Hibernate 技术栈

Hibernate 是一个开源的轻量级框架，所谓轻量级即不需要依赖其他。
JavaEE 的三层架构

1) Web 层，struts2 框架；
2) Service 层,Spring 框架；
3) dao（data access object）层,Hibernate 框架，对数据库进行 crud 操作；

三、 Hibernate 入门

3.1 搭建 hibernate 环境

1) 创建项目，java 项目和 web 项目都可以；
2) 导入 jar 包，lib 文件夹里面的 required 文件夹，jpa 文件夹（规范）和其他的日志 jar 包，这个日志 jar 包并不是 Hibernate 本身的，是外面的,包括 log4j-1.2.16、slf4j-api-1.6.1 和 slf4j-log4j12-1.6.1。导入 java 的规范是先在项目下建一个文件夹，在右键选中 build path,笔者经过多次的尝试，发现使用该方法导入 jar 包时，只能导入到 referance library 里面，程序运行时改文件夹下的 jar 包不能被自动加载。所以改规范不可行，最好的办法还是直接放在 WEB-INF 文件夹下咋 ilb 文件夹里面，程序自动把 jar 包放在 Web App Libraries 文件夹里面，在运行是实现自动加载。

3.2 创建实体类

1) 要求有一个属性是唯一的，比如 ID；
2) 要有 get 和 set 的方法；
3) 使用 hibernate 时不需要手动建表，其可以自动帮我们创建；

3.3 配置实体类和数据库表的映射关系

1) 创建 xml 格式的配置文件，虽然配置文件的名称和位置没有固定要求，但是建议把 xml 文件放在实体类的包里面，名称和实体类的名称+“.hbm.xml”；

2) 配置时首先引入 xml 约束（如 dtd/schema）,其实在下载包\project\hibernate-core\out\production\resources\org\hibernate 里面的 hibernate-mapping.dtd 有的：

```
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

3) 配置映射关系，注意属性 name 的值要和实体类中的属性名称要一致

```
<hibernate-mapping>
    <class name="com.lvhongbin.bean.User" table="t_User">
      <id name="id" column="id">
        <generator class="native"></generator>
      </id>
      <property name="name" column="name"></property>
      <property name="password" column="password"></property>
      <property name="email" column="email"></property>
      <property name="sex" column="sex"></property>
      <property name="date" column="date"></property>
    </class>
</hibernate-mapping>
```

3.4 创建 Hibernate 中的核心配置文件

1) 核心配置文件的格式为.xml，但是核心配置文件的名称和位置是固定的。
   位置：必须在 src 下面
   名称：必须为 hibernate.cfg.xml
2) 配置时首先引入 xml 约束（如 dtd/schema）,其实在下载包\project\hibernate-core\out\production\resources\org\hibernate 里面的 hibernate-configuration-3.0.dtd 有的：

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

3) hibernate 在操作过程中，只会加载核心的配置文件，其他的配置文件不会进行加载。
   a) 配置数据库信息（可以参考 property 文件 project\etc\hibernate.properties），必须要有；
   b) 配置 hibernate 信息，可以不写；
   c) 引入映射文件，具体的代码如下：

```
<hibernate-configuration>
    <session-factory>
        <!-- 数据库驱动 -->
        <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
```

```
        <!-- 数据库连接的 URL 数据库需要手动创建 -->
        <property
name="connection.url">jdbc:mysql://localhost:3306/jsp_db?useSSL=true</prope
rty>
        <!-- 数据库连接用户名 -->
        <property name="connection.username">root</property>
        <!-- 数据库连接密码 -->
        <property name="connection.password">12345687</property>
        <!-- Hibernate 方言 旧版的方言为
org.hibernate.dialect.MySQLDialect-->
        <property
name="dialect">org.hibernate.dialect.MySQL5InnoDBDialect</property>
        <!-- 打印 SQL 语句 -->
        <property name="show_sql">true</property>
        <!-- 设置自动建表 -->
        <property name="hbm2ddl.auto">update</property>
        <!-- 映射文件 -->
        <mapping resource="com/lvhongbin/bean/User.hbm.xml"/>
    </session-factory>
 </hibernate-configuration>
```

3.5 实现添加的功能

1) 加载 hibernate 核心配置文件；
2) 创建 SessionFactory 对象；
3) 使用 SessionFactory 创建 session 对象；
4) 开启事务；
5) 写具体的逻辑 crud 操作；
6) 提交事务；
7) 关闭资源；

代码如下：

```
    public void Test() {
        Configuration cfg=new Configuration();
        cfg.configure();
        SessionFactory sessionFactory=cfg.buildSessionFactory();
        Session session=sessionFactory.openSession();
        Transaction tx=session.beginTransaction();
        User user=new User();
        user.setName("lvhongbin");
        user.setPassword("12345687");
        user.setSex("male");
```

```
        session.save(user);
        tx.commit();
        session.close();
        sessionFactory.close();
    }
```

四、 Hibernate 入门注意事项

4.1 Hibernate 配置文件详解

1) 映射文件的名称和位置没有固定，一般放在对应实体类的包里面；
2) 映射中的 name 属性的值要与对应实体类的属性相同，column 属性与表中的字段名称相同；
3) 如果不写 column 值，则 column 值与 name 值一致；
4) property 标签中的 type 属性，用来设置表字段的类型；
5) 核心配置文件的格式为.xml，但是核心配置文件的名称和位置是固定的：
   位置：必须在 src 下面
   名称：必须为 hibernate.cfg.xml
6) 核心配置文件的数据库部分是必须的，hibernate 部分时=是可选的，映射文件是必须的；

4.2 Hibernate 核心 API 的说明

1) Configuration

   Configuration cfg = new Configuration();

   cfg.configure(); //到 src 下面找到名称为 hibernate.cfg.xml 的配置文件，官方称“加载核心配置文件”

2) SessonFactory

   使用 Configuration 对象创建 SessonFactory 对象，然后根据核心配置文件进行连接或者创建数据表。如果每次操作都进行连接或者创建，则特别耗资源，所以在 hibernate 操作中建议一个项目只创建一个 SessonFactory 对象。具体实现：写一个工具类（util），写静态代码块实现，代码如下：

