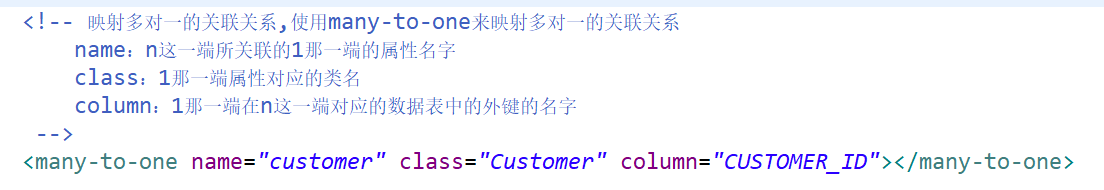
一对多关联关系

1. 单向 n-1
   1. 单向 n-1 关联只需从 n 的一端可以访问 1 的一端
   2. 域模型: 从 Order 到 Customer 的多对一单向关联需要在Order 类中定义一个 Customer 属性, 而在 Customer 类中无需定义存放 Order 对象的集合属性
   3. 关系数据模型:ORDERS 表中的 CUSTOMER\_ID 参照 CUSTOMER 表的主键
   4. 映射的方法



* 1. 多对一关系的增删改查方法的变化
     1. 增（save方法）

如果先save 1那一端的对象（Customer），再save n那一端的的对象（order），会仅发送INSERT语句

而如果先save n那一端的对象（Order），再save 1那一端的对象（customer），会发送INSERT语句和UPDATE语句，原因是，在插入n的一端时，无法确定1的那一端的外键值，所以只能等插入了1那一端之后才能确定外键值，然后，再UPDATE表

所以推荐使用先插入1这一端，这样不会发送UPDATE语句，效率较高

* + 1. 改（update方法）

若从n那一端导航到1的那一端，然后修改对象，则也会像原来那样，发送一条UPDATE语句进行修改

* + 1. 查（get方法）-延迟加载（和load方法一样）

若查询n的一端的对象，则默认情况下，只查询了n的一端的对象，而没有查询1那一端的对象

在需要使用到关联的对象时，才发送对应的SQL语句

在查询Customer对象时，由n的一端导航到1的一端时，若此时session已被关闭，则默认情况下，可以会发生LayerInstantiationException

获取Order对象时，默认情况下，其关联的Customer对象时代理对象

* + 1. 删（delete方法）

在不设定级联关系的情况下，且1这一端的对象有n那一端的引用，则不能删除

设定级联操作，具体见下面set节点的三个重要属性

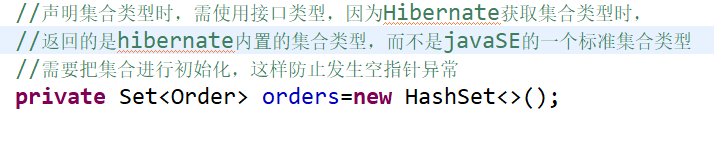
1. 双向1-n
   1. 双向 1-n 与 双向 n-1 是完全相同的两种情形
   2. 双向 1-n 需要在 1 的一端可以访问 n 的一端, 反之依然.
   3. 域模型:从 Order 到 Customer 的多对一双向关联需要在Order 类中定义一个 Customer 属性, 而在 Customer 类中需定义存放 Order 对象的集合属性
   4. 关系数据模型:ORDERS 表中的 CUSTOMER\_ID 参照 CUSTOMER 表的主键
   5. 映射的方法

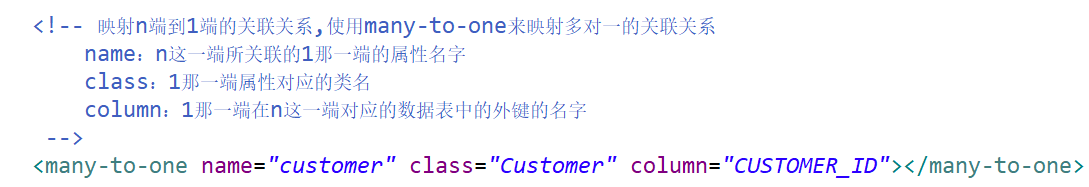
当 Session 从数据库中加载 Java 集合时, 创建的是 Hibernate 内置集合类的实例, 因此在持久化类中定义集合属性时必须把属性声明为 Java 接口类型

