UNIT 4

Servlets

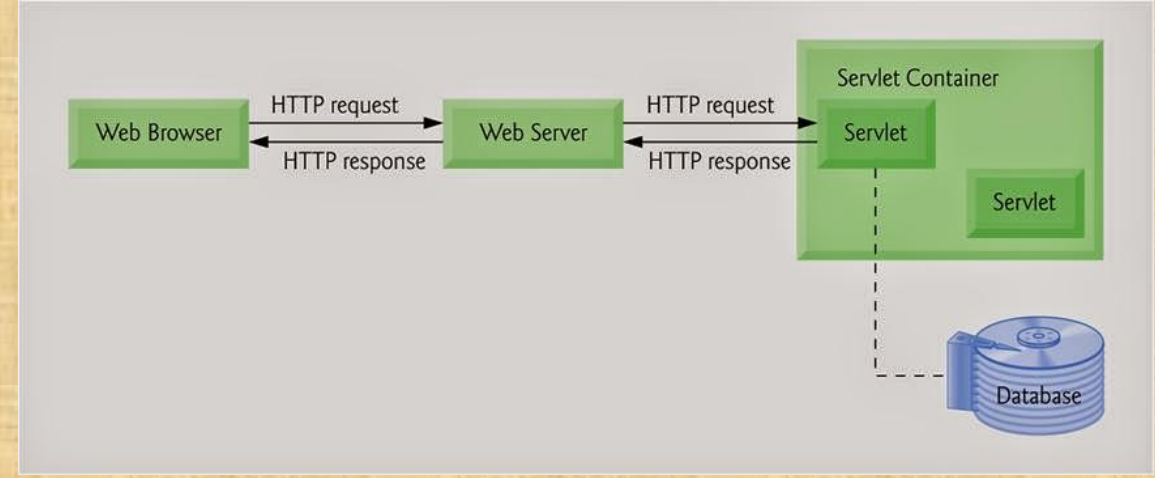
Introduction to Servlets

- A servlet can be thought of as a server-side applet.
- Servlets are modules that run inside request/response-oriented servers, such as Java-enabled web servers.
- Functionally they operate in a very similar way to CGI scripts, however, being Java based they are more platform independent.

What can you build with servlets

- Search engines
- E-commerce applications
- Shopping carts
- Product catalogs
- Personalization systems
- Intranet application
- Groupware applications: bulletin boards, file sharing, etc.

Servlet Architecture



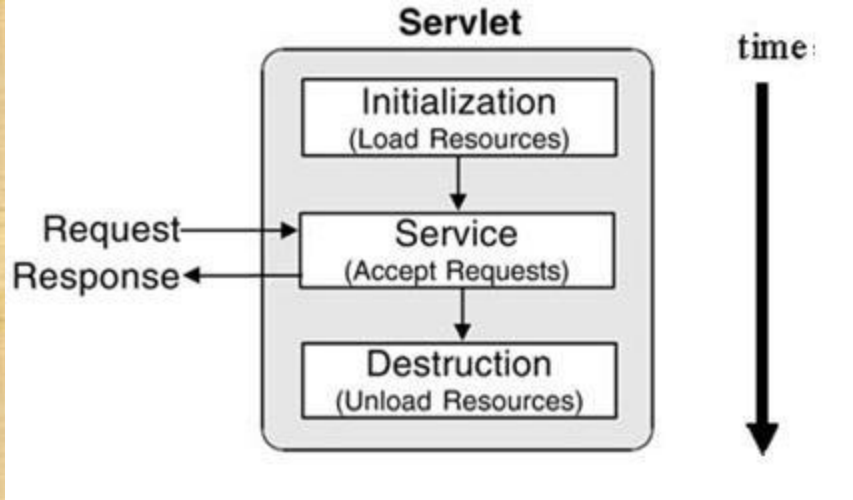
- The client makes a request via HTTP
- The web server receives the requests and forwards it to the servlet
  - If the servlet has not yet been loaded, the web server loads it into the JVM and executes it
- The servlet receives the HTTP request and performs some type of process
- The servlet returns a response to the web server
- The web server forwards the response to the client

Servlet Architecture:

- Servlets read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page.
- Read the implicit HTTP request data sent by the clients (browsers).
- Process the data and generate the results. This process may require talking to a database.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text , images, SpreadSheets etc.
- Send the implicit HTTP response to the clients (browsers).

Servlet Lifecycle

- The entire life cycle of a Servlet is managed by the Servlet container which uses the javax.servlet.Servlet interface.
  - Loading a Servlet.
  - Initializing the Servlet.
  - Request handling and Response Dispatching.
  - Destroying the Servlet.



