05/08/2024

⇒ JDBC :- (Java Database Connectivity)

- JDBC is a driver API.
- java.sql.*; (Package)
- 1. JDBC ODBC Bridge (Type 1)
  2. Native API Driver (Type 2)
  3. Network Protocol (Type 3)
  4. Thin Driver (Type 4)

  } Four main Type of Driver.

- 1. Unstructure storage
  2. Semi-structure storage
  3. Structure storage

  } Types of Data Storage.

- i = internet enable
  g = grid enable
  c = cloud enable

  ] (Database Software) Version.

- // ODBC = Open Database Connectivity.
- Client dependent code in Native API Driver.
- Pure Java code in Thin Driver (Type 4)
- JDK Version, Database, Database Version
  ↓
  Driver Change According to it
- throws SQLException [For handling Exception]

// National Language Support = NVARCHAR(10)

- Execute, ExecuteUpdate, ExecuteQuery.
- Public, boolean, => Signature of Query
- getInt() & getString() method.
- Statement object is local object.
- Dynamic SQL Statement.

=> Properties() class :— 06/08/2024
multiple value can pass using Properties class.
- String PropertyNames() method when you don't know the Property key.

→ Prepared Statement => Dynamic statement
Callable statement => Store Procedure

method => prepareStatement
Class => PreparedStatement

- Store Procedure is a Database object.
- " " is a callable code.
- " " has a store object that are Pre-Compiled.

Syntax :—
CallableStatement cstmt = con.prepareCall
("{call P1(?)}");

// raw() & longraw() datatypes Oracle Database &

→ registerOutParameter() = For mapping to backend to Java code.

→ java.sql.Types.*; (contain final types) where Types is a interface.

→ BLOB() & CLOB()

→ setBinaryStream() & setAsciiStream() methods are important methods.

24/08/2024

=> MetaData (JDBC) :-

1.] ResultSet MetaData } Both are interface.
2.] Database MetaData }

// rs = ResultSet // rsmd = ResultSetMetaData

- rs.getMetaData(): ResultSet MetaData object
- rsmd.getSchemaName(1) = For Single element Schema Name
- // .getTableName() // Table Name
- // .getColumnCount() // integer
- // .getColumnName() // String
- // .getColumnType()
- // .getColumnTypeName()
- // .getPrecision() // Size

- con.getMetaData = Database MetaData

☆ Networking :—

Package = java.net.*;

- TCP Based Communication ↔ (Request/Response Based) (Connection oriented)
- UDP Based Communication (Connection less)
- Multicast Based Communication

→ TCP is Synchronous Communication (Acknowledge)
→ UDP is Asynchronous communication (Unacknowledged) (Delay occur)
  - Every transaction different Package generate
  - Different Channel.

- Every Communication (created using different Packet.
- Main two Classes :—
  1. Socket Class
  2. Server Socket class → Server side only

Socket class object (IP., Port)
server

// Port is a reference to the buffer which is associate with application.

Server Socket class object (Port No)

→ accept() is a blocking method. (SS.accept())
When client is requesting, run on Server Socket

1. Request Listen
2. Request accept
3. Request Acknowledge

} Accept() method 3 work

→ Round Trip Communication in Server and Client. [Server side Socket to Socket]

→ Channel is already built so we just take using InputStream and OutputStream. control (Socket Stream).

// Components :— 10/09/2024

Server Side	Client Side
1. Server	1. Server
2. Server Socket	2. Input Stream
3. Input Stream	3. Output Stream
4. Output Stream	

(Created only at server side.) ← 2. Server Socket

// Port Number is associated with network purpose.

// UDP Protocol :—

- Asynchronous (Asynchronous) mode Communication
- Connection less
- Components :- (Server side)
  1. DatagramSocket
  2. DatagramPacket = Byte Array store in it.
  3. byte[] = byte array.
  4. Port

- VDP - client side component is
  1. InetAddress
  2. Port
  3. DatagramSocket
  4. DatagramPacket
  5. byte[]

=> InetAddress = It's class in java, used to create obj of IP address, InetAddress class doesn't have a constructor.

=> Factory Method = The methods which are used to create a class are known as Factory Method

→ Methods :-
1. getLocalHost();
2. getByName()
3. getAllByName()

→ MulticastSocket class is sub class of a DatagramSocket class.

→ Multicast has its IP address.

→ Range = 224.0.0.1 to 239.255.255.255

→ import sun.net.*; // For old classes and methods

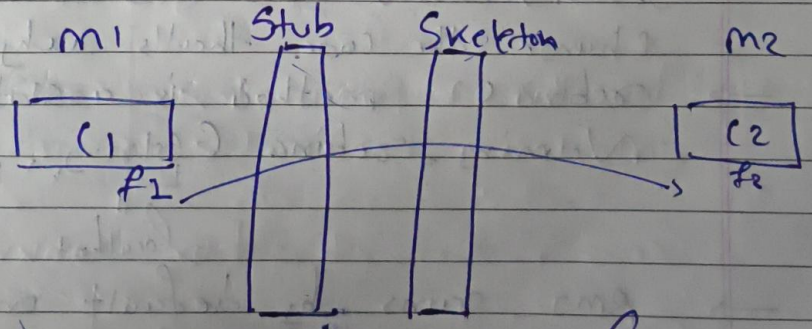
COBOL
// Common Business Oriented Language

Page
Date 12/9/24

=> RMI (Remote Method Invocation)

- RMI is a subset of CORBA architecture
  [Common Request Broker Architecture]
- CORBA architecture allow you to inter. language.
- Stub (Client) & Skeleton (server) → Two elements → working like Broker.
- → Marshalling Process is involved.

