Object/Relational Mapping with Hibernate

Practical ORM

Who is this guy?

- Gavin King
- gavin@hibernate.org

“Modern” ORM Solutions

- Transparent Persistence (POJO/JavaBeans)
- Persistent/transient instances
- Automatic Dirty Checking
- Transitive Persistence
- Lazy Fetching
- Outer Join Fetching
- Runtime SQL Generation
- Three Basic Inheritance Mapping Strategies

Why?

- Natural programming model
- Minimize LOC
- Code can be run and/or tested outside the “container”
- Classes may be reused in “nonpersistent” context
- Minimize database access with smart fetching strategies
- Opportunities for aggressive caching
- *Structural* mapping more robust when object/data model changes

Entity Beans?

- Transparent Persistence ☹
- Persistent/transient instances ☹
- Automatic Dirty Checking ☺
- Transitive Persistence ☹
- Lazy Fetching ☺
- Outer Join Fetching 😐
- Runtime SQL Generation 😐
- Three Basic Inheritance Mapping Strategies ☹

What do RDBs do well?

- Work with *large amounts* of data
  - Searching, sorting
- Work with *sets* of data
  - Joining, aggregating
- Sharing
  - Concurrency (Transactions)
  - Many applications
- Integrity
  - Constraints
  - Transaction isolation

What do RDBs do badly?

- Modeling
  - No polymorphism
  - Fine grained models are difficult
- Business logic
  - Stored procedures kinda suck
- Distribution
  - (arguable, I suppose)

Data is important

Even so, the relational model is important…

The data will be around much longer than the Java application!

The Goal

- Take advantage of those things that relational databases do well
- Without leaving the language of objects / classes

The Real Goal

- Do less work
- Happy DBA

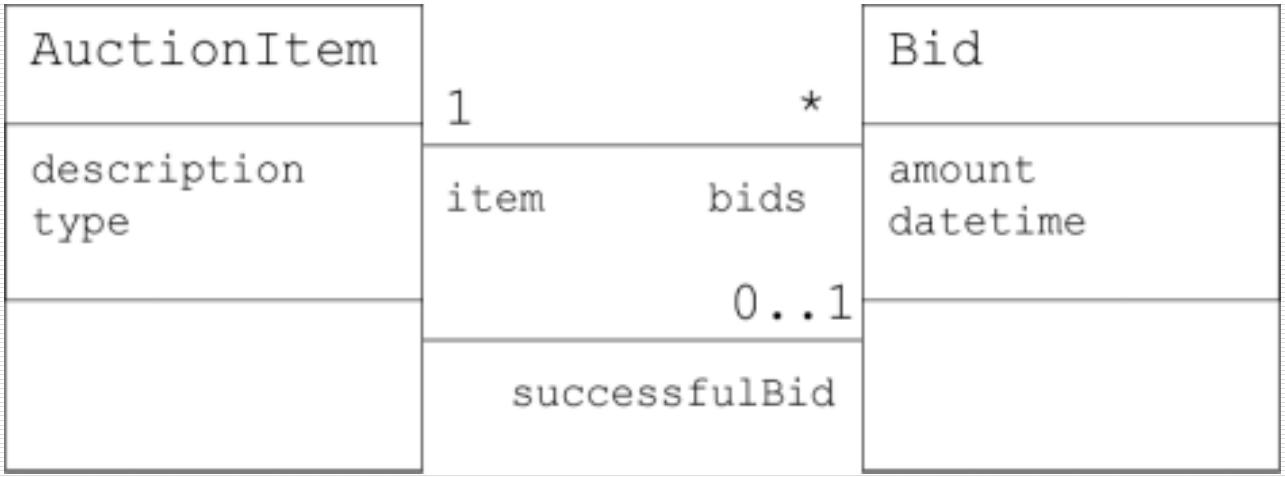
Hibernate

- Opensource (LGPL)
- Mature
- Popular (13 000 downloads/month)
- Custom API
- Will be core of JBoss CMP 2.0 engine

Hibernate

- Persistence for JavaBeans
- Support for very fine-grained, richly typed object models
- Powerful queries
- Support for *detached* persistent objects

Auction Object Model



Persistent Class

- Default constructor
- Get/set pairs
- Collection property is an interface type
- Identifier property

```
public class AuctionItem {
    private Long _id;
    private Set _bids;
    private Bid _successfulBid
    private String _description;

    public Long getId() {
         return  _id;
    }
    private void setId(Long id) {
         _id = id;
    }
    public String getDescription() {
         return _description;
    }
    public void setDescription(String desc) {
         _description=desc;
    }
    …
}
```

XML Mapping

- Readable metadata
- Column / table mappings
- Surrogate key generation strategy
- Collection metadata
- Fetching strategies

```
<class name="AuctionItem" table="AUCTION_ITEM">
    <id name="id" column="ITEM_ID">
        <generator class="native"/>
    </id>
    <property name="description" column="DESCR"/>
    <many-to-one name="successfulBid"
        column="SUCCESSFUL_BID_ID"/>
    <set name="bids"
        cascade="all"
        lazy="true">
        <key column="ITEM_ID"/>
        <one-to-many class="Bid"/>
    </set>
</class>
```

