**JNDI**

**JNDI**: (Java Naming and Directory Interface)

We use JNDI to develop a java application to interact with directory servers.

Similar to Database servers we have directory servers. Directory servers also use to store the data.

As part of Database server we store the data in the form of relational records as part of directory server. We store the data in the form of objects.

\***Difference between Database server and Directory servers**:

Directory servers give the best performance if we want to the store the data once and retrieve the data for multiple times.

As part of Database servers we can store huge amount of data. Directory servers are not meant to store huge amount of data. They can store only small amount of data. (Yahoo, Google and etc…)

To interact with database servers where using query language. To interact with directory server we have to use the predefined functions/methods.

Directory servers can’t store the data permanently. These servers are meant to store the data temporally. In most of the project we use both database servers and directory servers.

In most of the projects we write the java code to store the data into Database server permanently. Once if data into data is stored in server we write another java program to retrieve required data represent in the form of object and store it in directory server.

JNDI API is mainly used to develop a java application to interact with directory servers. The following is architecture of JNDI.

JAVA Application

Directory server

JNDI Driver software

They are so many directory servers are available some of them all.

1. LDAP (Light weight Directory Access Protocol) from open source apache.
2. ADS (Active Directory Server) from Micro soft
3. NDS (Novel Directory Server)
4. DNS (Domain Naming Server)……etc.

We no need to install directory servers separately. Now way days all the JEE servers are integrated directory servers. When we install JEE servers we get the directory server also. They following some of the JEE servers.

**Ex**:

1. Weblogic
2. JBOSS
3. Tomcat
4. Websphere
5. Resin

When install weblogic server it got install in a home directory called as BEA.

To work with weblogic server we must configure the weblogic server. That meaning of weblogic server is placing all the required files in a folder.

To work with weblogic server we have to configure the weblogic server (or) we have to create a Domain.

The meaning of creating a domain is create the folder and copy all the required files into it.

\***Procedure to create domain in weblogic server**:

Start the weblogic server configuration wizard (start -> All Programs -> oracle weblogic -> weblogic10gR3 -> tools -> click on configuration wizard).

The above step launches a window how’s name is oracle weblogic configuration wizard. From this select an option creates a new weblogic domain and click on NEXT.

Select an option generates a domain configured automatically and click on NEXT.

Enter the username and password which act as on administrator.

Select the development mode and available JDK. Choose the customization as NO and click on create BUTTON.

To start the weblogic server oracle guys are provide CMD files and SH files. We use start weblogic CMD to start the server.

Now a days people are not give separate client software the client software is integrated with the server in the form of web based application.

The advantage of this approach is we now to install the client software separately. We can access the client software by taking the help of BROWSER. To access oracle client software where using the following URL.

**Ex**:

<http://localhost:8080/apex/>

Browser 1

Web application

Server S/W

Browser 2

Server

The following is URL which is used to access admin console of web logic server.

**Ex**:

<http://localhost:7001/console/>

The default port number of weblogic server is 7001 and the default server name is “admin”. We can change the default port number and server name. We can use customization option to use our own port number as well as our own server names. In the projects most of the time we change the settings according to our project requirements.

Sun micro system as released JNDI API to develop a java application to communicate with directory server. Sun micro system as released all the classes and interfaces as part of javax.naming package. The most important interface is context. The most important class is initial context. Initial context class provide the implementation of context interface.

Context

Interface

Initial context

Class

As a programmer we are responsible to develop a java application to create an object and store the object into directory server Update, Delete and Retrieve objects directory server.

To develop a JNDI application we have to supply the following four values.

JNDI

INITIAL\_CONTEXT\_FACTORY

PROVIDER\_URL

SECURITY-PRINCIPAL

SECURITY\_CEDECTIALS

JDBC

driver class

url

username

password

The following are the steps to develop JNDI Application.

1. Create Hash table.
2. Store the details in the Hash table using keys and values.
3. Get the connection to directory server by supplying Hash table object as input.
4. Call the methods bind/unbind/rebind/lookup methods to perform the operations with Directory server.

\*Develop a JNDI application to store a string object into directory server. To store an object in the directory server we use a method bind.

