Hibernate3 Day2

讲师：BoBo老师

内容安排：

1. Hibernate的持久化对象（PO）相关状态和操作。
   1. Hibernate持久化对象的状态（3个）和转换。
   2. Session的一级缓存。
   3. Session一级缓存的快照（snapshot）。
   4. 持久化对象的操作方法(CRUD)。
2. 多表关联映射配置和操作。
3. 一对多关联映射配置和操作、以及级联配置、外键维护配置。
4. 多对多关联映射配置和操作。

学习目标：

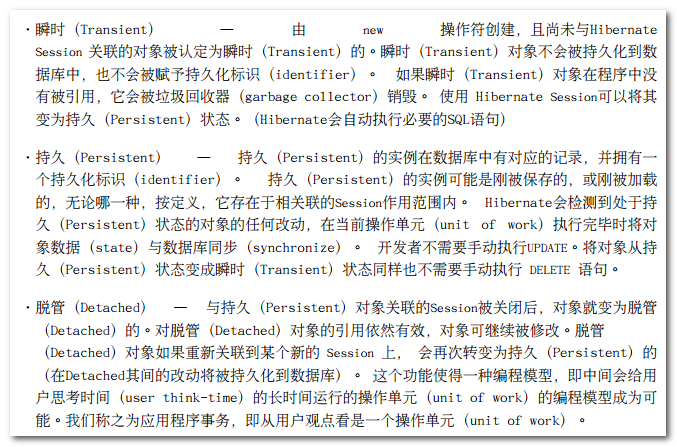
1. 掌握Hibernate的核心概念：PO的状态+一级缓存和快照
2. 掌握多表映射的配置、级联配置和增删改的操作
3. 逐步学会和习惯使用debug

# Hibernate的持久化对象相关概念和操作

## Hibernate持久化对象(po)的状态和转换

### 持久化对象的状态

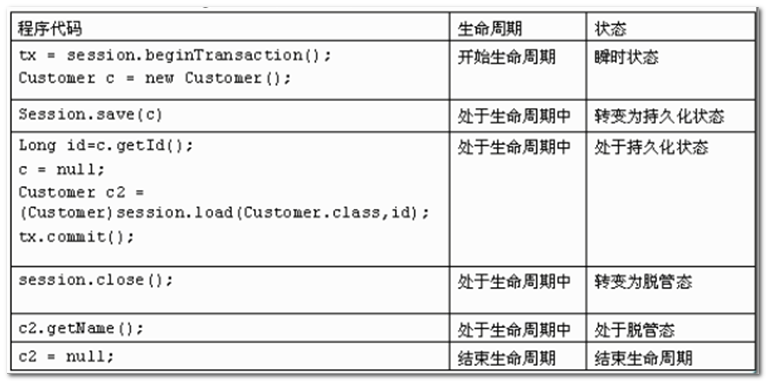
官方描述：



Hibernate 将操作PO对象，状态分为三种

* 瞬时 （Transient ） ： 通常new 创建对象(持久化类)，未与Session关联
* 持久 （Persistent） ： 在数据库存在对应实例，拥有持久化标识OID， 与Session关联（受session管理）
* 脱管 （Detached）：当Session关闭后，持久状态对象与Session断开关联，称为脱管对象，此时也持有OID

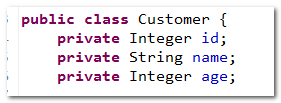
Hibernate中Java对象的状态的转换代码示例：



搭建测试环境：

新建包：cn.itcast.a\_postate

1. 导入开发jar包（12个），将Hibernate.cfg.xml、log4j.properties复制到src，修改Hibernate.cfg.xml的jdbc连接参数，导入hibernateUtils工具类。
2. 编写实体类



1. 编写hbm映射

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <!-- 配置映射 -->  <hibernate-mapping>  <!-- po和表的映射 -->  <class name=*"cn.itcast.a\_postate.Customer"* table=*"t\_customer"*>  <!-- 主键 -->  <id name=*"id"* column=*"id"*>  <!-- 主键策略 -->  <generator class=*"native"*/>  </id>  <!-- 其他属性 -->  <property name=*"name"*/>  <property name=*"age"*/>  </class>  </hibernate-mapping> |

1. 在hibernate.cfg.xml加载映射配置



编写测试类：

|  |
| --- |
| //po状态  @Test  **public** **void** testSaveForPO(){    //瞬时态：new一个对象  //特点：未与session关联，没有OID，数据库没有该数据  Customer customer = **new** Customer();  customer.setName("Rose");  customer.setAge(18);    Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //保存操作  //持久态  //特点：与session关联，“有OID，数据库有对应数据”  //立刻发出insert语句，进行抢占数据库的主键ID，并且hb会将该id返回给po  session.save(customer);    session.getTransaction().commit();    //脱管态  //特点：与session失去联系，有OID，“数据库有对应数据”  session.close();//session关闭  } |

测试内容：

* 查看是否是同一个对象
* 查看对象的主键情况

三种状态的区别分析：

* 持久态和瞬时态、脱管态：容易区别，持久态主要特点是与Session关联，而且数据库中有对应记录、拥有OID，因此，只要与session关联的就是持久态。
* 瞬时态和脱管态：相同点是都和Session没关联，不同点是瞬时态没有OID，而脱管态有OID。

【思考】

通过上述的分析，发现，瞬时态和脱管态对象就差一个OID，那么瞬时态的对象中给主键ID属性赋值后就是脱管态了么？

未必！

首先，需要区分持久化标识OID和对象中主键ID属性的关系：

* 在持久化之前，虽然有ID属性，但数据库中没有对应的数据，那么此时OID是null；
* 在持久化之后，这个ID属性值被插入数据库中当主键了，数据库中有对应的数据了，此时OID就有值了，而且与主键值保持一致性，比如类型、长度等。

因此：OID和PO对象中主键ID属性的区别就是：数据库存在不存在，如果存在就是OID，如果不存在，那就是个ID属性而已。

瞬时态和脱管态的区别扩展：

* 脱管态对象：有持久化的标识oid，并且在数据库中存在。
* 瞬时态对象：无持久化标识oid，或者有id但在数据库中不存在的(自然主键情况)。

例如：

Customer对象具有Id属性值，如果数据库中不存在，则该对象还是瞬时态对象，如果数据库中存在，则认为是脱管态的。

【三者的区别最终总结】：

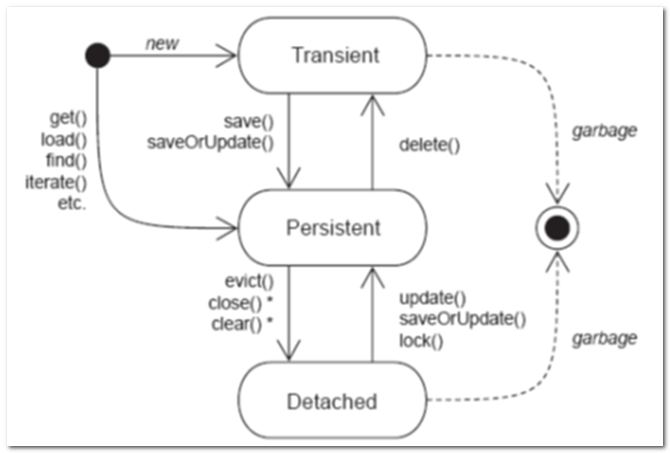
对于三者：在session中存在的，就是持久化对象，不存在的就是瞬时或脱管对象。

对于瞬时和脱管对象：有oid（持久化标识）的就脱管对象，没有的就是瞬时对象。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 是否有持久化标识 | session是否存在 | 数据库中是否有 |
| 瞬时态对象-临时状态 | n | n | n |
| 持久态对象 | y | y | y/（n：没有提交） |
| 脱管态对象 | y | n | y |

