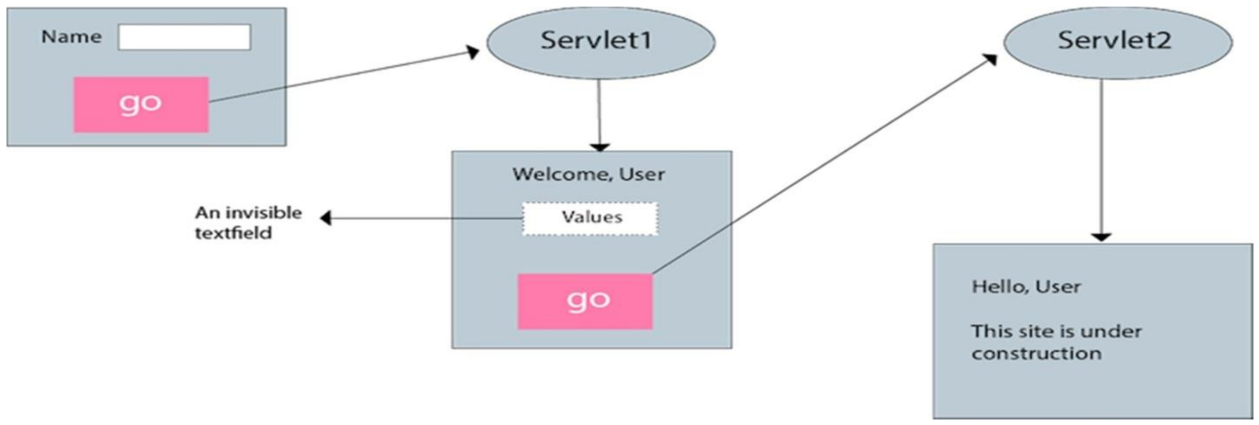


Hidden Form Field :

- In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet.
- Ex:

– <input type="hidden" name="uname" value=“PVPSIT">
Here, uname is the hidden field name and PVPSIT is the hidden field value.

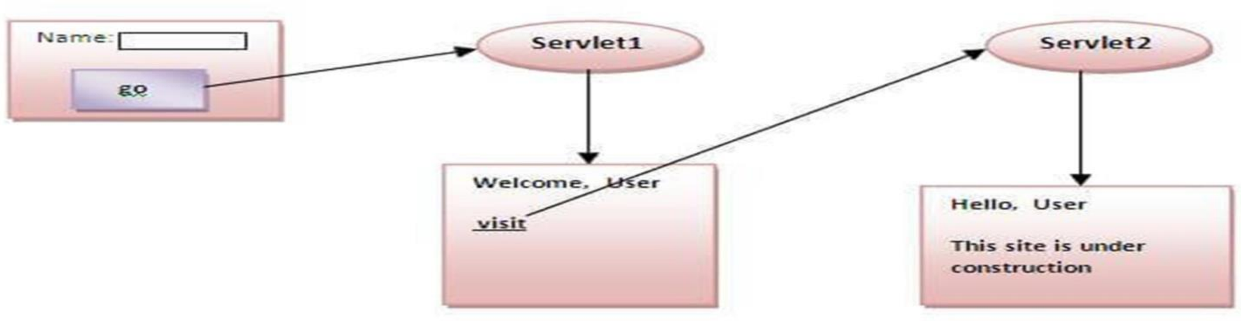
URL Rewriting



URL Rewriting

- In URL rewriting, we append **a token or identifier** to the URL of the next Servlet or the next resource.
- **Token or identifier** - Parameter name/value pairs using the following format:

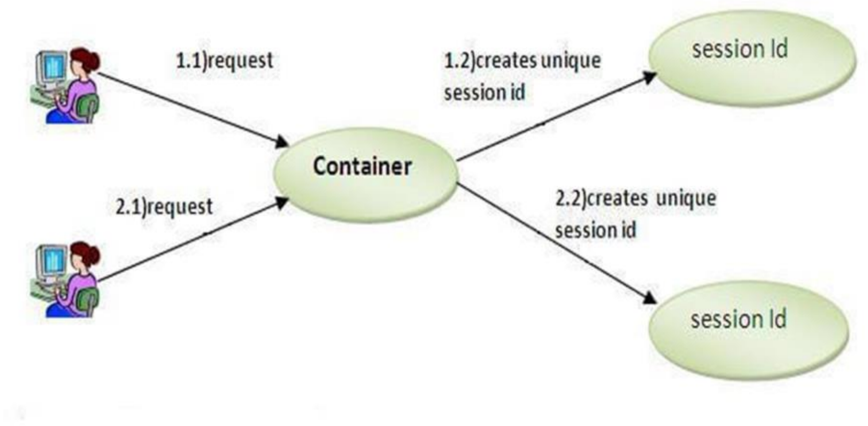
  url?name1=value1&name2=value2&??
- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.
- From a Servlet, we can use getParameter() method to obtain a parameter value.



- **Advantage of URL Rewriting**
  - It will always work whether cookie is disabled or not (browser independent).
  - Extra form submission is not required on each pages.
- **Disadvantage of URL Rewriting**
  - It will work only with links.
  - It can send Only textual information.

