D. Y. Patil Pratishthan's

Institute for Advanced Computing and Software Development

IACSD

Advanced Java Notes

Dr. D.Y. Patil Educational Complex Sector 29,Near Akurdi Railway Station,
Nigdi Pradhikaran,Akurdi,Pune-44

Introduction to Enterprise Java

Web application

The application which can be developed using web components like **HTML**, **JSP**, **JS**, **XML**, **Servlet** etc. is called web application. A web application is running on particular server.

Client-Server Architecture in java

Client

The media which is use to send request on server is called Client and receive request from server is called client.Our browser will act as client.

Server

The media which is use to receive request from client and give response to client is called Server.
The main purpose of the server is to pick up the request from client identify the requested and dispatch dynamic response to client.

Protocol

The main job of protocol in client-server architecture is to carry the request data from client to server and carry response from server to client.
e.g. TCP**P/IP,HTTP,FTP,SMTP etc..**

What is HTTP

HTTP stands for hyper text transfer protocol which is use to exchange data over the web.
HTTP is stateless protocol means every request treated as new request on server.
HTTP unable to manage clients previous request data.

Examples of Servers

1. Apache Tomcat
2. Weblogic
3. Websphere
4. Glashfish

What is Web Server?

Web server is server which will provide environment to execute web application only like Servlet, JSP ,Html … etc.

What is Application Server

Application server is a server that will provide environment to execute both web application and distributed application like Servlet, JSP and EJB etc…

What is Web Container?

It provides run time environment for **JEE** applications.

Web Container will perform following action

1. Life cycle management
2. Multi threaded support

3. Security
4. It support Connection pooling, Transaction Management, messaging, clustering, load balancing and persistence. etc…

Servlet

A servlet is a Java™ technology-based Web component, managed by a container that generates dynamic content. Like other Java technology-based components, servlets are platform-independent Java classes that are compiled to platform-neutral byte code that can be loaded dynamically into and run by a Java technology-enabled Web server. Containers, sometimes called servlet engines, are Web server extensions that provide servlet functionality. Servlets interact with Web clients via a request/response paradigm implemented by the servlet container. The servlet API is a standard Java extension and is therefore portable and platform-neutral. Servlets are secure, since they operate within the context of a security manager.

What is a Servlet Container?

The servlet container is a part of a Web server or application server that provides the network services over which requests and responses are sent. A servlet container also contains and manages servlets through their lifecycle. All servlet containers must support HTTP as a protocol for requests and responses, with additional protocols such as HTTPS (HTTP over SSL) etc… A servlet container may place security restrictions on the environment in which a servlet executes.

HTTP Basics

Servlets use the HTTP protocol, so a basic understanding of the protocol is assumed when using servlets.

- Requests, Responses, and Headers
- HTTP is a simple, stateless protocol. A client, such as a web browser, makes a request, the web server responds, and the transaction is done.
- When the client sends a request, the first thing it specifies is an HTTP command, called a method, that tells the server the type of action it wants performed.
- This first line of the request also specifies the address of a document (a URL) and the version of the HTTP protocol it is using. For example:
  GET /intro.html HTTP/1.0
- This request uses the GET method to ask for the document named intro.html, using HTTP Version 1.0.
- After sending the request, the client can send optional header information to tell the server extra information about the request, such as what software the client is running and what content types it understands.
- After the client sends the request, the server processes it and sends back a response. The first line of the response is a status line that specifies the version of the HTTP protocol the server is using, a status

code, and a description of the status code.For example:HTTP/1.0 200 OK. This status line includes a status code of 200, which indicates that the request was successful, hence the description "OK". Another common status code is 404, with the description "Not Found"—as you can guess, this means that the requested document was not found.

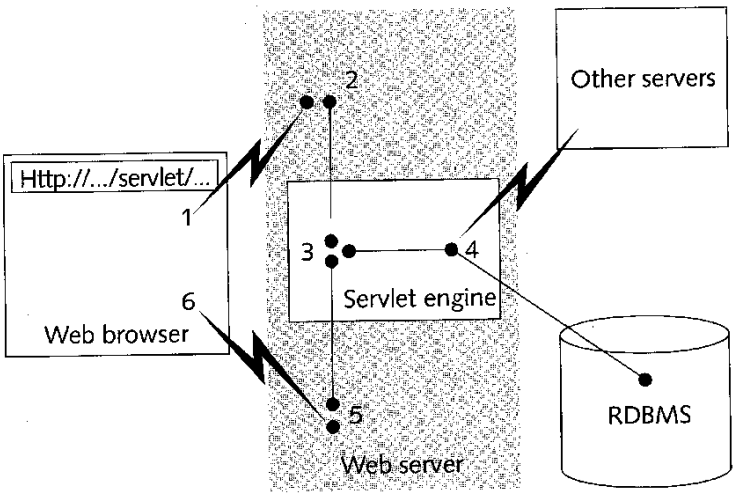
GET and POST methods

The GET and POST methods are used to get information and upload information to the webserver respectively. GET actually has the ability to pass information to the server and because of this is often used incorrectly. Many servers will limit the size of the url for a GET request to only 240 characters.POST is used for sending (posting) information to the server. It generally has an unlimited length.GET and POST are generally the only methods we will be dealing with in this unit, and generally speaking, are the only two methods most developers need.

Other methods

- In addition to GET and POST, there are several other lesser-used HTTP methods.
- There's the HEAD method, which is sent by a client when it wants to see only the headers of the response, to determine the document's size, modification time, or general availability.
- There's also PUT, to place documents directly on the server, and
- DELETE, to do just the opposite. These last two aren't widely supported due to complicated policy issues.
- The TRACE method is used as a debugging aid—it returns to the client the exact contents of its request.
- Finally, the OPTIONS method can be used to ask the server which methods it supports or what options are available for a particular resource on the server.

How a request is handled ?



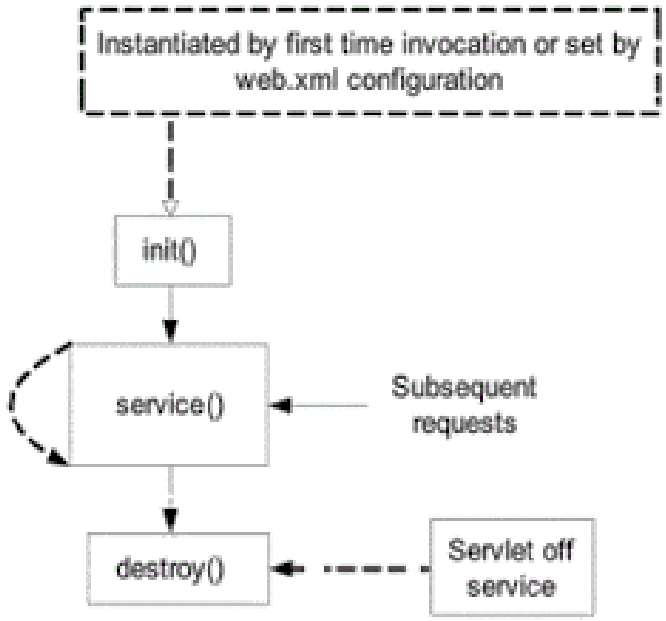
1. A client (e.g., a Web browser) accesses a Web server and makes an HTTP request.
2. The request is received by the Web server and handed off to the servlet container.
3. Note : The servlet container can be running in the same process as the host Web server, in a different process on the same host, or on a different host from the Web server for which it processes requests.
4. The servlet container determines which servlet to invoke based on the configuration of its servlets, and calls it with objects representing the request and response.
5. The servlet uses the request object to find out who the remote user is, what HTTP POST parameters may have been sent as part of this request, and other relevant data. The servlet performs whatever logic it was programmed with, and generates data to send back to the client. It sends this data back to the client via the response object.
6. Once the servlet has finished processing the request, the servlet container ensures that the response is properly flushed, and returns control back to the host Web server.

