# Hibernate学习

## 一、\* 基础

### 1、什么是Hibernate

一个框架

一个Java领域的**持久化**框架

一个**ORM**框架

### 2、对象的持久化

1）狭义的理解，“持久化”仅仅指把对象永久保存到数据库中。

2）广义的理解，“持久化”包括和数据库相关的各种操作：

保存：把对象永久保存到数据库中。

更新：更新数据库中对象（记录）的状态。

删除：从数据库中删除一个对象。

查询：根据特定的查询条件，把符合查询条件的一个或多个对象从数据库加载到内存中。

**加载**：根据特定的**OID**，把一个对象从数据库加载到内存中。

（为了在系统中能够找到所需对象，需要为每一个对象分配一个唯一的标识号。在关系数据库中称之为主键，而在对象术语中，则叫做对象标识(Object identifier-OID)）

### 3、ORM

**ORM(Object/Relation Mapping)：对象/关系映射**

1）ORM主要解决对象-关系的映射：



2）ORM的思想：将关系数据库中表中的记录映射成为对象，以对象的形式展现，**程序员可以把对数据库的操作转化为对对象的操作**。

3）ORM采用**元数据**来描述对象-关系映射细节，元数据通常采用XML格式，并且存放在专门的对象-关系映射文件中。

4）ORM架构图：



5）流行的ORM框架：

**（1）Hibernate：**

非常优秀、成熟的ORM框架。

完成对象的持久化操作。

Hibernate允许开发者**采用面向对象的方式来操作关系数据库**。

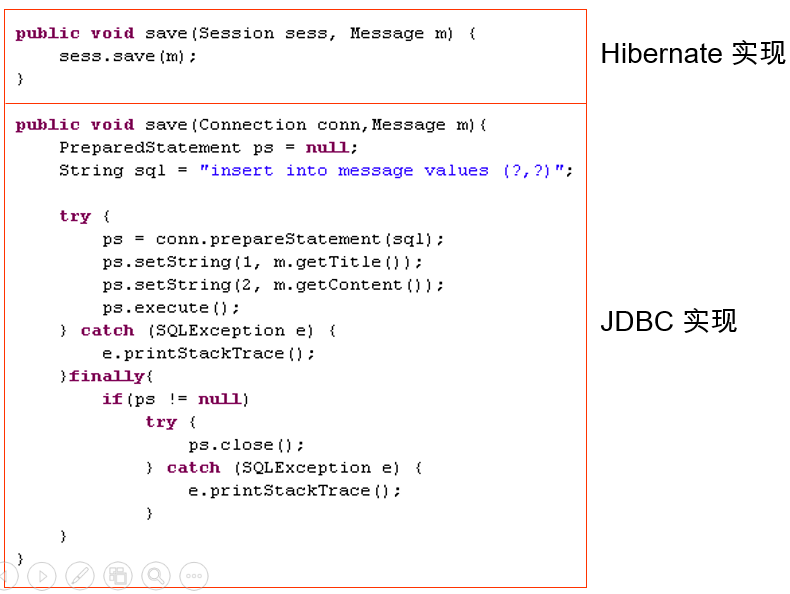
消除那些针对特定数据库厂商的SQL代码。

**（2）myBatis：**

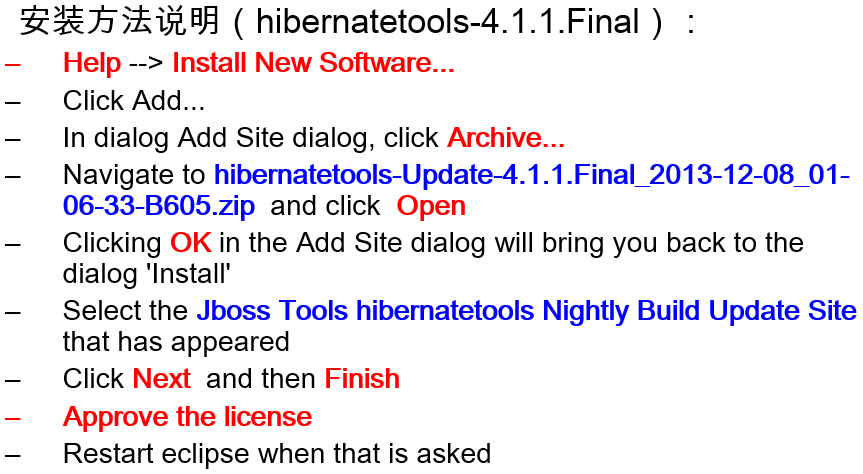
相比Hibernate灵活高，运行速度快。

开发速度慢，不支持纯粹的面向对象操作，需熟悉sql语句，并且熟练使用sql语句优化功能。

### 4、Hibernate与Jdbc代码对比



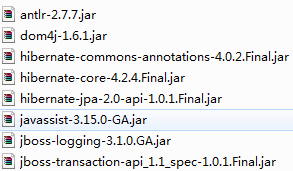
### 5、安装Hibernate插件



### 6、准备Hibernate环境

**1）导入Hibernate必须的jar包：**

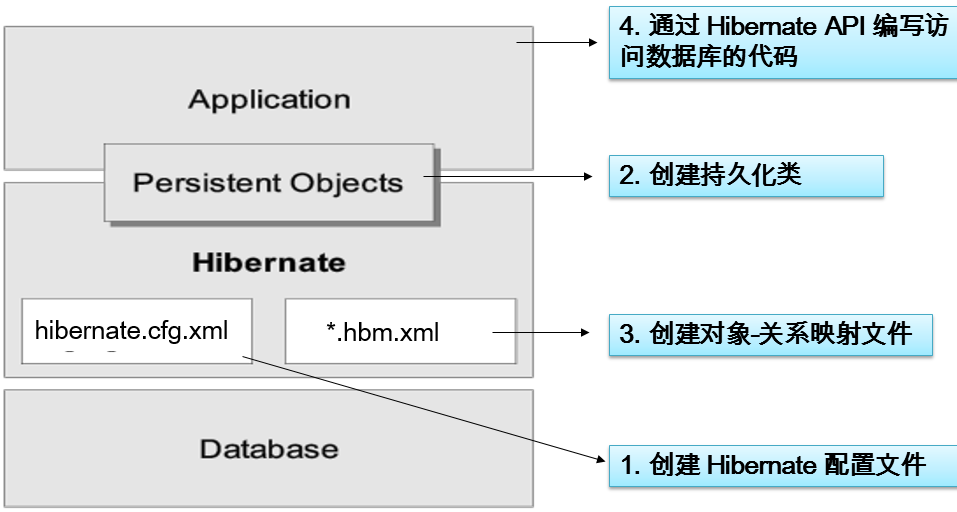
**路径：hibernate-release-4.2.4.Final\lib\required**



**2）加入数据库驱动的jar包：**



### 7、Hibernate开发步骤



#### 1）创建Hibernate配置文件

Hibernate从其配置文件（hibernate.cfg.xml）中读取和数据库连接的有关信息，这个文件应该位于应用的classpath下（src目录）。



#### 2）创建持久化Java类

（1）**提供一个无参的构造器**：使Hibernate可以使用Constructor.newInstance() 来实例化持久化类。

（2）**提供一个标识属性（identifier property）**：通常映射为数据库表的主键字段，如果没有该属性，一些功能将不起作用，如：Session.saveOrUpdate()。

（3）**为类的持久化类字段声明访问方法（get/set）**：Hibernate对JavaBeans风格的属性实行持久化。

（4）**使用非final类**：在运行时生成代理是Hibernate的一个重要的功能，如果持久化类没有实现任何接口，Hibnernate使用CGLIB生成代理；如果使用的是final类，则无法生成CGLIB代理。

（5）**重写eqauls和hashCode方法**：如果需要把持久化类的实例放到Set中（当需要进行关联映射时），则应该重写这两个方法。

（6）Hibernate不要求持久化类继承任何父类或实现接口，这可以保证代码不被污染。这就是Hibernate被称为低侵入式设计的原因。

#### 3）创建对象-关系映射文件

Hibernate采用XML格式的文件来指定对象和关系数据之间的映射。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



#### 4）通过Hibernate API编写访问数据库的代码

使用Hibernate进行数据持久化操作，通常有如下步骤：

编写持久化类：POJO + 映射文件

获取Configuration对象

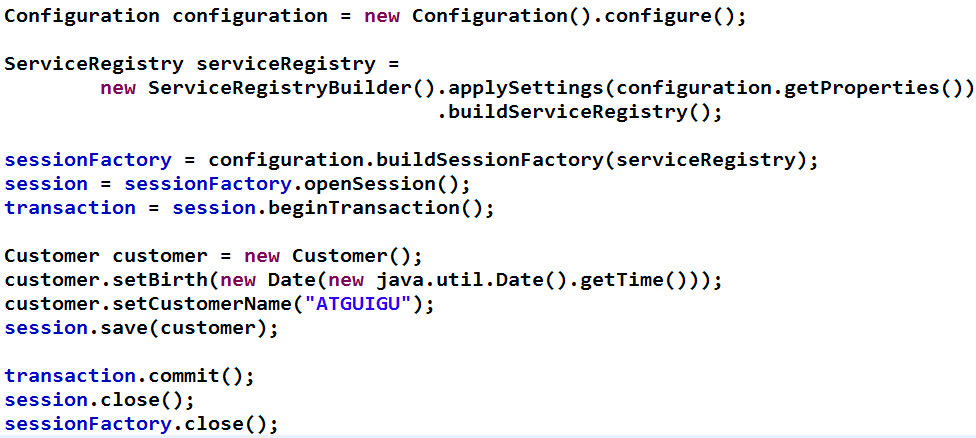
获取SessionFactory对象

获取Session，打开事务

用面向对象的方式操作数据库

关闭事务，关闭Session

**测试代码：**



**控制台输出的SQL语句：**



### 8、Hibernate API介绍

#### 1）Configuration类

（1）Configuration类负责管理Hibernate的配置信息。包括如下内容：

① Hibernate运行的底层信息：数据库的URL、用户名、密码、JDBC驱动类，数据库Dialect，数据库连接池等（对应**hibernate.cfg.xml**文件）。

② 持久化类与数据表的映射关系（**\*.hbm.xml**文件）。

（2）创建Configuration的两种方式：

**① xml文件（hibernate.cfg.xml）：**

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

② 属性文件（hibernate.properties）：

Configuration cfg = new Configuration();

③ Configuration的configure方法还支持带参数的访问：

File file = new File(“simpleit.xml”);

Configuration cfg = new Configuration().configure(file);

#### 2）SessionFactory接口

（1）针对单个数据库映射关系经过编译后的内存镜像，是线程安全的。

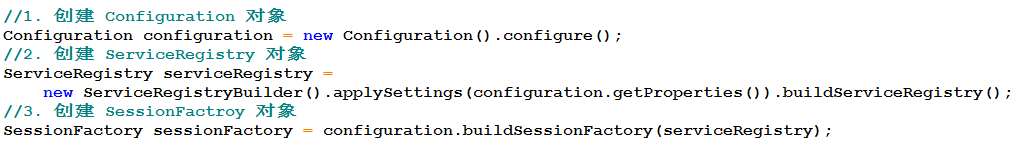
（2）SessionFactory对象一旦构造完毕，即被赋予特定的配置信息。

（3）SessionFactory是生成Session的工厂.

（4）构造SessionFactory很消耗资源，一般情况下一个应用中只初始化一个SessionFactory对象。

（5）Hibernate4新增了一个ServiceRegistry接口，所有基于Hibernate的配置或者服务都必须统一向这个ServiceRegistry注册后才能生效。

（5）Hibernate4中创建SessionFactory的步骤：



#### 3）Session接口

（1）Session是应用程序与数据库之间交互操作的一个**单线程对象**，是Hibernate运作的中心，所有持久化对象必须在session的管理下才可以进行持久化操作。此对象的生命周期很短。Session对象有一个一级缓存，显式执行flush之前，所有的持久层操作的数据都缓存在 session对象处。**相当于JDBC中的 Connection**。

（2）持久化类与Session关联起来后就具有了持久化的能力。

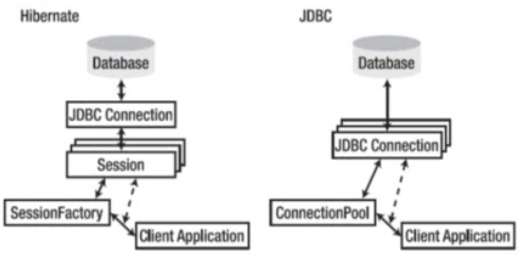
（3）Session类的方法：

取得持久化对象的方法：get()、load()

持久化对象都得保存，更新和删除：save()、update()、saveOrUpdate()、delete()

开启事务：beginTransaction()

管理Session的方法：isOpen()、flush()、clear()、evict()、close()等



### 9、Hibernate配置文件的两个配置项

#### 1）hbm2ddl.auto

该属性可帮助程序员实现正向工程，即由java代码生成数据库脚本，进而生成具体的表结构。取值：create | update | create-drop | validate

**create**：会根据 .hbm.xml文件来生成数据表，但是每次运行都会删除上一次的表，重新生成表，哪怕二次没有任何改变。

**create-drop**：会根据 .hbm.xml文件生成表，但是SessionFactory一关闭，表就自动删除。

**update**：**最常用的属性值**，也会根据 .hbm.xml文件生成表，但若 .hbm.xml文件和数据库中对应的数据表的表结构不同，Hiberante将更新数据表结构，但不会删除已有的行和列。

**validate**：会和数据库中的表进行比较，若 .hbm.xml文件中的列在数据表中不存在，则抛出异常。

#### 2）format\_sql

是否将SQL转化为格式良好的SQL，取值：true | false。

## 二、Session

### 1、Session概述

1）Session接口是Hibernate向应用程序提供的操纵数据库的**最主要**的接口，它**提供了基本的保存、更新、删除和加载Java对象的方法**。

2）**Session具有一个缓存，位于缓存中的对象称为持久化对象，它和数据库中的相关记录对应**。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，这一过程被称为刷新缓存（flush）。

