# Hibernate

## 搭建环境

### 软件分层

从上往下依次是：表示层（Action，JSP），业务逻辑层（Service），持久化层（Dao），数据库层，上层组件会访问下层组件的API，而下层组件不应该依赖上层组件

### 准备工作

拷贝hibernate解压出来的jar包

request里全部包，hibernate3.jar log4j.jar slf4j-log4j.jar

hibernate3.jar:

　　Hibernate的库，必须使用的jar包

dom4j.jar:

dom4j是一个Java的XML API，类似于jdom，用来读写XML文件的

commons-collections.jar：

Apache Commons包中的一个，包含了一些Apache开发的集合类，功能比java.util.\*强大。必须使用的jar包。

jta.jar：

Java Transaction API JTA规范，当Hibernate使用JTA的时候需要，不过App Server都会带上，所以也是多余的。

### 创建数据库

### 拷贝日志文件log4j.properties

改其中的内容log4j.logger.org.hibernate=info

### 创建一个持久化类

符合JavaBean的规范，包含一些属性，以及与之对应的getter 和setter方法，hibernate要求持久化类必须要有一个不带参数的构造方法，访问级别必须是public或protected，而不能是default或private

### O-R映射，拷贝映射文件 .hbm.xml

project\tutorials\eg\src\main\resources\org\hibernate

配置映射文件：每个实体类对应一个映射文件

数据库表必须要有主键

<hibernate-mapping>

<class name=”member.Member” table=”member”>

<id name=”id”>

<generator class=”native”/>

</id>

<property name=”username”>

<column name=”username” length=”20” />

</property>

</class>

</hibernate-mapping>

<generator class="native"/>表示根据数据库自增的数据

<property>的name告诉hibernate使用持久化类的哪个getter和setter方法，column属性指对应表中哪个字段

<hibernate-mapping>一些常见属性：

default-cascade 级联风格

default-access 访问属性的策略 field property ClassName

default-lazy 懒加载true false

package 指定包前缀

### 拷贝hibernate配置文件 hibernate.cfg.xml

<hibernate-configuration>

<session-factory>

<!-- 数据库链接的相关配置 -->

<!—连接数据库的hibernate的URL-->

<property name="hibernate.connection.url">

jdbc:mysql://localhost/hibernate

</property>

<property name="hibernate.connection.driver\_class">

com.mysql.jdbc.Driver

</property>

<property name="hibernate.connection.username">root

</property>

<property name="hibernate.connection.password">123456

</property>

<!-- 数据库方言 -->

<property name="hibernate.dialect">org.hibernate.dialect.

MySQLDialect

</property>

<!-- 自动打印出SQL语句 -->

<property name="hibernate.show\_sql">true</property>

<!-- 自动创建数据库表 -->

<property name="hibernate.hbm2ddl.auto">update

</property>

<!-- 映射文件列表 -->

<mapping resource="com/hibernate/User.hbm.xml"/>

</session-factory>

</hibernate-configuration>

## 基础知识

### 方法介绍

用测试单元进行测试，先创建一个source folder：Test ，然后选中持久化类 选择JUnit Test Case 把测试类放在test下

添加：session.save(user)

更新：session.update(user)

只更新一个字段时，会对该记录的每个字段都更新一遍

单个查询：session.get(User.class,21) 第一个参数是类名，第二个是id

hibernate是对整个对象进行操作，只根据ID不能删除一条记录

多查：session.createQuery(“from User”).list()

### 对语法的理解

//默认读取类路径根目录下的hibernate.cfg.xml配置文件

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

//创建SessionFactory

SessionFactory factory = cfg.buildSessionFactory();

//创建hibernate session

Session session = factory.openSession();

//开启事务

session.beginTransaction();

//提交事务

session.getTransaction().commit();

//出现异常事务回滚

session.getTransaction().rollback();

//关闭session

session.close();

Configuration接口负责配置并启动hibernate

创建Sessionfactory

配置hibernate两种方式:

第一种是属性文件hibernate.properties

第二种是配置hibernate.cfg.xml调用方法：

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

SessionFactory接口负责初始化hibernate，充当数据库存储源代理，并负责创建Session对象

从SessionFactory(会话工厂)获得Session实例。SessionFactory是线程安全的，采用单例创建，SessionFactory对象对应一个数据库，hibernate配置文件实际上是对SessionFactory的配置

session是线程不安全的，它与web层的HttpSession没有任何关系

connection是连接，生命周期较长，如果有连接池的话，服务器的开和关是connection开始和结束的标志

session是一次会话，生命周期是请求响应周期，一个session可以包含多个transaction

Transaction是事务，生命周期比session还短

一般每个数据库对应一个SessionFactory，每次请求响应周期对应一个session对象，每个session对象对应一个transaction对象

### 时间格式化

”年”用小写的y

bith = new

SimpleDateFormat("yyyy-MM-dd").parse("1992-10-19");

format 和 parse的区别：

parse()返回的是一个Date类型数据，format返回的是一个StringBuffer类型的数据

SimpleDateFormat中的parse方法可以把String型的字符串转换成特定格式的date类型

SimpleDateFormat中的format方法可以把Date型的字符串转换成特定格式的String类型

### Session的生命周期

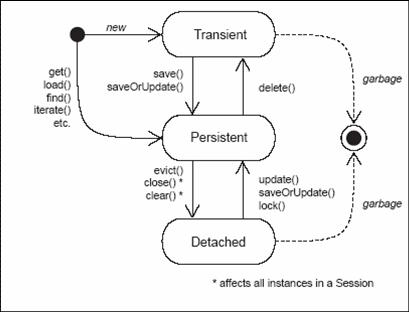
首先解释一下持久化对象：

瞬时对象(Transient Objects)：使用new 操作符初始化的对象不是立刻就持久的。它们的状态是瞬时的，也就是说它们没有任何跟数据库表相关联的行为，只要应用不再引用这些对象（不再被任何其它对象所引用），它们的状态将会丢失，并由垃圾回收机制回收。

