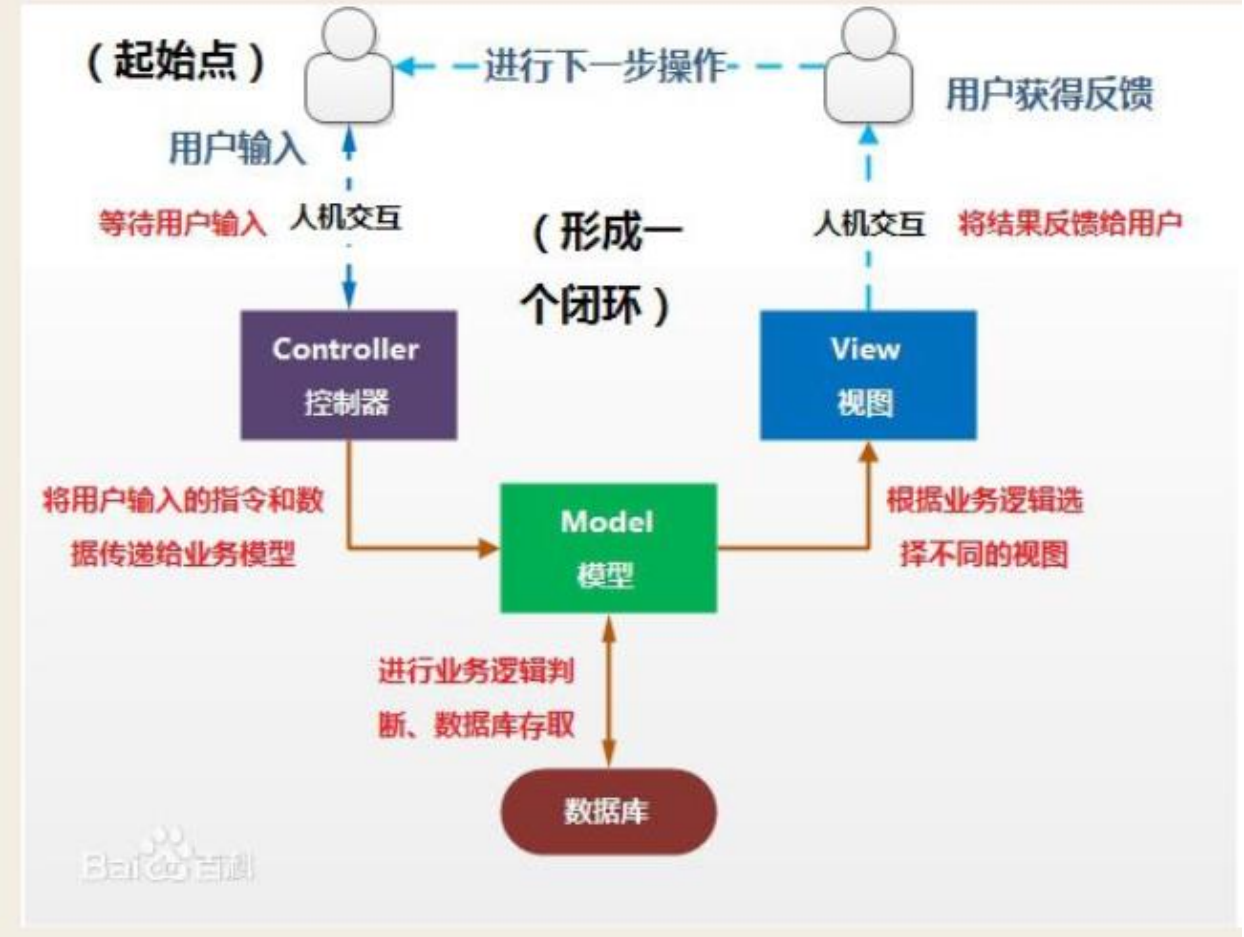
1、什么是MVC

（1）Model(业务模型)：应用程序中用于处理应用程序数据逻辑的部分，通常模型对象负责在数据库中存取数据 。

（2）view(视图)：应用程序中处理数据显示的部分。通常视图是依据模型数据创建的。

（3）Controller(控制器)：应用程序中处理用户交互的部分。通常控制器负责从视图中读取数据，控制用户的输入，并向模型发送数据。



2、Hibernate工作原理及为什么要用

工作原理：

（1）通过Configuration().configure(); 读取并解析hibernate.cfg.xml配置文件

（2）由hibernate.cfg.xml中的<mapping resource="com/xx/User.hbm.xml"/>读取并解析映射信息

（3）通过config.buildSessionFactory(); // 创建SessionFactory

（4）sessionFactory.openSession(); //打开Session

（5）session.beginTransaction(); //创建事务Transation

（6）persistent.operate //持久化操作

（7）session.getTransaction().commit(); //提交事务

（8）关闭Session

（9）关闭SessionFactory

hibernate.xml中：

```
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/fruit_db</property
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">ok</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">false</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping resource="com/gem/fruit/pojo/Fruit.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

POJO.hbm.xml:

```
<hibernate-mapping package="com.gem.fruit.pojo">
    <class name="Fruit" table="t_fruit">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="fname"/>
        <property name="price"/>
        <property name="count"/>
        <property name="remark"/>
    </class>
</hibernate-mapping>
```

DAOImpl中：

```
{
    Configuration cfg = new Configuration().configure();
    factory = cfg.buildSessionFactory();
}

private SessionFactory factory ;
private Session session ;

public List<Fruit> getFruitList() {
    session = factory.openSession();
    Query query = session.createQuery("from Fruit");
    return query.list();
}

public List<Fruit> getFruitList(int pageSize, int pageNum) {
    return getFruitList(null, pageSize, pageNum);
}

public int getFruitsCount() {
    session = factory.openSession();
    Query query = session.createQuery("select count(*) from Fruit");
    return ((Long)query.uniqueResult()).intValue();
}

public List<Fruit> getFruitList(String keyword, int pageSize, int pageNum) {
    if(StringUtil.isEmpty(keyword)){
        keyword = "" ;
    }
    String hql = "from Fruit where fname like ? or remark like ? " ; //limit ? , ? " ;
    session = factory.openSession();
    Query query = session.createQuery(hql);
    //Query<Fruit> query = session.createQuery(hql, Fruit.class);

