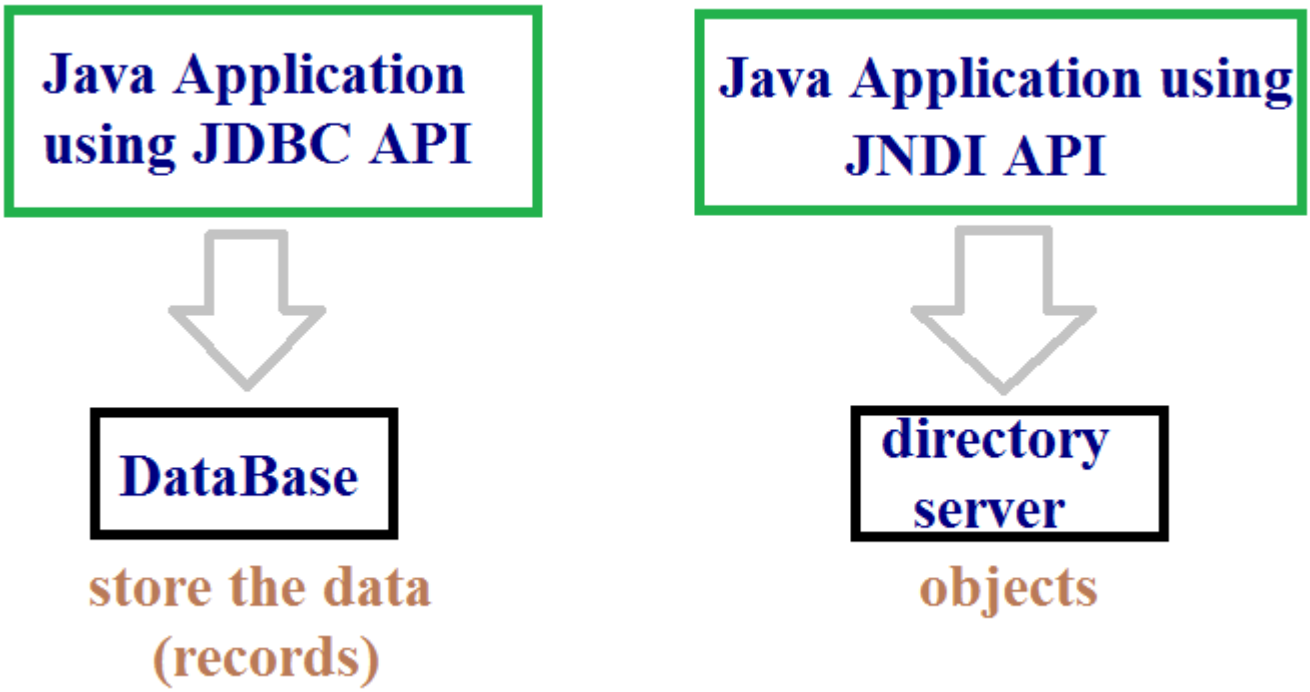
Java Naming and Directory Interface (JNDI)

Agenda :

1. Introduction

Introduction :



- JNDI API is used to communicate with directory server.
- Similar to the Database server, directory server is used to store the data.
- As part of the directory server the data will be store in the form of objects.
- We can choose directory server if insertion operations are very less and select operations are more number of times.

- We can store the data into database fast reader than database server.
- As part of directory server we can store very small amount of data.
- The SQL queries can not be applied of directory server.
- As a java programmer we can write a java application which can talks to directory server.
- If a java program need to talk directory server we have to use JNDI API.
- A java application which uses JNDI API can communicate with directory server.

There are different directory servers are available in market , they are

1. **LDAP** LightWeight Directory Access Protocol (open source)
2. **ADS** Active Directory Server from Microsoft
3. **NDS** Novel Directory Server from Novel

- Our java application which uses JNDI Api can communicate with any directory server.
- Generally directory servers are part of J2EE servers.If we buy the J2EE servers we get the directory servers.

The following are most popular J2EE servers

1. Weblogic
2. Websphere
3. JBoss
4. Resin

- LDAP directory server is applied with Weblogic server.
- If we start a Weblogic server by default the directory server will be started.

Procedure to configure Weblogic 10.3 version (pure java s/w) :

Step 1 :	start --> All Programs --> Oracle Weblogic --> weblogic 10g --> Tools --> config.wizard (bea-wlserver10.0/common/bin-config.sh)
Step 2 :	The above step we lanch the dialog box **OracleWeblogic Configuration Wizard** . In this dialog box choose create a new weblogic domain radio button and click on next.
Step 3 :	Choose generate a domain configured automatically Radio button and click on next.
Step 4 :	Provide the user name and password
Step 5 :	Choose to available SDK and click on next button. Provide the domain name (myDomain) and click on create button.