持久化对象(Persist Objects)：持久实例是任何具有数据库标识的实例。它有持久化管理器Session统一管理，持久实例是在事务中进行操作的——它们的状态在事务结束时同数据库进行同步。当事务提交时，通过执行SQL的INSERT、UPDATE和DELETE语句把内存中的状态同步到数据库中。

离线对象(Detached Objects)：Session关闭之后，持久化对象就变为离线对象。离线表示这个对象不能再与数据库保持同步，它们不再受Hibernate管理。

生命周期：



配置好后，新建一个HibernateUtils.java类：

Java代码

1. package com.mp.hibernate;
3. import org.hibernate.Session;
4. import org.hibernate.SessionFactory;
5. import org.hibernate.cfg.Configuration;
7. public class HibernateUtils {
8. public static SessionFactory factory;
10. static {
11. try {
12. Configuration cfg = new Configuration().configure();
13. factory = cfg.buildSessionFactory();
14. } catch(Exception e) {
15. e.printStackTrace();
16. }
17. }
19. public static SessionFactory getSessionFactory() {
20. return factory;
21. }
23. public static Session getSession() {
24. return factory.openSession();
25. }
27. public static void closeSession(Session session) {
28. if(session != null) {
29. if(session.isOpen()) {
30. session.close();
31. }
32. }
33. }
34. }

这个类就是把对session的基本操作封装起来。

在数据库中新建了hibernate\_session数据库，并且修改hibernate.cfg.xml文件中的配置

在test中建了一个测试类SessionTest.java

首先对session的三个状态进行测试：

Java代码

1. package com.mp.hibernate;

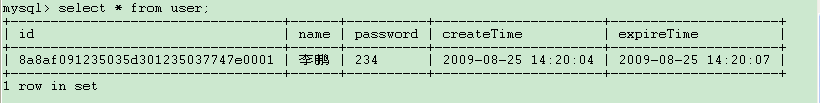
4. import java.util.Date;
6. import org.hibernate.Session;
7. import org.hibernate.Transaction;
9. import junit.framework.TestCase;
11. public class SessionTest extends TestCase {
12. public void testHello() {
13. System.out.println("-----------SessionTest.hello()------------");
14. //throw new RuntimeException();
15. }

18. public void testSave1() {
19. Session session = null;
20. Transaction ts =  null;
21. User user = null;
22. try {
23. session = HibernateUtils.getSession();
24. ts = session.beginTransaction();
26. //Transient状态
27. user = new User();
28. user.setName("哈哈");
29. user.setPassword("234");
30. user.setCreateTime(new Date());
31. user.setExpireTime(new Date());
33. //persistent状态,当属性发生改变的时候， hibernate会自动和数据库同步
34. session.save(user);
36. user.setName("李鹏");
38. //          session.update(user);
39. ts.commit();
41. } catch(Exception e) {
42. e.printStackTrace();
43. ts.rollback();
44. } finally {
45. HibernateUtils.closeSession(session);
46. }
48. //detached状态
49. user.setName("张三");
50. try {
51. session = HibernateUtils.getSession();
52. session.beginTransaction();
54. //persistent状态
55. session.update(user);
56. session.getTransaction().commit();
57. } catch (Exception e) {
58. // TODO: handle exception
59. e.printStackTrace();
60. session.getTransaction().rollback();
61. } finally {
62. HibernateUtils.closeSession(session);
64. }
65. }
67. }

进入debug模式调试，查看user 的变化过程

在执行了save方法后，id被创建，也就意味着user对象被创建，但是hibernate并没有执行插入语句

之后修改user的属性也是可以的

因为在提交事务后可以看到hibernate执行了两个语句，一个insert另外一个就是update；此时也能看到数据库中已经插入记录：  


update主要是由于使用了user对象的属性。

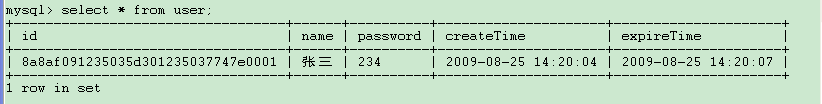
sava()操作使得它进入persistent状态,当属性发生改变的时候，hibernate会自动和数据库同步

之后的commit()执行时，进行了"脏数据检查"操作。

紧接着又关闭了session，在进入了detached状态（对象不存在于session中）。

执行到update 又转入到persistent状态。

之后提交事务，也能完成数据库的更新。

查看数据库：  


### 事务的概念

原子性：一个transaction是一个程序片段，在程序中如果出任何差错，则全部回退

一致性:回退前后要一致

隔离性：B事务一定要等A事务提交后才能进行

持久性：一旦数据库发生变化，则变化保持持久

并发问题：丢失跟新、脏读、非重复读、幻像读

在hibernate中设置隔离级别，级别越高越严格

在hibernate.cfg.xml

<session-factory>

<property name=”hibernate.connection.isolation”>4</property>

< /session-factory >

### 对象相关操作

#### 对象状态

映射文件对应的实体类创建出来的对象有三种状态：

瞬时对象：内存有，数据库没有，可以通过save() update()转化成持久对象，只是改变的状态，当commit的时候才发出语句

持久化对象：内存有，数据库有，而且通过session管理，两者自动同步，虽然不执行update() 和save()方法，仍自动同步，当执行delete()将变成瞬时对象，当执行session.close() clear() evict() 持久对象将变成离线对象

