Using ApacheDS for unit tests

Using ApacheDS for unit tests

The idea is to use ADS as an embedded server for Ldap junit tests. We will build an environment in which it will be convenient to test Ldap applications.

First steps

We have two choices : either we launch the server totally embedded, without having to communicate with it using the Ldap Protocol, or we want to use the server as a remote server, with Ldap protocol in the middle (it can be useful if one want to test specific applications)

Anyway, we will simply launch only one server (if one want to test referrals, it might be necessary to initialize 2 or more servers)

A unit test is a sequence in which three operations are run for each test :

1. setUp() is called
2. the test is launched
3. tearDown() is called

Each time *setUp() is called, a new instance of the server will be initialized, and it can take some time (2 to 5 seconds, depending on your processor). If you have many tests, be aware that each run will cost you the price of the setup, the test and the teardown. This is the only guarantee that those tests are unit tests.

ApacheDS sources provide a common class to initialize the server into unit tests : **AbstractServerTest**. This class can be found in the **apacheds-server-unit** jar file. This class is very usefull, it alleviates you from the burden of initializing the server.

We also have to define a layout for the files and directory we will use in this tutorial. Let's use the **maven** layout :

```
/
|
+--src
    |
    +--test
        |
        +--java      : we will put all the sources into this directory
        |
        +--resources : we will put the resources files into this directory
```


You will need some jars file in order to compile and run this sample :
*junit-3.8.1.jar
*apacheds-server-unit-1.0.2.jar
*nlog4j-1.2.25.jar

The apacheds-server-unit-1.0.2.jar file can't be found in the binary distribution. You will have to download it here : http://people.apache.org/repo/m2-ibiblio-rsync-repository/org/apache/directory/server/apacheds-server-unit/1.0.2/

Source for this test : DemoTest.java
Ldif file for this test : demo.ldif

Creating a blank test

So let's declare the framework for a simple unit test, using a server which will be used through socket (the first free port above 1023 will be used) :

```
package org.apache.directory.demo;

import org.apache.directory.server.unit.AbstractServerTest;

/**
 * Testcase using an embedded Apache Directory Server.
 *
 * @author <a href="mailto:dev@directory.apache.org">Apache Directory Project</a>
 */

public class DemoTest extends AbstractServerTest
{
    /**
     * Initialize the server.
     */
    public void setUp() throws Exception
    {
        super.setUp();
    }

    /**
     * Empty test just to avoid a warning to be thrown when launching the test
     */
    public void testEmpty()
    {
        // Do nothing
    }

    /**
     * Shutdown the server.
     */
    public void tearDown() throws Exception
    {
        super.tearDown();
    }
}
```

This is totally empty, we will fill the blanks now. But let's launch the test as is :

```
838 [main] WARN org.apache.directory.server.core.DefaultDirectoryService - ApacheDS shutdown hook has NOT
been registered with the runtime.  This default setting for standalone operation has been overriden.
1695 [main] WARN org.apache.directory.server.core.schema.bootstrap.BootstrapAttributeTypeRegistry -
Attribute dynamicSubtrees does not have normalizer : using NoopNormalizer
1696 [main] WARN org.apache.directory.server.core.schema.bootstrap.BootstrapAttributeTypeRegistry -
Attribute javaSerializedData does not have normalizer : using NoopNormalizer
...
// Lot of lines ...
...
2996 [main] WARN org.apache.directory.server.core.DefaultDirectoryService -
You didn't change the admin password of directory service instance 'default'.
Please update the admin password as soon as possible to prevent a possible security breach.
2996 [main] INFO org.apache.directory.server.jndi.ServerContextFactory -
LDIF load directory not specified.  No LDIF files will be loaded.
3285 [main] INFO org.apache.directory.server.jndi.ServerContextFactory -
Successful bind of an LDAP Service (1024) is complete.
3409 [main] INFO org.apache.directory.server.jndi.ServerContextFactory -
Unbind of an LDAP service (1024) is complete.
3409 [main] INFO org.apache.directory.server.jndi.ServerContextFactory -
Sending notice of disconnect to existing clients sessions.
```

We can see that the server is launched, some initialization is done, then server is listening on port 1024, and is immediatly shutdown.

Creating our own partition

At the moment, we have created a server which is not totally empty : one partition is created by default, the system partition. We won't use it for our tests, so we will need to create our own partition to play with. Let's call it 'o=sevenseas' (**o** stands for **organization**)