```
public class HibernateUtil {

    private final static Configuration cfg;
    private final static SessionFactory sessionFactory;

    static {
```

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

        public static SessionFactory getSessionFactory() {
            return sessionFactory;
        }
    }
```

然后在事务操作中添加代码：

```
SessionFactory sessionFactory=HibernateUtil.getSessionFactory();
```

3) Session

类似与 jdbc 中的 connection，调用 session 里面不同的方法实现 crud 操作：添加 save 方法，修改 update 方法，删除 delete 方法和跟据 id 查询 get 方法。session 对象是单线程（不能共用，只能自己使用，注意进程和线程的区别）对象。

4) Transaction

事务对象，两个重要操作事务提交和事务回滚，tx.commit();和 tx.rollback();事务有四个特性：原子性（一个事务操作要不全部成功，要么全部失败），一致性（数据操作前后总量不变），隔离性（多人同时操作不会相互影响）和持久性（数据在数据库提交后生效）。

4.3 解决配置文件中没有提示的问题

1) 可以上网；
2) 把约束文件引入项目，复制配置文件中的 http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd，然后点击 window -- Prefernce -- XML -- XML Catalog -- Add 在 key 中 URI 粘贴刚才复制的网址，然后在 Location 中寻找相应.dtd 文件的路径；

4.4 实体类（持久化类）的编写规则

1) 属性私有
2) 私有属性需要有公开的 get 和 set 方法
3) 要求实体类中有一个属性作为唯一值（一般使用 id 值）
4) 实体类属性建议不使用基本数据类型，而使用基本数据类型对应的包装。8 个基本数据类型,int 对应的是 integer，char 对应的是 Character，其他的都采用首字母大写，如 string 对应的是 String，double 对应的是 Doube，其原因是

基本数据类型没有 null 的值，而包装类有 null 值。

4.5 主键生成策略

1) increment 每次增加增量为 1；
2) identity 条件是数据库支持自动增长，如 Oracle 就不支持自动增长；
3) sequence 条件是数据库支持序列，如 MySql 不支持序列；
4) nativ 跟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，适合跨数据库平台的开发，适用于代理主键；
5) uuid 实体类 id 属性类型必须为字符串类型

4.6 实体类操作

1) save()操作；

```
User user =new User();
user.setName(“lvhongbin”);
session.copy(user);
```

2) 跟据 id 查询

Session 对象提供了两种对象装载的方法，分别是 get()和 load()。当开发人员不确定数据库是否有匹配的记录存在时，可以使用 get()方法，如果没有匹配的记录存在，返回 null。get()方法中含有两个参数，第一个是持久化对象，另一个是持久化对象的唯一标识值。其代码如下

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

load()方法返回对象的代理，只有在返回对象被调用的时候，Hibernate 才会发出 SQL 语句去查询对象。这叫 hibernate 的延迟加载策略，即使用 load()加载持久化对象时，它返回的是一个未初始化的代理（代理无须从数据库中抓取数据对象的数据）直到调用代理的某个方法时，hibernate 才会访问数据库。当装载的对象长时间没有调用时，就会被垃圾回收器回收，在程序中合理使用延迟加载策略，可以优化系统的性能，节省内存空间，减少不必要的开支。

```
// sql 语句 SELECT * FROM t_user t WHERE t.uid =1
User user =session.load(User.class, 1);
System.out.println(user);
```

或者在映射文件中的<property>标签的 lazy 属性改为“true”

```
<property lazy=“true”>
```

3) 修改

跟据 id 查询，返回对象

```
// UPDATE t-user SET username=?, address=? WHERE uid=?
```

第一种方法(建议)

User user =session.get(User.class, 1);
user.setName(“lvhongbin”);
session.update(user);
第二种方法(不建议,因为他会影响到别的参数，如果别的参数不修改的话会变成空值)
User user =new User();
user.setName(“lvhongbin”);
session.update (user);
第三种方法 强制刷新提交。session 的刷出（flush）过程是指 Session 执行的一些必须的 SQL 语句来把内存中的对象状态同步到 JDBC 中，刷出会在某些查询之前执行，在事物提交时执行，或者在程序中直接调用 flush（）时执行。
User user =new User();
user.setName(“lvhongbin”);
session.flush();

4) 删除

只有当对象在持久化状态时才能执行，所以在删除数据之前，首先将对象的状态转化为持久化状态。
//DELETE FROM t_user WHERE uid=?
第一种方法(建议)
User user =session.get(User.class, 1);
session.delete(user);
第二种方法
User user = new User();
User.setUid(3);
session.delete(user);

5) saveOrUpdate()

当实体类对象为瞬时态时作 save()的操作，当实体类对象为托管态或者持久态时作 update()的操作；

4.7 实体类对象的状态

1) 瞬时态：对象里面没有 id 值，对象与 session 没有关联。只是通过 new 关键字开辟内存空间创建 Java 对象，或者通过删除操作删除数据库表某一对象时，没有纳入 session 的管理里面，如

   User user =new User();
   user.setName(“lvhongbin”);
   或者：
   User user =session.get(User.class, 1);
   session.delete(user);

2) 持久态：在对象里面有 id 值，它总和会话状态（session）和事务（transaction）关联在一起。当持久化对象发生改动时并不会立即执行数据库操作，只有当

事务结束时，才会更新数据库，以便保证 Hibernate 的持久化对象和数据库操作的同步性，如插入，查询和更新的操作：

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

3) 托管状态：对象里面有 id 值，对象与 session 没有关联，close(),clear(), evict()

User user =new User();

user.setUid(3);

