# Hibernate简介

Hibernate是一个orm（对象关系映射）框架，充当项目的持久层。Hibernate实际上是对jdbc进行了轻量级的封装，使用java了反射机制。

# 使用hibernate好处

1 使用hibernate，让程序员更关心业务流程，让数据库人员关注数据库相关的各种操作。

2 分层清晰，耦合性更小。

3 通用性强：可以更轻松的从一个数据库平台转移到别的平台。

4 对象化：把关系型数据库变成了java对象，更方便操作。

5 性能保证：hibernate可能根据不同的数据库，处理不同操作时是用最优化的sql语句。

6 增加了程序的鲁棒性。

# POJO类

和数据库某张表相互映射的那个java类。

序列化之后，可以唯一的标识该对象，同时可以在网络和文件传输。

简单的java对象，应当具有：

1 有一个主键特性，用于唯一标识该对象。

2 有其他的属性。

3 有对各个属性操作的get和set方法

4 属性一般是private修饰

5 一定有一个无参构造函数（反射使用）

# Sessionfactory

SessionFactory是一个重量级对象，应该尽量保证其单例的。

//不允许其他类继承（单态）

//在使用hibernate开发项目时，保证只有一个SessionFactory

//一个数据库对应一个SessionFactory对象

**final** **public** **class** MySessionFactory {

//SessionFactory是一个重量级对象，应该保证其是单态的

**private** **static** SessionFactory *sessionFactory*=**null**;

**private** MySessionFactory(){

}

**static** {

*sessionFactory* = **new** Configuration().configure().buildSessionFactory();

}

**public** **static** SessionFactory getSessionFactory(){

**return** *sessionFactory*;

}

}

# Hibernate中的核心类和接口

## Configuration类

1 负责管理hibernate的配置信息

2 读取hibernate.cfg.xml文件

3 加载hibernate.cfg.xml配置文件中配置的驱动，url，用户名，密码，连接池

4 管理\*.hbm.xml对象关系映射文件

Configuration cf = new Configuration.configure();

### Hibernate.cfg.xml文件

1 该文件主要用于指定各个参数，是hibernate的核心文件

2 指定了连接数据库的驱动 用户名 密码 url 连接池

3 指定对象关系映射文件的位置

4 可以使用hibernate.propertities文件代替

### 对象关系映射文件

1 建立表和类的映射关系

2 一般放在与其映射的类同一个目录下

3 命名方式一般是 类名.hbm.xml

## SessionFactory接口

会话工厂接口

1 缓存sql语句和某些数据（称为session级缓存）

2 是一个重量级的类，尽量保证一个数据库有一个sessionfactory

Sessionfactory是一个session工厂，而session是一个会话，也就相当于一个connection。重量级和轻量级跟占用资源大小有关，sessionfactory中管理了session，存放了大量预定义的sql语句和映射元数据，需要很大的缓存。一般，一个数据库对应于一个sessionfactory。

Sessionfactory接口负责初始化hibernate，sessionfactory充当数据存储源的代理，并负责创建session对象。

Sessionfactory是线程安全的，多个线程可以同时访问一个。一般在启动的时候创建。大部分的实例变量都是final类型的

Session是非线程安全的，大部分的实例变量是非final的。可以使用ThreadLocal将session和当前线程绑定在一起。当session作为类的成员变量时，会存在多个线程同时操作一个session对象，对数据库进行crud操作，产生混乱。这样会产生线程不安全问题。

### 获取session的两个方法

openSession():每次获取一个新的session

getCurrentSession():获取和当前线程绑定的session。也就是说，在同一个线程中，获取到的session是同一个session，这样利于事务控制。

一个http请求，就是一个线程，这样就可以使用getCurrentSession()在不同的service中执行一个事务操作。

通过getCurrentSession()获取的session在事务提交之后，会自动关闭，通过openSession()获取到的session则必须手动关闭。

如果通过getCurrentSession()获取的session，在进行查询时，需要事务提交。

### 本地事务和全局事务

本地事务：在数据库的本地并且限制在单个进程内的事务。本地事务不涉及多个数据来源。

全局事务：资源管理器管理和协调的事务，可以跨越多个数据库和进程。

## Session接口

1 session一个实例代表与数据库的一次操作（可以是crud组合）

2 session实例要通过sessionFactory获取，用完后需要关闭

3 session是线程不安全的，因此要保证在同一个线程中使用，可以使用getCurrentSession()