离线对象：内存有，数据库有，session已关闭，两者不能同步

delete()删除

save()会把瞬时对象转化成持久化对象

get()出来的对象处于持久化状态，对持久化对象的改变无须手动update，hibernate会自动同步，如果状态没有改变，不会发出update语句

#### 获取对象

懒加载，即延迟加载，延迟发出sql语句

默认lazy=true 即支持懒加载

get()会立即发出查询语句

load()会延迟发出查询语句（在getId()之外的其他语句（不能是setId()）时发出查询语句），load出来的是一个代理对象

session必须打开，延迟机制才有效（懒加载异常）

get操作 get(Person.class,1)

get操作会马上发出sql语句查询相应的对象

get加载的对象如果不存在会报NullPointException异常

get操作返回的是User类型的对象

get只在一级缓存中查找，如果没有就直接调用查询语句

load操作

load操作将会应用懒加载机制

load对象如果不存在会抛出ObjectNotFoundException（因为返回的是一个代理对象，如果数据库没有对应对象的记录，当然代理会出错）

load操作返回的是User类型的代理类型

load可以在一、二级缓存中充分查找现有数据

#### 保存对象

用save() 或 persist()保存对象，当commit()的时候才会发出SQL语句

二者区别：persist() 无返回值，而save()返回保存对象的标识符

persist只能保存瞬时和持久化对象，如果保存其他状态的对象会出异常，而save可以保存任何状态的对象

#### 更新对象

session.update(user)

#### 删除对象

session.delete(user) 当commit()的时候才会发出SQL语句

### hibernate-mapping元素

#### hibernate-mapping元素

default-access(默认为property) 定义访问所有属性的策略

default-lazy(默认为true) 指定了未明确注明lazy的属性

#### class元素

name 持久化类的全限定名

table对应的数据库表名

where 指定一个附加的SQL WHERE条件，在抓取这个类的对象时会一直增加这个条件

batch-size 批量抓取的数量，默认为1

lazy定义懒加载

rowid hibernate可以使用额外的字段rowid实现快速更新

abstract 抽象超类

#### ID生成策略

assigned策略，ID不自增，需要手动赋值，缺省值为0，只能用缺省值插入一条记录

native针对整型 在mysql里是identity策略（依赖数据库生成）

helo 针对整型 （高低位算法生成的标识符，只有在一个特定的数据库中是唯一的，自动生成一张表）

uuid 针对字符串（hibernate自己生成，在网络中是唯一的）

sequence 针对整型 在oracel中使用序列

foreign使用另外一个相关联的对象的标识符。它通常和 <one-to-one> 联合起来使用。

#### property 元素

name 实体类的属性名

column 实体类中属性名对应的数据库表中的字段名

type: 图片实体类定义为byte[] type=binary， 长文本 type=text， 时间 type=date

insert，update默认为true，设置为false表示添加，更新时忽略本项

lazy默认为false

unique 添加唯一的约束

not-null 添加非空约束

access access=field指通过field访问实体类的属性，绕过setter和getter ，利用反射机制

access=property 通过setter 和getter访问，两个值要对应field和property的name

### 关联映射

#### 多对一单向关联映射

需要定义column ，不需要定义type，type由关联的对象字段类型决定

更新的时候，如果是new出来的对象，容易丢失原来的关联，需要把更新对象的关联也提交回去，如果get出来在更新就不会丢失

一个持久化对象不能引用一个瞬时对象

Group g = new Group();

g.setName(“朋友”);

Person p = new Person ();

p.setName(“zhangsan”);

p.setGroup(g);

session.save(p)

**此时会出TransientObjectException**

即g为瞬时对象，p在save()之后变成了持久化对象，此时持久化对象不能引用瞬时对象

在many-to-one 标签中加上cascade(级联)=”all”意思就是你在对p进行操作的时候会同时对g操作

<many-to-one name="group" class=” ” column="gid"

cascade="all"/>

class指定关联类的路径，name指定Person类中被关联类的属性名

cascade的属性：

none 默认值，在操作当前对象时忽略其他关联对象

save-update 当通过save() update()及saveOrupdate() 操作对象时，关联瞬时对象

delete 通过delete()操作对象时关联到瞬时对象

all 通过save，update，delete，evict ，lock 都会关联到瞬时对象

#### 一对多单向关联映射

在Group实体类定义一个集合

private Set<Person> persons;

在Group的hbm.xml里需要对应的配置

<set name=”persons” lazy=”true/false/extra”>

//lazy=true 会把整个集合元素加载到内存中，=extra 不会把整个集合加载到内存中，使用的时候才加载，建议使用lazy=extra，order-by=”id”按照ID排序

