MIDDLEWARE TECHNOLOGIES NOTES FOR B.E - IT

UNIT-I

CLIENT/SERVER CONCEPTS: Client – Server – File Server, Database server, Group server, Object server, Web server. Middleware – General middle ware – Service specific middleware. Client/Server Building blocks – RPC – Messaging – Peer – to – Peer. Web Services- SOA, SOAP, WSDL, REST Services.

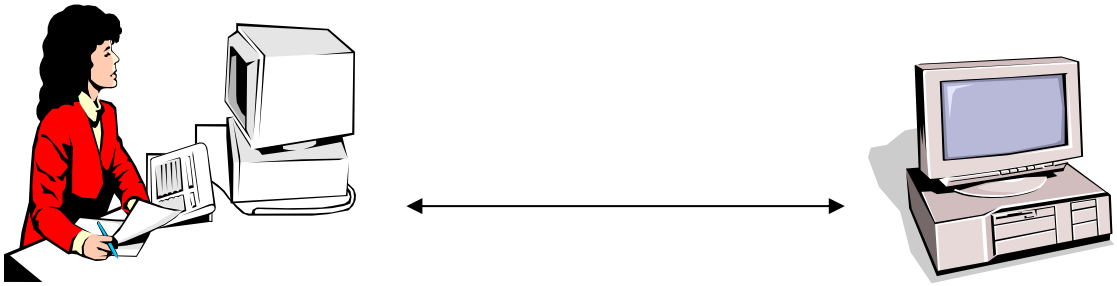
Source: http://middlewares.wordpress.com/2008/01/13/what-is-middleware/

Concepts:

Client – Server – File Server:

In client/server computing, processes are divided between the client and the server. This relationship is based on a series of requests and responses:

- Clients
- Servers
- Communication Networks



Client **Server**

Q1. What are Clients and Servers? Examples?

Clients: "Clients are Applications"

- Request sender is known as client
- Initiates requests
- Waits for and receives replies.
- Usually connects to a small number of servers at one time
- Typically interacts directly with end-users using a graphical user interface

Example: E-mail client

Servers: "Servers Manage Resources"

- Receiver of request which is send by client is known as server
- Passive (slave)

- Waits for requests from clients
- Upon receipt of requests, processes them and then serves replies
- Usually accepts connections from a large number of clients
- Typically does not interact directly with end-users

Real Time Examples:

Most transactions that occur on the Internet are client/server based. Some examples include:

- FTP (file transfer protocol) - An FTP client program contacts an FTP server and requests the transfer of a file; the FTP server responds by transferring the file to the client.
- WWW (World Wide Web) - In this case the client program is a browser. A browser requests the contents of a web page and displays the results on the user's computer.
- E-MAIL - A mail client program enables the user to interact with a server in order to access, read and send electronic mail messages.

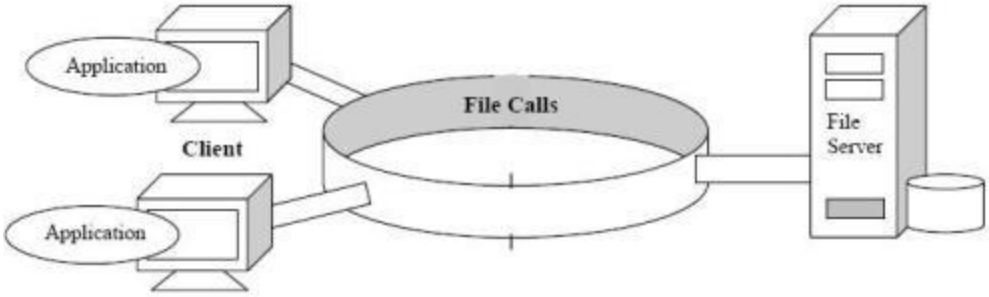
Q2. What is Server and Server types?

A server is a system (software and suitable computer hardware) that responds to requests across a computer network to provide, or help to provide, a network service and it can provide several services and have several servers running.

- File Server
- Database Server
- Transaction Server
- Groupware Server
- Object Server
- Web Server

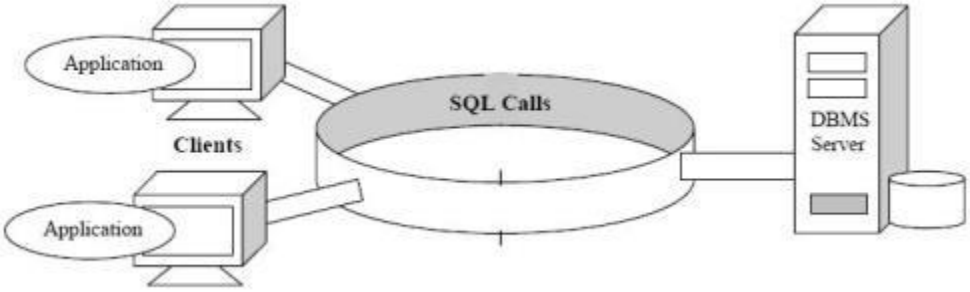
1. File Servers

a) The client passes requests for file records over a network to the file server.
b) This is a very primitive form of data service.
c) File Servers are useful for sharing files across a network.
d) These are acting as a repository of documents, images, engineering drawings and other large data objects.



2. Database Servers

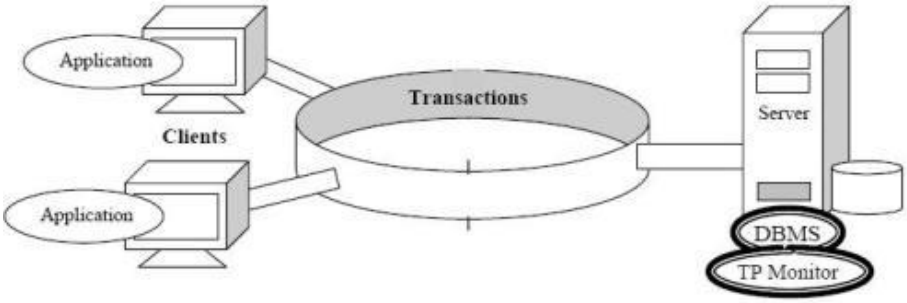
a) The client passes SQL requests as messages to the database server.
b) The result of each SQL command are returned over the network to the client.
c) The code in the server process will processes the SQL request and the data reside in the same machine.
d) Distributed database servers may increase the efficiency of the processing power.
e) These servers provide the foundation for decision-support systems that require adhoc queries and flexible reports.



3. Transaction Servers

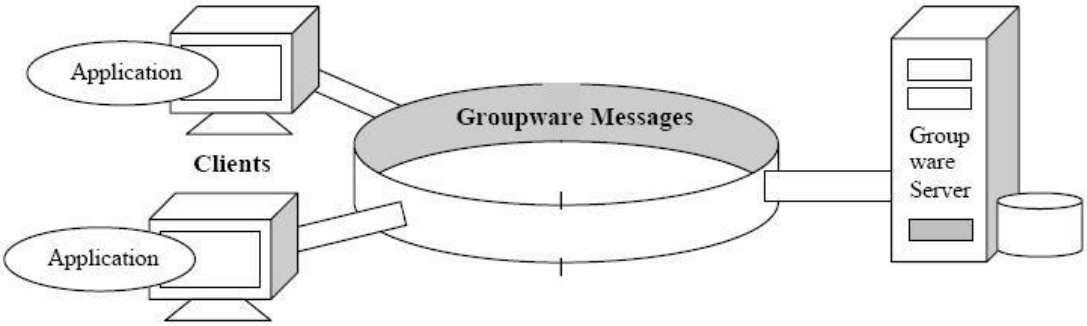
a) The client invokes remote procedures that reside on the server with an SQL database engine.
b) These remote procedures on the server execute a group of SQL statements.
c) The network exchange consists of a single request/reply message.
d) The SQL statements either all succeed or fail as a unit. These grouped SQL statements are called Transactions.
e) The server component usually consists of SQL transactions against a database
f) These are called Online Transaction Processing or OLTP.
g) OLTP applications also require tight controls over the security and integrity of the database.
h) Two forms of OLTP: based on the TP Monitors provided by the OLTP Vendors

- TP Lite
- TP Heavy



4. Groupware Servers

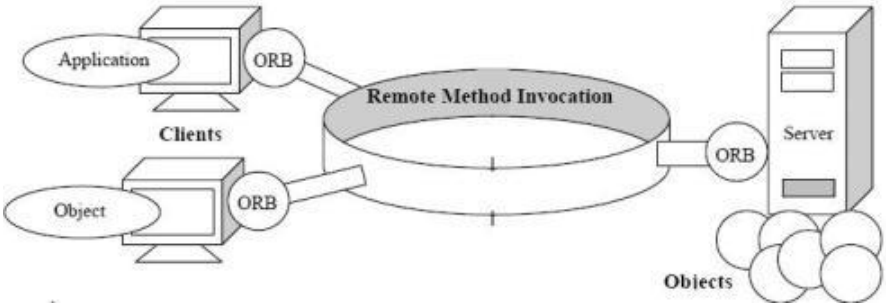
a) Groupware addresses the management of semi-structured information such as text, image, mail, bulletin boards, and the flow of work.
b) These client/server systems place people in direct contact with other people.
c) Lotus Notes is the Leading Example.
d) In most cases, applications are created using a scripting language and form-based interfaces provided by the vendor.
e) The communication middleware between the client and the server is vendor-specific.



5. Object Servers

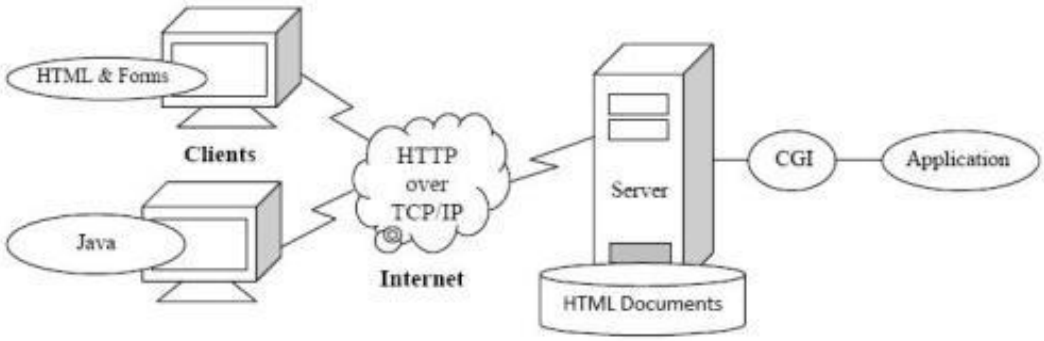
a) The client/server application is written as a set of communicating objects.
b) Client objects communicate with server objects using an Object Request Broker (ORB).
c) The client invokes a method on a remote object.
d) The ORB locates an instance of that object server class, invokes the requested method, and returns the results to the client object.
e) Server objects must provide support for concurrency and sharing. The ORB brings it all together.

Example: Digital's Object Broker, IBM's SOM 3.0, Sun's NEO, HP's ORB Plus, Expersoft's Power Broker, Microsoft's DCOM or Network OLE.



6. Web Servers

a) WWW is the first truly intergalactic client/server application.
b) This model of client/server consists of thin, portable, "universal" clients that talk to Superfast Servers.
c) The clients and servers communicate using an RPC-like protocol called HTTP.
d) This protocol defines a simple set of commands, parameters are passed as strings.
e) The collection of HTML documents are stored in the Web Server.



Q3. What is Middleware? What is General Middleware and Service Specific Middleware?

Middleware does not include the software that provides the actual service that's in the server's domain.

It also does not include the user interface or the application's logic that's in the client's domain.

It starts with the API set on the client side that is used to invoke a service, and it covers the transmission of the request over the network and the resulting response.

Middleware divided into two broad classes:

(a) General Middleware

(b) Service-Specific Middleware.

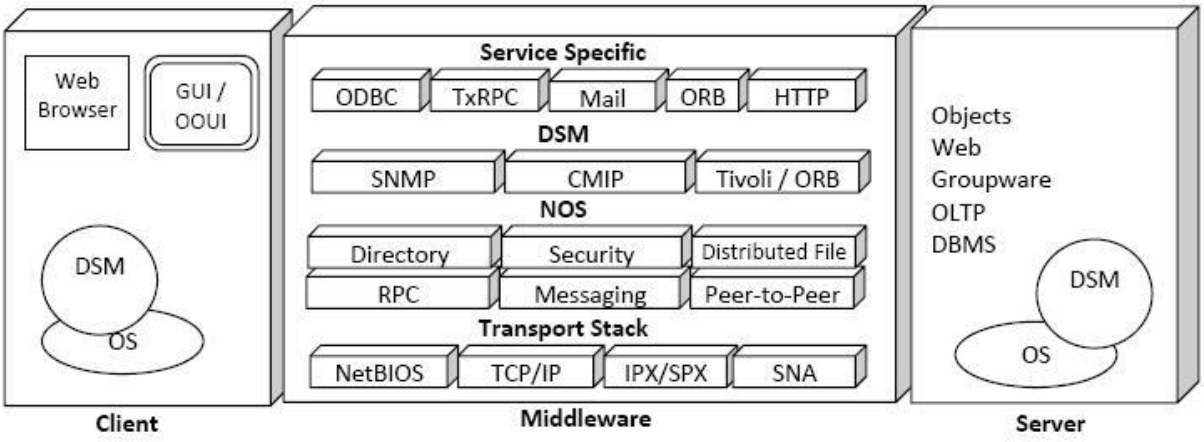
(a) General Middleware:

a) It is the substrate for most client/server interactions
b) It includes the communication stacks, distributed directories, authentication stacks, distributed directories, authentication services, network time, remote procedure calls, and queuing services.
c) Products that fall into the general middleware category include DCE, ONC+, NetWare, NamedPipes, LAN Server, LAN Manager, Vines, TCP/IP, APPC and NetBIOS.
d) Message Oriented Middleware (MOM) products from Peerlogic, Covia, Message Express, System Strategies and IBM.
e) These are depends on message queue system and increases portability, interoperability, flexibility.

(b) Service-Specific Middleware:

a) It is need to accomplish a particular client/server type of service.
b) Database-specific middleware such as ODBC, DRDA, EDA/SQL, SAG/CLI and Oracle Glue.
c) OLTP-specific middleware such as Tuxedo's ATMI and /WS, Encina's Transactional RPC, and X/Open's TxRPC and XATMI
d) Groupware-specific middleware such as MAPI, VIM, VIC, SMTP and Lotus Notes Calls
e) Object-specific middleware such as OMG's CORBA and Microsoft's Network OLE (or DCOM)
f) Internet-specific middleware such as HTTP, S-HTTP and SSL
g) System Management-specific middleware such as SNMP, CMIP and ORBs.

