RGPVNOTES.IN

Please do not share these notes on apps like WhatsApp or Telegram.

The revenue we generate from the ads we show on our website and app funds our services. The generated revenue **helps us prepare new notes and improve the quality of existing study materials**, which are available on our website and mobile app.

If you don't use our website and app directly, it will hurt our revenue, and we might not be able to run the services and **have to close them**. So, it is a humble request for all to **stop sharing the study material** we provide on various apps. Please **share the website's URL instead.**

Course Contents:

Unit 3. Software architecture implementation technologies: Software Architecture Description Languages (ADLs), Struts, Hibernate, Node JS, Angular JS, J2EE – JSP, Servlets, EJBs; middleware: JDBC, JNDI, JMS, RMI and CORBA etc. Role of UML in software architecture.

Unit-3

Software Architecture Description Languages (ADLs)

Architecture description languages (ADLs) are widely used to describe system architectures. Components and connectors are the main elements of ADLs and include rules and guidelines for well-formed architectures. The output of the architecture description process is the system architecture documents. It should describe how the system is structured into sub-systems and how each sub-system is structured into components. Each component is described based on many properties such as behaviour. The generated architecture description includes a static structure of components, their dynamic processes, interface modules, and the relationships among components. ADLs are important in system component design, since they affect system performance, robustness, disreputability and maintainability.

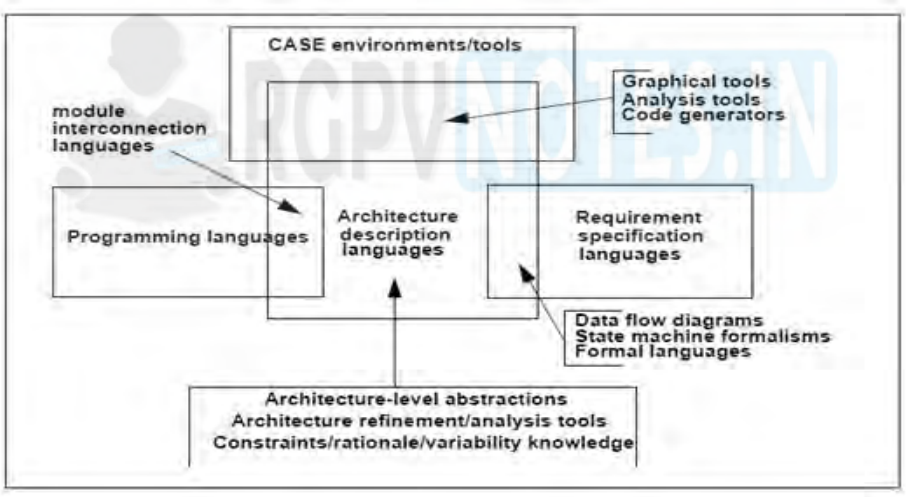


Figure 3.1: Relation of ADL's to other notations and Tools

Figure shows the relation of ADLs to other, more familiar software engineering notations and tools.

1. Parts of traditional programming languages (e.g., Ada package specifications) represent module interconnection, which is also important for ADLs.
2. Requirements specification languages share some notations with ADLs.
3. Computer Aided Software Engineering (CASE) environments significantly overlap with ADLs, but there is much Computer Aided Software Engineering (CASE) functionality that is not part of architecture (e.g. test tools).

Characteristics:

- ADLs support the routine use of existing designs and components in new application systems.
- ADLs support the evaluation of an application system before it is built.

Elements of ADL:

1. **Component -** Primitive building block.
2. **Connector -** Mechanisms of combining components.

3. **Abstraction-** Rules for referring to the combination of components and connectors.

Important Properties of ADL

- **Composition –** composition of independent components and connections.
- **Abstraction –** need to describe design elements clearly and explicitly.
- **Reusability –** ability to reuse components, connectors in different architectural descriptions.
- **Configuration –** ability to understand and change the architectural structure.
- **Heterogeneity –** ability to combine multiple, heterogeneous architectural descriptions.
- **Analysis -** possible to perform rich and varied analysis of architectural description.

Advantage

- ADLs represent a formal way of representing architecture.
- ADLs are intended to be both human and machine readable.
- It support describing a system at a higher level than previously possible.
- ADLs permit analysis of architectures – completeness, consistency, ambiguity, and performance.
- It can support automatic generation of software systems.

Disadvantage

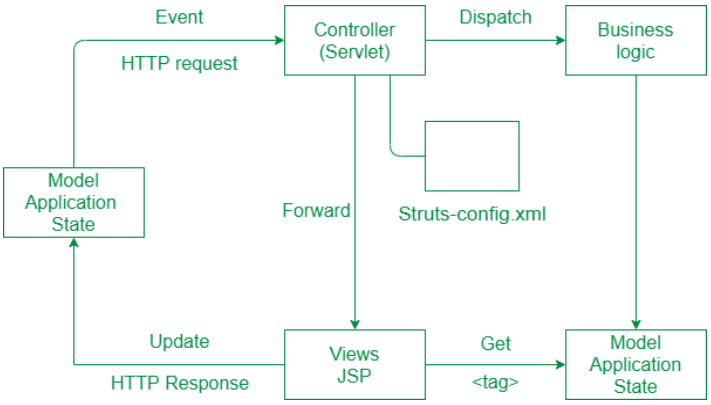
- There is not universal agreement on what ADLs should represent, particularly as regards the behaviour of the architecture.
- Representations currently in use are relatively difficult to parse and are not supported by commercial tools.
- Most ADL work today has been undertaken with academic rather than commercial goals in mind.
- Most ADLs tend to be very vertically optimized toward a particular kind of analysis.