Servlet Lifecycle: Loading the servlet

- Servlet lifecycle involves loading and initializing the Servlet by the Servlet container.
- The Servlet container performs two operations in this stage :
  - **Loading :** Loads the Servlet class.
  - **Instantiation :** Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.

Initializing a Servlet

- Once Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object.
- The container initializes the Servlet object by invoking the **Servlet.init(ServletConfig)** method which accepts ServletConfig object reference as parameter.
- The Servlet container invokes the **init()** method only once.
- This method is used to initialize the resources, such as JDBC datasource.

Handling request

- After initialization, the Servlet instance is ready to serve the client requests.
- The Servlet container performs the following operations when the Servlet instance is located to service a request :
  - It creates the ServletRequest and ServletResponse objects.
  - In this case, if this is a HTTP request then the Web container creates HttpServletRequest and HttpServletResponse objects
- After creating the request and response objects it invoke the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

Destroying a Servlet

- When a Servlet container decides to destroy the Servlet, it performs the following operations:
  - It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
  - After currently running threads have completed their jobs, the Servlet container calls the destroy() method on the Servlet instance.
- Once destroy() method is executed, the Servlet conatiner releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

Example of Servlet Life Cycle

```
public class MyServlet implements Servlet{
String output;
   public void init(ServletConfig config) throws ServletException
   { output = "Advance Java Concepts";          }
   public void service(ServletRequest res, ServletResponse res)
      throws ServletException, IOException {
         res.setContentType("text/html");
               PrintWriter out = resp.getWriter();
               out.println(output);
     }
   }
   public void destroy(){ System.out.println("Over");  }
   }
```

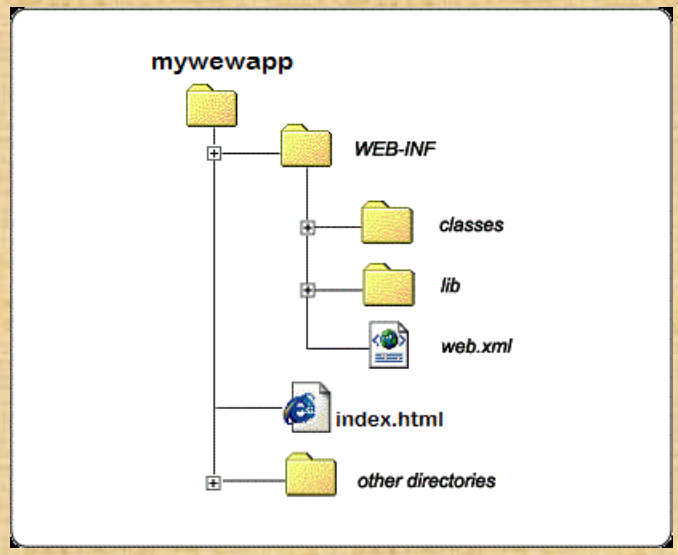
Deploying an Servlet Application

1)Downloading a servlet Conatiner

Jakarta/Apache Tomcat or GlassFish(bundled withNetBeans IDE)

2) Directory Structure of Servlet Application

- An application contain root **folder** with any name.
- Under root **folder** a sub **folder** is required with a name WEB-INF.
- Under WEB-INF two sub **folder** are required classes and lib.
- All jar files placed inside lib **folder**.
- deployment descriptor (web.xml file)



Note: the above directory structure is made at specific location in Servlet Container software. “webapps” folder in Apache Tomcat.

