Hibernate

1. Steps to create hibernate application…?

1**1. Create the persistence POJO class.**

**2. Create Hibernate mapping file.(Student.hbm.xml)**

**3. Create configuration files.(hibernate.cfg.xml)**

**4. Create class that retrieve and store the persistence object.**

**5. Load the jar.**

**6. Run the Hibernate Application.**

\* Data Persistency…?

1. Representing data permanently in the back-end system is called data persistency.

Technology Used :

1. Serialization & Deserialization.
2. JDBC
3. ORM :

Technology used : Hibernate, JPA, Open JPA, EJB-Entity Beans, Toplink, iBatis.

Hibernate Mapping File :

The main intention is to provide mapping between object oriented data model elements and Relational data model elements like :

Class -------------> Table

ID ------------------> Primary Key

Properties ------------> Column

In Single Hibernate application, is it possible to provide more than one mapping file :

Yes, it is possible to provide more then one mapping file in single Hibernate application.

Note : In general we will prepare a separate mapping file for each & every Pojo class.

Is it mandatory to provide mapping file in Hibernate application for the entities configuration :

it is Upto 3.2.4V XML is mandatory but form 3.2.5V it is optional. Alternative we can use Annotation.

Hibernate Interfaces :

Hibernate Configuration file :

The main intention of configuration object is to manage all the configuration details which are required by the hibernate software in order to interact with database.

It is an XML documents, it will provide all the Hibernate configuration details like :

1. Driver class configuration.
2. Driver URL configuration.
3. Database user name configuration.
4. Database password configuration.
5. Dialect class configuration. (The dialect specifies the type of database used in hibernate so that hibernate generate appropriate type of SQL statements. For connecting any hibernate application with the database, it is required to provide the configuration of SQL dialect.)
6. Mapping file configuration.
7. Connection pooling configuration.
8. Cache mechanisim confiuration.
9. Transaction Isolation level configuration.
10. Annotation support is not available for Hibernate configuration file.

To represent configuration object Hibernate has provided a separate predefined class in form of = org.hibernate.cfg.configuration.

Configuration cfg = new Configuration()

( It is just created a object in heap memory no data will be store in memory. After calling cfg.configure() method it will store data in memory)

It is Heavy weight object & it is Singleton object upto a particulate database & It is thread-safe upto a particular database.

Is it possible to create more then one configuration object in a single Hibernate application….??

Yes, it is possible more than one configuration but, we have to use more than one configuration file and more than one DB.

Hibernate SessionFactory :

1. The Main Intention of SessionFactory is to get all the required configuration details from configuration file & it will load and register the driver and it will establish connection between Hibernate application and database.
2. It is factory for session object.
3. To represent SessionFactory object Hibernate have provided a predefined Interface int the form of = org.hibernate.SessionFactory.

SessionFactory sf = cfg.buildSessionFactory();

1. SessionFactory is thread upto a single database.
2. It holds second level cache.

Hibernate Session Object:

1. The Main Intention of Session object is to create an environments in order to perform Database operation.
2. It is like statement created in JDBC.
3. To represent Session object Hibernate have provided a predefined Interface in the form of = org.hibernate.Session.
4. To get session object we have to use the following method form SessionFactory :

Session session = sf.openSession();

1. It is not a thread safe.
2. It holds first level cache.

Hibernate Transaction Object:

1. In case of JDBC connection is available in auto commit mode by default.
2. But in Hibernate connection are in non-auto commit mode.
3. We have to perform commit & rollback operation.
4. For non-select query we have to perform transaction operation.
5. To Perform operation Hibernate provide predefined interface :

org.hibernate.Transaction.

Transaction tx = session.beginTransaction();

tx.begin();

Difference between save() and persist() method…?

|  |  |
| --- | --- |
| save() | persist() |
| 1. Both the method are used to insert a persistence object record in database, but if we use save() method then hibernate software will insert a record in DB and return primary key value. | 1. If we use persist() method then it will insert a record in DB and it will not return primary key value. |
| 2. public serializable save(Object obj) | 2. public void persist(Object obj) |

Difference between update() and persist() method…?