4 session可以看作是持久化管理器，是与持久化操作相关的接口

# Get和load区别

1 Get方法直接返回实体类，如果查不到数据，则返回null。Load会返回一个实体代理对象（当前这个对象可以自动转化为实体对象），只有当代理对象被调用时，才去数据库中查找。

2 load先到缓存中去查，如果没有，则返回一个代理对象（并不马上到数据库中去查找），等后面使用这个代理对象操作的时候，才去数据库中查询。Load在默认情况下支持延迟加载。

3 get先到缓存中去查，如果没有就到数据库中去查（即马上发出sql）。如果确定数据库中有这个对象就用load，不确定就用get。

4 load查询时，如果数据不存在，会抛出ObjectNotFoundException异常。Get查询时，如果数据不存在，会返回null。

一级缓存是必须有的，二级缓存配置之后才有。

当使用load查询数据时，首先会查找一级缓存（session级缓存），没有的话，则进入二级缓存(在内存和磁盘之间)查找，如果还没有数据，就停止查找，返回的是一个代理对象。当需要使用对象的某些属性时，再去数据库中查找。同时会把查询到的数据放入到二级缓存中。

而get（）方法会依次进入到一级缓存/二级缓存（如果存在）中查找，如果都没有，则去查询数据库，并把数据存放到缓存中。（减少了对数据库的查询次数）

# 懒加载

懒加载减少了程序本身因为与数据库的频繁交互造成的处理速度缓慢。

Employee employee1 = (Employee) session.load(Employee.**class**, 1);

System.***out***.println(employee1.getClass());

输出的结果是class com.wy.domain.Employee\_$$\_javassist\_0得到的是代理对象。这个对象所属的类是Employee的子类，是hibernate自动实现的一个子类。因此domain对象不能是final的，如果是final，那么就不可以继承，当然也不可以产生代理对象，也就不能实现懒加载了。如果不使用懒加载，就可以把domain对象设计成final。代理类具有查询数据库的能力。当session没有关闭，就可以调用employee1.getName()方法，去查询数据库并返回相应的数据；如果session已经关闭了，再去调用上述方法，会报org.hibernate.LazyInitializationException异常。如果一定要在session关闭之后再使用代理对象，hibernate中定义了一个初始化代理对象的方法initialize()，通过该方法即可将代理对象初始化。(注：在使用代理对象的getId()和getClass()方法时，并不会抛出不能初始化异常，因为这两个属性不用查询数据库)

当去查询一个对象时，在某人情况下，返回的只是该对象的普通属性，当用户使用对象属性时，才会向数据库发出再一次的查询。

解决方法： 1 显示初始化Hibernate.initized(代理对象);

2 lazy=false

3 opensessioninview

实体的集合属性和所关联的实体默认为懒加载。

Hibernate 对于 Set 属性延迟加载关键就在于 PersistentSet 实现类。在延迟加载时，开始 PersistentSet 集合里并不持有任何元素。但 PersistentSet 会持有一个 Hibernate Session，它可以保证当程序需要访问该集合时“立即”去加载数据记录，并装入集合元素。与 PersistentSet 实现类类似的是，Hibernate 还提供了 PersistentList、PersistentMap、PersistentSortedMap、PersistentSortedSet 等实现类，它们的功能与 PersistentSet 的功能大致类似。熟 悉 Hibernate 集合属性读者应该记得：Hibernate 要求声明集合属性只能用 Set、List、Map、SortedSet、SortedMap 等接口，而不能用 HashSet、ArrayList、HashMap、TreeSet、TreeMap 等实现类，其原因就是因为 Hibernate 需要对集合属性进行延迟加载，而 Hibernate 的延迟加载是依靠 PersistentSet、PersistentList、PersistentMap、PersistentSortedMap、 PersistentSortedSet 来完成的——也就是说，Hibernate 底层需要使用自己的集合实现类来完成延迟加载，因此它要求开发者必须用集合接口、而不是集合实现类来声明集合属性。

