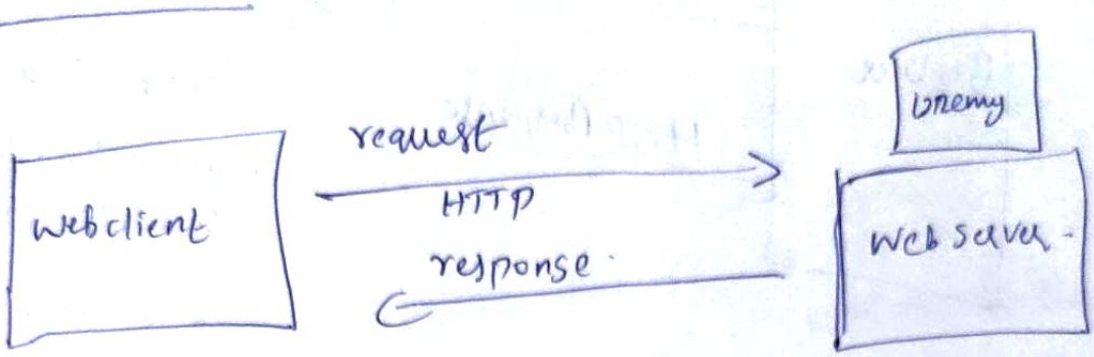
Java Web Development

Web application:



Example of webserver are Apache IIS

Webclients are google chrome

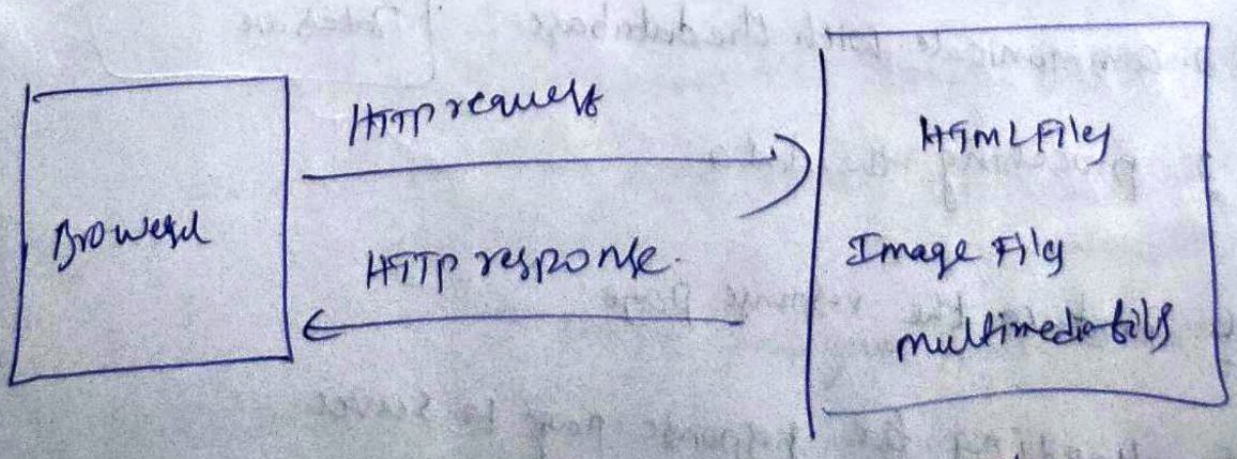
URL - Uniform Resource Locator

http://www.udemy.com:80

Protocol domain name : port no.

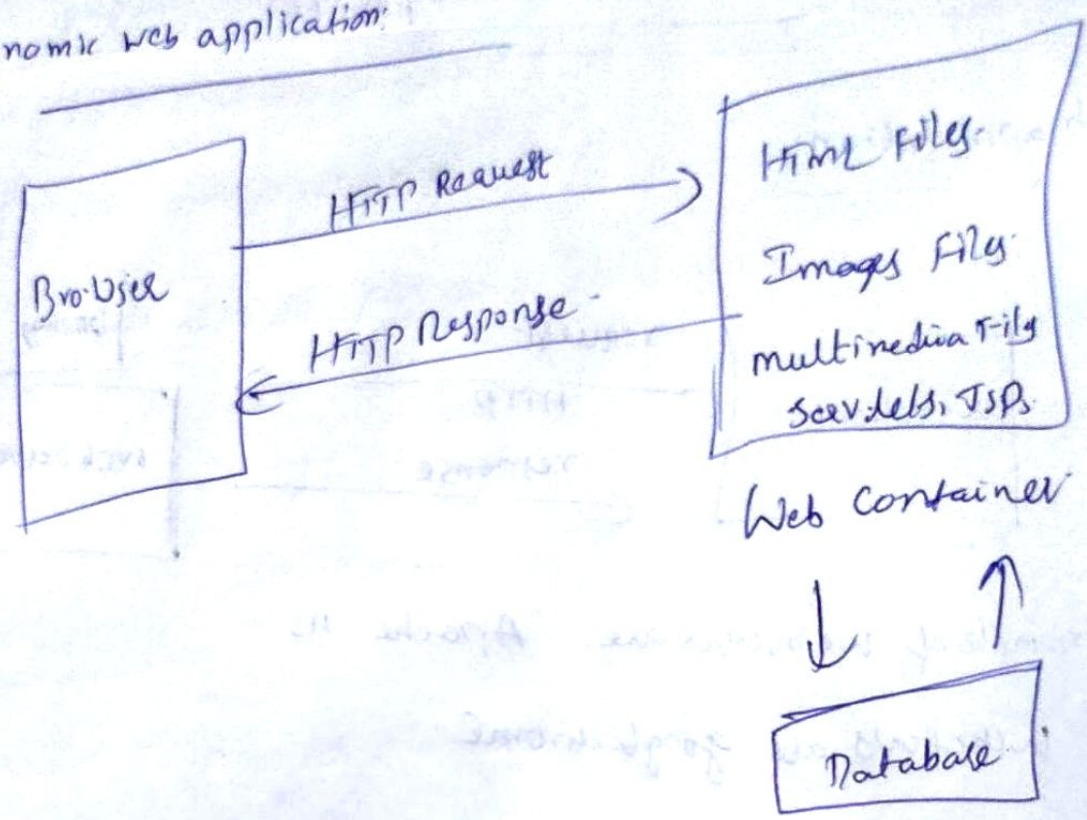
Static vs Dynamic web applications:

Static web application is non interactive web application



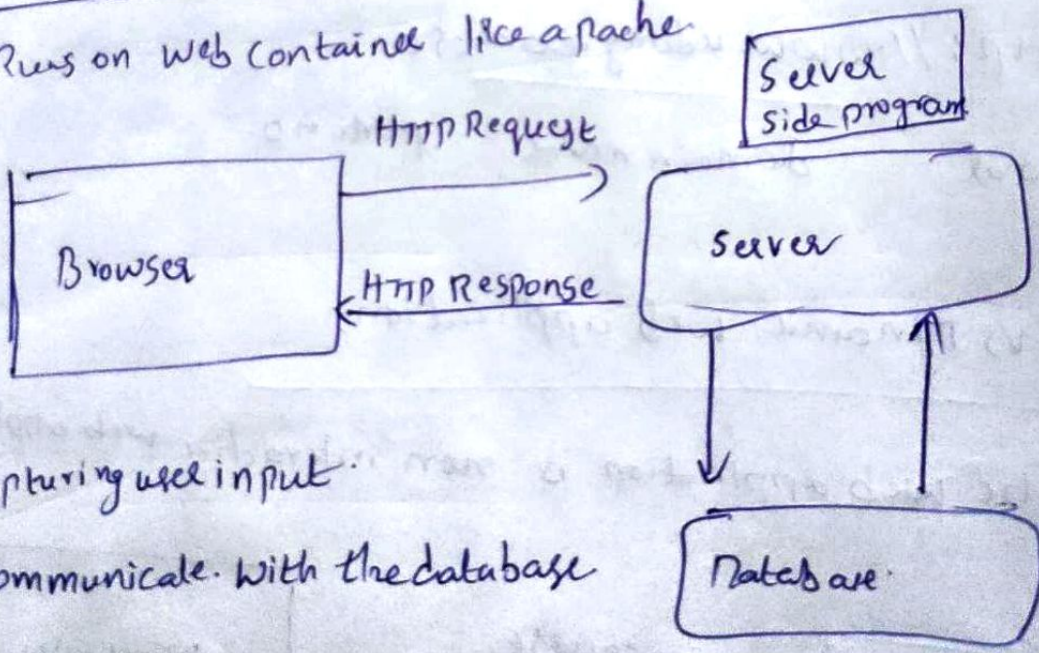
No saved side programming

Dynamic web application:



Serverside programming:

Runs on Web Container like apache



1. capturing user input.
2. Communicate with the database
3. processing the data.
4. produce the response page.
5. Handing the response page to server.

