What’s New and Exciting in JPA 2.0

Mike Keith
Java and Persistence Architect
Oracle
Something About Me

- Java and persistence architect at Oracle
- Almost 20 years experience in server-side and persistence implementations
- Member of JCP expert groups, including JPA 2.0 (JSR-317), Java EE 6 (JSR-316), EJB 3.1 (JSR-318)
- Book: Pro EJB 3: Java Persistence API (Apress)
- Contributor to other specifications (e.g. SCA, SDO, OSGi, etc.)
- Presenter at numerous conferences and events
It’s All About You
It’s All About You

How many people don’t know very much about JPA yet?
It’s All About You

- How many people don’t know very much about JPA yet?
- How many people are using JPA?
It’s All About You

- How many people don’t know very much about JPA yet?
- How many people are using JPA?
- How many people are still using proprietary persistence APIs (Hibernate, TopLink, etc.) without JPA?
It’s All About You

- How many people don’t know very much about JPA yet?
- How many people are using JPA?
- How many people are still using proprietary persistence APIs (Hibernate, TopLink, etc.) without JPA?
- How many people that are not using JPA right now are planning to use it later?
It’s All About You

- How many people don’t know very much about JPA yet?
- How many people are using JPA?
- How many people are still using proprietary persistence APIs (Hibernate, TopLink, etc.) without JPA?
- How many people that are not using JPA right now are planning to use it later?
- How many people think that they “don’t need no stinking JPA” and that they will never use it?
Where We Are

- Vendors have implemented it
- All the cool people are doing it
- The newbies are asking questions about it
- Architects are talking about it
- Consultants are charging for it

Meanwhile…

- We are trying to finish the 2.0 version of the specification in time for the Java EE 6 release
Main Focus

- Standardize useful properties
- Fill in ORM mapping gaps
- Make object modeling more flexible
- Offer simple cache control abstraction
- Allow advanced locking settings
- Provide more hooks for vendor properties
- Add API for better tooling support
- Enhance JP QL query language
- Support Java API based query language
- Integrate with established validation standards
Some properties are used by every provider

Need to duplicate JDBC properties in persistence.xml for each provider

```xml
<properties>
    ...
    <!-- TopLink -->
    <property name="toplink.jdbc.driver"
        value="oracle.jdbc.OracleDriver"/>
    <property name="toplink.jdbc.url"
        value="jdbc:oracle:thin:@localhost:1521:XE"/>
    <property name="toplink.jdbc.user"
        value="scott"/>
    <property name="toplink.jdbc.password"
        value="tiger"/>
</properties>
```
...<!

<!-- Hibernate -->

<property name="hibernate.connection.driver_class"
value="oracle.jdbc.OracleDriver"/>

<property name="hibernate.connection.url"
value="jdbc:oracle:thin:@localhost:1521:XE"/>

<property name="hibernate.connection.username"
value="scott"/>

<property name="hibernate.connection.password"
value="tiger"/>

...<//properties>
Persistence Unit Properties

Should simply be:

```xml
</properties>
<property name="javax.persistence.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
<property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@localhost:1521:XE"/>
<property name="javax.persistence.jdbc.user" value="scott"/>
<property name="javax.persistence.jdbc.password" value="tiger"/>
```

...
Question:

What are **YOUR** favorite properties and which properties do **YOU** think should be standardized?
More Mappings

Can use Join Tables more:

- Unidirectional/bidirectional one-to-one

```java
@Entity public class Vehicle {
    ...
    @OneToOne
    @JoinTable(name="VEHIC_REC")
    VehicleRecord record;
    ...
}
```

- Bidirectional many-to-one/one-to-many
Can use Join Tables less:

- Unidirectional one-to-many with target foreign key

```java
@Entity
public class Vehicle {
  ...
  @OneToMany
  @JoinColumn(name="VEHIC")
  List<Part> parts;
  ...
}
```
Collections of basic objects or embeddables

```java
@Entity
public class Vehicle {
  ...
  @ElementCollection(targetClass=Assembly.class)
  @CollectionTable(name="ASSEMBLY")
  Collection assemblies;

  @ElementCollection @Temporal(DATE)
  @Column(name="SRVC_DATE")
  @OrderBy("value")
  List<Date> serviceDates;
  ...
}
```
Collections of basic objects or embeddables

@Entity
public class Vehicle {
    ...
    @ElementCollection(targetClass=Assembly.class)
    @CollectionTable(name="ASSEMBLY")
    Collection assemblies;

    @ElementCollection @Temporal(DATE)
    @Column(name="SRVC_DATE")
    @OrderBy("value")
    List<Date> serviceDates;
    ...
}

Is there a better name?
Should we have separate annotations for each of these mappings?
Additional Collection Support

Collections of basic objects or embeddables

```java
@Entity
public class Vehicle {

  ...  
  @ElementCollection(targetClass=Assembly.class)
  @CollectionTable(name="ASSEMBLY")
  Collection assemblies;

  @ElementCollection @Temporal(DATE)
  @Column(name="SRVC_DATE")
  @OrderBy("value")
  List<Date> serviceDates;

  ...
}
```
Collections of basic objects or embeddables

```java
@Entity
public class Vehicle {
    ...
    @ElementCollection(targetClass=Assembly.class)
    @CollectionTable(name="ASSEMBLY")
    Collection assemblies;

    @ElementCollection @Temporal(DATE)
    @Column(name="SRVC_DATE")
    @OrderBy("value")
    List<Date> serviceDates;
    ...
}
```

Should this be some other reserved word instead of "value"?