//在person表中每个记录加一个字段g\_id 表示group的Id

<key column=”g\_id”></key>

//指出Group关联的是哪个类

<one-to-many class=”com.hebernate.Person”/>

</set>

class标签里的lazy控制加载group时是否懒加载，set标签里的lazy控制集合是否懒加载，二者不冲突

尽量避免在集合里管理对象

#### 一对多双向关联映射

在1的一端inverse=true 表示只能在对方一端维护关联关系（即在多的一端维护），默认是false，表示在两端都能维护

session.flush()强制hibernate 发出语句，而不是等到事务提交时才发出

tree练习中一对多，多对一都写在同一个.hbm.xml中

1的一端即student.hbm.xml

<set name=”scores” cascade=”save-update” inverse=”true”>

<key>

<column name=”sid”>//表示给对方加一个sid字段

</key>

<one-to-many class=”score.Score” />

</set>

多的一端即Score.hbm.xml

<many-to-one name=”student” class=”student.Student”

cascade=”save-update” >

<column name=”sid” />//定义一个记录Student ID 的字段，与set里的sid一致

</many-to-one>

专家模式

private Set<Person> persons;

public Group addPerson(Person p){

if (persons == null){

persons = new hashSet<Person>()

}

persons.add(p);

return this; //返回当前对象，Group group

}

#### 一对一单向关联映射

**外键关联**

在score.hbm.xml中

<many-to-one name=”student” class=”student.Student”

cascade=”all” column=”sid” unique=”true”>

unique=”true”用于表示多的一端也必须唯一，在多的一端增加了唯一约束，即成为单向一对一

**主键关联**

在score.hbm.xml中

基于主键关联的持久化类不能有自己的主键生成策略，它的主键由关联类负责生成

<id name=”id”>

<generator class=”foreign”>

<param name=”property”>student</param>

</generator>

<id>

<generator class=”foreign”>表名根据关联类Student生成Score类主键id，param 即在Score类定义的Student类型的属性student

<one-to-one name=”student” constrained=”true” />

constrained=”true”表明Score类主键由Student生成

#### 一对一双向关联映射

**外键关联**

在score.hbm.xml中

因为外键关联，所以主键不再作为外键，主键的生成策略不再需要关联类生成

<id name=”id”>

<generator class=”native” />

</id>

<many-to-one name=”student” class=”student.Student”

cascade=”all” column=”sid” unique=”true”>

双向一对一关联为什么要用many-to-one 而不使用

one-to-one 呢？

many-to-one 标签可以设置外键的column，而one-to-one不能，此处必须用many-to-one，否则无法建立外键关联

在Student.hbm.xml中

<one-to-one name=”score” cascade=”all”

property-ref=”student”/>

property-ref 指明关联类的一个属性，这个属性将会和外键相对应，如果没指定，会使用关联类的主键，因此在基于主键关联的中没有指定该参数

**主键关联**

本对象与目标对象id保持一致

在score.hbm.xml中

<id name=”id”>

<generator class=”foreign”>

<param name=”property”>student</param>

</generator>

<id>

<one-to-one name=”student” constrained=”true” />

在Student.hbm.xml中

<one-to-one name=”score” cascade=”all” />

#### 多对多关联映射

生成三个表

Person端：

<set column=”roles” table=”t\_person\_role”>

<key column=”personid”></key>

<many-to-many class=”com.hibernate.Role”

column=”roleid”></many-to-many>

</set>

Role端：

<set column=”persons” table=”t\_person\_role”>

<key column=”roleid”></key>

<many-to-many class=”com.hibernate.Person”

column=”personid”></many-to-many>

</set>

把多对多拆分成多对一，中间表为多，两边是一，在中间表维护

需要三个配置文件：

Role端和Person端都是 <one-to-many>

中间的Person\_Role 是两个<many-to-one>

《分析模式》

#### component映射

定义一个地址类，这个类包含在Person类当中，不需要有主键，只需要一个配置文件

<component name=”home” class=”com.Address”>

<property name=”city” column=”home\_city”>

<property name=”province” column=”home\_province”>

</ component >

### 继承映射

#### 单表继承

子类父类都放在一个表中

父类定义Id，子类继承父类并定义自己的属性

discriminator为区别字段

在Animal.hbm.xml中定义：

<class name=”com.Animal” table=”t\_animal”>

<id name=”id”>

<generator class=”native”>

</id>

<discriminator column=”animalType”></ discriminator >

<property name=”name”>

</class>

子类配置文件定义：

<subclass name=”com.Bird” extends=”com.Animal”>

<property name=”height”></properyty>

</subclass>

查询语句：

List list= session.creatQuery(“from Animal”).list()

Bird b = (Bird)session.get(“Animal.class”,2);

下面的写法会出异常

Bird b = (Bird)session.load(“Animal.class”,2);

因为load会为animal创建动态代理，不会自动找到它的子类型

可以把配置文件写在同一个里，此时subclass不需要写extends

<class name=”com.Animal” table=”t\_animal” type=”char”

