Introduction to Web Application and multitier application architecture

- Internet is a global system of inter-connected computer networks. It uses Internet Protocol Suit
- World Wide Web (www, W3) is an information system of interlinked hypertext documents and other digital resources that are accessed via the Internet. It uses Hyper Text Transfer Protocol.

Web Development

- Web development is the work involved in developing a web site for the Internet (World Wide Web) or an intranet (a private network).
- Web development can range from developing a simple single static page of plain text to complex web-based internet applications (web apps), electronic businesses, and social network services.
- A more comprehensive list of tasks to which web development commonly refers, may include:
    - web engineering,
    - web design,
    - web content development,
    - client liaison (A client liaison acts as an intermediary between the company or agency and the client to meet the client's need for information, support, assistance, reports and training.),
    - client-side/server-side scripting,
    - web server and network security configuration,
    - e-commerce development
- In computing, a web application or web app is a client–server computer program that the client (including the user interface and client-side logic) runs in a web browser.
- Common web applications include webmail, online retail sales, online banking, and online auction.

Web Technology

- **Client Side Technologies**: HTML, CSS, JavaScript, AJAX, FLASH, Angularjs
- **Server Side Technologies**: ASP, PHP, Perl, JSP, ASP.NET, Java, MySQL, SQL Server
- **Other Technologies**: XML, XSLT, RSS (Rich Site Summary), WSDL (Web Services Description Language), Ruby on Rails, GRAIL Framework (Grails was previously known as "Groovy on Rails"), REST (Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services.), SOAP (Simple Object Access Protocol)

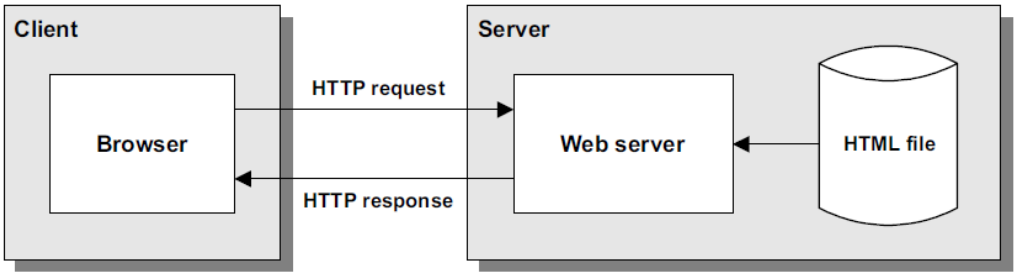
Java Web Technologies

Java Servlet, Java Server Pages (JSP), JSTL, JSF, JDBC ,JNDI ,JMS , JavaPersistenceAPI (old), Spring, Hibernate, Struts, Grails, Blade, Java Server Faces, etc.

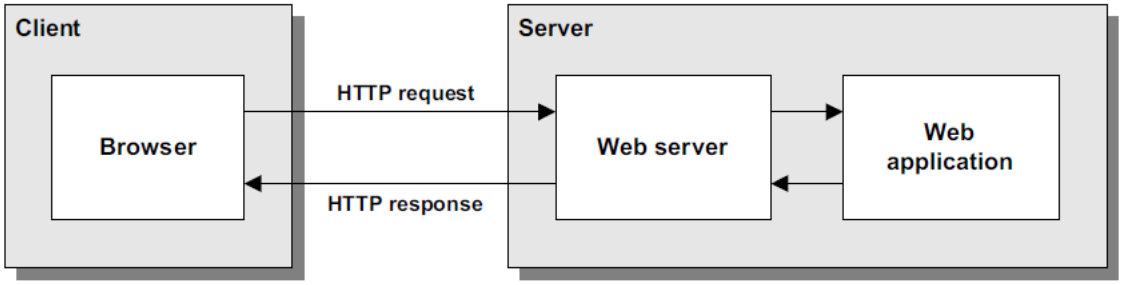
Web Applications

- A web-based application is any program that is accessed over a network connection using HTTP, rather than existing within a device's memory.
- Web-based applications often run inside a web browser.
- However, web-based applications also may be client-based, where a small part of the program is downloaded to a user's desktop, but processing is done over the internet on an external server.
- Web-based applications are also known as web apps.

Static Web Application



Dynamic Web Application



Server: Web vs. Application

- Server is a device or a computer program that accepts and responds to the request made by other program, known as client.
- It is used to manage the network resources and for running the program or software that provides services.

There are two types of servers: Web Server & Application Server

Web Server

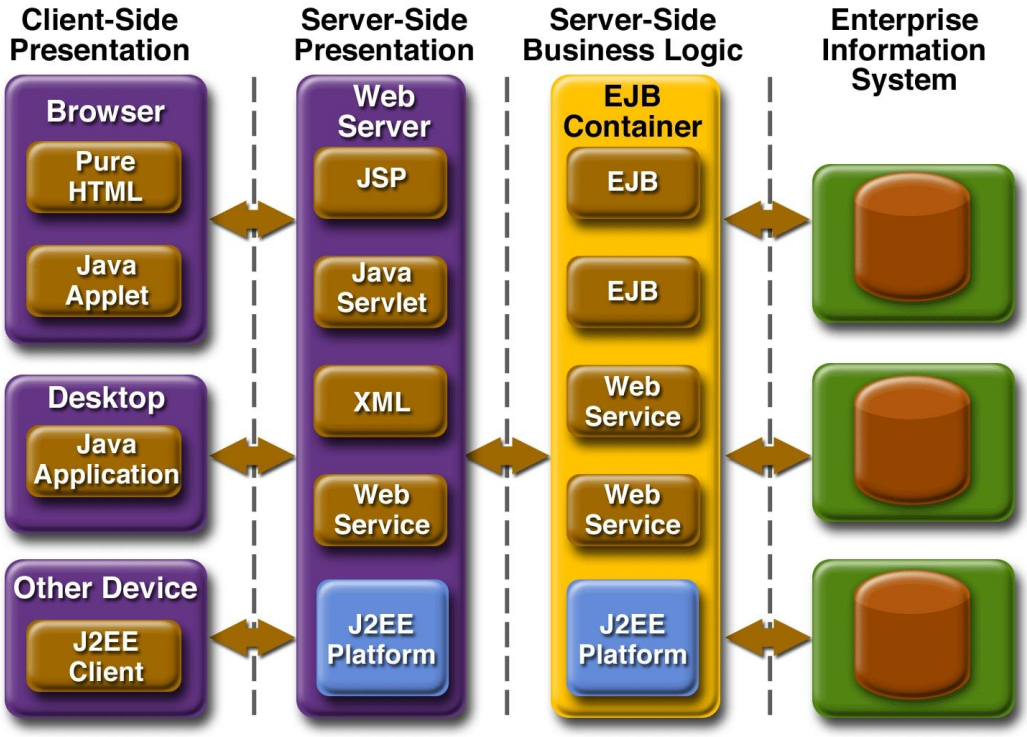
- Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.
- It is a computer where the web content can be stored. In general web server can be used to host the web sites but there also used some other web servers also such as FTP, email, storage, gaming etc.

