Java Hibenrate

Peter Williams
Matthew Fulmer

Object-Relational Mapping

- Framework for converting data between object-oriented programming languages and relational databases
- Replaces direct database access with high-level object-handling functions to overcome an impedance mismatch

Basic Features

- *Transparent persistence*
  - Persistent classes require only a no-argument constructor
- Supports natural OO idiom
  - polymorphism, inheritance, and Java collections
- Supports multiple query languages
  - HQL, Criteria and Example queries in Java, raw SQL
  - SQL is generated at runtime
- Wide range of database support
  - Oracle, DB2, MySQL, PostgreSQL, MS SQL Server, Ingres, etc.
- Portable
  - No need to write Java code for a specific DBMS

Supports All Development Approaches

- Top-down (start with an object model in Java)
  - Create an object model in Java
  - Write an mapping document (XML) or use annotations
  - Export to database tables using the built-in SchemaExport tool (hbm2ddl)
- Bottom-up (start with a data model)
  - Use HibernateTools to reverse-engineer mapping documents and hbm2java to generate skeletal Java code for the objects
- Middle-out (start with mapping documents)
  - Generate skeletal Java code for the objects using hbm2java†
  - Export to database tables using hbm2ddl
- Meet in the middle (start with a data model and Java classes)
  - Write a mapping document to adapt between the two models

Mapping Document vs. Java Object

- Mapping Document
  - XML file that specifies what properties of the Java class to map to the table and how to map them
- Java Object
  - JavaBean-style property accessor methods (getters and setters)
  - Empty constructor

Mapping Document vs. Java Object (continued)

```
package hello;
public class Message {
   private Long id;
   private String text;
   private Message nextMessage;
   private Message() {}
   public Message(String text) {
      this.text = text;
   }
   public Long getId() {
      return id;
   }
   private void setId(Long id) {
      this.id = id;
   }
   public String getText() {
      return text;
   }
   public void setText(String text) {
      this.text = text;
   }
   public Message getNextMessage() {
      return nextMessage;
   }
   public void setNextMessage(Message nextMessage) {
      this.nextMessage = nextMessage;
   }
}
```

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
   <class
      name="hello.Message"
      table="MESSAGES">
      <id
         name="id"
         column="MESSAGE_ID">
         <generator class="increment"/>
      </id>
      <property
         name="text"
         column="MESSAGE_TEXT"/>
      <many-to-one
         name="nextMessage"
         cascade="all"
         column="NEXT_MESSAGE_ID"/>
   </class>
</hibernate-mapping>
```

Hello World

```
Message message = new Message("Hello World");
System.out.println( message.getText() );
```

- Still prints "Hello World" like a normal, non-persistent object

```
Session session = getSessionFactory().openSession();
Transaction tx = session.beginTransaction();
Message message = new Message("Hello World");
session.save(message);
tx.commit();
session.close();
```

- Begin by opening a Session and beginning a Transaction
- Calling "session.save(Object persistentObject)" saves the message to the database
- Rough equivalent of executing the following SQL:

```
insert into MESSAGES (MESSAGE_ID, MESSAGE_TEXT, NEXT_MESSAGE_ID)
values (1, 'Hello World', null)
```

- What's missing from the Java code that appears in the SQL?

Hello World (continued)

- We never explicitly set MESSAGE_ID
  - Hibernate auto-generates values for the identifier column by default (can be configured in the .hbm.xml file for that class)
- Other Hibernate perks:
  - *Automated dirty checking*: modifications made to a persistent object within a transaction are automatically saved
  - *Cascading save*: objects referenced by a persistent instance are automatically saved to the database too
  - *Transactional write-behind*: use of an algorithm to maximize query efficiency behind the scenes automatically

Hello World (continued)

```
Session session = getSessionFactory().openSession();
Transaction tx = session.beginTransaction();
// 1 is the generated id of the first message
   Message message =
(Message) session.load( Message.class, new Long(1) );
message.setText("Greetings Earthling");
Message nextMessage = new Message("Take me to your leader (please)");
message.setNextMessage( nextMessage );
tx.commit();
session.close();
```

```
select m.MESSAGE_ID, m.MESSAGE_TEXT, m.NEXT_MESSAGE_ID
from MESSAGES m
where m.MESSAGE_ID = 1
insert into MESSAGES (MESSAGE_ID, MESSAGE_TEXT, NEXT_MESSAGE_ID)
values (2, 'Take me to your leader (please)', null)
update MESSAGES
set MESSAGE_TEXT = 'Greetings Earthling', NEXT_MESSAGE_ID = 2
where MESSAGE_ID = 1
```

Demo

- How do we deal with many-to-many relationships?
- Can we create bi-directional associations for the Java objects?
- What's the most efficient way to query over what become large collections in Java?

Object – Database Mapping

- Each row contains an "entity" made up of a tree of java objects that are part of a single "thing"
- One object cannot be stored in more than one row
- Object References are translated to the database in one of two ways:
  - If object B is clearly a property of object A, and is used by nobody else, B can be stored in the same row as A
  - If objects A and B are both mostly standalone, but have references to each other because they are related somehow, they should be stored in separate tables, and a Hibernate will use a join to rebuild that relationship
  - Corollary: every user-defined-type appears at most once in any database. Built-in types are exempt from this rule; the objects "false" and "42" can be in the database numerous times
- A table row can contain any tree of java objects as long as they are all attributes of a single thing

Transactions and Persistence Independence

- With each database conversation, the Java programmer creates a hibernate session
- Within the session, objects are manipulated within Transactions
- Java object identity is preserved within a single session, but not otherwise

Questions?