3）**站在持久化的角度，Hibernate把对象分为4种状态**：持久化状态，临时状态，游离状态，删除状态。Session的特定方法能使对象从一个状态转换到另一个状态。

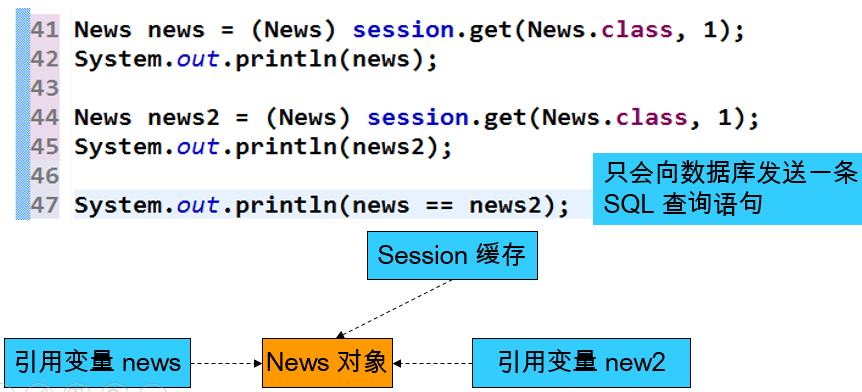
### 2、Session缓存

#### 1）概述

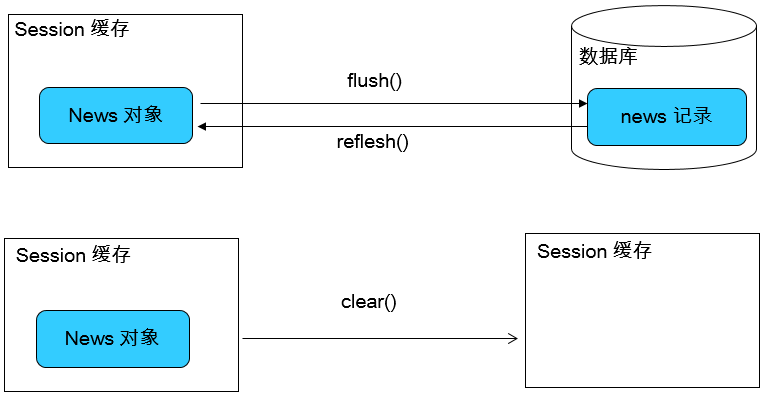
（1）在Session接口的实现中包含一系列的Java集合，这些Java集合构成了Session缓存。只要Session实例没有结束生命周期，且没有清理缓存，则存放在它缓存中的对象也不会结束生命周期。

（1）Session缓存可减少Hibernate应用程序访问数据库的频率。

#### 2）示例



#### 3）\* 操作Session缓存



##### （1）flush()

（1）flush：Session按照缓存中对象的属性变化来同步更新数据库。

（2）默认情况下，Session在以下时间点刷新缓存：

① 显式调用**Session的flush()** 方法。

② 当应用程序调用**Transaction的commit()** 方法的时，该方法先flush，然后在向数据库提交事务。

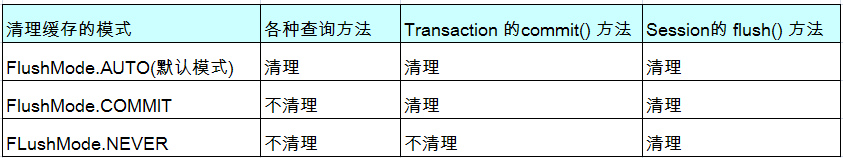
③ 当应用程序执行一些查询（HQL，Criteria）操作时，如果缓存中持久化对象的属性已经发生了变化，会先flush缓存，以保证查询结果能够反映持久化对象的最新状态。

（3）**flush缓存的例外情况**：如果对象使用native生成器生成OID，那么当调用Session的save() 方法保存对象时，会立即执行向数据库插入该实体的insert语句，因为save方法后必须保证对象的ID是存在的。

（4）commit() 和flush() 方法的区别：flush执行一系列sql语句，但不提交事务；commit方法先调用flush() 方法，然后提交事务，意味着对数据库操作永久保存下来。

（5）设定刷新缓存的时间点

若希望改变flush的默认时间点，可以通过Session的setFlushMode() 方法显式设定flush的时间点：



##### （2）reflush()

会强制发送SELECT语句，以使Session缓存中对象的状态和数据表中对应的记录保持一致。在这里要注意数据库的隔离级别问题。

##### （3）clear()

清理缓存。

### 3、持久化对象的状态

站在持久化的角度，Hibernate把对象分为4种状态：**持久化状态、临时状态、游离状态、删除状态**。Session的特定方法能使对象从一个状态转换到另一个状态。

#### 1）临时对象（Transient）

在使用代理主键的情况下，OID通常为null

不处于Session的缓存中

在数据库中没有对应的记录

#### 2）持久化对象(也叫”托管”)（Persist）

OID不为null

位于Session缓存中

若在数据库中已经有和其对应的记录，持久化对象和数据库中的相关记录对应

Session在flush缓存时，会根据持久化对象的属性变化，来同步更新数据库

在同一个Session实例的缓存中，数据库表中的每条记录只对应唯一的持久化对象

#### 3）删除对象（Removed）

在数据库中没有和其OID对应的记录

不再处于Session缓存中

一般情况下，应用程序不该再使用被删除的对象

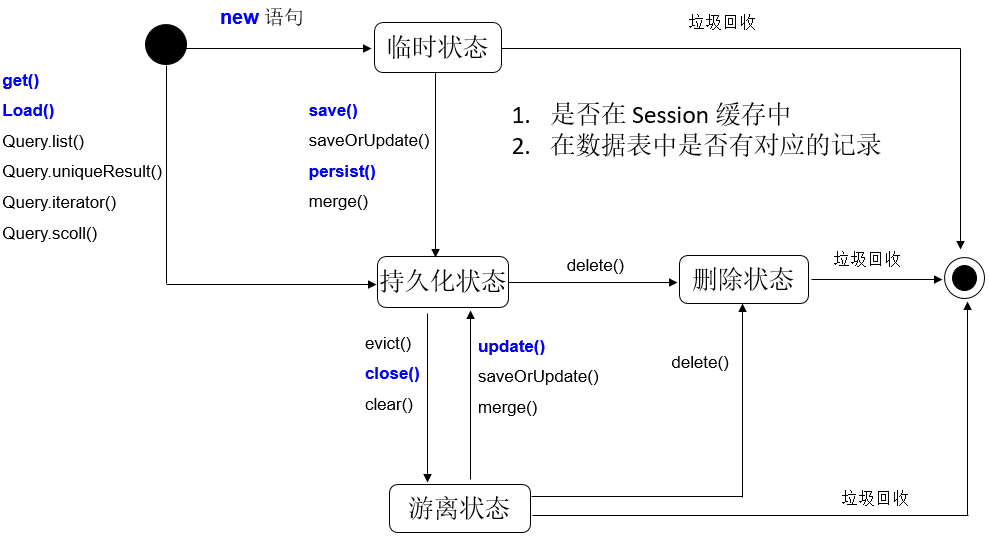
#### 4）游离对象(也叫”脱管”)（Detached）

OID不为null

不再处于Session缓存中

一般情况需下，游离对象是由持久化对象转变过来的，因此在数据库中可能还存在与它对应的记录

#### 5）\* 对象的状态转换



### 4、Session的常用方法

#### 1）save()

（1）Session的save() 方法使一个临时对象转变为持久化对象。

（2）Session的save() 方法完成以下操作：

① **把News对象加入到Session缓存中，使它进入持久化状态**。

② **选用映射文件指定的标识符生成器，为持久化对象分配唯一的OID**。在使用代理主键的情况下，setId() 方法为News对象设置OID使无效的。

③ **在flush缓存的时候，执行一条insert语句**。

（3）Hibernate通过持久化对象的OID来维持它和数据库相关记录的对应关系。**当News对象处于持久化状态时，不允许程序随意修改它的ID**。

（4）**persist() 和save() 区别：**

当对一个OID不为Null的对象执行save() 方法时，会把该对象以一个新生成的OID保存到数据库中，忽略对象原有的OID；但执行persist() 方法时会抛出一个异常。

#### 2）get() 和load()

（1）都可以根据指定的OID从数据库中加载一个持久化对象。

（2）区别:

① 执行get方法，会立即加载对象；执行load方法，若不使用该对象，则不会立即执行查询操作，而返回一个代理对象，**load方法支持延迟加载策略**。

② 如果数据库中不存在与OID对应的记录：

get() 方法返回null；执行load() 方法，若不使用该对象的任何属性，也就是还没有执行查询操作时，不会出现问题，但是一旦需要执行查询操作了，就会抛出ObjectNotFoundException异常。

③ load() 方法在真正执行查询操作之前关闭了Session，那么当执行查询操作时会抛出LazyInitializationException异常。

#### 3）update()

（1）若更新一个持久化对象，不需要显示的调用update() 方法，因为在调用Transaction的 commit() 方法时，会先执行session的flush() 方法。

（2）更新一个游离对象，需要显式的调用session的update() 方法，把该游离对象变为持久化对象。

（3）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。

如何能让updat方法不再盲目的出发update语句呢？

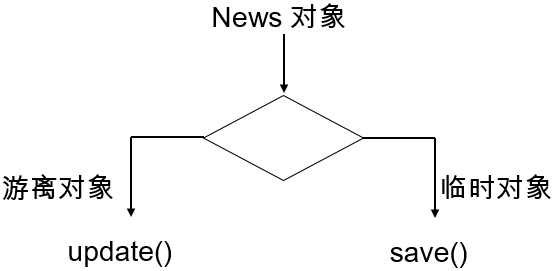
在 .hbm.xml文件的class节点设置**select-before-update=true**（默认为 false），但**通常不要设置该属性**，因为这样设置在每次更新之前都会发送一天select语句，效率反而变低了。

② 若数据表中没有游离对象对应的记录，但还调用了update() 方法，会抛出异常。

③ 当update() 方法关联一个游离对象时，如果在Session的缓存中已经存在相同OID的持久化对象，会抛出异常，因为在Session缓存中不能有两个OID相同的对象。

#### 4）saveOrUpdate()

（1）Session的saveOrUpdate() 方法同时包含了save() 与update() 方法的功能。



（2）判定对象为临时对象的标准：

① **Java对象的OID为null**

② （一般不用）映射文件中为 <id> 设置了unsaved-value属性，并且Java对象的OID取值与这个unsaved-value属性值匹配。

（3）注意：若OID不为null且不等于unsaved-value属性值，但数据表中还没有和其对应的记录，会抛出一个异常。

#### 5）delete()

（1）Session的delete() 方法既可以删除一个游离对象，也可以删除一个持久化对象，**只要对象OID和数据表中一条记录对应，就会准备执行delete操作，若OID在数据表中没有对应的记录，则抛出异常**。

（2）Session的delete() 方法处理过程：

① 计划执行一条delete语句。

② 把对象从Session缓存中删除，该对象进入删除状态。

（3）（一般不用）可以通过设置Hibernate配置文件的hibernate.use\_identifier\_rollback属性值为true（默认为false），使删除对象后，把该对象OID置为null：

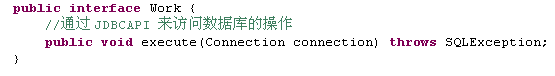


#### 6）evict()

从session缓存中把指定的持久化对象移除。

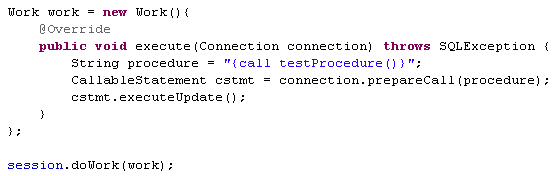
#### 7）通过Hibernate调用存储过程

（1）Work接口：直接通过JDBC API来访问数据库的操作。



（2）Session的doWork(Work) 方法用于执行Work对象指定的操作，即调用Work对象的execute() 方法。Session会把当前使用的数据库连接传递给execute() 方法。

**（3）示例：**



#### 8）Hibernate与触发器协同工作

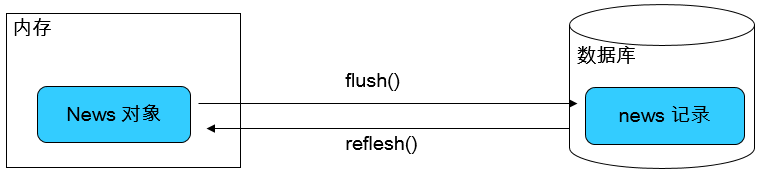
（1）Hibernate与数据库中的触发器协同工作时，会造成两类问题：

① 触发器使Session的缓存中的持久化对象与数据库中对应的数据不一致：触发器运行在数据库中，它执行的操作对Session是透明的。

② Session的update() 方法盲目地激发触发器：无论游离对象的属性是否发生变化，都会执行update语句，而update语句会激发数据库中相应的触发器。

（2）解决方案：

① 在执行完Session的相关操作后，立即调用Session的flush() 和refresh() 方法，迫使Session的缓存与数据库同步（refresh() 方法重新从数据库中加载对象）。



② 在映射文件的的 <class> 元素中设置 select-before-update 属性：当Session的update() 或saveOrUpdate() 方法更新一个游离对象时，会先执行Select语句，获得当前游离对象在数据库中的最新数据，只有在不一致的情况下才会执行update语句。