Q4. Client/Server Building blocks?



The Client Building Block

a) Runs the client side of the application
b) It runs on the OS that provides a GUI or an OOUI and that can access distributed services, wherever they may be.

c) The client also runs a component of the Distributed System Management (DSM) element.

The Server Building Block

a) Runs the server side of the application
b) The server application typically runs on top of some shrink-wrapped server software package.
c) The five contending server platforms for creating the next generation of client/server applications are SQL database severs, TP Monitors, groupware servers, Object servers and the Web server.
d) The server side depends on the OS to interface with the middleware building block.
e) The server also runs DSM component
f) It may be a simple agent or a shared object database etc.

The Middleware Building Block

Runs on both the client and server sides of an application

This broken into three category

- Transport Stacks
- NOS
- Service-specific middleware

Middleware is the nervous system of the client/server infrastructure

This also has the DSM component

DSM

a) Runs on every node in the client/server network.
b) A managing workstation collects information from all its agents on the network and displays it graphically.
c) The managing workstation can also instruct its agents to perform actions on its behalf.

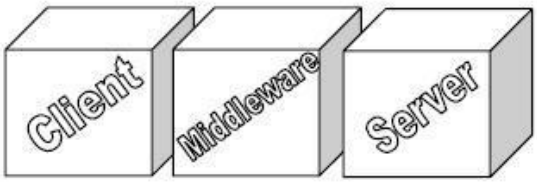
Server-to-server Middleware

- Server-to-server interactions are usually client/server in nature – servers are clients to other servers.
- However, some server-to-server interactions require specialized server middleware. For example, Two-Phase commit protocol may be used to coordinate a transaction that executes on multiple servers.
- Servers on mail backbone will use special server-to-server middleware for doing store-and-forward type messaging.
- But most modern software follows the client/server paradigm.

Q5. Client/ Server : A one size fits all model?

The building blocks of client/server applications are:

- Client
- Middleware
- Server



These building blocks can be rearranged to use them in the following **situations**:

1. Client/Server for tiny shops and nomadic tribes – This is a building-block implementation that runs the client, the middleware software, and most of the business services on the same machine. It is the suggested implementation for the one-person shops, home offices, and mobile users with well-endowed laptops.

2. Client/Server for small shops and departments - This is the classic Ethernet client/single-server, building block implementation. It is used in small shops, departments, and branch offices. This is the predominant form of client/server today.

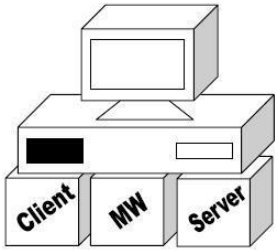
3. Client/Server for intergalactic enterprises – This is the multiserver building-block implementation of client/server. The servers present a single system image to the client. They can be spread out throughout the enterprise, but they can be made to look like they are part of the local desktop. This implementation meets the initial needs of intergalactic client/server computing.

4. Client/Server for a post-scarcity world – This model transforms every machine in the world into both a client and a server. Personal agents on every machine will handle all the negotiations with their peer agents anywhere in the universe. This dream is almost within reach.

Working:

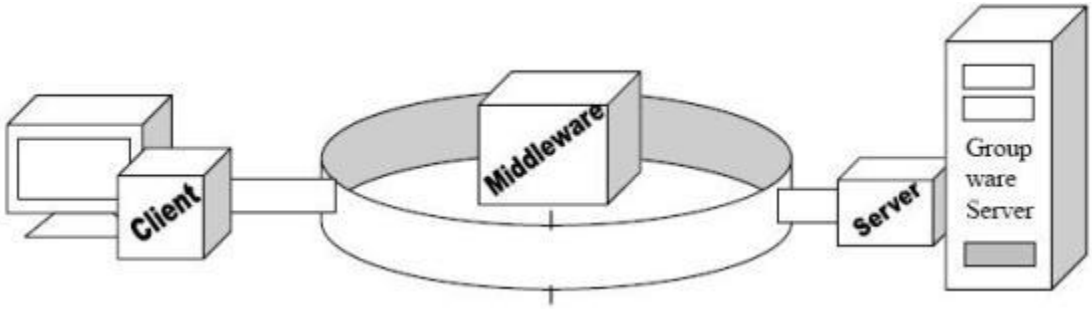
1) Client/Server for Tiny Shops and Nomadic Tribes

a) It is easy to run the client and server portion of an application on the same machine.
b) Vendors can easily package single-user versions of a client/server application.
c) The business critical client/server application runs on one machine and does some occasional communications with outside servers to exchange data, refresh a database and send or receive mail and faxes. Ex: Internet.



2) Client/Server for small shops and departments

a) The client/server architecture is particularly well-suited for the LAN-based single server establishments.
b) It consists of multiple clients talking to a local server.
c) This is the model used in small businesses.
d) The single-server nature of the model tends to keep the middleware simple.
e) The client only needs to look into a configuration file to find its server's name.
f) Security is implemented at the machine level and kept quite simple.
g) The network is usually relatively easy to administer; it's a part-time job for a member of the group.
h) There are no complex interactions between servers, so it is easy to identify failures- they're either on the client or on the local server.

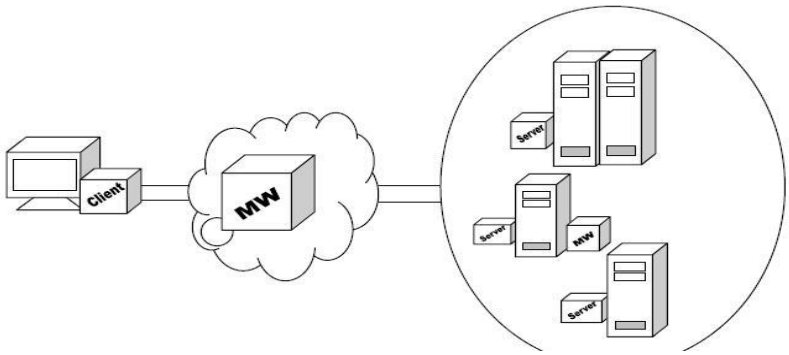


3) Client/Server for Intergalactic Enterprises:

a) The client/server enterprise model addresses the needs of establishments with a mix of heterogeneous servers.
b) These models are upwardly scalable.

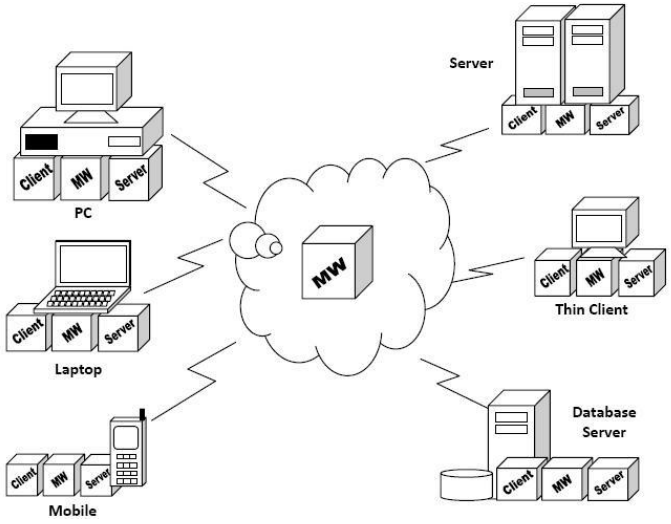
c) When more processing power is needed for various intergalactic functions, more servers can be added, or the existing server machine can be traded up for the latest generation of superserver machine.
d) Multiserver capability, when properly used, can provide an awesome amount of compute power and flexibility, in many cases rivaling that of mainframes.
e) To exploit the full power of multiservers, we need low-cost, high-speed bandwidth and an awesome amount of middleware features -including

- network directory services
- network security
- remote procedure calls and
- network time services.



4) Client/Server for a Post-Scarcity World

a) Every machine is both a client and a full-function server.
b) Because every machine is a full-function server, it will run, at a minimum, a file server, database server, workflow agent, TP Monitor, and Web server – all connected via an ORB.
c) This is in addition to all the client software and middleware.
d) In next few years, a hundred million machines or more may be running almost all the forms of client/server software
e) In this model instead of mobile agents, personal agents will be used.



Q6. MOM Vs RPC?

Feature	MOM	RPC
Metaphor	Post-office like	Telephone like
Client/Server time relationship	Asynchronous. Clients and Servers may operate at different times and speeds.	Synchronous. Clients and Servers must run concurrently. Servers must keep up with clients.
Client/Server Sequencing	No fixed sequence	Servers must first come up before clients can talk to them.
Style	Queued	Call-Return
Partners need to be available	No	Yes
Load-balancing	Single queue can be used to implement FIFO or priority based policy	Requires a separate TP Monitor.
Transactional Support	Yes (Some Products) Message Queue can participate in the commit synchronization	No. Requires a Transactional RPC.
Message Filtering	Yes	No
Performance	Slow. An intermediate hop is required	Fast
Asynchronous processing	Yes. Queues and triggers are required Limited.	Requires threads and tricky code for managing threads.

Q7. Remote Procedure Call?

a) "RPCs are not procedure calls at all, they are truly process invocations. The invoked program runs across the wire in a different resource domain"
b) A client process calls a function on a remote server and suspends itself until it gets back the results.
c) Parameters are passed like in any ordinary procedure.
d) The RPC, like an ordinary procedure is synchronous.
e) The process (or threads) that issue the call waits until it gets the results.
f) Under the covers, the RPC run-time software collects values for the parameters, forms a message, and sends it to the remote server.
g) The server receives the request, unpacks the parameters, calls the procedure, and sends the reply back to the client.
h) While RPCs make life easier for the programmer, they pose a challenge for the NOS designers who supply the development tools and run-time environments.

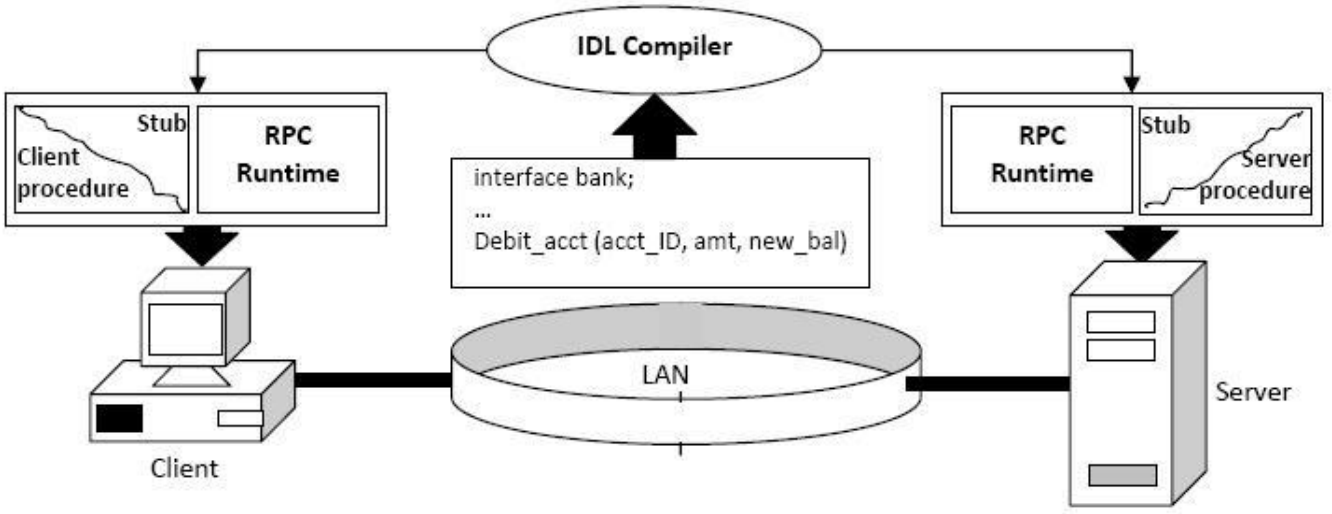
The Common issues are:

How are the Server functions located and started?

a) Server starts the process, when a remote invocation is received with necessary parameters and returns the response to the client.
b) What happens when multiple clients invoke the same function? Now an environment is needed to start and stop servers, prioritize requests, perform security checks, and provide some form of load-balancing.
c) Each incoming requests invokes a thread in the server side.
d) A server loop is created to manage the pool of threads waiting for work rather than create a thread for each incoming request.
e) TP Monitors are really need on the server side, which provides more functions than a NOS.

How are parameters defined and passed between the client and the server?

a) The better NOSs provide an Interface Definition Language (IDL) for describing the functions and parameters that a server exports to its clients.
b) An IDL compiler takes these descriptions and produces source code stubs (and header files) for both the client and server.
c) These stubs can then be linked with the client and server code.
d) The client stubs packages the parameters in an RPC packet, converts the data, calls the RPC run-time library and waits for the server's reply.
e) On the server side, the server stubs unpacks the parameters, calls the remote procedure, packages the results, and sends the reply to the client.



How are failures handled?

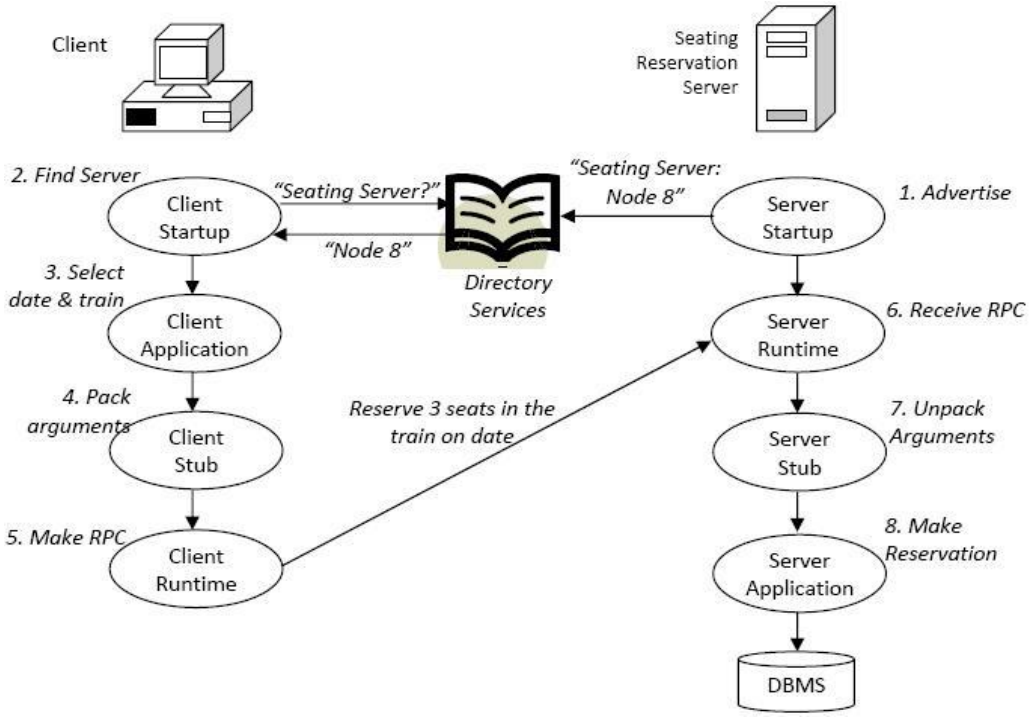
a) Both the sides of the RPC can fail separately, it is important for the software to be able to handle all the possible failure combinations.
b) If the server does not respond, the client side will normally block, timeout, and retry the call.
c) The server side must guarantee only once semantics to make sure that a duplicate request is not re-executed.
d) If the client unexpectedly dies after issuing a request, the server must be able to undo the effects of that transition.

