Java 2 Enterprise Edition (J2EE)

Reference Material on J2EE technology

Mr. P. Suresh Kumar

By
Mr. L. Madhav
(http://www.geocities.com/megamadhu
http://free.7host06.com/lmadhav)

My sincere thanks to Mr. T. Ravi Kiran for providing notes on JBBC, CORBA, RMI and other friends who have supported me.

Contents:

Chapter 1

Introduction 03
Required Concepts (JDBC, CORBA and RMI) 06

Chapter 2

XML 41

Chapter 3

HTTP 50
Servlets 52
JSP 71

Chapter 4

Java Beans 78
Struts 90
EJB 98

Chapter 5

JMS 117
Java Mail 121
Web Services 123

Introduction:

javap can be used by the programmers to find the info about a java class. It can also be used to disassemble the code that should in a class file.
Ex: *javap <options> <classes>*
javap java.lang.Integer

When we compile a java class file, the java compiler generates a class file in byte code (numbers representing instructions). When javap tool is executed with -c option these numbers will be converted as instructions like getstatic, return …
As part of the Object class we have an implementation of *toString(), hashCode()* methods. Depending upon our requirement we may need to provide the implementation of these methods.

The implementation of *hashCode* provided by Java Soft may not be suitable for our classes. In this case we need to implement our own logic for *hashCode.* The objects representing the same information must return the same hashcode.

Apart from implementing *hashCode* we need to implement *equals()* – returns true if your objects are considered to be equal.
Even though *equals* was implemented as part of object class this may not satisfy our requirements. In such a case we need to override *equals().*

*We have to override *equals()* whenever *hashCode()* is overridden.

toString() provided as part of *object* class returns the name of the class and the hashcode. This may not be that useful in debugging the code.

toString() — Used for debugging
hashCode()
equals() } Used to identify whether the objects are same or not

Most of the programmers debug their applications by including the statements that displays the information about the object. It is always advisable to provide *toString()* that returns the info about the object.
Its our responsibility to return useful info from *toString()* method.

Software Development Kit (SDK):

So many companies have released the products that can be used by the software developers to develop programs. These products are generally referred as SDK ex: Microsoft Windows SDK which contains image controls, cursor controls, text editor, c compiler, linker, debugger, header files and libraries used to develop programs in windows.

Java soft has provided a set of tools like javac, javap, appletviewer, javadoc, rmic, jdb, jar, idlj … to help programmers in developing the java applications. These set of tools are packaged together as Java Development Kit (JDK).

Most of the UNIX programmers place their executables (binaries) under Bin directory, libraries under Lib directory and header files under Include directory.

Instead of supporting the class files to a customer separately we can pack them as a single file using java archiver (JAR2 – Java Archive 2). Jar uses the format which is same as the format used by WinZip.

Native code is the code implemented in a language other than java (c, c++). For including native code some header files has to include.

As part of JDK JavaSoft has supplied Java Runtime Environment (JRE) which will be available under JRE directory. JRE is the set of tools, libraries that are required to run java applications.

JavaSoft for windows has supplied java.exe which is JVM (Java Virtual Machine). Apart from JavaSoft the companies like JRockit, TowerJ, Blackdown supplies JVM of their own.

JRockit, TowerJ are commercial products and they are considered to be very efficient than the JVM supplied by Sun Microsystems.

MFC is a set of classes used by c++ programmers to develop windows application.

JavaSoft has developed the extensions like Swing (code name for JFC), servlets with the package name like *javax.swing, javax.servlet.*

From Java 1.2 JavaSoft has started using the new term called Java 2 Platform.

Later on JavaSoft has released 3 different packages.

- J2SE used to develop the client applications
- J2EE used to develop the server side applications
- J2ME used to develop java applications that run inside small devices like Mobile phones …

Difference is on the set of packages available in each.

For a java programs an API (Application Program Interface) is a set of interfaces and classes.

JDBC (Java Data Base Connectivity)

For a C programmer an API is a set of functions ex: Windows API is supplied by Microsoft which consists of a set of functions that can be used to write windows applications.
Ex: A company xyz may supply a product like xyz printer API.

For accessing the Data Bases like Oracle, Sybase, and Informix from a C language we can use DB specific APIs (Native APIs).

Almost all DB server vendors provide server and client components. Server component is responsible for managing the DB and client component is responsible for connecting to server and perform different DB operations.

As part of the client software most of the DB vendors provide the tools which can be used by the developers and the administrators plus a set of libraries those can be used as part of the project.

If we have to use Oracle server from the C programs we can use Oracle Call Interface (OCI) i.e. supplied by Oracle corp.

Incase of Oracle we get OCI when we install Oracle client.
Similarly Sybase provides Sybase DB Lib and Microsoft provides DB Library to access SQL server.

We can develop the C program to access Oracle using OCI calls.

In almost all DB operations we follow

- Connect to DB
- Perform DB operation
- Disconnect

If native APIs are used we may need to write lot of code for supporting different Data Bases and the developer need to learn about multiple APIs.
To simplify writing the programs that access DB Xopen.org specified Client Level Interface (CLI) and Microsoft has specified ODBC.

An open specification is a specification that is available for public (anyone can use the specification).

There are so many companies which has released the specifications about an API or a technology. If the specification is available for all anyone can provide the implementation.

Specification means no code only prototype
Implementation means providing the code.

Once the specification is released multiple vendors can provide the implementations this gives wide choice to the customers in choosing the product.

Microsoft has released a specification called ODBC API with the functions like SQLconnect, SQLexecute, SQLfetch …. This specification gives the info about the purpose of each and every function, parameters to the functions, the expected return values of the function. This info can be used by any vendor to provide the implementation.

Anyone can provide the implementation of ODBC API functions. These implementations are called as ODBC Drivers.

ODBC drivers are supplied as Dynamic Link Libraries (DLL).

For a single DB there can be multiple ODBC drivers.

With the ODBC option the programmers need not learn the Native APIs. Once he learns the ODBC functions he can write the code to interact with any DB using the same set of functions.

To run the programs that use ODBC API requires the appropriate ODBC driver.

Programs using Native APIs are much more efficient than programs using ODBC API.

JDBC is an open API that can be implemented by anyone similar to ODBC API implementations.

As part of JDBC specification JavaSoft has provided a set of interfaces like *java.sql.Driver, java.sql.Connection, java.sql.CallableStatement, java.sql.ResultSet, java.sql.Blob, java.sql.Statement, java.sql.PrepareStatement, java.sql.Clob* plus the classes like *java.sql.DriverManager, java.sql.Date, java.sql.Time …*

Any vendor can provide the implementation of these interfaces. The implemented classes are called as JDBC drivers.

Every vendor provides the documentation in which he specifies the name of the class that provides the implementation of *java.sql.Driver.*

In the documentations provided by the vendors they will specify the JDBC URL.

To write the JDBC program:

- .zip or .jar containing JDBC drivers
- Name of Driver class
- JDBC URL

DriverManager class is responsible for maintaining the info about the drivers loaded by a program.

Class path: when we compile the source code the java compiler searches for the classes in the source in class path.

It is very common to place the class files under different directories. In this case we need to specify all the directories as part of the class path.
javac -classpath c:\dir1\dir2 one.java
javac -classpath c:\dir1\dir2;d:\dir1 one.java
As part of class path we can specify multiple directories separated by ';'.
The above command can be replaced by,
set classpath = c:\dir1\dir2;d:\dir1
javac one.java
Instead of using *–classpath* option with *javac and java* commands we directly set the classpath environment variable using the *set* command as shown above.
To view the value of classpath environment variable we can use,
echo %classpath%

It is very common to use jar files to combine multiple classes. We can create jar file as,

c:\dir1> java cvf myclass.jar .

c – create a jar file

v – verbose form

f – file name

. is used to represent current dir

Above command takes the files from the current directory and places them in myclass.jar.

To view the jar file,

C:\dir1> java tvf myclass.jar

t – table of contents

v - verbose form

f – file name.

We can also use WinZip to open jar files. We can use .zip extension. In JDK 1.0 (old versions) the class files are packaged in .zip files but the preferred extension now is .jar.

We can extract a jar file using,

C:\dir1> java xvf myclass.jar

x – extract v – verbose form f – file.

To ser class path for jar files, *set classpath = c:\dir1\myclass.jar*

For more info on jar see the link,
http://java.sun.com/j2se/1.3/docs/tooldocs/win32/jar.html.

In JDBC programs we need to

- Register the driver
- Connect to DB
- Perform the DB operations using SQL statements like *insert, update, delete, select.*
- Disconnect.

To register a driver we need to know the name of the class that provides the implementation of *java.sql.Driver.*

1. We can use class.forName followed by the name of the driver class as, *class.forName("oracle.jdbc.driver.oracledriver");*

2. we can use the code,
   java.sql.DriverManager.registerDriver(neworacle.jdbc.driver.OracleDriver());
3. instead of writing the code to register the driver we can set the system property *jdbc.drivers* on the command line as,
   java –D jdbc.drivers=oracle.jdbc.driver.OracleDriver
   syntax*: java –D<name>=<value>*

According to our requirement we can define our own system properties. JavaSoft has defined some standard system properties. We can get the info like the name of the Operating System, JVM's version … using standard system properties.

The system properties can also be set inside Users Home Directory (~)
~/.hotjava/properties file

On windows operating sys Home directories are created under "Documents and Settings".

We can never create an object based on an interface.

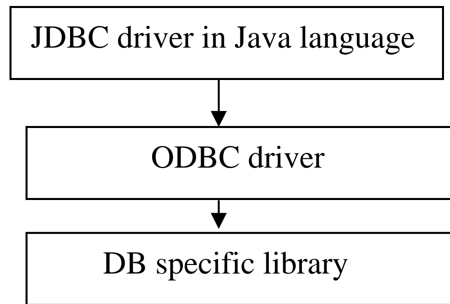
Connection, Statement and ResultSet are interfaces. They are properly implemented by its vendors. Here the creating objects are because of the interfaces are implemented as classes by vendors. The objects are of these classes. In the code we are not creating objects by interface because we cannot create objects by interface.

Types of Drivers:

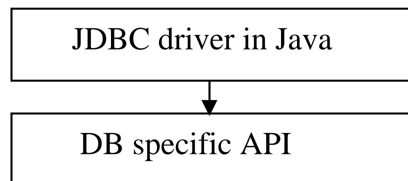
We have four types of drivers,

- Type1 or JDBC-ODBC Bridge
- Type2 or native-API partly Java technology-enabled driver
- Type3 or net-protocol fully Java technology-enabled driver
- Type4 or native-protocol fully Java technology-enabled driver (thin driver)

Type1:



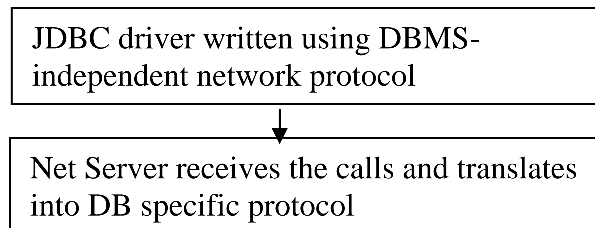
Type2:



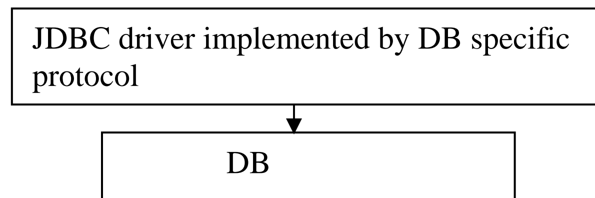
Problem with Type1 and Type2 drivers is we need to use additional software like DB specific client and ODBC driver software apart from java classes.

Type3 and Type4 drivers are implemented in Java.

Type3:



Type4:



Native API of Oracle is OCI. Incase of Type3 and Type4 we need not install additional softwares on the client machines but incase of Type3 on one of the machine we need to install netserver.

Oracle corp. has provided both Type2 (OCI) and Type4 (Thin) as part of the same set of classes (classesxx.zip).

If Type3, Type4, Type2 are not available then we can make use of Type1 driver supplied by JavaSoft or any other vendors.

JavaSoft has implemented Type1 i.e. JDBC-ODBC bridge.
URL: JDBC:ODBC:acc1
standard data source name (DNS)

When we use Type1 drivers we need to configure ODBC DNS as,
Start→Settings→Control Panel→Administrative Tools→ Data Sources (ODBC) → System DSN → Add → Microsoft Access Driver (*.mdb) → finish → select directory -- Name it that select from where you have data.

We can access the data available in the files using appropriate driver.

To execute data manipulation statements using *executeUpdate*. The return value of it is an *int* indicating the number of rows effected.

int nr = stmt.executeUpdate("insert into abc.txt values ('a''b''c')");

In case of DB like Quad base we can group a set of DB objects like Tables, views and store them in a schema (set of objects).
In case of Oracle we place the objects under a username i.e. username itself called as a schema in Oracle.

In case of Quad base we will establish the connection with the server by connecting with a specific schema.

As part of *executeUpdate* we can use any valid SQL statements. Ex: drop table, create table, create user, drop user …

Every DB server stores the Metadata in a set of standard tables. Ex: if you define a table the server will be storing the name of the table, the columns of the table and data types of the columns.

From our JDBC program we can get the metadata info using *ResultSetMetadata* object (*java.sql.ResultSetMetadata.getMetaData)*

Different DB vendors support different data types ex: SQL server supports BIT, SMALLINT, BIGINT, FLOAT, DATE, TIME, and TIMESTAMP etc… where as Oracle supports number, char, varchar, date …

You can find different types of SQL data types
http://www.idevelopment.info/data/Programming/java/jdbc/SQL_Datatypes_and_Java_Datatypes.html

There is as standard called SQL standard – defines the syntax of SQL and the data types that has to be supported by a DB server.

A DB vendor need not support all the data types of SQL standard directly. In JDBC *java.sql.Types* class provides a list of constants representing different SQL data types.

Metadata not only deals with data but also deals with drivers and versions. DB Metadata can be used to get the info about the JDBC driver we are using and the DB we are accessing.

Difference between *executeQuery* and *executeUpdate:*
executeQuery returns *ResultSet* type where as *executeQuery* returns *int* type.

How JDBC manages Transactions:

Whenever we start the operation the transaction is automatically started in case of JDBC. Once the transaction is committed/rollback the next transaction is automatically started.

In most of the DB applications being performed multiple operations on the DB in a single transaction by default. The JDBC drivers using Auto Commit Mode in this mode the driver issues the *commit* statement after we execute a SQL statement.
As most of the applications updates multiple tables as part of as single transaction we must not use Auto Commit Mode. For this we can use *Connection.setAutoCommit(false);* The transactions will not be committed automatically. *Connecton.commit()* makes the transaction committed permanently.
We can perform multiple operations in a transaction and commit/rollback the transaction using *Connection.commit() and Connection.rollback().*

For our java program when we execute *stmt.executeUpdate() or stmt.executeQuery* internally the server performs (a) parse the statement and (b) execute the statement.

In most of the applications we issue the statements like,
insert into tab1 values(1,2);
insert into tab1 values(2,3);
insert into tab1 values(3,4); repeatedly with a different set of values.
If we are use the *statement* objects to execute such a set of statements, the statements has be parsed every time. We can reduce this by making use of *PreparedStatement.*

For using *PreparedStatement* we need to follow the steps,

- Create a prepared statement using,
  PreparedStatement stmt = con.prepareStatement();
- Set the parameters [replace values (1,1) as values(?,?)]
- Execute the statement

Ex: *PreparedStatement stmt = con.prepareStatement("insert into tab1 values(?,?));*
stmt.setInt(1,100);
stmt.setInt(2,100);
System.out.prinln("No of rows effected = " +stmt.executeUpdate());

Parameters are called as bind variables in Oracle and defined with (:1,:2)
We will use *executeUpdate()* for *insert, update* or *drop* operations and *executeQuery()* for *select* operation.

URL depends upon the JDBC driver we are using.
To perform the DB operations we can use,

- *Statement* object
- *PreparedStatement* object
- *CallableStatement* object

To execute a *select* statement we can use *Statement.executeUpdate*
If the statement is executed successfully it will return an object of type *ResultSet.* From the *ResultSet* we can access the data retrieved by the query. From *ResultSet* we can access one row at a time, it internally maintains a pointer to the current row.

ResultSet.next() advances the current row pointer by one row and returns a true value if it points to a real row and a false value when it points to after last row.

There are imaginary rows with the real rows. They are (a) before first row (b) after last row

To get the column values of current row we can use *ResultSet.getxxx(column- no or column-name);* where xxx stands for String, int, float … This statement will fail when the current row points to imaginary rows.

In *getxxx* functions we can either pass the column-no or the column-name. Depending upon the column data type we need to choose appropriate getter like *getString, getInt …*

Most of the JDBC drivers support unidirectional *ResultSets* as well as bidirectional *ResultSets*. By default unidirectional *ResultSet* will be created. If we use a bidirectional *ResultSet* we can use the methods like *previous, first, last, absolute …*

To create a *ResultSet* that allows us to move forward and backward we have to create a statement object using,
Statement stmt = Connection.createStatement (ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);

By using *ResultSet.absolute(10)* we will directly go to 10th row.
Unidirectional result set are known as forward only resultset and bidirectional result sets are known as scrollable result sets.

In scrollable result sets we have scroll sensitive and scroll insensitive result sets.

Theoretically scrollable sensitive result sets are sensitive to changes and scrollable insensitive result sets are not sensitive to changes. But practically these are not implemented by the vendors. In JavaSoft documentation it is specified that sensitive resultsets are generally sensitive to changes.

The constant can be used to specify that we will read the values from the result set. The constant is *ResultSet.CONCUR_READ_ONLY* (concurrent read only).
A resultset can be opened for updation also. For this we need to use the constant. *ResultSet.CONCUR_UPDATABLE.*
We can develop our programs without using the features like *CONCUR_UPDATABLE, SCROLL_SENSITIVE.*

Not every driver available in the market supports scrollable and updatable resultsets.

These are the different ways in which the driver can be implemented.

Prepared statements: "select statements"

We can also use following queries like, *PreparedStatement stmt = con.PreparedStatement("select * from emp where sal >? And deptno=20");*

Don't use parameters in place of table names and column names we will get errors like "Invalid table name" and "Invalid no".
We have to use *stmt.executeQuery() in System.out.println()*

Update statements:

PreparedStatement stmt = con.PreparedStatement("update emp set col1=? where cal2 =?");
We have to use *stmt.executeUpdate() in System.out.println().*

Updatable Resultset:
We can read and update a row from a table ex:,

```
Connection con;
con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","scott","tiger");
Statement stmt = con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("select eno, ename, deptno from Emp");
rs.moveToInsertRow();           // creating the memory
rs.updateInt(1,1001);         
rs.updateString(2,"xyz");       // filling the memory
rs.updateInt(3,10);           
```

rs.insertRow(); // insert a row write the above values.
rs.close()

If in we *CONCUR_UPDATABLE* we can update the values and if we use *CONCUR_READ_ONLY* we can't update the values.
CONCUR_UPDATABLE is automated. Here it is internally using two SQL queries instead of single query by manually so it's better to go manually instead of automated. It will improve the performance.

Most of the Data Bases supports length in text (char long objects CLOB) and multimedia data like images, audio, video … using BLOB[Binary LOng Object]. Photo is of type BLOB in Oracle.
Directly we can't get photographs from the table. In-order to insert or get a photograph we have to write a program and the code is,

```
PreparedStatement stmt = con.prepareStatement("update bigtab set photo =
? where no = 10");
        File f = new File("verisign.bmp");
        FileInputStream fis = new FileInputStream(f);
        stmt.setBinaryStream(1,fis,(int)f.length());
        System.out.println("photo length = " + f.length());
        System.out.println("no of rows effected "+stmt.executeUpdate());
```

We can also insert by using *"Insert into bigtable values(?,?)"*
To retrieve the image ,

```
Statement stmt = con.CreateStatement();
ResultSet rs = stmt.executeQuery("select no, photo from bigtable");
rs.next();
System.out.println(rs.getString(1));
Blob b = rs.getBlob(2);
Byte b1[]= new byte[(int)b.lenght()];
System.out.println("blob length="+b.lenght());
fos = new FileOutputStream("x.bmp");
fos.write(b1);
fos.close();
```

To transfer the data i.e. available in a file to the DB we can use *setBinaryStream()* or we can also use *setBlob*. To retrieve the same data from the DB we can use *rs.getBlob or rs.getObject*

Blob is an interface. *getBlob* returns an object that provides the implementation of *java.sql.Blob*

Stored Procedure:

Most of the DB vendors support stored procedure that can be implemented directly in the DB.
Ex: (1) In Oracle we can use PL/SQL as well as java language to implement stored procedures.
(2) In SQL server we can use TSQL

Using JDBC the java programs can execute stored procedures stored in the DB irrespective of the language in which the stored procedure is created.
To call a stored procedure java programmers need to know the name of procedure, input parameters and output parameters.

To call a procedure we need to create *CallableStatement* using
*CallableStatement stmt=con.prepareCall("begin proc0;end;") //*oracle
CallableStatement stmt=con.prepareCall("{call pro0}") // any DB

If there are no input parameters we can create a *CallableStatement* using
con.prepareCall("{call procname}");

stmt.execute(); // to execute
SQLWarning sqlw =stmt.getWarnings(); // to capture the warnings

Once the *CallableStatement* is created we can execute it for any number of times same as a PreparedStatement.
If a procedure take parameters a callable statement has to be created using call procname(?n) – n is the number of parameters.

There are different modes in which the parameters can be passed.
In mode – the caller has to pass the value
Out mode – the caller receives the value after the procedure is executed.
Inout mode – the caller has to pass the value and it gets a return value after procedure is executed.

To call a procedure we need to identify the inputs [IN, INOUT] and outputs [OUT]. Ex:
Create or replace procedure proc1(inparam IN number, outparam OUT number, inoutparam INOUT number)

To call a stored procedure:

- Create a callable statement
- Set the input parameters
- Register the data types of output param
- Execute the procedure
- Read the output values

Ex:

```
CallableStatement stmt = con.prepareCall("{ call proc1 (?,?,?) }");
        int inparam=1;
        int inoutparam=33;
        //set input parameters
        stmt.setInt(1,inparam);
        stmt.setInt(3,inoutparam);
        //register output types
        stmt.registerOutParameter(2,Types.INTEGER);
        stmt.registerOutParameter(3,Types.INTEGER);
        stmt.execute();
        SQLWarning sqlw = stmt.getWarnings();
        System.out.println(sqlw);
        System.out.println(" output param ="+ stmt.getInt(2)+"  inout
parameter = " + stmt.getInt(3));
```

Functions:

In case of Oracle we can write the functions. A function can take input parameter and it can return a value. Default mode is IN.

To call a function the callable statement has to be created using the syntax given below
CallableStatement stmt = con.prepareCall("{?=call fun0(?)}");

Java program treats it as two parameters, parameter1 is return value and parameter2 is input value.

The function can be used in a select statement directly.
Ex: select sal, fun0(sal) from emp;

We can use a select statement with a function directly with statement object and prepared statement object.

In oracle only a table with the name dual is provided with single row and a single column. If a function has to be executed once, we can use a statement like “select function(parameters) from dual;”

There are two types of metadata used, (1) result set metadata – used to know the column name, data type, number of columns …
(2) DB metadata—used to know the version.

In the applications instead of hard coding the JDBC driver, URL we will be receiving them as parameters.

```
drv = System.getProperty(“Driver”);
url = System.getProperty(“URL”);
name = System.getProperty(“name”);
pwd = System.getProperty(“password”);
con=DriverManager.getconnection(url,name,pwd);
dbmd= con.getMetaData();
sop(“DB name” + dbmd.getDatabaseProductName());
sop(“DB version” + dbmd.getDatabaseProductVersion());
sop(“Driver Ver” + dbmd.getDatabaseDriverVersion());
sop(“User local files” + dbmd.Userlocalfiles());
```

If we use like this, DB metadata, we can give our application with no driver, URL, particular name and pass. The customer can set his own.

Introduction to 3 tier Architecture:
2 tier application – client/server
3 tier application – application using 3 tiers

In a 2 tired application the client/front-end is responsible for displaying the user interface, taking the inputs from the user and talking to the backend. The server is responsible for storing the data, manipulating the data and giving the data to front end as per the request of the client.

In every business application there will be user interface logic and business logic. User interface code is responsible for setting up the UI and allowing the user to interact with the application. Business logic is the code that implements business procedures.

