Unit – 1 (Foundation of Enterprise Programming)

MCQ Questions

1. What does JDBC stand for?

   A. Java Data Connection
   B. Java Database Connectivity
   C. Java Direct Connect
   D. Java Data Communication

2. Which of the following is NOT a component of JDBC?

   A. JDBC Driver
   B. JDBC API
   C. JDBC Connection Pool
   D. JDBC Database

3. Which type of JDBC driver is also known as the Native-API driver?

   A. Type 1: JDBC-ODBC Bridge driver
   B. Type 2: Native-API driver
   C. Type 3: Network Protocol driver
   D. Type 4: Thin driver

4. Which JDBC driver type is most suitable for web applications due to its platform independence and performance?

   A. Type 1: JDBC-ODBC Bridge driver
   B. Type 2: Native-API driver
   C. Type 3: Network Protocol driver
   D. Type 4: Thin driver

5. What is the correct way to load the Oracle JDBC driver in a Java program?

   A. Class.forName("oracle.jdbc.OracleDriver");
   B. DriverManager.loadDriver("oracle.jdbc.driver.OracleDriver");
   C. Connection conn = new oracle.jdbc.OracleDriver().connect();
   D. DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

6. Which method is used to establish a connection to an Oracle database?

   A. DriverManager.getConnection(String url)
   B. DriverManager.getConnection(String url, Properties info)
   C. DriverManager.getConnection(String url, String user, String password)
   D. All of the above

7. What is the default port number used by Oracle for database connections?

   A. 1521
   B. 1433
   C. 3306
   D. 8080

8. How do you create a Statement object in JDBC?

   A. Statement stmt = new Statement(conn);
   B. Statement stmt = conn.createStatement();
   C. Statement stmt = conn.newStatement();
   D. Statement stmt = conn.getStatement();

9. Which Eclipse plugin is commonly used to integrate Maven with Eclipse?

A. M2Eclipse
B. Maven2IDE
C. EclipseMaven
D. MavenEclipse

10. What is the purpose of the pom.xml file in a Maven project?

A. To configure the IDE settings
B. To manage project dependencies and build configuration
C. To specify the project's Java version
D. To list all Java classes in the project

11. How can you convert an existing Eclipse project to a Maven project?

A. Right-click on the project, select Convert to Maven Project
B. Right-click on the project, select Configure > Convert to Maven Project
C. Right-click on the project, select Add Maven Nature
D. Right-click on the project, select Build Path > Convert to Maven Project

12. Which Maven command in Eclipse is used to update the project's dependencies and configurations from the pom.xml file?

A. Maven > Update Project
B. Maven > Refresh Dependencies
C. Maven > Synchronize
D. Maven > Rebuild

Unit – 1 (Foundation of Enterprise Programming)

Fill in the blanks

1) **XML** is a markup language used to encode documents in a format that is both human-readable and machine-readable.
2) In an XML document, **Opening and Closing** tags are used to define the data elements.
3) **JDBC** is an API in Java used for connecting and executing queries with databases
4) In JDBC architecture, the **Driver Manager** acts as a bridge between the application and the database
5) The **Type-4** driver type is known as the Thin driver in JDBC.
6) To establish a connection with an Oracle database using JDBC, you typically use the **Oracle Thin** driver.
7) In JDBC, the **executeQuery()** method is used to execute a SQL query and return a ResultSet.
8) The **createStatement()** method in the Connection interface is used to create a Statement object for sending SQL statements to the database.
9) In JDBC, the **PreparedStatement()** object is used to execute precompiled SQL queries.
10) **MySql** is a popular open-source relational database management system used with JDBC.
11) In MySQL, the default port number for connecting to the database is **3306**
12) **Maven** is a build automation tool used primarily for Java projects
13) In Maven, the **POM.XML** file is used to configure project dependencies and other settings.
14) To integrate Maven with Eclipse, you typically install the **M2Eclipse** plugin.
15) In a POM.xml file, the **<dependencies>** element is used to specify dependencies for the project.
16) The **mvn compile** command in Maven is used to compile the source code of the project.
17) In Maven, the **<groupId>** element specifies the unique identifier for a project.
18) To add a MySQL JDBC driver dependency in a Maven project, you include it in the **<dependencies>** section of the POM.xml.
19) In a Maven POM.xml file, the **<main>** element is used to specify the main class to be executed.
20) In JDBC, a **CallableStatement** is used to call stored procedures in a database.

Unit – 1 (Foundation of Enterprise Programming)

Short Questions

1. What is JDBC?

JDBC (Java Database Connectivity) is an API that allows Java programs to interact with a wide range of databases.

2. Name the main components of JDBC.

The main components of JDBC are: JDBC API, JDBC Driver Manager, JDBC Test Suite, JDBC-ODBC Bridge.

3. What is the purpose of the JDBC Driver Manager?

The JDBC Driver Manager is used to manage a list of database drivers. It handles the establishment of connections between the Java application and the database.

4. List the types of JDBC drivers.

Type 1: JDBC-ODBC Bridge Driver, Type 2: Native-API Driver, Type 3: Network Protocol Driver, Type 4: Thin Driver.

5. What is the main advantage of the Type 4 JDBC driver?

The main advantage of the Type 4 driver is that it is platform-independent, as it is written entirely in Java.

6. Which JDBC driver type is known as the 'Pure Java' driver?

The Type 4 driver is known as the 'Pure Java' driver.

7. What are the core interfaces provided by JDBC?

The core interfaces are: Driver, Connection, Statement, PreparedStatement, CallableStatement, ResultSet, ResultSetMetaData, DatabaseMetaData.

8. Explain the Connection interface in JDBC.

The Connection interface provides methods for establishing a connection to a database, as well as for managing transactions and closing the connection.

9. How do you establish a connection to a database in JDBC?

You establish a connection by using the DriverManager.getConnection method with the appropriate database URL, username, and password.

10. Provide an example of a JDBC URL for connecting to a MySQL database.

jdbc:mysql://localhost:3306/mydatabase

11. What are CRUD operations?

CRUD operations refer to Create, Read, Update, and Delete operations performed on a database.

12. Write a sample code snippet for a SELECT query using JDBC.

```
Connection  conn = DriverManager.getConnection(url, username, password);
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
while (rs.next()) {
        System.out.println(rs.getString("username"));
}
conn.close();
```

13. What is the RowSet interface in JDBC?

The RowSet interface is a part of the JDBC API that encapsulates the retrieval of data in a tabular format and provides additional features like scrolling and updating rows.

14. Name different types of RowSet implementations.

JdbcRowSet,

CachedRowSet,

WebRowSet,

JoinRowSet,

FilteredRowSet.

15. What is a JdbcRowSet?

A JdbcRowSet is a connected RowSet object, meaning it maintains a connection to the database while processing data.

16. Explain CachedRowSet.

A CachedRowSet is a disconnected RowSet object, meaning it can operate without maintaining a connection to the database. It caches data in memory.

17. What is a WebRowSet?

A WebRowSet is an extension of CachedRowSet that can read and write XML data.

18. Describe JoinRowSet.

A JoinRowSet allows two or more RowSet objects to be combined based on a common column.

19. What is a FilteredRowSet?

A FilteredRowSet provides the capability to apply filtering criteria to rows in a RowSet.

20. How do you configure Apache Tomcat server in Eclipse IDE?

Download and install Apache Tomcat.

Open Eclipse and go to Window > Preferences > Server > Runtime Environments.

Click Add and select Apache Tomcat.

Specify the Tomcat installation directory and click Finish.

Unit – 1 (Foundation of Enterprise Programming)

Long Questions

1. What is XML (eXtensible Markup Language)? How is it different from HTML?

XML is a markup language designed to store and transport data. Unlike HTML, which defines the presentation of data, XML focuses on describing the structure and meaning of data. XML allows for custom tags and is designed to be self-descriptive, making it ideal for data interchange between different systems and platforms.

2. Explain XML Schema (XSD) and its importance in XML documents.

XML Schema (XSD) is a language for describing the structure and constraints of XML documents. It defines the elements, attributes, data types, and rules for valid XML documents in a particular namespace. XSD ensures that XML documents conform to a specific structure and format, providing validation and ensuring interoperability between systems that exchange XML data.

3. What is JDBC (Java Database Connectivity)? Describe its architecture and the role of JDBC drivers.

JDBC is an API that allows Java applications to interact with databases. Its architecture consists of a JDBC API, Driver Manager, Drivers, and Database. The JDBC Driver Manager manages a list of database drivers. Drivers translate Java calls into database-specific calls.

JDBC Drivers are categorized into four types:

JDBC-ODBC Bridge,

Native-API Driver,

Network Protocol Driver, and

Thin Driver.

JDBC facilitates connection establishment, statement execution, result set retrieval, and transaction management with databases.

4. How do you establish a connection to a database using JDBC?

Provide a step-by-step explanation.

To connect to a database using JDBC:

1. Load the database driver using Class.forName() or let JDBC 4.0+ auto-load.

2. Create a connection URL specifying the database location, username, and password.

3. Establish a connection using DriverManager.getConnection(url, username, password).

4. Optionally, set auto-commit or transaction isolation levels.

5. Execute SQL queries/statements using Statement, PreparedStatement, or CallableStatement. 6. Handle results and exceptions appropriately.

5.What are PreparedStatement objects in JDBC? How are they different from Statement objects?

PreparedStatement in JDBC is a precompiled SQL statement that can accept input parameters at runtime. It enhances performance by precompiling and caching the SQL statement, making it efficient for repeated executions with different parameters. Unlike Statement, which is used for static SQL queries, PreparedStatement is suitable for dyna

mic queries and prevents SQL injection attacks through parameterized queries.

