DataNucleus

Java LDAP Persistence with DataNucleus

Stefan Seelmann
seelmann@apache.org

- Stefan Seelmann
- Freelancer
  - Software Development with Java
  - LDAP, Identity- and Access-Management
- Open Source Developer
  - Apache Directory Project
  - DataNucleus LDAP Store

Agenda

- Motivation
- Java Persistence, JDO and DataNucleus
- Basic Demo
- DataNucleus LDAP Store
- Advanced Demo
- Status and Conclusion

Java LDAP Development

- Java APIs for LDAP
  - Mature: Netscape LDAP SDK, JLDAP (Novell/OL)
  - Modern: Unbound ID, Apache Directory, OpenDS
    - Hopefully a common Java LDAP API soon?
  - JNDI, Spring-LDAP
- Drawback:
  - Developer has to deal with LDAP
    - DN, RDN, filters, modification items, error codes
  - Boiler-Plate code, exception handling

Java Persistence

- Standards
  - JPA (Java Persistence API): JSR 220, RDBMS only
  - SDO (Service Data Objects): JSR 235
  - JDO: (Java Data Object): JSR-12 and JSR-243
- Products
  - O/R Mapper: Hibernate, TopLink/EclipseLink, ...
  - Apache iBATIS, Cayenne, OpenJPA, Tuscany, ...
  - DataNucleus
  - ...

JDO

- Java-centric API to access persistent data
- Datastore independent
- Started by Sun, now driven by Apache JDO
- Versions 1.0, 2.0, 2.1, 2.2, 2.3 in progress
- Three main parts
  - Persistence definition (metadata)
  - Persistence API
  - Query language/API

DataNucleus

- Reference implementation of JDO
- Apache licence
- Designed to be extensible
- Data stores:
  - LDAP, RDBMS, db4o, NeoDatis, Excel, ODF, XML, HBase, BigTable
- APIs:
  - JDO, JPA, REST
- Query languages
  - JDOQL, JPQL, native

Persistence Definition

- What data to persist, Java centric
- Annotation, XML metadata, API

```
public class User
{
   String uid;
   Group group;
}
```

```
@PersistenceCapable
public class User
{
   @Persistent
   String uid;
   @Persistent
   Group group;
}
```

```
<class name=“User“>
  <field name=“uid“/>
  <field name=“group“
    persistence-modifier="persistent/>
</class>
```

Persistence API

- Used to persist and retrieve objects

```
PersistenceManagerFactory pmf =
  JDOHelper.getPersistenceManagerFactory(...);
PersistenceManager pm =
  pmf.getPersistenceManager();
...
pm.makePersistent(someObject);
pm.getObjectById(id);
pm.deletePersistent(someObject)
```

Query Language and API

- Used to retrieve objects on criteria
  - API or SQL-like string
  - Elements: filter, ordering, aggregation
  - DataNucleus has in-memory evaluation
- Fetch groups
  - Which fields to fetch

```
Query q = pm.newQuery(User.class);
q.setFilter(“accountId=bbunny“);
Collection result = q.execute();
```

Basic Demo

- Class User
  - Single-valued fields (String, long, Calendar)
  - Multi-valued fields
- Annotations
- Persist, Query, Delete
- Interjection: Reusal of ORM metadata!

LDAP Store Objective

- Use standard API to access LDAP data
- Easier LDAP usage
  - No need to learn (too much) LDAP
  - Work with Java objects instead of LDAP
  - Declarative persistence
- Need full control how data is persisted
- Support existing LDAP data
  - DIT Structure
  - Schema
- Leverage LDAP best practices and patterns

Basic Mapping

- Java Object ↔ LDAP Entry
  - `@PersistentCapable`
  - Object classes defined in „schema“
    - Automatically added to entry
    - Base filter for retrieval
    - Rule: one object class per object
- Java Fields ↔ LDAP Attribute
  - `@Persistent`
  - String, Primitives, Number, Calendar → SV attribute
  - Collection, Set, List of above → MV attribute
  - byte[] → binary attribute

Object Identity

- Application identity and single-field identity
  - One field marked as „primary key“ → RDN
    - Must be unique per object type!
  - String, Primitives, Number, UUID
  - Only single-valued RDN supported
- Parent of Entry determined by „table“
  - Distinguished Name
  - LDAP URL
    - Discriminator for inherited classes
  - Reference to parent field

Relationships

- Relationship Strategies
  - By DN (default)
  - By attribute
  - Hierarchical
  - Embedded
- Dangerous without transactions!