Should there even be a reserved word for basic objects?
List order can be persisted without being mapped as part of the entity

```java
@Entity
public class Vehicle {

    //...

    @ManyToMany
    @JoinTable(name="VEH_DEALERS")
    @OrderColumn(name="SALES_RANK")
    List<Dealer> preferredDealers;

    //...
}
```
More “Map” Flexibility

Map keys and values can be:

- Basic objects, embeddables, entities

```java
@Entity
public class Vehicle {
    ...
    @OneToOne
    @JoinColumn(name="VEH_ID")
    @MapKeyJoinColumn(name="PART_ID")
    @JoinColumn(name="SUPP_ID")
    Map<Part,Supplier> suppliers;
    ...
}
```
Enhanced Embedded Support

- Embeddables can be nested
- Embeddables can have relationships

```java
@Embeddable
public class Assembly {
    ...
    @Embedded
    ShippingDetail shipDetails;
    ...
    @ManyToOne
    Supplier supplier;
    ...
}
```
Access Type Options

- Mix access modes in a hierarchy
- Combine access modes in a single class

```java
@Entity @Access(FIELD)
public class Vehicle {
    @Id int id;
    @Transient double fuelEfficiency;

    @Access(PROPERTY) @Column(name="FUEL_EFF")
    public double getDbFuelEfficiency() {
        return convertToMetric(fuelEfficiency);
    }
    public void setDbFuelEfficiency(double fuelEff) {
        fuelEfficiency = convertToImperial(fuelEff);
    }
    ...
}
```
Derived Identifiers

- Identifier that includes a relationship
  - Require a additional foreign key field
  - Indicate one of the mappings as read-only
  - Duplicate mapping info

```java
@Entity
public class Part {

    @Id int partNo;
    @Column(name="SUPP_ID")
    @Id int suppId;

    @ManyToOne(insertable=false, updatable=false)
    @JoinColumn(name="SUPP_ID")
    Supplier supplier;

    ...
}
```
Derived Identifiers

Identifiers can be derived from relationships

```java
@Entity @IdClass(PartPK.class)
public class Part {
    @Id int partNo;
    @Id @ManyToOne
    Supplier supplier;
    ...
}
```

```java
public class PartPK {
    int partNo;
    int supplier;
    ...
}
```
Can use different identifier types

```java
@Entity
public class Part {
    @EmbeddedId PartPK partPk;
    @ManyToOne @MappedById
    Supplier supplier;
    ...
}

@Embeddable
public class PartPK {
    int partNo;
    int supplier;
    ...
}
```
Shared Cache API

- API for operating on entity cache shared across all EntityManagers within a given persistence unit
  - Accessible from EntityManagerFactory
- Supports only very basic cache operations
  - Can be extended by vendors

```java
public class Cache {
    public boolean contains(Class cls, Object pk);
    public void evict(Class cls, Object pk);
    public void evict(Class cls);
    public void evictAll();
}
```
Previously only supported optimistic locking, will now be able to acquire pessimistic locks

New LockMode values introduced:

- OPTIMISTIC (= READ)
- OPTIMISTIC_FORCE_INCREMENT (= WRITE)
- PESSIMISTIC
- PESSIMISTIC_FORCE_INCREMENT

Optimistic locking still supported in pessimistic mode

Multiple places to specify lock (depends upon need)
Advanced Locking

Read **then** lock:

```java
Account acct = em.find(Account.class, acctId);
// Decide to withdraw $100 so lock it for update
em.lock(acct, PESSIMISTIC);
int balance = acct.getBalance();
acct.setBalance(balance - 100);
```

Read **and** lock:

```java
Account acct = em.find(Account.class, acctId, PESSIMISTIC);
// Decide to withdraw $100 (already locked)
int balance = acct.getBalance();
acct.setBalance(balance - 100);
```
Advanced Locking

- Trade-offs of getting lock too soon, or using stale data for the update and getting opt lock exception
- In this case: read **then** lock **and** refresh

    Account acct = em.find(Account.class, acctId);
    // Decide to withdraw $100 - lock and refresh
    em.refresh(acct, PESSIMISTIC);
    int balance = acct.getBalance();
    acct.setBalance(balance - 100);

- The “right” approach will depend on the requirements and expectations of the application
API Additions

- Additional API provides more options for vendor support and more flexibility for the user

- EntityManager:
  - LockMode parameter added to find, refresh
  - Properties parameter added to find, refresh, lock

- Other useful additions
  - `void clear(Object entity)`
  - `<T> T unwrap(Class<T> cls)`
  - `getEntityManagerFactory()`
API Additions

