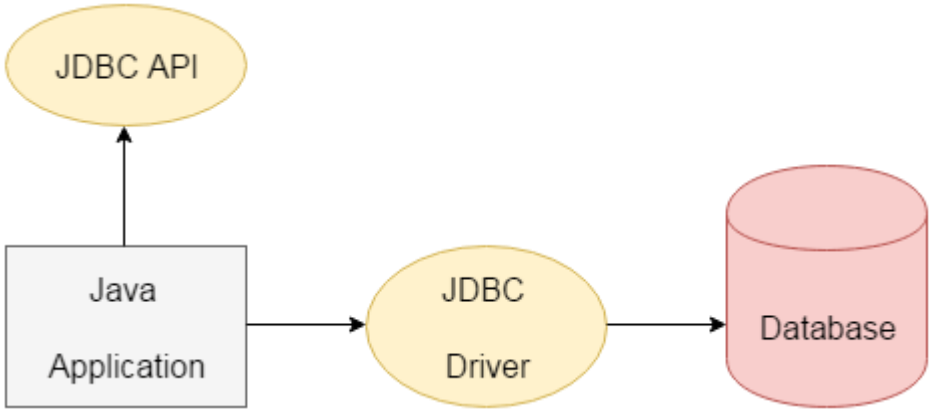
Q1. What is the use of JDBC in java?

Answer: JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

We have discussed the above four drivers in the next chapter.

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database.
2. Execute queries and update statements to the database.
3. Retrieve the result received from the database.

Q2. What are the steps involved in JDBC?

Answer:

Fundamental Steps in JDBC

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

1. Import JDBC packages.
2. Load and register the JDBC driver **// This step is not required in Java 6 and in JDBC 4.0**
3. Open a connection to the database.
4. Create a statement object to perform a query.
5. Execute the statement object and return a query resultset.
6. Process the resultset.
7. Close the resultset and statement objects. **// This step is not required because we use a try-with-resource statement to auto-close the resources**

8. Close the connection. // **This step is not required because we use a try-with-resource statement to auto-close the resources**

From the above steps, we actually require below five steps to connect a Java application to the database (example: MySQL):

1. Import JDBC packages.
2. Open a connection to the database.
3. Create a statement object to perform a query.
4. Execute the statement object and return a query resultset.
5. Process the resultset.

Q3. What are the types of statement in JDBC in java?

Answer: There are three types of statements in JDBC namely, Statement, Prepared Statement, Callable statement.

Statement

The Statement interface represents the static SQL statement. It helps you to create a general purpose SQL statements using Java.

Creating a statement

You can create an object of this interface using the **createStatement()** method of the **Connection** interface.

Create a statement by invoking the **createStatement()** method as shown below.

```
Statement stmt = null;
try {
  stmt = conn.createStatement( );
  . . .
}
catch (SQLException e) {
  . . .
}
finally {
  . . .
}
```

Executing the Statement object

Once you have created the statement object you can execute it using one of the execute methods namely, execute(), executeUpdate() and, executeQuery().

- **execute():** This method is used to execute SQL DDL statements, it returns a boolean value specifying whether the ResultSet object can be retrieved.
- **executeUpdate():** This method is used to execute statements such as insert, update, delete. It returns an integer value representing the number of rows affected.
- **executeQuery():** This method is used to execute statements that returns tabular data (example SELECT statement). It returns an object of the class ResultSet.

Prepared Statement

The **PreparedStatement** interface extends the Statement interface. It represents a precompiled SQL statement which can be executed multiple times. This accepts parameterized SQL quires and you can pass 0 or more parameters to this query.

Initially, this statement uses place holders "**?**" instead of parameters, later on, you can pass arguments to these dynamically using the **setXXX()** methods of the **PreparedStatement** interface.

Q4. What is Servlet in Java?

Answer: A Servlet in Java is a server-side component that runs on a web server and handles client requests and generates dynamic responses. It is a Java class that conforms to the Java Servlet API, which provides a standard way to extend the functionality of a web server and serve web applications.

