一、什么是Hibernate：

Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，它将POJO与数据库表建立映射关系，是一个全自动的orm框架，hibernate可以自动生成SQL语句，自动执行，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。 Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用，最具革命意义的是，Hibernate可以在应用EJB的J2EE架构中取代CMP，完成数据持久化的重任。

二、ORM：

利用描述对象和数据库表之间映射的元数据，自动把java应用程序中的对象，持久化到关系型数据库的表中，通过操作java对象，，就可以完成对数据库表的操作，可以把ORM理解为关系型数据和对象的一个纽带，开发人员只需要关注纽带一端映射的对象即可。

ORM可分为4级：

hibernate：4级，完全面向对象操作数据库

mybatis：2级

dbutils：1级

三、Hibernate优点：

a.Hibernate对JDBC访问数据库的代码做了轻量级封装，大大简化了数据访问　　层繁琐的重复性代码，并减少了内存的消耗，加快了运行效率。

b.有丰富的映射方式将Java对象之间的关系转换为数据库表之间的关系。

c.优秀的ORM框架，将对数据库的操作转换为对Java对象的操作，从而简化开 发。通过修改一
个持久化"对象的属性从而修改数据库表中对应的记录数据。

四、Hibernate两个配置文件详解：

1、orm映射文件详解（customer.hbm.xml）：

```xml
<!-- 配置表与实体对象的关系 -->
<!-- package属性:填写一个包名.在元素内部凡是需要书写完整类名的属性,可以直接写简答类名了. -->
<hibernate-mapping package="com.itcast.domain" >
    <!--
        class元素: 配置实体与表的对应关系的
            name: 完整类名
            table:数据库表名
     -->
    <class name="Customer" table="cst_customer" >
        <!-- id元素:配置主键映射的属性
                name: 填写主键对应属性名
                column(可选): 填写表中的主键列名.默认值:列名会默认使用属性名
                type(可选):填写列(属性)的类型.hibernate会自动检测实体的属性类型.
                        每个类型有三种填法: java类型|hibernate类型|数据库类型
                not-null(可选):配置该属性(列)是否不能为空. 默认值:false
                length(可选):配置数据库中列的长度. 默认值:使用数据库类型的最大长度
         -->


        <id name="cust_id"  >
            <!-- generator:主键生成策略.就是每条记录录入时,主键的生成规则.(7个)
                    identity : 主键自增.由数据库来维护主键值.录入时不需要指定主键.
                    sequence: Oracle中的主键生成策略.
                    increment(了解): 主键自增.由hibernate来维护.每次插入前会先查询表中id最大值.+1作为新主键值.
                    hilo(了解): 高低位算法.主键自增.由hibernate来维护.开发时不使用.
                    native:hilo+sequence+identity 自动三选一策略.
                    uuid: 产生随机字符串作为主键. 主键类型必须为string 类型.
                    assigned:自然主键生成策略. hibernate不会管理主键值.由开发人员自己录入.

             -->
            <generator class="native"></generator>
        </id>
        <!-- property元素:除id之外的普通属性映射
                name: 填写属性名
                column(可选): 填写列名
                type(可选):填写列(属性)的类型.hibernate会自动检测实体的属性类型.
                        每个类型有三种填法: java类型|hibernate类型|数据库类型
                not-null(可选):配置该属性(列)是否不能为空. 默认值:false
                length(可选):配置数据库中列的长度. 默认值:使用数据库类型的最大长度
         -->


        <property name="cust_name" column="cust_name" >
            <!--  <column name="cust_name" sql-type="varchar" ></column> -->
        </property>
        <property name="cust_source" column="cust_source" ></property>
        <property name="cust_industry" column="cust_industry" ></property>
        <property name="cust_level" column="cust_level" ></property>
        <property name="cust_linkman" column="cust_linkman" ></property>
        <property name="cust_phone" column="cust_phone" ></property>
        <property name="cust_mobile" column="cust_mobile" ></property>
    </class>
</hibernate-mapping>
```