Struts

Struts is used to create a web applications based on servlet and JSP. It depend on the MVC (Model View Controller) framework and its application is a genuine web application. Struts are thoroughly useful in building J2EE (Java 2 Platform, Enterprise Edition) applications because struts takes advantage of J2EE design patterns. Struts follows these J2EE design patterns including MVC.

Struts consists of a set of own custom tag libraries. Struts are based on MVC framework which is pattern oriented and includes JSP custom tag libraries. Struts also supports utility classes.

Working of Struts



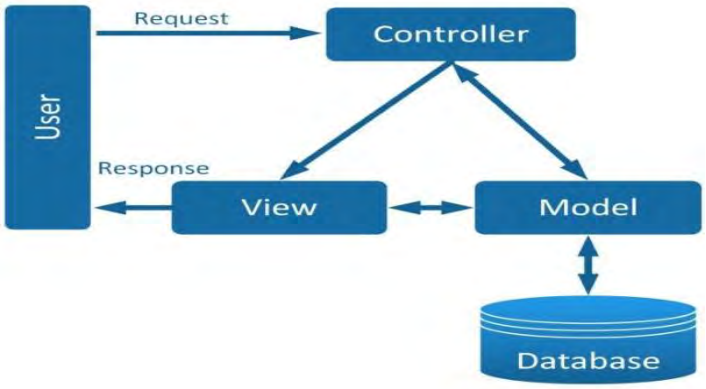
In the initialization phase, the controller rectifies a configuration file and used it to deploy other control layer objects. Struts configuration is form by these objects combined. The struts configuration defines among other things the action mappings for an application. Struts controller servlet considers the action mappings and routes the HTTP requests to other components in the framework. Request is first delivered to an action and then to JSP.

Figure 3.2: Working of Struts

The mapping helps the controller to change HTTP requests into application actions. The action objects can handle the request from and responds to the client (generally a web browser). Action objects have access to the applications controller servlet and also access to the servlet's methods. When delivering the control, an action objects can indirectly forward one or more share objects, including java-beans by establish them in the typical situation shared by java servlets.

Model View Controller Architecture

The MVC design pattern consists of three modules model, view and controller.



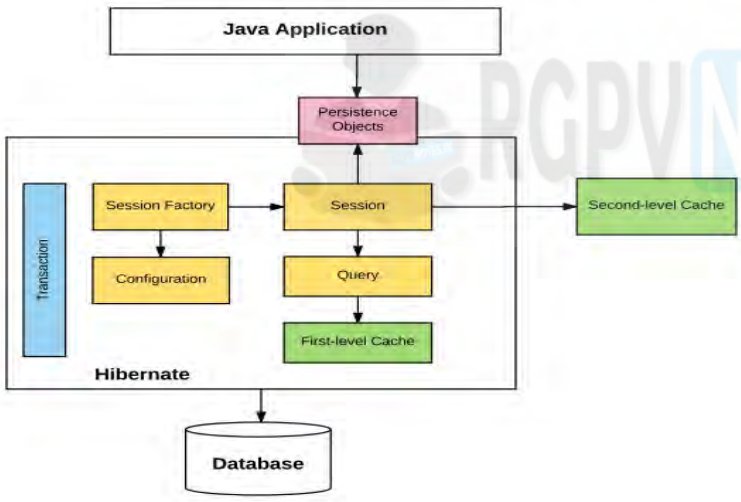
1. **Model** The model represents the state (data) and business logic of the application.
2. **View** The view module is responsible to display data i.e. it represents the presentation.
3. **Controller** The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

Figure 3.3: MVC Architecture

Advantage of MVC architectures	**Disadvantage of MVC Architecture**
• Navigation control is centralized only controller contains the logic to determine the next page. • Easy to maintain, extend and test. • Better separation of concerns.	• We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

Hibernate

Hibernate Architecture summarizes the main building blocks in hibernate architecture.



Hibernate is an open source Java persistence framework project. It performs powerful object-relational mapping and query databases using HQL and SQL. Hibernate is a great tool for ORM mappings in Java. Hibernate's primary feature is mapping from Java classes to database tables, and mapping from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

Figure 3.4: Hibernate Architecture

1. **Configuration**: Generally written in hibernate.properties or hibernate.cfg.xml files. For Java configuration, you may find class annotated with @Configuration. It is used by Session Factory to work with Java Application and the Database. It represents an entire set of mappings of an application Java Types to an SQL database.
2. **Session Factory**: Any user application requests Session Factory for a session object. Session Factory uses configuration information from above listed files, to instantiates the session object appropriately.
3. **Session**: This represents the interaction between the application and the database at any point of time. This is represented by the org.hibernate.Session class. The instance of a session can be retrieved from the SessionFactory bean**.**
4. **Query**: It allows applications to query the database for one or more stored objects. Hibernate provides different techniques to query database, including NamedQuery and CriteriaAPI**.**
5. **First-level cache:** It represent the default cache used by Hibernate Session object while interacting with the database. It is also called as session cache and caches objects within the current session. All

