Java Institute for Advanced Technology

UNIT NAME: WEB COMPONENT DEVELOPMENT I

UNIT ID: HF2J 04

ASSESSMENT NAME: RESEARCH ASSIGNMENT

NAME: M.R.P.N.THARUKSHA RAJAPAKSHA

STUDENT ID: 2019/2020/CO/SE/I2/029

SCN NO: 207977608

NIC: 200019401866

BRANCH: JAVA INSTITUTE, COLOMBO

Contents

1. Introduction to Java EE Application

1.1 Evaluation of Java EE Platform

Java EE is the Java Enterprise Edition, formerly known as J2EE and now known as Jakarta EE. It is a set of specifications wrapped around Java SE (Standard Edition). Java EE provides a platform for developers with enterprise features such as distributed computerization and web services. Java EE applications typically run on reference runtimes, such as a microserver or application server. Examples of some contexts in which Java EE is used are e-commerce, banking, and accounting information systems.

Java EE has several specifications that are useful for creating web pages, transactional reading and writing from the database, and managing distributed queues. Java EE includes a number of APIs that have the functionality of basic Java SE APIs, such as Enterprise JavaBeans, connectors, Servlets, Java Server Pages, and web service technologies.

The Specifications of JavaEE can be categorized as follows:

- Web Specifications
  - Servlet- This specification defines how you can manage HTTP requests synchronously or asymmetrically. It is low level and other specifications depend on it.
  - WebSocket- WebSocket is a computer communication protocol, and this API provides an API kit to facilitate WebSocket connections.
  - Java Server Faces - A service that helps build GUI components.
  - Unified Expression Language - A simple language designed to facilitate web application developers.

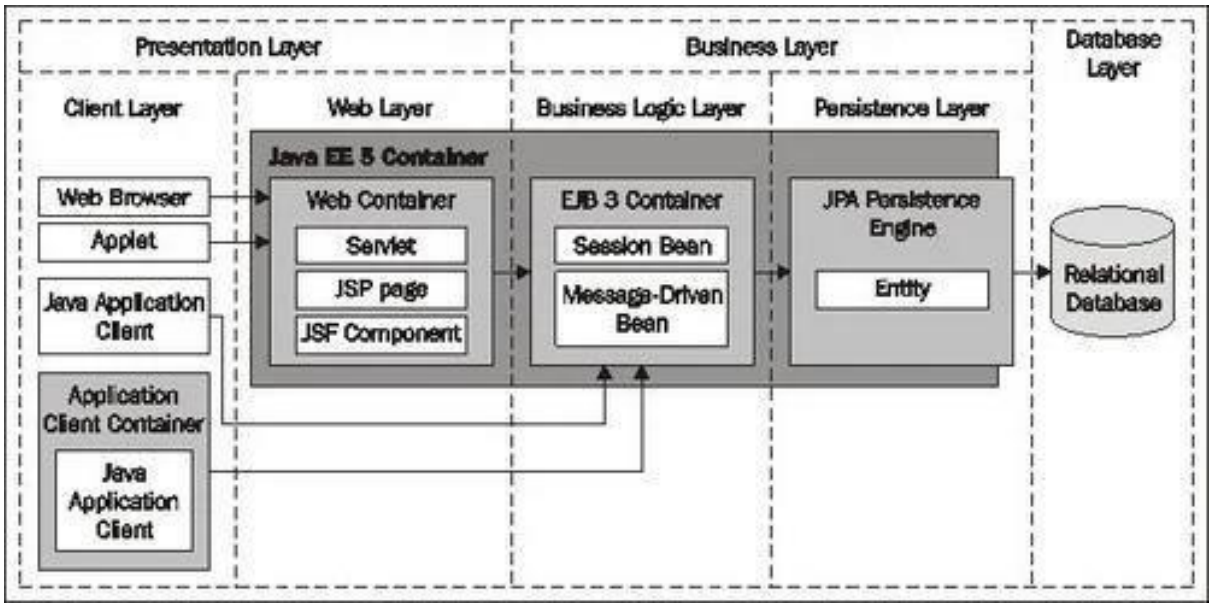
- Web Service Specifications
  - Java API for RESTful Web Services - It helps to provide services having a representational state transfer schema.
  - Java API for JSON Processing - It is a set of specifications for managing the information provided in the JSON format.
  - Java API for JSON Binding - A set of specifications that provide for binding or parsing a JSON file into Java classes.
  - Java Architecture for XML Binding - It allows to binding XML to Java objects.
  - Java API for XML Web Services- SOAP is an XML based protocol for accessing web services via HTTP. This API allows you to create SOAP web services.