Application Server

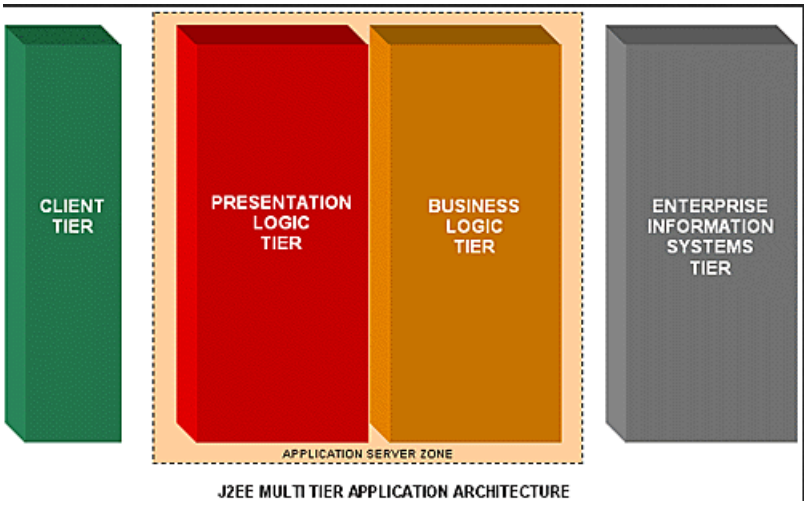
- Application server contains Web and EJB containers.
- It can be used for servlet, jsp, struts, jsf, ejb etc.
- It is a component based product that lies in the middle-tier of a server centric architecture.
- It provides the middleware services for state maintenance and security, along with persistence and data access.
- It is a type of server designed to install, operate and host associated services and applications for the IT services, end users and organizations.
- Typically, it contains a web server within it.

Web server: serves content to the web using http protocol. (Apache is a web server)
Application server: hosts and exposes business logic and processes. (Tomcat is an application server.)

Java 2 Platform, Enterprise Edition (J2EE): Application Model



(ImageCourtesy:https://www.oracle.com/technetwork/java/javaee/appmodel-135059.html)



Advantages:

- Scalable
- Better and Finer Security Control
- Easy to maintain/upgrade
- Better Reusability
- Better Fault Tolerance Ability

Servlet is a server-side, presentation logic component that resides in the Web container. It extends the functionality of a Web server—providing programming capabilities and the capability to generate dynamic contents, apart from serving static HTML.

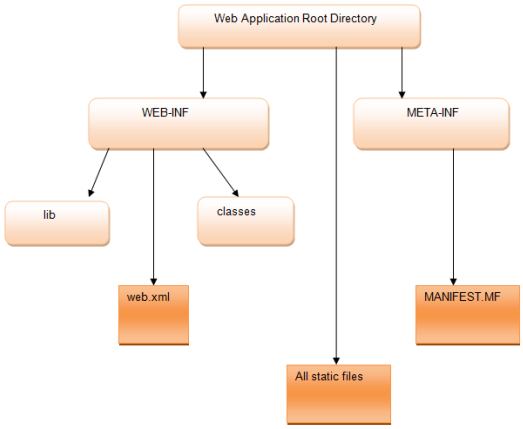
Java Server Page (JSP) is presentation logic components that reside, along with servlet, in the Web container.

- J2EE application components in different tiers come to life in their runtime environments, which are called containers in J2EE terminology.
- These containers provide services like life-cycle management, security, transaction management, JNDI lookups, Remote connectivity etc... to their components.

J2EE standard defines four different containers:

1. Applet container: hosts applets
2. Application client container: hosts standard Java application clients (including swing windows applications)
3. Web container: hosts servlets and JSPs in the presentation logic tier
4. EJB container: hosts Enterprise Javabeans in the business logic tier

Structure of web application



Web Application Root Directory – This is the main or Root folder of web application. Usually name of this folder becomes your web application context.
E.g. if web application name is FirstWebApp, then folder name will be FirstWebApp and web application will be accessible via *http://localhost:8080/FirstWebApp*

WEB-INF- This is the special directory under web application root directory. This is special because this is secured folder and files available within this folder will not be accessible to client directly. If this directory has a file "index.html" then this file cannot be accessed directly via *http://localhost:8080/FirstWebApp/index.html*

- **WEB-INF/lib-** Contains JAR files used by the Web application, including JSP tag libraries.
- **WEB-INF/classes-** Contains server-side classes such as HTTP servlets and utility classes.
- **WEB-INF/web.xml** – it is also known as **deployment descriptor**.
  All the configuration of web application like servlets configuration, filters configuration, welcome file list etc are configured in web.xml.

- **META-INF/MAINFEST.MF-** This is the manifest file: Manifest-Version: 1.0
- **META-INF/Context.xml :** <?xml version="1.0" encoding="UTF-8"?>
  <Context path="/FirstWebApp "/>

Static Files – All static files like HTML, css, javascript will be placed directly under web application root directory. If we want to make these files secure, we need to place these files under WEB-INF directory

EXAMPLE:

```
myWebApp/
        WEB-INF/
                web.xml
                lib/
                        MyLib.jar
                classes/
                        MyPackage/
                                MyServlet.class
        index.html
        index.jsp
```

Servlets

- **Servlet** technology is used to create web application (resides at server side and generates dynamic web page).
- **Servet** technology is robust and scalable as it uses the java language.
- There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.
- Before Servlet, CGI (Common Gateway Interface) scripting language was used as a server-side programming language.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

1. Servlet is a technology i.e. used to create web application.
2. Servlet is an API that provides many interfaces and classes including documentations.
3. Servlet is an interface that must be implemented for creating any servlet.
4. Servlet is a class that extends the capabilities of the servers and responds to the incoming request. It can respond to any type of requests.
5. Servlet is a web component that is deployed on the server to create dynamic web page.

CGI(Commmon 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

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

Advantage 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:
    1. **Better performance:** because it creates a thread for each request not process.
    2. **Portability:** because it uses java language.
    3. **Robust:** Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.
    4. **Secure:** because it uses java language.

