www.tutorial4us.com

Tutorial4Us

College Projects

```
void main()
{
 int no;
 clrscr();
 printf("Hello");
}
```

Basic Program

C Tutorial

JAVA Programming

tutorial4us.com

A Perfect Place for All Tutorials Resources

Core Java | Servlet | JSP | JDBC | Struts | Hibernate | Spring

Java Projects | C | C++ | DS | Interview Questions | JavaScript

College Projects | eBooks | Interview Tips | Forums | Java Discussions

For More Tutorials Stuff Visit

www.tutorial4us.com

A Perfect Place for All Tutorials Resources

JDBC, Servlet, JSP Interview Question

www.tutorial4us.com

1.What is the JDBC?

Java Database Connectivity (**JDBC**) is a standard Java API to interact with relational databases form Java. **JDBC** has set of classes and interfaces which can use from Java application and talk to database without learning RDBMS details and using Database Specific JDBC Drivers.

2.What are the new features added to JDBC 4.0?

The major features added in JDBC 4.0 include :

- Auto-loading of JDBC driver class
- Connection management enhancements
- Support for `RowId` SQL type
- DataSet implementation of SQL using Annotations
- SQL exception handling enhancements
- SQL XML support

3.Explain Basic Steps in writing a Java program using JDBC?

JDBC makes the interaction with RDBMS simple and intuitive. When a Java application needs to access database :

- Load the RDBMS specific JDBC driver because this driver actually communicates with the database (Incase of JDBC 4.0 this is automatically loaded).
- Open the connection to database which is then used to send SQL statements and get results back.
- Create JDBC Statement object. This object contains SQL query.

- Execute statement which returns resultset(s). ResultSet contains the tuples of database table as a result of SQL query.
- Process the result set.
- Close the connection.

4.Exaplain the JDBC Architecture.

The JDBC Architecture consists of two layers:

- The JDBC API, which provides the **application-to-JDBC Manager** connection.
- The JDBC Driver API, which supports the **JDBC Manager-to-Driver** Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. The location of the driver manager with respect to the JDBC drivers and the Java application is shown in Figure 1.

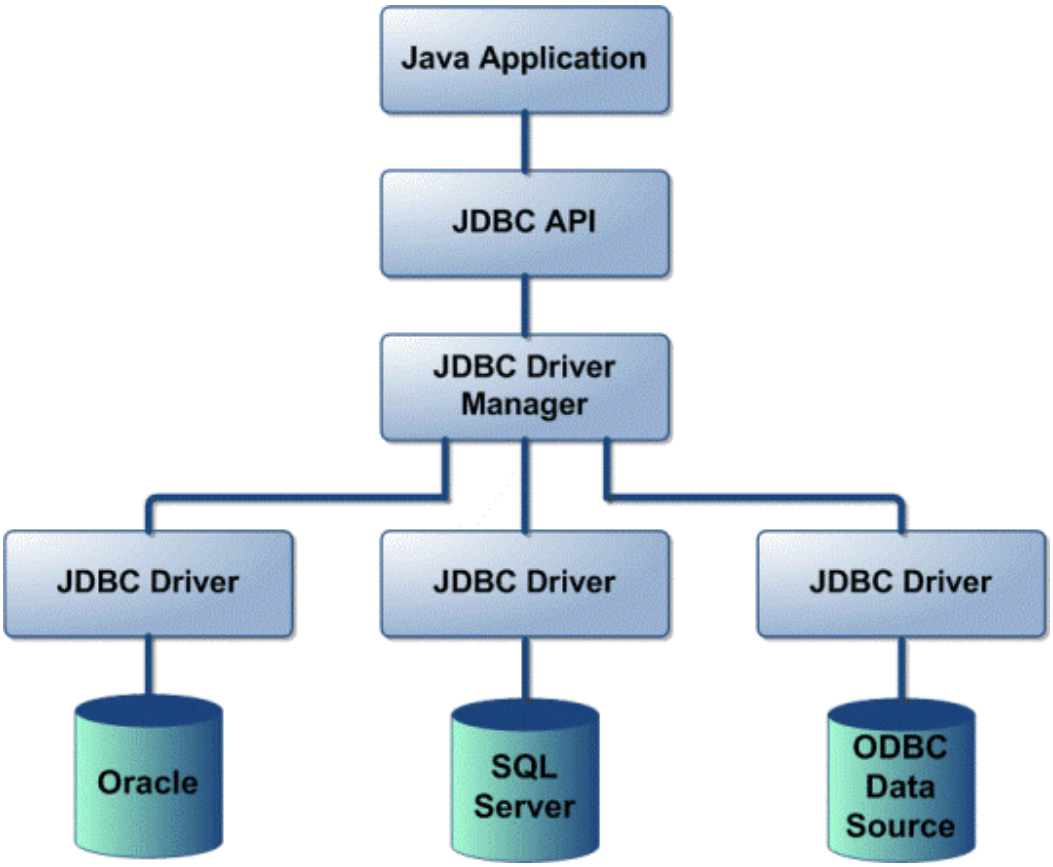


Figure 1: JDBC Architecture

5.What are the main components of JDBC ?

The life cycle of a servlet consists of the following phases:

- **DriverManager**: Manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication subprotocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

- **Driver**: The database communications link, handling all communication with the database. Normally, once the driver is loaded, the developer need not call it explicitly.

- **Connection** : Interface with all methods for contacting a database.The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement** : Encapsulates an SQL statement which is passed to the database to be parsed, compiled, planned and executed.

- **ResultSet**: The ResultSet represents set of rows retrieved due to query execution.

6.How the JDBC application works?

A JDBC application can be logically divided into two layers:

1. **Driver layer**

2. **Application layer**

- Driver layer consists of DriverManager class and the available JDBC drivers.
- The application begins with requesting the DriverManager for the connection.
- An appropriate driver is choosen and is used for establishing the connection. This connection is given to the application which falls under the application layer.
- The application uses this connection to create Statement kind of objects, through which SQL commands are sent to backend and obtain the results.

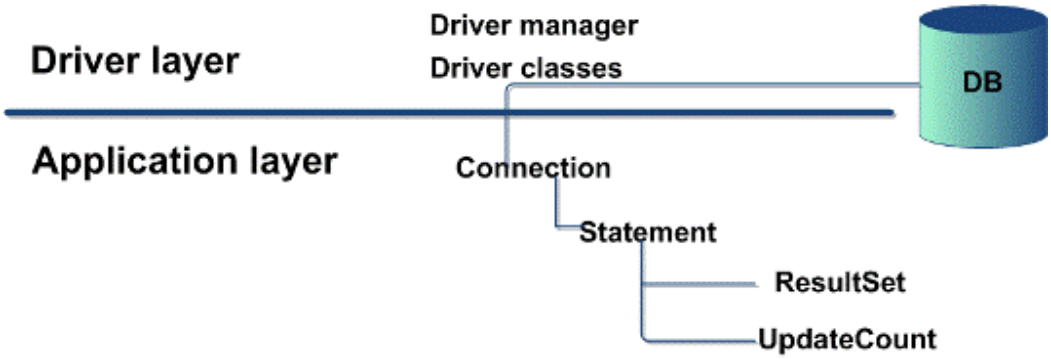


Figure 2: JDBC Application

7.**How do I load a database driver with JDBC 4.0 / Java 6?**

Provided the JAR file containing the driver is properly configured, just place the JAR file in the classpath. Java developers **NO** longer need to explicitly load JDBC drivers using code like `Class.forName()` to register a JDBC driver.The DriverManager class takes care of this by automatically locating a suitable driver when the `DriverManager.getConnection()` method is called. This feature is backward-compatible, so no changes are needed to the existing JDBC code.

8.**What is JDBC Driver interface?**

The JDBC Driver interface provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each vendor driver must provide implementations of the `java.sql.Connection,Statement,PreparedStatement, CallableStatement, ResultSet` and `Driver`.

9.**What does the connection object represents**?

The connection object represents communication context, i.e., all communication with database is through connection object only.

10.**What is Statement** ?

