**Hibernate项目需要的一些配置：**

1. 实体类（POJO类）：持久化JavaBean类
2. 对象-关系映射文件（\*.hbm.xml）

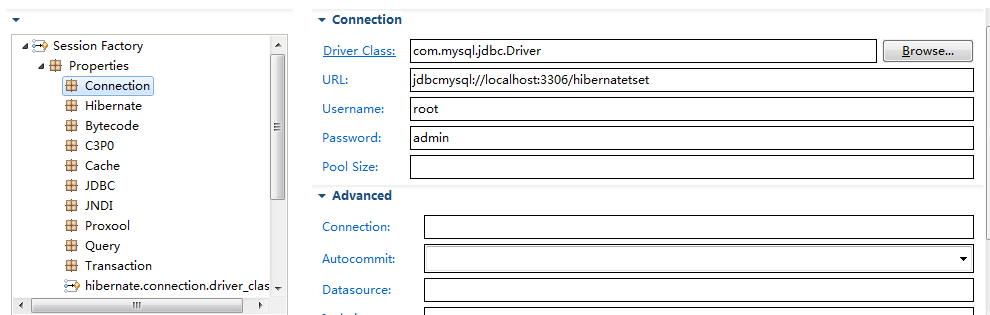
文件要求：文件名：实体类名称.hbm.xml 如：Person.hbm.xml

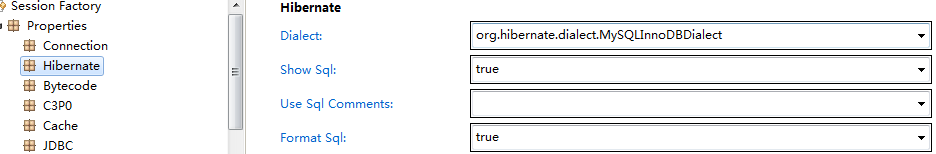
文件位置：和实体类在同一目录下

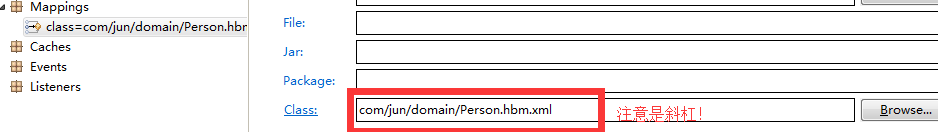
1. 主配置文件（hibernate.cfg.xml）

在src目录下创建hibernate.cfg.xml文件，配置Properties下的Connection和Hibernate，最后要映射实体类的*hbm.xml*

Connection：



Hibernate:

Mapping：

<!-- 数据库连接配置 -->

<session-factory >

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/hibernate5?characterEncoding=UTF-8</property>

<property name=*"connection.characterEncoding"*>utf-8</property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.password"*>admin</property>

<!--数据库方言 -->

<property name=*"hibernate.dialect"*>org.hibernate.dialect.MySQL5InnoDBDialect</property>

<!-- 是否打印sql -->

<property name=*"hibernate.show\_sql"*>true</property>

<!-- 是否格式化sql -->

<property name=*"hibernate.format\_sql"*>true</property>

<!-- ddl是数据库定义语言 -->

<property name=*"hibernate.hbm2ddl.auto"*>update</property>

<!-- 将实体类.hbm.xml映射上来 -->

<mapping resource=*"com/jun/domain/Person.hbm.xml"*/>

</session-factory>

1. 编写Hibernate提供的API执行程序（）
2. 创建Configuration读取配置文件:

