Testing LDAP Implementations

Emmanuel Lécharny
Symas

Do who need tests anyway ?

OSS projects don't need it...

We have users !

DO WHO NEED TESTS ANYWAY ?

LDAP project phases

- Initial analysis
- Developments + tests <-------+
- Conformance tests <----------+
                               |
                               |
                             costs

Common issues

- Complex setup
- Server reset
- Servers diversity

What are tests anyway ?

What are tests anyway ?

- Unit tests
- Integration tests
- Performance tests

LDAP Unit Tests in Java

- Need a server
- Need an API
- Has to be simple
- Need a mechanism to speed up tests

ApacheDS test framework

- Starts servers with @nnotations
- Easy to use API
- Multiple levels
- Don't bother cleaning up between tests !

A simple test...

- Create of a DirectoryService
- Create of a LdapServer
- Extend AbstractLdapTestUnit
- Get a LdapConnection instance...
- ...and run the test !

CODE

```
@RunWith (FrameworkRunner.class)

// Define the DirectoryService
@CreateDS()

// Define the LDAP protocol layer
@CreateLdapServer( transports = { @CreateTransport(protocol = "LDAP" ) } )

public class A_SimpleServerTest extends AbstractLdapTestUnit {
    /** A simple test */
    @Test public void test() throws Exception {
        LdapServer ldapServer = getLdapServer();

        // Get an admin connection on the defined server
        LdapConnection connection = new LdapNetworkConnection( "localhost", ldapServer.getPort() );
        connection.bind( "uid=admin,ou=system", "secret" );

        // Check that we can read an entry
        assertNotNull( connection.lookup( "ou=system" ) );

        // And close the connection
        connection.close();
    }
}
```

Let's inject some data !

- Same as the previous example
- Injection of **entries**
- Injection of LDIF **files**

ApplyLdifs

```
@RunWith (FrameworkRunner.class)

// Define the DirectoryService
@CreateDS()

// Define the LDAP protocol layer
@CreateLdapServer( transports = { @CreateTransport(protocol = "LDAP" ) } )

// Inject an entry
@ApplyLdifs
    ({
        // Entry # 1
        "dn: uid=elecharny,ou=users,ou=system",
        "objectClass: uidObject",
        "objectClass: person",
        "objectClass: top",
        "uid: elecharny",
        "cn: Emmanuel Lécharny",
        "sn: lecharny",
        "userPassword: emmanuel"
        // More entries...
    })
...
```

ApplyLdifFiles

```
@RunWith (FrameworkRunner.class)

   // Define the DirectoryService
   @CreateDS()

   // Define the LDAP protocol layer
   @CreateLdapServer( transports = { @CreateTransport(protocol = "LDAP" ) } )

   // Inject an file containing entries
   @ApplyLdifFiles( {
            "ldif/muppets.ldif"
      }
   )
```

…

Test with your own partition

- Stop using default
- Define indexes
- Define context entry...

Paritions definition

```
@RunWith (FrameworkRunner.class)

// Define the DirectoryService
@CreateDS( partitions = {
    @CreatePartition(
        name = "example",
        suffix = "dc=example,dc=com",
        contextEntry = @ContextEntry(
            entryLdif = "dn: dc=example,dc=com\n" +
            "dc: example\n" +
            "objectClass: top\n" +
            "objectClass: domain\n\n"
        ),
        indexes = {
            @CreateIndex( attribute = "objectClass" ),
            @CreateIndex( attribute = "dc" ),
            @CreateIndex( attribute = "ou")
        } )
    } )

// Define the LDAP protocol layer
@CreateLdapServer( transports = { @CreateTransport(protocol = "LDAP" ) } )
...
```

Protocols

- SSL
- Ports and servers
- Kerberos

SSL, TLS, SASL

```
@RunWith (FrameworkRunner.class)

// Define the DirectoryService
@CreateDS()

@CreateLdapServer(
   transports = {
       @CreateTransport( protocol = "LDAP", port=10389 ),
       @CreateTransport( protocol = "LDAPS", port=10636 )
   },
   saslHost = "localhost",
   saslMechanisms = {
       @SaslMechanism(name = SupportedSaslMechanisms.GSSAPI, implClass = GssapiMechanismHandler.class),
       @SaslMechanism(name = SupportedSaslMechanisms.GSS_SPNEGO, implClass = NtlmMechanismHandler.class)
   },
   extendedOpHandlers =
         { StartTlsHandler.class }
)
...
```

Kerberos

```
@RunWith (FrameworkRunner.class)

@CreateDS(name = "KerberosUdpIT-class",
   partitions = {
         @CreatePartition(
               name = "example",
               suffix = "dc=example,dc=com")
   },
   additionalInterceptors = {
         KeyDerivationInterceptor.class
   })

@CreateLdapServer(
   transports = {
         @CreateTransport(protocol = "LDAP")
   })

@CreateKdcServer(
   transports = {
         @CreateTransport(protocol = "TCP", address = "127.0.0.1", port = 6087),
         @CreateTransport(protocol = "UDP", address = "127.0.0.1", port = 6087)
   })

@ApplyLdifFiles("org/apache/directory/server/kerberos/kdc/KerberosIT.ldif")
```