#### 9）\* 结论

（1）对象的OID是自动生成的，唯一的，并且应该是固定不变的，所以无论对象处于什么状态最好都不要改变对象的OID。

（2）一般情况下，使用save()、get() 方法即可。

（3）更新操作一般先查询出来再对查出的对象进行修改，所以update() 方法一般不用显示调用。

## 三、对象关系映射文件

### 1、概述

1）POJO类和关系数据库之间的映射可以用一个XML文档来定义。

2）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。

3）在运行时Hibernate将根据这个映射文件来生成各种SQL语句。

4）映射文件的扩展名为：.hbm.xml。

### 2、映射文件说明

#### 1）概述

（1）hibernate-mapping

① 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

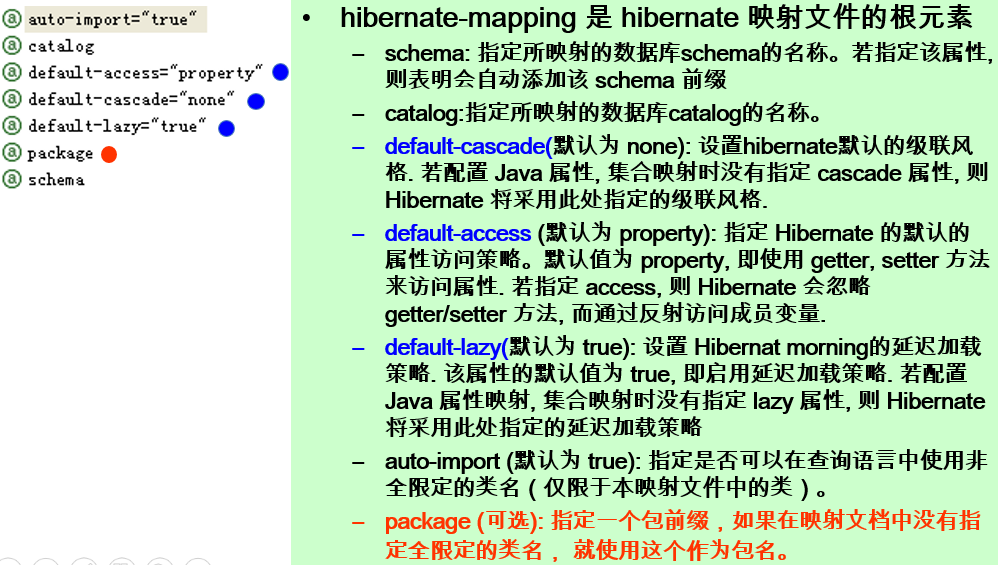
子类：subclass | joined-subclass

其它：component | any 等

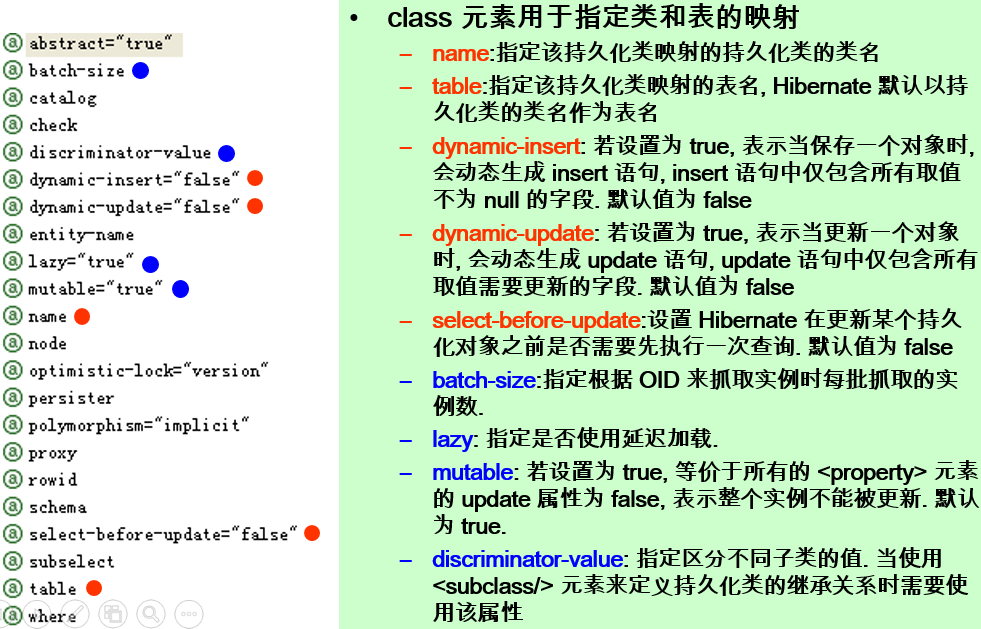
② 查询语句：query（用来放置查询语句，便于对数据库查询的统一管理和优化）

（2）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**。

#### 2）hibernate-mapping元素



#### 3）class元素



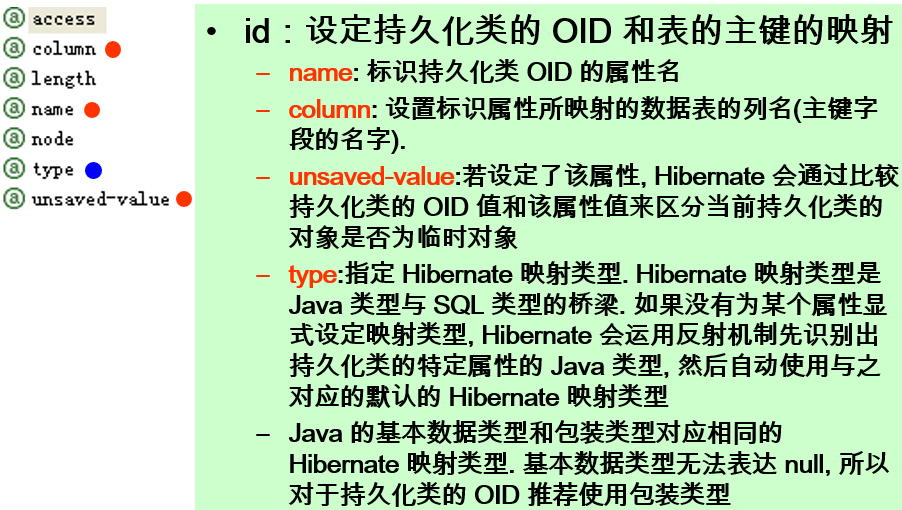
#### 4）id元素

（1）Hibernate使用对象标识符（OID），来建立内存中的对象和数据库表中记录的对应关系。对象的OID和数据表的主键对应。Hibernate通过标识符生成器来为主键赋值。

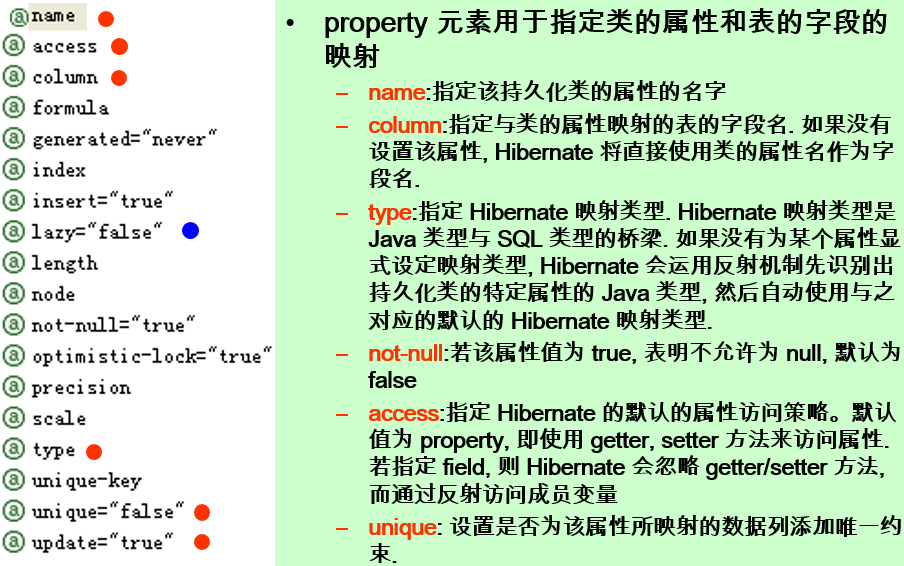
（2）Hibernate推荐在数据表中使用代理主键，即不具备业务含义的字段。代理主键通常为整数类型，因为整数类型比字符串类型要节省更多的数据库空间。

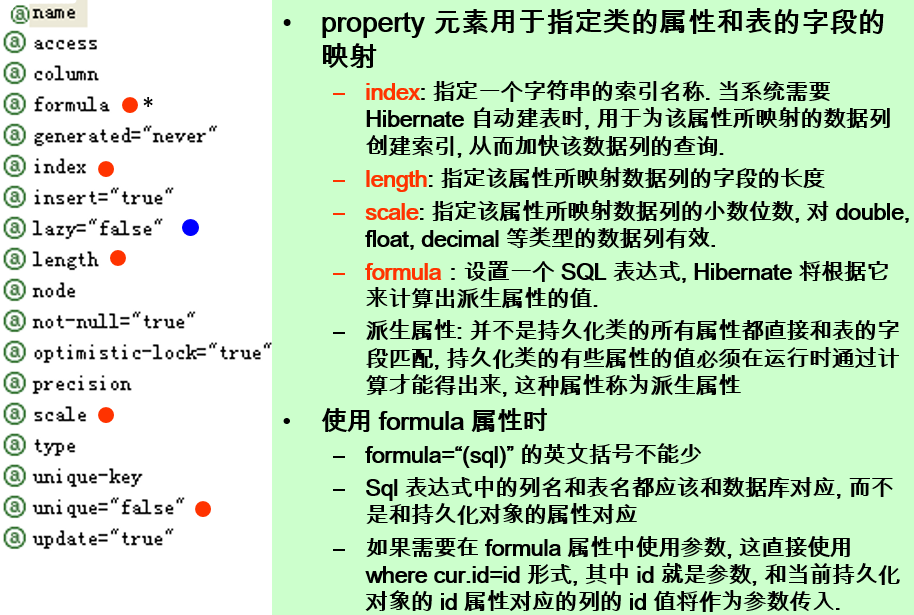
（3）在对象-关系映射文件中，<id> 元素用来设置对象标识符。<generator> 子元素用来设定标识符生成器。

（4）Hibernate提供了标识符生成器接口：IdentifierGenerator，并提供了各种内置实现。

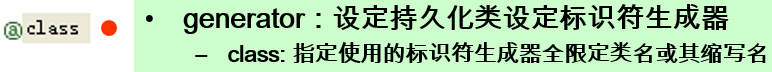


#### 5）property元素



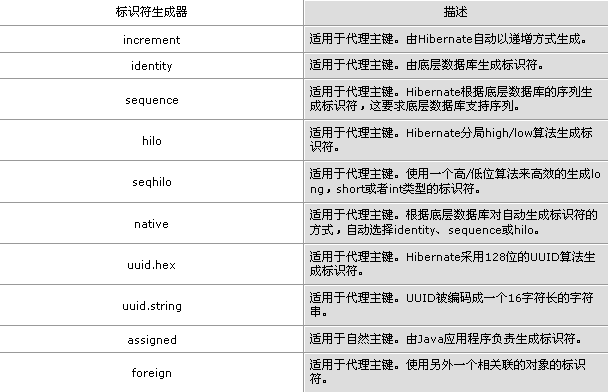


#### 6）generator元素



### 3、主键生成策略generator

Hibernate提供的内置标识符生成器：



#### 1）increment标识符生成器

increment标识符生成器**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。

适用范围：

① 由于increment生存标识符机制不依赖于底层数据库系统，因此它适合所有的数据库系统。

② 适用于只有单个Hibernate应用进程访问同一个数据库的场合，在集群环境下不推荐使用它，会发生并发问题，所以**只适合于测试，不能用于开发环境**。

③ OID必须为long、int或short类型，如果把OID定义为byte类型，在运行时会抛出异常。

#### 2）identity标识符生成器

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。

适用范围：

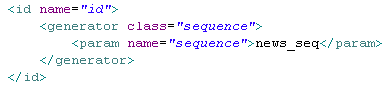
① 由于identity生成标识符的机制依赖于底层数据库系统，因此，**要求底层数据库系统必须支持自动增长字段类型。支持自动增长字段类型的数据库包括：DB2、Mysql、MSSQLServer、Sybase等**。

② OID必须为long、int或short类型，如果把OID定义为byte类型，在运行时会抛出异常。

#### 3）sequence 标识符生成器

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**。

**示例：**



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。

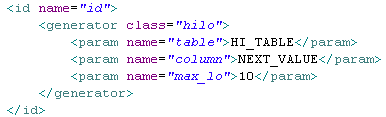
适用范围：

① 由于sequence生成标识符的机制依赖于底层数据库系统的序列，因此，**要求底层数据库系统必须支持序列。支持序列的数据库包括：DB2、Oracle等**。

② OID必须为long、int或 hort类型，如果把OID定义为byte类型，在运行时会抛出异常。

#### 4）hilo标识符生成器

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。



Hibernate在持久化一个News对象时，由Hibernate负责生成主键值。**hilo标识符生成器在生成标识符时，需要读取并修改HI\_TABLE表中的NEXT\_VALUE值**。

适用范围：

① 由于hilo生存标识符机制**不依赖于底层数据库系统，因此它适合所有的数据库系统**。

② OID必须为long、int或short类型，如果把OID定义为byte类型，在运行时会抛出异常。

#### 5）\* native标识符生成器

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**。

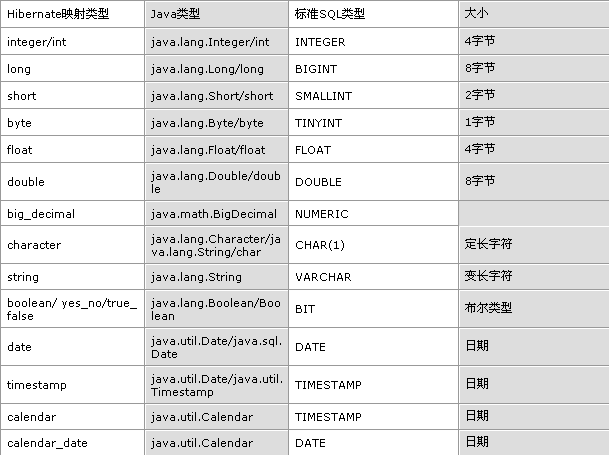
适用范围：

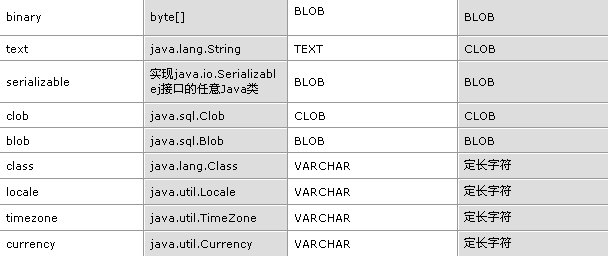
① 由于native能根据底层数据库系统的类型，自动选择合适的标识符生成器，因此**很适合于跨数据库平台开发**。

