Important Concepts Of Hibernate

22 May 2015

What is Hibernate?

Hibernate is the implementation of JPA (Java Persistence API). Hibernate is also called **Hibernate ORM**. Hibernate is an object-relational mapping framework for the Java language. Hibernate's primary feature is mapping from Java classes to database tables and vice versa. Hibernate supported query is known as Hibernate Query Language (HQL).

Core components of Hibernate

Below are the important elements of Hibernate

- **hibernate.cfg.xml:** This file has database connection details
- **hbm.xml or Annotation:** Defines the database table mapping with POJO. Also defines the relation between tables in java way.
- **SessionFactory:**
  - There will be a session factory per database.
  - The SessionFacory is built once at start-up
  - It is a **thread safe** class
  - SessionFactory will create a new Session object when requested
- **Session:**
  - The Session object will get physical connection to the database.
  - Session is the Java object used for any DB operations.
  - Session is not **thread safe**. Hence do not share hibernate session between threads
  - Session represents unit of work with database
  - Session should be closed once the task is completed

Standalone Hibernate program

The objective of this program is to

- refresh your knowledge of hibernate.cfg.xml (https://github.com/ashismo/repositoryForMyBlog/blob/master/hibernate/HibernateExample/src/main/resources/hibernate.cfg.xml) and entity class (https://github.com/ashismo/repositoryForMyBlog/blob/master/hibernate/HibernateExample/src/main/java/com/ashish/entity/EmployeeEntity.java)
- build session factory (https://github.com/ashismo/repositoryForMyBlog/blob/master/hibernate/HibernateExample/src/main/java/com/ashish/util/HibernateUtil.java) with a database
- insert a record (https://github.com/ashismo/repositoryForMyBlog/blob/master/hibernate/HibernateExample/src/main/java/com/ashish/main/MainApp.java) into a database and select the same record (https://github.com/ashismo/repositoryForMyBlog/blob/master/hibernate/HibernateExample/src/main/java/com/ashish/main/MainApp.java) from the database

Hibernate Example zip(18kb)
(https://github.com/ashismo/repositoryForMyBlog/blob/master/hibernate/HibernateExample.zip)

Hibernate Exampl
(https://github.com

Hibernate Caching

The performance of the database access is handled using **caching techniques**. Following are the two different caching techniques available in the hibernate.

- **First Level Caching** is the default caching in hibernate. It executes query at the end of the transaction to reduce the interaction with the DB. It is associated with the session object.
- **Second Level Caching** is associated with the **SessionFacory** object. It loads the entity objects at the SessionFacory level while executing a transaction. As a result, those objects are available to the entire application and do not bound to a single user.
  Second level caching is configured in hibernate.cfg.xml file. Following are the popular open source cache implementations
  - EH Cache (Easy Hibernate Cache)

- OS Cache
- Swarm Cache
- JBoss Tree Cache
- Infinispan

load() vs get() in hibernate

SL	load()	get()
1	**load()** is a method of hibernate session	**get()** is also a method of hibernate session
2	**load()** is used when we are sure that an object exists	**get()** is used when we are not sure about the existence of an object
3	Throws **exception** if unique id not found in the database	Returns **NULL** if the unique id not found in the database
4	**load()** returns a proxy by default and the database won't be hit until the proxy is invoked	**get()** hits the database immediately
5	Example: User user = (User)session.load(User.class, userId)	Example: User user = (User)session.get(User.class, userId)

update() vs merge() in Hibernate

update()	merge()
update() is used when a session does not have another persistence object with the same identifier	**merge()** is used when we want to merge our changes without considering the state of the session

Object states in Hibernate

In Hibernate, an object can remain in three states

- **Transient:** Newly created instance of a persistence class which is never associated with any hibernate session.
- **Persistence:** An object which is associated with the hibernate session is called **Persistence object**. Any change in the object will reflect in database upon the flash strategy i.e.
- automatic flush whenever any property of the persistence object changes **OR**
- explicit flush by calling **session.flush()**
- **Detached:** The object which was associated with session earlier but currently not associated with it are called **detached object**. We can reattach the detached object by calling **session.update()** or **session.saveOrUpdate()** method. Once the session will be closed then the same object will be **detached again**

Immutable entity class in hibernate

- Immutable entity class is a read-only entity so it does not allow application to update or delete.
- Mark @Entity(mutable=false) to make a class immutable. Default is mutable=true