Statement acts like a vehicle through which SQL commands can be sent. Through the connection object we create statement kind of objects.
Through the connection object we create statement kind of objects.

```
Statement stmt  = conn.createStatement();
```

This method returns object which implements statement interface.

People who read this, also read:-

- Tibco Interview Questions
- webMethods Questions

- SCBCD Certification
- JSF Integration with Spring Framework
- Spring Interview Questions

11.What is PreparedStatement?

A prepared statement is an SQL statement that is precompiled by the database. Through precompilation, prepared statements improve the performance of SQL commands that are executed multiple times (given that the database supports prepared statements). Once compiled, prepared statements can be customized prior to each execution by altering predefined SQL parameters.

```
PreparedStatement pstmt = conn.prepareStatement("UPDATE
EMPLOYEES SET SALARY = ? WHERE ID = ?");
            pstmt.setBigDecimal(1, 153833.00);
            pstmt.setInt(2, 110592);
```

*Here: **conn** is an instance of the Connection class and "**?**" represents parameters.These parameters must be specified before execution.*

12.What is the difference between a Statement and a PreparedStatement?

Statement	PreparedStatement
A standard Statement is used to create a Java representation of a literal SQL statement and execute it on the database.	A PreparedStatement is a precompiled statement. This means that when the PreparedStatement is executed, the RDBMS can just run the

	PreparedStatement SQL statement without having to compile it first.
Statement has to verify its metadata against the database every time.	While a prepared statement has to verify its metadata against the database only once.
If you want to execute the SQL statement once go for STATEMENT	If you want to execute a single SQL statement multiple number of times, then go for PREPAREDSTATEMENT. PreparedStatement objects can be reused with passing different values to the queries

13.**What are callable statements ?**

Callable statements are used from JDBC application to invoke stored procedures and functions.

14.How to call a stored procedure from JDBC ?

PL/SQL stored procedures are called from within JDBC programs by means of the prepareCall() method of the Connection object created. A call to this method takes variable bind parameters as input parameters as well as output variables and creates an object instance of the CallableStatement class.

The following line of code illustrates this:

```
                    CallableStatement stproc_stmt =
conn.prepareCall("{call procname(?,?,?)}");
```

Here conn is an instance of the Connection class.

15.What are types of JDBC drivers?

There are four types of drivers defined by JDBC as follows:

- **Type 1: JDBC/ODBC**—These require an ODBC (Open Database Connectivity) driver for the database to be installed. This type of driver works by translating the submitted queries into equivalent ODBC queries and forwards them via native API calls directly to the ODBC driver. It provides no host redirection capability.

- **Type2: Native API (partly-Java driver)**—This type of driver uses a vendor-specific driver or database API to interact with the database. An example of such an API is Oracle OCI (Oracle Call Interface). It also provides no host redirection.

- **Type 3: Open Protocol-Net**—This is not vendor specific and works by forwarding database requests to a remote database source using a net server component. How the net server component accesses the database is transparent to the client. The client driver communicates with the net server

using a database-independent protocol and the net server translates this protocol into database calls. This type of driver can access any database.

- **Type 4: Proprietary Protocol-Net(pure Java driver)**—This has a same configuration as a type 3 driver but uses a wire protocol specific to a particular vendor and hence can access only that vendor's database. Again this is all transparent to the client.

Note: *Type 4 JDBC driver is most preferred kind of approach in JDBC.*

16.**Which type of JDBC driver is the fastest one**?

JDBC Net pure Java driver(Type IV) is the fastest driver because it converts the JDBC calls into vendor specific protocol calls and it directly interacts with the database.

17.**Does the JDBC-ODBC Bridge support multiple concurrent open statements per connection?**

No. You can open only one Statement object per connection when you are using the JDBC-ODBC Bridge.

18.Which is the right type of driver to use and when?

- Type I driver is handy for prototyping
- Type III driver adds security, caching, and connection control
- Type III and Type IV drivers need no pre-installation

Note: *Preferred by 9 out of 10 Java developers: Type IV*. Click here to learn more about JDBC drivers.

19.**What are the standard isolation levels defined by JDBC?**

The values are defined in the class java.sql.Connection and are:

- TRANSACTION_NONE
- TRANSACTION_READ_COMMITTED
- TRANSACTION_READ_UNCOMMITTED
- TRANSACTION_REPEATABLE_READ
- TRANSACTION_SERIALIZABLE

Any given database may not support all of these levels.

20.What is resultset ?

The ResultSet represents set of rows retrieved due to query execution.

```
ResultSet rs = stmt.executeQuery(sqlQuery);
```

21.What are the types of resultsets?

The values are defined in the class java.sql.Connection and are:

- `TYPE_FORWARD_ONLY` specifies that a resultset is not scrollable, that is, rows within it can be advanced only in the forward direction.
- `TYPE_SCROLL_INSENSITIVE` specifies that a resultset is scrollable in either direction but is insensitive to changes committed by other transactions or other statements in the same transaction.
- `TYPE_SCROLL_SENSITIVE` specifies that a resultset is scrollable in either direction and is affected by changes committed by other transactions or statements within the same transaction.

Note: *`A TYPE_FORWARD_ONLY` resultset is always insensitive.*

22.**What's the difference between TYPE_SCROLL_INSENSITIVE and TYPE_SCROLL_SENSITIVE?**

TYPE_SCROLL_INSENSITIVE	TYPE_SCROLL_SENSITIVE
An insensitive resultset is like the	A sensitive resultset does NOT

snapshot of the data in the database when query was executed.	represent a snapshot of data, rather it contains points to those rows which satisfy the query condition.
After we get the resultset the changes made to data are not visible through the resultset, and hence they are known as insensitive.	After we obtain the resultset if the data is modified then such modifications are visible through resultset.
Performance not effected with insensitive.	Since a trip is made for every '**get'** operation, the performance drastically get affected.

22.**What is rowset?**

A RowSet is an object that encapsulates a set of rows from either Java Database Connectivity (JDBC) result sets or tabular data sources like a file or spreadsheet. RowSets support component-based development models like JavaBeans, with a standard set of properties and an event notification mechanism.

24.**What are the different types of RowSet ?**

There are two types of RowSet are there. They are:

- **Connected** - A connected RowSet object connects to the database once and remains connected until the application terminates.

- **Disconnected -** A disconnected RowSet object connects to the database, executes a query to retrieve the data from the database and then closes the connection. A program may change the data in a disconnected RowSet while it is disconnected. Modified data can be updated in the database after a disconnected RowSet reestablishes the connection with the database.

25.**What is the need of BatchUpdates?**

The BatchUpdates feature allows us to group SQL statements together and send to database server in one single trip.

26.**What is a DataSource?**

A DataSource object is the representation of a data source in the Java programming language. In basic terms,

- A DataSource is a facility for storing data.
- DataSource can be referenced by JNDI.
- Data Source may point to RDBMS, file System , any DBMS etc..

27.**What are the advantages of DataSource?**

The few advantages of data source are :

- An application does not need to hardcode driver information, as it does with the `DriverManager`.

- The `DataSource` implementations can easily change the properties of data sources. *For example*: There is no need to modify the application code when making changes to the database details.
- The DataSource facility allows developers to implement a `DataSource` class to take advantage of features like connection pooling and distributed transactions.

28.**What is connection pooling? what is the main advantage of using connection pooling?**

A connection pool is a mechanism to reuse connections created. Connection pooling can increase performance dramatically by reusing connections rather than creating a new physical connection each time a connection is requested..

1.What is the Servlet?

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.

2.What are the new features added to Servlet 2.5?

Following are the changes introduced in Servlet 2.5:

- A new dependency on J2SE 5.0
- Support for annotations
- Loading the class
- Several web.xml conveniences
- A handful of removed restrictions
- Some edge case clarifications

Learn more about Servlets 2.5 features

3.What are the uses of Servlet?

Typical uses for HTTP Servlets include:

- Processing and/or storing data submitted by an HTML form.
- Providing dynamic content, e.g. returning the results of a database query to the client.
- A Servlet can handle multiple request concurrently and be used to develop high performance system
- Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

4.What are the advantages of Servlet over CGI?