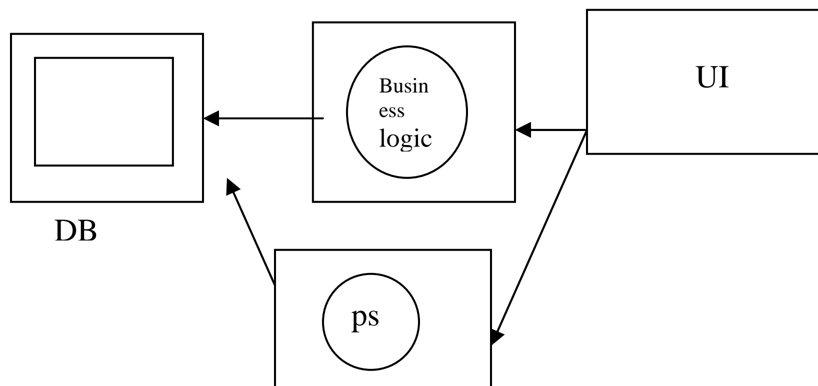
There are 2 approaches to develop 2-tiered applications,

- DB will be used to store the data, the UI logic and the business logic will be implemented in the front end.
- Only UI and the code that executes stored procedures will be written in the client and the business logic will be implemented in the DB server as a stored procedure. This approach is adopted by most of the companies because of,
  - Increasing the performance
  - Ease of maintenance

We can maintain the application easily if the business logic is separated from UI.

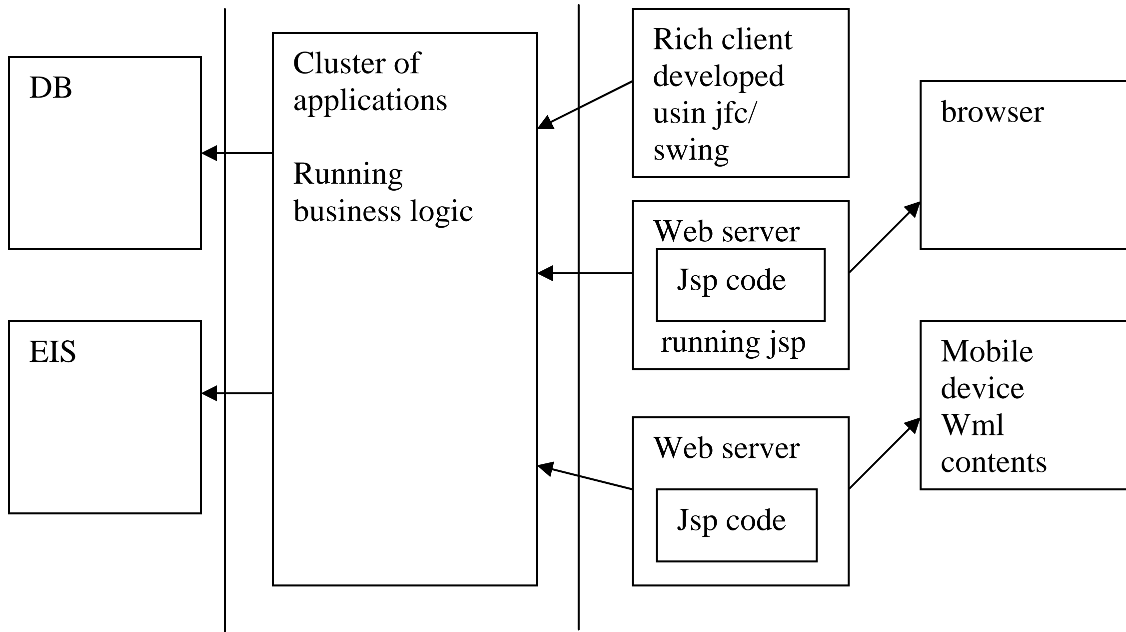
These applications are not scalable i.e. these are applicable effectively with less number of clients. If the number of clients increases the response time increase i.e. decreases performance.

To take care of scalability most of the companies has started 3-n-multi tired applications.



If the number of the clients increases addition of I more server to decrease the burden at middle tier.

3Tier Architecture:



We can develop a business application using the above architecture. The application server like Web logic, Web sphere, I-planet, sun one studio, jboss … can be used in the middle tier to implement business logic. This will be interacting with DB, existing Enterprise Information Systems (EIS).

For integrating different applications we can use wide variety of techniques. Most of the developers prefer to use XML for exchanging the data, web services, JMS, EJB and Java Connector architecture.

If we write a program as multiple pieces and deploy them on multiple machines, we can reduce the load on a single machine. To do this we need to write lot of networking code apart from the business logic and user interface. We can make use of the distributed technologies RPC, CORBA, RMI and DCOM to reduce the code we have to develop. With these technologies the network code will be generated automatically and developers using these technologies can concentrate on writing business logic and UI. The implementation of above technologies provides the tools like *rmic* for RMI, idltojava, idltoc++ … for CORBA and MIDL compiler for DCOM.

If we need to develop a client-server application we need to design a protocol. If we implement the network code on our own, we need to design a protocol that is used to send request fir execution of methods and sending the response.

In distributed computing before the data is send from one end to other end the data has to be packed or assembled this is called as Marshalling. At the receiving end the data has to be unpacked which is called as Unmarshalling.

In above distributed component technologies the programmers need not write the code for marshalling and unmarshalling.

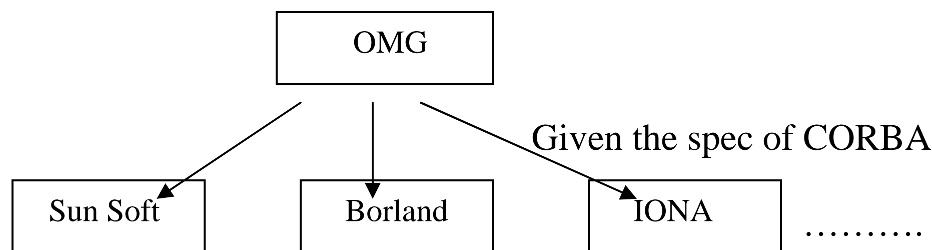
The above tools generate the stub used on client side and skeleton used on the server.

In case of java 1.2 RMI complier will not be generate the skeleton.

Main difference between above distributed technologies is the protocol.

CORBA – Common Object Request Broker Architecture

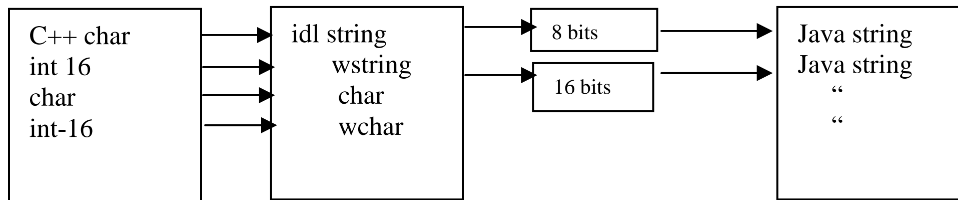
Corba is independent of language, hardware and software. It is a specification from OMG (Object Management Group).
The specifications are open and they can be implemented by any vendor.



Implemented CORBA for various languages by different vendors

The central piece of CORBA tech is Object Request Broker
For java programs ORB software is a set of classes and interfaces.
For c programs ORB is a library with a set of functions that can be called from our programs.
When we purchase a product from a vendor implementing CORBA tech, we get the ORB software.

OMG group has specified a language called Interface Definition Language (IDL). IDL language can be used to define the interfaces with a set of functions. As part of IDL language, OMG specified IDL Primitive Data types and specified how these data types has to be mapped in various languages. These are also called as IDL to java, IDL to C++, IDL to xxx…



Wstring – wide string

For ex, in IDLJava mapping OMG has specified how char, wchar, module, interface has to be mapped. The vendors are responsible for providing IDLxxx where xxx is a language.

As a CORBA developer we need to first identify the methods in an interface and create an IDL file with these methods.

A java programmer can use the ORB and the other tools supplied as part of JDK or he can use the ORB and tools provided by other vendors.

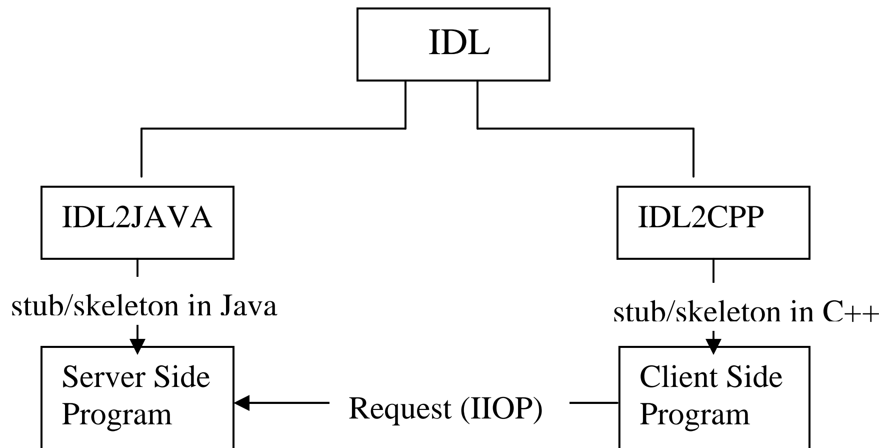
IDL2XXX tools checks whether the IDL file conforms to the IDL syntax or not and it generates the files in xxx language.

Modules in IDL language are equivalent to packages in java.
These xxx files provide the stubs and skeletons.

Microsoft DCOM is same as IDL language. Both are language neutral. Theoretically both are platform independent but MDCOM is partially implemented on UNIX platform. That’s why IDL has advantage over DCOM.

IIOP is the protocol used by CORBA where as MDCOM is used by Microsoft. Theoretically DCOM is almost similar to CORBA.

There are so many implementations of CORBA tech for different languages on different platforms. DCOM is implemented in Windows and UNIX.



CORBA uses IIOP (Internet Inter ORB Protocol). Java RMI tech uses RMI compiler in place of IDL2Java and takes a java file as input to generate stub/skeleton.

RMI compiler can generate stub/skeletons using JRMP (Java Remote Method Protocol) as well as IIOP.

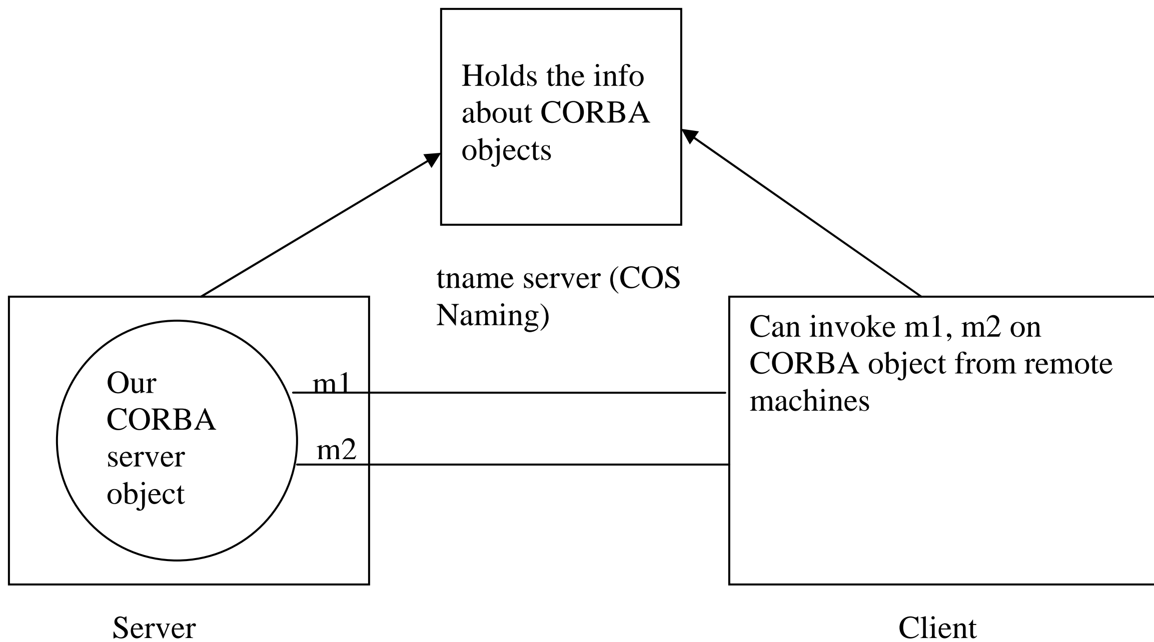
CORBA	RMI
IDL	Java interface
IDL2XXX	RMIC
Stub/skeleton for xxx using IIOP	stub/skeleton in Java using JRMP

Developer need not worry about the internal protocol used by stub/skeleton.

RMI internally uses CORBA nowadays.

Data type long in IDL is equivalent to int in Java
long long --- long

All the methods in interface are public but when we are in class, we have to declare the methods as public.



A directory server is a server program that can be used to store any info.
Ex:

- tname server (transient name server) is a server in which we can store info about CORBA object.
- RMIC – is a directory server that can be used to store info about RMI objects.
- NIS+ (Network Info Server +) is a directory server available on Solaris. This can be used to store user info, info about the other systems in the network, info about the hardware like printers. Similar to NIS+ Microsoft has Active Directory Service and Novel has Novel Directory Server (NDS)
- Yahoo, Bigfoot, whowhere uses LDAP [Lightweight Directory Access Protocol]
- DNS server used to hold the name and IP address mapping is also an example of directory service.

Java programmers can use Java Naming & Directory Interface (JNDI) to access the directory service.

As part of CORBA specification, OMG has specified different services like COS Naming, Object Transaction Service …

To develop CORBA/RMI/DCOM,

- First we need to decide about the methods that have to be exposed by the object.
- Develop an IDL file with an interface and the methods that have to be exposed
- Generate the stub/skeleton using the command,

  idlj file-name

  the above step creates an interface in java.
- Create a class implementing the interface.
  Use -fall to generate additional files that are required for implementation.
- Develop a server. In server, create a CORBA object.

In the server:

- Initialize ORB
- Create the CORBA object.
- Register the info about CORBA object in COSNaming server.

In old versions of IDLJ generates -*xxximplbase* as a skeleton. To generate this, *idlj - oldimplbase -fall greeting.idl*

From our CORBA programs, we need to use the package *org.omg.CosNnaming* to interact write the directory server.

In the client,

- We need to contact the directory server and get the info about the CORBA object.
- Invoke the methods on the CORBA object.

We will not directly create the stub objects. The stub objects will be created automatically when we contact the COSNaming server for the info on CORBA object.

ORB used to establishes communication between server and client.

As a programmer no need to establishing the communication link between the client and the server, it can be used to locate the objects.
Implementing CORBA applications:

1. Create idl with interface (with a set of functions)
2. Use idlj to create java stub/skeleton.

3. Implement a class by extending skeleton for CORBA sever objects
4. Write the server program.
    a. Initialize ORB.
    b. Create CORBA server object.
    c. Register the info of CORBA server objects with CosNaming server
    d. Wait for incoming request.
5. Write the client program
    a. Initialize ORB
    b. Get the stub object using CosNaming server.
    c. Execute the methods on CORBA object using the stub object.

Most of the directory servers allow us to store the data in a hierarchy or in a tree form as shown in the fig1. ex: CosNaming server. Some of the directory servers may not allow us to store the data in a tree form (flat structured as shown in fig2)

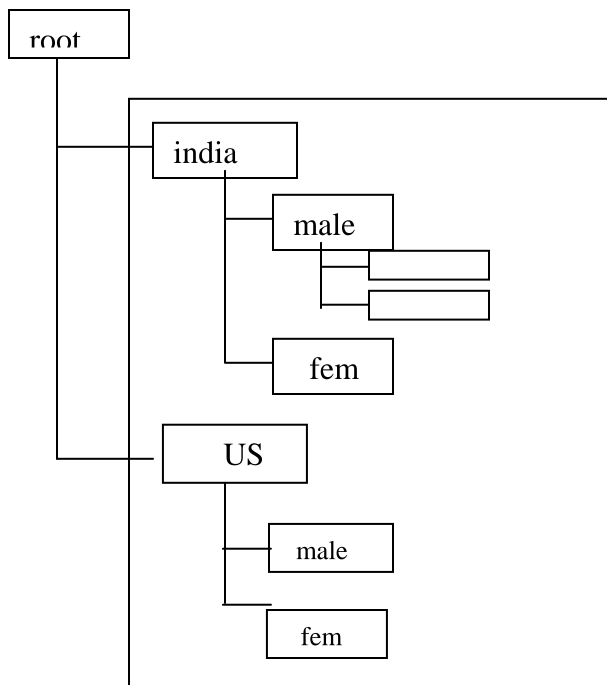


Fig 1

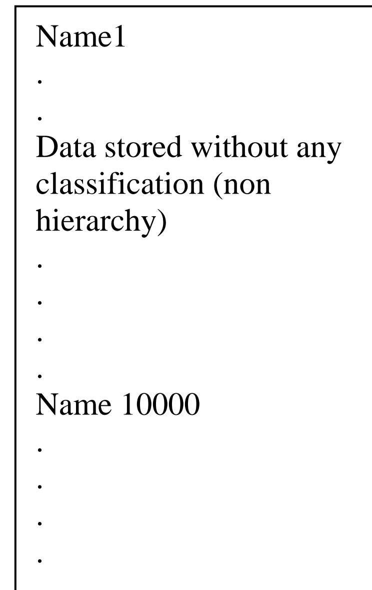


Fig 2

RMI registry implemented by SUN Microsystems provides a flat structure. In above ex in fig1 the items shown in blocks indicate the directories in a file system. They are known as contexts. A context may contain multiple sub contexts and/or multiple data items. Starting point is

known as Initial Context. To deal with a directory server we need to get hold of initial context.

As part of COSNaming server, we can create tree kind of structure and stores the data (info about the CORBA objects).

To reach a data item, we need to follow a path. In above ex, ic:India:female:above25:xxx is path to the data item xxx. In this, elements India, female … are called as path elements.

In CORBA methods are called as operations.
Most of the software developers use the naming convention xxximpl for the name of the class that provides the implementation of xxx.

The cosnaming servers are responsible for creating an initial context when the server is started.
If required we can create a sub context and bind (storing the info with name) the info about one CORBA object to a name under the sub context.

Binding is associating the info with a name.
Narrowing is converting a super class reference to a subclass reference.
Widening is converting a subclass reference to a super-class reference.

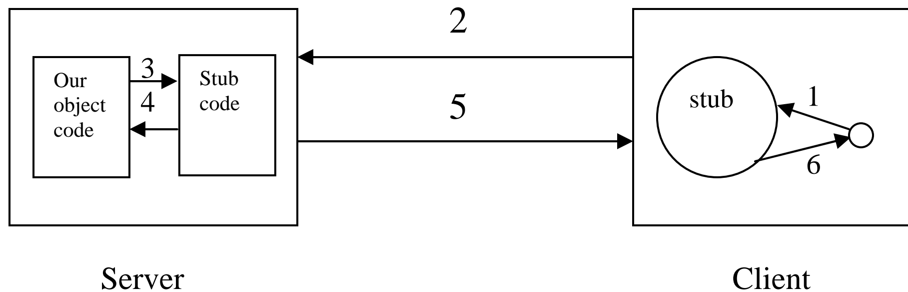
In the client programs the stub object will be created when we execute the *resolve( )*.

In our code we can use *thread.currentThread()* to identify the thread from which the code is executing.
In RMI as well as in CORBA, the methods on one server objects will be executed internally from different threads created by the CORBA/RMI implementation.

In RMI, CORBA, DCOM the following sequence of events occur when a client executes the method,

- Client executes the method on stub
- The stub sends the request to the skeleton
- The skeleton executes the method provided as part of the server object.

- After the method is executed the result will be given to the skeleton.
- The skeleton passes back the result to stub
- The stub gives the result to client.



Connection between server and client

In real time applications we will be using same server program to deploy our CORBA server objects.

From our programs we can detect the value of the current JVM version using the system property "java.vm.ver" as,
System.getProperty(Java.Vm.Ver)

The environment variable values can be obtained by using getenv() in c language. In java in old versions we can use System.getenv(). Use getProperties() instead of getenv in new versions.

Remote Method Invocation (RMI)

In the old versions of Java (before 1.2), RMI implementation requires a skeleton. From 1.2 onwards RMI is re-implemented. In this version skeletons are not required (internally it will take care).
When we run the RMI compiler, it generates the stub and skeleton if we use v1.2 flag (version 1.2) otherwise it will generate only stub. The compiler generates the class files directly.

To see the java source files generated by the rmic we can use the flag "*keepgenerated*".
We can use the flag *-iiop* to ask the compiler to generate stub and skeleton according to iiop protocol.

RMI can be implemented by any vendor but today Java soft is recommended all the vendors to support iiop protocol.

To develop a RMI application:

- Define an interface by extending *java.rmi.Remote.* As part of this interface we can define any number of methods. These methods can throw any exception but we need to declare that these methods will throw *java.rmi.RemoteException.*

- Compile the interface.

- Provide the implementation of the interface. The implementation class must extends from *java.rmi.server.UnicastRemoteObject.*

- Compile the implementation class.

- Using *rmic implemenationclass* generates the stub and skeleton.

- In RMI server program:
    a. Create the server object
    b. Register the server object info in rmiregistry

  For storing the info about our object we can either use.
  Naming.bind() or Naming.rebind().
  We can start rmiregistry by using *rmiregistry (or)rmiregistry portno*

When the port no is specified to startup the registry, the server and client programs must specify the port no.
If the rmiregistry is started without a port no, it makes use of default port no 1099.
Naming.rebind("rmi://localhost:1099/servername", remoteobject);
We may face a problem like *port is already in use* while starting the rmi registry if the port no is already running in some other program.

- In the client :
  a. Contact the rni registry using the name of the server object and get it.

  Object obj= java.rmi.Naming.lookup("rmi://localhost/server");

  In RMI client the stub object will be created when the *lookup* operation is performed successfully.

  We can register the info about a single object using multiple names.

  It's an error if we use same name for multiple objects. We cannot have multiple objects registered with single name.

bind() method fails if the info is already stored with the same name. *rebind()* always succeed even if the info is already stored with the same name.

The state of an object depends on instance variables. Default values of instance variables are 0 and false.
The state of the object is represented by the values of instance variables.
We cannot change the state of an object after it is created such an object is called an *immutable object.* Ex: String object. If we can change the state of an object it is called as *mutable object.* Ex: string buffer

Serialization is the process of writing the state of the object in a stream. A stream can be *FileOutputStream, SocketOutputStream …*
Deserialization is the process of recreating an object by reading the data from a stream like *FileInputStream, NetworkInputStream…*

For more info on serialization follow the link,
http://java.sun.com/docs/books/tutorial/essential/io/serialization.html

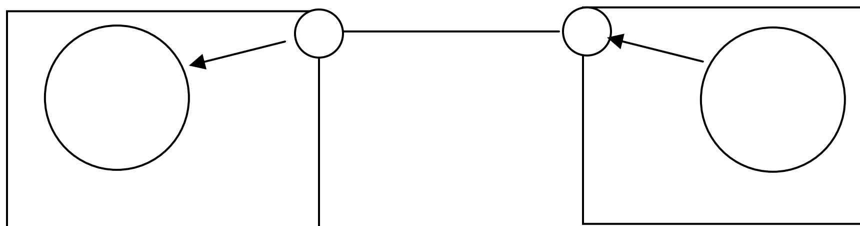
While designing the projects the developers prepare the class-diagrams, object state-diagrams and object interaction diagrams.

A state diagram gives the info about different states in which the object can exists and it also gives the info about how we can move from one state to other state.

Object interaction diagram gives the info about how different objects interact with each other. Most of the developers use Unified Modeling Language (UML).

You can find more info on UML and UML diagrams at,
http://www.developer.com/design/article.php/2238131
http://unicoi.kennesaw.edu/~jbamford/csis4650/uml/UML_tutorial/diagrams.htm

Not every object can be serialized. Ex: Current News
We use *FileOutputStream* to write and *FileInputStream* to read.
To transfer the objects from one JVM to another JVM we can use serialization and deserialization.



Flattening the object

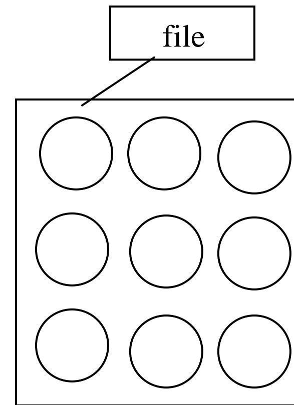
Most of the application servers maintain the info about multiple objects in a file or in a database if the space is not enough using serialization and deserialization.