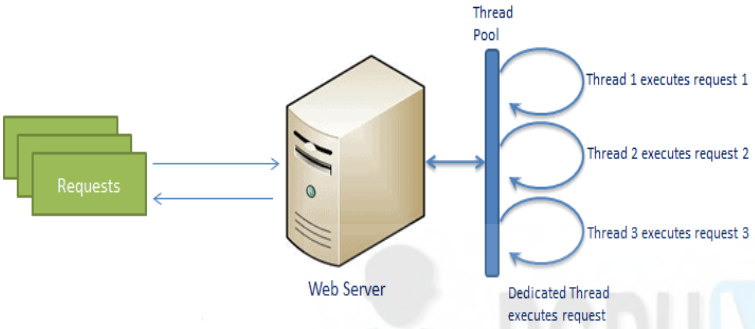
requests from the session object to the database must pass through the first level cache or session cache.

6. **Transaction**: enables you to achieve data consistency, and rollback in case something goes unexpected.
7. **Persistent objects**: These are plain old Java objects (POJOs), which get persisted as one of the rows in the related table in the database by hibernate. They can be configured in configurations files (hibernate.cfg.xml or hibernate.properties) or annotated with @Entity annotation.
8. **Second-level cache**: It is used to store objects across sessions. This needs to be explicitly enabled and one would be required to provide the cache provider for a second-level cache. One of the common second-level cache providers is EhCache.

Node.js

Node.js is an open-source server-side runtime environment built on Chrome's V8 JavaScript engine. It provides an event driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side application using JavaScript.

Traditional Web Server Model



In the traditional web server model, each request is handled by a dedicated thread from the thread pool. If no thread is available in the thread pool at any point of time then the request waits till the next available thread. Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.

Figure 3.5: Traditional Web Server Model

Node.js Process Model

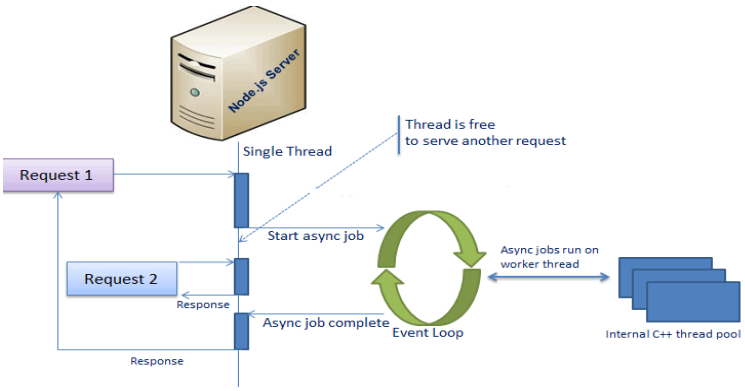


Figure 3.6: Node.js Process Model

Node.js processes user requests differently when compared to a traditional web server model. It runs in a single process and the application code runs in a single thread. All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request. So, this single thread doesn't have to wait for the request to complete and is free to handle the next request. When asynchronous I/O work completes then it processes the request further and sends the response.

Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.

Angular JS

AngularJS is a client side JavaScript MVC framework to develop a dynamic web application. It was originally started as a project in Google but now, it is open source framework. AngularJS is entirely based on HTML and JavaScript. It changes static HTML to dynamic HTML. It extends the ability of HTML by adding built-in attributes and components and also provides an ability to create custom attributes using simple JavaScript.

Example

```
<html>
<head>
<title>AngularJS First Application</title>
</head>
<body>
<h1>Sample Application</h1>
   <div ng-app = "">
      <p>Enter your Name: <input type = "text" ng-model = "name"></p>
      <p>Hello <span ng-bind = "name"></span>!</p>
    </div>
<script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
   </script>
</body>
</html>
```

Output: Enter your Name: 

Hello !

Advantages of AngularJS

- **Dependency Injection**: Dependency Injection specifies a design pattern in which components are given their dependencies instead of hard coding them within the component.
- **Two way data binding:** AngularJS creates a two way data-binding between the select element and the orderProp model. orderProp is then used as the input for the orderBy filter.
- **Testing:** Angular JS is designed in a way that we can test right from the start. So, it is very easy to test any of its components through unit testing and end-to-end testing.
- **Model View Controller:** In Angular JS, it is very easy to develop application in a clean MVC way. You just have to split your application code into MVC components i.e. Model, View and the Controller.

JSP (Java Server Page)

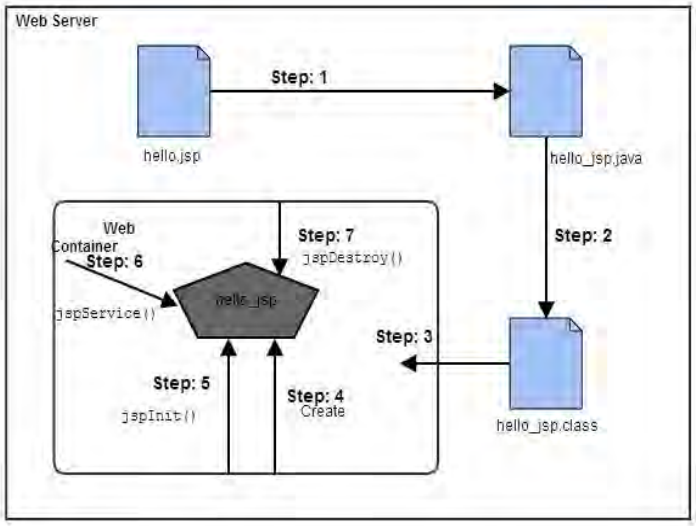
JSP technology is used to create dynamic web applications. JSP pages are easier to maintain then a Servlet, as servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags. Everything a Servlet can do, a JSP page can also do it. JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs.