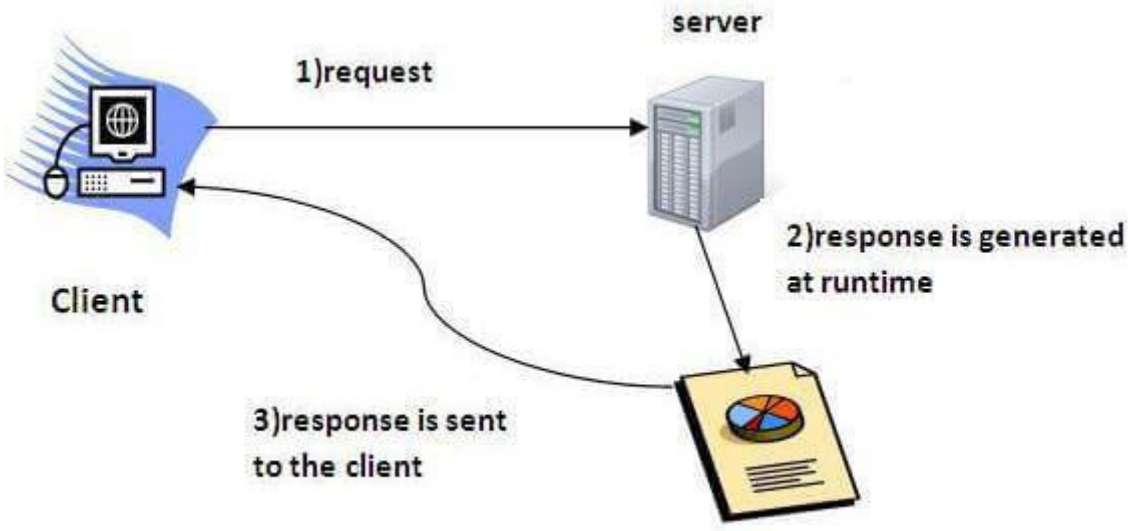
When a client sends an HTTP request to a web server, the server determines which servlet should handle the request based on the URL or other request parameters. The server then invokes the servlet's methods to process the request and generate a response, which is sent back to the client.

Servlets are often used in Java web applications to handle tasks such as gathering user input, interacting with databases, generating dynamic web content, and managing session state. They can handle different types of HTTP requests, such as GET and POST, and can respond with various types of data, including HTML, XML, JSON, or binary content.

Servlets provide a high-level abstraction for web development in Java, allowing developers to focus on application logic rather than low-level details of handling HTTP requests and responses. They can be used in conjunction with other Java technologies, such as JavaServer Pages (JSP), JavaServer Faces (JSF), or frameworks like Spring MVC, to build robust and scalable web applications.

Servlet can be described in many ways, depending on the context.

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



Q5. Explain the life Cycle of servlet?

Answer:A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

The servlet is initialized by calling the init() method.

The servlet calls service() method to process a client's request.

The servlet is terminated by calling the destroy() method.

Finally, servlet is garbage collected by the garbage collector of the JVM.

the life cycle methods in detail.

The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this −

```
public void init() throws ServletException {

   // Initialization code...

}
```

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method −

```
public void service(ServletRequest request, ServletResponse response)

   throws ServletException, IOException {

}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)

   throws ServletException, IOException {

   // Servlet code

}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)

   throws ServletException, IOException {

   // Servlet code

}
```

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this −

```
public void destroy() {

   // Finalization code...

}
```

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

Answer: The `RequestDispatcher.forward()` and `HttpServletResponse.sendRedirect()` methods are both used in Java servlets to redirect the flow of a request to another resource. However, they differ in how they handle the redirection process.

1. `RequestDispatcher.forward()`:

- This method allows the servlet to forward the current request to another resource, such as another servlet, JSP page, or HTML file within the same web application.
- The forward is performed internally by the servlet container, and the client is unaware of the forward. The client's URL, as well as any request parameters, remain unchanged.
- The target resource receives the same request and can modify the request attributes or generate a response.
- The control is transferred to the new resource, and it becomes responsible for generating the response.
- The forward method is typically used for server-side processing where multiple servlets or resources need to collaborate to generate the final response.

2. `HttpServletResponse.sendRedirect()`:

- This method instructs the client's browser to send a new request to a different URL, either within the same web application or on a different server.
- The server sends an HTTP response with a 302 status code and the new URL to the client's browser.
- The browser then initiates a new request to the provided URL, and the server processes this new request independently.
- The URL change is visible to the client, and the client's browser updates the address bar with the new URL.
- Any request parameters or attributes from the original request are not carried over to the new request unless explicitly included in the URL.
- The sendRedirect method is commonly used for client-side redirects, such as after form submissions or to redirect users to a different page.

Q7. What is the purpose of the doGet() and doPost() methods in a servlet?