Hibernate 对集合属性默认采用延迟加载，在某些特殊的情况下，为 <set.../>、<list.../>、<map.../> 等元素设置 lazy="false"属性来取消延迟加载。

# Opensession和getcurrentsession区别

1 Opensession每次创建session都是新的session实例对象，而getcurrentsession创建的session会绑定到当前线程中，在事务结束之前使用的是当前线程中的session。

2 在事务进行commit或者rollback时，getcurrentsession创建的session会自动关闭，而Opensession创建的session需要手动关闭。

（**openSession**创建session时autoCloseSessionEnabled参数为false，即在事物结束后不会自动关闭session，需要手动关闭，如果不关闭将导致session关联的数据库连接无法释放，最后资源耗尽而使程序宕掉。**getCurrentSession**创建session时autoCloseSessionEnabled，flushBeforeCompletionEnabled都为true，并且session会同sessionFactory组成一个map以sessionFactory为主键绑定到当前线程。）

在使用spring的gethibernateTemplate就不需要考虑事务的管理和session关闭问题。

3 使用getcurrentsession需要在hibernate.cfg.xml文件中配置：

本地事务（jdbc事务）

<property name=”hibernate.current\_session\_context\_class”>thread</property>

全局事务(jta事务) 多个数据库资源或事务资源

<property name=” hibernate.current\_session\_context\_class”>jta</property>

在 SessionFactory 启动的时候， Hibernate 会根据配置创建相应的 CurrentSessionContext ，在 getCurrentSession() 被调用的时候，实际被执行的方法是 CurrentSessionContext.currentSession() 。在 currentSession() 执行时，如果当前 Session 为空， currentSession 会调用 SessionFactory 的 openSession 。所以 getCurrentSession() 对于 **Java EE** 来说是更好的获取 Session 的方法。

**final** **public** **class** HibernateUtil {

**private** **static** SessionFactory *sessionFactory* = **null**;

//使用线程局部模式

**private** **static** ThreadLocal<Session> *threadLocal* = **new** ThreadLocal<Session>();

**private** HibernateUtil(){

}

**static** {

*sessionFactory* = **new** Configuration().configure().buildSessionFactory();

}

//获取全新的session

**public** **static** Session openSession(){

**return** *sessionFactory*.openSession();

}

//获取和线程关联的session

**public** **static** Session getCurrentSession(){

Session session = *threadLocal*.get();

//判断是否得到

**if**(session==**null**){

session = *sessionFactory*.openSession();

//把session对象设置到threadLocal里面，相当于该session已经和线程绑定

*threadLocal*.set(session);

}

**return** session;

}

}

# Hibernate中的查询

## Query接口

Query query = session.createQuery("from Employee where name ='王元'");

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

在hibernate中使用hql语句查询时，返回的是对象。

如果检索的是部分属性，返回的是一个包含这些属性的对象数组，不建议使用。

Hibernate中建议把整个对象的属性都查取出来。而在jdbc中查询时，需要什么字段就去取出什么字段。

### UniqueResult()

Student s=(Student) session.createQuery("from Student where sid='20050003'").uniqueResult();

检索最多只有一个对象的时候，使用uniqueResult()方法。

### Distinct

使用distinct关键字对查询的结果去重

### \*between and

List list=session.createQuery("select distinct sage,ssex,sname from Student where sage between 20 and 22").list();

for(int i=0;i<list.size();i++){

Object [] objs=(Object[]) list.get(i);

System.out.println(objs[0].toString()+" "+objs[1].toString()+objs[2].toString());

}

### \*in /not in

List<Student> list=session.createQuery("from Student where sdept in ('计算机系','外语系')").list();

//取出1. for 增强

for(Student s:list){

System.out.println(s.getSname()+" "+s.getSaddress()+" "+s.getSdept());

}

### \*group by

//显示各个系的学生的平均年龄

List<Object[]> list=session.createQuery("select avg(sage),sdept from Student group by sdept").list();

//取出1. for 增强

for(Object[] obj:list){

System.out.println(obj[0].toString()+" "+obj[1].toString());

}

### having

//1.对分组查询后的结果，进行筛选:比如请显示**人数大于3的系名称**

//a. 查询各个系分别有多少学生.