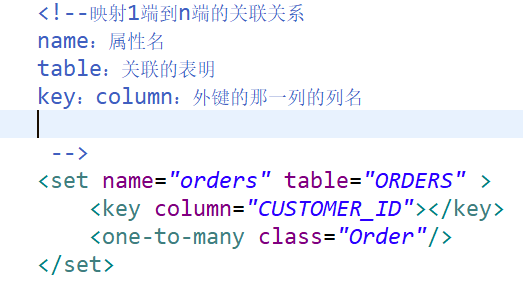
Hibernate 的内置集合类具有集合代理功能, 支持延迟检索策略

事实上, Hibernate 的内置集合类封装了 JDK 中的集合类, 这使得 Hibernate 能够对缓存中的集合对象进行脏检查, 按照集合对象的状态来同步更新数据库。

在定义集合属性时, 通常把它初始化为集合实现类的一个实例. 这样可以提高程序的健壮性, 避免应用程序访问取值为 null 的集合的方法抛出 NullPointerException







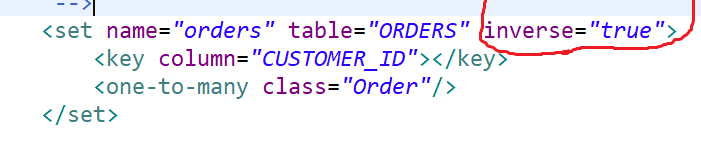
* 1. 双向多对一增删改查方法的变化
     1. 增

如果先save 1那一端的对象（Customer），再save n那一端的的对象（order），会仅发送INSERT语句还会有（两条）UPDATE语句

而如果先save n那一端的对象（Order），再save 1那一端的对象（customer），会发送INSERT语句和（4条）UPDATE语句

多出两条UPDATE的原因：默认情况下因为1的一端和n的一端都维护关联关系，所以会多出两条UPDATE语句

解决方式：让1的一端不维护关联关系（具体见下面set节点的三个重要属性）



* + 1. 查

若查询1的一端的对象，则默认情况下，只查询了1的一端的对象，而没有查询n那一端的对象

在需要使用集合中元素的时候，进行初始化

在查询Order对象时，由1的一端导航到n的一端时，若此时session已被关闭，则默认情况下，可以会发生LayerInstantiationException

获取Order对象时，默认情况下，其关联的Order的集合类型是Hibernate内置的集合类型，该类型具有延迟加载和存放代理对象的功能

* + 1. 改

若从1那一端导航到n的那一端，然后修改对象，则也会像原来那样，发送一条UPDATE语句进行修改

* + 1. 删

和单向一对多一样

* 1. Set节点里的三个重要属性
     1. inverse
* 在hibernate中通过对 inverse 属性的来决定是由双向关联的哪一方来维护表和表之间的关系. inverse = false 的为主动方，inverse = true 的为被动方, 由主动方负责维护关联关系
* 在没有设置 inverse=true 的情况下，父子两边都维护父子

关系

* 在 1-n 关系中，将 n 方设为主控方将有助于性能改善(如果要国家元首记住全国人民的名字，不是太可能，但要让全国人民知道国家元首，就容易的多)
* 在 1-N 关系中，若将 1 方设为主控方
  + **会额外多出 update 语句**。
  + 插入数据时无法同时插入外键列，因而无法为外键列添加非空约束
    1. cascade

用于指定如何操纵与当前对象关联的其他对象.

