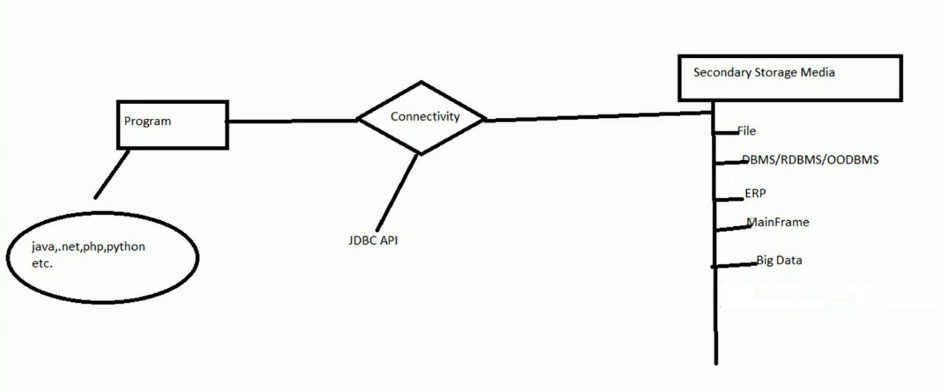
# Database

**Secondary storage media:** file, RDMS, DBMS, RADIS this are secondary storage media. bigdata, mainframe, ERP system.

**connectivity :** The process of connect the program with database is called connectivity.



# Database Connectivity :

There is many database software we use for data store. As like MySQL, oracle, MSSQL. We can use that software by install it. When we install database, all data base provides three facilities for use data.

1. **CUI (Character User Interface):** we use database by CUI in command line interface. we can create, insert, delete we with help CUI.
2. **GUI (graphical user interface):** we can also use database with GUI.
3. **API (application programming interface):** All database provides us API to use database with programming language. This API varies from database to database. We can connect programming language with database using API in three ways ----
4. using vender Specific API.
5. ODBC
6. JDC
7. ORM
8. **Vender Specific API:**

All database have many library which are use for insert, delete, update in database. We use this library for comminicate with program. This library are called vendor specific API. Every database libarbry are different form other. MYSQL API are diffarent from Oracle API. So if we need to impliment diffarent diffarent database with diffarent diffarent vender API.

Problem: if we need multiple database we need know multiple vendor specific API for every database. This is very difficult for user to know all database API.

1. **ODBC(Open DataBase Connectivity):**

Vendor specific API are solve by ODBC concept. ODBC are make by microsoft.ODBC are combination of ODBC driver and ODBC API.

1. **ODBC API:** OBDC collect all database vendor specific API and write it with a C programing language make ODBC API for various database.
2. **ODBC driver :** after make ODBC API implement are also make for program. This implemention called ODBC driver.

# Java Data Base Connectivity (JDBC)

1. JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database.
2. It is a part of JavaSE (Java Standard Edition).
3. JDBC API uses JDBC drivers to connect with the database.
4. JDBC solve platform dependent problem of ODBC.
5. JDBC all method stores in java.sql package.

There are four types of JDBC drivers:

1. JDBC-ODBC Bridge Driver,
2. Native Driver,
3. Network Protocol Driver
4. Thin Driver
5. Type-1) JDBC-ODBC bridge driver: this driver uses ODBC driver to connect to the database. This driver first converts JDBC calls into the ODBC calls. Then ODBC call the Database. this process is not use now.



1. Type-2) Native-API driver: this driver direct called the database specific vendor API. It not uses ODBC driver for connect Database. The driver converts JDBC calls into native calls of the database API. It is not written entirely in java.



1. **Type 3**) Network Protocol driver (**JDBC-Net pure Java**): The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.



1. **Type 4: - Thin driver (Pure Java):** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.so it is platform independent driver.



# JDBC API

All the JDBC API are found in java.sql package. There many class and interface in JDBC API.

1. Driver
2. Drivermanager(C)
3. connection
4. Statement
5. PreparedStatement()
6. Resultset()
7. ResultSetMetaData
8. databaseMetaData
9. callableStatement
10. SQlException(C)
11. **Driver:**

Driver is an interface that every “driver class” must implement. This is providing implementation accordingly different different venders. Driver convert one language called to other language. It is an interface of database and program.