    query.setString(0, "%"+keyword+"%");
    query.setString(1, "%"+keyword+"%");
```

为什么要用hibernate:

（1）对JDBC访问数据库的代码进行了封装，大大简化了数据访问层繁琐的重复性代码

（2）Hibernate是一个基于JDBC的主流持久化框架，是一个优秀的ORM实现。他它很大程度的简化DAO层的编码工作

（3）hibernate使用java反射机制，而不是字节码增强程序来实现透明

（4）性能好

hibernate是什么?

hibernate是基于ORM对象关系映射（完成对象数据到关系数据映射的机制）实现的，做数据持久化的工具

如何优化hibernate?

（1）使用双向一对多关联，不使用单向一对多

（2）灵活使用单向一对多关联

（3）不用一对一，用多对一

（4）配置对象缓存，不使用集合缓存

（5）表的字段要少，表关联不要怕多，有二级缓存

3、Hibernate是如何延迟加载的

a）实体对象　b）集合（Collection）

当hibernate在查询数据的时候，数据并没有存在内存中，当程序真正对数据的操作时，对象才存在于内存中，就实现了延迟加载，它节省了服务器的内存开销，从而提高了服务的性能。

4、Hibernate中的对象的状态及如何转化的

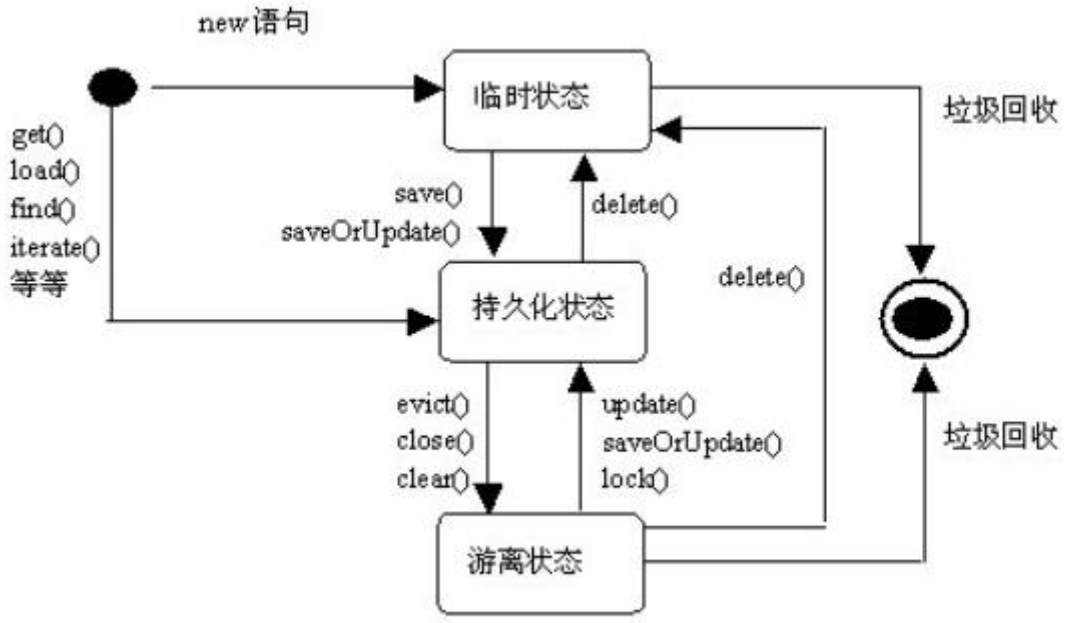
hibernate中对象的状态有三种：瞬时、游离和持久态，三种状态转化的方法都是通过session来调用。

瞬时到持久态的方法有：save()、saveOrUpdate()、get()、load();

持久到瞬时态的方法有：delete();

游离到持久态的方法有：update()、saveOrUpdate() 、lock();

持久到游离态的方法有：session.close()、session.evict()、session.clear().



5、hibernate中get() 和 load() 的区别

（1）get() 加载方法：不管我们 使不使用这个对象，此时都会发出sql语句从数据库中查询出来。

（2）load()加载方法：当我们使用session.load() 方法来加载一个对象时，此时并不会发出sql语句（因为有延迟加载），当前得到的这个对象其实是一个代理对象，这个代理对象只保存了实体对象的id值，只有当我们真正使用这个对象的时候，得到它的属性 时，这个时候才会发出sql语句，从数据库中去查询我们的对象。

注意：

(1)、如果使用get方式来加载对象，当我们试图得到一个id不存在的对象时，此时会报NullPointexception

(2)、如果使用load() 方法加载对象，当我们试图得到一个不存在的对象时，此时会报ObjectNotFoundException