**Configuration cfg = new Configuration();**

**cfg.configure();**

1. 创建一个SessionFactory工厂类

**SessionFactory sf = cfg.buildSessionFactory();**

1. 通过工厂开启Session对象

**Session session = sf.openSession();**

1. 开启事务

**Transaction transaction = session.beginTransaction();**

1. 执行数据库的操作

**session.save(Object);**

1. 提交事务

**transaction.commit();**

1. 关闭session

**session.close();**

1. 关闭工厂类

**sf.close();**

**简单的CRUD操作**：可能出现两种错误：更新时没传入主键；更新时没有对应的主键

**save:插入数据**

**update：更新数据，必须有主键**

**saveOrUpdate：插入或更新数据，如有主键参与则为更新，否则为插入**

**load: load(Object.Class,id) 根据id得到数据;当没有匹配到数据时报错**

**get：get(Object.Class,id)根据id得到数据；当没有匹配到数据时返回null**

**session中的一级缓存**：session中存在缓存，这种缓存叫做一级缓存，存在整个session生命周期中，缓存的对象不会随着引用变量消失而清空，只要缓存中存在该对象，在查询时，hibernate**优先从缓存中取出数据**，而不会再次访问数据库，只有缓存中不存在数据，才会发起sql语句查询数据库，缓存极大地减少了访问数据库的频率

**作用：1）减少访问数据库的频率**

**2）保证缓存中的对象与数据库的记录保持同步，当缓存的对象改变后，session不会立即执行sql，而是将多个sql语句合并为一条sql执行，提高效率**

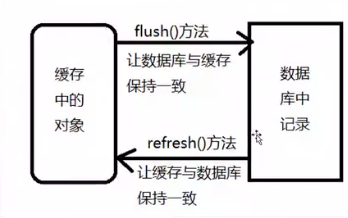
**注意：如果对于同一个属性修改了多次，其实hibernate的session缓存并不是执行了多个update语句，而是以最后一个更新为准而发送一条更新的sql**

缓存相关方法：

1. flush：强制让数据库中的记录与session中的对象数据保持一致，不一致就发起update语句进行修改 （**缓存是主人，数据库跟着缓存走**）**平常不使用，批量处理使用**

**注意：**事务的commit()中已经自动调用flush()，若缓存数据发生改变，flush()会执行update语句，但不会提交事务，提交事务就意味着将数据持久化到数据库

1. reflush：强制发出select语句，保证session缓存中数据和数据库中数据是一直的，若不一致他会**修改缓存中的对象的数据，让它与数据库一致（数据库是主人，缓存跟着数据库走）**
2. clear：清理session，会重新发起sql语句



**---------------------------------------------------------------------------------------------------------------------------------**

**Hibernate对象状态**

1. 状态分类

Hibernate为了管理持久化类：将持久化类分成3个状态

* 瞬时态：Transient Object

没有持久化标识OID（主键），没有被纳入session对象的管理

* 持久态：persistent Object

有持久化表示OID，已经被纳入session对象的管理

* 脱管态(游离态)：Detached Object

有持久化表示OID，没有被纳入session对象的管理

1. 详细描述

**瞬时态**：（Person p=new Person()）对象由**new操作服创建**，且尚未与Hibernate session关联的对象。瞬时态对象不会被持久化到数据库中，也不会被赋予持久化标识。如果程序中失去了瞬时态对象的引用，瞬时态对象将被垃圾回收机制销毁。使用Hibernate session可以将其变为持久态

**持久态：**（session.get();）持久化实例在数据库中有相对应的记录，并拥有一个持久化标识，持久化的实例可以是刚刚保存的，也可以是刚被加载的。无论哪一种，持久化对象都必须与指定的Hibernate session关联。Hibernate会检测到处于持久化对象的改动，在当前操作完成是将**对象数据写回数据库**