Lifecycle of JSP

A JSP page is converted into Servlet in order to service requests. The translation of a JSP page to a Servlet is called Lifecycle of JSP. JSP Lifecycle is exactly same as the Servlet Lifecycle, with one additional first step, which is, translation of JSP code to Servlet code.

Following are the JSP Lifecycle steps:

1. Translation of JSP to Servlet code.
2. Compilation of Servlet to byte code.
3. Loading Servlet class.
4. Creating servlet instance.
5. Initialization by calling jspInit() method.
6. Request Processing by calling jspService() method Destroying by calling jspDestroy() method.



Web Container translates JSP code into a servlet class source(.java) file, then compiles that into a java servlet class. In the third step, the servlet class bytecode is loaded using classloader. The Container then creates an instance of that servlet class.

The initialized servlet can now service request. For each request the Web Container call the jspService() method. When the Container removes the servlet instance from service, it calls the jspDestroy() method to perform any required clean up.

Figure 3.7: Lifecycle of JSP

TAGS in JSP

Writing a program in JSP is nothing but making use of various tags which are available in JSP. In JSP we have three categories of tags- scripting elements, directives and standard actions.

SCRIPTING ELEMENTS: Scripting elements are basically used to develop preliminary programming in JSP such as, declaration of variables, expressions and writing the java code. Scripting elements are divided into three types- declaration tag, expression tag and scriptlet.

1. Declaration tag: Whenever we use any variables as a part of JSP we have to use those variables in the form of declaration tag i.e., declaration tag is used for declaring the variables in JSP page.

Syntax: <%! Variable declaration or method definition %>

When we declare any variable as a part of declaration tag these variables will be available as data members in the servlet and they can be accessed throughout the entire servlet. When we use any methods definition as a part of declaration tag they will be available as member methods in servlet and it will be called automatically by the servlet container as a part of service method.

For example-1:

<%! int a=10,b=30,c; %>

Example 2: <%! Int count () { return(a+b); } %>

2. Expression tag: Expression tags are used for writing the java valid expressions as a part of JSP page.

Syntax: <%=java valid expression %>

Whatever the expression we write as a part of expression tags that will be given as a response to client by the servlet container. All the expression we write in expression tag they will be placed automatically in out.println () method and this method is available as a part of service method. Expressions in the expression tag should not be terminated by semi-colon (;).

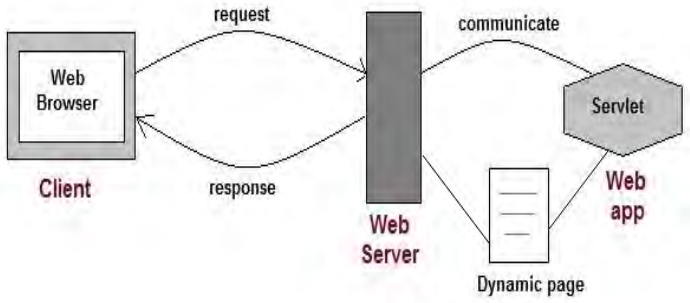
Example-1 <%! Int a=10, b=20 %> <%=a+b%>

The equivalent servlet code for the above expression tag is out.println (a+b); out is implicit object of JSPWriter class.

3. scriplet tag: scriplets are basically used to write a pure java code. Whatever the java code we write as a part of scriplet, that code will be available as a part of service () method of servlet.

Syntax: <%pure java code% >

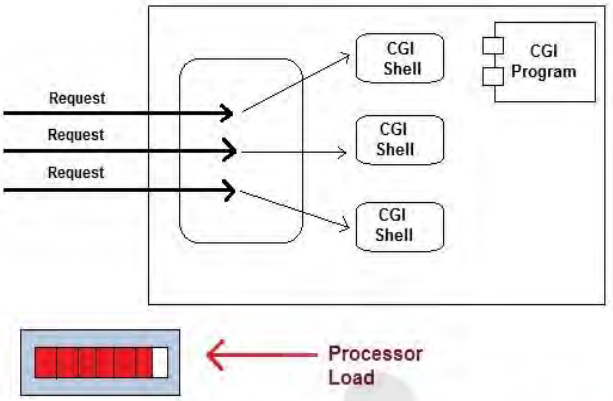
Servlet



Servlet Technology is used to create web applications. Servlet technology uses Java language to create web applications. Web applications are helper applications that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.

Figure 3.8: Servlet

CGI (Common Gateway Interface) – CGI programming was used to create web applications.



Here's how a CGI program works:

- User clicks a link that has URL to a dynamic page instead of a static page.
- The URL decides which CGI program to execute.
- Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
- The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.

Figure 3.9: Common Gateway Interface

Drawbacks of CGI programs

- High response time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

Advantages of using Servlets

- Less response time because each request runs in a separate thread.
- Servlets are scalable.
- Servlets are robust and object oriented.
- Servlets are platform independent.