Dirty Checking

Retrieve an `AuctionItem` and change description

```
Session session = sessionFactory.openSession();
Transaction tx = s.beginTransaction();

AuctionItem item =
   (AuctionItem) session.get(ActionItem.class, itemId);
item.setDescription(newDescription);

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

Transitive Persistence

Retrieve an `AuctionItem` and create a new persistent Bid

```
Bid bid = new Bid();
bid.setAmount(bidAmount);

Session session = sf.openSession();
Transaction tx = session.beginTransaction();

AuctionItem item =
   (AuctionItem) session.get(ActionItem.class, itemId);
bid.setItem(item);
item.getBids().add(bid);

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

Detachment

Retrieve an `AuctionItem` and create a new persistent Bid

```
Session session = sf.openSession();
Transaction tx = session.beginTransaction();
AuctionItem item =
   (AuctionItem) session.get(ActionItem.class, itemId);
tx.commit();
session.close();

item.setDescription(newDescription);

Session session2 = sf.openSession();
Transaction tx = session2.beginTransaction();
session2.update(item);
tx.commit();
session2.close();
```

More on this later!

Optimizing Data Access

- Lazy Fetching
- Eager (Outer Join) Fetching
- Batch Fetching

Transparent Lazy Fetching

```
AuctionItem item = (AuctionItem) session.get(ActionItem.class, itemId);

    SELECT … FROM AUCTION_ITEM ITEM WHERE ITEM.ITEM_ID = ?

Iterator iter = item.getBids().iterate();

    SELECT … FROM BID BID WHERE BID.ITEM_ID = ?

item.getSuccessfulBid().getAmount();

    SELECT … FROM BID BID WHERE BID.BID_ID = ?
```

Outer Join Fetching

```
<class name="AuctionItem" table="AUCTION_ITEM">
    <id name="id" column="ITEM_ID">
        <generator class="native"/>
    </id>
    <property name="description" column="DESC"/>
    <many-to-one name="successfulBid"
        outer-join="true"
        column="SUCCESSFUL_BID_ID"/>
    <set name="bids"
        cascade="all"
        outer-join="true">
        <key column="ITEM_ID"/>
        <one-to-many class="Bid"/>
    </set>
</class>
```

Outer Join Fetching

```
AuctionItem item = (AuctionItem) s.get(ActionItem.class, itemId);
```

```
SELECT …
FROM AUCTION_ITEM ITEM
LEFT OUTER JOIN BID BID1 ON BID1.ITEM_ID = ITEM.ITEM_ID
LEFT OUTER JOIN BID BID2 ON BID2.BID_ID = ITEM.SUCCESSFUL_BID
WHERE ITEM.ITEM_ID = ?
```

Optimizing Data Access

- Minimize row reads
- Minimize database roundtrips
- (*Much* less important) Minimize column reads

Optimizing Data Access

- Minimize row reads
  - **Use lazy fetching**
- Minimize database roundtrips
  - **Use outer join fetching**
- (*Much* less important) Minimize column reads
  - *Come back to this one later…*

Optimizing Data Access

- Minimize row reads
  - Use lazy fetching
    - **N+1 Selects Problem (too many roundtrips)**
- Minimize database roundtrips
  - Use outer join fetching
    - **Cartesian Product Problem (huge result set)**

Optimizing Data Access

Solution: *Runtime Fetch Strategies*

1. Say what objects you need
2. Navigate the object graph

Hibernate Query Options

- Hibernate Query Language (HQL)
  - "Minimal" OO dialect of ANSI SQL
- `Criteria` Queries
  - Extensible framework for expressing query criteria as objects
  - Includes "query by example"
- Native SQL Queries

Hibernate Query Language

- Make SQL be object oriented
  - Classes and properties instead of tables and columns
  - Polymorphism
  - Associations
  - *Much* less verbose than SQL
- Full support for relational operations
  - Inner/outer/full joins, cartesian products
  - Projection
  - Aggregation (max, avg) and grouping
  - Ordering
  - Subqueries
  - SQL function calls

Hibernate Query Language

- HQL is a language for talking about “sets of objects”
- It unifies *relational operations* with *object models*

Hibernate Query Language

Simplest HQL Query:

```
from AuctionItem
```

i.e. get all the AuctionItems:

```
List allAuctions = session.createQuery("from AuctionItem")
    .list();
```

Hibernate Query Language

More realistic example:

```
select item
from AuctionItem item
   join item.bids bid
where item.description like 'hib%'
   and bid.amount > 100
```

i.e. get all the **AuctionItem**s with a **Bid** worth > 100 and description that begins with "hib"