为什么使用load的方式和get的方式来得到一个不存在的对象报的异常不同呢？其原因是因为load的延迟加载机制，使用load时，此时的User对象是一个代理对象，仅仅保存了当前的这个id值，当我们试图得到这个对象的UserName属性时，这个属性其实是不存在的，所以就会报ObjectNotFoundException

(3)还是因为load的延迟加载机制，当我们通过load()方法来加载一个对象时，此时并没有发出sql语句去从数据库中查询出该对象，当前这个对象仅仅是一个只有id的代理对象，我们还并没有使用该对象，但是此时我们的session已经关闭了，所以当我们在测试用例中使用该对象时就会报LazyInitializationException这个异常了。

解决这个的方法有两种，一种是将load改成get的方式来得到该对象，另一种是在表示层来开启我们的session和关闭session。

6、谈一谈Hibernate中list方法和iterate方法的区别

（1）使用list（）方法获取查询结果，每次发出一条查询语句，获取全部的数据

（2）iterate方法获取查询结果，先发出一条Sql语句用来查询满足条件数据的id，然后依次按这些id查询记录，也就是要执行N+1条SQL语句

7、hibernate中为什么使用缓存，及个缓存的各自的特点

如果通过list（）方法来获取得到对象，hibernate会发出一条sql语句将所有对象查询出来；

如果通过iterator（）方法来获取对象的时候，hibernate首先会发出一条sql语句查询出所有对象的id值，当我们如果要查询到某个对象的具体信息的时候，hibernate此时会根据查询出来的id值再发sql语句去从数据库中查询对象的信息，这就是典型的N+1的问题，**以下解决办法：**

一级缓存：我们考虑这样一种情况，如果我们需要在一个session当中要两次查询出很多对象，此时我们如果写两条 list()时，hibernate此时会发出两条 sql 语句，而且这两条语句

是一样的，但是我们如果第一条语句使用 list()，而第二条语句使用 iterator()的话，此时我们也会发两条sql语句，但是第二条语句只会将查询出对象的id，所以相对应取出所有的对象而已，显然这样可以节省内存，而如果再要获取对象的时候，因为第一条语句已经将对象都查询出来了，此时会将对象保存到session的一级缓存中去，所以再次查询时，就会首先去缓存中查找，如果找到，则不发sql语句了。这里就牵涉到了接下来这个概念:hibernate的一级缓存。

hibernate的一级缓存是session级别的，所以如果session关闭后，缓存就没了，此时就会再次发sql去查数据库。

二级缓存：是sessionFactory级别的缓存，需要配置；当我们session关闭以后，我们再去查询对象的时候，此时hibernate首先回去二级缓存中查询是否有该对象，有就不会再发sql了；

二级缓存缓存的仅仅是对象，如果查询出来的是对象的属性，就不会加到缓存中。