The Servlet Interface

Servlet interface is the central abstraction of the Java Servlet API. All servlets implement this interface either directly, or more commonly, by extending a class that implements the interface. The two classes in the Java Servlet API that implement the Servlet interface are GenericServlet and HttpServlet. For most purposes, Developers will extend HttpServlet to implement their servlets.

Servlet Life Cycle

A servlet is managed through a well defined life cycle that defines how it is loaded and instantiated, is initialized, handles requests from clients, and is taken out of service. This life cycle is expressed in the API by the init, service, and destroy methods of the javax.servlet.Servlet interface that all servlets must implement directly or indirectly through the GenericServlet or HttpServlet abstract classes.



- **Loading and Instantiation**
  The servlet container is responsible for loading and instantiating servlets. The loading and instantiation can occur when the container is started, or delayed until the container determines the servlet is needed to service a request.
  When the servlet engine is started, needed servlet classes must be located by the servlet container. The servlet container loads the servlet class.After loading the Servlet class, the container instantiates it for use.
- **Initialization**
  After the servlet object is instantiated, the container must initialize the servlet before it can handle requests from clients. Initialization is provided so that a servlet can read persistent configuration data, initialize resources and perform other one-time activities. The container initializesthe servlet instance by calling the init method of the Servlet interface with aunique (per servlet declaration) object implementing the ServletConfig interface.
- **Request Handling**
  After a servlet is properly initialized, the servlet container may use it to handle client requests. Requests are represented by request objects of type ServletRequest. The servlet fills out response to requests by calling methods of a provided object of type ServletResponse. These objects are passed as parameters to the service method of the Servlet interface.
  In the case of an HTTP request, the objects provided by the container are of types HttpServletRequest and HttpServletResponse.
- **End of Service**
  The servlet container is not required to keep a servlet loaded for any particular period of time. When the servlet container determines that a servlet should be removed from service, it calls the destroy method of the Servlet interface to allow the servlet to release any resources it is using and save any persistent state. After the destroy method completes, the servlet container must release the servlet instance so that it is eligible for garbage collection.

What is Request?

The request object encapsulates all information from the client request. In the HTTP protocol, this information is transmitted from the client to the server in the HTTP headers and the message body of the request.

Request parameters for the servlet are the strings sent by the client to a servletcontainer as part of its request. The container populates the parameters from the URI query string and POST-ed data. The parameters are stored as a set of name-value pairs. The following methods of theServletRequest interface are available to access parameters:

- getParameter
- getParameterNames

- getParameterValues
- getParameterMap

- **Headers**

A servlet can access the headers of an HTTP request through the following methods of the HttpServletRequest interface:

- getHeader
- getHeaders
- getHeaderNames
-

- **Attributes**

Attributes are objects associated with a request. Attributes may be set by thecontainer to express information that otherwise could not be expressed via the API,or may be set by a servlet to communicate information to another servlet (via theRequestDispatcher). Attributes are accessed with the following methods of theServletRequest interface:

- getAttribute
- getAttributeNames
- setAttribute

- **Request Path Elements**

The request path that leads to a servlet servicing a request is composed of manyimportant sections. The following elements are obtained from the request URI path:

- **Context Path**: The path prefix associated with the ServletContext that this servlet is a part of.
- **Servlet Path**: The path section that directly corresponds to the mapping which activated this request.
- **PathInfo**: The part of the request path that is not part of the Context Path or the Servlet Path.

The following methods exist in the HttpServletRequest interface to access thisinformation:

- getContextPath
- getServletPath
- getPathInfo

requestURI = contextPath + servletPath + pathInfo

- **Lifetime of the Request Object**

Each request object is valid only within the scope of a servlet's service method, or within the scope of a filter's doFilter method, unless the asynchronous processing is enabled for the component. Containers commonly recycle request objects in order to avoid the performance overhead of request object creation.

HTTP request types in servlets

The GET and POST requests are to be handled by the doGet and doPost methods of your servlet class that extends the abstract HttpServlet class. These methods are automatically called by the HttpServlet class's service method, which is called when a request arrives at the server. The method service performs the following steps in order:

- determines the request type
- calls the appropriate method (doGet or doPost)

There is no need for you to call the service method in your servlets.Every call to doGet and doPost for an HttpServlet receives an object that implements the interface HttpServletRequest. This object contains the request from the client. A variety of methods are available in this interface and the ServletRequest interface (super-interface of HttpServletRequest) to enable the servlet to process the client's request,

- getContextPath()
- Cookie[] getCookie()
- HTTPSessiongetSession()
- String getParameter(String name)

The web server that executes the servlet creates an HttpServletResponse object. The web server passes the HttpServletResponse object to the servlet's service method (which, in turn, passes it to doGet or doPost). This object contains the response to the client. A variety of methods are available in this interface and the ServletResponse (super-interface of HttpServletResponse) to formulate the response to the client, egs

- ServletOutputStreamgetOutputStream()
- PrintWritergetWriter()
- void setContentType(String type)

Apart from the doGet and doPost methods, the abstract class HttpServlet also provides methods such as doDelete (for client to remove documents from the server) and doPut (for client to place documents onto the server).

ServletContext

The ServletContext interface defines a servlet's view of the Web application within which the servlet is running. The Container Provider is responsible for providing animplementation of the ServletContext interface in the servlet container. Using theServletContext object, a servlet can log events, obtain URL references to resources,and set and store attributes that other servlets in the context can access.A ServletContext is rooted at a known path within a Web server. ServletContext object is global to entire web application. Object of ServletContext will be created at the time of web application deployment. It's *Scopeis a*s long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server.

Context Attributes

A servlet can bind an object attribute into the context by name. Any attribute bound into a context is available to any other servlet that is part of the same Web application. The following methods of ServletContext interface allow access to this functionality:

- setAttribute
- getAttribute
- getAttributeNames
- removeAttribute

ServletConfig

Servlet Container creates ServletConfig object for each Servlet during initialization, to pass information to the Servlet. This object can be used to get configuration information such as parameter name and values from deployment descriptor file(web.xml).ServletConfig object is one per servlet class.Object of ServletConfig will be created during initialization process of the servlet. This Config object is public to a particular servlet only It's Scope is as long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet execution is completed.We should give request explicitly, in order to create ServletConfig object for the first time.

