Contents

1) Abstract

2) Introduction

3) Key terms

4) How Hibernate works?

5) Hibernate overview

6) Hibernate Query Language

7) Sample code

8) Tools for writing HQL

9) Conclusion and bibliography

Chapter 1

ABSTRACT

Hibernate is a powerful, high performance object/relational persistence and query service. Hibernate lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. Hibernate allows you to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented criteria.

Hibernate offers sophisticated query options, you can write plain SQL, object-oriented HQL (Hibernate Query Language), or create programmatic Criteria and Example queries. Hibernate can optimize object loading *all the time*, with various fetching and caching options. Hibernate adapts to your development process, no matter if you start with a design from scratch or work with an existing database and it will support any application architecture.

User-defined data types and dynamic beans are also supported. Hibernate is released under the Lesser GNU Public License, which is sufficient for use in commercial as well as open source applications. It supports numerous databases, including Oracle and DB2, as well as popular open source databases such as PostgreSQL and MySQL. An active user community helps to provide support and tools to extend Hibernate and make using it easier.

The main advantage of using hibernate is there reduces writing huge code. Actually hibernate developers developed for easy use not to write huge code.
Here there is no need to write the jdbc code here we will use the connection pooling technique and it happens internally and because of connection pooling we can reuse the connection from the pool.

Hibernate supports wide range of databases including Oracle, DB2, Sybase, MS SQL Server, PostgreSQL, MySQL, HypersonicSQL, Mckoi SQL, SAP DB, Interbase, Pointbase, Progress, FrontBase, Ingres, Informix, and Firebird.

Chapter 2

INTRODUCTION

A major portion of the development of an enterprise application involves the creation and maintenance of the persistence layer used to store and retrieve objects from the database of choice. If changes are made to the underlying database schema, it can be expensive to propagate those changes to the rest of the application. Hibernate steps in too fill this gap, providing an easy-to-use and powerful object-relational persistence framework for java applications.

Hibernate provides support for collections and object relations, as well as composite types. In addition to persisting objects, hibernate provides a rich query language to retrieve objects from the database, as well as an efficient caching layer and Java Management Extensions (JMX) support. User defined data types and dynamic beans are also supported.

Hibernate automates to a large extent the creation of an efficient persistence layer for the enterprise application. Hibernate makes mapping objects to be persisted to underlying database easier. In other words, Hibernate allows representing an underlying database by using simple Java objects and vice versa.

By facilitating direct retrieval of persistent objects from the database. Hibernate automates/hides the process of creating objects and populating them with data retrieved from the database (common in JDBC-based applications), saving the developer from such tedious routine tasks.

Hibernate uses the following ways to retrieve objects from the database:

- Hibernate Query Language (HQL).
- Query By Criteria (QBC) and Query BY Example (QBE) using Criteria API.
- Native SQL queries.

Chapter 3

Key terms:

3.1) HQL

Although it is possible to use native SQL queries directly with a Hibernate-based persistence layer, it is more efficient to use HQL instead. The reasons of choosing HQL over the other two methods are given below:

- HQL allows representing SQL queries in object-oriented terms—by using objects and properties of objects.
- Instead of returning plain data, HQL queries return the query result(s) in the form of object(s)/tuples of object(s) that are ready to be accessed, operated upon, and manipulated programmatically. This approach does away with the routine task of creating and populating objects from scratch with the "resultset" retrieved from database queried.
- HQL fully supports **polymorphic queries**. That is, along with the object to be returned as a query result, all child objects (objects of subclasses) of the given object shall be returned.
- HQL is easy to learn and implement, as its syntax and features are very similar to SQL.
- HQL contains many advance features such as pagination, fetch join with dynamic profiling, and so forth, as compared to SQL.
- HQL facilitates writing database-type independent queries that are converted to the native SQL dialect of the underlying database at runtime. This approach helps tap the extra features the native SQL query provides, without using a non-standard native SQL query.

3.2) ORM

The term object/relational mapping (ORM) refers to the technique of mapping a data representation from an object model to a relational data model with a SQL-based schema.

So what can an ORM do for you? A ORM basically intends to takes most of that burden of your shoulder. With a good ORM, you have to define the way you map your classes to tables once - which property maps to which column, which class to which table, etc. With a good ORM you can take the plain java objects you use in the application and tell the ORM to persist them. This will automatically generate all the SQL needed to store the object. An ORM allows you to load your objects just as easily: A good ORM will feature a query language too. The main features include:

1. Less error-prone code
2. Optimized performance all the time
3. Solves portability issues
4. Reduce development time

3.3) Persistent class

Hibernate provides transparent persistence, the only requirement for a persistent class is a no-argument constructor. In a persistent class no interfaces have to be implemented and no persistent super class has to be extended. The Persistent class can be used outside the “persistence” context. Persistent classes are classes in an application that implement the entities of the business problem.