2、主配置文件（hibernate.cfg.xml）：

```xml
<hibernate-configuration>
    <session-factory>
        <!-- 数据库驱动 -->
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <!-- 数据库url -->
        <property name="hibernate.connection.url">jdbc:mysql:///hibernate_32</property>
        <!-- 数据库连接用户名 -->
        <property name="hibernate.connection.username">root</property>
        <!-- 数据库连接密码 -->
        <property name="hibernate.connection.password">123456</property>
        <!-- 数据库方言
            不同的数据库中,sql语法略有区别. 指定方言可以让hibernate框架在生成sql语句时.针对数据库的方言生成.
            sql99标准: DDL 定义语言 库表的增删改查
                       DCL 控制语言 事务 权限
                       DML 操纵语言 增删改查
            注意: MYSQL在选择方言时,请选择最短的方言.
         -->
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>


        <!-- #hibernate.show_sql true
             #hibernate.format_sql true
        -->
        <!-- 将hibernate生成的sql语句打印到控制台 -->
        <property name="hibernate.show_sql">true</property>
        <!-- 将hibernate生成的sql语句格式化(语法缩进) -->
        <property name="hibernate.format_sql">true</property>
        <!--
            create: 自动建表.每次框架运行都会创建新的表.以前表将会被覆盖,表数据会丢失.(开发环境中测试使用)
            create-drop: 自动建表.每次框架运行结束都会将所有表删除.(开发环境中测试使用)
            update(推荐使用): 自动生成表.如果已经存在不会再生成.如果表有变动.自动更新表(不会删除任何数据).
            validate: 校验.不自动生成表.每次启动会校验数据库中表是否正确.校验失败.
         -->
        <property name="hibernate.hbm2ddl.auto">update</property>
        <!-- 引入orm元数据
            路径书写: 填写src下的路径
         -->
        <mapping resource="com/itcast/domain/Customer.hbm.xml" />

    </session-factory>
</hibernate-configuration>
```

五、Hibernate的API详解：

1、Configuration: 配置加载类，用于加载主配置，orm元素加载。

```java
//1创建：调用空参构造
Configuration conf = new Configuration();
//2读取指定主配置文件---空参加载方法，加载src下的hibernate.cfg.xml文件
conf.configure();
//3根据配置信息，创建SessionFactory对象
SessionFactory sf = conf.buildSessionFactory();
```

2、SessionFactory:创建session对象

注意:1.sessionfactory 负责保存和使用所有配置信息.消耗内存资源非常大.

2.sessionFactory属于线程安全的对象设计.

结论: 保证在web项目中,只创建一个sessionFactory.

```
//获得session，打开一个新的session对象
sf.openSession();
//获得一个与线程绑定的session对象
sf.getCurrentSession();
```

3、Session:表达hibernate框架与数据库之间的连接(会话).session类似于JDBC年代的connection对象.

还可以完成对数据库中数据的增删改查操作.

session是hibernate操作数据库的核心对象

```
//3获得session
Session session = sf.openSession();
//4 session获得操作事务的Transaction对象
//获得操作事务的tx对象
//Transaction tx = session.getTransaction();
//开启事务并获得操作事务的tx对象(建议使用)
Transaction tx2 = session.beginTransaction();
```

4、Transaction:

```
打开事务有两种方式：
    a.Transaction tx = session.getTransaction();
    b.Transaction tx2 = session.beginTransaction();(获得事务并开启事务，建议使用)
```