Servlets have several advantages over CGI:

- A Servlet does not run in a separate process. This removes the overhead of creating a new process for each request.
- A Servlet stays in memory between requests. A CGI program (and probably also an extensive runtime system or interpreter) needs to be loaded and started for each CGI request.
- There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
- Several web.xml conveniences
- A handful of removed restrictions
- Some edge case clarifications

5.What are the phases of the servlet life cycle?

The life cycle of a servlet consists of the following phases:

- **Servlet class loading** : For each servlet defined in the deployment descriptor of the Web application, the servlet container locates and loads a class of the type of the servlet. This can happen when the servlet engine itself is started, or later when a client request is actually delegated to the servlet.

- **Servlet instantiation** : After loading, it instantiates one or more object instances of the servlet class to service the client requests.

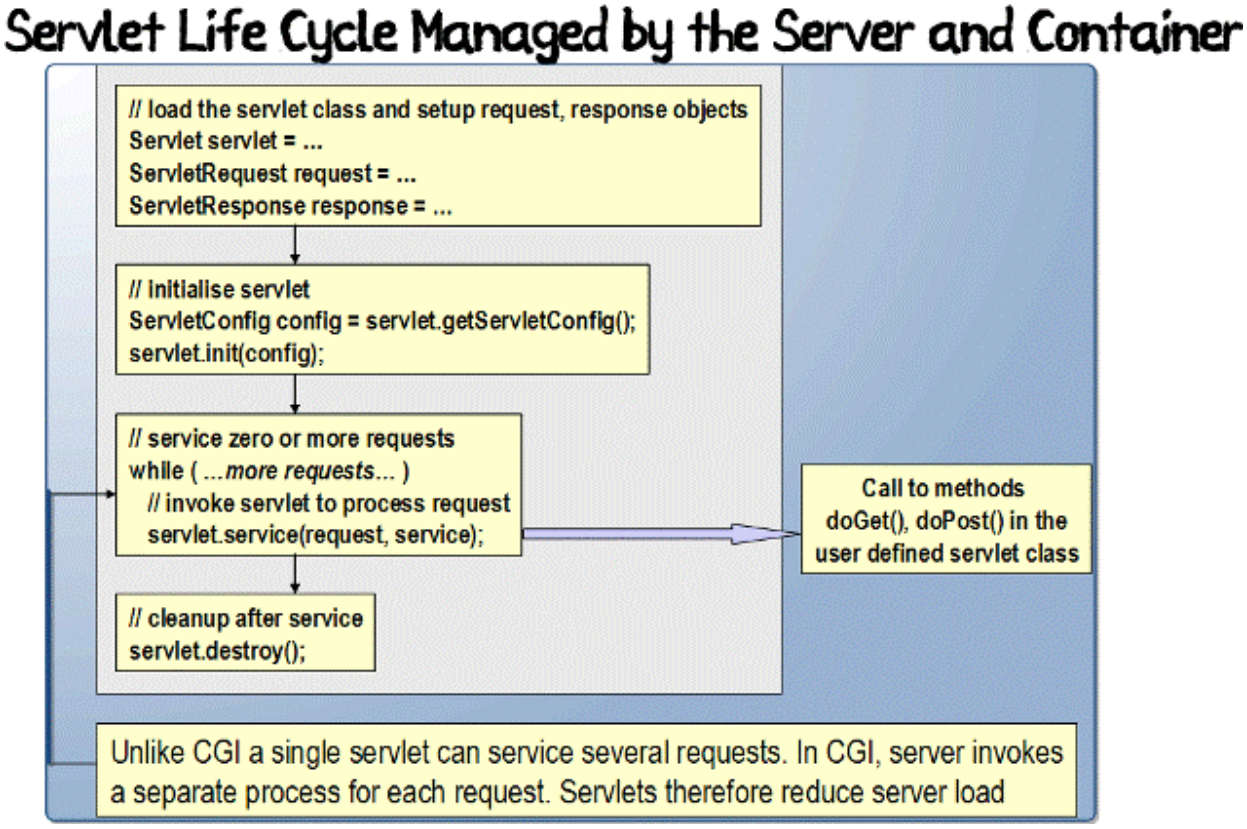
- **Initialization (call the init method)** : After instantiation, the container initializes a servlet before it is ready to handle client requests. The container

initializes the servlet by invoking its init() method, passing an object implementing the ServletConfig interface. In the init() method, the servlet can read configuration parameters from the deployment descriptor or perform any other one-time activities, so the init() method is invoked once and only once by the servlet container.

- **Request handling (call the service method)** : After the servlet is initialized, the container may keep it ready for handling client requests. When client requests arrive, they are delegated to the servlet through the service() method, passing the request and response objects as parameters. In the case of HTTP requests, the request and response objects are implementations of HttpServletRequest and HttpServletResponse respectively. In the HttpServlet class, the service() method invokes a different handler method for each type of HTTP request, doGet() method for GET requests, doPost() method for POST requests, and so on.

- **Removal from service (call the destroy method)** : A servlet container may decide to remove a servlet from service for various reasons, such as to conserve memory resources. To do this, the servlet container calls the destroy() method on the servlet. Once the destroy() method has been called, the servlet may not service any more client requests. Now the servlet instance is eligible for garbage collection

  The life cycle of a servlet is controlled by the container in which the servlet has been deployed.



6.Why do we need a constructor in a servlet if we use the init method?

Even though there is an init method in a servlet which gets called to initialize it, a constructor is still required to instantiate the servlet. Even though you as the developer would never need to explicitly call the servlet's constructor, it is still being used by the container (the container still uses the constructor to create an instance of the servlet). Just like a normal POJO (plain old java object) that might have an init method, it is no use calling the init method if you haven't constructed an object to call it on yet.

7.How the servlet is loaded?

A servlet can be loaded when:

- First request is made.
- Server starts up (auto-load).
- There is only a single instance which answers all requests concurrently. This saves memory and allows a Servlet to easily manage persistent data.
- Administrator manually loads.

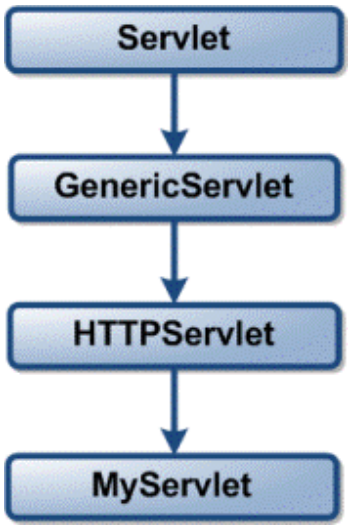
8.How a Servlet is unloaded?

A servlet is unloaded when:

- Server shuts down.
- Administrator manually unloads.

9.What is Servlet interface?

The central abstraction in the Servlet API is the Servlet interface. All servlets implement this interface, either directly or , more commonly by extending a class that implements it.



Note: Most Servlets, however, extend one of the standard implementations of that interface, namely `javax.servlet.GenericServlet` and `javax.servlet.http.HttpServlet`.

10.What is the GenericServlet class?

GenericServlet is an abstract class that implements the Servlet interface and the ServletConfig interface. In addition to the methods declared in these two interfaces, this class also provides simple versions of the lifecycle methods init and destroy, and implements the log method declared in the ServletContext interface.

Note: This class is known as generic servlet, since it is not specific to any protocol.

11.What's the difference between GenericServlet and HttpServlet?

GenericServlet	HttpServlet
The GenericServlet is an abstract class that is extended by HttpServlet to provide HTTP protocol-specific methods.	An abstract class that simplifies writing HTTP servlets. It extends the GenericServlet base class and provides an framework for handling the HTTP protocol.
The GenericServlet does not include protocol-specific methods for handling request parameters, cookies, sessions and setting response headers.	The HttpServlet subclass passes generic service method requests to the relevant doGet() or doPost() method.
GenericServlet is not specific to any protocol.	HttpServlet only supports HTTP and HTTPS protocol.

12.Why is HttpServlet declared abstract?

