Hibernate 配置

# Hibernate介绍

Hibernate属于一种ORM框架（Object Relational Mapping，对象关系映射），ORM主要进行面向对象与关系型数据库之间的映射，属于数据持久层技术。

参考资料：http://www.zhihu.com/question/21104468

## Hibernate与mybatis相比的优劣：

Hibernate具有良好的数据映射机制，不用写sql语句，数据库移植性好，缓存机制完善。但是表的关联操作配置麻烦，对sql的优化不方便。

Mybatis要手动编写sql，可以细粒度的优化sql，对数据库操作方便灵活。但是数据库移植需要重写sql，需要维护sql与结果的映射，自带的缓存自己不佳。

## Hibernate与jdbc

jdbc是用什么查什么，减少带宽。

Hibernate建议将对象的字段全部取出，方便封装成一个类，并且可以支持表格关联。如果使用HQL选取部分属性，不仅要再进行封装进行使用，而且不利于表格关联操作。

# Hibernate配置及demo

Hibernate开发方式一般分为从3种：

1.是先写Pojo，再写mapping，再自动生成db（这是官方推荐，部分人用这种方式）。

2.是建立db，用工具生成mapping和pojo（使用较多）。

3.是先写mapping，再写Pojo和db，比较少见。

我们采用第二种方式，先建好数据库和表，代码来自javaTools。

## Domain对象/javabean/pojo/data对象

一般一张表对应一个pojo对象（plain ordinary java object，简单Java对象）。

一个pojo序列化，含主键、私有成员、无参构造器、get与set方法。

一般Pojo类名用首字母大写的表名，字段名也相似。

Hibernate推荐每一张表都有一个独立的主键，并且这个主键和业务逻辑无关，这样能够保证主键独立性。

package hibernate.domain;

import java.io.Serializable;

/\*\*

\* 序列化实现对象的唯一标识，便于网络传输等

\*

\* @author dell

\*

\*/

public class Customer implements Serializable {

private static final long serialVersionUID = 1L;

private int id;

private String userName;

private String passwd;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public String getPasswd() {

return passwd;

}

public void setPasswd(String passwd) {

this.passwd = passwd;

}

}

## Mapping文件

映射文件描述Pojo对象与数据表字段的映射关系。一般mapping文件用Pojo名加上后缀名.hbm.xml。注意在xml中开头第一行不能是空行或其他，必须是<?xml>.

<?xml version="1.0" encoding="UTF-8"?>

<!-- 该文件用于ORM映射 -->

<!-- mapping文件需要的dtd -->

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="hibernate.domain">

<!-- 配置domain对象与表的对应关系 -->

<class name="Customer" table="customer">

<!-- 配置主键 -->

<id name="id" column="id" type="java.lang.Integer">

<!-- 主键生成策略hilo，native，increment，sequence... -->

<generator class="increment">

<!-- <param name="auto\_inc">emp\_inc</param> -->

</generator>

</id>

<!-- 配置其他属性 -->

<property name="userName" type="java.lang.String">

<column name="username" not-null="false"></column>

</property>

<!-- 配置其他属性 -->

<property name="passwd" type="java.lang.String">

<column name="password" not-null="false"></column>

</property>

</class>

</hibernate-mapping>

## hibernate.cfg.xml

hibernate.cfg.xml是hibernate的配置文件，包括数据库配置，mapping文件路径等，该名称是默认的配置文件名。

<?xml version="1.0" encoding="UTF-8"?>

<!-- 配置文件需要的dtd -->

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<!-- 配置与数据库的session会话 -->

<session-factory>

<!-- 配置数据库参数 -->

<property name="connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="connection.url">jdbc:mysql://127.0.0.1:3306/javatools</property>

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

<property name="connection.password">1234</property>

<!-- 配置数据库方言，hibernate自动会根据翻译成sql方言 -->

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

<!-- 配置可以使用getCurrentSession，线程绑定 -->

<property name="current\_session\_context\_class">thread</property>

<!-- 显示运行的sql -->

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

<!-- 根据映射文件自动创建数据表,create表示每次创建，update为更新，若没有创建 -->

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

<!-- 配置mapping文件 -->

<mapping resource="hibernate/Customer.hbm.xml" />

</session-factory>

</hibernate-configuration>

## 业务与控制

一般一个SessionFactory对应一个数据库，利用该工厂建立多个连接（session），所以该工厂可以抽取出来（只使用一个数据库时）。

### SessionFactory工具类

package hibernate.util;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

/\*\*

\* 写一个sessionFactory的工具类，且不允许继承，因为一个数据库最好只用一个sessionFactory

\*

\* @author dell

\*

\*/

final public class MySessionFactory {

private static SessionFactory sessionFactory = null;

private MySessionFactory() {

}

// 静态化的代码块，只会被执行一次

static {

// 创建Configuration来读取hibernate.cfg.xml，默认该文件名可以不用写

Configuration configuration = new Configuration().configure("hibernate.cfg.xml");