List<Object[]> list=session.createQuery("select count(\*) as c1,sdept from Student group by sdept having count(\*)>3").list();

//取出1. for 增强

for(Object[] obj:list){

System.out.println(obj[0].toString()+" "+obj[1].toString());

}

### 聚集函数

**//1.查询计算机系共多少人?->如果我们返回的是一列数据**

//这时我们的取法是直接取出list->object 而不是 list->Object[]

List<Object[]> list=session.

createQuery("select sage from Student where sdept='计算机系'").list();

//取出1. for 增强

for(Object obj:list){

System.out.println(obj.toString());

}

**2 查询总学分**

List<Object> list = session.createQuery("select sum(grade) from Stucourse ").list();

for(Object objs : list) {

System.out.println(objs.toString());

}

**3.查询选修11号课程的最高分和最低分.**

**List<Object[]> list=session.**

**createQuery("select 11,max(grade),min(grade) from Studcourse where course.cid=11").list();**

**//取出1. for 增强**

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

**System.out.println(obj[0].toString()+" max="+obj[1].toString()+" min="+obj[2].toString());**

**}**

**4 显示各科不及格的学生名字 科目 分数**

List<Object[]> list = session.createQuery("select student.sname,course.cname,grade from Stucourse where grade<60").list();

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

System.***out***.println(objs[0].toString()+" "+objs[1].toString()+" "+objs[2].toString());

}

查询语句等价于：

**select s.sname,c.cname,sc.grade from student as s,course as c,stuCourse as sc where s.sid=sc.sid and c.cid=sc.cid and sc.grade>60;**

**//计算各个科目不及格的学生数量.(学生练习!)**

注意是从哪个表选择的。（选择的那张表与其他的表都有联系）

List<Object[]> list=session.

createQuery("select count(\*),student.sdept from Studcourse where grade<60 group by student.sdept").list();

//取出1. for 增强

for(Object[] obj:list){

System.out.println(obj[0].toString()+" "+obj[1].toString());

}

### 模糊查询

From Student where name like ‘刘%’;

### 分页查询

//分页函数

**private** **static** **void** showResultByPage(**int** pageSize){

//设置分页变量

**int** pageNow=1;

**int** pageCount=1;//总页数--计算

**int** rowCount =1 ;//记录总数 这个需要查询

Session session = **null**;

Transaction ts = **null**;

**try** {

session = HibernateUtil.*getCurrentSession*();

ts= session.beginTransaction();

rowCount = Integer.*parseInt*(session.createQuery("select count(\*) from Student").uniqueResult().toString());

pageCount = (rowCount-1)/pageSize + 1;

//循环显示每页信息

**for**(**int** i=1;i<=pageCount;i++){

System.***out***.println("-----第"+i+"页------");

List<Student> list = session.createQuery("from Student").setFirstResult((i-1)\*pageSize).setMaxResults(pageSize).list();

**for**(Student s : list) {

System.***out***.println(s.getSname()+"--"+s.getSdept());

}

}

ts.commit();

} **catch** (Exception e) {

e.printStackTrace();

**if**(ts!=**null**){

ts.rollback();

}

**throw** **new** RuntimeException(e.getMessage());

} **finally** {

**if**(session !=**null** && session.isOpen()){

session.close();

}

}

}

### 参数绑定

防止sql注入漏洞。

参数绑定有两种形式

**Query q=session.createQuery(from Student where sdept=:dept and sage>:age)**

如果我们的参数是 :冒号形式给出的，则我们的参数绑定应当这样:

List<Student> list=session.createQuery("from Student where sdept=:a1 and sage>:sage")

.setString("a1", "计算机系").setString("sage", "2").list();

还有一种形式:

**Query q=session.createQuery(from Student where sdept=? and sage>?)**

**如果我们的参数是以 ? 形式给出的则，参数绑定应当:**

List<Student> list=session.createQuery("from Student where sdept=? and sage>?")