The HttpServlet class is declared abstract because the default implementations of the main service methods do nothing and must be overridden. This is a convenience implementation of the Servlet interface, which means that developers do not need to implement all service methods. If your servlet is required to handle `doGet()` requests for example, there is no need to write a `doPost()` method too.

13.Can servlet have a constructor ?

One can definitely have constructor in servlet.Even you can use the constrctor in servlet for initialization purpose,but this type of approch is not so common. You can perform common operations with the constructor as you normally do.The only thing is that you cannot call that constructor explicitly by the new keyword as we normally do.In the case of servlet, servlet container is responsible for instantiating the servlet, so the constructor is also called by servlet container only.

14.**What are the types of protocols supported by HttpServlet ?**

It extends the GenericServlet base class and provides a framework for handling the HTTP protocol. So, HttpServlet only supports HTTP and HTTPS protocol.

15.What is the difference between doGet() and doPost()?

#	doGet()	doPost()
1	In doGet() the parameters are appended to the URL and sent along with header information.	In doPost(), on the other hand will (typically) send the information through a socket back to the webserver and it won't show up in the URL bar.
2	The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.	You can send much more information to the server this way - and it's not restricted to textual data either. It is

		possible to send files and even binary data such as serialized Java objects!
3	doGet() is a request for information; it does not (or should not) change anything on the server. (doGet() should be idempotent)	doPost() provides information (such as placing an order for merchandise) that the server is expected to remember
4	Parameters are not encrypted	Parameters are encrypted
5	doGet() is faster if we set the response content length since the same connection is used. Thus increasing the performance	doPost() is generally used to update or post some information to the server.doPost is slower compared to doGet since doPost does not write the content length
6	doGet() should be idempotent. i.e. doget should be able to be repeated safely many times	This method does not need to be idempotent. Operations requested through POST can have side effects for which the user can be held accountable.
7	doGet() should be safe without any side effects for which user is held responsible	This method does not need to be either safe
8	It allows bookmarks.	It disallows bookmarks.

16.When to use doGet() and when doPost()?

Always prefer to use GET (As because GET is faster than POST), except mentioned in the following reason:

- If data is sensitive
- Data is greater than 1024 characters
- If your application don't need bookmarks.

17.How do I support both GET and POST from the same Servlet?

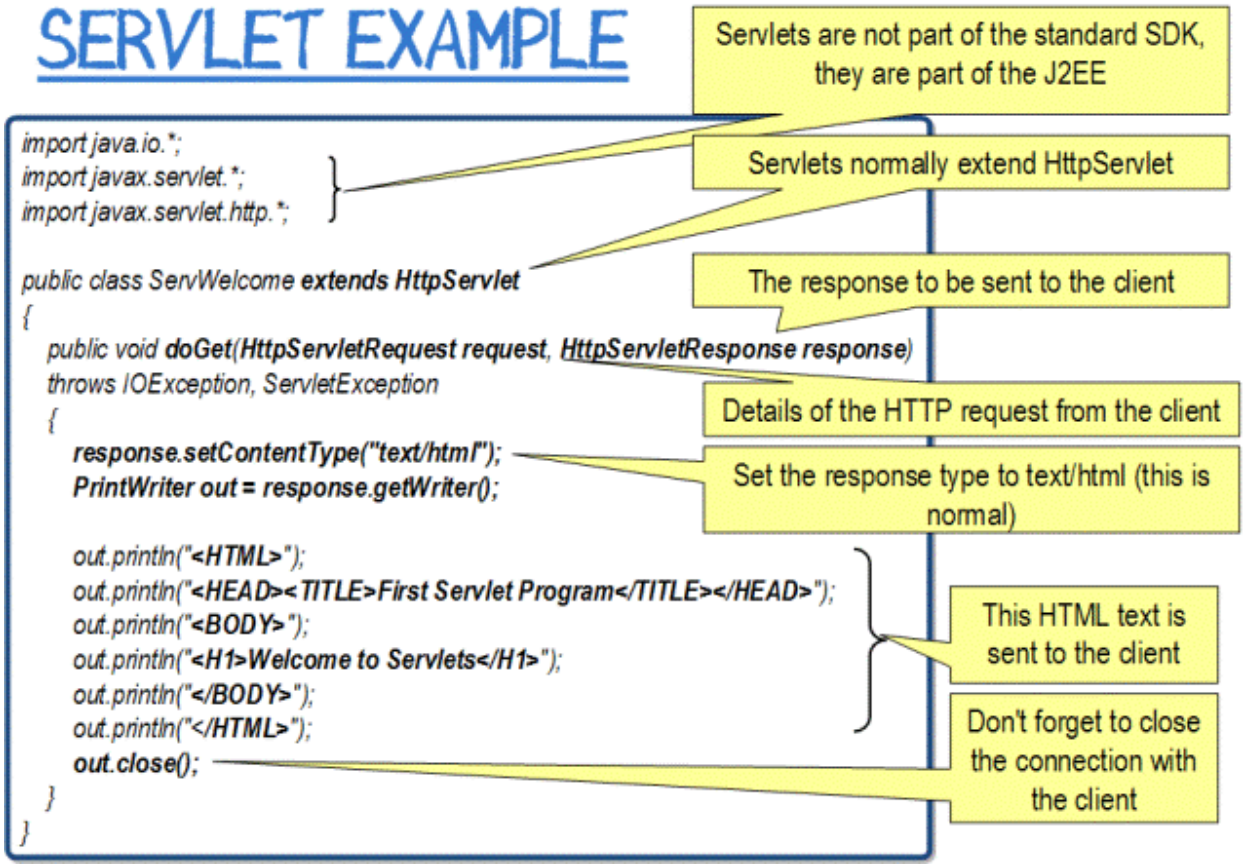
The easy way is, just support POST, then have your doGet method call your doPost method:

```
 public void doGet(HttpServletRequest request,
HttpServletResponse response)
                        throws ServletException, IOException
{
    doPost(request, response);
}
```

18.Should I override the service() method?

We never override the service method, since the HTTP Servlets have already taken care of it . The default service function invokes the doXXX() method corresponding to the method of the HTTP request.For example, if the HTTP request method is GET, doGet() method is called by default. A servlet should override the doXXX() method for the HTTP methods that servlet supports. Because HTTP service method check the request method and calls the appropriate handler method, it is not necessary to override the service method itself. Only override the appropriate doXXX() method.

19.How the typical servlet code look like ?



20.What is a servlet context object?

A servlet context object contains the information about the Web application of which the servlet is a part. It also provides access to the resources common to all the servlets in the application. Each Web application in a container has a single servlet context associated with it.

21.What are the differences between the ServletConfig interface and the ServletContext interface?

ServletConfig	ServletContext
The ServletConfig interface is implemented by the servlet container in order to pass configuration information to a servlet. The server passes an object that implements the ServletConfig interface to the servlet's init() method.	A ServletContext defines a set of methods that a servlet uses to communicate with its servlet container.
There is one ServletConfig parameter per servlet.	There is one ServletContext for the entire webapp and all the servlets in a webapp share it.
The param-value pairs for ServletConfig object are specified in the <init-param> within the <servlet> tags in the web.xml file	The param-value pairs for ServletContext object are specified in the <context-param> tags in the web.xml file.

22.What's the difference between forward() and sendRedirect() methods?

forward()	sendRedirect()
A forward is performed internally by the servlet.	A redirect is a two step process, where the web application instructs the browser to fetch a second URL, which differs from the original.
The browser is completely unaware that it has taken place, so its original URL remains intact.	The browser, in this case, is doing the work and knows that it's making a new request.
Any browser reload of the resulting page will simple repeat the original request, with the original URL	A browser reloads of the second URL ,will not repeat the original request, but will rather fetch the second URL.
Both resources must be part of the same context (Some containers make provisions for cross-context communication but this tends not to be very portable)	This method can be used to redirect users to resources that are not part of the current context, or even in the same domain.
Since both resources are part of same context, the original request context is retained	Because this involves a new request, the previous request scope objects, with all of its parameters and attributes are no longer available after a redirect. (Variables will need to be passed by via the session object).

