Hibernate Research Paper

Eurico Costa

CIT-360 W16

Hibernate -> High performance object/relational persistence and query service, licensed under open source GNU Lesses General Public License (LGPL) and free.

Takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), and also provides data query and retrieval capabilities.

On a typical application, access to a relational database would be provided by JDBC (Java Database Connectivity), which is an API for accessing databases from a Java program. This allows a program to execute SQL statements against a database and retrieve and persist data. The problem is that, by using JDBC, a programmer will have to deal with all the details of the implementation and of interacting with the database. It has a flexible architecture, but forces the programmer to do more implementation work during implementation.

# Pros and Cons of JDBC

|  |  |
| --- | --- |
| Pros | Cons |
| * Clean and simple SQL processing * Good performance with large data sets * Very good for small applications * Simple syntax easy to learn | * Complex if used in large projects * Large programming overhead * No encapsulation * Query is RDBMS specific |

# Object Relational Mapping (ORM)

ORM is a programming technique for converting data models between relational databases and object oriented programming languages such as Java, C#, etc. An ORM framework has the following advantages over plain JDBC:

* Business code accesses objects rather than tables
* Hides the details of SQL queries from the Object Oriented logic
* Based on JDBC under the hood
* No need to deal with the database implementation. Abstracts the database implementation detail from the actual business logic and application in general.
* Entities are based on business concepts rather than database structure
* Provides transaction management and automatic key generation.
* Fast development of the application by not having to spend too long architecting the data access layer.

An ORM usually contains the following four components:

* An API to perform basic CRUD operations on objects of persistent classes
* A language or API to specify queries that refer to classes and properties of classes
* A configurable facility for configuring mapping metadata
* A technique to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization features.

Hibernate is an ORM framework that supports these four basic components.

# Hibernate

Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.



Figure - High Level Hibernate Architecture

# Working with Hibernate

Hibernate is composed of a number of classes that are involved in the Hibernate Application Architecture:

## Configuration Object

Is the first object that is created in any hibernate application and usually created only once during the application initialization. It represents a configuration or properties file required by the framework. It provides two key components:

* Database Connection: handled through one or more config files supported by hibernate (hibernate.properties, hibernate.cfg.xml)
* Class Mapping Setup: Creates the connection between java classes and database tables.

## SessionFactory Object

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

The SessionFactory is an heavyweight object so it is usually created during the application start up and kept for later use. One SessionFactory object is required per database using a separate configuration file. So if the application uses multiple databases then one would have to create multiple SessionFactory objects.

## Session Object

A Session is used to get a physical connection with a database. The Session object is lightweight 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 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 (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.

# Using Hibernate

To use hibernate in an application:

* Download the Hibernate package from <http://www.hibernate.org/downloads>.
* Unzip the directory and extract the /lib directory to a location under the application CLASSPATH. All the jars should be included.
* Copy the file hibernate3.jar from the root directory of the extracted folder structure, into a location under the application CLASSPATH.

## Hibernate Config file for MySQL database (hibernate.cfg.xml)

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE hibernate-configuration SYSTEM

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">

org.hibernate.dialect.MySQLDialect

</property>

<property name="hibernate.connection.driver\_class">

com.mysql.jdbc.Driver

</property>

<!-- Assume test is the database name -->

<property name="hibernate.connection.url">

jdbc:mysql://localhost/test

</property>

<property name="hibernate.connection.username">

root

</property>

<property name="hibernate.connection.password">

root123

</property>

<!-- List of XML mapping files -->

<mapping resource="Employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

## Creating a Session Object

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

// do some work

...

tx.commit();

}

catch (Exception e) {

if (tx!=null) tx.rollback();

e.printStackTrace();

}finally {

session.close();

}

## A simple POJO example

The entire concept of Hibernate is to take the values from Java class attributes and persist them to a database table. A mapping document helps Hibernate in determining how to pull the values from the classes and map them with table and associated fields.

Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate. Hibernate works best if these classes follow some simple rules, also known as the Plain Old Java Object (POJO) programming model. There are following main rules of persistent classes, however, none of these rules are hard requirements.

All Java classes that will be persisted need a default constructor.

All classes should contain an ID in order to allow easy identification of your objects within Hibernate and the database. This property maps to the primary key column of a database table.

All attributes that will be persisted should be declared private and have getXXX and setXXX methods defined in the JavaBean style.

A central feature of Hibernate, proxies, depends upon the persistent class being either non-final, or the implementation of an interface that declares all public methods.

All classes that do not extend or implement some specialized classes and interfaces required by the EJB framework.

The POJO name is used to emphasize that a given object is an ordinary Java Object, not a special object, and in particular not an Enterprise JavaBean.

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>