**Mysql:** com.msql.jdbc.Driver.(mysql-driver)

1. **DriverManager:**

Driver manager is a class which do everything for database and programing. It do the following action-

1. Register the drivers with database and java program.
2. De-register the driver
3. Create connection between java program and database.
4. Return connection.

**Method of Driver manager:**

* + - 1. registerDriver ()
      2. getConnection ()

1. **RegisterDriver:** This method is use for register the driver with database. In this method we cannot change our driver name in runtime. You should use the registerDriver() method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

try {

Driver myDriver = new oracle.jdbc.driver.OracleDriver();

DriverManager.registerDriver( myDriver );

}

Catch (Exception e) {

System.out.println(e);

System.exit(1);

}

1. **getConnection () :** After the register the driver, we create connection with getConnection() method. we can have overloaded DriverManager.getConnection() methods –

**getConnection(String url):**

String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";

Connection conn = DriverManager.getConnection(URL);

**getConnection (String url, Properties prop):**

import java.util.\*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";

Properties info = new Properties( );

info.put( "user", "username" );

info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);

**getConnection(String url, String user, String password):**

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";

String USER = "username";

String PASS = "password"

Connection conn = DriverManager.getConnection(URL, USER, PASS);

1. **Connection:**

it is an interface which implementation is create by database. Every database connection interface is different. When getConnection method is connect the database it returns an object which is responsible for create a session between java program to database. This object is Connection interface type.

Connection conn = DriverManager.getConnection(URL, USER, PASS);

1. **statement:**

Once a connection is obtained, we can interact with the database. The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

**Statement stmt=con.createStatement();**

1. **Execute Method: I**n java there methods for execute Query :
2. executeQuery( )
3. executeUpdate( )
4. execute( )
5. executeBatch( )
6. **executeQuery():** The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

ResultSet rs = stmt.executeQuery("select \* from emp");

1. **executeUpdate( ):** **the executeUpdate( )** is used to execute specified query, it may be created, drop, insert, update, delete etc.

ResultSet rs = stmt.executeUpdate ("select \* from emp");

1. **execute( ):** **execute()** is used to execute queries that may return multiple results.

ResultSet rs = stmt.execute ("select \* from emp");

1. **executeBatch( ):** **executeBatch( )** is used to execute batch of commands.

ResultSet rs = stmt.executeBatch("select \* from emp");

1. **Close the connection object:**

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

con.close();

**Example of Oracle Database:**

**import** java.sql.\*;

**class** OracleCon{

**public** **static** **void** main(String args[]){

**try**{

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521","system","oracle");

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**( rs.next() ){

System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

con.close();

}

}**catch**(Exception e){

System.out.println(e);

}

}

}

# Java Database Connectivity

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

1. Create connection
2. Create statement
3. Execute method
4. Close connection

**Import JDBC Packages:**

import java.sql.\* ; // for standard JDBC programs

import java.math.\* ; // for BigDecimal and BigInteger support

To use JDBC we first register the driver with database. For registration we use various method.

1. **Class.forName():** The most common approach to register a driver is to use Java's

Class.forName() method, to dynamically load the driver's class file into memory, which automatically registers it. We can change the driver name in runtime. This method is preferable because it allows you to make the driver registration configurable and portable.

|  |  |  |
| --- | --- | --- |
| RDBMS | JDBC driver name | URL format |
| MySQL | com.mysql.jdbc.Driver | jdbc:mysql://hostname/ databaseName |
| ORACLE | oracle.jdbc.driver.OracleDriver | jdbc:oracle:thin:@hostname:port Number:databaseName |
| DB2 | COM.ibm.db2.jdbc.net.DB2Driver | jdbc:db2:hostname:port Number/databaseName |
| Sybase | com.sybase.jdbc.SybDriver | jdbc:sybase:Tds:hostname: port Number/databaseName |

try {

Class.forName("oracle.jdbc.driver.OracleDriver");

}

catch(ClassNotFoundException ex) {

System.out.println("Error: unable to load driver class!");

System.exit(1);

}

1. **DriverManager.registerDriver():** Next approach you can use to register a driver, is to use the static DriverManager.registerDriver() method. In this method we cannot change our driver name in runtime. You should use the registerDriver() method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

try {

Driver myDriver = new oracle.jdbc.driver.OracleDriver();

DriverManager.registerDriver( myDriver );