.setString(0, "计算机系").setString(1, "2").list();

## Criteria接口

Criteria接口是比hql更加面向对象方式的查询。

Criteria cri=session.createCriteria(Employee.class).

setMaxResults(2).addOrder(Order.desc("id") );

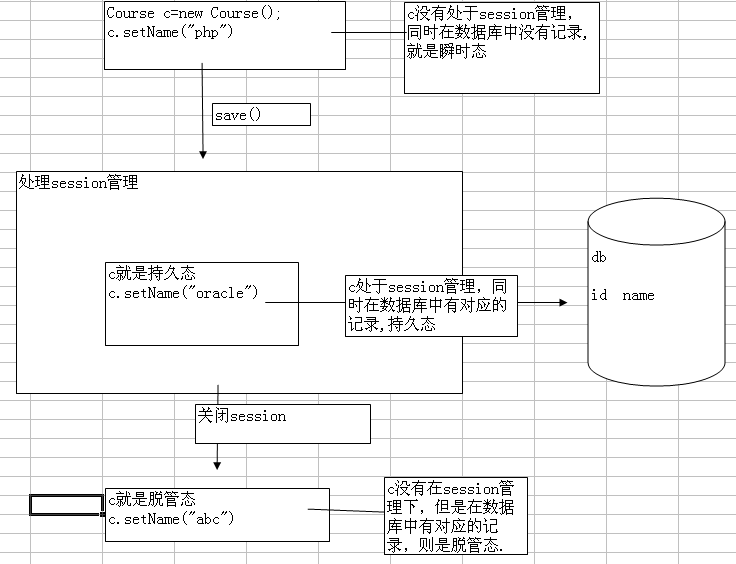
List<Employee> list=cri.list();

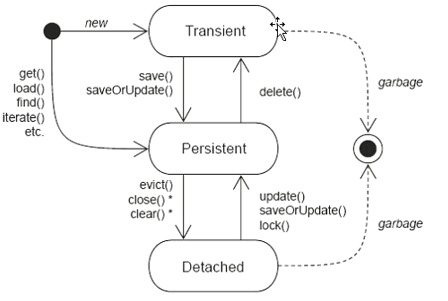
# Hibernate中对象的状态

**瞬时(transient)：**数据库中没有数据与之对应，超过作用域会被JVM垃圾回收器回收，一般是new出来且与session没有关联的对象。

**持久(persistent)：**数据库中有数据与之对应，当前与session有关联，并且相关联的session没有关闭，事务没有提交。持久对象状态发生改变，在事务提交时会影响数据库。

**脱管/游离(detached)：**数据库中有数据与之对应，但当前没有session与之关联，脱管对象状态发生改变，hibernate不能检测到。





# Hibernate的映射文件创建数据库

hibernate开发的三种方式中的:

编写domain object + 映射文件 ------> 创建出对应的数据库,

这里我们说明如果要自动的创建出对应的数据库，需要做配置(hibernate.cfg.xml).

<property name="hbm2ddl.auto">create</property>

这里有四个配置值: create , update , create-drop, validate

create : 当我们的应用程序加载hibernate.cfg.xml [ new Configuration().config(); ]就会根据映射文件，创建出数据库, 每次都会重新创建， 原来表中的数据就没有!!!

(只需要加载这个文件即可）

update: 如果数据库中没有该表，则创建，如果有表，则看有没有变化，如果有变化，则更新.

create-drop: 在显示关闭 sessionFactory时，将drop掉数据库的schema

validate: 相当于每次插入数据之前都会验证数据库中的表结构和hbm文件的结构是否一致

# 缓存

缓存的作用是提高性能，减少与数据库的交互。相当于一个Map。

一级缓存是session级缓存，当session关闭之后，就会自动销毁。不需要配置就可以使用。它本身没有保护机制，不能控制缓存的数量，所以大批量操作数据时可能造成内存溢出，可以使用evict clear方法清除缓存中的内容。

在查询数据时，首先向一级缓存中取数据，如果有，则不去查询数据库，否则会去查询数据库。

save,update,saveOrUpdate,load,get,list,iterate,lock会向一级缓存中存数据。（save时会把数据放在一级缓存中，commit之后才会向数据库中插入数据）

get / load 会首先从一级缓存中取，如没有.再有不同的操作[get 会立即向数据库发请求，而load 会返回一个代理对象，直到用户真的去使用数据，才会向数据库发请求]

从上面的案例，我看出 query.list() query.uniueResut() 不会从一级缓取数据! 但是query.list 或者query.uniqueResult() 会向一级缓存放数据的.

