Q1. What is the use of JDBC in Java?
JDBC (Java Database Connectivity) is an API (Application Programming Interface) in Java that provides a set of classes and methods to interact with relational databases. It allows Java applications to connect to and perform various operations on databases, such as querying, inserting, updating, and deleting data.

Q2. What are the steps involved in JDBC?
The steps involved in using JDBC to interact with a database are as follows:

1. Load the JDBC driver: Before establishing a connection to the database, the appropriate JDBC driver needs to be loaded using the Class.forName() method. The driver class name varies depending on the database vendor.
2. Establish a connection: Use the DriverManager.getConnection() method to establish a connection to the database. Provide the necessary connection details such as the database URL, username, and password.
3. Create a statement: Create a Statement object or a PreparedStatement object to execute SQL statements. The Statement interface provides methods to execute SQL queries or updates.
4. Execute SQL statements: Use the executeQuery() method to execute SQL queries that retrieve data from the database. Use the executeUpdate() method to execute SQL statements that update the database (e.g., INSERT, UPDATE, DELETE).
5. Process the result: If the SQL statement is a query, use the ResultSet object to retrieve the results of the query. Iterate over the result set to access the individual rows and columns of the retrieved data.
6. Close resources: Close the ResultSet, Statement, and Connection objects to release database resources and free up memory.

Q3. What are the types of statements in JDBC in Java?

In JDBC, there are three types of statements that can be used to execute SQL queries or updates:

7. Statement: The Statement interface allows you to execute static SQL statements that do not contain input parameters. It is created using the createStatement() method of the Connection object. Statements are suitable for executing simple SQL queries and updates.
8. PreparedStatement: The PreparedStatement interface extends the Statement interface and is used to execute parameterized SQL statements. It is created using the prepareStatement() method of the Connection object. Prepared statements are precompiled and can be reused with different parameter values, improving performance and preventing SQL injection attacks.
9. CallableStatement: The CallableStatement interface is used to execute stored procedures or functions in the database. It is created using the prepareCall() method of the Connection object. Callable statements allow the execution of parameterized SQL statements that return multiple result sets.

Q4. What is Servlet in Java?
A servlet in Java is a server-side component that dynamically generates web content and provides a way to handle client requests and send responses. It is a Java class that follows the Java Servlet API and is executed within a web container or servlet container.
Servlets are used to develop web applications and are an integral part of Java's server-side

technology stack. They provide a scalable and platform-independent way to create web applications that can handle HTTP requests, process user input, interact with databases, and generate dynamic content.

Q5. Explain the life cycle of a servlet.
The life cycle of a servlet consists of several phases:

Instantiation: When a servlet is first accessed, the servlet container creates an instance of the servlet class using its constructor. This happens only once during the lifetime of the servlet.

Initialization: After the servlet is instantiated, the servlet container calls the init() method of the servlet. This method is used to perform any necessary initialization tasks, such as setting up database connections, loading configuration parameters, or initializing resources. The init() method is called only once during the servlet's lifetime.

Request Handling: Once the servlet is initialized, it can handle incoming client requests. The servlet container calls the service() method of the servlet to process each request. The service() method determines the type of request (e.g., GET, POST) and dispatches it to the appropriate method (doGet(), doPost(), etc.) of the servlet based on the HTTP method.

Request Processing: The appropriate method (doGet(), doPost(), etc.) is invoked to handle the specific type of request. The servlet can access request parameters, headers, and input streams to process the request data. It can also generate a response, which is sent back to the client.

Destruction: When a servlet is being taken out of service (e.g., when the web application is stopped or reloaded), the servlet container calls the destroy() method of the servlet. This method allows the servlet to clean up any resources it has allocated, such as closing database connections or releasing memory. The destroy() method is called only once during the servlet's lifetime.
During the life cycle, a servlet can handle multiple requests, and the service() method is called for each request. The servlet container manages the life cycle of the servlet and ensures that the appropriate methods are called at the appropriate times.

Q6. Explain the difference between the RequestDispatcher.forward() and HttpServletResponse.sendRedirect() methods.

- RequestDispatcher.forward(): This method is used to forward the control from one servlet to another servlet, JSP page, or HTML page within the same web application. The forward() method transfers the control to the specified resource without the client being aware of it. The URL in the browser's address bar remains the same. The request and response objects are shared between the original servlet and the forwarded resource. It is typically used for server-side forwarding.