How is security handled by the RPC?

a) Modern NOSs, like DCE – make it easy to automatically incorporate their security features into the RPC.
b) All you need to specify is the level of security required; then the RPC and security feature will cooperate to make it happen.

How does the client find its server?

a) The association of a client with a server is called binding.
b) The binding information may be hardcoded in the client.
c) The client can find its server by consulting a configuration file or an environment parameter.
d) A client can also find its server at run time through the network directory services. The server must, of course, advertise their services in the directory.
e) The process of using the directory to find a server at runtime is called dynamic binding
f) RPC can be used to find a server. The RPC client stub will locate a server from a list of servers that support the interface. This is called automatic binding.



How is data representation across systems handled?

a) The problem here is that different CPUs represent data structures differently (Ex: bigendian Vs little endian)
b) To maintain machine independence, the RPC must provide some level of data format translation across systems.

c) Example: Sun RPC requires that clients convert their data to a neutral canonical format using the External Data Representation (XDR) APIs.
d) In contrast, DCE's Network Data Representation (NDR) service is multicanonical, meaning that it supports multiple data format representations.
e) The client chooses one of these formats, tags the data with chosen format, and then leaves it up to the server to transform the data into a format it understands.
f) In other words, the server makes it right. It lets the client to do translation, which makes the life easy for the server.
g) With Sun, all clients look the same to the server: The Client makes it right.

Q8. Peer – to – Peer Communications?

a) Most early client/server applications were implemented using low-level, conversational, peer-to-peer protocols – such as sockets, TLI, CPIC/APPC, NetBIOS and Named Pipes.
b) These low-level protocols are hard to code and maintain.
c) Instead now, the programmers are using RPCs, MOMs, and ORBs, which provide high level abstraction.
d) The term, "peer-to-peer" indicates that the two sides of a communication link use the same protocol interface to conduct a networked conversation.
e) The protocol is symmetrical, and it is sometimes called "program-to-program".
f) The peer-to-peer interface not fully mask the underlying network from the programmer.
g) Programmer have to handle the transmission timeouts, race conditions, and other network errors.
h) The peer-to-peer protocols started as stack-specific APIs.

1.Sockets

a) Sockets were introduced in 1981 as the UNIX BSD 4.2 generic interface that would provide Unix-to-Unix communications over network.
b) In 1985, SUN OS introduced NFS and RPC over sockets.
c) Sockets are supported on virtually every OS.
d) The windows socket API, known as WinSock, is a multivendor specification that standardizes the use of TCP/IP under windows.
e) In BSD Unix System, sockets are part of the Kernel and provide both a standalone and networked IPC service.

Socket = Net_ID . Host_ID . Port_ID = IP Address + Port Address.

The three most popular socket types are

- Stream
- Datagram
- Raw

Stream and datagram sockets interface to the TCP and UDP protocols, and raw sockets interface to the IP protocol. A port is an entry point to an application that resides on the host. It is represented by a 16-bit

integer. Ports are commonly used to define the entry points for services provided by the server applications.

2. TLI

a) In 1986, AT&T introduced the Transport Layer Interface that provides functionality similar to sockets but in a more network-independent fashion.
b) Sockets and TLI are very similar from a programmer's perspective.
c) TLI is just cleaner version of the sockets.
d) It should run on IPX/SPX (or) TCP/IP with very few modifications.
e) The TLI API consists of 25 API calls.
f) Later standardized as XTI, X/Open Transport Interface.

3) NetBIOS:

a) It is the premier protocol for LAN-based, program-to-program communications.
b) Introduced by IBM and Sytek in 1984 for the IBM PC network.
c) It is used as an interface to a variety of stacks – including NetBEUI, TCP/IP, XNS, Vines, OSI and IPX/SPX.
d) The NetBIOS services are provided through a set of commands, specified in a structure called the Network Control Block (NCB)
e) It does not support the routing of messages to other networks.

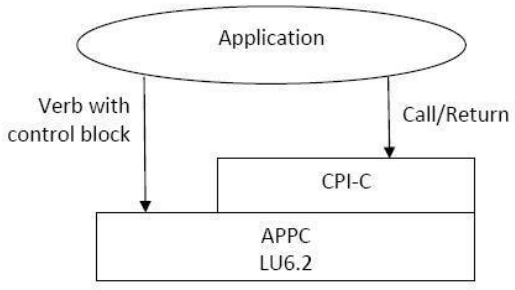
4) Named Pipes:

a) Provide highly reliable, two-way communications between clients and a server.
b) They provide a file-like programming API that abstracts a session-based two-way exchange of data.
c) Using named pipes, processes can exchange data as if they were writing to, or reading from, a sequential file.
d) These are suitable for implementing server programs that require many-to-one pipelines.
e) Important benefit of named pipes are part of the base interprocess communications service.
f) Named pipes interface is identical, whether the processes are running on an individual machine or distributed across the network.
g) Named pipes run on NetBIOS, IPX/SPX, and TCP/IP stacks.
h) Named pipes are built-in networking features in Windows NT, Windows for Workgroups, Windows 95 and Warp Server.
i) Unix support for Named Pipes is provided by LAN Manager/X.

5) CPI-C/APPC:

a) Common Programming Interface for Communications (CPI-C) build on top of APPC and marks its complexities and irregularities.
b) Writing to the CPI-C API allows you to port your programs to all SNA platforms.
c) The CPI-C API consists of about 40 calls; APPC consists of over 60 calls.
d) Most of these calls deals with configuration and services.
e) Advanced program-to-program communication is a protocol which computer programs can use to communicate over a network.

f) APPC was developed as a component of IBM's Systems Network Architecture (SNA).
g) APPC is linked with the term LU6.2.
h) LU6.2. (Logic Unit Version 6.2) is a device independent SNA Protocol.
i) It was developed to allow computers in IBM environments to setup their own communications sessions, rather than rely on a hos computer to do so.
j) Contrary to TCP/IP, in which both communication partners always possess a clear role, the communication partners in APPC are equal, i.e., everyone can be both servers and clients equally.
k) With the wide success of TCP/IP, APPC has declined.



Q8. Web Services?

Service-oriented architecture:

A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

Service-oriented architectures are not a new thing. The first service-oriented architecture for many people in the past was with the use DCOM or Object Request Brokers (ORBs) based on the CORBA specification.

In software engineering, a Service-Oriented Architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functionalities that are built as software components (discrete pieces of code and/or data structures) that can be reused for different purposes. SOA design principles are used during the phases of systems development and integration.

SOA also generally provides a way for consumers of services, such as web-based applications, to be aware of available SOA-based services. For example, several disparate departments within a company may develop and deploy SOA services in different implementation languages; their respective clients will benefit from a well-understood, well-defined interface to access them. XML is often used for interfacing with SOA services, though this is not required. JSON is also becoming increasingly common.

SOA defines how to integrate widely disparate applications for a Web-based environment and uses multiple implementation platforms. Rather than defining an API, SOA defines the interface in terms of protocols and functionality. An endpoint is the entry point for such a SOA implementation.

Service-orientation requires loose coupling of services with operating systems, and other technologies that underlie applications. SOA separates functions into distinct units, or services, which developers make

accessible over a network in order to allow users to combine and reuse them in the production of applications.

These services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services

SOA can be seen in a continuum, from older concepts of distributed computing and modular programming, through SOA, and on to current practices of mashups, SaaS, and cloud computing.

Q9. SOAP (Simple Object Access Protocol)?

SOAP (Simple Object Access Protocol) is a way for a program running in one kind of operating system (such as Windows 2000) to communicate with a progam in the same or another kind of an operating system (such as Linux) by using the World Wide Web's Hypertext Transfer Protocol (HTTP)and its Extensible Markup Language (XML) as the mechanisms for information exchange. Since Web protocols are installed and available for use by all major operating system platforms, HTTP and XML provide an already at-hand solution to the problem of how programs running under different operating systems in a network can communicate with each other. SOAP specifies exactly how to encode an HTTP header and an XML file so that a program in one computer can call a program in another computer and pass it information. It also specifies how the called program can return a response.

SOAP was developed by Microsoft, DevelopMentor, and Userland Software and has been proposed as a standard interface to the Internet Engineering Task Force (IETF). It is somewhat similar to the Internet Inter-ORB Protocol (IIOP), a protocol that is part of the Common Object Request Broker Architecture (CORBA). Sun Microsystems' Remote Method Invocation (RMI) is a similar client/server interprogram protocol between programs written in Java.

An advantage of SOAP is that program calls are much more likely to get through firewall servers that screen out requests other than those for known applications (through the designated port mechanism). Since HTTP requests are usually allowed through firewalls, programs using SOAP to communicate can be sure that they can communicate with programs anywhere.

Q10. What is WSDL?

- WSDL stands for Web Services Description Language
- WSDL is written in XML
- WSDL is an XML document
- WSDL is used to describe Web services
- WSDL is also used to locate Web services
- WSDL is a W3C recommendation
- WSDL stands for Web Services Description Language.
- WSDL is a document written in XML. The document describes a Web service. It specifies the location of the service and the operations (or methods) the service exposes.

Hence, a WSDL document uses the following elements in the definition of network services:

1. Types– a container for data type definitions using some type system (such as XSD).
2. Message– an abstract, typed definition of the data being communicated.

3. Operation– an abstract description of an action supported by the service.
4. Port Type–an abstract set of operations supported by one or more endpoints.
5. Binding– a concrete protocol and data format specification for a particular port type.
6. Port– a single endpoint defined as a combination of a binding and a network address.
7. Service– a collection of related endpoints.

Q11. Rest Services?

The RESTful Web services are completely stateless. This can be tested by restarting the server and checking if the interactions are able to survive.

Restful services provide a good caching infrastructure over HTTP GET method (for most servers). This can improve the performance, if the data the Web service returns is not altered frequently and not dynamic in nature.

The service producer and service consumer need to have a common understanding of the context as well as the content being passed along as there is no standard set of rules to describe the REST Web services interface.

REST is particularly useful for restricted-profile devices such as mobile and PDAs for which the overhead of additional parameters like headers and other SOAP elements are less.

REST services are easy to integrate with the existing websites and are exposed with XML so the HTML pages can consume the same with ease. There is hardly any need to refactor the existing website architecture. This makes developers more productive and comfortable as they will not have to rewrite everything from scratch and just need to add on the existing functionality.

REST-based implementation is simple compared to SOAP.

UNIT-II

SERVLETS: Servlet Lifecycle, sending HTML information, Session tracking, JDBC API, Applications.

STRUTS: An introduction to Struts Framework, Basic components of struts, Model Layer, view Layer, Controller Layer, and Validator.

Concepts:

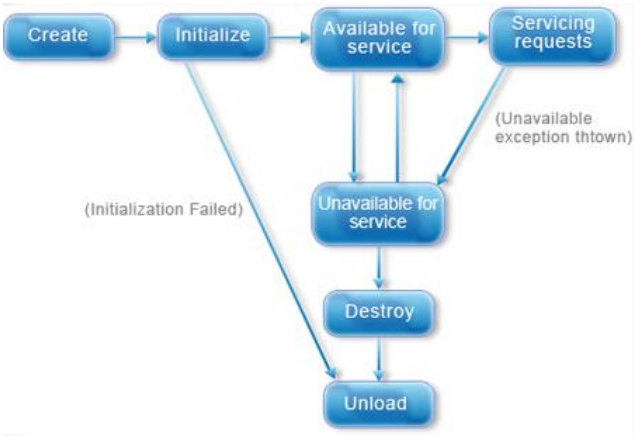
Q1.Servlet Life Cycle?

The lifecycle of a Servlet involves creating an instance of the Servlet, associating the Servlet with initialization information, dispatching request to the servlet instance and finally destroying the servlet instance. The lifecycle of a servlet begins when it is loaded into Application Server memory and ends when the servlet is terminated or reloaded.

Basically There are four stages in Servlet life Cycle

1. Loading and Instantiation
2. Initialization
3. Servicing the Request
4. Destroying the Servlet

Process Diagram of The Servlet Life-cycle



Loading a Servlet: The first stage of the Servlet life-cycle involves loading and initializing the servlet by container. When ever the web container is started, it loads and initializes the Servlet. the web container may delay this untill the web container determines which servlet is needed to service a request. The Servlet Container loads the servlet during startup or when the first request made. The Loading of servlet depends on <load-on-startup> attribute in web.xml file. If the attribute <load-on-startup> has the positive values then the servlet is load with loading of the container otherwise it will load when the first request comes for service. After loading the servlet, the container creates the instance of the servlet.

Servlet Initializing: After creating the instances, the servlet container calls the init() method and pass the servlet initialization parameters to the init() method. The init() method must be called by the servlet container before the servlet can service any request. The initialization parameters persist until the servlet is destroyed. The init() method is called only once throughout the life cycle of the servlet. The Servlet will be available for service if its loaded successfully otherwise the servlet container unloads the servletServlet's can be dynamically loaded and instantiated when their services are first requested, the Web server can be configured so that specific servlets are loaded and instantiated when the Web server initializes. In either case, the init method of the servlet performs any necessary servlet initialization, and is guaranteed to be called once for each servlet instance, before any requests to the servlet are handled. An example of a task which may be performed in the init method is the loading of default data parameters or database connections.

The most common form of the init method of the servlet accepts a ServletConfig object parameter. This interface object allows the servlet to access name/value pairs of initialization parameters that are specific to that servlet. The ServletConfig object also gives us access to the SevletContext object that describes information about our servlet environment. Each of these objects will be discussed in more detail in the servlet examples sections.