How a Servlet Application works

Web container is responsible for managing execution of servlets and JSP pages for Java EE application. When a request comes in for a servlet, the server hands the request to the Web Container. **Web Container** is responsible for instantiating the servlet or creating a new thread to handle the request. It is the job of Web Container to get the request and response to the servlet. The container creates multiple threads to process multiple requests to a single servlet.

Servlets don't have a main() method. Web Container manages the life cycle of a Servlet instance.

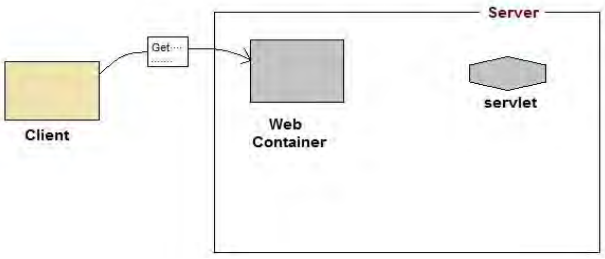


Figure 3.10 (a): User sends request

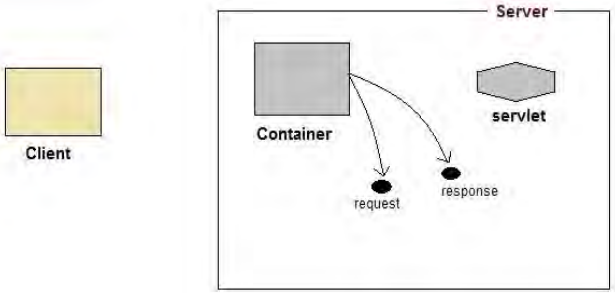


Figure 3.10 (b): Container find servlet

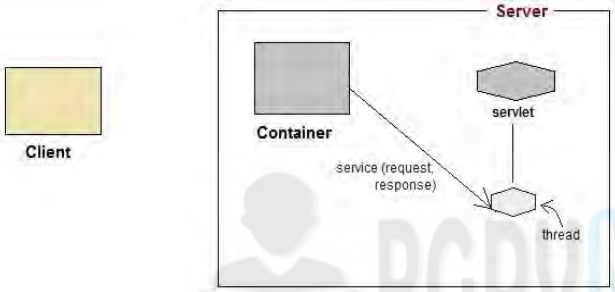


Figure 3.10 (c): Container allocates thread

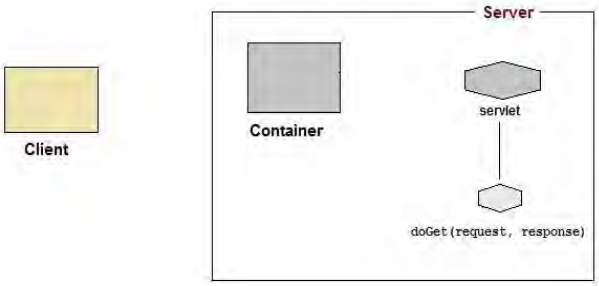


Figure 3.10 (d): Service method used

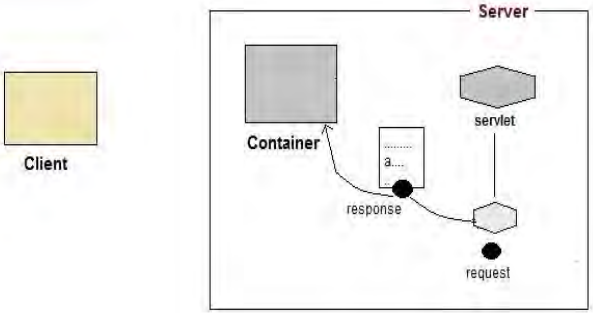


Figure 3.10 (e): Servlet Response

1. User sends request for a servlet by clicking a link that has URL to a servlet (Figure 3.10 (a)).
2. The container finds the servlet using **deployment descriptor** and creates two objects - **HttpServletRequest, HttpServletResponse** (Figure 3.10 (b))**.**
3. Then the container creates or allocates a thread for that request and calls the Servlet's service() method and passes the **request, response** objects as arguments (Figure 3.10 (c)).
4. The service () method, then decides which servlet method, doGet()or doPost() to call, based on **HTTP Request Method**(Get, Post etc) sent by the client (Figure 3.10 (d).
5. Then the Servlet uses response object to write the response back to the client(Figure 3.10 (e).
6. After the service() method is completed the thread dies. And the request and response objects are ready for garbage collection(Figure 3.10 (f).

