一、实体类(持久化类)的编写规则

1. 实体类的属性都是私有的
2. 提供空参构造器
3. 所有属性都有公开的get和set方法
4. 要有一个属性与表的主键对应
5. 实体类的属性建议我们使用基本数据类型的包装类，包装类可以表示空

   > 假设表中有一列员工工资，如果使用 double 类型，如果这个员工工资忘记录入到系统中，系统会将默认值 0 存入到数据库，如果这个员工工资被扣完了，也会向系统中存入 0.那么这个 0 就有了多重含义，而如果使用包装类类型就会避免以上情况，如果使用 Double 类型，忘记录入工资就会存入 null，而这个员工工资被扣完了，就会存入 0，不会产生歧义。

6. 实体类尽量不要使用final进行修饰

二、Hibernate主键生成策略

自然主键：

把具有业务含义的字段作为主键，称之为自然主键。不常用，同名咋办？给数据库维护增加难度。

代理主键：

把不具备业务含义的字段作为主键，称之为代理主键。该字段一般取名为"ID"，通常为整数类型，因为整数类型比字符串类型要节省很多数据库空间。

主键生成策略

名称	描述
increment	用于 long、short、或 int 类型，由 Hibernate 自动以递增的方式生成唯一标识符，每次增量为 1。只有当没有其它进程向同一张表中插入数据时才可以使用，不能在集群环境下使用。适用于代理主键。
identity	采用底层数据库本身提供的主键生成标识符，条件是数据库支持自动增长数据类型。在 DB2、MySQL、MS SQL Server、Sybase 和 HypersonicSQL 数据库中可以使用该生成器，该生成器要求在数据库中把主键定义成为自增长类型。适用于代理主键。
sequence	Hibernate 根据底层数据库序列生成标识符。条件是数据库支持序列。适用于代理主键。
native	根据底层数据库对自动生成表示符的能力来选择 identity、sequence、hilo 三种生成器中的一种，适合跨数据库平台开发。适用于代理主键。
uuid	Hibernate 采用 128 位的 UUID 算法来生成标识符。该算法能够在网络环境中生成唯一的字符串标识符，其 UUID 被编码为一个长度为 32 位的十六进制字符串。这种策略并不流行，因为字符串类型的主键比整数类型的主键占用更多的数据库空间。适用于代理主键。
assigned	由 java 程序负责生成标识符，如果不指定 id 元素的 generator 属性，则默认使用该主键生成策略。适用于自然主键。

三、实体类的crud操作

3.1 save方法实现添加操作

```
34          //添加功能
35          User user=new User();
36          user.setUsername("lisi");
37          user.setAddress("beijing");
38          user.setPassword("232323");
39          //调用session的方法实现添加
40          session.save(user);
```

3.2 get方法实现根据id查询

```
20          //4.根据id查询
21          //第一个参数，实体类的class
22          //第二个参数 id值
23          User user = session.get(User.class, 4);
24          System.out.println(user);
```

3.3 update方法实现更新操作

```
20          //4.根据id修改,先查询，然后根据实体类重新设置即可，最后调用更新操作
21          //第一个参数，实体类的class
22          //第二个参数 id值
23          User user = session.get(User.class, 5);
24          user.setUsername("wangwu");
25          session.update(user);
```

3.4 delete方法实现删除操作

```
20          //4.根据id删除,先查询，再调用删除操作
21          //第一个参数，实体类的class
22          //第二个参数 id值
23          User user = session.get(User.class, 1);
24          session.delete(user);
25          //或者方法二
26  //      User user1=new User();
27  //      user1.setUid(1);
28  //      session.delete(user1);
```

上述除了添加操作，基本都要进行查询操作

四、实体类(持久化类)的三种状态

4.1 瞬时态

瞬时态也称为临时态或者自由态，瞬时态的实例是由new命令创建、开辟内存空间的对象，**不存在持久标识(相当于主键值)**，尚未与Hibernate Session关联，在数据库中也没有记录，失去引用后将被JVM回收。瞬时状态的对象在内存中是孤立存在的，与数据库中的数据无任何关联，仅是一个信息携带的载体。

```
35          User user=new User();
36          user.setUsername("lisi");
37          user.setAddress("beijing");
38          user.setPassword("232323");
```

