An Introduction to Hibernate Three

Gavin King

gavin@hibernate.org
gavin.king@jboss.com
Road Map

- Overview
- New features of Hibernate 3.0
  - Filters
  - Handwritten SQL support
  - JSR-220 annotation support
  - Representation independence
  - XML Mappings
  - Other new features
- Hibernate and Eclipse
- Hibernate and EJB 3.0
- Integration (the bigger picture)
Overview

- Major revision of Hibernate implementation and APIs
  - Straightforward migration from Hibernate 2.x (mostly automatable)
  - Hibernate 3.0 and Hibernate 2.1 may safely run side by side
  - Currently in rc1, 3.0 final this weekend
  - More than a year in development

- Hibernate3 forms the core implementation of JBoss EntityManager (EJB 3.0)
  - JBoss EJB 3 preview 4 available now
  - Standalone JBoss EntityManager soon!
Hibernate3 Filters

- For temporal, regional, permissioned data
- Allows data visible in a transaction to be subsetted using a global, named, parameterized filter
- Filter conditions defined as SQL fragments in mapping document
- Filters may be applied cumulatively
- Turns a very common case for messy dynamic queries into a simple declarative mechanism
Hibernate3 Filters

- Filters are defined in the mapping metadata

```xml
<filter-def name="region">
    <filter-param name="regionCode" type="string"/>
    region = :regionCode
</filter-def>

<filter-def name="access">
    <filter-param name="accessLevel" type="integer"/>
</filter-def>
```
<class name="SalesPerson" ...

    ...

    <property name="region" not-null="true"/>

    <set name="customers" ...>
        <key column="contactPointId">
            <one-to-many class="Customer">
                <filter name="region"/>
                <filter name="access">accessLvl <= :accessLevel</filter>
            </set>
            <filter name="region"/>
        </set>
        <filter name="region"/>
    </class>

    <class name="Customer" ...

    ...

    <property name="region" not-null="true"/>

    <property name="accessLevel"
        column="accessLvl"
        not-null="true"/>

    <many-to-one name="contactPoint" column="contactPointId"/>
    <filter name="region"/>
    <filter name="access">accessLvl <= :accessLevel</filter>
</class>
Hibernate3 Filters

Filters are enabled at runtime, for a particular session, using `enableFilter()`

```java
session.enableFilter("region")
    .setParameter("regionCode", "TX");
session.enableFilter("access")
    .setParameter("accessLevel", 1);
```
Hibernate3 Filters

- Filters are applied transparently whenever we run a query, or navigate a many-valued association

```java
//returns all salespeople in my region
Iterator iter =
    session.createQuery("from SalesPerson").list().iterator();

//get the customers in my region, which I have access to
while ( iter.hasNext() ) {
    SalesPerson salesPerson = (SalesPerson) iter.next();
    Iterator customers = salesPerson.getCustomers().iterator();
    ...
}
```
Handwritten SQL support

- Most SQL in an application is of a tedious kind
  - So generated SQL is a Good Thing
- Sometimes, we would like to customize the SQL
  - For performance
  - To take advantage of vendor-specific features
  - When working with legacy data
  - Occasionally, to achieve specific functionality
- JDBC frameworks like iBATIS allow you to write your own SQL, and bind to JavaBeans
  - Hibernate3 allows you to write your own SQL, and bind to managed persistent objects
  - This adds higher level semantics and performance optimizations possible in ORM, that a simple JDBC framework does not have
Handwritten SQL support

- Handwritten \texttt{CREATE}, \texttt{UPDATE}, \texttt{DELETE} statements
  - Including support for calling stored procedures
- Native SQL queries
  - Enhancements to Hibernate 2.1 functionality
- Native SQL for entity or collection loading
  - Re-use native SQL query facility, by defining a \texttt{<loader>} query
Handwritten SQL support

• Handwritten SQL for **INSERT, UPDATE**

```xml
<class name="SalesPerson">
    <id name="id">...</id>
    <property name="name"/>
    <property name="creationDate" insert="false" update="false"/>
    <property name="modifiedDate" insert="false" update="false"/>

    <sql-insert>
        INSERT INTO SALES_PERSON (NAME, CREATED, MODIFIED, ID)
        VALUES (?, CURRENT_DATE, CURRENT_DATE, ?)
    </sql-insert>
    <sql-update>
        UPDATE SALES_PERSON
        SET NAME = ?, MODIFIED = CURRENT_DATE
    </sql-update>
</class>
```
Handwritten SQL support

- Handwritten SQL for loading

```
<class name="SalesPerson">
  <id name="id">...</id>
  <property name="name"/>
  <property name="creationDate" insert="false" update="false"/>
  <property name="modifiedDate" insert="false" update="false"/>
  <loader query-ref = "loadSalesPerson"/>
</class>

<sql-query name = "loadSalesPerson">
  <return alias="sp" class="SalesPerson"/>
  SELECT ID AS {sp.id}, TRIM(NAME) AS {sp.name},
          CREATED AS {sp.creationDate}, MODIFIED as {sp.modifiedDate}
  FROM (SELECT * FROM SALES_PERSON UNION SELECT * FROM OLD_SALES_PERSON)
  WHERE ID = ?
</sql-query>
```
JSR-220 annotation support