The following methods of ServletConfiginterface allow access to the functionality:

- **getInitParameter**
- **getServletName**
- **getServletContext**

What is Response?

The response object encapsulates all information to be returned from the server to the client. In the HTTP protocol, this information is transmitted from the server to the client either by HTTP headers or the message body of the request.

Lifetime of the Response Object

Each response object is valid only within the scope of a servlet's service method, or within the scope of a filter's doFilter method. Containers commonly recycle response objects in order to avoid the performance overhead of response object creation.

Sessions

The Hypertext Transfer Protocol (HTTP) is by design a stateless protocol. To buildeffective Web applications, it is imperative that requests from a particular client beassociated with each other. Many strategies for session tracking have evolved overtime. The following are some approaches for tracking a user's sessions. The standard name of the session tracking cookie must be JSESSIONID. Containers may allow the name of the session tracking cookie to be customized through container specific configuration.

- **Using 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. A cookie's value can uniquely identify a client, so cookies are commonly used for session management. Cookie is a key value pair of information.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number. The servlet sends cookies to the browser by usingthe addCookie() method, which adds fields to HTTP response headers to send cookies to the browser, one at a time. The browser is expected to support 20 cookies for each Web server, 300 cookies total, and may limit cookie size to 4 KB each.The browser returns cookies to the servlet by adding fields to HTTP request headers. Cookies can be retrieved from a request by using the getCookies() method. Several cookies might have the same name but different path attributes.

This should be saved by the browser in its space in the client computer. Whenever the browser sends a request to that server it sends the cookie along with it. Then the server can identify the client using the cookie.

In java, following is the source code snippet to create a cookie:

```
Cookie cookie = new Cookie("userID", "7456");
res.addCookie(cookie);
```

Session tracking is easy to implement and maintain using the cookies. Disadvantage is that, the users can opt to disable cookies using their browser preferences. In such case, the browser will not save the cookie at client computer and session tracking fails.

- **Hidden Fields**

<INPUT TYPE="hidden" NAME="technology" VALUE="servlet">

Hidden fields like the above can be inserted in the webpages and information can be sent to the server for session tracking. These fields are not visible directly to the user, but can be viewed using view source option from the browsers. This type doesn't need any special configuration from the browser of server and by default available to use for session tracking. This cannot be used for session tracking when the conversation included static resources like html pages.

- **URL Rewriting**

Original URL: http://server:port/servlet/ServletName

Rewritten URL: http://server:port/servlet/ServletName?sessionid=1234

When a request is made, additional parameter is appended with the URL. In general added additional parameter will be sessionid or sometimes the userid. It will suffice to track the session. This type of session tracking doesn't need any special support from the browser. Disadvantage is, implementing this type of session tracking is tedious. We need to keep track of the parameter as a chain link until the conversation completes and also should make sure that, the parameter doesn't clash with other application parameters.

- **Session tracking API**

Session tracking API is built on top of the first three methods. This is inorder to help the developer to minimize the overhead of session tracking. This type of session tracking is provided by the underlying technology. Lets take the java servlet example. Then, the servlet container manages the session tracking

task and the user need not do it explicitly using the java servlets. This is the best of all methods, because all the management and errors related to session tracking will be taken care of by the container itself.
Every client of the server will be mapped with a javax.servlet.http.HttpSession object. Java servlets can use the session object to store and retrieve java objects across the session. Session tracking is at the best when it is implemented using session tracking api.

How to access HttpSession object?

Every request is associated with an HttpSession object. It can be retrieved using getSession(booleancreateFlag) available in HttpServletRequest. It returns the current HttpSession associated with this request or, if there is no current session and create is true, and then returns a new session. A session can be uniquely identified using a unique identifier assigned to this session, which is called session id. getId() gives you the session id as String.
Method isNew()returns true if the client does not know about the session or if the client chooses not to join the session. getCreationTime() returns the time when this session was created. getLastAccessedTime() returns the last time the client sent a request associated with this session.

How to store data in session?

Once you have got access to a session object, it can be used as a HashTable to store and retrieve values. It can be used to transport data between requests for the same user and session. setAttribute(String name, Object value) adds an object to the session, using the name specified. Primitive data types cannot be bound to the session.
Note that your object needs to implement Serializable interface if you are going to store it in session and carry it over across different web servers.

How to retrieve data from session?

getAttribute(String name) returns the object bound with the specified name in this session.

How to invalidate a session object?

By default every web server will have a configuration set for expiry of session objects. Generally it will be some xxxx seconds of inactivity. That is when the user has not sent any request to the server for the past xxxx seconds then the session will expire. When a session expires, the HttpSession object and all the data it contains will be removed from the system. When the user sends a request after the session has expired, server will treat it as a new user and create a new session.
Apart from that automatic expiry, it can also be invalidated by the user explicitly. HttpSession provides a method invalidate() this unbinds the object that is bound to it. Mostly this is used at logout. Or the application can have an absurd logic like after the user logins he can use the application for only 30 minutes. Then he will be forced out. In such scenario you can use getCreationTime().

JSP

JavaServer Pages (JSP) technology enables you to mix regular, static HTML with dynamically generated content from servlets. You simply write the regular HTML in the normal manner. Separating the static HTML from the dynamic content provides a number of benefits over servlets alone, and the approach used in JavaServer Pages offers several advantages over competing technologies such as ASP, PHP.

JSP is widely supported and thus doesn't lock you into a particular operating system or Web server and that JSP gives you full access to servlet and Java technology for the dynamic part, rather than requiring you to use an unfamiliar and weaker special-purpose language.

Although what you write often looks more like a regular HTML file than a servlet, behind the scenes, the JSP page is automatically converted to a normal servlet, with the static HTML simply being printed to the output stream associated with the servlet's service method. This translation is normally done the first time the page is requested.

Aside from the regular HTML, there are three main types of JSP constructs that you embed in a page: scripting elements, directives, and actions.

Scripting Elements [Sciptlets]

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

1. Expressions of the form <%= expression %>, which are evaluated and inserted into the servlet's output
2. Scriptlets of the form <% code %>, which are inserted into the servlet's _jspService method (called by service)
3. Declarations of the form <%! code %>, which are inserted into the body of the servlet class, outside of any existing methods.
4. Scriptlets have access to the same automatically defined variables as expressions (request, response, session, out, etc). So, for example, if you want output to appear in the resultant page, you would use the out variable, as in the following example.

```
<%
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
%>
```

JSP Expressions

A JSP expression is used to insert values directly into the output. It has the following form:

```
<%= Java Expression %>
```

The expression is evaluated, converted to a string, and inserted in the page. For example, the following shows the date/time that the page was requested:

Current time: <%= new java.util.Date() %>.