- HttpServletResponse.sendRedirect(): This method is used to redirect the client's browser to a different resource, which can be another servlet, JSP page, or HTML page. The sendRedirect() method sends an HTTP redirect response to the client, which causes the client to make a new request to the specified URL. The URL in the browser's address bar changes to the new URL. It is typically used for client-side redirection.
In summary, forward() is used for server-side forwarding within the same web application, while sendRedirect() is used for client-side redirection to a different URL.

Q7. What is the purpose of the doGet() and doPost() methods in a servlet?
The doGet() and doPost() methods are used to handle HTTP GET and POST requests, respectively, in a servlet.

- doGet(): This method is called by the servlet container to handle HTTP GET requests. It is used to process requests that retrieve data or perform read-only operations. In the doGet() method, the servlet can access request parameters through the HttpServletRequest object and generate a response using the HttpServletResponse object.
- doPost(): This method is called by the servlet container to handle HTTP POST requests. It is used to process requests that submit data or perform write operations. In the doPost() method, the servlet can retrieve request parameters, process the data, and generate a response using the HttpServletResponse object.

By overriding these methods in a servlet, you can define the logic to handle specific types of HTTP requests and generate appropriate responses.

Q8. Explain the JSP Model-View-Controller (MVC) architecture.
The JSP Model-View-Controller (MVC) architecture is a design pattern that separates the application logic into three interconnected components:

- Model: The model represents the data and business logic of the application. It encapsulates the data and provides methods to manipulate and access it. In the case of JSP, the model can be implemented using JavaBeans or other data access objects.
- View: The view is responsible for presenting the data to the user. It includes the user interface components, such as HTML, CSS, and JSP tags, that render the data. The view retrieves the necessary data from the model and formats it for display.
- Controller: The controller handles the user's input and coordinates the interaction between the model and the view. It receives the user's requests, processes them, and updates the model or selects the appropriate view. In JSP, the controller can be implemented using servlets or other request-handling mechanisms.

The MVC architecture promotes a separation of concerns, where each component has a specific role and can be developed independently. It improves code modularity, maintainability, and reusability. Changes to one component do not affect the others, allowing for easier modifications and updates to the application.

Q9. What are some of the advantages of Servlets?
Some advantages of using servlets in Java web development are:
Platform Independence: Servlets are written in Java and can run on any platform that supports the Java Virtual Machine (JVM). This makes servlets platform-independent and allows developers to write web applications that can be deployed on different operating systems.

Portability: Servlets can be easily deployed across different web servers or servlet containers. They provide a consistent way to develop web applications that can run on various servers without modifications.

Performance: Servlets are efficient and provide high performance. They are initialized once and can handle multiple requests concurrently. Servlet containers also provide built-in mechanisms for managing thread pooling and request handling, optimizing performance.

Scalability: Servlets can handle a large number of concurrent requests, making them suitable for scalable applications. With proper configuration and deployment, servlets can efficiently handle increased traffic and workload.

Extensibility: Servlets can be extended and customized using

Q10. What are the limitations of JSP?
Some limitations of JSP include:

Complexity: JSP can become complex and difficult to maintain as the application grows. Mixing business logic and presentation code in JSP files can lead to code clutter and reduced code modularity.

Steep Learning Curve: JSP requires knowledge of both Java and HTML, which can make it challenging for beginners to learn and understand. It may require additional effort and time to become proficient in JSP development.

Tight Coupling: JSP files are tightly coupled with the underlying Java code. Changes in the Java code may require modifications in the associated JSP files, leading to a high degree of coupling between the presentation and business logic.

Limited Separation of Concerns: JSP does not provide a clear separation of concerns between the presentation layer and the business logic. This can make it difficult to achieve a clean and modular design in larger applications.

Limited Reusability: JSP pages are specific to a particular web application and may not be easily reusable in other projects. Reusing JSP components across multiple applications may require extra effort or refactoring.

Performance Overhead: JSP pages need to be compiled into servlets before they can be executed, which introduces an overhead during the first request. Additionally, the dynamic nature of JSP pages can lead to slower performance compared to static HTML pages.