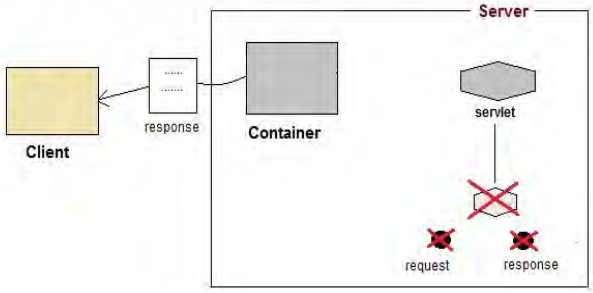


Figure 3.10 (f): Servlet Destroy

Servlet Life Cycle

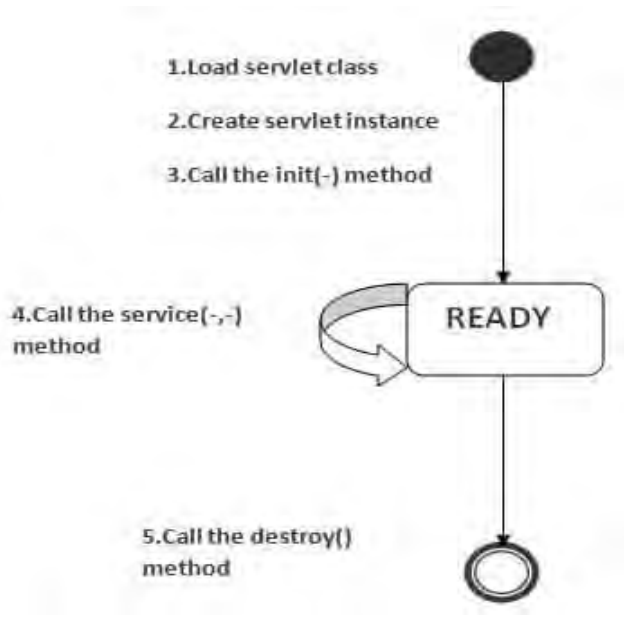


Figure 3.11: Lifecycle of Servlet

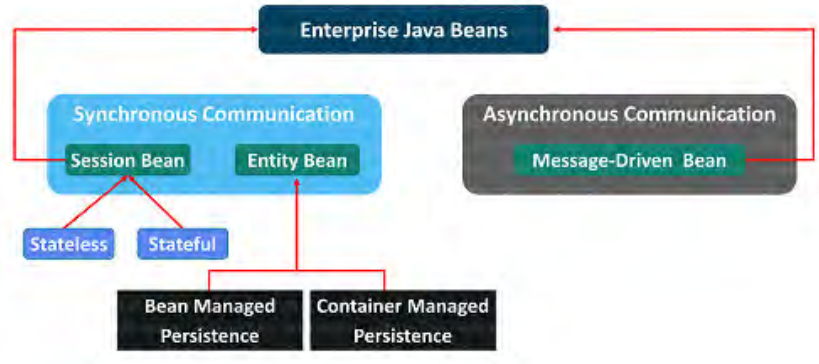
A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

There are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy () method, it shifts to the end state.

1. **Servlet class is loaded** - The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.
2. **Servlet instance is created -** The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.
3. **init method is invoked-** The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:
   public void init(ServletConfig config) **throws** ServletException
4. **service method is invoked -** The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:
   public void service (ServletRequest request, ServletResponse response)
   throws ServletException, IOException
5. **Destroy method is invoked** - The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:
   public void destroy()

EJB (Enterprise Java Beans)



EJB is server-side software that helps to summarize the business logic of a certain application. EJB was provided by sun micro-systems in order to develop robust, secure applications. The EJB enumeration is a subset of the Java EE enumeration.

Figure 3.12: Enterprise Java Beans

Types of EJB - There are several types of enterprise Java beans-

1. Session bean
2. Entity bean
3. Message-driven beans

Advantages of EJB-

- EJB is an API, hence the application's build on EJB can run on Java EE web application server.
- The EJB developer focuses on solving business problems and business logic.
- Java beans are portable components that help the JAVA application assembler to formulate new applications for the already existing JavaBeans.
- EJB container helps in providing system-level services to enterprise Java beans.
- EJB contains business logic hence the front end developer can focus on the presentation of the client interface.

Disadvantages of EJB-

- The specification of EJB is pretty complicated and large.
- It creates costly and complex solutions.
- It takes time for development.
- Continuous revision of the specifications takes place.
- There are more complex cities than straight Java classes.

JDBC (Java Database Connectivity)

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access any kind of tabular data, especially relational database. It is part of Java Standard Edition platform, from Oracle Corporation. It acts as a middle layer interface between java applications and database.

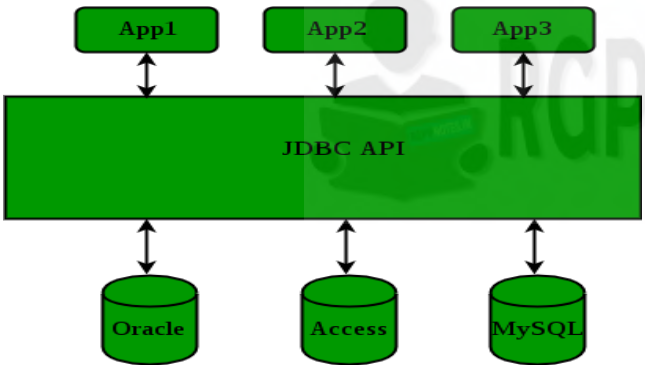


Figure 3.13: JDBC

The JDBC classes are contained in the Java Package java.sql and javax.sql.
JDBC helps you to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database.
2. Send queries and update statements to the database.
3. Retrieve and process the results received from the database in answer to your query.

DBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver.
2. Type-2 driver or Native-API driver.
3. Type-3 driver or Network Protocol driver.
4. Type-4 driver or Thin driver.