The setUp() method will be completed with all the needed instruction to create a new partition

```
import java.io.File;
import java.util.HashSet;
import java.util.Hashtable;
import java.util.Set;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.naming.directory.Attribute;
import javax.naming.directory.Attributes;
import javax.naming.directory.BasicAttribute;
import javax.naming.directory.BasicAttributes;
import javax.naming.directory.DirContext;

import org.apache.directory.server.core.configuration.MutablePartitionConfiguration;
import org.apache.directory.server.unit.AbstractServerTest;
...

    /**
     * Initialize the server.
     */
    public void setUp() throws Exception
    {
        // Add partition 'sevenSeas'
        MutablePartitionConfiguration pcfg = new MutablePartitionConfiguration();
        pcfg.setName( "sevenSeas" );
        pcfg.setSuffix( "o=sevenseas" );

        // Create some indices
        Set<String> indexedAttrs = new HashSet<String>();
        indexedAttrs.add( "objectClass" );
        indexedAttrs.add( "o" );
        pcfg.setIndexedAttributes( indexedAttrs );

        // Create a first entry associated to the partition
        Attributes attrs = new BasicAttributes( true );

        // First, the objectClass attribute
        Attribute attr = new BasicAttribute( "objectClass" );
        attr.add( "top" );
        attr.add( "organization" );
        attrs.put( attr );

        // The the 'Organization' attribute
        attr = new BasicAttribute( "o" );
        attr.add( "sevenseas" );
        attrs.put( attr );

        // Associate this entry to the partition
        pcfg.setContextEntry( attrs );

        // As we can create more than one partition, we must store
        // each created partition in a Set before initialization
        Set<MutablePartitionConfiguration> pcfgs = new HashSet<MutablePartitionConfiguration>();
        pcfgs.add( pcfg );

        configuration.setContextPartitionConfigurations( pcfgs );

        // Create a working directory
        File workingDirectory = new File( "server-work" );
        configuration.setWorkingDirectory( workingDirectory );

        // Now, let's call the upper class which is responsible for the
        // partitions creation
        super.setUp();
    }
```

Ok, now the partition **sevenseas** should be created. How can we be sure of that ? let's write a test to replace the emptytest() method :

```
/**
     * Test that the partition has been correctly created
     */
    public void testPartition() throws NamingException
    {
        Hashtable<Object, Object> env = new Hashtable<Object, Object>( configuration.toJndiEnvironment() );

        // Create a new context pointing to the overseas partition
        env.put( Context.PROVIDER_URL, "o=sevenSeas" );
        env.put( Context.SECURITY_PRINCIPAL, "uid=admin,ou=system" );
        env.put( Context.SECURITY_CREDENTIALS, "secret" );
        env.put( Context.SECURITY_AUTHENTICATION, "simple" );
        env.put( Context.INITIAL_CONTEXT_FACTORY, "org.apache.directory.server.jndi.ServerContextFactory" );

        // Let's open a connection on this partition
        InitialContext initialContext = new InitialContext( env );

        // We should be able to read it
        DirContext appRoot = ( DirContext ) initialContext.lookup( "" );
        assertNotNull( appRoot );

        // Let's get the entry associated to the top level
        Attributes attributes = appRoot.getAttributes( "" );
        assertNotNull( attributes );
        assertEquals( "sevenseas", attributes.get( "o" ).get() );

        Attribute attribute = attributes.get( "objectClass" );
        assertNotNull( attribute );
        assertTrue( attribute.contains( "top" ) );
        assertTrue( attribute.contains( "organization" ) );
        // Ok, everything is fine
    }
```

The test should succeed. Is that all ? Well, almost. As you can see, a working space has been created ("server-work", at the end of the setup). Do we have to take care of this working space? No. It has been cleaned by the super class !

So everything is fine, the partition is up and running, you are ready to add more tests.

One line in the logs is interesting :

```
3879 [main] INFO org.apache.directory.server.jndi.ServerContextFactory -
LDIF load directory not specified.  No LDIF files will be loaded.
```

The setup has tried to load an LDIF file to inject some data into the partition, but as we didn't specify any Ldif file to be loaded, nothing has been done. let's add some data !

Adding some data into the partition

The **AbstractServerTest** class provide a method called **importLdif(InputStream in)**. It allows you to inject some entries into the newly created partition. How d we use it?

First, we need to have a valid LDIF file containing your entries. We will create two branches in our **sevenseas** organization :

- one for groups
- one for people

Here is the ldif file we will create :

```
dn: ou=groups,o=sevenSeas
objectClass: organizationalUnit
objectClass: top
ou: groups
description: Contains entries which describe groups (crews, for instance)

dn: ou=people,o=sevenSeas
objectClass: organizationalUnit
objectClass: top
ou: people
description: Contains entries which describe persons (seamen)
```

Save it as a text file into a directory where we will be able to read it directly. But where?

We have created the test into a directory src/test/java/org/apache/directory/demo. This is the **maven** way to organized the sources, as seen before. Let's create another directory for the resources : src/test/resources/org/apache/directory/demo. Tis is the place where we will save the ldif file.

Now, to let the server know about the ldif file, just add this line *after* the call to the **setup()** method :

```
...
        // Now, let's call the upper class which is responsible for the
        // partitions creation
        super.setUp();

        // Load a demo ldif file
        importLdif( this.getClass().getResourceAsStream( "demo.ldif" ) );
    }
```


This is important to add the import after the setup : you can't import data while the partition has not been created ...

The **getResourceAsStream** call will automatically read the file from the resources directory, based on the current class package.