JSP Declarations

A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class (outside of the _jspService method that is called by service to process the request). A declaration has the following form:

<%! Java Code %>

For example, a JSPfragment that prints the number of times the current page has beenrequested since the server was,

```
<%! privateintaccessCount = 0; %>
Accesses to page since server reboot:
<%= ++accessCount %>
```

Predefined Variables

To simplify code in JSP expressions and scriptlets, you are supplied with some automatically defined variables, sometimes called implicit objects.

- **request**

  This variable is the HttpServletRequest associated with the request;it gives you access to the request parameters, the request type (e.g., GET or POST), and the incoming HTTP headers (e.g., cookies).
- **response**

  This variable is the HttpServletResponse associated with the response to the client.
- **out**

  This is the PrintWriter used to send output to the client. However, to make the response object useful, this is a buffered version of Print- Writer called JspWriter.
- **session**

  This variable is the HttpSession object associated with the request. Recall that sessions are created automatically, so this variable is bound even if there is no incoming session reference.
- **application**

  This variable is the ServletContext as obtained via getServletConfig().getContext().
- **config**

  This variable is the ServletConfig object for this page.
- **pageContext**

  JSP introduced a new class called PageContext to give a single point of access to many of the page attributes and to provide a convenient place to store shared data.
- **page**

  This variable is simply a synonym for this

JSP directives

JSP directive affects the overall structure of the servlet that results from the JSP page. In JSP, there are three types of directives: page, include, and taglib. The page directive lets you control the structure of the servlet by importing classes, customizing the servlet superclass, setting the content type, and the like.

Page Directive

The page directive lets you define one or more of the following case-sensitive attributes: import, contentType, isThreadSafe, session, buffer, autoflush, extends, info, errorPage, isErrorPage, and language. These attributes are explained below,

1. The import Attribute

The import attribute of the page directive lets you specify the packages thatshould be imported by the servlet into which the JSP page gets translated.

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

2. The contentType Attribute

The contentType attribute sets the Content-Type response header, indicating the MIME type of the document being sent to the client.

e.g.<%@ page contentType="text/plain" %>

3. The isThreadSafe Attribute

The isThreadSafe attribute controls whether or not the servlet that results from the JSP page will implement the SingleThreadModel interface. The default value is true, which means that the system assumes you made your code thread safe.

e.g.<%@ page isThreadSafe="true" %>

4. The session Attribute

The session attribute controls whether or not the page participates in HTTP sessions. Use of this attribute takes one of the following two forms:

<%@ page session="true" %>

A value of true (the default) indicates that the predefined variable session (of type HttpSession) should be bound to the existing session if one exists; otherwise, a new session should be created and bound to session.

5. The buffer Attribute

The buffer attribute specifies the size of the buffer used by the out variable, which is of type JspWriter (a subclass of PrintWriter).

e.g.<%@ page buffer="sizekb" %>

6. The autoflush Attribute

The autoflush attribute controls whether the output buffer should be automatically flushed when it is full or whether an exception should be raised when the buffer overflows

e.g.<%@ page autoflush="true" %>

7. The extends Attribute

The extends attribute indicates the superclass of the servlet that will be generated for the JSP page

e.g. <%@ page extends="package.class" %>

8. The info Attribute

The info attribute defines a string that can be retrieved from the servlet by means of the getServletInfo method.

e.g.<%@ page info="Some Message" %>

9. The errorPage Attribute

The errorPage attribute specifies a JSP page that should process any exceptions (i.e., something of type Throwable) thrown but not caught in the current page. It is used as follows:

<%@ page errorPage="Relative URL" %>

The exception thrown will be automatically available to the designated error page by means of the exception variable.

10. The isErrorPage Attribute

The isErrorPage attribute indicates whether or not the current page can act as the error page for another JSP page.

e.g.<%@ page isErrorPage="true" %>

Using JavaBeans with JSP

JavaBean

JavaBeans are reusable software components for Java that can be manipulated visually in a builder tool. Practically, they are classes written in the Java programming language conforming to a particular convention. They are used to encapsulate many objects into a single object (the bean), so that they can be passed around as a single bean object instead of as multiple individual objects. A JavaBean is

Defined with following standards,

- A bean class must have a zero-argument (empty) constructor.
- A bean class should have no public instance variables (fields).
- Persistent values should be accessed through methods called getXxx and setXxx.

Basic Bean Use

The jsp:useBean action lets you load a bean to be used in the JSP page.Beans provide a very useful capability because they let you exploit the reusability of Java classes.

The simplest syntax using bean is:

<jsp:useBean id="name" class="package.Class" />

This usually means “instantiate an object of the class specified by Class, and bind it to a variable with the name specified by id.”Even you can specify a scope attribute that makes the bean associated with more than just the current page.

Accessing Bean Properties

Once you have a bean, you can access its properties with jsp:getProperty, which takes a name attribute that should match the id given in jsp:useBean and a property attribute that names the property of interest.

e.g.
```
<jsp:useBean id="book1" class="com.MyBook" />
    <jsp:getProperty name="book1" property="title" />
    <%= book1.getTitle() %>
```

Setting Bean Properties

To modify bean properties, you normally use jsp:setProperty.

- *To set indivisual specific property*

```
<jsp:setProperty
name="book1"
property="title"
value='<%= request.getParameter("title") %>' />
```

- *To set all properties or Associating All Properties with Input Parameters*

```
<jsp:setProperty name="book1" property="*" />
```

Sharing Beans

Although the beans are indeed bound to local variables, that is not the only behavior. They are also stored in one of four different locations, depending on the value of the optional scope attribute of jsp:useBean. The scope attribute has the following possible values:

- **Page**

This is the default value. It indicates that, in addition to being bound to a local variable, the bean object should be placed in the PageContext object for the duration of the current request.Beans created with page scope are almost always accessed by jsp:getProperty, jsp:setProperty, scriptlets, or expressions later in the same page.

- **application**

This very useful value means that, in addition to being bound to a local variable, the bean will be stored in the shared ServletContext available through the predefined application variable or by a call to getServletContext().Values in the ServletContext can be retrieved by the getAttribute method.

- **session**

This value means that, in addition to being bound to a localvariable, the bean will be stored in the HttpSession objectassociated with the current request, where it can be retrievedwith getValue.

- **request**

This value signifies that, in addition to being bound to a localvariable, the bean object should be placed in theServletRequest object for the duration of the current request.

JSTL tutorial examples

JSTL stands for JSP Standard Tag Library and JSTL represents a set predefined tags.The main purpose of JSTL is to simplify the JSP development. JSTL is not language like JSP it's a tag library by which we can write java program in JSP, like for loop , if else statement etc.

Advantages of JSTL

1. Fast development.
2. Easy understandable for Humans.
3. No need to use scriptlet tag in JSP page.

How to use JSTL in JSP?