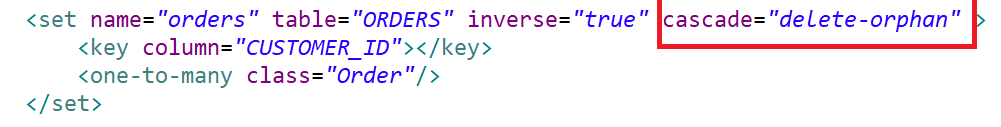


详细说明几个属性值

delete：即在删除时可以级联式删除

delete-orphan：集合调用了clear，即解除了关联关系，那么久把这个对象删除

save-update：级联保存，例如，当保存customer时，也保存了order

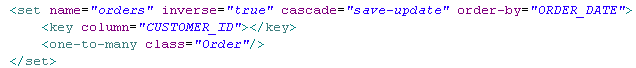


* + 1. Order-by

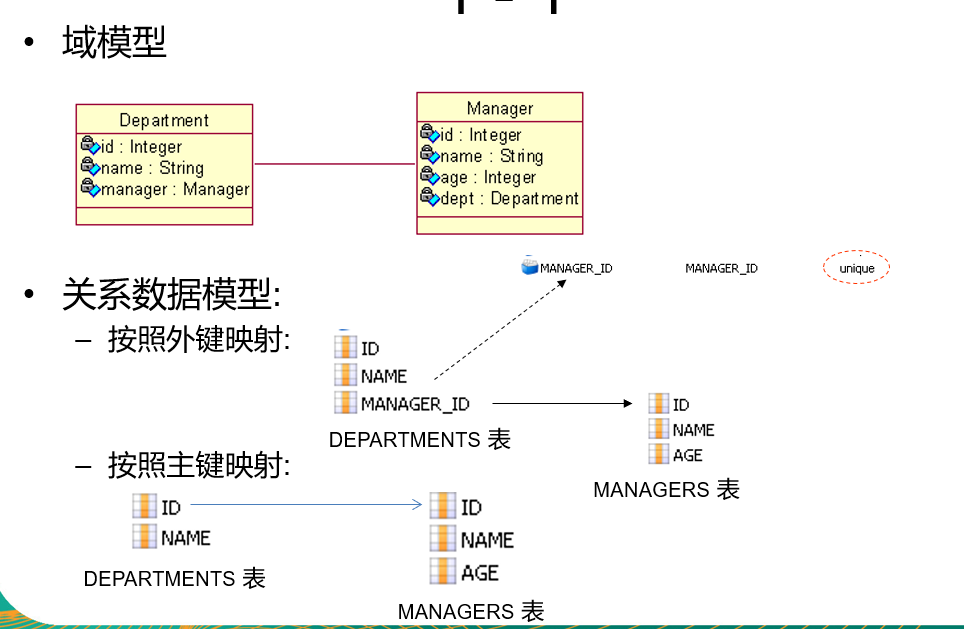
如果设置了该属性, 当 Hibernate 通过 select 语句到数据库中检索集合对象时, 利用 order by 子句进行排序

order-by 属性中还可以加入 SQL 函数

注意：在order-by里面中放置的是表中的列名而不是实体类的属性名

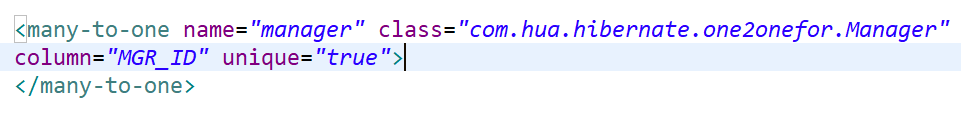


1. 基于外键的1对1关联关系
   1. 域模型
   2. 关系数据模型



* 1. 映射的方法

对于基于外键的1-1关联，其外键可以存放在任意一边，在需要存放外键一端，增加many-to-one元素。为many-to-one元素增加unique=“true” 属性来表示为1-1关联



另一端需要使用one-to-one元素，该元素使用 property-ref 属性指定使用被关联实体主键以外的字段作为关联字段

* 1. 基于外键的一对一映射的增删改查方法的效果
     1. 增

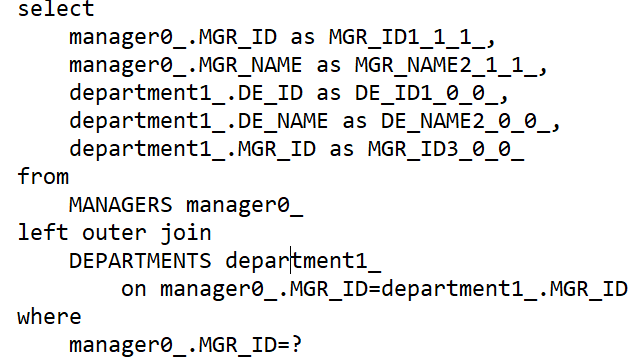
顺序问题会导致多UPDATE语句，所以，建议先保存没有外键列的对象，这样会减少UPDATE语句。