6. Explain the ResultSet interface in JDBC. How do you navigate through a ResultSet and retrieve data?

ResultSet in JDBC represents the result set of a SQL query. It provides methods to navigate through rows and retrieve data from columns. Steps to navigate and retrieve data: 1. Use next() to move to the next row. 2. Use getXxx() methods (getString(), getInt(), etc.) to retrieve column values by index or column name. 3. Iterate until next() returns false. 4. Close the ResultSet and Statement objects after use to release resources.

7. What are transactions in JDBC? How do you manage transactions using JDBC?

Transactions in JDBC are sequences of operations that are treated as a single unit of work, ensuring data consistency and integrity.

Manage transactions using:

1. Begin a transaction with connection.setAutoCommit(false) or connection.setTransactionIsolation().

2. Execute SQL statements.

3. Commit the transaction using connection.commit() or rollback using connection.rollback() in case of errors.

4. Set auto-commit to true or handle exceptions in a try-catch-finally block. JDBC supports transaction management through Connection methods and SQL commands (COMMIT, ROLLBACK)

8. How does JDBC batch processing improve performance? Explain with an example.

JDBC batch processing allows executing multiple SQL statements in a batch, reducing communication overhead with the database and improving performance.

Example:

1. Create a Statement or PreparedStatement.

2. Add SQL queries using addBatch().

3. Execute batch using executeBatch().

4. Process results or handle exceptions. Batch processing is useful for bulk inserts, updates, or deletes where multiple similar queries are executed together.

9. How do you connect to an Oracle database using JDBC? Describe Oracle-specific considerations.

To connect to Oracle using JDBC:

1. Include Oracle JDBC driver in the classpath (ojdbc.jar).

2. Use Class.forName() to load the driver.

3.Construct connection URL (jdbc:oracle:thin:@hostname:port:SID).

4. Obtain connection using DriverManager.getConnection(url, username, password).

5. Handle exceptions and close resources. Considerations: Oracle specific JDBC URL format, driver version compatibility, and configuration of Oracle JDBC properties like connection pooling and performance tuning.

10. Explain how to connect to a MySQL database using JDBC. Discuss MySQL-specific configurations.

Connect to MySQL using JDBC:

1. Include MySQL JDBC driver (mysql-connector-java.jar).

2. Load driver using Class.forName() or allow auto-loading.

3. Create connection URL (jdbc:mysql://hostname:port/database).

4. Establish connection with DriverManager.getConnection(url, username, password).

5. Handle exceptions and close resources. MySQL considerations: JDBC URL format, driver version compatibility, and configuring connection parameters like SSL, timezone, and character encoding.

11. What is Maven? Describe its features and benefits in software development.

Maven is a build automation tool that manages project dependencies, builds, and documentation. Features include dependency management, project lifecycle management, build automation, and plugins for various tasks. Benefits include standardized project structure, dependency resolution, continuous integration support, and ease of project management with centralized configuration (pom.xml).

12. Explain the typical structure of a Maven project. What are the key files and directories in a Maven project?

Maven project structure:

1. src/main/java: Java source files.

2. src/main/resources: Non-Java resources.

3. src/test/java: Test source files.

4. src/test/resources: Test resources.

5. pom.xml: Project Object Model (POM) configuration.

6. target: Compiled output and generated artifacts. Key files: pom.xml (configuration), source directories (java, resources), and target (output directory).

13. How does Maven handle dependencies? Describe the process of adding and managing dependencies in a Maven project.

Maven manages dependencies through pom.xml. Steps to add dependencies: 1. Add <dependency> tag with artifact coordinates (groupId, artifactId, version). 2. Maven downloads dependencies and their transitive dependencies from repositories. 3. Dependencies are managed centrally, ensuring version compatibility and reducing conflicts. Use mvn clean install to download dependencies and build project. Maven resolves dependencies based on <dependencyManagement> and <dependencies> sections in pom.xml.

14. What are Maven plugins? How do you configure and use plugins in a Maven project?

Maven plugins provide additional functionality like compiling code, running tests, packaging applications, etc. Configure plugins in pom.xml using <plugin> tags with <groupId>, <artifactId>, and <version>. Plugins can be bound to phases of Maven lifecycle (compile, test, package) or executed standalone (mvn plugin:goal). Use mvn clean install to execute plugins configured in pom.xml and achieve project-specific tasks like code generation or static analysis.

15. How do you integrate Maven with Eclipse IDE? Describe the steps to create and manage Maven projects in Eclipse.

Integrate Maven with Eclipse:

1. Install Maven plugin (m2eclipse) in Eclipse.

2. Create a new Maven project (File > New > Maven Project).

3. Select archetype (project template) and configure pom.xml.

4. Import existing Maven projects (File > Import > Existing Maven Projects).

5. Manage dependencies, build lifecycle, and run Maven goals (clean, install) from Eclipse using Run As and Maven options. Eclipse provides integration with Maven for seamless project management and development.

16. What is pom.xml in Maven? Describe its structure and the essential elements it contains.

pom.xml is the Project Object Model configuration file in Maven. Structure:

1. <project>: Root element.

2. <modelVersion>: Maven version.

3. <groupId>, <artifactId>, <version>: Project coordinates.

4. <dependencies>: Project dependencies.

5. <build>: Build configuration (plugins, resources, testResources).

6. <repositories>: Dependency repositories. pom.xml defines project metadata, dependencies, plugins, build configurations, and repository locations for Maven projects.

17. Explain the Maven build lifecycle phases and their sequence. How can you customize build lifecycle in Maven? Maven build lifecycle consists of phases (validate, compile, test, package, install, deploy) executed sequentially. Customize lifecycle using plugins (<plugin>), goals (<execution>), and lifecycle bindings (<phase>). Define custom phases and execute specific goals (mvn phase) or skip phases (-DskipTests). Maven plugins and configurations in pom.xml control build process, dependencies resolution, testing, packaging, and deployment lifecycle phases.

18. How do you integrate JDBC drivers with Maven? Provide an example configuration in pom.xml for MySQL and Oracle JDBC drivers.

Integrate JDBC drivers in Maven pom.xml:

1. Add <dependency> for MySQL (mysql-connector-java) or Oracle (ojdbc) JDBC driver.

2. Specify <groupId>, <artifactId>, and <version> for driver dependency.

Example configurations:

<dependency> for MySQL:

<groupId>mysql</groupId>,

<artifactId>mysql-connector-java</artifactId>,

<version>8.0.26</version>.

<dependency>

for Oracle:

<groupId>com.oracle.database.jdbc</groupId>,

<artifactId>ojdbc8</artifactId>,

<version>19.8.0.0</version>.

19. Describe the JDBC architecture with a detailed explanation of each component.

JDBC architecture:

1. JDBC API: Interfaces and classes for database access (Connection, Statement, ResultSet).

2. JDBC Driver Manager: Manages JDBC drivers and establishes connections (DriverManager).

3. JDBC Drivers: Implementations translating JDBC calls to database-specific protocols (JDBC-ODBC, Native-API, Network Protocol, Thin). 4. Database: Target data repository (Oracle, MySQL). JDBC API and drivers facilitate database connection, SQL execution, and result processing in Java applications.

20. How can Maven be used for deployment? Discuss strategies for deploying Maven-built artifacts to servers or repositories.

Deploy Maven artifacts:

1. Build artifact (mvn clean install) generates target files (JAR, WAR).

2. Deploy to local repository (~/.m2/repository) using mvn deploy.

3. Publish to remote repository (Nexus, Artifactory) for shared access. Deployment strategies: Automated CI/CD pipelines integrate Maven (Jenkins, GitLab CI) for continuous integration, testing, and deployment. Maven plugins (maven-deploy-plugin) configure artifact publishing and version management for centralized artifact storage and distribution.

21. What is RowSet interface and explain its types?

The RowSet interface in Java is part of the JDBC (Java Database Connectivity) API, which provides a higher-level abstraction over the traditional ResultSet interface. It was introduced to simplify the process of working with database data in Java applications.The RowSet interface in Java is part of the JDBC (Java Database Connectivity) API, which provides a higher-level abstraction over the traditional ResultSet interface. It was introduced to simplify the process of working with database data in Java applications.

Common Characteristics of RowSet Interfaces:

Disconnected Operation: Except for JdbcRowSet, all other types are disconnected from the database after fetching data, which improves performance and reduces resource usage.

Serializable: All types can be serialized and used in distributed applications or stored in session state.

Cursor Movement: They all support cursor movement similar to ResultSet, allowing navigation through rows.

Update Capabilities: With the exception of JdbcRowSet, all others can be updated while disconnected and later synchronized with the database.

Types of RowSet Interfaces:

1. JdbcRowSet:

Description: This is a connected rowset, meaning it maintains a connection to the database throughout its lifecycle.

2. CachedRowSet:

Description: This rowset is disconnected, meaning it fetches data from the database and then closes the connection, allowing it to operate independently of the database.

3. WebRowSet:

Description: Extends CachedRowSet and adds additional features for use in web applications.

4. FilteredRowSet:

Description: Implements the FilteredRowSet interface, allowing rows to be filtered programmatically based on specific criteria.