### 持久化对象状态的相互转换

持久化对象状态转换图（官方规范）：



分解分析（对象的获取和如何转换到其他状态）：

* 瞬时对象：

如何直接获得 --- new 出来

转换到持久态 ---- save、saveOrUpdate 保存操作

转换到脱管态 ---- setId 设置OID持久化标识（这个id是数据库中存在的） （不是规范中的）

* 持久对象

如何直接获得 ---- 通过session查询方法获得 get、load、createQuery、createCriteria

转换到瞬时态 ---- delete 删除操作 （数据表不存在对应记录 ）（其实还有id，只是不叫OID）

转换到脱管态 ---- close 关闭Session， evict、clear 从Session清除对象

* 脱管对象

如何直接获得 ----- 无法直接获得 ，必须通过瞬时对象、持久对象转换获得

转换到瞬时态 ---- 将id设置为 null,或者手动将数据库的对应的数据删掉

转换到持久态 ---- update、saveOrUpdate、~~lock~~ （对象重新放入Session ，重新与session关联）

## Session的一级缓存

### 什么是一级缓存？



* 缓存作用：

将数据缓存到内存或者硬盘上，访问这些数据，直接从内存或硬盘加载数据，无需到数据表查询.好处:快!降低数据库压力。

* 一级缓存的生命周期：

Session中对象集合(map)，在Session创建时，对象集合(map)也就创建，缓存保存了Session对象数据，当Session销毁后，集合（map）销毁， 一级缓存释放 ！

* 什么对象会被放入一级缓存？

只要是持久态对象，都会保存在一级缓存 （与session关联的说法的本质，是将对象放入了一级缓存）

### 一级缓存的存在性

【示例】

通过多次查询同一个po对象数据，得到同一个对象，且第二次不再从数据库查询，证明一级缓存测存在。

|  |
| --- |
| //测试一级缓存的存在性  @Test  //测试方法：多次查询，是不是一个对象，看看有没有发出sql  **public** **void** testFirstCacheExist(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //第一次查询:根据id查询  //发生了：立即发出sql语句，查询数据，并将po放入一级缓存  Customer customer =(Customer)session.get(Customer.**class**, 1);  System.*out*.println(customer);    //第二次查询  //发生了：不发出sql，直接从一级缓存拿到数据（通过地址一样可以判断：是引用的同一个对象）  Customer customer2 =(Customer)session.get(Customer.**class**, 1);  System.*out*.println(customer2);    session.getTransaction().commit();  session.close();    } |

测试（同一个对象）：

### 一级缓存的生命周期

一级缓存的生命周期就是session的生命周期，不能跨Session，可以说，一级缓存和session是共存亡的！

【示例】

使用两个不同Session来测试生命周期。（一级缓存和session共存亡）

|  |
| --- |
| //测试一级缓存的生命周期  @Test  //测试方法：关session看还能不能取到缓存的数据（看发不发sql）  **public** **void** testFirstCachelifecycle(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //从 数据库查询出数据:根据id，query对象，cretiea对象，都可以  Customer customer = (Customer) session.get(Customer.**class**, 21);  System.*out*.println(customer);    //再次查询  Customer customer2 = (Customer) session.get(Customer.**class**, 21);  System.*out*.println(customer2);      session.getTransaction().commit();  session.close();//一级缓存随之销毁    //-------------------------------------  System.*out*.println("--------------------------------------");  Session session2 = HibernateUtils.*openSession*();//一级缓存又重新诞生（是另外一个），但此时没数据，空的。  session2.beginTransaction();    //从 数据库查询出数据:根据id，query对象，cretiea对象，都可以  Customer customer3 = (Customer) session2.get(Customer.**class**, 21);//将数据又放入一个新的一级缓存中  System.*out*.println(customer3);    //再次查询  Customer customer4 = (Customer) session2.get(Customer.**class**, 21);//不发语句  System.*out*.println(customer4);//地址相同  session2.getTransaction().commit();  session2.close();    } |

测试（两者地址不同）：

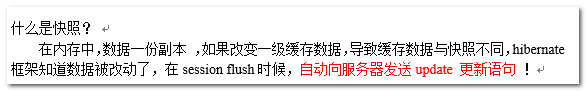
小结：缓存的作用，可以提高性能，减少数据库查询的频率。

[补充：原则]所有通过hibernate操作(session操作)的对象都经过一级缓存。

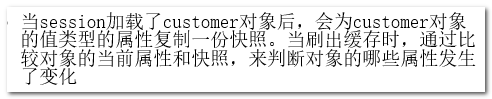
一级缓存是无法关闭的！内置的！hibernate自己维护的！

## Session一级缓存的快照

### 什么是一级缓存的快照(snapshot)

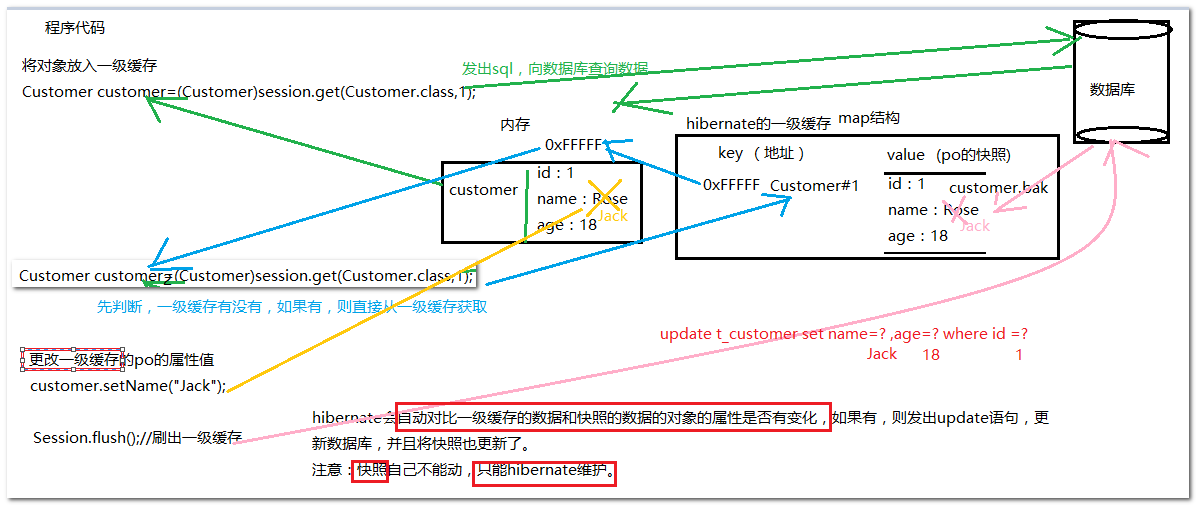


如：



一句话：快照的作用就是用来更新数据的。

### 一级缓存快照的原理（图）



保持一级缓存、数据库、快照的一致性

【注意】

1. 持久态对象原则：po对象尽量保持与数据库一一致。
2. 当一级缓存和快照不一致的时候，会先发出update语句，将一级缓存同步到数据库（发出update语句），然后当同步成功之后，再自动内部同步快照。保证三者的一致性。

### 一级缓存快照的能力

一级缓存的快照是由Hibernate来维护的，用户可以更改一级缓存（PO对象的属性值），但无法手动更改快照！

快照的主要能力（作用）是：用来更新数据！当刷出缓存的时候，如果一级缓存和快照不一致，则更新数据库数据。

【示例】

通过改变查询出来的PO的属性值，来查看一级缓存的更改；通过提交事务，来使用快照更新数据。

|  |
| --- |
| @Test  //测试一级缓存快照的能力（更新数据）  **public** **void** testSnapShot(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //立即发出sql，并且将将数据放入一级缓存（customer是一级缓存对象的引用）和快照  Customer customer =(Customer)session.get(Customer.**class**, 1);  System.*out*.println(customer);    //改变一级缓存的数据  customer.setName("Jack1");  //快照可以局部更新（不用更新所有字段）  //原理：将旧的值更新上去了。  // customer.setAge(88);    //等待提交（会自动flush刷新缓存）  //在刷新缓存的时候，会自动对比一级缓存和快照是否一致，如果不一致，则发出update语句，更新数据到数据库，，并且同步到快照  //不需要手动发出update方法动作  session.getTransaction().commit();  session.close();  } |

