Study of Hibernate

Akash Ravindra. Khapare
Dept Name: Computer Science & Engineering
College: H.V.P.M'S, C.O.E.T, Amravati
Amravati, India
E-mail: akashkhapare@gmail.com

Abstract:-***Generally we use Object Oriented Programming with the relational databases. Thus there is a mismatch between the objects and the relational databases as relational databases are in the form of tables which need to be mapped with the objects of Object Oriented technology. So the solution to avoid this mismatch is to provide Object Relational Mapping (ORM) which will create a virtual Object Oriented Database (OOD). Hibernate, an open source persistent framework provides an Object Relational Mapping for JAVA.***

I. Introduction

Hibernate is an open source object/relational mapping tool for Java. Hibernate lets you develop persistent classes following common Java idiom - including association, inheritance, polymorphism, composition and the Java collections framework.

Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities and can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC.Hibernates goal is to relieve the developer from 95 percent of common data persistence related programming tasks.
Hibernate makes use of persistent objects commonly called as POJO (POJO = "Plain Old Java Object".) along with XML mapping documents for persisting objects to the database layer. The term POJO refers to a normal Java objects that does not serve any other special role or implement any special interfaces of any of the Java frameworks (EJB, JDBC, JDO).

II. History Of Hibernate

Hibernate was started in 2001 by Gavin King with colleagues from Cirrus Technologies as an alternative to using EJB2-style entity beans. Its original goal was to offer better persistence capabilities than offered by EJB2 by simplifying the complexities and supplementing missing features.
In early 2003, the Hibernate development team began Hibernate2 releases, which offered many significant improvements over the first release.
JBoss, Inc. (now part of Red Hat) later hired the lead Hibernate developers in order to further its development.

In 2005, Hibernate version 3.0 was released. Key features included a new Interceptor/Callback architecture, user defined filters, and JDK 5.0 Annotations (Java's metadata feature). As of 2010, Hibernate 3 (version 3.5.0 and up) was a certified implementation of the Java Persistence API 2.0 specification via a wrapper for the Core module which provides conformity with the JSR 317 standard.
In Dec 2011, Hibernate Core 4.0.0 Final was released. This includes new features such as multi-tenancy support, introduction of ServiceRegistry (a major change in how Hibernate builds and manages "services"), better Session opening from SessionFactory, improved integration via org.hibernate.integrator.spi.Integrator and auto discovery, internationalization support and message codes in logging, and a clearer split between API, SPI and

implementation classes. In Dec 2012, Hibernate ORM 4.1.9 Final was released. In 2012, development was started on Hibernate 5.

III. ORM Overview

JDBC stands for Java Database Connectivity and provides a set of Java API for accessing the relational databases from Java program. These Java APIs enables Java programs to execute SQL statements and interact with any SQL compliant database.
JDBC provides a flexible architecture to write a database independent application that can run on different platforms and interact with different DBMS without any modification.
Pros and Cons of JDBC

Pros of JDBC

- Clean and simple SQL processing
- Good performance with large data
- Very good for small applications
- Simple syntax so easy to learn

Cons of JDBC

- Complex if it is used in large projects
- Large programming overhead
- Query is DBMS specific

Why Object Relational Mapping (ORM)?

When we work with an object-oriented systems, there's a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects.

First problem, what if we need to modify the design of our database after having developed few pages or our application? Second, Loading and storing objects in a relational database exposes us to the following five mismatch problems impedance mismatches problem is occurred.
Mismatch
1) Granularity
Sometimes you will have an object model which has more classes than the number of corresponding tables in the database.

2) Inheritance
RDBMSs do not define anything similar to Inheritance which is a natural paradigm in object-oriented programming languages.

3) Identity
A RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).

4) Associations
Object-oriented languages represent associations using object references where as a RDBMS represents an association as a foreign key column.

5) Navigation
The ways you access objects in Java and in a RDBMS are fundamentally different.

The Object-Relational Mapping (ORM) is the solution to handle all the above impedance mismatches.

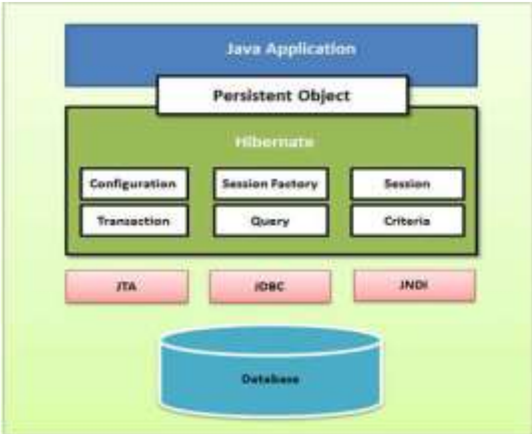
What is ORM?

ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc. An ORM system has following advantages.

- Hides details of SQL queries from OO logic.
- Transaction management and automatic key generation
- Fast development of application.
- Let's business code access objects rather than DB tables.