Life cycle method and phases:

3 life cycle methods: for servlets

1. init()
2. service()
3. destroy()

} These life cycle methods are called by Container like apache server

4 life cycle phases:

1. instantiation.
2. initialization
3. Servicing
4. destruction.

first goes to instantiation phase → OrderServlet.class

Container → service business logic code written

called n times.

Init method will be called once.

destroy() method will be called once.

WEBAPP

- home.html
- login.jsp

} files.

WEB-INF → Interface b/w tomcat & developers

- web.xml → Deployment Descriptor file or Configuration file → use to configure servlets & files
- classes
- lib → libraries goes here mysql.jar spring.jar hibernate.jar

Servletes Introduction:

1. It is a technology in Java EE that allows to develop dynamic web applications using Java.

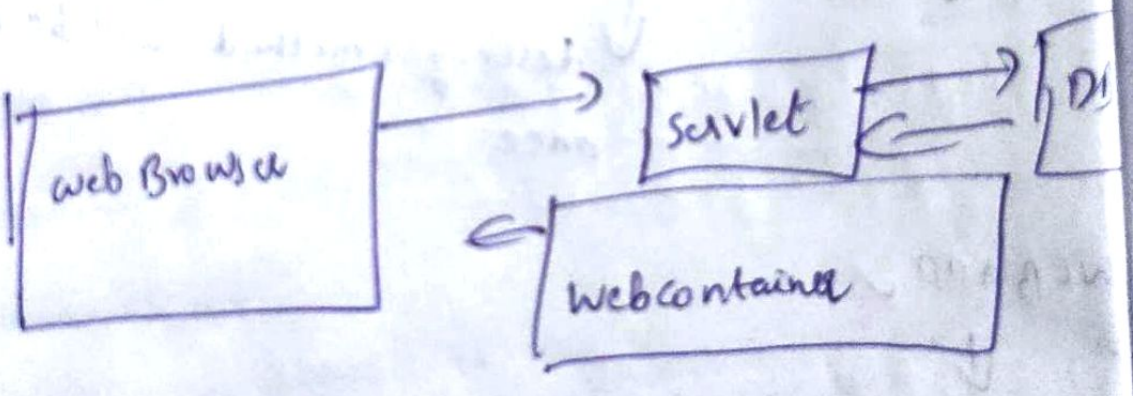
2. servlets also an API and specification [set of rules in English].

↓

It is for programmers to develop web applications set of classes & interface

Servlet:

Servlet is a Java progrom that run on server web containes.



Servlet Annotations:

@WebServlet

@WebInitParam

@WebFilter

@WebListener

```
class HelloServlets extends GenericServlet {
@Override
public void service(ServletRequest req, ServletResponse res)
{
    res.setContentType("    ");
PrintWriter out = res.getWriter();
                    ↳ writing the content.
out.println("<html>");
out.println("<body>");
out.println("<h1> Hello Servlets </h1>");
out.println("</body>");
out.println("</html>");
}
}
```

JDBC Architecture:

Containg

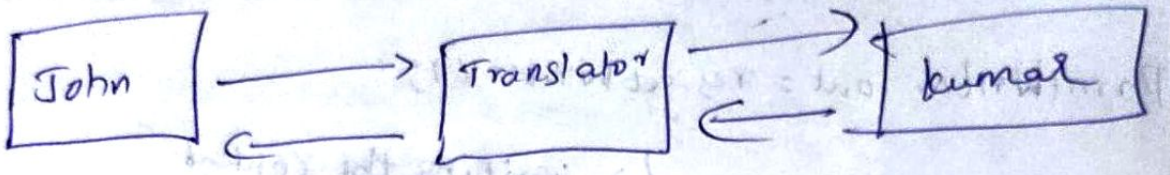
4 components.

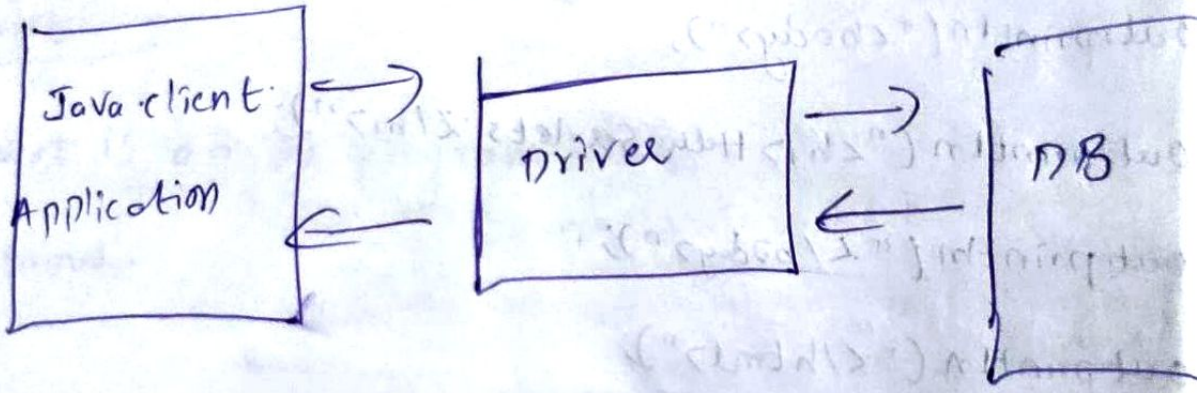
1. JDBC client - application code
2. JDBC API - standard API from oracle
3. JDBC Driver - program. interface b/w JDBC client
4. Drive Manager - establishes connection to database

JDBC API:

JDBC Driver:

Driver is like a translator:-

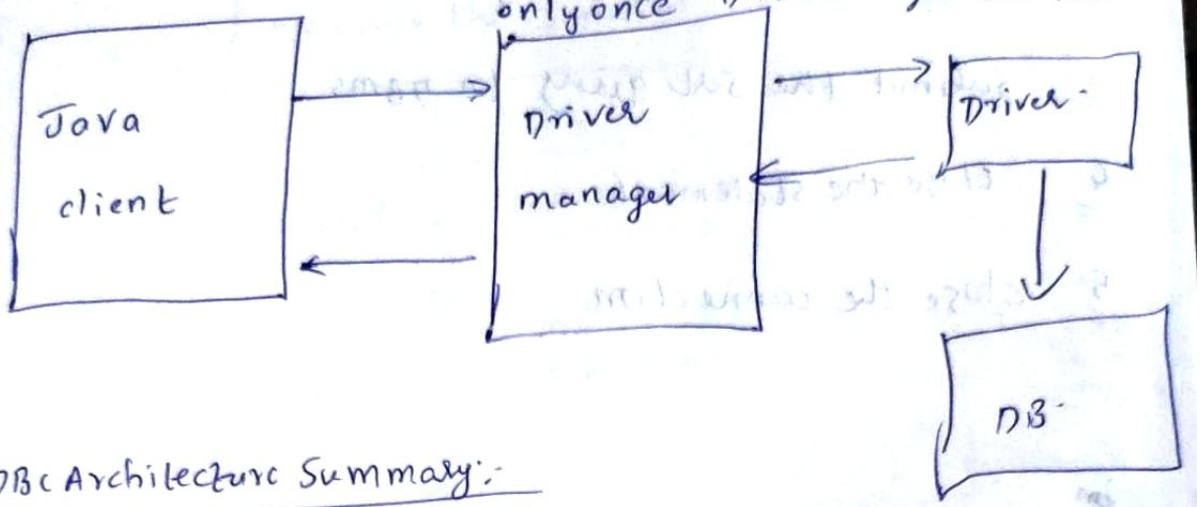




JDBC client:

1. connect to DB
2. perform CRUD
3. process the Response
4. Handle the Exceptions
5. Do Transaction management
6. Close connection.