Passivation is the process of writing the state of the object to a persistent storage (file/DB) and removing the object. Activation is the process of creating the object by reading the data from persistent storage.

Suppose if the space in service is to hold 9 objects and we have 10 objects to hold. At that time serialization and deserialization techniques are useful. We keep one object in a file or in DB i.e. serialization.

The object is passive or inactive.
Now there is room in server to place 10th object.

Again when we want to use the 9th object, we will keep 8th object in the file and we reconstruct the 9th object from the file in the place of 8th object i.e. we are making active objects into passive and vice versa.



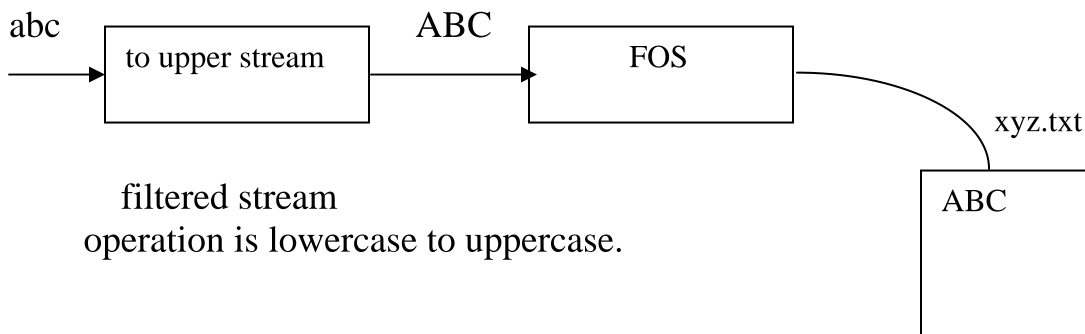
As part of the standard java classes, we can serialize objects like String, StringBuffer.
If a class implements *java.io.Serializable* interface then we can serialize the objects.

Some of the examples of non serializable objects are System, Socket, DBConnection …

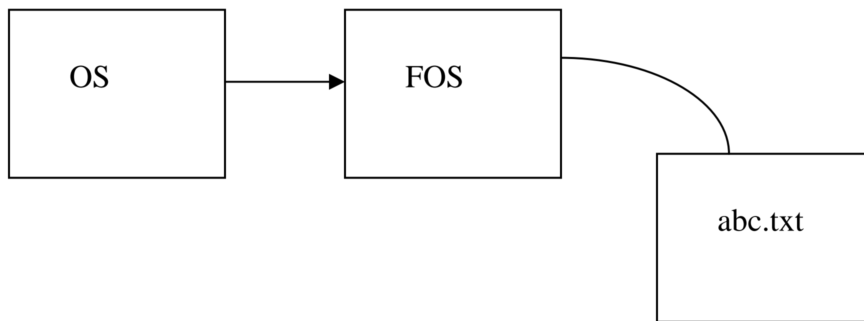
When we design our classes we need to decide whether it makes sense to serialize our objects or not.

In java we can use filtered streams to perform filtering operations. Filter actions like converting lowercase to uppercase, compression, uncompress, removing the spaces.

For serialization and deserialization we can use the classes *ObjectOutputStream, ObjectInputStream.*



FileOutputStream fos = new FileOutputStream("abc.txt");
ObjectOutputStream os = new ObjectOutputStream(fos);



The 1st statement open a file with the name abc.txt and 2nd statement creates object output stream and creates the link between object output stream and file output stream as shown in the diagram.

In this ex, when we write the data to object output stream using writexxx() methods. The data will be alternately stored inside the file abc.txt.

os.writeInt(i);
os.writeInt(b);
os.writeFloat(x);

When we use writexxx() methods, the data will be written using Object Serialization Protocol.

Reading the data from a file:
FileInputStream fis = new FileInputStream("abc.txt");
ObjectInputStream ois = new ObjectInputStream(fis);
System.out.println(ois.readInt(i));
………
For strings:
os.writeObject(str);
System.out.println(os.readObject(str));

Socket object: we are unable to create socket object,
java.net.Socket s= new java.net.Socket("127.0.0.1",80);
os.writeObject(s); --- it will give an error.
When we develop our classes we can decide whether the objects of our classes are serialized or not.
java.io.Serializable can be implemented by a class. By implementing this interface we are saying that our objects can be serialized.

class myobj implements Serializable {
int I;
...   }

There are no methods defined as part of serializable interface such as interface is called as Marker Interface—interface without an object and marker gives more info about the object.

While serializing the instance variables will be return to the stream but not the class variables.

If we use an instance variable which cannot be serialized then the serialization of the object fails. In such a case we need to declare the instance variables as transient variables. A transient instance variable will not be written to a file
transient Socket s; (o/p is null).

If we have special requirements like,

- Storing the info about the class variables.
- Storing the info about transient variables.

We need to provide *readObject()* and *writeObject()* methods by our own.
We can use *readObject()*, *writeObject()* methods in above cases.
The above two methods must be declared as private.
private void readObject(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException { …… }
private void writeObject(java.io.ObjectOutputStream stream) throws IOException { …… }

When we provide the above methods internally *writeObject* will be called during Serialization and during deserialization *readObject* will be called.

By default it's not possible to write or read a socket object. Based on our requests we will write the IP address and port no. While deserialization we will read these IP address and port no and creates a new socket.
Most of the tool vendors define their own protocols of storing the objects. In such case our classes must implement *Externalizable* interface which extends *Serializable* interface.
In this interface we have the methods *writeExternal()* and *readExternal()*.

Externalization is used to implement our own objects.

There are two types of objects (1) Serializable and (2) non-serializable
To identify whether the object implements serilizable or not we can use, *isInstanceOf()*

Good info on serialization http://www.javacaps.com/java_serial.html

If the object is remote we can invoke the methods from different machine and if the object is local we can invoke the methods from same machine.

According to RMI specification, an object that provides the implementation of *java.rmi.Remote* is called as Remote object.

To develop an RMI solution:

- Identify the methods that have to be exposed by the Remote object.
- Define a remote interface with the above methods following the steps
    a. Our remote interface must extend from *java.rmi.Remote.*
    b. Define the methods by following the rules
        i. The methods must throw *java.rmi.RemoteException* apart from application specific exceptions.
        ii. All the parameters and return types must be (1) java primitive types (2) objects that are serializable (3) it can be a remote type. (parameter restrictions is because internally RMI uses serialization)
        iii. Compile the interface.
- Provide a class implementing the interface
    a. Create the class extending *java.rmi.UnicastRemoteObject* implements our own interface.
    b. Provide the implementation of the methods.
    [*java.rmi.Remote* identifies whether the object is remote or local
    RMI system throws *RemoteException* because of three reasons (a) System crash (b) Program crash (c) Problem in network connectivity. These exceptions are not under our control.

Java soft has defined different exception classes as sub classes of *RemoteException* whenever there is a problem like executing a method that will be reported by throwing the above exception. We can develop our own class without extending *UnicastRemoteObject*]

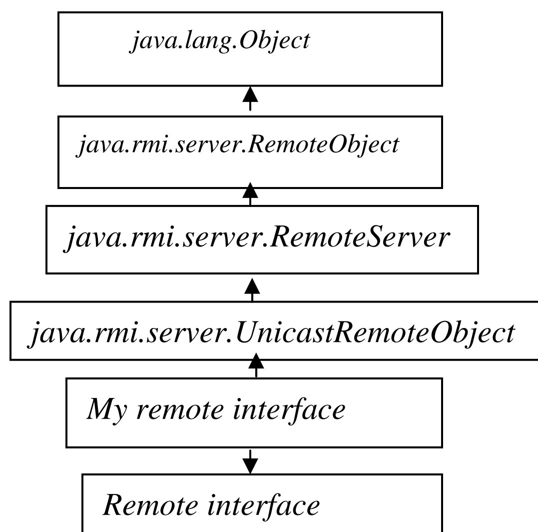
- Compile the class xxximpl
- Use rmic and generate stubs/skeletons [*rmic xxximpl.java*]
- Develop the server
  a. Create server object.
  b. Register the server object with rmiegistry.
  c. Compile the server
- Develop the client
  a. Lookup for the server object. This creates the stub object.
  b. Call the methods on the server object.
  c. Compile the client.

http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

The implementation of *toString(),equals() and hashCode()* of *java.lang.Object* are implemented keeping the objects in mind. These are reimplemented in *java.rmi.server.RemoteObject* to sent remote objects.

java.rmi.server.RemoteServer provides the methods like *getClientHost, java.rmi.server.UnicastRemoteObject* provides the methods like *exportObject(), unexportObject().*

Class Hierarchy



To make a remote object available to the external world we need to manipulate and export the object.

When we directly implementing the remote interfaces we need to provide the methods like *toString(), hashCode() and equals()* and we need to write the code to export the object.
When the object is exported the port no, the IP addresses will be decided and every object is identified using an ID.

The *UnicastRemoteObject* constructors will be internally calling *ExportObject().* The advantage of *UnicastRemoteObject* is exporting is done automatically.

Remote object is not creating directly. It is created by extending *UnicastRemoteObject* because exporting is done automatically. No need to write extra code to export.

If an object can remember the conversational state of the client then we can say that the object is stateful object. If an object cannot remember the state of the client then the object is called as stateless object.

If we have 'n' number of server objects then we can maintain the state of 'n' objects. It's not possible to maintain the state of 'n+1' clients.

It is not possible to anticipate the number of clients so we cannot write our server program to create the number of objects using the technique shown in part4 (in Suresh Examples) of RMI.

As part of a single JVM there can be multiple class loaders. Class loaders are used to download the class files from web server in applet.

By default the JVM used an internal class loader to load the classes from different directories specified in classpath.

When we need to download the classes from a source like http server or a DB server then we need to implement our own class loaders.

We can configure the web servers to serve the class files.

```
import java.rmi.registry.*;
locateRegistry.createRegistry(1099);
```

We can directly register through our program using above statements. For this no need to use rmiregistry.

To simplify the maintenance and insulation of client applications RMI provides an option of loading the classes from web server, http server…

XML

When we create .rtf document apart from saving the actual info the tool saves additional info like start of a paragraph, bold, size of the font ... Etc. This info will be used by the tools to display the actual info.

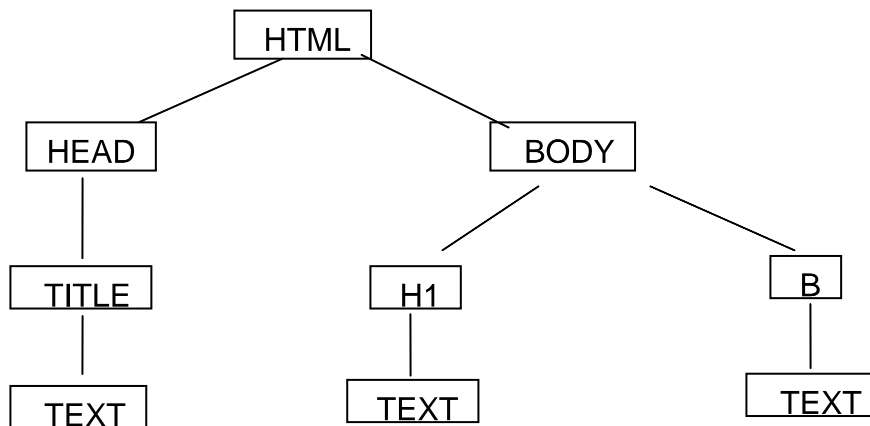
In html also we use several tags to tell the browser how the data has to be display. In html we use tags like H, B, FONT, TABLE… etc to tell how the data has to be formatted.

As part of html pages we can use an entity like nbsp, copy, lt, gt, reg … Entities can be used for replacement.

By looking at html tags we can say how the data has to be displayed but we can't understand the semantics of the data.

Every html document can be represented as DOM tree (document object model)

DOM tree:



All the boxes are different objects representing various html elements.
Object can be called as nodes.
Various scripting languages like java script support the DOM.
In xml we can use a DOM parser to read the document and create the objects representing a tree.

Validation of XML

There are so many tools available that can validate the html documents. www.w3c.org provides a facility to validate the html files.

Meta languages like sgml and xml can be used to define a new language like html.

Xml can be used to define our own language; we can define our own tags either by using DTD (document type definition) or xml schema standard.

A well form xml document is a document containing at least one element.

- For every start tag there must be an end tag.
- All the elements must be nested properly.
- All the attribute values must be placed in quotes.

A valid xml document is a well formed xml document and it is written according to xml schema or xml DTD.
An element in xml can have attributes, element content, simple content, empty content or mixed content.
An empty element can be written either as

<abc atr1="value"> </abc>

(or)

<abc atr1="value" />

Attributes are used to provide additional info about the data

Limitations on Attributes :

- Can't contain multiple values
- Not easily expandable
- Can't describe structures
- More difficult to manipulate programmatically
- Values are not easy to test against a DTD

Most of the xml authors' use attributes to store Meta data

In xml documents we can use standard entities like <, > …etc

DTD

Xml DTD can be used to define our own language.
In a DTD we define

- Root of the document
- The content of an element
- The sequence in which the elements has to appear
- Define the entities and attributes of an element

Ex :
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT page (title+, content, comment?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA)>
<!ELEMENT comment (#PCDATA)>

+ one or more
? Zero or one
* Zero or more
PCDATA or CDATA contains characters or simple content

We use ELEMENT to define content of an element
<! ELEMENT element name (content)>

To define attributes we can use,
<! ATTLIST element name attribute name attribute type default value>

In ATTLIST we can specify list of valid values

We need to specify whether the document is valid or not according to DTD

We can directly have DTD as part of an xml document called internal DTD.A DTD can be external.

In the xml file we can refer to an external DTD using DOCTYPE as,
<! DOCTYPE root element system “file.dtd”>

A name space is a set of unique names.
In java we use a package to provide a name space ex :

Since both of them are two different name spaces we can have the same class name socket.

In above case also there will be ambiguity if we refer to names directly. To resolve the ambiguity we can use the fully qualified names ex: namespace-name followed by name in namespace

com.inet.package.socket

While debugging the tags we can define the name of the namespace

In xml we can write the fully qualified name as,

xyz:Element where xyz is namespace prefix

According to xml specification the name of the namespace must be an URI

Instead of writing *<http://www.w3.org/.....:author>* we can define an alias for namespace and use it as,

<h : ELEMENT xmlns: h =“name of the namespace”>

Where h is prefix

Xml schemas

In xml DTDs we can’t specify the data types and can’t specify the restrict on the values.

Xml schema can be used in place of DTDs. The standard schema language supported by all the software vendors today is w3c xml schema language.

Why schemas:

- Extensible to future addition
- Richer and more useful than DTDs
- Written in xml

- It support data types and namespaces

Xml schema uses xml syntax

W3c has defined a set of tags as part of the name space www.w3.org/2001/xmlschema. (You can download the xsd file www.w3.org/2001/xmlschema.xsd or dtd file www.w3.org/2001/XMLSchema.dtd).

Some of these tags are *ELEMENT, SIMPLE TYPE, COMPLEXTYPE, SEQUENCE, STRING, DATE……* Apart from this name space w3c has defined another name space *www.w3.org/2001/xmlschema-instance* (www.w3.org/2001/XMLSchema-instance.xsd)

When we have to define our own tags we have to use the above tags defined by W3C.

According to xml schema if an element contains other elements or attributes the element is known as complex. You can see different types of elements at http://www.cs.rpi.edu/~puninj/XMLJ/classes/class4/all.html

Parsers

We can produce the xml files directly without using any additional software. To consume xml we need to use a parser.

In our application for reading xml data we can use xml parser. Most of the parsers available in the market can validate the xml files against xml DTD as well as xml schema. These parsers are responsible for reading xml content, breaking that into multiple tokens, analyzing the content and gives the content to our program.

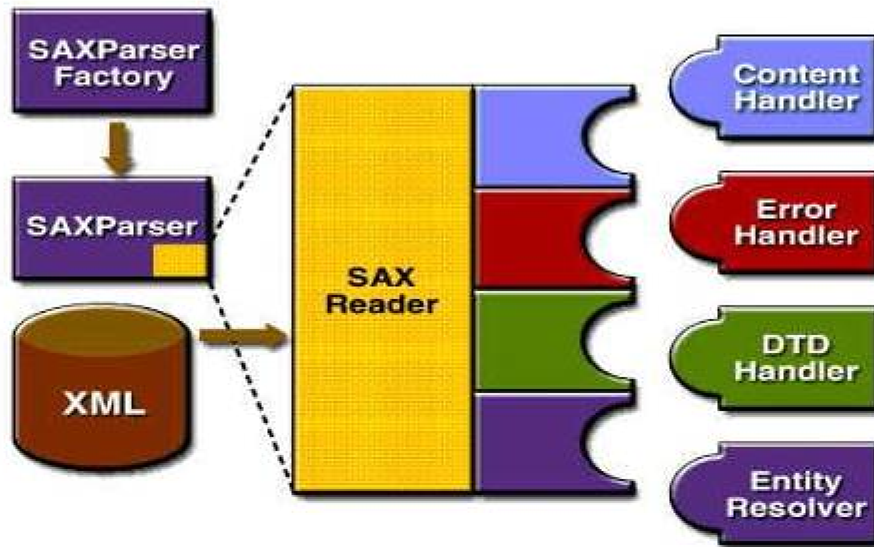
There are different types of parser soft wares available in the market. We can use these soft wares using JAXP.

Mainly there are two types of parsers

a) Sax parser b) DOM parser

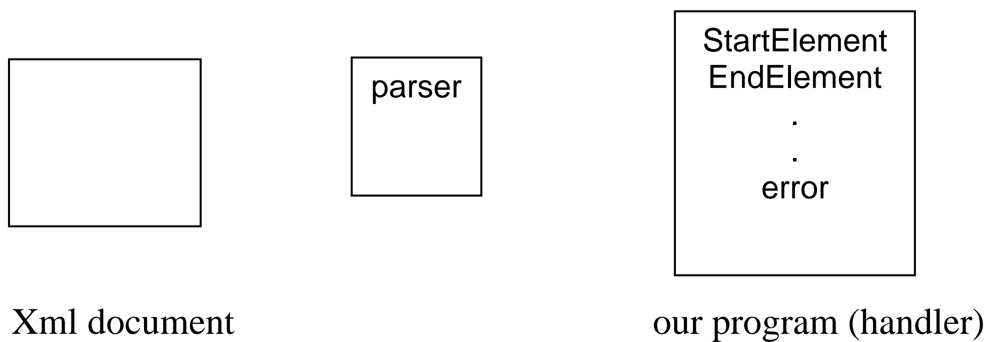
SAX PARSER:

You can see details of sax parser at this site http://www.saxproject.org/



SAX PARSER

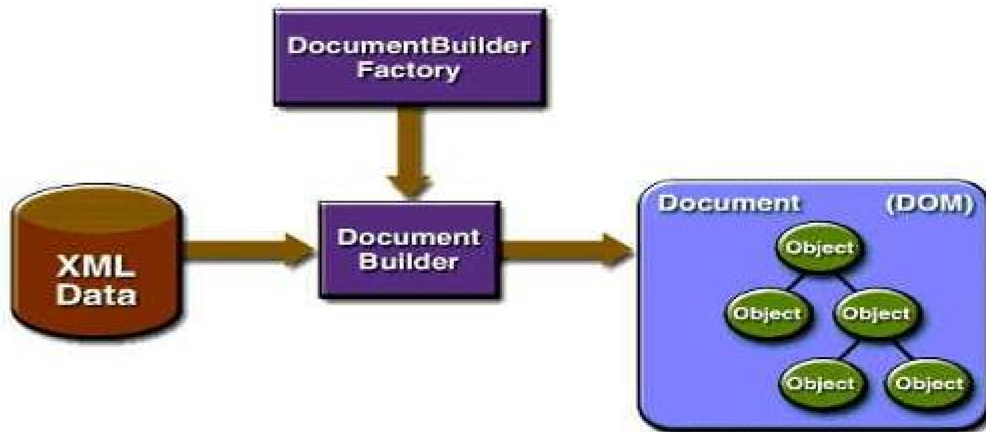
Processing xml document through sax parser :



Xml document our program (handler)

To use a SAX parser we need to create a handler with a set of methods *startDocument, endDocument, startElement, endElement* … and create a link between the handler and the parser.
Sax parser reads the XML content serially and it calls various methods for ex: when it sees the start of the element like Student, Name, Rno parser calling the method *startElement*, when it discover any error it calls Error.

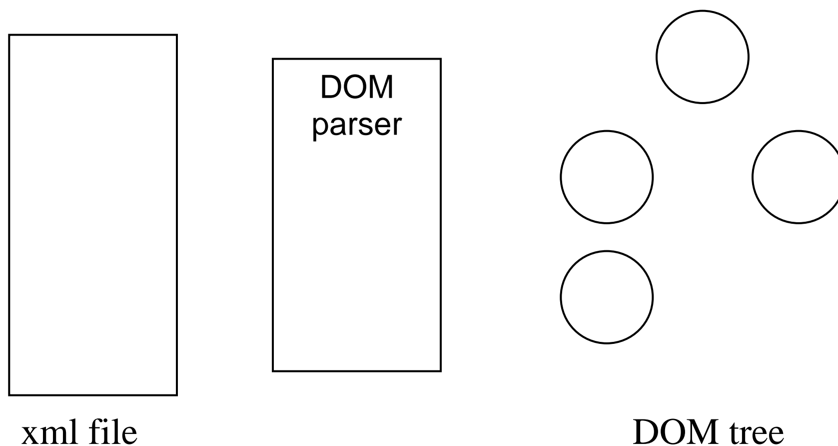
DOM PARSER:



DOM PARSER

For more info DOM parser
http://www.cafeconleche.org/books/xmljava/chapters/ch09.html

A DOM parser reads the XML file if there are no errors it produces a set of objects represents the DOM tree. Even though it is easy to program the applications will perform very slowly as the parser need to create too many objects.



xml file DOM tree

We need to use SAX parser factory to create SAX parser.

If the document is not well formed the parser will detect the errors and calls the appropriate method in error handler.

To validate the XML file we need to set the property validating = true as,
    parserFactory.setValidating(true);

In a standalone file we can't make reference to external entities. While creating XML documents to improve the readability we will be using the spaces, tabs, new line characters – these are called as ignorable white spaces.

We need to implement *org.xml.Sax.DocumentHandler* interface that has the method *setDocumentLocator, startDocument, startElement* …. If errors have to handle we need to implement error handler.

Before the parser starts parsing a document it calls the method *stLocator()* by passing an object Locator.

When the methods in error handler are called, the parser parses *SaxParseException* object. We can execute the methods like *getLinenumber, getSystemID, getMessage*.

As soon as the parsing is started the parser calls the method *setLocator*. After it calls the method *startDocument* – after this depending upon what it has seen (start of element/ end of element…) it will call the appropriate methods. By using the locator object we are able to find the location where the parser is currently parsing.

When the *characters*() is called the parser gives us a buffer which contains the character that it has parsed plus some other data.

In characters method we need to start reading from offset up to length.

We can create our handler classes by extending *org.xml.sax.helpers.DefaultHandler*. This class provides the implementation of the handler interfaces.

Similar to *setValidator = true* we can use *setNamespaceAware* to true indicating that the parser must using the namespace.

To validate an xml file against xml schema we need to specify

(i) Schema language
(ii) Schema source.

Even though we can specify schema file in xml file it is recommended to programmatically set the values.

Code we develop for xml files using xml schema as well as xml DTDs is same except

- Set the schema language
- Set schema source
- We need to setup NamespaceAware to true.

To use the DOM parser:

- Creates a document builder factory
- Using factory creates a document builder
- Ask the document builder to parse
- If there is no problem in parsing the parser returns an object of type Document.

org.w3c.dom.Document is an interface and it is implemented by the parser vendors. In our code we will not creating objects directly.

Once the document is created we can execute the *getDocumentElement* method which returns the root element. Using root element we can get child nodes using *root.getChildNodes(return type nodelist)*

For every node we can find out its type, name, value, parent, child nodes using *hasChildNode, hasAttribute, getAttribute* ….

SERVLETS:

Introduction to web applications

HTTP is used by the browsers to communicate with the web servers. In place of browser we can develop our own programs using Socket API to communicate with a web server.
According to HTTP the user agent/http client/browser sends a request for a resource and gets back response. According to the protocol the web server/http server sends the response with a status code.

Most of the web servers are configured not to list the contents of a directory. Almost every web server provides an option of defining a list of files as welcome files.