Forward is marginally faster than redirect.	redirect is marginally slower than a forward, since it requires two browser requests, not one.

23.What is the difference between the include() and forward() methods?

include()	forward()
The `RequestDispatcher include()` method inserts the the contents of the specified resource directly in the flow of the servlet response, as if it were part of the calling servlet.	The `RequestDispatcher forward()` method is used to show a different resource in place of the servlet that was originally called.
If you include a servlet or JSP document, the included resource must not attempt to change the response status code or HTTP headers, any such request will be ignored.	The forwarded resource may be another servlet, JSP or static HTML document, but the response is issued under the same URL that was originally requested. In other words, it is not the same as a redirection.
The `include()` method is often used to include common "boilerplate" text or template markup that may be included by many servlets.	The `forward()` method is often used where a servlet is taking a controller role; processing some input and deciding the outcome by returning a

	particular response page.

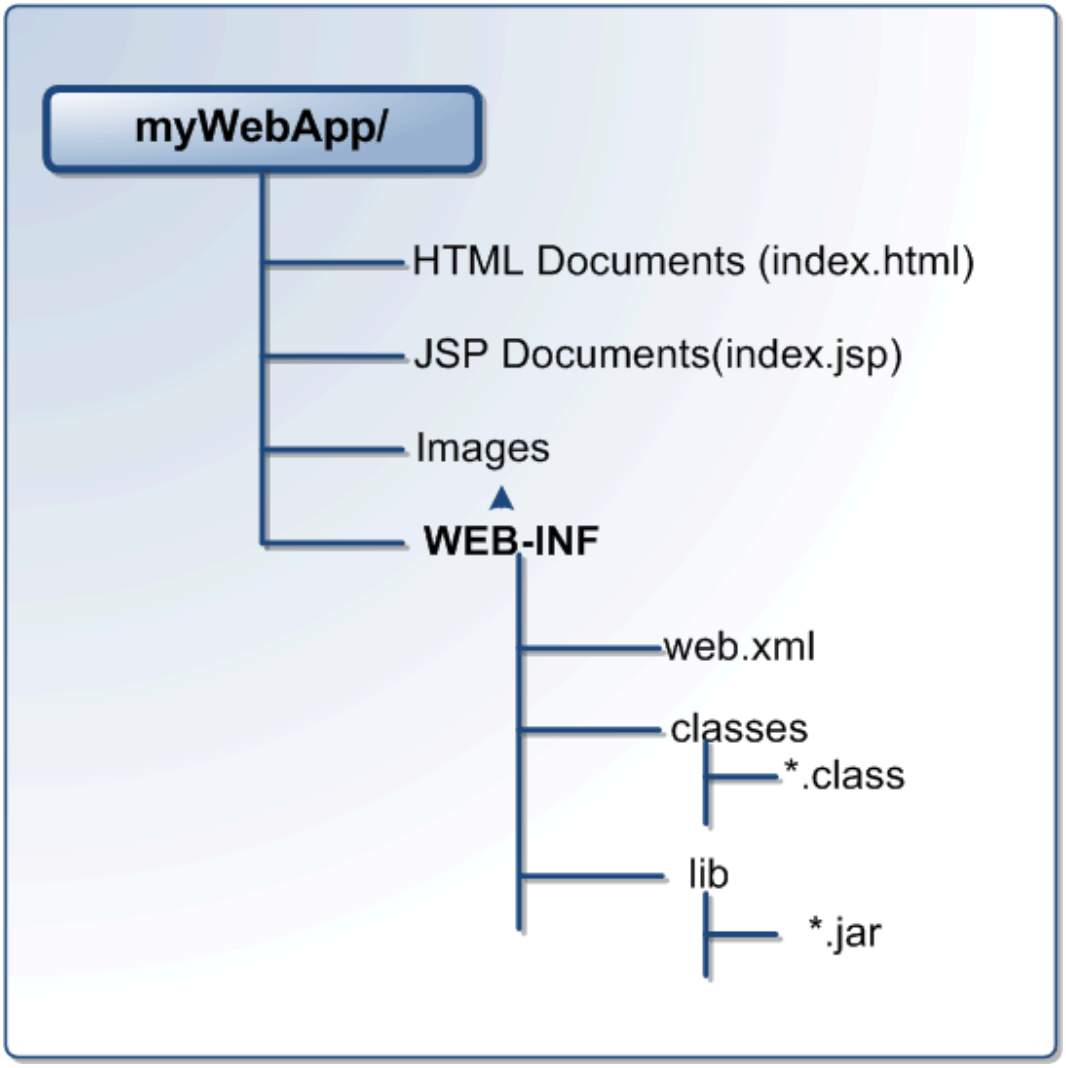
24.What's the use of the servlet wrapper classes??

People who read this, also read:-

- Webservices Interview Questions
- JSF Standard Components
- WebSphere Certification
- Hibernate Vs iBatis
- Core Java Questions

The `HttpServletRequestWrapper` and `HttpServletResponseWrapper` classes are designed to make it easy for developers to create custom implementations of the servlet request and response types. The classes are constructed with the standard `HttpServletRequest` and `HttpServletResponse` instances respectively and their default behaviour is to pass all method calls directly to the underlying objects.

25.What is the directory structure of a WAR file?



26.What is a deployment descriptor?

A deployment descriptor is an XML document with an .xml extension. It defines a component's deployment settings. It declares transaction attributes and security authorization for an enterprise bean. The information provided by a deployment descriptor is declarative and therefore it can be modified without changing the source code of a bean.

The JavaEE server reads the deployment descriptor at run time and acts upon the component accordingly.

27.What is the difference between the getRequestDispatcher(String path) method of javax.servlet.ServletRequest interface and javax.servlet.ServletContext interface?

ServletRequest.getRequestDispatcher (String path)	ServletContext.getRequestDispatcher (String path)
The `getRequestDispatcher(String path)` method of `javax.servlet.ServletRequest` interface accepts parameter the path to the resource to be included or forwarded to, which can be relative to the request of the calling servlet. If the path begins with a "/" it is interpreted as relative to the current context root.	The `getRequestDispatcher(String path)` method of `javax.servlet.ServletContext` interface cannot accept relative paths. All path must start with a "/" and are interpreted as relative to current context root.

28.What is preinitialization of a servlet?

A container does not initialize the servlets as soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading. The servlet specification defines the element, which can be specified in the deployment descriptor to make the servlet container load and initialize the

servlet as soon as it starts up. The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

29.What is the <load-on-startup> element?

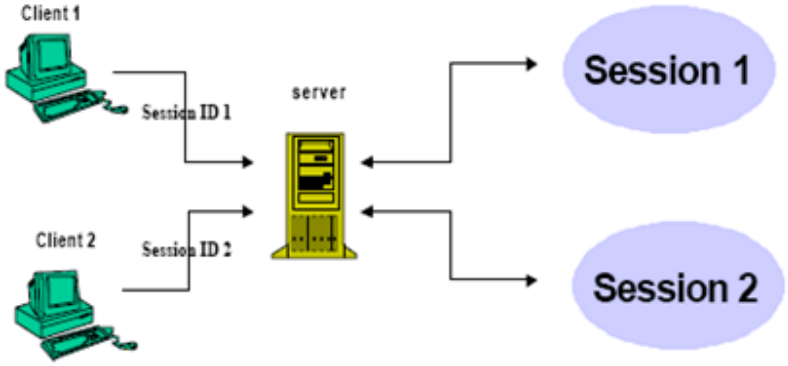
The `<load-on-startup>` element of a deployment descriptor is used to load a servlet file when the server starts instead of waiting for the first request. It is also used to specify the order in which the files are to be loaded. The <load-on-startup> element is written in the deployment descriptor as follows:

```
<servlet>
   <servlet-name>ServletName</servlet-name>
   <servlet-class>ClassName</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>
```

Note: The container loads the servlets in the order specified in the <load-on-startup> element.

30.What is session?

A session refers to all the requests that a single client might make to a server in the course of viewing any pages associated with a given application. Sessions are specific to both the individual user and the application. As a result, every user of an application has a separate session and has access to a separate set of session variables.



31.What is Session Tracking?

Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time.

32.What is the need of Session Tracking in web application?

HTTP is a stateless protocol i.e., every request is treated as new request. For web applications to be more realistic they have to retain information across multiple requests. Such information which is part of the application is reffered as "state". To

keep track of this state we need session tracking.

Typical example: Putting things one at a time into a shopping cart, then checking out--each page request must somehow be associated with previous requests.

33.What are the types of Session Tracking ?

Sessions need to work with all web browsers and take into account the users security preferences. Therefore there are a variety of ways to send and receive the identifier:

- **URL rewriting :** URL rewriting is a method of session tracking in which some extra data (session ID) is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session. This method is used with browsers that do not support cookies or where the user has disabled the cookies.

- **Hidden Form Fields :** Similar to URL rewriting. The server embeds new hidden fields in every dynamically generated form page for the client. When the client submits the form to the server the hidden fields identify the client.

- **Cookies :** Cookie is a small amount of information sent by a servlet to a Web browser. Saved by the browser, and later sent back to the server in

subsequent requests. A cookie has a name, a single value, and optional attributes. A cookie's value can uniquely identify a client.

- **Secure Socket Layer (SSL) Sessions :** Web browsers that support Secure Socket Layer communication can use SSL's support via HTTPS for generating a unique session key as part of the encrypted conversation.

Learn more about Session Tracking

34.How do I use cookies to store session state on the client?

In a servlet, the HttpServletResponse and HttpServletRequest objects passed to method HttpServlet.service() can be used to create cookies on the client and use cookie information transmitted during client requests. JSPs can also use cookies, in scriptlet code or, preferably, from within custom tag code.

- To set a cookie on the client, use the addCookie() method in class HttpServletResponse. Multiple cookies may be set for the same request, and a single cookie name may have multiple values.
- To get all of the cookies associated with a single HTTP request, use the getCookies() method of class HttpServletRequest

35.What are some advantages of storing session state in cookies?

- Cookies are usually persistent, so for low-security sites, user data that needs to be stored long-term (such as a user ID, historical information, etc.) can be maintained easily with no server interaction.
- For small- and medium-sized session data, the entire session data (instead of just the session ID) can be kept in the cookie.

36.What are some disadvantages of storing session state in cookies?

- Cookies are controlled by programming a low-level API, which is more difficult to implement than some other approaches.
- All data for a session are kept on the client. Corruption, expiration or purging of cookie files can all result in incomplete, inconsistent, or missing information.
- Cookies may not be available for many reasons: the user may have disabled them, the browser version may not support them, the browser may be behind a firewall that filters cookies, and so on. Servlets and JSP pages that rely exclusively on cookies for client-side session state will not operate properly for all clients. Using cookies, and then switching to an alternate client-side session state strategy in cases where cookies aren't available, complicates development and maintenance.

- Browser instances share cookies, so users cannot have multiple simultaneous sessions.
- Cookie-based solutions work only for HTTP clients. This is because cookies are a feature of the HTTP protocol. Notice that the while package `javax.servlet.http` supports session management (via class `HttpSession`), package `javax.servlet` has no such support.

37.What is URL rewriting?

URL rewriting is a method of session tracking in which some extra data is appended at the end of each URL. This extra data identifies the session. The server can associate this session identifier with the data it has stored about that session.

Every URL on the page must be encoded using method `HttpServletResponse`.encodeURL(). Each time a URL is output, the servlet passes the URL to encodeURL(), which encodes session ID in the URL if the browser isn't accepting cookies, or if the session tracking is turned off.
E.g., http://abc/path/index.jsp;jsessionid=123465hfhs

Advantages

- URL rewriting works just about everywhere, especially when cookies are turned off.
- Multiple simultaneous sessions are possible for a single user. Session information is local to each browser instance, since it's stored in URLs in each page being displayed. This scheme isn't foolproof, though, since users can start a new browser instance using a URL for an active session, and

confuse the server by interacting with the same session through two instances.

- Entirely static pages cannot be used with URL rewriting, since every link must be dynamically written with the session state. It is possible to combine static and dynamic content, using (for example) templating or server-side includes. This limitation is also a barrier to integrating legacy web pages with newer, servlet-based pages.

DisAdvantages

- Every URL on a page which needs the session information must be rewritten each time a page is served. Not only is this expensive computationally, but it can greatly increase communication overhead.
- URL rewriting limits the client's interaction with the server to HTTP GETs, which can result in awkward restrictions on the page.
- URL rewriting does not work well with JSP technology.
- If a client workstation crashes, all of the URLs (and therefore all of the data for that session) are lost.

38.How can an existing session be invalidated?

An existing session can be invalidated in the following two ways:

- Setting timeout in the deployment descriptor: This can be done by specifying timeout between the `<session-timeout>`tags as follows:

```
<session-config>
        <session-timeout>10</session-timeout>
</session-config>
```

This will set the time for session timeout to be ten minutes.

- Setting timeout programmatically: This will set the timeout for a specific session. The syntax for setting the timeout programmatically is as follows:

  `public void setMaxInactiveInterval(int interval)`
  The `setMaxInactiveInterval()` method sets the maximum time in seconds before a session becomes invalid.
  Note :Setting the inactive period a*s negative(-1), makes the container stop tracking session, i.e, session never expires.*

39.How can the session in Servlet can be destroyed?

An existing session can be destroyed in the following two ways:

- Programatically : Using `session.invalidate()` method, which makes the container abonden the session on which the method is called.
- When the server itself is shutdown.

40.A client sends requests to two different web components. Both of the components access the session. Will they end up using the same session object or different session ?

Creates only one session i.e., they end up with using same session .

Sessions is specific to the client but not the web components. And there is a 1-1 mapping between client and a session.

41.What is servlet lazy loading?

- A container doesnot initialize the servlets ass soon as it starts up, it initializes a servlet when it receives a request for that servlet first time. This is called lazy loading.
- The servlet specification defines the <load-on-startup> element, which can be specified in the deployment descriptor to make the servlet container load and initialize the servlet as soon as it starts up.
- The process of loading a servlet before any request comes in is called preloading or preinitializing a servlet.

42.What is Servlet Chaining?

Servlet Chaining is a method where the output of one servlet is piped into a second servlet. The output of the second servlet could be piped into a third servlet, and so on. The last servlet in the chain returns the output to the Web browser.

43.How are filters?

Filters are Java components that are used to intercept an incoming request to a Web resource and a response sent back from the resource. It is used to abstract any useful information contained in the request or response. Some of the important functions performed by filters are as follows:

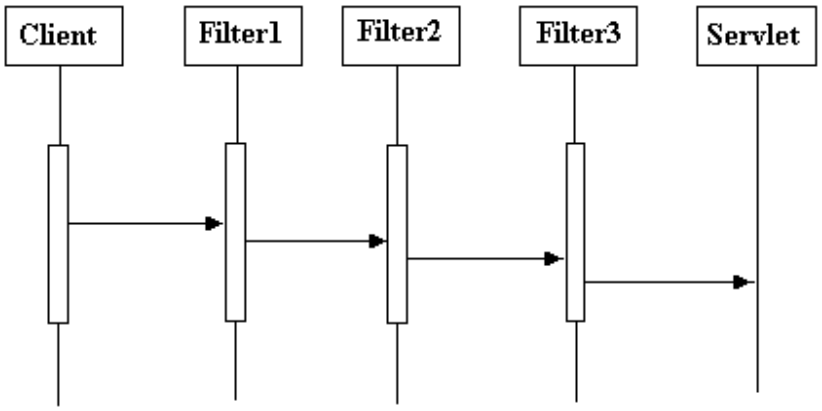
- Security checks
- Modifying the request or response
- Data compression
- Logging and auditing
- Response compression

Filters are configured in the deployment descriptor of a Web application. Hence, a user is not required to recompile anything to change the input or output of the Web application.

44.What are the functions of an intercepting filter?

The functions of an intercepting filter are as follows:

- It intercepts the request from a client before it reaches the servlet and modifies the request if required.
- It intercepts the response from the servlet back to the client and modifies the request if required.
- There can be many filters forming a chain, in which case the output of one filter becomes an input to the next filter. Hence, various modifications can be performed on a single request and response.