IV. Architecture Hibernate

The Hibernate architecture is layered to keep you isolated from having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application. Following is a detailed view of the Hibernate Application Architecture with few important core classes.



Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers. Following section gives brief description of each of the class objects involved in Hibernate Application Architecture. Configuration Object

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two keys components: Database Connection: This is handled through one or more configuration files supported by Hibernate. These files are hibernate.propertiesand hibernate.cfg.xml.

 Class Mapping Setup: This component creates the connection between the Java classes and database tables.

SessionFactory Object Configuration object is used to create a SessionFactoryobject which inturn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactoryis a thread safe object and used by all the threads of an application.

The SessionFactoryis heavy weight object so usually it is created during application start up and kept for later use. You would need one SessionFactoryobject per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactoryobjects. Session Object a Session is used to get a physical connection with a database. The Session object is light weight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object. The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed. Transaction Object A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA). This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code. Query Object Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query. Criteria Object Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

V. Hibernate Mapping

An Object/relational mappings are usually defined in an XML document. This mapping file instructs Hibernate how to map the defined class or classes to the database tables. Though many Hibernate users choose to write the XML by hand, a number of tools exist to generate the mapping document. These include XDoclet, Middlegen and AndroMDA for advanced Hibernate users.

```
public class Employee {
private int id;
private String firstName;
private String lastName;
private int salary;
public Employee() {}
public Employee(String fname, String lname, int salary) {
this.firstName = fname;
this.lastName = lname;
this.salary = salary;
}
public int getId() {
return id;
}
public void setId( int id ) {
this.id = id;
}
public String getFirstName() {
return firstName;
}
public void setFirstName( String first_name ) {
this.firstName = first_name;
}
public String getLastName() {
return lastName;
}
public void setLastName( String last_name ) {
this.lastName = last_name;
}
public int getSalary() {
return salary;
```

```
}
public void setSalary( int salary ) {
this.salary = salary;
}
}
```

There would be one table corresponding to each object you are willing to provide persistence. Consider above objects need to be stored and retrieved into the following RDBMS table:

```
create table EMPLOYEE (
id INT NOT NULL auto_increment,
first_name VARCHAR(20) default NULL,
last_name VARCHAR(20) default NULL,
salary INT  default NULL,
PRIMARY KEY (id)
);
```

Based on the two above entities we can define following mapping file which instructs Hibernate how to map the defined class or classes to the database tables.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="Employee" table="EMPLOYEE">
<meta attribute="class-description">
This class contains the employee detail.
</meta>
<id name="id" type="int" column="id">
<generator class="native"/>
</id>
<property name="firstName" column="first_name" type="string"/>
<property name="lastName" column="last_name" type="string"/>
<property name="salary" column="salary" type="int"/>
</class>
</hibernate-mapping>
```

You should save the mapping document in a file with the format <classname>.hbm.xml. We saved our mapping document in the file Employee.hbm.xml. Let us see little detail about the mapping elements used in the mapping file:

- The mapping document is an XML document having <hibernate-mapping> as the root element which contains all the <class> elements.
- The <class> elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the name attribute of the class element and the database table name is specified using the table attribute.

- The <meta> element is optional element and can be used to create the class description.
- The <id> element maps the unique ID attribute in class to the primary key of the database table.
- The name attribute of the id element refers to the property in the class and the column attribute refers to the column in the database table. The type attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.
- The <generator> element within the id element is used to automatically generate the primary key values. Set the class attribute of the generator element is set to native to let hibernate pick up either identity, sequence to create primary key depending upon the capabilities of the underlying database.
- The <property> element is used to map a Java class property to a column in the database table. The name attribute of the element refers to the property in the class and the column attribute refers to the column in the database table. The type attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

VI. Advantages Of Hibernate

- Hibernate takes care of mapping Java classes to database tables using XML files.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in Database or in any table then the only need to change XML file properties.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- A real good Caching Mechanism for faster retrieval of data.
- Your application support almost relational databases.

Conclusion

Hibernate is an Object-relational mapping (ORM) tool. Object-relational mapping or ORM is a programming method for mapping the objects to the relational model where entities/classes are mapped to tables, instances are mapped to rows and attributes of instances are mapped to columns of table.

Hibernate is persistence framework which is used to persist data from Java environment to database. Thus “Virtual object Oriented database” is created.

References

[1] Bhushan S. Sapre, Rohan V. Thakare, Santosh V. Kakade, Dr. B. B. Mesh ram, “Design and Application of the Hibernate Persistence Layer Data Report System using JasperReports,” (IJEIT)Volume 1, Issue 5, May 2012

[2] www.hibernate.org

[3] http://www.gontu.org

[4] http://roseindia.net/.../hibernate/hbm.xml

[5] http://www.javabeat.net/hibernate-ormobjectrelational-framework-an-introduction