3.4) Transparent persistence

Hibernate provides *transparent persistence*, the only requirement for a persistent class is a no-argument constructor. You don't even need classes, you can also persist a model using Maps of Maps, or just about anything else. You don't even need tables; Hibernate can map entities and particular properties to SQL expressions.

Chapter 4

4) How Hibernate Works?

Rather than utilize bytecode processing or code generation, Hibernate uses runtime reflection to determine the persistent properties of a class. The objects to be persisted are defined in a mapping document, which serves to describe the persistent fields and associations, as well as any subclasses or proxies of the persistent object. The mapping documents are compiled at application startup time and provide the framework with necessary information for a class.

Additionally, they are used in support operations, such as generating the database schema or creating stub Java source files. A SessionFactory is created from the compiled collection of mapping documents. The SessionFactory provides the mechanism for managing persistent classes, the Sessioninterface. The Session class provides the interface between the persistent data store and the application. The Session interface wraps a JDBC connection, which can be user-managed or controlled by Hibernate, and is only intended to be used by a single application thread, then closed and discarded.

Chapter 5

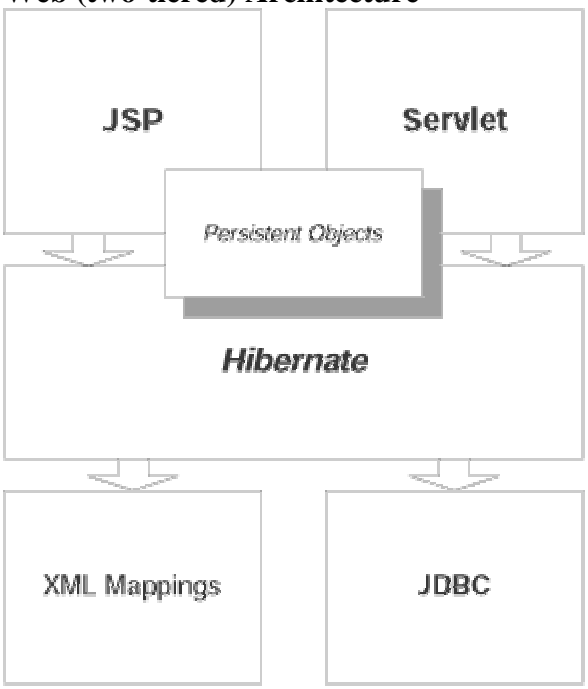
5) Hibernate overview

5.1) Hibernate architecture:

. In Web (two-tiered) Architecture Hibernate may be used to persist Java Beans used by servlets/JSPs in Model/View/Controller architecture.

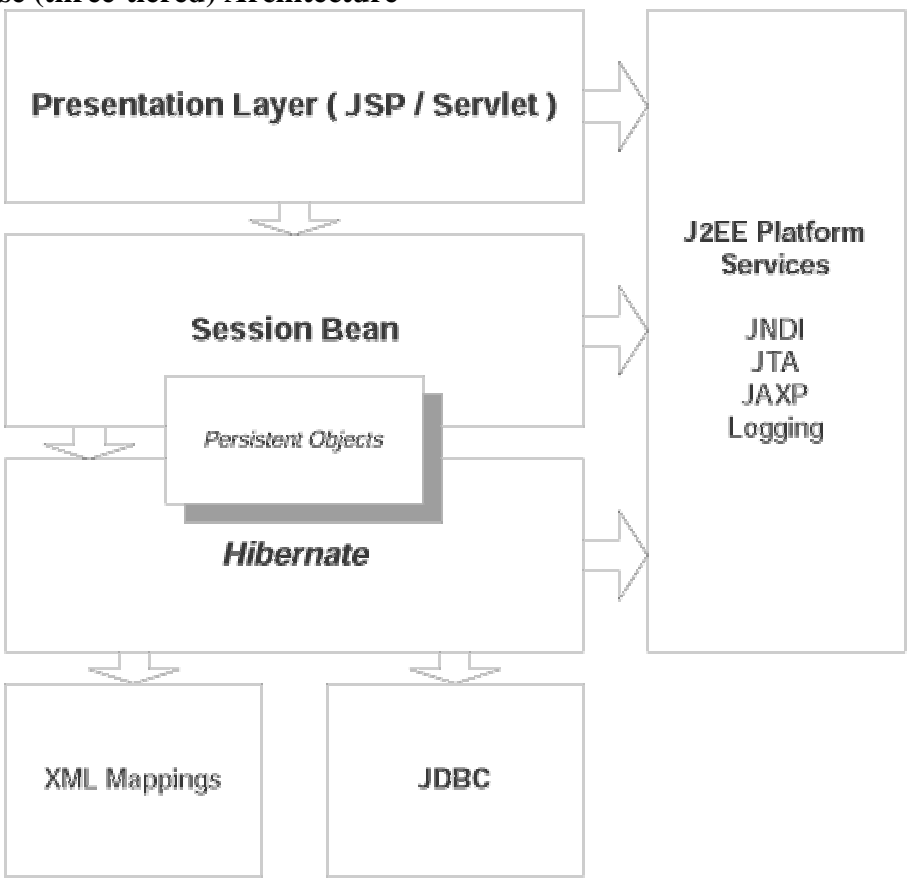
In enterprise (three-tiered) architecture Hibernate may be used by a Session EJB that manipulates persistent objects. Hibernate is architecture-agnostic. Because Hibernate provides persistence as a service, rather than as a framework, it integrates seamlessly with various application architectures. We will show two common (recommended) architectures incorporating Hibernate as a persistence layer