* + 1. 查

默认情况下，对关联属性使用延迟加载

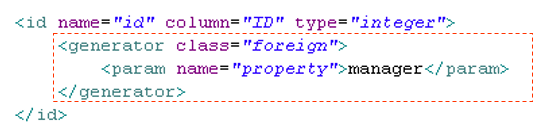
所以会出现懒加载异常的问题（关闭session后得所关联的那个对象）

还有，如果查询并引用没有设置外键的那个实体类对象，则会调用一个左外链接的查询

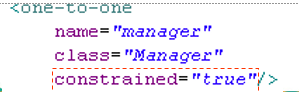


* + 1. 改
    2. 删

1. 基于主键映射的 1-1
   1. 基于主键的映射策略:指一端的主键生成器使用 foreign 策略,表明根据”对方”的主键来生成自己的主键，自己并不能独立生成主键. <param> 子元素指定使用当前持久化类的哪个属性作为 “对方”



* 1. 采用foreign主键生成器策略的一端增加 one-to-one 元素映射关联属性，其one-to-one属性还应增加 constrained=“true” 属性；另一端增加one-to-one元素映射关联属性。
  2. constrained(约束):指定为当前持久化类对应的数据库表的主键添加一个外键约束，引用被关联的对象(“对方”)所对应的数据库表主键



