**Hibernate技术速查**

目录

[1.hibernate的引入 3](#_Toc426709404)

[2.安装配置 3](#_Toc426709405)

[3.基本概念和CURD 3](#_Toc426709406)

[4. 对象状态 5](#_Toc426709407)

[5. HQL和Criteria 5](#_Toc426709408)

[6. 基本功能练习 6](#_Toc426709409)

[7.关联映射 6](#_Toc426709410)

[1> 多对一(Employee - Department) 6](#_Toc426709411)

[2> 一对多(Department-Employee) 7](#_Toc426709412)

[3>一对一(Person - IdCard) 7](#_Toc426709413)

[4>多对多(teacher - student) 7](#_Toc426709414)

[5>组件映射(User-Name 8](#_Toc426709415)

[8.继承映射 8](#_Toc426709416)

[1>对象模型(Java类结构) 8](#_Toc426709417)

[2> 一个类继承体系一张表(subclass) 9](#_Toc426709418)

[3> 每个子类一张表(joined-subclass) 9](#_Toc426709419)

[4>混合使用“一个类继承体系一张表”和“每个子类一张表” 10](#_Toc426709420)

[9.集合映射 11](#_Toc426709421)

[1> 集合映射(set, list, array,bag, map) 11](#_Toc426709422)

[2> cascade和inverse (Employee – Department) 12](#_Toc426709423)

[10.懒加载 13](#_Toc426709424)

[11. 缓存 13](#_Toc426709425)

[12.事务 14](#_Toc426709426)

[1> JDBCTransaction 14](#_Toc426709427)

[2>JTATransaction 14](#_Toc426709428)

[13. 其他问题 15](#_Toc426709429)

[14. Hibernate不适合的场景 16](#_Toc426709430)

[15. 与JPA的集成(annotation方式) 17](#_Toc426709431)

# 1.hibernate的引入

模型不匹配(阻抗不匹配)

Java面向对象语言，对象模型，其主要概念有：继承、关联、多态等；数据库是关系模型，其主要概念有：表、主键、外键等。

解决办法

1使用JDBC手工转换。

2使用ORM(Object Relation Mapping对象关系映射)框架来解决，主流的ORM框架有Hibernate、TopLink、OJB。

# 2.安装配置

下载地址http://www.hibernate.org，本教程使用3.2.5。

将下载目录/hibernate3.jar和/lib下的hibernate运行时必须的包加入classpath中：

antlr.jar,cglib.jar,asm.jar,commons-collections.jar,commons-logging.jar,jta.jar,dom4j.jar

配置文件hibernate.cfg.xml和hibernate.properties，XML和properties两种，这两个文件的作用一样，提供一个即可，推荐XML格式，下载目录/etc下是示例配置文件。

可以在配置文件指定：数据库的URL、用户名、密码、JDBC驱动类、方言等。

启动时Hibernate会在CLASSPATH里找这个配置文件。

映射文件(hbm.xml，对象模型和关系模型的映射)。在/eg目录下有完整的hibernate示例。

快速开始小例子

# 3.基本概念和CURD

开发流程

1由Domain object -> mapping->db。(官方推荐)

2由DB开始，用工具生成mapping和Domain object。(使用较多)

3由映射文件开始。

Domain Object限制

1.默认的构造方法(必须的)。

2有无意义的标示符id（主键）(可选)

3非final的，对懒加载有影响（可选）

Domain Java Object(User)

**public** **class** User {

**private** **int** id;

**private** String name;

**private** Date birthDay;

**//getter setter…**

}

**1.hbm.xml**

<?xml version="1.0"?>

<hibernate-mapping package=“cn.itcast.domain">

<class name="User" table="user">

<id name="id">

<generator class="native"/>

</id>

<property name="name"/>

<property name="birthday”/>

</class>

</hibernate-mapping>

主键生成器，参考文档P65 5.1.4

**2.详细信息见参考文档P62**

**Java代码**

**1.初始化代码(只做一次)**

Configuration cfg = new Configuration();

cfg.configure(“config.cfg.xml”);

也可以通过cfg.setProperty设置属性。

SessionFactory sessionFactory = cfg.buildSessionFactory();

**2.模板代码**

Session session = null;Transaction tx = null;

try{

session = sessionFactory.openSession();

tx = session.beginTransaction();

//…你的代码save,delete,update,get…

tx.commit();

}catch(Exception e){

if(tx !=null)tx.rollback();throw e;