4.8 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本身效率并不高，把数据存到内存里面，不需要使用流的方式，可以直接读取内存里面的数据，提高读取效率。
2) Hibernate 框架中有很多优化的手段，其中缓存是其中一个手段，Hibernate 有一级缓存和二级缓存，一级缓存是默认打开的，属于 session 级缓存，所以它的生命周期与 session 相同，随 session 创建而创建，随 session 销毁而销毁，而且存储数据必须是持久化数据。过程：首先，查询一级缓存发现没数据，才会去查询数据，返回查询对象（持久化对象），然后，把持久化对象的值放到一级缓存中；若查询一级缓存中存在相同的数据，直接返回。
3) 一级缓存的更新特性：持久态自动更新数据库（不需要写 update 方法）。过程：创建 session 对象时，自动创建一级缓存对象和快照区（副本），当持久态对象被创建时，持久态被放到一级缓存及其对应的快照区中

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

   user.setName(“lvhongbin”);

   tx.commit();　　//修改 user 对象里面的值，修改持久态对象的值的同时，修改一级缓存中的内容，但是不会修改对照区中的内容。最后提交事务的时候，比较一级缓存和快照区的内容是否相同，如果不相同，则把一级缓存中的内容更新到数据库里面，如果相同，则不更新。
4) Hibernate 的二级缓存，目前已经不用了，替代技术为 redis，默认是不打开的，需要配置，属于 sessionFaction 级缓存。

   验证一级缓存的存在， 跟据 uid=1 查询，返回对象；再次根据 uid=1 查询，返回对象，通过断点 debug 查看。

4.9 Hibernate 事务的的那些事

1) 什么是事务；
2) 事物的特性：原子性，隔离性，一致性和持久性；
3) 不考虑隔离产生的问题：脏读，不可重复读和幻读。
   a) 脏读 指一个事务正在访问数据并且准备修改数据，在该事务提交到数据库前，另一个事务访问并且使用了这个数据。
   b) 不可重复读 指一个事务在多次读同一数据期间，存在其他事务访问并且修改了该数据，使得第一个事务在多次读同一数据过程中，无法读取到

相同的数据。

c) 幻读 是指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。

4) 设置事务的隔离级别。

为了避免事务并发问题的发生，在标准 SQL 规范中，定义了 4 个事务隔离级别，如：

a) 读未提交（Read Uncommitted，1 级）一个事务在执行过程中，可以访问其他未提交事务的插入数据或者是修改数据。多个事务不可同时执行写的操作，读的操作则可以同时进行。这样做可以防止丢失更新。

b) 已提交读（Read Committed，2 级） 一个事务在执行过程中，可以访问其他已提交事务的插入数据或者是修改数据。读的操作则可以同时进行。但是未提交的写事务将禁止其他事务进行访问，这样做可以有效防止脏读。

c) 可重复读（Repeatable Read，4 级）一个事务在执行过程中，可以访问其他已提交事务的插入数据，但不可以或访问成功修改的数据。读取数据的事务将禁止写事务，多个读取数据的事务可以同时进行，写事务则禁止任何其他的事务，这样可以有效防止不可重复读和脏读。

d) 序列化/串行化（Serializable，8 级） 非常严格的事务隔离，要求事务序列化执行，即串行，不能并发，这样可以有效防止脏读，不可重复读和幻读。

5) mysql 的默认隔离级别 repeatable read，但也可以配置隔离级别，代码如下：
<property name=“hibernate.connection.isolaton”>4</ property>

6) 规范写法：

```
SessionFactory sessionFactory=null;
    Session session=null;
    Transaction tx=null;
    try {
        sessionFactory=HibernateUtil.getSessionFactory();
        session=sessionFactory.openSession();
        tx=session.beginTransaction();
        user=new User();
        user.setName("lvhongbin");
        user.setPassword("12345687");
        user.setSex("male");
        session.save(user);
        tx.commit();
    } catch (Exception e) {
        e.printStackTrace();
        tx.rollback();
```

```
}finally {
    session.close();
    //sessionFactory.close();
}
```

4.10session 与本地线程绑定

1) 在 hibernate 核心配置文件中配置；
<property name="hibernate.current_session_context_class">thread</property>
2) 调用 sessionFactory 里面的方法得到；

```
public class HibernateUtil {

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

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }

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

3) 通过单例构建的对象最后一般不能使用 close（）方法结束并回收资源，否则再次使用对象时会报错；
4) 另外，session 对象不能设置为 static 和 final 类型，因为每次对话都需要新建，但是挂钩的 sessionFactory 可以是 static 和 final 类型

五、 Hibernate 的 api 使用

5.1 Query 对象

使用 query 对象不需要写 sql 语句，但是写 hql（hibernate query language）sql 操作的是操作表和字段，hql 操作的是实体类和属性。方法：

1) 首先创建 query 对象，代码如下：

System.out.println("使用 Query 方法进行查询");
@SuppressWarnings("unchecked")
Query<User> query=session.createQuery("from User");//注意 from 后面跟着是实体类名字。

2) 然后调用 query 对象里面的方法，其返回值是一个实体类对象，代码如下：

```
List<User> list1=query.list();
for(User user2:list1) {
    System.out.println(user2);
}
```

5.2 Criteria 对象

使用该对象进行查询操作时，不需要写语句，直接可以调用方法，方法：

1) 首先创建 Criteria 对象，代码如下：

```
System.out.println("使用 Criteria 方法进行查询");
@SuppressWarnings("deprecation")
Criteria criteriaAll =session.createCriteria(User.class);
```

2) 然后调用 Criteria 对象里面的方法，其返回值是一个实体类对象，代码如下：

```
@SuppressWarnings("unchecked")
List<User> list3= criteriaAll.list();
for(User user4:list3) {
    System.out.println(user4);
}
```

5.3 SQLQuery 对象

用于调用底层 sql 语句，需要先去判断获取的 list 对象是否非空，实现过程：

1) 首先创建 SQLQuery 对象，代码如下：

```
System.out.println("使用 SQLQuery 方法进行查询");
@SuppressWarnings({ "deprecation", "unchecked" })
SQLQuery<Object[]> sqlQuery =session.createSQLQuery("SELECT * from tbl_user1");
```

2) 然后调用 SQLQuery 对象里面的方法，其返回值是一个数组，数组转换为字符串需要调用 Arrays.toString(Object[])方法，主要代码如下：