② OID必须为long、int或short类型，如果把OID定义为byte类型，在运行时会抛出异常。

### 4、Java类型、Hibernate映射类型及SQL类型之间的对应关系

#### 1）映射表

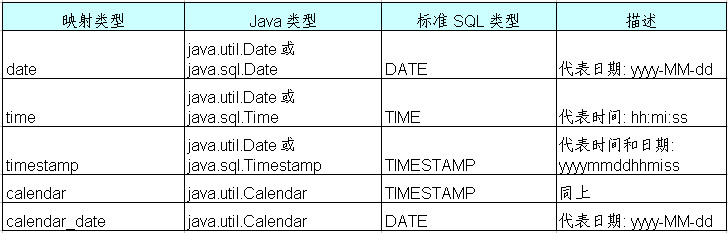




#### 2）Java时间和日期类型的Hibernate映射

（1）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。

（2）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。



##### \* 注意：

① 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。

② 基于①所以在**设置持久化类的Date类型时，建议设置为java.util.Date**。

③ 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：**

**<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>**

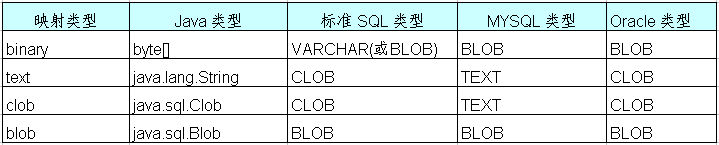
其中timestamp，date，time是Hibernate映射类型。

④ 以下情况下必须显式指定Hibernate映射类型：

一个Java类型可能对应多个Hibernate映射类型，例如：如果持久化类的属性为java.util.Date类型，对应的Hibernate映射类型可以是date，time或timestamp，此时必须根据对应的数据表的字段的SQL类型，来确定Hibernate映射类型，如果字段为DATE类型，那么Hibernate映射类型为date；如果字段为TIME类型，那么Hibernate映射类型为time；如果字段为TIMESTATMP类型，那么Hibernate映射类型为timestamp。

#### 3）Java大对象类型的Hiberante映射

（1）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）。



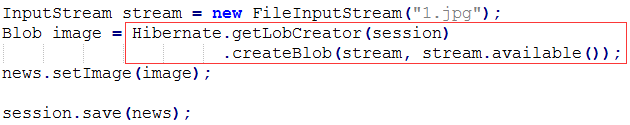
（2）Mysql不支持标准SQL的CLOB类型，在Mysql中，用TEXT、MEDIUMTEXT及LONGTEXT类型来表示长度操作255的长文本数据。

**（3）在持久化类中，二进制大对象可以声明为byte[] 或java.sql.Blob类型；字符串可以声明为java.lang.String或java.sql.Clob。**

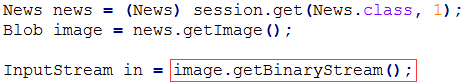
**（4）实际上在Java应用程序中处理长度超过255的字符串，使用java.lang.String比java.sql.Clob更方便。**

**示例：**

存入blob对象：



获取blob对象：

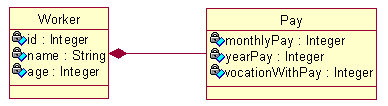


### 5、映射组成关系

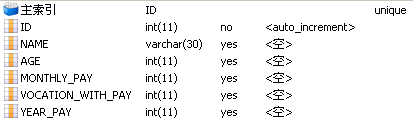
#### 1）域模型和关系数据模型

**建立域模型和关系数据模型有着不同的出发点：**

（1）域模型：由程序代码组成，通过细化持久化类的的粒度可提高代码的可重用性，简化编程。



（2）在没有数据冗余的情况下，应该尽可能减少表的数目，简化表之间的参照关系，以便提高数据的访问速度。



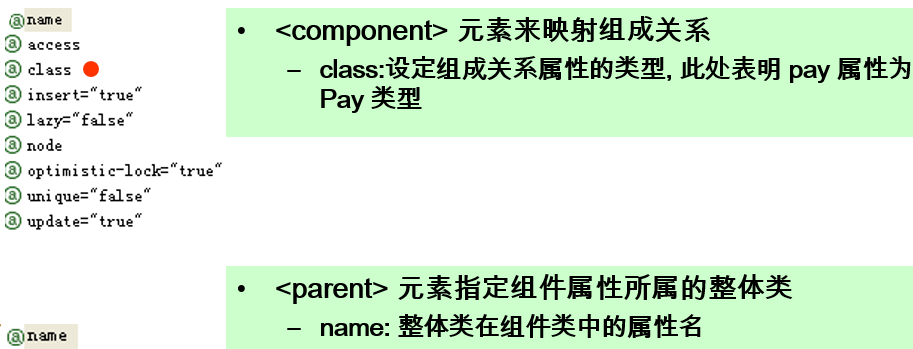
#### 2）Hibernate把持久化类的属性分为两种

（1）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，也就是作为实体类型的一个属性存在。

（2）实体（entity）类型：有OID，可以被单独持久化，有独立的生命周期。

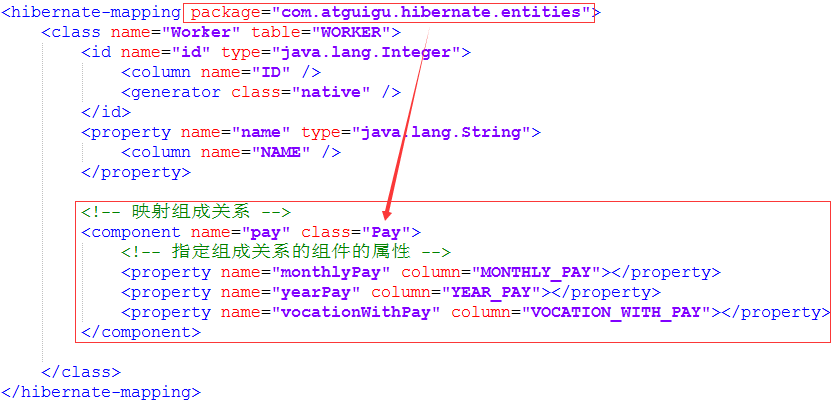
#### 3）component元素

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。



**注意：只有当组件类中包含一个整体类时才能使用parent元素，也就是说当Pay这个类中包含一个Worker对象时才能使用parent元素。**

**示例：**



## 四、映射关联关系

### 1、映射一对多关联关系

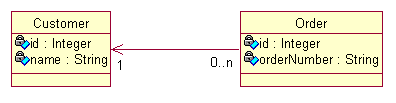
#### 1）概述

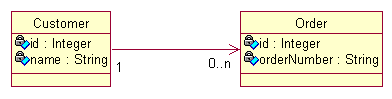
（1）在领域模型中，类与类之间最普遍的关系就是关联关系。

（2）在UML中，关联是有方向的：

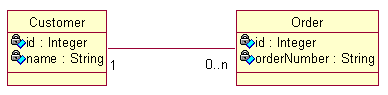
① 以Customer和Order为例：一个用户能发出多个订单，而一个订单只能属于一个客户。从Order到Customer的关联是多对一关联；而从Customer到Order是一对多关联。

② 单向关联：





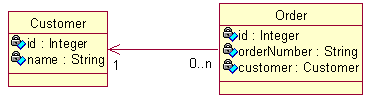
③ 双向关联：



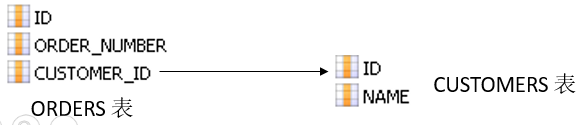
#### 2）单向n-1

（1）单向n-1关联只需从n的一端可以访问1的一端。

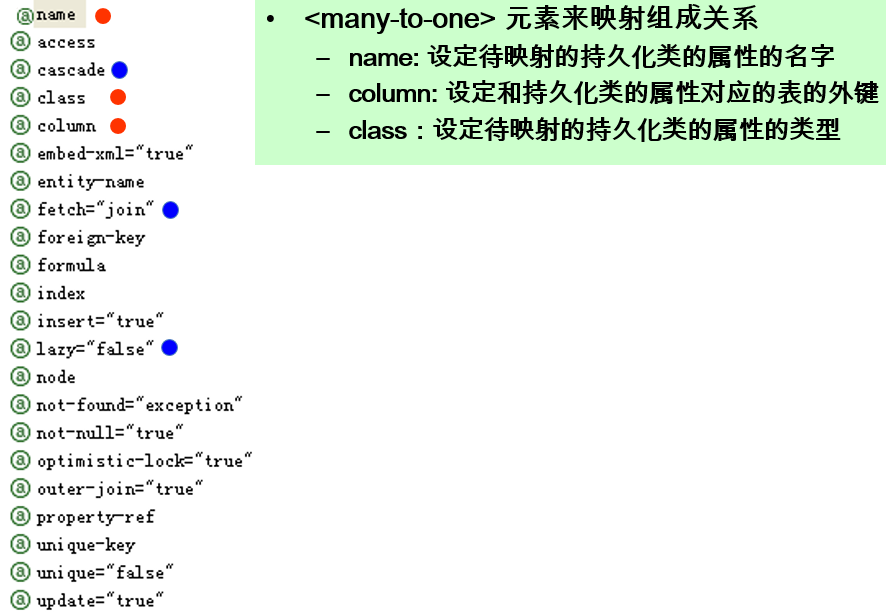
（2）域模型：从Order到Customer的多对一单向关联需要在Order 类中定义一个 Customer属性，而在Customer类中无需定义存放Order对象的集合属性。



（3）关系数据模型：ORDERS表中的CUSTOMER\_ID参照CUSTOMER表的主键。



（4）显然无法直接用property映射customer属性，**Hibernate使用 <many-to-one> 元素来映射多对一关联关系**。



**示例：**



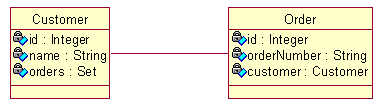
（5）在使用get方法查询Order时，对于它的属性customer来说，会使用延迟加载技术，首先返回一个代理对象，只有当使用到customer对象时，才会发送对应的sql语句查询该对象。

#### 3）双向1-n

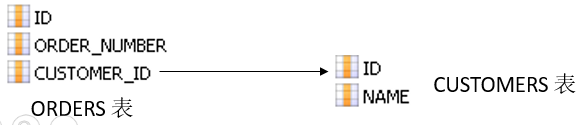
（1）双向1-n与双向n-1是完全相同的两种情形。

（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。



（3）关系数据模型：ORDERS表中的CUSTOMER\_ID参照CUSTOMER表的主键。

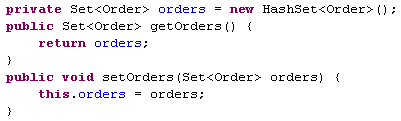


（4）当Session从数据库中加载Java集合时，创建的是Hibernate内置集合类的实例，因此**在持久化类中定义集合属性时必须把属性声明为Java接口类型**（也就是Set，而不能声明为HashSet）：

① Hibernate的内置集合类具有集合代理功能，支持延迟检索策略。

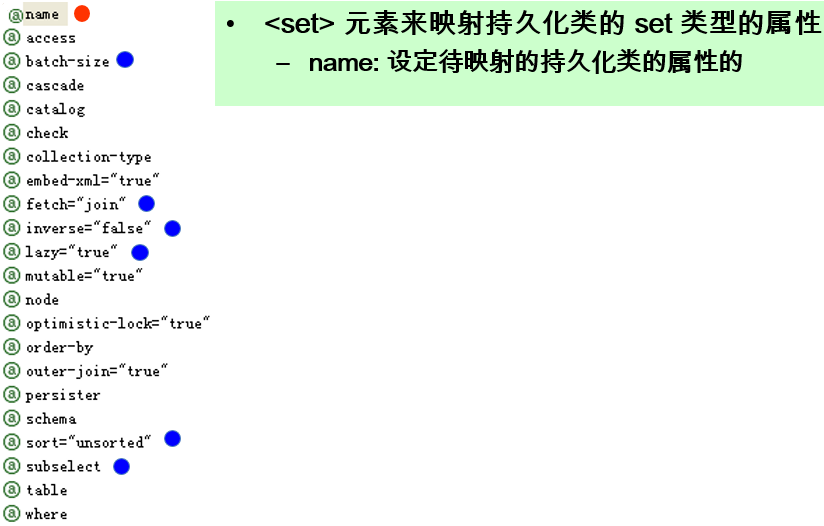
② 事实上，Hibernate的内置集合类封装了JDK中的集合类，这使得Hibernate能够对缓存中的集合对象进行脏检查，按照集合对象的状态来同步更新数据库。

（5）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。



（6）set元素

**Hibernate 使用 <set> 元素来映射set类型的属性。**



inverse属性：

① 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。

② 在没有设置inverse=true的情况下，父子两边都维护父子关系。

③ **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）。

cascade属性：

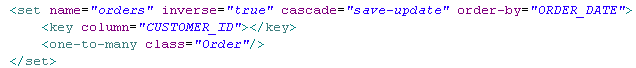
在对象–关系映射文件中，用于映射持久化类之间关联关系的元素。<set>，<many-to-one>和 <one-to-one> 都有一个cascade属性，它用于指定如何操纵与当前对象关联的其他对象。