```
@Test
public void testCache2()
{
    Session session = null;
    try
    {
        session = HibernateUtil.openSession();

        /**
         * 注意：二级缓存中缓存的仅仅是对象，而下面这里只保存了姓名和性别两个字段，所以 不会被加载到二级缓存里面
         */
        List<Object[]> ls = (List<Object[]>) session
                .createQuery("select stu.name, stu.sex from Student stu")
                .setFirstResult(0).setMaxResults(30).list();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        HibernateUtil.close(session);
    }
    try
    {
        /**
         * 由于二级缓存缓存的是对象，所以此时会发出两条sql
         */
        session = HibernateUtil.openSession();
        Student stu = (Student) session.load(Student.class, 1);
        System.out.println(stu);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

查询缓存：当我们如果通过list（）方法查询两次对象时，二级缓存虽然会缓存查询出来对象，但是我们看到发出了两条相同的查询语句，这是因为二级缓存不会缓存sql查询语句，要解决这个问题，我们就要 配置查询缓存：

四、查询缓存(sessionFactory级别)

我们如果要配置查询缓存，只需要在hibernate.cfg.xml中加入一条配置即可：

```
        <!-- 开启查询缓存 -->
        <property name="hibernate.cache.use_query_cache">true</property>
```

然后我们如果在查询hql语句时要使用查询缓存，就需要在查询语句后面设置这样一个方法:

```
List<Student> ls = session.createQuery("from Student where name like ?")
                    .setCacheable(true)    //开启查询缓存，查询缓存也是SessionFactory级别的缓存
                    .setParameter(0, "%王%")
                    .setFirstResult(0).setMaxResults(50).list();
```

8、描述hibernate中实体间的关系及其注解配置

```
 */
@Entity
@Table(name = "user_login_log")
public class UserLoginLog implements java.io.Serializable {

    private static final long serialVersionUID = 8686390190132322570L;

    private Integer loginId;
    private String ip;
    private String address;
    private Integer type;
    private Integer status;
    @DateTimeFormat( pattern = "yyyy-MM-dd HH:mm:ss" )
    private Date loginTime;

    //private UserBaseInfo userBaseInfo;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "login_id", unique = true, nullable = false)
    public Integer getLoginId() {
        return loginId;
    }

    public void setLoginId(Integer loginId) {
        this.loginId = loginId;
    }

    @Column(name = "ip", nullable = true)
    public String getIp() {
        return ip;
    }
```

实体类组件以及注解标签映射关系：

实体类----------@Entity / @Table----------数据表

Id----------@Id----------主键

普通属性---------@Column----------普通键

集合属性--------@OneToMany / @ManyToOne / @ManyToMany / @OneToOne------外键

A、写在类声明之前有：

a. @Entity.以表明此Bean为EntityBean.每一个持久化POJO类都是一个实体Bean 。

b. @Table(name="TableName"),表示此实体Bean对应的数据库表名。@Table是类一级的注解，通过@Table注解可以为实体Bean映射指定表（Table）,目录（catelog）和schema的名字。