```
@SuppressWarnings("deprecation")
List<Object[]> list5=sqlQuery.list();
for(Object[] Objects:list5) {
    System.out.println(Arrays.toString(Objects));
}
```

3) 可以用 session. addEntity()方法将返回的数组转换为实体类对象，代码如下：

```
System.out.println("使用 Criteria 方法进行查询 2");
```

```
SQLQuery sqlQuery2 =session.createSQLQuery("SELECT * from tbl_user1");
sqlQuery2.addEntity(User.class);
List<User> list6=sqlQuery2.list();
for(User user7:list6) {
    System.out.println(user7);
}
```

六、 Hibernate 高级应用

6.1 表与表之间关系回顾

1) 一对多
   用的场景很多：分类和商品关系，一个分类里面有很多商品，一个商品只能属于一个分类；客户（与公司有业务往来的集体，如 360，百度和腾讯等）和联系人（客户里面的员工）的关系。建表时表与表 通过外键建立关系。
2) 多对多
   订单和商品的关系，一个订单里面有很多商品，一个商品属于多个订单。需要创建第三张表，至少有两个字段作为主键，分别指向两个表的主键。
3) 一对一
   在中国，一个男人只能有一个妻子，一个女人只能有一个丈夫。

6.2 一对多的操作

1) 映射配置
   以客户和联系人为例
   a) 创建实体类，分别为客户和联系人；
   b) 让两个实体类之间相互表示，即在客户实体类里面用 Set 集合(无序)表示多个联系人：

```
private Set<LinkMan> setLinkman=new HashSet<LinkMan>();
/**
* @return the setLinkman
*/
public Set<LinkMan> getSetLinkman() {
    return setLinkman;
}
/**
 * @param setLinkman the setLinkman to set
*/
public void setSetLinkman(Set<LinkMan> setLinkman) {
    this.setLinkman = setLinkman;
```

```
}
一个联系人只能属于一个客户：
private Customer customer;
/**
* @return the customer
*/
public Customer getCustomer() {
    return customer;
}
/**
* @param customer the customer to set
*/
public void setCustomer(Customer customer) {
    this.customer = customer;
}
```

c) 为客户配置映射关系：

```
<set name="setLinkMan">    //其中 name 的属性是 set 集合的名称
    <key column="clid"></key>  //key 是外键，本来只需要在多的一方建立，但在 hibernate 的机制中，外键需要双方来维护
    <one-to-many class="com.lvhongbin.bean.LinkMan"/> //其中 class 的属性是多的一方实体类的全路径
</set>
```

为联系人配置映射关系：

```
<many-to-one name="customer" class="com.lvhongbin.bean.Customer" column="clid"></many-to-one>
```

//其中 name 的属性值指的是在多的一方实体类中建立的一的一方的成员变量名，class 的属性值是一的一方实体类的全路径，column 属性值是外键的名称

2) 一对多级联添加操作

法一：复杂的做法

```
Customer customer= new Customer(); //添加客户对象
customer.setCustName("上海交通大学");
customer.setCustLevel("vip");
customer.setCustSource("网络");
customer.setCustPhone("54749110");
customer.setCustMobile("110");
LinkMan linkMan1 = new LinkMan(); //添加联系人对象
linkMan1.setLkm_name("Lucy");
linkMan1.setLkm_gender("female");
linkMan1.setLkm_phone("911");
LinkMan linkMan2 = new LinkMan();
```

```
linkMan2.setLkm_name("Ben");
linkMan2.setLkm_gender("male");
linkMan2.setLkm_phone("120");
customer.getSetLinkMan().add(linkMan1); 配置客户和联系人的映射关系
customer.getSetLinkMan().add(linkMan2);
linkMan1.setCustomer(customer);
linkMan2.setCustomer(customer);
session.save(customer);  //添加客户和联系人的业务操作
session.save(linkMan1);
session.save(linkMan2);
```

法二：简单的做法

a) 首先在客户映射文件中的 set 标签下添加 cascade 属性 save-update

```
<set name="setLinkMan" cascade="save-update">
    <key column="clid"></key>
    <one-to-many class="com.lvhongbin.bean.LinkMan"/>
</set>
```

b) 然后添加客户和联系人对象，把联系人放到客户里面，最终保存客户即可

```
System.out.println("级联保存");
Customer customer= new Customer();
customer.setCustName("百度");
customer.setCustLevel("vip");
customer.setCustSource("网络");
customer.setCustPhone("54749110");
customer.setCustMobile("110");
LinkMan linkMan1 = new LinkMan();
linkMan1.setLkm_name("Jenny");
linkMan1.setLkm_gender("female");
linkMan1.setLkm_phone("911");
LinkMan linkMan2 = new LinkMan();
linkMan2.setLkm_name("Micle");
linkMan2.setLkm_gender("male");
linkMan2.setLkm_phone("120");
customer.getSetLinkMan().add(linkMan1);
customer.getSetLinkMan().add(linkMan2);
//linkMan1.setCustomer(customer);
//linkMan2.setCustomer(customer);
session.save(customer);
//session.save(linkMan1);
//session.save(linkMan2);
```

c) 具体过程：

Hibernate: insert into tbl_Customer (custName, custLevel, custSource, custPhone, custMobile) values (?, ?, ?, ?, ?)

Hibernate: insert into tbl_LinkMan (lkm_name, lkm_gender, lkm_phone, clid) values (?, ?, ?, ?)

Hibernate: insert into tbl_LinkMan (lkm_name, lkm_gender, lkm_phone, clid) values (?, ?, ?, ?)

3) 一对多级联删除操作

a) 首先在客户映射文件中的 set 标签下添加 cascade 属性 delete

```
<set name="setLinkMan" cascade="save-update, delete">
    <key column="clid"></key>
    <one-to-many class="com.lvhongbin.bean.LinkMan"/>
</set>
```

b) 然后跟据 id 查询对象，调用 session 中的 delete 方法删除即可