Servlet Terminology

1. HTTP
2. HTTP Request Types
3. Get vs Post method
4. Container
5. web server vs. application server
6. Content Type

HTTP (Hyper Text Transfer Protocol)

- Http is the protocol that allows web servers and browsers to exchange data over the web.
- It is a request response protocol.
- Http uses reliable TCP connections by default on TCP port 80.
- It is **stateless** means **each request is considered as the new request**.
- In other words, server doesn't recognize the user by default.

HTTP Request:

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

1. A Request-line
2. Zero or more header (General|Request|Entity) fields followed by CRLF(Carriage Return, Line Feed)
3. An empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields
4. Optionally a message-body

HTTP Request types or Http Request Methods

Every request has a header that tells the status of the client.

	Methods	**Description**
1	GET	• used to request data from a specified resource.
2	POST	• used to send data to a server to create/update a resource. • The data sent to the server with POST is stored in the request body of the HTTP request
3	HEAD	• HEAD is almost identical to GET, but without the response body.
4	PUT	• PUT is used to send data to a server to create/update a resource. • The difference between POST and PUT is that PUT requests are idempotent. • That is, calling the same PUT request multiple times will always produce the same result. • In contrast, calling a POST request repeatedly has side effects of creating the same resource multiple times.
5	DELETE	• The DELETE method deletes the specified resource.

6	TRACE	• The TRACE method echoes the received request so that a client can see what (if any) changes or additions have been made by intermediate servers.
7	OPTIONS	• Returns the HTTP methods that the server supports
8	CONNECT	• The CONNECT method converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy
9	PATCH	• The PATCH method applies partial modifications to a resource.

	GET	**POST**
Visibility	• Data is visible to everyone in the URL	• Data is not displayed in the URL
Security	• GET is less secure compared to POST because data sent is part of the URL • Never use GET when sending passwords or other sensitive information!	• POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Data type	• Only ASCII characters allowed	• No restrictions. • Binary data is also allowed
Data length	• Restricted. 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
History	• Parameters remain in browser history	• Parameters are not saved in browser history
Bookmarked	• Can be bookmarked	• Cannot be bookmarked
Encoding type	• application/x-www-form-urlencoded	• application/x-www-form-urlencoded or multipart/form-data. • Use multipart encoding for binary data
BACK button / Reload	• Harmless	• Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)

GET Request

Request Line:	HTTP method	The path to the web resource on the server	In a Get request, parameters (if there are any) are appended to the first part of the request URL starting with a "?". Parameters are separated with an ampersand "&"	The protocol version that the web browser is requesting.
Request Headers				

POST Request

Request Line:	HTTP method	The path to the web resource on the server	The protocol version that the web browser is requesting.
Request Headers:			
Message Body:	the parameters are in the body, so they are not limited as in GET method		

Content of Request Header:

No.	Header	Description
1	**Accept**	• This header specifies the MIME types that the browser or other clients can handle. • Values of image/png or image/jpeg are the two most common possibilities.
2	**Accept-Charset**	• This header specifies the character sets the browser can use to display the information. • For example ISO-8859-1.
3	**Accept-Encoding**	• This header specifies the types of encodings that the browser knows how to handle. • Values of gzip or compress are the two most common possibilities.
4	**Accept-Language**	• This header specifies the client's preferred languages in case the servlet can produce results in more than one language. • For example en, en-us, ru, etc.
5	**Authorization**	• This header is used by clients to identify them when accessing password-protected Web pages.
6	**Connection**	• This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files with a single request. • A value of Keep-Alive means that persistent connections should be used

7	**Content-Length**	• This header is applicable only to POST requests and gives the size of the POST data in bytes.
8	**Cookie**	• This header returns cookies to servers that previously sent them to the browser.
9	**Host**	• This header specifies the host and port as given in the original URL.
10	**If-Modified-Since**	• This header indicates that the client wants the page only if it has been changed after the specified date. The server sends a code, 304 which means Not Modified header if no newer result is available.
11	**If-Unmodified-Since**	• This header is the reverse of If-Modified- Since; it specifies that the operation should succeed only if the document is older than the specified date.
12	**Referer**	• This header indicates the URL of the referring Web page. • For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the Referer header when the browser requests Web page 2.
13	**User-Agent**	• This header identifies the browser or other client making the request and can be used to return different content to different types of browsers.

HTTP Response

Content Type

- Content Type aka MIME (Multipurpose internet Mail Extension) Type.
- It is a **HTTP header** that provides the description about what are you sending to the browser.
- The simplest MIME type consists of a type and a subtype: **type/subtype**
- There are many content types:

application/octet-stream
application/pdf
application/zip
audio/mp3
audio/mpeg
font/ttf
image/jpeg
image/png,
text/plain
text/csv
text/html
video/mp4

What does the Container do??

1. **Communication Support**: The container provides an easy way for servlets to talk to web server.
2. **Lifecycle management**: The container takes care of loading the classes, instantiating and initializing the servlets, invoking the servlet methods and making servlet instances eligible for garbage collection. i.e controls the lifecycle of servlets.
3. **Multithreading support**: the container automatically creates a new java thread for every servlet request it receives.
4. **Declarative security**: DD (deployment descriptor) can be used to configure and modify security without hard-core in the servlets. Security can be changed without having to recompile classes.
5. **JSP support**: translation of JSP to java code is done by the container.

Servlet API

- The **javax.servlet and javax.servlet.http** packages represent interfaces and classes for servlet API.
- The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
- The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

javax.servlet package

Interfaces	Classes
1. Servlet	1. GenericServlet
2. ServletRequest	2. ServletInputStream
3. ServletResponse	3. ServletOutputStream
4. RequestDispatcher	4. ServletRequestWrapper
5. ServletConfig	5. ServletResponseWrapper
6. ServletContext	6. ServletRequestEvent
7. SingleThreadModel	7. ServletContextEvent
8. Filter	8. ServletRequestAttributeEvent
9. FilterConfig	9. ServletContextAttributeEvent
10. FilterChain	10. ServletException
11. ServletRequestListener	11. UnavailableException
12. ServletRequestAttributeListener	
13. ServletContextListener	
14. ServletContextAttributeListener	

javax.servlet.http package

Interfaces	Classes
1. HttpServletRequest	1. HttpServlet
2. HttpServletResponse	2. Cookie
3. HttpSession	3. HttpServletRequestWrapper
4. HttpSessionListener	4. HttpServletResponseWrapper
5. HttpSessionAttributeListener	5. HttpSessionEvent
6. HttpSessionBindingListener	6. HttpSessionBindingEvent
7. HttpSessionActivationListener	7. HttpUtils (deprecated now)
8. HttpSessionContext (deprecated)	

Servlet Interface

- **Servlet interface** provides common behavior to all the servlets.
- Servlet interface needs to be implemented for creating any servlet (either directly or indirectly).
- It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

- The init, service and destroy are the life cycle methods of servlet which are invoked by the web container.

