1、添加操作（create）

Hibernate 通过 session 的 save 方法来添加数据。

注：Hibernate自动建表是在 sessionFactory创建时期进行建表，一般都是用 static 修饰 *sessionFactory* ，所以一般如果需要完成hibernate建表直接在封装 sessionFactory 的工具类中执行main即可

代码示例：

```
        Transaction transaction = session.beginTransaction();

        //curd操作
        User user = new User();
        user.setUsername("chenlong");
//      不要需要添加id属性
        user.setPassword("123456");
        user.setAddress("nankang");

        session.save(user);
        //提交事务
        transaction.commit();
```

2、查询操作（retrieve）

根据 id 查询： Hibernate 通过 session 的 get 方法来进行查询

session.get(Class<T>, id) 包含俩个参数，第一个是查询返回数据的泛型，第二个是要查询数据的id

代码示例：

```
        Transaction transaction = session.beginTransaction();

        //curd操作
        User user = session.get(User.class, 1);
        System.out.println(user);


        //提交事务
```

```
        transaction.commit();
```

3、修改操作（update）

Hibernate 通过 session 的 update 方法实现更新

实现步骤：

使用 get 方法，通过 id 查询 ----> 向返回的对象设置修改值 -----> 调用 session 中的 update 方法，更新数据

代码实现：

```
    @Test
    public void testUpdate() {
        SessionFactory sessionFactory = HibernateUtils.getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();
        try {
            tx.begin();
            User user = session.get(User.class, 1);
            user.setUsername("Daiwei");
//          session.save(user);
            session.update(user);
            tx.commit();
        } catch(Exception exception){
            tx.rollback();
            exception.printStackTrace();
        } finally {
            if(session != null) {
                session.close();
            }
        }
    }

```

注：在这个更新的代码中，使用 save 方法也能实现，但是主要还是用 update 方法。

4、删除操作（delete）

删除操作是通过 hibernate 调用 session 中的 delete 方法实现。。

实现步骤：

方法一： 使用 get 方法 获得要删除的对象---> 调用 session 的delete 方法进行删除（提倡）

方法二：new 一个数据库数据模型pojo---> 设置 pojo 的 id 值为要删除记录的id值---> 调用 session 的 delete 方法进行删除

代码实现：

```
//方法
    @Test
    public void testDelete() {
        SessionFactory sessionFactory = HibernateUtils.getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();
        try{
            tx.begin();
            User user = session.get(User.class, 1);
            session.delete(user);
            tx.commit();
        }catch (Exception e) {
            tx.rollback();
            e.printStackTrace();
        }finally {
            if(session != null) {
                session.close();
            }
        }
    }
```

```
//方法2
    @Test
    public void testDelete() {
        SessionFactory sessionFactory = HibernateUtils.getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();
        try{
            tx.begin();
            User user = new User();
            user.setUid(2);
//          User user = session.get(User.class, 1);
            session.delete(user);
            tx.commit();
        }catch (Exception e) {
```

```
            tx.rollback();
            e.printStackTrace();
        }finally {
            if(session != null) {
                session.close();
            }
        }
    }
```

5、实体类对象状态（概念）

瞬时状：对象里面没有 id，对象与 session 也没有关联。
持久态：对象中有 id ，且对象与session 也有关联。
托管态：对象里面有id， 但对象与session没有关联。

6、saveOrUpdate方法

如果 saveOrUpdate实体类对象的状态是瞬时态的话，saveOrUpdate 做的是 save操作即插入操作；

```
    @Test
    public void saveOrUpdate() {
        SessionFactory sessionFactory = HibernateUtils.getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();
        try {
            tx.begin();
            User user = new User();
            user.setUsername("DaiWei");
            user.setAddress("GongQingCheng");
            user.setPassword("123456");

            session.saveOrUpdate(user);

            tx.commit();
        } catch (Exception e) {
            // TODO: handle exception
            tx.rollback();
            e.printStackTrace();
        } finally {
            if(session != null) {
```

```
                session.close();
            }
        }
    }
```

如果 saveOrUpdate实体类对象的状态是持久态 或 托管态的话，saveOrUpdate 做的是 save操作即更新操作；

```
    @Test
    public void saveOrUpdate() {
        SessionFactory sessionFactory = HibernateUtils.getSessionFactory();
        Session session = sessionFactory.openSession();
        Transaction tx = session.beginTransaction();
        try {
            tx.begin();
//          瞬时态
//          User user = new User();
//          user.setUsername("DaiWei");
//          user.setAddress("GongQingCheng");
//          user.setPassword("123456");

//          持久态
//          User user = session.get(User.class, 4);
//          user.setPassword("345");
//          托管态
            User user = new User();
            user.setUid(4);
            user.setPassword("123456");
            session.saveOrUpdate(user);
            tx.commit();

        } catch (Exception e) {
            // TODO: handle exception
            tx.rollback();
            e.printStackTrace();
        } finally {
            if(session != null) {
                session.close();
            }
        }
    }
```