}finally{

if(session != null)session.close();

}

Session的几个主要方法

1.save,persist保存数据，persist在事务外不会产生insert语句。

2.delete,删除对象

3.update,更新对象，如果数据库中没有记录，会出现异常。

4.get,根据ID查，会立刻访问数据库。

5.Load，根据ID查，(返回的是代理，不会立即访问数据库)。

6.saveOrUpdate,merge(根据ID和version的值来确定是save或update),调用merge你的对象还是托管的。

7.lock(把对象变成持久对象，但不会同步对象的状态)。

# 4. 对象状态

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到数据库(hibernate能检测到)。

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



# 5. HQL和Criteria

HQL(Hibernate Query Language)

面向对象的查询语言，与SQL不同，HQL中的对象名是区分大小写的（除了JAVA类和属性其他部分不区分大小写）；HQL中查的是对象而不是和表，并且支持多态；HQL主要通过Query来操作，Query的创建方式：

Query q = session.createQuery(hql);

from Person

from User user where user.name=:name

from User user where user.name=:name and user.birthday < :birthday

Criteria

Criteria是一种比HQL更面向对象的查询方式；Criteria的创建方式：

Criteria crit = session.createCriteria(DomainClass.class);

简单属性条件如：criteria.add(Restrictions.eq(propertyName, value)),

criteria.add(Restrictions.eqProperty(propertyName,otherPropertyName))

# 6. 基本功能练习

实现UserDao

public interface UserDao {

public void saveUser(User user);

public User findUserById(int id);

public User findUserByName(String name);

public void updateUser(User user);

public void remove(User user);

}

实验步骤：

1.设计domain对象User。

2.设计UserDao接口。

3.加入hibernate.jar和其依赖的包。

4.编写User.hbm.xml映射文件，可以基于hibernate/eg目录下的org/hibernate/auction/User.hbm.xml修改。

5.编写hibernate.cfg.xml配置文件，可以基于hibernate/etc/hibernate.cfg.xml修改；必须提供的几个参数：

connection.driver\_class、connection.url、connection.username、connection.password、dialect、hbm2ddl.auto。

6.编写HibernateUtils类，主要用来完成Hibnerate初始化和提供一个获得Session的方法；这步可选。

7.实现UserDao接口。

# 7.关联映射

## 1> 多对一(Employee - Department)

映射文件<many-to-one name=”depart” column=”depart\_id”/>

ER图



## 2> 一对多(Department-Employee)

<set name=”employees”>

<key column=”depart\_id”/>

<one-to-many class=”Employee”/>

</set>

## 3>一对一(Person - IdCard)

1)基于主键的one-to-one(person的映射文件)