}

catch(ClassNotFoundException ex) {

System.out.println("Error: unable to load driver class!");

System.exit(1);

}

# PreparedStatement and CallableStatement

1. **PreparedStatement:**

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query. The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

String sql="insert into emp values( ?, ?, ?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

**Methods of PreparedStatement interface:**  The important methods of PreparedStatement interface are given below:

1. setInt(int paramIndex, int value)
2. setString(int paramIndex, String value)
3. setFloat(int paramIndex, float value)
4. setDouble(int paramIndex, double value)
5. executeUpdate()
6. ResultSet executeQuery()

code:

import java.sql.\*;

class InsertPrepared{

public static void main(String args[]){

try{

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con = DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:xe", "system", "oracle" );

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");

stmt.setInt(1,101);//1 specifies the first parameter in the query

stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();

System.out.println(i+" records inserted");

con.close();

}catch(Exception e){

 System.out.println(e);

}

}

}

1. **CallableStatement :**

# Result Sets

A ResultSet is a Java object that contains the results of executing an SQL query. In other words, it contains the rows that satisfy the conditions of the query. The data stored in a ResultSet object is retrieved through a set of get methods that allows access to the various columns of the current row.

**import** java.sql.\*;

**class** FetchRecord{

**public** **static** **void** main(String args[])**throws** Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

Statement stmt=con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);

ResultSet rs=stmt.executeQuery("select \* from emp765");

rs.absolute(3);

System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();

} }

Methods of DatabaseMetaData interface:

1. public boolean next():
2. public boolean previous():
3. public boolean first():
4. public boolean last():
5. public boolean absolute(int row):
6. public boolean relative(int row):
7. public int getInt(int columnIndex):
8. public int getInt(String columnName):
9. public String getString(int columnIndex):
10. public String getString(String columnName):

# Java Enterprise Edition

The **Java EE** stands for **Java Enterprise Edition**, which was earlier known as J2EE and is currently known as Jakarta EE. It is a set of specifications wrapping around Java SE (Standard Edition). The Java EE provides a platform for developers with enterprise features such as distributed computing and web services.

**WebApp:**

1. Java Servlets
2. JSP (Java Servlet Pages

**Enterprise Application:**

1. RMI
2. EJB
3. WEB SERVICE

# Servlet

What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

**Web Terminology:**

1. Website: static vs dynamic
2. HTTP
3. HTTP Requests
4. Get vs Post
5. Container
6. Server: Web vs Application
7. Content Type

CGI (Common Gateway Interface):

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



**Disadvantages of CGI:**

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Servlet

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Advantages of Servlet



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. Better performance: because it creates a thread for each request, not process.
2. Portability: because it uses Java language.
3. Robust: JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. Secure: because it uses java language.

Servlet API

You need to use Servlet API to create servlets. There are two packages that you must remember while using API.

**1)javax.servlet. package:**

**Interfaces:**

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

**Classes:**

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

**2)javax.servlet.http package:**

**Interfaces :**

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

**Classes :**

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

**Hierarchy of servlet interface:**

Diagram

Description automatically generated

**How to create servlet**

**Web Server**: It is also known as HTTP Server, it can handle HTTP Requests send by client and responds the request with an HTTP Response.

**Web Container**: Also known as Servlet Container and Servlet Engine. It is a part of Web Server that interacts with Servlets. This is the main component of Web Server that manages the life cycle of Servlets.

**Servlet life cycle method:**

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

When the web server (e.g. Apache Tomcat) starts up, the servlet container deploy and loads all the servlets. Once all the Servlet classes loaded, the servlet container creates instances of each servlet class. Servlet container creates only once instance per servlet class and all the requests to the servlet are executed on the same servlet instance.

**init() method:**

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

**public** **void** init(ServletConfig config){

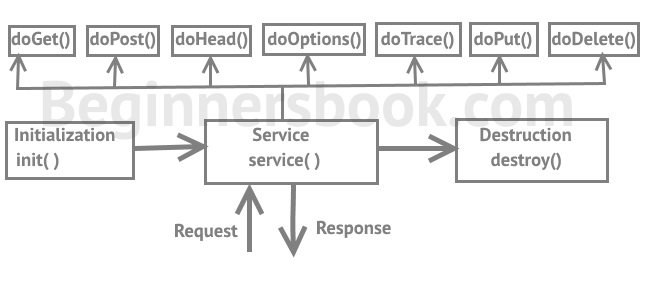
**this**.config=config;

System.out.println("servlet is initialized");

}

**The service() Method:**

The service() method is the main method to perform the actual task. the web container calls the service method each time when request for the servlet is received. Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.