Type-1 driver or JDBC-ODBC bridge driver- It uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. Type-1 driver is also called Universal driver because it can be used to connect to any of the databases.

- As a common driver is used in order to interact with different databases, the data transferred through this driver is not so secured.
- The ODBC bridge driver is needed to be installed in individual client machines.
- Type-1 driver isn't written in java, that's why it isn't a portable driver.
- This driver software is built-in with JDK so no need to install separately.
- It is a database independent driver.

Advantage

- Easy to use.
- Can be easily connected to any database.

Disadvantage

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver nee.ds to be installed on the client machine.

Type 2 driver or Native-API driver - The Native API driver uses the client -side libraries of the database. This driver converts JDBC method calls into native calls of the database API. In order to interact with different database, this driver needs their local API, that's why data transfer is much more secure as compared to type-1 driver.

- Driver needs to be installed separately in individual client machines.
- Type-2 driver isn't written in java, that's why it isn't a portable driver.
- It is a database dependent driver.

Advantage

- Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

Type-3 driver - The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. Here all the database connectivity drivers are present in a single server, hence no need of individual client-side installation.

- Type-3 drivers are fully written in Java, hence they are portable drivers.
- Switch facility to switch over from one database to another database.

Advantage

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantage

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.

Type-4 driver native protocol driver - This driver interact directly with database. It does not require any native database library that's why it is also known as Thin Driver.

- Does not require any native library and Middleware server, so no client-side or server-side installation.
- It is fully written in Java language, hence they are portable drivers.

Advantage

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage

- Drivers depend on the Database.

Connectivity - There are 5 steps to connect any java application with the database using JDBC.

- Register the Driver class.
- Create connection.
- Create statement.
- Execute queries.
- Close connection.

JNDI

JNDI provides a common-denominator interface to many existing naming services, such as LDAP (Lightweight Directory Access Protocol) and DNS (Domain Name System). These naming services maintain a set of bindings, which relate names to objects and provide the ability to look up objects by name. JNDI allows the components in distributed applications to locate each other.

WebLogic Server JNDI - The WebLogic Server implementation of JNDI supplies methods that:

- Give clients access to the Web Logic Server naming services.
- Make objects available in the Web Logic namespace.
- Retrieve objects from the Web Logic namespace.

Each Web Logic Server cluster is supported by a replicated cluster wide JNDI tree that provides access to both replicated and pinned RMI and EJB objects. While the JNDI tree representing the cluster appears to the client as a single global tree, the tree containing the cluster-wide services is actually replicated across each Web Logic Server instance in the cluster.

JNDI in a Clustered Environment. Other Web Logic services can use the integrated naming service provided by Web Logic Server JNDI. For example, Web Logic RMI can bind and access remote objects by both standard RMI methods and JNDI methods.

JMS (Java Message Service)

JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication. It is also known as a messaging service. Messaging is a technique to communicate applications or software components. JMS is mainly used to send and receive message from one application to another.

Requirement of JMS - Generally, user sends message to application. But, if we want to send message from one application to another, we need to use JMS API. Consider a scenario, one application A is running in INDIA and another application B is running in USA. To send message from A application to B, we need to use JMS.

Advantage of JMS

- **Asynchronous:** To receive the message, client is not required to send request. Message will arrive automatically to the client.
- **Reliable:** It provides assurance that message is delivered.

Messaging Domains - There are two types of messaging domains in JMS-

1. Point-to-Point Messaging Domain- In PTP model, one message is delivered to one receiver only. Here, Queue is used as a message oriented middleware (MOM).The Queue is responsible to hold the message until receiver is ready. In PTP model, there is no timing dependency between sender and receiver.

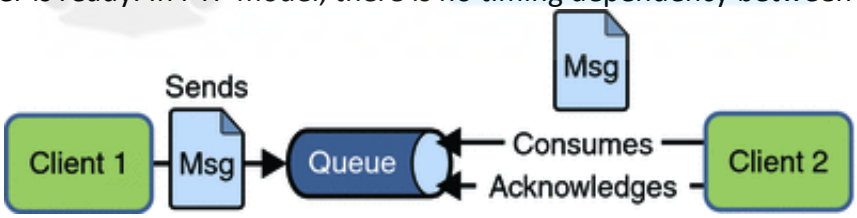


Figure 3.14: Point to Point Messaging Domain

2. Publisher Messaging Domain - In Pub/Sub model, one message is delivered to all the subscribers. It is like broadcasting. Here, Topic is used as a message-oriented middleware that is responsible to hold and deliver messages.

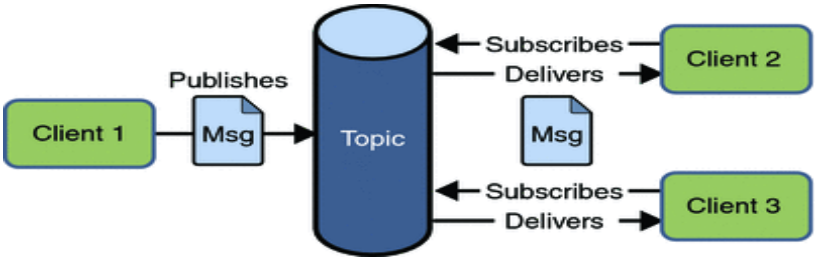


Figure 3.15: Publisher Messaging Domain

RMI (Remote Method Invocation)

Remote Method Invocation (RMI) is the standard for distributed object computing in Java. RMI enables an application to obtain a reference to an object that exists elsewhere in the network, and then invoke methods on that object as though it existed locally in the client's virtual machine. RMI specifies how distributed Java applications should operate over multiple Java virtual machines.