二级缓存是sessionfactory级缓存。因为一级缓存生命周期短，所以需要二级缓存。

二级缓存需要配置，是交给第三方去处理，如HashTable,OSCache,EHCache，二级缓存可能在内存或者硬盘上。首先去一级缓存中查找，查找不到，再去二级缓存中查找。

<!-- 启动二级缓存 -->

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

<!-- 指定使用哪种二级缓存 -->

<property name="cache.provider\_class">org.hibernate.cache.OSCacheProvider</property>

<!-- 指定哪个domain启用二级缓存

特别说明二级缓存策略:

1. read-only

2. read-write

3. nonstrict-read-write（bbs被浏览了多少次）

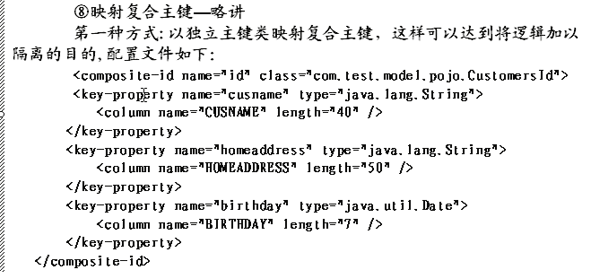
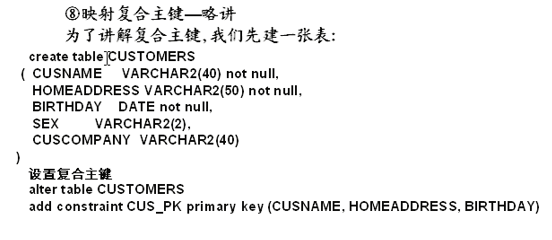
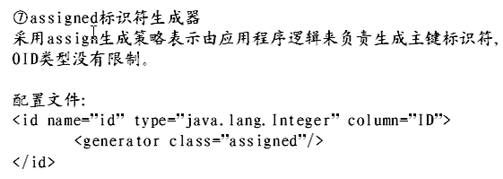
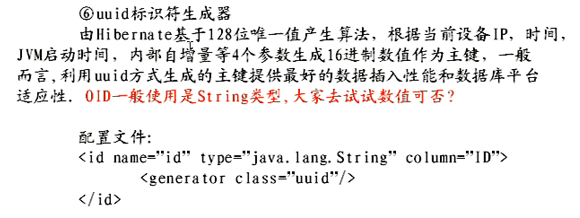
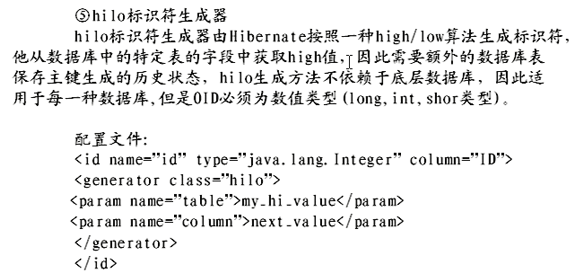
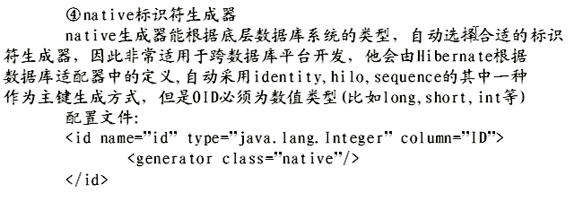
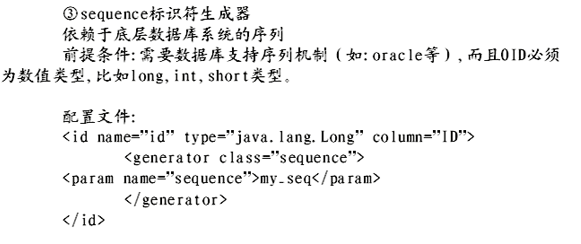
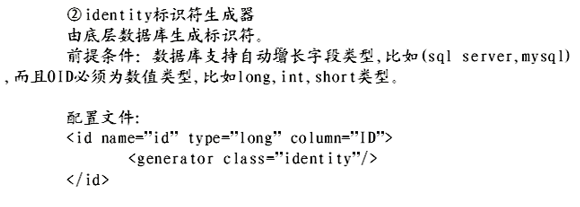
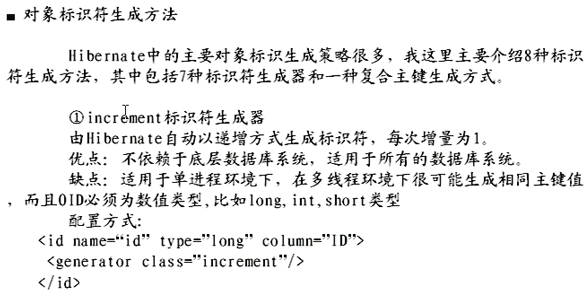
4. transcational