B、写在getXxx（）方法声明之前的有：

a. @Column注释定义了映射到列的所有属性，如列名是否唯一，是否允许为空，是否允许更新等

```
(1)    name 可选,列名(默认值是属性名)
(2)    unique 可选,是否在该列上设置唯一约束(默认值false)
(3)    nullable 可选,是否设置该列的值可以为空(默认值false)
(4)    insertable 可选,该列是否作为生成的insert语句中的一个列(默认值true)
(5)    updatable 可选,该列是否作为生成的update语句中的一个列(默认值true)
(6)    columnDefinition 可选: 为这个特定列覆盖SQL DDL片段 (这可能导致无法在不
(7)    table 可选,定义对应的表(默认为主表)
(8)    length 可选,列长度(默认值255)
(8)    precision 可选,列十进制精度(decimal precision)(默认值0)
(10)   scale 可选,如果列十进制数值范围(decimal scale)可用,在此设置(默认值0)
```

b. @Id 注释指定personId 属性为表的主键，它可以有多种生成方式：

```
·TABLE：容器指定用底层的数据表确保唯一。

    ·SEQUENCE：使用数据库的SEQUENCE 列来保证唯一

    ·IDENTITY：使用数据库的INDENTIT列来保证唯一

   ·AUTO：由容器挑选一个合适的方式来保证唯一

   ·NONE：容器不负责主键的生成，由调用程序来完成。
```

c. @GeneratedValue注释定义标识字段的生成方式

d. @Version 映射版本号属性

e. @Column 指定属性对应的列的信息

f . @Temporal 指定日期时间的类型(TIMESTAMP,DATE,TIME)

g. 简单属性可以不用注解。默认就是@Basic

h. @Transient 指定属性不需要映射

i. 复杂属性：关联，继承，组件，联合主键，集合

1>targetEntity

Class 类型的属性。定义关系类的类型，默认是该成员属性对应的类类型，所以通常不需要提供定义。

2>mappedBy

String 类型的属性。定义类之间的双向关系。如果类之间是单向关系，不需要提供定义，如果类和类之间形成双向关系，我们就需要使用这个属性进行定义，否则可能引起数据一致性的问题。

3>cascade

CascadeType[]类型。

该属性定义类和类之间的级联关系。定义的级联关系将被容器视为对当前类对象及其关联类对象采取相同的操作，而且这种关系是递归调用的。

4>fatch

FetchType 类型的属性。

可选择项包括：FetchType.EAGER和FetchType.LAZY。

前者表示关系类(本例是OrderItem类)在主类(本例是Order类)加载的时候同时加载；

后者表示关系类在被访问时才加载。默认值是FetchType. LAZY。

@OrderBy(value = "id ASC")注释指明加载OrderItem 时按id 的升序排序。

optional 属性是定义该关联类对是否必须存在，值为false 时，关联类双方都必须存在，如果关系被维护端不存在，查询的结果为null。值为true 时，关系被维护端可以不存在，查询的结果仍然会返回关系维护端，在关系维护端中指向关系被维护端的属性为null。

optional属性的默认值是true。

举个例：某项订单(Order)中没有订单项(OrderItem)，如果optional 属性设置为false，获取该项订单（Order）时，得到的结果为null，如果optional属性设置为true，仍然可以获取该项订单，但订单中 指向订单项的属性为null。

实际上在解释Order 与OrderItem的关系成SQL时，optional属性指定了他们的联接关系

optional=false 联接关系为inner join,

optional=true 联接关系为left join。

@JoinColumn(name = "order_id")注释指定OrderItem 映射表的order_id 列作为外键与Order 映射表的主键列关联。

@ManyToMany 注释：表示此类是多对多关系的一边，mappedBy 属性定义了此类为双向关系的维护端，注意：mappedBy 属性的值为此关系的另一端的属性名。

例如，在Student类中有如下方法：

```
@ManyToMany(mappedBy = "students")

    public Set<Teacher> getTeachers() {

    return teachers;

}
```

@OneToOne的属性有：tragetEntity 、cascade、fetch、optional、mappedBy

：：：：Person类中

```java
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;

@Entity(name="t_person")
public class Person {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int pid;
    private String pname;
    private String address;

    @OneToOne(cascade=CascadeType.ALL)
    @PrimaryKeyJoinColumn
    private Card card ;

    public Person() {
    }
```

