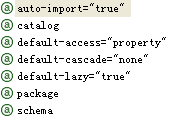
* POJO 类和关系数据库之间的映射可以用一个XML文档来定义。
* 通过 POJO 类的数据库映射文件，Hibernate可以理解持久化类和数据表之间的对应关系，也可以理解持久化类属性与数据库表列之间的对应关系
* 在运行时 Hibernate 将根据这个映射文件来生成各种 SQL 语句
* 映射文件的扩展名为 .hbm.xml
* hibernate-mapping
  + 类层次：class
    - 主键：id
    - 基本类型:property
    - 实体引用类: many-to-one | one-to-one
    - 集合:set | list | map | array
      * one-to-many
      * many-to-many
    - 子类:subclass | joined-subclass
    - 其它:component | any 等
  + 查询语句:query（用来放置查询语句，便于对数据库查询的统一管理和优化）
* 每个Hibernate-mapping中可以同时定义多个类. 但更推荐为每个类都创建一个单独的映射文件

1. hibernate-mapping节点
   1. hibernate-mapping街店里面的属性详情

* **hibernate-mapping 是 hibernate 映射文件的根元素**
  + **schema: 指定所映射的数据库schema的名称。若指定该属性, 则表明会自动添加该 schema 前缀**
  + **catalog:指定所映射的数据库catalog的名称。**
  + **default-cascade(默认为 none): 设置hibernate默认的级联风格. 若配置 Java 属性, 集合映射时没有指定 cascade 属性, 则 Hibernate 将采用此处指定的级联风格.**
  + **default-access (默认为 property): 指定 Hibernate 的默认的属性访问策略。默认值为 property, 即使用 getter, setter 方法来访问属性. 若指定 access, 则 Hibernate 会忽略 getter/setter 方法, 而通过反射访问成员变量.**
  + **default-lazy(默认为 true): 设置 Hibernat morning的延迟加载策略. 该属性的默认值为 true, 即启用延迟加载策略. 若配置 Java 属性映射, 集合映射时没有指定 lazy 属性, 则 Hibernate 将采用此处指定的延迟加载策略**
  + **auto-import (默认为 true): 指定是否可以在查询语言中使用非全限定的类名（仅限于本映射文件中的类）。**
  + **package (可选): 指定一个包前缀，如果在映射文档中没有指定全限定的类名， 就使用这个作为包名。**



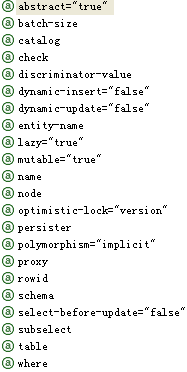
2 。hibernate-mapping下的class子节点

**class 元素用于指定类和表的映射**

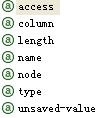
* + **name:指定该持久化类映射的持久化类的类名**
  + **table:指定该持久化类映射的表名, Hibernate 默认以持久化类的类名作为表名**
  + **dynamic-insert: 若设置为 true, 表示当保存一个对象时, 会动态生成 insert 语句, insert 语句中仅包含所有取值不为 null 的字段. 默认值为 false**
  + **dynamic-update: 若设置为 true, 表示当更新一个对象时, 会动态生成 update 语句, update 语句中仅包含所有取值需要更新的字段. 默认值为 false**

**说明：dynamic-insert和dynamic-update类似，这两个子节点设置为true的话，那么在执行更改某个属性的时候，就只会更改那一个属性，而不是所有的属性都进行更新**

* + **select-before-update:设置 Hibernate 在更新某个持久化对象之前是否需要先执行一次查询. 默认值为 false**
  + **batch-size:指定根据 OID 来抓取实例时每批抓取的实例数.**
  + **lazy: 指定是否使用延迟加载.**
  + **mutable: 若设置为 true, 等价于所有的 <property> 元素的 update 属性为 false, 表示整个实例不能被更新. 默认为 true.**
  + **discriminator-value: 指定区分不同子类的值. 当使用 <subclass/> 元素来定义持久化类的继承关系时需要使用该属性**