1. **public void init(ServletConfig config) :** initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
2. **public void service(ServletRequest request, ServletResponse response) :** provides response for the incoming request. It is invoked at each request by the web container.
3. **public void destroy() :** is invoked only once and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig() :** returns the object of **ServletConfig**, which contains **initialization and startup parameters** for this servlet.
5. **public String getServletInfo() :** returns information about servlet such as **author, copyright, version** etc.

GenericServlet class

- **GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces.
- It provides the implementation of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.

Methods of GenericServlet class

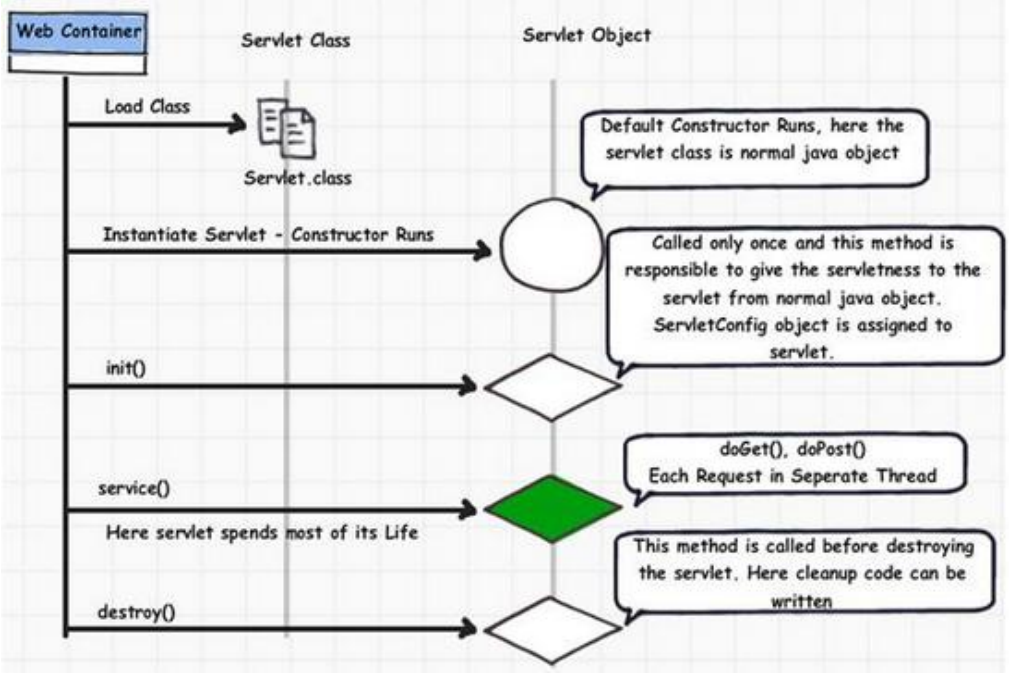
1. public void init(ServletConfig config)
2. public abstract void service(ServletRequest request, ServletResponse response)
3. public void destroy()
4. public ServletConfig getServletConfig()
5. public String getServletInfo()
6. public void init(): it eliminates the need to call super.init(config)
7. public ServletContext getServletContext()
8. public String getInitParameter(String name)
9. public Enumeration getInitParameterNames()
10. public String getServletName()
11. public void log(String msg) 12. public void log(String msg,Throwable t)

HttpServlet class

- extends the **GenericServlet** class and implements **Serializable** interface.
- It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

1. public void service(ServletRequest req,ServletResponse res) dispatches the request to the protected service method by converting the request and response object into http type.
2. protected void service(HttpServletRequest req, HttpServletResponse res) receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. protected void doGet(HttpServletRequest req, HttpServletResponse res) handles the GET request. It is invoked by the web container.
4. protected void doPost(HttpServletRequest req, HttpServletResponse res) handles the POST request. It is invoked by the web container.
5. protected void doHead(HttpServletRequest req, HttpServletResponse res) handles the HEAD request. It is invoked by the web container.
6. protected void doOptions(HttpServletRequest req, HttpServletResponse res) handles the OPTIONS request. It is invoked by the web container.
7. protected void doPut(HttpServletRequest req, HttpServletResponse res) handles the PUT request. It is invoked by the web container.
8. protected void doTrace(HttpServletRequest req, HttpServletResponse res) handles the TRACE request. It is invoked by the web container.
9. protected void doDelete(HttpServletRequest req, HttpServletResponse res) handles the DELETE request. It is invoked by the web container.
10. protected long getLastModified(HttpServletRequest req) returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Life Cycle of a Servlet (Servlet Life Cycle) [container maintains the life cycle]



1. **Servlet class is loaded**

- The class loader is responsible to load the servlet class.
- The servlet class is loaded when the first request for the servlet is received by the web container.

2. **Servlet instance is created**

- The web container creates the instance of a servlet after loading the servlet class.
- The servlet instance is created only once in the servlet life cycle.

3. **init method is invoked**

- The web container calls the init method only once after creating the servlet instance.
- The init method is used to initialize the servlet.
- It is the life cycle method of the javax.servlet.Servlet interface
- public void **init (ServletConfig config)** throws ServletException

4. **service method is invoked**

- The web container calls the service method each time when request for the servlet is received.
- If servlet is not initialized, it follows the first three steps as described above then calls the service method.
- If servlet is initialized, it calls the service method. Notice that servlet is initialized only once.
- **public void service** (ServletRequest request, ServletResponse response) throws ServletException, IOException

5. **destroy method is invoked**

- The web container calls the destroy method before removing the servlet instance from the service.
- It gives the servlet an opportunity to clean up any resource for example memory, thread etc. Syntax: **public void destroy()**

How Servlet works?

The server checks, if the servlet is requested **for the first time**.

If yes, web container does the following tasks:

- loads the servlet class.
- instantiates the servlet class.
- calls the init method passing the ServletConfig object

else

- calls the service method passing request and response objects

The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the application.

How web container handles the servlet request?

The web container is responsible to handle the request.