【能力扩展】快照可以用来更新数据，而且，可以用来更新部分数据。

### 刷出缓存的时机

什么叫刷出缓存？

Session能够在某些时间点，按照缓存中对象的变化来执行相关的SQL语句，来同步更新数据库，这一过程被成为刷出缓存（flush）。

什么情况下session会执行 flush操作？

刷新缓存的三个时机：

* 事务提交commit()：该方法先刷出缓存（session.flush()），然后再向数据库提交事务。
* 手动刷出flush()：直接调用session.flush()。
* 查询操作：当Query查询（get、load除外,这两个会优先从一级缓存获取数据）时，如果缓存中持久化对象的属性已经发生了变化，（一级缓存和快照不一样了），则先刷出缓存，以保证查询结果能够反映出持久化对象的最新状态。

【补充理解】：

关于Hibernate如何识别是同一一个对象？（持久态）

根据OID，

问题：假如先查询出来一个对象oid是1001，数据库主键也是1001，但其他字段的属性不一样，那么，再次查询数据库的数据出来的对象，和原来的对象是否是一个对象？

答案：是！

【示例】

三种方式刷出一级缓存

|  |
| --- |
| @Test  //刷出一级缓存的时机  **public** **void** testflushcacheTime(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //前提：先将对象放入一级缓存，--持久态对象  Customer customer =(Customer)session.get(Customer.**class**, 1);  System.*out*.println(customer);    //改变一级缓存对象的属性值！  customer.setName("Jack13");  customer.setAge(88);  //等待Hibernate对一级缓存进行刷新    //先用get  //优先走一级缓存，只会从一级缓存取对象，不会去数据库校验，数据是否过期  Customer customer11 =(Customer)session.get(Customer.**class**, 1);  System.*out*.println(customer11);    //query.list是也走一级缓存，但会校验，先得查询，然后flush一下，保证同步  Customer customer12 =(Customer)session.createQuery("from Customer where id=1").uniqueResult();  System.*out*.println(customer12);    //显式的刷新  session.flush();//刷出一级缓存，对比快照，发出update语句    //隐式的刷新session.flush();  session.getTransaction().commit();  //flush和commit区别  //flush作用是控制发出语句的，向数据库发sql的。  //commit:本质：提交事务，只是，hierarnate隐式的flush了一下  session.close();    } |

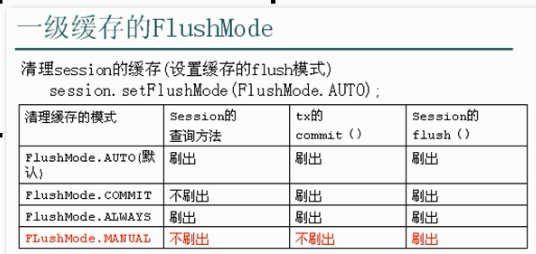
【提示】

flush和commit的区别：

* flush是发语句的。对比快照
* commit的是数据库层面的是否保存更改的数据(是否提交数据，是否持久化数据到数据库),若不手动发出flush， hibernate在commit之前自动先flush()；

### 一级缓存的刷出模式-了解

问题:能否改变一级缓存的刷出时机？答案是可以的.



【示例】

通过在session上设置手动flush模式测试：只能通过flush刷出缓存

|  |
| --- |
|  |

### 一级缓存的常用操作

一级缓存除了可以flush之外，还可以清除（clear，evict）、重载（reflesh）。

* flush：刷出一级缓存

作用：当一级缓存发生变化时，即和快照不同时，刷出一级缓存，会自动向数据库提交update语句。

【扩展提示】

如果是Mysql数据库，开启事务后，当手动flush后，不手动commit，Hibernate不会自动commit；

如果是Oracle数据库，开启事务后，当手动flush后，不手动commit，Hibernate自动commit（oracles事务的提交：手动提交、正常关闭连接的时候也会隐式提交）；

* clear：清除一级缓存

作用：清除一级缓存中的所有对象，这些对象被清除后，会从持久态对象变成脱管态.

【扩展理解】

持久态对象与session关联的另一层含义就是对象在一级缓存中存在。

* evict：清除一级缓存指定对象

作用：清除一级缓存中的指定对象，使对象变成脱管态的。

* reflesh：刷新一级缓存

作用：刷新一级缓存中指定的对象，即发出SQL语句，

将对象重新放入一级缓存，并试图将数据库的数据同步到一级缓存中（同时更新一级缓存和快照），一级缓存之前的如果有修改将被覆盖掉。与flush有相反的感觉。

相关操作示例：

|  |
| --- |
|  |

为什么要清除一级缓存：

大批量处理数据的时候，有些数据无需在一级缓存存在，或者已经处理完了，为了防止内存泄漏，一级缓存爆满，可以手动清除一级缓存的对象。

## 持久化对象的CRUD操作方法

本节将结合持久化对象状态和一级缓存、快照等概念，对PO对象的一些操作进行深入分析和学习。

### 保存操作save()

【状态变化】将瞬时态对象转换为持久态对象。

【深入分析】不同主键策略对保存时发出的insert语句时机的影响。

下面分别对主键策略生成的位置层面（数据库层面和应用程序层面）和不同数据库，进行分析：

1．主键是数据库层面生成的，代表性的主键策略是native（包含identity和sequence），Hibernate都会主动抢占数据表中的主键id的值，但Oracle和MySql的抢占方式不同。

* Oracle：采用序列抢占，即save()执行后，先发出序列查询语句，抢占序列ID值；仅当commit的时候，才会发出insert语句。
* MySQL：采用自增长列抢占，即save()执行后，立即发出insert语句，来抢占自增长列的ID值。

【示例】

测试save的insert语句发出时机—抢占主键

|  |
| --- |
| //测试save的insert语句发出时机  @Test  **public** **void** testFirstCacheExist(){  Customer customer = **new** Customer();  customer.setName("Jack");  customer.setAge(88);    Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //保存操作  session.save(customer);//瞬时转持久  //oracle情况下啊，会只发出序列查询语句，抢占id。只有在commit的时候再发出insertsql  //mysql的情况下：会通过直接发出insert语句来进行抢占id，commit的时候就不需要再发insert    session.getTransaction().commit();  session.close();    } |

2．主键是应用程序层面生成的，代表性的主键策略是assigned（自然主键），因为不需要抢占主键，因此执行save()后，不会立刻发出任何sql，会在事务提交时发出insert语句。

【示例】

测试save的insert语句发出时机—无需抢占主键

|  |
| --- |
| //测试save的insert语句发出时机  @Test  **public** **void** testFirstCacheExist(){  Customer customer = **new** Customer();  customer.setId(1001);//自然主键：手动指定  customer.setName("Jack");  customer.setAge(88);    Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //保存操作  session.save(customer);//瞬时转持久  //oracle情况下啊，会只发出序列查询额语句，抢占id。只有在commit的时候再发出insertsql  //mysql的情况下：会通过直接发出insert语句来进行抢占id，commit的时候就不需要再发insert  //如果是自然主键，（应用程序层面生成主键），由于不需要抢占id，所以，save方法时不发出sql    session.getTransaction().commit();  session.close();    } |

【扩展注意】持久态对象的ID不允许随意更改！，原因是：

Hibernate通过持久态对象的OID来维持它（一级缓存）和数据库相关记录的对应关系，因此，当PO对象处于持久化状态时，不允许程序随意修改它的ID。

【示例】

保存数据之后，更改刚刚保存的数据的id，

|  |
| --- |
| /测试save的insert语句发出时机  @Test  **public** **void** testFirstCacheExist(){  Customer customer = **new** Customer();  customer.setName("Jack22");  customer.setAge(88);    Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //保存操作  session.save(customer);//瞬时转持久    //改变id  customer.setId(2222);//将原先持久态对象的id给改变了，当提交的时候，Hibernate就不知道这个对象是谁了，那么就会报错！      session.getTransaction().commit();  session.close();    }  报错： |