The above steps will create a folder with name myDomain in user_projects.
http://localhost:7001/console/

Once if a configure the WebLogic server to start the Weblogic server we need we need to double click on **startweblogic.cmd** or **startweblogic.sh**

To access the Weblogic server perform the following steps :

1. Open the Internet Explorer (any browser)
2. Go to the URL and type the
   url **http://localhost:7001/console/**
   provide userName, password

   Environment --> server --> Admin Server --> view JNDI Tree
3. All the classes and interfaces of JNDI API are available as part of **javax.naming** package
4. The most important interface of JNDI is **Context**
5. The most important class name is **InitialContext**
6. In case of weblogic server LDAP directory server integrated with our server.
7. If we start the weblogic server internally its going to start the directory server also.
8. We can develop java application which interacts with directory server.
9. Weblogic people as provided the implementation of JNDI API , this implementation is called as **JNDI driver**
10. The JNDI driver is available with weblogic server in the form of **jar** files
    weblogic 8.1 ---> weblogic.jar
    weblogic 8.3 ---> wlclient.jar
    (bea/wlserver 10.3/server/lib)

Develop JNDI application which can get the InitialContext object :

```
package com.jndi;
```

```
import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JndiDemo {
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://localhost:7001");
h.put(Context.SECURITY_CREDENTIALS, "weblogic");
h.put(Context.SECURITY_PRINCIPAL, "weblogic");

InitialContext ic=new InitialContext(h);
System.out.println("Object is : "+ ic);
}
}
```

Note : To run the above java program we have to set **wlclient.war** in the class path.

Procedure to develop java application which interacts with the directory server :

- To establish connection with directory server, we need the JNDI driver **class name** , which provide the implementation of **Context** interface, **url**, **user name** of weblogic server and **password** of weblogic server.
- We need to place all these details inside a Hashtable object.

- We need to supply the Hashtable object has input to InitialContext constructor.

Once if we get the InitialContext object, we can perform the following operations :

Admin server

1. Create object - **bind()** method
2. Remove object - **unbind()** method
3. Update object - **rebind()** method
4. Retrieve object - **lookup()** method

We have to use the following key in the Hashtable :

javax.naming package

context.INITIAL_CONTEXT_FACTORY
context.PROVIDER_URL
context.SECURITY_PRINCIPAL
context.SECURITY_CREDENTIALS

Develop a JNDI application which add a name object to the directory server :

```
package com.jndi;

import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JndiCreateObject {
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://localhost:7001");
h.put(Context.SECURITY_CREDENTIALS, "weblogic");
h.put(Context.SECURITY_PRINCIPAL, "weblogic");

InitialContext ic=new InitialContext(h);

String name="Ashok JavaProgrammer";
ic.bind("same", name);
System.out.println("object is added to directory
server");

}
}
```

Develop a JNDI application which adds an ArrayList object to Directory server :

To store the ArrayList object use the key name as **emplist**

```
package com.jndi;

import java.util.ArrayList;
import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JndiAddEmpObject {
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://localhost:7001");
h.put(Context.SECURITY_CREDENTIALS, "weblogic");
h.put(Context.SECURITY_PRINCIPAL, "weblogic");

InitialContext ic=new InitialContext(h);

ArrayList  a=new ArrayList();
a.add("EmpOne");
a.add("EmpTwo");

ic.bind("emplist", a);
System.out.println("object is added to directory
server");

}
}
```

- When we use the bind() method it adds the object to the directory server.
- The bind() method converts the object into super class reference object.

Ex 1 :

```
String name="Sai Java";
Object o=name;
```

Ex 2 :

```
ArrayList a = new ArrayList();
Object o=a;
```

// ArrayList is type casted to Object class

Develop a JNDI application which delete an object stored in Directory server :

Delete object which is associated to **sname** key

```
package com.jndi;

import java.util.ArrayList;
```

```
import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JndiDeleteObject {
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://localhost:7001");
h.put(Context.SECURITY_CREDENTIALS, "weblogic");
h.put(Context.SECURITY_PRINCIPAL, "weblogic");

InitialContext ic=new InitialContext(h);
ic.unbind("sname");
System.out.println("object is removed from
directory server");

}
}
```

If we try to add different objects with the same key the directory server will not allow.It will give the Exception i.e., **javax.naming.NameAlreadyBoundException**

Develop a JNDI application, update an object stored in Directory server :

Update the object which is associated to 'sname' key

```
package com.jndi;

import java.util.Hashtable;

import javax.naming.Context;
```

```
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JndiUpdateObject {
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://localhost:7001");
h.put(Context.SECURITY_CREDENTIALS, "weblogic");
h.put(Context.SECURITY_PRINCIPAL, "weblogic");

Context ic=new InitialContext(h);
ic.rebind("sname", "Student name modified");
System.out.println("object is updated in directory
server");

}
}
```