六、Hibernate中的实体规则：

1、实体类创建注意事项：

持久化类需提供无参构造

持久化类中的属性，应尽量使用包装类型

持久化类需提供oid，与数据库中的主键列对应

不要用final修饰class

2、主键类型：

自然主键（少见）：具有业务含义的字段，并且不重复，作为主键，

代理主键（常见）：没有业务含义的字段，作为主键。

3、Hibernate的主键生成策略：

```
<!-- generator:主键生成策略.就是每条记录录入时,主键的生成规则.(7个)
        identity : 主键自增.由数据库来维护主键值.录入时不需要指定主键.
        sequence: Oracle中的主键生成策略.
        increment(了解): 主键自增.由hibernate来维护.每次插入前会先查询表中id最大值.+1作为新主键值.
        hilo(了解): 高低位算法.主键自增.由hibernate来维护.开发时不使用.
        native:hilo+sequence+identity 自动三选一策略.
        uuid: 产生随机字符串作为主键. 主键类型必须为string 类型.
        assigned:自然主键生成策略. hibernate不会管理主键值.由开发人员自己录入.

    -->
    <generator class="native"></generator>
```

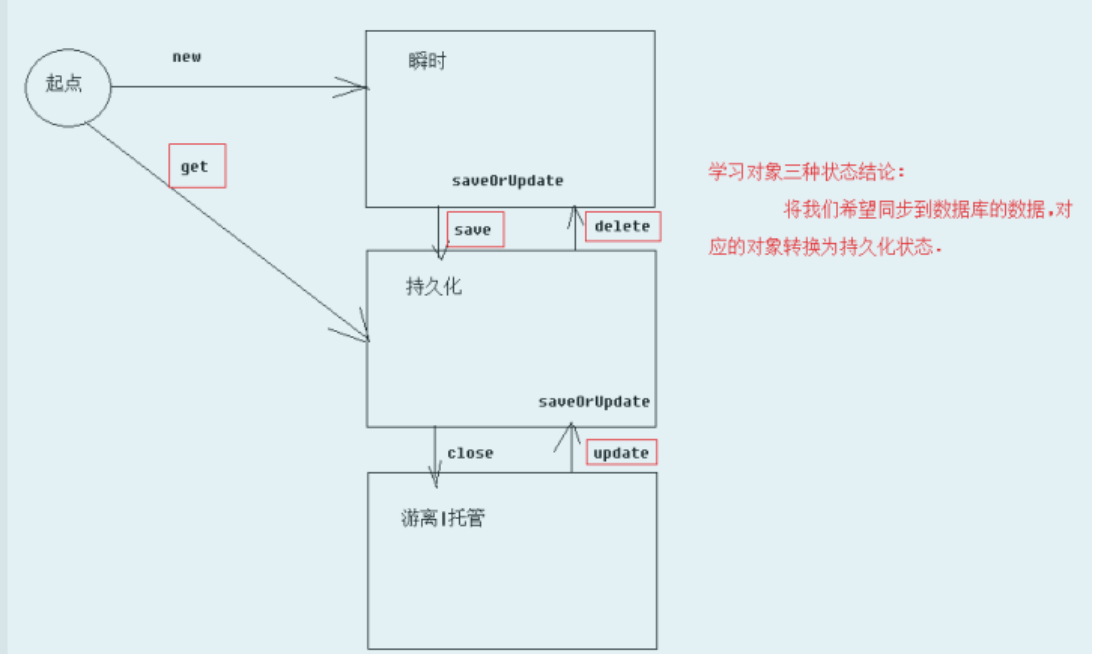
七、Hibernate对象中的三种状态：

瞬时状态：没有id,在session缓存中。

持久状态；有id，在session缓存中。（持久状态对象能够自动更新数据库）

游离状态：有id,不在session缓存中。

三种状态转换图：

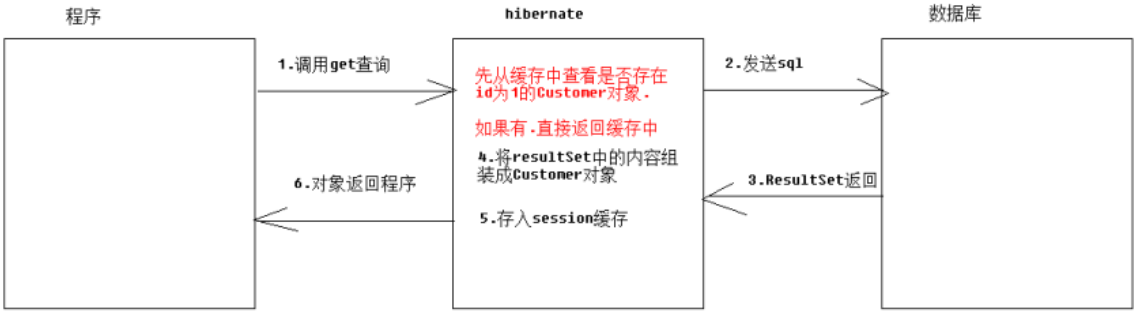


八、Hibernate的缓存：

一级缓存：(提高查询效率)，就指session缓存，减少对数据库的访问次数。

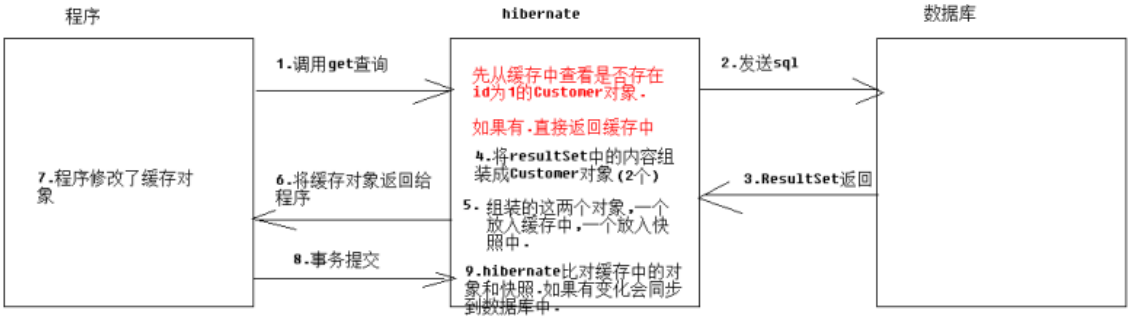
当调用session的close()方法时，Session缓存会被清空。

缓存原理



快照：(减少不必要的修改语句发送)，比对缓存区和快照区的数据是否一致，如果一致，不更新数据库；如果不一致自动更新数据库。