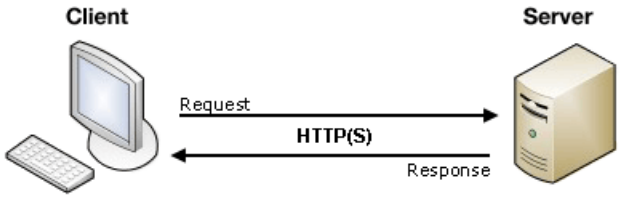
Request Handling: After completing the initialization process, the servlet will be available for service. Servlet creates separate thread for each request. The servlet container calls the service() method for servicing any request. The service() method determines the kind of request and calls the appropriate method (doGet() or doPost() for handling the request and sends response to the client using the method of the response object. Application Server - Express receives a client request. The servlet engine creates a request object and a response object. The servlet engine invokes the servlet service() method, passing the request and response objects.The service() method gets information about the request from the request object, processes the request, and uses methods of the response object to create the client response. The service method can invoke other methods to process the request, such as doGet(), doPost(), or methods you write.

Destroying the Servlet: If the Servlet is no longer needed for servicing any request, the servlet container calls the destroy() method. like init() method it is also called only once throughout the life cycle of the servlet. The servlet engine stops a servlet by invoking the servlet's destroy() method. Typically, a servlet's destroy() method is invoked when the servlet engine is stopping a Web application which contains the servlet. The destroy() method runs only one time during the lifetime of the servlet and signals the end of the servlet.After a servlet's destroy() method is invoked, the servlet engine unloads the servlet, and the Java virtual machine eventually performs garbage collection on the memory resources associated with the servlet.

Q2. Sending HTML Information?

About client/server architecture

The web is based on a very basic client/server architecture that can be summarized as follows: a client (usually a Web browser) sends a request to a server (most of the time a web server like Apache, Nginx, IIS, Tomcat, etc.), using the HTTP protocol. The server answers the request using the same protocol.



On the client side, an HTML form is nothing more than a convenient user-friendly way to configure an HTTP request to send data to a server. It makes it possible for the user to provide information to be delivered in the HTTP request.

Q3. Session Tracking?

There are a number of problems that arise from the fact that HTTP is a "stateless" protocol. In particular, when you are doing on-line shopping, it is a real annoyance that the Web server can't easily remember previous transactions. This makes applications like shopping carts very problematic: when you add an entry to your cart, how does the server know what's already in your cart? Even if servers did retain contextual information, you'd still have problems with e-commerce. When you move from the page where you specify what you want to buy (hosted on the regular Web server) to the page that takes your credit card number and shipping address (hosted on the secure server that uses SSL), how does the server remember what you were buying?

There are three typical solutions to this problem.

1.Cookies. You can use HTTP cookies to store information about a shopping session, and each subsequent connection can look up the current session and then extract information about that session from some location on the server machine. This is an excellent alternative, and is the most widely used approach. However, even though servlets have a high-level and easy-to-use interface to cookies, there are still a number of relatively tedious details that need to be handled:

- Extracting the cookie that stores the session identifier from the other cookies (there may be many, after all),
- Setting an appropriate expiration time for the cookie (sessions interrupted by 24 hours probably should be reset), and
- Associating information on the server with the session identifier (there may be far too much information to actually store it in the cookie, plus sensitive data like credit card numbers should never go in cookies).

2.URL Rewriting. You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session. This is also an excellent solution, and even has the advantage that it works with browsers that don't support cookies or where the user has disabled cookies. However, it has most of the same problems as cookies, namely that the server-side program has a lot of straightforward but tedious processing to do. In addition, you have to be very careful that every URL returned to the user (even via indirect means like Location fields in server

redirects) has the extra information appended. And, if the user leaves the session and comes back via a bookmark or link, the session information can be lost.

3.Hidden form fields. HTML forms have an entry that looks like the following: <INPUT TYPE="HIDDEN" NAME="session" VALUE="...">. This means that, when the form is submitted, the specified name and value are included in the GET or POST data. This can be used to store information about the session. However, it has the major disadvantage that it only works if every page is dynamically generated, since the whole point is that each session has a unique identifier.

Deleting Session Data:

When you are done with a user's session data, you have several options:

Remove a particular attribute: You can call public void removeAttribute(String name) method to delete the value associated with a particular key.

Delete the whole session: You can call public void invalidate() method to discard an entire session.

Setting Session timeout: You can call public void setMaxInactiveInterval(int interval) method to set the timeout for a session individually.

Log the user out: The servers that support servlets 2.4, you can call logout to log the client out of the Web server and invalidate all sessions belonging to all the users.

web.xml Configuration: If you are using Tomcat, apart from the above mentioned methods, you can configure session time out in web.xml file as follows.

```
<session-config>
 <session-timeout>15</session-timeout>
 </session-config>
```

The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.

The getMaxInactiveInterval( ) method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, getMaxInactiveInterval( ) returns 900.

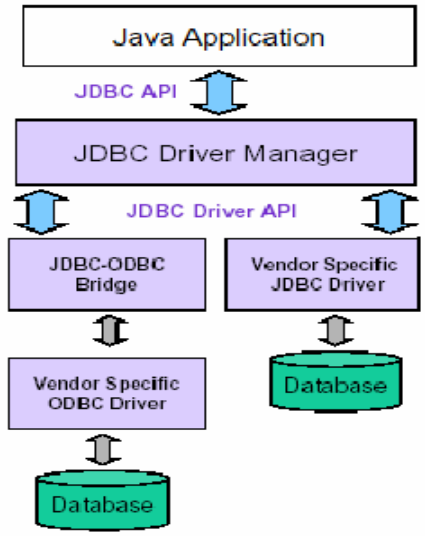
Q4. What is JDBC API?

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks commonly associated with database usage:

1. Making a connection to a database
2. Creating SQL or MySQL statements
3. Executing that SQL or MySQL queries in the database

4. Viewing & Modifying the resulting record.



General Architecture

Make sure you have done following setup:

1. Core JAVA Installation
2. SQL or MySQL Database Installation

Apart from the above you need to setup a database which you would use for your project. Assuming this is EMP and you have created on table Employees within the same database.

There are following steps required to create a new Database using JDBC application:

Import the packages . Requires that you include the packages containing the JDBC classes needed for database programming. Most often, using import java.sql.* will suffice.

Register the JDBC driver . Requires that you initialize a driver so you can open a communications channel with the database.

Open a connection . Requires using the DriverManager.getConnection() method to create a Connection object, which represents a physical connection with datbase server.

To create a new database, you need not to give any database name while preparing database URL as mentioned in the below example.

Execute a query . Requires using an object of type Statement for building and submitting an SQL statement to the database.

Clean up the environment . Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

When you run JDBC Program, it produces following result:

C:\>java JDBC Program

Connecting to database...

Creating database...

Database created successfully...

Goodbye!

C:\>

Q5. Struts Framework?

The Struts Framework is a standard for developing well-architected Web applications. It has the following features:

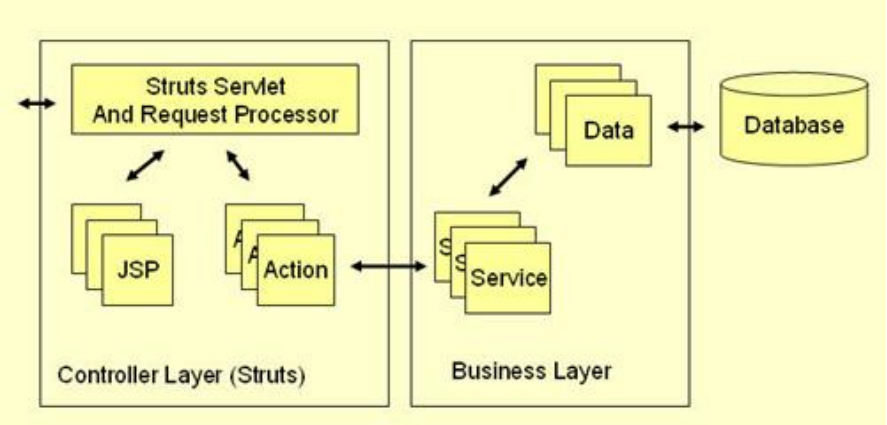
1. Open source
2. Based on the Model-View-Controller (MVC) design paradigm, distinctly separating all three levels:

- Model: application state
- View: presentation of data (JSP, HTML)
- Controller: routing of the application flow

3. Implements the JSP Model 2 Architecture

4. Stores application routing information and request mapping in a single core file, struts-config.xml

The Struts Framework, itself, only fills in the View and Controller layers. The Model layer is left to the developer.



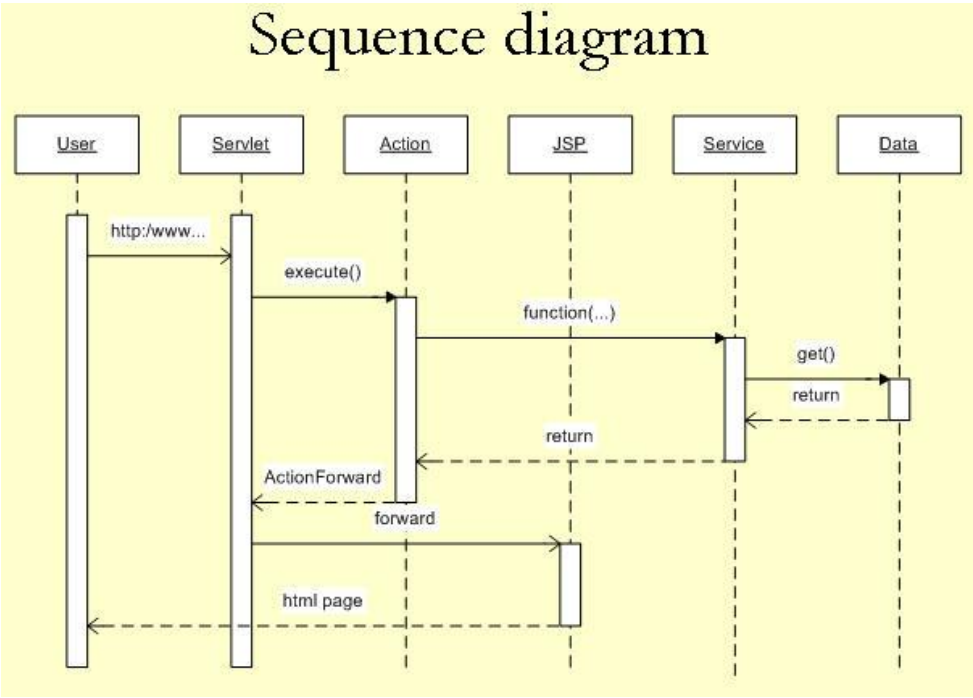
Architecture Overview

All incoming requests are intercepted by the Struts servlet controller. The Struts Configuration file struts-config.xml is used by the controller to determine the routing of the flow. This flows consists of an alternation between two transitions:

From View to Action -A user clicks on a link or submits a form on an HTML or JSP page. The controller receives the request, looks up the mapping for this request, and forwards it to an action. The action in turn calls a Model layer (Business layer) service or function.

From Action to View -After the call to an underlying function or service returns to the action class, the action forwards to a resource in the View layer and a page is displayed in a web browser.

The diagram below describes the flow in more detail:



- User clicks on a link in an HTML page.
- Servlet controller receives the request, looks up mapping information in struts-config.xml, and routes to an action.
- Action makes a call to a Model layer service.
- Service makes a call to the Data layer (database) and the requested data is returned.
- Service returns to the action.
- Action forwards to a View resource (JSP page)
- Servlet looks up the mapping for the requested resource and forwards to the appropriate JSP page.
- JSP file is invoked and sent to the browser as HTML.
- User is presented with a new HTML page in a web browser.

Struts Components:

1. The Controller

This receives all incoming requests. Its primary function is the mapping of a request URI to an action class selecting the proper application module. It's provided by the framework.

2. The struts-config.xml File

This file contains all of the routing and configuration information for the Struts application. This XML file needs to be in the WEB-INF directory of the application.

3. Action Classes

It's the developer's responsibility to create these classes. They act as bridges between user-invoked URIs and business services. Actions process a request and return an ActionForward object that identifies the next component to invoke. They're part of the Controller layer, not the Model layer.

4. View Resources

View resources consist of Java Server Pages, HTML pages, JavaScript and Stylesheet files, Resource bundles, JavaBeans, and Struts JSP tags.

5. ActionForms

These greatly simplify user form validation by capturing user data from the HTTP request. They act as a "firewall" between forms (Web pages) and the application (actions). These components allow the validation of user input before proceeding to an Action. If the input is invalid, a page with an error can be displayed.

6. Model Components

The Struts Framework has no built-in support for the Model layer. Struts supports any model components:

- JavaBeans
- EJB
- CORBA
- JDO
- any other

Q6. Basic Components of Struts?

The Struts framework is a rich collection of Java libraries and can be broken down into the following major pieces:

- Base framework
- JSP tag libraries
- Tiles plugin
- Validator plugin

A brief description of each follows.

Base Framework

The base framework provides the core MVC functionality and is comprised of the building blocks for your application. At the foundation of the base framework is the Controller servlet: ActionServlet. The rest of the base framework is comprised of base classes that your application will extend and several utility classes. Most prominent among the base classes are the Action and ActionForm classes. These two classes are used extensively in all Struts applications. Action classes are used by ActionServlet to process specific requests. ActionForm classes are used to capture data from HTML forms and to be a conduit of data back to the View layer for page generation.

JSP Tag Libraries

Struts comes packaged with several JSP tag libraries for assisting with programming the View logic in JSPs. JSP tag libraries enable JSP authors to use HTML-like tags to represent functionality that is defined by a Java class.

Following is a listing of the libraries and their purpose:

- **HTML** Used to generate HTML forms that interact with the Struts APIs.
- **Bean** Used to work with Java bean objects in JSPs, such as accessing bean values.
- **Logic** Used to cleanly implement simple conditional logic in JSPs.
- **Nested** Used to allow arbitrary levels of nesting of the HTML, Bean, and Logic tags that otherwise do not work.

Tiles Plugin

Struts comes packaged with the Tiles subframework. Tiles is a rich JSP templating framework that facilitates the reuse of presentation (HTML) code. With Tiles, JSP pages can be broken up into individual 'tiles' or pieces and then glued together to create one cohesive page. Similar to the design principles that the core Struts framework is built on, Tiles provides excellent reuse of View code. As of Struts 1.1, Tiles is part of and packaged with the core Struts download. Prior to Struts 1.1, Tiles was a third-party add-on, but has since been contributed to the project and is now more tightly integrated.

Validator Plugin

Struts comes packaged, as of version 1.1, with the Validator subframework for performing data validation. Validator provides a rich framework for performing data validation on both the server side and client side (browser). Each validation is configured in an outside XML file so that validations can easily be added to and removed from an application declaratively versus being hard-coded into the application. Similar to Tiles, prior to Struts 1.1, Validator was a third-party add-on, but has since been included in the project and is more tightly integrated.

Q7. Model layer?

