**Name-Nabanita Sarkar**

**Emp ID – 815944**

**Java Web Development Notes**

Java Web Application is used to create dynamic websites. Java provides support for web application through Servlets and JSPs. We can create a website with static HTML pages but when we want the information to be dynamic, we need a web application.

**Servlets**

* Servlet is a technology which is used to create a web application.
* Servlet is an API that provides many interfaces and classes including documentation.
* Servlet is an interface that must be implemented for creating any Servlet.
* Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
* Servlet is a web component that is deployed on the server to create a dynamic web page.

**Servlet API:**

We need to use Servlet API to create servlets. There are two packages that you must remember while using API, the javax.servlet package that contains the classes to support generic servlet (protocol-independent servlet) and the javax.servlet.http package that contains classes to support http servlet.

**Generic Servlet:**

GenericServlet class has an abstract service() method. Which means the subclass of GenericServlet should always override the service() method.  
**Signature of service() method:**

public abstract void service(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException

The service() method accepts two arguments ServletRequest object and ServletResponse object. The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

**HTTP Servlet**

If we creating Http Servlet you must extend javax.servlet.http.HttpServlet class, which is an abstract class. Unlike Generic Servlet, the HTTP Servlet doesn’t override the service() method. Instead it overrides one or more of the following methods. It must override at least one method from the list below:

* **doGet()** – This method is called by servlet service method to handle the HTTP GET request from client. The Get method is used for getting information from the server
* **doPost()** – Used for posting information to the Server
* **doPut()** – This method is similar to doPost method but unlike doPost method where we send information to the server, this method sends file to the server, this is similar to the FTP operation from client to server
* **doDelete()** – allows a client to delete a document, webpage or information from the server
* **init() and destroy()** – Used for managing resources that are held for the life of the servlet
* **getServletInfo()** – Returns information about the servlet, such as author, version, and copyright.

**Life Cycle of Servlet:**

Servlet life cycle contains five steps: 1) Loading of Servlet 2) Creating instance of Servlet 3) Invoke init() once 4) Invoke service() repeatedly for each client request 5) Invoke destroy()

**Step 1: Loading of Servlet**  
When the web server (e.g. Apache Tomcat) starts up, the servlet container deploy and loads all the servlets.

**Step 2: Creating instance of Servlet**  
Once all the Servlet classes loaded, the servlet container creates instances of each servlet class. Servlet container creates only once instance per servlet class and all the requests to the servlet are executed on the same servlet instance.

**Step 3: Invoke init() method**  
Once all the servlet classes are instantiated, the init() method is invoked for each instantiated servlet. This method initializes the servlet. There are certain init parameters that you can specify in the deployment descriptor (web.xml) file

**Step 4: Invoke service() method**  
Each time the web server receives a request for servlet, it spawns a new thread that calls service() method. If the servlet is GenericServlet then the request is served by the service() method itself, if the servlet is HttpServlet then service() method receives the request and dispatches it to the correct handler method based on the type of request.

**Step 5: Invoke destroy() method**  
When servlet container shuts down(this usually happens when we stop the web server), it unloads all the servlets and calls destroy() method for each initialized servlets.

**Web Container**

Tomcat is a web container, when a request is made from Client to web server, it passes the request to web container and it’s web container job to find the correct resource to handle the request (servlet or JSP) and then use the response from the resource to generate the response and provide it to web server. Then the webserver sends the response back to the client.

When web container gets the request and if it’s for servlet then container creates two Objects HTTPServletRequest and HTTPServletResponse. Then it finds the correct servlet based on the URL and creates a thread for the request. Then it invokes the servlet service() method and based on the HTTP method service() method invokes doGet() or doPost() methods. Servlet methods generate the dynamic page and write it to the response. Once servlet thread is complete, the container converts the response to HTTP response and send it back to the client.

**JDBC Concepts:**

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction (API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database (RDB) can be accessed with the help of JDBC APIs. The JDBC classes are contained in the Java Package **java.sql** and **javax.sql**.