Unit – 2 (Servlet Programming)

MCQ Questions

1. What is the first method called when a servlet is initialized?

   A. service()
   B. init()
   C. doGet()
   D. doPost()

2. Which method is called by the servlet container to handle a client request?

   A. init()
   B. service()
   C. doGet()
   D. destroy()

3. Which method is called when a servlet is taken out of service?

   A. init()
   B. service()
   C. destroy()
   D. doGet()

4. Which method is called when a servlet is taken out of service?

   A. init()
   B. service()
   C. destroy()
   D. doGet()

5. What is the purpose of the doGet() and doPost() methods in a servlet?

   A. To initialize the servlet
   B. To destroy the servlet
   C. To handle HTTP GET and POST requests, respectively
   D. To configure the servlet

6. When is the init() method of a servlet called?

   A. Every time a request is made to the servlet
   B. When the servlet is first loaded into memory
   C. When the servlet is destroyed
   D. When the servlet is reloaded

7. Which class must a Java class extend to become an HTTP Servlet?

   A. java.servlet.HttpServlet
   B. javax.servlet.HttpServlet
   C. javax.servlet.GenericServlet
   D. java.servlet.GenericServlet

8. Which of the following methods is used to handle HTTP POST requests in an HTTP Servlet?

   A. doGet()
   B. doPost()
   C. service()
   D. init()

9. Which of the following methods in an HTTP Servlet handles HTTP DELETE requests?

   A. doDelete()
   B. doPost()

C. doGet()
D. doRemove()

10. How do you set the content type of the response to "text/html" in an HTTP Servlet?

A. response.setHeader("Content-Type", "text/html");
B. response.setContentType("text/html");
C. response.setContent("text/html");
D. response.setType("text/html");

11. What method is used to retrieve a parameter sent in an HTTP request in a servlet?

A. request.getParameter(String name)
B. request.getAttribute(String name)
C. request.getProperty(String name)
D. request.getValue(String name)

12. Which tag is used to map a URL pattern to a servlet in the web.xml file?

A. <url-mapping>
B. <servlet-mapping>
C. <mapping>
D. <url-pattern>

13. Where do you specify initialization parameters for a servlet in the web.xml file?

A. Inside the <servlet> tag using <param>
B. Inside the <servlet-mapping> tag using <param>
C. Inside the <servlet> tag using <init-param>
D. Inside the <servlet-mapping> tag using <init-param>

14. How do you retrieve an initialization parameter in a servlet?

A. config.getInitParameter("paramName")
B. context.getInitParameter("paramName")
C. request.getParameter("paramName")
D. response.getInitParameter("paramName")

15. What is the purpose of the <load-on-startup> element in the web.xml file?

A. To specify the order in which servlets should be loaded
B. To delay the loading of servlets until they are first requested
C. To load the servlet immediately after the server starts
D. To configure the servlet's URL pattern

16. Which of the following methods is used to track sessions using URL rewriting?

A. response.encodeURL(String url)
B. response.rewriteURL(String url)
C. response.trackURL(String url)
D. response.modifyURL(String url)

17. What is the default timeout period for an HTTP session in minutes, if not configured?

A. 15 minutes
B. 30 minutes
C. 60 minutes
D. 120 minutes

18. Which method is used to invalidate a session?

A. session.destroy()
B. session.terminate()
C. session.invalidate()
D. session.end()

19. How can you set an attribute in an HTTP session?

A. session.putAttribute(String name, Object value)
B. session.addAttribute(String name, Object value)
C. session.setAttribute(String name, Object value)
D. session.storeAttribute(String name, Object value)

20. Which of the following is NOT a method for session tracking in servlets?

A. Cookies
B. URL Rewriting
C. Hidden Form Fields
D. Page Refreshing

Unit – 2 (Servlet Programming)

Fill in the blanks

1) The **Init()** method is called by the servlet container to initialize a servlet.
2) The **service()** method is called by the servlet container to handle a client request.
3) The **destroy()** method is called by the servlet container to remove a servlet from service.
4) The **HttpServlet** class provides methods to handle HTTP-specific services.
5) The **doGet()** method of HttpServlet is used to handle HTTP GET requests.
6) The **doPost()** method of HttpServlet is used to handle HTTP POST requests.
7) A **ServletContext()** object provides information about the servlet's environment.
8) The **<servlet>** element in web.xml is used to configure a servlet.
9) A **ServletConfig** object contains initialization parameters for a servlet.
10) A **RequestDispather** object can be used to forward a request from one servlet to another.
11) The **getParameter()** method of HttpServletRequest retrieves the value of a request parameter.
12) A **HttpSession** is used to track sessions in servlets.
13) The **setAttribute()** method of HttpSession stores an attribute in the session.
14) The **removeAttribute()** method of HttpSession removes an attribute from the session.
15) The **sendRedirect()** method of HttpServletResponse sends a redirect response to the client.
16) In a CRUD operation, the **POST()** method is typically used to create a new resource.
17) In a CRUD operation, the **GET()** method is typically used to read a resource.
18) In a CRUD operation, the **PUT()** method is typically used to update a resource.
19) In a CRUD operation, the **DELETE()** method is typically used to delete a resource.
20) The **getCookies()** method of HttpServletRequest returns an array of cookies sent by the client.

Unit – 2 (Servlet Programming)

Short Questions

1. What is HTTP and what are its common methods used in web development?

HTTP (Hypertext Transfer Protocol) is the protocol used for transmitting web pages over the internet. Common HTTP methods include:

- GET: Requests data from a server (e.g., fetching a webpage).

- POST: Submits data to be processed to a server (e.g., form submission).

- PUT: Updates existing resources on the server.

- DELETE: Deletes resources from the server.

- HEAD: Similar to GET, but it retrieves only the headers, not the body of the response.

- OPTIONS: Describes the communication options for the target resource.

2. What is a web server, and how does it work?

A web server is software that serves web pages to clients over the HTTP protocol. It processes incoming requests from clients, retrieves the requested resources (such as HTML pages, images, or data), and sends them back as HTTP responses. Common web servers include Apache HTTP Server, Nginx, and Microsoft IIS.

3. Explain the difference between a web server and an application server.

A web server handles HTTP requests and serves static content (HTML, CSS, images). An application server, on the other hand, provides business logic to application programs through various protocols, including HTTP. Application servers can serve dynamic content, manage transactions, and provide middleware services. Examples include Apache Tomcat (which can also serve as a web server) and JBoss.

4. Describe the servlet lifecycle in detail.

The servlet lifecycle includes the following stages:

1. Loading and Instantiation: The servlet container loads the servlet class and creates an instance.

2. Initialization (`init()` method): Called once when the servlet is first loaded. It is used for one-time setup like resource allocation.

3. Request Handling (`service()` method): Called for each client request. It determines the request type (GET, POST, etc.) and dispatches it to the appropriate method (`doGet()`, `doPost()`, etc.).

4. Destruction (`destroy()` method): Called once when the servlet is being taken out of service. It is used for cleanup activities like releasing resources.

5. What is the purpose of the `init()` method in a servlet?

The `init()` method is called by the servlet container to initialize the servlet. It is executed only once when the servlet is first loaded into memory. This method is used for one-time initialization tasks such as reading configuration parameters, setting up database connections, or preparing resources needed by the servlet during its lifecycle.

6. What is the difference between `init()` and `service()` methods in a servlet?

A: The `init()` method is called once during the servlet's lifecycle, when the servlet is first loaded and initialized by the servlet container. It is used for one-time setup and initialization tasks, such as loading configuration parameters or establishing database connections.

The `service()` method is called for each client request to the servlet. It determines the type of request (GET, POST, etc.) and dispatches it to the appropriate method (`doGet()`, `doPost()`, etc.) for handling.

Servlets API

7. What are the main classes and interfaces in the Servlet API, and what are their purposes?

Key classes and interfaces in the Servlet API include:

- `Servlet` interface: Defines the basic methods a servlet must implement (`init`, `service`, `destroy`, `getServletConfig`, `getServletInfo`).

- `GenericServlet` class: An abstract class that implements the `Servlet` interface and is protocol-independent.

- `HttpServlet` class: Extends `GenericServlet` to provide methods specific to handling HTTP requests (`doGet`, `doPost`, etc.).

- `ServletConfig` interface: Used to pass configuration information to a servlet.

- `ServletContext` interface: Provides methods for interacting with the servlet container, sharing information between servlets, and accessing resources.

- `HttpServletRequest` and `HttpServletResponse` interfaces: Provide methods to access request data and construct responses, respectively.

8. Explain the `HttpServlet` class and its role in servlet development.

The `HttpServlet` class is an abstract class provided by the Servlet API that extends `GenericServlet` and implements the `Servlet` interface. It provides methods to handle HTTP-specific services. Subclasses of `HttpServlet` override methods like `doGet()`, `doPost()`, `doPut()`, and `doDelete()` to handle HTTP GET, POST, PUT, and DELETE requests, respectively. This class simplifies the development of HTTP-based web applications by providing a framework for handling common HTTP request types.

HTTP Servlets

9. What is the `doGet()` method in an `HttpServlet`, and when should it be used?

The `doGet()` method handles HTTP GET requests. It is used when the client requests data from the server, typically to retrieve a web page or query parameters. GET requests should be idempotent and safe, meaning they do not modify the server state and can be repeated without side effects. protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { response.setContentType("text/html"); PrintWriter out = response.getWriter(); out.println("<html><body>"); out.println("<h1>Hello, World!</h1>"); out.println("</body></html>"); }

10. Describe the `doPost()` method in an `HttpServlet` and its use cases.

The `doPost()` method handles HTTP POST requests, which are used to send data to the server, typically from a form submission. POST requests can change the server state and are not idempotent. They are used for operations like creating or updating resources, processing forms, and uploading files. protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { String username = request.getParameter("username"); String password = request.getParameter("password"); // Process the data response.setContentType("text/html"); PrintWriter out = response.getWriter(); out.println("<html><body>"); out.println("<h1>Welcome, " + username + "!</h1>"); out.println("</body></html>"); }

11. How can servlets be configured using the `web.xml` file? Provide an example.

Servlets can be configured using the `web.xml` deployment descriptor file by defining servlets and their mappings. Here is an example: <web-app> <servlet> <servlet-name>ExampleServlet</servlet-name> <servlet-class>com.example.ExampleServlet</servlet-class> </servlet>

<servlet-mapping> <servlet-name>ExampleServlet</servlet-name> <url-pattern>/example</url-pattern> </servlet-mapping> </web-app>

This configuration maps the `ExampleServlet` class to the URL pattern `/example`.