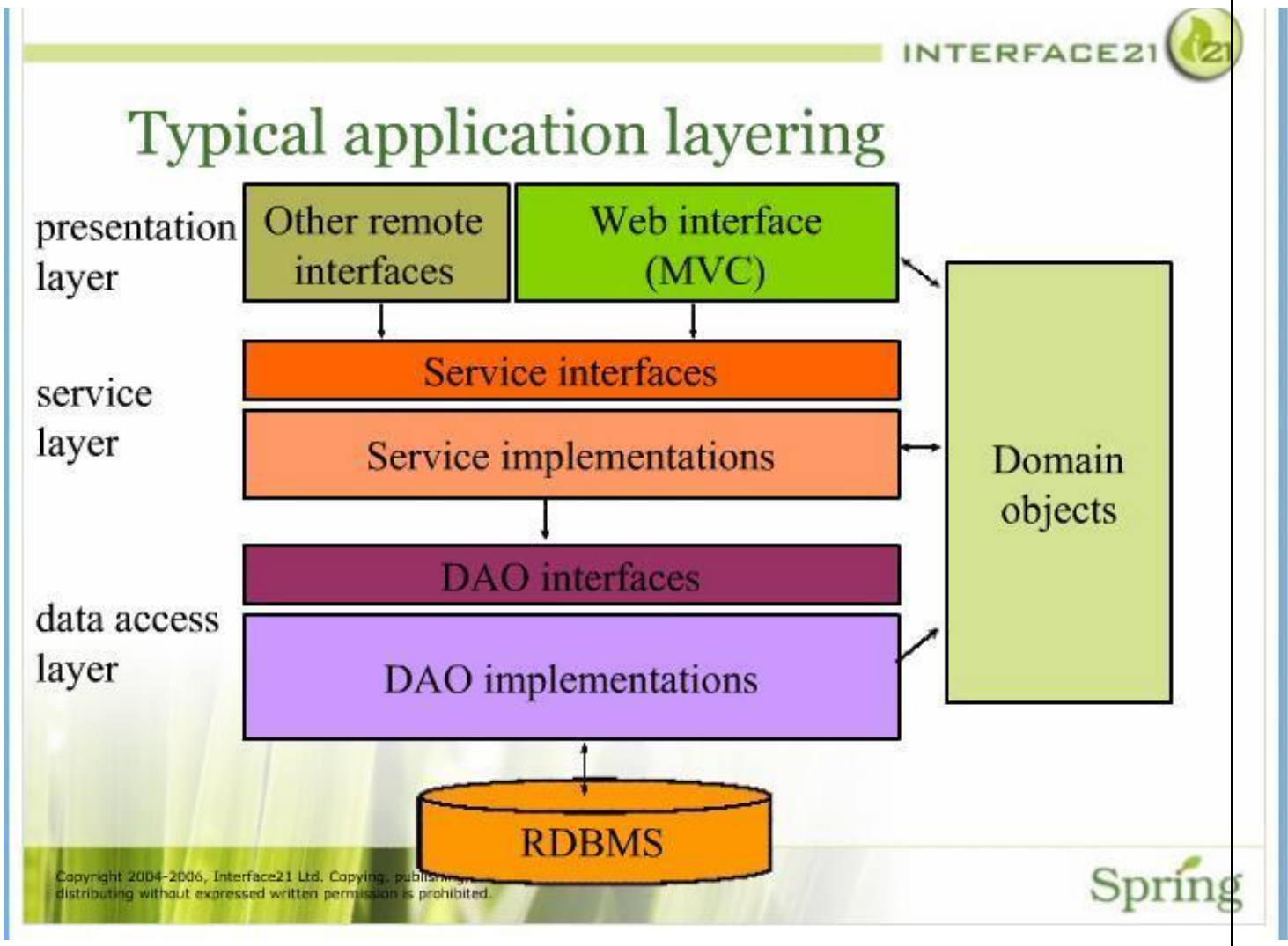
To use JSTL in our JSP pages, we need to download the **JSTL** jars and we need to add jar into WEB-INF/lib directory.JSTL has some JSTL Core Tags and JSTL Functions. Before writing any JSTL Core tag or JSTL function into JSP page we need to add tag library into JSP page Header section.

JSTL Core Tags library

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

JSTL Functions Tag library

```
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```


INTERFACE21
Typical application layering
presentation layer
Other remote interfaces
Web interface (MVC)
service layer
Service interfaces
Service implementations
Domain objects
DAO interfaces
data access layer
DAO implementations
RDBMS
Copyright 2004-2006, Interface21 Ltd. Copying, publishing, distributing without expressed written permission is prohibited.
Spring

Hibernate

Hibernate is an object relational mapper (**ORM**) Framework which is used to turn our java objects into a way to persists them into the database and vice versa.

What is ORM

Object-Relational Mapping (**ORM**) is a technique which provides us facility to query and manipulate data from a database using an object-oriented programming pradigm.

ORM binds our tables or stored procedures in java classes, so that instead of writing SQL statements to interact with your database, you use methods and properties of objects.

Advantage of Hibernate Framework

1. Hibernate is non-invasive framework means it does not force use to extend or implement API classes or interfaces.
2. Simple to learn and work means no need to learn complicated SQL queries.
3. Hibernate is a bit faster than JDBC queries.
4. By default transaction support is given.
5. Database independent query
6. Light-weight framework.

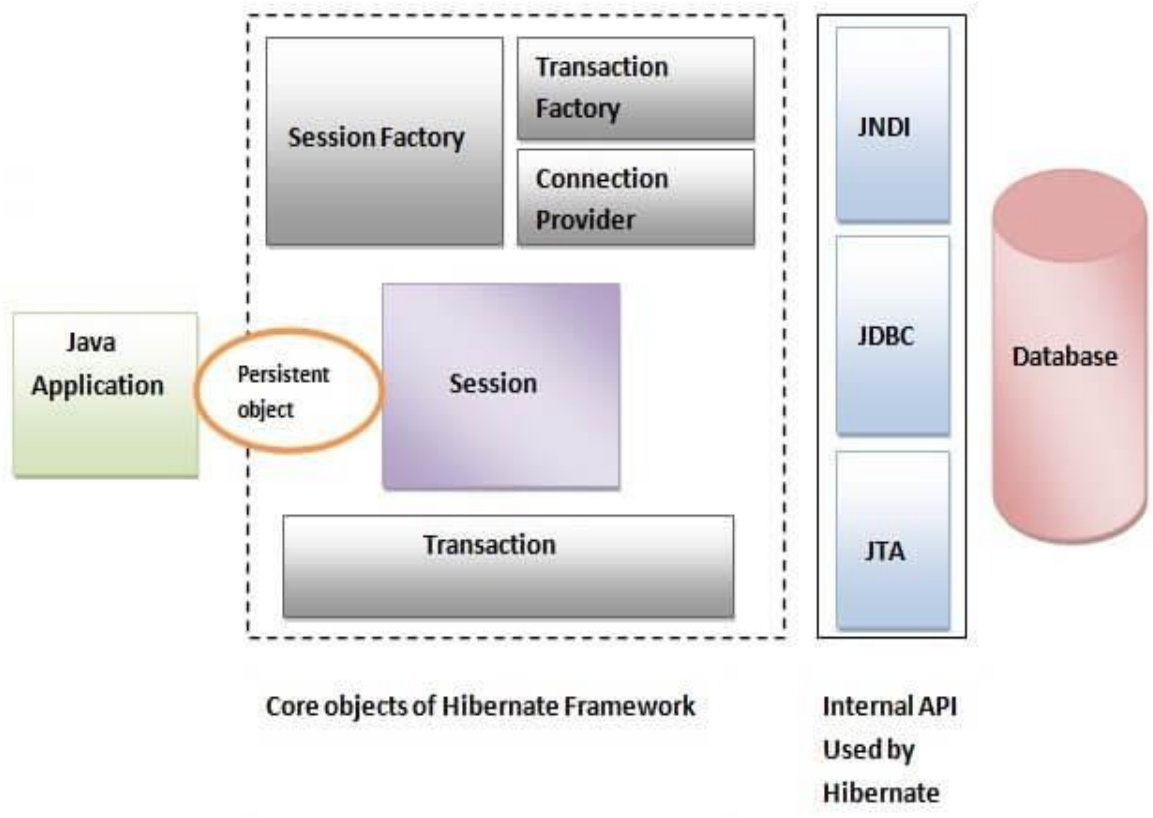
Limitation of Hibernate

1. Hibernate is slower than pure JDBC driver API.
2. Hibernate is not recommended for small project.
3. Hibernate is not suitable for Batch processing
4. So Many configurations needed for simple query also.

Hibernate Architecture

Elements of Hibernate Architecture

1. Configuration
2. SessionFactory
3. Session
4. Transaction
5. Query



What is Configuration?

Configuration is a class given by hibernate people to load the hibernate XML configuration file.