|  |  |
| --- | --- |
| update() | saveOrUpdate() |
| 1. In hibernate application if we use update() operation to perform updation then hibernate software will check whether the respective record is existed or not in the Database, if the record is existed then hibernate software will perform updation in the respective record. If the respective record is not existed in database then hibernate software raise an exceptions. | 1. In hibernate application if we use saveOrUpdate() operation to perform updation then hibernate software will check whether the respective record is existed or not in the Database, if the record is existed then hibernate software will perform updation in the respective record. If the respective record is not existed in database then hibernate software will perform save operation. |
| 2. public void update(Object obj) | 2. public void saveOrUpdate(Object obj) |

Difference between get() and load() method…?

|  |  |
| --- | --- |
| get() | load() |
| 1. It is early loading, performance is slow. | 1. It is lazy loading, and hence performance is faster. |
| 2. if object not found for the given identifier then it will return null object. | 2. if object not found then it will throw objectNotFoundException. |
| 3. After creating object it will direct interact with database. | 3. It will not interact with database after creating object, it will create proxy object and after calling getName() and other fields then it will interact with database |
| 4. It should be used if you are not sure that the data exists in the database or not. | 4. It should be used if you are sure that the data exists in the database. |

Composite key in Hibernate :

1. **Create a class and define two or more field in it that you want to make it unique and extend that class with serializable and override equals method and hash code method, and @classId(EmployeeId.class) and add two Id annotation for both fields. (First Approach).**
2. **Create a class and define two or more field in it and extend that class with serializable and override equals method and hash code method, and add @Embeddable annotation in this new class and define this class in main Employee class.(Second Approach)**

Foreign Key In Hibernate :

1. **Consider a class forming one to many mapping, primary class is customer and contains product list(Product Class).**

* Class Customer{
* @Id
* @Generated Value
* Filed…………
* @OneToMany(targetEntity = Product.class, cascade= CascadeType.ALL)
* @joinColumn(name=”product\_foreginKey”, referenceColumnName =”id”)// This foreign key column
* }

Primary key and foreign key…?

1. For primary key and foreign key we can use Inheritance mapping Table per sub class. So we can use below annotation for that :
2. In parent Table :

@Inheritance(strategy = InheritanceType.*JOINED*)

1. In Child table :

@PrimaryKeyJoinColumn(name = "id")

4. Difference between openSession and getCurrentSession…?

1**1. getCurrentSession method returns the session bound to context. Since this object belongs to context of the hibernate, it is ok if we don’t close it. Once the sessionFactory closed, this session object gets closed. It is created new session object if not exists else use same session object which is current hibernate context.**

**2. openSession() method helps in opening new session. You should close this session object once you are done with all database operation. Ans also, you should open a new session object for each request in a multi-threaded environment. It is always create new session.**

6. What is Hibernate Cache and It’s Type…????

**1. First level cache.**

**i. Hibernate caches query data to make our application faster and improve performance.**

**ii. The idea behind cache is to reduce the number of database queries.**

**iii. Hibernate first level cache is associated with the session object.**

**iv. Hibernate first level catch is enabled by default and there is not way to disable it.**

**v. Still Hibernate provides methods through which we can delete selected object from the cache or clear the cache completely.**

**vi. Any object cached in a section will not be visible to other sessions and when the session is closed, all the cached objects will be lost.**

**2. Second level cache :**

**i. Hibernate second level cache is disable by default, but we can enable it through configuration.**

**ii. Currently EHCache and Infinispan provides implementation for Hibernate second cache and we can use them.**

**iii. Add hibernate-ehcache dependency in your maven project, if it’s not maven then add corresponding jars.**

**<dependency>**

**<groupId>org.hibernate</groupId>**

**<artifactId>hibernate-ehcache</artifactI**

**</dependency>**

**iv. Add below property in hibernate configuration file :**

**<property name**="hibernate.cache.use\_second\_level\_cache"**>**true**</property>**

**<property name**="hibernate.cache.region.factory\_class"**>**org.hibernate.cache.ehcache.EhCacheRegionFactory**</property>**

**<property key**="hibernate.cache.use\_query\_cache"**>**true**</property>**

**Query level cache in not work with first level cache it is always associate with second level cache only.**

**<property name**="net.sf.ehcache.configurationResourceName"**>**/ehcache.xml**</property>**

**v. Create EHCache configuration file, a sample file configuration.xml.**

**vi. Use your cache, Annotate entity beans with @Cache annotation and caching strategy to use**

**@Entity**

**@Table( name = “Address”)**

**@Cache( usage = cacheConcurrencyStrategy.READ\_ONLY, region=”employee”)**

**Public class Address{}**

### 6. What are the states of the object in hibernate…????

There are mainly four states of the Hibernate Lifecycle :