Updating the schema

```
@RunWith(FrameworkRunner.class)

@CreateDS(name = "SchemaAddAT-test")

@ApplyLdifs(
{
      // Inject an AT
      "dn: m-oid=1.3.6.1.4.1.18060.0.4.1.2.999,ou=attributeTypes,cn=other,ou=schema",
      "m-usage: USER_APPLICATIONS",
      "m-equality: integerOrderingMatch",
      "objectClass: metaAttributeType",
      "objectClass: metaTop",
      "objectClass: top",
      "m-name: numberOfGuns",
      "m-oid: 1.3.6.1.4.1.18060.0.4.1.2.999",
      "m-singleValue: TRUE",
      "m-description: Number of guns of a ship",
      "m-collective: FALSE",
      "m-obsolete: FALSE",
      "m-noUserModification: FALSE",
      "m-syntax: 1.3.6.1.4.1.1466.115.121.1.27",
...
```

Loading schemas

```
@CreateDS( name = "MethodDSWithPartition",
    partitions = {
        @CreatePartition(
            name = "example",
            suffix = "dc=example,dc=com",
            contextEntry = @ContextEntry(
                  entryLdif =
                  "dn: dc=example,dc=com\n" +
                  "dc: example\n" +
                  "objectClass: top\n" +
                  "objectClass: domain\n\n"),
            indexes = {
                  @CreateIndex(attribute = "objectClass"),
                  @CreateIndex(attribute = "dc"),
                  @CreateIndex(attribute = "ou")
            }
        )
    },
loadedSchemas = {
      @LoadSchema(name = "nis", enabled = true),
      @LoadSchema(name = "posix", enabled = false)
})
```

Need more than one server ?

- Sometime needed : **replication**
- @nnotations at different **level**
  - test suite
  - class
  - method

Replication, producer

```
@CreateDS(
    allowAnonAccess = true,
    name = "provider-replication",
    enableChangeLog = false
)

@CreateLdapServer(transports =
{ @CreateTransport(port = 16000, protocol = "LDAP") })
public static void startProvider( final CountDownLatch counter ) throws Exception
{
DirectoryService provDirService = DSAnnotationProcessor.getDirectoryService();

providerServer = ServerAnnotationProcessor.getLdapServer( provDirService );
providerServer.setReplicationReqHandler( new SyncReplRequestHandler() );
providerServer.startReplicationProducer();


...
```

Replication, consumer

```
@CreateDS(
    allowAnonAccess = true,
    enableChangeLog = false,
    name = "consumer-replication"
    @CreateLdapServer(transports = { @CreateTransport(port = 17000, protocol = "LDAP") })
@CreateConsumer(
      remoteHost = "localhost",
      remotePort = 16000,
      replUserDn = "uid=admin,ou=system",
      replUserPassword = "secret",
      useTls = false,
      baseDn = "dc=example,dc=com",
      refreshInterval = 1000,
      replicaId = 1
)
public static void startConsumer( final CountDownLatch counter ) throws Exception
{
DirectoryService provDirService = DSAnnotationProcessor.getDirectoryService();
consumerServer = ServerAnnotationProcessor.getLdapServer( provDirService );

...
```

Class + Method @nnotation

Class level :

```
@CreateDS(name = "classDS")
public class DirectoryServiceAnnotationTest
{
    …
```

Method level :

```
@Test
@CreateDS(name = "methodDS")
public void testCreateMethodDS() throws Exception
{
    …
```

Let's Save time !

- Don't **restart** the server
- But start with a **clean** place
  - clean data
  - clean schema
- Even if the test **fails**...
- And **don't bother** the tester !

Reverter

- registers each update
- replay them in the reverse order
- let's JUnit does that for you
- You still can't have multiple tests running concurrently with the same updates...

The LDAP API

- Easy to use
- Schema aware
- Deal locally with LDAP syntax and comparison

What's the problem ???

```
@ApplyLdifs({
    // Entry # 1
    "dn: cn=Test Lookup,ou=system",
    "objectClass: person",
    "cn: Test Lookup",
    "sn: sn test"
})
public void testLookupCn() throws Exception {
    LdapConnection connection = getWiredConnection( getLdapServer(), "uid=admin,ou=system", "secret");
    Entry entry = connection.lookup( "cn=test lookup,ou=system", "cn");
    assertNotNull( entry );

    // Check that the CN, or cn, or even 2.5.4.3 attribute is present
    // and that its case insensitive value is there too !
    assertTrue( entry.contains( "cn", "Test Lookup" ) );
    assertTrue( entry.contains( "cn", "test lookup" ) );          // Pathetic failure...
    assertTrue( entry.contains( "2.5.4.3", "test lookup" ) );     // Dumb failure...
    assertTrue( entry.contains( "CN", " test LOOKUP ") );         // Idiotic failure...
}
```

...

What's the Solution then ?

```
@ApplyLdifs({
    // Entry # 1
    "dn: cn=Test Lookup,ou=system",
    "objectClass: person",
    "cn: Test Lookup",
    "sn: sn test"
})
public void testLookupCn() throws Exception {
    LdapConnection connection = getWiredConnection( getLdapServer(), "uid=admin,ou=system", "secret");
    Entry entry = connection.lookup( "cn=test lookup,ou=system", "cn");
    assertNotNull( entry );

    // Make the connection schema aware
    connection.loadSchema();

    // Check that the CN, or cn, or even 2.5.4.3 attribute is present
    // and that its case insensitive value is there too !
    assertTrue( entry.contains( "cn", "Test Lookup" ) );
    assertTrue( entry.contains( "cn", "test lookup" ) );          // Works...
    assertTrue( entry.contains( "2.5.4.3", "test lookup" ) );     // Works too !
    assertTrue( entry.contains( "CN", " test LOOKUP ") );         // And works !!!
}
```

...

What's next ?

The LDAP API

- Reboot SLAMD effort
- LDAP assertions
- Start another external server
- LDAPUnit...


QUESTIONS
ANSWERS