HTTP Session Interface

- Web container creates a session id for each user.
- The container uses this id to identify the particular user.
- An object of HttpSession can be used to perform two tasks:
  - bind objects
    - setAttribute(name,value)
  - view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?:

- The HttpServletRequest interface provides two methods to get the object of HttpSession:
  - public HttpSession getSession():Returns the current session associated with this request, or if the request does not have a session, creates one.
  - public HttpSession getSession(boolean create):Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session. If is false it don’t create new session.

Commonly used methods of HttpSession interface :

- **public String getId():**Returns a string containing the unique identifier value.
-
- **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
- **public void invalidate():**Invalidates this session then unbinds any objects bound to it. (clears the session)

HTTPServletResponse Interface :

- HTTPServletResponse object helps
  - To provide response by obtaining stream
  - To specify the content type
  - To add the header
  - To redirect the HTTP request to another URI
- To send information to the client, we are using the functionality of HTTPServletResponse object by obtaining output stream with either getWriter() or getOutputStream() methods.
- Use only one method to obtain output stream, otherwise exception will come,

Servlet Config and Servlet context in Detail :

The Servlet Container

Servlet container, also known as **Servlet engine** is an integrated set of objects that provide a run time environment for Java Servlet components.
In simple words, it is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

ServletConfig:

Signature: public interface ServletConfig

ServletConfig is implemented by the servlet container to initialize a single servlet using init(). That is, you can pass initialization parameters to the servlet using the web.xml deployment descriptor. For understanding, this is similar to a constructor in a java class.

Example code:

```
<servlet>
<servlet-name>ServletConfigTest</servlet-name>
<servlet-class>com.javapapers.ServletConfigTest</servlet-class>
<init-param>
<param-name>sum</param-name>
<param-value>14 </param-value>
</init-param>
</servlet>
```

ServletContext:

Signature: public interface ServletContext

ServletContext is implemented by the servlet container for all servlet to communicate with its servlet container, for example, to get the MIME type of a file, to get dispatch requests, or to write to a log file. That is to get detail about its execution environment. It is applicable only within a single Java Virtual Machine. If a web application is distributed between multiple JVM this will not work. For understanding, this is like a application global variable mechanism for a single web application deployed in only one JVM.

The ServletContext object is contained within the ServletConfig object. That is, the ServletContext can be accessed using the ServletConfig object within a servlet. You can specify param-value pairs for ServletContext object in <context-param> tags in web.xml file.

Example code:

```
<web-app>
 <context-param>
<param-name>globalVariable</param-name>
<param-value>gv </param-value>
</context-param>
<web-app>
```

Differences between Servelt config and servlet context

ServletConfig

1.ServletConfig available in javax.servlet.*; package
2.ServletConfig object is one per servlet class
3.destroyed once the servlet execution is completed.
4.We should give request explicitly, in order to create ServletConfig object for the first time
5.Object of ServletConfig will be created during initialization process of the servlet
6.This Config object is public to a particular servlet only

ServletContext

1.ServletContext available in javax.servlet.*; package
2.ServletContext object is global to entire web application
3.Object of ServletContext will be created at the time of web application deployment
4.Scope: As long as web application is executing, ServletContext object will be available, and 5.it will be destroyed once the application is removed from the server.
6.ServletContext object will be available even before giving the first request

<u>Deployment Descriptor :</u>

web. xml file is also called Deployment descriptor defines mappings between URL paths and the servlets that handle requests with those paths. The web server uses this configuration to identify the servlet to handle a given request and call the class method that corresponds to the request method.

Servlet Mapping   (Example web.xml)

```
<servlet>
  <servlet-name>watermelon</servlet-name>
  <servlet-class>myservlets.watermelon</servlet-class>
</servlet>

<servlet>
  <servlet-name>garden</servlet-name>
  <servlet-class>myservlets.garden</servlet-class>
</servlet>

<servlet>
  <servlet-name>list</servlet-name>
  <servlet-class>myservlets.list</servlet-class>
</servlet>

<servlet>
  <servlet-name>kiwi</servlet-name>
  <servlet-class>myservlets.kiwi</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>watermelon</servlet-name>
  <url-pattern>/fruit/summer/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>garden</servlet-name>
  <url-pattern>/seeds/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>list</servlet-name>
  <url-pattern>/seedlist</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>kiwi</servlet-name>
  <url-pattern>*.abc</url-pattern>
</servlet-mapping>
```

Table 4-1 url-patterns and Servlet Invocation

URL	**Servlet Invoked**
`http://host:port/mywebapp/fruit/summer/index.html`	`watermelon`
`http://host:port/mywebapp/fruit/summer/index.abc`	`watermelon`
`http://host:port/mywebapp/seedlist`	`list`
`http://host:port/mywebapp/seedlist/index.html`	The default servlet, if configured, or an HTTP 404 File Not Found error message. If the mapping for the `list` servlet had been `/seedlist*`, the `list` servlet would be invoked.
`http://host:port/mywebapp/seedlist/pear.abc`	`kiwi` If the mapping for the `list` servlet had

	been `/seedlist*`, the `list` servlet would be invoked.
`http://host:port/mywebapp/seeds`	`garden`
`http://host:port/mywebapp/seeds/index.html`	`garden`
`http://host:port/mywebapp/index.abc`	`kiwi`