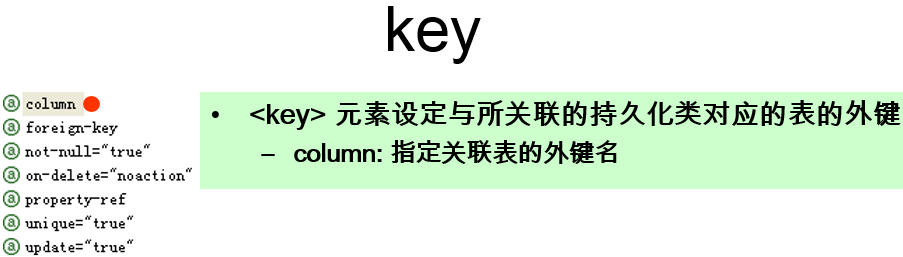
order-by属性：

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。

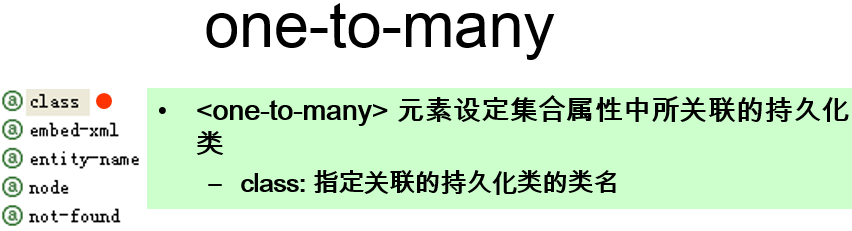
order-by属性中还可以加入SQL函数。



（7）key元素



（8）one-to-many元素



**示例：**



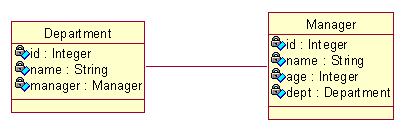
**说明：**



### 2、映射一对一关联关系

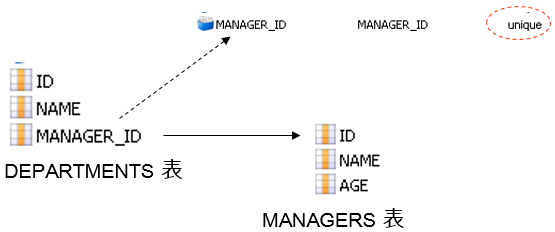
#### 1）概述

（1）域模型：



（2）关系数据模型：

按照外键映射：



按照主键映射：

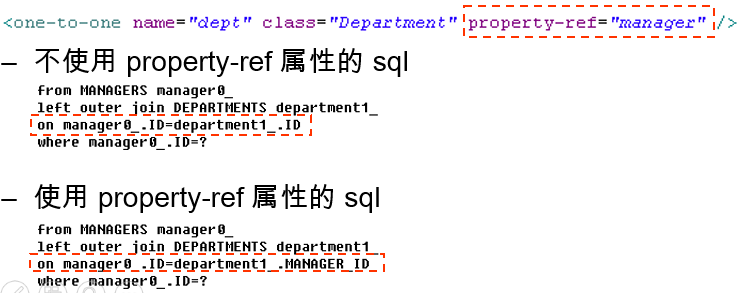


#### 2）基于外键映射的1-1

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：

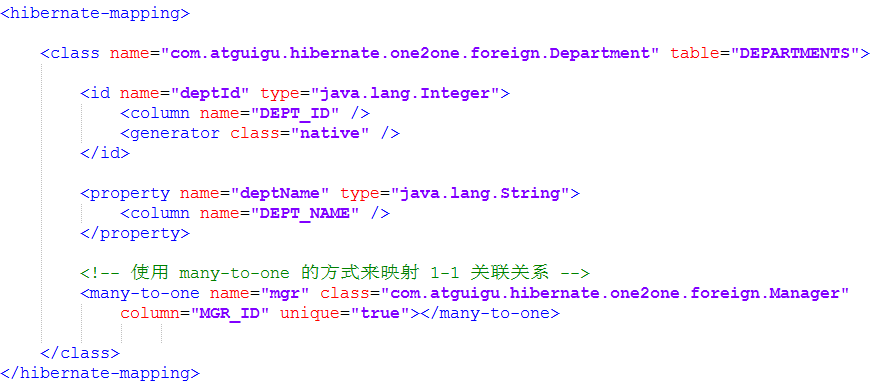


另一端需要使用**one-to-one元素**，该元素使用**property-ref属性**指定使用被关联实体主键以外的字段作为关联字段：



**示例：**

Department.hbm.xml：

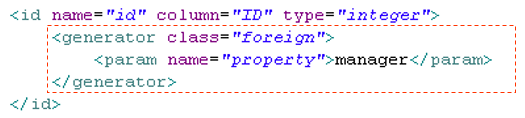


Manager.hbm.xml：



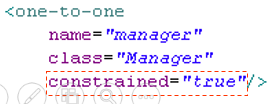
#### 3）基于主键映射的1-1

（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”。



（2）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。

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，

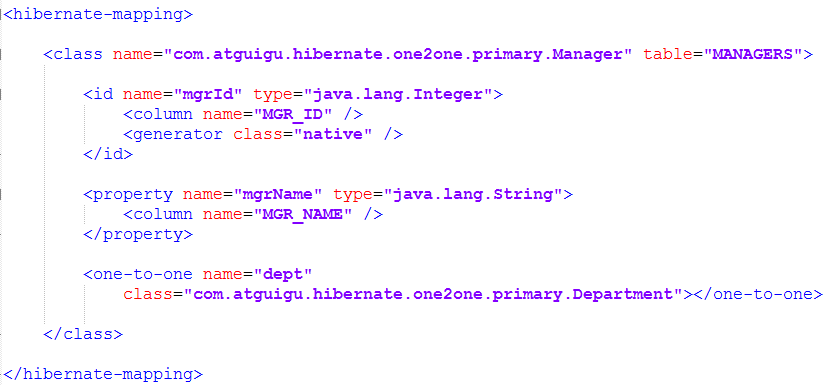


**示例：**

Department.hbm.xml：



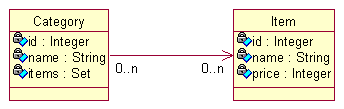
Manager.hbm.xml：



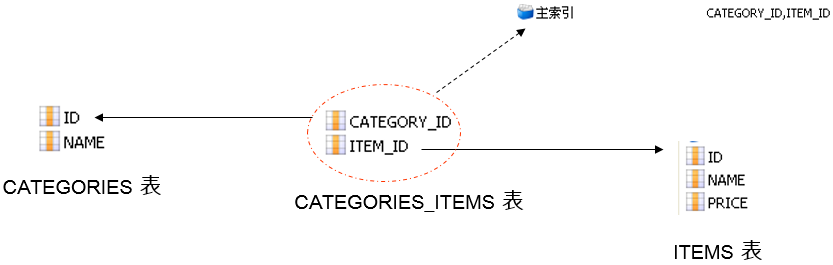
### 3、映射多对多关联关系

#### 1）单向n-n

（1）域模型



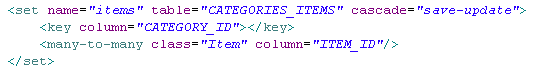
（2）关系数据模型



**（3）n-n的关联必须使用连接表**

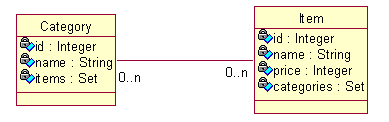
与1-n映射类似，必须为set集合元素添加key子元素，指定CATEGORIES\_ITEMS表中参照CATEGORIES表的外键为CATEGORIY\_ID；与1-n关联映射不同的是，建立n-n关联时，集合中的元素使用many-to-many，many-to-many子元素的class属性指定items集合中存放的是Item对象，column属性指定CATEGORIES\_ITEMS表中参照ITEMS表的外键为ITEM\_ID。

**示例：**

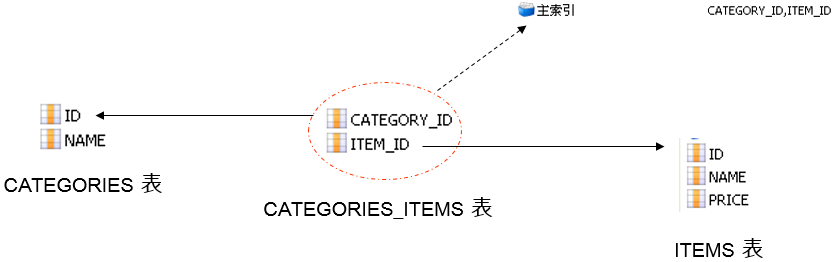


#### 2）双向n-n

（1）域模型



（2）关系数据模型



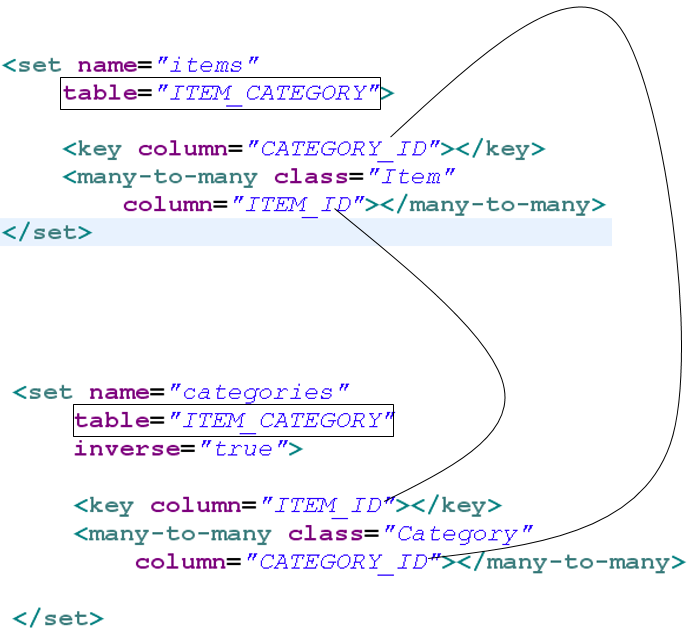
**（3）双向n-n关联必须使用连接表。**

① 双向n-n关联需要**两端都使用集合属性**。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。

② 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。

③ **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。

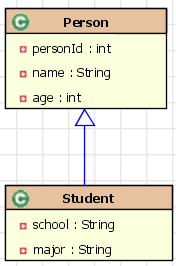
**示例：**



### 4、映射继承关系

#### 1）概述

（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。



（2）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。

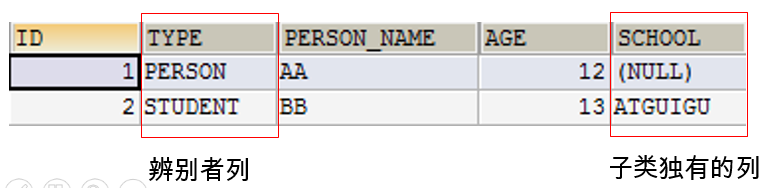
③ **使用union-subclass进行映射**：域模型中的每个类映射到一个表，通过关系数据模型中的外键来描述表之间的继承关系。这也就相当于按照域模型的结构来建立数据库中的表，并通过外键来建立表之间的继承关系。

#### 2）采用subclass元素的继承映射

（1）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）。

（2）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**。

（3）**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

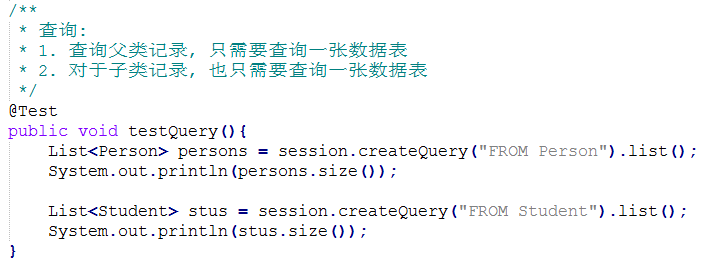


**（4）示例：**

配置文件（父类子类存储在一张表上只需要一个配置文件）：



查询操作：

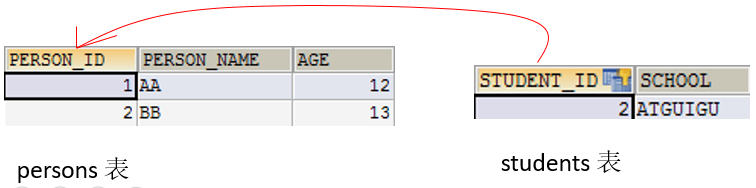


#### 3）采用joined-subclass元素的继承映射

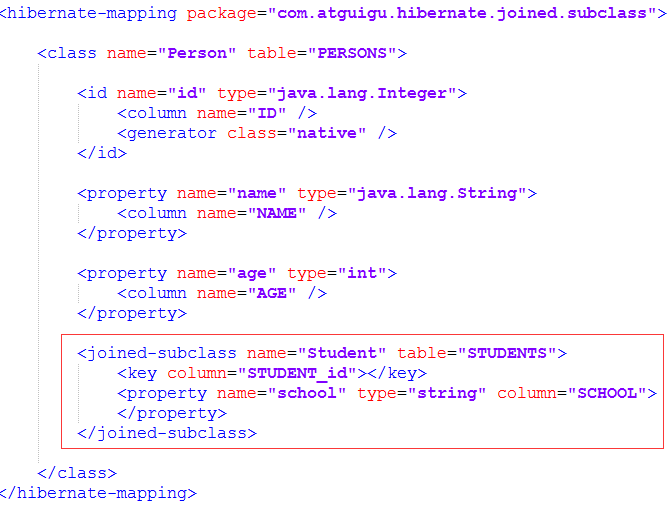
（1）采用joined-subclass元素的继承映射可以实现**每个子类一张表**。采用这种映射策略时，**父类实例保存在父类表中，子类实例由父类表和子类表共同存储**。因为子类实例也是一个特殊的父类实例，因此必然也包含了父类实例的属性。于是将子类和父类共有的属性保存在父类表中，子类增加的属性，则保存在子类表中。