TOMCAT:

Tomcat_home is a directory in which TOMCAT is installed. Ex: C:\Tomcat\Tomcat5.0 (most it look like C:\ jakarta-tomcat-5.0.19/jakarta-tomcat-5.0.19 when you extracted zip file to C drive)

Starting and Stopping Tomcat :
To start Tomcat run *Tomcat_Home\bin\startup.bat*
To stop Tomcat run *Tomcat_Home\bin\shutdown.bat*

Tomcat requires the location in which java is installed for this we need to add set *JAVA_HOME="path where jdk is installed"* and add set *CATALINA_HOME=..* in both *startup.bat* and *shutdown.bat*.

If you are not installed the tomcat, for accessing Admin and Manager we need to add
<role rolename="manager"/>
<role rolename="admin"/>
<user username="uname" password="pwd"
roles="admin,manager,tomcat"/>
to *Tomcat_Home\conf\tomcat-users.xml*

Tomcat is configured to listen at port no 8080. We can change the port no using configuration files.*(Tomcat_Home\conf\server.xml file change <Connector port="8080" ... >).*

Request and Response formats:

Request format:
Initial request line
header1
header2
header3
.
.
.
Body (optional)

Most of the browsers either implements http 1.0 or 1.1 protocol.

According to http, initial request line contains

- Request method
- Requested resource
- Protocol version

Header is split into (1) Header name (2) Header value
Ex: User-Agent : MSIE

Response Format:

Initial response line
header1
header2
header3
.
.
Body

In the response the server sets several heads like

- Server ----- Name of the server
- Content length ----- Length of the content

MIME is standard format in which different types of contents can be placed standard format used by e-mail clients

Status Codes:

1xx	Informational message only
2xx	Success of some find
3xx	Redirects the client to another URL
4xx	Error on clients part
5xx	Error on servers part

Standard Request Methods:

- GET Used to request a resource
- HEAD Used to request only header part
- POST Used to send the data from client to server ex: when we will fill the form the browser may send the data
- OPTIONS Used to identify the options supported by the browser
- PUT Used by the clients to ask the server to store the content on it
- DELETE Used to ask the server to delete a file

Diff Btw Get and Post :

In our programs most of the time we will use *GET* or *POST* methods.
In HTML forms we can specify the method as *POST* or *GET*
When we use GET method the browser appends the form data to the URL but when we use POST method the data will not appended to the URL. Data will be placed as part of request body.
In order to upload files from browser to web server we can use *<INPUT Type= "file">* for these types of forms we need to use *POST* instead of *GET*.

Web-Containers:

Java soft has released Servlet and JSP specifications publicly. There are so many vendors who has provided the implementation of these specifications. All these products are known as web containers.
A web application developer develops a web application using different resources like html pages, image files, servlets, jsp pages, ……………..

Configuring web applications:

Java soft has defined a standard approach for configuring web applications. Once the web application is configured we can deploy it on any of the available web containers.
Steps:

- Create WEB_APP_ROOT directory
- Under WEB_APP_ROOT create WEB_INF
- Create classes and lib directories under WEB_INF (place java classes in classes directory and .jar files in lib directory)
- Create *web.xml* and place it under WEB_INF
- Create a WAR (Web Application Archive) file using java tool for this move to WEB_APP_ROOT dir and use *jar cvf webapp.war* .

Above command places all files in *webapp.war*

We use content paths to identify the web applications. Ex:
http://localhost:8080/myweebapp1/index.html

By default index.html is consider as a welcomefile. According to our application we can configure a different set of welcome files.
Ex:

<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
</welcome-file-list>

The web containers will not be sending the information about WEB_INF directory to the user agent.

Even though we can place all the files under the root it is not recommended. It is not recommended to use the absolute paths of the resource in a web application. We have to use relative paths.

Managing web applications in web containers:

Most of the vendors provide the tools to manage the web applications. We can use these tools to upload our war files to the server and deploy them.

Ex: In Tomcat

- Use the URL http://localhost:8080
- Choose Tomcat Manager (by using uname and pwd given in config file)
- Choose the war file to be uploaded choose install

In Web Logic

- Login to web logic server using admin uname and pwd
- Choose deployments web applications node
- Choose configure a new application.
- Choose upload the war files
- Select the war file place it on server , configure and deploy it

Today most of the websites generates the content dynamically (when a request is send the web server, the content is generated)

First generation web servers are designed to serve the static content. Later on the web servers provided an option of extending its capability using CGI

A CGI program or script is the one that runs external to a web server.

Earlier most of the programmers have used CGI programs to generate content dynamically.
The main problems with this approach are

- Server has to spend some time in starting a new process
- As web server gets too many requests the server has to create more no of processes and this reduces the performance.

As scripting languages are easier than programming languages most of the web developers has started writing web applications using scripting languages.

To run the perl scripts we require a program called perl interpreter. Even it is easy to develop perl scripts we need an interpreter to run them.
Later on most of the web server vendors started supporting different scripting languages directly.
Java soft designed an alternative for CGI programs as well as script getting executed in a web server called as a Servlet.

In servlets the request send by the browser and the response send by the server will be represented as objects.

While compiling the java servlets we need to set the classpath point to a jar file that contains *javax.servlet* and *javax.servlet.http* packages (in most cases the jar file will be *servlet.jar* or *servlet-api.jar* or ….).

Every container vendor supplies the jar files that contain *javax.servlet* and *javax.servlet.http* packages.

<u>Deploying servlets</u>:

After developing the servlet classes to deploy the servlets,

- Copy the servlet classes under WEB_APP_ROOT/WEB-INF/classes
- Modify *web.xml* by adding the information about the servlet element as

```
<servlet>
<servlet-name>servlet1</servlet-name>
<servlet-class>class of servlet</servlet-class>
</servlet>
<!-- mapping our servlet -->
<servlet-mapping>
<servlet-name>servlet1</servlet-name>
<url-pattern>/serv1</url-pattern>
</servlet-mapping>
```

We can use any url pattern like * or *.do or *.xyz etc… ex:

```
<url-pattern>/*</url-pattern>
```

To get the name of the user agent we can use
stringbuffer = request.getHeader("USER_AGENT");
We can get the list of all headers using
Java.util.Enumuration list = request.getHeaderNames

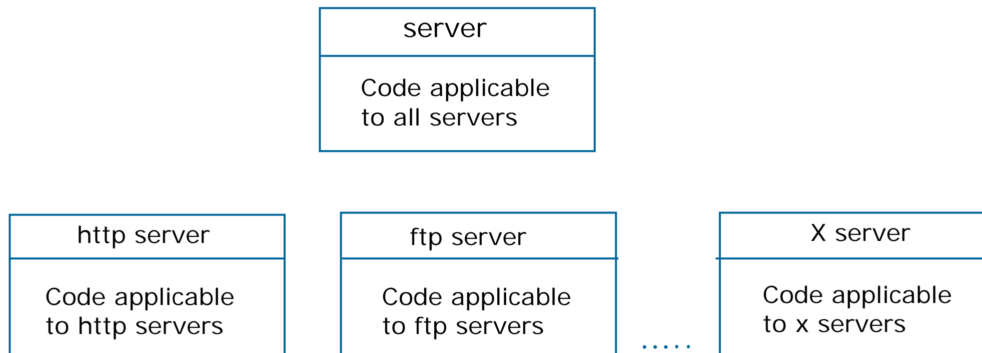
To invoke a servlet from our applications (standalone java apps/applets/ servlet/ejb …)

- Establish a socket connection with web server
- Send http request to the server

- Read the response

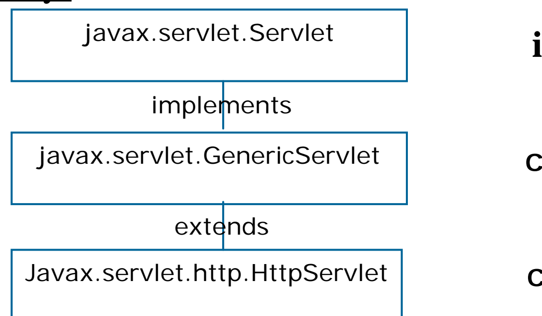
Servlets are initially designed to extend any type of server (web server/ ftp server / smtp …) but almost all the companies has implemented this technology to extend http servers.

In object oriented projects to eliminate redundancy we will be using class hierarchy as,



In above ex, code specific to web server in http server class, code specific to ftp server in ftp server class … and code that is common to all the servers is implemented in server class.

Servlet class hierarchy:



A servlet is an object that provides the implementation of *javax.servlet.Servlet* interface.

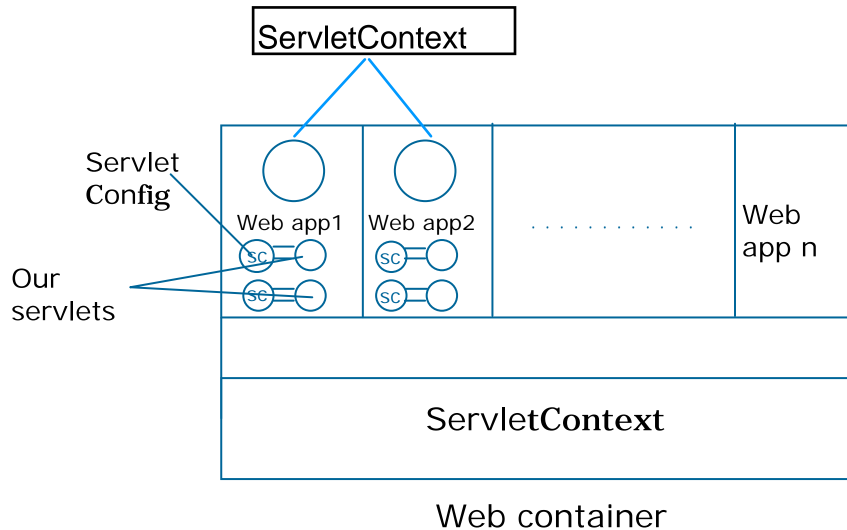
Web container must provide one *ServletConfig* object to a servlet. *ServletContext* is also called as application object.

javax.servlet.Servlet has the following methods

- *init* – called by the web container after our servlet object is created.
- *service* – called when the client sends the request.

- *destroy* – called before our servlet object is removed.

A servlet life cycle has 3 stages, (a) servlet is created (b) servlet will provide the service (c) servlet will be destroyed.



Web container

getServletInfo() must return a string describing the purpose of our servlet. *getServletConfig()* method must return the servlet config associated with the servlet.

It is the responsibility of servlet containers to create the servlet objects and destroy.

<load-on-startup>10</load-on-startup> where 10 is the priority.
If we use it, the web container will be created our servlet object when our application is deployed.

A servlet object is created

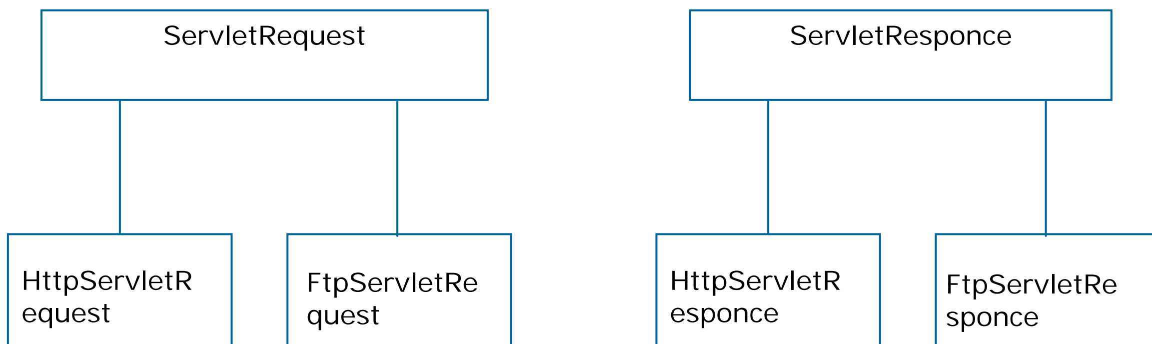
- At the startup if we use load-on-startup
- When the initial request is send and the servlet is not yet available or
- Whenever it is required.

Servlet object will be destroyed

- When the web application is undeployed
- Whenever the server wants to remove.

The *GenericServlet* implemented as part of *javax.servlet* package provides the implementation of *init* method and *getServletConfig* method that fits almost all the servlets.
javax.servlet.http.HttpServlet provides the implementation of *service* method. We can implement a servlet as a sub class of *GenericServlet*.

service method of *GenericServlet* takes *ServletRequest* and *ServletResponce* as parameters.



HttpServletRequest and *HttpServletResponce* contains methods specific to http and same as with ftp.
If we create a servlet as subclass of *GenericServlet* we may need to write common code like converting the reference of type *ServletRequest*, *ServletResponce* to *HttpServletRequest* and *HttpServletResponce*.
This can be eliminated by creating our servlet as a subclass of *javax.servlet.http.HttpServlet*.

Inside the servlet containers our servlets are pointing by using reference of type *javax.servlet.Servlet*.

Whenever a method is called on an object the JVM will start searching for the method from the class based on which the object is created. If it is not available then the method will be searched in the super class.

The containers always calls the methods exposed by Servlet interface.

```
init(ServletConfig)    {
        init()   }
service(ServletRequest, ServletResponce) {
        service(HttpServletRequest,HttpServletResponce)      }
```

service(HttpServletRequest,HttpServletResponce) {
doXXX() }

The implementation of *doXXX* methods of *HttpServlet* sends an error to the user agent.

To report an error from our servlets we can use
response.SendError("status code")

Before we start development of any solution we need to understand the business requirements and functional specifications.

We can use *request.getParameter* to get values sumbitted using a form.

Instead of using *System.out* for debugging its better to use log method. The data will be stored in a destination like in a log file or as database.

To know the location of log file we need to read the documentation of the container.

Typical sequence of steps we carry out in a servlet to process the form

- Read the input parameters
- Validate the input
- Process the input
- Generate the view.

Instead of hard coding we can place the information as servlet initialization parameters or context parameters.
driverManager.registerDriver(driver);
con=DriverManager.getConnection(url,dbuser,dbpass);

In above ex, instead of hard code we have used variables. Values of these variables can be picked up from web.xml file.

ServletConfig provides the methods *getInitparameter()* and *getInitParameterNames()*. As this is implementation of *GenericServlet* we can call these methods in our servlet.

String driver = getInitParameter("driver");
String url = getInitParameter("dburl");

Above code read the initialization parameters.

In *web.xml* we can add the names of initialization parameters,

```
<init-param>
<param-name>driver</param-name>
<param-value>oracle.jdbc.driver.OracleDriver</param-value>
</init-param>
<init-param>
<param-name>dburl</param-name>
<param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-value>
</init-param>
```

If multiple servlets uses the same set of parameters instead of placing them as servlet initialization parameters we can use context parameters.

```
<context-param>
    <param-name>name1</param-name>
    <param-value>value1</param-value>
  </context-param>
```

getInitparameter() available in *ServletContext* will be getting the initialization parameters configured as context parameters.

```
ServletContext scon = getServletContext();
url = scon.getInitParameter("dburl")
```

1st line gets the reference of servlet context
2nd line gets context parameter.

We can add any no of context parameters in web.xml file.

The web container is responsible for reading context parameters from *web.xml* and places them in *ServletContext*. For every servlet the container is responsible for creating *ServletConfig* and stores the init parameters in this object.

```
public class init extends HttpServlet {
public void doGet(...) {
        HttpServletRequest req;
        HttpServletResponce res;
        out.getInitParameters("name");      }
```

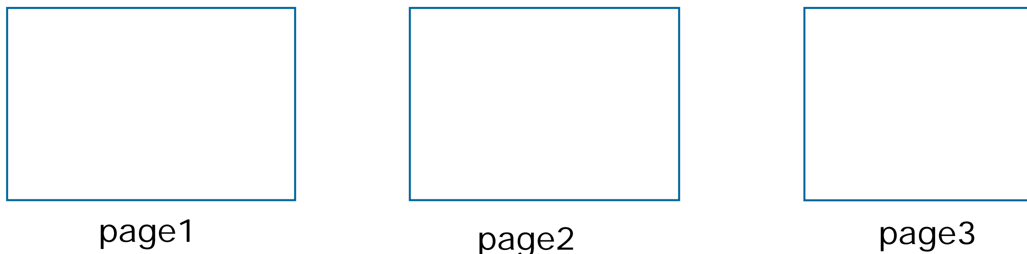
public void init(ServletConfig config) throws ServletException{
System.out.println(ServletConfig.getInitParameter("name");
}

Above code in *doGet()* will be failing to get the initial parameters. To avoid this failure we need to add *super.init(config)* if init method is overridden. Methods like *getInitParameter, getServletConfig, getServletContext*, log available in *GenericServlet* will fail if we write code with calling *super.init()*.

Instead of overriding *init(ServletConfig)* we can override *init()*

We can store the strings as resource bundles instead of hard coding the strings in source code.

It's very common to submit the information using multiple forms ex: user registration form can be split-up into multiple forms.



In above ex information is gathered from the user using 3 different forms. Once the user fills the 1st page he gets the 2nd page after it is filled he gets 3rd page. Once the 3rd page filled, information will be saved in database. If the protocol is designed to remember what is done earlier by the client then it is called as stateful protocol otherwise it is called as stateless protocol.

** Http 1.0 as well as Http 1.1 is stateless protocols.

If we have application with multiple forms the web server will be able to give the information submitted in the current form, it will not be able to give us the info submitted earlier.

A web application can use following techniques to remember the state of the client.

- Hidden Fields
- Cookies
- Sessions using cookies
- Sessions using URL rewriting.

We can use *<input type= hidden name="n1" value="v1">* as part of our html forms. Browser will not be developing the info about the hidden fields. If there are too many no of forms more amount of N/W traffic will be generated if we use hidden fields.

Cookie is a piece of info set by the server on the client using http.

We can use *response.setCookie()* to set a cookie on a browser, to get the cookies we can use *request.getCookie()*

Steps to set cookie:

- Create a cookie object *cookie c1 = new cookie(name, value)*
- Add the cookie *response.addCookie(name)*

Cookies are mainly used to serve personalized content according to user requirement.
Problems:

- If used heavily this generate more N/W traffic
- There are some limitations in some browsers in maximum no of cookies per domain.

It is not advisable to store sensitive info using cookie.
When we execute *response.addCookie(),* it adds set-cookie header to the response.

Browser sends back the cookie as part of request header as, cookies name1=value1&name2=value2

Most of the browsers provide an option of allowing or denying the cookies. Most of web application developers display a message saying that their web application works properly if the cookies are allowed.

The cookie class provides the methods *setMaxAge*, *setDomain*, *setPath* … In most of he cases we may not use these methods.

We can store any java object using *session.setAttribute(key, java object)* , *request.setAttribute, ServletContext.setAttribute(key, jo)…*

We can retrieve the objects by using *session.getAttribute(key) request.getAttribute(key) ServletContext.getAttribute(key)*

To remove the java object we can use *xxx.removeAttribute(key)*

The web containers can create the session objects on behalf of a client. For accessing the session object we use,
request.getSession(true/false).

For every session object that is created a session id will be generated. *request.getSession(true)* creates a session object if it is not already exists.

We can remove the session objects by using *session.invalidate().* As the server can't detect the closure of browser, the server deletes the session object after inactivatetimeout which can be configured in web.xml or we can use
session.setMaxInactiveInterval(int)
session.setAttribute("key1","value1");
session.setAttribute("key1","value2"); session.getAttribute("key1");
session.getAttribute("key2");

When 1st stmt executed value1 will be stored using key1 and when 2nd stmt executed value1 is removed and value2 is stored in key1. 3rd stmt gets value2 and 4th line gets null.

A web application based on session using cookie to carry session id may failed (if user doesn't accept cookies). To overcome this we can use sessions maintain using URL rewriting technique.

To use URL rewriting in our servlets, we need to use *response.encodeURL(string)* instead of directly writing the URL.

When we use *response.encodeURL,* the URL will rewritten as, /xyz/abc + session id
This technique places a bit of burden on server (server has to spend some time in writing URLs with session id).

To upload the files we have to use *post* method. In the form we can use any no of inputs. For files the input type must be specified as file.
There are so many file uploading components available in the market one such component is javazoom file upload component. You can see details at http://www.javazoom.net/jzservlets/uploadbean/uploadbean.html

To deals with upload files

- Create a newreq object using the class javazoom
- To get the list of the files we can use *getFiles()* – uses a list of files in a hash table.

To store the files:

- Create uploadBean
- Set the folder
- Set override policy
- Execute *store* method with newreq as parameter.

To read the parameter submitted as part of the form we can use *newreq.getParameter()* .Ex: book store provided by java soft

We can consider deploying a web application, un-deploying a web application, creating a session, adding an attribute, removing an attribute from ServletContext, HttpSession as events. We can write the code by implementing the listeners interface provided as part of javax.servlet.http package.

Names of the listener interface,
ServletContextListener, HttpSessionListener,
ServletContextAttributeListener and HttpSessionAttributeListener

ContextInitialized is called when the application is deployed.
ContextDestroyed is called when the application is undeployed.

We can implement the above listeners and configure them as listeners in *web.xml* file as,
<listener><listener-class> Class Name</listener-class> </listener>

SessionListener interface has (a) *sessionCreated* – called when the session is created (b) *sessionDestroyed* – called when session is destroyed.

Whenever we need to add an object as soon as the session object is created. We can write the code in *sessionCreated* of *HttpSessionListener*.

Along with the cookie we can pass a comment indicating the purpose of the cookei using *cookie.setComment()*

Netscape 4.75 (old versions) stores the cookies in cookies.txt file and IE uses multiple files.

When we specify the age of the cookie the browser stores the info about the cookie in a file,

1) Name of the cookie
2) Value of the cookie
3) Path of cookie
4) Domain of the cookie and
5) Expire time of the cookie.

In most of the applications the parameters will not setting the path explicitly using *setPath()* In such case path of the cookie will be set to the path of resource.

It is very rare to use *setDomain()* on a cookie. By default domain value will be set to the domain name where the cookie is set ex: http://localhost or www.yahoo.com

A servlet running at www.domain1.com can't set a cookie specifying the domain name of the cookie as www.domain2.com.

If the cookies are set by hotmail.com the browser will send back to hotmail.com only. This is why the browser stores the domain of a cookie.

While sending back the cookies, browser will check whether the path of requested resource is matching with the path of the cookie or not.

If we are running 2 web applications in same server one servlet in application1 can set a cookie that can be reuse by servlet in application2.

We can remove the cookie that is set earlier by setting its *MaxAge* to 0; if we have not used *setMaxAge -1* will be used as default.

If *MaxAge is -1* the browser will not storing the cookie in file, cookie will held in memory and its get deleted when the browser is closed. We need to pass the no of seconds the cookie has to live. Ex: *setMaxAge(600)* – cookie will live 10 min.

We can assign multiple names like a.com, b.com, www.a.com … to the same machine. On the global internet the N/W administrators will be configuring the DNS servers with this mapping.

We can place the mapping between the name and IP address in
C:/winnt/system32/drivers/etc (in 2000/NT)
C:/windows/system32/drivers/etc (in win Xp).
Ex: 102.54.94.97 xyz.com
abc.com

Http 1.1 supports virtual hosting – hosting the web site with different domain names on a single sever. This is supported by using the *Host* header.

According to Http 1.1 *Host* header is mandatory.

Using this header the web server will be able to identify the resource that has to be served.

It is very common to refer images, applets, activex controls in a webpage.
In older Http protocol the browsers used to establish a connection, sends the request, get the response, and disconnect the connection. If we have a html file with 10 images embedded inside it. The browser has to repeat the same steps for 11 times. This may reduce the performance of browser as establishing a connection is time consuming process.

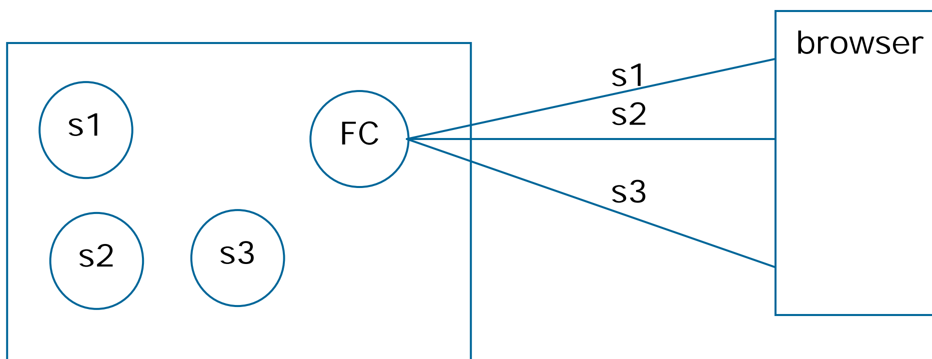
Browsers can send connection keep alive header asking the web browser to keep the connection alive. The server may or may not keep the connection alive.