12. What are servlet initialization parameters, and how are they defined in `web.xml`?

A: Servlet initialization parameters are configuration parameters passed to a servlet during initialization. They are defined in the `web.xml` file within the `<init-param>` tags. Here is an example: <servlet> <servlet-name>ExampleServlet</servlet-name> <servlet-class>com.example.ExampleServlet</servlet-class> <init-param> <param-name>param1</param-name> <param-value>value1</param-value> </init-param> </servlet>

The servlet can retrieve these parameters using the `getInitParameter` method: public void init(ServletConfig config) throws ServletException { super.init(config); String param1 = config.getInitParameter("param1"); }

Servlets Context

13. What is the `ServletContext` interface, and how is it used? Provide examples.

A: The `ServletContext` interface provides methods for servlets to interact with the web application environment. It allows servlets to:

- Retrieve application-wide parameters: String paramValue = getServletContext().getInitParameter("globalParam");

- Set and get attributes for sharing data between servlets: getServletContext().setAttribute("sharedData", data); Object data = getServletContext().getAttribute("sharedData");

- Access resources like files or request dispatchers: InputStream input = getServletContext().getResourceAsStream("/WEB-INF/config.properties"); RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/anotherServlet"); dispatcher.forward(request, response);

14. How can servlets log messages using `ServletContext`?

A: Servlets can log messages using the `log()` method of the `ServletContext` interface. This method writes log messages to the servlet container's log file, which helps in debugging and monitoring. Here are examples: getServletContext().log("This is an informational message."); getServletContext().log("This is an error message.", new Exception("Sample Exception"));

The first method logs a simple message, while the second logs a message with an exception.

15. What is servlet collaboration, and how is it achieved?

A: Servlet collaboration refers to the mechanism by which servlets communicate and share data with each other. It can be achieved using:

- Request Dispatcher: Forwards a request from one servlet to another resource within the same application. RequestDispatcher dispatcher = request.getRequestDispatcher("/anotherServlet"); dispatcher.forward(request, response);

- Shared Attributes: Stores and retrieves shared data using request, session, or context attributes.

request.setAttribute("key", "value"); Object value = request.getAttribute("key");

- Redirects: Sends a client-side redirect to another servlet or resource.

response.sendRedirect("anotherServlet");

Servlets Collaboration (continued)

16. Explain the `RequestDispatcher` interface and its methods.

RequestDispatcher dispatcher = request.getRequestDispatcher("/header.jsp");

dispatcher.include(request, response);

This method includes the content of `header.jsp` in the current response.

17. What is session tracking, and why is it important in web applications?

Session tracking is a mechanism used to maintain state and data for a user across multiple requests in a web application. It is important because HTTP is a stateless protocol, meaning it does not retain any information about previous requests. Session tracking allows web applications to recognize users, store user preferences, manage login information, and maintain the state of a user's interaction with the application.

18. Explain the different techniques used for session tracking.

Techniques for session tracking include:

- Cookies:

Small pieces of data stored on the client-side.

They are sent with every request to the server.

Cookie userCookie = new Cookie("username", "john");

userCookie.setMaxAge(60*60*24); // 1 day response.addCookie(userCookie);

- URL Rewriting:

Appends session information to the URL.

String url = response.encodeURL("nextPage.jsp");

- Hidden Form Fields:

Includes session data in hidden fields of HTML forms. <input type="hidden" name="sessionID" value="12345">

- HttpSession API: Provides a more robust and easier way to manage sessions. HttpSession session = request.getSession(); session.setAttribute("username", "john");

19. Describe how the `HttpSession` interface is used to manage sessions in servlets.

The `HttpSession` interface provides methods to create and manage sessions. It allows servlets to store and retrieve user-specific data across multiple requests.

- Creating/Retrieving a Session: HttpSession session = request.getSession(); // true by default

- Storing Data in a Session: session.setAttribute("username", "john");

- Retrieving Data from a Session: String username = (String) session.getAttribute("username");

- Invalidating a Session:

session.invalidate();

20. How can you handle exceptions in servlets?

A: Exceptions in servlets can be handled using:

- `try-catch` blocks: Surrounding code that might throw exceptions with `try-catch` blocks.

- `error-page` configuration in `web.xml`: Configuring custom error pages for specific exceptions or status codes.

<error-page> <error-code>404</error-code> <location>/error404.html</location> </error-page> <error-page> <exception-type>java.lang.Throwable</exception-type> <location>/error.jsp</location> </error-page>

- Logging: Using logging mechanisms to log the details of exceptions.

- `HttpServletResponse.sendError`: Sending an HTTP error response.

response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, "An internal error

Unit – 2 (Servlet Programming)

Long Questions

1. What is Servlet ? Explain advantages of servlet ?

- Servlets is a java-based web technology from Sun Microsystems. Servlet technology is J2EE technology.
- A servlet is a user-defined public java Class that implements javax.servlet.Servlet interface and it is managed by the servlet container (is also called Servlet Engine).
- Properties of servlet are:
    - A servlet is a Dynamic Web Resource that enhances the functionality of the webserver.
    - A servlet is a Web Server-Side Piece of Code (server-side program).
    - A servlet is a Web Component.
- Servlets use the classes within the java packages javax.servlet and javax.servlet.http.
- For each client request servlet container will generate a separate thread on the respective servlet object. If we increase the number of requests even containers will create a number of threads instead of processes so it increase the performance of the server-side application.

Advantages of Servlet.

1. Portability: Servlets are highly portable across operating systems and server implementations because servers are written in java and follow well known standardized APIs so they are.
2. Powerful: It is possible to do several things with the servlets which were difficult or even impossible to do with CGI.
3. Efficiency: As compared to CGI the invocation of the servlet is highly efficient. When the servlet gets loaded in the server, it remains in the server's memory as a single object instance.
4. Safety: As servlets are written in java, servlets inherit the strong type safety of java language. Servlets are generally safe from memory management problems because of Java's automatic garbage collection and a lack of pointers.
5. Integration: Servlets are tightly integrated with the server. Servlet basically uses the server to translate the file paths, perform logging, check authorization, and MIME type mapping, etc.
6. Extensibility: The servlet API is designed in such a way that it can be easily extensible. As it stands today, the servlet API supports HTTP Servlets, but later it can be extended for another type of servlets.

2. Explain the life cycle of servlet with an example.

The Java Servlet Life cycle includes three stages right from its start to the end until the Garbage Collector clears it. These three stages are described below.

1. init()

2. service()

3. destroy()

1. init()

The init() is the germinating stage of any Java Servlet. When a URL specific to a particular servlet is triggered, the init() method is invoked.

Another scenario when the init() method gets invoked is when the servers are fired up. With every server starting up, the corresponding servlets also get started, and so does the init() method.

One important specialty of the init() method is the init() method only gets invoked once in the entire life cycle of the Servlet, and the init() method will not respond to any of the user's commands.

The init() method Syntax:

```
public void init() throws ServletException {

//init() method initializing

}
```

2. service()

The service() method is the heart of the life cycle of a Java Servlet. Right after the Servlet's initialization, it encounters the service requests from the client end.

The client may request various services like:

• GET

• PUT

• UPDATE

• DELETE

The service() method takes responsibility to check the type of request received from the client and respond accordingly by generating a new thread or a set of threads per the requirement and implementing the operation through the following methods.

• doGet() for GET

• doPut() for PUT

• doUpdate() for UPDATE

• doDelete() for DELETE

The service() method Syntax:

```
public void service(ServletRequest request, ServletResponse response)  throws ServletException, IOException {

}
```

3. destroy()

Like the init() method, the destroy() method is also called only once in the Java Servlet's entire life cycle. When the destroy() method is called, the Servlet performs the cleanup activities like, Halting the current or background threads, Making a recovery list of any related data like cookies to Disk.

After that, the Servlet is badged, ready for the Garbage collector to have it cleared.

The destroy() method Syntax:

```
public void destroy() {
   //destroy() method finalizing
}
```

3. What is a deployment descriptor? Give an example to map the servlet

• A web application's deployment descriptor maps the http request with the servlets.

• When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.

• The deployment descriptor should be named as web.xml. It resides in the application's WAR file under the WEB-INF/ directory.

• Root element of web.xml should be <web-app>. <servlet> element map a URL to a servlet using <servlet-mapping> element. To map a URL to a servlet, you declare the servlet with the <servlet> element, then define a mapping from a URL path to a servlet declaration with the <servlet-mapping> element.

• The <servlet> element declares the servlet class and a logical name used to refer to the servlet by other elements in the file.

• You can declare multiple servlets using the same class but name for each servlet must be unique across the deployment descriptor. The <servlet-mapping> element specifies a URL pattern and the name of a declared servlet to

use for requests whose URL matches the pattern. The URL pattern can use an asterisk (*) at the beginning or end of the pattern to indicate zero or more of any character.

• The standard does not support wildcards in the middle of a string, and does not allow multiple wildcards in one pattern. The pattern matches the full path of the URL, starting with and including the forward slash (/) following the domain name. The URL path cannot start with a period (.).

Structure of deployment descriptor file – web.xml

```
<servlet>
<servlet-name>Practice1</servlet-name>
    <servlet-class>
        com.parul.controller.servlet
    </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Practice1</servlet-name>
    <url-pattern>/ </url-pattern>
</servlet-mapping>
```

4. What is the difference between GET and POST method?

1. GET is a safe method (idempotent) where POST is non-idempotent method.
2. We can send limited data with GET method and it's sent in the header request URL whereas we can send large amount of data with POST because it's part of the body.
3. GET method is not secure because data is exposed in the URL and we can easily bookmark it and send similar request again, POST is secure because data is sent in request body and we can't bookmark it.
4. GET is the default HTTP method whereas we need to specify method as POST to send request with POST method.
5. Hyperlinks in a page uses GET method.