（2）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**。

（3）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。



**（4）示例：**



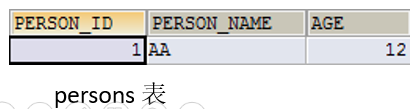
#### 4）采用union-subclass元素的继承映射

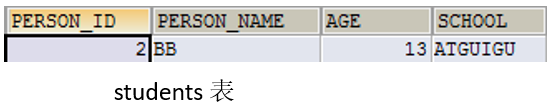
（1）采用union-subclass元素可以实现将**每一个实体对象映射到一个独立的表中**。子类增加的属性可以有非空约束，即**父类实例的数据保存在父表中，而子类实例的数据保存在子类表中**。子类实例的数据仅保存在子类表中，而在父类表中没有任何记录。

（2）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。

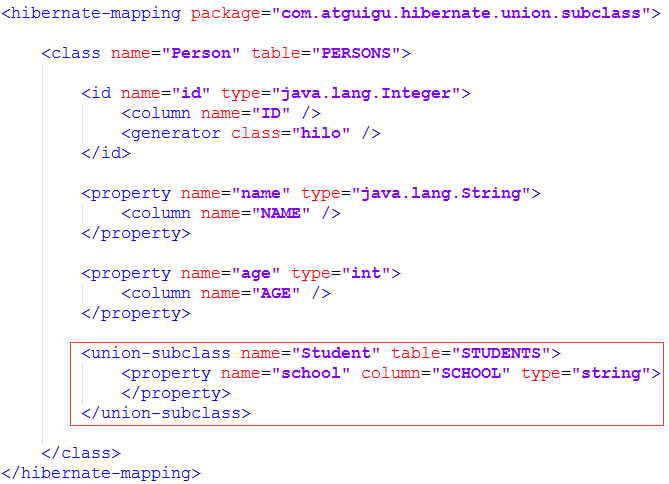
（3）在这种映射策略下，**既不需要使用鉴别者列，也无须使用key元素来映射共有主键**。

（4）**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。





**（5）示例：**



#### 5）三种继承映射方式的比较



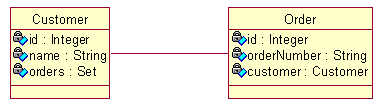
## 五、Hibernate检索策略

### 1、概述

检索数据时的2个问题：

① 不浪费内存：当Hibernate从数据库中加载Customer对象时，如果同时加载所有关联的Order对象，而程序实际上仅仅需要访问Customer对象，那么这些关联的Order对象就白白浪费了许多内存。

② 更高的查询效率：发送尽可能少的SQL语句。



### 2、类级别的检索策略

1）类级别可选的检索策略包括立即检索和延迟检索，默认为延迟检索。

立即检索：立即加载检索方法指定的对象

延迟检索：延迟加载检索方法指定的对象。在使用具体的属性时，再进行加载。

**2）类级别的检索策略可以通过 <class> 元素的lazy属性进行设置。**

3）如果程序加载一个对象的目的是为了访问它的属性，可以采取立即检索；如果程序加载一个持久化对象的目的是仅仅为了获得它的引用，可以采用延迟检索。注意出现懒加载异常！

4）**无论 <class> 元素的lazy属性是true还是false，Session的get() 方法及Query的list() 方法在类级别总是使用立即检索策略，也就是说该属性只对Session的load() 方法起作用**。

5）若 <class> 元素的lazy属性为true或取默认值，Session的load() 方法不会执行查询数据表的SELECT语句，仅返回代理类对象的实例，该代理类实例有如下特征：

（1）由Hibernate在运行时采用CGLIB工具动态生成。

（2）Hibernate创建代理类实例时，仅初始化其OID属性。

（3）在应用程序第一次访问代理类实例的非OID属性时，Hibernate会初始化代理类实例。

### 3、一对多和多对多的检索策略

1）在映射文件中，用 <set> 元素来配置一对多关联及多对多关联关系。<set> 元素有lazy、fetch属性和batch-size属性：

**Lazy**：**主要决定orders集合被初始化的时机**，即到底是在加载Customer对象时就被初始化还是在程序访问orders集合时被初始化。

**fetch：取值为“select”或“subselect”时，决定初始化orders的查询语句的形式；取值为“join”，则决定orders集合被初始化的时机，若把fetch设置为“join”，lazy属性将被忽略。**

**batch-size**：用来为延迟检索策略或立即检索策略设定批量检索的数量，批量检索能减少 SELECT语句的数目，提高延迟检索或立即检索的运行性能。



**2）laze属性**

（1）在延迟检索（lazy属性值为true）集合属性时，Hibernate在以下情况下初始化集合代理类实例：

① 应用程序第一次访问集合属性：iterator()、size()、isEmpty()、contains()等方法。

② 通过Hibernate.initialize() 静态方法显式初始化。

（2）增强延迟检索（lazy属性为extra）：与lazy=”true” 类似，主要区别是**增强延迟检索策略能进一步延迟Customer对象的orders集合代理实例的初始化时机**：

① 当程序第一次访问orders属性的iterator() 方法时，会导致orders集合代理类实例的初始化。

② 当程序第一次访问order属性的size()、contains()和isEmpty() 方法时，Hibernate不会初始化orders集合类的实例，仅通过特定的select语句查询必要的信息，不会检索所有的Order对象。

**3）fetch属性**

（1）当fetch属性为“subselect”时：

① 假定Session缓存中有n个orders集合代理类实例没有被初始化，Hibernate能够通过带子查询的select语句，来批量初始化n个orders集合代理类实例。

**② batch-size属性将被忽略。**

③ 子查询中的select语句为查询CUSTOMERS表OID的SELECT语句。

（2）当fetch属性为“join”时：

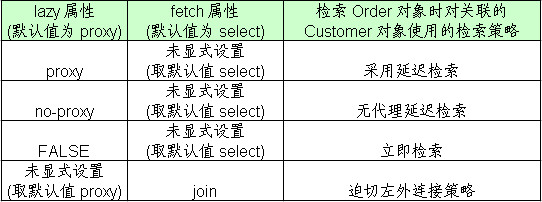
① 检索Customer对象时，会采用**迫切左外连接**（通过左外连接加载与检索指定的对象关联的对象）策略来检索所有关联的Order对象。

**② lazy属性将被忽略。**

③ Query的list() 方法会忽略映射文件中配置的迫切左外连接检索策略，而依旧采用延迟加载策略。

### 4、多对一和一对一关联的检索策略

1）和 <set> 一样，<many-to-one> 元素也有一个lazy属性和fetch属性。



（1）**若fetch属性设为join，那么lazy属性被忽略**。

（2）迫切左外连接检索策略的优点在于比立即检索策略使用的SELECT语句更少。

（3）无代理延迟检索需要增强持久化类的字节码才能实现。

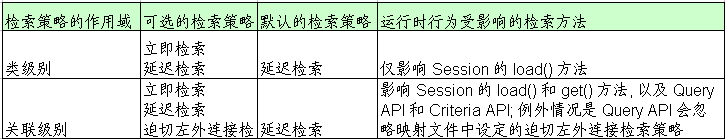
**2）Query的list方法会忽略映射文件配置的迫切左外连接检索策略，而采用延迟检索策略。**

3）如果在关联级别使用了延迟加载或立即加载检索策略，可以设定批量检索的大小，以帮助提高延迟检索或立即检索的运行性能。

4）Hibernate允许在应用程序中覆盖映射文件中设定的检索策略。

### 5、检索策略小结

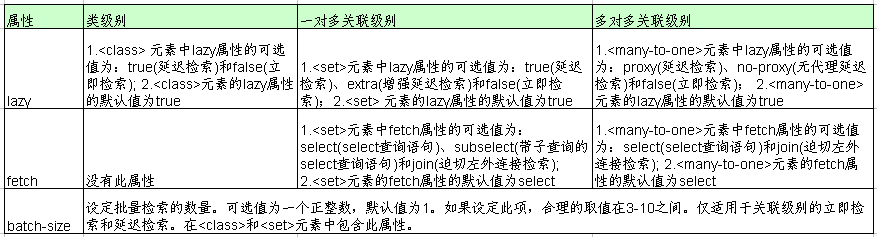
1）类级别和关联级别可选的检索策略及默认的检索策略



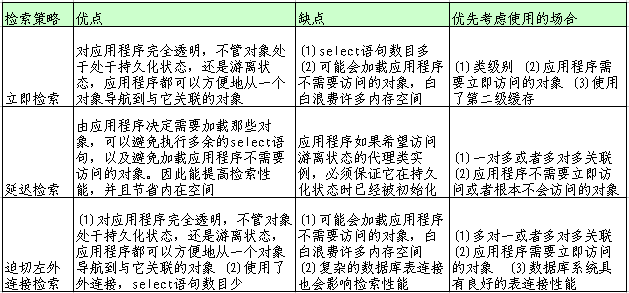
2）3种检索策略的运行机制



3）映射文件中用于设定检索策略的几个属性



4）比较Hibernate的三种检索策略



## 六、Hibernate 检索方式

### 1、概述

Hibernate提供了以下几种检索对象的方式：

① 导航对象图检索方式：根据已经加载的对象导航到其他对象

② OID检索方式：按照对象的OID来检索对象

③ HQL检索方式：使用面向对象的 HQL 查询语言

④ QBC检索方式：使用QBC（Query By Criteria） API来检索对象，这种API封装了基于字符串形式的查询语句，提供了更加面向对象的查询接口

⑤ 本地SQL检索方式：使用本地数据库的 SQL 查询语句

### 2、HQL检索方式

#### 1）概述

HQL（Hibernate Query Language）是面向对象的查询语言，它和SQL查询语言有些相似。在Hibernate提供的各种检索方式中，HQL是使用最广的一种检索方式，它有如下功能：

在查询语句中设定各种查询条件

支持投影查询，即仅检索出对象的部分属性

支持分页查询

支持连接查询

支持分组查询，允许使用HAVING和GROUP BY关键字

提供内置聚集函数，如sum()、min() 和max()

支持子查询

支持动态绑定参数

能够调用 用户定义的SQL函数或标准的SQL函数

#### 2）基础知识

（1）HQL检索方式包括以下步骤：

① 通过Session的createQuery() 方法创建一个Query对象，它包括一个HQL查询语句，HQL查询语句中可以包含命名参数。

② 动态绑定参数。

③ 调用Query相关方法执行查询语句。

（2）**Qurey接口支持方法链编程风格**，它的setXxx() 方法返回自身实例，而不是void类型。

（3）HQL vs SQL：

① **HQL查询语句是面向对象的，Hibernate负责解析HQL查询语句**，然后根据对象-关系映射文件中的映射信息，把HQL查询语句翻译成相应的SQL语句。**HQL查询语句中的主体是域模型中的类及类的属性**。

② SQL查询语句是与关系数据库绑定在一起的。SQL查询语句中的主体是数据库表及表的字段。

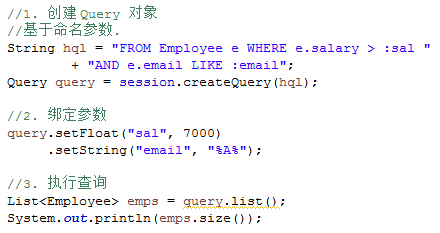
（4）绑定参数：

① Hibernate的参数绑定机制依赖于JDBC API中的PreparedStatement的预定义SQL语句功能。

② HQL的参数绑定由两种形式：

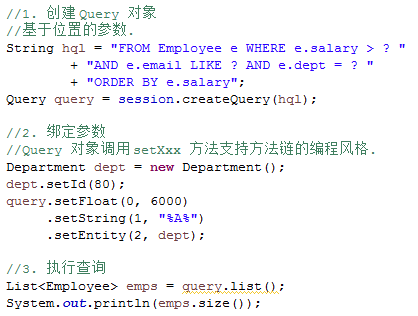
按参数名字绑定：在HQL查询语句中定义命名参数，命名参数以“:”开头。

示例：



按参数位置绑定：在HQL查询语句中用“?”来定义参数位置。

示例：



③ 相关方法：

setEntity()：把参数与一个持久化类绑定。

setParameter()：绑定任意类型的参数，该方法的第三个参数显式指定Hibernate映射类型。

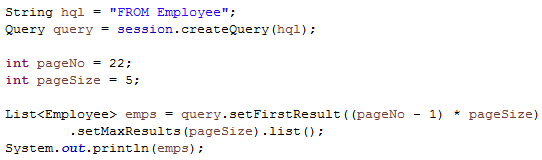
（5）HQL采用ORDER BY关键字对查询结果排序。

#### 3）分页查询

（1）setFirstResult(int firstResult)：设定从哪一个对象开始检索，参数firstResult表示这个对象在查询结果中的索引位置，索引位置的起始值为0。默认情况下，Query从查询结果中的第一个对象开始检索。

（2）setMaxResults(int maxResults)：设定一次最多检索出的对象的数目。在默认情况下，Query和Criteria接口检索出查询结果中所有的对象。

**（3）示例：**



#### 4）命名查询

（1）在映射文件中定义命名查询语句：

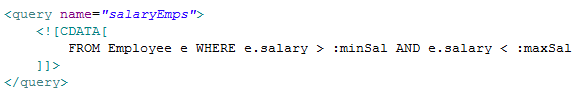
① Hibernate允许在映射文件中定义字符串形式的查询语句。

② **<query> 元素**用于定义一个HQL查询语句，它和 <class> 元素并列。

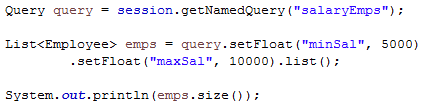
③ 在程序中通过Session的getNamedQuery() 方法获取查询语句对应的Query对象。

**（2）示例：**

配置文件：



查询代码：



#### 5）投影查询

（1）投影查询：**查询结果仅包含实体的部分属性，通过SELECT关键字实现**。

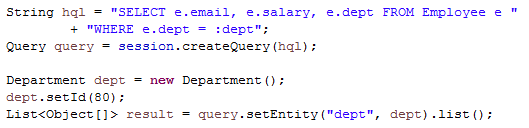
（2）Query的list() 方法返回的集合中包含的是数组类型的元素，每个对象数组代表查询结果的一条记录。

