JOD Snapshot Quick Start Guide

Table of contents

1. Introduction

In this quick tutorial I will guide you through some little JOD Snapshot code. At the end you'll be able to read a simple object from a directory

Obviously you will need a LDAP directory server. If you don't have one I suggest Apache Directory Server. It is very simple to get it up and running. A LDAP browser is very useful, try JXplorer.

Before starting put snapshot.jar in your classpath and do so for all the following JOD Snapshot dependencies:

- asm.jar
- asm-attrs.jar
- cglib-2.1.3.jar
- commons-beanutils.jar
- commons-logging-1.0.4.jar
- log4j-1-2-11.jar

You can find them in the JOD Snapshot distribution under /lib directory.

2. Configuring directory access

The first thing to do is to create an xml configuration file where you will write directory access informations. Call it snapshot.cfg.xml and put it in the default package.

The content of this file should be similar to the following

```
<?xml version='1.0' encoding='utf-8'?>
<snapshot-configuration>

        <property name="jndiUrl">ldap://localhost:10389</property>
        <property name="user">uid=admin,ou=system</property>
        <property name="pwd">secret</property>
        <property name="searchBase">dc=example,dc=com</property>

</snapshot-configuration>
```

Here you have to define four parameters:

jndiUrl
The url of your ldap server
user
The username you have to provide to connect to the server
pwd
Your user password
searchBase
The distinguished name of the ldap node from which all your searches should

start

The values provided here are the default for an installation of Apache Directory Server

3. Writing the user bean

In this little example we will read from the directory some user informations, so we have to write a User class with the properties that will contain all these informations.

```
package example;

public class User {
        private String id;
        private String name;
        private User manager;
        private Collection mailAddresses;

        public String getId() {
                return id;
        }

        public void setId(String id) {
                this.id = (String)id;
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public Collection getMailAddresses() {
                return mailAddresses;
        }

        public void setMailAddresses(Collection mailAdresses) {
                this.mailAddresses = mailAdresses;
        }

        public User getManager() {
                return manager;
        }

        public void setManager(User manager) {
                this.manager = manager;
        }

        public String toString() {
                return (String)getId();
        }

        public boolean equals(Object obj) {
                return obj instanceof User
                    && ((User)obj).getId().equals(getId());
        }
```

```
        public int hashCode() {
                return getId().hashCode();
        }

}
```

The id property will contain the distinguished name of the user while the other properties will contain other information taken from the directory:

- The name property will be the "common name" of the user contained in the "cn" LDAP attribute
- The manager property is the "boss" of our user, the "manager" LDAP attribute contains the distinguished name of this user
- The mailAddresses collection is a set of strings eaach of which is a possible mail address of our user. The corresponding LDAP attribute is mailAddress and can have multiple values

4. Mapping the user bean

Now we have to say to snapshot the correspondence between our User class and the entries in the directory. We will do it using an xml file called User.odm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectdir-mapping>
        <class name="example.User"
               objectClass="inetorgperson">
                <dn name="id"/>
                <property name="name" ldapName="cn"/>
                <bag name="mailAddresses">
                        <property ldapName="mail"/>
                </bag>
                <reference name="manager"
                           ldapName="manager"
                           class="net.sf.snapshot.User"/>
        </class>
</objectdir-mapping>
```

The root element of this file is objectdir-mapping, but there is no much to say about it. It contains a class element for each Java class we want to map. In this case we have only the User class. The attributes of this element are:

- name, that defines the name of the Java class that we want to map
- objectClass, that is the name of the LDAP objectClass

So we are saying that objects of Java class User will be searched in the directory among entries that have objectClass=inetorgperson.

Eache element inside class defines a property mapping:

- The dn element says in which property we want to copy the distinguished name of the entry

- The property element defines the correspondence between a Java String property and a simple attribute in the directory
- The bag element defines the mapping for a Java Collection property. Putting a property element inside it we say that the collection will contains all the values of a LDAP attribute
- The reference element says that the mapped attribute will contain the distinguished name of another directory entry and this referenced entry will be put in the Java property

Put this file in the same directory of the User class, then modify snapshot.cfg.xml file putting a reference to this file inside it:

```
<?xml version='1.0' encoding='utf-8'?>
<snapshot-configuration>

        <property name="jndiUrl">ldap://localhost:10389</property>
        <property name="user">uid=admin,ou=system</property>
        <property name="pwd">secret</property>
        <property name="searchBase">dc=example,dc=com</property>

        <mapping resource="example/User.odm.xml"/>

</snapshot-configuration>
```

5. Reading a user from the directory

Put a couple of inetorgperson entries in your LDAP directory. If you want you can use the following ldif file or something similar

```
version: 1
dn: cn=user1,ou=users,dc=example,dc=com
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: top
cn: user1
mail: user1@gmail.com
mail: user1@yahoo.it
sn: user1
uid: user1

dn: cn=user2,ou=users,dc=example,dc=com
objectclass: inetOrgPerson
objectclass: organizationalPerson
objectclass: person
objectclass: top
cn: user2
mail: user2@yahoo.it
manager: cn=user1,ou=users,dc=example,dc=com
sn: user2
uid: user2
```

Now we are ready to use snapshot to read these entries into objects of class User. Look at the following class!

```
public class Hello {

        public static void main(Strig[] args) {
                ServerFactory sf = new ServerFactory();
                Server srv = sf.createServer("snapshot.cfg.xml");
                Session session = srv.newSession();
                User u =
(User)session.load("cn=user1,ou=users,dc=example,dc=com",User.class);
                System.out.println("Name: "+u.getName());
                System.out.println("Manager name:
"+u.getManager().getName());
        }

}
```

This code should be self-explanatory. Objects of the Server class represents a server you can access and contains all the mapping informations. From Server objects you can obtain Session objects and this will be used to access the directory. The load() method of the Session class allow you to retrieve an object knowing its distinguished name.

Obviously we can do more than this, for example we can do queries:

```
public class Hello {

        public static void main(Strig[] args) {
                ServerFactory sf = new ServerFactory();
                Server srv = sf.createServer("snapshot.cfg.xml");
                Session session = srv.newSession();
                Query q =
session.createQuery(User.class,"dc=example,dc=com");
                q.addEqualsCriterion("name","user1");
                Collection c = q.execute();
                for (Iterator iter = c.iterator; iter.hasNext();) {
                        User u = (User)iter.next();
                        System.out.println("Name: "+u.getName());
                        System.out.println("Manager name:
"+u.getManager().getName());
                }
        }

}
```