Enterprise Computing

Behzad Bordbar
Introducing Object Relational Mapping and Hibernate

Strategy

• Why ORM and Hibernate?
• A simple example to provide an overview
• Log4j
• Object lifecycle in Hibernate
• Equality
• Session and its use
• Querying (some HQL!)
• Cascading persistence
• Transactions
• Mapping
• Advanced topics

Contents

• Conventional Java vs JDBC
• Exploring object relation mismatch:
  • granularity
  • subtyping
  • identify
  • navigation
• ORM and its Advantages
• Hibernate
• Starting with an example

A high level view of the course

• Aim: to learn engineering of multi-tiered web based systems

As a java programmer

Now that you know your JDBC:
What is the most striking difference between writing a conventional Java program and writing a program that involves JDBC and database connections?
As a java programmer

What are your observations?

• They don't map very well
• Writing CRUD (create, read, update, delete) primitive statements in JDBC is very time consuming and error prone
• How do we achieve portability when it comes to different dialects of SQL in DBMSs
• Imagine how hard it is to have to use different databases and different flavors of SQL

Object relational paradigm mismatch

is often caused by

• Different levels of granularity
• The problem related to subtypes
• The problem of identity, when two things are equal
• Implementation of associations
• Challenges of navigation

Simple case

No mismatch here

```
public class User {
    private String username;
    private String name;
    private String address;
    private Set billingDetails;
}
```

Simple case

```
public class BillingDetails {
    private String accountNumber;
    private String accountName;
    private String accountType;
    private User user;
}
```

You can create the same model in SQL with another table for BillingDetails

```
create table USERS {
    USERNAME varchar(15) not null primary key,
    NAME varchar(50) not null,
    ADDRESS varchar(100)
}
```

What if we want to include users Address?

```
Address <-> User 1:* BillingDetails
```

Shall we add a new table?

There are different variants of address, shall we use user-defined addresses?

To Java it is just an address, yet another attribute!

Different level of granularity!

Subtyping

```
User 1:* BillingDetails
```

Consider the above:

In Java we have subtyping

In SQL we probably use supertable and subtables.

These two are different- ORM gives proper solution.
Identity

In Java there are two different notions of equality:

- Object identity (a==b i.e. Same memory location)
- Equality by value, using equals() method

In databases it is on the basis of primary key or system defined keys (surrogate keys)

Association and navigation

OO representation of associations is via object reference.

In DBs it is key value matching, hence bidirectional. In OO, the object reference is a pointer and can be unidirectional (in any of the two directions) or bidirectional.

In OO navigation is via pointer mechanism, in databases it is via table join.

What is ORM?

object/relational mapping is the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database.

What is metadata/metamodel?

Metadata is data about data. Schema description (DDL) vs. data description, DTD vs HTML, XSD to XML, Annotation vs. java code, UML metamodel vs. UML model.

Advantages of ORM

- Productivity, because of automation
- Maintainability, raising the level of abstraction via use of mappings
- Performance, by using complex optimisation techniques
- Vendor independence, a middleware to cater for various SQL dialects

ORM

works by (reversibly) transforming data from one representation to another-like compilers:

- An API for performing basic CRUD operations on objects of persistent classes
- A language or API for specifying queries that refer to classes and properties of classes
- A facility for specifying mapping metadata
- A technique for the ORM implementation to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions

What follows

Step 1: study of a Hello World example so that you can see a complete- although very simple application

Step 2: we will elaborate on samples of the development processes- in particular different mappings
Different Hibernate Development Processes

1. Start from Java implementation (POJO)
2. Create mapping XML file for the metadata
3. Use it to generate the database schema

(You can also use Java Annotations instead of the XML file)

What is a Java annotation?

Libraries used in Hibernate

In Hibernate we make use of various libraries:

+ lib
  antlr.jar
  asm.jar
  asm-attrs.jar
  c3p0.jar
  cglib.jar
  hibernate3.jar
  jta.jar
  commons-collections.jar
  commons-logging.jar
  dom4j.jar
  log4j-1.2.11.jar
  postgresql-8.2-504.jdbc2.jar
  ...

What are these?