discriminator-value=”A” >

<id name=”id”><generator class=”native”></id>

<property name=”name”>

<subclass name=”com.Bird” discriminator-value=”B” >

<property name=”height”></properyty>

</subclass>

</class>

在discriminator标签中可以指定区别字段类型

type=”char”，此时标记字段只能取一个字符

#### 具体表继承

父类、子类拥有自己的表，共性的属性会存在父类表中，子类特有的属性会存在子类自己的表中（子表中只有自己的字段）

<class name=”com.Animal” table=”t\_animal”>

<id name=”id”>

<generator class=”native”>

</id>

<property name=”name”>

<joined-subclass name=”com.Bird” table=”t\_bird”>

<key column=”animalId”></key>

<property name=”height”></properyty>

</ joined-subclass >

</class>

#### 类表继承

父类、子类都拥有自己的表，但是子表中包括所有的字段（父类字段和自己的字段）此时id生成策略不能是native

<class name=”com.Animal” table=”t\_animal”>

<id name=”id”><generator class=”hilo”></id>

<property name=”name”>

<union-subclass name=”com.Bird” table=”t\_bird”>

<property name=”height”></properyty>

</ union-subclass >

</class>

#### 集合的配置

集合如果是实体类，<set>里用one-to-many many-to-many

集合如果是普通类型，<set>里用 composit-element , element

<set name=”addresses” table=”t\_address”>

<key column=”personId”></key>

<composite-element class=”com.hibernate.Address”>

<property name=”city”></property>

</ composite-element >

</set>

<set name=”qqs” table=”t\_qq”>

<key column=”personId”></key>

<element type=”string” column=”qqnumber”></element>

</set>

如果集合类型为list则：

<list name=”qqs” table=”t\_qq”>

<key column=”personId”></key>

<list-index column=”qqIndex”></list-index>

<element type=”string” column=”qqnumber”></element>

</list>

如果是map类型：

<map name=”addresses” table=”t\_address”>

<key column=”personId”></key>

<map-key column=”addressType” type=”string”>

</map-key>

<composite-element class=”com.hibernate.Address”>

<property name=”city”></property>

</ composite-element >

</map>

### hibernate查询语句

#### HQL（hibernate query language）

##### 语法

是面向对象的查询语言，不要涉及表名

大小写敏感，类名、字段名敏感

session.createQuery(“from Person”)

from可以是实体类的类名，也可以是父类型

如java.lang.Object ，from可以是类的全限定名，在查询一个类时，hibernate会自动查询这个类的所有继承类，这就是多态查询

用as指定实体类的别名，select p from Person p where p.name……，as可以省略

##### 索引参数查询

Query query = session. createQuery (“select p from Person p where p.name like ? and p.age= ?”);

query.setParameter(0,“%张%”); //索引从零开始

query.setParameter(1,20);

##### 命名参数查询

Query query = session. createQuery (“select p from Person p where p.name like :pn and p.age= :pa”); //必须加冒号

query.setParameter(“pn”,“%张%”);

query.setParameter(“pa”,20);

##### 投影查询（查询某几项字段）

Query query = session. createQuery (“select p.name,p.age from Person p where p.name like :pn and p.age= :pa”);

此时查询出来的list存的每一项都是一个数组，数组的第一项是name，第二项是age

List list = query.list();

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

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

System.out.println(o[0]);

System.out.println(o[1]);

}

如果只查询一项，则取出来的是对应的类型

特性：

Query query = session. createQuery (“select p.name,p.age from Person p where p.id in (:ids)”);

List ids = new ArrayList();

ids.add(1);

ids.add(2);

ids.add(3);

query.setParameterList(“ids”,ids);//ids不能为null ,empty

query.list()

查询出来的是数组

当确定查出来的结果是零条或一条时，可以用

query.uniqueResult()

##### 调用组函数

max() min() vg() count() sum()

Query query = session. createQuery (“select count(\*) from Person p”);

Long l = (Long)query.uniqueResult(); //返回长整型

##### 给结果排序

from Admin a order by a.level

##### 消除重复记录

select distinct a.username from Admin a

##### 分组

select a.level,count(\*) from Admin a group by a.level having a.level in (1,2);

having过滤分组条件

where 子句设置查询条件

数学运算符 + - \* /

比较运算符 > < = >= <= <> != like

逻辑运算符and or not

其他运算符 in , not in , is null , is not null , is empty , is not empty between…and… , not between…and…

##### 分页

Query query = session. createQuery (“select p from Person p”);

query.setFirstResult(0); //起始位置

query.setMaxResult(10);//显示的记录数

public List findByPage(int pageNo,int pageSize){

Query query = getSession().createQuery(“from Admin”);

query.setFirstresult((pageNo-1)\*pageSize);

query.setMaxResults(pageSize);

return query.list()

}

##### N次查询问题

iterate方法：N+1查询 ，只有查实体对象的时候iterate和list才有区别，其他非实体对象没有区别

在执行list()时，首先在查询缓存找，找不到则去数据库查，如果获得是直接的结果集，则返回结果集，如果查到的是ID，则会调用load()，此时会产生N次查询

在查询缓存和二级缓存都开启的情况下，在查询缓存找到了对应的ID，然后再根据ID到二级缓存找到对应实体对象

在执行iterate()时，首先查到ID列表，然后load()

Query query = session. createQuery (“select p from Person p”);

Iterator iterator = query.iterate();//返回iterator类型

会发出N+1条查询语句，先查询所有的id，然后每迭代一次便发出一条查询语句，如果缓存有将要查询的结果则不发出查询语句

##### 查询结果的转换

之前查询某几项数据时，查询结果是一个数组类型，可以通过类型转换变成实体类型

Query query = session. createQuery (“select

new Person(p.id,p.name) from Person p”);

Person实体类要有对应的构造方法，此时就可以把查询的结果当作实体对象操作了

把结果转化成map类型

Query query = session. createQuery (“select new map(p.id as pid,p.name as pname) from Person p”);

此时取出来的就是map类型，第一个key，第二个为value

##### scroll方法

用于处理大数据

Query query = session. createQuery (“select p from Person p”);

ScrollableResults results =

query.scroll(ScrollMode.FORWORD\_ONLY);

while(results.next()){

Person p = (Person)results.get(0);

system.out.println( p.getId() + P.getName() )

}

##### 命名查询

可以把查询语句定义在配置文件中

在class标签下面：

<query name=”searchPersons”>

<! [ CDATA [select p from Person p where p.age > ? ] ]>

</query>

Query query =session. getNamedQuery (“searchPersons”);

query.setParameter(0,30);也可以使用命名参数查询

##### HQL嵌套子查询

相关子查询：指子查询语句引用了外层语句定义的别名，

无关子查询：子查询语句与外层查询语句无关

#### 标准化对象查询 QBC

##### 语法

Criteria criteria = session.createCriteria(Person.class);

criteria.add( Restrictions.like(“name”,”%11%”) ); //添加查询条件

List list = criteria.list();