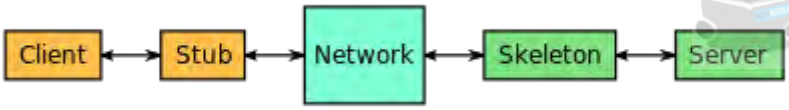


Figure 3.16: Implementation model of Java RMI

Features of Web Logic RMI

Table 3.1 Features of Web Logic implementation of RMI

S.No.	Features	Web logic RMI
1.	Overall performance	Enhanced by Web Logic RMI integration into the Web Logic Server framework, which provides underlying support for communications, scalability, management of threads and sockets, efficient garbage collection, and server-related support.
2.	Standards compliant	Compliance with the Java Platform Standard Edition 6.0 API Specification.
3.	Failover and Load balancing	Web Logic Server support for failover and load balancing of RMI objects.

CORBA (Common Object Request Broker Architecture)

CORBA is based on the Request-Response architecture. There is an object implementation on the server, which client requests to execute. The client and the server object implementation do not have any restrictions on the address space, for example the client and the server can exist in the same address space or can be located in separate address spaces on the same node or can be located on separate nodes altogether.

Object IDL interface and the Remote references are called Object References. There is a specification of the CORBA IDL Language and how it is mapped with other languages. It essentially provides a means for the interface definitions. The Proxy or a local representative for the client side is called the IDL stub; the server-side proxy is the IDL skeleton.

The proxy represents an object created on the client side, which is used for more functionality like support for Dynamic invocation. For marshalling the request and the response, the information is delivered in a canonical format defined by the IIOP protocol used for CORBA interoperability on the Internet.

IDL stub makes use of dynamic invocation interface for marshalling on the client side. Similarly on the server side, IDL Skeletons use the Dynamic Skeleton Interface for marshalling the information. The request (response) can also contain Object Reference as parameters; remote object can be passed by reference.

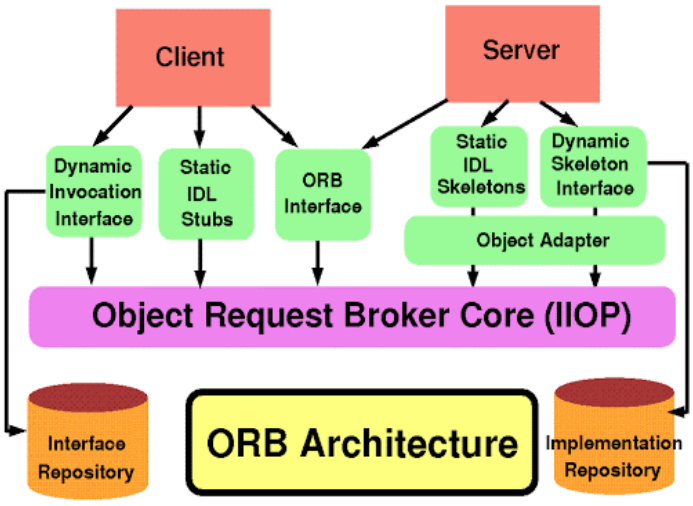


Figure 3.17: CORBA Architecture

CORBA architecture follows a Broker pattern. The ORB is used for connecting the client and the server and it acts as a broker object. The IDL stub along with DII stub performs the client-side proxy and the POA IDL skeleton (along with the DSI skeleton used for dynamic delivery) does the Server-Side proxy. Many CORBA implementations use a direct communication between the client stubs and the Server Skeleton. The Interface Repository is used for introspection (particularly in case of a dynamic invocation and dynamic delivery). The Implementation Repository is used for reactivation of servers.

Role of UML in Software Architecture

A model is a simplified representation of the system. To visualize a system, we will build various models. The subset of these models is a view. Architecture is the collection of several views. The stakeholders (end users, analysts, developers, system integrators, testers, technical writers and project managers) of a project will be interested in different views.

Architecture can be best represented as a collection five views:

Table 3.2: Description of Different View in Software Architecture

View	Stakeholder	Static Aspects	Dynamic Aspects
Use Case View	End users Analysts Tester	Use case Diagrams	Interaction Diagrams State chart Diagrams Activity Diagrams
Design View	End users	Class Diagrams	Interaction Diagrams State chart Diagrams Activity Diagrams
Implementation View	Programmers Configuration Managers	Component Diagrams	Interaction Diagrams State chart Diagrams Activity Diagrams
Process View	Integrator	Class Diagram (Active Classes)	Interaction Diagrams State chart Diagrams Activity Diagrams
Deployment View	System Engineer	Deployment Diagrams	Interaction Diagrams State chart Diagrams Activity Diagrams

RGPVNOTES.IN

Thank you for using our services. Please support us so that we can improve further and help more people.
https://www.rgpvnotes.in/support-us

If you have questions or doubts, contact us on
WhatsApp at +91-8989595022 or by email at hey@rgpvnotes.in.

For frequent updates, you can follow us on
Instagram: https://www.instagram.com/rgpvnotes.in/.