Web (two-tiered) Architecture



.

Enterprise (three-tiered) Architecture



5.2) Steps in hibernate:

The hibernate involves some steps in order to perform the action

1)creation of persistent classes and objects:

here simple classes are created using simple java beans with some properties. Here naming conventions are used for getter and setter methods. Then no-argument constructor is used. Various id are used for identifiers.

2)mapping file:

Hibernate needs to know how to load and store objects of the persistent class. This is where the Hibernate mapping

file comes into play. The mapping file tells Hibernate what table in the database it has to access, and what columns in that table it should use.

2.1) Major elements in mapping file:

1. <hibernate-mapping> element

The root element of hibernate mapping document is <hibernate-mapping> element. This element has several optional attributes.

2. <class> element

The <Class> element maps the domain object with corresponding entity in the database. In simple words the <class> element maps a table with corresponding class . <hibernate-mapping> element allows you to nest several persistent <class> mappings, as shown above. It is however good practice to map only a single persistent class in one mapping file and name it after the persistent superclass, e.g. User.hbm.xml

3. <id> element

The <id> element defines the mapping from that property to the primary key column. The <id> element represents the primary key column, and its associated attribute in the domain object. Mapped classes must declare the primary key column of the database table. Most classes will also have a JavaBeans-style property holding the unique identifier of an instance.

4. <generator> element

The optional <generator> child element names a Java class used to generate unique identifiers for instances of the persistent class. If any parameters are required to configure or initialize the generator instance, they are passed using the <param> element

.Some commonly used generators are :

1. Increment - generates identifiers of type long, short or int that are unique only when no other process is inserting data into the same table. Do not use in a cluster.

2. Sequence - uses a sequence in DB2, PostgreSQL, Oracle, SAP DB, or a generator in Interbase. The returned identifier is of type long, short or int

3. Assigned - lets the application to assign an identifier to the object before save() is called. This is the default strategy if no <generator> element is specified.

4. Foreign - uses the identifier of another associated object. Usually used in conjunction with a <one-to-one> primary key association.

5. <property> element

The <property> element declares a persistent, JavaBean style property of the class. The <property> elements represent all other attributes available in the domain object. type name could be:

1. The name of a Hibernate basic type (eg. integer, string, character, date, timestamp, float, binary, serializable, object, blob).

2. The name of a Java class with a default basic type (eg. int, float, char, java.lang.String, java.util.Date, java.lang.Integer, java.sql.Clob).

3. The name of a serializable Java class.

6. <many-to-one> element

An ordinary association to another persistent class is declared using a many-to-one element. The relational model is a many-to-one association: a foreign key in one table is referencing the primary key column(s) of the target table. A typical many-to-one declaration looks like this:

<many-to-one name="product" class="Product" column="PRODUCT_ID"/>

7. <one-to-one> element

A one-to-one association to another persistent class is declared using a one-to-one

element . A typical many-to-one declaration looks like this:

<many-to-one name="product" class="Product" column="PRODUCT_ID"/>

8.<natural-id> element

A natural key is a property or combination of properties that is unique and non-null. If it is also immutable, even better. Map the properties of the natural key inside the <natural-id> element.

9.<component>,<dynamic-component> element

The <component> element maps properties of a child object to columns of the table of a parent class.The <dynamic-component> element allows a Map to be mapped as a component, where the property names refer to keys of the map.