Develop a JNDI application which retrieves an object stored in Directory server :

The key name of the object is 'sname'

```
package com.jndi;

import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JndiRetrieveObject {
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://localhost:7001");
h.put(Context.SECURITY_CREDENTIALS, "weblogic");
h.put(Context.SECURITY_PRINCIPAL, "weblogic");

Context ic=new InitialContext(h);
Object o=ic.lookup("sname");
String s=(String)o;
System.out.println("object is retrieved : " + s );

}
}
```

- When we add an object to the directory server the object will added to InitialContext directly.
- If we want to place all the related objects in one group then create a subcontext.

Note : For a java programmer subcontext is some thing similar to package

```
ic.createSubcontext("Student");
ic.createSubcontext("Student.MCA");
ic.destroySubcontext("Student");
```

The above code create a subcontext under the InitialContext

Note : We can store the objects inside the subcontext.

Connection Pool Technique :

- Without this technique will not develop any projects.

- Every DB Administrator will configure , how many max number of connections can be given by DB server.
- When the Administrator configure that it store inside a file
  oracle --> config --> init.ora // open with notepad
- Every database server will having limits and issue the connections to the applications.
- Connection pool is a java program, which can maintain set of connections with him, The connection pool program aquires the connection from the DB and holds those connections.

Procedure to configure the connections through in the Weblogic server :

Step 1 :	To configure the Connection pool, we have to start the JEE server (start Weblogic server) **Note :** Connection pool is a program which is available as part of J2EE server.
Step 2 :	Access Weblogic server Admin console using the following 'URL' **http://localhost:7001/console/**
Step 3 :	We need to configure DataSource , to configure the DataSource from the domain structure menu goto services option. From the services option choose JDBC service. From the JDBC service click on DataSource **Note :** Once we click on DataSource we get a webpage which gives the summary of JDBC DataSource.

	If any DataSources already configure we can use those DataSources also.
Step 4 :	To create the new DataSource we have to click on new button from summary of JDBC DataSources wep page.
Step 5 :	We have to provide the following details, we create new DataSource page. **Name : MyDs** **JNDI Name : myPool** **Database Type : Oracle** **Database Server : Oracle Thin Driver** After fill the above form click on next button.
Step 6 :	Once we click the next button , it displays transactions option. **Note :** do not change any option and click on next button.
Step 7 :	*We need to provide the following connection properties* **DataBase Name : xe** **Host Name : localhost** **Port : 1521** **Database userName : lms** **Password : ***** **Confirm Password : *****
Step 8 :	After filling the above form, to click next button. Once we click the before next button , Test database connection , in this screen we can check the connection pool program is able to communicate with Database server or not.

	To test this click on button **Test Configuration** other wise click on next button. Once we click on next button we get the screen to associate the connection pool with the server.
Step 9 :	Select the name of the server to which we want to associate the connection pool and click on Finish.

- In the Weblogic server perform all the above steps , it creates the connection pool in **initial capacity as 1**.
- To modified capacity go to **services** and goto **DataSources** option, and click on **DataSourceName** then goto **connection pool tab** and modified it.

Start --> Oracle 10g --> goto DB Home page --> Administration --> Monitor --> sessions

Weblogic --> services --> JDBC --> Data Sources --> new

Procedure to get a connection from Connection Pool :

- Write the JNDI code to get DataSource object from directory server.
- From the DataSource object get the connection.
  Note : If we try to get the connection object from DataSource , that object is type logical.

1. We can get the connection from Weblogic server.
2. create the Statement object.
3. Execute the Query.

4. Close the connection.

DataSource interface is part of **javax.sql** package.

```
package com.jndi;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class DBConnect {
public static void main(String args[])
             throws NamingException, SQLException{

Hashtable h=new Hashtable();
h.put(Context.INITIAL_CONTEXT_FACTORY,

     "weblogic.jndi.WLInitialContextFactory");
h.put(Context.PROVIDER_URL,
"t3://localhost:7001");
h.put(Context.SECURITY_CREDENTIALS, "weblogic");
h.put(Context.SECURITY_PRINCIPAL, "weblogic");

Context ic=new InitialContext(h);
Object o=ic.lookup("myPool");
DataSource ds=(DataSource)o;
Connection con=ds.getConnection();
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from
product");

while(rs.next()){
System.out.println(rs.getString(1));
```

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

}
}
```

- To run above connection pool object, we have create a command **setDomainEnv.cmd** , this file is available inside bin directory at our domain (myDomain/bin)
- In connection pool program, if we write the code to close the connections using **con.close()** , then connection will be return back to the Connection pool.