The Model layer represents the part of your application that implements the business logic. It is responsible for retrieving data and converting it into meaningful concepts for your application. This includes processing, validating, associating or other tasks related to handling data.

At a first glance, Model objects can be looked at as the first layer of interaction with any database you might be using for your application. But in general they stand for the major concepts around which you implement your application.

In the case of a social network, the Model layer would take care of tasks such as saving the user data, saving friends associations, storing and retrieving user photos, finding new friends for suggestions, etc. While the model objects can be thought as “Friend”, “User”, “Comment”, or “Photo”.

Q8. The View layer?

The View renders a presentation of modeled data. Being separated from the Model objects, it is responsible for using the information it has available to produce any presentational interface your application might need.

For example, as the Model layer returns a set of data, the view would use it to render a HTML page containing it. Or a XML formatted result for others to consume.

The View layer is not only limited to HTML or text representation of the data, it can be used to deliver a wide variety of formats depending on your needs, such as videos, music, documents and any other format you can think of.

Q9. The Controller layer?

The Controller layer handles requests from users. It’s responsible for rendering back a response with the aid of both the Model and the View Layer.

Controllers can be seen as managers taking care that all needed resources for completing a task are delegated to the correct workers. It waits for petitions from clients, checks their validity according to authentication or authorization rules, delegates data fetching or processing to the model, and selects the correct type of presentational data that the client is accepting, to finally delegate this rendering process to the View layer.

Q10.Validator Layer?

The Struts validator framework provides many generic validation methods to make the validation work more easily and maintainability. With Struts validator, you need to declared the validation function into a xml file instead of the ActionForm validate() method, it can make the Struts validation more standardization, reusable and less duplicated codes.

Example:

The best way to understand about the Struts validator framework is create a simple application and walk through the validation works. Here's a simple user registration form to use the Struts validator framework to check the username, password and email.

Table 1. Basic Validators provided by the framework.

Name	Description
`byte,short,integer, long,float,double`	Checks if the value can safely be converted to the corresponding primitive.
`creditCard`	Checks if the field is a valid credit card number.
`date`	Checks if the field is a valid date.
`email`	Checks if the field is a valid email address.
`mask`	Succeeds if the field matches the corresponding regular expression mask.
`maxLength`	Checks if the value's length is less than or equal to the given maximum length.
`minLength`	Checks if the value's length is greater than or equal to the given minimum length.
`range`	Checks if the value is within a minimum and maximum range.
`required`	Checks if the field isn't null and length of the field is greater than zero, not including whitespace.

EJB ARCHITECTURE: EJB –EJB Architecture – Overview of EJB software architecture – View of EJB – Conversation – Building and Deploying EJBs – Roles in EJB.

EJB APPLICATIONS: EJB Session Beans – EJB entity beans – EJB clients – EJB Deployment – Building an application with EJB.

Concepts:

Q1. EJB –EJB Architecture?

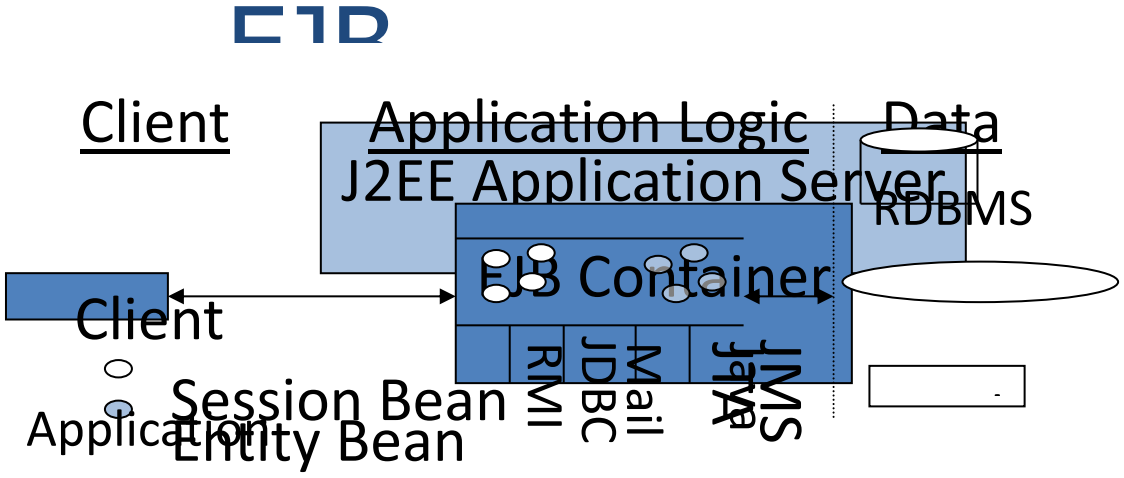
Enterprise beans-An enterprise bean is a non-visual component of a distributed, transaction-oriented enterprise application. Enterprise beans are typically deployed in EJB containers and run on EJB servers.

There are three types of enterprise beans: session beans, entity beans, and message-driven beans.

Session beans: Session beans are non-persistent enterprise beans. They can be stateful or stateless. A stateful session bean acts on behalf of a single client and maintains client-specific session information (called conversational state) across multiple method calls and transactions. It exists for the duration of a single client/server session. A stateless session bean, by comparison, does not maintain any conversational state. Stateless session beans are pooled by their container to handle multiple requests from multiple clients.

Entity beans: Entity beans are enterprise beans that contain persistent data and that can be saved in various persistent data stores. Each entity bean carries its own identity. Entity beans that manage their own persistence are called bean-managed persistence (BMP) entity beans. Entity beans that delegate their persistence to their EJB container are called container-managed persistence (CMP) entity beans.

Message-driven beans: Message-driven beans are enterprise beans that receive and process JMS messages. Unlike session or entity beans, message-driven beans have no interfaces. They can be accessed only through messaging and they do not maintain any conversational state. Message-driven beans allow asynchronous communication between the queue and the listener, and provide separation between message processing and business logic.



EJB Logical Architecture

A EJB system is logically a three-tier system.The three tiers are as follows:

a) The Client

b) The EJB Server

c) The Database (or other persistent store)

This is a logical architecture because these tiers don't necessarily have to reside on three different machines.

For example,

a) The EJB server and the database might reside on the same machine, when the EJB server includes built-in functionalities for persistent storage.

b) The client and the EJB server might reside on the same machine, when an EJB bean makes a call to another bean in the same container. The caller bean is acting as a client here.

c) These two cases can be combined to have all the tiers in a single machine.

EJB's role in each of these tiers are:

a) A program on the client side makes call to remote EJBs. The client needs to know how to find the EJB server and how to interact with the objects that reside on the EJB server.

b) The EJB components live in the middle tier. The EJB objects reside inside an EJB container, which in turn resides in an EJB server.

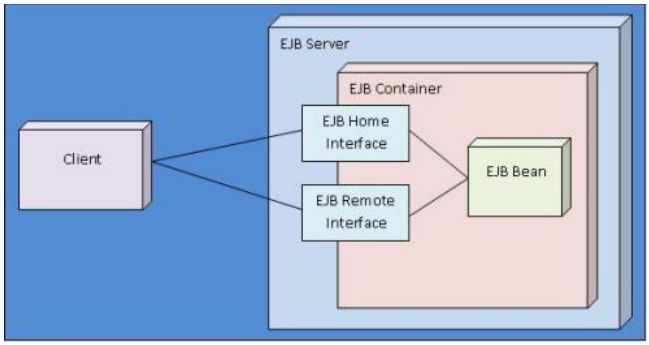
c) EJB can access the database themselves, typically via Java Database connecting (JDBC), or they can allow the container to handle their data storage needs for them.

EJB Advantages:

- Persistent- Beans need to load and store data.You can let the server do it.
- Transactional - Support for distributed transactions as Server manages all the transactions.
- Secure-SSL/RMI protects transmitted data.
- Multithreaded-Programmer delegates(Allocate a task to a person) all responsibility for multithreading to server.

Q2. Overview of EJB's Software Architecture?

The diagram presents a high-level view of the EJB architecture, showing the various relationships between the various components.



This illustrates the Key Features of the EJB Architecture

- EJB bean exists within the container
- Client never communicates directly with the EJB bean; rather it talks to the bean through its home interface and its remote interface, both of which are provided by the container.
- EJB server does not take part in the logical conversation between client and EJB bean. Actually, the server is doing a lot of work behind the scenes. Remembers the must handle requests for the container, and feed the container incoming client requests. And the bean and the client not directly access the server – all access is performed against the container.

The EJB architecture simplifies application development by:

- providing pre-integrated solution framework
- separating the business logic from distributed system services
- providing standard service interfaces, including naming, transactions, and security
- managing life cycle functions.

Q3. Views in EJB?

Remote client view

When your EJB and its clients will be in a distributed environment - meaning EJBs and clients will reside on separate Java virtual machines. Example : EJBs hosted on a WebSphere Application Server and Servlets that consume EJB APIs hosted on a Tomcat server.

Local client view

Only when it is guaranteed that other enterprise beans or clients will only address the bean within a single JVM. Example, EJBs as well as the Servlets deployed on the same WebSphere server.

Q4. Building and Deploying EJBs?

EJB must provide three required classes:

- EJB implementation Class
- Home Interface
- Remote Interface

Each EJB must provided with a deployment descriptor

The deployment descriptor is serialized instance of a Java class that contains information about how to deploy the bean.

Two flavors of deployment descriptors:

- Session Descriptors – apply to Session EJBs
- Entity Descriptors – apply to Entity EJBs.

A Deployment Descriptor contains information such as the following:

- The name of the EJB Class
- The name of the EJB Home Interface
- The name of the EJB Remote Interface
- ACLs of entities authorized to use each class or method.
- For Entity Beans, a list of container – managed fields.
- For Session Beans, a value denoting whether the bean is stateful or stateless.

Properties File may be generated, which contains environment properties that will be activated during runtime.

Manifest File is needed to create the ejb-jar file, which is the medium used to distribute EJBs.

Name : <filename>

Enterprise-Bean:True

Deploying the EJBs:

- At the deployment time, the EJB container must read the ejb-jar file, create implementations for the home and remote interfaces.
- Reads the deployment-descriptor and ensures that the bean has what it wants and ad the bean's property settings to the environment so that they're available to the bean at runtime.

Connecting to the EJB

- The client can use either RMI or CORBA to connect to the EJB
- The client looks up the EJB's home interface using a naming service (JNDI or COS)
- Invokes a find() or create() method on the home interface to get a reference to an EJB object.
- And then uses the EJB object as if it were an ordinary object

Q5. Roles in EJB?

EJB specification gives a clear description of each of these roles and their associated responsibilities.

a) Enterprise Bean Provider

- It is the person or group that writes and packages EJBs.
- It might be a third-party vendor of EJB components, or it might be an information systems programmer who writes EJBs to implement their company's business logic.
- Because the container must provide transaction and network communication support, the enterprise bean provider does not need to be an expert at low-level system programming, networking or transaction processing.
- The end product is an ejb-jar file, which contains the Enterprise bean, its supporting classes, and information that describes how to deploy the bean.

b) Deployer

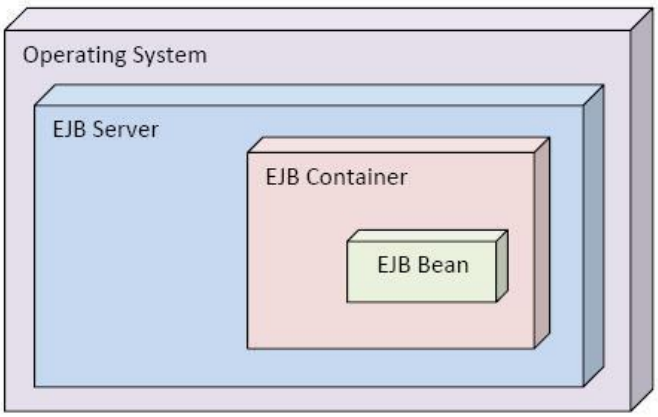
- The deployer takes an ejb-jar file and installs it into an EJB container.
- The deployer's task begins with the receipt of an ejb-jar file, and ends with the installation of the ejb-jar file in the container.
- This task is typically handled by the System Administrator in MIS world.

c) Application Assembler

- An application assembler builds applications using EJBs classes.
- An MIS programmer may purchase a prepackaged set of EJBs that implement an accounting system.
- Then the programmer might use tools to customize the beans.
- The application assembler might also need to provide a user-interface client program.
- This role is roughly analogous to that of a Systems Integrator in the MIS world.

d) EJB Server Provider

- An EJB Server Provider provides a server that can contain EJB containers.
- The EJB 1.0 specification does not describe the interface between the server and the container, so the EJB server provider and the EJB container provider will likely be one and the same for any given product.
- The diagram shows the relationships among the EJB server, the EJB container, and the EJB bean.
- An EJB server provides services to an EJB container and an EJB container provides services to an Enterprise Java Bean.
- The role of the EJB server provider is similar to that of a database system vendor.



e) EJB Container Provider

- An EJB container is the world in which an EJB bean lives.
- The container services requests from the EJB, forwards requests from the client to the EJB, and interacts with the EJB server.
- The container provider must provide transaction support, security and persistence to beans.
- It is also responsible for running the beans and for ensuring that the beans are protected from the outside world.

f) System Administrator

- System Administrators are responsible for the day-to-day operation of the EJB server.
- Their responsibilities include keeping security information up-to-date and monitoring the performance of the server.

Q6. EJB Session Beans?

- These are generally tied to the lifetime of a given client session.
- They are relatively short-lived.
- The stateful session objects are created in response to a single client's request, communicate exclusively with a single client and die when the client no longer needs them.
- A session bean is required to implement the javax.ejb.SessionBean interface.

```
public interface javax.ejb.SessionBean extends javax.ejb.EnterpriseBean
{
 public abstract void ejbActivate();
 public abstract void ejbPassivate();
 public abstract void ejbRemove();
 public abstract void setSessionContext(SessionContext ctx);
}
```

- Al of the methods declared in this interface are callback methods; that is they are invoked by the container to notify the Session EJB that some event has occurred.

ejbPassivate()

- Container calls this method to swap the EJB class out to "semi-persistent" storage.
- The storage is "semi-persistent" because Session EJBs do not carry the same sort of guarantee of long life as Entity EJBs do.
- If the EJB container crashes or restarts, the session beans that existed before the crash or restart are destroyed.
- Session bean also have an associated timeout value; after the timeout expires, they are destroyed.
- If none of these events happens the session bean can be sapped back in at some later time.

ejbActivate()

- The ejbActivate() method is essentially the inverse of the ejbPassivate() call.
- The container calls this method after a session bean's state has been restored from "semi-persistent" storage, but before it becomes available to the outside world.
- A bean can use the ejbActivate() call to restore any resources that is closed before being passivated.

ejbRemove()

- This callback method in invoked by the container as a result of a client calling the remove() method of a bean's home or remote interface.
- The client's invocation of the remove() method notifies the container that this particular session bean instance should be destroyed.
- setSessionContext(SessionContext ctx)
- This method is called at the very beginning of a session bean's life.
- The container passes the session bean an object that implements the SessionContext interface.
- The bean stores this object in an object variable, and can then use the SessionContext to interact with container-provided services such as security and transaction management.