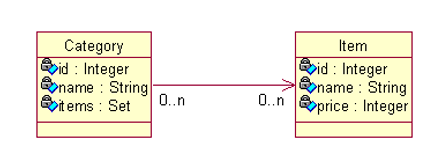
* 1. 增删改查
     1. 增

插入顺序和save方法的先后没有关系，所以不管怎么写save，都会先插入被参照的表，都不会产生多余的UPDATE

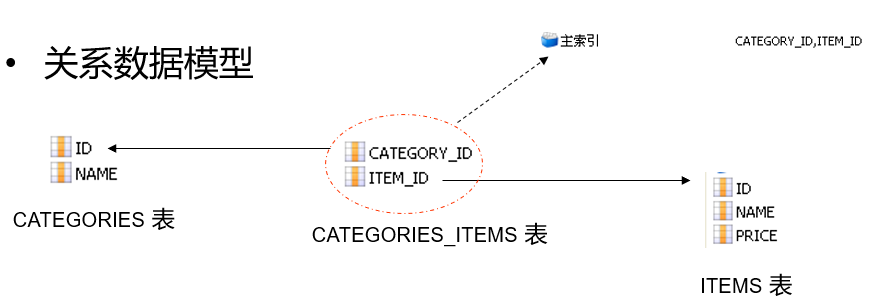
* + 1. 删
    2. 改
    3. 查

当查询主键为参照的那个实体类对象时，跟上边的相同，会调用一个左外链接，

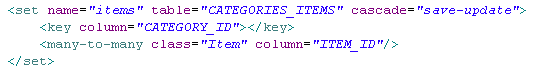
1. 单向多对多的关联关系
   1. 域模型



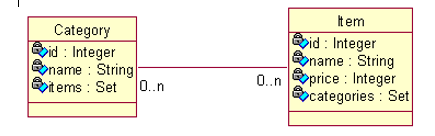
* 1. 关系数据模型



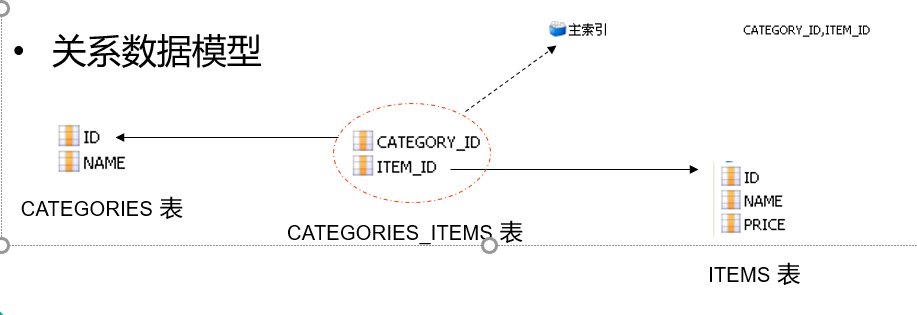
* 1. n-n 的关联必须使用连接表
  2. 与 1-n 映射类似，必须为 set 集合元素添加 key 子元素，指定 CATEGORIES\_ITEMS 表中参照 CATEGORIES 表的外键为 CATEGORIY\_ID. 与 1-n 关联映射不同的是，建立 n-n 关联时, 集合中的元素使用 many-to-many. many-to-many 子元素的 class 属性指定 items 集合中存放的是 Item 对象, column 属性指定 CATEGORIES\_ITEMS 表中参照 ITEMS 表的外键为 ITEM\_ID