（3）可以在持久化类中定义一个对象的构造器来包装投影查询返回的记录，使程序代码能完全运用面向对象的语义来访问查询结果集。

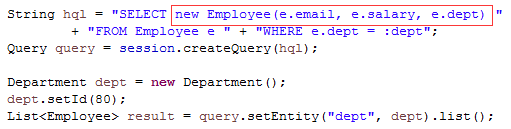
（4）可以通过DISTINCT关键字来保证查询结果不会返回重复元素。

**（5）示例：**

查询结果用数组存储：



查询结果封装成对象：

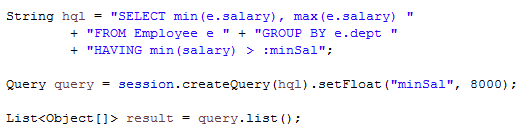


#### 6）报表查询

（1）报表查询用于对数据分组和统计，与SQL一样，HQL利用GROUP BY关键字对数据分组，用HAVING关键字对分组数据设定约束条件。

（2）在HQL查询语句中可以调用以下聚集函数：count()、min()、max()、sum()、avg()。

**（3）示例：**



#### 7）迫切左外连接

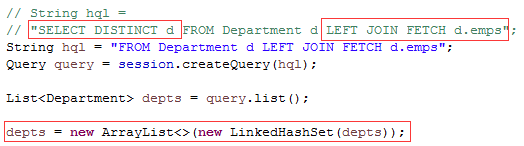
（1）**LEFT JOIN FETCH关键字**表示迫切左外连接检索策略。

（2）list() 方法返回的集合中存放实体对象的引用，每个Department对象关联的Employee集合都被初始化，存放所有关联的Employee的实体对象。

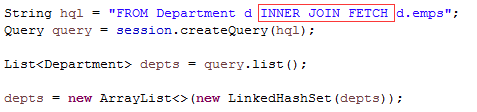
（3）查询结果中可能会包含重复元素，可以通过DISTINCT关键字去重或者通过一个HashSet来过滤重复元素。

**（4）示例：**

迫切左外连（返回所有部门不论有没有员工）：



迫切内连接（关键字INNER JOIN FETCH，也可以省略INNER关键字，只返回有员工的部门）：



#### 8）关联级别运行时的检索策略

（1）如果在HQL中没有显式指定检索策略，将使用映射文件配置的检索策略。

（2）HQL会忽略映射文件中设置的迫切左外连接检索策略，**如果希望HQL采用迫切左外连接策略，就必须在HQL查询语句中显式的指定它**。

（3）若在HQL代码中显式指定了检索策略，就会覆盖映射文件中配置的检索策略。

### 3、QBC检索和本地SQL检索

#### 1）QBC查询

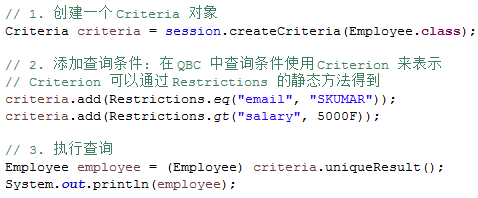
通过使用Hibernate提供的Query By Criteria API来查询对象，这种API封装了SQL语句的动态拼装，对查询提供了更加面向对象的功能接口。

#### 2）本地SQL查询

用来完善HQL不能涵盖所有的查询特性，比如HQL不能进行的插入操作。

#### 3）QBC查询示例

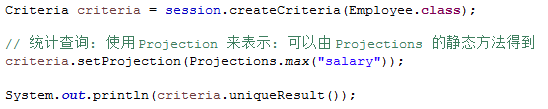
（1）添加查询条件



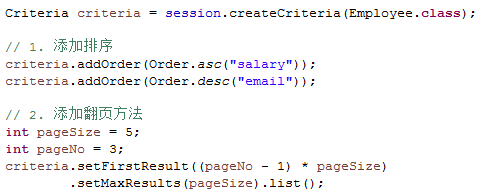
（2）添加AND和OR条件



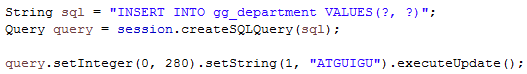
（3）统计查询



（4）排序、翻页



#### 4）本地SQL查询示例



## 七、Hibernate二级缓存

### 1、概述

#### 1）缓存（Cache）

计算机领域非常通用的概念。它**介于应用程序和永久性数据存储源（如硬盘上的文件或者数据库）之间，其作用是降低应用程序直接读写永久性数据存储源的频率，从而提高应用的运行性能**。缓存中的数据是数据存储源中数据的拷贝。**缓存的物理介质通常是内存。**

#### 2）Hibernate中提供了两个级别的缓存

（1）第一级别的缓存是Session级别的缓存，它是属于事务范围的缓存。这一级别的缓存由hibernate管理的。

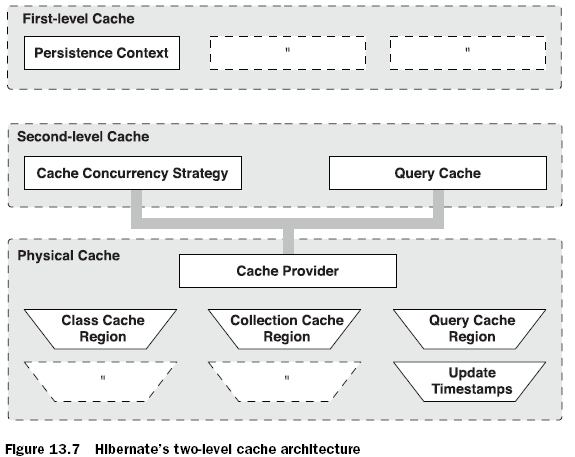
（2）第二级别的缓存是SessionFactory级别的缓存，它是属于进程范围的缓存。

#### 3）SessionFactory的缓存可以分为两类

（1）内置缓存：**Hibernate自带的，不可卸载。**通常在Hibernate的初始化阶段，Hibernate会把映射元数据和预定义的SQL语句放到SessionFactory的缓存中，映射元数据是映射文件中数据（.hbm.xml 文件中的数据）的复制。该内置缓存是只读的。

（2）外置缓存（二级缓存）：**一个可配置的缓存插件。**在默认情况下，SessionFactory不会启用这个缓存插件。外置缓存中的数据是数据库数据的复制，外置缓存的物理介质可以是内存或硬盘。

#### 4）二级缓存的架构



### 2、二级缓存的适用情况

1）适合放入二级缓存中的数据：

① 很少被修改

② 不是很重要的数据, 允许出现偶尔的并发问题

2）不适合放入二级缓存中的数据：

① 经常被修改

② 财务数据，绝对不允许出现并发问题

③ 与其他应用程序共享的数据

### 3、二级缓存的并发访问策略

1）两个并发的事务同时访问持久层的缓存的相同数据时，也有可能出现各类并发问题。

2）二级缓存可以设定以下4种类型的并发访问策略，每一种访问策略对应一种事务隔离级别：

（1）非严格读写（Nonstrict-read-write）：不保证缓存与数据库中数据的一致性。提供Read Uncommited事务隔离级别，对于极少被修改，而且允许脏读的数据，可以采用这种策略。

（2）**读写型（Read-write）**：提供Read Commited数据隔离级别。对于经常读但是很少被修改的数据，可以采用这种隔离类型，因为它可以防止脏读。

（3）事务型（Transactional）：仅在受管理环境下适用。它提供了Repeatable Read事务隔离级别。对于经常读但是很少被修改的数据，可以采用这种隔离类型，因为它可以防止脏读和不可重复读。

（4）只读型（Read-Only）：提供Serializable数据隔离级别，对于从来不会被修改的数据，可以采用这种访问策略。

### 4、管理Hibernate的二级缓存

1）Hibernate的二级缓存是进程或集群范围内的缓存。

2）二级缓存是可配置的的插件，Hibernate允许选用以下类型的缓存插件：

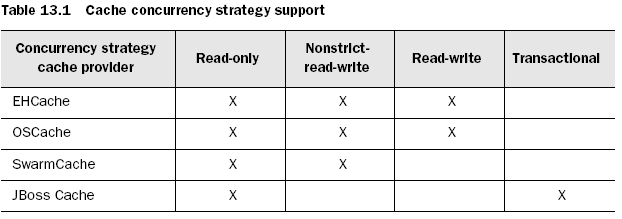
**EHCache**：可作为进程范围内的缓存，存放数据的物理介质可以使内存或硬盘，对Hibernate的查询缓存提供了支持。

OpenSymphony OSCache：可作为进程范围内的缓存，存放数据的物理介质可以使内存或硬盘，提供了丰富的缓存数据过期策略，对Hibernate的查询缓存提供了支持。

SwarmCache：可作为集群范围内的缓存，但不支持Hibernate的查询缓存。

JbossCache：可作为集群范围内的缓存，支持Hibernate的查询缓存。

3）4种缓存插件支持的并发访问策略（x代表支持，空白代表不支持）：



### 5、使用Hibernate二级缓存的步骤

（选用EHCache插件为例）

1）加入二级缓存插件的jar包及配置文件：

（1）复制 **\hibernate-release-4.2.4.Final\lib\optional\ehcache\\*.jar**到当前Hibrenate应用的类路径下。

（2）复制**hibernate-release-4.2.4.Final\project\etc\ehcachexml**到当前WEB应用的类路径下。

2）配置hibernate.cfg.xml

（1）配置启用hibernate的二级缓存

**<property name="cache.use\_second\_level\_cache">true</property>**

（2）配置hibernate二级缓存使用的产品

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

**org.hibernate.cache.ehcache.EhCacheRegionFactory**

**</property>**

（3）配置对哪些类使用hibernate的二级缓存

**<class-cache usage="read-write" class="com.atguigu.hibernate.entities.Employee"/>**

实际上也可以在 .hbm.xml文件中配置对哪些类使用二级缓存，及二级缓存的策略是什么。

### 6、集合级别的二级缓存的配置

1）配置对集合使用二级缓存

**<collection-cache usage="read-write" collection="com. hibernate.entities.Department.emps"/>**

也可以在 .hbm.xml文件中进行配置

**<set name="emps" table="GG\_EMPLOYEE" inverse="true" lazy="true">**

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

**<key>**

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

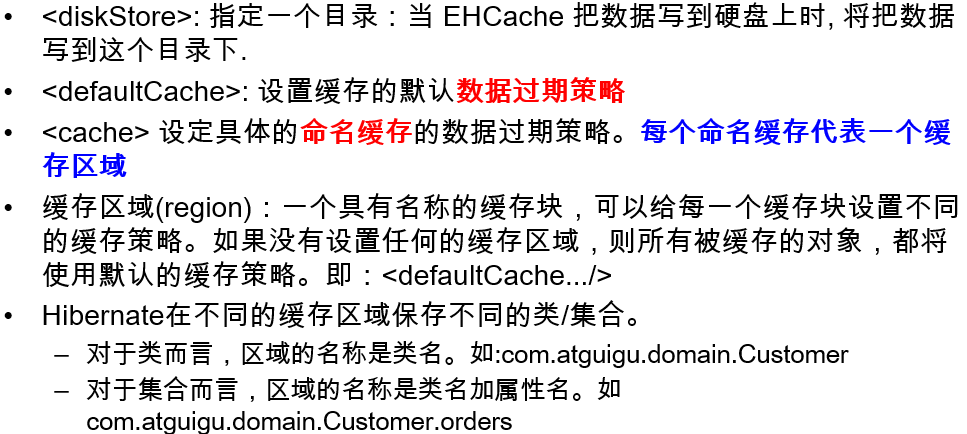
**</key>**

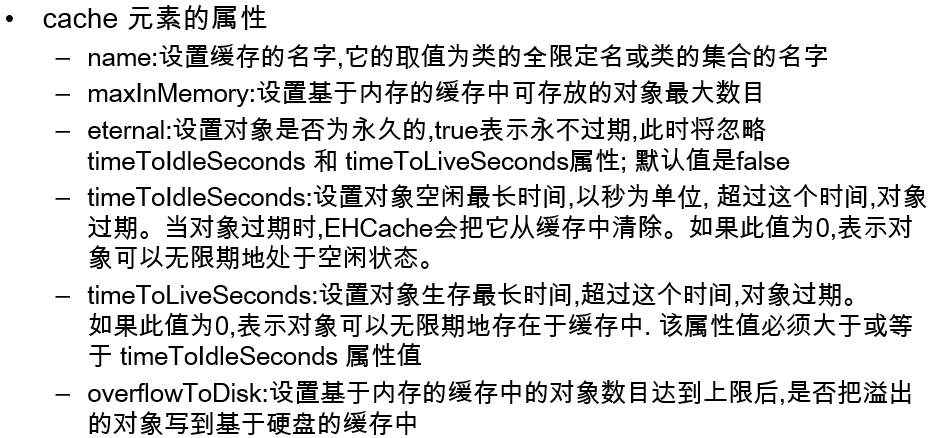
**<one-to-many class="com.atguigu.hibernate.entities.Employee" />**

**</set>**

2）注意：还需要配置集合中的元素对应的持久化类也使用二级缓存，否则将会多出n条SQL语句。

### 7、ehcache.xml说明





### 8、查询缓存

1）对于经常使用的查询语句，如果启用了查询缓存，当第一次执行查询语句时，Hibernate会把查询结果存放在查询缓存中，以后再次执行该查询语句时，只需从缓存中获得查询结果，从而提高查询性能。