Q7. Stateful and Stateless Session Beans?

A stateless session bean is one that doesn't keep track of any information from one method call to another – essentially, a bean with no instance variables. (Ex: CalculateInterest Bean)

Any two instances of a stateless bean are equivalent.

A container can easily handle a stateless Session bean, because instances of a stateless bean can be created and destroyed at will, without worrying about their state at the time of their destruction.

A stateful session bean, is a bean that changes state during a conversation – that is, a bean that has instance variables. (Ex: Shopping Cart Bean)

The life cycle of a stateful bean is more complex for the container than the stateless bean.

Because, the container have to maintain the state of the stateful session bean during swap out and swap in process.

According to the EJB specification, EJB Objects are not allowed to have static variables.

Q8. EJB Containers?

- It is an environment in which EJB execute.
- Its primary role is to serve as a buffer between an EJB and the outside world
- The container is having high degree of flexibility in servicing client requests.

The container provides following services:

- Support for transactions
- Support for management of multiple instances
- Support for persistence
- Support for security

The container generally provides support for managing and monitoring EJBs as well as for version management.

a) Support for transactions

EJB container provides the beans with ACID properties. EJB supports transaction by having the transaction manager for 'begin' and 'commit' operations. It also provides another approach, called declarative transaction management.

Here the author can specify the type of transaction support for that bean.

When the bean is deployed, the container reads the deployment descriptor associated with that particular EJB Bean, and provides necessary transaction support automatically.

EJB Provides six modes of transaction management

1) TX_NOT_SUPPORTED

2) TX_BEAN_MANAGED

3) TX_REQUIRED

4) TX_SUPPORTS

5) TX_REQUIRES_NEW

6) TX_MANDATORY

b) Support for Management of Multiple Instances

Normally stateless session beans are accessed by only one client.Other types of beans may be used concurrently by multiple clients .The EJB container must ensure that each client's requests are serviced in a timely manner.

To accomplish this goal, the server performs several tasks behind the screen.

i) Instance Passivation

ii) Instance Pooling

iii) Database connection pooling

iv) Preached Instances

v) Optimized method invocation

i) Instance Passivation

- It is the temporary swapping of an EJB bean out to storage.If a container needs resources, it may choose to temporarily swapout a bean.
- Both session & entity beans can be passivated.
- Passivation on session beans are performed only to free up space. But on the entity beans, also used to synchronize its state with the underlying data store.

ii) Instance Pooling

- This means that the container shares instances of EJB beans among multiple clients.To increase the efficiency the container instantiate fewer copies of a given bean to service requests. This not possible in all the cases.
- Specially, the container must provide one session bean per client otherwise conversational state can't be tracked by the containers.

iii) Database Connection Pooling

- It is a strategy commonly employed in middleware applications.
- When a component wishes to access a database, it does not create a new database connection; instead, it simply grabs a connection from a pool of available connections.
- This is beneficial for two reasons
- It avoids the overhead of establishing a new database connection
- Maintaining an open database connection consumes resources on the database. Connection pooling allows to limit the no. of simultaneous connections and reduces the overhead on performance.

iv) Precached instances

- This may be used when an EJB needs to load some state information from a database when it is created.
- EJB State information can be cached in some readily available place and this speeds up the loading process.

v) Optimized method invocations

- These are employed to combat overhead costs associated with remote invocation
- Whether the client of an EJB is a remote client or another EJB in the same container, the APIs used to gain access to it are always the same.
- A typical EJB interaction is as follows:
- Use a naming service to locate the EJB's home interface
- Make a call on the EJB's home interface to gain access to its remote interface.
- Make calls to the business methods against the remote interface.

Contain short circuit the remote protocol, when both the beans live in the same container.

Q9. EJB Servers?

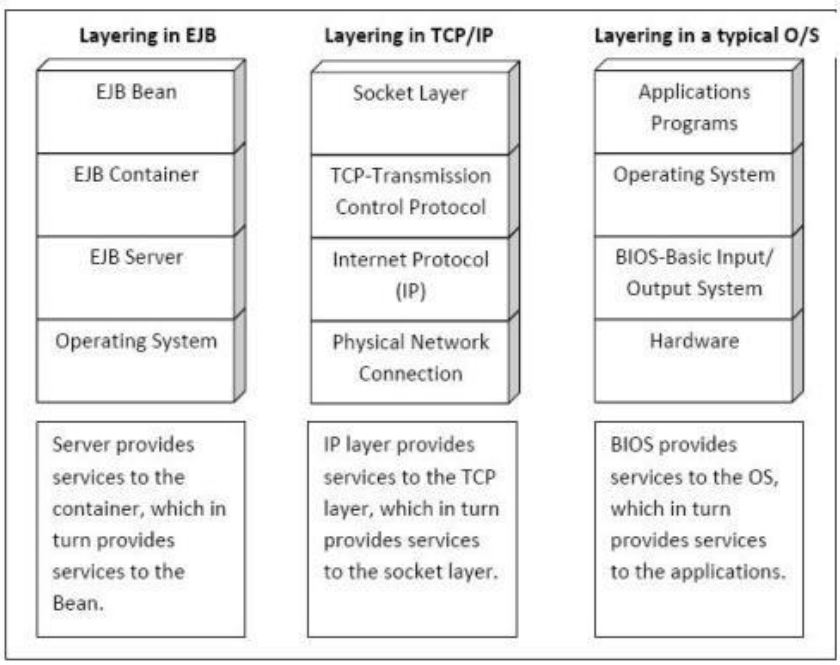
It is a “Container Container” – it contains the EJB container

It is responsible for providing the container with lower-level services such as network connectivity.

The EJB Specification 1.0 did not outline the exact interface between the EJB container and the EJB server. So, interface is currently left up to the individual vendor.

At an abstract level, though, the relationship between the container and server is such that the server will provide the container with low-level implementation of the services required by the container.

The layered architecture is shown below:



Q10. Building an EJB Application?

NetBeans is a Java Integrated Development Environment, IDE, which enables fast application development with the most adopted frameworks, technologies, and servers. Different than other IDEs, NetBeans comes already pre-packaged with a wide range of functionality out of the box, such as support for different frameworks, servers, databases, and mobile development.

Steps:

- Creating an EJB project
- Adding JPA support
- Creating Stateless Session Bean
- Creating Stateful Session Bean
- Sharing a service through Web Service
- Creating a Web Service client

UNIT-IV

CORBA: CORBA – Distributed Systems – Purpose – Exploring CORBA alternatives – Architecture overview – CORBA and networking model – CORBA object model – IDL – ORB – Building an application with CORBA.

Concepts:

Definition:

- CORBA is a distributed object-based systems.
- Provides inter-operability
- CORBA is a middle ware neither 2-tier or 3-tier architecture.
- CORBA is a technology to communicate 2 objects which are of heterogeneous type.
- CORBA is based on complete object-approach in which a client sends a massage to a remote object.
- CORBA can be written in c, c++, COBOL and JAVA
- CORBA was created by OMG(Object management group).
- OMG was created in 1989 initially with 8 companies.
- OMG provide the specification for object communication.
- OMG does not provide any s/w products.
- OMG focus on creating specification that can be implemented and used by all.
- The first version of CORBA was released in July ,1992 as
- "OBJECT MANAGEMENT ARCITECTURE GUIDE"
- Concept of CORBA came from OLE.

CORBA's technical background is based on 3 important concepts:

- Object –oriented Model
- Open distributed computing environment
- Component integration & reuse.

SOCKET PROGRAMMING:

- A socket is peer-to-peer communication end point.
- It has a name & network address.

- There are 3 types sockets.
    1. Datagram(UDP)
    2. Stream(TCP)
    3. Raw stream(ICMP)

Q1. Exploring CORBA Alternatives?

1. Socket Programming

Socket is a channel through applications can connect with each other and communicate

The most straight forward way to communicate between application components

The API for socket programming is rather low-level.

However, because the API is low-level, socket programming is not well suited to handling complex data types, especially when application components reside on different types machines (or) are implemented in different programming languages.

2. Remote Procedure Call (RPC):

Provides a function-oriented interface to socket-level communications

Using RPC, rather than directly manipulating the data that flows to and from a socket, the developer defines a function and generates codes that makes that function look like a normal functions to the caller.

Because RPC provides a function-oriented interface, it is often much easier to use than raw socket programming.

3. DCE

Set of Standards by the OSF, includes a standard for RPC.

It was never gained wide acceptance and exists today as little more than an historical curiosity.

4. DCOM

It is a relatively robust object model, that enjoys particular good support on Microsoft O/S.

However, being a Microsoft technology, the availability of DOM is sparse outside the realm of Windows O/S.

CORBA-DCOM bridges enable CORBA objects to communicate with DCOM objects vice versa.

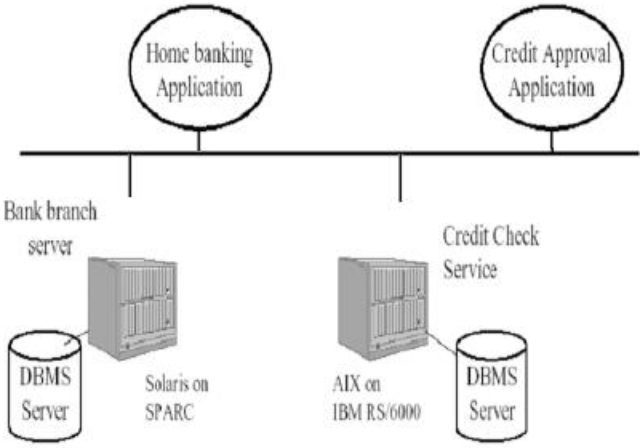
5. RMI

Advantage – it supports the passing of objects by value, a feature not (currently) supported by CORBA.

Disadvantage – it is a java-only solution, that is RMI servers and clients must be written in JAVA.

Q2. CORBA Ditributed Systems?

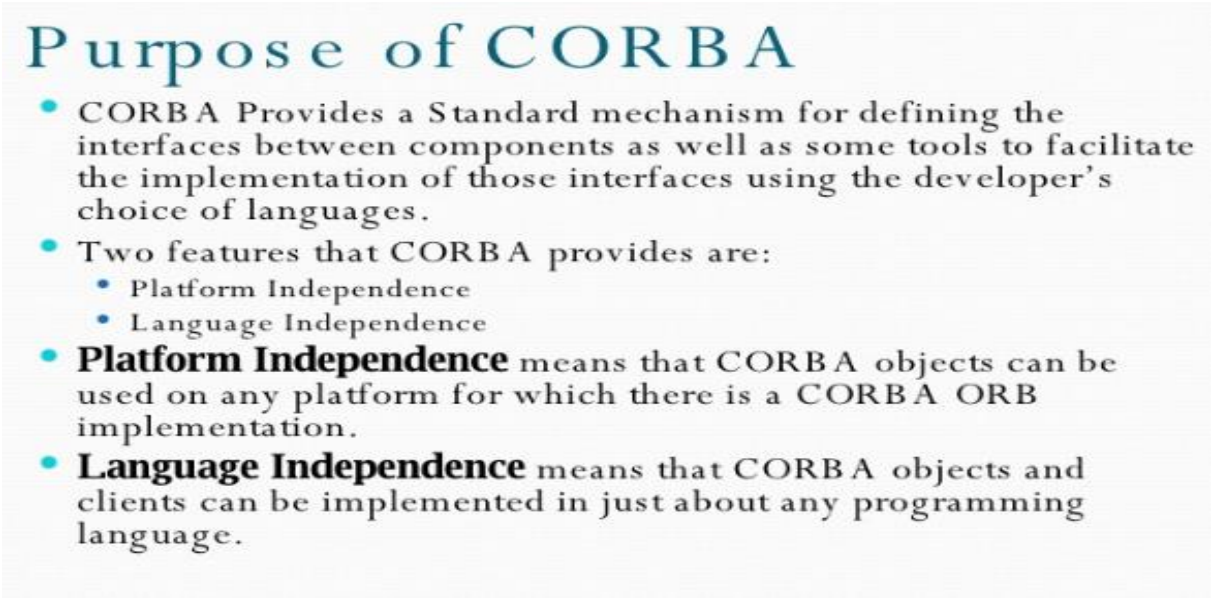
A distributed architecture is an architecture supporting the development of applications and services that can exploit a physical architecture consisting of multiple, autonomous processing elements.Those elements do not share primary memory but cooperate by sending messages over the network.



Some of the key challenges involved in designing, implementing and operating a distributed system include:

1. ***Heterogeneity***
   -Heterogeneous hardware, OS, languages, protocols etc.
2. ***Concurrency***
   -Resource sharing and concurrent access to data pose the issue of data integrity
3. ***Failure***
   -Independent failure: often we want the system to keep working after one or more have failed.
   -Unreliable communication: connections may be unavailable; messages may be lost.
4. ***Insecure communication***
   -The interconnections among the computers may be exposed to unauthorized eavesdropping and message modification.
5. ***Costly Communication***
   -The interconnections among the computers usually provide lower bandwidth, higher latency and higher cost communication compared to independent processes in a single machine.

Q3. Purpose of CORBA?



Q4. IDL Data Types?

PURPOSE OF IDL

IDL stands for Interface Definition Language. Its purpose is to define the capabilities of a distributed service along with a common set of data types for interacting with those distributed services. IDL meets a number of objectives:

- Language Independence:
- Distributed service specification:
- Definition of complex data types:
- Hardware Independence

Primitive Types

IDL features a variety of primitive types. These types store simple values such as integral numbers, floating point numbers, character strings, and so on.

a) void

- This is analogous to the void type in C, C++ and Java
- This is useful for methods that don't return any value.

b) boolean

- Stores a boolean value either 0 (false) or 1(true)
- Depending on the programming language used, the IDL boolean type can map to an integral type (short int in C/C++) or to the language's native Boolean type (as in Java)

boolean aBoolean;

c) char and wchar

- This is analogous to char type in C/C++ and Java. And directly maps.
- Stores a single character value
- The char data type is an 8-bit quantity

char aChar;

Floating Point Types

d) float

- The IDL float type represents an IEEE single-precision floating point value
- Analogous to C, C++, and Java

float aFloat;

f) double and long double

- Represents an IEEE double-precision floating point value
- Corresponds to the double type in leading languages

double aDouble;

The long double type, introduced in CORBA 2.1, represents IEEE double-extended floating point value, having an exponent of at least 15 bits and a signed fraction of at least 64 bits.

g)Integer Types

Unlike most familiar programming languages, IDL doesn't define a plain int type only short and long integer types.

a) long and long long

The IDL long type represents a 32-bit signed quantity, like C/C++'s int and Java's int

Ex: long aLong;

In CORBA 2.1 the long type was added, which is a 64-bit signed quantity.

b) unsigned long and unsigned long long

The unsigned long type in IDL is an unsigned version of the long type

Ex: unsigned long anUnsignedLong;

CORBA 2.1 added the unsigned long long type, which is a 64-bit unsigned quantity.

c) short

- Represents a 16-bit signed quantity, as in C, C++, and Java
- It's range -215 … 215-1.