Hibernate Query Language

Projection:

```
select item.description, bid.amount
from AuctionItem item
   join item.bids bid
where bid.amount > 100
order by bid.amount desc
```

i.e. get the description and amount for all the **`AuctionItems`** with a **`Bid`** worth > 100

Hibernate Query Language

Aggregation:

```
select max(bid.amount), count(bid)
from AuctionItem item
   left join item.bids bid
group by item.type
order by max(bid.amount)
```

Hibernate Query Language

Runtime fetch strategies:

```
from AuctionItem item
    left join fetch item.bids
    join fetch item.successfulBid
where item.id = 12
```

```
AuctionItem item = session.createQuery(…)
    .uniqueResult(); //associations already fetched
item.getBids().iterator();
item.getSuccessfulBid().getAmount();
```

Criteria Queries

```
List auctionItems =
   session.createCriteria(AuctionItem.class)
        .setFetchMode(“bids”, FetchMode.EAGER)
        .add( Expression.like(“description”, description) )
        .createCriteria(“successfulBid”)
               .add( Expression.gt(“amount”, minAmount) )
        .list();
```

Equivalent HQL:

```
   from AuctionItem item
        left join fetch item.bids
   where item.description like :description
        and item.successfulbid.amount > :minAmount
```

Example Queries

```
AuctionItem item = new AuctionItem();
item.setDescription("hib");
Bid bid = new Bid();
bid.setAmount(1.0);
List auctionItems =
   session.createCriteria(AuctionItem.class)
        .add( Example.create(item).enableLike(MatchMode.START) )
        .createCriteria("bids")
              .add( Example.create(bid) )
        .list();
```

Equivalent HQL:

```
   from AuctionItem item
        join item.bids bid
   where item.description like 'hib%'
        and bid.amount > 1.0
```

Fine-grained Persistence

- “More classes than tables”
- Fine-grained object models are good
  - Greater code reuse
  - More typesafe
  - Better encapsulation

Components

- **`Address`** class
- **`street`**, **`city`**, **`postCode`** properties
- **`STREET`**, **`CITY`**, **`POST_CODE`** columns of the **`PERSON`** and **`ORGANIZATION`** tables
- Mutable class

Components

```
<class name="Person" table="PERSON">
   …
   <component name="address">
        <property name="street" column="STREET"/>
        <property name="city" column="CITY"/>
        <property name="postCode" column="POST_CODE"/>
   </component>

</class>
```

Custom Types

- **`MonetoryAmount`** class
- Used by lots of other classes
- Maps to **`XXX_AMOUNT`** and **`XXX_CURRENCY`** columns
- Performs currency conversions (behaviour!)
- Might be mutable or immutable

Custom Types

```
<class name="Bid" table="BID">
   …
   <property name="amount" type="MonetoryAmountUserType">
        <column name="AMOUNT"/>
        <column name="CURRENCY"/>
   </property>

</class>
```

We still have to write the **`MonetoryAmountUserType`** class!

DTOs are Evil

- “Useless” extra LOC
- Not objects (no behavior)
- Parallel class hierarchies smell
- Shotgun change smell

Solution: detached object support

Detached Object Support

- For applications using servlets + session beans
- You don't need to `select` a row when you only want to `update` it!
- You don't need DTOs anymore!
- You may serialize objects to the web tier, then serialize them back to the EJB tier in the next request
- Hibernate lets you *selectively* reassociate a subgraph! (essential for performance)

Detached Object Support

Step 1: Retrieve some objects in a session bean:

```
public List getItems() throws … {
  return getSession()
    .createQuery(“from AuctionItem item where item.type = :itemType”)
    .setParameter(“itemType”, itemType)
    .list();
}
```

Detached Object Support

Step 2: Collect user input in a servlet/action:

```
item.setDescription(newDescription);
```

Detached Object Support

Step 3: Make the changes persistent, back in the session bean:

```
public void updateItem(AuctionItem item) throws … {
   getSession().update(item);
}
```

Detached Object Support

Even transitive persistence!

```
Session session = sf.openSession();
Transaction tx = session.beginTransaction();
AuctionItem item =
   (AuctionItem) session.get(ActionItem.class, itemId);
tx.commit();
session.close();

Bid bid = new Bid();
bid.setAmount(bidAmount);
bid.setItem(item);
item.getBids().add(bid);

Session session2 = sf.openSession();
Transaction tx = session2.beginTransaction();
session2.update(item);
tx.commit();
session2.close();
```

The Big Problem

- Detached objects + Transitive persistence!
- How do we distinguish between newly instantiated objects and detached objects that are already persistent in the database?

The Big Problem (solution)

1. Version property (if there is one)
2. Identifier value e.g. `unsaved-value="0"` (only works for generated surrogate keys, not for natural keys in legacy data)
3. Write your own strategy, implement `Interceptor.isUnsaved()`

Hibernate Info

- http://hibernate.org
- Hibernate in Action (Manning, 2004)
- Tool support
  - http://xdoclet.sf.net
  - http://boss.bekk.no/boss/middlegen
  - http://www.andromda.org/