1. maps the request with the servlet in the web.xml file.
2. creates request and response objects for this request
3. calls the service method on the thread
4. The public service method internally calls the protected service method
5. The protected service method calls the doXXX method depending on the type of request.
6. The doXXX method generates the response and it is passed to the client.
7. After sending the response, the web container deletes the request and response objects. The thread is contained in the thread pool or deleted depends on the server implementation.

What is written inside the public service method?

```
1. public void service(ServletRequest req, ServletResponse res)
2.                                 throws ServletException, IOException {
3.       HttpServletRequest request;
4.       HttpServletResponse response;
5.       try {
6.          request = (HttpServletRequest) req;
7.          response = (HttpServletResponse) res;
8.       }
9.       catch(ClassCastException e)  {
10.         throw new ServletException("non-HTTP request or response");
11.      }
12.      service(request, response);  // calls protected service method
13.   }
```

What is written inside the protected service method?

```
1. protected void service(HttpServletRequest req, HttpServletResponse resp)
2.                                 throws ServletException, IOException {
3.         String method = req.getMethod();
4.         if(method.equals("GET")) {  // checks the type of request
5.            long lastModified = getLastModified(req);
6.            if(lastModified == -1L) {
7.               doGet(req, resp);
8.            }
9.      //rest of the code
10.        }
11.    }
```

Code of a servlet

```
import java.io.IOException;                    import java.io.PrintWriter;
import javax.servlet.ServletException;         import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;
public class FirstServlet extends HttpServlet {
   protected void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         out.println("<!DOCTYPE html>");
         out.println("<html>" + "<body>");
         out.println("<h1>Servlet FirstServlet at"+request.getContextPath() + "</h1>");
         out.println("</body>" + "</html>");
} } }
```

Deployment Descriptor (web.xml)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
        <servlet>
                <servlet-name>helloServlet</servlet-name>
                <servlet-class>com.example.HelloServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name> helloServlet </servlet-name>
                <url-pattern>hello</url-pattern>
        </servlet-mapping>
</web-app>
```

ServletConfig Interface

- An object of ServletConfig is created by the web container for each servlet.
- This object can be used to get configuration information of a servlet from web.xml
- If the configuration information is modified from the web.xml file, we don't need to change the servlet.
- So it is easier to manage the web application if any specific content is modified from time to time.

Methods of ServletConfig interface

1. **public String getInitParameter(String name)** returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames()** returns an enumeration of all the initialization parameter names.
3. **public String getServletName()** returns the name of the servlet.
4. **public ServletContext getServletContext()** returns an object of ServletContext.

How to get the object of ServletConfig

getServletConfig() method of Servlet interface returns the object of ServletConfig.
Syntax : public ServletConfig getServletConfig()

Code of DemoServlet.java to get initialization parameter using ServletConfig

```
ServletConfig config=this.getServletConfig();
String driver=config.getInitParameter("driver");
out.print("Driver is: "+driver);
```

web.xml

```
<web-app>
 <servlet>
    <servlet-name>DemoServlet</servlet-name>
    <servlet-class>DemoServlet</servlet-class>
    <init-param>
        <param-name>driver</param-name>
       <param-value>com.mysql.jdbc.Driver</param-value>
    </init-param>
 </servlet>
 <servlet-mapping>
    <servlet-name>DemoServlet</servlet-name>
    <url-pattern>/servlet1</url-pattern>
</servlet-mapping>
</web-app>
```

Example of ServletConfig to get all the initialization parameters

```
ServletConfig config=getServletConfig();
Enumeration<String> e=config.getInitParameterNames();
String str="";
while(e.hasMoreElements()){
    str=e.nextElement();
    out.print("<br>Name: "+str);
    out.print(" value: "+config.getInitParameter(str));
}
```

```
<servlet>
    <servlet-name>DemoServlet</servlet-name>
    <servlet-class>DemoServlet</servlet-class>
     <init-param>
          <param-name>username</param-name>
          <param-value>admin</param-value>
     </init-param>
     <init-param>
          <param-name>password</param-name>
          <param-value>admin</param-value>
     </init-param>
</servlet>
```

ServletContext Interface

- An object of ServletContext is created by the web container at time of deploying the application.
- This object can be used to get configuration information from web.xml file.
- It is used to communicate with ServletContainer.
- There is only one ServletContext object per web application.
- If any information is shared to many servlets, it is better to provide it from the web.xml file using the **<context-param>** element.

Usage of ServletContext Interface

1. set the attributes for application level
2. and get the initialization parameter values from web.xml
3. get the MIME type of a file
4. get dispatch requests
5. write to a log file

Commonly used methods of ServletContext interface

public String getInitParameter(String name)	returns the parameter value for the specified parameter name
public Enumeration getInitParameterNames()	returns the names of the context's initialization parameters
public void setAttribute(String name, Object object)	sets the given object in the application scope
public Object getAttribute(String name)	returns the attribute for the specified name
public void removeAttribute(String name)	removes the attribute with the given name from the servlet context

How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

Syntax: public ServletContext getServletContext()

Example of ServletContext to get the initialization parameter

DemoServlet.java

```
//creating ServletContext object
        ServletContext context=getServletContext();
//Getting the value of the initialization parameter and printing it
        String driverName=context.getInitParameter("dname");
        out.println("driver name is="+driverName);
```

web.xml

```
<web-app>
        <context-param>
          <param-name>dname</param-name>
          <param-value>com.mysql.jdbc.Driver</param-value>
        </context-param>
</web-app>
```

Example of ServletContext to get all the initialization parameters

```
ServletContext context=getServletContext();
Enumeration<String> e=context.getInitParameterNames();
String str="";
while(e.hasMoreElements()){
        str=e.nextElement();
        out.print("<br> "+context.getInitParameter(str));
}
```

web.xml

```
<web-app>
        <context-param>
                <param-name>dname</param-name>
                <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
        </context-param>
        <context-param>
                <param-name>username</param-name>
                <param-value>admin</param-value>
        </context-param>
        <context-param>
                <param-name>password</param-name>
                <param-value>admin</param-value>
        </context-param>
</web-app>
```

ServletConfig	**ServletContext**
ServletConfig object is one per servlet class	ServletContext object is global to entire web application (One ServletContext per web apps)
Object of ServletConfig will be created during initialization process of the servlet	Object of ServletContext will be created at the time of web application deployment
As long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet is destroyed.	As long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server.
In web.xml – <*init-param*> tag is used under <*servlet*> tag	In web.xml – <*context-param*> tag is used under <*web-app*> tag
Use it to pass deploy-time information to the servlet like name of database that you don't want to hard –code into the servlet.	Use it to access web-app parameters configured in DD.
Use it to access the ServletContext	Use it to get server info including the name and version of the Container and the version of the API that is supported.

Request Parameters

- In many situations user needs to pass some information from browser to web server and ultimately to backend program.
- The browser uses two methods to pass this information to web server.
- These methods are GET Method and POST Method.
- The information passed by user to server program through form fields is called request parameters.
- Servlets handles form data parsing automatically using the following methods of request object depending on the situation −
  1. **getParameter()** − Call this method to get the value of a form parameter.
  2. **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
  3. **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.