**Syntax**: void bind(key, Object)

**Ex**:

import java.util.\*;

import javax.naming.\*;

public class StoreObject{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context ic = new InitialContext(h);

String name = "Raju";

ic.bind("uname",name);

}

}

The bind() method convert the object into super class object and stored it in directory server. In the above example string object is stored into directory server. By converting it into super\_class\_object object.

\*The following example demonistrate how to retrieve the data from directory server.

import java.util.\*;

import javax.naming.\*;

public class RetrieveObject{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext(h);

Object o = c.lookup(“uname”);

String s = (String)o;

System.out.println(s);

}

}

With key we are using to searching if it is not available. It got a error message NameNotFoundException.

\*We using a method rebind to update a Record a directory server.

import java.util.\*;

import javax.naming.\*;

public class UpdateRecord{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext(h);

c.rebind(“uname”,”Naveen”);

}

}

\*Unbind method is used to delete the Record a directory server.

import java.util.\*;

import javax.naming.\*;

public class DeleteRecord{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext(h);

c.unbind(“uname”);

}

}

It is not recommended to store data directory into initial context. This is because of if we store directory the search operation takes time. It always recommended store the data into sub context (sub context is like a folder).

Initial Context

Sub Context

Object

Sub Context

\*To create the sub context use a method.

import java.util.\*;

import javax.naming.\*;

public class CreateSubContext{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext();

c.createSubcontext(“Btech”);

c.createSubcontext(“mca”);

}

}

\*To create the sub context inside the sub context we can create the separate dot.

import java.util.\*;

import javax.naming.\*;

public class CreateSubContext1{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext();

c.createSubcontext(“Btech.1styear.mech”);

c.createSubcontext(“Btech.1styear.it”);

c.createSubcontext(“Btech.1styear.csc”);

}

}

\*if we want to create a sub context in another sub context we need to make sure that the base sub context is available.

import java.util.\*;

import javax.naming.\*;

public class CreateSubContext{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext();

c.createSubcontext(“btech”);

c.createSubcontext(“btech.1styear”);

c.createSubcontext(“Btech.1styear.mech”);

c.createSubcontext(“Btech.1styear.it”);

c.createSubcontext(“Btech.1styear.csc”);

}

}

\*To store an object into a specific context we have to specify absolute context path.

import java.util.\*;

import javax.naming.\*;

public class CreateBind{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context ic = new InitialContext();

ic.bind(“betch.1styear.csc.ramesh”,”abc”);

}

}

\*To remove sub context we use a method destroy sub context.

import java.util.\*;

import javax.naming.\*;

public class CreateBind{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext();

c.destroySubcontext(“betch.1styear.csc”);

}

}

\*\*\*\***Connection pool**:

When we develop a java application to get the connection from Database server. By using DriverManager.getConnection we always get physical connections.

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

Physical

If we got a physical connection and we are not closing the connection after we above work the other application can not use the same connections.

Connection pool is java program which manages the connections of Database server. Connection pool contains set of connections.

There are so many connection pool programs are available some of them are.

1. DBCP (Database connection pool)
2. C3p0 connection pool
3. Weblogic connection pool
4. JBOSS connection pool
5. Tomcat connection pool ……. Etc.

DB Server

Physical logical

Connection pool java application

Generally connections pool program is released in the form of jar files. The jar file contains set of class.

The meaning of using connection pool program is creating the object class and supply Driver class, url, username, password and initial capacity.

\*The following is an example of using DBCP connection pool program.

import org.apache.commons.dbcp.\*;

import java.io.\*;

import java.sql.\*;

public class DBC3P0{

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

BasicDataSource bds = new BasicDataSource();

bds.setDriverClassName(“oracle.jdbc.driver.OracleDriver”);

bds.setUrl(“jdbc:oracle:thin:@localhost:1521:xe”);

bds.setUsername(“system”);

bds.setPassword(“malli”);

bds.setInitialSize(3);

Connection con1 = bds.getConnection();

System.out.println(con1);

System.in.read();

System.in.read();

Connection con2 = bds.getConnection();

System.out.println(con2);

System.in.read();

System.in.read();

Connection con3 = bds.getConnection();

System.out.println(con3);

System.in.read();

System.in.read();

Connection con4 = bds.getConnection();

System.out.println(con4);

System.in.read();

System.in.read();

Connection con5 = bds.getConnection();