// 创建数据库的sessionFactory，这是一个重量级的对象，一个sessionFactory对应一个数据库

sessionFactory = configuration.buildSessionFactory();

}

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

}

### 测试主函数

package hibernate.view;

import java.util.Date;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import hibernate.domain.Customer;

import hibernate.util.MySessionFactory;

/\*\*

\* 使用hibernate实现数据库crud

\*

\* @param args

\*/

public class MyHibernate {

public static void main(String[] args) {

addCustomer();

addCustomer();

addCustomer();

retrievalCustomer();

updateCustomer();

deleteCustomer();

}

/\*\*

\* 插入一条数据

\*/

public static void addCustomer() {

System.err.println("准备插入一个数据");

// 创建Configuration来读取hibernate.cfg.xml，默认该文件名可以不用写

Configuration configuration = new Configuration().configure("hibernate.cfg.xml");

// 创建数据库的sessionFactory，这是一个重量级的对象，一个sessionFactory对应一个数据库

SessionFactory sessionFactory = configuration.buildSessionFactory();

// 申请数据库session（connection）使用后必须记得关闭

Session session = sessionFactory.openSession();

// hibernate要求增改删操作必须使用事务，否则不会执行

Transaction transaction = session.beginTransaction();

// 添加一个customer

Customer customer = new Customer();

customer.setUserName("zhangsan");

customer.setPasswd(new Date().toString());

session.save(customer);

transaction.commit();

session.close();

System.err.println("插入数据完成");

}

/\*\*

\* 查询一条数据

\*/

private static void retrievalCustomer() {

System.err.println("准备查询数据记录");

// 通过工具获取session

Session session = MySessionFactory.getSessionFactory().openSession();

// 获取主键=2的对象，通过类名反射获取mapping

Customer customer = session.load(Customer.class, 2);

System.err.println("查询到主键为2的customer姓名为：" + customer.getUserName());

}

/\*\*

\* 更新一条数据

\*/

private static void updateCustomer() {

System.err.println("准备修改数据记录");

// 通过工具获取session

Session session = MySessionFactory.getSessionFactory().openSession();

Transaction transaction = session.beginTransaction();

// 获取主键=3的对象，通过类名反射获取mapping

Customer customer = session.load(Customer.class, 3);

// set会产生数据游离状态，hibernate会检测到修改，所以commit会发生改变

customer.setUserName(customer.getUserName() + "1");

customer.setPasswd(new Date().toString());

transaction.commit();

session.close();

System.err.println("修改主键3的姓名为：" + customer.getUserName() + "1");

}

/\*\*

\* 删除一条数据

\*/

private static void deleteCustomer() {

System.err.println("准备删除记录");

// 通过工具获取session

Session session = MySessionFactory.getSessionFactory().openSession();

Transaction transaction = session.beginTransaction();

try {

// 获取主键=1的对象，通过类名反射获取mapping

Customer customer = session.load(Customer.class, 1);

session.delete(customer);

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

System.err.println("删除主键1的记录成功");

}

}

# Hibernate核心类、接口与配置

## Configuration类

配置管理的类。

## Hibernate.cfg.xml

配置数据库管理、mapping文件管理。

## Pojo.hbm.xml

配置Pojo与数据库之间的mapping。

## SessionFactory接口

SessionFactory是session工厂，获取数据库连接。

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

## Session类操作数据库

### getSession与openSession

SessionFactory可以进行sql缓存，被称为session级缓存（属于一级缓存）。

一次请求不返回之前都可以看成一个线程。

SessionFactory.openSession()每次获取新的session。如果在不同的service中用不同的session就用openSession，但是注意必须手动close。

SessionFactory.getCurrentSession()获取当前线程绑定的session，可以进行事务控制，本方法可以自动关闭session，但是查询也要事务提交，并且要配置cfg.xml，thread为本地事务（jdbc事务，针对一个数据库），jta为全局事务（跨数据库）。

可以通过threadLocal将openSession与线程绑定实现getCurrentSession的功能。

package hibernate.util;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

/\*\*

\* 获取session的两种方法

\*

\* @author dell

\*

\*/

public class SessionUtil {

private static SessionFactory sessionFactory = null;