Ex: short aShort;

d) unsigned short

- Unsigned version of short
- Range – 0…216-1

Ex: unsigned short aUnsignedShort;

e) octet

- It is an 8-bit quantity that is not translated in any way during transmission.
- The similar type is available in Java as byte.

Ex: octet aOctet;

f) string

- Represents a collection of characters. Similar to C++'s Cstring and Java's String class.
- In 'C' character arrays are used as string.
- IDL supports fixed-length and variable-length strings.

Ex: string aFixedLength[20];

string aVariantLength;

g)The const modifier

- const values are useful for values that should not change.
- The scope of the const value is interface or module.

Ex: const float pi = 3.14;

Q4. Why CORBA?

- CORBA Provides a Standard mechanism for defining the interfaces between components as well as some tools to facilitate the implementation of those interfaces using the developer's choice of languages.
- Two features that CORBA provides are:
    - Platform Independence
    - Language Independence
- Platform Independence means that CORBA objects can be used on any platform for which there is a CORBA ORB implementation.
- Language Independence means that CORBA objects and clients can be implemented in just about any programming language.

Q5. Interface Definition Language (IDL)?

IDL Ground Rules

a) Case Sensitivity

- Identifiers are case sensitive.
- The names of identifiers in the same scope cannot differ in case only. Ex. in the myObject interface, IDL would not allow an operation named anOperation and another operation named an OPERATION to be simultaneously.

b) IDL Definition Syntax

- All definitions in IDL are terminated by a semicolon (;), much as they are in C, C++, and Java
- Definitions that enclose other definitions do so with braces. (modules & interfaces)
- When a closing brace also appears at the end of a definition, it is also followed by a semicolon it represents a module.

c) IDL Comments

- Both C-Style and C++-Style comments are allowed
- // C++ comment allowed
- /* C – Style Comment allowed */

d) Use of the C Preprocessor

- IDL assumes the existence of a C preprocessor to process constructs such as macro definitions and conditional compilation.

Ex: #ifdef….#endif, #include etc.

e) The Module

- The module construct is used to group together IDL definitions that share a common purpose.
- A module declaration specifies the module name and encloses its members in braces.

Ex:

```
module Bank {

interface Customer {

…

};

interface Account {

…

};

…

};
```

The grouping together of similar interfaces, constant values, and the like is commonly referred to as partitioning and is a typical step in the system design process.

Partitions are also often referred to as modules or as packages.

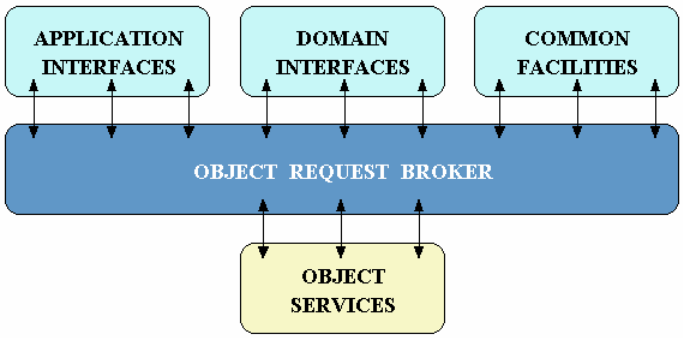
f) Coupling and Cohesion

- Loose coupling means that components in separate modules are not tightly integrated with each other; an application using components in one module generally need not know about components in another module. When there is little or no dependency between components, they are said to be loosely coupled.
- Tight cohesion means that interfaces within the module are tightly integrated with each other.
- Loose coupling of components reduces the possibility that changes to one component will require changes to another.

Q6. Overview of CORBA?

The Common Object Request Broker Architecture (CORBA)is an emerging open distributed object computing infrastructure being standardized by the Object Management Group (OMG). CORBA automates many common network programming tasks such as object registration, location, and activation; request demultiplexing; framing and error-handling; parameter marshalling and demarshalling; and operation dispatching. See the OMG Web site for more overview material on CORBA. See my CORBA page for additional information on CORBA, including our tutorials and research on high-performance and real-time ORBs. Results from our research on high-performance and real-time CORBA are freely available for downloading in the open-source TAO ORB.

The following figure illustrates the primary components in the OMG Reference Model architecture. Descriptions of these components are available further below.



Object Services -- These are domain-independent interfaces that are used by many distributed object programs. For example, a service providing for the discovery of other available services is almost always necessary regardless of the application domain. Two examples of Object Services that fulfill this role are:

- The Naming Service -- which allows clients to find objects based on names;
- The Trading Service -- which allows clients to find objects based on their properties.

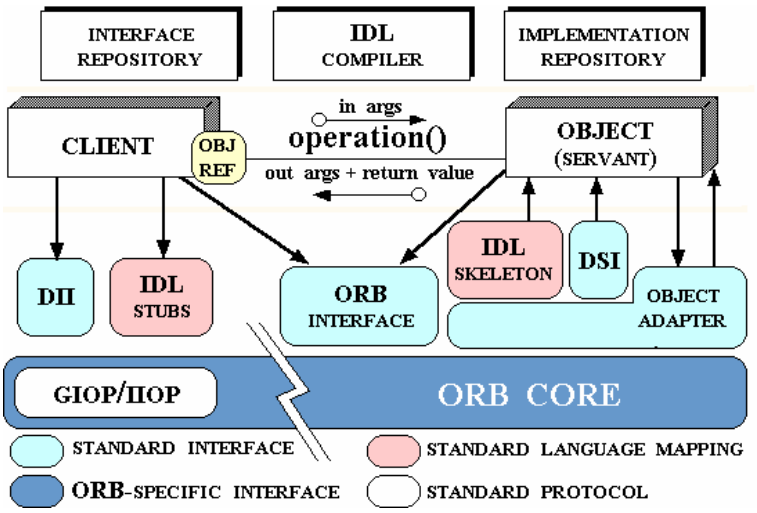
There are also Object Service specifications for lifecycle management, security, transactions, and event notification, as well as many others .

Common Facilities -- Like Object Service interfaces, these interfaces are also horizontally-oriented, but unlike Object Services they are oriented towards end-user applications. An example of such a facility is the Distributed Document Component Facility (DDCF), a compound document Common Facility based on OpenDoc. DDCF allows for the presentation and interchange of objects based on a document model, for example, facilitating the linking of a spreadsheet object into a report document.

Domain Interfaces -- These interfaces fill roles similar to Object Services and Common Facilities but are oriented towards specific application domains. For example, one of the first OMG RFPs issued for Domain Interfaces is for Product Data Management (PDM) Enablers for the manufacturing domain. Other OMG RFPs will soon be issued in the telecommunications, medical, and financial domains.

Application Interfaces - These are interfaces developed specifically for a given application. Because they are application-specific, and because the OMG does not develop applications (only specifications), these interfaces are not standardized. However, if over time it appears that certain broadly useful services emerge out of a particular application domain, they might become candidates for future OMG standardization.

CORBA ORB Architecture



Object -- This is a CORBA programming entity that consists of an identity, an interface, and an implementation, which is known as a Servant.

Servant -- This is an implementation programming language entity that defines the operations that support a CORBA IDL interface. Servants can be written in a variety of languages, including C, C++, Java, Smalltalk, and Ada.

Client -- This is the program entity that invokes an operation on an object implementation. Accessing the services of a remote object should be transparent to the caller. Ideally, it should be as simple as calling a method on an object, i.e., obj->op(args). The remaining components in Figure 2 help to support this level of transparency.

Object Request Broker (ORB) -- The ORB provides a mechanism for transparently communicating client requests to target object implementations. The ORB simplifies distributed programming by decoupling the client from the details of the method invocations. This makes client requests appear to be local procedure calls. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.

ORB Interface -- An ORB is a logical entity that may be implemented in various ways (such as one or more processes or a set of libraries). To decouple applications from implementation details, the CORBA specification defines an abstract interface for an ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the dynamic invocation interface described below.

CORBA IDL stubs and skeletons -- CORBA IDL stubs and skeletons serve as the ``glue'' between the client and server applications, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by a CORBA IDL compiler. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimizations.

Dynamic Invocation Interface (DII) -- This interface allows a client to directly access the underlying request mechanisms provided by an ORB. Applications use the DII to dynamically issue requests to objects without requiring IDL interface-specific stubs to be linked in. Unlike IDL stubs (which only allow RPC-style requests), the DII also allows clients to make non-blocking deferred synchronous (separate send and receive operations) and oneway (send-only) calls.

Dynamic Skeleton Interface (DSI) -- This is the server side's analogue to the client side's DII. The DSI allows an ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.

Object Adapter -- This assists the ORB with delivering requests to the object and with activating the object. More importantly, an object adapter associates object implementations with the ORB. Object adapters can be specialized to provide support for certain object implementation styles (such as OODB object adapters for persistence and library object adapters for non-remote objects).

Q7. Building a CORBA Application?

The outline of the process is

- Define the server's interfaces using IDL
- Choose an implementation approach for the server's interfaces
- Use the IDL compiler to generate client stubs and server skeletons for the server interfaces.
- Implement the server interfaces.
- Compile the server application
- Run the server application

Building a CORBA Server

Since a server can be tested without a client, the first step is to implement the server.

The server interfaces define the capabilities that will be made available by the server and how these capabilities are accessed.

Implement those interfaces and finally compile and run the server.

1.Defining the Server Interfaces (Stock Market Server Example):

- A Service is desired that, when given a stock symbol, will return the value of that stock at that particular time.
- As an added convenience, the service will also return a list of all known stock symbols upon the request.
- So, the StockServer interface could be defined to provide two services: (1) getStockValue() (2) getStockSymbols()
- getStockValue() should take a StockSymbol as a parameter and return a floating-point result.
- getStockSymbol() need not take any arguments and should return a list of StockSymbol objects.

```
StockMarket.idl
module StockMarket {
typedef string StockSymbol;
typedef sequence <StockSymbol> StockSymbolList;
interface StockServer {
float getStockValue (in StockSymbol symbol);
StockSymbolList getStockSymbols();
};
};
```

- Group the related interfaces and types into IDL modules.
- IDL offers two constructs to represent lists. (1) the sequence (2) the array.
- Here, the size of the list is unknown, so the sequence is used.

However a method cannot return a sequence directly. So

typedef sequence <StockSymbol>

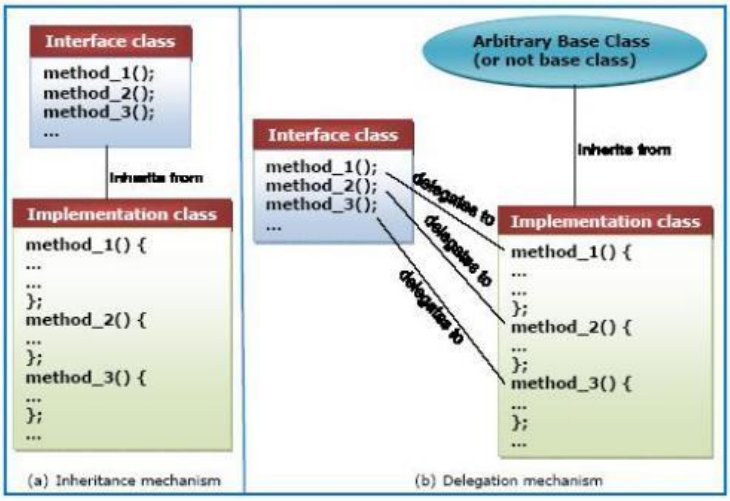
StockSymbolList; is used.

2.Choosing an implementation approach:

CORBA supports two mechanisms for implementation of IDL interfaces:

i. **Inheritance mechanism** – in which a class implements an interface by inheriting from that interface class.

ii. **Delegation mechanism** – in which the methods of the interface class call the methods of the implementing class.



- Implementation by inheritance consists of a base class that defines the interfaces of a particular object and a separate class, inheriting from this base class, which provides the actual implementations of these interfaces.
- Implementation by delegation consists of a class that defines the interfaces for an object and then delegates their implementations to another class or classes.
- The primary differences between the inheritance and delegation approaches is that in delegation, the implementation classes need not derive from any class in particular.
- A tie class, or simply a tie, is the class to which implementations are delegated in the delegation approach. Thus, the approach is often referred to as the tie mechanism or tying.
- Most IDL compilers accept command-line arguments to determine which implementation approach to generate code for.

How to choose an implementation approach??

- If an application makes use of legacy code to implement an interface, it might not be practical to change the classes in that legacy code to inherit from a class generated by the IDL compiler.
- Therefore, for such an application it would make more sense to use the delegation approach; existing classes can readily be transformed into the classes.

Using the IDL Compiler

The command to invoke the IDL compiler, included with Sun's Java IDL product is,

idltojava –fno-cpp –fclient –fserver StockMarket.idl

-fno-cpp : switch instructs the IDL compiler to not invoke the C preprocessor before compiling the file.

-fclient and –fserver : switches instruct the IDL compiler to generate client stubs and server skeletons, respectively.

Client Stubs and Server Skeletons

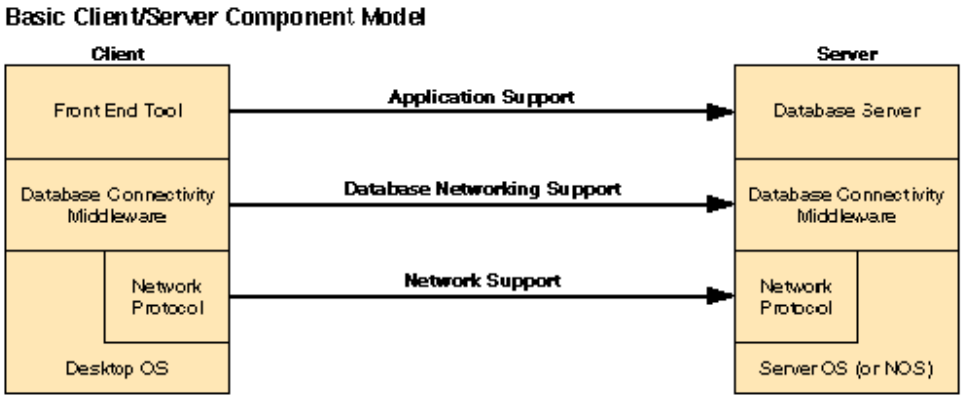
- When the IDL compiler is invoked, it generates code that conforms to the language mapping used by that particular product.
- The IDL compiler will generate a number of files – some of them helper classes, some them client stub classes, and some of them server skeleton classes.
- The stubs do nothing more than tell the client's ORB to marshal and unmarshal outgoing and incoming parameters.
- The skeletons pass incoming parameters to the implementation code and passing outgoing parameters back to the client.
- The names of the files generated by the IDL compiler are dependent on the language mapping used and sometimes on command-line arguments passed to the IDL compiler.
- The contents of these files will remain the same, for the most part, regardless of the IDL compiler used.