- Enterprise Specifications
  - Contexts and Dependency Injection- It provides a container for dependency injection as in Swing.
  - Enterprise JavaBean - is a set of lightweight APIs that an object container possesses in order to provide transactions, remote procedure calls, and concurrency control.
  - Java Persistence API - These are the specifications for object-relationship mapping between contact database tables and Java classes.

- Java Transaction API - Contains interfaces and annotations to establish interaction between the transaction supports offered by Java EE.
- Java Message Service- It provides a common way for the Java program to create, send and read messages in the enterprise messaging system.

- Other Specifications
  - Validation - This package includes various interfaces and annotations to support the declarative validation offered by the Bean Validation API.
  - Batch applications - It provides the means to execute long-running background tasks that involve large volumes of data and need to be executed from time to time.
  - Java EE Connector Architecture - This is a Java-based technology solution for connecting Java servers to the Enterprise Information System.

1.2 Architecture of an Enterprise application



The architecture of enterprise applications has its tasks divided into 3 layers as below:

- Presentation Layer

  This layer is responsible for presenting a user interface and interacting with the client. Servlets and JSP technologies are used in this layer. The client can interact with them using a web browser. These requests are handled by a Web Container.

- Business Layer

  The business logic of the application works on this layer and it is responsible for executing business logic. Enterprise Java Beans (EJB), and Java Persistence API are the specifications used in this layer.

- Database Layer

This layer is responsible for the storage of business data and this layer stores all the data related to enterprise applications on a database server. Typically a relational database management system is used for this layer.

In the above diagram, our 3-layer model has become 5-layers. The distinction between client/web and business logic/persistence layers is not always made. Consequently, we refer to Java EE architecture simply as n-layer or multi-layer.

1.3 Compatible servers

Web Containers such as apache tomcat and application servers like glassfish, JBoss, etc. can be used to run JavaEE applications. Application servers have more features compared to simple web containers like apache tomcat.

2. The Servlet Technology Model

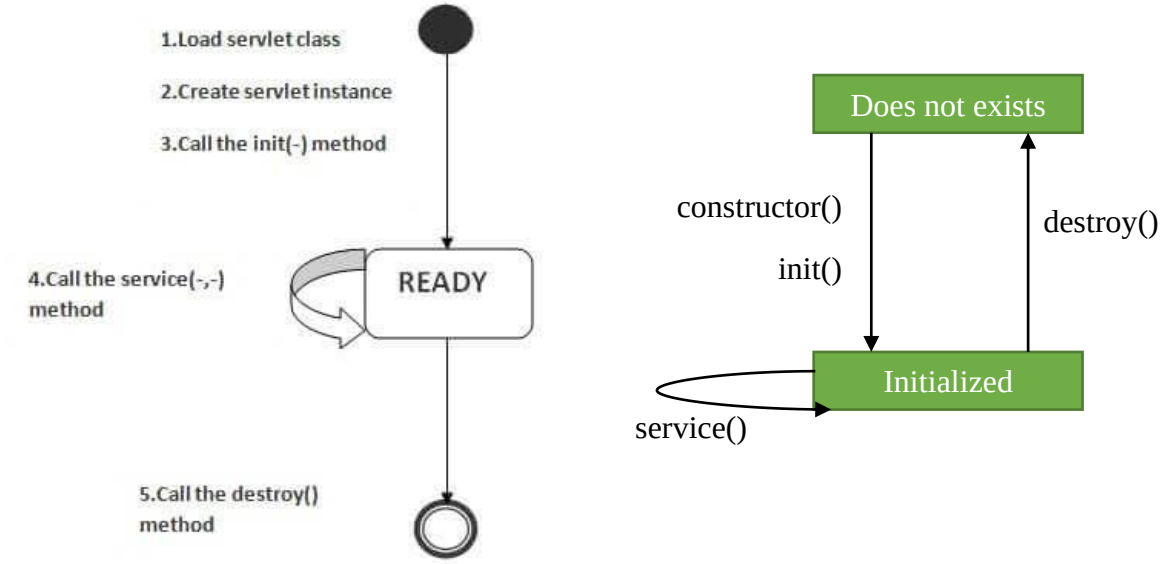
2.1 Components in a web container

The components in a web container are server-side objects used by a web-based client (browser) to interact with J2EE applications.