Drivemanager: key classes in JDBC act as a key holder or helper b/w Driver & Java application. only once Drivemanager is used.



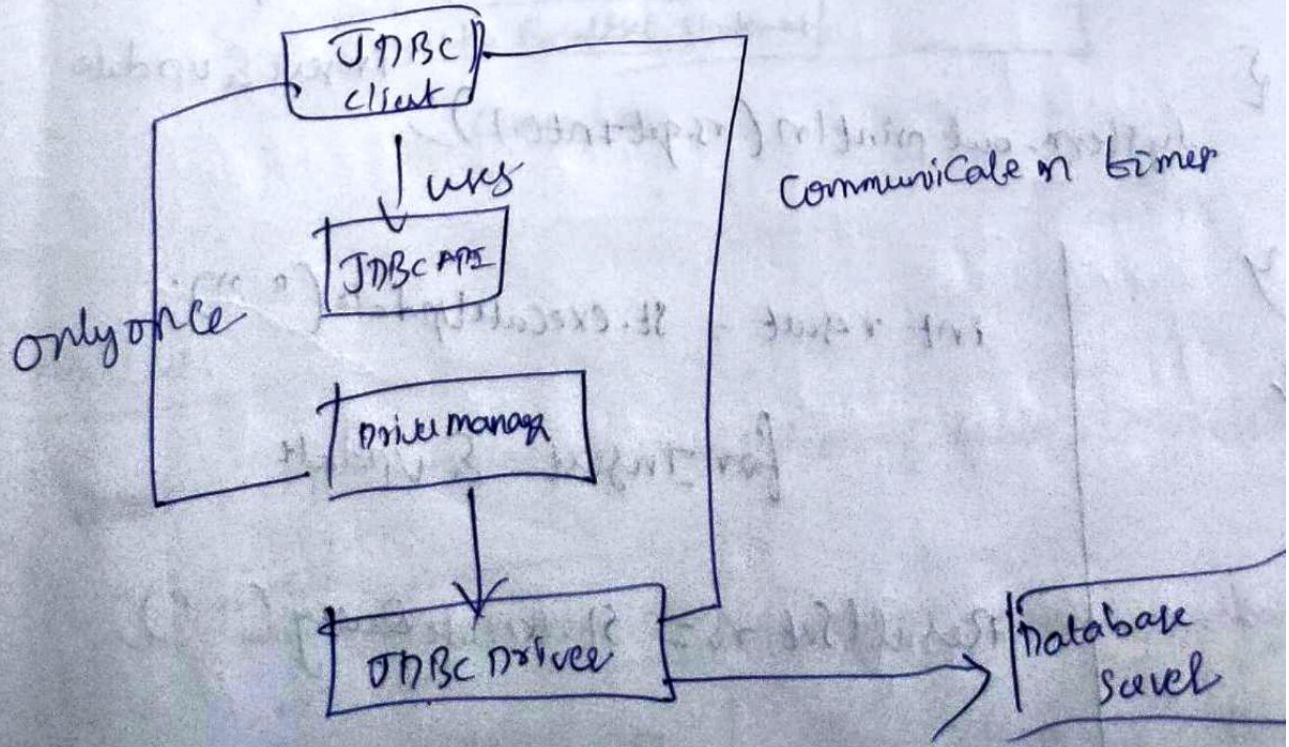
JDBC Architecture Summary:-

1. JDBC client:- Any piece of java code connect to DB

2. JDBC API:- The client develop API connect to DB. do operation.

3. DRIVER Manager:

4. JDBC Driver:- Implement JDBC specification



Step 1: Establish the connection.

2. Create the statement object

3. Submit the SQL query to DBMS.

4. Close the statement

5. Close the connection

in
class App {

psvm (String[] args)

{ String url = "jdbc:mysql://127.0.0.1:3306/database name";
Connection con = DriverManager.getConnection(url, "root",
Statement st = con.createStatement(); "password");
ResultSet rs = st.executeQuery("select * from emp");

while (rs.next())

{
System.out.println(rs.getInt(1));

}
}

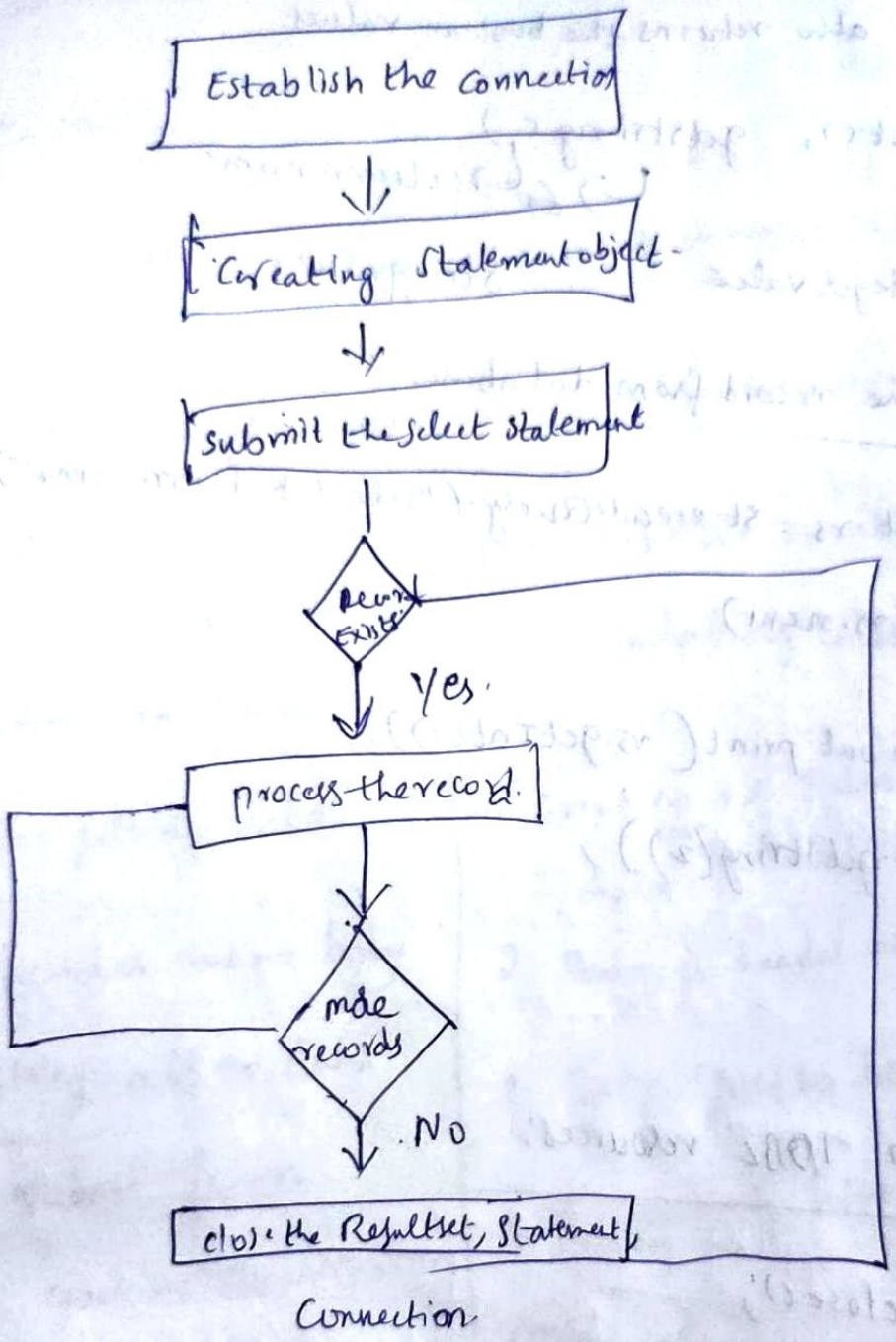
st.executeUpdate() for insert & update.

int result = st.executeUpdate(" ");

for Insert & Update.