缓存进阶-快照



九、Hibernate中的事物：

1、事物：

事物的特性（ACID）:原子性、一致性、隔离性、持久性；

事物的并发问题：脏读、不可重复读、虚读/幻读;

事物的隔离级别：读未提交（1）、读已提交（2）、可重复读（4）、串行化（8）；

2、如何在hibernate中指定事物的隔离级别：

```
<!-- 指定hibernate操作数据库时的隔离级别
  #hibernate.connection.isolation 1|2|4|8
  0001    1    读未提交
  0010    2    读已提交
  0100    4    可重复读
  1000    8    串行化
-->
<property name="hibernate.connection.isolation">4</property>
```

在hibernate中确保使用同一个session的问题，hibernate已经帮我们解决了，只需调用sf.getCurrentSession(); 但前提要在主配置文件中配置：

```
<!-- 指定session与当前线程绑定 -->
<property name="hibernate.current_session_context_class">thread</property>
```

通过这样获得的session，当事物提交时，session会自动关闭，不需要手动close关闭；

十、Hibernate中的批量查询：

1、HQL查询：多表查询，但不复杂时使用。属于面向对象的查询语言；

a.基本查询：

```
//1>书写Hql语句
//String hql = "from com.itcast.domain.Customer"; //类的全名
String hql = "from Customer";//查询所有customer对象
//2>根据Hql语句创建查询对象
Query query = session.createQuery(hql);
//3>根据查询对象获得查询结果
List<Customer> list = query.list();//返回list结果
//query.uniqueResult();//接受唯一的查询结果
System.out.println(list);
```