45.What are the functions of the Servlet container?

The functions of the Servlet container are as follows:

- **Lifecycle management** : It manages the life and death of a servlet, such as class loading, instantiation, initialization, service, and making servlet instances eligible for garbage collection.
- **Communication support** : It handles the communication between the servlet and the Web server.
- **Multithreading support** : It automatically creates a new thread for every servlet request received. When the Servlet service() method completes, the thread dies.
- **Declarative security** : It manages the security inside the XML deployment descriptor file.
- **JSP support** : The container is responsible for converting JSPs to servlets and for maintaining them.

1 What are the advantages of JSP over Servlet?

JSP is a serverside technology to make content generation a simple appear.The advantage of JSP is that they are document-centric. Servlets, on the other hand, look and act like programs. A Java Server Page can contain Java program fragments that instantiate and execute Java classes, but these occur inside an HTML template file and are primarily used to generate dynamic content. Some of

the JSP functionality can be achieved on the client, using JavaScript. The power of JSP is that it is server-based and provides a framework for Web application development.

2.What is the life-cycle of JSP?

When a request is mapped to a JSP page for the first time, it translates the JSP page into a servlet class and compiles the class. It is this servlet that services the client requests.
A JSP page has seven phases in its lifecycle, as listed below in the sequence of occurrence:

- Translation
- Compilation
- Loading the class
- Instantiating the class
- jspInit() invocation
- _jspService() invocation
- jspDestroy() invocation

More about JSP Life cycle

3.What is the jspInit() method?

The jspInit() method of the javax.servlet.jsp.JspPage interface is similar to the init() method of servlets. This method is invoked by the container only once when a JSP page is initialized. It can be overridden by a page author to initialize

resources such as database and network connections, and to allow a JSP page to read persistent configuration data.

4.What is the _jspService() method?

SThe _jspService() method of the javax.servlet.jsp.HttpJspPage interface is invoked every time a new request comes to a JSP page. This method takes the HttpServletRequest and HttpServletResponse objects as its arguments. A page author cannot override this method, as its implementation is provided by the container.

5.What is the jspDestroy() method?

The jspDestroy() method of the javax.servlet.jsp.JspPage interface is invoked by the container when a JSP page is about to be destroyed. This method is similar to the destroy() method of servlets. It can be overridden by a page author to perform any cleanup operation such as closing a database connection.

6.What JSP lifecycle methods can I override?

You cannot override the _jspService() method within a JSP page. You can however, override the jspInit() and jspDestroy() methods within a JSP page. jspInit() can be useful for allocating resources like database connections, network connections, and so forth for the JSP page. It is good programming practice to free any allocated resources within jspDestroy().

7.How can I override the jspInit() and jspDestroy() methods within a JSP page?

The jspInit() and jspDestroy() methods are each executed just once during the lifecycle of a JSP page and are typically declared as JSP declarations:

```
<%!
      public void jspInit() {
            . . .
      }
%>
<%!
      public void jspDestroy() {
            . . .
      }
%>
```

8.What are implicit objects in JSP?

Implicit objects in JSP are the Java objects that the JSP Container makes available to developers in each page. These objects need not be declared or instantiated by the JSP author. They are automatically instantiated by the container and are accessed using standard variables; hence, they are called implicit objects.The implicit objects available in JSP are as follows:

- request
- response
- pageContext
- session

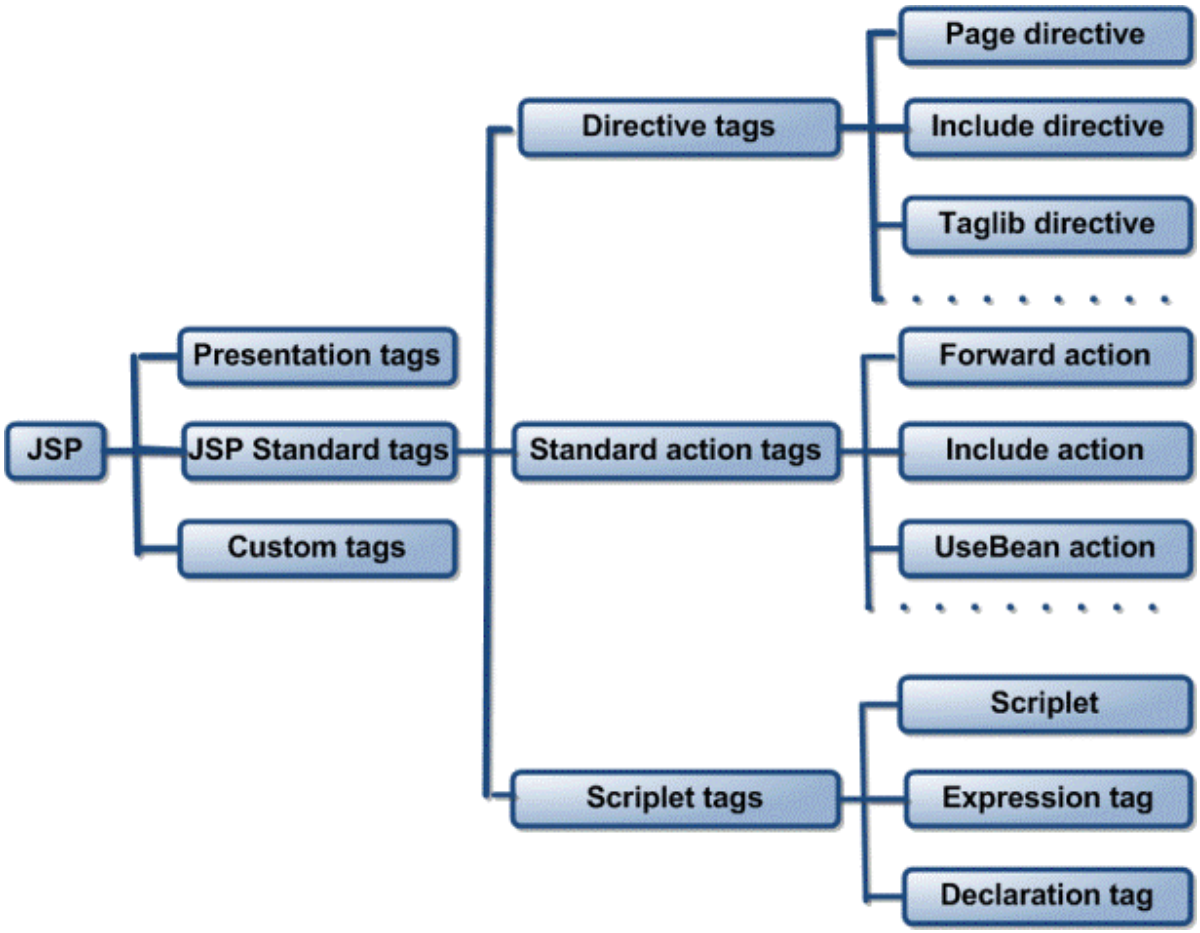
- application
- out
- config
- page
- exception

The implicit objects are parsed by the container and inserted into the generated servlet code. They are available only within the jspService method and not in any declaration.

Check more about implicit objects

9.What are the different types of JSP tags?

The different types of JSP tags are as follows:



10.What are JSP directives?

- JSP directives are messages for the JSP engine. i.e., JSP directives serve as a message from a JSP page to the JSP container and control the processing of the entire page
- They are used to set global values such as a class declaration, method implementation, output content type, etc.
- They do not produce any output to the client.
- Directives are always enclosed within <%@ ….. %> tag.
- Ex: page directive, include directive, etc.

11.What is page directive?

- A page directive is to inform the JSP engine about the headers or facilities that page should get from the environment.
- Typically, the page directive is found at the top of almost all of our JSP pages.
- There can be any number of page directives within a JSP page (although the attribute – value pair must be unique).
- The syntax of the include directive is: <%@ page attribute="value">
- Example:`<%@ include file="header.jsp" %>`

12.What are the attributes of page directive?