```
System.out.println("级联删除");
Customer customer3=session.get(Customer.class, 2);
session.delete(customer3);
```

c) 具体过程：

跟据主键查询客户

Hibernate: select customer0_.cid as cid1_0_0_, customer0_.custName as custName2_0_0_, customer0_.custLevel as custLeve3_0_0_, customer0_.custSource as custSour4_0_0_, customer0_.custPhone as custPhon5_0_0_, customer0_.custMobile as custMobi6_0_0_ from tbl_Customer customer0_ where customer0_.cid=?

跟据外键查询联系人

Hibernate: select setlinkman0_.clid as clid5_1_0_, setlinkman0_.lkm_id as lkm_id1_1_0_, setlinkman0_.lkm_id as lkm_id1_1_1_, setlinkman0_.lkm_name as lkm_name2_1_1_, setlinkman0_.lkm_gender as lkm_gend3_1_1_, setlinkman0_.lkm_phone as lkm_phon4_1_1_, setlinkman0_.clid as clid5_1_1_ from tbl_LinkMan setlinkman0_ where setlinkman0_.clid=?

把外键设置为 null

Hibernate: update tbl_LinkMan set clid=null where clid=?

Hibernate: update tbl_LinkMan set clid=? where lkm_id=?

Hibernate: update tbl_LinkMan set clid=? where lkm_id=?

删除联系人

Hibernate: delete from tbl_LinkMan where lkm_id=?

Hibernate: delete from tbl_LinkMan where lkm_id=?

删除客户

Hibernate: delete from tbl_Customer where cid=?

4) 一对多级联修改操作

a) 跟据主键查询需要修改的联系人和客户

```
System.out.println("级联修改");
LinkMan linkMan4=session.get(LinkMan.class, 5);
Customer customer4=session.get(Customer.class, 1);
```

b) 建立联系人和客户的映射关系

```
customer4.getSetLinkMan().add(linkMan4);
linkMan4.setCustomer(customer4);
```

c) 具体过程：

跟据主键查询需要修改的联系人

```
Hibernate: select linkman0_.lkm_id as lkm_id1_1_0_, linkman0_.lkm_name as lkm_name2_1_0_, linkman0_.lkm_gender as lkm_gend3_1_0_, linkman0_.lkm_phone as lkm_phon4_1_0_, linkman0_.clid as clid5_1_0_ from tbl_LinkMan linkman0_ where linkman0_.lkm_id=?
```

跟据主键查询需要修改到的另外的客户

```
Hibernate: select customer0_.cid as cid1_0_0_, customer0_.custName as custName2_0_0_, customer0_.custLevel as custLeve3_0_0_, customer0_.custSource as custSour4_0_0_, customer0_.custPhone as custPhon5_0_0_, customer0_.custMobile as custMobi6_0_0_ from tbl_Customer customer0_ where customer0_.cid=?
```

添加客户和联系人的映射关系

```
Hibernate: select setlinkman0_.clid as clid5_1_0_, setlinkman0_.lkm_id as lkm_id1_1_0_, setlinkman0_.lkm_id as lkm_id1_1_1_, setlinkman0_.lkm_name as lkm_name2_1_1_, setlinkman0_.lkm_gender as lkm_gend3_1_1_, setlinkman0_.lkm_phone as lkm_phon4_1_1_, setlinkman0_.clid as clid5_1_1_ from tbl_LinkMan setlinkman0_ where setlinkman0_.clid=?
Hibernate: update tbl_LinkMan set lkm_name=?, lkm_gender=?, lkm_phone=?, clid=? where lkm_id=?
Hibernate: update tbl_LinkMan set clid=? where lkm_id=?
```

因为 hibernate 是外键双向维护机制，所以需要修改外键两次，造成性能的浪费，解决的方式是让其中一方放弃外键的维护，一般让客户放弃外键的维护。在客户的映射文件中的 set 标签中添加 inverse 属性，默认是 false，表示不放弃维护，true 表示放弃维护。

级联修改

```
Hibernate: select linkman0_.lkm_id as lkm_id1_1_0_, linkman0_.lkm_name as lkm_name2_1_0_, linkman0_.lkm_gender as lkm_gend3_1_0_, linkman0_.lkm_phone as lkm_phon4_1_0_, linkman0_.clid as clid5_1_0_ from tbl_LinkMan linkman0_ where
```

linkman0_.lkm_id=?

Hibernate: select customer0_.cid as cid1_0_0_, customer0_.custName as custName2_0_0_, customer0_.custLevel as custLeve3_0_0_, customer0_.custSource as custSour4_0_0_, customer0_.custPhone as custPhon5_0_0_, customer0_.custMobile as custMobi6_0_0_ from tbl_Customer customer0_ where customer0_.cid=?

Hibernate: select setlinkman0_.clid as clid5_1_0_, setlinkman0_.lkm_id as lkm_id1_1_0_, setlinkman0_.lkm_id as lkm_id1_1_1_, setlinkman0_.lkm_name as lkm_name2_1_1_, setlinkman0_.lkm_gender as lkm_gend3_1_1_, setlinkman0_.lkm_phone as lkm_phon4_1_1_, setlinkman0_.clid as clid5_1_1_ from tbl_LinkMan setlinkman0_ where setlinkman0_.clid=?

Hibernate: update tbl_LinkMan set lkm_name=?, lkm_gender=?, lkm_phone=?, clid=? where lkm_id=?

6.3 多对多的操作

1) 映射配置

以用户和角色为例

a) 创建实体类，分别为用户和角色；

b) 让两个用户相互表示

一个用户里面有多个角色，使用 set 集合；

```
private Set<Role> setRole=new HashSet<Role>();
/**
* @return the setRole
*/
public Set<Role> getSetRole() {
    return setRole;
}
/**
* @param setRole the setRole to set
*/
public void setSetRole(Set<Role> setRole) {
    this.setRole = setRole;
}
```

一个角色里面有多个用户，使用 set 集合；

```
private Set<User> setUser = new HashSet<User>();
/**
* @return the setUser
*/
```

```
public Set<User> getSetUser() {
    return setUser;
}
/**
* @param setUser the setUser to set
*/
public void setSetUser(Set<User> setUser) {
    this.setUser = setUser;
}
```

c) 配置映射关系

基本配置