Most of the web administrators either disable keep alive request or disable persistent connection or set the connection timeout to a small value.

If we expect too many concurrent users then to improve the performance of web server. We use small value for no of connections that are kept alive or we can total disable persistent connection.

**Irrespective of the type of connection http is stateless.

Front Controller:



The above web application uses Front Controller design pattern

In this design pattern we use one servlet to act as a controller sitting in front of other resources. In this design pattern all the requests will be first handled by FC then it dispatches the request to the appropriate resource.

The servlet engine internally uses *RequestDispatcher* to dispatch the request to the appropriate servlet or jsp.

We can use *ServletContext.getRequestDispatcher()* to get the request dispatcher.

There are two methods (a) forward (b) include

It's very common to develop a web application with more servlets processing the same request.

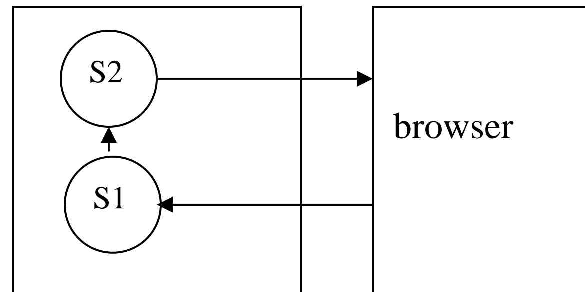
** We can use both include and forward to dispatch the request to other servlets. When we use include the outputs generated by both the servlets will be sent to the client when forward is used the output generated by 1st servlet will be discarded and only 2nd servlet output will be send to the client.

Difference between Request Forward and Redirect :

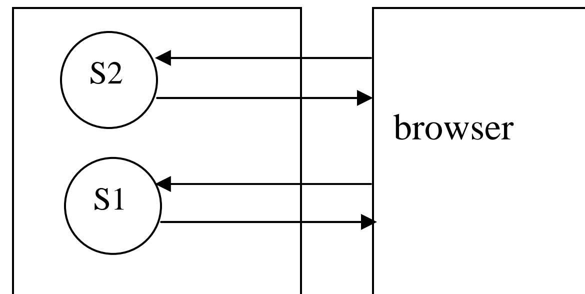
When we forward method only one request will be send by the browser and it is processed by two servlets/jsps. But when we SendRedirect method two requests will be send by the browser.

Ex:

class s2 { service() { ……………….. }
}
class s1 {
service() {
rd= sc.getRequestDispatcher("/two");
…………
rd.forward("req,resp); }



In this case browser sends the request to the resource s1, this will forward to s2 finally the browser will getting the response.



class s2 {
service() { ……… }
}
class s1 {
service() {
resp.sendRedirect(http://…./two); …..
}

In this case two different requests will send by the browser and it will get two responses.

Filters:

We can setup the files to perform the operations like logging, gathering performance statistics, user authentication, authorization of user to carryout

some operation, compressing the output generated by servlet. We can also configure chain of filters.

A filter is a class that provides the implementation of *javax.servlet.Filter.*

Life cycle methods of a filter are defined in this interface.

- When filter object is created by web container it calls *init()*
- Whenever there is a request that ha to be processed by the filter the container will call the method *doFilter()*
- When web container decides to remove the filter object then it will call *destroy()*

Filter chain object holds the details of objects (filter & resources like servlet/jsp/…)

We need to configure the web application with the info about the filters in *web.xml* as,

<filter>
<filter-name> name </filter-name><filter-class> class name <filter-class>
</filter>
<filter-mapping>
<filter-name> name </filter-name>
<url-pattern> / </url-pattern>*
</filter-mapping>
Most of the web containers creates the filter object and calls *init()* when the web application is deployed. When we undeploy the web application *destroy()* will be called and the filter object will be deleted.

See the URLs for more info on filters,
http://www.onjava.com/pub/a/onjava/2003/11/19/filters.html
http://www.informit.com/guides/content.asp?g=java&seqNum=95

In most of the applications which generates content dynamically we need to set the expiry time. If the expiry time is not set, the browser assumes that the content will never expire.

To set the expiry time we can use the,
Date d= new Date();

long l = d+6000;
response.setDateHeader("Expires",l);

When we use *response.setDateHeader("Expires",l)* a response header with name Expires will be added. It looks like,
Expires: Tue, 15 Nov 1994 08:22:31 GMT
Date: Tue, 15 Nov 1994 08:12:31 GMT

Every database driver sets a value for *Fetchsize*. In our programs we can use *getFetchSize* and *setFetchSize.*

You can see more details at
http://javaalmanac.com/egs/java.sql/GetFetchSize.html

JSP:

Most of the web developers deploying web applications using servlets mixes the presentation logic and business logic.
Separation of business logic from presentation logic helps us in simplifying the development of application(web authors can concentrates on developing the presentation logic and other developers in the team who has good knowledge of java can concentrate on business logic in (1)Java Beans (2)EJB and (3)Tag Handlers)

Java Server Page is designed to simplify the process of developing a web application on the top of servlet technology. Writing and configuring the application using JSP is easy

Javax.servlet.jsp and javax.servlet.jsp.tagext contains the interfaces to support JSP technology and Tag libraries.

When we send a request to JSP file if there is no servlet representing the JSP file the web container runs JSP compiler that generates the servlet code. Whenever we modify the JSP file the servlet will be regenerated. Some of the web containers provide an option of pre-compiling of JSP.

We can place our JSP files directly under WEB_APP_ROOT or any other sub directory except WEB-INF.
We need not add an entry in web.xml file for most of JSPs. When we need to pass initialization parameters to JSPs then we have to make an entry in web.xml as,

```
<servlet>
<servlet-name>servlet1</servlet-name>
<jsp-file>one.jsp</jsp-file>
</servlet>
<!-- mapping our servlet -->
<servlet-mapping>
<servlet-name>servlet1</servlet-name>
<url-pattern>/jsp1</url-pattern>
</servlet-mapping>
```

If there are any errors in JSP file the jsp compiler fails to compile the java source code. We can check the log files for more information on session for failure.
ASP allows us to mix html with a scripting language like jscript, vb script…
JSP also allows us to mix html with java language (scriptlet) or scripting languages.

Most of the web containers support only java language code be mixed with html. Resin supports jscript as part of JSP.
In JSP files we can use Page directives like Language, Import, Include …
These directives very much similar to preprocessor directives we use in C programs.

It is very common to have the same header and footer for multiple web pages, to simplify the development of such applications we develop header.jsp – responsible for generating only the header portion and footer.jsp to generate the footer. In all other JSP pages we write the code as,

Page1.jsp :	Page2.jsp:
Include header.jsp	Include header.jsp
Content1	Content1
Include footer.jsp	Include footer.jsp

Ex: www.oracle.com (same header and footer for all pages)

When we send the request for page1.jsp the JSP compiler internally generates Page1.jsp by merging header.jsp, footer.jsp with Page1 and this JSP page will be converted as a servlet.
According to JSP specification the container need not regenerates the servlet whenever there is a change in the files that are included using include directive. Tomcat regenerates the servlet even if there is a change in the included file.
When we used Info directive JSP compiler generates getServletInfo method.
Language directive can be used to specify the language that is used as part of our JSP. Ex:
Page Directive:

```
<%@ page
  [ language="java" ]
  [ extends="package.class" ]
  [ import= "{ package.class | package.* }, ..." ]
```

```
[ session="true|false" ]
[ buffer="none|8kb|sizekb" ]
[ autoFlush="true|false" ]
[ isThreadSafe="true|false" ]
[ info="text" ]
[ errorPage="relativeURL" ]
[ contentType="mimeType [
  ;charset=characterSet ]" |
  "text/html ; charset=ISO-8859-1" ]
[ isErrorPage="true|false" ]
%>
```

For more information on JSP page directives see the URL
http://www.developer.com/java/other/article.php/626391

Whenever we use an object in System.out.println internally toString() will be executed.

```
Ex:  Date d = new Date();
     System.out.println(d);
     System.out.println(d.toString());
```

If we need to use a class in a package other than the default package we can use import package or fully qualified name of the class in our code

In our JSP pages to import the classes we can use

- *<%@ page import = "java.util.Date" %>*
  <%@ page import = "java.io. %>*

  (or)

- *<%@ page import = "java.util.Date, java.io.*" %>*

We can include the java code directly as part of JSP files ,

<% Java Code %> called as scriptlet. When the JSP compiler compiles the JSP file it copies the scriptlet as it is in jspService method.

```
Html content
<%= new java.util.Date() %>
<%  int i=10; out.println("i=",+i);  %>
Html content
```

In above code we have declared a variable 'i' and used in print stmt but we are using a variable 'out' without declaring it but it is perfectly valid in JSP scriptlets.

Implicit Variables:

JSP compiler generates JSP code with a set of variable like
(i) out ii) request iii) response iv) pageContext v) session vi) application vii) config and viii) page. All the variables can be used in the scriptlets without declaring them. These variables are called as well known variable or implicit variables.

For more info on implicit variable see the following link
http://javaboutique.internet.com/tutorials/JSP/part11/page07.html

page refers to this – current object and pageContext is an object (*javax.servlet.jsp.pageContext*) that is created for every JSP page and it provides access to the other objects like out, session ….

We have methods like forward and include performing operations like
RequestDispatcher.forward and RequestDispatcher.include

In JSP pages we can use declarative statements to define the variables or implement the methods as,

```
<%! int i; %>
```

We can write our own methods, static variables and instance variables in declarative statement as,

```
<%! Static int x =10;
int add (int a, int b) { return a+b; }
%>
```

It is highly recommend to use tag libraries instead of using scriptlets and declarative statement.

In java we use try catch blocks to separate out the actual code that performs the action and code the code that deals with error. In JSP we can provide the implementation of code that handles the error in a separate page. This JSP page is known as error page.
For supporting the above concept JSP provides isErrorPage and errorPage directives.

```
<%@ isErrorPage="True" %>
```

errpage.jsp

```
<%@ isErrorPage="True" %>
<%@ errorPage="errpage.jsp"%>
```

one.jsp

When an exception is thrown while executing one.jsp file the container will start executing errpage.jsp. We can access the exception variable without declaring it in error pages.

Instead of sending the data byte by byte to the browser the web container collects the data in a buffer. If the size of the buffer is enough to accommodate the whole o/p of JSP page the o/p is collected and send at once to the browser. This reduces the amount of Network traffic.

If the jsp generates more amount of data than the size of the buffer the container frees the buffer once it is filled (this is done if autoFlush is set to true)

<%@ page buffer="1kb"%>
<%@ page autoFlush="false"%> (default is true)

When we encounter JSP buffer overflow exception we can solve it either by increasing the size of the buffer or by setting the value of autoFlush to true.

In JSP pages we can use action tags like *include, forward, useBean, setProperty, getProperty, param…* these tags are supported by all the web containers.

See url http://javaboutique.internet.com/tutorials/JSP/part11/page08.html

<jsp:include page="page2.html" flush="true" /> It includes the content of page2 (it may be of type jsp/html/img ….) in the o/p sent to the client (internally it uses *requestDispatcher.include*) similarly we can use *jsp:forward* tag to forward the request from one page to other page. This is equivalent to *requestDispatcher.forward*.

If the data is already committed before the execution of *jsp:forward* then forward will be failed. We can solve the problem by increasing the buffer.

There will be no difference in the o/p produced by using *jsp:include* and include directive. When we use include directive the contents will merge together by jsp compiler and generates a single servlet but when we use action tag include two servlets will be generated and in first servlet code similar to *requestDispatcher.include* will be added.

There may be slight benefit in terms of performance using include directive.

When we use *jsp:include* a change in main file and change in sub file causes regeneration of the servlets.

According to jsp specification the container need not regenerates the servlet if there is a change in included file when we use include directive

We can write the functions as part of declarations and in these function we can't access implicit variables. Implicit variables are local to *jspService* and can be accessed from the scriptlets.

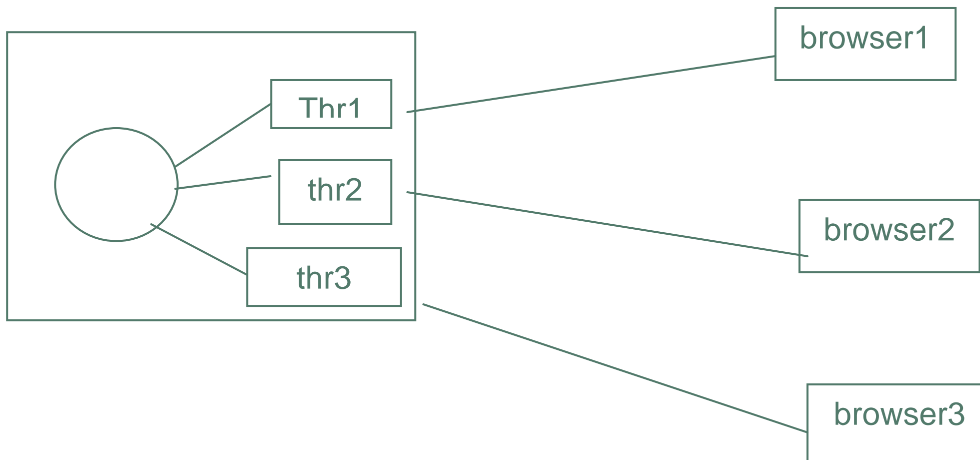
We can provide the implementation of *jspInit* and *jspDestory* which are equivalent to *init()* and destroy() methods in the servlets.

As part of JSP files we can use jsp expression as,
<%= Java Expression %>
ex: Current time: <%= new java.util.Date() %>

In jsp 2.0 we have support for **EL – Expression Language**

Tomcat as well as other web containers internally creates the threads and access the service method from the threads.

A servlet container creates multiple threads and accesses the service method from these threads concurrently as shown below same point of time it sends the response.



Since the servlets are accessing from multiple threads the code in servlets must be **ThreadSafe**. Code will be thread safe if it can be executed concurrently from multiple threads.
All the objects we will use from our servlets using instance variables must be thread safe. Not every java class is thread safe ex: AWT, JFC classes

If our servlet uses only locals variables we can claim that our servlet is thread safe.

We can develop a servlet by implementing single thread model. This is a marker interface or tagged interface. From servlets 2.4 this interface is deprecated and it is not recommended. By implementing this interface we are telling the web container that my service method can't executed concurrently from multiple threads.

Most of the web container vendors creates multiple servlet objects based on same class if the servlet implements single thread model. In this case more amount of memory space is required.

Java Beans:

We can assemble a computer or fan very easily by choosing different components manufactured by different vendors. We can take a screw from company one and use it to fit the Mother board to cabinet as they are manufactured according to a standard. Observing to this point to simplify the process of developing software, different software companies have proposed different component technologies. Ex: java soft java bean component tech, EJB component tech, Microsoft COM, BONOBO component model.

Java Bean and EJB are two different specifications from java soft.

EJB can be used to implement business logic on the server side. Most of the developers uses to assume Java Bean components are for developing GUI components and they can be used only on the client side but we can develop any kind of software using Java Bean standard (GUI/ non GUI). Java Bean can be used either on the client side or on the server side.

AWT, JFC components are implemented according to Java Bean standard.

According to Java Bean standard a Bean component can support a set of properties, set of events and any number of additional methods.

A property can be read-write or it can be just read only property. For read write property we need to provide setXXX and getXXX methods (isXXX if the property is Boolean)

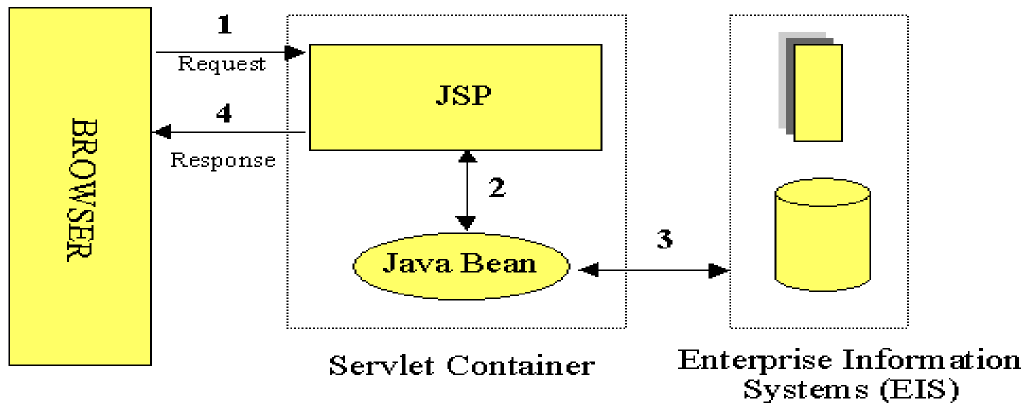
To support the following properties (i) uname (ii) email (iii) age according to Java bean standard we need to write the code as,

```
public class UserBean
{String uname;
 String email;
 int age;
 public void setUsername( String value ) {uname = value; }
 public void setEmail( String value ) { email = value; }
 public void setAge( int value ) { age = value; }
 public String getUsername() { return uname; }
 public String getEmail() { return email; }
 public int getAge() { return age; }
}
```

Java Beans like JButton supports the events by providing the methods with naming patterns (*i) addXXXListener (ii) removeXXXListener*

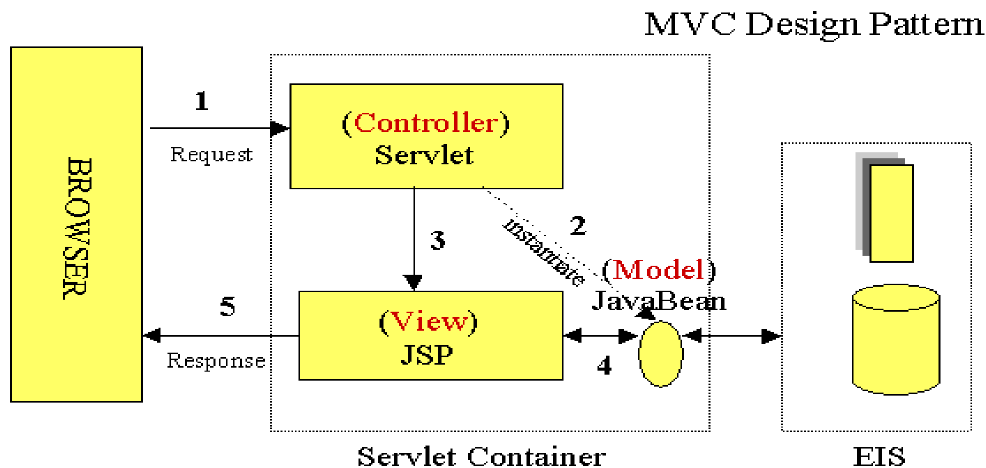
Apart from developing Java bean class we can also provide BeanInfo class. In this class we can provide *(i) Information about properties (ii) Information about the events and (iii) Information about the icon that represents our bean.*

According to JSP model1 we can develop the application as,



According to above model the presentation logic has to be implemented in JSP page and the business logic has to be implemented as part of Java bean This model help us in separating the presentation and business logic.

For a large scale projects instead of using model1 it is better to use model2 (MVC). Stuts frame work is based on model 2.



JSP compiler generates the following code equivalent to
<jsp:useBean id =“var1” scope =“session” class =“class name” />
as,

```
ourpack.ourbean var1 = null;
var1 = session.getAttribute(“var1”);
if(var1 == null) {
var1 = (ourpack.ourbean)class.forName(“ourpack.ourbean”).new
instance();
Session.setAttribute(“var1”, var1);    }
```

JSP compiler generates the following code equivalent to
<jsp:useBean id =“var1” scope =“application” class =“class name” />
as,

```
ourpack.ourbean var1 = null;
var1 = application.getAttribute(“var1”);
if(var1 == null) {
var1 = (ourpack.ourbean)class.forName(“ourpack.ourbean”).new
instance();
application.setAttribute(“var1”, var1);    }
```

In useBean tag we can specify the following scopes,

- Session -- Bean object will be stored in session object. We will be using this scope if we have to store data specific to the client.
- Application – Bean object will be stored in application object. We will use this scope when global data has to be stored.
- Page – Used to store the bean in pageContext object. Uses when we need to hold the data only while processing the request. The bean object is accessible only in that page.
- Request – Bean object will be stored in request object. Bean object can be accessible by multiple pages that process the request. This can be used if we need to hold the data only during the request processing using multiple JSP/ servlets.

We can use the tags *jsp: setProperty and jsp:getProperty* to set and get the properties of a bean.

To access additional methods available in java bean we can write var1.somemethod in a scriptlet.

We can automate the process of setting bean properties with the values submitted by the user in a form.

Most of the application developers develops a JSP/ servlet for (i) Generating the form and (ii) Process the form.

If we need to process a form as follows,

Ename [ ] (ename)

Salary [ ] (sal)

Dept no [ ] (dept)

[Submit]

We can create a java bean with property names ename, sal, dept so that in our JSP we can use,
<jsp:setProperty name = "emp" property = "" />*

Let us assume that we are developing an application that has to deployed at different customer sides, for this application we can store the preferences of the customer in a xml file or in a database table as,

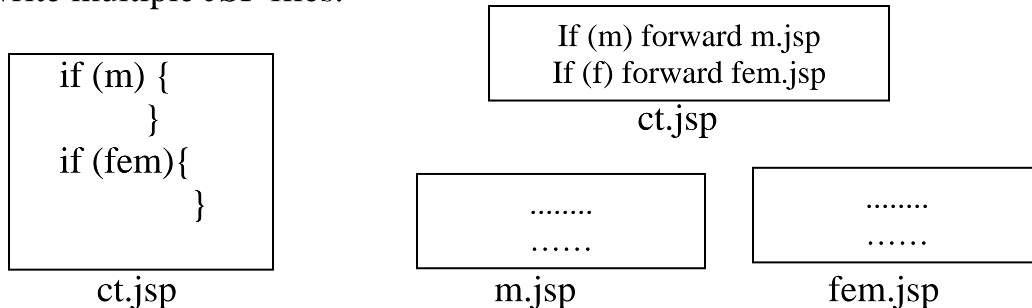
App_prof

Prof_name	prof_value
Logo	one.jpg
Name	comp1
bgcolor	white

In this project we can create a java bean that holds this information. In this above data is not specific to the client and it need to be accessed by multiple jsps we can create a java bean and store it inside Application obj. The data that is specific to the client has to be stored inside a session object if multiple jsp/servlets need to access the same data.

When we use page scope, object will be available only in that page. Multiple jsps processing same request need to access same data then we need to store the data in request scope.

Let us assume we need to compute the tags of a application and generates different views for male & female applicants. For this we can develop a single jsp like CT.jsp and write the whole code in a single file or we can write multiple JSP files.



In our earlier ex we establish the connection with DB in our servlets whenever a connection is required and closes the connection when the task is completed. This reduces the performance as it takes some amount of time in establishing the connection.

To improve performance of servlet/jsp/EJB we can use connection pools. A connection pool pre creates a set of connections to DB. In our code whenever we require a connection we will take it from the pool and when the task is completed instead of really closing the connection we will be putting back the connection in the connection pool.
If we need to use connection pool we can store it inside application scope and access the connection pool from all the JSPs. Today every web container provides an action of creates a connection pool and accessing it using javax.sql.DataSource.
To use connection pool with Tomcat server,

- Copy the JDBC drivers under Catalina_HOME/Commom/lib
- Open the browser and login into Admin page.
- Choose Data Sources and use create new Data Source by feeding the information like name of the jdbc driver class, DB Uname, pwd and jdbc URL by giving a name to this. In our programs we can use this name to access the connection pool.

We can write the following code to detect the jsp version currently supported by the container.

```
JspEngineInfo  ei = jspf.getEngineInfo();
out.println(ei.getSpecificationVersion() );
```

We can use the code to check the jsp version and display error message if our web application deployed on a engine that is not enough to support our web application.
Most of the shell scripting languages supports $ {varname} for getting value of a variable. Same kind of syntax is supported in EL that is part of JSP 2.0
EL supports basic arithmetic operations (+,-,*,/,%)
To display ${1} in jsp we need to write \${1}. Similar to any other programming language in an expression we can use a variable or a constant.
Java soft has initially used EL as part of JSTL (Java Standard Tag Libraries). As most of the developers started using this language Java soft started supporting it directly in jsp.