**脱管态：**某个实例曾经处于是持久化状态，但随着与之**关联的session被关闭，**该对象变成脱管态。脱管态的引用依然有效，对象可继续被修改。如果重新让脱管态与某个session关联，这个脱管对象会重新转换为持久化状态



对象状态之间的转换：

1. 状态转换

* 瞬时态：Transient Object

如何获得---new出来

持久态--save、saveOrUpdate 保存操作

脱管态—setld设置OID持久化标识（这个id是数据库中存在的）

* 持久态：persistent Object

如何过得—通过session查询方法获得get、load、creatQuery、createCriteria

瞬时态—delete删除操作（数据表不存在对应记录）（其实还有id，是是不叫OID）

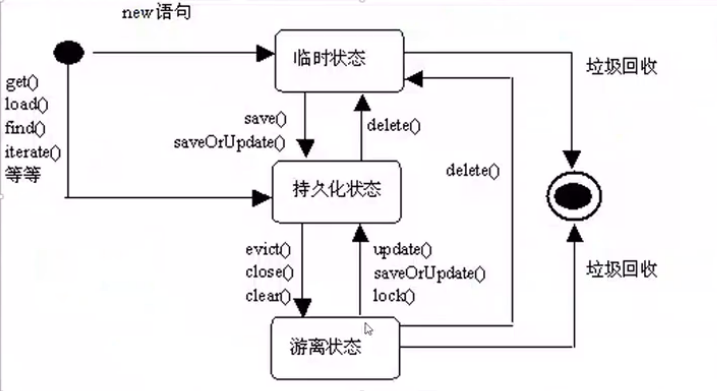
脱管态—close关闭session，evict、clear从session清除对象

* 脱管态(游离态)：Detached Object

如何获得—无法直接获得，必须通过瞬时对象、持久对象转换获得

瞬时态--将id设置为null，或者手动将数据库对应的数据删除

持久态—update、saveOrUpdate、lock(对象重新放入session，重写session关联)



**Session核心方法：**

**save**:

1. 使一个临时对象变为持久化对象
2. 为对象分配ID
3. 执行save会发送一条insert语句，**不是立马作用于数据库，需要等待事务提交**
4. 在save之前配置的ID是无效的(前提是在.hbm.xml中设置的主键策略决定)
5. 持久化对象的ID是不能被修改的

**persist:**与save相似

与save区别：在调用persist之前，若对象已有了ID，会抛出异常

**update：**

1. 若更新一个持久化对象，不需要显式的调用update，因为在调用transaction的commit时会先执行session的flush
2. 更新一个游离对象，需要显式调用session的update。Update一把一个游离对象变成持久化对象

**注意：**

1. 无论要更新的游离对象和数据表的记录是否一致，都会发送update语句。可以通过设置.hbm.xml文件的class节点设置select-before-update=ture，但通常不需要设置该属性
2. 若数据表中没有对应的记录，但调用了update，会抛出异常
3. 当update关联一个游离对象是，若session的缓存中已经存在相同的OID持久化对象，会抛出异常，因为在session缓存中不能有两个OID相同的对象

**get&load：**都能从数据库中取数据

**区别：**

* + - 1. 执行get方法：会立即加载对象（即**立即检索，执行sql查询语句**）

执行load方法：若不使用该对象，则不会立即执行查询操作，而是返回一个hibernate代理对象（即**延迟检索**）

* + - 1. get若没找到会返回null

load若没找到会报错

* + - 1. load可能会抛出懒加载异常，如：在初始化代理对象之前关闭session

**saveOrUpdate：**包含了save和update

如果对象是**游离**状态，执行**update**

如果对象是**临时**状态，则执行**save**

判断对象是游离状态还是临时状态的标准：**java对象的OID是否为null**

**注意：若对象的OID不为null，但数据表中还没有和其对应的记录，会抛出异常**

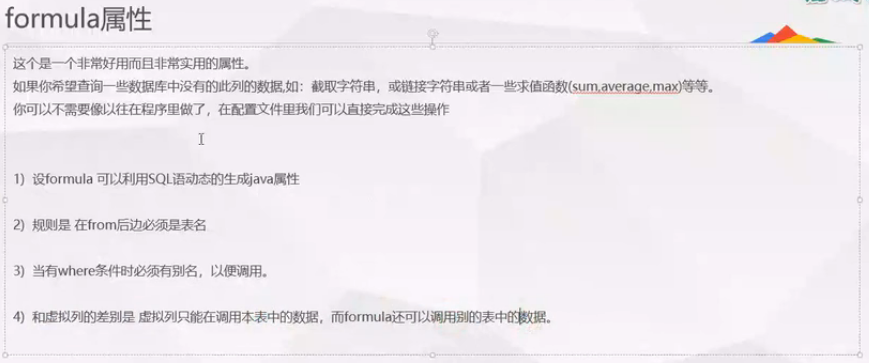
了解：OID的值对于id的unsaved-value属性值的对象，也被认为是一个游离状态