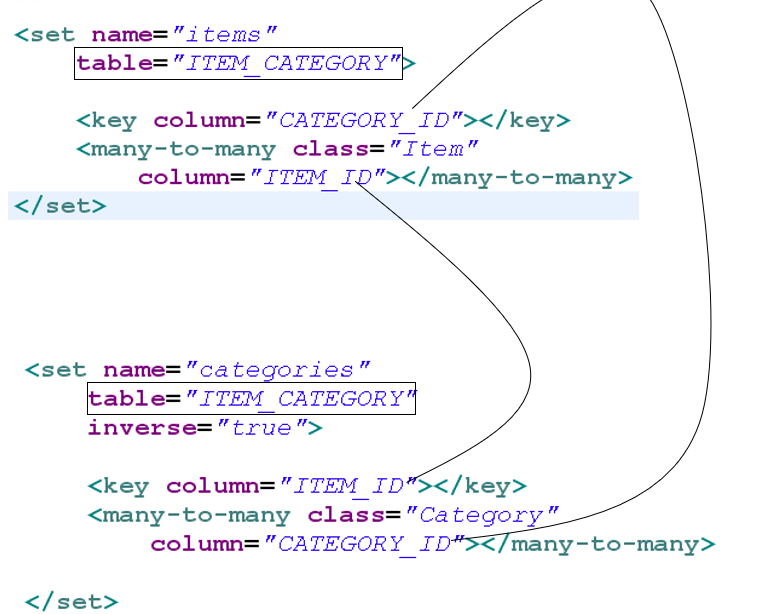
1. 双向n-n关联
   1. 域模型



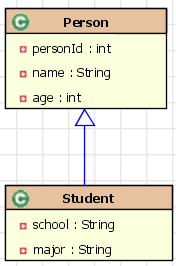
* 1. 关系数据模型



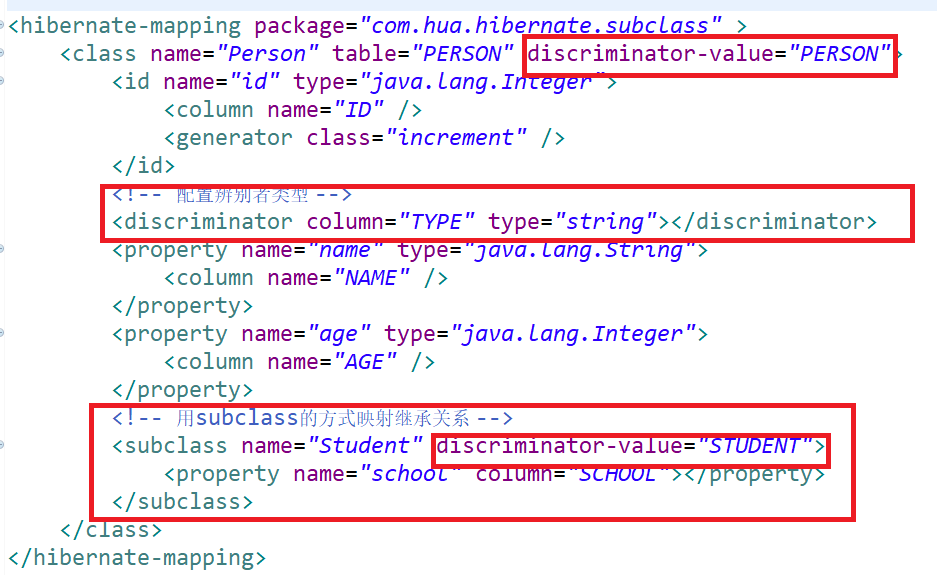
* 1. 双向 n-n 关联需要两端都使用集合属性
  2. 双向n-n关联必须使用连接表
  3. 集合属性应增加 key 子元素用以映射外键列, 集合元素里还应增加many-to-many子元素关联实体类
  4. 在双向 n-n 关联的两边都需指定连接表的表名及外键列的列名. 两个集合元素 set 的 table 元素的值必须指定，而且必须相同。set元素的两个子元素：key 和 many-to-many 都必须指定 column 属性，其中，key 和 many-to-many 分别指定本持久化类和关联类在连接表中的外键列名，因此两边的 key 与 many-to-many 的column属性交叉相同。也就是说，一边的set元素的key的 cloumn值为a,many-to-many 的 column 为b；则另一边的 set 元素的 key 的 column 值 b,many-to-many的 column 值为 a.
  5. 对于双向 n-n 关联, 必须把其中一端的 inverse 设置为 true, 否则两端都维护关联关系可能会造成主键冲突.



1. 映射继承关系
   1. 对于面向对象的程序设计语言而言，继承和多态是两个最基本的概念。**Hibernate 的继承映射可以理解持久化类之间的继承关系**。例如：人和学生之间的关系。学生继承了人，可以认为学生是一个特殊的人，如果对人进行查询，学生的实例也将被得到



* 1. Hibernate支持三种继承映射策略：
     + - 1. 使用 subclass 进行映射：将域模型中的每一个实体对象映射到一个独立的表中，也就是说不用在关系数据模型中考虑域模型中的继承关系和多态。
         2. 使用 joined-subclass 进行映射： 对于继承关系中的子类使用同一个表，这就需要在数据库表中增加额外的区分子类类型的字段。
         3. 使用 union-subclass 进行映射：域模型中的每个类映射到一个表，通过关系数据模型中的外键来描述表之间的继承关系。这也就相当于按照域模型的结构来建立数据库中的表，并通过外键来建立表之间的继承关系。
  2. 使用subclass进行映射
* 采用 subclass 的继承映射可以实现对于继承关系中**父类和子类使用同一张表**
* 因为父类和子类的实例全部保存在同一个表中，因此**需要在该表内增加一列**，使用该列来区分每行记录到低是哪个类的实例----这个列被称为辨别者列(**discriminator**).
* 在这种映射策略下，**使用 subclass 来映射子类**，**使用 class 或 subclass 的 discriminator-value 属性指定辨别者列的值**
* **所有子类定义的字段都不能有非空约束**。如果为那些字段添加非空约束，那么父类的实例在那些列其实并没有值，这将引起数据库完整性冲突，导致父类的实例无法保存到数据库中



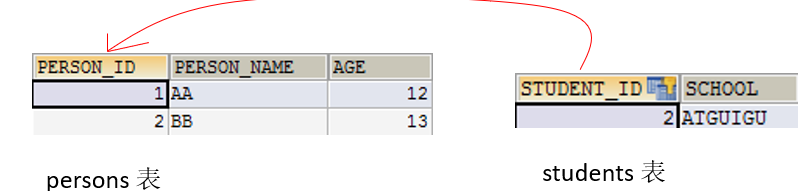
* save方法：对于子类对象只需要把记录插入到一张数据表中，辨别者列由Hibernate自动委会
* 查询方法：查询父类记录，只需要查询一张数据表。对于子类记录也只需要查询一张数据表
* 缺点：使用了辨别者列，使得数据表中多维护一列数据
  + - 子类独有的字段不能添加非空约束
    - 若继承层次较深，则数据表的字段也会较多
  1. 使用joined-subclass元素继承映射
     1. 基本描述