Relationship by DN

- Typical usage: groupOfNames

```
@PersistenceCapable(table="ou=Groups,...", schema="top,groupOfNames")
public class Group
{
    @Persistent(column="member")
    @Extension(vendorName="datanucleus", key="empty-value", value="...")
    Set<User> members = new HashSet<User>();
    ...
}

@PersistenceCapable(table="ou=Users,...",
    schema="top,person,organizationalPerson,inetOrgPerson")
public class User
{
    @Persistent(mappedBy="members")
    Set<Group> memberOf = new HashSet<Group>();
    ...
}
```

Relationship by Attribute

- Typical usage: posixAccount, posixGroup

```
@PersistenceCapable(table="ou=Groups,...", schema="top,posixGroup")
public class Group
{
    @Persistent(column="memberUid")
    @Join(column="uid")
    private Set<Account> members = new HashSet<Account>();
    ...
}

@PersistenceCapable(table="ou=Accounts,...", schema="top,posixAccount")
public class Account
{
    @Persistent(mappedBy="members")
    Set<Group> secondaryGroups = new HashSet<Group>();
    ...
}
```

Hierarchical Relationship

- Typical usage: Organizational structure

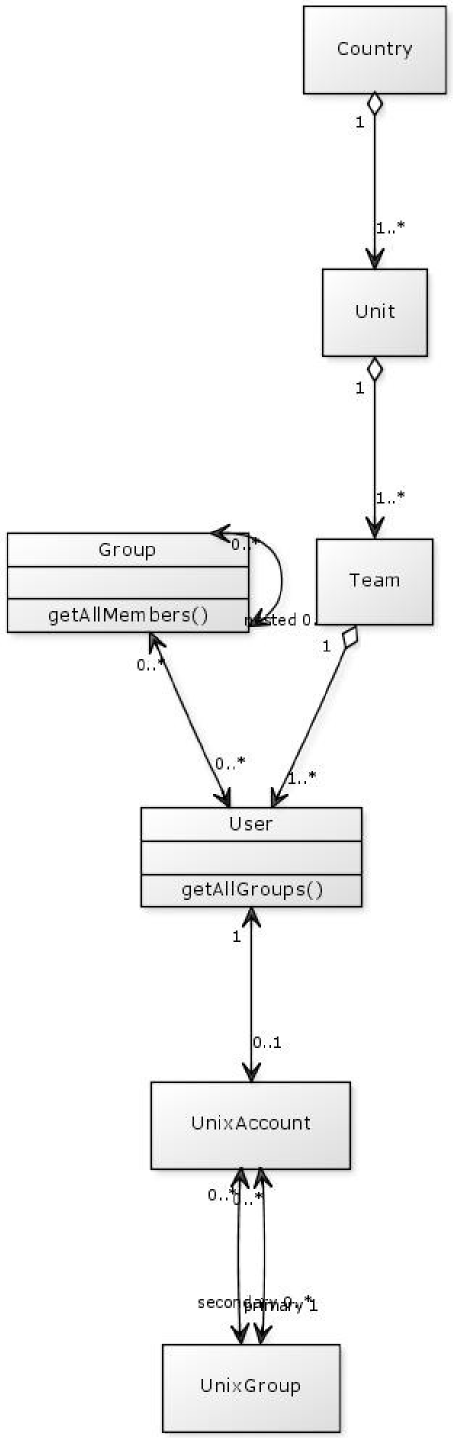
```
@PersistenceCapable(table="dc=example,dc=com",
    schema="top,organizationalUnit")
public class Department
{
    ...
}

@PersistenceCapable(table="{department}",
    schema="top,person,organizationalPerson,inetOrgPerson")
public class User
{
    ...
    @Persistent(defaultFetchGroup = "true")
    private Department department;
    ...
}
```

Embedded Objects

- Containment relationship
- Embedded objects don't have their own identity
  - Loaded together with the owner object
- Two variants:
  - As child
  - Into owner entry

Queries

- Only a subset of JDQL filter is translated to LDAP
  - Logical: & |
  - Operators: ==, !=, <, <=, >=, >
  - Methods: startsWith() and endsWith()
- Not schema aware!
- Advanced filters, aggregation, ordering is evaluated in-memory!



Advanced Demo

- Organization hierarchy
  - country, organization, organizationalUnit
- Users
  - inetOrgPerson
- Nested groups
  - groupOfNames
- Unix accounts and groups
  - posixAccount, posixGroup
- Wicket web application

DataNucleus

Status

- Prototype implementation
- Basic mapping works well
- Relationships by DN and attribute works well
- Hierarchical relationship sucks
- Performance is partially poor
  - Collections
  - Hierarchical relationships
- Transactions are missing!

Future Steps

- Connection handling (encryption, authn)
- Hierarchical relationships
  - compound identity
- Large data
  - lazy loading, paged search, VLV
- Query
- Schema awareness
- java.util.Map
- Auto-creation of schema and DIT
- Tooling to create beans and metadata from LDAP

Conclusion

- Pros:
  - Standardized API for different datastores
  - When using LDAP as datastore
- Cons:
  - Setup (dependencies, enhancer)
  - Learning curve (lifecycle, metadata, configuration)

Resources

- DataNucleus
  - http://www.datanucleus.com
- DataNucleus LDAP store documentation
  - http://www.datanucleus.org/products/accessplatform_2_0/ldap/support.html
- JDO specification
  - http://db.apache.org/jdo/specifications.html
- Paper, slides, examples
  - http://github.com/seelmann/ldapcon2009-datanucleus