瞬时态，没有关联与数据库

4.2 持久态

持久态的对象存在持久化标识，加入到了Session缓存中，并且相关联的Session没有关闭，在数据库中有对应的记录，每条记录只对应唯一的持久化对象，需要注意的是，持久态对象是在事务还未提交前变成持久态的。

```
20          //4.根据id查询
21          //第一个参数，实体类的class
22          //第二个参数 id值
23          User user = session.get(User.class, 4);
24          System.out.println(user);
```

4.3 托管态

托管态也称离线态或者游离态，当某个持久化状态的实例与Session的关联被关闭时就变成了托管态。托管态对象存在持久化标识，并且仍然与数据库中的数据存在关联，只是失去了与当前Session的关联，托管状态对象发生改变时Hibernate不能检测到。

```
//      User user1=new User();
//      user1.setUid(1);
```

五、 Hibernate一级缓存

缓存是计算机领域非常通用的概念。它介于应用程序和永久性数据存储源(如硬盘上的文件或者数据库)之间，其作用是降低应用程序直接读写永久性数据存储源的频率，从而提高应用的运行性能。缓存中的数据是数据存储源中数据的拷贝。缓存的物理介质通常是内存。
Hibernate的缓存分为一级缓存和二级缓存，Hibernate的这两级缓存都位于持久化层，存储的都是数据库数据的备份。其中第一级缓存为Hibernate的内置缓存，不能被卸载。

1. 一级缓存默认是打开的。
2. 一级缓存使用范围是session域，从session打开到session关闭期间。
3. 一级缓存存的数据是持久化数据。
   二级缓存由redis解决。是sessionFactory范围的。

5.1 验证一级缓存的存在

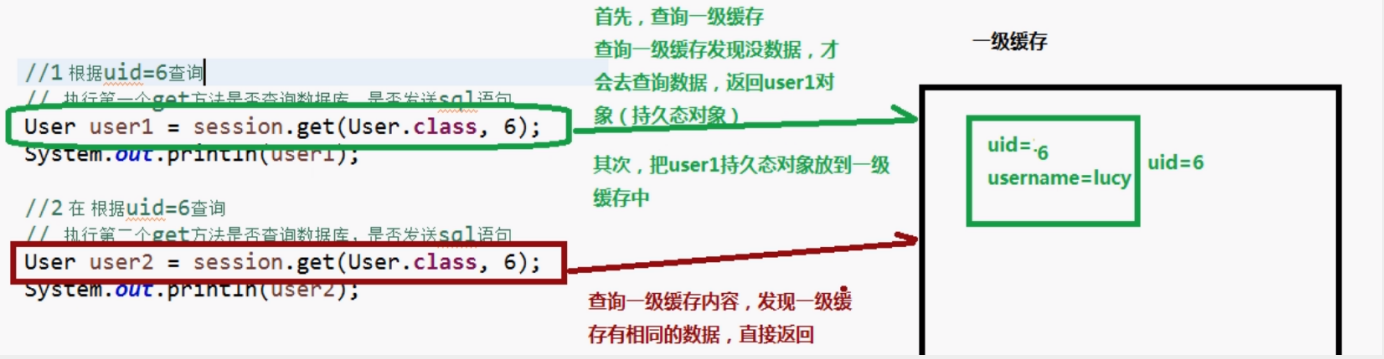
根据**两次查询同一个对象时是否都会出现sql语句输出**确定是否存在一级缓存

```
21          //根据uid=4查询
22          //是否发送sql与语句
23          User user1 = session.get(User.class, 4);
24          System.out.println(user1);
25        //根据uid=4再次查询
26          //是否发送sql语句
27          User user2 = session.get(User.class, 4);
28          System.out.println(user2);
```

```
Hibernate:
    select
        user0_.uid as uid1_0_0_,
        user0_.username as username2_0_0_,
        user0_.password as password3_0_0_,
        user0_.address as address4_0_0_
    from
        t_User user0_
    where
        user0_.uid=?
User [uid=4, username=wangwu, password=12323, address=jiangsu]
User [uid=4, username=wangwu, password=12323, address=jiangsu]
```