【补充扩展】

补充方法：JPA规范中 ，提供 persist 方法，作用和save类似 ，可以通用。



【示例】

Persist的用法

|  |
| --- |
| //保存  @Test  **public** **void** testSave(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //保存操作:瞬时态-》持久态  Customer customer = **new** Customer();  customer.setId(101);//自然主键的时候必须手动设置id  customer.setName("xiaohong");  customer.setAge(28);    // session.save(customer);//持久态  session.persist(customer);//该方法是jpa规范中    //更改持久态的oid,：一级缓存被更新，快照是旧的，等待flush  // customer.setId(102);  customer.setName("xiaohong2");    session.getTransaction().commit();  session.close();  } |

### 更新操作update()

【状态变化】脱管态的变成持久态的

1． Update方法直接更新

【示例】

使用瞬时态对象构造一个脱管态对象（手动加ID），然后执行update()操作，变成持久态（一级缓存中存在了）

|  |
| --- |
| @Test  //更新操作：脱管态-》持久态  **public** **void** testUpdate(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //脱管态  Customer customer = **new** Customer();  customer.setId(1);//只要设置oid，就将瞬时态变成脱管态  customer.setName("Jack");  //update方法会更新所有字段    //因为不需要抢占主键，所以不发出sql  session.update(customer);//根据id更新    //提交的时候统一发出  session.getTransaction().commit();  session.close();  } |

【注意】

update动作之后，会将所有字段一起更新

2. 持久态对象的快照更新

如果可以拿到持久态对象，那么更新的时候就不需要显式的使用update()，而直接使用一级缓存的快照更新，即使你使用了update()方法，也没有意义了。

相对update()方法，一级缓存的快照更新的优点是只需要改变PO的部分字段，而不用担心其他字段置空。

【示例】

|  |
| --- |
| @Test  //更新操作快照方式：持久  **public** **void** testUpdateBySnapshot(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //弄持久态  Customer customer =(Customer) session.get(Customer.**class**, 1);  //更改一级缓存  customer.setAge(88);  //不需要显示的update方法，写了也没用。  //快照更新的本质还是底层发出了update方法了，还是更新了所有字段，只是快照更新的时候，po拥有就的属性值。  session.getTransaction().commit();  session.close();  } |

3．【关于update方法和快照更新的选择】

* update方法主要是针对脱管态对象的更新，一般要更新PO中的几乎全部的属性字段值时，可以使用。
* 快照更新主要针对持久态对象的更新，一般要更新PO中的部分属性字段值时，推荐使用。

实际开发中，我们一般会更多的使用快照更新，操作方法是：先get(id)查询出持久态的PO，然后改变其属性，再事务提交时实现快照更新。

【update的几个问题】-了解

1．update()方法存在如下问题：不管PO对象数据是否发生变化，都一定会发出更新sql语句，即使程序对象内容和数据表完全一致！

解决办法：

设置hbm的<class>select-before-update属性为true，默认值是false（优化手段）

|  |
| --- |
|  |

【示例】

通过设置更新前查询的属性，在PO和数据库数据一致时不发出更新语句，只有在不一致的时候才发出更新语句。

|  |
| --- |
| //测试更新：脱管态变成持久态  @Test  **public** **void** testUpdate(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //更新  //脱管态  Customer customer = **new** Customer();  customer.setId(1);//设置OID（数据库存在的主键id）  customer.setName("Jack");  customer.setAge(88);  //采用脱管态更新数据的时候，会更新"所有"属性（字段）  //更新操作的时候，会自动将customer放入session，意味着与session关联了。  session.update(customer);  //任何通过session操作的数据，基本上都会经过一级缓存。    session.getTransaction().commit();  session.close();    } |

【扩展了解】

关系型数据库的select语句的成本是最低的（只查询，不用管约束。。等关系），保存、更新、删除成本偏高。（牵扯到约束等关系，效率肯定低）

2．PO对象的ID值在数据库中不存在时，更新会报错。

【示例】

更新一个数据库中不存在ID的对象。

|  |
| --- |
| // 更新不存在的ID的对象会报错。  Customer c = **new** Customer();//瞬时  c.setId(100);//脱管了(数据库不存在的话,其实不是脱管态)  c.setName("Jack2222");  c.setAge(31);  //更新操作  session.update(c); |

3．使用持久态中已经存在的ID的不同对象进行更新会报错，原因是一级缓存中已经存在了这个OID了，假如能更新成功，相当于一级缓存中的两个对象拥有同一个id了！也就是说OID重复了,这是不允许的!

【示例】

|  |
| --- |
| //先将1号员工放入session  session.get(Customer.**class**, 1);    //更新  //脱管态  Customer customer = **new** Customer();  customer.setId(1);//设置OID（数据库存在的主键id）  customer.setName("Jack");  customer.setAge(88);  //采用脱管态更新数据的时候，会更新"所有"属性（字段）  //更新操作的时候，会自动将customer放入session，意味着与session关联了。  session.update(customer);//注意：update的时候，如果sesion中已经有该oid的对象的时候，会报错  //任何通过session操作的数据，基本上都会经过一级缓存。 |

【思考】

如果一级缓存中已经有了要更新的对象，那么如何更新呢？直接采用快照更新(不要再用脱管去更新)。

### 根据OID查询get(),load()

【状态变化】直接拿到持久态对象。

【理解 session的get方法和load方法区别】

两者的区别：

get()方法是立即加载，即执行get方法后，立即发出查询语句进行查询，直接返回目标对象。

load()方法是延迟加载，即执行load方法后，不会立即发出查询语句，返回具有id的目标对象代理类子对象，再访问对象的除了ID之外的其他属性的时候，才发出SQL语句进行查询。

【示例】

分别用get和load，来根据不同id查询不同对象，使用debug来查看语句发出的时机以及延迟加载时的代理类子对象。

|  |
| --- |
| //get和load的区别  @Test  **public** **void** testGetAndLoad(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //get：立即加载，立即发出sql  Customer customer =(Customer)session.get(Customer.**class**, 1);  System.*out*.println(customer);    //load:延迟加载，不会立即发出sql,生成一个仅包含id的代理子对象  Customer customer2 =(Customer)session.load(Customer.**class**, 2);  //只有在访问除id之外属性的时候才发出sql  System.*out*.println(customer2.getId());  System.*out*.println(customer2.getName());      session.getTransaction().commit();  session.close();  }    C2引用的Customer的代理子对象  继续调试    初始化后，内部handler中initialized变为true （已经初始化）， target 指向真正查询结果对象 |

【关于load延迟加载的几个情况提示】

* 代理子对象，handler属性,包含了主键id，因此只访问id时，不需要发出sql语句。

【示例】

延迟加载时，访问ID属性不发出sql，访问ID之外的属性时发出sql

|  |
| --- |
| //load:不会立即发出sql，只有在访问除id之外的其他属性的时候，才发出sql  Customer customer2 =(Customer) session.load(Customer.**class**, 2);  System.*out*.println(customer2.getId());//不需要发出sql  System.*out*.println(customer2.getName());//此时才发出sql |

* 当id在数据库中不存在的时候，访问其他属性时会发生 ObjectNotFoundException。

【示例】

两者查不到数据的区别：

|  |
| --- |
|  |

结论：get会返回null，load会报错：

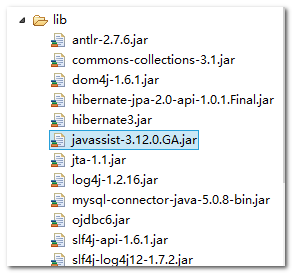


【扩展了解】

代理子对象是谁来负责生成的？



答：通过javassist.jar来进行生成子对象(反射机制)



延迟加载的好处：节约资源，需要的时候再加载（提高内存利用率），如果不需要的话，先不加载。

### 删除操作delete()

【状态变化】可以将持久或脱管，变成瞬时态。

【提示注意】

Hibernate的delete()方法，只通过oID来删除！

【示例】

持久态和脱管态对象都可以删除

|  |
| --- |
| @Test  //删除  **public** **void** testDelete(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //删除脱管  Customer customer = **new** Customer();  customer.setId(101);  session.delete(customer);//不发出sql    //删除持久对象  Customer customer2=(Customer)session.get(Customer.**class**, 100);  session.delete(customer2);    session.getTransaction().commit();//统一发  session.close();  } |