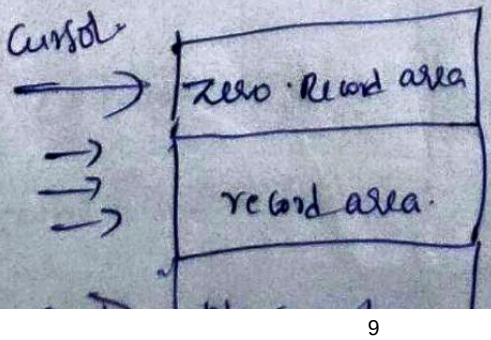
ResultSet rs = st.executeQuery(" ");

JDBC steps to Read Data:-



ResultSet:

~~Object~~ objectoriented representation of table records.



The process of reading each record is called

next() → move the cursor to next record

↳ also returns the boolean values

getInt(), getString()
↳ ↳ column name
↳
integer value String value

Read the record from database:

```
ResultSet rs = st.executeQuery("select * from emp");
while(rs.next)
{
System.out.print(rs.getInt(1));
Sop(rs.getString(2));
}
}
```

Clean up JDBC resources:

```
rs.close();
st.close();
Con.close();
```

Service provider mechanism:

GET vs POST:

GET is used default

default

3 cases get method is used.

User specified URL

. User click on hyperlink

User submits the form

1. used for getting data.

2. only header but no body

3 Querystring: all the data we submit from browser goes as querystring appended to URL

4. No sensitive Data

5. Restriction on length.

6. we can execute any no of time (Idempotent)

POST

mentioned Explicitly

<form method = "POST">

Post is used for creating, updating & deleting

1. Creating or submitting a data

2. Body & header are present.

3. Data goes to bodyserver as part of http body or payload

4. Can be secure.

5 No restriction on length.

6. Not Idempotent

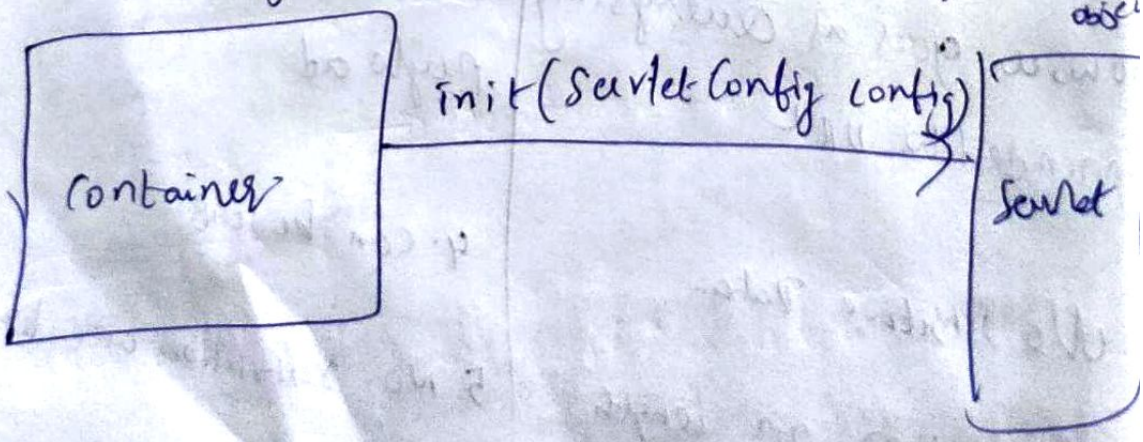
Init params: are

Name value pairs of textual information that are supplied to a servlet declaratively through web.xml during its initialization phase.

We create initparam in web.xml.

```
<servlet>
    <servlet-name>
    <servlet-class>
    <init-param>
        <param-name>dbuser</param-name>
        <param-value>root</param-value>
    </init-param>
```

during initid phase The container creates a ServletConfig obj



String dbUser = Config.getInitParameter("dbuser");

Advantage is no hard coded values

```
WebServlet (urlPatterns = "/addservlet", initParams =
{ @WebInitParam(name = "dbv", value = "jdbc:mysql:
                    //localhost/mydb"),
@WebInitparam(name = "dbuser", value = "root"),
@WebInitparam(name = "dbpassword", value = "madankumar@9")
});

Public void init(ServletConfig config)
{
Connection con = DriverManager.getConnection(config.getInit
Parameter("dburl"), config.getInitParam("dbuser"),
config.getInitparam("dbpassword"));
}
```

Initparam using web.xml

```
<servlet>
<init-param>
<param-name> dburl </param-name>
<param-value> jdbc:mysql://localhost/mydb </param-value>
<param-name> dbuser </param-name>
<param-value> root </param-value>
```

<param-name>dbpassword</param-name>

<param-value>madanbunal</param-value>

Public void init (ServletConfig config)

{

Connection con = DriverManager.getConnect (Obj Con.getInit Param("dburd"), config.getConnection InitParam("dbU

, config.getInitparam("dbpassword"));

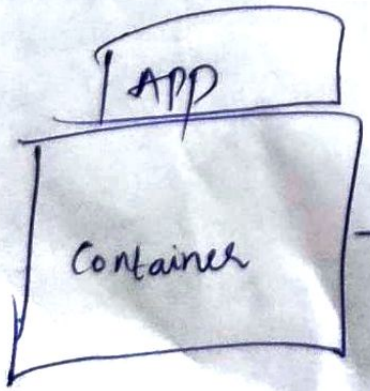
javax.servlet.ServletContext:-

Servlet content is library ~~api~~ interface in servlet API

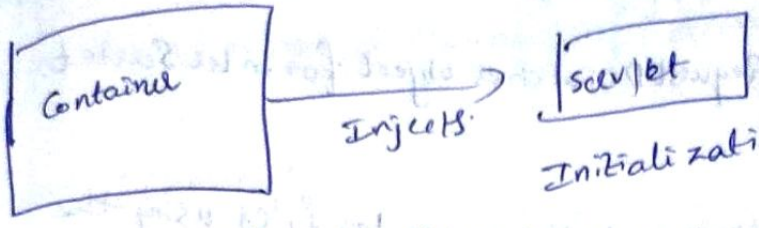
in javax.servlet.ServletContext.

Containel implements the interface

There will be one servlet context for entire application



Container with destroy the servletContext when we undeploy our application.



we can retrieve it by

init method with config

```
Public void init(ServletConfig config) {
  Servlet Context Context = config.getServletContext();
```

init method without config

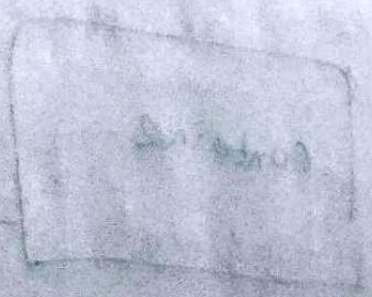
```
public void ~~config~~ init() {
ServletContext context = getServletContext();
}
```

init method without config:

```
Public void Service() {
ServletContent context = getServletContext();
}
```

Uses of ServletContext:

1. Share and manipulate Data.



SetAttribute()

getAttribute()

getAttributeNames();

2. To deal with content params.

3. To create the RequestDispatcher object for inter Servlet communication

4. To store information in to the server log files using the log() method.

Context parameters:

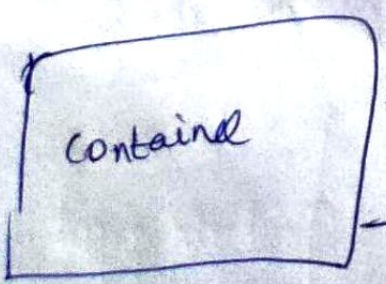
are name value pairs that are supplied through web.xml

Context parameters are accessed by any servlet or any Jsp across the application.