<id name=”id”>

<generator class=”foreign”><param name=”property”>idCard</param></generator>

<id>

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



2)基于外健的one-to-one，可以描述为多对一，加unique=“true”约束

<one-to-one name=”idCard” property-ref=“person”/>

property-ref用于指定关联类的一个属性，这个属性将会和本外键相对应

<many-to-one name=”person” column=”person\_id” unique=”true” not-null=”true”/>

 <!-唯一的多对一，其实就便成了一对一了-->



## 4>多对多(teacher - student)

在操作和性能方面都不太理想，所以多对多的映射使用较少，实际使用中最好转换成一对多的对象模型；Hibernate会为我们创建中间关联表，转换成两个一对多。

<set name="teacher" table="teacher\_student">

<key column="teacher\_id"/>

<many-to-many class="Student" column="student\_id"/>

</set>



## 5>组件映射(User-Name

关联的属性是个复杂类型的持久化类，但不是实体即：数据库中没有表与该属性对应，但该类的属性要之久保存的。

<component name=”name” class=”com.test.hibernate.domain.Name”>

<property name=”initial”/>

<property name=”first”/>

<property name=”last”/>

</component>

当组件的属性不能和表中的字段简单对应的时候可以选择实现：

org.hibernate.usertype. UserType或

org.hibernate.usertype. CompositeUserType

# 8.继承映射

## 1>对象模型(Java类结构)



## 2> 一个类继承体系一张表(subclass)

**(表结构)**



**(映射文件)**

<class name="Employee" table="employee" discriminator-value="0">

<id name="id">

<generator class="native"/>

</id>

<discriminator column="type" type="int"/>

<property name="name"/>

<many-to-one name=”depart” column=”depart\_id”/>

<subclass name="Skiller" discriminator-value="1">

<property name=”skill”/>

</subclass>

<subclass name="Sales" discriminator-value="2">

<property name="sell"/>

</subclass>

</class>

## 3> 每个子类一张表(joined-subclass)

**(表结构)**



**(映射文件)**

<class name="Employee" table="employee">

<id name="id">

<generator class="native"/>

</id>

<property name="name"/>

<joined-subclass name="Skiller" table="skiller">

<key column="employee\_id"/>

<property name="skill"/>

</joined-subclass>

<joined-subclass name="Sales" table="sales">

<key column="employee\_id"/>

<property name="sell"/>

</joined-subclass>

</class>

## 4>混合使用“一个类继承体系一张表”和“每个子类一张表”

**(表结构)**



**(映射文件)**

<class name="Employee" table="employee">

<id name="id">

<generator class="native"/>

</id>

<discriminator column="type"/>

<property name="name"/>

<subclass name="Skiller">

<property name="net"/>

</subclass>

<subclass name=”Sales”">

<join table="sales">

<key column="employee\_id"/>

<property name="sell"/>

</join>

</subclass>

</class>

**5> 每个具体类一张表(union-subclass)**

**(表结构)**



**(映射文件)**

<class name="Employee" abstract="true">

<id name="id">

<generator class="hilo"/>

</id>

<property name="name"/>

<union-subclass name="Skiller" table="skiller">

<property name="skill"/>

</union-subclass>

<union-subclass name="Sales" table="sales">

<property name="sell"/>

</union-subclass>

</class>

主健不能是identity类型，如果父类是abstract=”true”就不会有表与之对应。

隐式多态，映射文件没有联系，限制比较多很少使用。

# 9.集合映射

## 1> 集合映射(set, list, array,bag, map)

<set name=”employees” >

<key column=”depart\_id”/>

<one-to-many class=”Employee”/>

<!-- <element type="string" column="name"/> -->

<!--

<composite-element class=”YourClass”>

<property name=”prop1”/>

<property name=”prop2”/>

</composite>

-->

</set>

<list name=”employees” >

<key column=”depart\_id”/>

<!—表中有单独的整型列表示list-index 🡪

<list-index column=”order\_column”/>

<one-to-many class=”Employee”/>

</list>

<array name=”employees” >

<key column=”depart\_id”/>

<!—表中有单独的整型列表示list-index à

<list-index column=”order\_column”/>

<one-to-many class=”Employee”/>

</array>

<bag name="employees " order-by="id desc">

<key column=”depart\_id”/>

<one-to-many class=”Employee”/>

</bag>

<map name="employees ">

<key column=”depart\_id”/>

<map-key type="string" column="name"/>

<one-to-many class=”Employee”/>

</map>

这些集合类都是Hibernate实现的类和JAVA中的集合类不完全一样，set,list,map分别和JAVA中的Set,List,Map接口对应，bag映射成JAVA的List；这些集合的使用和JAVA集合中对应的接口基本一致；在JAVA的实体类中集合只能定义成接口不能定义成具体类， 因为集合会在运行时被替换成Hibernate的实现。

集合的简单使用原则：大部分情况下用set，需要保证集合中的顺序用list，想用java.util.List又不需要保证顺序用bag。

## 2> cascade和inverse (Employee – Department)

Casade用来说明当对主对象进行某种操作时是否对其关联的从对象也作类似的操作，常用的cascade:

none,all,save-update ,delete, lock,refresh,evict,replicate,persist,

merge,delete-orphan(one-to-many) 。一般对many-to-one,many-to-many不设置级联，在<one-to-one>和<one-to-many>中设置级联。

inverse表“是否放弃维护关联关系”(在Java里两个对象产生关联时，对数据库表的影响)，在one-to-many和many-to-many的集合定义中使用，inverse=”true”表示该对象不维护关联关系；该属性的值一般在使用有序集合时设置成false（注意hibernate的缺省值是false）。

one-to-many维护关联关系就是更新外键。many-to-many维护关联关系就是在中间表增减记录。

注: 配置成one-to-one的对象不维护关联关系

# 10.懒加载

通过asm和cglib二个包实现；Domain是非final的。

1.session.load懒加载。

2.one-to-one(元素)懒加载：

必需同时满足下面三个条件时才能实现懒加载

(主表不能有constrained=true,所以主表没有懒加载)

lazy!=false 2)constrained=true 3)fetch=select

3.one-to-many (元素)懒加载：1)lazy!=false 2)fetch=select

4.many-to-one (元素) ：1)lazy!=false 2)fetch=select

5.many-to-many (元素) ：1)lazy!=false 2)fetch=select

6.能够懒加载的对象都是被改写过的代理对象，当相关联的session没有关闭时，访问这些懒加载对象（代理对象）的属性（getId和getClass除外）hibernate会初始化这些代理，或用Hibernate.initialize(proxy)来初始化代理对象；当相关联的session关闭后，再访问懒加载的对象将出现异常。