5. What is HttpServlet and how it is different from GenericServlet?

GenericServlet

GenericServlet is an abstract class that provides a generic, protocol-independent servlet. It is intended to handle any type of protocol such as HTTP, FTP, SMTP, etc., although it is commonly used with HTTP. Here are some key points about GenericServlet:

Protocol Independence: It is designed to be protocol-independent, meaning it can handle any type of protocol. This is achieved by not assuming any specific details about the request or response.

Methods: The key methods provided by GenericServlet are:

init(ServletConfig config): Initializes the servlet with configuration data.

service(ServletRequest req, ServletResponse res): Handles the client request.

destroy(): Cleans up resources before the servlet instance is removed.

Implementation Requirements: Subclasses of GenericServlet must override the service method to provide specific handling for the protocol they intend to support.

HttpServlet

HttpServlet is an abstract subclass of GenericServlet that specifically handles HTTP requests and responses. It extends the capabilities of GenericServlet to provide functionality tailored for web applications that communicate over HTTP. Here's how HttpServlet differs:

HTTP-Specific Functionality: HttpServlet is designed to handle HTTP-specific details such as HTTP methods (GET, POST, PUT, DELETE, etc.), HTTP headers, cookies, session management, etc.

Methods: In addition to the methods inherited from GenericServlet, HttpServlet adds:

doGet(HttpServletRequest req, HttpServletResponse res): Handles GET requests.

doPost(HttpServletRequest req, HttpServletResponse res): Handles POST requests.

doPut(HttpServletRequest req, HttpServletResponse res): Handles PUT requests.

doDelete(HttpServletRequest req, HttpServletResponse res): Handles DELETE requests.

doHead(HttpServletRequest req, HttpServletResponse res): Handles HEAD requests.

doOptions(HttpServletRequest req, HttpServletResponse res): Handles OPTIONS requests.

doTrace(HttpServletRequest req, HttpServletResponse res): Handles TRACE requests.

These methods are specific to HTTP and provide a way to process requests based on the HTTP method used.

Usage: HttpServlet is typically used for building web applications that handle HTTP requests and generate HTTP responses. It encapsulates common web application logic, such as handling form submissions, managing user sessions, and interacting with HTTP-specific headers and parameters.

Key Differences

Protocol Support: GenericServlet is protocol-independent, whereas HttpServlet is specifically tailored for HTTP.

Methods: HttpServlet adds HTTP-specific methods (doGet, doPost, etc.) on top of the methods provided by GenericServlet.

Purpose: GenericServlet is more generic and can be used for any protocol, whereas HttpServlet is specialized for handling HTTP requests and responses in web applications.

In summary, HttpServlet extends GenericServlet to provide specialized support for handling HTTP requests and responses, making it the preferred choice for developing servlets in web applications that operate over the HTTP protocol.

6. What are the advantages of Servlet over CGI?

Servlets provide better performance in terms of processing requests, better memory usage by using the advantage of multithreading (a new thread is created for each request, which is faster than allocating memory for a new object for each request, as in CGI).

Servlets, platform and system are independent. Thus, a web application written using servlets can be run in any servlet container that implements the standard and in any operating system.

Servlet usage improves program reliability, because the servlet container itself takes care of the servlet life cycle (and therefore memory leaks), security, and garbage collection.

Servlets are relatively easy to learn and maintain, so the developer only needs to care about the business logic of the application, and not the internal implementation of web technologies.

7. What is the process to implement doGet and doPost methods?

doGet():

doGet() methods are the service methods that is included in the servlets. This allow the servlet to handle the GET request that is being processed by the client. This overrides the method to support the GET request that automatically supports the HTTP head request that is given in the no body in the response and included in the header fields. It is being implemented as:

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, java.io.IOException

It is used to read the request data and provide the response of it by putting the solution in the header. It uses the response's writer or the object stream object that provides the response to the client. This is safe to use and can be safely repeated in case of any other request. .

doPost():

doPost() is used to handle a POST request that is also given at the time of filling up the form or any other action that is related to the user submission. This method allows the client to send the data of any length to the web server and that is also at single time. To read the request data the response headers are included that takes the response writer class to write that uses the output stream object. It uses the function of PrintWriter object that returns the response to set the content type of accessing the object. It is given as:

doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, java.io.IOException"

What is a web application and what is it's directory structure in Servlet? "Web Applications are modules that requires to run on server to handle the request and return the response. Java provides web application support through Servlets and JSPs that can run in a servlet container and provide dynamic content to client browser.

Java Web Applications are packaged as Web Archive (WAR) and it has following folders in it.

WEB-INF : contains the lib directory(application dependencies), classes(contains java servlet files) and web.xml(Deployment Descriptor)

META-INF: contains MANIFEST.MF

webapp: contains all the static pages like .html , .js etc"

8. What is Servlet API? Explain the Servlet API packages

The Java Servlet API is a class library for Servlets. Servlets are Java programs that run on the server and send response to the client. Two packages are available: javax.servlet and javax.servlet.http.

The first one contains basic classes and interfaces that we can use to write Servlets from the scratch. The second package offers more advanced classes and interfaces that extend classes and interfaces from the first package. It is much more convenient to program using the second package.

The Servlet API packages are listed below.

Packages

javax.servlet

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a servlet class and the runtime environment provided for an instance of such a class by a conforming servlet container.

javax.servlet.http

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

9. What is the ServletConfig object? Explain with an example

• ServletConfig is an object, it will store all the configuration details of a specific servlet, where the configuration details include the logical name of the servlet, initialization parameters, reference of ServletContext object, and so on.

• ServletConfig is an object, it will provide the entire view of a specific servlet. In the web application, the container will prepare ServletConfig objects individually to every and each servlet.

• In web application execution, the container will prepare ServletCofig object immediately after servlet instantiation and just before calling init() method in servlet initialization.

• The container will destroy the ServletConfig object just before servlet de-instantiation.

• If we declare any data in the ServletConfig object then that data is going to be shared up to the respective servlet. Due to the above reasons, the scope of the ServletConfig object is up to a particular servlet.

```
<web-app>
 <servlet>
   <servlet-name>firstservlet</servlet-name>
   <servlet-class>com.teknobizz.controller</servlet-class>
   <init-param>
     <param-name> username </param-name>
     <param-value> jones@aol.com </param-value>
   </init-param>
   <init-param>
     <param-name> fullname </param-name>
     <param-value> Mark Jones  </param-value>
   </init-param>
 </servlet>
 <servlet-mapping>
   <servlet-name>firstservlet</servlet-name>
   <url-pattern>/</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
   <welcome-file>index.html</welcome-file>
 </welcome-file-list>
</web-app>
protected void doPost(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException {
        PrintWriter pw=res.getWriter();
      res.setContentType(""text/html"");

      ServletConfig conf = getServletConfig();
      String username = conf.getInitParameter(""username"");
      String fullname = conf.getInitParameter(""fullanme"");
```

```
        pw.println(""Username :" + username + "<br/>Fullname :" + fullname);
        pw.close();
}
```

10. What is Servlet Context ? Explain with an example

• It will manage all the context details of a particular web application, where the context details include the logical name of the web application and context parameters, and so on.

• It will provide a complete view of a particular web application. ServletContext object will be prepared by the container the moment when we start the server i.e. the time when we deploy the web application.

• It can be created and removed only by the server, not by the programmer.

• ServletContext object is created at the time of deploying the project.

• ServletContext object will be removed by the server when we un-deploy the project.

• ServletContext object will be destroyed by the container when we shut down the server i.e. the time when we un-deploy the web application. Due to the above reasons, the life of the ServletContext object is almost all the life of the respective web application.

• If we declare any data in the ServletContext object then that data will be shared with all the number of resources that are available in the present web application. Due to the above reason, ServletContext will provide more shareability.

• In web applications, the container will prepare ServletContext object irrespective of getting requests from the client.

• In the web application, ServletContext will allow both parameters and attributes data. Due to the above reason, ServletContext will allow both Static Inclusion and Dynamic Inclusion of data.

Web.xml

```
<web-app>
  <context-param>
    <param-name> num1 </param-name>
    <param-value> 100 </param-value>
  </context-param>
  <context-param>
    <param-name> num2  </param-name>
    <param-value> 200 </param-value>
  </context-param>
 <servlet>
   <servlet-name>ctxservlet</servlet-name>
   <servlet-class>com.teknobizz.controller</servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name> ctxservlet </servlet-name>
   <url-pattern>/</url-pattern>
 </servlet-mapping>
 <welcome-file-list>
```

```
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
protected void doPost(HttpServletRequest req,HttpServletResponse res)throws ServletException,IOException
   {
      PrintWriter pw=res.getWriter();
      res.setContentType(""text/html"");
      ServletContext context = getServletContext();
      String num1 = context.getInitParameter(""num1"");
      String num2 = context.getInitParameter(""num2"");
      pw.println(""num1 value is :"" + num1+ "" and num2 is :""  + num2);
     pw.close();
   }
```

11. What is the difference between PrintWriter and ServletOutputStream?

PrintWriter is a class for working with a character stream, and ServletOutputStream is a class for working as a byte stream. PrintWriter is used to write character-based information, such as an array of characters or a string in the response, while ServletOutputStream is used to write a byte array to the response.

To get an instance of ServletOutputStream, use the ServletResponse getOutputStream () method , and for PrintWriter , use the ServletResponse getWriter () method

12. What is Servlet container ? What tasks does a servlet container perform usually?

Servlet containers are part of a web server that offers network services. They depend on the MIME-based requests and responses. A servlet container handles servlets.