Init parameters are accessed by only for particular servlets.

In web.xml

```
<context-param>
<param-name> drive </param-name>
<param-value> jbl </param-value>
</context-param>
```



we can retrive by

String driver = sc.getInitParameter("driver");

Prepared statement:-

It is a child Interface of Statement interface

It is a precompiled version of SQL statement.

PreparedStatement stmt = con.prepareStatement(" ")
↳ ipd query

insert into employee values (???);

stmt.setInt(1)

stmt.setString("123");

ALTER USER root@localhost IDENTIFIED WITH mysql_native_password BY 'Kumar98';

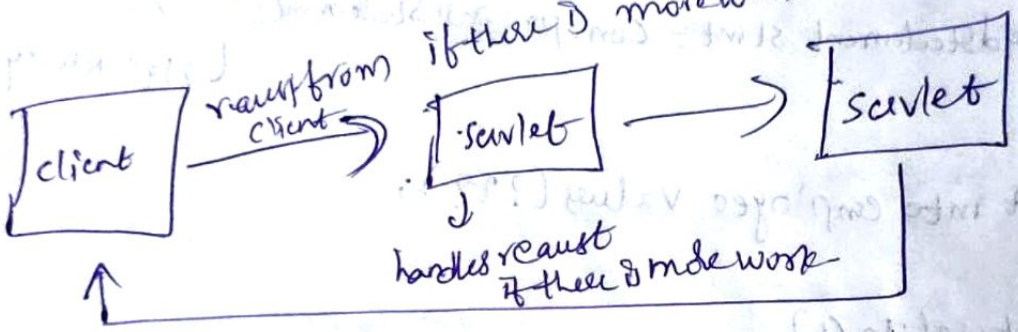
Jsp and servlet communicate with each other is called Interservlet communication. or requestDispatcher

RequestDispatcher rd = request.getRequestDispatcher(url);

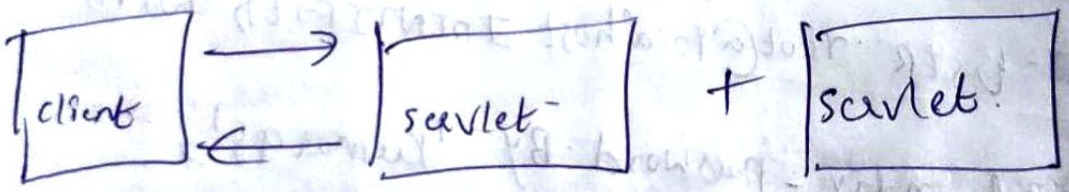
↓

Interface.

rd.forward(request, response);



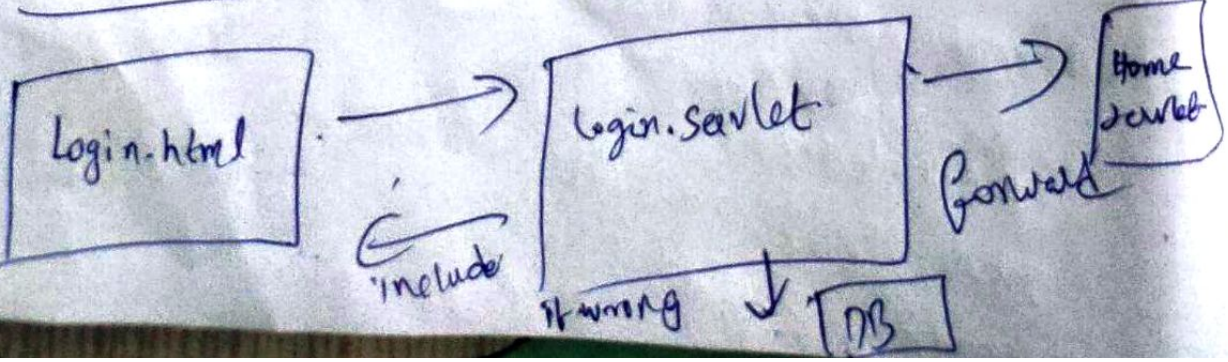
rd.include(request, response);



contain (or) Tomcat Server merges

both responses are sends to client.

Useage for interservlet application.

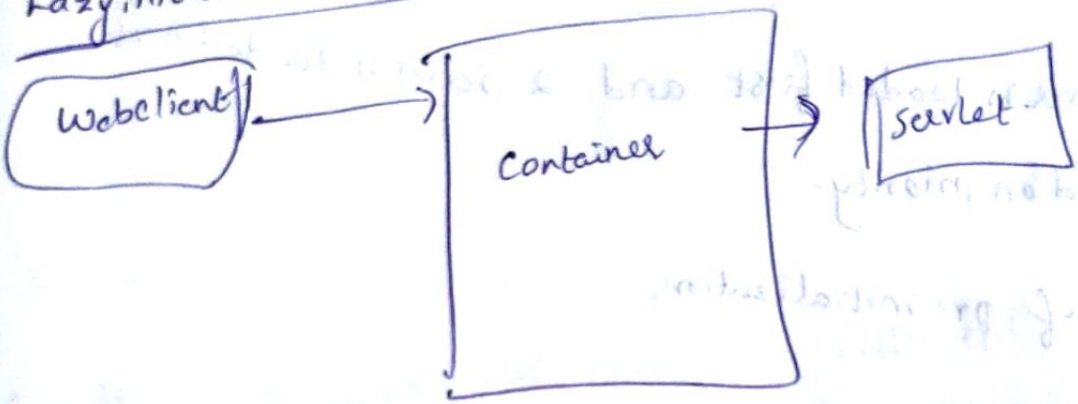


Initialization:

Two types:

1. Lazy initialization.
2. pre initialization.

Lazy initialization:



Lazy initialization.

When the first user request comes, the servlet is initialized and created.

If container initializes the servlet before user request is called pre initialized.

By using <load-on-start-up> element on web.xml file
(or) Child element of servlet element

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

```
<servlet>
  <load-on-startup> 1 </load-on-startup>
</servlet>
<servlet>
  <load-on-startup> 2 </load-on-startup>
</servlet>
```

1 servlet is loaded first and 2 servlet is loaded next based on priority

Ex: of pre initialization

webservices using CXF or Spring mvc framework

Listeners:

It enables our application to react for exceptions

Event Handling in web application

1. Request
2. Session
3. Context
4. Async

To create listner implement HttpSessionListener.