# 11. 缓存

缓存的作用主要用来提高性能，可以简单的理解成一个Map；使用缓存涉及到三个操作：把数据放入缓存、从缓存中获取数据、删除缓存中的无效数据。

一级缓存，Session级共享。

save,update,saveOrUpdate,load,get,list,iterate,lock这些方法都会将对象放在一级缓存中，一级缓存不能控制缓存的数量，所以要注意大批量操作数据时可能造成内存溢出；可以用evict,clear方法清除缓存中的内容。

二级缓存，SessionFactory级共享。

实现为可插拔，通过修改cache.provider\_class参数来改变;

hibernate内置了对EhCache,OSCache,TreeCache,SwarmCache的支持，可以通过实现CacheProvider和Cache接口来加入Hibernate不支持的缓存实现。

在hibernate.cfg.xml中加入:

<class-cache class="className" usage="read-only"/>

或在映射文件的class元素加入子元素:

<cache usage="read-write"/>

其中usage:read-only,read-write,nonstrict-read-write,transactional

Session的:save(这个方法不适合native生成方式的主键),

update,saveOrUpdate,list,iterator,get,load,以及Query,Criteria都会填充二级缓存，但只有(没打开查询缓存时)Session的iterator,get,load会从二级缓存中取数据(iterator可能存在N+1次查询)。

Query,Criteria(查询缓存)由于命中率较低，所以hibernate缺省是关闭；修改cache.use\_query\_cache为true打开对查询的缓存，并且调用query.setCacheable(true)或criteria.setCacheable(true)。

SessionFactory中提供了evictXXX()方法用来清除缓存中的内容。

统计信息打开generate\_statistics，用sessionFactory.getSatistics()获取统计信息。

分布式缓存和中央缓存。

使用缓存的条件

1.读取大于修改。

2.数据量不能超过内存容量。

3.对数据要有独享的控制。

4.可以容忍出现无效数据。

# 12.事务

## 1> JDBCTransaction

单个数据库（一个SesisonFactory对应一个数据库），由JDBC实现。

Session session = null;

Transaction tx =null;

try {

session = sessionFactory.openSession();

tx = session.beginTransaction();

//process

tx.commit();

} catch(HibernateException e){

if(tx != null)tx.rollback();throw e;

}finally {

if (session != null)session.close();

}

connection.setAutoCommit(false);

connection.commit();conn.rollback();

## 2>JTATransaction

可以简单的理解成跨数据库的事物，由应用JTA 容器实现；使用JTATransaction需要配置hibernate.transaction.factory\_class参数，该参数缺省值是org.hibernate.transaction. JDBCTransactionFactory，当使用JTATransaction时需要将该参数改成org.hibernate.transaction.JTATransactionFactory，并配置jta.UserTransaction参数JNDI名(Hibernate在启动JTATransaction时要用该值到JNDI的上下文Context中去找javax.transaction.UserTransaction)。

javax.transaction.UserTransactin tx = context.lookup(“jndiName”);

try{

tx.begin();

//多个数据库的session操作;

//session1….

//session2….

tx.commit();

}catch(Exception e){

tx.rollback(); throw e;

}

session context和事务边界

用current\_session\_context\_class属性来定义context（用sessionFactory.getCurrentSession()来获得session），其值为：

1.thread：ThreadLocal来管理Session实现多个操作共享一个Session，避免反复获取Session，并控制事务边界，此时session不能调用close当commit或rollback的时候session会自动关闭(connection.release\_mode:after\_transaction)。