Call stored procedure from hibernate

Suppose you have below stored procedure

```
CREATE PROCEDURE `GET_STOCK`(stockCode VARCHAR(20))
BEGIN
   SELECT * FROM STOCK WHERE STOCK_CODE=stockCode;
END
```

There are three different ways to call the stored procedure

Native SQL procedure call

```
<code class="language-java" data-lang="java>
Query q = session.ceateSQLQuery("CALL GET_STOCK(stockCode)")
                                .addEntity(Stock.class)
                                .setParameter("stockCode", "1111");

List result = q.list();

for(int i = 0; i < result.size(); i++) {
        Stock s = (Stock) result.get(i);
        System.out.println(s.getStockCode());
}
</code>
```

SQL procedure call by Named Query

query mapping in your **hbm.xml** file

```
<hibernate-mapping>
    <class name="com.ashish.Stock" table="STOCK">
        <id name='stockId' type="java.lang.Integer">
            <column="STOCK_ID"/>
            <generator class="identity"/>
        </id>


.....


        <sql-query name="stockStoredProcCall">
            <return alias="stock" class="com.ashish.Stock"/>
            <![CDATA[CALL GET_STOCK(:stockCode)]]>
        </sql-query>
    </class>
</hibernate-mapping>
```

Java code to call the procedure

```
Query query = session.getNamedQuery("stockStoredProcCall")
                                    .setParameter("stockCode", "1111");
List result = query.list();
for(int i=0; i<result.size(); i++){
        Stock stock = (Stock)result.get(i);
        System.out.println(stock.getStockCode());
}
```

SQL procedure call by Named Native Query

query mapping in your entity class (Stock.java)

```
@NamedNativeQueries({
        @NamedNativeQuery (
                name="stockStoredProcCall",
                query="CALL GET_STOCK(:stockCode)"
                resultClass=Stock.class
        )
})

@Entity
@Table(name="stock")
public class Stock implements Serializable {
        ....
        ....
}
```

```
Query query = session.getNamedQuery("stockStoredProcCall")
                                    .setParameter("stockCode", "1111");
List result = query.list();
for(int i=0; i<result.size(); i++){
        Stock stock = (Stock)result.get(i);
        System.out.println(stock.getStockCode());
}
```

Dirty read, Phantom Read and Non Repeatable Read

- **Dirty read** occurs when one transaction is changing records/tuple and second transaction is trying to read this tuple/record before the original change has been committed or rolled back. This is known as a dirty read scenario because there is always the possibility that the first transaction may rollback the change, resulting in the second transaction having read an invalid value.
- **Phantom read** occurs where in a transaction same query executes twice, and the second result set includes rows that weren't visible in the first result set. This situation is caused by another transaction inserting new rows between the execution of the two queries
- **Non Repeatable Reads** happen when in a same transaction same query yields different results. This happens when another transaction updates the data returned by other transaction.

Isolation and Propagation

- **Isolation:** The degree to which this transaction is isolated from the work of other transactions. For example, can this transaction see uncommitted writes from other transactions?
- **Propagation:** In case of propagation, the code will always run in a transaction scope. Create a new transaction or reuse one if availble.

Datasource

Datasource is a name given to the connection set up to a Database from a server. The name is commonly used when creating a query to the database. The advantages of using datasource are

- Improves performance as the connections are created and managed by the application server
- Provides the facilities of creating connection pool

If you want to configure a BasicDataSource for MySQL, you would do something like this

```
BasicDataSource dataSource = new BasicDataSource();

dataSource.setDriverClassName("com.mysql.jdbc.Driver");
dataSource.setUsername("username");
dataSource.setPassword("password");
dataSource.setUrl("jdbc:mysql://:/");
dataSource.setMaxActive(10);
dataSource.setMaxIdle(5);
dataSource.setInitialSize(5);
dataSource.setValidationQuery("SELECT 1");

</code></pre>
```

java-Java/J2EE Popular Topics [9] (/categories.html#java-Java/J2EE Popular Topics-ref)

Java/J2EE popular topics [8] (/tags.html#Java/J2EE popular topics-ref)

« Previous (/upcoming-spring%20security/2015/05/21/Spring-Security-And-LDAP-Integration) Archive (/archive.html)
Next » (/java-java/j2ee%20popular%20topics/2015/05/22/Important-Concepts-of-the-Spring-Framework)