10.<properties> element

The <properties> element allows the definition of a named, logical grouping of properties of a class. The most important use of the construct is that it allows a combination of properties to be the target of a property-ref. It is also a convenient way to define a multi-column unique constraint.

11.<subclass>element

Finally, polymorphic persistence requires the declaration of each subclass of the root persistent class. For thetable-per-class-hierarchy mapping strategy, the <subclass> declaration is used.

Each subclass should declare its own persistent properties and subclasses. <version> and <id> properties are assumed to be inherited from the root class.

3) Hibernate code:

the hibernate code is written for the above created classes, objects with the mappings by using hibernate classes and tags which will communicate with the database.

5.3) Key features of Hibernate include:

1. Integrates elegantly with all popular J2EE application servers , web containers and in standalone applications .

- Hibernate is typically used in Java Swing applications, Java Servlet-based applications, or J2EE applications using EJB session beans

2. Free / open source

- Hibernate is licensed under the LGPL (Lesser GNU PublicLicense).critical component of the JBoss Enterprise Middleware System (JEMS) suite of products

3. Natural programming model

- Hibernate supports natural OO idiom; inheritance, polymorphism, composition and the Java collections framework

5. Extreme scalability

- Hibernate is extremely performant, has a dual-layer cache architecture, and may be used in a cluster

6. The query language

- Hibernate addresses both sides of the problem; not only how to get objects into the database, but also how to get them out again

7. EJB 3.0

- Implements the persistence API and query language defined by EJB 3.0 persistence

Chapter 6
6) Hibernate Query Language (HQL):

Hibernate is equipped with an extremely powerful query language that looks very much like SQL. Queries are case-insensitive, except for names of Java classes and properties. HQL is a language for talking about ¡§sets of objects¡¨. It unifies relational operations with object models. Make SQL be object oriented. It uses Classes and properties instead of tables and columns. It supports Polymorphism, Associations , Much less verbose than SQL .The Hibernate Query Language, designed as a minimal object-oriented extension to SQL, provides an elegant bridge between the object and relational worlds.
Key features include:

1. Integrates elegantly with all popular J2EE application servers, web containers and in standalone applications.
- Hibernate is typically used in Java Swing applications, Java Servlet-based applications, or J2EE applications using EJB session beans

2. Free / open source
- Hibernate is licensed under the LGPL (Lesser GNU Public License).critical component of the JBoss Enterprise Middleware System (JEMS) suite of products

3. Natural programming model
- Hibernate supports natural OO idiom; inheritance, polymorphism, composition and the Java collections framework

5. Extreme scalability
- Hibernate is extremely performant, has a dual-layer cache architecture, and may be used in a cluster

6. The query language
- Hibernate addresses both sides of the problem; not only how to get objects into the

database, but also how to get them out again

7. EJB 3.0

- Implements the persistence API and query language defined by EJB 3.0 persistence

Other features include:

1. Full support for relational operations
2. Inner/outer/full joins, Cartesian products
3. Projection
4. Aggregation (max, avg) and grouping
5. Ordering
6. Sub queries
7. SQL function calls

6.1) HQL Syntax

As described earlier, most of HQL's syntax and features are very similar to SQL. An HQL query may consist of following elements:

- Clauses
- Aggregate functions
- Subqueries

Chapter 7
7) Sample code:

1) Preparing database

USER_ID <<PK>> int(11)

USER_NAME varchar(20)

USER_ PASSWORD varchar(10)

USER_ EMAIL varchar(20)

2. Creating persistent java objects

Following code sample represents a java object structure which represents the User table.

Generally these domain objects contain only getters and setters methods.

Source code for User.java

```
public class User {

    /** identifier field */
    private Long id;

    /** persistent fields */
    private String userName;
    private String userPassword;
    private String userEmail;

    /** default constructor */
    public User() {
    }

    public Long getId() {
        return this.id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getUserName() {
        return this.userName;
    }
    public void setUserName(String userName) {
```

```
        this.userName = userName;
    }
    public String getUserPassword() {
        return this.userPassword;
    }
    public void setUserPassword(String userPassword) {
        this.userPassword = userPassword;
    }
    public String getUserEmail() {
        return this.userEmail;
    }
    public void setUserEmail(String userEmail) {
        this.userEmail = userEmail;
    }
}
```

3. Mapping POJO with persistence layer using hibernate mapping document

Each persistent class needs to be mapped with its configuration file. Following code represents Hibernate mapping file for User class.