**Hbm映射配置文件**

Hibernate-mapping节点的子节点

1. **class**：为当前映射文件指定持久类名和对应的数据库表名
   * + - **name：**为当前映射文件指定对应的持久类类名
       - **table：**映射到数据库里的表名，这是表名和类名一直
       - **lazy：**是否使用延迟加载
2. **subclass**：指定多态持久化操作时当前映射文件对应的持久化子类
3. **joined**-**subclass**：指定当前联结子类
4. **union-subclass:**指定当前联结子类
5. **query：**定义一个HQL查询

property: formular



**复杂的关联映射配置**

表与表之间关系有：一对多（单向/双向）（one-to-many）、一对一(one-to-one)、多对多

特殊映射：**组件映射**、**继承映射**

**单向多对一**

**业务场景：**学生表，老师表，多个学生对应一个老师

学生为多端，老师是一端

单向：仅多端引用一端

**save**：先增加一端，发起的sql少，先增加多端，发起的sql多

**query**:默认情况下，查询多端的对象，只要没有使用到关联对象，不会发起关联对象的查询，因为hibernate采用懒加载策略，所以在使用关联对象之前，若session关闭，则会出发懒加载异常

**update**:和单表一样

**delete**:删除是需要考虑，一端有没有被多端关联，若有关联，则需要先删除多端相关约束数据

**双向多对一**

**业务场景：**学生表，老师表，多个学生对应一个老师，一个老师对应多个学生，在原有单向基础上，在一端加入多端的引用

Set元素的其他属性：

inverse=true:交出控制权，由另一端来维护关系。默认为false(我方维护)，一般的，在一端的set上配置为true，多端会自动接管控制权

cascade:负责控制关联对象的级联操作，包括更新、删除，也就是对一个对象更新时，其他对象也会受影响，属性值如下：

**save-update**：在session.save/update一个对象时，级联操作关联对象，或者执行save语句或者执行update语句或者什么都不执行

**delete**:在session.delete一个对象的时候，表示执行级联删除，如删除一端数据时，会将多端的关联数据全部删除，一般不建议使用

**all**:save-update和delete结合

order-by:排序字段值：如 order-by=”student\_id ASC”