很明显，第一次查询有sql语句输出，第二次查询没有，说明有一级缓存的存在

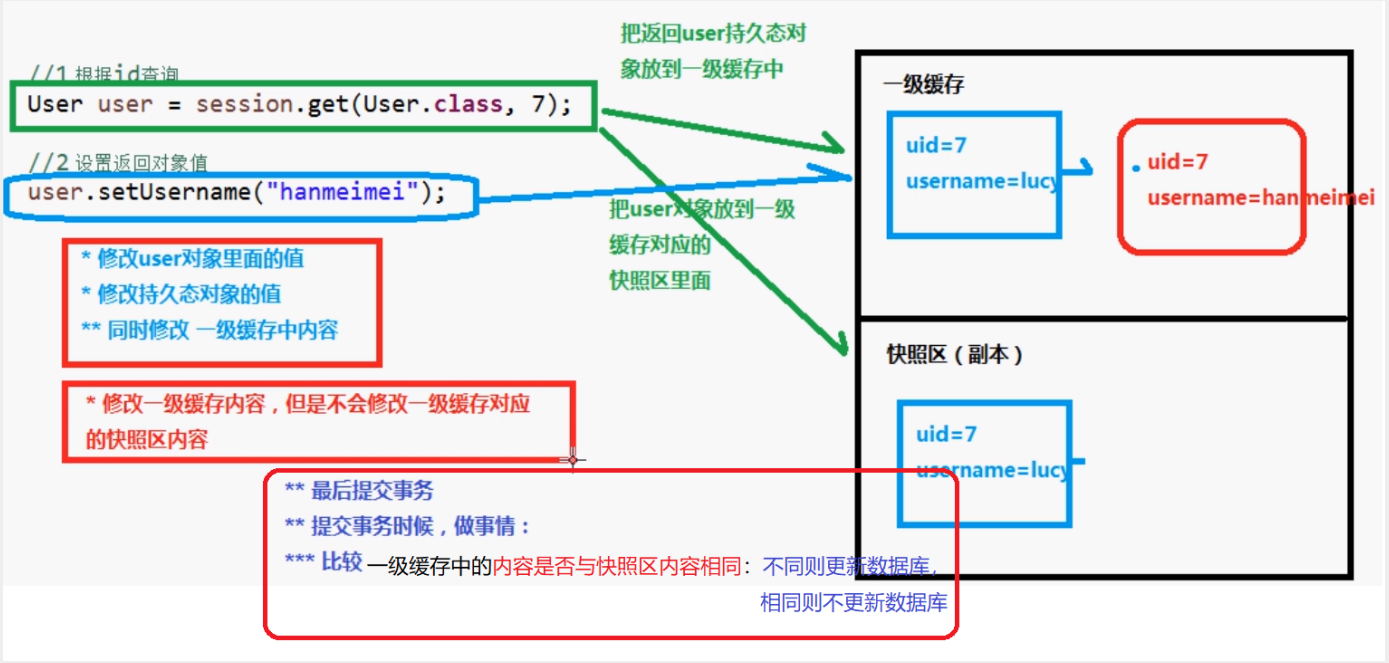
5.2 一级缓存的执行过程



5.3 一级缓存的特性演示

特性：持久态自动更新数据库

```
21          //持久态自动更新
22          User user1 = session.get(User.class, 4);
23          user1.setUsername("hanmeimei");
24                                  不需要执行update语句
25          //session.update(user1);
```