```
Configuration cfg = new Configuration();
```

What is SessionFactory?

SessionFactory is an interface which is used to create the session object and SessionFactory contains the connection information, hibernate configuration information and mapping files,location path.Instances of SessionFactory are *thread-safe* and typically shared throughout an application.SessionFactory holds second level cache also.

```
SessionFactorysessionFactory = configurationObject.buildSessionFactory();
```

What is Session?

Session is an interface which is used to execute SQL queries like insert , update , delete and it is holds first level cache.

What is Transaction?

Transaction is an interface using which we maintain transaction in hibernate application.

What is Query?

Query is an interface which is used to execute DML statement in hibernate application like select , delete etc.

Hibernate Mapping and Configuration file

Hibernate Mapping and Configuration file are heart of any **ORM** based framework application, here in hibernate application mapping is possible in two ways either using XML file or using Hibernate given annotation.

What is Hibernate Mapping file?

Hibernate mapping file is used to configure java persistent class details like class name, fields name etc.
In hibernate application our java class name will treated as table name for any database and fields name will be treated as column name of table.
We can construct one or more hibernate mapping files in hibernate application.
The naming convention of hibernate mapping file is **javaClassName.hbm.xml.**

Syntax

```
<hibernate-mapping>
        <class name="java _persistent_class" table="table_name">
                <!– Here id should be primary key of database –>
                <id name="field_name" column="database_column_name" type="Java_data_type" />
                <property name="field_name" column="database_column_name"type="Java_data_type"/>
        </class>
</hibernate-mapping>
```

What is Hibernate Configuration file?

Hibernate Configuration file is used to pass java JDBC driver name, driver class name, username and password and configuration file will be loaded file **Configuration** hibernate class in hibernate application.
Note that we have to pass hibernate mapping file into Hibernate Configuration file.

Syntax

```
<hibernate-configuration>
        <session-factory>
                <!– JDBC Connection info –>
                        <property name="connection_driver_class">Driver Class Name</property>
                        <property name="connection_url">URL </property>
                        <property name="connection_user">user </property>
                        <property name="connection_password">password</property>
                <!– Hibernate properties –>
                        <property name="show_sql">true/false</property>
                        <property name="Hibernate_Dialect">Hibernate Gievn dialect</property>
                        <property name="hbm2ddl.auto">create/update/delete/select</property>
                        <!– Hibernate mapping file –>
                        <mapping resource="mappingfilename.xml" / >
        </session-factory>
</hibernate-configuration>
```

Steps to create hibernate application in eclipse

To use hibernate framework in java application we need to follow some steps.We can use hibernate framework in both application in standalone application and in web application also, So steps is similar for both of the application but to add jar some differences are there like.

1. For java application we need to create **lib** folder and we need to place the all the hibernate jar files and then we have to configure build path.
2. And for web application we need to add hibernate jar files inside WEB-INF/lib folder and we no need to configure build path for web application.

Steps to create hibernate application

1. Create the java project
2. Add Hibernate jar files.
3. Create Persistent class inside any package.
4. Create the hibernate mapping file for Persistent class.
5. Create the Hibernate Configuration file for database information like driver name , URL username and password.
6. Create class to retrieves or stores java persistent object.

Step 1: Create java project using below steps
New -> Java Project -> Give name of project and press next and next.
Step 2: Create lib folder in created java project.
After creating lib folder add all hibernate application related jar files and configure build path.
Step 3: Create Persistent class
Step 4: Create hibernate mapping file
Step 5: Create hibernate configuration file
Step 6: Create Test Class having main method.

Hibernate SQL Dialects List

Hibernate Dialects is used to connect with specific database and it convert hibernate query language(**HQL**) todatabase specific query language.In simple word we can say Hibernate uses “Dialects ” configuration to know which database you are using so that it can switch to the database specific SQL generator code wherever/whenever necessary.

Hibernate Query Language

Hibernate Query Language (**HQL**) is a new language given by Hibernate people and it is powerful query language. **HQL** is fully object-oriented and understands notions like inheritance, polymorphism.

Why HQL came ?

As we know for every relational database will have their own query language which differs database to database so hibernate came up with new HQL language which is nothing but query language but it is not database dependent.We can say **HQL** is database independent query language.
HQL uses java class name as database table name and class fields name as table column name.
e.g. To fetch roll ,name from student table.

SQL in Oracle

```
>select roll , name from student;
```

HQL

```
selects.roll , s.name from Student s;
```

Here Student is class name and roll and name is fields name of Student class. HQL query will be executed with Query interface given by hibernate.

Hibernate Criteria API

Hibernate Criteria API is used to retrieve data from database based on some criteria like greater than , less than where clause.We can use Hibernate Criteria API for data retrieval purpose only.

Advantages of Hibernate Criteria API

By the help of **Criteria API** we can impose a lot of condition on SQL queries in easy way because Hibernate has given some predefined methods like gt() for greater than etc.

Disadvantages of Criteria API

Criteria API lags a lot in performance when we have huge amount of data to retrieve.

How to use Criteria API

We can get Criteria object by calling **createCriteria()** of session interface.