**一对一主键关联**

业务场景：一个用户信息，分成两个表储存，一个用户表，一个扩展表

**单向一对一：**

**扩展表端.hbm.xml设置**

<id name=*"userId"* type=*"java.lang.Integer"*>

<column name=*"USERID"* />

<!--1、 class中参数改成 foreign 外键-->

<generator class=*"foreign"* >

<!-- 2、指向User对象的这个user对象，一定要在UserExt这个类中配置 -->

<param name=*"property"*>user</param>

</generator>

</id>

<!-- 3、constrained="true" -->

<one-to-one name=*"user"* constrained=*"true"*></one-to-one>

// 4、在扩展表中

// 要依赖另一个表，ext扩展信息基于user这个主键来产生的，ext主键id也依赖user主表

**private** User user;

**双向一对一:**

在另一端加入一个对象类型的属性，并在xml中配置**one-to-one**

**主键关联的特点：一个表中的主键就是外键，指向另一个表中的主键，所以两张表的主键是相同的，但有一个缺点，就是必须依赖另一张表的主键**

第一步：// User类中。双向：通过user可以获得userExt，需要到user.hbm.xml中配置

**private** UserExt userExt;

第二步：主类.hbm.xml<!-- 也要配置one-to-one -->

<one-to-one name=*"userExt"* ></one-to-one>

**一对一外键关联**

业务场景：一个用户信息，分成两个表储存，一个用户表，一个扩展表

**扩展表端.hbm.xml设置**，扩展类中要含有主类：**private** User user;

<!-- 2、加入many-to-one， -->

<many-to-one name=*"user"* class=*"User"* >

<!-- 设置外键列名FUID 并控制唯一性unique="true" -->

<column name=*"FUID"* unique=*"true"*></column>

</many-to-one><!-- 3、constrained="true" -->

<one-to-one name=*"user"* constrained=*"true"*></one-to-one>

**主表端.hbm.xml设置 ：**主类中要含有扩展类： **private** UserExt userExt;

<!-- 4、加入one-to-one 附属表指向本对象属性：property-ref="user"-->

<one-to-one name=*"userExt"* property-ref=*"user"* class=*"com.jun.onetoone.more.UserExt"*></one-to-one>

**多对多关系**

业务场景：班级表，老师表，一个老师教多个班级，一个班级被多个老师教

需求：需要引入**第三方关联关系表，**才能充分体现多对多的关系，但在hibernate中无需建立关系表，框架会自动生成关系表

双向多对多：两边都需要配置**set**标签

<!—clazz中配置 set中的table是中间表 inverse="true":维护关系表的主键-->

<set name=*"teachers"* table=*"R\_TEACHER\_CLAZZ"* inverse=*"true"* lazy=*"true"*>

<key>

<!-- 关系表中指向本类的外键列 -->

<column name=*"CLASS\_ID"* />

</key>

<!-- 此时需要配置关系表和另外一个关联表中外键对应关系 -->

<many-to-many class=*"Teacher"* column=*"TEACHER\_ID"*/>

</set>

<!—teacher中配置： set中的table是中间表 -->

<set name=*"teachers"* table=*"R\_TEACHER\_CLAZZ"* inverse=*"false"* lazy=*"true"*>

<key>

<!-- 关系表中指向本类的外键列 -->

<column name=*"CLASS\_ID"* />

</key>

<!-- 此时需要配置关系表和另外一个关联表中外键对应关系 -->

<many-to-many class=*"Teacher"* column=*"TEACHER\_ID"*/>

</set>

</class>

**组件关系映射**

业务场景：两个对象映射在一张表中，**B对像是A对象的一部分**（A对象和B对象为组合关系）

需求：一辆汽车Car，有用汽车名称，颜色，车轮，车轮数量

关系：此时可以抽取Car和Wheel两个对象，Wheel是Car的一部分，作为Car的组件出现

<!-- 组件 -->

<component name=*"wheel"* class=*"Wheel"*>

<!-- 组件对应的列属性 -->

<property name=*"count"* column=*"wheel\_count"* type=*"Integer"*></property>

<property name=*"size"* column=*"wheel\_size"* type=*"Integer"*></property>

</component>

**Hibernate的检索方式**

1. 检索数据的5中方式
2. **OID检索方式（按照对象的OID检索对象）**
3. **导航对象图检索方式。（根据已加载的对象，导航到其他对象）**
4. **HQL检索方式（使用面向对象的HQL查询语言）**
5. **QBC检索方式（使用QBC API来检索对象）**
6. **本地SQL检索方式**

**HQL检索：（！！！重要）**

1. **全表查询**

// 1、准备HQL语句

String hql = "**from BuyOrder**";// 查询实体名

Query<BuyOrder> query = session.createQuery(**hql, BuyOrder.class**);

List<BuyOrder> list = query.list();

1. **别名查询**

// 1、准备HQL语句

String hql = "**select bo from BuyOrder as bo**";// 查询实体名

List<BuyOrder>list=session.createQuery(**hql,BuyOrder.class**).list();

1. **条件占位符查询**

String hql = "**from BuyOrder where orderId=?0**";// 查询实体名

List<BuyOrder> list = session.createQuery**(hql,BuyOrder.class).setParameter(0, 1)**

.list();

1. **具名查询**

String hql = "from BuyOrder where orderId = :orderId";// 查询实体名

List<BuyOrder> list =

session.createQuery(**hql, BuyOrder.class).setParameter("orderId",2).list()**;