Restrictions.sqlRestriction() 添加SQL语句

##### 动态SQL查询

应用条件查询可以很方便的实现动态SQL查询

public static List findStudent(String name,String number){

Criteria criteria = getSession().createCriteria(Student.class);

if(name !=null)

criteria.add(Restrictions.eq(“name”,name));

if(number !=null)

criteria.add(Restrictions.eq(“number”, number));

return criteria.list();

}

##### 举例查询（携带查询条件的对象）

Person c = new Person();

c.setId(100); //ID会被忽略掉

c.setName(“11”);

c.setCity(“1”);

c.setAddress(“1”);

Criteria criteria = session.createCriteria(Person.class);

criteria.add(

Example.create(c) //c为携带查询条件的对象

.enableLike(MatchMode.ANYWHERE) //对所有的字符串类型，如果非空，均使用模糊查询，就是在字符串两端加上%

.excludeZeroes() //取0的值不作为查询条件

.excludeProperty(“city”) //忽略city，不作为查询条件

);

example 的ignoreCase() 忽略大小写

excludeNone() 不排除查询为null或者为0的属性

//投影查询，设置要查询哪些属性

criteria.setProjection(

Projections.property(“name”)

);

List list = criteria.list();

查询多个属性

criteria.setProjection(

Projections.projectionList()

.add(Projections.property(“id”))

.add(Projections.property(“name”))

);//此时查询出的类型为数组

##### 查询总记录数

criteria.setProjection(

Projections.rowCunt()

);

Long total = (Long)criteria.uniqueResult();

##### 分页

Criteria criteria = getSession().createCriteria(Admin.class);

.setFirstResult((pageNo-1)\*pageSize)

.setMaxResults(pageSize)

.list();

聚集和分组

Projections类的count()方法用于设定统计的属性名，groupProperty()方法用于设定分组的属性名

##### 查询结果的转换

转化成实体类型(要有对应的构造方法)

criteria.setResultTransformer(new AliasToBeanResultTransformer(Admin.class));

转换成map类型

criteria.setProjection(

Projections.projectionList() //必须命别名，别名和属性不一样

.add(Projections.property(“id”),”pid”)

.add(Projections.property(“name”),”pname”)

);

criteria.setResultTransformat(CriteriaSpecification.ALIAS\_TO\_ENTITY\_MAP); //把结果转换成map类型

转换成VO类型，别名必须与VO定义的属性名一致，VO必须有空的构造方法

criteria.setResultTransformat(new AliasToBeanResultTransformat(PersonVO.class));

#### 普通的SQL语句查询

要想进行类型转化必须是查所有的字段

session.createSQLQuery(“select \* from t\_person”)

.addEntity(Person.class)

.list();

##### 隐式多态查询

session.createQuery(“from java.lang.Object”).list()

能够查出所有的实体类的记录，很少用

通过接口查询，两个类都实现了某个接口

session.createQuery(“from com.hibernate.Something”).list();

通过某个属性查询

session.createQuery(“select s from com.hibernate.Something s

where s.name like ?”)

.setParameter(0,”%1%”)

.list();

Query接口也支持方法链编程风格，setParameter()方法和setXXX()方法都将返回本身的实例，而不是void或其他类型

get()第一个参数必须是一个实体类

##### DML语句

删除

Query query =session.createQuery(“delete Person p where p.id in (:ids)”);

List ids = new ArrayList();

ids.add(2);

ids.add(3);

query.setParameterList(“ids”,ids);

query.executeUpdate();

批量删除

Query query = session.createQuery(“delete User”);

query.executeUpdate();

更新

Query query = session.createQuery(“update Person p set

p.name=? ”);

query.setParameter(0,”张三”);

query.executeUpdate();

#### 连接查询

两个表必须是双向关联的

##### 内连接

不满足连接条件的记录查不出来

**抓取内连接**

在加载数据时，不存在延迟加载，强制加载所有对象，可能会加载无用的对象，因此是浪费内存，但是当需要使用全部关联对象而且立即读取时，采用抓取内连接会大大提高查询性能

**隐式内连接**

隐式内连接指根本看不到使用内连接的join关键字，但是事实上已经发生内连接，通过点结构连接一个对象，只适用于一对一和多对一的关联

Query query = session.createQuery(

“select p.id,p.name,p.group.name from Person p”);

List list = query.list() 结果是数组

**显示内连接**

Query query = session.createQuery(

“select p.id,p.name,g.name from Person p join p.group g”);

##### 外连接

不满足条件的也能查出来

**左外连接**

可以使用hashset()过滤掉重复对象

Iterator I = new Hashset(getsession().createQuery(“查询语句”).list()).iterator();

Query query = session.createQuery(

“select p.id,p.name,g.name from Person p left join p.group g”);

**右外连接**

Query query = session.createQuery(

“select p.id,p.name,g.name from Group g right join g.persons p”);

**抓取左外连接**

from Student st left join fetch st.scores sc;

##### 交叉连接

from Student s,Admin a where s.id=a.id;

