Java Object/Relational Persistence with Hibernate

David Lucek

11 Jan 2005

Object Relational Persistence

- Maps objects in your Model to a datastore, normally a relational database.
- Why?
  - EJB Container Managed Persistence (CMP) is Dead.
  - The business logic and domain are normally represented as an object model.
  - The developer should concentrate and work with the object model
  - Persistence code is complex to write in-house
  - Java types vs SQL datatype mismatch
  - Java Collections versus SQL Joins

Object Relational Mapping (ORM) Tools

- A good ORM tools should provide
  - Transparent Persistence
  - Transitive Persistence
  - Automatic dirty checking
  - Support for Inheritance
  - Efficient fetching and caching
- Transparent Persistence means any class can be persisted without:
  - Any implementation of specific interfaces
  - No required persistence superclass
- A good ORM should be able to be run outside of a container. Good for testing.

Popular Object Relational Mapping Tools

- TopLink, Oracle Tool
- JDO, POJO persistence,
  - Official POJO persistence standard
  - Kodo JDO from SolarMetric is popular
  - There are many JDO vendors
- Castor – Open Source
- Hibernate – Open Source
- EJB 3.0 will support POJO, probably too late?

The Transparent Persistence Goal

- The goal is to write a POJO and let a toolkit provide the persistence.
  - We are not there yet, but close
- JDO and Hibernate provide transparent persistence
- They are competing toolkits in base functionality
  - Hibernate at this point has more advanced query capability and raw SQL support.
  - But, the JDO 2.0 Spec is coming out soon, which addresses these concerns

Why Hibernate

- Its Open Source (LGPL)
- Mature toolkit, supported by JBoss
- Very popular
- Web Site: www.hibernate.org
- Latest Stable Version: 2.1.7
- Hibernate 3.0 Beta coming out soon.
- Lots of resources on the web

Main Hibernate Features

- POJO persistence
- Support for fine grained objects
- Flexible Mapping (table to object mapping)
- Two layer Cache Architecture
- Support for Detached Objects
- High Performance Queries
- Eclipse IDE Plug-in
- Round trip toolkits
- 3 Query Options

Basic Hibernate Usage Work Flow

1)Create Java Object Model
2)Create mapping file, hibernate.cfg.xml
    1)Mappings POJO to a SQL Table
    2)Add your JDBC connection settings here or in a separate hibernate.properties file
3)Use the persistence manager API to save/delete or update your object.

- XDoclet Tags and Ant can be used to automate. This is the most common approach

Basic Hibernate POJO

- Must use JavaBean Specification with Accessor methods
- No-arg Constructor
- Collections must use an Interface
- Add XDoclet tags to “get” methods for persistence.
- Add class level tag for table name.
- Can use sequence, either from database or Hibernate generated.

POJO Example

```
/**
 * @hibernate.class table="dvds"
 */
public class DVD implements Serializable {
    private Long id;
    private String title;
    private int type;
    private Account account;  // Parent Object
   Dvds() {}
    /**
     * @hibernate.id generator-class="sequence" type="long" column="id"
     * @hibernate.generatorParam name="sequence" value="dvdseq"
     */
    public Long getId() {  return this.id;    }

    /**
     * @hibernate.property column="title"
   */
    public String getTitle() { return this.title; }

   /** @hibernate.property column="type"
   */
   public int getType() { return this.type; }
}
```

Example Mapping Document

```
<hibernate-mapping>
 <class name="DVD" table="dvds">
    <id name="id" column="id" type="long" >
       <generator class="sequence"/>
          <param name="sequence">dvdseq</param>
       </generator>
    </id>
    <property name="title" column="title" type="string"
       length="15" not-null="true"/>
    <property name="type" column="type" type="short"  not-
    null="true"/>
  </class>
</hibernate-mapping>
```

Persistence API Usage

- API Order
  - Get Session
  - Begin Transaction
  - Save your object by calling save(), saveOrUpdate()
  - Commit and close transaction
- Normally this should be wrapped in a DAO layer

API Usage Example - Saving

```
// Create business Object
DVD dvd = new DVD();
dvd.setTitle("Shrek 2");
dvd.setType(1);  // Single layer
Session session = getSessionFactory().openSession();
Transaction txt = session.BeginTransaction();

// Saves Object to database
session.save(dvd);

tx.commit();
session.close();
```

API Usage – Loading/Retrieving

- Can load by Key or Query.
- Loading by Key:

```
Long id = xxx;
DVD dvd = session.get(DVD.class,id);
```

- Hibernate Query Language (HQL)
  - Native Hibernate Language
  - Can also use native SQL if desired

```
List dvdsList = session.find("from DVD");
// Walk through list as required
```

Hibernate Associations

- Defines relationships in the object model
- Relationship are automatically saved. (Transitive Persistence)
- Example: An Account can have many DVDs

An Account can have 1..* DVDs

Association Example

```
class Account {
      private Long id;
      private Set dvds = new HashSet();
      Account() {}
      public Set getDvds() { return this.dvds; }
      public addDvd(DVD dvd) {
        // Set the Parent Reference
        dvd.setAccount(this);
        // Add to list
            dvds.add(dvd):
      }
}
```

Association Mappings

- Account Mapping

```
<class name=”Account” table=”account” >  ....
  <set name=”dvds”>
    <key column=”id” />
    <one-to-many class=”DVD” />
  </set>
</class>
```

- DVD Mapping

```
<class name=”DVD” table=”dvds” >  ....
  ....
  <many-to-one name=account” column=”account” class
  =”Account” not-null=”true” />
</class>
```

API Example of Transitive Persistence

```
DVD dvd = new DVD();
dvd.setTitle("Kill Bill");
Transaction tx = session.getTransaction();
Long id = xxx;
Account acct = (Account) session.get(Account.class,id);
acct.addDvd(dvd);
// This will auto-magically save the DVD object.
tx.commit();
session.close();
```

Hibernate Query's

- Three types
  1) Hibernate Query Language (HQL)
     - Object orientated version of SQL
     - Has full support for inner/outer joins
     - Support for aggregation and grouping
     - Supports sub queries
  2)Criteria queries (QBC)
     Allows creation and execution of object orientated criteria queries.
  3) SQL queries
     - Direct SQL pass through to the database
     - Can name the SQL queries in the meta data

Other Hibernate Features

- Detached Objects
  - Get an object, close database Session
  - Serialize Object to Web Tier
  - Web Tier sends back the Object and it is reattached to it parent and saved.
  - Eliminates DTOs
- Automatic Versioning
  - Provides optimistic locking based on timestamp or number.