The Components in a web container can be found in two types:

- Java Servlet: A server-side web component used to process requests and build responses. A servlet is a Java class that responds to HTTP requests.
- JavaServer Pages (JSP): Used to create dynamic web content and server/platform independent web-based applications. We can put Java code inside HTML using JSP. JSP is converted to a servlet when running.

2.2 Managing the lifecycle of servlets and Java Server Pages (JSP)



Steps of lifecycle of servlets and Java Server Pages (JSP):

1. Load Servlet class - The classloader is responsible for loading the servlet class. The servlet class loads when the first request for the servlet is received by the web container.
2. Load Servlet instance - 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. Call the init() method - The web container calls the init() method only once after creating the servlet instance. Uses the init() method to initialize the servlet. It is the life cycle of the *javax.servlet.Servlet* interface. The phrase of the init() method is *public void init(ServletConfig config) throws ServletException* .
4. Call the service() method - The web container calls the service() method each time when a request for the servlet is received. If the servlet is not initialized, it follows the first three steps as described above and then calls the service() method. If the servlet is initialized, it calls the service() method. Notice that the servlet is initialized only once. The syntax of the service() method of the Servlet interface is *public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException* .
5. Call the destroy() method - The web container calls the destroy() method before removing the servlet instance from the service. It gives the servlet an opportunity to clean any resource for example memory, thread, etc. The syntax of the destroy() method of the Servlet interface is *public void destroy()* .

2.3 Usage of Implicit objects in a JSP

The Implicit objects in a JSP are created during the translation phase of the JSP to the servlet. The Implicit objects can be directly used in scriptlets that go into the service() method. Furthermore, they are created by the container automatically, and they can be accessed using objects.

- Out
  Out is one of the implicit objects used to write data to the buffer and send output to the client in response. Out object allows us to access the output stream of the servlet. Out is *the javax.servlet.jsp.jspWriter* class object. When working with servlet, we need the printwriter object.

- Request
  The request object is an instance of *java.servlet.http.HttpServletRequest*, which is one of the arguments of the service() method. It will be made by the container for every request. It will be used to request information such as parameters, header information, server name and etc. It uses getParameter() to access the request parameter.

- Response
  "Response" is an instance of a class that implements the *HttpServletResponse* interface. The container generates this object and passes it to the _jspservice() method as a parameter. The "Response Object" will be created by the container for each request. It represents the response that can give to the client. The response implicit object is used to content type, add cookies, and redirect to the response page.

- Config
  "Config" is the *java.servlet.servletConfig* type. It is created by the container for each JSP page. It is used to obtain the initialization parameter of web.xml.

- Application
  The application object is a case in point *javax.servlet.ServletContext* which is used to get the context information and attributes in JSP. When the application gets deployed, the application object is created by container one per application. The servletcontext object contains a set of methods used to interact with the servlet container.

- Session
  The session is holding “httpsession” object. The session object is used to get, set, and remove attributes to the session scope and is also used to get session information.

- PageContext
  This object is a type of pagecontext. It is used to get, set, and remove the attributes from a specific scope.

- Page
  The page implicit variable holds the currently executed servlet object for the relevant JSP. Acts as this object for the current JSP page.

- Exception
  The exception is the implicit object of the throwable class. It is used for exception handling in JSP. The exception object can only be used in error pages.

3. Process of securely handling multiple requests in a web-based application via the Session Management

3.1 Session lifecycle diagram

When the client first creates the session, it gives the client a unique session id. It stores a session object in the server memory and the reference cookie containing the session id is stored in the client browser. Thereafter the application session cookie is parsed with the HTTP Request each time the client accesses it. This helps to identify the client correctly. The created session objects are in the ready state and can be accessed to read and write data to it. If a session is not currently in use, the server changes its state to passive and stores it in secondary storage to preserve memory. If it is re-accessed, the server will quickly move the passive session object to the ready state. The Sessions have a timeout period, the session will be destroyed when their life is over. When the session is destroyed, the client has to create a new session.