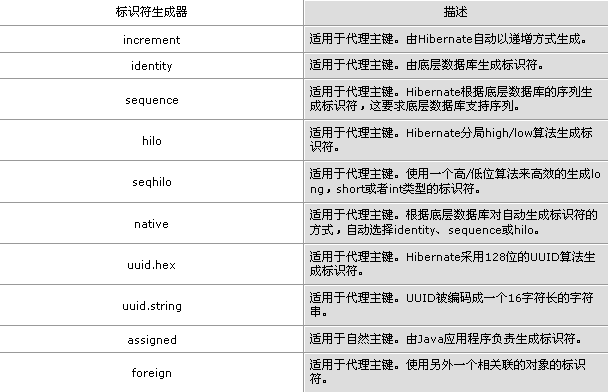
1. **id：设定持久化类的 OID 和表的主键的映射**
   1. **name: 标识持久化类 OID 的属性名**
   2. **column: 设置标识属性所映射的数据表的列名(主键字段的名字). 可以直接用<column>子节点来设置**
   3. **unsaved-value:若设定了该属性, Hibernate 会通过比较持久化类的 OID 值和该属性值来区分当前持久化类的对象是否为临时对象**
   4. **type:指定 Hibernate 映射类型. Hibernate 映射类型是 Java 类型与 SQL 类型的桥梁. 如果没有为某个属性显式设定映射类型, Hibernate 会运用反射机制先识别出持久化类的特定属性的 Java 类型, 然后自动使用与之对应的默认的 Hibernate 映射类型**
   5. **Java 的基本数据类型和包装类型对应相同的 Hibernate 映射类型. 基本数据类型无法表达 null, 所以对于持久化类的 OID 推荐使用包装类型**



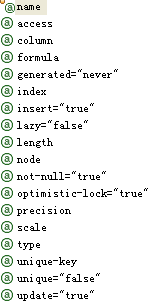
* 1. ．id下的子节点**generator**

**generator：设定持久化类设定标识符生成器**

**class: 指定使用的标识符生成器全限定类名或其缩写名**



* -increment，存在并发问题，一般只在测试阶段使用，不用于生产，increment是**由** Hibernate 以递增的方式为代理主键赋值。Hibernate 会先读取 NEWS 表中的主键的最大值, 而接下来向 NEWS 表中插入记录时, 就在 max(id) 的基础上递增, 增量为 1.。这个东西不分数据库都可以使用。
* identity 标识符生成器由底层数据库来负责生成标识符, 它要求底层数据库把主键定义为自动增长字段类型
* sequence 标识符生成器利用底层数据库提供的序列来生成标识符. Hibernate 在持久化一个 News 对象时, 先从底层数据库的 news\_seq 序列中获得一个唯一的标识号, 再把它作为主键值
* hilo 标识符生成器由 Hibernate 按照一种 high/low 算法\*生成标识符, 它从数据库的特定表的字段中获取 high 值.
* native 标识符生成器依据底层数据库对自动生成标识符的支持能力, 来选择使用 identity, sequence 或 hilo 标识符生成器.
* 4. **property 元素用于指定类的属性和表的字段的映射**
  + **name:指定该持久化类的属性的名字**
  + **column:指定与类的属性映射的表的字段名. 如果没有设置该属性, Hibernate 将直接使用类的属性名作为字段名.**
  + **type:指定 Hibernate 映射类型. Hibernate 映射类型是 Java 类型与 SQL 类型的桥梁. 如果没有为某个属性显式设定映射类型, Hibernate 会运用反射机制先识别出持久化类的特定属性的 Java 类型, 然后自动使用与之对应的默认的 Hibernate 映射类型.**
  + **not-null:若该属性值为 true, 表明不允许为 null, 默认为 false**
  + **access:指定 Hibernate 的默认的属性访问策略。默认值为 property, 即使用 getter, setter 方法来访问属性. 若指定 field, 则 Hibernate 会忽略 getter/setter 方法, 而通过反射访问成员变量**
  + **unique: 设置是否为该属性所映射的数据列添加唯一约束.**
  + **index: 指定一个字符串的索引名称. 当系统需要 Hibernate 自动建表时, 用于为该属性所映射的数据列创建索引, 从而加快该数据列的查询.**
  + **length: 指定该属性所映射数据列的字段的长度**
  + **scale: 指定该属性所映射数据列的小数位数, 对 double, float, decimal 等类型的数据列有效.**
  + **formula：设置一个 SQL 表达式, Hibernate 将根据它来计算出派生属性的值.**
  + **派生属性: 并不是持久化类的所有属性都直接和表的字段匹配, 持久化类的有些属性的值必须在运行时通过计算才能得出来, 这种属性称为派生属性**
  + **使用 formula 属性时**
  + **formula=“(sql)” 的英文括号不能少**
  + **Sql 表达式中的列名和表名都应该和数据库对应, 而不是和持久化对象的属性对应**
  + **如果需要在 formula 属性中使用参数, 这直接使用 where cur.id=id 形式, 其中 id 就是参数, 和当前持久化对象的 id 属性对应的列的 id 值将作为参数传入.**