@WebListner

In Web.xml

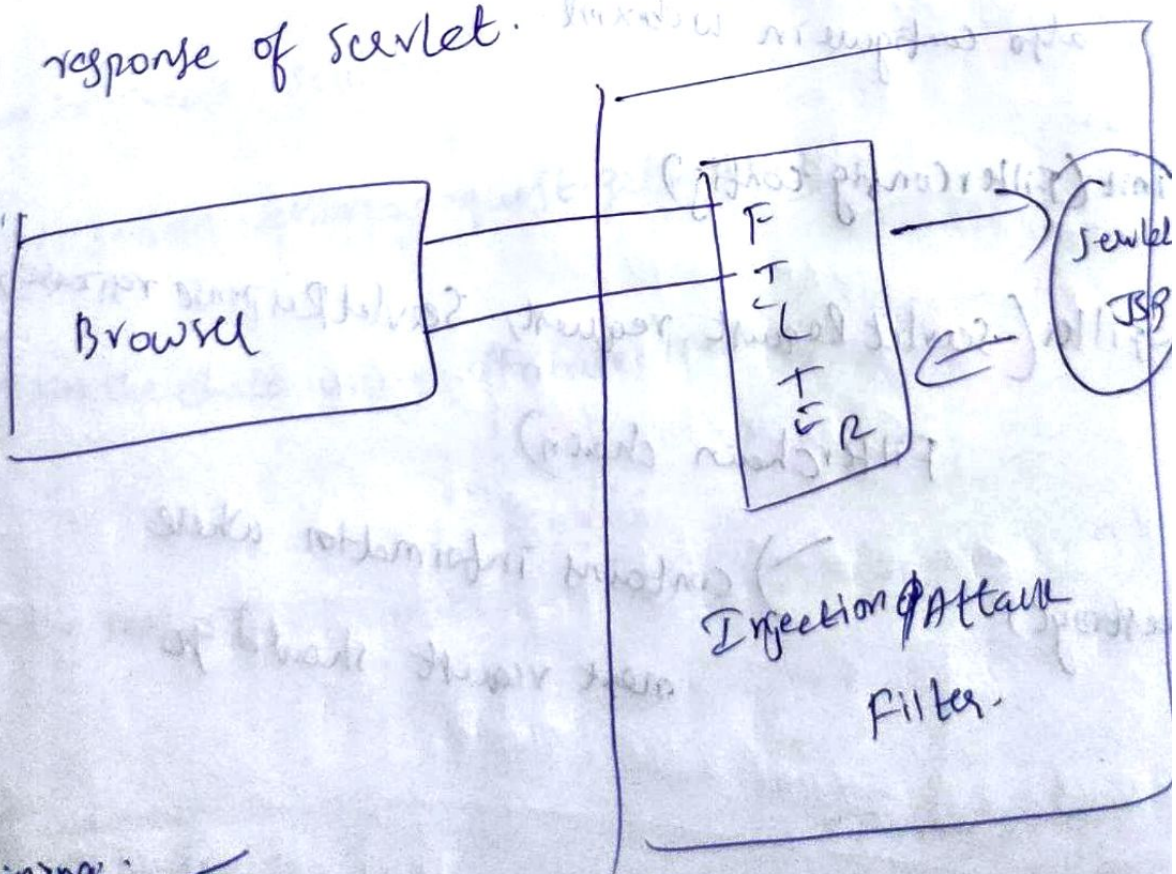
<listner>

<listner-class>UserCountListener</listnerclass>

</listener>.

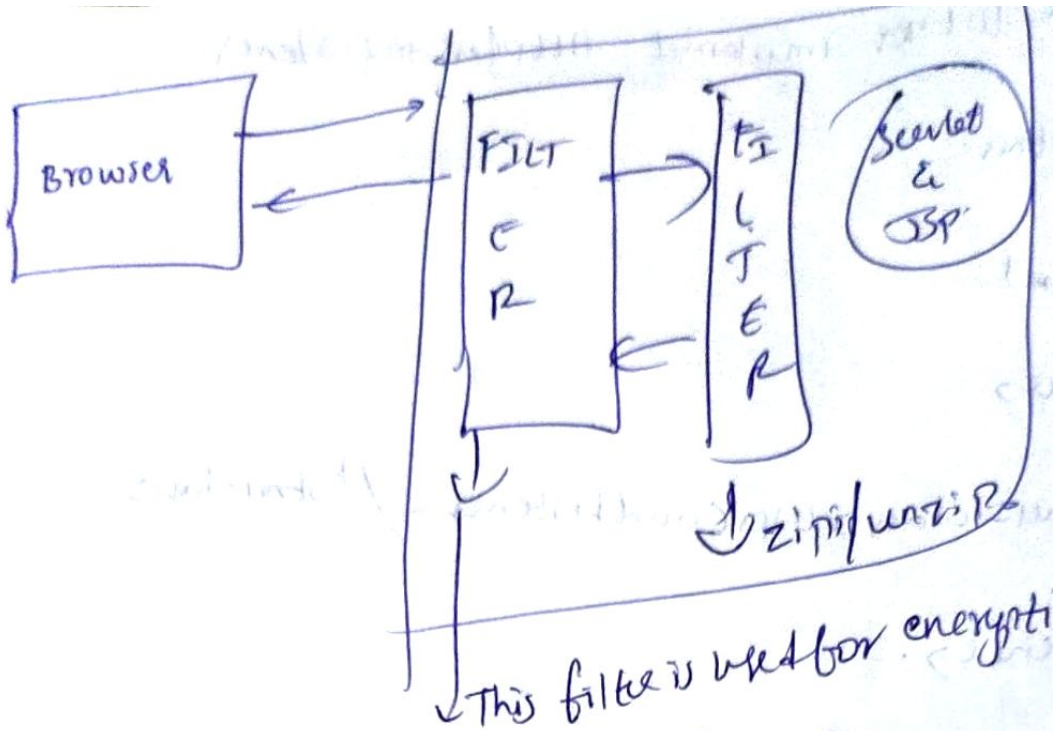
Filters:

A filter is a java class that can intercept request and response of servlet.



Filter chaining:

multiple filters.



Filter can be implemented by filter interface Filter

marked up by @Filter

also configure in web.xml.

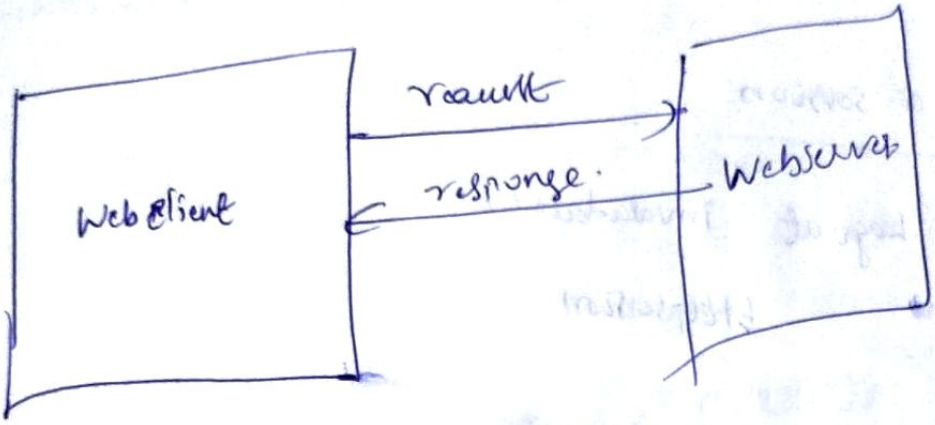
init(FilterConfig config)

doFilter(ServletRequest request, ServletResponse response, FilterChain chain) → contains information where next request should go.

destroy()

Session management

1. HTTP is stateless protocol



Advantage:

1. performance
2. scalability

How to create session:

1. HttpSession session = request.getSession();
2. Main the data using attribute method
3. End the session

Session Tracking:

To maint. statelessness of HTTP

Tracking:

Track user interaction

Session: means Login to Logout.

1. client/user Identification
2. State maintainance

} Remembering the thing → session tracking

Ending a session:

Explicit Logout → invalidate()
HttpSession.

Session Expiry: Webcontainer

overriding the expiry:

Session.SetMaxInactiveInterval(600);
(many minutes)

or

In web.xml

~~Session~~

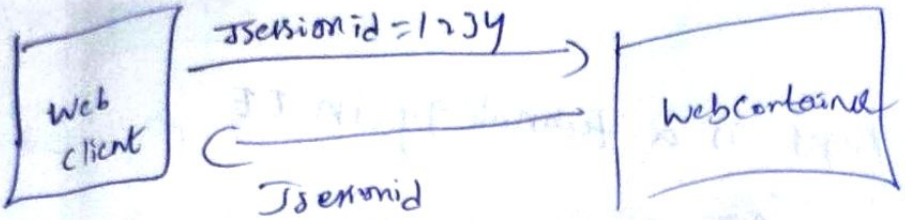
<session-config>

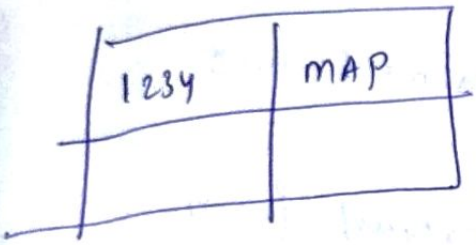
<session-timeout>5</session-timeout>

</session-config>

Cookies are name value pairs:

Used to maintain session b/w client and server





Cookie c = new Cookie ("jsessionid", "1234");

response.addCookie(c);

Cookie [] cookies = request.getCookies()

Cookies[0].getValue();

URL rewritting works only with url.