Open session in view：在生成（渲染）页面时保持 session打开。

2.jta：由JTA事务管理器来管理事务(connection.release\_mode:after\_statement)。

悲观锁和乐观锁

悲观锁由数据库来实现；乐观锁hibernate用version和timestamp来实现

# 13. 其他问题

hibernate.cfg.xml和hbm.xml内容解释

数据类型

1.<property name=“name” type=“java.lang.String”/>

type可以是hibernate、java类型或者你自己的类型(需要实现hibernate的一个接口)。

2.基本类型一般不需要在映射文件(hbm.xml)中说明，只有在一个JAVA类型和多个数据库数据类型相对应时并且你想要的和hibernate缺省映射不一致时，需要在映射文件中指明类型(如:java.util.Date,数据库DATE,TIME,DATATIME,TIMESTAMP，hibernate缺省会把java.util.Date映射成DATATIME型，而如果你想映射成TIME，则你必须在映射文件中指定类型)。

3.数据类型的对应关系见参考文档5.2.2

Session是非线程安全的，生命周期较短，代表一个和数据库的连接，在B/S系统中一般不会超过一个请求；内部维护一级缓存和数据库连接，如果session长时间打开，会长时间占用内存和数据库连接。

SessionFactory是线程安全的，一个数据库对应一个SessionFactory，生命周期长，一般在整个系统生命周期内有效；SessionFactory保存着和数据库连接的相关信息（user，password，url）和映射信息，以及Hibernate运行时要用到的一些信息。

flush时将一级缓存与数据库同步

大批处理

大量操作数据时可能造成内存溢出，解决办法如下：

1.清除session中的数据

for(int i=0;i<100000;i++)session.save(obj);

for(int i=0;i<100000;i++){

session.save(obj);

if(i% 50 == 0){session.flush(); session.clear();}

}

2.用StatelessSession接口：它不和一级缓存、二级缓存交互，也不触发任何事件、监听器、拦截器，通过该接口的操作会立刻发送给数据库，与JDBC的功能一样。

StatelessSession s = sessionFactory.openStatelessSession();该接口的方法与Session类似。

3.Query.executeUpdate()执行批量更新，会清除相关联的类二级缓存(sessionFactory.evict(class))，也可能会造成级联，和乐观锁定出现问题

HQL

1查询多个对象select art, user from Article art, User user where art.author.id=user.id and art.id=:id这种方式返回的是Object[]，Object[0]:article,Object[1]:user。

2分页query.setFirstResult,query.setMaxResults.

查询记录总数query.iterate(“select count(\*) from Person”).next()

3批量更新query.executeUpdate()可能造成二级缓存有实效数据。

Criteria

1排序Criteria.addOrder(Order.desc(propertyName));

2关联查询criteria.setFetchMode(“propertyName”, FetchMode.SELECT)与映射文件中关联关系的fetch作用一致。

3投影Projections.rowCount(),max(propertyName), avg, groupProperty…

4分页Projections.rowCount(),criteria.setFirstResult(),criteria.setMaxResults()

5DetachedCriteria可在session外创建（在其他层创建比如在Service中创建）然后用getExecutableCriteria(session)方法创建Criteria对象来完成查询。

6Example查询,Example.create(obj);criteria.add(example)。

N+1次查询和懒加载

1.用Query.iterator可能会有N+1次查询。

2.懒加载时获取关联对象。

3.如果打开对查询的缓存即使用list也可能有N+1次查询。

拦截器与事件

拦截器与事件都是hibernate的扩展机制，Interceptor接口是老的实现机制，现在改成事件监听机制；他们都是hibernate的回调接口，hibernate在save,delete,update…等会回调这些类。

SQL和命名查询

用Map代替Domain对象；将对象转化为XML。

# 14. Hibernate不适合的场景

不适合OLAP(On-Line Analytical Processing联机分析处理)，以查询分析数据为主的系统；适合OLTP（on-line transaction processing联机事务处理）。

对于些关系模型设计不合理的老系统，也不能发挥hibernate优势。

数据量巨大，性能要求苛刻的系统，hibernate也很难达到要求, 批量操作数据的效率也不高。

# 15. 与JPA的集成(annotation方式)

需要添加的包ejb3-persistence.jar, hibernate-entitymanager.jar, hibernate-annotations.jar, hibernate-commons-annotations.jar, jboss-archive-browsing.jar, javassist.jar

配置文件%CLASSPATH%/META-INF/persistence.xml

JAVA代码：

EntityManagerFactory emf = Persistence.createEntityManagerFactory(name);

//(Name:在persistence.xml中指定。)

EntityManager em = emf.createEntityManager();

EntityTransaction tx = em.getTransaction();

Tx.begin();

Em.persist(entity);//remove,merge,find

Tx.commit();

Em.close();

Emf.close();