1. 实体类编写规则

1. 实体类里面属性私有的
2. 私有属性使用公开的set和get方法操作
3. 要求实体类有属性作为唯一值（一般使用id值）
4. 实体类属性建议不使用基本数据类型，使用基本数据类型对应的包装类

4-1. 八个基本数据类型对应的包装类

- int – Integer
- char—Character、
- 其他的都是首字母大写 比如 double – Double

4-2. 比如表示学生的分数，假如 int score;

- 比如学生得了0分 ， int score = 0;
- 如果表示学生没有参加考试，int score = 0;不能准确表示学生是否参加考试

l 解决：使用包装类可以了， Integer score = 0，表示学生得了0分，

表示学生没有参加考试，Integer score = null;

2. Hibernate主键生成策略

2-1. hibernate要求实体类里面有一个属性作为唯一值，对应表主键，主键可以不同生成策略

2-2. hibernate主键生成策略有很多的值

```
<!-- 设置数据库表id增长策略   native：生成表id值就是主键自动增长-->
<generator class= "native"></generator>
```

2-3. 在class属性里面有很多值

（1）native： 根据使用的数据库帮选择哪个值

```
CREATE TABLE `t_user` (
  `uid` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `address` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`uid`)
```

（2）uuid：之前web阶段写代码生成uuid值，hibernate帮我们生成uuid值

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。

2-4. 演示生成策略值 uuid

（1）使用uuid生成策略，实体类id属性类型 必须字符串类型

private String uid;

（2）配置部分写出uuid值

<generator class= "uuid"></generator>

```
CREATE TABLE `t_user` (
  `uid` varchar(255) NOT NULL,
  `username` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `address` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`uid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

uid	username	password	address
8a7e85cf573085dd01573085df5b0000	小马	1314520	美国

3. 对实体类crud操作

3-1. 添加操作

1. 调用session里面的save方法实现

```
//添加功能
User user = new User();
user.setUsername("小马");
user.setPassword("1314520");
user.setAdress("美国");
//调用session的方法实现添加
session.save(user);
```

3-3. 查询操作

1. 调用session里面的get方法实现

```
//根据id查询，调用session里面的get方法。第一个参数：实体类的class，第二个参数：id值
User user = session.get(User.class, 1);
```

```
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=1, username=lucy, password=123, address=china]
```

3-2. 修改操作

1. 首先查询，再修改值

```
User user = session.get(User.class, 2);
user.setUsername("东方不败");
session.update(user);
```

```
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=?
Hibernate:
    update
        t_user
    set
        username=?,
        password=?,
        address=?
    where
        uid=?
```

3-3. 删除操作

1. 调用session里面delete方法实现

```
User user = session.get(User.class, 2);
session.delete(user);
```

4. 实体类对象状态（概念）

4-1. 实体类状态有三种

（1）瞬时态：对象里面没有id值，对象与session没有关联

```
User u = new User();
u.setUsername("jack");
u.setPassword("124");
u.setAddress("china");
```

（2）持久态：对象里面有id值，对象与session关联

```
User user = session.get(User.class, 1);
```

（3）托管态：对象有id值，对象与session没有关联

```
User user = new User();
user.setUid(3);
```

4-2. 演示操作实体类对象的方法

（1）saveOrUpdate方法：实现添加、实现修改

```
//1 添加操作
User user = new User();
user.setUsername("jack");
user.setPassword("520");
user.setAddress("朝鲜");

//实体类对象状态是瞬时态，做添加
session.saveOrUpdate(user);


User user = new User();
user.setUid(6);
user.setUsername("rose");
user.setPassword("1314");
user.setAddress("阿尔巴尼亚");

//实体类对象状态是托管态，做修改
session.saveOrUpdate(user);
```

```
User user = session.get(User.class, 7);
user.setUsername("lilei");

//实体类对象状态是持久态，做修改
session.saveOrUpdate(user);
```

5. Hibernate的一级缓存

5-1. Hibernate的一级缓存

1. 数据存到数据库里面，数据库本身是文件系统，使用流方式操作文件效率不是很高。

（1）把数据存到内存里面，不需要使用流方式，可以直接读取内存中数据

（2）把数据放到内存中，提高读取效率

5-2. Hibernate缓存

1. hibernate框架中提供很多优化方式，hibernate的缓存就是一个优化方式
2. hibernate缓存特点：

第一类 hibernate的一级缓存

（1）hibernate的一级缓存默认打开的

（2）hibernate的一级缓存使用范围，是session范围，从session创建到session关闭范围

（3）hibernate的一级缓存中，存储数据必须持久态数据

第二类 hibernate的二级缓存

（1）目前已经不使用了，替代技术 redis

（2）二级缓存默认不是打开的，需要配置

（3）二级缓存使用范围，是sessionFactory范围

5-3. 验证一级缓存存在

1. 验证方式

（1）首先根据uid=1查询，返回对象

（2）其次再根据uid=1查询，返回对象

```java
//1 根据uid=6查询
// 执行第一个get方法是否查询数据库，是否发送sql语句
User user1 = session.get(User.class, 6);
System.out.println(user1);

//2 在 根据uid=6查询
// 执行第二个get方法是否查询数据库，是否发送sql语句
User user2 = session.get(User.class, 6);
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=6, username=rose, password=1314, address=阿尔巴尼亚]
User [uid=6, username=rose, password=1314, address=阿尔巴尼亚]
```

第一步执行get方法之后，发送sql语句查询数据库

第二个执行get方法之后，没有发送sql语句，查询一级缓存内容

5-4. Hibernate一级缓存执行过程

5-5. Hibernate一级缓存特性

1. 持久态自动更新数据库