【问题】

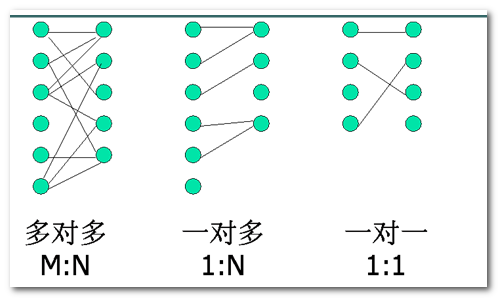
删除持久态和删除脱管态对象，有区别么？

答：单表无区别，但多表会有区别：

* 删除脱管态，不具备级联删除效果 ！
* 删除持久态，在进行级联删除配置之后，可以具有级联删除的效果。

# 多表关联映射配置和操作

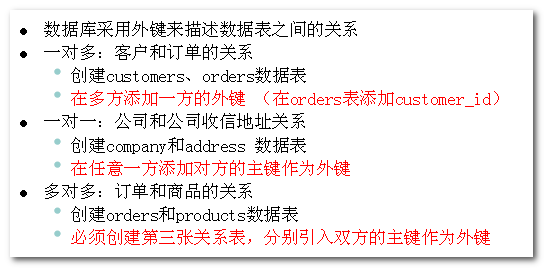
## 多表设计



多对多： 学生选课 （一个学生选多门课， 一门课被多个学生选择）

一对多： 客户和订单 （一个客户可以产生很多订单， 一个订单属于一个客户）

一对一： 一个公司对应一个地址 (一个人员的基本信息会对应一个扩展信息)



**建表规则**：

一对多： 在多方表，添加一方主键作为外键 ，不需要第三张表 ， 需两张表（客户表 、订单表），在订单表添加客户id

多对多： 一定产生第三方关系表， 需要三张表 （学生表、 课程表、 选课表）， 关系表联合主键，引入两张实体表主键，作为外键

一对一： 在任意一方添加对方主键作为外键

## java对象(po)描述表关系

Hibernate 是一个完全ORM框架，使用hibernate 编程，可以完成类和表映射

一对多 ：

Customer {

// 一个客户 多个订单

Set<Order> orders ;

}

Order {

// 一个订单 一个客户

Customer customer ;

}

多对多 ：

Student {

// 一个学生对应 多门课

// Set、List、bag、数组 代表复数，基本区别：set不能重复，没顺序、list可以重复，有顺序。bag：不能重复，而且有顺序（缺点：效率低）

Set<Cource> cources ;

}

Cource {

// 一门课，多个学生

Set<Student> students ;

}

一对一：

Company {

// 一个公司 一个地址

Address address;

}

Address {

// 一个地址 对应一个公司

Company company ;

}

我们下面重点学习一对多和多对多！

**hbm中的多表映射关系配置的一个原则（便于记忆）：**

对多的情况下，先配置集合(set)后配置关系

对一的情况下，直接配置关系。

## 一对多关联映射配置

案例：客户和订单

建立一个包用于测试: cn.itcast.hibernate.d\_onetomany

### 映射配置

编写方法：实体类的编写，先写单表的属性和配置，再加关系。

实体类编写

|  |
| --- |
|  |

hbm映射文件编写

Order.hbm.xml

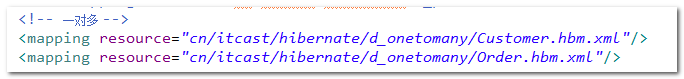
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"cn.itcast.hb.c\_onetomany.Order"* table=*"t\_order"*>  <id name=*"id"*>  <generator class=*"native"*/>  </id>  <property name=*"name"*/>  <!-- 配置关联的属性：直接配置关系 -->  <!-- many-to-one多对一的关系  name:属性的名字  class：一方的类全名  column:自己表中的外键的名称,必须和customer.hbm中配置的key的名字一致！！！  -->  <many-to-one name=*"customer"* class=*"cn.itcast.hb.c\_onetomany.Customer"* column=*"customer\_id"*/>    </class>  </hibernate-mapping> |

Customer.hbm.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  <hibernate-mapping>  <class name=*"cn.itcast.hb.c\_onetomany.Customer"* table=*"t\_customer"*>  <!-- oid属性 -->  <id name=*"id"*>  <generator class=*"native"*/>  </id>  <!-- 普通属性 -->  <property name=*"name"*/>  <!-- 配置关联集合属性 -->  <!--  set标签要和po中属性类型对应  name属性：po中属性名字对应  -->  <set name=*"orders"*>  <!-- 自己在关系表中的外键名称 -->  <key column=*"customer\_id"*/>  <!-- 配置关系  one-to-many代表一对多的关系  class:多方的类全名  -->  <one-to-many class=*"cn.itcast.hb.c\_onetomany.Order"*/>  </set>  </class>  </hibernate-mapping> |

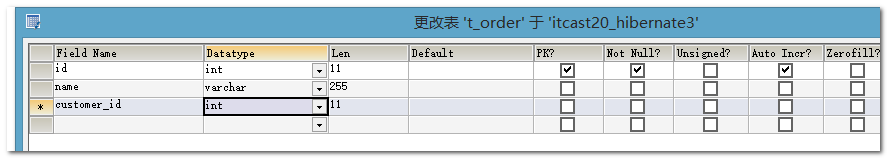
注意：两个配置文件的外键必须对应！！！！！

核心配置文件中添加映射



建表测试

|  |
| --- |
|  |
| 建表成功: |



提示：外键的数据类型自动会使用对方主键的类型

## 相关操作

本节难点：cascade级联和inverse外键维护

### 保存操作Save

多表保存的原则：双方都必须是持久态的对象！

目标：学习几种保存方法。

1. **双向关联保存**

|  |
| --- |
| @Test  //双向关联保存（瞬时态-》持久态）  **public** **void** testSave(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //客户  Customer customer = **new** Customer();  customer.setName("Rose");    //两个订单  Order order1 = **new** Order();  order1 .setName("电视机");  Order order2 =**new** Order();  order2.setName("戒指");    //建立双方的关系（双向建立）  //客户关联订单  customer.getOrders().add(order1);  customer.getOrders().add(order2);  //订单关联客户  order1.setCustomer(customer);  order2.setCustomer(customer);    //保存操作  session.save(customer);  session.save(order1);  session.save(order2);    session.getTransaction().commit();  session.close();    } |

这种保存要求:必须双方都建立关系,而且都要执行保存操作.

1. **级联保存**

【需求】

保存客户的同时自动保存订单。

|  |
| --- |
| 默认情况下：    会报错： |

原因结论： 在hibernate代码中，在session.flush前，不允许 持久态对象 关联 瞬时态对象

持久态对象只能关联持久态对象！

解决：**采用级联 ，cascade ：cascade="save-update"**

它的作用：

**可以使持久态对象”关联”瞬时态对象， 自动会隐式执行save操作，变为持久态，**

**可以使持久态对象”关联”脱管对象，自动会隐式执行update操作，变为持久态**

如果通过操作customer来级联保存 order ，需要在Customer.hbm.xml(谁是持久的) 配置级联.

Customer.hbm.xml:

|  |
| --- |
|  |

使用级联之后,Hibernate会对瞬时态的这个对象，会自动执行save操作.