String url = "targetServletServlet ?sessionId=123";

out.println("click");

Jsp;

Java Server Pages is a technology in EE

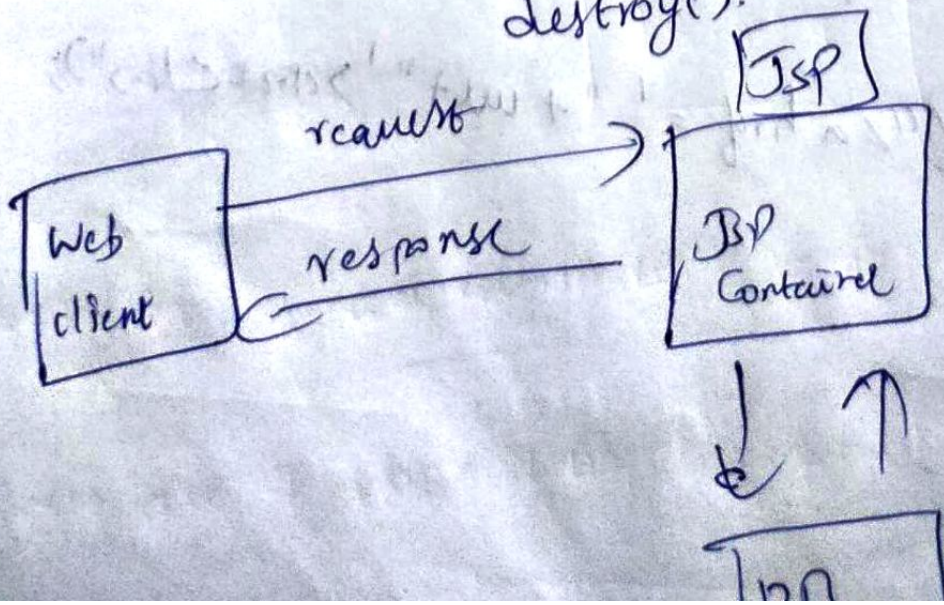
Jsp <
- API for Developers
- Specification.
  - Set of rules for container Creators

It runs on Jsp container takes request from client and process the request make call from database receives response and process the response

Jsp seperates the Jsp code and Html

It Avoids Broiler plate coding like

Httprequest
init()
destroy().



JSP Elements:

1. Scripting Elements

   are to embed java code in to a JSP page

2. Directives:

   Translation time instructions to the container

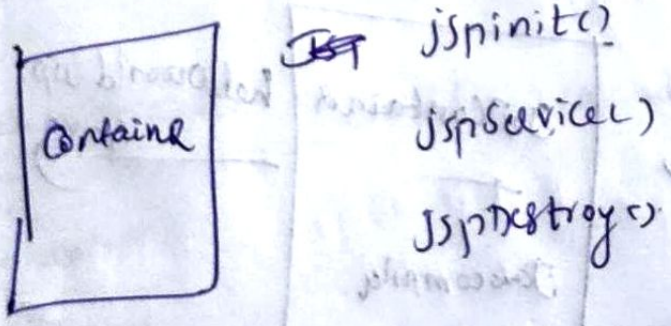
   import files

3. Actions:

   Runtime Instructions to the JSP Container

   at Runtime

JSP Life cycle methods:

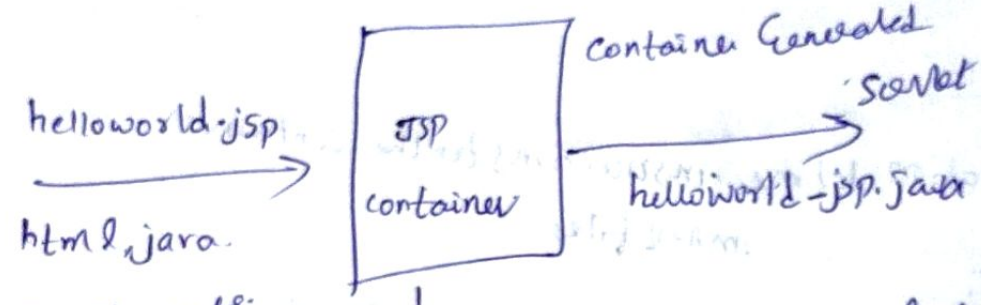


6 phases:

1. Translation
2. Compilation
3. Instantiation
4. Initialization
5. Destruction

Translation:

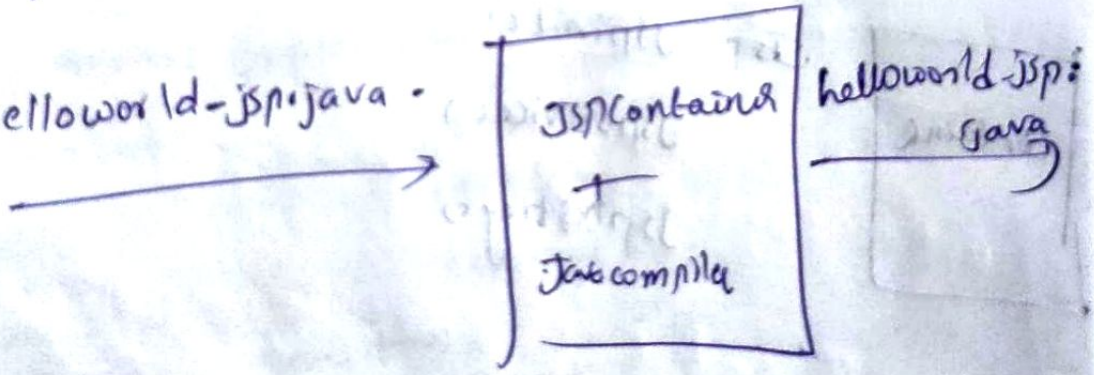
First client Request:



converts jsp file to servlet is called Translation.

Compilation:

Servlet creat in translation is converted to .class file is called compilation.



Translation & compilation will happen only once

There are 9 implicit objects:

config	ServletConfig
request	HttpServlet
response	
session	
application	ServletContext.
page	java.lang.object
pageContext	javax.servlet.jsp.PageContext
exception.	
out.	
↳ instance of JspWriter	

Scripting elements: embed javacode into jsp.

1. Declaration.
2. Expression
3. scriptlet

1. Declaration. start with. use for define.

<%!

we can put javacode

```
<%!
    int x;
    int y;
    void display()
    {
    }
%>
```

(2) Expression: evaluation & sending response to Browser

<%= a+b %>

<%= use.getName() %>

Evaluates

Sends the Response

- JspService.

(3) Scriptlet:

<%

any number of Java statements

into service Block

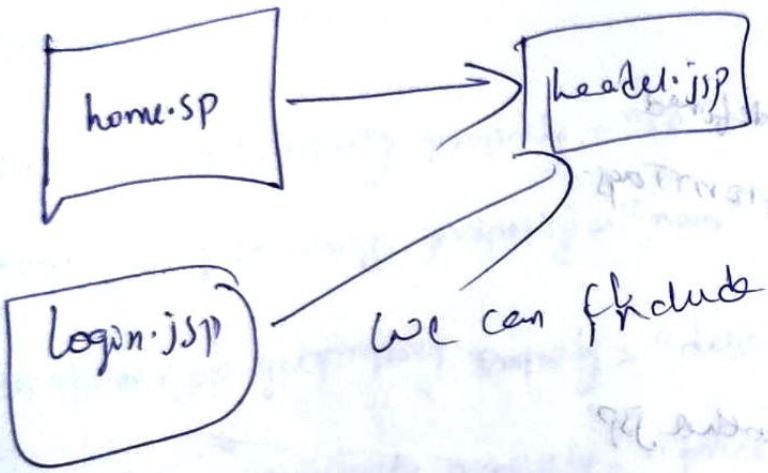
%>

JSP directives:-

3 types of

1. include use for
   ↓
   implements reusability.

<%@ include file = "" %>



We can include header.jsp in all JSP page.