b.条件查询：

```
//1>书写Hql语句
String hql = "from Customer where cust_id = 1";//查询所有customer对象。cust_id是属性名，不是列名
//2>根据Hql语句创建查询对象
Query query = session.createQuery(hql);
//3>根据查询对象获得查询结果
Customer c = (Customer) query.uniqueResult();//接受唯一的查询结果
System.out.println(c);
```

c.？占位符查询：

```
//1>书写Hql语句
String hql = "from Customer where cust_id = ?";//查询所有customer对象。
//2>根据Hql语句创建查询对象
Query query = session.createQuery(hql);
//设置参数
//query.setLong(0, 1l);
query.setParameter(0, 1l);
//3>根据查询对象获得查询结果
Customer c = (Customer) query.uniqueResult();//接受唯一的查询结果
System.out.println(c);
```

d.命名占位符：

```
//1>书写Hql语句
String hql = "from Customer where cust_id = :cust_id";//查询所有customer对象。
//2>根据Hql语句创建查询对象
Query query = session.createQuery(hql);
//设置参数
//query.setLong(0, 1l);
query.setParameter("cust_id", 1l);
//3>根据查询对象获得查询结果
Customer c = (Customer) query.uniqueResult();//接受唯一的查询结果
System.out.println(c);
```

e.分页查询：

```
//1>书写Hql语句
String hql = "from Customer ";//查询所有customer对象。
//2>根据Hql语句创建查询对象
Query query = session.createQuery(hql);
//设置分页信息 limt ?,?
query.setFirstResult(0);
query.setMaxResults(1);
//3>根据查询对象获得查询结果
 List<Customer> list = query.list();
System.out.println(list);
```

f.排序查询：

```
String hql1 = " from  cn.itcast.domain.Customer order by cust_id asc ";//完整写法
String hql2 = " from  cn.itcast.domain.Customer order by cust_id desc ";//完整写法

Query query = session.createQuery(hql2);

List list = query.list();
```

g.聚合查询：

```
String hql1 = " select count(*) from  cn.itcast.domain.Customer  ";//完整写法
String hql2 = " select sum(cust_id) from  cn.itcast.domain.Customer  ";//完整写法
String hql3 = " select avg(cust_id) from  cn.itcast.domain.Customer  ";//完整写法
String hql4 = " select max(cust_id) from  cn.itcast.domain.Customer  ";//完整写法
String hql5 = " select min(cust_id) from  cn.itcast.domain.Customer  ";//完整写法

Query query = session.createQuery(hql5);

Number number  = (Number) query.uniqueResult();
```

h.多表查询：

```
    //回顾-原生SQL
    //交叉连接--笛卡尔积（避免）
//      select * from A,B
    //内连接
//      |-隐式内连接
//          select * from A,B where b.aid=a.id
//      |-显示式内连接
//          select * from A inner join B on b.aid=a.id
    //内连接
//      |-左外
//          select * from A left[outer] join B on b.aid=a.id
//      |-右外
//          select * from A right[outer] join B on b.aid=a.id
//-----------------------------------------------------------------
    //HQL的多表查询
        //内连接(迫切)
        //外连接
//          |左外(迫切)
//          |右外(迫切)
```

hql内连接查询：

```
//HQL内连接---将连接的两端对象分别返回，放到数组中
@Test
public void fun1(){
    Session session = HibernateUtils.openSession();
    Transaction tx = session.beginTransaction();
    //----------------------------------------
    String hql = "from Customer c inner join c.linkMens";

    Query query = session.createQuery(hql);
    List<Object[]> list = query.list();

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

    //----------------------------------------
    tx.commit();
    session.close();
}
```

hql迫切内连接：

```java
//HQL 迫切内连接---帮我们进行封装，返回值就是一个对象；
@Test
public void fun2(){
    Session session = HibernateUtils.openSession();
    Transaction tx = session.beginTransaction();
    //----------------------------------------
    String hql = "from Customer c inner join fetch c.linkMens";

    Query query = session.createQuery(hql);
    List<Customer> list = query.list();

    System.out.println(list);

    //----------------------------------------
    tx.commit();
    session.close();
}
```

hql左外连接：

```java
String hql = " from Customer c left join c.linkMens ";

Query query = session.createQuery(hql);

List<Object[]> list = query.list();

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

hql右外连接：