问题:如果保存顺序反过来呢？即先保存订单，同时保存客户呢？  
分析:先操作order ，级联保存 customer ，需要在 Order.hbm.xml 配置级联

Order.hbm.xml :

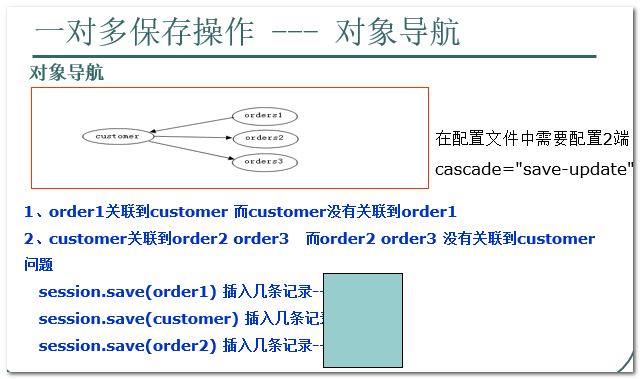
|  |
| --- |
| //双向关联保存测试  @Test  **public** **void** testSave(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //操作：双向关联保存  //弄几个瞬时态对象  //客户  Customer customer= **new** Customer();  customer.setName("xiaoming");  //2个订单  Order order1 = **new** Order();  order1.setName("空调");  Order order2 = **new** Order();  order2.setName("风扇");    //双向关联  //客户关联订单  customer.getOrders().add(order1);  customer.getOrders().add(order2);  //订单关联客户  order1.setCustomer(customer);  order2.setCustomer(customer);    //保存方法  // session.save(customer);//变成持久态：持久态对象不能主动关联瞬时态对象（不受Hibernate管理的），否则事务提交的时候会报错。  session.save(order1);////变成持久态  session.save(order2);//变成持久态    session.getTransaction().commit();  session.close();  } |

级联对于大量的保存或更新操作非常有用。

1. **对象导航—连续保存—其实是依赖于级联。**

**对象导航概念：通过一个对象的保存操作，可以自动导航到另外一个关联对象的保存操作。**

下面有个面试题，前提：customer和order都配置了级联保存（双向都配置），那么请问下面的1，2，3语句分别产生几条插入的sql语句，



|  |
| --- |
| @Test  //测试保存：对象导航  **public** **void** testSaveByCascadeNavi(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    Customer c = **new** Customer();  c.setName("Rose");    Order o1 = **new** Order();  o1.setName("洗衣机1");    Order o2 = **new** Order();  o2.setName("洗衣机2");    Order o3 = **new** Order();  o3.setName("洗衣机3");    //关系  o1.setCustomer(c);  c.getOrders().add(o2);  c.getOrders().add(o3);    //分别产生几条插入sql：  //1.  // session.save(o1);//订单保存的顺序。    //2.  // session.save(c);    //3.  session.save(o2);    session.getTransaction().commit();  session.close();  } |

Hibernate的外键：是由关系来提供

导航保存对于大量的保存或更新操作非常有用。

### 删除操作delete

表之间的依赖关系:在一对多中 ，多方表(从表) 依赖 一方表(主表) （order依赖customer）。

1. 删除多方的一条数据

|  |
| --- |
| //删除多方，比如：3号订单  // Order o = new Order();  // o.setId(3);  // session.delete(o); |

删除多方 （订单），直接删除

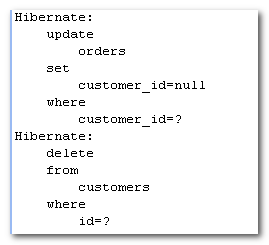
1. 删除一方的一条数据(问:会如何呢?)

|  |
| --- |
| //删除一方的数据:  //会自动解除外键关系，再删除一方  Customer customer = **new** Customer();  customer.setId(6);  session.delete(customer); |

删除一方（客户），被依赖

结果：多方的外键被置空了，一方被删除了(内部机制)





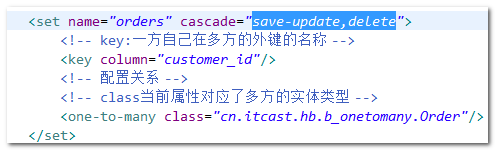
原理：Hibernate 先解除对一方外键依赖，然后进行删除

（如果外键设置 not-null ， 无法删除 ）课后可以试试

1. 级联删除-删除客户自动删除对应的订单（知识点：这里就可以看出删脱管对象和持久对象的区别了）

问题： 从关系型数据库的角度来说:在一对多数据模型中，多方对一方存在依赖的， 如果一方数据被删除，多方数据不完整，无意义。（客户删除的同时，将订单一块都删了）

解决方法： 在一方 配置 cascade=”delete” ,会级联删除.



|  |
| --- |
| //删除一方的数据:  //会自动解除外键关系，再删除一方  //脱管态对象删除，没有级联效果！！！！  // Customer customer = new Customer();  // customer.setId(5);  // session.delete(customer);    //持久态对象删除才有级联效果！！  Customer customer = (Customer)session.get(Customer.**class**, 5);  session.delete(customer); |

结论：删除操作中，删除托管对象没有级联效果，删除持久对象可以进行级联删除

1. 孤儿删除---解除关系的同时并自动删除被解除的对象对应的数据-（主要是删除多方）

订单数据 依赖客户而存在，如果订单没有关联客户，订单是无效的

-------- 没有一方依赖的 多方数据 ： 孤儿

注意：孤儿删除必须持久态对象的操作（解除双方关系）。

在客户的hbm文件中加入配置：



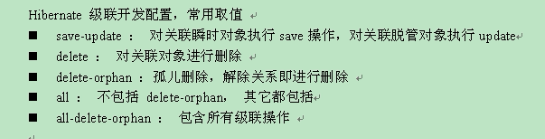
|  |
| --- |
| //孤儿删除：只需要解除关系，即可删除关联的集合对象(必须是持久的)  Customer customer = (Customer)session.get(Customer.**class**, 4);  Order order = (Order) session.get(Order.**class**, 7);  //孤儿删除会隐式的发出delete  customer.getOrders().remove(order); |

注意：要求双方都是持久态的！

结论：孤儿删除就是解除关系后，多方会自动被删除。

### 级联属性配置-cascade





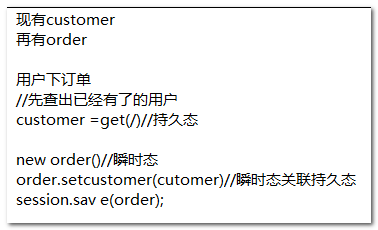
在实际开发中，一对多模型中，一方一般是主动的一方（多方要依赖一方），如果配置级联，通常在一方进行配置！！！

没有说的是删除订单自动就删除了客户的业务逻辑吧。

实际项目开发中，一般级联主要是用来级联删除用的，很少用来级联保存。实际业务中，一般都是先有一方的数据，再有多方的数据，即先有客户，再有订单。所以，保存多方不配置级联（不在多方配置级联）。

前提:现有一方数据了,那如何保存多方数据呢？一般我们也会用多方去关联一方的脱管（有oid）或持久对象。

实际业务系统中，先存在客户，后建立订单。



下面举个保存案例：

【需求】

保存一个订单，这个订单是1号客户的。

|  |
| --- |
| @Test  //现有客户，保存订单（如何给外键赋值）  **public** **void** testSaveOrder(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    //下单：  Order order = **new** Order();  order.setName("床上用品2");  //外键如何赋值：  //只需要给一个有Oid的对象就行（不管是脱管(需要级联)还是持久都可以）  // Customer customer= new Customer();  // customer.setId(4);    Customer customer = (Customer)session.get(Customer.**class**, 4);    //建立关系：给外键赋值  order.setCustomer(customer);    session.save(order);    session.getTransaction().commit();  session.close();    } |