(Sending) Marshalling = Object → Byte Stream
(Receiving) Demarshalling = Bytestream → Object



Stub.class Skeleton.class files.

- After JDK 1.8 → only one file stub.class
- Put the two files on both side client side & Server side.

- RMI is web IIOP Protocol
  // Internet Inter Object Protocol

Play old java objects = POJO

→ Pojo :— 4 Files (Java)

1. Interface file → (1) method Declaration

2,3,4 - Class file → (2) Client, (3) Server, (4) method Implementation

For Communication

25/09/24 Package :— import java.rmi.*;

→ Remote is a interface
→ Interface can not extends (inherit) class.
→ UnicastRemoteObject is a class.
→ SerialVersionUID = 0 (Different developers change the code - that's why its used).
→ rebind() method is a static method
Naming.rebind(String, Object);
localhost Object of class.
→ RMI runs by default on port no 1099.
→ URL = To locate resources in uniform form. (Uniform Resource Locator)
→ Using RMI Protocol we map the URL. (Bind the object)
→ binds & rebind() methods.
→ Naming.lookup() = For Search URL
URL
return the object of implementation class

Note:-

→ Compile all the file = javac *.java

→ Another Compiler = rmic <class files>
↳ For create stub & skeleton files for the remote method

→ For versioning :-
rmic -v1.2 <class files>

→ Two generated files :-
ServerImpl_Stub.class
ServerImpl_Skeleton.class

→ rmiregistry is a Services.
start rmiregistry. // windows

rmiregistry & // Linux

→ When the port is change you have to give it to in URL.

→ start java server

→ Note: A device can have multiple IP address via virtualization.

→ LocateRegistry.CreateRegistry();
↓ ↓
interface method
// For starting the services.

Boon Reference // Sun Microsystem Volume 2

Page
Date

→ Distributed Computing
  ↳ master → slave
  ↳ Completion, Mergin

26/9/24 => Servlet (Java code for work like web application)

javaee = java extended.
import javaee.servlet.*;
import javaee.servlet.http.*; // Package

[// executed //
1. init (Once)
2. Service (multiple)
3. Destroy (Once)] Lifecycle methods

→ Whenever Servlet class load the application
→ Call with every request.

methods [doGet = Parameter come from get Parameter
doPost = Parameter come from Post "

Generic Servlet = For unknown protocol

PrintWriter , PrintStream
↓ ↓
response.getWriter(); response.getStream();

response.setContentType("text/html");

classpath set [// Apache - Tomcat = Server For Run application
// Servlet-api.jar

Internet = Hardware = Network of Networks
www = Software = Use the internet

Web.xml

→ <servlet-mapping>
<servlet-name>HelloWorldName</>
<url-pattern>/HelloWorld</>
</servlet-mapping>

→ <servlet>
<servlet-name> /servlet-name>
<servlet-class> </servlet-class>
</servlet>

Notes: Startup.bat → For starting Apache tomcat
shutdown.bat → For stop Apache tomcat

28/9/24 // Parameters From User
request.getParameter("User");
↓
Text Field

<Form action="servlet-url", method=GET>

// Parameters From Startup
→ Database connectivity in init method.
<servlet>
<init-Param>
<Param-name> </Param-name>
<Param-value> </Param-value>
</init-Param>
</servlet>

// Overloading methods:-
public void init(ServletConfig sc){
myInitName = sc.getInitParameter("InitName");
}

<load-on-startup> 1 </load-on-startup>
→ For First servlet run. Give Sequence number for Get /Loaded first.

→ request.getParameter("User");

→ RequestHeader = For checking the request from

→ Request Info = Informations about client to send a request.

// request.getHeaderNames();
// request.getProtocol();

ServletContext	ServletConfig
- Gives the data about Servlet Container - i.p. port. http - about application Server	- To get all the info about servlet [which is stored in web.xml] - init parameters

// CGI = Common Gateway Interface Scripting
// Jsp = Java Server Pages
↳ Internally connected into Servlet (Java embedded with html)

☆ Session Handling Mechanism :—

=> Cookies :—
- enables to Save data on client side
- Example: Gmail

Advantages :—
- Every Time You use device data is available.
- Needs limited storage
- Faster

Disadvantages :—
- If You change the device, data is lost
- Less secure

List :- request.getCookies()

=> Http Session :-
- This is on Server Side
- Example: Exam Portal
- More Secure
- Needs larger Storage
- bandwidth can be issue while connecting.
- If You Change device, You will still have session
- Congestion issues associate
- Can Expire

//ORoms = Object Roms
response.addCookies()

HttpSession session = request.getSession(true)
.setId
.setAttributeName
.setAttribute

session.invalidate(); ——> to destroy session

Page
Date 03/10/24

=> HTTP Filters :-

→ Allowing necessary details only
→ It's Working like a middle man.

Package : import javax.servlet.Filter;
↓
Interface

Init() method parameter ← FilterConfig fg = Object creation in init method.

doFilter() method = (ServletRequest, ServletResponse, FilterChain) } 3 parameters

Type Casting = HttpServletRequest req = (↓) req;
HttpServletResponse res = (↓) res;

web.xml

```
<filter>
    <filter-name> </>
    <filter-class> </>
</filter>
// Note : This block is always be written before normal <servlet> tag.
```

```
<filter-mapping>
    <filter-name> </>
    <url-pattern> </>
</filter-mapping>
```

=> Calling Servlet Within Same application Server.

request.getRequestDispatcher("/HelloWorld");
↳ For Forwarding Servlet request & response

.WAR = Web Archive

Command = jar -cvf <app-name>.war -C ./.

=> calling Servlet from another application Servlet :-

response.sendRedirect(url);