一，创建工具类的目的？

- 为了简化代码
- 因为SessionFactory最好在一个项目中，只有一个
- 单例模式
- 无法把SessionFactory的构造方法私有化

二，将SessionFactory的获取过程，封装起来

```
    SessionFactory sf = new Configuration().configure().buildSessionFac
tory();
```

三，HibernateUtil工具类

- 如果要通过工具类获得sf
- 则sf应该是工具类的一个属性
    - 并且这个属性的赋值过程，应该只有一次

```
    static SessionFactory SF;

    static{
        SF = new Configuration().configure().buildSessionFactory();

    }
```

四，我们到底是要获得SF，还是要获得session？

```
    public static Session getSession(){
        return SF.openSession();
    }
```

五，Transaction怎么办？

- **我们发现，每次使用session的时候，都需要手动开启和提交事务**
- **把session关闭也是每次都要做的**

```
interface ResultHandler{
    void resultHandle(Session session);
}
```

```
public static <T> T handle(ResultHandle rh){
    Session session = SF.openSession();
    Transaction t = session.beginTransaction();

    //TODO 执行我们具体要做的业务逻辑
    //rh.resultHandle(session);
    //这句话是在第六步后更改的
    T tt = rh.resultHandle(session);
    t.commit();
    session.close();
    //这里是在第六步加上的
    return tt;
}
```

六，怎么拿到在rh中使用session获得的返回值？

- 调用的时候，一定是使用HibernateUtil
- 比如
  - Student stu = HibernateUtil.handle();

```
    Student stu = HibernateUtil.handle(new HibernateUtil.ResultHandler(
){

            //返回值应该做一个修改
            //应该返回的是获取数据时传入的类型
            //但是这个类型不能具体化，因为是可变的
```

```
            //public void resultHandle(Session session){
                //Student stu = session.get(Student.class,1L);

            //}

            //所以应该是下面这样
            public <T> T resultHandle(Session session){
                Student stu = session.get(Student.class,1L);
                return stu;
            }
    });
```

根据上述代码中，需要更改返回值类型为泛型，所以应该把Hibernate中内部接口的方法的返回值更改

```
    interface ResultHandler{
        <T> T resultHandle(Session session);
    }
```