总结：

一般在业务开发中，不要两端都配置级联，（多方尽量不要配置级联。）

扩展（作业），如果都配置级联，必须操作持久态对象，否则可能会出现非预期效果。

### 外键的维护问题-inverse

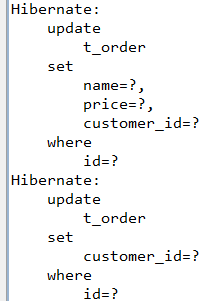
问题:多余sql的问题

示例：

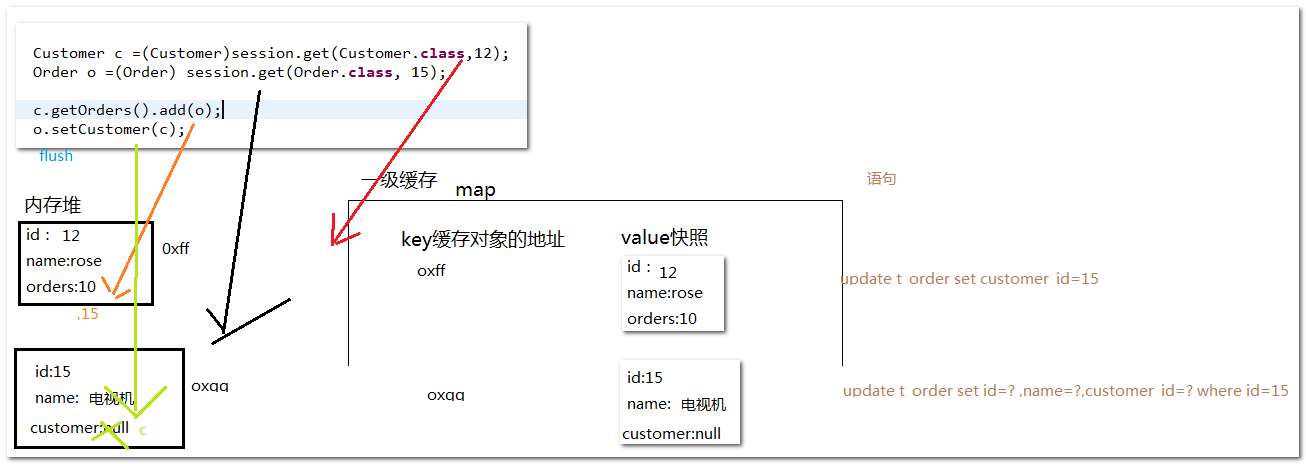
将没有关系的一个客户和一个订单建立关系。(双方)

|  |
| --- |
| //目标：建立客户和订单的关系  //查询1号客户  Customer c = (Customer)session.get(Customer.**class**, 1);  //查询1号订单  Order o = (Order) session.get(Order.**class**, 1);    //分别建立关系  c.getOrders().add(o);//客户关联订单  o.setCustomer(c);//订单关联客户 |

产生的sql:多余sql



分析图：



解决方法：

采用inverse属性来配置。



简单的说这个属性谁是true，就放弃了主键维护权。

inverse默认值是false，即双方都有外键维护权。

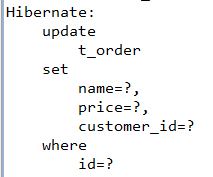
inverse只对集合起作用，也就是只对one-to-many或many-to-many有效.

在业务开发中，一般是在一方放弃。（从业务场景上来分析，一般先存1 方，再存多方，那么就存多方的时候建立关系就比较合理。）





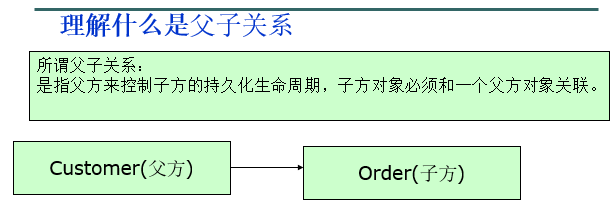
再次运行，发现就一条语句了：



一般，我们都让1方放弃外键维护权！

### 小结

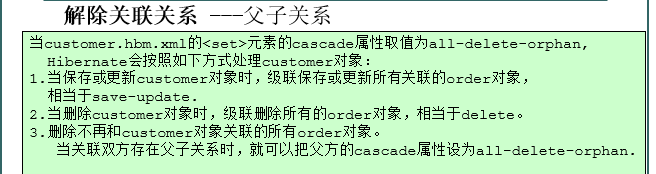
理解一对多的父子关系



一对多 存在父子关系，如果配置级联，在父方配置（一方配置 客户）； 如果配置外键维护权，在父方配置（一方配置 客户 inverse=true）， 将外键维护权 交给多方维护 。

所以，一般，**级联和放弃外键维护的配置都放在一方。**

配置级联、外键维护权



面试题：inverse和cascade的区别（cascade操作的是记录级联的层面，inverse操作的是外键值的层面）。

|  |
| --- |
| @Test  //inverse和cascade的区别  **public** **void** testSaveCascadeInverse(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    Customer c = **new** Customer();  c.setName("小明1");    Order o = **new** Order();  o.setName("彩电1");    c.getOrders().add(o);  //这里并没有order来关联客户  session.save(c);//必须配置级联      session.getTransaction().commit();  session.close();  } |