```
Criteria criteria = session.createCriteria(Entity class);
```

To impose any condition like greater than or less than we have to use **Restriction** interface with **Criterion**.

Syntax

```
Criterion c = Restrictions.gt("roll", new Integer(101));//greater than
```

Hibernate mapping

To reduce data redundancy in our database table we will divide the properties of one class into two classes and then we will apply relationship between objects of two classes is called **Hibernate mapping.**

In hibernate we can apply 4 types of relationship between two **POJO** classes.

1. One-to-Many
2. Many-to-One
3. Many-to-Many
4. One-to-One

In Hibernate if we want to apply relationship mapping between two classes then first we need to add Child class object to collection and then we need to set that collection object to Parent class.We can use List, Set or Map collection for mapping.

Hibernate Caching – First Level Cache

Hibernate supports three level of cache to optimize the performance of hibernate application.

1. First level cache
2. Second level cache
3. Query level cache

Why Hibernate support Cache ?

There are so many problems we have to face if we will not use any cache concept in our application like.

1. Performance issue we will get widely.
2. Network bandwidth issue we will.

When to use Hibernate cache

If same data is available around the application and if data will change very rarely then we can implement the cache concept in our application. But if data is changing frequently then we should not use cache.

What is Hibernate Caching – First Level Cache

Hibernate supports First Level Cache in the form of Session object and it is configured by default we can not disable first level cache but we can remove some of the object from Session object using some predefined method.

The scope of Hibernate First level cache is of per session object. Once session is closed cached objects are gone forever.

Hibernate second level cache

Hibernate Second Level cache will be created and associated with Hibernate SessionFactory and It will be available to in all sessions.

We can use Second level cache in all sessions within the application.Once Hibernate SessionFactoryobjectwillbe closed then all cached values will be gone.

Working of Hibernate second level cache

1. Hibernate application will try to load Entity from Session cache first if found that entity then it won't go in Second level cache.
2. If Hibernate does not find any entity in session object cache then it will search in Second level cache , if that entity found in second level cache then it will process.
3. If that entity does not find then Hibernate query will hit database.

How to implement Second level cache in hibernate?

Hibernate does not support Second level cache by default so we need to use some vendor provided second level cache like EHCache is one of the most popular second level cache vendor using this we can implement.

Spring framework

Spring is complete application development framework which is use to develop all type of application like standalone, distributed, EJB, Enterprise application.

Advantage of spring framework

1. Spring is **light-weight**.
2. Spring is **versatile**.
3. It is **non-invasive** framework.
4. It provides **rapid application development**.
5. It has given boiler plate code to reduce development effort of programmer.
6. It has strong dependency management.
7. It provides loosely coupling.

What is Non-Invasive Framework?

A framework that does not forces us to implement or extend any interface or class is called **non-invasive**framework.

Example Spring is non-invasive because it passes all that need to our application development at run time.

What is Invasive Framework?

A framework that forces us to implement or extend any interface or class is called **invasive** framework.

e.g. Struts is invasive framework because it forces us to **extend** their own **ActionServlet** class to create normal application.

Why Spring framework came?

When we design our application on the basis of **J2EE** then It will take **more time to develop project** and it complex to develop our application and API enforces us to write **boiler plate code**.

What is Problem with J2EE?

J2EE API has lots of problems that is why people switched to spring framework.

1. It is **not giving boiler plate code,**It enforces us to write **boiler plate code**.
2. It is taking more time develop application.
3. It is taking **more cost to develop** the application.
4. It provides **tightly coupling.**

What is boilerplate code?

The code which is repeatedly used in our application development is called **Boiler Plate Code**.

For example in **JDBC** we have to write **4 or 5** line to **save a single record in our database**.

J2EE API enforces us to write boiler plate code.

Why Spring is Light-Weight?

Spring framework is **Light-Weight** because it has provided all the modules separately.

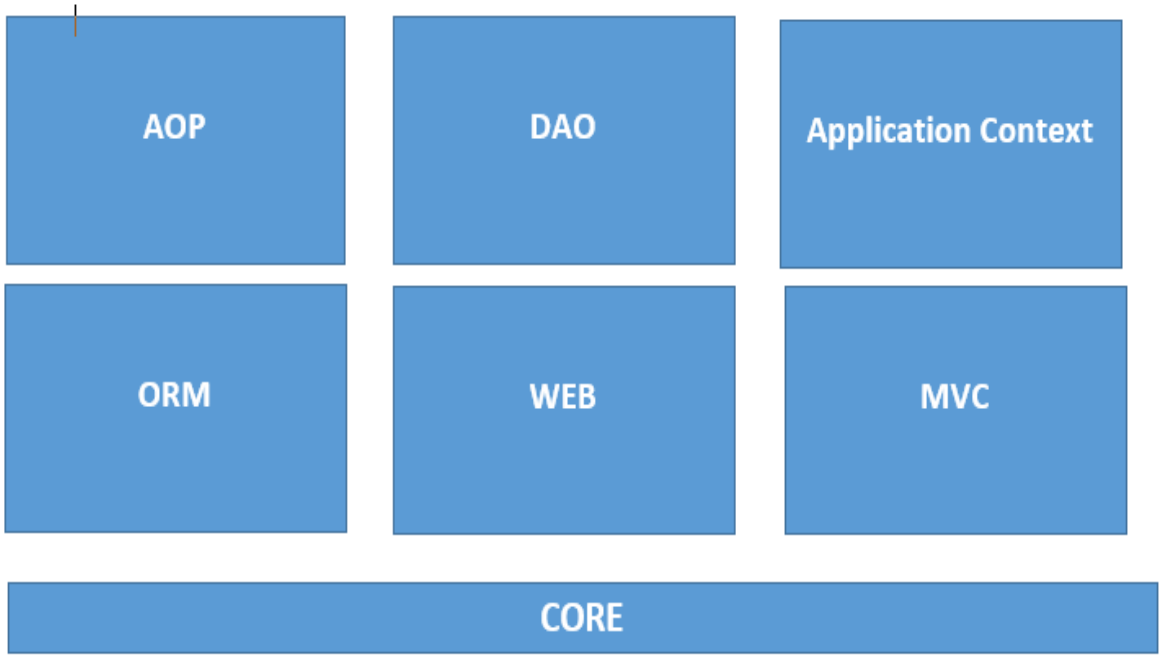
When we want to use only **Core Module of Spring** then we can use only **core module,** no need to integrate other module like **J2EE** or **JDBC** ..etc

Modules of spring framework

There are six modules in spring framework, But this spring framework will vary with version to version.

Spring Modules

1. **Spring Core(IOC Container)**
2. **Spring MVC**
3. **Spring JDBC**
4. **AOP**
5. **ORM**
6. **JEE**



What is dependency injection in spring?

Dependency Injection is a process which is use to inject dependent object into target object by container(IOC Container) itself.

Advantage of Dependency Injection

1. **Dependency injection** makes the code loosely coupled so easy to maintain
2. It makes **configuration** and **code separation**.
3. makes the code easy to test

Who does manage dependency injection?

IOC Container is responsible to manage our object in spring framework.

Ways to inject dependency in spring

In three ways we can inject dependency using **application-context.xml** configuration file in spring.

1. Setter injection

2. Constructor injection
3. Interface injection

What is IOC container in spring?

IOC Container is a principle which is use to manage and collaborate life cycle of object and it is responsible to instantiate and manage the objects.IOC Container is use to manage dependency between two class in spring framework.IOC Container is logical memory which is located on top of java JVM.

Types of IOC Container

1. BeanFactory
2. ApplicationContext

Spring Application Context

The ApplicationContext works on top of the BeanFactory (it's a subclass) and adds other functionality such as easier integration with Springs AOP features, message resource handling
(for use in internationalization), event propagation, declarative mechanisms to create the ApplicationContext and optional parent contexts.
ApplicationContext, a superset of BeanFactory does support the Annotation-based dependency Injection in spring application.

Spring ApplicationContext implementations

ApplicationContext is an interface which is use to provide configuration information of bean to Spring ioc container and it is use to manage life cycle of object within the ioc container.
We can say ApplicationContext is another name of ioc container which is used to create the Spring ioc container.
Spring ApplicationContext implementations example

1. ClassPathXmlApplicationContext
2. FileSystemXmlApplicationContext
3. AnnotationConfigApplicationContext
4. AnnotationConfigWebApplicationContext
5. XmlPortletApplicationContext
6. XmlWebApplicationContext

e.g. ApplicationContext ctx = new ClasspathXmlApplicationContext("spring-context.xml");

What is Dependency Injection?

The Dependency injection is when all dependent objects are automaticallybinded when an instance is initialized.
ApplicationContext applicationContext = new ClassPathXmlApplicationContext("/application-context.xml");

For example when we create bean in Spring XML configuration file then all dependent object will be binded at the time of creation itself.