Sample code for web.xml :

```
<web-app>

<servlet>
      <servlet-name>redteam</servlet-name>
      <servlet-class>mysite.server.TeamServlet</servlet-class>
      <init-param>
          <param-name>teamColor</param-name>
          <param-value>red</param-value>
      </init-param>
      <init-param>
          <param-name>bgColor</param-name>
          <param-value>#CC0000</param-value>
      </init-param>
  </servlet>

  <servlet>
      <servlet-name>blueteam</servlet-name>
      <servlet-class>mysite.server.TeamServlet</servlet-class>
      <init-param>
          <param-name>teamColor</param-name>
          <param-value>blue</param-value>
      </init-param>
      <init-param>
```

```
            <param-name>bgColor</param-name>
            <param-value>#0000CC</param-value>
        </init-param>
    </servlet>

  <servlet-mapping>
        <servlet-name>redteam</servlet-name>
        <url-pattern>/red/*</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>blueteam</servlet-name>
        <url-pattern>/blue/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Sample code for Servlet :

HelloWorldServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HelloWorldServlet extends HttpServlet

{
public void service(HttpServletRequest req,
                    HttpServletResponse res)
    throws IOException
  {
   // Must set the content type first
   res.setContentType("text/html");
   // Now obtain a PrintWriter to insert HTML into
   PrintWriter out = res.getWriter();
```

```
     out.println("<html><head><title>" +
               "Hello World!</title></head>");
     out.println("<body><h1>Hello World!</h1></body></html>");
    }
  }
```

Link for creating servlets in Netbeans using Glassfish server :
https://www.geeksforgeeks.org/servlet-web-application/

What is a Web Application?

A Web application is sometimes called a Web app, Thus, it is an application that is accessed using a Web browser over the network such as the Internet or an Intranet. While many of the Web applications are written directly in PHP or Perl, Java remains a commonly used programming language for writing Web applications. This is especially true for Web-based enterprise applications as usually referred to as enterprise Web applications. Java EE provides several useful components such as JSP, JSF, Servlets, client-side applets, Enterprise JavaBeans, JDBC, and several other Web service technologies for writing enterprise Web applications. This chapter focuses on the writing Servlets as the Web application of choices.

Basically, a Web application is defined as a hierarchy of directories and files within a standard layout. There are two ways by which such a hierarchy can be accessed:

- The first is where each directory and file exist in the file system separately. This is termed as the application in its unpacked form and used during the application development
- The second is where all the subdirectories are zipped together in a packed form known as a Web ARchive or WAR file. It is used when distributing the applications to be installed

Organization Root Of A Web Application

Before beginning to work with the Servlets, it is useful to examine the runtime organizations of Web applications. Before the Servlet API Specification and version 2.2, across the servers, there was little consistency in the WAR file structure. However, Web servers that conform to the Servlet API version 2.2 [or later] the Web application ARchive must be in a standard format before it is accepted. These are discussed below.

Web Resources

In the Java EE architecture, Web components and static Web content files such as images are called Web resources. A web module is the smallest deployable and usable unit of Web resources. A Java EE Web module corresponds to the Web application as defined in the Java Servlet specification, And The top-level directory of A Web application hierarchy is also a document root of the application. Here, all the HTML files and JSP pages can be placed that comprise the application's user interface.

WEB-INF: *The Document root contains a subdirectory named /WEB-INF/. This subdirectory holds the following files and directories:*

- *web.xml: Is the Web application deployment descriptor file*
- *Classes: Is a directory that holds server side classes such as Servlets, utility classes JavaBeans components and so on*

Deployment Descriptor

Servlet API version 3.0 onwards, the deployment descriptor of [web.xml] takes precedence over the annotations. This deployment descriptor overrides configuration information specified through the annotation mechanism. Version 3.0 of the web deployment descriptor contains a new attribute called metadata complete on the <web-app> element, which defines whether the web descriptor is complete or whether the class files of the web application should be examined for its annotations that specify the deployment information, and if this attribute is set to true, the deployment tool must be ignoring any servlet annotations are present in the class files and it is used only the configuration details mentioned in the descriptor. Otherwise, if the value is not specified or set to be false, the container must scan all classes files of the application for annotations. This will provide a way to enable or disable the scanning of the annotation and its processing during the startup of the application.

Context Path
When the application is being deployed on the particular Web server, Then a context path to the application has been assigned within it. Thus it means if the application is assigned to the context path as a bookshop, then a request of URI referring to /bookshop/index.html will retrieve the index.html file from that document root.

Servlet environment objects :

Servletconfigure
Se

- A servlet is a Java program that runs in a Web server, as opposed to an applet that runs in a client browser. Typically, the servlet takes an HTTP request from a browser, generates dynamic content (such as by querying a database), and provides an HTTP response back to the browser.