```
<set name="setRole" table="user_role">
        <key column="userid"></key>
        <many-to-many class="com.lvhongbin.bean.Role"
column="roleid"></many-to-many>
    </set>

    <set name="setUser" table="user_role">
        <key column="roleid"></key>
        <many-to-many class="com.lvhongbin.bean.User"
column="userid"></many-to-many>
    </set>
```

配置多对多关系

2) 多对多级联保存

a) 首先在用户映射文件中的 set 标签下添加 cascade 属性 save-update

```
<set name="setLinkMan" cascade="save-update">
    <key column="clid"></key>
    <one-to-many class="com.lvhongbin.bean.LinkMan"/>
</set>
```

b) 然后添加用户和角色对象，把角色放到用户里面，最终保存用户即可

```
User user1=new User();
user1.setName("李小龙");
user1.setPassword("12345687");
user1.setSex("male");
User user2=new User();
user2.setName("霍元甲");
user2.setPassword("12345687");
user2.setSex("male");
User user3=new User();
user3.setName("爱迪生");
user3.setPassword("12345687");
user3.setSex("male");
```

```
Role role1 =new Role();
role1.setRole_name("中国人");
role1.setRole_memo("武术家");
Role role2 =new Role();
role2.setRole_name("美国人");
role2.setRole_memo("发明家");
user=new User();
user.setName("lvhongbin");
user.setPassword("12345687");
user.setSex("male");
user1.getSetRole().add(role1);
user2.getSetRole().add(role1);
user3.getSetRole().add(role2);
session.save(user1);
session.save(user2);
session.save(user3);
```

3) 多对多级联删除
   a) 首先在用户映射文件中的 set 标签下添加 cascade 属性 delete

```
<set name="setLinkMan" cascade="save-update">
    <key column="clid"></key>
    <one-to-many class="com.lvhongbin.bean.LinkMan"/>
</set>
```

   b) 然后跟据 id 查询所删除的用户，添加删除操作

```
User user1=session.get(User.class, 1);
Session.delete(user1);
```

4) 维护第三张表关系
   a) 让某个用户拥有某个角色

   首先根据 id 查询用户和角色

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

   Role role ==session.get(Role.class,1);

   然后把角色对象放到用户 set 集合里面即可

   user.getSetRole().add(role);

   具体过程如下：

   先查询相应的用户及其所属角色

   多对多级联修改-添加角色

   Hibernate: select user0_.id as id1_3_0_, user0_.name as name2_3_0_, user0_.password as password3_3_0_, user0_.email as email4_3_0_, user0_.sex as sex5_3_0_ from tbl_user1 user0_ where user0_.id=?

   Hibernate: select role0_.role_id as role_id1_2_0_, role0_.role_name as role_nam2_2_0_, role0_.role_memo as role_mem3_2_0_ from tbl_role role0_ where role0_.role_id=?

Hibernate: select role0_.role_id as role_id1_2_0_, role0_.role_name as role_nam2_2_0_, role0_.role_memo as role_mem3_2_0_ from tbl_role role0_ where role0_.role_id=?

再进行添加操作

Hibernate: select setrole0_.userid as userid1_4_0_, setrole0_.roleid as roleid2_4_0_, role1_.role_id as role_id1_2_1_, role1_.role_name as role_nam2_2_1_, role1_.role_memo as role_mem3_2_1_ from user_role setrole0_ inner join tbl_role role1_ on setrole0_.roleid=role1_.role_id where setrole0_.userid=?

Hibernate: select setrole0_.userid as userid1_4_0_, setrole0_.roleid as roleid2_4_0_, role1_.role_id as role_id1_2_1_, role1_.role_name as role_nam2_2_1_, role1_.role_memo as role_mem3_2_1_ from user_role setrole0_ inner join tbl_role role1_ on setrole0_.roleid=role1_.role_id where setrole0_.userid=?

b) 让某个用户没有某个角色

首先根据 id 查询用户和角色

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

Role role ==session.get(Role.class,1);

然后把角色对象放到用户 set 集合里面即可

user.getSetRole().remove(role);

6.4 查询方式介绍

1) 对象导航查询

跟据 id 查询客户，再查询该客户的所有联系人

```
System.out.println("对象导航查询");
Customer customer8 = session.get(Customer.class,97);
Set<LinkMan> linkMan8= customer8.getSetLinkMan();
System.out.println("对象导航查询结果：");
for(LinkMan linkMan8For:linkMan8) {
    System.out.println(linkMan8For.toString());
}
```

具体过程如下：

对象导航查询

Hibernate: select customer0_.cid as cid1_0_0_, customer0_.custName as custName2_0_0_, customer0_.custLevel as custLeve3_0_0_, customer0_.custSource as custSour4_0_0_, customer0_.custPhone as custPhon5_0_0_, customer0_.custMobile as custMobi6_0_0_ from tbl_Customer customer0_ where customer0_.cid=?

对象导航查询结果：

Hibernate: select setlinkman0_.clid as clid5_1_0_, setlinkman0_.lkm_id as lkm_id1_1_0_, setlinkman0_.lkm_id as lkm_id1_1_1_,

setlinkman0_.lkm_name as lkm_name2_1_1_, setlinkman0_.lkm_gender as lkm_gend3_1_1_, setlinkman0_.lkm_phone as lkm_phon4_1_1_, setlinkman0_.clid as clid5_1_1_ from tbl_LinkMan setlinkman0_ where setlinkman0_.clid=?

2) OID 查询

跟据 id 查询某一条记录，返回对象，使用的是 get 方法

```
Customer customer8 = session.get(Customer.class,97);
```

3) hql 查询

Query 对象，写 hql 语句实现查询。hql 全称：hibernate query language，与 sql 语言最大的不同在于 hql 操作的是实体类对象和属性，而 sql 操作的是数据库表和字段。

使用方法为先创建 query 对象，写 hql 语句，然后调用 query 对象里面的方法得到结果。常用的 hql 语句有：

a) 查询所有记录