System.out.println(con5);

System.in.read();

System.in.read();

}

}

\***Using C3P0 connection pool**:

The following is an example of using c3p0 connection pool.

import com.mchange.v2.c2p0.\*;

import java.sql.\*;

public class DBc3p0Connection{

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

ComboPoolDataSource cpds = new ComboPoolDataSource();

cpds.setDriverClass(“oracle.jdbc.driver.OracleDriver”);

cpds.setJdbcUrl(“jdbc:oracle:thin:@localhost:1521:xe”);

cpds.setUser(“system”);

cpds.setPassword(“malli”);

cpds.setInitialPoolSize(3);

Connection con = cpds.getConnection();

System.out.println(“Connection:” +con);

System.in.read();

System.in.read();

}

}

\***Procedure to work with weblogic connection pool**:

MyPool(ds object)

Data source

DB Server

Physical

Java application

(MyApp.java)

Weblogic server

\***Procedure to configure weblogic connection pool**:

As part of the admin console go to Domain Structure services JDB

Data Source.

This will list out all available Data sources. To create the new Data Source select

NEW

button. Supply the following details for JDBC Data Source properties.

**Ex**:

NEXT

Name:

JNDI:

Database Type:

Database Driver:

BMS Data Source

MyPool

oracle

Oracle thin driver

Name is used to identify that available Data Source. JNDI name is used to store the data source object into directory server. DB type and DB driver are used to which DB Server it connects.

It will display the transaction options by default “one-phase commit” option are coming. Supply the following connection properties.

**Ex**:

xe

Database Name:

Host Name:

Port No:

DB User Name:

Conform Password:

malli

system

1521

local host

NEXT

When we fill the above form it will display driver class url, username and password. To check whether. The details are valid or not we use on option “Test Configuration”

Associate the connection pool program with admin server and click on FINISH button.

To change the initial capacity and capacity increment select the Data Source connection pool tab.

When we Run the above connection pool program it is acquired five connections and the information about the connections is stored in Data Source object and it is stored a Directory server.

\*The following java program get the connection from connection pool.

import java.util.\*;

import javax.naming.\*;

import java.sql.\*;

import javax.sql.\*;

public class DBConnect2{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext(h);

Object o = c.lookup(“mypool”);

DataSource ds = (DataSource)o;

Connection con = ds.getConnection();

System.out.println(con);

System.in.read();

System.in.read();

}

}

// set CLASSPATH=wlclient.jar;.;

// set CLASSPATH=c:\bea\wlserver\_10.3\server\lib\weblogic.jar;.;

// echo %CLASSPATH%

// echo %PATH%

//c:\bea\user\_projects\domains\mydomain\bin\setDomainEnv.cmd

// c:\bea\user\_projects\domains\mydomain\cd\

To set the class path to “weblogic.jar;.;” we can use “set DomainEnv.cmd”.

\*Retrieve the Records program?

import java.util.\*;

import javax.naming.\*;

import java.sql.\*;

import javax.sql.\*;

public class RetrieveRecords1{

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

Hashtable h = new Hashtable();

h.put(Context.INITIAL\_CONTEXT\_FACTORY,"weblogic.jndi.WLInitialContextFactory");

h.put(Context.PROVIDER\_URL,"t3://lacalhost:7001/");

h.put(Context.SECURITY\_PRINCIPAL,"admin");

h.put(Context.SECURITY\_CREDENTIALS,"inetsolv");

Context c = new InitialContext(h);

Object o = c.lookup(“mypool”);

DataSource ds = (DataSource)o;

Connection con = ds.getConnection();

Statement stmt = con.createStatement();

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

while(rs.next()){

System.out.println(rs.getString(1));

System.out.println(rs.getString(2));

System.out.println(rs.getString(3));

}

}

}

By default weblogic server is uses java1.6 some times we may get error message saying that “unsupported class version” as part of connection pool program. The problem this is to compile the java program by using one version and Run the program another version.

To resolve this problem after “setDomainEnv.cmd” is executed recompile the program and run it.

If the maximum capacity connection pool is ‘10’ and trying to get more number of connection to we getting an Exception is “PoolLimitSqlException”.

When we use the connection pool also it’s mandatory that we must close the connection. In con.close() method if it is a logical connection it will be return to connection pool.