EL supports basic comparison operations like <,> …
EL provides support for implicit objects like pageContext, pagescope, sessionscope, applicationscope, param, paramvalue, cookie and initparam.

Tag Libraries:
With tag libraries…

```
< arth:add param1 = 5 param2 = 10 />
```

Without tag libraries…

```
<% int a =5, b =10, c = 0;
    c = a+b ; %>
```

Above two ex shows the jsp page performs the same operation but the first one is written without using java code. Any html developer can very easily learn and use tags like this. A jsp tag library provides the support for multiple tags or at least one tag.

By using tag libraries we can eliminate the scriptlets this help a non java programmer to develop a web application using jsp technology.
Tag library implementer provides a tld file and jar file containing java classes. These classes are called as tag handlers.
When a tag is encounter in a jsp file the web container internally executes the methods on tag handlers.
Java soft has identified a set of commonly used tags in almost all jsp applications and released a specification called JSTL.
Any one can provide the implementation on JSTL tags. One such implementation is available at http://www.jakarta.apache.org
Every tag library will be identified uniquely by URI.

The JSTL tags are divided into different categories

- Core Tags : out, if, forEach, forTokens, redirect, import, set. Remove
- SQL tags : used to perform DB operations – transaction, query, update …
- XML tags : used to deal with xml content – parse, other core tags like set, … that deals with xml.
- For developing internationalized applications we can use i18n tags – setLocale, timezone, bundle, setbundle, message, formatnumber, formatdate

Procedure for using a Tag Lib :

- Copy tag lib jar files under WEB-INF/lib (if tag handler supplied as classes without package a jar file then we need to copy the classes under WEB_INF/classes.
- Copy the tld files files under WEB-INF or create a directory under WEB-INF and copy tld files under this directory.
- We need to give the information about the tag libraries in web.xml by adding the following lines to it as,

```
<taglib>
    <taglib-uri>http://www.inet.com/taglibs/bodytags</taglib-uri>
    <taglib-location>/WEB-INF/tlib/tlib1.tld </taglib-location>
  </taglib>
```

To use tags in jsp file we need to add taglib directive as ,

```
<%@ taglib uri="http://www.inet.com/taglibs/samptags" prefix="abc"%>
```

Every tag can support zero or more number of attributes and a tag may or may not support body content.

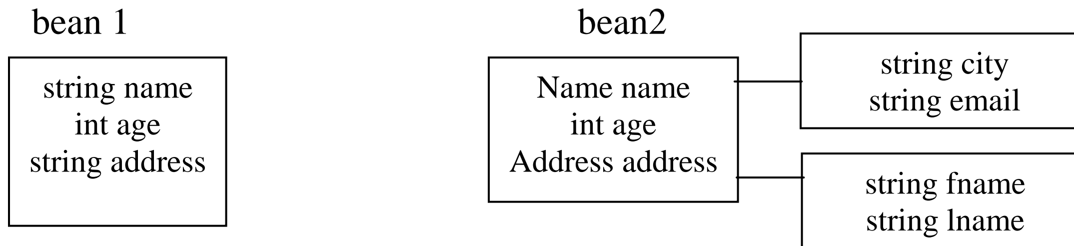
We can use setDataSource tag with driver, url, uname, pwd as attributes to the tag. We need to specify the name of the variable using var tag. This variable name has to be used in query, update tags.

We can execute sql queries using the query tag by specifying the datasource, select statement. The result set can be accessed using the variable name that is defined in var attribute.

Before running jdbc code from the servlets we need to copy jdbc driver classes under Catalina_Home/common/lib

We can run multiple sql stmts as part of a single transaction. For this we need to group update and query tags as part of transaction tag.

Using EL we can access the properties of a customer bema using
${cust.name}



In the second bean not all properties are of type java primitive or java strings. In this case we can access different properties using expression ${cust.Name.fname} …
As part of EL expressions we can use logical AND and logical OR.
In EL expression it is better to write xxxScope.beanname.property instead of using beanname.property.
paramvallues, headervalues are implicit objects. These objects are collections

In html forms we can use different input fields with same name. ex :
http://localhost:8080/1.jsp?p1=val1&p2=val2&p3=val3&p3=val4&p3=val5
In above URL even though we have five parameters there are three parameters with same name.
We can use forTokens to split-up a string into multiple tokens separated by a delimiter. If the delimiter is a ‘, ‘ we can use forTokens or forEach.
To store multiple objects we can use classes like vector, arraylist, hashmap, hashtable … Whenever we need to iterate though the list of objects using forEach tag we can store the objects inside list, linkedlist, array, stack, set, hashmap, hashtable, property…
Instead of writing the loops for copying elements from one array to another array we can use System.ArrayCopy

Internationalization

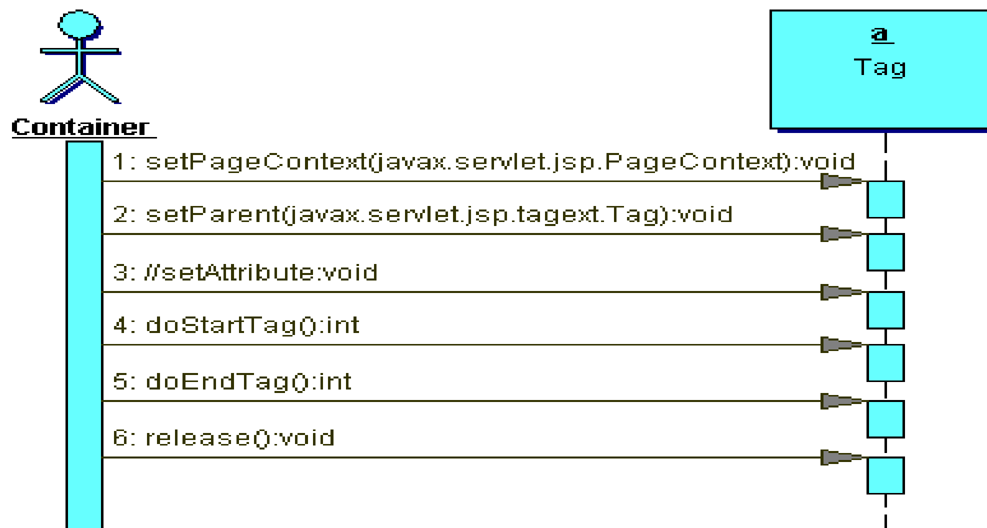
When we develop application targeting the customers of multiple locale we need to take care about (i) Data format (ii) Time zone (iii) Number format (iv) Messages we display (v) Currency symbol. These kind of applications are called as internationalized applications.
Once we develop i18n applications we can localize it to different locale.
When we install OS we choose the locale settings. We can modify these settings at later of time. Ex: regional settings in windows
To represent a locale in java we can use java.util.locale class.
If we develop the code targeting one locale as,
JButton jb1 = new JButton ("save");
Localizing into other locale takes lot of time. Instead of hard coding the strings that will be used as labels or messages in the code as shown above. We can separate the strings and store them in property files.

In the resource files the label of delete button is
button.delete.label = delete – app_en_US.property
button.delete.label = loschen – app_de_DE.property
If we create resource files in one locale in future we can add a new resource file to support a new locale. In this case we need not change the code.
In order to read resource files we can use java.util.ResourceBundle. In a single program we can use multiple resource bundles.
We can use ResourceBundle.getBundle(basename, locale) -- this loads the resources from the files specific to locale if it is not available the default resource bundle will be loaded.
errors.invalid = {0} is invalid
errors.maxlenght = {0} can't be greater than {1} characters
In above ex at the run time we need to supply the parameter values to the messageFormatter.

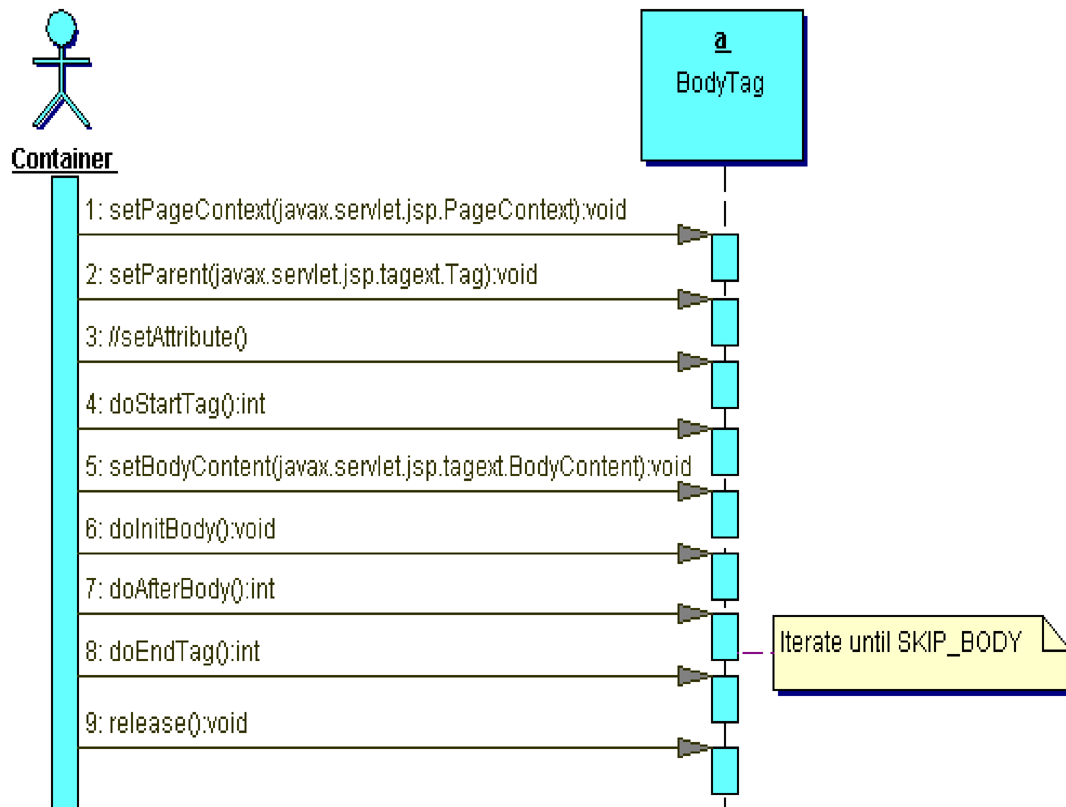
In internationalized applications we can use java.text.DateFormat, java.text.NumberFormat, java.text.MessageFormat to format date, time, numbers and parameterized messages.
We can support <pre:tag without body and attributes />
<pre:tag without body attr1="val1" />
<pre:tag without body and attributes>body content<pre:tag />
<pre:tag with body > body content <pre:tag />

To support tag library Java soft has provided the package javax.servlet.jsp.tagext. As part of this package Tag, iterationTag, bodytag interfaces are defined. As part of same package we have classes TagSupport and BodyTagSupport.
Tag handler is a class that provides the implementation of TagSupport to support a tag without body and BodyTagSupport to support a tag with body. The most important methods as part of Tag interface are doStartTag and doEndTag.

Similar to servlet specification in which java soft define the purpose of each and every method and when the container has to call these methods. In jsp tag specification java soft has defined the purpose of methods in Tag and BodyTag interfaces and it has defined the sequence in which the methods has to be called by the container.
We can use a tag inside another tag like otherwise, when inside choose tag. In this case choose tag is known as parent tag.
Javax.servlet.jsp.PageContext provides the methods like setAttribute, getAttribute, getOut, getSession … we can use PageContext object to access session, application … objects.
setParent will be added by the container to give the information about the parent.
Since the code for setPageContext, setParent, getParent is common for most of the tags these are implemented in TagSupport.
Variables like id, pageContext are provided by TagSupport class. id is used to hold an attribute and pageContext variable is used to hold PageConext object.



Most of the 1st generation web containers used to create a tag handler object when a tag is encounter and call the method as shown above. Some web containers have implemented techniques like pooling the tag handlers and reusing the tag hadlers to improve the performance.
When we write a tag handler for a tag like <tag1 /> in the doStartTag method after carrying out initialization we have to return SKIP_BODY. If we have to support <tag2> Body </tag2> then we need to return EVAL_BODY_INCLUDE.
doEndTag will be called when the container sees the end of tag. In this method if we encounter a problem it is advice to return SKIP_PAGE other wise EVAL_PAGE. When the SKIP_PAGE is returned the remaining page will be skipped if EVAL_PAGE is returned web container evaluates the remaining part of the page.
To support a tag that accepts body we need to provide the methods defined in BodyTag interface. Some of these tag handlers (forEach) has to support evaluation of body for multiple times. Instead of directly implementing BodyTag interface we can create a sub class of BodyTagSupport class.



doInitBody will be called only once but doAfterBody will be called multiple times till the method returns SKIP_BODY.
JSTL tag libraries provide a pair of tlds for every tag library. Ex: For core tag library we have c.tld and c-rt.tld. In c.tld for attributes we have, <rtexprvalue>false</rtexprvalue> and in c-rt.tld we have , <rtexprvalue>true</rtexprvalue> for all attributes.

<inettag:mathops op1 =“14” op2=“10” oper=“+” />
<inettag:mathops op1 =“14” op2=“10” />
While designing the tags depending upon the requirement developers can stress to support set of attributes ex: op1, op2 and specify whether an attribute is required or not in the tld file. Ex: op1, op2 are required and oper is not required.
In order to support an attribute x as part of tag handler we need to provide setX method. For above tags we need to provide setop1, setop2 and setoper methods.

As most of the tags support an attribute ID, setID is provided.
We need not provide the implementation of general purpose tags like for, switch, forward… as these are available in JSTL. We need to implement application specific tags.

STRUTS:

A frame work like struts provides a set of classes implemented using architecture. Frame work provides a proven procedure for implementing a project. *Struts* is based on *MVC* architecture. It contains a set of tag libraries like struts-bean, struts-html, struts-logic …
To simplify the development of struts based projects struts team has provided struts-blank.war. This can be used as starting point for the development of a web application using struts frame work.
Struts-bean tag library provides functionality similar to i18n tag library of *JSTL.*

In the project using struts frame work we need to copy struts related jar file in *WEB-INF/lib* and tld files of struts tag lib has to copy under *WEB-INF*.
To develop a struts based application we need to add struts tag library description to *web.xml* plus we need to add,

```
<servlet>
<servlet-name>action</servlet-name>
   <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
</servlet>
```

We need to provide servlet mapping as,

```
 <servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
  </servlet-mapping>
```

The above servlet handles all the requests that matches with *.do. Action servlet is called as struts controller. This servlet has to be loaded when the application is deployed. Action servlet reads the information provided in configuration file *struts-config.xml.*

When a struts based application is deployed web container creates the controller servlet and this servlet reads the information available in *struts-config.xml*. We can manually edit *struts-config.xml* or use a product struts console to create the file.

It is highly recommended to use the resource files instead of hard coding the messages in JSP file.

Html tag library is used to generate the html tags.

The bean tag library provides the tags like *message* – to get the message from the resource bundle, *cookie*, *include*, *resource*, *write* – to write the value of bean property, *logic tag library* provides the tags like *present*, *equal*, *greaterequal*, ….
Struts automatically the process of getting the form data, validating form data and processing the form data. It is always advisable to use *<html:form action="xyz.do" />* instead of using form tag (normal html tag)
In struts frame work to handle the form data we need to create a form bean.
Procedure:

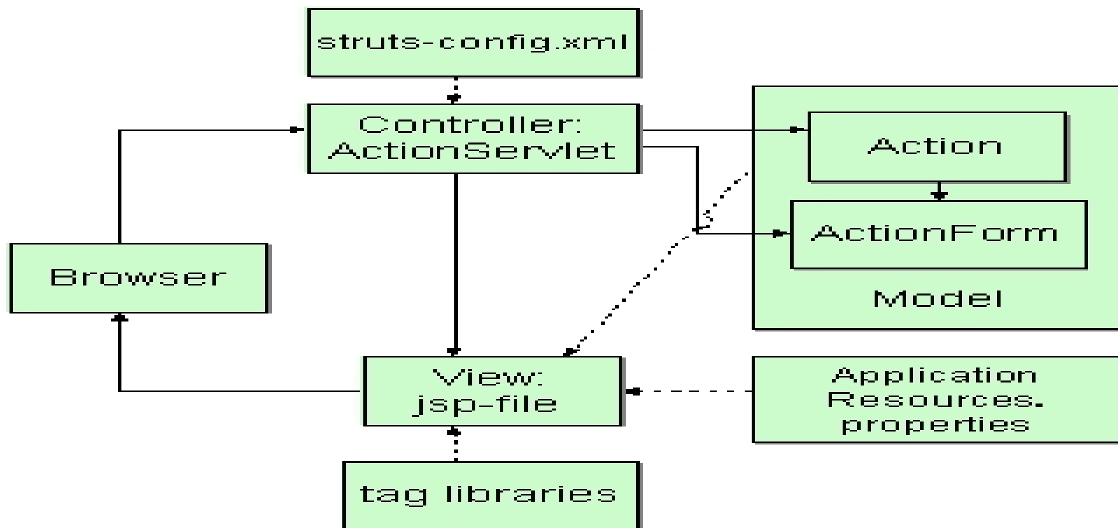
- Identify the names of the form fields ex: assume the form contains fields like empno, ename, desg, sal …
- Create a class xxxform as a subclass of *org.apache.struts.action.ActionForm* and provide the setters and getters corresponding to form fileds.
- Provide the *reset()* method in formbean class. In this method set the default values for the properties.
- If required provide the method *validate()*.

In validate method we need to provide the code that validates the input and returns a null if there are no errors. If there are any errors we need to return the errors.
As part of struts frame work we have *actionError* class that can be used to represent a single error. We can store multiple such errors inside an object of type *ActionErrors*.
To process an action request we need to provide an Action object
Procedure:

- Create an action class by extending *Action* class
- Provide the method *execute()*

In almost all web applications we need to carry out the following steps when user submits the data

- Get the data
- Validate the data if data is valid process it otherwise redisplay input form with the list of errors.



For struts UML diags visit the site
http://rollerjm.free.fr/pro/Struts11.html

When a request is send to perform an action like addemp.do struts framework creates *formBean* and sets the properties of *formBean* using the data submitted by user. It validates the data if the data is not valid it resends the form with errors. If there are no errors it executes the action. Depending upon the result of action struts framework picks up the view and sends the result to the browser.
See the link for more info http://www.exadel.com/products_tutorials.htm (struts framework tutorials)

The developers of web applications using struts framework need to provide the *formBean, Action* classes and provide the information about these classes in *struts-config* file.

When we start struts bases applications the framework loads the resource that is listed in *message-resources* element in *struts-config* file as
<message-resources parameter="ApplicationResources" null="false"/>
<message-resources parameter="inetRes" null="false" key="Inet-key" />

In struts 1.1 we can specify multiple resource files as shown above, 1st resource will be loaded and stored inside servlet context using the key *org.apache.struts.actionmessage* and 2nd resource will be loaded by struts framework and it will be stored in the servlet context using the key *Inet-key.*

We can access the messages using *bean* tag as
<bean:message bundle="Inet-key" key="index.title">

Above tag picks up the value of index.title from the bundle *Inet-key*

If we use *null="True"* in *message-resource* element the framework throws an exception saying that the resource is not available. If we set *null="false"* instead of throwing an exception it returns *?? locale key ??*

We can access the parameterized messages using *bean* tag. In struts framework there is a limitation on no of parameters we pass to a parameterized message   arg{0} … arg{4}

As part of struts some facilities are provided if we want to extend those capabilities of struts framework we can provide our own plugins. Ex: struts framework can't validate our inputs automatically. A plug-in called struts-validator is provided to take care of validation.

To use a validator plug-in we need to add the *plug-in* element as
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">

In order to handle the form we need not develop a formbean class on our own. In place of our own formBean we can use *dynamic action form* supplied as part of struts. In order to use the *dynamicformbean* we need to add entries in struts-config file as
<form-bean name="userinfo"
type="org.apache.struts.action.DynaActionForm">
 <form-property name="fname" type="java.lang.String"/>
 <form-property name="lname" type="java.lang.String"/>
</form-bean>

We can avoid writing the form beans by using *DynaAactionForm*

To access the properties of the bean we need to use formbean.get("name of property");

To use validator framework
- Add validator plug-in info to struts-config.xml
- Add errorxxx entries to application resources file (these resources will be used by struts framework)

- Design the form and place validation rules in *validation.xml* file

It is better not to modify validator-rules.xml file. This contains the information about the java scripts that are required for validations on client side and the classes that are used for validation.

To use validator framework we need to create a formbean by extending *ValidatorForm* and provides *reset*, *getxxx*, *setxxx* methods. If we need to provide additional validations that are not supported by the validator framework we have to implement *validate()* method.
We need to define the validation rules specific to our application in *validation.xml*

```
<formset>
    <form    name="vform">
      <field    property="age" depends="required,integer"> </field>
      <field    property="e-mail" depends="required,email"> </field>
</formset>
```

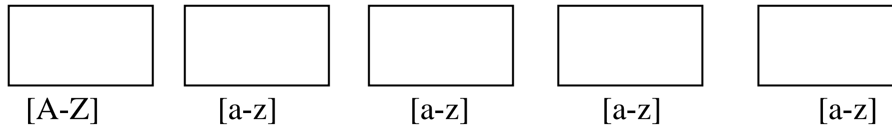
Developing a web application that validate a form on the browser is easy with struts.
Procedure for generating java script with validator framework

- In jsp file that generates the form we need to add *<html:javascript formName="vform">* script uses a variable bcancel if the value of the bcancel is true validator will not be carried out.

- In html:form tag add, *<html:form action="/tv.do" onsubmit="return validateVform(this);">*
- Before we call validatevform we need to set bcancel to false. This can be done by adding, *<html:submit property="submit" onclick="bCancel=false;">*

The validator framework uses the java script functions that are part of *validator-rules.xml*
Let us assume there is a form field with the name UserName, from this field we want to accept the inputs like (i) Abc (ii) Xyz but not (iii) 12xy. For this in struts framework we can use the mask validator. For this validator we need to provide regular expression.

Let us assume we want to accept 5 chars from the user and 1st character must be capital and remaining 4 chars must be small letters. For this requirement we can use Regular Expression as shown below,



/w stands for a word character
Some times we may need to specify an option like A or X or Z for that similar to the range we can specify as, *[AXZ]*
We can fix the value for a char position by directly writing the char as, *[A]x[Z]*

We can define a constant in validation.xml as,

<constant>
<constant-name>zip</constant-name>
<constant-value>^\d{5}\d$</constant-value>*
</constant>

In RE ^ stands for start with, $ - end with

/W – non word char
/d – digit
/D – non digit
* -- *zero or more* + -- *one or more* *? – Zero or one*
we can use |(or) as [A|Z]

When ^ is used as *[^abc]* it accepts the char other than abc.
In some programs if we need to accept RE we can use *regexp* package available at jakarta site.

In a typical strut based application some of the URLs will be pointing to jsp files ex: form.jsp and some of the URLs pointing to strut actions ex: process.do. By using the Action class *org.apache.struts.actions.Forward* we can map a struts action to a jsp file.

For using this we need to add the following mapping.
<action input="/index.jsp" path="/abc" type=" org.apache.struts.ForwardAction" parameter="/two.jsp"> </action>

With above mapping struts framework will execute two.jp when a request is send for abc.do
In most of the cases to process a form and automatically validate it we did not use our own form beans. For this we can make use of *DynaValidatorForm*.
Most of the IDEs supports struts framework. These IDEs automatically generate the configuration files plus generate the skeleton code. We can also use struts console to edit *config file*.

TILES:

Before we actually develop the web application we need to decide about the layout of the pages. Most of the websites uses the layout. Once the layout is designed, the designers create the template according to the layout.