```
<bean id="myBean" class="class_name"/>
```

What is Dependency Lookup?

Dependency lookup is concept which is trying to find a dependency .For example when we create ApplicationContext object in spring application thenthis ApplicationContext object is trying to get all dependency object from Spring XML Configuration file**.**

Note : Dependency Lookup **is slow** and Dependency injection **is fast**.

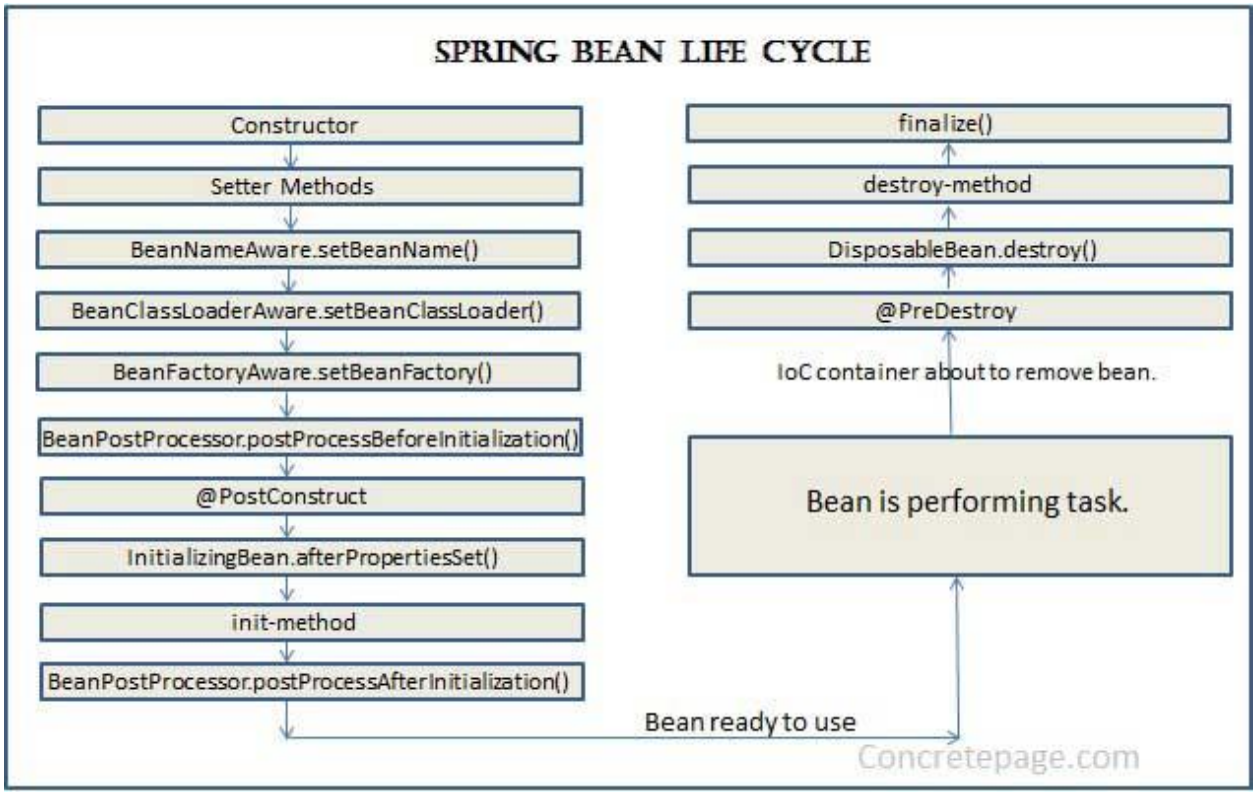
What is bean in spring?

A bean is an object which is instantiated, assembled, and managed by a Spring IoC container.

These beans are created with the spring XMLconfiguration file that you given to the IoC container at the time of creating BeanFactory or ApplicationContext**.**

Beans are use to configure database connection parameters, security etc in spring XML Confugurationfile.Beans are use to avoid hardcoding**.**Beans are use to manage the dependency of related class objects in context of IoCContainer**.**

How to create bean in spring?



Here id you have to pass in main class at the time of creating **BeanFactory** or **ApplicationContext**.

And You have to pass fully qualified class path and name in main class.

What is bean inheritance in spring?

Inheritance is one of the most important features of OOPS of any language by which we can reuse the existing functionality.

Like any language spring also has given **parent** attribute to reuse existing bean feature.

What is bean scope in spring?
In spring framework all **bean** has ascope , means every **bean** has its own visibility.
When we declare a class as a **bean** then by **default** the bean will be created under **singleton** scope.
How many types of bean scope

1. Singleton
2. Prototype
3. Request
4. Session
5. Global Session

What is singleton scope?
When will try to create multiple object of same bean then spring **IoC** container will return same object.
Singleton scope is default scope in bean.**Singleton** scope functionality is same as **singleton** class in java.

What is prototype scope?
When we will declare bean as **prototype** scope then every time **new object** will be return by spring **IoC**container.

What is request scope?
When we will declare a bean scope as **request** then for every **HTTPRequest** a **new bean instance** will be injected.

What is session scope?
When we will declare a bean scope as session then for every new **HttpSession new bean instance** will be injected.

What is auto-wiring in spring?
Auto-wiring is feature given by spring framework which is use to manage the dependency automatically.
Spring IoC container is able to find dependency and manage dependency implicitly.By **default Auto-Wiring** is **disable** in **spring framework.**
If we want to enable **Auto-Wiring** to managedependency automatically then we have to use autowire attribute in **spring XML configuration** file.

How to enable auto-wire?
We can enable auto-wire **spring IoC container** in four ways.

1. byName
2. byType
3. Constructor
4. Autodetect