- Tools need the ability to do introspection
- Additional APIs on EntityManager:
  - `Set<String> getSupportedProperties()`
  - `Map getProperties()`
  - `LockModeType getLockMode(Object entity)`
- Additional APIs on Query:
  - `int getFirstResult(), int getMaxResults`
  - `Map getHints()`
  - `Set<String> getSupportedHints()`
  - `FlushModeType getFlushMode()`
  - `Map getNamedParameters()`
Enhanced JP QL

- **Timestamp literals**
  
  ```sql
  SELECT t from BankTransaction t
  WHERE t.txTime > '2008-06-01 10:00:01.0-09:00'
  ```

- **Non-polymorphic queries**
  
  ```sql
  SELECT e FROM Employee e
  WHERE CLASS(e) = FullTimeEmployee
  OR e.wage = "SALARY"
  ```

- **IN expression may include collection parameter**
  
  ```sql
  SELECT emp FROM Employee emp
  WHERE emp.project.id IN [:projectIds]
  ```
**Enhanced JP QL**

- **Ordered List indexing**

```sql
SELECT t FROM CreditCard c
  JOIN c.transactionHistory t
WHERE INDEX(t) BETWEEN 0 AND 9
```

- **CASE statement**

```sql
UPDATE Employee e SET e.salary =
  CASE e.rating WHEN 1 THEN e.salary * 1.1
               WHEN 2 THEN e.salary * 1.05
               ELSE e.salary * 1.01
  END
```
Expressions and Criteria API

- Have had many requests for an object-oriented query API
- Most products already have them
- Dynamic query creation without having to do string manipulation
- Additional level of compile-time checking
- Equivalent JP QL functionality, with vendor extensibility
- Objects represent JP QL concepts, and are used as building blocks to build the query definition
- Natural Java API allows constructing and storing intermediate objects
Expressions and Criteria API

**QueryDefinition**
- Objectification of JP QL string
- Constructed from a QueryBuilder factory
- Housed inside Query object -- leverages Query API
- Contains one or more “query roots” representing the domain type(s) being queried over

**DomainObject**
- Equivalent to an identification variable
- Represents single instance of entity/embeddable type
- Extends QueryDefinition for increased ease of use
Expressions and Criteria API

**JP QL:**

```
SELECT a FROM Account a
```

```
Query q = em.createQuery(
        "SELECT a FROM Account a");
```

**QueryDefinition:**

```
QueryDefinition qdef = em.getQueryBuilder().createQueryDefinition(Account.class);
Query q = em.createQuery(qdef);
```
Expressions and Criteria API

```java
SELECT a.id FROM Account a
    WHERE a.balance > 100

queryBuilder qb = em.getQueryBuilderFactory();
DomainObject acct = qb.
    createQueryDefinition(Account.class);
acct.select(acct.get("id"))
    .where(acct.get("balance")
        .greaterThan(100));
```
Question: Do you think the ease-of-use/complexity trade-off is worth it?

Alternative:
```java
queryBuilder qb = em.getQueryBuilder();
QueryDefinition qdef =
    qb.createQueryDefinition();
DomainObject acct = qdef.addRoot(Account.class);
qdef.select(acct.get("id"))
    .where(acct.get("balance")
    .greaterThan(100));
```
SELECT e
    FROM Employee e, Employee mgr
    WHERE e.manager = mgr AND mgr.level = "C"

DomainObject emp = qb.
    createQueryDefinition(Employee.class);
DomainObject mgr = emp.addRoot(Employee.class);
emp.select(emp)
    .where(emp.get("manager").equal(mgr)
        .and(mgr.get("level").equal("C")));

Expressions and Criteria API
SELECT c.id, a
FROM Account a JOIN a.customer c
WHERE c.name LIKE :custName

DomainObject a = qb.
   createQueryDefinition(Account.class);
DomainObject c = a.join("customer");
a.select(c.get("id"), a)
    .where(c.get("name")
        .like(a.param("custName")));
Summary

- JPA 2.0 shipped as part of the Java EE 6 release (J1 09)
- JPA 2.0 Reference Implementation will be EclipseLink project (open source TopLink)
  - Shipped with Glassfish, Spring, or standalone
  - [http://www.eclipse.org/eclipselink](http://www.eclipse.org/eclipselink)
- Download JPA 2.0 Public draft and have a look
- If you have any suggestions talk to an expert group member or send email to feedback alias:
  - jsr-317-edr-feedback@sun.com
Summary

- JPA 2.0 is introducing many of the things that were missing and that people asked for
- Have reached the 90-95% level
- JPA will never include *everything* that *everybody* wants
- There are now even fewer reasons to use a proprietary persistence API without JPA
- Just because a feature is there doesn’t mean you have to use it!
Early Access

You can access some of the new features as they are available and offer your input:

http://www.eclipse.org/eclipselink

If you like the challenge of developing the internals you can also get involved in the project!
Q&A

Are we not men?

No...we are Devoxx.