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

private SessionUtil() {

}

static {

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

}

// 使用原生的openSession，获取全新的session

public static Session openSession() {

return sessionFactory.openSession();

}

// 通过threadLocal方法实现getCurrentSession的功能

public static Session getCurrentSession() {

Session session = threadLocal.get();

// 判断是否得到

if (session == null) {

session = sessionFactory.openSession();

// 将新建的session和线程进行绑定，之后获得的都是该session

threadLocal.set(session);

}

return session;

}

}

下面介绍session几个重要的方法。

### 保存session.save

Session.save(obj);

### 删除session.delete

Session.delete(obj);

### 查询get、load、query、criteria

#### get与load

Session.get()与session.load().

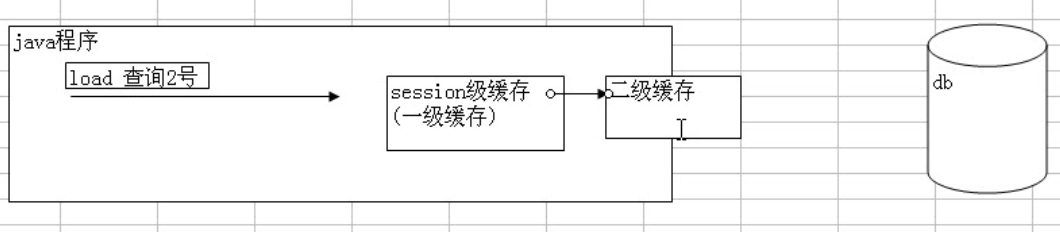
Get直接返回实体类，查不到返回null。Load会返回一个实体代理对象，如果数据不存在会抛出异常，类似于中间件到数据的查找映射。

Load先到session缓存（一级缓存）与二级缓存中去查，没有也会返回代理对象（不马上到DB中找），等到使用代理对象操作时才到DB查找，这就是“load在默认情况下支持延迟加载（lazy）”，会导致DB中数据不存在的时候会抛出异常。比如连续多次load同一数据最多执行一次sql。

Get先到session缓存与二级缓存中去查，如果没有就马上去DB中查。

总之，如果确定DB中存在就用load，不确定就用get，这样会提高效率。取消load的懒加载可以配置mapping文件：<class name="Customer" lazy="false" table="customer">。

缓存图解如下。



#### query

获取query引用，这里用的还是Pojo对象不是表名，条件中的可以是表的字段名也可以是Pojo成员名，推荐是Pojo成员名。

Query query = session.createQuery("from Customer where userName='zhangsan'");

public static void main(String[] aStrings) {

Session session = SessionUtil.openSession();

Transaction transaction = session.beginTransaction();

// 获取query引用，这里用的还是Pojo对象不是表名，条件中的可以是表的字段名也可以是Pojo成员名，推荐是Pojo成员名

Query query = session.createQuery("from Customer where userName='zhangsan'");

// Query query = session.createQuery("from Customer where id=1);

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

for (Customer customer : list) {

System.out.println("查询到的数据信息为：");

System.out.println(customer.getId() + " " + customer.getPasswd());

}

session.close();

}

如果确定一个查询结果最多一个结果，推荐使用uniqueResult()，取代list()，可以在查找到一个结果后及时返回结果。

#### criteria

criteria不太常用，主要是面向Pojo成员添加条件限制。

private static void criteria() {

Session session = SessionUtil.openSession();

Transaction transaction = session.beginTransaction();

Criteria criteria = session.createCriteria(Customer.class).add(Restrictions.like("userName", "zhang%"))

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

List<Customer> list = criteria.list();

for (Customer customer : list) {

System.out.println("查询到的数据信息为：");

System.out.println(customer.getId() + " " + customer.getUserName() + " " + customer.getPasswd());

}

session.close();

}

### 修改

Update

## 事务

Hibernate的操作除了查询之外，其他都必须使用事务进行操作，基本方式如下。

Session session = MySessionFactory.getSessionFactory().openSession();

Transaction transaction = null;

try {

transaction = session.beginTransaction();

// do something

transaction.commit();

} catch (Exception e) {

if (transaction != null) {

transaction.rollback();

}

} finally {

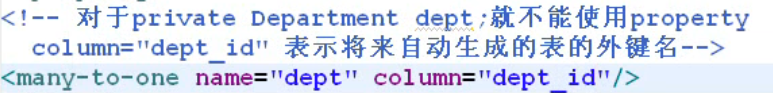
session.close();