```java
//1 根据id查询
User user = session.get(User.class, 7);

//2 设置返回对象值
user.setUsername("hanmeimei");

//3 调用方法实现
session.update(user);
```

2. 执行过程（了解）

6. Hibernate事务操作

6-1. 事务相关概念

1. 什么是事务
2. 事务特性
3. 不考虑隔离性产生问题

（1）脏读

（2）不可重复读

（3）虚读

4. 设置事务隔离级别

（1）mysql默认隔离级别 repeatable read

6-2. Hibernate事务代码规范写法

1. 代码结构

```
try {
  开启事务
  提交事务
}catch() {
  回滚事务
}finally {
  关闭
}
```

```java
@Test
public void testTx() {
        SessionFactory sessionFactory = null;
```

```
Session session = null;
Transaction tx = null;
try {
        sessionFactory = HibernateUtils.getSessionFactory();
        session = sessionFactory.openSession();
        //开启事务
        tx = session.beginTransaction();
        //添加
        User user = new User();
        user.setUsername("小马");
        user.setPassword("250");
        user.setAddress("美国");
        session.save(user);
        int i = 10/0;
        //提交事务
        tx.commit();
        }catch(Exception e) {
                e.printStackTrace();
                //回滚事务
                tx.rollback();
                        }finally {
                        //关闭操作
                        session.close();
                        sessionFactory.close();
                                }
                }
```

6-3. Hibernate绑定session

1. session类似于jdbc的connection，之前web阶段学过 ThreadLocal
2. 实现与本地线程绑定session，session是单线程对象，实际企业项目中要与当前线程绑定
3. 获取与本地线程session

（1）在hibernate核心配置文件中配置

```
<!-- 在hibernate核心配置文件中配置 -->
<property name="hibernate.current_session_context_class">thread<,
```

（2）调用sessionFactory里面的方法得到

```java
//提供返回与本地线程帮的session的方法
public static Session getSessionobject() {
    return sessionFactory.getCurrentSession();
}
```

4. 获取与本地线程绑定session时候，关闭session报错，不需要手动关闭了

```
org.hibernate.SessionException: Session was already closed
```

7. Hibernate的API使用

7-1. Query对象

1. 使用query对象，不需要写sql语句，但是写hql语句

   （1）hql：hibernate query language，hibernate提供查询语言，这个hql语句和普通sql语句很相似

   （2）hql和sql语句区别：

   - 使用sql操作表和表字段
   - 使用hql操作实体类和属性

2. 查询所有的记录的hql语句：

   （1）from 实体类名称

3. Query对象使用

   （1）创建Query对象

   （2）调用query对象里面的方法得到结果

```java
//1 创建Query对象
//方法里面写hql语句
Query query = session.createQuery("from User");

//2 调用query对象里面的方法得到结果
List<User> list = query.list();

for (User user : list) {
    System.out.println(user);
}
```

7-2. Criteria对象

1. 使用这个对象查询操作，但是使用这个对象时候，不需要写语句，直接调用方法实现

2. 实现过程

（1）创建criteria对象

（2）调用对象里面的方法得到结果

```
//1 创建criteria对象
//方法里面参数是实体类class
Criteria criteria = session.createCriteria(User.class);
//2 调用方法得到结果
List<User> list = criteria.list();
```

7-3. SQLQuery对象

1. 使用hibernate时候，调用底层sql实现

2. 实现过程

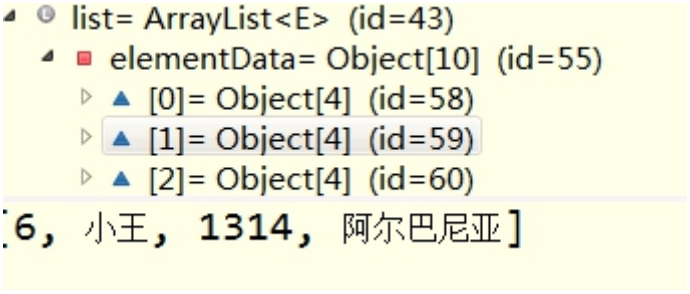
（1）创建对象

（2）调用对象的方法得到结果

```
//1 创建对象
//参数普通sql语句
SQLQuery sqlQuery = session.createSQLQuery("select * from t_user");
//调用sqlQuery里面的方法
//返回list集合，默认里面每部分数组结构
List<Object[]> list = sqlQuery.list();

for (Object[] objects : list) {
    System.out.println(Arrays.toString(objects));
}
```

返回list集合每部分是Object数组

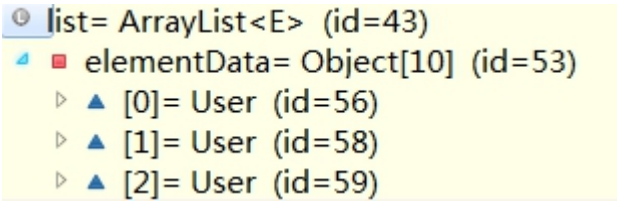


返回list中每部分是实体类形式

```
//1 创建对象
//参数普通sql语句
SQLQuery sqlQuery = session.createSQLQuery("select * from t_user");

//返回的list中每部分是对象形式
sqlQuery.addEntity(User.class);

//调用sqlQuery里面的方法
List<User> list = sqlQuery.list();
```


list= ArrayList<E> (id=43)
elementData= Object[10] (id=53)
[0]= User (id=56)
[1]= User (id=58)
[2]= User (id=59)