：：：**Card类中**

```java
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity(name="t_card")
public class Card {
    @Id
    @GeneratedValue(generator="myforeign")
    @GenericGenerator(
            name="myforeign",
            strategy="foreign",
            parameters={
                @Parameter(name="property",value="person")
            }
    )
    private int cid;
    private String cno;

    @OneToOne(mappedBy="card")
    private Person person ;

    public Card() {
    }
```

test中

```java
public class TestHibernate {

    private Session session ;

    @Before
    public void setup(){
        Configuration cfg = new Configuration().configure();
        SessionFactory factory = cfg.buildSessionFactory() ;
        session = factory.openSession() ;
        session.getTransaction().begin();
    }

    @Test
    public void test01(){
        Person person = new Person("jim", "USA");
        Card card = new Card("3231231989090887 89");
        card.setPerson(person);
        person.setCard(card);
        session.save(person);
    }

    @After
    public void teardown(){
        session.getTransaction().commit();
        session.close();
    }
}
```

7、foreign：用于一对一关系共享主健时，两id值一样。
本文讲解Hibernate中hbm的generator子元素的一些内置生成器的快捷名字。Generator子元素是一个非常简单的接口；某些应用程序可以选择提供他们自己特定的实现。
在*.hbm.xml必须声明的< generator>子元素是一个Java类的名字，用来为该持久化类的实例生成唯一的标识。

@OneToMany的属性有：

Student类中

```java
@Entity(name="t_stu")
public class Student {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int sid ;
    private String sname ;

    @ManyToOne(targetEntity=ClassBean.class)
    @JoinColumn(name="classid")
    private ClassBean cb ;

    public Student(){}

    public Student(String sname, ClassBean cb) {
        super();
        this.sname = sname;
        this.cb = cb;
    }

    public int getSid() {
        return sid;
    }
```

ClassBean类中：

```java
@Entity(name="t_class")
public class ClassBean {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int cid ;

    private String cname ;

    @OneToMany(cascade=CascadeType.ALL,mappedBy="cb")
    private Set<Student> students;

    public ClassBean(){}

    public ClassBean(String cname) {
        super();
        this.cname = cname;
    }

    public int getCid() {
        return cid;
    }
```

test类中：

```java
@Test
public void test02(){
    ClassBean cb = session.load(ClassBean.class, 1);
    session.delete(cb);
}

@Test
public void test03(){
    ClassBean cb = session.load(ClassBean.class, 2);

    //客户端重定向
    //OpenSessionInViewFilter
    //students.size();
    //Hibernate.initialize(cb);
    session.close();
    Set<Student> students = cb.getStudents();
    for(Student s : students){
        System.out.println(s);
    }
}

@After
public void teardown(){
    //session.getTransaction().commit();
    //session.close();
}
```

@ManyToMany

Student类中

```java
@Entity(name="t_stu")
public class Student {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int sid ;
    private String sname ;

    @ManyToMany(mappedBy="students")
    private Set<Teacher> teachers;

    public Student(){}

    public Student(String sname) {
        super();
        this.sname = sname;
    }
```

Teacher类中

```java
@Entity(name = "t_teacher")
public class Teacher {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int tid;
    private String tname;

    @ManyToMany(targetEntity=Student.class,cascade=CascadeType.ALL)
    @JoinTable(
        name="t_tea_stu",
        joinColumns={
                @JoinColumn(name="teaid")
        },
        inverseJoinColumns={
                @JoinColumn(name="stuid")
        }
    )
    public Set<Student> students;

    public Teacher() {
    }
```

Test类中

```
    @Test
    public void test02(){
        Teacher teacher = session.load(Teacher.class, 1);   //teacher是持久态
        Set<Student> students = teacher.getStudents();
        Student studentTarget = null ;
        for(Student s : students){
            if(s.getSid()==3){
                studentTarget = s ;
            }
        }
        students.remove(studentTarget);
//      session.update(teacher);        //当前这句话可以不写
    }

    @After
    public void teardown(){
        session.getTransaction().commit();
        session.close();
    }
```

9、hibenate常用的接口和类有哪些

(1)、Configuration类的实例首先定位映射文件的位置，读取这些配置，然后创建一个SessionFactory对象

(2)、用户程序从工厂类SessionFactory中取得Session的实例；注意：Session对象是非线程安全的，因此最好是一个线程只创建一个Session对象。

(3)、Transaction接口，是一个可选的API，是对实际事务实现的一个抽象，这些实现包括JDBC的事务……

(4)、Query和Criteria接口Query接口让你方便的对数据库及持久对象进行查询，它可以有两种表达方式：HQL语言和SQL语句；Query经常被用来绑定查询参数，限制查询记录数量，并最终执行查询操作。

Criteria接口允许你创建并执行面向对象的标准化查询