A servlet container performs the following tasks:

It facilitates communication between the servlets, JSPs and the web client. You don't have to build a server socket to receive requests, parse them and generate responses because of the container. The container takes care of these tasks, allowing you to focus on the business logic.

The servlet container handles the life cycle of servlets. It loads the servlets into memory, initialises them, invokes the necessary methods and destroys them. Servlet containers also simplify resource management by offering utilities such as JNDI.

Servlet containers create new threads for every request and give servlets request and response objects. This way, you don't have to initialise the servlets for every request, saving a lot of memory and time. "

13. Differences between ServletConfig and ServletContext

ServletConfig	ServletContext
Defined in javax.servlet package.	Defined in the same javax.servlet package.
Values are placed in web.xml file.	Values are placed in web.xml file.
getInitParameter() defined ServletConfig is used to read values.	Same method getInitParameter() is also present in ServletContext interface also to read the values
When the values of <param-value> are changed, the Servlet need not be compiled again and thereby maintenance of code is easier.	The same case also with <context-param>.
No limit on the existence of <init-param> tags in <servlet>.	the web.xml fiel can have any number of <context-param> tags.
Servlet is initialized with the initialization data provided in <init-param>. This data is specific for one Servlet only.	The data provided in <context-param> tag is meant for the usage of all Servlets under execution.
Data in included within <servlet> tag.	Data is included within <context-param> tag.
It is local data for a particular Servlet.	It is global data sharable by all servlets.
Each Servlet comes with a separate ServletConfig object.	There will be only one ServletContext object available accessed by all Servlets.

14. What is RequestDispatcher ? explain forward() and include method() with an example

The RequestDispatcher is an Interface that comes under package javax.servlet. Using this interface we get an object in servlet after receiving the request. Using the RequestDispatcher object we send a request to other resources which include (servlet, HTML file, or JSP file). A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

There are two methods defined in the 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.

The forward() method is used to transfer the client request to another resource (HTML file, servlet, jsp etc). When this method is called, the control is transferred to the next resource called. On the other hand, the include() method is used to include the content of the calling file into the called file. After calling this method, the control remains with the calling resource, but the processed output is included into the called resource.

Example

```
RequestDispatcher rd = request.getRequestDispatcher(""RequestDispacherEx1_1"");
        rd.forward(request, response);
        RequestDispatcher rd = request.getRequestDispatcher(""RequestDispacherEx2_1"");
        rd.include(request,response);"
```

15. What is the difference between sendRedirect() and Forward() in a Servlet?

The difference between sendRedirect() and Forward() can be explained as follows:

sendRedirect():

sendRedirect() method is declared in HttpServletResponse Interface.

The syntax for the function is as follows:

1

void sendRedirect(String URL)

This method redirects the client request for further processing, the new location is available on a different server or different context. The web container handles this and transfers the request using the browser, this request is visible in the browser in the form of a new request. It is also called a client-side redirect.

Forward():

Forward() method is declared in the RequestDispatcher Interface.

The syntax for the function is as follows:

1

forward(ServletRequest request, ServletResponse response)

This passes the request to another resource for processing within the same server, another resource could be any servlet, JSP page. Web container handles the Forward() method. When we call Forward() method, a request is sent to another resource without informing the client, about the resource that will handle the request. It will be mentioned on the requestDispatcher object which we can be got in two ways. Either using ServletContext or Request. "

17. What is session tracking ? Explain the various techniques with an example

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 particular user.There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

Cookie

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

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

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.

There are 2 types of cookies in servlets.

Non-persistent cookie

Persistent cookie

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.

Hidden Field

The Server embeds new hidden Fields in every dynamically generated From page for the client. when the client submits the form to the server the hidden fields identify the client.

Hidden field is an invisible text box of the form page, hidden field value goes to the server as a request parameter when the form is submitted.

It always work whether cookie is disabled or not.

URL rewriting

URL-Rewriting Session Tracking Mechanism is almost all same as HttpSession Tracking Mechanism, in URL-Rewriting Session Tracking Mechanism we will not depending on a Cookie to maintain Session-Id value, we will manage SessionId value as an appended to URL in the next generated form.

In URL-Rewriting Session Tracking Mechanism, every time we need to rewrite the URL with Session-Id value in the next generated form. So that this mechanism is called a URL-Rewriting Session Tracking Mechanism.

HTTP Session

HttpSession interface enables a servlet to read and write the state information that is associated with an HTTP session.

A session contains information, specific to a particular user across the whole application. When a user enters into a website for the first time.

Session Management facility creates a unique session ID and typically sends it back to the browser as a cookie or store in request parameter.

Each subsequent request from this user passes the cookie containing the session ID, and the Session Management facility uses this ID to find the user's existing HttpSession object.

HttpSession is obtained via request.getSession(). The default timeout value is 30 minutes.

19. What are the different methods involved in generic servlet?

Generic servlet is a base class of servlet. It involves the following methods:

init() method: This method is called once by the servlet container in its lifecycle . It takes a ServletConfig object that contains the initialization parameters and configuration of the servlet.

Service() method : This method is defined as a public void service as ServletRequest “req” OR ServletResponse “res” which gives Servlet Exception, as IOException .If once the servlet starts getting requests, the service() method is called by the servlet container to respond to the requests.

Getservlet config(): The method comprises of the initialization and the startup of the servlet. It is also responsible for returning the Server Config object.

Getservlet info(): This method is defined as public String getServletInfo() .It is responsible for returning a string which is in the form of plain text and not any kind of markup.

destroy(): This method is defined as public destroy().this method is called when want to close the servlet."

20. What is the difference between request attributes, session attributes, and ServletContext attributes?

A ServletContext attribute is an object bound into a context through ServletContext.setAttribute() method and which is available to ALL servlets (thus JSP) in that context, or to other contexts via the getContext() method. By definition a context attribute exists locally in the VM where they were defined. So, they’re unavailable on distributed applications.

Session attributes are bound to a session, as a mean to provide state to a set of related HTTP requests. Session attributes are available ONLY to those servlets which join the session. They’re also unavailable to different JVMs in distributed scenarios. Objects can be notified when they’re bound/unbound to the session implementing the HttpSessionBindingListener interface.

Request attributes are bound to a specific request object, and they last as far as the request is resolved or while it keep dispatched from servlet to servlet. They’re used more as comunication channel between Servlets via the RequestDispatcher Interface (since you can’t add Parameters…) and by the container. Request attributes are very useful

in web apps when you must provide setup information between information providers and the information presentation layer (a JSP) that is bound to a specific request and need not be available any longer, which usually happens with sessions without a rigorous control strategy.

Thus we can say that context attributes are meant for infra-structure such as shared connection pools, session attributes to contextual information such as user identification, and request attributes are meant to specific request info such as query results.

Unit – 3 (JSP Programming)

MCQ Questions

1. Which JSP scripting element is used to declare variables and methods?

<%! %>

<% %>

<%= %>

<%@ %>

2. Which JSP scripting element is used to embed Java expressions that are evaluated and inserted into the output

<%! %>

<% %>

<%= %>

<%@ %>

3. Which JSP scripting element is used to include Java code that is executed when the JSP is requested?

<%! %>

<% %>

<%= %>

<%@ %>

4. What is the purpose of the <%@ %> directive in JSP?

To declare variables and methods .

To embed Java expressions

To include Java code

To provide instructions to the JSP engine

5. Which JSP directive is used to include another file in a JSP at page translation time?

<%@ include file="filename" %>

<% include file="filename" %>

<%= include file="filename" %>

<%! include file="filename" %>

6. Which JSP directive is used to specify page-related attributes, like content type and error page?

<%@ taglib %>

<%@ include %>

<%@ page %>

<%@ directive %>

7. Which directive is used to define the scripting language in a JSP page?

<%@ include %>

<%@ taglib %>

<%@ page %>

<%@ language %>

8. Which directive is used to include a file during the translation phase of the JSP?

<%@ include file="filename" %>

<% include file="filename”%>

<%@ file include="filename" %>

<% file include="filename" %> 1

9. Which directive declares the content type of a JSP page?

<%@ page contentType="type" %>

<%@ page type="content" %>

<%@ include contentType="type" %>

<%@ taglib contentType="type" %>

10. How do you specify an error page in a JSP using directives?

<%@ page errorPage="error.jsp" %>

<%@ page error="error.jsp" %>

<%@ include errorPage="error.jsp" %>

<%@ taglib errorPage="error.jsp" %>

11. Which directive is used to import classes, interfaces, or packages in a JSP page?

<%@ import="java.util.*" %>

<%@ page import="java.util.*" %>

<%@ include import="java.util.*" %>

<%@ taglib import="java.util.*" %>

12. Which directive is used to specify that the current JSP page is an error page?

<%@ page errorPage="true" %>

<%@ page isErrorPage="true" %>

<%@ include isErrorPage="true" %>

<%@ taglib isErrorPage="true" %>

13. What is the correct syntax for the taglib directive in JSP?

<%@ taglib uri="uri" prefix="prefix" %>

<%@ taglib uri="prefix" prefix="uri" %>

<%@ taglib prefix="uri" uri="prefix" %>

<%@ taglib prefix="prefix" uri="uri" %>

14. Which page directive attribute is used to define a buffer size for the JSP page?

buffer

autoFlush

contentType

session

Unit – 3 (JSP Programming)

Fill in the Blanks

1. JSP stands for **Java Server Page**
2. Scripting elements in JSP allow embedding **Java Code** within HTML pages.
3. Directive elements in JSP provide instructions to the container about JSP **page compilation and execution**
4. CRUD operations typically involve **Creating, Reading, Updating, Deleting**
5. The <%@ page import="..." %> directive is used in JSP for **Importing Java classes**
6. In JSP, <%! ... %> denotes **declaration**
7. The <%= ... %> syntax in JSP is used for **Outputting data**
8. JSP pages are compiled into **Java bytecode**
9. The action element <jsp:useBean> in JSP is used for **Instantiating a JavaBean**
10. The <jsp:forward> action in JSP is used for **Redirecting to another resource**
11. The <jsp:include> action in JSP is used for **Including the output of another resource or file**
12. In JSP, the <%@ taglib %> directive is used for **Importing a tag library**
13. The request.getParameter("paramName") method in JSP is used to **Extract data from an HTTP request**
14. The method used to delete data from a database in JSP is **doDelete()**
15. JSP technology allows **Static HTML pages** to be generated dynamically.
16. The <%@ include file="..." %> directive in JSP is used for **Including the content of another file**
17. A JSP page is compiled into a(n) **Servlet**
18. The <c:forEach> tag in JSP is used for **Iterating over a collection**
19. The <form> element in HTML is used to **Collect user input** in JSP.
20. In JSP, the <jsp:scriptlet> tag is used for **Embedding Java code**

Unit – 3 (JSP Programming)

Short Questions

1. What is JSP?

JSP (JavaServer Pages) is a technology that allows developers to create dynamically generated web pages based on HTML, XML, or other document types.

