Eclipse JPA

Building JPA Applications with EclipseLink and Dali

Neil Hauge, Oracle
Dali Project Lead

Doug Clarke, Oracle
EclipseLink Project co-Lead
Agenda

• JPA Primer
• EclipseLink JPA
• WTP’s JPA Tooling Project : Dali
• Eclipse JPA in action … the demo
JPA—in a Nutshell

- A Java standard that defines:
  - how Java objects are stored in relational databases (specified using a standard set of mappings)
  - a programmer API for reading, writing, and querying persistent Java objects (“Entities”)
  - a full featured query language
  - a container contract that supports plugging any JPA runtime in to any compliant container.
JPA—POJO Entities

• Concrete classes
• No required interfaces
  – No required business interfaces
  – No required callback interfaces
• new() for instance creation
• Direct access or getter/setter methods
  – Can contain logic (e.g. for validation, etc.)
• “Managed” by an EntityManager
• Can leave the Container (become “detached”)
Eclipse Persistence Services Project – “EclipseLink”

- JPA
- MOxy
- EIS
- SDO
- DBWS
EclipseLink 1.0.1

- JPA: Object-Relational
  - JPA 1.0 with many advanced features
  - Simplified configuration of using annotations and/or XML
  - All leading RDMS with platform specific features
    - Best ORM for the Oracle Database
- MOXy: Object-XML Binding (JAXB)
- SDO: Service Data Objects
- EIS using JCA Resource Adapters
- Containers
  - WebLogic, OracleAS, WebSphere, GlassFish/SunAS, JBoss
Where does EclipseLink JPA fit?

Design Time

- Java Classes
- Mapping Metadata
- Database Schema

Runtime

- JPA Persistence Provider
- Relational Database
- Java SE/EE/OSGi
Mapping

- The activity of ‘Mapping’ is the process of connecting objects/attributes to tables/columns
Standard JPA Mappings

• Core JPA Mappings
  – Id
  – Basic
  – Relationships
    • OneToOne
    • ManyToOne
    • OneToMany
    • ManyToMany
  – And more…

• Annotations and/or XML
EclipseLink JPA Config

- JPA (portable)
  - Persistence.xml with EclipseLink properties
  - Mapping: Annotations and/or orm.xml
- EclipseLink JPA
  - Query hints
  - Standard JPA +
  - EclipseLink annotations
- EclipseLink orm.xml
Advanced Mapping Example

```
@Entity
@Cache(type=SOFT_WEAK, coordinationType=SEND_OBJECT_CHANGES)
@OptimisticLocking(type=CHANGED_COLUMNS)
@Converter(name="money", converterClass=MoneyConverter.class)
public class Employee {
    @Id
    private int id;

    private String name;

    @OneToMany(mappedBy="owner")
    @PrivateOwned
    private List<PhoneNumbers> phones;

    @Convert("money")
    private Money salary

    ...
}
```
Advanced Mappings

- **@BasicCollection** - stores a collection of simple types, such as String, Number, Date, etc., in a single table
- **@BasicMap** - stores a collection of key-value pairs of simple types, such as String, Number, Date, etc., in a single table
- **@PrivateOwned** - supports orphan management
- **@JoinFetch** - enables the joining and reading of a referenced object(s) in the same query as the source object
- **@Mutable** - indicates that the value of a complex field itself can be changed or not changed (instead of being replaced)
- **@Transformation** - enables the mapping of a single field to to one or more database columns.
- **@VariableOneToOne** - supports OneToOne mappings to an interface rather than an Entity
- **@ReadOnly** - makes an Entity read only
Converters

- New converter mappings for type conversion and user defined types include:
  - @Converter
    - @TypeConverter, @ObjectTypeConverter, @StructConverter
  - @Convert

```java
@Entity
@Converter(
    name="Currency",
    converterClass=CurrencyConverter.class)
public class Employee {
    @Convert("Currency")
    private Currency salaryCurrency;
```
Query Framework

- Queries can be defined using
  - Entity Model: JPQL, Expressions, Query-by-example
  - Native: SQL, Stored Procedures

- Customizable
  - Locking, Cache Usage, Refreshing
  - Optimizations: Joining, Batching, parameter binding
  - Result shaping/conversions

- Static or Dynamic
  - Stored Procedure support
Stored Procedure Query

- Stored procedure usage has been simplified through the
  - @Named Stored Procedure Query
  - @Named Stored Procedure Queries annotations.
- These annotations encapsulate stored procedure calls as named queries.
- Client code is unaware that the query they are executing is a stored procedure and not a JP QL or native SQL query.
Additional Query Hints

• Optimized Graph Loading
  – eclipselink.batch
  – eclipselink.join-fetch

• Results & Caching
  – eclipselink.cache-usage
  – eclipselink.read-only
  – eclipselink.result-collection-type
  – eclipselink.refresh

• JDBC
  – eclipselink.jdbc.timeout
  – eclipselink.jdbc.fetch-size
  – eclipselink.jdbc.max-rows
  – eclipselink.jdbc.bind-parameters
Lazy Loading & Fetch Groups

- Supports lazy loading of relationship and @Basic mappings
- Two default Fetch Groups defined automatically: eagerly loaded fields and lazily loaded fields.
  - Initial Entity select fetches only eager fields.
  - Referencing *any* lazy field will fetch all lazy fields
  - Especially useful with CLOB/BLOB fields to avoid loading until necessary
- Enabled by byte code weaving
- Fetch Groups can be manually defined and added to a query as a hint—overrides default fetch groups.
Returning Policy

- Return policy options can be configured using:
ERROR: undefined
OFFENDING COMMAND: G00GFFEncoding

STACK:

/Encoding