Answer: The `doGet()` and `doPost()` methods are two important methods in a Java servlet that handle HTTP GET and POST requests, respectively. They are part of the HttpServlet class, which is the base class for creating servlets in Java.

1. `doGet()` method:

- The `doGet()` method is called by the servlet container to handle HTTP GET requests.

- It is responsible for processing the request and generating the response for GET requests.

- The `doGet()` method receives two parameters: the HttpServletRequest object, which encapsulates the request information, and the HttpServletResponse object, which allows the servlet to send the response back to the client.

- In the `doGet()` method, you can access the query parameters, request headers, and other information from the request object to generate a response.

- The doGet() method is typically used for retrieving data or performing read-only operations.

2. `doPost()` method:

- The `doPost()` method is called by the servlet container to handle HTTP POST requests.

- It is responsible for processing the request and generating the response for POST requests.

- Similar to the `doGet()` method, the `doPost()` method receives the HttpServletRequest and HttpServletResponse objects as parameters.

- In the `doPost()` method, you can access the request parameters, request body, and other information to perform operations such as data submission, updating a resource, or executing business logic that modifies the server's state.

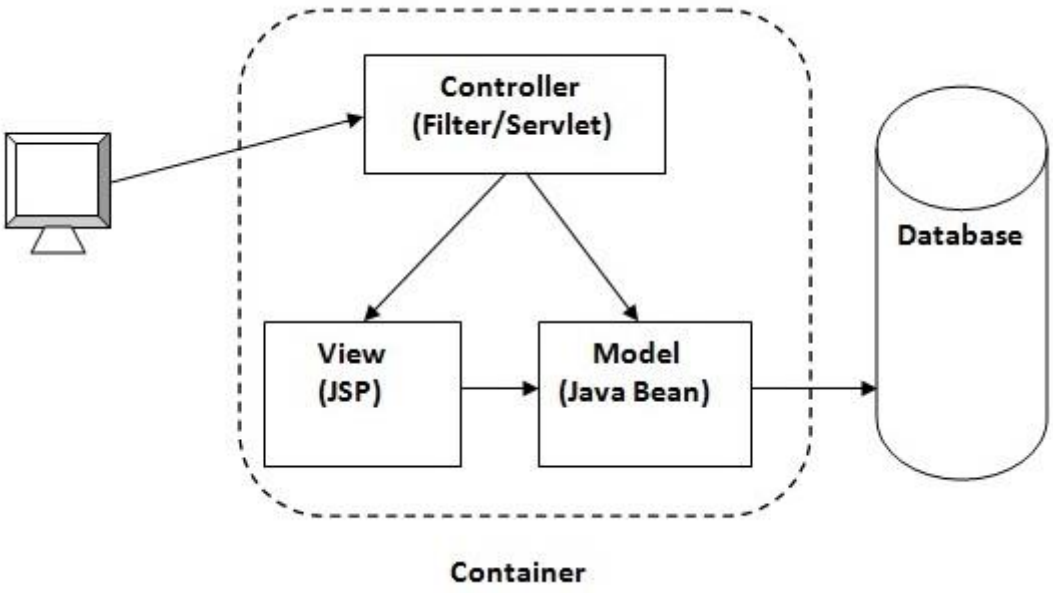
- The `doPost()` method is commonly used for operations that modify the server's state or require data to be sent securely, such as submitting a form, uploading files, or performing transactions.

Both `doGet()` and `doPost()` methods can be overridden in your servlet class to provide custom implementation based on the specific requirements of your web application. By distinguishing between GET and POST requests, servlets can handle different types of requests appropriately and generate the corresponding responses.

Q8. Explain the JSP Model-View-Controller (MVC) architecture.

Answer: MVC stands for Model View and Controller. It is a design pattern that separates the business logic, presentation logic and data.Controller acts as an interface between View and Model. Controller intercepts all the incoming requests.Model represents the state of the application i.e. data. It can also have business logic.

View represents the presentaion i.e. UI(User Interface).



Model Layer

- This is the data layer which consists of the business logic of the system.
- It consists of all the data of the application
- It also represents the state of the application.
- It consists of classes which have the connection to the database.
- The controller connects with model and fetches the data and sends to the view layer.
- The model connects with the database as well and stores the data into a database which is connected to it.

View Layer

- This is a presentation layer.
- It consists of HTML, JSP, etc. into it.
- It normally presents the UI of the application.
- It is used to display the data which is fetched from the controller which in turn fetching data from model layer classes.
- This view layer shows the data on UI of the application.