2. **How is a JSP page processed?**

A JSP page is first translated into a servlet, compiled into bytecode, loaded and instantiated, and then executed by the servlet container.

3. What is the life cycle of a JSP page?

The life cycle includes translation, compilation, instantiation, initialization, execution, and destruction of the JSP page.

4. How do you create and execute a JSP page?

To create a JSP page, create a .jsp file containing HTML and JSP elements. Execute by deploying on a servlet container like Tomcat.

5. Discuss the directory structure of a JSP page.

Typical structure includes .jsp files in the webapp directory, with WEB-INF containing web.xml and other configuration files.

6. What is the JSP API?

The JSP API provides classes and interfaces to support the development of web applications based on JSP technology.

Discuss the packages of the JSP API. Packages include javax.servlet.jsp for JSP-specific classes and javax.servlet.jsp.tagext for tag library support.

7. What is a scripting element in JSP?

A scripting element in JSP allows embedding Java code directly into the JSP page.

8. Discuss the types of scripting elements in JSP.

Types are:

include declaration (<%! %>) for variable and method declarations,

scriptlet (<% %>) for executable Java code, and

expression (<%= %>) for outputting data.

9. Provide an example of a declaration in JSP.

jsp
<%! int count = 0; %>

10. What are implicit objects in JSP?

Implicit objects are predefined objects accessible without explicit declaration in JSP pages.

11. Discuss the types of implicit objects in JSP.

Types include request, response, session, application, out, config, pageContext, exception, page, pageScope, param, header, cookie.

12. Explain the usage of the request implicit object in JSP with an example.

The request object provides access to client request information. Example: String name = (String)request.getParameter("username");

13. Explain request and response methods in JSP with examples.

Request methods (getParameter(), getAttribute()) retrieve data sent by the client.

Response methods (setContentType(), getWriter()) manage server responses.

14. Discuss JSP configuration

(<jsp-config>), application (<jsp:useBean>), session (<jsp:session>), out (<jsp:out>), context (<jsp:context>), and exception handling (<%@ page errorPage="error.jsp" %>) in JSP. These elements configure JSP behavior, manage application data, session attributes, output content, application-wide context, and handle exceptions.

15. What are JSP action tags?

JSP action tags are XML-based tags that provide server-side processing logic in JSP pages.

16. Provide an example of a JSP action tag.

<jsp:include page="header.jsp" />

17. What is expression language (EL) in JSP?

Expression Language (EL) simplifies accessing data stored in JavaBeans, session, request, and application scopes directly in JSP pages.

18. Discuss the usage of expression language (EL) in a JSP page with an example.

Example: ${user.name} accesses the name property of the user bean stored in session or request scope.

19. Discuss core JSTL tags and custom tags in JSP.

JSTL tags (<c:forEach>, <c:if>, <c:set>) provide common programming tasks, while custom tags extend JSP functionality with custom behaviors.

20. Discuss core JSTL tags and custom tags in JSP.

JSTL tags (<c:forEach>, <c:if>, <c:set>) provide common programming tasks, while custom tags extend JSP functionality with custom behaviors.

Unit – 3 (JSP Programming)

Long Questions

1. What is JSP and why do we need it? Discuss the benefits of using JSP over traditional Java servlets.

- JSP stands for Java Server Pages.
- JSP is java server-side technology to create dynamic web pages.
- JSP is extension of Servlet technology to help developers create dynamic pages with HTML like syntax.
- We can create user views in servlet also but the code will become very ugly and error-prone. Also, most of the elements on a web page are static, so the JSP page is more suitable for web pages.
- We should avoid business logic in JSP pages and try to use it only for view purpose.
- JSP scripting elements can be used for writing Java code in JSP pages but it's best to avoid them and use JSP action elements, JSTL tags or custom tags to achieve the same functionalities.
- One more benefit of JSP is that most of the containers support hot deployment of JSP pages. Just make the required changes in the JSP page and replace the old page with the updated jsp page in the deployment directory and the container will load the new JSP page.
- We don't need to compile our project code or restart server whereas if we make a change in servlet code, we need to build the complete project again and deploy it. Although most of the containers now provide hot deployment support for applications still it's more work that JSP pages.

Some benefits of using JSP over traditional Java servlets are:

1. Simplified development: JSP allows embedding Java code within HTML, reducing the need for explicit servlet code.
2. Rapid prototyping: JSP facilitates the quick and easy creation of dynamic web pages with its simple syntax.
3. Enhanced maintainability: JSP separates presentation logic from business logic, making it easier to update and maintain the codebase.
4. Reusability: JSP fragments, tag files, and custom tags promote code reuse and modular design.
5. Improved productivity: JSP provides a large set of built-in tags and libraries, such as JSTL, for common web development tasks.
6. Platform independence: JSP runs on any server that supports Java, providing portability and flexibility.

2. Explain the advantages and disadvantages of using JSP over other web technologies like Servlets or JavaScript frameworks?

JSP (JavaServer Pages) is a technology that allows developers to embed Java code within HTML pages to dynamically generate content on the server side. When comparing JSP to other web technologies like Servlets or JavaScript frameworks, there are several advantages and disadvantages to consider:

Advantages

1. Simplicity: JSP allows developers to mix Java code directly within HTML, making it easier to create dynamic web pages compared to writing Java Servlets, which require more code for similar functionality.
2. Reusable Components: JSP supports custom tag libraries, which can be used to create reusable components and simplify complex page structures.
3. Integration with Java EE: JSP is a part of the Java EE stack, making it easy to integrate with other Java EE technologies like Enterprise JavaBeans (EJB) and Java Persistence API (JPA).
4. Separation of Concerns: JSP promotes a separation of concerns by encouraging the use of JavaBeans for business logic and JSP pages for presentation. This separation makes it easier to maintain and update web applications.

Disadvantages

1. Performance: JSP pages can have performance overhead due to the need to compile them into Servlets at runtime. Compiled Servlets tend to perform better than JSP pages.
2. Mixing Logic with Presentation: While JSP encourages separation of concerns, it is still possible to mix business logic with presentation, leading to code that is hard to maintain and test.
3. Limited Front-End Functionality: JSP is primarily a server-side technology and lacks the rich front-end capabilities provided by modern JavaScript frameworks like React or Angular.

4. Steep Learning Curve: For developers who are new to Java and web development, JSP may have a steeper learning curve compared to simpler templating languages or JavaScript frameworks.

3. What are the JSP lifecycle phases? JSP lifecycle phases are:

1. Translation - JSP container checks the JSP page code and parse it to generate the servlet source code. For example in Tomcat you will find generated servlet class files at TOMCAT/work/Catalina/localhost/WEBAPP/org/apache/jsp directory. If the JSP page name is home.jsp, usually the generated servlet class name is home_jsp and file name is home_jsp.java
2. Compilation - JSP container compiles the jsp class source code and produce class file in this phase.
3. Class Loading - Container loads the class into memory in this phase.
4. Instantiation - Container invokes the no-args constructor of generated class to load it into memory and instantiate it.
5. Initialization - Container invokes the init method of JSP class object and initializes the servlet config with init params configured in deployment descriptor. After this phase, JSP is ready to handle client requests. Usually from translation to initialization of JSP happens when first request for JSP comes but we can configure it to be loaded and initialized at the time of deployment like servlets using load-on-startup element.
6. Request Processing - This is the longest lifecycle of JSP page and JSP page processes the client requests. The processing is multi-threaded and similar to servlets and for every request a new thread is spawned and ServletRequest and ServletResponse object is created and JSP service method is invoked.
7. Destroy - This is the last phase of JSP lifecycle where JSP class is unloaded from memory. Usually it happens when application is undeployed or the server is shut down.

4. What are JSP lifecycle methods?

JSP lifecycle methods are:

1. jspInit(): This method is declared in JspPage and it's implemented by JSP container implementations. This method is called once in the JSP lifecycle to initialize it with config params configured in deployment descriptor. We can override this method using JSP declaration scripting element to initialize any resources that we want to use in JSP page.
2. _jspService(): This is the JSP method that gets invoked by JSP container for each client request by passing request and response object. Notice that method name starts with underscore to distinguish it from other lifecycle methods because we can't override this method. All the JSP code goes inside this method and it's overridden by default. We should not try to override it using JSP declaration scripting element. This method is defined in HttpJspPage interface.
3. jspDestroy(): This method is called by container when JSP is unloaded from memory such as shutting down application or container. This method is called only once in JSP lifecycle and we should override this method to release any resources created in JSP init method

5. Differentiate between JSP and servlet.

1. JSP (JavaServer Pages) is an extension of servlet technology that allows the embedding of Java code within HTML pages, making it easier to create dynamic content. Servlets, on the other hand, are Java classes that handle HTTP requests and generate responses dynamically.
2. JSP pages are primarily used for presentation logic, while servlets are used for both handling requests and generating dynamic content.
3. JSP pages are translated into servlets during runtime, whereas servlets are Java classes that are compiled before execution.
4. JSP pages are more suitable for web page development, while servlets provide more flexibility and control over the request-handling process.