```
<form action="NewServlet" method="get">
        <input type="text" name="name"/>
        <input type="checkbox" name="hobby" value="reading" />Reading
        <input type="checkbox" name="hobby" value="singing"/>Singing
        <input type ="submit"/>
 </form>
```

```
out.println("Welcome " + request.getParameter("name"));
Enumeration<String> parameterNames = request.getParameterNames();
while (parameterNames.hasMoreElements()) {
        String paramName = parameterNames.nextElement();
        out.println("<br/>" + paramName);
        String[] paramValues = request.getParameterValues(paramName);
        for (int i = 0; i < paramValues.length; i++) {
                String paramValue = paramValues[i];
                out.println(" " + paramValue);
}       }
```

Html page	**Response of servlet when "button" is press on html page**
niyati ☑ Reading ☐ Singing Submit Query	Welcome niyati name niyati hobby reading
niyati ☑ Reading ☑ Singing Submit Query	Welcome niyati name niyati hobby reading singing

HTTP Response

After receiving and interpreting a request message, a server responds with an HTTP response message:

HTTP Response structure:

- A Status-line
- Zero or more header fields followed by CRLF
- An empty line indicating the end of the header fields
- Optionally a message-body

Message Status-Line

- A Status-Line consists of the protocol version followed by a numeric status code and its associated textual phrase.
- The elements are separated by **space SP characters**.

Status-Line = HTTP-Version SP **Status-Code** SP **Reason-Phrase CRLF**

E.g. HTTP/1.1 200 OK

E.g. HTTP/1.1 404 Not Found

HTTP Version: A server supporting HTTP version 1.1 will return the following version information: HTTP-Version = HTTP/1.1

Status Code

- The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role.
- There are 5 values for the first digit:

No.	Code and Description
1	**1xx: Informational** It means the request was received and the process is continuing.
2	**2xx: Success** It means the action was successfully received, understood, and accepted.
3	**3xx: Redirection** It means further action must be taken in order to complete the request.
4	**4xx: Client Error** It means the request contains incorrect syntax or cannot be fulfilled.
5	**5xx: Server Error** It means the server failed to fulfill an apparently valid request.

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all registered status codes.

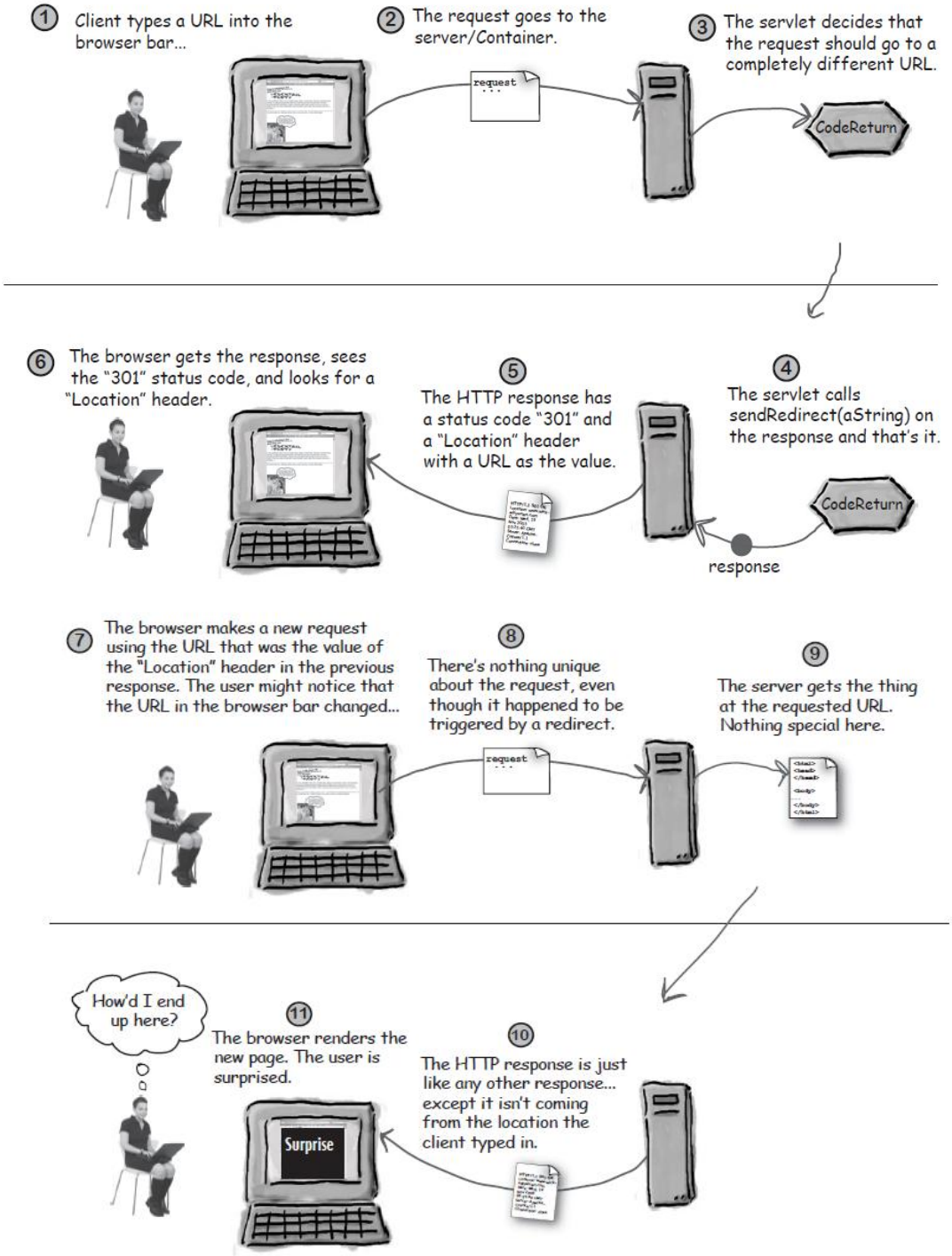
Response Header Fields

	Header & Description
1	**Allow:** This header specifies the request methods (GET, POST, etc.) that the server supports.
2	**Cache-Control:** This header specifies the circumstances in which the response document can safely be cached. It can have values **public**, **private** or **no-cache** etc. Public means document is cacheable, Private means document is for a single user and can only be stored in private (non-shared) caches and nocache means document should never be cached.