Source code for User.hbm.xml

```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC

   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

 <class name="User" table="user">

     <id column="USER_ID" name="id" type="java.lang.Long">

       <generator class="increment"/>
```

```
    </id>

<property column="USER_NAME" length="20" name="userName" not-    null="true"
type="java.lang.String"/>

<property column="USER_PASSWORD" length="10" name="userPassword" not-
null="true" type="java.lang.String"/>

<property column="USER_EMAIL" length="20" name="userEmail"
type="java.lang.String"/>

  </class>

</hibernate-mapping>
```

Source code for hibernate.cfg.xml

```
<hibernate-configuration>

      <session-factory>
    <!-- Database connection settings -->
       <property name="show_sql">true</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect
</property>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver
</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/quickstart
</property>
<property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password"></property>

<!-- Mapping files -->
    <mapping resource="User.hbm.xml"/>

</session-factory>
</hibernate-configuration>
Configuration config = new Configuration().addResource(¡§User.hbm.xml");
Configuration config = new Configuration().addClass(User.class)
.setProperty("hibernate.dialect", "org.hibernate.dialect. MySQLDialect")
.setProperty("hibernate.connection.driver_class", " com.mysql.jdbc.Driver")
.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/quickstart");
 SessionFactory sessions = config.buildSessionFactory();
```

4. Querying the database

Source code for UserQueryHibernate.java

```
import java.util.Iterator;
import org.hibernate.*;
```

```
import org.hibernate.cfg.*;

public class UserQueryHibernate {
      public static void main(String[] args)
                        throws Exception {

      //Fire up Hibernate
      SessionFactory sessionFactory = new Configuration()
         .configure().buildSessionFactory();

      //Open Session
      Session session = sessionFactory.openSession();

      //Query using Hibernate Query Language
      String SQL_STRING = "  FROM User as users";
      Query query = session.createQuery(SQL_STRING);
      System.out.println("aftr createQuery" );
      for (Iterator it = query.iterate(); it.hasNext();) {
         User user = (User) it.next();
         System.out.println("User name " + user.getUserName() );
         System.out.println("User Email " + user.getUserEmail() );
         }

      //Close Session
      session.close();
        }
}
```

Chapter 8

8.1) Tools for writing HQL:

The *Hibernate Tools* currently include plugins for the Eclipse IDE as well as Ant tasks.

- ***Mapping Editor:*** An editor for Hibernate XML mapping files, supporting auto-completion and syntax highlighting. It also supports semantic auto-completion for class names and property/field names, making it much more versatile than a normal XML editor.
- ***Console:*** The console is a new view in Eclipse. In addition to a tree overview of your console configurations, you also get an interactive view of your persistent classes and their relationships. The console allows you to execute HQL queries against your database and browse the result directly in Eclipse.
- ***Development Wizards:*** Several wizards are provided with the Hibernate Eclipse tools; you can use a wizard to quickly generate Hibernate configuration (cfg.xml) files, or you may even completely reverse engineer an existing database schema into POJO source files and Hibernate mapping files. The reverse engineering wizard supports customizable templates.
- ***Hql editor:*** hql editor is used for the writing and running of the hql.

8.2)Advantages of Hibernate:

1) Writing queries are avoided.
2) Using ORM we can avoid the jdbc API completely and also provides ease to the developer in developing the classes.
3) Easily migrate your code between different databases. Good for updating, maintaining your application
4) Support for a wide range of databases including Oracle, DB2, Sybase, MS SQL Server, PostgreSQL, MySQL, HypersonicSQL, Mckoi SQL, SAP DB, Interbase, Point base, Progress, Front Base, Ingress, Informix, Firebird.
5) Free software and plugins are available in internet.

Chapter 9
9.1) Conclusion:

Hibernate Query Language (HQL) is a rich and powerful object-oriented query language available with the Hibernate O/R Mapping Framework. This query language, designed as a "minimal object-oriented extension to SQL," allows you to represent SQL queries in object-oriented terms—by using objects and properties of objects.

HQL provides many advanced features compared to SQL, yet is easier to learn and use as its syntax and basic features are very similar to SQL. It facilitates writing database-type independent queries that are converted to a local SQL dialect of the underlying database at runtime

Hibernate delivers a high performance, open source persistence framework comparable to many of its open source and commercial counterparts. Developers utilizing Hibernate can greatly reduce the amount of time and effort needed to code, test, and deploy applications.

9.2)Bibliography:

- http://www.hibernate.org
- http://hibernate.bluemars.net
- http://tools.hibernate.org
- http://forum.hibernate.org
- http://www.andromda.org