How can we be sure that the data has been imported ? Let's do a search request !

Cleanup the code

Before that, let's do some cleanup. The context creation is something we will have to do in each test. We should create a common method called every time to avoid duplicating this code. Let's create this method :

```
/**
     * Create a context pointing to a partition
     */
    private DirContext createContext( String partition ) throws NamingException
    {
        // Create a environment container
        Hashtable<Object, Object> env =
            new Hashtable<Object, Object>( configuration.toJndiEnvironment() );

        // Create a new context pointing to the partition
        env.put( Context.PROVIDER_URL, partition );
        env.put( Context.SECURITY_PRINCIPAL, "uid=admin,ou=system" );
        env.put( Context.SECURITY_CREDENTIALS, "secret" );
        env.put( Context.SECURITY_AUTHENTICATION, "simple" );
        env.put( Context.INITIAL_CONTEXT_FACTORY,
                    "org.apache.directory.server.jndi.ServerContextFactory" );

        // Let's open a connection on this partition
        InitialContext initialContext = new InitialContext( env );

        // We should be able to read it
        DirContext appRoot = ( DirContext ) initialContext.lookup( "" );
        assertNotNull( appRoot );

        return appRoot;
    }
```

This method is added into the body of the test class. This method is very simple and quite straightforward : we just create an initial context pointing to the requested partition, and return a directory context on this partition. It takes a parameter, the partition name.

Let's modify the testpartition method :

```
/**
     * Test that the partition has been correctly created
     */
    public void testPartition() throws NamingException
    {
        DirContext appRoot = createContext( "o=sevenSeas" );

        // Let's get the entry associated to the top level
        Attributes attributes = appRoot.getAttributes( "" );
        ...
```

We just replaced the first lines by a call to the newly created createContext() method.

If you launch the unit test, it should still be ok.

Searching for entries

This is really simple. What we will do is to search for the imported entries. Let's go directly to the code. We will add a new unit test

```
/**
     * Test that the ldif data has correctly been imported
     */
    public void testImport() throws NamingException
    {
        // Searching for all
        Set<String> result = searchDNs( "(ObjectClass=*)", "o=sevenSeas", "",
                                        SearchControls.ONELEVEL_SCOPE );

        assertTrue( result.contains( "ou=groups,o=sevenSeas" ) );
        assertTrue( result.contains( "ou=people,o=sevenSeas" ) );
    }
```

Here, we are looking for all the entries starting at the top level of the partition, within the level. We should only get two entries.

It's not enough : the searchDNs() method does not exist. It is a private method we have created to avoid duplicating some code all over the unit tests. Here is its code :

```
/**
     * Performs a single level search from a root base and
     * returns the set of DNs found.
     */
    private Set<String> searchDNs( String filter, String partition, String base, int scope )
       throws NamingException
    {
        DirContext appRoot = createContext( partition );

        SearchControls controls = new SearchControls();
        controls.setSearchScope( scope );
        NamingEnumeration result = appRoot.search( base, filter, controls );

        // collect all results
        HashSet<String> entries = new HashSet<String>();

        while ( result.hasMore() )
        {
            SearchResult entry = ( SearchResult ) result.next();
            entries.add( entry.getName() );
        }

        return entries;
    }
```

As for the test partiton, we call the createContext() method, then we just do some JNDI magic :

- creating a SearchControl,
- setting the scope,
- calling the serach
- and gathering the returned DN if any

If the test is successful, you get the imported DNs !

Adding your own schema

Ok, let's go deeper into the server configuration. Working with default schema is fine, but some point, you may want to use your own ObjectClasses and AttributeTypes. let's assume you have created them, and that you have been throw the process of generating the class files for it (this process is described in Custom Schema) into a new package (org.apache.directory.demo.schema) where all the generated files will be put.

You will just have to add those lines at the end of the setUp() method (just before the call to the super() method) :

```
...
import org.apache.directory.server.core.configuration.MutablePartitionConfiguration;
import org.apache.directory.server.core.schema.bootstrap.AbstractBootstrapSchema;
import org.apache.directory.server.unit.AbstractServerTest;

import org.apache.directory.demo.schema.DemoSchema;

...
        ...
        ///
        /// add the Demo schema
        ///
        Set<AbstractBootstrapSchema> schemas = configuration.getBootstrapSchemas();
        schemas.add( new DemoSchema() );

        configuration.setBootstrapSchemas(schemas);


        // Now, let's call the upper class which is responsible for the
        // partitions creation
        super.setUp();
    }
```

If we launch the test, nothing special will happen, except that the test will succeed. That's not very impressive...

Conclusion

Ok, this tutorial was a short one, but you get everything you need to play with Apache Directory Server as a Unit Test Engine for your Ldap application. Just create your own partition, define your schema, import your ldif file, and add all the tests you need. it's as simple as explained 🙂

If you have any problem, just feel free to post a mail to users@directory.apache.org, we will be there to help !

Resources

- Embedding ApacheDS - Conference Materials