Using the Java™ Data Objects Specification With the Enterprise JavaBeans™ (EJB™) Specification

Persistence Options for the Enterprise

Craig Russell
Architect
Sun Microsystems, Inc.
Learning Goal

Learn how to use Java™ Data Objects ("JDO") technology together with the Enterprise JavaBeans™ (EJB™) specification
As a result of this presentation, you will be able to:

- Name three choices for Enterprise Computing Persistence
- Explain the access patterns for the JDBC™ API, the JDO specification, and Entity Beans
- Describe benefits of CMP, JDO technology, and the JDBC API as implementation alternatives
Craig Russell’s Qualifications

- Specification Lead for Java™ Data Objects technology, Java™ Specification Request 12, (JSR-12)

- Transparent Persistence Architect at Sun Microsystems, and currently implementing CMP using the Java DO specification

- Frequent contributor to public forum discussions on transparent persistence for the Java platform
You Think Programming Persistence for EJB™ Specification-based Components (“EJB components”) Is Hard Work?

Maybe you haven’t heard how JDO can make your life easier
Presentation Agenda

- What is Persistence?
- Persistence Alternatives for the Enterprise
- Session Bean Facade Pattern
- Entity Bean Delegate Pattern
- Batch Programs
What Is Persistence?

- Depends on your point of view:
  - From the programming perspective
    - Long term storage of information after program end; and
  - From the database perspective
    - Shared repository of information among programs
- Today’s presentation focuses on the programming perspective
Persistence Differentiators

Persistence Solutions address:
- Data sharing among users
- Transactional access
- Portability of applications
- Ease of use, productivity, quality:

Database programming represents up to 30% of coding in a project.
Persistence Alternatives

- File I/O (generally illegal in servers)
- Serialization
- Java DataBase Connectivity (JDBC™) API
- EJB-specification Container Managed Persistence
- Transparent Persistence (JDO)
Serialization for Persistence

- Simple API on the surface
  - `writeObject`, `readObject`
  - But... `Externalizable`, `replaceObject`...

- No query

- No partial read or update
  - All or nothing gets read/written

- No transactions

- No sharing (last update wins)
JDBC API for Persistence

- Full access to SQL database functionality
- No domain object model
- Standard API, but
  - Not portable due to SQL variants
- Manual, field by field storage of data
- Hand coded data transformations
- Same data appears multiple times in VM
- Little reuse of code, data models
CMP Entity Beans

- Guaranteed portability
- Declarative query
- Simple domain object model
  - No inheritance, or complex data models
- Optimized database update strategy
- Automatic distribution and security are provided by “EJB container”
Java Data Object Technology for Persistence

- Simple API
- Extended query support
- Complex domain object model
  - Inheritance, complex data models
- Optimized database read/update strategy
- Distribution and security are provided by “EJB container”
JDO Technology Interfaces

Application

PersistenceManager

Query

Transaction

Persistent Object Life Cycle
Query factory

Filter database extent
Java boolean expression

Transaction completion
void makePersistent (Object o)
void deletePersistent (Object o)
Object getObjectById (Object oid)
Transaction currentTransaction()
Query newQuery (Extent ex, String filter)
JDO Technology
Query Interface

- Object execute (Object[] parameters)

- Query filter:
  - Java programming language, boolean expression, e.g.,
    "name.startsWith("Research")"
  - Navigation via Collection.contains(), e.g.,
    "emps.contains(e) & e.salary > param0"
JDO Technology
Transaction interface

Modeled on javax.transaction.UserTransaction

\[
\begin{align*}
\mu \text{commit}(); \\
\mu \text{rollback}(); \\
\mu \text{begin}();
\end{align*}
\]
Inheritance Example

Subclasses inherit fields and methods from superclass

Subclasses redefine methods, and add fields and methods

Mapping object model to database is transparent to programmer

```java
class Employee {
    String name;
}
class PTEmployee extends Employee {
    BigDecimal hourlyWage;
}
class FTEmployee extends Employee {
    BigDecimal salary;
}
```
JDO implementation “knows” class of each persistent instance

- Automatic instantiation of the correct class:
  - `query`  
  - `Navigation`  
  - `getObjectById`  

```java
PersistenceManager pm = pmf.getPersistenceManager();
EmployeeKey ek = new EmployeeKey (empId);
Employee emp = pm.getObjectById(ek);
/* here JDO automatically instantiates instance of the correct class. */
emp.computePaycheck();
```
Persistence in the Enterprise

- Java™ Servlet API/JavaServer Pages™ (JSP™) specification, with JDBC API or JDO API
- Session Beans with JDBC API or JDO API
- BMP Entity Beans with JDBC API or JDO API
- CMP Entity Beans
Session Beans/JSP Specification/Java Servlet API With the JDBC API

DataSource provides transactional Connection
Connection returns ResultSet with data
Session Bean extracts data from ResultSet
InitialContext ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/PersonnelDB");
Connection cx = ds.getConnection();

PreparedStatement st = cx.prepareStatement("SELECT ID, NAME, SALARY FROM EMPLOYEE
WHERE ID = ?");
st.setInt(1, empName);
ResultSet rs = st.executeQuery();

String name = rs.getString(2);
BigDecimal salary = rs.getBigDecimal(3);
Session Beans/JSP Specification/Java Servlet API With the JDO API

PersistenceManagerFactory provides transactional PersistenceManager
PersistenceManager returns Persistent Instance
Session Bean uses Persistent Instance methods

PersistenceManager

Persistent Instance

Session Bean

JSP/Servlet

PersistenceManagerFactory
InitialContext ctx = new InitialContext();
PersistenceManagerFactory pmf =
(PersistenceManagerFactory)
ctx.lookup("java:comp/env/jdo/PersonnelPMF");
PersistenceManager pm =
pmf.getPersistenceManager();

EmployeeKey empKey = new EmployeeKey(empId);
Employee emp = (Employee) pm.getObjectById(empKey);

String name = emp.getName();
BigDecimal salary = emp.getSalary();
Entity Bean Home provides transactional Entity Bean via finder method
Session Bean uses Entity Bean methods
Access to Database is Entity Bean-specific
InitialContext ctx = new InitialContext();
EmployeeHome empHome = (EmployeeHome)
ctx.lookup("java:comp/env/ejb/employee");

EmployeeEJB emp = empHome.findByPrimaryKey
(empId);

String name = emp.getName();
BigDecimal salary = emp.getSalary();
BMP Entity Bean With the JDBC API

- `ejbLoad` reads values from database
- `ejbStore` stores values in database
- `ejbCreate` creates new row
- `ejbRemove` deletes row

Entity Bean

Connection

Result Set

Database

DataSource
BMP Entity Beans With the JDO API

- `ejbLoad` accesses persistent instance
- `ejbStore` is ignored
- `ejbCreate` creates new persistent instance
- `ejbRemove` deletes persistent instance
CMP Entity Beans

Concrete Bean class is generated by deployment tools
Database access is transparent to bean developer
Batch Programs (Non-server)

- The JDO API can be used in non-server environments
- Applications use same domain object model as for server environments
- PersistenceManagerFactory constructed by Properties
- No transaction coordination or distributed transaction support
Summary

- Persistence options for the enterprise include CMP, the Java™ Data Objects (“JDO”) API, and the Java DataBase Connectivity (JDBC™) API.
- CMP provides portable persistence for containers.
- The JDO API can be used with Session Beans, BMP Entity Beans, or outside the container.
Conclusion

JDO should be in every enterprise application programmer’s toolkit
Contact Information

For more information:
http://access1.sun.com/jdo
http://JDOcentral.com