Q8. CORBA and Networking Model?

This involves:

- CORBA application
- IIOP's(Internet Inter ORB Protocol)
- TCP/IP Network
- DCE Network

Network application model to help explain the concept of middleware, and we'll use it throughout this guide. Instead of the OSI network model, we have basic chunks on servers and clients, including operating system, application program, and the connectivity pieces like network protocols and middleware software. Here's our model which is defined in the context of database networking middleware:



The middleware is right there in the middle (called database connectivity middleware in our diagram). Notice that middleware components are instantiated on both client and server platforms. Since some definitions of middleware include code that isolates operating systems from hardware platform differences (like the hardware abstraction layer or HAL in Windows NT), let's keep our discussion focused on middleware for distributed systems. In some cases, there will be more middleware services provided by an intermediate device like a database gateway. The next diagram provides a more specific example using Oracle and PowerBuilder on common platforms:

UNIT-V

COM: COM – Data types – Interfaces – Proxy and stub – Marshalling – Implementing server/Client – Interface pointers – Object Creation, Invocation, Destruction – Comparison COM and CORBA – Introduction to .NET – Overview of .NET architecture–Marshalling – Remoting .

Concepts:

Q1. .NET Framework?

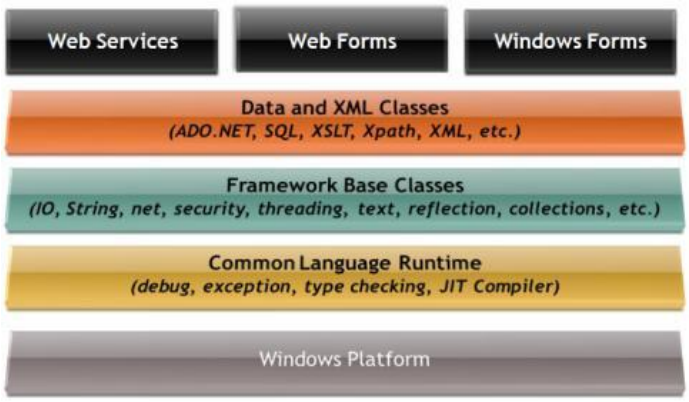
Microsoft .NET supports not only language independence, but also language integration. This means that you can inherit from classes, catch exceptions, and take advantage of polymorphism across different languages. The .NET Framework makes this possible with a specification called the Common Type System (CTS) that all .NET components must obey. For example, everything in .NET is an object of a specific class that derives from the root class called System.Object. The CTS supports the general concept of classes, interfaces, delegates (which support callbacks), reference types, and value types.

Additionally, .NET includes a Common Language Specification (CLS), which provides a series of basic rules that are required for language integration. The CLS determines the minimum requirements for being a .NET language. Compilers that conform to the CLS create objects that can interoperate with one another. The entire Framework Class Library (FCL) can be used by any language that conforms to the CLS.

The .NET Framework sits on top of the operating system, which can be any flavor of Windows, and consists of a number of components, currently including:

- Four official languages: C#, VB.NET, Managed C++, and JScript.NET.
- The CLR, an object-oriented platform for Windows and web development that all these languages share.
- A number of related class libraries, collectively known as the FCL.

Figure breaks down the .NET Framework into its system architectural components.



The most important component of the .NET Framework is the CLR, which provides the environment in which programs are executed. The CLR includes a virtual machine, analogous in many ways to the Java virtual machine. At a high level, the CLR activates objects, performs security checks on them, lays them out in memory, executes them, and garbage-collects them. (The Common Type System is also part of the CLR.)

In Figure 1, the layer on top of the CLR is a set of framework base classes, followed by an additional layer of data and XML classes, plus another layer of classes intended for web services, Web Forms, and Windows Forms. Collectively, these classes make up the FCL, one of the largest class libraries in history and one that provides an object-oriented API for all the functionality that the .NET platform encapsulates. With more than 4,000 classes, the FCL facilitates rapid development of desktop, client/server, and other web services and applications.

The set of Framework base classes, the lowest level of the FCL, is similar to the set of classes in Java. These classes support rudimentary input and output, string manipulation, security management, network communication, thread management, text manipulation, reflection and collections functionality, etc.

Above this level is a tier of classes that extend the base classes to support data management and XML manipulation. The data classes support persistent management of data that is maintained on backend databases. These classes include the Structured Query Language (SQL) classes to let you manipulate persistent data stores through a standard SQL interface. Additionally, a set of classes called ADO.NET allows you to manipulate persistent data. The .NET Framework also supports a number of classes to let you manipulate XML data and perform XML searching and translations.

Extending the Framework base classes and the data and XML classes is a tier of classes geared toward building applications using three different technologies: Web Services, Web Forms, and Windows Forms. Web services include a number of classes that support the development of lightweight distributed components, which will work even in the face of firewalls and NAT software. Because web services employ standard HTTP and SOAP as underlying communications protocols, these components support Plug and Play across cyberspace.

Web Forms and Windows Forms allow you to apply Rapid Application Development techniques to building web and Windows applications. Simply drag and drop controls onto your form, double-click a control, and write the code to respond to the associated event.

Introduction to .NET

The .NET Platform

The .NET platform is a new development framework that provides a fresh application programming interface (API) to the services and APIs of classic Windows operating systems (especially the Windows 2000 family), while bringing together a number of disparate technologies. This includes COM+ component services, the ASP web development framework, a commitment to XML and object-oriented design, support for new web services protocols such as SOAP, WSDL, and UDDI, and a focus on the Internet, all integrated within the DNA architecture.

The platform consists of four separate product groups:

- A set of languages, including C# and Visual Basic .NET, a set of development tools including Visual Studio .NET, a comprehensive class library for building web services and web and Windows applications, as well as the Common Language Runtime (CLR) to execute objects built within this framework.
- A set of .NET Enterprise Servers, formerly known as SQL Server 2000, Exchange 2000, BizTalk 2000, and so on, that provide specialized functionality for relational data storage, email, B2B commerce, etc.
- An offering of commercial web services, called .NET My Services. For a fee, developers can use these services in building applications that require knowledge of user identity, etc.
- New .NET-enabled non-PC devices, from cell phones to game boxes.

Q2. COM Data types?

The COM uses several data types to communicate with the host computer. Before using a variable, it is best to declare the variable as the type of data it will store. It saves memory and is usually faster to access. The following are the most common data types:

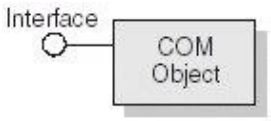
- **Long Integer**- Long (long integer) variables are stored as signed 32-bit (4-byte) numbers ranging in value from -2,147,483,648 to 2,147,483,647.
- **Single Precision (Real)**- Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values.
- **Double Precision (Real)-** Double (double-precision floating-point) variables are stored as IEEE 64-bit (8-byte) floating-point numbers ranging in value from -1.79769313486232E308 to -4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to 1.79769313486232E308 for positive values.
- **Boolean-** Boolean variables are stored as 16-bit (2-byte) numbers, but they can only be True or False. Use the keywords True and False to assign one of the two states to Boolean variables.bWhen other numeric types are converted to Boolean values, 0 becomes False and all other values become True. When Boolean values are converted to other data types, False becomes 0 and True becomes -1. In PNA release 5.26, the following properties were changed to return True rather than 1 to conform with this definition. This change may affect the functionality of your COM program:
- **String-** String variables hold character information. A String variable can contain approximately 65,535 bytes (64K), is either fixed-length or variable-length, and contains one character per byte. Fixed-length strings are declared to be a specific length. Variable-length strings can be any length up to 64K, less a small amount of storage overhead.

- **Object-** Object variables are stored as 32-bit (4-byte) addresses that refer to objects within the analyzer or within some other application. A variable declared as Object is one that can subsequently be assigned (using the Set statement) to refer to any actual analyzer object.

- **Enumeration-** Enumerations (Enum) are a set of named constant values. They allow the programmer to refer to a constant value by name instead of by number. For example:
  Enum Days Of Week
  Sunday = 0
  Monday = 1
  Tuesday = 2
  Wednesday = 3
  Thursday = 4
  Friday = 5
  Saturday = 6
  End Enum
- **Variant-** If you don't declare a data type ("typed" data) the variable is given the Variant data type. The Variant data type is like a chameleon — it can represent many different data types in different situations.The PNA provides and receives Variant data because there are programming languages that cannot send or receive "typed" data. Variant data transfers at a slower rate than "typed" data.

Q3. COM Interfaces?

COM clients communicate with objects through COM interfaces. Interfaces are groups of logically or semantically related routines which provide communication between a provider of a service (server object) and its clients. The standard way to depict a COM interface is as follows:



For example, every COM object must implement the basic interface, The Fundamental COM Interface, IUnknown. Through a routine called QueryInterface in IUnknown, clients can request other interfaces implemented by the server.

Objects can have multiple interfaces, where each interface implements a feature. An interface provides a way to convey to the client what service it provides, without providing implementation details of how or where the object provides this service.

Key aspects of COM interfaces are as follows:

Once published, interfaces are immutable; that is, they do not change. You can rely on an interface to provide a specific set of functions. Additional functionality is provided by additional interfaces.

By convention, COM interface identifiers begin with a capital I and a symbolic name that defines the interface, such as IMalloc or IPersist.

Interfaces are guaranteed to have a unique identification, called a Globally Unique Identifier (GUID), which is a 128-bit randomly generated number. Interface GUIDs are called Interface Identifiers (IIDs). This eliminates naming conflicts between different versions of a product or different products.

Interfaces are language independent. You can use any language to implement a COM interface as long as the language supports a structure of pointers, and can call a function through a pointer either explicitly or implicitly.

Interfaces are not objects themselves; they provide a way to access an object. Therefore, clients do not access data directly; clients access data through COM Interface Pointers. Windows 2000 adds an additional layer of indirection known as an interceptor through which it provides COM+ features such as just-in-time activation and object pooling.

Interfaces are always inherited from the fundamental interface, [[The Fundamental COM Interface, IUnknown].

Interfaces can be redirected by COM through proxies to enable interface method calls to call between threads, processes, and networked machines, all without the client or server objects ever being aware of the redirection. For more information, see In-process, Out-of-process, and Remote Servers.

Q4. What is the difference between a proxy, Skeleton and a stub?

stub

A routine that doesn't actually do anything other than declare itselfand the parameters it accepts. Stubs are used commonly as placeholdersfor routines that still need to be developed. The stub contains just enough code to allow it to be compiled and linked with the rest of theprogram.

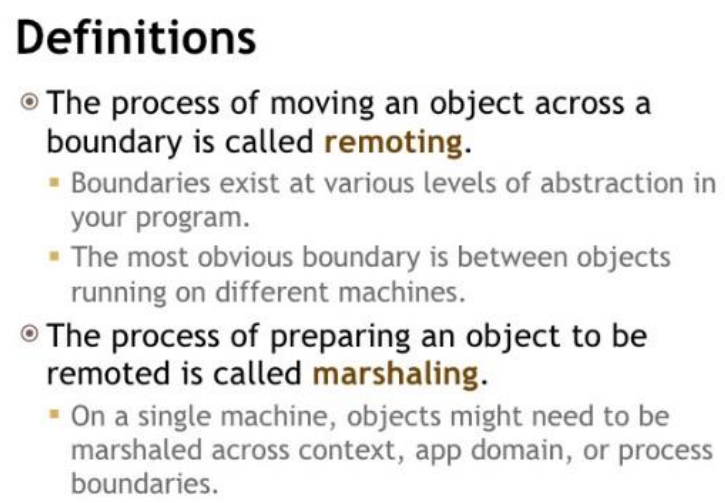
proxy

A server that sits between a client application, such as a Web browser, and a real server. It intercepts all requests to the real server to see if it can fulfill the requests itself. If not, it forwards the request to the real server.

Skeleton

A skeleton is a proxy for a remote object that runs on the server. Skeletons return the results of server methodinvocations to clients via stubs. Skeleton is a server side proxy,the purpose of skeleton is converting the network oriented stream into java program(human readable format).To generate stubs and skeletons we can use "rmic" tool in JDK.

Q5. Marshalling and Remoting?



The days of integrated programs all running in a single process on a single machine are, if not dead, at least seriously wounded. Today's programs consist of complex components running in multiple processes, often across the network. The Web has facilitated distributed applications in a way that was unthinkable even a few years ago, and the trend is toward distribution of responsibility.

A second trend is toward centralizing business logic on large servers. Although these trends appear to be contradictory, in fact they are synergistic: business objects are being centralized while the user interface and even some middleware are being distributed.

The net effect is that objects need to be able to talk with one another at a distance. Objects running on a server handling the web user interface need to be able to interact with business objects living on centralized servers at corporate headquarters.

The process of moving an object across a boundary is called remoting. Boundaries exist at various levels of abstraction in your program. The most obvious boundary is between objects running on different machines.

The process of preparing an object to be remoted is called marshaling. On a single machine, objects might need to be marshaled across context, app domain, or process boundaries.

A process is essentially a running application. If an object in your word processor wants to interact with an object in your spreadsheet, they must communicate across process boundaries.

Processes are divided into application domains (often called app domains); these in turn are divided into various contexts. App domains act like lightweight processes, and contexts create boundaries that objects with similar rules can be contained within. At times, objects will be marshaled across both context and app domain boundaries, as well as across process and machine boundaries.

When an object is remoted, it appears to be sent through the wire from one computer to another. The actual transmission of your message is done by a channel. The channel works with a formatter. The formatter makes sure the message is in the right format.

Q6. Marshalling and UnMarshalling?

In few words, "marshalling" refers to the process of converting the data or the objects inbto a byte-stream, and "unmarshalling" is the reverse process of converting the byte-stream back to their original data or object. The conversion is achieved through "serialization".

The purpose of the "marshalling/unmarshalling" process is to transfer data between the RMI system.