##### 内连接、外连接、交叉连接的区别

**内联接**

（典型的联接运算，使用像 = 或 <> 之类的比较运算符）。包括相等联接和自然联接。

内联接使用比较运算符根据每个表共有的列的值匹配两个表中的行。例如，检索 students和courses表中学生标识号相同的所有行。

**外联接**

外联接可以是左向外联接、右向外联接或完整外部联接。

在 FROM子句中指定外联接时，可以由下列几组关键字中的一组指定：

1）LEFT JOIN或LEFT OUTER JOIN

左向外联接的结果集包括 LEFT OUTER子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值。

2）RIGHT JOIN 或 RIGHT OUTER JOIN

右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。

3）FULL JOIN 或 FULL OUTER JOIN

完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。

**交叉联接**

交叉联接返回左表中的所有行，左表中的每一行与右表中的所有行组合。交叉联接也称作笛卡尔积。

FROM 子句中的表或视图可通过内联接或完整外部联接按任意顺序指定；但是，用左或右向外联接指定表或视图时，表或视图的顺序很重要。

#### Filter技术

定义filter，可以在任一配置文件定义

<filter-def name=”animalFilter”>

<filter-param name=”atype” type=”char” />

</filter-def>

配置：在配置文件的类标签里边<class></class>

<filter name=”animalFilter” condition=”animalType != :atype”>

<fileter>

在hibernate.cfg.xml 加上映射文件

启用filter,在Query之前

session.enableFilter(“animalFilter”).setParemeter(“atype”,’p’);

List list = session.createQuery(“from Animal”).list();

### 事务的隔离级别

       隔离级别定义了事务与事务之间的隔离程度。

       隔离级别与并发性是互为矛盾的：隔离程度越高，数据库的并发性越差；隔离程度越低，数据库的并发性越好。

       ANSI/ISO SQL92标准定义了一些数据库操作的隔离级别：

          未提交读（read uncommitted）：在读取数据时不会检查和使用任何锁，在这种隔离级别可能读取未提交的数据

          提交读（read committed）：只读取提交的数据并等待其他事务释放排它锁，读数据的共享锁在读操作完成后立即释放

          重复读（repeatable read）：像已提交读那样读数据，但会保持共享锁到事务最后

          序列化（serializable）：工作方式类似于可重复度，但它不仅会锁定受影响的数据，还会锁定这个范围，这就阻止了添加新数据及查询数据所涉及的范围

共享锁：防止其他事务更改数据，但不阻止读数据

排他锁：每一次数据的更新会阻止其他事务访问数据

更新锁：在数据被update语句读取时，他、其他事务可以读取数据，但其他的update语句必须等到更新锁被释放才可以读取数据

意向锁：

      通过一些现象，可以反映出隔离级别的效果。这些现象有：

更新丢失（lost update）：当系统允许两个事务同时更新同一数据是，发生更新丢失。

          脏读（dirty read）：当一个事务读取另一个事务尚未提交的修改时，产生脏读。

          非重复读（nonrepeatable read）：同一查询在同一事务中多次进行，由于其他提交事务所做的修改或删除，每次返回不同的结果集，此时发生非重复读。

          幻像读（phantom read）：同一查询在同一事务中多次进行，由于其他提交事务所做的插入操作，每次返回不同的结果集，此时发生幻像读。

### 缓存

#### 缓存介绍

缓存的范围：

事务范围：缓存只能被当前事务访问，缓存的生命周期依赖于事务的生命周期（一级缓存）

进程范围：缓存被进程内的所有事务访问，缓存的生命周期依赖于进程的生命周期（二级缓存）

集群范围：在集群范围内，缓存被一个或多个计算机的进程共享

一二级缓存存的是实体对象，key是ID，value是实体对象

查询缓存存的是普通结果集，key是HQL语句和参数，

value是：如果查询的是普通结果集则value是结果集

如果查询的是实体对象则value是实体对象的ID列表

#### 一级缓存

原理：

Hibernate加载一个对象时，首先在Session缓存中查询该对象，如果找到则直接引用，如果没有找到则从数据库查询，然后加入一级缓存，当清理缓存时，hibernate会根据缓存中对象的状态变化来同步更新数据库



一级缓存：存的是实体对象，必须有，session级别的

get、load、list、iterate方法可以把实体对象放入一级缓存

get、load、iterate方法能够利用一级缓存，list不会利用一级缓存

session.evict(p);将p 从缓存里清除

session.clear();将缓存里数据全部清空

session.flush();强制hibernate把一级缓存里的数据持久化到数据库

session.refresh(p);把数据库p的信息重新装载到缓存

往数据库插入大量数据时，要适时的使用flush()和clear()

一个session里不允许存在有两个相同数据库标识的对象

第一种方法，先移除具有相同标识的对象之一，然后在update

先flush 再evict 然后update

第二种方法，session.merge(p)，把p对象合并到数据库与之具有同名的对象里边去，p对象不会变成一个持久化对象

#### 二级缓存

存的是实体对象，不是必须有的，session factory级别的

适合使用二级缓存的数据：

很少被修改的数据；

不是很重要的数据，偶尔允许出现并发的数据；

不会被并发访问的数据

供应用程序参考的常量数据

需要配置文件 hibernate.cfg.xml

启用二级缓存

<property name=”hibernate.cache.use\_second\_level\_cache”>true

</property>

二级缓存策略提供商

<property name=”hibernate.cache.provider\_class”>

org.hibernate.cache.HashtableCacheProvider</property>

在.hbm.xml中配置：

<class>

(在最开始的位置)<cache usage=”read-only” />表示对此类启用二级缓存

</class>

把Student类对象加入二级缓存，在Student.hbm.xml加<cache usage=”read-write” />

把Student关联的Score对象加入二级缓存，在Student.hbm.xml的<set>元素中加入<cache usage=”read-write” />

把所有的Score对象加入二级缓存，在Score.hbm.xml加<cache usage=”read-write” />

清空指定的二级缓存：

session.getSessionFactory().getCache().evictEntityRegion(Person.class);