1. **参数为对象（**可以传入实际参数，对象指定ID即可进行查询**）**

BuyOrder buyOrder = **new** BuyOrder();

buyOrder.setOrderId(1);// 给定ID即可进行查询

String hql = "**from OrderDetail where buyOrder = :order**";// 查询实体名

List<OrderDetail> list = session.createQuery

(hql, OrderDetail.**class**).setParameter(**"order", buyOrder**).list();

1. **分页查询**

String hql = "**from OrderDetail**";

Query<OrderDetail> query = session.createQuery(hql, OrderDetail.**class**);

// 当前页pageNo，每页显示的行数pageSize

**int pageNo = 1;**

**int pageSize = 10;**

**query.setFirstResult((pageNo - 1) \* pageSize);**// 起始记录，从哪一行开始查询

**query.setMaxResults(pageSize)**;// 每页最大的显示行数

1. **查询排序**

//排序的是属性，不是列

String hql = "**from OrderDetail order by detialid desc**";// 倒序

Query<OrderDetail> query = session.createQuery(hql, OrderDetail.**class**);

1. **命名查询**

// 命名需要在hbm.xml中定义

Query<OrderDetail> query = session.**createNamedQuery**("**nameTest**", OrderDetail.**class**);

Hbm.xml中

<!-- 在class外面，<![CDATA[[此处内容不做xml处理]]> -->

<query name=*"****nameTest****"*><![CDATA[select o from OrderDetail as o where detialid !=2]]></query>

1. **投影查询**

//new 需要提供与之对应的构造方法

String hql = "select **new BuyOrder(o.orderName)** from BuyOrder o ";

**public BuyOrder(String orderName) {**

**this.orderName = orderName;}**

1. **聚合查询**

String hql = "select o.buyOrder.orderId, count(1),sum(o.price),avg(o.price) from OrderDetail o group by o.buyOrder.orderId";

Query query = session.createQuery(hql);

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

**for** (Object[] objects : list) {

System.***out***.println(Arrays.*asList*(objects));}

**多表查询**

1. 查询方式

和标准的sql查询一样，HQL也支持各种各样的连接查询，如**内连接、外连接**。我们可以通过SQL语句中通过**Join**等关键字实现，表之间的连接查询。HQl同样提供了多表连接查询机制，并新增了**迫切内连接和迫切外连接（立即查询）**

HQL的连接查询适用于：**有关联关系的，并且映射文件中对这种关联关系做了映射持久化类**



**迫切连接：连接是不仅只用了连接查询方式，而且显示的指定了关联级别的查询策略，若使用fetch则关联对象会立即被初始化**。**建议使用fetch。减少查询次数**

使用fetch和没有使用fetch得到的返回类型不一样

**没有使用fetch，不能使用对象接收**

// 连接b中orderDetails对象： b.orderDetails

String hql = "from BuyOrder b inner join b.orderDetails";

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

**使用fetch，只能用对象接收（获取另一对象时只会出发一次查询）**

String hql = "from BuyOrder b inner join **fetch** b.orderDetails";

List<**BuyOrder**> list = session.createQuery(hql).list();

**集成Ehcache**

1. 准备ehcache所需的jar包（hibernate：lib->optional->ehcache）
2. 创建配置文件ehcache.xml到src目录下（直接从lib->project->ect拷贝）
3. 在hibernate.cfg.xml中开启二级缓存Cache下，再加上下面的属性



<property name=*"hibernate.cache.region.factory\_class"*>

org.hibernate.cache.ehcache.internal.EhcacheRegionFactory

</property>

1. 配置使用缓存类（主配置文件最底部）

在主配置文件最下方

<class-cache usage=*"read-only"*class=*"com.jun.hql.OrderDetail"*/>

或在子配置文件**class**中

<!-- 缓存当前类 -->

<cache usage=*"read-only"*/>

开启查询缓存：(查询相同的数据时，会到二级缓存中查找)

在hibernate.cfg.xml中开启查询缓存：

**<propertyname=*"hibernate.cache.use\_query\_cache"*>true</property>**

在查询方法中：

**createQuery.setCacheable(true);**