Cascade 操作记录层面，级联相关操作，保存数据关联数据也被保存

Inverse 操作外键字段层面，如果配置inverse=true， 创建数据没有外键

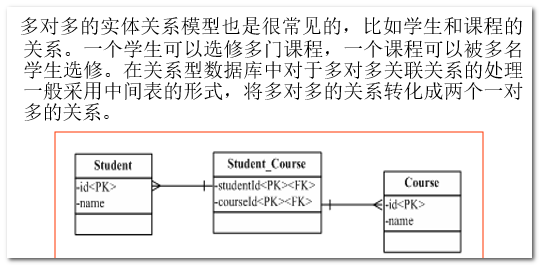
# 多对多关联映射

学习点：掌握多对多配置方式。

案例：学生和课程，学生选课

## 映射配置方式

### 多对多关系分析（student\course）



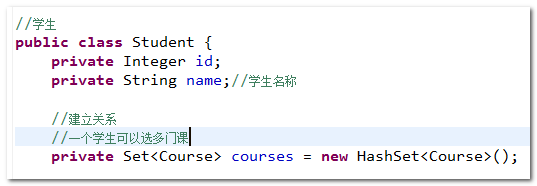
学生和课程 是经典 多对多关系，学生选课是关系表数据

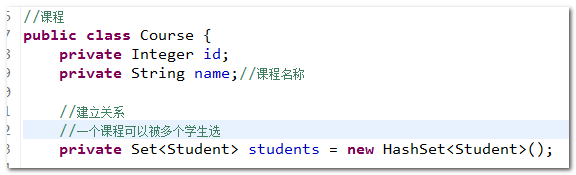
多对多和一对多区别很大，最明显区别，多对多没有父子关系的

在多对多中，一般情况 没必要使用 cascade 级联的！！！

### 映射配置

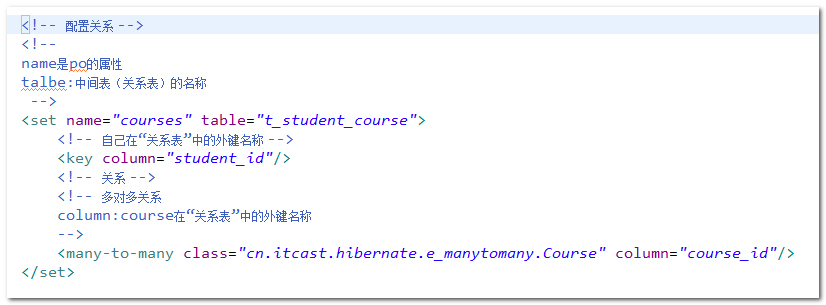
1. 实体类编写





1. hbm映射文件编写

student.hbm.xml



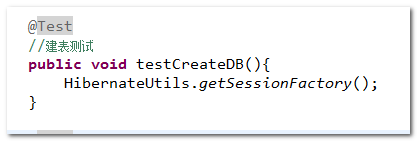
Course.hbm.xml



1. 核心配置中添加映射，

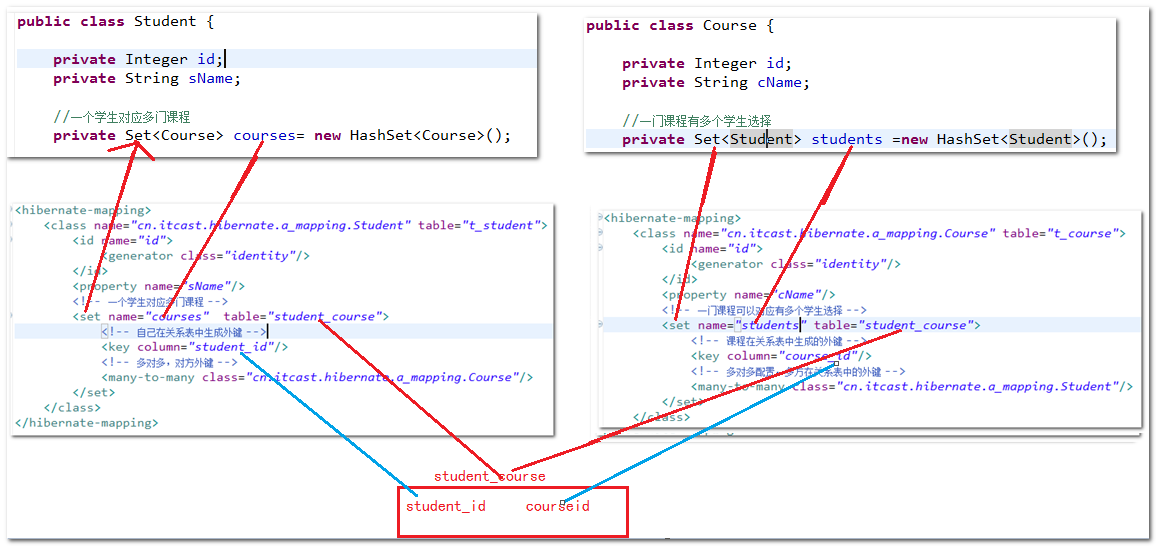


1. 建表测试



### 映射图解

图解：



只要是多对一的标签，都配置自己在对方的外键属性。

## 相关操作

### 保存操作save，同时建立关系

即保存学生和课程数据的同时，在中间选课表插入关联数据。

|  |
| --- |
| @Test  //保存  **public** **void** testSave(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //两个瞬时态对象  //一个学生  Student student = **new** Student();  student.setName("Jack");    //一个课程  Course course = **new** Course();  course.setName("画画");    //建立关系  //学生选课：学生关联课程  student.getCourses().add(course);  //课程被学生选：课程关联学生  // course.getStudents().add(student);    //发生了两次插入中间表：有两种解决方案：  //1.外键维护权，一方放弃inverse="true"  //2.建立关系是，只需要建立一方的关系即可    //保存操作  session.save(student);  session.save(course);        session.getTransaction().commit();  session.close();  }  错误： |

实际业务中，一般不会去级联保存，因为学生和课程是各自产生存在的.

结论：多对多模型中，一次创建关系，对应中间表 一条insert语句 ！不需要双方发生关系。

在实际业务中，要么是只一方建立关系，要么是如果两方都建立关系，就配置inverse，让一方主动放弃维护权。

### 解除关系

两个对象解除关系就是删除中间表的关系数据，即将两个对象解除关系。

类似于孤儿删除（但不是）

|  |
| --- |
| //解除关系：学生不选某们课了，：处理方式是删除中间表的对应的数据  @Test  **public** **void** deleteRelation(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();    // session.delete(arg0);  //只要两个对象解除关系，就会自动删除中间表数据--类似于孤儿删除  //双方都是持久态对象  Student student=(Student)session.get(Student.**class**, 3);  Course course=(Course)session.get(Course.**class**, 3);  //学生不选这个课，解除关系(谁解除谁都可以)  student.getCourses().remove(course);//针对性的发出sql，删除中间表  //等快照flush    session.getTransaction().commit();  session.close();  } |

语句问题：查看语句。

### 改变关系

变更选课内容，原来选的语文，改为选数学

----- hibernate 无法生成update语句 （只能由我们自己先delete，后insert—先解除关系，在增加关系 ）

分析：假如能update，你要update中间表，但中间表无法通过实体操作。

插入测试数据，-

|  |
| --- |
| @Test  //改变关系  **public** **void** changeRelation(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //做法：先解除，再添加新关系    //只要两个对象解除关系，就会自动删除中间表数据--类似于孤儿删除  //双方都是持久态对象  Student student=(Student)session.get(Student.**class**, 2);  Course course=(Course)session.get(Course.**class**, 2);  //学生不选这个课，解除关系(谁解除谁都可以)  student.getCourses().remove(course);//针对性的发出sql，删除中间表    //弄要添加的  Course course3=(Course)session.get(Course.**class**,3);  student.getCourses().add(course3);    //等快照flush    session.getTransaction().commit();  session.close();  } |
|  |

### 删除实体

删除学生的时候，Hibernate会自动删除关系表(中间表)的数据（无需级联）

|  |
| --- |
| //rose退学了  @Test  **public** **void** testDelete(){  Session session = HibernateUtils.*openSession*();  session.beginTransaction();  //直接删除rose，Hibernate会自动删除关系表中的数据，无需手动操作(删除持久态)  Student student =(Student)session.get(Student.**class**, 2);  session.delete(student);    session.getTransaction().commit();  session.close();  } |

注意： 多对多不建议使用 级联 （delete 级联），造成数据丢失 ！！！！！！！(如果两边都配置级联,会两边删除,造成表被清空)

测试一下：（实际业务中是不存在的！！）

在课程或者学生方配置级联



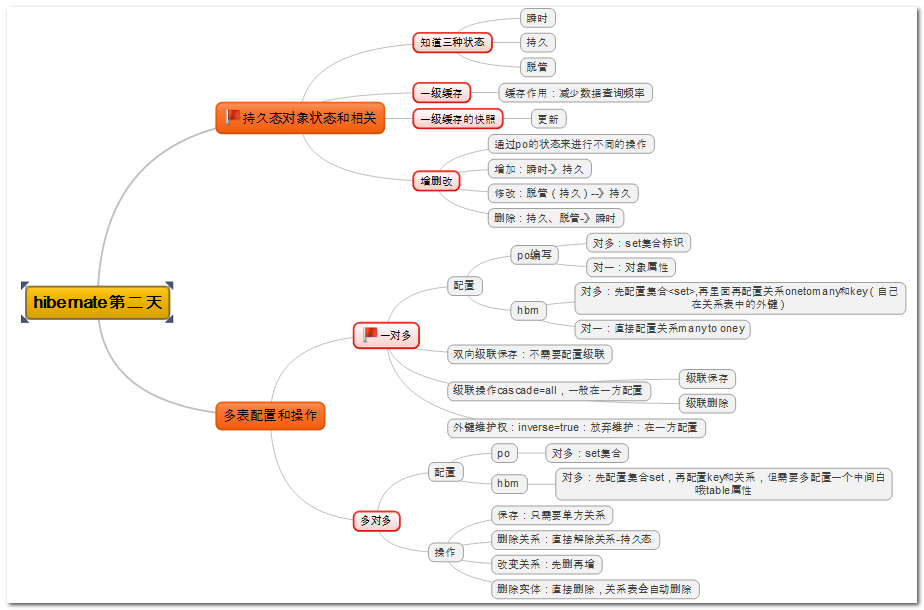


如果在学生方面配置了级联，那么当删除持久态的学生对象时，会将对应的课程也同时删除掉了！从而造成数据丢失！（删除脱管的不会，因为脱管态对级联不起作用）

# 小结+重点

复习：

* 1. 持久化对象的状态：瞬时，持久，脱管
  2. 持久态对象的转换：瞬时态—》持久态：save 托管态---》持久态：update
  3. Session的一级缓存：作用:减少数据库的查询频率，提升性能，生命周期：和session共存亡。
  4. 一级缓存的快照：主要能力：更新数据（快照更新（更新部分）+update方法（更新几乎所有））
  5. CRUD的操作。Save：抢占主键（insert语句发出的时机）（数据库生成的主键的时候，需要抢，其他的时候不需要抢）；update（上面略）；get和load（延迟加载）；delete（单表，删除脱管和持久效果一样。多表有级联效果）
  6. 一对多的映射配置和po的编写。
  7. 一对多的操作：保存（双向保存）、级联的知识点（在关联集合的属性上增加cascade=all），放弃外键维护权（属性inverse=true）--这些配置一般都在一方配置。
  8. 多对多的映射配置和po的编写
  9. 多对多的相关操作。



学会debug，多用debug

# 作业

【作业一】

完成全天课程练习。

【作业二】

完成课前资料中的：《hibernate知识点作业练习》文档中的练习。