1. **Transient State**
2. **Persistent State**
3. **Detached State**
4. **Removed State**

**1. Transient State :**

**The transient state is the first state of an entity object. When we instantiate an object of pojo classes using the new operator then the object is in the transient state. This object is not connected with any hibernate session. As it is not connected to any Hibernate Session, so this state is not connected to any database table. So, if we make any changes in the data of the POJO Class then the database table is not altered. Transient objects are independent of Hibernate, and they exist in the heap memory.**

**There are two layouts in which transient state will occur as follows:**

1. **When objects are generated by an application but are not connected to any session.**
2. **The objects are generated by a closed session.**

**2. Persistent State :**

**Once the object is connected with the Hibernate Session then the object moves into the Persistent State. So, there are two ways to convert the Transient State to the Persistent State :**

1. **Using the hibernated session, save the entity object into the database table.**
2. **Using the hibernated session, load the entity object into the database table.**

**In this state. each object represents one row in the database table. Therefore, if we make any changes in the data then hibernate will detect these changes and make changes in the database table.**

**Following are the methods given for the persistent state:**

* **session.persist(e);**
* **session.save(e);**
* **session.saveOrUpdate(e);**
* **session.update(e);**
* **session.merge(e);**
* **session.lock(e);**

**3. Detached State :**

**For converting an object from Persistent State to Detached State, we either have to close the session or we have to clear its cache. As the session is closed here or the cache is cleared, then any changes made to the data will not affect the database table. Whenever needed, the detached object can be reconnected to a new hibernate session. To reconnect the detached object to a new hibernate session, we will use the following methods as follows:**

**Following are the methods used for the detached state :**

* **session.detach(e);**
* **session.evict(e);**
* **session.clear();**
* **session.close();**

**4. Removed State :**

**In the hibernate lifecycle it is the last state. In the removed state, when the entity object is deleted from the database then the entity object is known to be in the removed state. It is done by calling the *delete() operation*. As the entity object is in the removed state, if any change will be done in the data will not affect the database table.**

***Note: To make a removed entity object we will call session.delete().***

##### **What is Cascade…??**

Cascade is the feature provided by hibernate to **automatically manage the state of mapped entity** whenever the state of its relationship owner entity is affected.

In other words,  
When **relationship owner is saved**,then **associated mapped entity**should also be **saved** and when relationship owner is **deleted**, associated **mapped entity** should also be **deleted** etc.

**Cascade Typs :**

1. **CascadeType.ALL**

**2. CascadeType.PERSIST**

**3. CascadeType.MERGE**

**4. CascadeType.REMOVE**

**5. CascadeType.REFRESH**

**6. CascadeType.DETACH**

**7. CascadeType.REPLICATE**

**8. CascadeType.SAVE\_UPDATE**

OneToMany Mapping…????

1. To create oneToMany relation we have to create two entity classes like below:

Class Question{

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name = "emp\_id")

**private** **int** id;

@OneToMany(cascade = CascadeType.***ALL***)

@JoinColumn(name = "question\_id")

@OrderColumn(name = "type")

**private** List<Answer> answer;

}

**class** Answer {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** qid;

**private** String answername;

}

ManyToMany Mapping…????

**class** Question1 {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String qname;

@ManyToMany(targetEntity = Answer1.**class**, cascade = CascadeType.***ALL***)

@JoinTable(name = "q\_ans\_Many", joinColumns = { @JoinColumn(name = "q\_id") }, I inverseJoinColumns = {

@JoinColumn(name = "ans\_id") })

List<Answer1> answers;

}

**class** Answer1 {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

**private** String answername;

**private** String postedBy;

}

OneToOne Mapping…????

1. While making One To One mapping we have define @OneToOne mapping annotation in the both class Employee and Address.
2. In this case two table will be created.

**class** Employee {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@PrimaryKeyJoinColumn

**private** **int** employeeId;

**private** String name;

**private** String email;

@OneToOne(targetEntity = Address.**class**, cascade = CascadeType.***ALL***)

**private** Address address;

}

**class** Address {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** addressId;

**private** String city;

@OneToOne(targetEntity = Employee.**class**, cascade = CascadeType.***ALL***)

**private** Employee employee;

}