3	**Connection:** This header instructs the browser whether to use persistent in HTTP connections or not. A value of **close** instructs the browser not to use persistent HTTP connections and **keepalive** means using persistent connections.
4	**Content-Disposition:** This header lets you request that the browser ask the user to save the response to disk in a file of the given name.
5	**Content-Encoding:** This header specifies the way in which the page was encoded during transmission.
6	**Content-Language:** This header signifies the language in which the document is written. For example en, en-us, ru, etc
7	**Content-Length:** This header indicates the number of bytes in the response. This information is needed only if the browser is using a persistent (keep-alive) HTTP connection.
8	**Content-Type:** This header gives the MIME (Multipurpose Internet Mail Extension) type of the response document.
9	**Expires:** This header specifies the time at which the content should be considered out-of-date and thus no longer be cached.
10	**Last-Modified:** This header indicates when the document was last changed.
11	**Location:** This header should be included with all responses that have a status code in the 300s. This notifies the browser of the document address. The browser automatically reconnects to this location and retrieves the new document.
12	**Refresh:** This header specifies how soon the browser should ask for an updated page. You can specify time in number of seconds after which a page would be refreshed.
13	**Retry-After:** This header can be used in conjunction with a 503 (Service Unavailable) response to tell the client how soon it can repeat its request.
14	**Set-Cookie:** This header specifies a cookie associated with the page.

Delegating the Work

Sometimes you don’t want to deal with the response generation for the request you are getting from the client You can delegate your work to other component using either of the techniques: Redirect or Request Dispatch

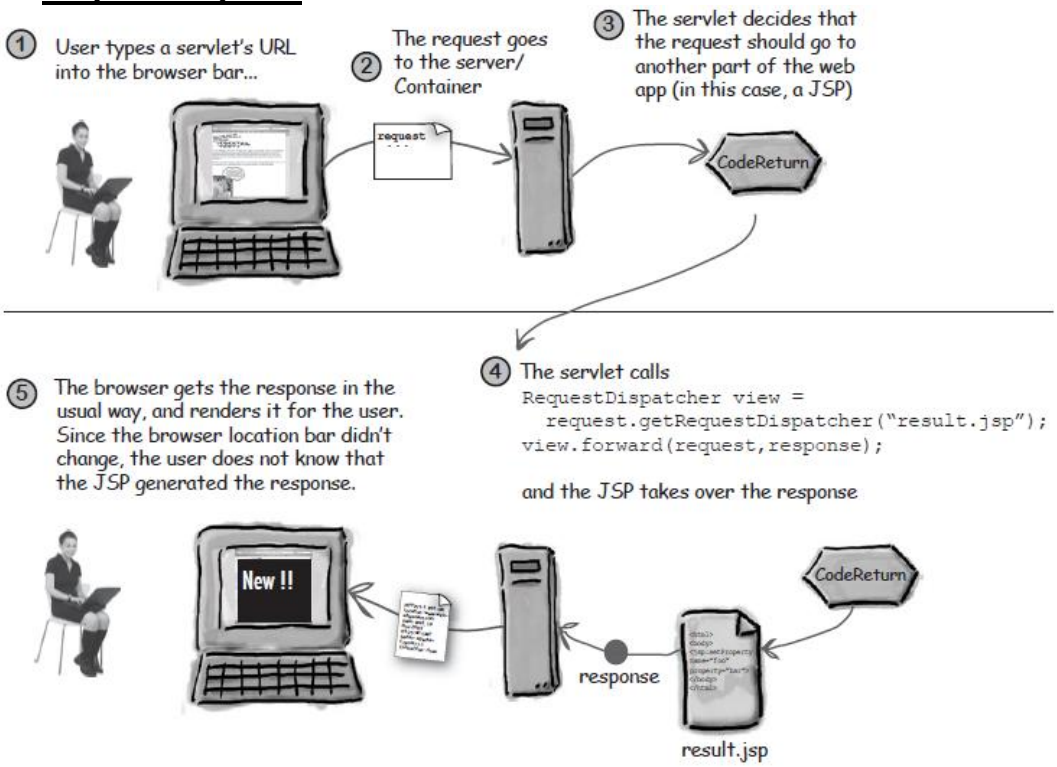
1. **Redirect**



sendRedirect()

- 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.
- So it can go for resources inside or outside the server
- **Syntax:** public void sendRedirect(String URL)throws IOException;
- **Usage:** response.sendRedirect("http://www.google.com");

2. Request Dispatch

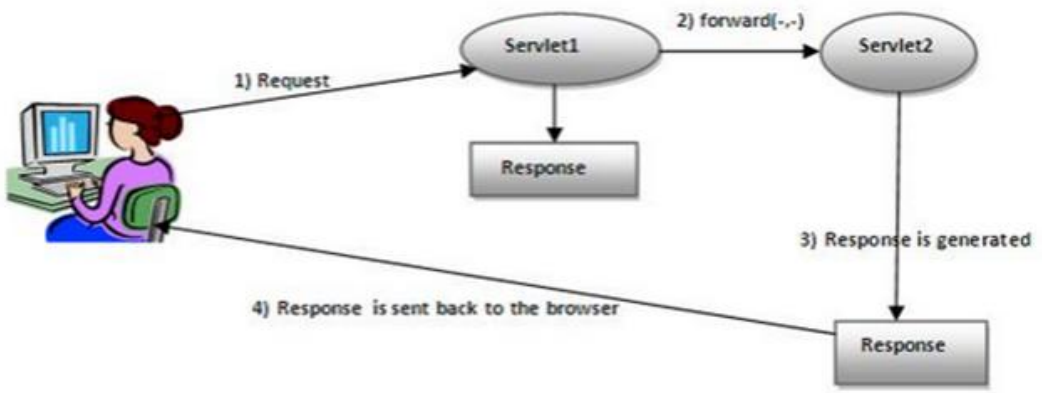


RequestDispatcher – An Interface

- The RequestDispatcher interface provides the facility of **dispatching** the request to another resource it may be html, servlet or jsp.
- This interface can also be used to **include** the content of another resource also.
- There are two methods defined in the RequestDispatcher interface for delegation of work.: forward & include

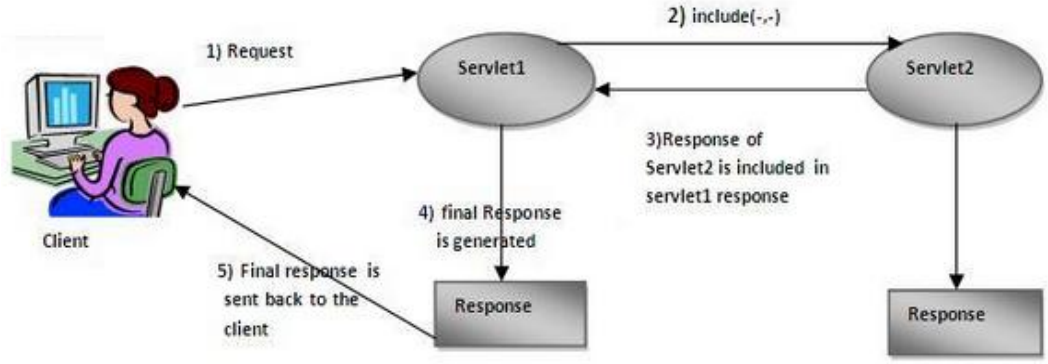
forward() Method of RequestDispatcher

- 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.



include() Method of RequestDispatcher

- 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.

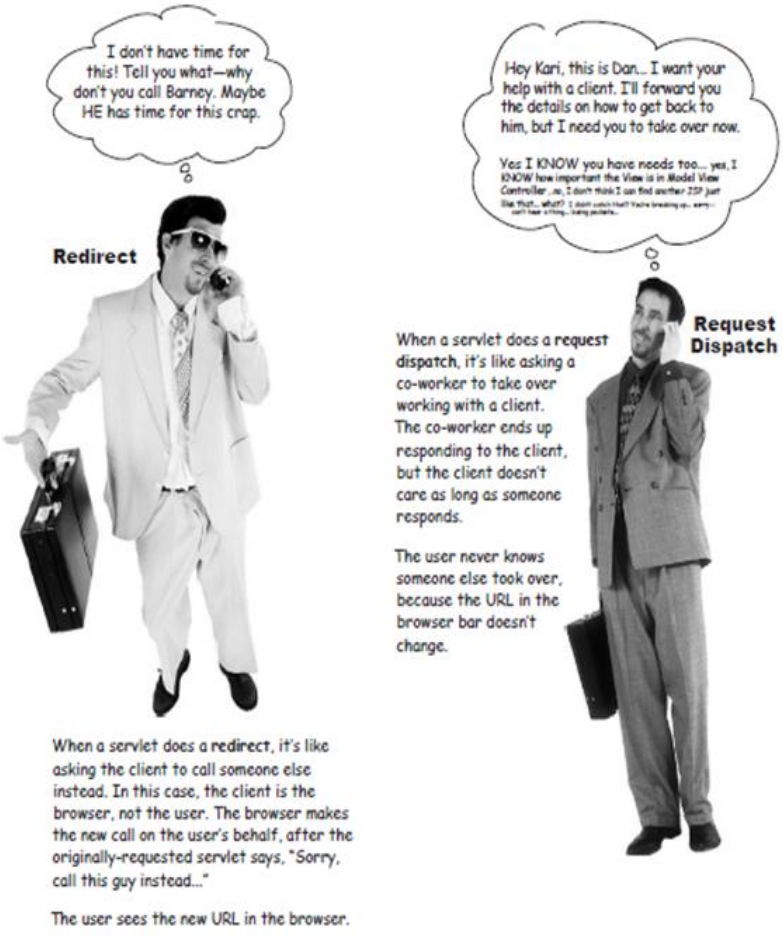


Working with RequestDispatcher

- The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher.
  - **Syntax:** public RequestDispatcher getRequestDispatcher(String resource);
- **Usage:**