6. What are JSP directives? Explain with an example.

List the types of directives. JSP directives are instructions that provide directives to the JSP container on how to handle the JSP page during translation and execution. There are three types of JSP directives:

1. Page Directive: Specifies page-specific attributes, such as error handling, language, content type, session management, and more.
2. Include Directive: Includes the contents of an external file during translation.
3. Taglib Directive: Declares custom tag libraries and their prefixes for use on the JSP page.

7. What is a JSP expression? How is it different from a scriptlet?

A JSP expression is a piece of Java code that is enclosed within <%=%> tags in a JSP page. It is used to dynamically output a value or expression directly into the generated HTML response. The expression is evaluated at runtime, and the result is converted to a string and inserted into the HTML output. In contrast, a scriptlet (<% … %>) allows you to embed arbitrary Java code within a JSP page. It can be used to perform more complex logic and computations, but its output is not directly written to the response. Instead, it can manipulate variables, call methods, or control the flow of the page.

8. Explain JSP scripting elements. ? Give an examples

JSP scripting elements allow you to embed Java code within a JSP page. There are three types of scripting elements:

1. Scriptlet: Enclosed within <% … %> tags, it allows you to write Java code that is executed when the page is processed.
2. Declaration: Enclosed within <%! … %> tags, it allows you to declare variables, methods, and other members that can be accessed throughout the JSP page.
3. Expression: Enclosed within <%= … %> tags, it allows you to embed Java expressions whose values are converted to strings and outputted directly to the response.

9. What is the purpose of the page directive in a JSP page?

This is one of the other most asked JSP Interview questions. The page directive in a JSP (JavaServer Pages) page serves a critical role in providing instructions to the JSP container on how to process and manage the current page. It essentially acts as a configuration element at the top of the JSP file, allowing developers to set specific attributes that impact the behaviour and characteristics of the JSP page.

One of the primary purposes of the page directive is to define the content type that the JSP page will generate when it is rendered in the browser. This is done using the contentType attribute, and it specifies whether the JSP page will produce HTML, XML, JSON, or some other type of content. Setting the correct content type is crucial for ensuring that the browser interprets the page's output correctly.

Additionally, the page directive allows developers to specify the scripting language to be used within the JSP file, with options like Java or JavaScript. It also enables error handling by allowing the specification of an error page to be displayed in case an unhandled exception occurs during the execution of the JSP.

10. Differentiate between forward and redirect in JSP.

1. Forward: Forwarding is a server-side operation where the control is transferred from one JSP page to another JSP page or servlet within the same request. The browser is unaware of the forward, and the URL in the address bar remains the same. It allows sharing of request attributes and maintains a single request/response cycle.
2. Redirect: Redirecting is a client-side operation where the control is transferred to a different URL or resource. The server sends an HTTP redirect response to the browser, which then sends a new request to the redirected URL. The browser’s address bar displays the redirected URL. Redirecting creates a new request/response cycle and does not share request attributes

11. What is the JSP implicit object? List some commonly used implicit objects.

JSP implicit objects are pre-defined objects provided by the JSP container that is available for use without explicitly declaring or instantiating them. Some commonly used implicit objects in JSP include:

- request: Represents the client’s request to the server.
- response: Represents the server’s response to the client.
- out: Used for writing output to the client.
- session: Represents the user’s session.
- application: Represents the entire web application.
- pageContext: Provides access to various objects and information about the JSP page.
- config: Represents the configuration of the JSP page.
- exception: Represents any exception thrown during JSP page execution.

12. What is the significance of the JSP page directive isThreadSafe attribute, and when would you use it?

The isThreadSafe attribute in a JSP page directive allows you to specify whether a JSP page should be thread-safe or not. When set to true, it indicates that the JSP container should make the JSP page thread-safe by synchronising access to it, ensuring that multiple threads cannot concurrently execute the page.

This attribute is uncommonly used because most JSP pages are inherently thread-safe, as they follow a model where each request is handled by a separate thread, and JSP instances are not shared among requests. However, in rare cases where you might want to disable this automatic thread-safety mechanism, you can set isThreadSafe to false. This can be useful if you have specific optimization requirements and are confident that your JSP page does not have any shared state or critical sections that need synchronization.

It is worth noting that explicitly setting isThreadSafe to false should be done cautiously, as it can lead to issues with concurrency if not managed properly. In most cases, leaving it to the default value of true is recommended for safety and predictability.

13. Which categories can be divided JSTL tags, give examples.

JSTL tags are divided into five categories according to their functionality:

1. Core Tags – Core tags provide opportunities for iteration, exception handling, url, forward and redirect response, etc.
2. Formatting and Localization Tags – provide options for formatting Numbers, Dates, and support for i18n localization and resource bundles.
3. SQL Tags – JSTL SQL Tags support for working with databases like MySQL, Oracle, etc.
4. XML Tags – used to work with XML documents. For example, for parsing XML, transforming XML data and executing XPath expressions.
5. JSTL Functions Tags – provides a set of functions that allow you to perform various operations with strings, etc. For example, by concatenation or splitting strings.

14. What do you know about jsp expression language (JSP Expression Language – EL)?

In most cases, we use JSP for viewing purposes, and all business logic is present in the servlet or model classes. After receiving the client's request, it is processed in the servlet, and then we add attributes to the request / session / context scope, which must be extracted in the JSP code. The request response , headers , cookies, and init parameters are also used to create response views .

We can use scriptlets in JSP expressions to get attributes and parameters in JSP with Java code and use it for views. The problem is that web designers usually do not know java code, which is why JSP 2.0 introduces an expression language (EL) through which we can get attributes and parameters using HTML tags.

The syntax expression language looks like $ {name} , and we can use EL implicit objects and expression language operators to extract attributes from different scopes and use them in the JSP page.

15. How do you handle JSP page redirection? Give an example of each JSP page redirection can be achieved in multiple ways:

1. Using the response.sendRedirect() method to redirect the client's browser to a different URL or JSP page.
2. Using the JSP <jsp:forward> action to forward the request to another resource or JSP page.
3. Utilizing JavaScript or HTML meta refresh tags for client-side redirection.
4. Handling redirection logic in Java servlets or controller classes based on business rules or user actions.

16. What is JSP standard tag library (JSTL)? How is it useful? What is the syntax for using JSTL core tags in JSP?

JSP Standard Tag Library (JSTL) is a collection of custom tags provided as part of the JavaServer Pages specification. It offers a set of tags for common tasks such as iteration, conditional execution, database access, and XML processing. JSTL simplifies JSP development by providing a standard way to perform these tasks, reducing the need for scriptlets and promoting a more modular and reusable code structure.

To use JSTL core tags, you need to include the JSTL core tag library in your JSP using the taglib directive: <%@ taglib prefix=""c"" uri=""http://java.sun.com/jsp/jstl/core"" %>. Then, you can use the tags with the c: prefix.

For example, <c:forEach> or <c:if>.

17. What is JSP session tracking? Discuss different methods.

JSP session tracking is a mechanism to maintain stateful interactions with users across multiple requests. Different methods of session tracking in JSP include:

Cookies: Storing a session ID in a browser cookie and associating it with user data on the server.

URL Rewriting: Appending the session ID to URLs as a query parameter.

Hidden Form Fields: Including the session ID as a hidden field in HTML forms.

HttpSession: Using the HttpSession object to store session-related data on the server.

SSL Session: Utilizing the SSL protocol to establish a secure session between the client and the server.

18. What is a JSP Custom Tag? How does it differ from JSP Standard Tag Libraries (JSTL)?

JSP Custom Tags are user-defined tags created by developers to encapsulate custom logic and functionality in JSP pages. They are implemented as Java classes and can be reused across multiple JSP pages.

JSTL (JavaServer Pages Standard Tag Library), on the other hand, provides a set of pre-defined tags for common tasks like iteration, conditional statements, and formatting. JSTL tags are not user-defined; they are part of the JSTL library.

In summary, JSP Custom Tags are created by developers for custom logic, while JSTL provides pre-built tags for common tasks. These are some of the most popular interview questions for jsp and servlet in Java.

19. What are the different scopes in JSP? Provide examples.

In JSP, there are four main scopes to store and access data:

1. Page scope: Data is accessible within the current JSP page only.
2. Request scope: Data is accessible across multiple pages within the same request.
3. Session scope: Data is accessible across multiple requests from the same client.
4. Application scope: Data is accessible by all clients and all requests throughout the application's lifetime.

Examples:

<jsp:useBean> with scope=" page" creates a page-scoped bean.

request.setAttribute(""name"", value) sets a request-scoped attribute.

session.setAttribute(""username"", value) sets a session-scoped attribute.

application.setAttribute(""counter"", value) sets an application-scoped attribute."

Explain the use of JSP actions for database access.

20. JSP actions provide a convenient way to interact with databases within JSP pages. Some commonly used JSP actions for database access are:

<jsp:useBean>: Instantiates or retrieves a JavaBean representing a database connection or a data access object (DAO).

<jsp:setProperty>: Sets properties of a JavaBean representing a database entity or a query parameter.

<jsp:getProperty>: Retrieves properties of a JavaBean representing a database entity or a query result.

<jsp:include>: Includes the content of a JSP page containing database-related logic or query execution.

<jsp:forward>: Forwards the request to a servlet or another JSP page for database processing.

Custom tag actions: Custom tags can be created to encapsulate complex database operations and improve code modularity and reusability.