- JSR-220 (EJB 3.0) defines annotations for O/R mapping
  - You can define your domain model, and mapping “all in one place”
  - Annotations may be overridden at deployment time (but this will not usually be necessary)
- There are two levels of annotation
  - Logical level, for making explicit, things that are implicit in the object model
    - `@Entity`
    - `@ManyToOne`, `@OneToOne`
  - Schema-level, for defining the mapping to a pre-existing relational schema
    - `@Table`
    - `@Column`
    - `@JoinColumn`
JSR-220 annotation support

```java
@Entity(access=FIELD)
@Table(name="sales_person")
public class SalesPerson {

    @Id(generated=AUTO)
    private Long id;

    private String name;

    @ManyToOne
    @JoinColumn(name="user_id")
    private User user;

    ...}
```
Representation independence

• Support for non-POJO representation of the domain model
  ✓ Persistence engine is abstracted from the Java-level representation of the data
  ✓ “property” and “entity name” are now logical constructs
  ✓ Built-in support for graphs of Maps
  ✓ Built-in support for dom4j trees
  ✓ User extension points
  ✓ Future support for SDO?
  ✓ Implement EJB 2.1 on top of Hibernate?
Map attributes = new HashMap();
attributes.put("color", "blue");
attributes.put("mass", new BigDecimal(3.0));
attributes.put("item", item);
item.setAttributes(attributes);
session.persist("ItemAttributes", attributes);
XML Mappings

- Not generic XML databinding functionality
  ✓ Specifically for representing entity data in XML
- Data import/export
- Presentation via XSLT
- XML serialization in some circumstances
- Map properties to nodes
  ✓ node="element"
  ✓ node="@attribute"
  ✓ node="element/@attribute"
  ✓ node="."
<class entity-name="ItemAttributes" node="item-attributes">
    <id name="id" type="int" node="@id">
        <generator class="native"/>
    </id>
    <property name="color" type="string"/>
    <property name="mass" type="big_decimal"/>
    <many-to-one name="item" node="item/@id" embed-xml="false"/>
</class>

Session dom4jSession = session.getSession(EntityMode.DOM4J);
List elements = dom4jSession.createQuery("from ItemAttributes").list();
for (int i=0; i<elements.size(); i++) {
    print( (Element) elements.get(i) );
}

<item-attributes id="123">
    <color>BLUE</color>
    <mass>132.4</mass>
    <item id="455"/>
</item-attributes>
More flexible inheritance support

- Mix table-per-subclass and discriminated-single-table together in the same hierarchy
- Brand new `<union-subclass>` mapping strategy
- “content-based” discrimination for all mapping strategies
  ✓ Discriminator column never necessary!
Mixed strategy mapping

```xml
<class name="Identity">
  <id name="id">
    ...
  </id>
  <discriminator column="type" type="string"/>
  <property name="phoneNumber" length="20"/>

  <subclass name="Person"
    discriminator-value="P">
    <join table="Person">
      <key column="id"/>
      <component name="name">
        ...
      </component>
    </join>
  </subclass>

  <subclass name="Organization"
    discriminator-value="P">
    <property name="name"/>
  </subclass>

</class>
```
<class name="Identity" abstract="true">
    <id name="id">
        ...
    </id>
    <property name="phoneNumber"
        length="20"/>

    <union-subclass name="Person">
        <component name="name">
            ...
        </component>
    </union-subclass>

    <union-subclass name="Organization">
        <property name="name"/>
    </union-subclass>
</class>
Union-subclass mapping

- Queries against this class result in a SQL union

```java
session.createQuery("from Party p left join fetch p.identity").list();

select ...
from Party p
left join (select ... from Person union select ... from Organization) i
  on p.id = i.id
```
Much, much more...

- More new features
  - Unchecked exception model
  - Many new O/R mapping features
    - Much better support for “weird” legacy schemas
    - Exotic inheritance mappings
    - Exotic association mappings
    - Single class to multiple table mappings
  - Further alignment with JSR-220
    - Addition of compliant create(), merge() operations
    - More flexible transitive persistence model
  - JMX-based reporting
    - Access to performance-related statistics in production application
  - Extensible event-driven architecture
    - Extend the behavior of Hibernate with user-written code
    - Complements the many existing extension points
  - New AST-based query parser supporting a superset of HQL and EJBQL
    - HQL was re-implemented from scratch
  - Property-level lazy fetching
    - Requires use of build-time bytecode processing
  - Much more at http://hibernate.org/About/RoadMap
Hibernate and Eclipse

• New eclipse-based O/R mapping toolset
  ✓ Hibernate-aware XML editor
  ✓ Integrated reverse-engineering tool
  ✓ Domain model / mapping visualization
  ✓ Refactoring / code generation plugins
    • Add association
    • Rename association
    • Add persistent property
  ✓ Integrated query prototyping tool and persistent data browser
  ✓ Support for JSR-220 annotations
  ✓ Integration with JBoss IDE EJB 3.0 toolset

• Early preview available soon
  ✓ Preview out now
  ✓ New revisions throughout Q2/Q3
More information

- *Hibernate in Action* (Manning, 2004)
- [http://hibernate.org](http://hibernate.org)