```
RequestDispatcher rd=request.getRequestDispatcher("servlet2");
rd.forward(request, response);
```

Redirect vs. RequestDispatch



Program 1: FirstServlet dispatches a forward/include request to SecondServlet of same web

(A) Without parameters

```
RequestDispatcher rd = request.getRequestDispatcher("SecondServlet");
rd.forward(request, response);
rd.include(request, response);
```

Ques. What happpens to the request parameters when request is dispatched?
answer: they get passed.

Ques. Can i add parameters when i dispatch request?
answer: yes.

(B) With parameters

ndex.html

```
<form action="FirstServlet" method="post">
        <input type="text" name="uname"/>
        <input type="submit" name="ok"/>
    </form>
```

FirstServlet.java

```
RequestDispatcher rd = request.getRequestDispatcher("SecondServlet?pass=abc");
rd.forward(request, response);
```

SecondServlet.java

```
out.println(request.getParameter("uname"));
out.println(request.getParameter("pass"));
```

Program 2:

FirstServlet dispatches a forward/include request to a different web app

→ not possible as RequestDispatcher works with same context i.e. within web application only

→following code will not work

```
RequestDispatcher rd =
request.getRequestDispatcher("http://localhost:8084/DemoWebApp/index.html");
rd.forward(request, response);
```

OUPTUT:
HTTP Status 404 - /LecturePrep/http:/localhost:8084/DemoWebApp/index.html

Program 2:

A. FirstServlet redirects request to SecondServlet of same web app

response.sendRedirect("SecondServlet");

-> show 302 in request header

B. FirstServlet redirects request to google homepage

response.sendRedirect("http://www.google.co.in");

C. FirstServlet redirects request to index.html of different web app

response.sendRedirect("http://localhost:8084/DemoWebApp/index.html");

D. FirstServlet redirects request to MyServlet of different web app

response.sendRedirect("http://localhost:8084/DemoWebApp/MyServlet");

NOTE: request parameters are not passed.
They must be hard-coded in the url in argument of sendRedirect

response.sendRedirect("http://localhost:8084/DemoWebApp/MyServlet?uname=nb");

Program 3: Using Response Header

response.setHeader("Refresh","5");
out.println(new java.util.Date());

Program 4:

response.addHeader("name","njb");
OUTPUT: observe response header in browser

Program 5: set status code

response.setStatus(HttpServletResponse.SC_NOT_FOUND);
OUTPUT: observe status code in browser.

What is difference between getRequestDispatcher() of ServletContext and getRequestDispatcher() ServletRequest??

request getRequestDispatcher can be used relative to same context only.
context getRequestDispatcher can be used to dispatch request to other context also.

```
ServletContext context = this.getServletContext().getContext("/DemoWebApp");
RequestDispatcher rd = context.getRequestDispatcher("/index.html");
rd.forward(request, response);
```