3.2 How to identifying and maintaining the client-side and server-side using Cookies, HTTPServletRequest, HTTPServletResponse and HttpSession

Uses cookies and sessions to identify users returning to a web application. Cookies can be used to store information in the client's web browser, but sessions store information on the server and associate the client with a session id. Each time the client sends an HTTP Request, the cookie data is also sent with it in the request header when a cookie is stored in the web browser. Using sessions can be done with or without the help of cookies. The session data of a session object created for that specific client is stored in the server and the session id is sent to the client with the HTTP Response as a reference. This session id can be stored using a cookie on the client-side or by using URL re-writing where the session id is connected with the URL as a get parameter.

3.3 Working with the session attributes

The Java Session object provides methods to create, update, or remove attributes within a session. These methods are:

- setAttribute(String, Object) - This method is used to create new attributes or change the values of existing attributes.
- getAttribute(String) - This method is used to read the value of a certain attribute.
- removeAttribute(String) - This method is used to remove an attribute from a session.

4. Listening to events in a web container

4.1 Creating and mapping a listener

Listeners can be used to execute tasks on certain web container events. To create and map a listener,

1. Create a class and implement the required listener interface.
2. Map the listener using the <listener> tag on the deployment descriptor or use the annotation @WebListener on the created class.

4.2 Common event interfaces and usages

- ServletContextListener
  This listener can be used to get notified when the servlet context is initialized and destroyed. This event can be used when something is needed to do when the application is deployed or destroyed. Gets the context init parameters from the ServletContext. Stores the database connection as an attribute, so that all parts of the web app can access it. Closes the database connection.

- ServletContextAttributeListener
  This listener can be used to execute something when an attribute in a web app context has been added, removed, or replaced.

- HttpSessionListener
  This listener can be used to execute something when a session has been created and destroyed. (Ex: - If you want to know how many users using the application at the moment you can use this listener.)

- ServletRequestListener
  This listener can be used to execute something each time a request comes in.

- ServletRequestAttributeListener
  This listener can be used to execute something when a request attribute has been added, removed, or replaced.

- HttpSessionBindingListener
  This listener can be used to notify objects when they are bound to or removed from a session.

- HttpSessionAttributeListener
  This listener can be used to execute something when a session attribute has been added, removed, or replaced.

- HttpSessionActivationListener
  This listener can be used to notify objects when the session to which they're bound is migrating to and from another JVM.

5. References

[1] "Wikipedia," [Online]. Available: https://en.wikipedia.org/.

[2] "oracle," [Online]. Available: https://www.oracle.com.

[3] "javatpoint," [Online]. Available: https://www.javatpoint.com/java-ee.

[4] "packtpub," [Online]. Available: https://subscription.packtpub.com/book/application-development/9781847195609/1/ch01lvl1sec01/introduction-to-the-java-ee-architecture#:~:text=Java%20Platform%20Enterprise%20Edition%20(Java,supporting%20the%20Java%20EE%20standard..

[5] "techopedia," [Online]. Available: https://www.techopedia.com/definition/24353/web-components.

[6] "javatpoint," [Online]. Available: https://www.javatpoint.com/life-cycle-of-a-servlet.

[7] "geeksforgeeks," [Online]. Available: https://www.geeksforgeeks.org/life-cycle-of-jsp/.

[8] "tutorialspoint," [Online]. Available: https://www.tutorialspoint.com/jsp/jsp_life_cycle.htm.

[9] "guru99," [Online]. Available: https://www.guru99.com/jsp-implicit-objects.html#:~:text=JSP%20implicit%20objects%20are%20created,can%20be%20accessed%20using%20objects..

[10] "tutorialspoint," [Online]. Available: https://www.tutorialspoint.com/jsp/jsp_implicit_objects.htm.

[11] "javatpoint," [Online]. Available: https://www.javatpoint.com/jsp-implicit-objects.

[12] "oracle," [Online]. Available: https://docs.oracle.com/cd/E19226-01/820-7627/gipln/index.html.

[13] "javatpoint," [Online]. Available: https://www.javatpoint.com/cookies-in-servlet.

[14] "journaldev," [Online]. Available: https://www.journaldev.com/1907/java-session-management-servlet-httpsession-url-rewriting.

[15] "baeldung," [Online]. Available: https://www.baeldung.com/java-servlet-cookies-session.

[16] "baeldung," [Online]. Available: https://www.baeldung.com/spring-mvc-session-attributes.

[17] "journaldev," [Online]. Available: https://www.journaldev.com/1945/servletcontextlistener-servlet-listener-example.