-->

<class-cache class="com.hsp.domain.Student" usage="read-write"/>

通过SessionFactory.getStatistics()查看命中率。

# 主键增长策略



1. foreign

在one-to-one的关系中，由另一张表的主键(Person) 来决定 自己主键/外键

# Hibernate不适合的场景

Hibernate不适合OLAP（联机分析处理），以查询分析数据为主的系统。

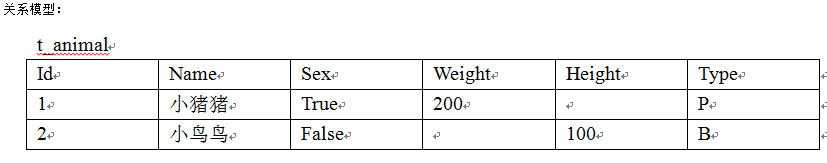
适合OLTP（联机事务处理）

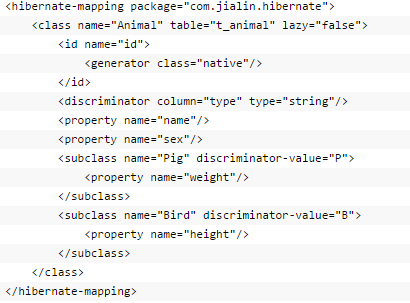
对于数据量大，性能要求高的系统，不适合使用hibernate。主要用于事务操作比较多的项目。如crm，oa等。