```
System.out.println("使用 Query 方法进行查询");
Query<User> query=session.createQuery("from User");
list1=query.list();
for(User user2:list1) {
    System.out.println(user2);
}
```

b) 条件查询

from 实体类名称 where 实体类属性名称=？and 实体类属性名称=？

```
Query queryCondition=session.createQuery("from Customer othername where othername.custName=?");

// othername 使用的是 Customer 的别名
queryCondition.setParameter(0, "百度");
//添加属性值的时候使用的是.setParameter(int arg0, Object arg1)方法，第一个数字从 0 开始，代表属性值的位置，后面的参数表示属性值的具体内容
List<Customer> listCondition=queryCondition.list();
for(Customer Customer9:listCondition) {
    System.out.println("客户姓名："+Customer9.getCustName()+"\n客户手机号："+Customer9.getCustMobile());
}
```

具体过程：

Hibernate: select customer0_.cid as cid1_0_, customer0_.custName as custName2_0_, customer0_.custLevel as custLeve3_0_, customer0_.custSource as custSour4_0_, customer0_.custPhone as custPhon5_0_, customer0_.custMobile as custMobi6_0_ from tbl_Customer

customer0_ where customer0_.custName=?

from 实体类名称 where 实体类属性名称 like？（实体类查询）

c) 模糊查询

from 实体类名称 where 实体类属性名称 like ？

```
System.out.println("使用 Query 方法进行模糊查询");
Query queryConditionFuzzy=session.createQuery("from Customer othername where othername.custName like ?");
queryConditionFuzzy.setParameter(0, "% 腾 %");
//前面的%表示查询首字，后面的%表示查询末字，两个都有的表示查询中间出现的字
List<Customer> listConditionFuzzy=queryConditionFuzzy.list();
for(Customer Customer9:listConditionFuzzy) {
    System.out.println(" 客 户 姓 名 ："+Customer9.getCustName()+"\n 客 户 手 机 号 ："+Customer9.getCustMobile());
}
```

具体过程：

```
Hibernate: select customer0_.cid as cid1_0_, customer0_.custName as custName2_0_, customer0_.custLevel as custLeve3_0_, customer0_.custSource as custSour4_0_, customer0_.custPhone as custPhon5_0_, customer0_.custMobile as custMobi6_0_ from tbl_Customer customer0_ where customer0_.custName like ?
```

d) 排序查询

from 实体类名称 order by 实体类属性名称 asc/desc

```
System.out.println("使用 Query 方法进行排序查询");
Query queryOrder=session.createQuery("from Customer order by cid desc");
listOrder=queryOrder.list();
for(Customer Customer9:listOrder) {
    System.out.println("客户 id："+Customer9.getCid()+"客户姓名："+Customer9.getCustName()+"\n 客 户 手 机 号 ："+Customer9.getCustMobile());
}
```

具体过程：

```
Hibernate: select customer0_.cid as cid1_0_, customer0_.custName as custName2_0_, customer0_.custLevel as custLeve3_0_, customer0_.custSource as custSour4_0_, customer0_.custPhone as custPhon5_0_, customer0_.custMobile as custMobi6_0_ from tbl_Customer customer0_ order by customer0_.cid desc
```

e) 分页查询

在数据库表中使用关键字 limit 后面跟着两个参数：开始位置和记录数，但是在 hql 操作中，不能写 limit，而用 query 的两个方法。

首先查询全部

```
Query queryPagination=session.createQuery("from Customer");
```

然后设置开始位置

```
queryPagination.setFirstResult(0);
```

接着设置每页显示的记录数

```
queryPagination.setMaxResults(2);
```

最后显示结果

```
List<Customer> listPagination=queryPagination.list();
for(Customer Customer9:listPagination) {
    System.out.println("客户 id："+Customer9.getCid()+"客户姓名："
"+Customer9.getCustName()+"\n 客户手机号："+Customer9.getCustMobile());
}
```

具体过程：

```
Hibernate: select customer0_.cid as cid1_0_, customer0_.custName
as custName2_0_, customer0_.custLevel as custLeve3_0_, customer0_.custSource as
custSour4_0_, customer0_.custPhone as custPhon5_0_, customer0_.custMobile as
custMobi6_0_ from tbl_Customer customer0_ limit ?
```

f) 投影使用

用于查询某一字段的集合，用 hql 语句表示为：

```
System.out.println("使用 Query 方法进行投影查询--客户姓名");
Query queryProjection=session.createQuery("select custName from
    Customer");
List<Object> listProjection=queryProjection.list();
for(Object Customer9:listProjection) {
    System.out.println("客户姓名："+Customer9.toString());
}
```

具体过程：

```
Hibernate: select customer0_.custName as col_0_0_ from
    tbl_Customer customer0_
```

g) 聚合函数使用

常用的聚合函数有 count，sum，avg，max，min，使用 hql 语句：

查询表记录数 select count(*) from 实体类名称，代码如下：

```
System.out.println("使用 Query 方法进行聚合查询--表记录数");
Query queryAggregate=session.createQuery("select count(*) from
Customer");
Object listAggregate=queryAggregate.uniqueResult();  //返回的是
一个 Object 类型，但是 Object 类型不能直接强转为 int 类型，否则会出
错正确的应该先把 Object 类型转为 Long 类型，然后再通过 Long.intValue()
方法转为 int 类型
System.out.println("客户 id："+listAggregate.toString());
```

具体过程：

```
Hibernate: select count(*) as col_0_0_ from tbl_Customer
```

customer0_

4) QBC 查询

Criteria 对象

a) 查询所有

如 5.2 节所示

b) 条件查询

没有对应的语句，而使用相应的方法

```
System.out.println("使用 Criteria 方法进行条件查询--id>190");
        Criteria
criteriaCondition=session.createCriteria(Customer.class);
criteriaCondition.add(Restrictions.eq("custName", "百度"));
criteriaCondition.add(Restrictions.ge("cid", 190));
if (null!=criteriaCondition.list()) {
    listCriteriaCondition=criteriaCondition.list();
    for(Customer criteriaCustomer:listCriteriaCondition) {
        System.out.println("客户 id："+criteriaCustomer.getCid()+"
客户姓名： "+criteriaCustomer.getCustName()+"\n 客户手机号：
"+criteriaCustomer.getCustMobile());
    }
}else {
System.out.println("使用 Criteria 方法进行条件查询,记录数为 0");
}
```

c) 模糊查询