• 采用 joined-subclass 元素的继承映射可以实现每个子类一张表

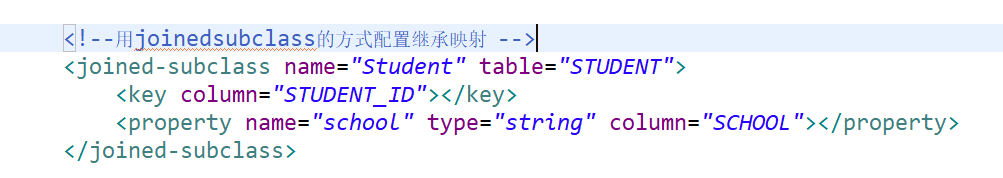
• 采用这种映射策略时，父类实例保存在父类表中，子类实例由父类表和子类表共同存储。因为子类实例也是一个特殊的父类实例，因此必然也包含了父类实例的属性。于是将子类和父类共有的属性保存在父类表中，子类增加的属性，则保存在子类表中。

• 在这种映射策略下，无须使用鉴别者列，但需要为每个子类使用 key 元素映射共有主键。

• 子类增加的属性可以添加非空约束。因为子类的属性和父类的属性没有保存在同一个表中



* + 1. 具体映射方法



* + 1. 增删改查方法的区别

Save方法：在插入子类对象时，会发送两条INSERT语句，因为，一个子类对象由两张表来维护，，，插入偏弱

查询方法：查询父类记录，做一个左外链接查询

查询子类记录，做一个内连接查询。。。。查询偏弱

* + 1. 优点

不需要使用辨别者列。子类字段能添加非空约束。没有冗余的字段

1. 采用 union-subclass 元素的继承映射
   1. 简单描述

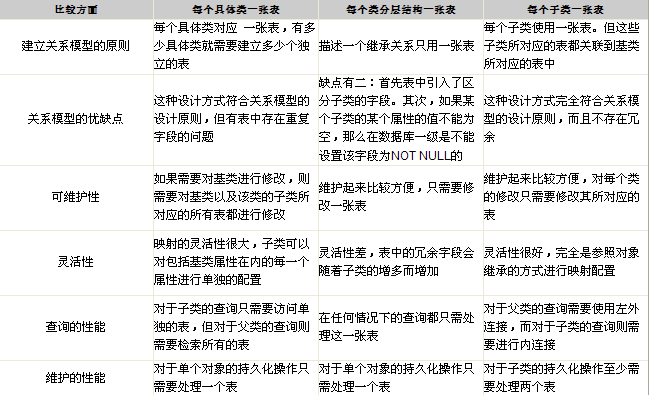
-采用 union-subclass 元素可以实现将每一个实体对象映射到一个独立的表中。

子类增加的属性可以有非空约束 --- 即父类实例的数据保存在父表中，而子类实例的数据保存在子类表中。

-子类实例的数据仅保存在子类表中, 而在父类表中没有任何记录

-在这种映射策略下，子类表的字段会比父类表的映射字段要多,因为子类表的字段等于父类表的字段、加子类增加属性的总和

-在这种映射策略下，既不需要使用鉴别者列，也无须使用 key 元素来映射共有主键.



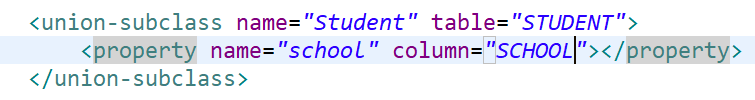
union-subclass

subclass

joined-subclass

-使用 union-subclass 映射策略是不可使用 identity 的主键生成策略, 因为同一类继承层次中所有实体类都需要使用同一个主键种子, 即多个持久化实体对应的记录的主键应该是连续的. 受此影响, 也不该使用 native 主键生成策略, 因为 native 会根据数据库来选择使用 identity 或 sequence.

* 1. 映射方法



* 1. 增删改查的变化

1. 注意，继承关系的映射在开发时并不是很常用，了解即可，在使用时一般使用union-subclass和joined-subclass的方式