```java
String hql = " from Customer c right join c.linkMens ";

Query query = session.createQuery(hql);

List<Object[]> list = query.list();

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

2、Criteria查询：单表条件查询，无语句面向对象查询；

a.基本查询：

```java
//创建criteria查询对象
Criteria criteria = session.createCriteria(Customer.class);
List<Customer> list = criteria.list();

System.out.println(list);
```

b.条件查询：

```
//条件查询
// >                gt
// >=               ge
// <                lt
// <=               le
// ==               eq
// !=               ne
// in               in
// between and      between
// like             like
// is not null      isNotNull
// is null          isNull
// or               or
// and              and
```

```
//创建criteria查询对象
Criteria criteria = session.createCriteria(Customer.class);
//添加查询参数--查询cust_id为1的customer对象
criteria.add(Restrictions.eq("cust_id", 2l));
//执行查询获得结果
Customer c =(Customer) criteria.uniqueResult();

 System.out.println(c);
```

c.分页查询：

```
//创建criteria查询对象
Criteria criteria = session.createCriteria(Customer.class);
//设置分页信息
criteria.setFirstResult(0);
criteria.setMaxResults(1);
List<Customer> list = criteria.list();

System.out.println(list);
```

d.查询总记录数：

```
//创建criteria查询对象
Criteria criteria = session.createCriteria(Customer.class);
//设置查询的聚合函数---总行数
criteria.setProjection(Projections.rowCount());
Long count = (Long) criteria.uniqueResult();
System.out.println(count);
```

e.排序语法：

```
Criteria c = session.createCriteria(Customer.class);

c.addOrder(Order.asc("cust_id"));
//c.addOrder(Order.desc("cust_id"));

List<Customer> list = c.list();

System.out.println(list);
```

离线查询：脱离session的查询方式

```
@Test
public void fun(){
    //web层/service层
    DetachedCriteria dc = DetachedCriteria.forClass(Customer.class);
    dc.add(Restrictions.idEq(31));

    //-------------------------------------
    Session session = HibernateUtils.openSession();
    Transaction tx = session.beginTransaction();
    //-------------------------------------

    Criteria c = dc.getExecutableCriteria(session);
    List list = c.list();

    System.out.println(list);

    //-------------------------------------
    tx.commit();
    session.close();
}
```

3.原生SQL查询：

a.基本查询(返回数组list)：

```
//1.书写sql语句
String sql = "select * from cst_customer";
//2.创建sql查询对象
SQLQuery query = session.createSQLQuery(sql);
//3.调用方法查询结果
List<Object[]> list = query.list();
```

b..基本查询(返回对象list)：

```java
//1.书写sql语句
String sql = "select * from cst_customer";
//2.创建sql查询对象
SQLQuery query = session.createSQLQuery(sql);
//指定将结果集封装到哪个对象中
query.addEntity(Customer.class);
//3.调用方法查询结果
List<Customer> list = query.list();

System.out.println(list);
```

c.条件查询：

```java
//1.书写sql语句
String sql = "select * from cst_customer where cust_id = ?";
//2.创建sql查询对象
SQLQuery query = session.createSQLQuery(sql);

query.setParameter(0, 1l);
//指定将结果集封装到哪个对象中
query.addEntity(Customer.class);
//3.调用方法查询结果
List<Customer> list = query.list();

System.out.println(list);
```

d.分页查询：

```java
//1.书写sql语句
String sql = "select * from cst_customer limit ?,?";
//2.创建sql查询对象
SQLQuery query = session.createSQLQuery(sql);

query.setParameter(0, 0l);
query.setParameter(1, 1l);
//指定将结果集封装到哪个对象中
query.addEntity(Customer.class);
//3.调用方法查询结果
List<Customer> list = query.list();

System.out.println(list);
```