六、 Hibernate的事务编写规范

```
14          public void testDemo(){
15           SessionFactory sessionFactory=null;
16           Session session=null;
17           //3.开启事务
18           Transaction transaction=null;
19           try {
20                   sessionFactory = HibernateUtils.getSessionFactory();
21                   session = sessionFactory.openSession();
22                   transaction = session.beginTransaction();
23
24
25                   User user1 = new User();
26                   user1.setAddress("beijing");
27                   user1.setUsername("hhhh");
28                   user1.setPassword("000");
29                   session.save(user1);
30                   //int i=1/0;
31                   //提交事务
32                   transaction.commit();
33           } catch (Exception e) {
34                   //回滚事务
35                   transaction.rollback();
36                   e.printStackTrace();
37           }finally{
38               //关闭操作
39              session.close();
40              sessionFactory.close();
41          }
42
43    }
44  }
```

Hibernate绑定session

1. session类似于jdbc的connection
2. 与本地线程绑定session对象类似于jdbc中ThreadLocal对象绑定connection
3. 获取与本地线层相关的session

   1）在核心配置文件中配置

```
<!-- 配置与本地线程绑定的session对象 -->
<property name="hibernate.current_session_context_class">thread</property>
```

   2）调用sessionFactory里面的方法`getCurrentSession()`得到

```java
package cn.scct.hibernateUtils;

import org.hibernate.Session;

public class HibernateUtils {
    private static Configuration cfg;
    private static SessionFactory sessionFactory;

    static{
        cfg=new Configuration();
        cfg.configure();
        sessionFactory = cfg.buildSessionFactory();
    }

    public static SessionFactory getSessionFactory(){
        return sessionFactory;
    } 获得当前的session对象，和ThreadLocal对象类似

    public static Session getCurrentSession(){
        return sessionFactory.getCurrentSession();
    }
}
```

从此这个工具类完整了好像

```java
public class HibernateAffairs2 {
    @Test
        public void testDemo(){
         Session session=null;
         //开启事务
         Transaction transaction=null;
         try {

                session = HibernateUtils.getCurrentSession();
                transaction = session.beginTransaction();
  此时已经不需要关闭

  session对象了User user1 = new User();
                user1.setAddress("beijing");
                user1.setUsername("ttt");
                user1.setPassword("0566");
                session.save(user1);
                //int i=1/0;
                //提交事务
                transaction.commit();
         } catch (Exception e) {
                //回滚事务
                transaction.rollback();
                e.printStackTrace();
         }
         //已经不需要关闭session了，因为与线程同生死
         //sessionFactory也不需要，
   }
}
```

从此若非单元测试，可以不写sessionFactory对象，否则会造成内存溢出。

七、 Hibernate其他API（针对查询所有）

7.1 Query对象

1. 使用hql（hibernate query language）语句实现
2. sql操作表名及其字段，hql操作实体类及其属性
3. 查询所有的hql语句：from 实体类名称

```
26                  Query query = session.createQuery("from User");
27                  List <User> list = query.list();
28
29                  for(User user :list){
30                      System.out.println(user);
31                  }
32
```

```
Hibernate:
    select
        user0_.uid as uid1_0_,
        user0_.username as username2_0_,
        user0_.password as password3_0_,
        user0_.address as address4_0_
    from
        t_User user0_
User [uid=2, username=lisi, password=232323, address=beijing]
User [uid=3, username=zhangsan, password=122, address=guangzhou]
User [uid=4, username=hanmeimei, password=12323, address=jiangsu]
User [uid=5, username=zhouliu, password=we2, address=shanghai]
User [uid=6, username=ttt, password=0566, address=beijing]
```

7.2 Criteria对象

1. 无需写sql和hql语句

```
33                  Criteria criteria = session.createCriteria(User.class);
34                  List <User> list = criteria.list();
35
36                  for(User user :list){
37                      System.out.println(user);
38                  }
```

7.3 SQLQuery对象

1. 调用底层sql语句

```
44                  SQLQuery sqlQuery = session.createSQLQuery("select * from t_user");
45
46                  List<Object[]>list = sqlQuery.list();
47                  for(Object[] shuzu:list){
48                      System.out.println(Arrays.toString(shuzu));
49                  }
50
```

或者

```
51                  SQLQuery sqlQuery = session.createSQLQuery("select * from t_user");
52                  sqlQuery.addEntity(User.class);
53
54                  List <User> list = sqlQuery.list();
55
56                  for(User user :list){
57                      System.out.println(user);
58                  }
```