Hibernate Session

The session is the heart of Hibernate. Every Operation is associated with Session like insert, update, delete. Session Object is created from SessionFactory and it is a lightweight/noncostly Object.Hibernate talks with Database using Session, also Hibernate maintains a First Level cache in Session. The session is not thread- safe so it is advisable that not open session for a long time unless data can be dirty if we work in a multithreaded environment where multiple threads are involved with the same session.

Sessions also maintain the Transaction. A transaction is a unit of work which must be done in an atomic way. Either whole Transaction is the success and committed to database otherwise it is retracted.

From SessionFactory we can create a Session so Sessionfactory tracks the Session.

**Following Table provides some of the important methods of Session Interface.**

|  |  |
| --- | --- |
| Method name | Description |
| **beginTransaction()** | By calling this method Session returns a Transaction Object. Any set of operations done on this context treat as a unit of work. Either Success or Failure |
| **cancelQuery()** | Buy calling this method developer can cancel the execution of the current query. |
| **clear()** | This method clears any data cached in a session. |
| **close()** | By calling this session is going to close and release the JDBC connection and cleaning up the caches. |
| **createCriteria(Class persistentClass)** | Criteria Create a new Criteria instance by passing the class Object, we use Criteria for implement dynamic conditional query. |
| **createCriteria(String entityName)** | Criteria Create a new Criteria instance by passing the entity name, we use Criteria for implement dynamic conditional query. |
| **getIdentifier(Object object)** | By calling this method Hibernat**e** returns the identifier value of the given entity which is currently attach with this session. |
| **createQuery(String queryString)** | This method returns a new instance of Query from the passed HQL query String.. |
| **createSQLQuery(String queryString)** | This method returns a new instance of SQL Query for the passed SQL query string. |
| **void delete(Object object)** | By calling this we can remove the passing entity from Database. |
| **delete(String entityName, Object object)** | By calling this we can remove the passing entity from Database but it takes the entity name and the Object. |
| **get(String entityName, Serializable id)** | Used for eager fetching. It always hit the database if Object not found in database returns null |
| **getSessionFactory()** | Get the Sessionfactory from which current Session has been created. |
| **refresh(Object object)** | By calling refresh Hibernate reads the Object's state from database although it has been present in Session |
| **getTransaction()** | Get the transaction attached with this session. |
|  |  |
| **boolean isDirty()** | This method returns true if the Entity Object has been modified but underlying database row for this entity is not modified. |
| **save(Object object)** | Used for save a POJO/Transient object into database and attach a identifier with it. |
| **saveOrUpdate(Object object)** | If Object is not found in database it creates a new one and generates an Identifier for it else update the state of the row in database. |
| **update(Object object)** | It will update the entity and save the updates state on database while closing /flushing the session. |
| **load(String entityName, Serializable id)** | Used for Lazy loading, when developer tries to load an Object from database it does not hit database rather return a Proxy Object. Later when developers call any method on that proxy then actual database calls happens. If the Object is not found in database throws ObjectNotFoundException |
| **merge(Object object)** | It also used for update but the main difference from update is after merge it returns a new Object which will copy of Original object and copied the updated values, so it does not touched the original entity, and this Copied entity will be tracked by Hibernate so any further changes on copied entry is being tracked but if any changes does on original entity it is not tracked by Hibernate so that will not reflect in database while closing or flushing the session. |

**Difference between load and get**

**load():** Used for Lazy loading, when a developer tries to load an Object from the database it does not hit database rather return a Proxy Object. Later when developers call any method on that proxy then actual database calls happens. If the Object is not found in database throws ObjectNotFoundException.

**get() :** Used for eager fetching. It always hit the database if Object not found in database returns null.

let See with an Example

Step 1. Create an Employee entity

package com.example.hibernate.entity;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.persistence.UniqueConstraint;

@Entity

@Table(name = "Employee", uniqueConstraints = {@UniqueConstraint(columnNames = "ID"), @UniqueConstraint(columnNames = "EMAIL")})

public class Employee

{

@Id

@Column(name = "ID", unique = true, nullable = false)

private Integer empId;

@Column(name = "EMAIL", unique = true, nullable = false, length = 100)

private String email;

@Column(name = "NAME", unique = false, nullable = false, length = 100)

private String name;

public Integer getEmpId()

{

return empId;

}

public void setEmpId(Integer empId)

{

this.empId = empId;

}

public String getEmail()

{

return email;

}

public void setEmail(String email)

{

this.email = email;

}

public String getName()

{

return name;

}

public void setName(String name)

{

this.name = name;

}

@Override

public String toString() {

return "Employee [empId=" + empId + ", email=" + email + ", name=" + name + "]";

}

}

Step 2: Crte the Test class to insert data and fetch data by get and load.

package com.example.hibernate.test;

import org.hibernate.Session;

import com.example.hibernate.core.HibernateUtil;

import com.example.hibernate.entity.Employee;

public class TestApplication {

public static void main(String[] args) {

TestApplication app = new TestApplication();

app.insertEmployee();

app.findEmployeeById(1);

app.loadEmployeeById(1);

HibernateUtil.shutdown();

}

public void insertEmployee() {

Session session = HibernateUtil.getSessionFactory().openSession();

session.beginTransaction();

Employee emp = new Employee();

emp.setEmpId(1);

emp.setEmail("mitrashamik@mail.com");

emp.setName("Shamik Mitra");

session.save(emp);

session.getTransaction().commit();

session.close();

}

public void findEmployeeById(Integer id) {

Session session = HibernateUtil.getSessionFactory().openSession();

Employee emp2 = (Employee) session.get(Employee.class, id);

System.out.println("Successfully Fetch " + emp2);

session.close();

}

public void loadEmployeeById(Integer id) {

Session session = HibernateUtil.getSessionFactory().openSession();

Employee emp2 = (Employee) session.load(Employee.class, id);

System.out.println("Successfully lazy Fetch");

System.out.println("No query fired on database , Proxy returned.");

System.out.println("Now query fired on database: " + emp2.getName());

session.close();

}

}

**Output:**

Hibernate: drop table Employee if exists

Hibernate: create table Employee (ID integer not null, EMAIL varchar(100) not null, NAME varchar(100) not null, primary key (ID))

Hibernate: alter table Employee add constraint UK\_ardf0f11mfa6tujs3hflthwdv unique (EMAIL)

===================================================================

Hibernate: insert into Employee (EMAIL, NAME, ID) values (?, ?, ?)

Hibernate: select employee0\_.ID as ID1\_0\_0\_, employee0\_.EMAIL as EMAIL2\_0\_0\_, employee0\_.NAME as NAME3\_0\_0\_ from Employee employee0\_ where employee0\_.ID=?

Successfully Fetch Employee [empId=1, email=mitrashamik@mail.com, name=Shamik Mitra]

===================================================================

Successfully lazy Fetch

No query fired on database , Proxy returned.

Hibernate: select employee0\_.ID as ID1\_0\_0\_, employee0\_.EMAIL as EMAIL2\_0\_0\_, employee0\_.NAME as NAME3\_0\_0\_ from Employee employee0\_ where employee0\_.ID=?

Now query fired on data base: Shamik Mitra

If we look the Output we can see after inserting Employee Objects in the in-memory database, when we try to retrieve the Employee Object with Id 1 using get, It immediately query the database and query are printed in the console.

But when we retrieve the Employee Object with Id 1 using load, It does not fire the database immediately but when we try to get the name of the employee at that time database will be accessed.

So, If we find a situation where the only id of an Object is sufficient to perform a task it is better to use load as it is not hit the database so performance can be improved.