Page:-

by 13 attributes

1. import (import java.package)
2. errorpage
3. iserrorpage
4. language
5. contenttype
6. session
7. extends
8. isELignored
9. autoflush
10. pageencoding
11. buffer
12. isthreadsafe
13. [illegible]

3. taglib directive

Jetof tags of html

but Java Code runs in background

JspActions:-

are JSP Tags are run time instructions

two type

1. ~~include~~ 1. predefined

2. customTags

1. predefined

1. Include - another JSP

2. forward -> one JSP to other JSP

Param in case of JSP

Usebean

Setpropeaty

Getpropeaty

Custom tags:

JSP UseBean tag:

<jsp:useBean id="product" class="com.exam.ple.jsp.product" />

<jsp:setproperty name="product" property="*" />

</jsp:useBean>

product Details

Id: <jsp:getproperty property="id" name="product" />

Name: <jsp:getproperty property="name" name="product" />

Description <jsp:getproperty property="description" name="Product" />

price: <jsp:getproperty property="price" name="product" />

MVC:

MVC is a design pattern or framework that

Splits Weblayer in to 3 parts

Model, View, Controller

↓ ↓ acts a request for Model & View

Data displaying the current model html page

1. Advantages:

1. maintainance

2. parallel development.

model is represented by Java class

View is JSP

Controller Servlet.

Custom Tags:

It is a two step process

1. create the Tag Handler class.

2. create the Tag Lib Descriptor TLD file

JSTL:

Java server page Standard tag library

Set of tags.

c: if conditional check

c: for

JSTLdemo.jsp:

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

<body>
  <c:out value="${10+9}" />
```

c:set and c:remove:

```
<c:set var="testScore" value="${80}" scope="session" />
<c:out value="${scope}" />
```

testScore

scope: request, session, application

```
<c:remove var="testScore" />
<c:out value="${testScore}" />
```

c:if Conditional View:

```
<c:if test="${testScore>=80}">
    <p>your score is good</p>
    <c:out value="${testScore}" /></p>
</c:if>
```

c:choose like switch in java.

```
<c:choose>
<c:when test="${testScore>=80}">
    A Grade
<c:when test="${testScore<=80 && testScore>=60}">
    B Grade
<c:when>
<c:otherwise>
    C Grade
</c:choose>
```

c:forEach:

```
<c:forEach var="i" begin="1" end="3">
    <c:out value="${i}"/>
</c:forEach>
```

```
<%
List<String> StudentNames = new ArrayList<>();
StudentNames.add("John");
StudentNames.add("peter");
StudentNames.add("mike");
request.setAttribute("StudentNames", StudentNames);
```

```
<c:forEach var="studentName" items="${StudentNames}">
<c:out value="${studentName}" />
```

Formatting library:-

↦ format

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/
jstl/fmt" %>
<c:set var="accountBalance" value="123.456" />
<fmt:parseNumber var="i" type="number" value="
${accountBalance}" />
<p> Amount is: <c:out value="
${i}" /></p>
```

format a given number:

```
<c:set var="accountBalance" value="7777.4567" />

<fmt:formatNumber value="${accountBalance}"
                  type="currency" />
                  maxFractionDigits="2" />
```

parseDate:

```
<c:set var="myDate" value="12-07-2022" />

<fmt:parseDate var="parseDate" value="${myDate}"
               pattern="dd-mm-YYYY" />

<c:out value="${parseDate}" />
```

Configure users and Role: Configure user and Role in tomcat-users.xml

in web.xml Configuring simple.jpp file for security.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>myResources</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>myuserrole</role-name>
  </auth-constraint>
</security constraint>
<login-config> → we need basic authentication.
  <auth-method>BASIC</auth-method>
  <realm-name>FILE</realm-name>
    ↳ Database file
</login-config>
```

for form based authentication.

change

<auth-method> FORM </auth-method>

<form-login-config>

<form-login-page>

</form-login-page>

<form-error-page>

</form-error-page>

</form-login-config>.

<form action = "j_security_check" method = "post">

UserName : <input name = "j_userName" />

password : <input type = "password" name = "j_passw

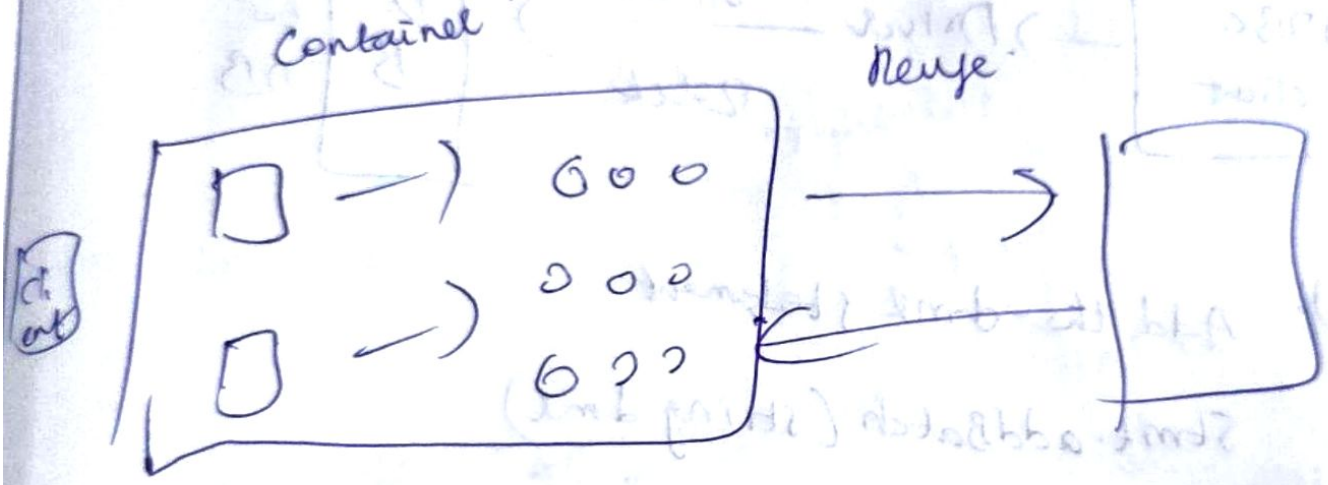
ord" />

<input type = "submit" value = "submit" />

Connection Pooling:

allows request the container to create sets of JDBC Connections

1. performance
2. reuse



Configuring a Connection pooling:

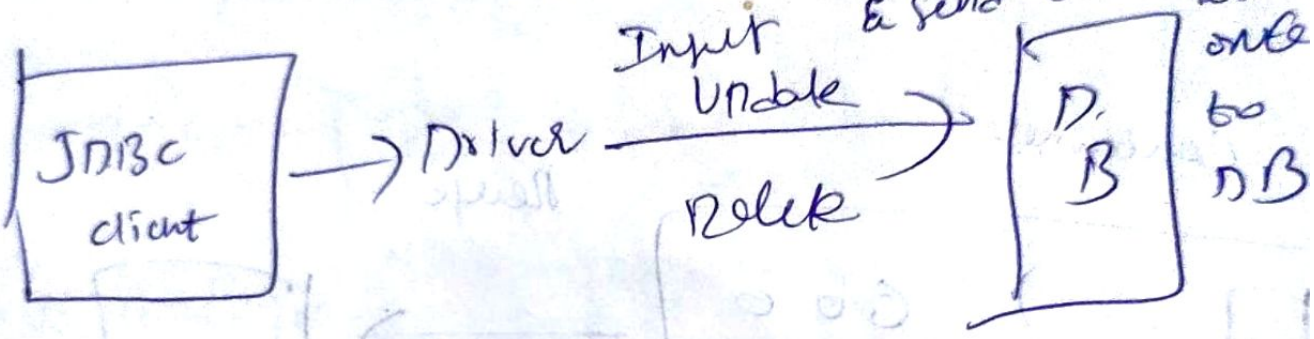
1. copy Drive Jar --) To establish Connection
2. Configure Resource Element in context.xml.

Advanced JDBC:

1.

JDBC Batch Updates: using JDBC Bath

We can group set of dml & send them all at once to DB



Step 1): Add the dml statement

```
stmt.addBatch(string dml)
```

Step 2) Execute batch.

```
int result[] = stmt.executeBatch();
```