#### 查询缓存

所谓查询缓存，即让hibernate缓存list、iterator、createQuery等方法的查询结果集。如果没有打开查询缓存，hibernate将只缓存load方法获得的单个持久化对象。

是针对各种查询操作进行缓存的，采用键值存储方式，键就是SQL语句等，值就是结果集

查询缓存基于二级缓存，在使用查询缓存时必须先启用二级缓存

在配置文件中配置查询缓存：

<property name=”hibernate.cache.use\_query\_cache”>true</property>

与二级缓存使用相同的提供商

每次查询的时候都需要启用查询缓存

session.createQuery(“select p.id,p.name from Person p where p.name like ?”).setCacheable(true) //启用查询缓存

.setParameter(0,”%11%”).list();

查询缓存如果查询的是实体对象，则缓存的是实体对象的ID

### 抓取策略

#### 在many-to-one / one-to-one 上设置fetch

抓取：有一个对象，如何得到它的关联对象

1. fetch=”select” 查询抓取，通过第二条查询语句抓取关联对象的策略，这是默认值

Person p = (Person)session.load(Person.class,1);

p.getId(); //此时不会发出查询语句

p.getName();//此时会发出第一条查询语句

Group g = p.getGroup();

g.getId();

g.getName(); //此时发出第二条查询语句

1. fetch=”join” 连接抓取，必须手动更改，在加载对象的同时通过一条外连接已经把关联对象加载上来了，所以lazy配置失效（确切的说是关联对象（Group对象）的懒加载失效）

Person p = (Person)session.load(Person.class,1);

p.getId(); //此时不会发出查询语句

p.getName();//此时会发出一条带有外连接查询语句

Group g = p.getGroup();

g.getId();

g.getName();

批量抓取：有一批对象，想得到这一批对象的关联对象

当fetch=”select”的时候，可以定义它的批量抓取策略，在对方的class标签<class ……. batch-size=””>，设置分批抓取被关联对象时每批的数量

List persons = session.createQuery(“from Person”).list();//发出查询语句，把所有的Person对象都拿出来了，包括gId

for(Iterator iterator = persons.iterator();iterator.hasNext();){

Perosn p =(Person)iterator.next();

p.getId();

p.getName();

Group g= p.getGroup(); //每次根据gid发出查询语句（定义批量抓取就按照定义的数量加载，反之，一条一条的发）

g.getId();

g.getName();

}

#### 在集合上设置fetch

抓取

fetch=”select” 查询抓取，通过第二条查询语句抓取关联对象的策略，这是默认值

fetch=”join” 连接抓取，必须手动添加，在加载对象的同时通过一条外连接已经把关联对象加载上来

批量抓取

fetch=”select”，可以定义批量抓取策略，在集合上配置batch-size

fetch=”subselect”，批量抓取对象，前一条查询语句查询出来的结果决定了这条查询语句中查询对象的个数，把前一条语句当作子查询

可以在<set>元素中加batch-size=””定义集合上抓取的数量

### 批量处理数据的一些配置

提高批量处理的性能

Id策略不能是自增的

select count(\*) from t\_user;查记录数

<property name=”hibernate.connection,url”>jdbc:mysql;//localhost/hibernate?rewriteBatchedStatements</property>

<property name=”hibernate.jdbc.batch\_size”>25</property>

### 并发访问和锁定策略

#### 悲观锁定

一个线程操作时其他线程不能操作，容易造成死锁

设置悲观锁：

调用load()

Session session = getSession();

Student student =

(Student)session.load(Student.class,6,LockMode.OPGRADE);

调用lock()

Session session = getSession();

Student student =

(Student)session.load(Student.class,6);

session.lock(student, LockMode.OPGRADE);

调用Query.setLockMode()

Query query = getsession().createQuery(“from Student student”);

query.setLockMode(“student”, LockMode.OPGRADE);

query.list();

#### 乐观锁定

在实体类中加一个版本号属性

Private int Number;

在配置文件中映射这个属性<version name=”Number”>要紧跟在<id>后面，版本号不一致就会报错

也可以使用时间戳作为版本字段<timestamp>，用法与version相同

hibernate自动维护，提高性能，避免死锁

### VO、DTO、DO、PO有什么区别

VO（View Object）：视图对象，用于展示层，它的作用是把某个指定页面（或组件）的所有数据封装起来。

DTO（Data Transfer Object）：数据传输对象，这个概念来源于J2EE的设计模式，原来的目的是为了EJB的分布式应用提供粗粒度的数据实体，以减少分布式调用的次数，从而提高分布式调用的性能和降低网络负载，但在这里，泛指用于展示层与服务层之间的数据传输对象。

DO（Domain Object）：领域对象，就是从现实世界中抽象出来的有形或无形的业务实体。

PO（Persistent Object）：持久化对象，它跟持久层（通常是关系型数据库）的数据结构形成一一对应的映射关系，如果持久层是关系型数据库，那么，数据表中的每个字段（或若干个）就对应PO的一个（或若干个）属性。

vo不需要映射文件，只是用来封装视图对象，用于呈现层

public class PersonVO{

private int id;

private String name;

public Person(){

}

public Person(int id,String name){

this.id=id;

this.name= name;

}

getters 和setters

}

session.createQuery(select new vo.PersonVO(p.id,p.name) from Person p).list();