Deploying an Servlet Application

- The **deployment descriptor** is an xml file, from which Web Container gets the information about the servlet to be invoked.
  - There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

```
<web-app>
<servlet>
<servlet-name>map</servlet-name>
<servlet-class>HelloServlet</servlet-
class>
</servlet>
<servlet-mapping>
<servlet-name>map</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Session Tracking Mechanisms

- Three mechanisms of session tracking
  - Cookies
  - URL rewriting
  - Hidden form fields

What is Cookie

- Cookie is a small amount of information sent by a servlet to a web browser
- Saved by the browser, and later sent back to the server in subsequent requests
  - A cookie has a name, a single value, and optional attributes (name/value pair)
  - A cookie's value can uniquely identify a client
- Server uses cookie's value to extract information about the session from some location on the server

Cookies as Session Tracking Mechanism

- Advantage
  - Very easy to implement
  - Highly customable
  - Persist across browser shut-downs
- Disadvantage
  - Users may turn off cookies from privacy or security reason
  - Not quite universal browser support

URL Rewriting

- URLs can be rewritten or encoded to include session information
- URL rewriting usually includes a session ID
- Session ID can be sent as an added parameters:
  - http://.../servlet /Rewritten?sessionid=678

URL Rewriting as Session Tracking

- Advantages
  - Users remain anonymous
  - There are universally supported
- Disadvantages
  - Tedious to rewrite all URLs
  - Only works for dynamically created documents

Hidden Form Fields

- Hidden form fields do not display in the browser, but can be sent back to the server by submit

  <INPUT TYPE="HIDDEN" Name="session" Value ='…'>
- Fields can have identification (session id) or just something to remember
- Servlet reads the fields using request.getParameter()

Hidden Form Fields as Session Tracking

- Advantages
  - Universally supported
  - Allow anonymous users
- Disadvantages
  - Only works for a sequence of dynamically generated forms
  - Breaks down with static documents, emailed documents, bookmarked documents
  - Cannot support browser shutdown

Steps of Doing Session Tracking

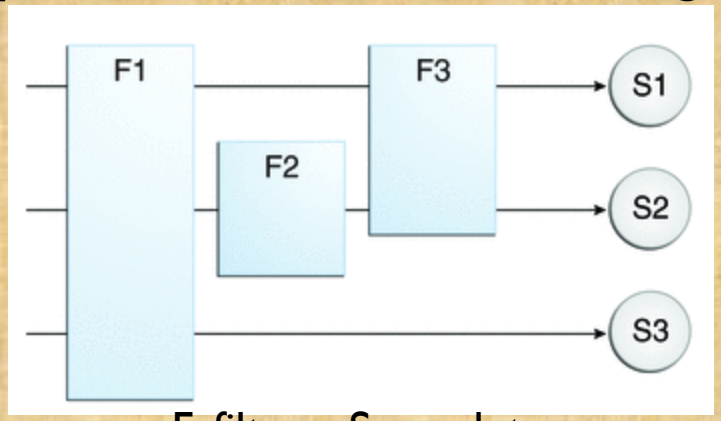
- Programmers have to do the following steps in order to use the aforementioned tracking mechanisms:
  - Generating and maintaining a session id for each session
  - Passing session id to client via either cookie or URL
  - Extracting session id information either from cookie or URL
  - Creating and maintaining a hashtable in which session id and session information are stored
  - Coming up with a scheme in which session information can be added or removed
- These mechanisms can pass "session id", but
  - do not provide high-level programming APIs
  - do not provide a framework from managing sessions

HttpSession class

- To get a user's existing or new session object:
  - HttpSession session = request.getSession(true)
    - flag = true to create a new session if none exists
  - HttpSession is java interface containing methods to
    - View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
    - Bind objects to sessions, allowing user information to persist across multiple user connections
- To Store and retrieve of attribute
  - session.setAttribute("cartItem", cart)
  - session.getAttribute("cartItem")
- All session data are kept on the server
  - Only session ID sent to client

Servlet Filter

- A **filter** is an object that is invoked at the preprocessing and postprocessing of a request.
- It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.
- The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.



F: filters S: servlet

Example: Filter

```
public class MyFilter implements Filter
{
public void init(FilterConfig arg0) throws
                                    ServletException
  { }
public void doFilter(ServletRequest req,
                          ServletResponse resp,
                             FilterChain chain)
  throws IOException, ServletException { }
}
    public void destroy() {}
}
```

Filter Updation in web.xml

```
<filter>
    <filter-name>f1</filter-name>
    <filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>f1</filter-name>
    <url-pattern>/servlet1</url-pattern>
</filter-mapping>
```

Servlet Event Listeners

- The servlet specification includes the capability to track key events in your Web applications through *event listeners*.
- This functionality allows more efficient resource management and automated processing based on event status.
- There are two levels of servlet events:
  - Servlet context-level (application-level) event

    This event involves resources or state held at the level of the application servlet context object.
  - Session-level event

    This event involves resources or state associated with the series of requests from a single user session; that is, associated with the HTTP session object.

Event Listener Scenario

- An event listener mechanism would be to create a servlet context lifecycle event listener to manage the database connection.
- Declaration and Invocation in web.xml

```
<display-name>MyListeningApplication</display-name> <listener>
<listener-class>MyConnectionManager</listenerclass> </listener>
<listener> <listener-class>MyLoggingModule</listener-class>
</listener>
```

Example : Listener class

```
public class SessionLifeCycleEventExample implements
ServletContextListener, HttpSessionListener
{
ServletContext servletContext;
ServletContextListener
public void contextInitialized(ServletContextEvent sce)
{}
public void contextDestroyed(ServletContextEvent sce) {
}
public void sessionCreated(HttpSessionEvent hse) {}
public void sessionDestroyed(HttpSessionEvent hse) {}
}
```

Thank You