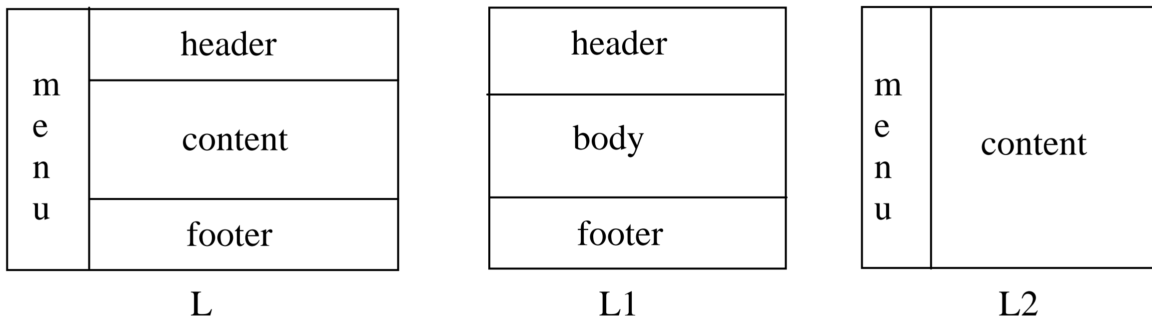
<table>
<tr><td rowspan="3">M
e
n
u</td><td>header</td></tr>
<tr><td>content</td></tr>
<tr><td>footer</td></tr>
</table>

In above ex the webpage split-up into multiple pieces we can treat these pieces as tiles. Tiles framework can be used to design the web pages with a consistent look.

As part of tiles plug-in we have support for a set of tags, as part of tiles tag lib we have the support *importattribute, useattribute, getasstrin, get, beanname, add, put, insert…*

To use struts tiles framework we need to define a layout or use the layout files that are provided as part of the tiles. Once the layout is defined we can use the same layout as part of multiple pages.

We can design the websites using the layout shown below, for this we will define a layout that splits the page into 3 parts.



As part of tiles we get a set of standard layouts these layouts are available in \layout.
Instead of defining the layouts in jsp files we can create a layout definition.
We store the definition of layout in xml file.

EJB (Enterprise Java Beans)

EJB is a component specification that can be used on the server side. Big advantages are

- During the development of projects we can make use of components developed by other companies. This reduces the cost of application, amount of time we spend in developing the application.
- As the EJB based on RMI over IIOP we get the benefits like
    a. Fail-safe applications
    b. Scalability

The servers (EJB containers) provide

- Object management services
- Management of Threads
- Transactional services
- Security services
- Access to directory service

In any business application apart from business logic we need to write the code for above services.

When we use EJB we develop the business logic and other things are taken care by the container.

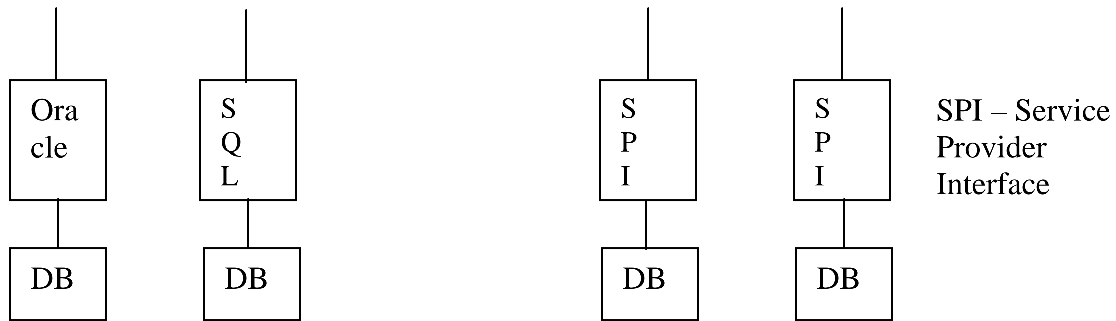
Today the products like Web logic, Web sphere, Pramti j2ee server, Jboss, Tomcat, Oracle application server, Sunone server … are called as J2EE servers. Not all the products support all APIs that are part of J2EE.
Ex; Tomcat acts as web container but not as EJB container. Web logic, Web sphere can acts as web containers as well as EJB containers. Jboss acts as EJB container but it bundled with Tomcat can be used to deploy both EJB and servlets.

We need not use EJB kind of technology to develop every business application.
For a small business houses which performs less number of transactions for developing the software, we need not use the technologies like EJB, JMS …

Similar to DB servers we have a number of Directory Services. We can use directory service to store info about anything. Similar to JDBC we can use JNDI to access the directory services.

JDBC code

Program using JNDI
API
Access Directory service



Almost all J2EE server vendors are providing a directory service as part of their products. Ex: Web logic has its own directory service.
In our J2EE projects we use directory service to store

- Info about connection pool
- JMS topics
- JMS queues
- User transaction (part of Java Transaction API)
- Info about mail server
- Any java object
- EJB

We can copy the web application as well as ejb under the application directory.

Procedure for configuring connection pool in web logic:

- Collect the following details
    a. JDBC driver class
    b. JDBC URL
    c. Other parameters like user name, password …
- Login to web logic console and choose connection pools. In the create connection pool wizard provide the name of the pool and above parameters.
- If require copy the JDBC driver in class path of web logic.
- Configure tx data source by providing JNDI name and connection pool. With this step the info about the connection pool will stored inside directory server.

In order to use the connection pool we need to configure data source or transactional data source.

To access the info available in directory server we need to get hold of the root of the tree (initial context). Getting the initial context object is similar to establishing a connection with DB server.

In order to get hold of initial context we need to create a hash table and fill it with properties like user name, password, name of the initial context factory class, URL giving the info about the server.

bind() method stores the info in the directory server. This method throws an exception if already an object is stored in directory server with the same name. To remove the info about an object we can use *unbind()*.

Once we obtain the initial context we can use *initialcontext.lookup* by giving the name.

As part of *javax.sql* JavaSoft has specified an interface with *javax.sql.DataSource.* We can use *DataSource.getConnection()* to get the connection from the pool.

Steps to connect from the pool:

- Obtain initial context
- Obtain the data source i.e. lookup
- Execute *getConnection()* method of the data source

When we execute *Connection.close()* the connection to DB server will not be closed, connection will be marked as free.

If we are performing a transaction on a single DB we can commit the transaction by using *Connection.commit()* and *Connection.rollback()* methods but we perform transactions on multiple resources then we have to use java JTA (Java Transaction API) which uses internally java transaction service. As part of JTA we have an interface *UserTransaction.* We can use *UserTransaction* object with the name *javax.transaction.UserTransaction.*

Even if we need to perform a transaction on a single resource its better to use JTA to control the transaction.

In any business application we need to maintain the info about different entities ex: students, employee, bank, bank transaction, voter ….

In the project using EJB technology we can use entity beans.

In every project we develop the code according to the business processes. In EJB projects we will be using session beans to implement a business process

A web server is a product that provides the implementation of http protocol ex: IIS, Apache web server. Application server is producer that provides the facilities like object creation, object management, security, transactional services, connection pooling. These facilities are required for all business application ex: MTS, Bea Web logic server, IBM Web sphere…

Most of the application servers provide the implementation of http protocol.

While developing EJB we will be using the package *javax.ejb. javax.ejb.EnterpriseBean* is a tagged interface. This interface is extended by *SessionBean, EntityBean, MessageDrivenBean.*

Above three interfaces represents three different types of beans.

Similar to *ServletContext* in EJB we have *SessionContext, EnityContext, MessageDrivenContext* interfaces. All the interfaces extends from *EJBContext.*

A session bean provides the implementation of *SessionBean* interface.

Before we start the development of EJB we need to identify the methods that have to be exposed by EJB to its client.

To develop a session bean:

- Identify the methods that has to be exposed
- Define a remote interface by extending *javax.ejb.EJBObject*
- Compile the remote interface using *javac -d . voterRemote.java*
- Create a home interface with a create method, create it by extending *javax.ejb.EJBHome.*
- In the home interface provide a *create()* method. Ex: *public voterRemote create() throws RemoteException, Createexception*
- Compile the home interface using *javac -d . voterHome.java*
- Provide a bean class by implementing *SessionBean* interface.

(As part of *SessionBean* interface we have *ejbActivate, ejbPassivate, ejbRomove, ejbSessionContext*).
In this class we need to provide *ejbCreate()* method. Provide the implementation of all the methods defined in *SessionBean* interface. Provide the implementation of methods that are exposed by remote interface.

- Compile the bean class.

Once the code is compiled we need to package EJB in a jar file (we can also use ear file). Create a jar file by packaging classes.

Similar to packing a web application with *web.xml* in EJBs we need to create deployment descriptors (xml files). For creating these files we use a tool *startWLBuilder.* The tool creates the deployment descriptors and places them in jar file.
For more info on WebLogic Builder visit the site
http://e-docs.bea.com/wls/docs70/wlbuilder/weblogicbuilder.html

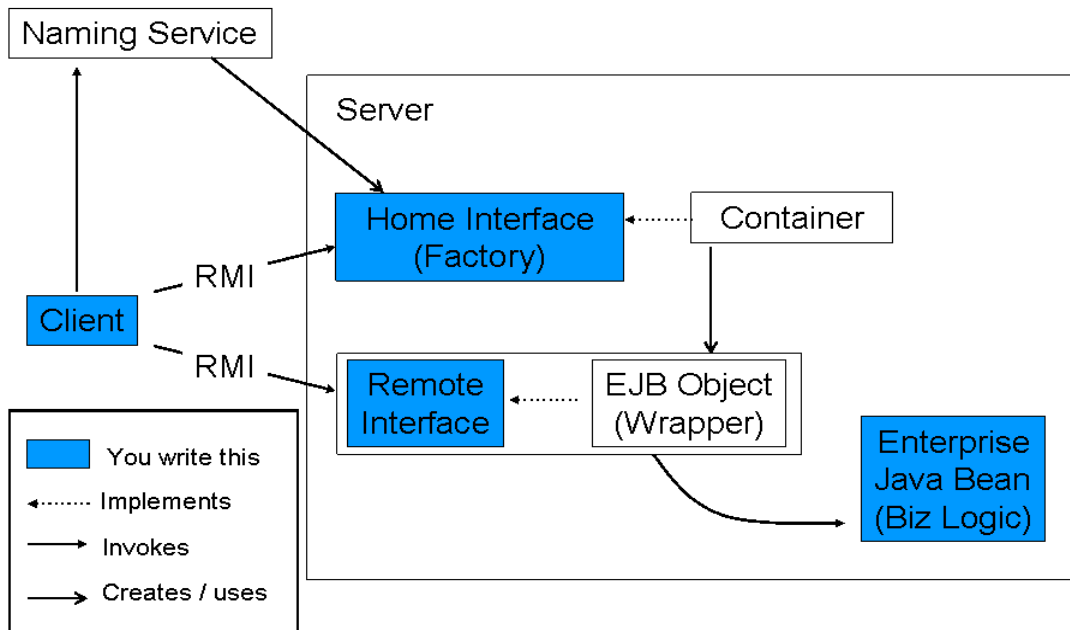
To deploy the jar file copy the jar file under applications directory or use web logic console.

We can manually create required class by running
java weblogic.ejbc input file output file

Tools will be generating the classes that provide the implementation of our remote interface and home interface.

When EJB is deployed, the info about our EJB home object will be registered in the directory server.

According to EJB specification the server vendor has to supply the tools that generate the classes implementing remote and home interface defined by EJB developers.



(For EJB architecture you can refer fig: Ejbcontainer3.bmp in Suresh_Programs\j2eec\ejb1\ex1)

According to EJB specification a client can't directly access the methods on a bean.

A stateful session bean can remember the state of the client. A stateless session bean can't remember the state of the client.

If the client has to execute on remote machine we need to copy client class, remote interface, home interface and other classes which are used as parameters and return types.

As part of EJB object the code will be provided for take care of security, transactions, object management …

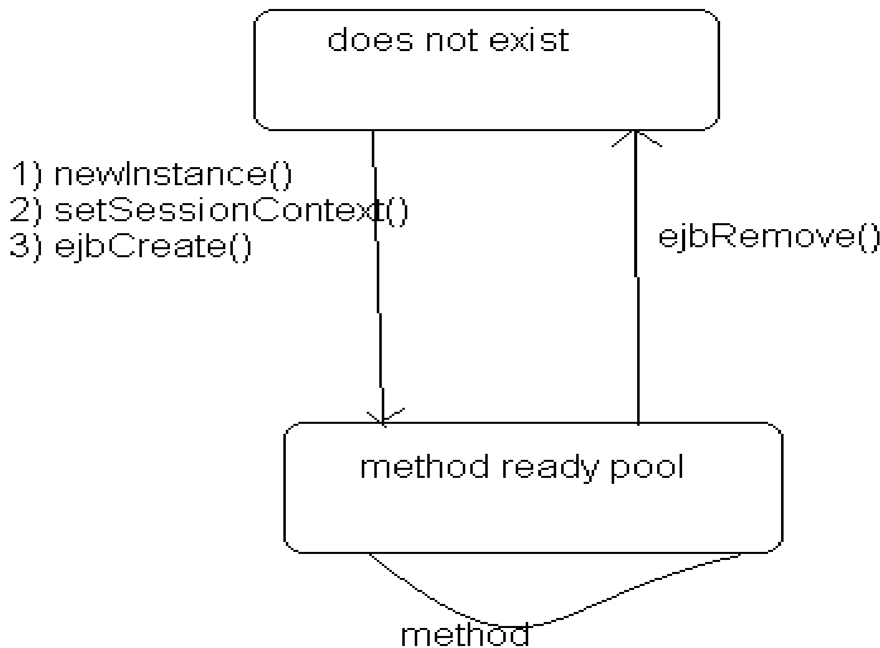
Similar to lifecycle of a *servlet, SessionBean, EntityBean, MessageDrivenBean* has a lifecycle. State diagrams given as part of EJB specification provide us the info about the lifecycle of beans.

Similar to a servlet an EJB will be born (created), it will provide the services to its client (by executing business methods) and it will dead (destroy).

For more info on lifecycle of session beans follow the link
http://java.sun.com/j2ee/sdk_1.2.1/techdocs/guides/ejb/html/Session4.html or

In case of stateless session bean an EJB container decides about when to create, when to destroy a stateless session bean. Big advantage of using EJB is developer need not worry about multithreading. It is the responsibility of server to manage threads.

According to EJB specification the EJB containers has to call *setSessionContext()* followed by *ejbCreate*() method after creating the bean. When the container decides to destroy the bean it will call *ejbRemove()*.



Stateless Session Bean Lifecycle

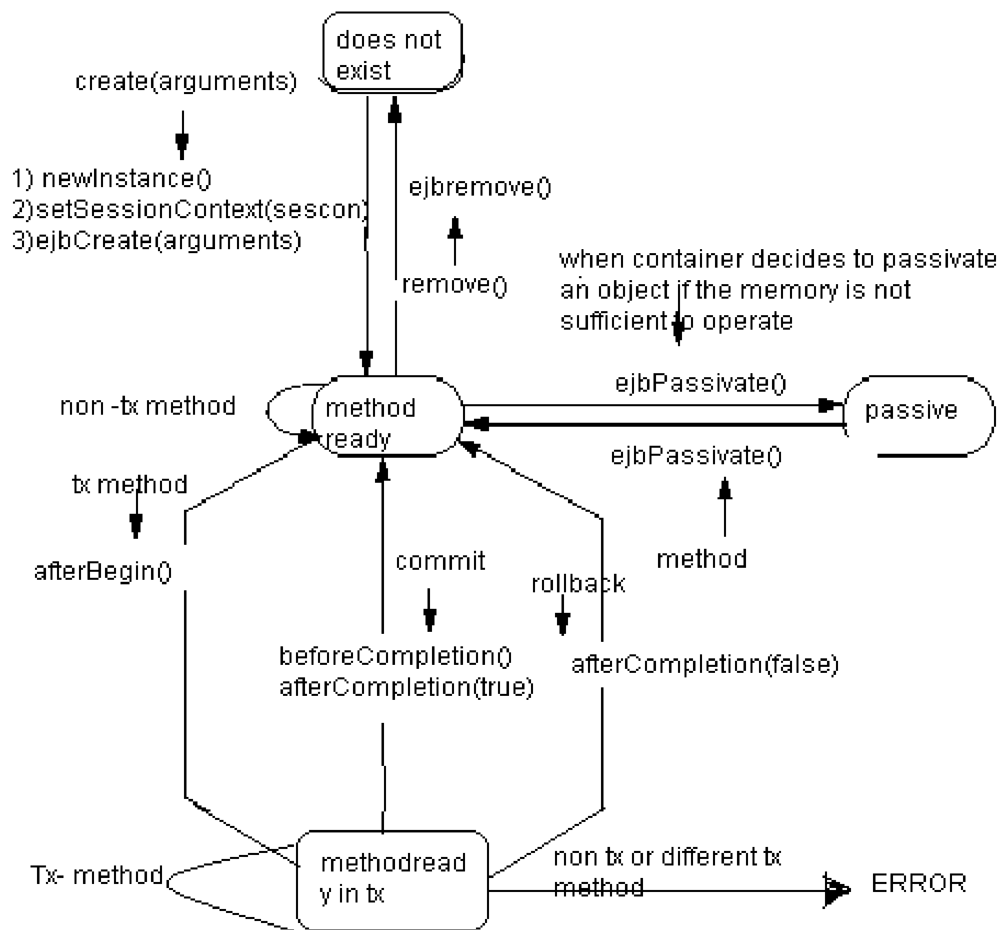
After getting the reference to EJB object, using *home.create* we can execute the business methods. To say that I don't require this stateless session bean we can call the *remove()* method. When we call *remove()* on a stateless session bean the server may or may not destroy our bean.

In case of stateful session bean the server removing the bean after calling *ejbRemove()* when the client executes the *remove()* method.

Since the stateless session bean not created exclusively for a specific client the server need not destroy the bean when the client calls *remove()* method.

In case of stateless session bean the server will never call *ejbActivate()* and *ejbPassivate().*

In case of stateful session bean the server calls *ejbPassivate* before passivating the object (serialization). The server calls *ejbActivate* when it decides to activate the object (deserialization).



Lifecycle of Stateful Session Bean

Most of the server vendors provide the facility ideal timeout and maximum beans in the pool.

We need to write the code that deals with transient objects in *ejbPassivate(), ejbActivate().*

In every business application we categorized the users under different groups ex: in a banking application we can define the roles like Manager, clerk, cashier …. Cashiers perform some tasks which can't be performed by clerks. Most of the EJB containers allow us to define the security constraints like manager can execute *issuedd* method.

Procedure for defining permissions on the methods (in web logic)

- Open the jar file using WLBuilder
- Highlight the jar file name. On right hand side add the names of the roles. To add the principles highlight the name of the role and add the user names.
- To define the permissions on the methods choose methods node of the bean. Choose the add button → choose method name and roles.

In old versions of web logic we can login to the system using username and password that is not defined in web logic. In such a case the user is treated as Guest.

As we are saving the security constraints in deployment descriptors in future we can easily change them according to business requirement.

We can use the methods *getCallerPrinciple, isCallerInRole()* of *EJBContext*

In our EJB projects we can use one of the following methods or both methods to take care of security.

- We can define the security constraints in deployment descriptors
- In our EJB methods we write the code to check the user and role by using the methods provided in *context.*

In case of transactions also we can use automatic transaction management by giving the info in deployment descriptors or in the client we can use JTA and write our own code to control transactions.

To run EJB that uses DB we have to create a data source with JNDI name.

In EJB projects there are 2 ways to access DB

1. By directly writing the JDBC code in session bean.
2. Accessing the data by implementing entity bean.

From EJB to access the DB we need to get the connection from connection pool, perform DB operations and return the connection to connection pool.

To get the initial context in EJB we can use,

initctx = new initialContext();

In EJB code we can't use *Connection.setAutoCommit(true), Cnnection.commit, Connection.rollback*.

If a transaction has to rollback we need to ask the container to rollback transaction. For this we have two different procedures (a) In EJB code we can throw an exception (b) We can use *context.setRollbackOnly*

To test if the transaction is marked for rollback we can use *getRollbackOnly*.

Session Synchronization: In case of stateful session beans we hold the state of client in instance variables. While performing the transaction in some applications we need to write the code that sets the state to the old state when the transaction fails. For these kinds of applications we need to implement *SessionSynchronization* interface.

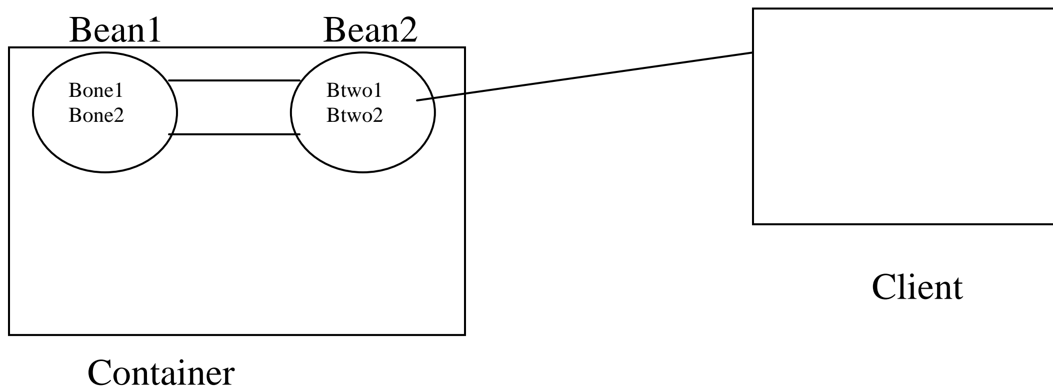
When *SessionSynchronization* is implemented by stateful session bean the server will call *afterBegin()* to indicate that a new transaction is started, before committing the transaction the container will call *beforeCompletion()*, after the transaction is completed it will call *afterCompletion(boolean)* with a value True indicating that it has committed the transaction.

We can set the transactional attributes of an EJB and its methods in deployment descriptors.

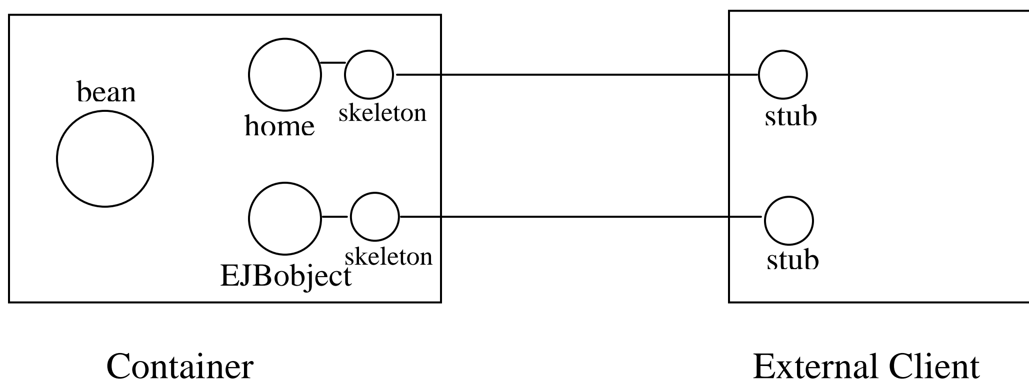
ENTITY BEAN

As part of EJB 2.0 JavaSoft has added the following facilities,

1. New type of bean
2. Support for local interfaces to improve performance
3. Support for new query language EJB QL
4. Support for container managed relationships between entities
5. Support of *ejbSelect*
6. Support for Home methods



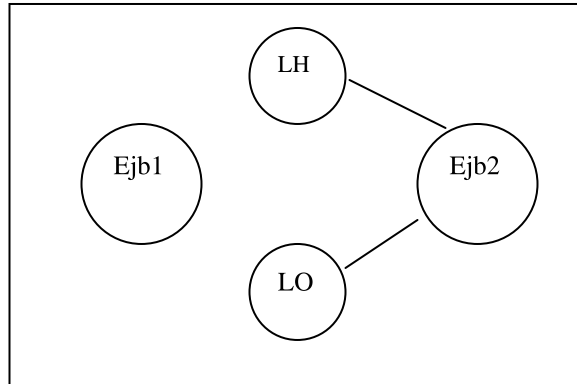
In the above fig we have an external client calling a method on bean2. In this case we can call bean2 as a server object. Bean2 is calling a method on bean1 in this case bean2 is acting as a client – bean2 is an internal client.



External client uses the stubs and skeletons to invoke the methods from other JVM.

In old versions of EJB even the internal clients has to use stub and skeleton. This reduces the performance to improve it JavaSoft has introduced *local* and *localHome* interfaces.

When we use *localHome and localObject* stub and skeletons are eliminated. This reduces the total amount of code that has to be executed and thus improves the performance.



To use above feature we need to provide two more interfaces (a) Local interface (b) Local Home interface

In the client code while looking up for home object we need to use local JNDI name.

We can very easily convert remote interface as a local interface

- Local interface must extend from *EJBLocalObject*
- Methods in this interface must not throw *RemoteException*
- Local home interface must extend *EJBLocalHome*
- *create()* must throw only *CreateException* and the return type of *create()* must be local interface.