```
System.out.println("使用 Criteria 方法进行模糊查询");
Criteria
criteriaConditionFuzzy=session.createCriteria(Customer.class);
criteriaConditionFuzzy.add(Restrictions.like("custName", "%腾%"));
    if (null!=criteriaConditionFuzzy.list()) {
        listCriteriaConditionFuzzy=criteriaConditionFuzzy.list();
        for(Customer  criteriaCustomer:listCriteriaConditionFuzzy)
{
        System.out.println("客户 id："+criteriaCustomer.getCid()+"
客户姓名： "+criteriaCustomer.getCustName()+"\n 客户手机号：
"+criteriaCustomer.getCustMobile());
        }
    }else {
    System.out.println("使用 Criteria 方法进行模糊查询,记录数为
0");
    }
```

d) 排列和分页查询

```
System.out.println("使用 Criteria 方法进行排列和分页查询");
Criteria
```

```
criteriaOrderAndPagination=session.createCriteria(Customer.class);
                criteriaOrderAndPagination.addOrder(Order.desc("cid"));
                criteriaOrderAndPagination.setFirstResult(0);
                criteriaOrderAndPagination.setMaxResults(2);
                if (null!=criteriaOrderAndPagination.list()) {

    listCriteriaOrderAndPagination=criteriaOrderAndPagination.list();
                    for(Customer
criteriaCustomer:listCriteriaOrderAndPagination) {
                    System.out.println("客户 id："+criteriaCustomer.getCid()+"
客 户 姓 名 ： "+criteriaCustomer.getCustName()+"\n 客 户 手 机 号 ：
"+criteriaCustomer.getCustMobile());
                        }
                }else {
                    System.out.println("使用 Criteria 方法进行模糊查询,记录
数为 0");
                }
```

e) 统计和离线查询

离线查询实质上是将查询条件从业务层中剥离到 Web 层，从而使查询条件与持久化和查询解耦。其中主要用到 DetachedCriteria 类，代码如下：

```
                System.out.println("使用 Criteria 方法进行统计和离线查询");
                DetachedCriteria
detachedCriteria=DetachedCriteria.forClass(Customer.class);
                Criteria
criteriaStatistical=detachedCriteria.getExecutableCriteria(session);
                criteriaStatistical.setProjection(Projections.rowCount());
                listCriteriaStatistical=criteriaStatistical.uniqueResult();
                if (null!=listCriteriaStatistical) {
                    Long                listCriteriaStatisticalLong=(Long)
listCriteriaStatistical;
                System.out.println("使用 Criteria 方法进行模糊查询,记录数为
"+listCriteriaStatisticalLong.intValue());
                }else {
                    System.out.println("使用 Criteria 方法进行模糊查询,记录
数为 0");
                }
```

5) 本地 sql 查询

SQLQuery 对象，使用普通的 sql 实现查询

6.5 HQL 多表查询

以客户和联系人为例

1) 内连接
   两个表都有关联的数据才会显示
   SELECT * FROM t_customer c, t_linkMan l WHERE c.cid=l.clid
   或者 SELECT * FROM t_customer c INNER JOIN t_linkMan l ON c.cid=l.clid
   hql 语言中：
   from Customer c inner join c.setLinkMan
   返回的 list 对象是数组的形式
2) 左外连接
   左边的表是全的，右边的表只有跟左边表相关联的记录才会显示
   SELECT * FROM t_customer c LEFT OUTER JOIN t_linkMan l ON c.cid=l.clid
   from Customer c left outer join c.setLinkMan
3) 右外连接
   右边的表是全的，左边的表只有跟右边表相关联的记录才会显示
   SELECT * FROM t_customer c RIGHT OUTER JOIN t_linkMan l ON c.cid=l.clid
   from Customer c right outer join c.setLinkMan
4) 迫切内连接（hql 特有）
   迫切内连接与内连接的相同之处在于两者的底层实现是一样的，区别在于内连接返回的 list 中每部分是数组，而迫切内连接返回 list 的每部分是对象。
   from Customer c inner join fetch c.setLinkMan
5) 迫切左外连接（hql 特有）
   from Customer c left outer join fetch c.setLinkMan

6.6 检索策略

1) 检索策略的概念
   检索策略分为两类，分别是立即查询和延迟查询。前者跟据 id 查询，调用 get 方法，一调用 get 方法马上发送发送语句查询数据库。后者调用 load 的方法，调用 load 方法不会马上发送语句查询数据，只有返回的对象非 id 属性被调用的时候才会发送语句查询数据库。而延迟查询又分为两类，分别是类级别延迟和关联级别延迟，前者跟据 id 查询返回实体类对象，调用 load 方法不会马上发送语句；后者跟据 id 查询客户，再根据客户查询其所有联系人的过程的延迟，只有联系人被调用的时候才会被发送 sql 语句，这个无论用 get 或者用 load 方法都可以，因为这是系统默认的。
2) 如何修改关联级别延迟
   需要在客户的映射文件中进行相应的配置，在 set 标签的 fetch（值：select）

属性和 lazy（值：true 延迟 | false 不延迟 | extra 极其延迟）属性，两个属性都要设，默认是 fetch=”select” 和 lazy=”true”。如果 lazy=”false”时，执行 get 或者 load 进行查询时，同时发送查询客户和联系人的 sql 语句，若 lazy=” extra”，需要什么才执行什么。

6.7 批量抓取

应用场景：查询所有客户，并得到每个客户的所有联系人
小白可能会想，只要遍历查询到的客户，再通过对象导航查询遍历每个客户的联系人即可。但是这样发送太多的 sql 语句，有严重的性能问题。
优化：在客户的映射文件中色 set 标签添加 batch-size 属性，值为任意正整数，盖正整数越大，发送的语句越少，性能越高

JavaBean,更正确的叫法为实体类，拥有 get 和 set 的方法；