2）查询缓存使用于如下场合：

（1）应用程序运行时经常使用查询语句

（2）很少对与查询语句检索到的数据进行插入、删除和更新操作

3）启用查询缓存的步骤：

（1）配置二级缓存，因为查询缓存依赖于二级缓存

（2）在Hibernate配置文件中启用查询缓存

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

（3）对于希望启用查询缓存的查询语句，用Query或Criteria的setCacheable(true) 方法

### 9、时间戳缓存区域

时间戳缓存区域存放了对于查询结果相关的表进行插入，更新或删除操作的时间戳。Hibernate通过时间戳缓存区域来判断被缓存的查询结果是否过期，其运行过程如下：

① T1时刻执行查询操作，把查询结果存放在QueryCache区域，记录该区域的时间戳为T1

② T2时刻对查询结果相关的表进行更新操作，Hibernate把T2时刻存放在UpdateTimestampCache区域

③ T3时刻执行查询结果前，先比较QueryCache区域的时间戳和UpdateTimestampCache区域的时间戳，若T2 > T1，那么就丢弃原先存放在QueryCache区域的查询结果，重新到数据库中查询数据，再把结果存放到QueryCache区域；若T2 < T1，直接从QueryCache中获得查询结果

### 10、Query接口的iterate() 方法

1）Query接口的iterator() 方法

（1）同list() 一样也能执行查询操作

（2）list() 方法执行的SQL语句包含实体类对应的数据表的所有字段

（3）Iterator() 方法执行的SQL语句中仅包含实体类对应的数据表的ID字段

（4）当遍历访问结果集时，该方法先到Session缓存及二级缓存中查看是否存在特定OID的对象，如果存在，就直接返回该对象，如果不存在该对象就通过相应的SQL Select语句到数据库中加载特定的实体对象

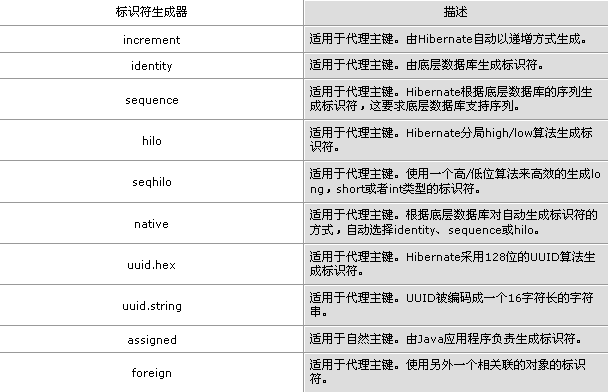
2）**大多数情况下，应考虑使用list() 方法执行查询操作。**iterator() 方法仅在满足以下条件的场合，可以**稍微**提高查询性能：

（1）要查询的数据表中包含大量字段

（2）启用了二级缓存，且二级缓存中可能已经包含了待查询的对象

## 八、散碎的知识点

### 1、Hibernate主键生成策略



### 2、数据库的隔离级别

#### 1）多个事务导致的并发问题

对于同时运行的多个事务，当这些事务访问数据库中相同的数据时，如果没有采取必要的隔离机制，就会导致各种并发问题：

**脏读**：对于两个事物T1、T2，T1读取了已经被T2更新但还没有被提交的字段，之后，若T2回滚，T1读取的内容就是临时且无效的。

**不可重复读**：对于两个事物T1、T2，T1读取了一个字段，然后T2更新了该字段，之后，T1再次读取同一个字段，值就不同了。

**幻读**：对于两个事物T1、T2，T1从一个表中读取了一个字段，然后T2在该表中插入了一些新的行，之后，如果T1再次读取同一个表，就会多出几行。

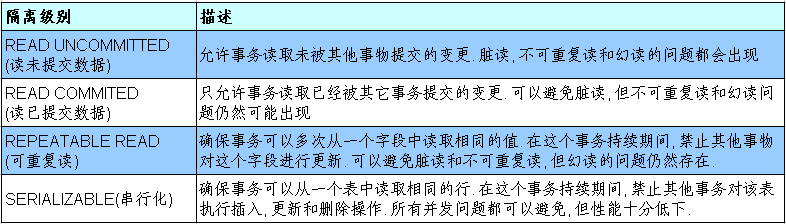
#### 2）数据库事务的隔离级别

##### （1）概述

数据库系统必须具有隔离并发运行各个事务的能力，使它们不会相互影响，避免各种并发问题。

一个事务与其他事务隔离的程度称为隔离级别。数据库规定了多种事务隔离级别，不同隔离级别对应不同的干扰程度，隔离级别越高，数据一致性就越好，但并发性越弱。

##### （2）数据库提供的4种事务隔离级别



Oracle支持的2种事务隔离级别：READ COMMITED、SERIALIZABLE，Oracle默认的事务隔离级别为：READ COMMITED。

Mysql支持4中事务隔离级别，Mysql默认的事务隔离级别为：REPEATABLE READ。

##### （3）在MySql中设置隔离级别

① 每启动一个mysql程序，就会获得一个单独的数据库连接，每个数据库连接都有一个全局变量：@@tx\_isolation，表示当前的事务隔离级别。MySQL默认的隔离级别为：Repeatable Read。

② 查看当前的隔离级别：**SELECT @@tx\_isolation;**

③ 设置当前mysql连接的隔离级别：

**set transaction isolation level read committed;**

④ 设置数据库系统的全局的隔离级别：

**set global transaction isolation level read committed;**

#### 3）在Hibernate中设置隔离级别

JDBC数据库连接使用数据库系统默认的隔离级别。在Hibernate的配置文件中可以指定**connection.isolation**属性显式的设置隔离级别。每一个隔离级别都对应一个整数：

1. READ UNCOMMITED

2. READ COMMITED

4. REPEATABLE READ

8. SERIALIZEABLE

**示例：**



### 3、Hibernate的配置文件

#### 1）概述

（1）Hibernate配置文件主要用于配置数据库连接和Hibernate运行时所需的各种属性。

（2）每个Hibernate配置文件对应一个Configuration对象。

（3）Hibernate配置文件可以有两种格式：

**hibernate.cfg.xml**

hibernate.properties

#### 2）常用属性

##### （1）JDBC连接属性

connection.url：数据库URL

connection.username：数据库用户名

connection.password：数据库用户密码

connection.driver\_class：数据库JDBC驱动

dialect：配置数据库的方言，根据底层的数据库不同产生不同的 sql 语句，Hibernate 会针对数据库的特性在访问时进行优化

##### （2）C3P0数据库连接池属性

**（C3P0 jar包路径：hibernate-release-4.2.4.Final\lib\optional\c3p0\\*.jar）**

hibernate.c3p0.max\_size：数据库连接池的最大连接数

hibernate.c3p0.min\_size：数据库连接池的最小连接数

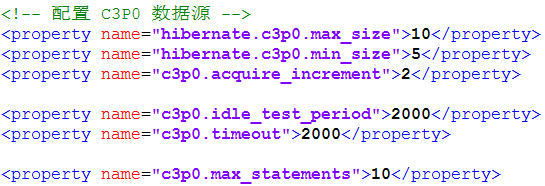
hibernate.c3p0.acquire\_increment：当数据库连接池中的连接耗尽时, 同一时刻获取多少个数据库连接

hibernate.c3p0.timeout：数据库连接池中连接对象在多长时间没有使用过后，就应该被销毁

hibernate.c3p0.idle\_test\_period：表示连接池检测线程多长时间检测一次池内的所有链接对象是否超时。连接池本身不会把自己从连接池中移除，而是专门有一个线程按照一定的时间间隔来做这件事，这个线程通过比较连接对象最后一次被使用时间和当前时间的时间差来和timeout做对比，进而决定是否销毁这个连接对象。

hibernate.c3p0.max\_statements：缓存Statement对象的数量

**示例：**



##### （3）其他

show\_sql：是否将运行期生成的SQL输出到日志以供调试。取值：true | false

format\_sql：是否将SQL转化为格式良好的SQL。取值：true | false

hbm2ddl.auto：在启动和停止时自动地创建，更新或删除数据库模式。取值：create | update | create-drop | validate

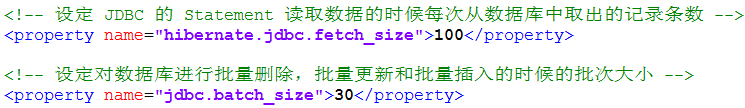
hibernate.jdbc.fetch\_size：实质是调用Statement.setFetchSize()方法**设定JDBC的Statement读取数据的时候每次从数据库中取出的记录条数**。

例如一次查询1万条记录，对于Oracle的JDBC驱动来说，是不会1次性把1万条取出来的，而只会取出fetchSize条数，当结果集遍历完了这些记录以后，再去数据库取fetchSize条数据。因此大大节省了无谓的内存消耗。fetchSize设的越大，读数据库的次数越少，速度越快；fetchSize越小，读数据库的次数越多，速度越慢。Oracle数据库的JDBC驱动默认的fetchSize=10，是一个保守的设定，根据测试，当fetchSize=50时，性能会提升1倍之多，当**fetchSize=100**，性能还能继续提升20%，fetchSize继续增大，性能提升的就不显著了。并不是所有的数据库都支持fetchSize特性，例如MySQL就不支持。

hibernate.jdbc.batch\_size：**设定对数据库进行批量删除，批量更新和批量插入的时候的批次大小**，类似于设置缓冲区大小的意思。batchSize越大，批量操作时向数据库发送sql的次数越少，速度就越快。

测试结果是当batchSize=0的时候，使用Hibernate对Oracle数据库删除1万条记录需要25秒，batchSize=50的时候，删除仅仅需要5秒！Oracle数据库**batchSize=30**的时候比较合适。

**示例：**



### 4、管理Session

1）Hibernate自身提供了三种管理Session对象的方法：

**（1）Session对象的生命周期与本地线程绑定**

（2）Session 对象的生命周期与JTA事务绑定

（3）Hibernate 委托程序管理Session对象的生命周期

2）在Hibernate的配置文件中，hibernate.current\_session\_context\_class属性用于指定Session管理方式，可选值包括：

**Thread**：Session对象的生命周期与本地线程绑定

jta\*：Session对象的生命周期与JTA事务绑定

managed：Hibernate委托程序来管理Session对象的生命周期

3）Session对象的生命周期与本地线程绑定、

（1）如果把Hibernate配置文件的hibernate.current\_session\_context\_class属性值设为thread，Hibernate就会按照与本地线程绑定的方式来管理Session。

（2）Hibernate按一下规则把Session与本地线程绑定：

① 当一个线程（threadA）第一次调用SessionFactory对象的getCurrentSession() 方法时，该方法会创建一个新的Session(sessionA) 对象，把该对象与threadA绑定，并将sessionA返回。

② 当threadA再次调用SessionFactory对象的getCurrentSession() 方法时，该方法将返回sessionA对象。

③ 当threadA提交sessionA对象关联的事务时，Hibernate会自动flush sessionA对象的缓存, 然后提交事务，关闭sessionA对象。当threadA撤销sessionA对象关联的事务时，也会自动关闭sessionA对象。

④ 若threadA再次调用SessionFactory对象的getCurrentSession() 方法时，该方法会又创建一个新的Session(sessionB) 对象，把该对象与threadA绑定，并将sessionB返回。

4）批量处理数据

（1）批量处理数据是指在一个事务中处理大量数据。

（2）在应用层进行批量操作，主要有以下方式：

通过Session

通过HQL

通过StatelessSession

**通过JDBC API**

（3）通过Session来进行批量操作

① Session的save() 及update() 方法都会把处理的对象存放在自己的缓存中。如果通过一个Session对象来处理大量持久化对象，应该及时从缓存中清空已经处理完毕并且不会再访问的对象。具体的做法是在处理完一个对象或小批量对象后，立即调用flush() 方法刷新缓存，然后在调用clear() 方法清空缓存。

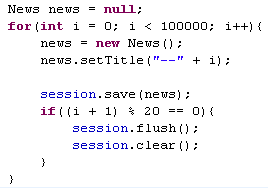
② 通过Session来进行处理操作会受到以下约束：

需要在Hibernate配置文件中设置JDBC单次批量处理的数目，应保证每次向数据库发送的批量的SQL语句数目与batch\_size属性一致。

若对象采用“identity”标识符生成器，则Hibernate无法在JDBC层进行批量插入操作。

进行批量操作时，建议关闭Hibernate的二级缓存。

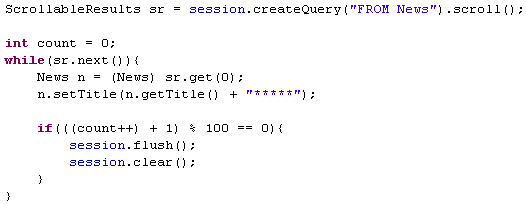
③ 批量插入数据：



④ 批量更新：在进行批量更新时，如果一下子把所有对象都加载到Session缓存，然后再缓存中一一更新，显然是不可取的。

⑤ 使用可滚动的结果集org.hibernate.ScrollableResults，该对象中实际上并不包含任何对象，只包含用于在线定位记录的游标。只有当程序遍历访问ScrollableResults对象的特定元素时，它才会到数据库中加载相应的对象。

⑥ org.hibernate.ScrollableResults对象由Query的scroll方法返回：



（4）通过HQL来进行批量操作

注意：HQL只支持INSERT INTO … SELECT形式的插入语句，但不支持INSERT INTO … VALUES形式的插入语句。所以使用HQL不能进行批量插入操作。

（5）通过StatelessSession来进行批量操作

从形式上看，StatelessSession与session的用法类似。StatelessSession与session相比，有以下区别：

① StatelessSession没有缓存，通过StatelessSession来加载、保存或更新后的对象处于游离状态。

② StatelessSession不会与Hibernate的第二级缓存交互。

③ 当调用StatelessSession的save()、update() 或delete() 方法时，这些方法会立即执行相应的SQL语句，而不会仅计划执行一条SQL语句。

④ StatelessSession不会进行脏检查，因此修改了Customer对象属性后，还需要调用StatelessSession的update()方法来更新数据库中数据。

⑤ StatelessSession不会对关联的对象进行任何级联操作。

⑥ 通过同一个StatelessSession对象两次加载OID为1的Customer对象，得到的两个对象内存地址不同。

⑦ StatelessSession所做的操作可以被Interceptor拦截器捕获到，但是会被Hibernate的事件处理系统忽略掉。