Local objects can be used as part of internal clients only. Remote objects can be used as part of internal as well as external clients.

In the servlets/jsp has to be deployed into Web logic for accessing the EJB we need to copy the classes and interfaces that are required into *WEB_INF/classes* of the web application.

To run the servlet that accesses the EJB running on Web logic we need to copy *weblogic.jar* file under *tomcat-home/shared/lib.*

We can write our own code to control the transactions in an external client or in a bean or using *javax.transaction.UserTransaction* object.

J2EE servers store the info about the *UserTransaction* object using the name *javax.transaction.UserTransaction* in directory server.

To control the transaction

- Lookup for *UserTransaction* object using *javax.transaction.UserTransaction* in directory server.
- Start the transaction   *ut.begin()*
- End the transaction   *ut.commit() or ut.rollback()*.

There are two different cases,

- Client may be performing a transaction before it invokes a method.
- Client may not be performing a transaction before it invokes a method

Transactional attributes:

1. RequiresNew:

- If the client is running within a transaction and invokes the enterprise bean's method, the container takes the following steps:
    - **a)** Suspends the client's transaction
    - **b)** Starts a new transaction
    - **c)** Delegates the call to the method
    - **d)** Resumes the client's transaction after the method completes
- If the client is not associated with a transaction, the container starts a new transaction before running the method.

2. Required:

- If the client is running within a transaction and invokes the enterprise bean's method, the method executes within the client's transaction.
- If the client is not associated with a transaction, the container starts a new transaction before running the method.

3. Mandatory:

- If the client is running within a transaction and invokes the enterprise bean's method, the method executes within the client's transaction

- If the client is not associated with a transaction, the container throws the `TransactionRequiredException`

4. Supports:
   - If the client is running within a transaction and invokes the enterprise bean's method, the method executes within the client's transaction.
   - If the client is not associated with a transaction, the container does not start a new transaction before running the method.

5. NotSupported:
   - If the client is running within a transaction and invokes the enterprise bean's method, the container suspends the client's transaction before invoking the method. After the method has completed, the container resumes the client's transaction.
   - If the client is not associated with a transaction, the container does not start a new transaction before running the method.

6. Never:
   - If the client is running within a transaction and invokes the enterprise bean's method, the container throws a `RemoteException`.
   - If the client is not associated with a transaction, the container does not start a new transaction before running the method.

For more info on transactional attributes visit,
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Transaction3.html
To avoid hard coding we can use *EnivironmentEntries* in EJB (similar to initialization parameters in servlets). *EnivironmentEntries* will be stored in deployment descriptors.

To access *EnivironmentEntries*

- Get the *initialcontext*   *Context initial = getInitialContext();*
- Get Environment Context
  Context ctx = (Context)someContext.lookup("java:comp/env");
- *Env1=((String)ctx.lookup("name of the EnvironmentEntry"));*

In stateful session bean we can provide multiple *create()* methods. Corresponding to every *create()* we have to write *ejbCreate()* in bean class.

In our EJB we must not write the code that returns the reference of bean directly (*return this*)
Instead of *return this* we must *return handle* representing the EJB.

For every EJB there will be a handle. *Handle* can be serialize i.e. we can save the *handle* inside a file.

Once the *handle* is saved in a file we can use the *handle* to get the reference of *EJBObject* directly without using *home object*.
To get the reference of *EJBObject* using *handle*

- Deserialize the handle
- Use *Handle= ejo.getHandle();*

To relieve the programmer from the burden of writing threadsafe code JavaSoft place the following restriction,

- EJB container should not call methods concurrently from multiple threads

Container can use

- Serialize the request (place it on queue)

  (Or)
- Maintain a pool of stateless session bean.

On any entity we can perform CURD (Create, Update, Read, Delete).

Two types of entity beans

- BMP (Bean Managed Persistent) entity bean – in this case we need to write JDBC code to manage bean
- CMP (Container Managed Persistent) entity bean – in this we need not write JDBC code

In case of entity beans we can define

- Any number of *create()* methods in home interface
- Any number of *finder()* methods in home interface (minimum one with *findByPrimaryKey)*

An entity bean can survive even if the server crashes.
In BMP we need to provide the code for selecting, deleting, inserting, updating the data in DB.

If multiple columns are used in a primary key the we need to create a *primarykey* class.
In the remote interface we can provide the getters and setters like *getEname, setEname ……*

In the home interface we can provide any number of *create()* methods and one *findByPrimaryKey* method – must throw *FinderException* and return type of this method must be remote.

Additional finders can be provided and return type must be *Collection*.

Entity bean class must extend from *EntityBean* interface. Apart from the methods in session beans, in entity bean we have (a) *ejbLoad* (b) *ejbStore* (c) *unsetEntityContext*

In the *ejbCreate()* we need to provide the code to insert the data in DB.

The *ejbCreate()* must return the primarykey if the data is added successfully, if it fails to insert then it must throw *CreateException.*

Corresponding to *create()* we provide *ejbPostCreate()*

In the *ejbRemove* we need to provide code to delete the entity. If deletion fails we need to throw *NoSuchEntityException*.

ejbStore called by the container to store the data in DB. In this method we write the code that updates the DB.

In *ejbLoad* we need to write the code that reads the data of an entity.

In finders methods we need to use the select statement and retrieve a primary key or a collection of primary key.

In case of entity beans EJB containers need not serialize the entity bean as the data is available in DB.

Today in 90% cases we can use CMP bean.
In following cases we use BMP,

- If we are not able to debug the code generated by the container.
- If there are bugs in the code generated by the container.

- If the code generated by the container is insufficient.

CMP:

- The procedure for developing Home and Remote interfaces in CMP is same as BMP.
- In EJB 2.0 we need not declare instance variables to hold the data
- In CMP beans we need not write the code in *ejbLoad, ejbStore, ejbRemove* for performing DB operations.
- Provide abstract methods to get and set the attributes.
- In *ejbCreate* we need to set the attribute by calling *setxxx* methods.
- The container specific tool will generate the subclass of our bean class. In this subclass the abstract methods will be implemented.
- Code for the finder methods will be generated automatically by the container using *ejbQL* provided by us in deployment descriptors.
- In the old versions a developer need to learn the query language specific to a container but with 2.0 all containers support the same query language (ejbQL).
- JDBC code will be available as part of the subclass.

I included EJB design patterns in chapter 6 (all design patterns)

QL:

With EJB 2.0 JavaSoft has introduced the Home methods. As part of Home interface we can provide additional methods.
We can use these methods to perform an operation that is not specific to an entity. Ex: Get the number of employee working in a dept, get the total salary paid to all employees.

JavaSoft has provided *ejbSelect* method to eliminate JDBC code in our bean (very much similar to finder method).

ejbSelectxxx must be declared as an abstract method in CMP bean.
Ex: *public abstract Set ejbSelectGetNOS() throws javax.ejb.FinderException;*

In every business application we need to store data about different entities. During the design the DB developers will be identifying the relation between different entities. Relationships depend upon the project requirements

ex: (a) If a bank maintains only one address for every customer then the relationship is one to one relationship. (b) If the application allows a customer to give multiple addresses then the relationship between customer and addresses is one to many. (c) If one customer of a bank can maintain multiple accounts and multiple customers can jointly hold a single account then the relationship is many to many.

Some of the DB servers provide cascading delete option. Ex: in such DB when we execute Delete from customer where cust_id =1 the server delete the rows that are marked with *

	cid	name	…
*	1	one	…
	2	two	…
	3	three	….

Customer

	Cust-Addrid	addrid	addr
*	1	1	…
*	1	2	….
	2	1	…

cust_addr

If a student can take multiple courses and multiple students can take a course then the relationship is many to many. For representing this info we can use three different tables as,

Sid	name	…
1	one	…
2	two	…
3	three	…

Student

cid	cname	..
1	J2EE	…
2	.NET	
3	Oracle	

courses

Scid	sid	cid
1	1	1
2	1	2
3	1	3
4	2	3

Student_course

For more info on DB structure visit the site,
http://www.georgehernandez.com/xDatabases/aaIntro/DBStructure.htm

In EJB we can specify cascade delete option. In this case the detail entities will be removed when we remove the master entity. This is supported even if the DB doesn’t directly support cascade delete option.

If a primary key has more than one column then we need to create a separate class representing primary key.

Primary key class must be serializable and we must provide *equals()* and *hashcode()* as part of primary key class.

Message Driven Bean (MDB):

To simplify the implementation of message processing applications we can utilize Message Driven Beans (MDB). Advantages are,

- We can reduce the amount of code
- As the code runs inside J2EE server we can utilize all the facilities that are provided by the server.

A message bean doesn't have a home, localhome, remote, local interfaces.

Message beans are stateless. They cannot give a return value, but they can post a message back on the queue.

Provide a bean class implementing *MessageDrivenBean* and *MessageListener*. MDB has to implement *javax.jms.MessageListener's onMessage* method and *javax.ejb.MessageDrivenBean's ejbRemove* and *setMessageDrivenContext* methods. Message driven bean have a single method *onMessage()*. This method will be called when the message is placed in queue.

In deployment descriptor of the bean we have to specify the destination type and JNDI name of the destination.

MDB can be associated with a queue or with a topic.

JMS (Java Messaging Service)

IBM MQ series, MSMQ servers, Oracle message queues are some of MOM (Message Oriented Middleware) products.

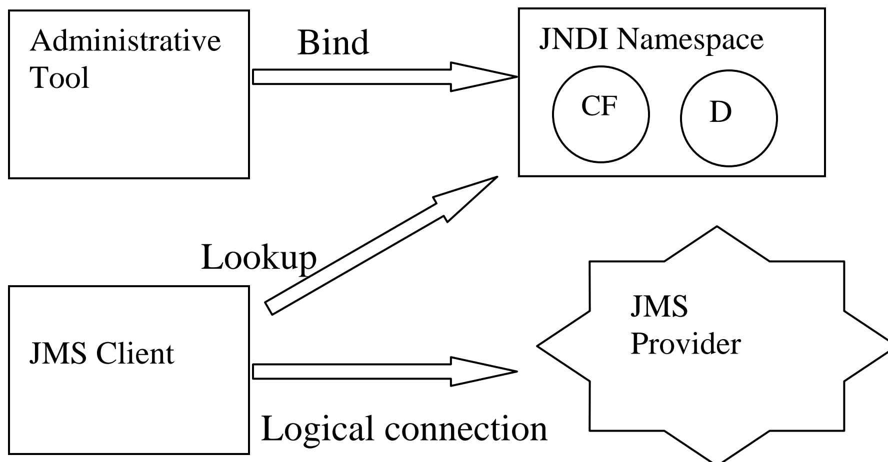
Apart from these products most of J2EE venders has provided a messaging server as part of their products. These products can be used to build highly reliable, loosely coupled applications.

In projects uses distributed technologies like RMI, CORBA the clients are tightly coupled with the server i.e. a change in the interface exposed by the server requires a change in the client.

In MOM products the client need not be in Sync with the server. The client need not even bother whether the server is running or not.

While sending the message we can specify whether the message has to be persisted or not. To balance the load and to take care of failure we can configure a cluster of MQ servers.

Java programmer can use JMS API to deal with MOM products.

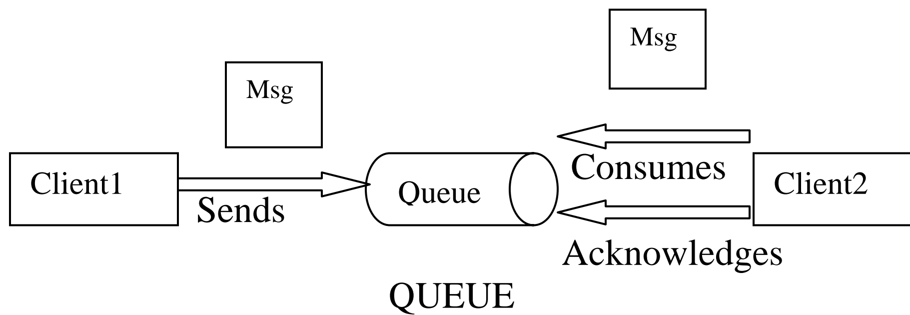


Configuring Weblogic to act as messaging server

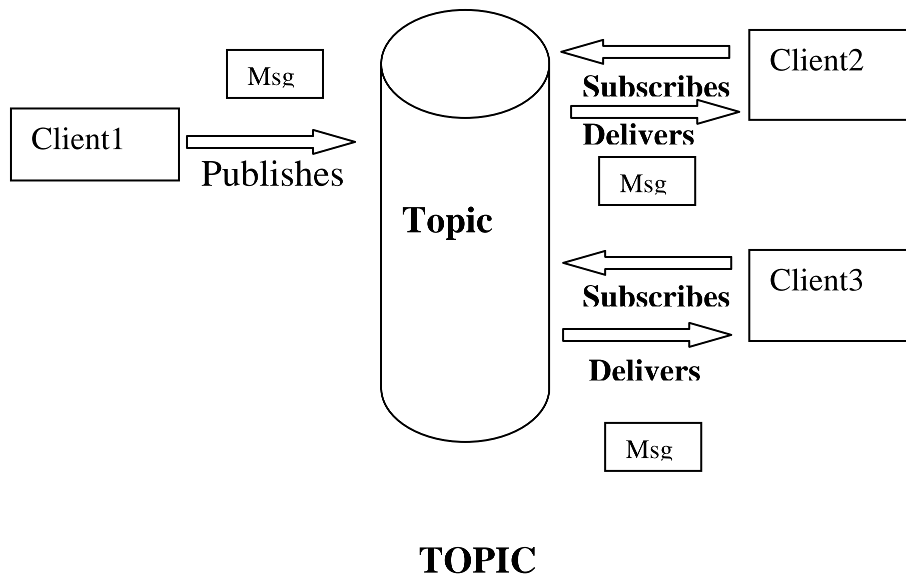
- Choose JMS→stores
- Configure new jms file store

- Enter the name for store and the directory → create
- JMS servers → configure new JMS server → enter the name of the server → choose the store → create. On targets tab choose the target and click apply
- JMS connection factories → configure new JMS connection factory → create. Choose target and click apply
- We need to create the destinations (topic/ queue)
- Choose JMS servers→ server created by you→ destination→ configure new destination→ give the name, JNDI name → enable store→ create.

There are two types of destinations (a) Queue (b) Topic.



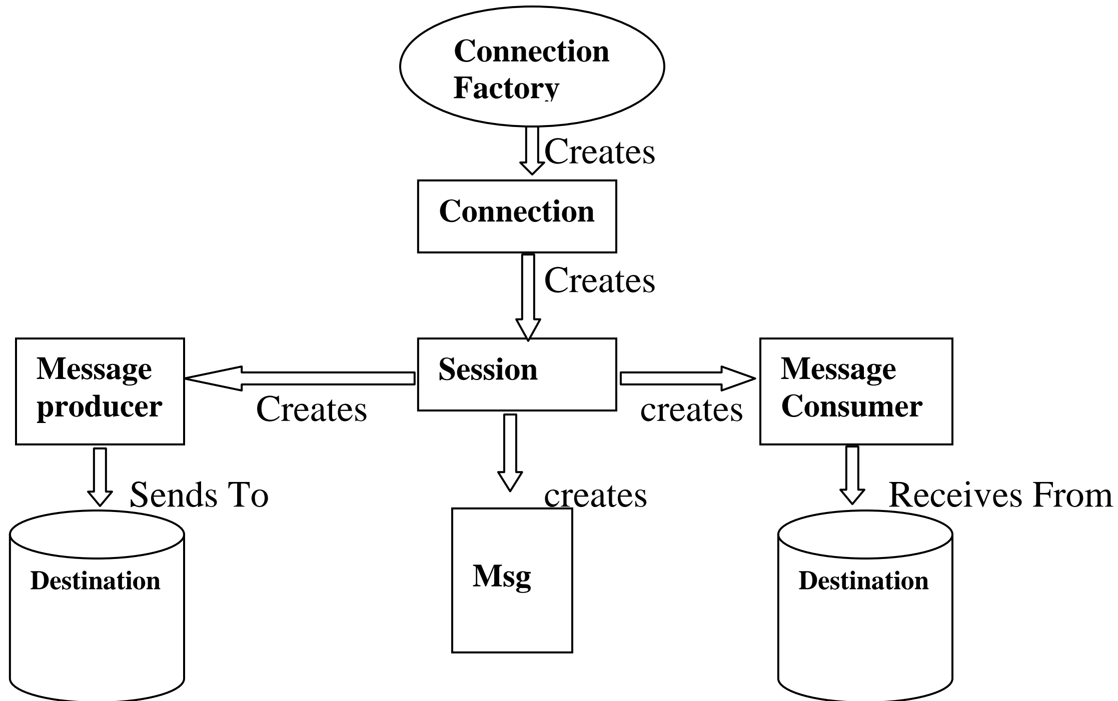
QUEUE



TOPIC

There will be timing dependency between publisher and subscriber. Same message will be delivered to multiple subscribers.

Compare to Queue Topic is not that reliable but we can use durable subscription to improve the reliability of Topic.



There are two ways of receiving the messages (a) By constantly waiting for incoming messages (b) By implementing *MessageListener* interface. This interface contains *OnMessage()*.

JMS system will be calling *OnMessage()* whenever a message has to be delivered.

JMS implementation is responsible for creating the threads and executing *OnMessage().*

In *CreateQueueSession* we can use the constant *Session. AUTO_ACKNOWLEDGE.* In this case we need not to write code to send acknowledgement. We can also use *Session.CLIENT_ACKNOWLEDGE* –in this case we need to write our own code to send an acknowledgement.

After receiving a message we need to execute *message.acknowledge()*

When we use *AUTO_ACKNOWLEDGE* the acknowledgement will send automatically when *OnMessage()* is executed completely. When we want to control, when the acknowledgement is send then we can set *CLIENT_ACKNOWLEDGE.*

Programmatically we can specify that the message must not be persistent using *qsender.setDeliveryMode(DeliveryMode.NON_PERSISTENT)*

We can perform the transactions by specifying first parameter as true in *createSession().*

In case of DB we perform operations like CURD in a transaction similarly in case of JMS we perform operations like sending a message or receiving a message in a transaction.

On senders side if we post a couple of messages in a transaction and execute *commit* the messaging system assumes that two messages are posted if we execute *rollback* the messages will be removed from the queue.

In the receiver side after receiving n messages if *commit* is executed an acknowledgement will be send to the server and the server removes the messages from the server. If the receiver executes *rollback* the server assumes that the receiver has not received the messages and it will redeliver the messages.

To use durable subscriptions we need to create JMS factory with a client id.

JAVA MAIL

There are multiple protocols used for sending and receiving the mails. Most popular protocols for sending the mail is SMTP and for downloading the mails is POP3.

Java Mail API can be used to access or send the mails using different protocols. SUN has provided the implementation of Java Mail API using POP3, SMTP and I-map protocols.

So many vendors have implemented Java Mail for other protocols.

Setting up James Server:

- Create a directory like D:\mailser
- Extract the contents of jamesxxx.zip to it
- Check documentation for configuring James Server.
- Set JAVA_HOME environment variable.
- Start the James Server using *run* command available in *bin*

To create James server users start telnet (*run/telnet*) and *open sysname portno username root pwd root*

To add the users use *adduser uname pwd*

To setup mail client we need

- Our e-mail address
- Incoming mails POP server address
- Outgoing SMTP server address

To develop a program for sending a mail we need to know the name and IP address of outgoing mail server.

For sending the mails we need not provide the username and password.

To send a mail using Java Mail API

- Establish a connection with mail server
- Create a message object and set different properties like To, From, Subject….
- Send the message using *Transport.send*

To exchange different types of content as part of a mail MIME format is designed.

Using MIME format we can send multiple attachments. We can assume a MIME message as a combination of multiple MIME parts.

Different parts of MIME message is separated by a boundary.

To send the attachments from one java mail program we have to use *MIMEmessage, MIMEMultipart, MIMEBodyPart* classes. We will not directly wriing the code to create a MIME message.

To deal with different kinds of content we use classes, interfaces that are part of *JavaActivationFrameWork.*

Some mail servers provide a facility of creating a folder on the server.

Even though POP server doesn't provide the concept of folders Java programmers can assume that it provides one folder with the name inbox.

To download the mails

- Establish the session
- Connect to the store
- Open the folder
- Download the messages
- Close the session

Most of the mail servers provide a facility of marking the mail for deletion. After marking we can expunge the messages.

According to Java Mail API documentation the messages can use *folder.expunge* to permanently delete the messages. This method is not implemented for POP3 for this we can use *folder.close(true)*

WEB SERVICES

A web service is a piece of software that explores a set of operations. Web service can be accessed by using a URL. According to the specification, web service is platform independent, language neutral and it can be accessed using wide variety of protocols.

A java programmer can assume a web service is like a RMI object.

Setting-up WebLogic for developing web services using WebLogic Workshop.

- Start Domain Configuration wizard
1. Choose WebLogic workshop template
2. Enter username, password and create.
- Setting up Workshop tool
1. open WebLogic Workshop
2. choose tools→preferences→paths→ setup the configuration directory → choose the domain workshop and click ok
- Start the WebLogic server. Tools→ Start WebLogic server

Different server vendors provide different options for implementing a web service. A J2EE server may provide a facility to implement a web service as a servlet or as a plain java object or as a EJB …

In object orientation, calling a method is considered as passing a message. In web services calling a method on a web service is considered to be passing a request message. Once the request is processed web service returns a response message.

Most of the web services implementation uses SOAP for sending a request message and getting response message.

SOAP with attachments can be used to pass non xml data (image, html, audio …)

A java programmer can use SAAJ (SOAP with Attachment API for Java)

According to the SOAP an application can return an exceptional conditional by passing a message containing SOAP Fault.

Most of the applications display the error code, description of the error and cause of the error. In case of SOAP, to report a fault we can use SOAP Fault which contains *faultcode*, *faultstring*, *faultactor*…

In web services we will be using WSDL (Web Service Description Language) files to provide the info about one web service. This file is very much similar to IDL in CORBA.

In case of web services we use the term proxy in place of stub. A proxy can be used on the client to access a web service.

In case of WebLogic we need to implement

- One or more java classes
- Session bean
- Message Driven Bean

To access WSDL file representing the web service we can use,
http://server:port/contest patth/service uri?WSDL

WSDL file contains
Types – our own data types will be defined
Message – in this info about request message and response message for our operations will be defined.
Port—operations will be defined
Binding—in this the info about transport protocol as well as how the messages will be formatted is defined.
Service—port name and URL to access the service will be defined.

In WebLogic we can implement a handler which can intercept request, response and fault messages. An intercept is very similar to a filter in servlet.

Similar to CORBA, web services supports one-way operations. If an operation is one-way then the client sends a request for execution of methods but it will not wait for response.

In WebLogic we can use MDB for implementing one-way operation.

Procedure for developing web service in WebLogic:

- Setup the environment by running *setenv* command and set classpath to current directory
- Run the *ant* command and ensure that it is working
- Develop a session bean, MDB or multiple plain java classes
- Create a build file and in this use *servicegen* and *clientgen*
- Run the *ant* tool to generate ear and jar files.
- To deploy the web service copy the ear file to applications directory
- To test the web service open the browser use the URL

There are multiple ways of developing client application.

Simple way of developing client application is,

- Create a *service* object
- From *service* object create a *port* object
- Execute the web service methods by using *port* object

To run the client use *ant run*

Companies like Microsoft, xmethods, IBM are running public UDDI register in which anyone can publish the info about their web services.

We can write the client programs to access a web service using WSDL file.

Client type1 uses the classes generated by clientgen tool

- Create service object using the class generated by clientgen
- Using service create a port object
- Using port call the operations.

Client type2

- Create service factory
- Using service factory create service object with the name of the service and WSDL file as parameters.
- Using port name and the interface name create a port object.
- Using port invoke the operations

Client type3 – dynamically involve method without using WSDL

- Create service factory
- Create service using service factory with only service name as parameter
- Create a call object from service factory
- Set input/output parameters
- Using call object invoke operations

As web services technology is independent of language, platform we can use this technology to integrate different applications easily ex: if a company is already using a purchase order application implemented in c++ instead of throwing away the old code we can create the web services in c++ to access the purchase order application. The new project can be developed in java and this project can use the purchase order system by using the methods exposed by web services.

Similar to session timeout incase of servlet most of the servers provide an option of defining timeout for stateful session bean and entity bean. If the bean is not used for this much amount of time server will be removing the objects. This is how we can get rid of unused objects.