**JDBC Drivers**

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

* Type-1 driver or JDBC-ODBC bridge driver
* Type-2 driver or Native-API driver
* Type-3 driver or Network Protocol driver
* Type-4 driver or Thin driver

**Type-1 driver**

Type-1 driver or JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

**Type-2 driver**

The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that’s why data transfer is much more secure as compared to type-1 driver.

**Type-3 driver**

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

**Type-4 driver**

Type-4 driver is also called native protocol driver. This driver interact directly with database. It does not require any native database library, that is why it is also known as Thin Driver.

**JSTL (JSP Standard Tag Library)**

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

**Advantage of JSTL**

* Fast Development JSTL provides many tags that simplify the JSP.
* Code Reusability We can use the JSTL tags on various pages.
* No need to use scriptlet tag It avoids the use of scriptlet tag.

**JSTL Tags**

There JSTL mainly provides five types of tags:

|  |  |
| --- | --- |
| Tag Name | Description |
| [Core tags](https://www.javatpoint.com/jstl-core-tags) | The JSTL core tag provide variable support, URL management, flow control, etc. The URL for the core tag is **http://java.sun.com/jsp/jstl/core**. The prefix of core tag is **c**. |
| [Function tags](https://www.javatpoint.com/jstl-function-tags) | The functions tags provide support for string manipulation and string length. The URL for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**. |
| [Formatting tags](https://www.javatpoint.com/jstl-formatting-tags) | The Formatting tags provide support for message formatting, number and date formatting, etc. The URL for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**. |
| [XML tags](https://www.javatpoint.com/jstl-xml-tags) | The XML tags provide flow control, transformation, etc. The URL for the XML tags is **http://java.sun.com/jsp/jstl/xml** and prefix is **x**. |
| [SQL tags](https://www.javatpoint.com/jstl-sql-tags) | The JSTL SQL tags provide SQL support. The URL for the SQL tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**. |

**CRUD Operation:**

CRUD is an acronym for CREATE, READ, UPDATE and DELETE which are basic functions of persistent storage. CRUD operations can use forms or an interface view to retrieve and return data from a database.

|  |  |
| --- | --- |
| Procedures | Function |
| CREATE | This is a form view to create a new record in the database |
| READ | Reads the table records based on the primary key within the input parameter. |
| UPDATE | Updates the content of the table based on the specified primary key for a record. |
| DELETE | Deletes a specified row in the table. |

**Create Application with CRUD Implementation**

* [Java](https://www.mitrais.com/software-development/java/) Servlets and Java Server Pages (JSP)
* JSP Standard Tag Library (JSTL)
* Java Database Connectivity (JDBC)
* MySQL database
* Apache Tomcat Server
* The following tools can be used for the development:
* Eclipse IDE for Java EE Developers (one of the newer versions is recommended)
* Apache Tomcat ver 8.5
* MySQL Community Server and MySQL Workbench (GUI Tool)
* MySQL Connector for Java

**Steps implementing CRUD Operation:**

1. Create Database
2. Create Project with Maven

(In Eclipse IDE, click File > New > Other and Maven Project.)

1. Writing Model Class

(Next, create a model package. We will use the package name. Then we can create a Java class to model an entity in the database)

1. Writing DAO Class

* Create Datasource Class
* Create a datasource properties file in folder resources:
* Create DAO Interface
* Create new interface class to implement DAO Interface

1. **Create View/Form file**

**(**Create folders for css (in which to save all css/style files) and jsp (in which to save all jsp files) in WebContent folder)

* CSS File

(Create the file style.css in folder css)

* JSP File

(Create a page with JSP for displaying all information from the database)

1. **Creating Controller Servlet Class**

Now, we can implement a Java Servlet that acts as a page controller to handle all requests from the client.

1. **Deploying and Testing the Application**

After we have completed the project code we need to deploy and test the application to check that it works.

Right click on project -> Run As -> Run on Server