Controller Layer

- It acts as an interface between View and Model.
- It intercepts all the requests which are coming from the view layer.
- It receives the requests from the view layer and processes the requests and does the necessary validation for the request.
- This requests is further sent to model layer for data processing, and once the request is processed, it sends back to the controller with required information and displayed accordingly by the view.

Advantages of MVC Architecture

- Easy to maintain
- Easy to extend

- Easy to test
- Navigation control is centralized

Q9. What are some of the advantages of Servlets?

Answer: Servlets offer several advantages in Java web application development:

- **Platform Independence:** Servlets are written in Java, which is platform-independent. They can run on any server that supports the Java Servlet API, making them highly portable across different operating systems and web servers.
- **Server-side Processing:** Servlets enable server-side processing, allowing complex operations and business logic to be executed on the server. They can interact with databases, perform data processing, authentication, and other server-side tasks, reducing the workload on the client-side.
- **Extensibility:** Servlets can be easily extended and customized to meet specific application requirements. Developers can implement the Servlet interface or extend the HttpServlet class to add application-specific logic and functionality.
- **Performance:** Servlets are efficient in terms of performance. They are typically loaded once and reused to process multiple requests, minimizing the overhead of creating new objects for each request. Servlet containers also provide various optimizations, such as multithreading, connection pooling, and caching, to improve performance.
- **Scalability:** Servlets can handle a large number of concurrent requests, making them suitable for building scalable web applications. Servlet containers manage the lifecycle of servlets and handle request dispatching, allowing applications to handle multiple simultaneous requests efficiently.
- **Reusability:** Servlets promote code reuse and modular design. They can be easily integrated with other Java technologies and frameworks, such as JavaServer Pages (JSP), JavaServer Faces (JSF), or frameworks like Spring MVC, allowing developers to build robust and maintainable web applications.
- **Standardization:** Servlets adhere to the Java Servlet API specification, providing a standardized approach to web development. This ensures compatibility across different servlet containers and simplifies deployment and maintenance of servlet-based applications.
- **Support for Various Protocols:** Servlets are primarily designed to handle HTTP requests and responses. However, they can also support other protocols such as HTTPS, WebSocket, and more, making them versatile for different types of web applications.

Q10.What are the limitations of JSP?

Answer: JSP (JavaServer Pages) has certain limitations that developers should be aware of:

- **Mixing of Logic and Presentation:** JSP allows the mixing of Java code and HTML markup within the same page. While this can be convenient for small applications, it can lead to maintenance issues and poor code organization when complex logic and presentation code are intertwined. This mixing can make the code harder to read, test, and maintain.
- **Steep Learning Curve:** JSP requires knowledge of both Java and HTML, which can be challenging for developers who are primarily proficient in either one. Learning JSP and understanding how to effectively separate concerns between the presentation and logic can take time and effort.
- **Limited Separation of Concerns:** Although JSP supports the concept of custom tags and tag libraries to separate concerns, it can still be challenging to maintain a clean separation of concerns between the presentation layer (HTML) and the business logic (Java code). This can lead to less maintainable and harder-to-test code.
- **Limited Reusability:** JSP pages tend to have a strong coupling with the specific web application they are developed for. Reusing JSP pages in different contexts or across multiple projects can be difficult due to the tight integration with the application's specific Java code and dependencies.
- **Performance Overhead:** JSP pages are converted into Java servlets before being executed, resulting in a slight performance overhead during the compilation process. Although modern Java Servlet containers are efficient in handling JSP compilation, large-scale applications with numerous JSP pages may experience some performance impact.
- **Lack of Flexibility for Front-End Development:** JSP focuses primarily on server-side processing, and it can be cumbersome to integrate modern front-end development frameworks and tools, such as client-side

JavaScript frameworks or Single-Page Application (SPA) frameworks . JSP is more suited for generating dynamic HTML on the server-side rather than providing rich client-side interactions.

- **Limited Support for Asynchronous Processing:** JSP traditionally follows a request-response model, and handling asynchronous processing and long-running tasks can be challenging. Although newer versions of Java and Servlet specifications provide support for asynchronous processing, JSP's inherent architecture and design may limit its ability to handle such scenarios effectively.

Despite these limitations, JSP continues to be widely used and is still a viable option for certain types of web applications. However, developers should be aware of these limitations and consider alternative technologies, such as server-side frameworks like Spring MVC or client-side frameworks like React or Angular, depending on the specific requirements of their projects.