Example:

**public** **void** service(ServletRequest req,ServletResponse res)  **throws** IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello simple servlet</b>");

out.print("</body></html>");

}

**The destroy() Method:**

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

**public** **void** destroy(){

System.out.println("servlet is destroyed");

}

Way of Creating servlet

the servlet can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class
3. By inheriting HttpServlet class
4. **By implementing Servlet interface:**

Here is the list of methods available in Servlet interface.

1. **void destroy():** This method is called by Servlet container at the end of servlet life cycle. Unlike

service() method that gets called multiple times during life cycle, this method is called only once by Servlet container during the complete life cycle. Once destroy() method is called the servlet container does not call the service() method for that servlet.

1. **void init(ServletConfig config):** When Servlet container starts up (that happens when the web server starts up) it loads all the servlets and instantiates them. After this init() method gets called for each instantiated servlet, this method initializes the servlet.
2. **void service(ServletRequest req, ServletResponse res):** This is the only method that is called multiple times during servlet life cycle. This methods serves the client request, it is called every time the server receives a request.
3. **ServletConfig getServletConfig():** Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.
4. **java.lang.String getServletInfo():** Returns information about the servlet, such as author, version, and copyright.

**Example:**

import java.io.\*;

import javax.servlet.\*;

public class DemoServlet implements Servlet{

ServletConfig config=null;

public void init(ServletConfig config){

this.config=config;

System.out.println("Initialization complete");

}

public void service(ServletRequest req,ServletResponse res) throws IOException,ServletException{

res.setContentType("text/html");

PrintWriter pwriter=res.getWriter();

pwriter.print("<html>");

pwriter.print("<body>");

pwriter.print("<h1>Servlet Example Program</h1>");

pwriter.print("</body>");

pwriter.print("</html>");

}

public void destroy(){

System.out.println("servlet life cycle finished");

}

public ServletConfig getServletConfig(){

return config;

}

public String getServletInfo(){

return "A Demo program written by Chaitanya";

}

}

1. **By inheriting GenericServlet class:**
2. GenericServlet class implements Servlet, ServletConfig and Serializable interfaces.
3. It provides the implementation of all the methods of these interfaces except the service method.
4. GenericServlet class can handle any type of request, so it is protocol independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

**Methods of GenericServlet class:**

1. **public void init (ServletConfig config) :** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response):** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy ()**: is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig():** returns the object of ServletConfig.
5. **public String getServletInfo():** returns information about servlet such as writer, copyright, version
6. **public void init() :** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext():** returns the object of ServletContext.
8. **public String getInitParameter(String name):** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames():** returns all the parameters defined in the web.xml file.
10. **public String getServletName():** returns the name of the servlet object.
11. **public void log(String msg):** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)**: writes the explanatory message in the servlet log file and a stack trace.

Example:

**import** java.io.\*;

**import** javax.servlet.\*;

**public** **class** First **extends GenericServlet**{

**public** **void** service(ServletRequest req,ServletResponse res)  **throws** IOException,ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();

out.print("<html><body>");

out.print("<b>hello generic servlet</b>");

out.print("</body></html>");

}

}

1. **By inheriting HttpServlet class**

The HttpServlet class extends the GenericServlet class and implements Serializable interface. HTTP Servlet doesn’t override the service() method. Instead it overrides the doGet() method or doPost() method or both.  The doGet() method is used for getting the information from server while the doPost() method is used for sending information to the server

**Methods of HttpServlet class:**

1. **public void service(ServletRequest req, ServletResponse res):** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res):** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res):** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)**: handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res):** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res):** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res):** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

**Example:**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class ExampleHttpServlet extends HttpServlet

{

private String mymsg;

public void init() throws ServletException

{

mymsg = "Http Servlet Demo";

}

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<h1>" + mymsg + "</h1>");

out.println("<p>" + "Hello Friends!" + "</p>");

}

public void destroy()

{

// Leaving empty. Use this if you want to perform

//something at the end of Servlet life cycle.

}

}