There are thirteen attributes defined for a page directive of which the **important** attributes are as follows:

- **import**: It specifies the packages that are to be imported.
- **session**: It specifies whether a session data is available to the JSP page.
- **contentType**: It allows a user to set the content-type for a page.
- **isELIgnored**: It specifies whether the EL expressions are ignored when a JSP is translated to a servlet.

13.What is the include directive?

There are thirteen attributes defined for a page directive of which the **important** attributes are as follows:

- The include directive is used to statically insert the contents of a resource into the current JSP.
- This enables a user to reuse the code without duplicating it, and includes the contents of the specified file at the translation time.
- The syntax of the include directive is as follows:
  `<%@ include file = "FileName" %>`
- This directive has only one attribute called `file` that specifies the name of the file to be included.

14.What are the JSP standard actions?

- The JSP standard actions affect the overall runtime behavior of a JSP page and also the response sent back to the client.
- They can be used to include a file at the request time, to find or instantiate a JavaBean, to forward a request to a new page, to generate a browser-specific code, etc.
- Ex: `include, forward, useBean,etc.` object

15.What are the standard actions available in JSP?

The standard actions available in JSP are as follows:

- **<jsp:include>**: It includes a response from a servlet or a JSP page into the current page. It differs from an include directive in that it includes a resource at request processing time, whereas the include directive includes a resource at translation time.

- **<jsp:forward>**: It forwards a response from a servlet or a JSP page to another page.
- **<jsp:useBean>**: It makes a JavaBean available to a page and instantiates the bean.
- **<jsp:setProperty>**: It sets the properties for a JavaBean.
- **<jsp:getProperty>**: It gets the value of a property from a JavaBean component and adds it to the response.
- **<jsp:param>**: It is used in conjunction with <jsp:forward>;, <jsp:, or plugin>; to add a parameter to a request. These parameters are provided using the name-value pairs.
- **<jsp:plugin>**: It is used to include a Java applet or a JavaBean in the current JSP page.

16.What is the <jsp:useBean> standard action?

The `<jsp:useBean>` standard action is used to locate an existing JavaBean or to create a JavaBean if it does not exist. It has attributes to identify the object instance, to specify the lifetime of the bean, and to specify the fully qualified classpath and type.

17.What are the scopes available in <jsp:useBean>?

The scopes available in <jsp:useBean> are as follows:

- **page scope:**: It specifies that the object will be available for the entire JSP page but not outside the page.

- **request scope**: It specifies that the object will be associated with a particular request and exist as long as the request exists.
- **application scope**: It specifies that the object will be available throughout the entire Web application but not outside the application.
- **session scope**: It specifies that the object will be available throughout the session with a particular client.

18.What is the <jsp:forward> standard action?

- The `<jsp:forward>` standard action forwards a response from a servlet or a JSP page to another page.
- The execution of the current page is stopped and control is transferred to the forwarded page.
- The syntax of the <jsp:forward> standard action is :
  `<jsp:forward page="/targetPage" />`
  Here, targetPage can be a JSP page, an HTML page, or a servlet within the same context.

- If anything is written to the output stream that is not buffered before `<jsp:forward>`, an IllegalStateException will be thrown.

Note : Whenever we intend to use <jsp:forward> or <jsp:include> in a page, buffering should be enabled. By default buffer is enabled.

19.What is the <jsp:include> standard action?

The <jsp:include> standard action enables the current JSP page to include a static or a dynamic resource at runtime. In contrast to the include directive, the include action is used for resources that change frequently. The resource to be included must be in the same context.The syntax of the <jsp:include> standard action is as follows:

```
<jsp:include page="targetPage" flush="true"/>
```

Here, targetPage is the page to be included in the current JSP.

20.What is the difference between include directive and include action?

Include directive	Include action
The *include* directive, includes the content of the specified file during the translation phase–when the page is converted to a servlet.	The *include* action, includes the response generated by executing the specified page (a JSP page or a servlet) during the request processing phase–when the page is requested by a user.
The include directive is used to statically insert the contents of a resource into the current JSP.	The include standard action enables the current JSP page to include a static or a dynamic resource at runtime.
Use the include directive if the file changes rarely. It’s the fastest mechanism.	Use the include action only for content that changes often, and if which page to include cannot be decided until the main page is requested.

21.Differentiate between pageContext.include and jsp:include?

The `<jsp:include>` standard action and the `pageContext.include()` method are both used to include resources at runtime. However, the `pageContext.include()` method always flushes the output of the current page before including the other components, whereas `<jsp:include>` flushes the output of the current page only if the value of flush is explicitly set to true as follows:

```
<jsp:include page="/index.jsp" flush="true"/>
```

22.What is the jsp:setProperty action?

You use `jsp:setProperty` to give values to properties of beans that have been referenced earlier. You can do this in two contexts. First, you can use `jsp:setProperty` after, but outside of, a `jsp:useBean` element, as below:

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName" property="myProperty" ... />
```

In this case, the jsp:setProperty is executed regardless of whether a new bean was instantiated or an existing bean was found.

A second context in which `jsp:setProperty` can appear is inside the body of a `jsp:useBean` element, as below:

```
<jsp:useBean id="myName" ... >
  ...
  <jsp:setProperty name="myName"
                   property="someProperty" ... />
</jsp:useBean>
```

Here, the jsp:setProperty is executed only if a new object was instantiated, not if an existing one was found.

23.What is the jsp:getProperty action?

The <jsp:getProperty> action is used to access the properties of a bean that was set using the <jsp:getProperty> action. The container converts the property to a String as follows:

- If it is an object, it uses the toString() method to convert it to a String.
- If it is a primitive, it converts it directly to a String using the valueOf() method of the corresponding Wrapper class.
- The syntax of the <jsp:getProperty> method is: `<jsp:getProperty name="Name" property="Property" />`

Here, name is the id of the bean from which the property was set. The property attribute is the property to get. A user must create or locate a bean using the <jsp:useBean> action before using the <jsp:getProperty> action.

24.What is the <jsp:param> standard action?

The <jsp:param> standard action is used with <jsp:include> or <jsp:forward> to pass parameter names and values to the target resource. The syntax of the <jsp:param> standard action is as follows:

```
<jsp:param name="paramName" value="paramValue"/>
```

25.What is the jsp:plugin action ?

This action lets you insert the browser-specific `OBJECT` or `EMBED` element needed to specify that the browser run an applet using the Java plugin.

26.What are scripting elements?

JSP scripting elements let you insert Java code into the servlet that will be generated from the current JSP page. There are three forms:

1. **Expressions** of the form `<%= expression %>` that are evaluated and inserted into the output,
2. **Scriptlets** of the form `<% code %>` that are inserted into the servlet's service method,
3. **Declarations** of the form `<%! code %>` that are inserted into the body of the servlet class, outside of any existing methods.

27.What is a scriptlet?

A scriptlet contains Java code that is executed every time a JSP is invoked. When a JSP is translated to a servlet, the scriptlet code goes into the `service()` method. Hence, methods and variables written in scriptlets are local to the `service()` method. A scriptlet is written between the **<% and %>** tags and is executed by the container at request processing time.

28.What are JSP declarations?

As the name implies, JSP declarations are used to declare class variables and methods in a JSP page. They are initialized when the class is initialized. Anything defined in a declaration is available for the whole JSP page. A declaration block is enclosed between the **<%! and %>** tags. A declaration is not included in the `service()` method when a JSP is translated to a servlet.

29.What is a JSP expression?

A JSP expression is used to write an output without using the `out.print statement`. It can be said as a shorthand representation for scriptlets. An expression is written between the **<%= and %>** tags. It is not required to end the expression with a semicolon, as it implicitly adds a semicolon to all the expressions within the expression tags.

30.How is scripting disabled?

Scripting is disabled by setting the scripting-invalid element of the deployment descriptor to true. It is a subelement of jsp-property-group. Its valid values are true and false. The syntax for disabling scripting is as follows:

```
<jsp-property-group>
   <url-pattern>*.jsp</url-pattern>
   <scripting-invalid>true</scripting-invalid>
</jsp-property-group>
```