5. 映射 Java 的时间, 日期类型

1). 两个基础知识:

I. 在 Java 中, 代表时间和日期的类型包括: java.util.Date 和 java.util.Calendar.

此外, 在 JDBC API 中还提供了 3 个扩展了 java.util.Date 类的子类: java.sql.Date, java.sql.Time

和 java.sql.Timestamp, 这三个类分别和标准 SQL 类型中的 DATE, TIME 和 TIMESTAMP 类型对应

II. 在标准 SQL 中, DATE 类型表示日期, TIME 类型表示时间, TIMESTAMP 类型表示时间戳, 同时包含日期和时间信息.

2). 如何进行映射 ?

I. 因为 java.util.Date 是 java.sql.Date, java.sql.Time 和 java.sql.Timestamp 的父类, 所以 java.util.Date

可以对应标准 SQL 类型中的 DATE, TIME 和 TIMESTAMP

II. 基于 I, 所以在设置持久化类的 Date 类型是, 设置为 java.util.Date.

III. 如何把 java.util.Date 映射为 DATE, TIME 和 TIMESTAMP ?

可以通过 property 的 type 属性来进行映射:

例如:

<property name="date" type="timestamp">

<column name="DATE" />

</property>

<property name="date" type="data">

<column name="DATE" />

</property>

<property name="date" type="time">

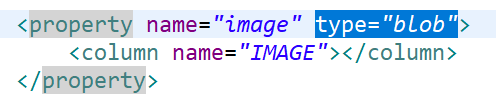
<column name="DATE" />

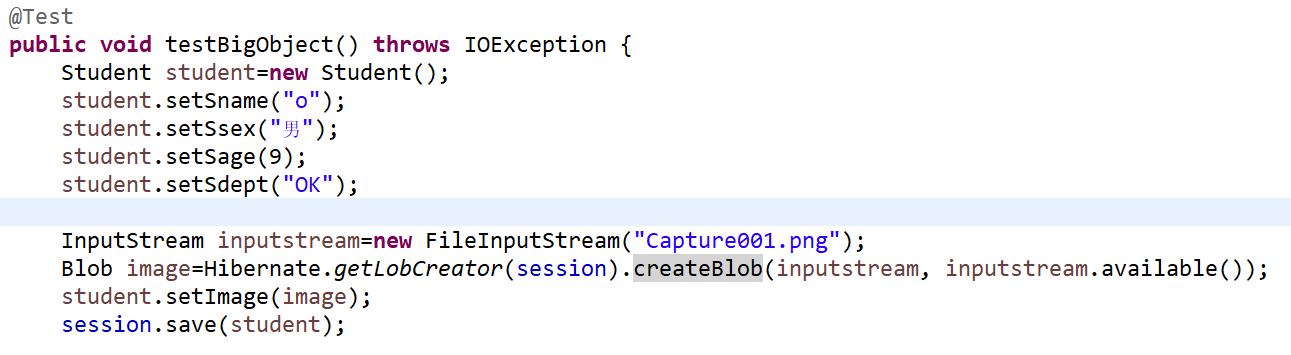
</property>

其中 timestamp, date, time 既不是 Java 类型, 也不是标准 SQL 类型, 而是 hibernate 映射类型.

6. Java 大对象类型的 Hiberante 映射

* 在 Java 中, java.lang.String 可用于表示**长字符串(长度超过 255),** 字节数组 byte[] 可用于存放图片或文件的**二进制数据**. 此外, 在 JDBC API 中还提供了 java.sql.Clob 和 java.sql.Blob 类型, 它们分别和标准 SQL 中的 CLOB 和 BLOB 类型对应. CLOB 表示字符串大对象(Character Large Object), BLOB表示二进制对象(Binary Large Object)





开发时并不直接存入图片等，一般是将图片等存到文件夹，在数据库存入路径

7. 映射组成关系

* Hibernate 把持久化类的属性分为两种:
  + 值(value)类型: **没有 OID**, **不能被单独持久化**, **生命周期依赖于所属的持久化类的对象的生命周期**
  + 实体(entity)类型: 有 OID, 可以被单独持久化, 有独立的生命周期
* 显然无法直接用 property 映射 pay 属性
* Hibernate 使用 <component> 元素来映射组成关系, 该元素表名 pay 属性是 Worker 类一个组成部分, 在 Hibernate 中称之为**组件**

