University of Liverpool

Comp220/Comp285

Version 1.1   Seb Coope   3/3/2020

The first section of this document is an introduction to Hibernate, the second part shows you how to get the example code working.

Contents

Brief notes on using Hibernate 2

  Introduction 2

  Why use an ORM such as Hibernate........ 2

  Language support 2

  Example Hibernate code........ 2

Getting the example code working........ 6

Brief notes on using Hibernate

Introduction

Hibernate is what termed an object relational management system or ORM. It allows you to save objects to an external database and loading them up again. It supports a wide range of databases including Oracle, MySQL and Microsoft SQL Server. In general Hibernate requires a SQL type database.

Why use an ORM such as Hibernate

You don't have to spend a lot of time writing SQL and managing table names and database schemas, the ORM details with a lot of that detail.

Potential to build a more scalable application using features of hibernate such as clustering and sharding automatically.

Hibernate will automatically store the structure of sub-classes and super-classes and relationships between classes when handling objects, the programmer only needs to add a simple annotation to the class to tell hibernate about these.

Language support

Hibernate is a Java API, there are ORMs for other languages such as nHibernate for C# and SQLAlchemy and Django for Python.

Example Hibernate code

The example code for hibernate is on Canvas and is called HibernateExample.zip.

This example shows how a set of objects defined for the hotel booking system can be marked up for storage using Hibernate annotations. In the example code, all the model objects are stored in the package, model, looking in src/model directory.

Here is an example of the start of a class definition annotated for storage by Hibernate, this class is in this Person.java.

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(uniqueConstraints= @UniqueConstraint(columnNames = {"surname",
"forename1","forename2","dateOfBirth"}))
public class Person  {
```

@Entity tells hibernate you want to store objects of this class on the database.
@Inheritance tells hibernate to objects of this class and its children (sub-class objects) in different tables and use a table join when loading up the object hierarchy. The final annotation establishes database uniqueness over the given fields/columns.

Row id handling
If you look in the body of the class, you will see the following;

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private int personID;
```

This makes personID a unique id for each row of the database table, this id will be automatically generated for each successful insertion of a row into the table.

Saved attributes

By default all of the attributes objects of the class are saved, so attributes such as

```
private String forename2;
```

Need no special annotations, Hibernate uses reflection to save all the attributes as default.

Hibernate can either access a field directly (field access) or via a getter or setter, this is called property access. When the @Id annotation is placed in the class determines which is the default. In our case, we have places @Id on the field, so our access strategy is field based.

Attributes/fields we don't wish to save

To mark up a field, we don't to wish to save to the database we use the @Transient in front of the field.

Saving an object

An object in Hibernate can be saved, by using a Session object. Have a look in the class DatabaseConnector in the database\hibernate directory, there are a series of static methods, which provide some basic save, update and delete functions.

So save, it intended to perform an SQL insert for an object which has no actual presence within the database. Update updates the data with respect to the object which has a valid id.

There is also a call called saveOrUpdate which performs a save if the object does not have an id assigned or an SQL update if it does. For many circumstances, you can use saveOrUpdate exclusively as you can just let Hibernate decide if the object is already persisted.

Transaction handling and ACID compliance

As long as the underlying database provides transaction handling then Hibernate allows transactions that have problems to be rolled back, in MySQL this means the database engine chosen has to be support transactions, for InnoDB is a good choice. If you use MyISAM you will not get this service.

Look at the following code.

```
Session session =getFactory().openSession();
                    Transaction tx = null;
                    try {
                              tx = session.beginTransaction();
                              session.save(object);
                              // TO DO
                               // Add in more transactions here… any problems… just throw an exception
                              tx.commit();
                              System.out.println("Status is "+session.getStatistics());

                              } catch (Exception e) {
                              if (tx != null)
                                  tx.rollback();

                              //e.printStackTrace();

                    } finally {
                              session.close();

                    }
```

You can add in as many transactions as you like, if you have some kind of error, tx.rollback will undo all the changes.

Loading up objects

Loading up objects can be done if we know the id of the object we are loading up, here is some code (look in DatabaseConnector.loadObject for full code)

```
Person person=new Person();
Integer id=new Integer(3);
Session session =getFactory().openSession();
Transaction tx = null;
try {
  tx = session.beginTransaction();
  session.load(person, id);
```

The problem however with this approach is we often have no idea what the id is, we for Person for example might just know somebodies surname.

To do a more conventional query for example for search objects based on other fields, we can use a piece of what is called Hibernate Query Language (HQL). This is a database independent query language which is generally a lot easier to use than standard SQL dialects. You don't need to know the table name (just the class name), and it is capable of what are called polymorphic queries, these automatically load up associated classes (like super classes).

Here is an example of loading up a list of user's with a given username.

```
String hql = "FROM User U where U.username='" + username + "'";
Session session = DatabaseConnector.getFactory().openSession();
Query<User> query = session.createQuery(hql, User.class);
List<User> results = query.list();
 System.out.println("size of list is " + results.size());
  if (results.size() > 0) {
            this.user=results.get(0);
             password=user.getPassword();
  }
```

Notice the code automatically will load a list of all objects defined by the where clause. This can be more complicated and include AND and OR logical operators as well.

Connecting to the database

There is example code shown in the database\hibernate\DatabaseConfig showing how you connect to the database Have a look at the method

```
public Configuration getHibernateConfig()
```

The example code given stores the configuration information in a JSON file so it can be saved for later and loaded up when the application is loaded.

The code also needs to tell Hibernate all classes of the objects you want to control.

To do this you use the addAnnnotedClass notation like this;

```
config.addAnnotatedClass(model.Person.class);
```

Getting the example code working

To get this code working you need the following;

Ant and Ivy working, you can do this be following the instructions on how to install Ant and Junit on Canvas.

An installation of MySQL.

Follow the instructions to install MySQL on MAC or Windows, the download links are here. You can install the database locally.

https://dev.mysql.com/downloads/mysql/

For the Liverpool lab machines, there is a Google SQL instance (https://cloud.google.com/sql)

The details of the Google instance are as follows;

Connection URL
jdbc:mysql://35.242.177.134:3306/hotelbooking

Username
root
Password
password1234

To build the example code, just type:

```
ant
```

To get the test the connection and configure your username and password, run the database helper code, you can set the connection URL, the username and password.

The connection URL

Configuring the connection to the database

You can test and configure the connection to the database by running DatabaseHelper.

```
java -cp build\classes;.\lib\*    database.hibernate.DatabaseHelper
```

Press test to check the connection, once the connection works, press OK and the details will be saved.

Guest handling

Examine the code in   src\model\Guest.java

You can now add a guest as follows;

```
java -cp build\classes;.\lib\*     model.Guest    add  Surname Forename MiddleName dobDay
dobMonth dobYear
```

You can list guests like this

```
java -cp build\classes;.\lib\*    model.Guest   list
```

or list all guests for a given surname

```
java -cp build\classes;.\lib\*     model.Guest   list surname
```

You can delete all users of a given surname like this

```
java -cp build\classes;.\lib\*     model.Guest     delete surname
```

Exercise

Try and provide a facility to create bookings associated to guests

Add this code to the Booking class.