十一、Hibernate一对多关系映射：

orm元数据表达：

```
<!-- 集合,一对多关系,在配置文件中配置 -->
<!--
    name属性:集合属性名
    column属性: 外键列名
    class属性: 与我关联的对象完整类名
 -->
 <!--
    级联操作: cascade
        save-update: 级联保存更新
        delete:级联删除
        all:save-update+delete
    级联操作: 简化操作.目的就是为了少些两行代码.
  -->
  <!-- inverse属性: 配置关系是否维护.
        true: customer不维护关系
        false(默认值): customer维护关系

    inverse属性: 性能优化.提高关系维护的性能.
    原则: 无论怎么放弃,总有一方必须要维护关系.
    一对多关系中: 一的一方放弃.也只能一的一方放弃.多的一方不能放弃.
  -->
<set name="linkMens" inverse="true" cascade="save-update"  >
    <key column="lkm_cust_id" ></key>
    <one-to-many class="LinkMan" />
</set>
```

```
<!-- 多对一 -->
<!--
    name属性:引用属性名
    column属性: 外键列名
    class属性: 与我关联的对象完整类名
 -->
 <!--
    级联操作: cascade
        save-update: 级联保存更新
        delete:级联删除
        all:save-update+delete
    级联操作: 简化操作.目的就是为了少些两行代码.
  -->
  <!-- 多的一方: 不能放弃维护关系的.外键字段就在多的一方.  -->
<many-to-one name="customer" column="lkm_cust_id" class="Customer"  >
</many-to-one>
```

级联操作:cascade，简化操作.目的就是为了少些两行代码.

save-update: 级联保存更新

delete:级联删除

all:save-update+delete

inverse属性: 配置关系是否维护.

true: customer不维护关系

false(默认值): customer维护关系

inverse属性: 性能优化.提高关系维护的性能.

原则: 无论怎么放弃,总有一方必须要维护关系.

一对多关系中: 一的一方放弃.也只能一的一方放弃.多的一方不能放弃.

十二、Hibernate多对多关系映射:

orm元数据表达:

```
<!-- 多对多关系表达 -->
<!--
    name: 集合属性名
    table: 配置中间表名
    key
     |-column:外键,别人引用"我"的外键列名
     class: 我与哪个类是多对多关系
     column:外键.我引用比人的外键列名
 -->
<set name="roles" table="sys_user_role" cascade="save-update"  >
    <key column="user_id" ></key>
    <many-to-many class="Role" column="role_id" ></many-to-many>
</set>


结论: 将来在开发中,如果遇到多对多关系.一定要选择一方放弃维护关系.
     一般谁来放弃要看业务方向. 例如录入员工时,需要为员工指定所属角色.
     那么业务方向就是由员工维护角色. 角色不需要维护与员工关系.角色放弃维护
 -->
<set name="users" table="sys_user_role" inverse="true"  >
    <key column="role_id" ></key>
    <many-to-many class="User" column="user_id" ></many-to-many>
</set>
```

十三、Hibernate的查询优化:

类级别查询:

get方法:没有任何策略.调用即立即查询数据库加载数据.

load方法: 应用类级别的加载策略

关联级别查询:

集合策略: lazy属性:决定是否延迟加载

true(默认值):延迟加载，懒加载

false:立即加载

extra:及其懒惰

fetch属性:决定加载策略，使用什么类型的sql语句加载集合数据

select(默认值):单表查询加载

join:使用多表查询加载集合

subselect:使用子查询加载集合

关联属性策略:

结论:为了提高效率.fetch的选择上应选择select. lazy的取值应选择 true. 全部使用默认值.

no-session问题解决: 扩大session的作用范围.添加一个过滤器;