# 继承映射

1 一张总表

整个继承结构一张表，要求字段不能有非空约束，而且要添加一个字段表示类型。





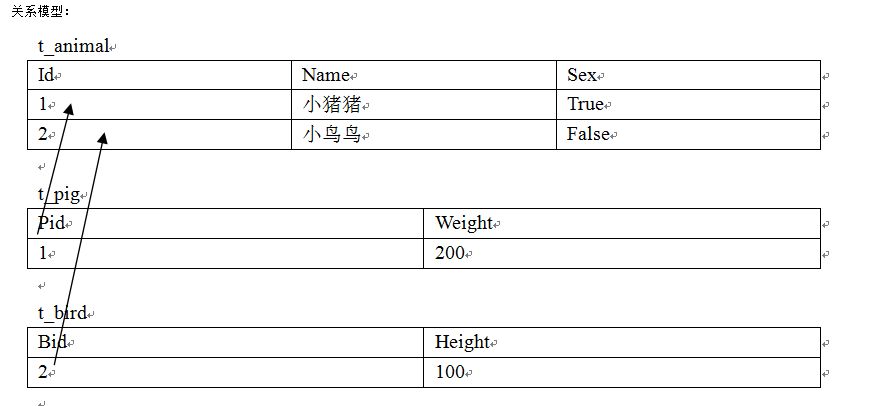
父类用<class>标签定义

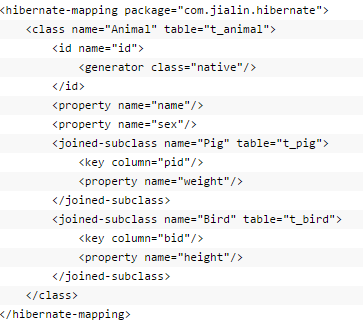
子类用<subclass>标签定义 注意：subclass可以被class标签包含(这种包含关系表明了类之间的继承关系)，也可以与class标签平行。平行时，需要在subclass标签中添加extends属性，里面的值是父类的全路径名称。<subclass name=”” discriminator=””></subclass>

在父类中定义一个discriminator，指定这个区分的字段的名称和类型 <discriminator column=”” type=””/>

鉴别值在存储时，hibernate会自动存储，在加载时，会根据鉴别值取得相关的对象。

2 每个类一张表

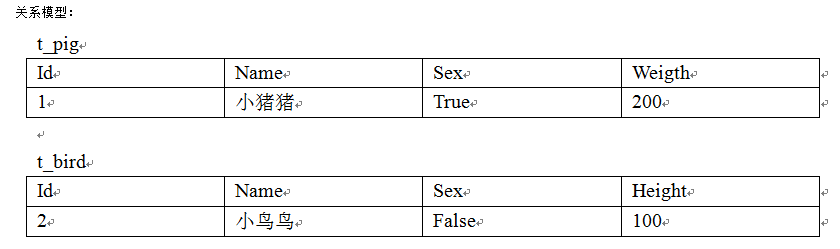




这种策略是使用joined-subclass标签来定义子类的。父类、子类，每个类都对应一张数据库表。  
在父类对应的数据库表中，实际上会存储所有的记录，包括父类和子类的记录；在子类对应的数据库表中，这个表只定义了子类中所特有的属性映射的字段。子类与父类，通过相同的主键值来关联。

实现这种策略的时候，有如下步骤：  
1、父类用普通的<class>标签定义即可  
2、父类不再需要定义discriminator字段  
3、子类用<joined-subclass>标签定义，在定义joined-subclass的时候，需要注意如下几点：  
\*Joined-subclass标签的name属性是子类的全路径名  
\*Joined-subclass标签需要包含一个key标签，这个标签指定了子类和父类之间是通过哪个字段来关联的。如：<key column=”PARENT\_KEY\_ID”/>，这里的column，实际上就是父类的主键对应的映射字段名称。  
\*Joined-subclass标签，既可以被class标签所包含（这种包含关系正是表明了类之间的继承关系），  
也可以与class标签平行。 当Joined-subclass标签的定义与class标签平行的时候，  
需要在Joined-subclass标签中，添加extends属性，里面的值是父类的全路径名称。  
子类的其它属性，像普通类一样，定义在joined-subclass标签的内部。

3 每个子类一张表



这种策略是使用union-subclass标签来定义子类的。每个子类对应一张表，而且这个表的信息是完备的，  
即包含了所有从父类继承下来的属性映射的字段（这就是它跟joined-subclass的不同之处，joined-subclass定义的子类的表，只包含子类特有属性映射的字段）。

实现这种策略的时候，有如下步骤：  
1、父类用普通<class>标签定义即可  
2、子类用<union-subclass>标签定义，在定义union-subclass的时候，需要注意如下几点：  
\*Union-subclass标签不再需要包含key标签（与joined-subclass不同）  
\*Union-subclass标签，既可以被class标签所包含（这种包含关系正是表明了类之间的继承关系），  
也可以与class标签平行。 当Union-subclass标签的定义与class标签平行的时候，需要在Union-subclass标签中，  
添加extends属性，里面的值是父类的全路径名称。  
\*子类的其它属性，像普通类一样，定义在Union-subclass标签的内部。这个时候，虽然在union-subclass  
里面定义的只有子类的属性，但是因为它继承了父类，所以，不需要定义其它的属性，在映射到数据库表的时候，  
依然包含了父类的所有属性的映射字段。

注意：在保存对象的时候id不能重复（不能使用数据库的自增方式生成主键）