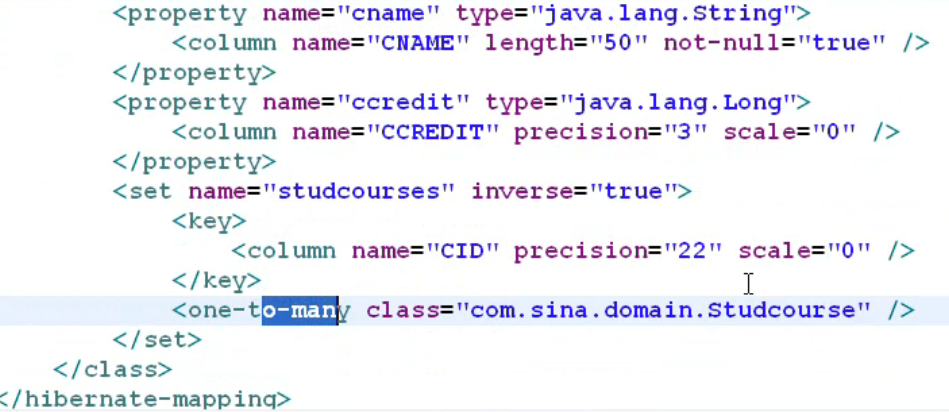
}

## Hibernate表格关联

Hibernate的表格关联，在pojo中引用的外键变成引用对象，关系包括one-to-many，many-to-one，one-to-one。相应的属性配置就不能用<property>。

多个雇员属于一个部门的多对一情况下，配置部门如下。





比如Customer拥有外键IdCard：

Customer Pojo中的外键是集合：private Set <IdCard> IdCard;

可以获取一个customer引用的所有外键：Set <IdCard> list = customer.getIdCard();

可以直接将student，course，grade三张表进行关联查询，只需在Studcourse的Pojo中配置对相应的字段。

List<Object[]> list = session.createQuery(“Select student.sname,cource.cname,grade from Studcourse”)

Hibernate存在懒加载现象，即默认情况下，返回的只是对象的普通属性，当用户使用对象属性（外键属性）的时候，才会向数据库再发出一次请求，这就是lazy加载。解决方法可以有：

M1.显示初始化hibernate.initized（代理对象）

M2.修改mapping文件lazy=false

M3.web项目可以过滤器，openSessionView

# HQL-Hibernate Query Language

HQL是hibernate官方推荐的操作语言，可以弥补单薄的save、delete、query等操作方法。

HQL与SQL不同，HQL区分大小写（仅区分java类和属性），HQL查找的是对象不是表，且支持多态，HQL主要通过query进行操作。

public static void main(String[] args) {

Session session = SessionUtil.getCurrentSession();

Transaction transaction = session.beginTransaction();

// 检索所有的customer，是从对象

List<Customer> list = session.createQuery("from Customer").list();

for (Customer customer : list) {

System.out.println(customer.getUserName() + " " + customer.getId());

}

transaction.commit();

if (transaction != null) {

transaction.rollback();

}

session.close();

}

HQL的query与SQL相似，但是注意查询对象是Pojo对象不是表。HQL的条件、排序等都和SQL语法类似。如where、order by、group by、函数等。Hibernate推荐选取所有属性，选取部分属性的话使用Object[]存储一个对象，但如果只有一个属性，获取的时候是Object[],，输出的时候是Object.toString而不是数组了。

Transaction transaction = session.beginTransaction();

// 检索所有的customer，是从对象

List<Object[]> list = session.createQuery("select userName,passwd from Customer where id = 2").list();

for (Object[] customer : list) {

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

}

System.out.println();

transaction.commit();

session.close();

## HQL分页

分页的时候可以使用setFirstResult和setMaxResults设置起止数据条数。

session.createQuery(“…”).setFirstResult(10).setMaxResults(20).list()

分页时后台首先查找所有数据总数，再查找，hibernate一共会进行3次过滤。

## HQL参数绑定

参数绑定的好处：可读性好、性能提高、防止sql注入。

常用方法：

Query query1 = session.createQuery("from Customer where id=:a");

query1.setString("a","2");

Query query2 = session.createQuery("from Customer where id=?");

query2.setString(0,"2");

可以用循环setString的方法设置多个参数。

## HQL语句放到xml文件中

一些经常变化的HQL语句就可以放到Pojo的xml中，通过session.getNamedQuery()获取这个Query。一般不太常用。

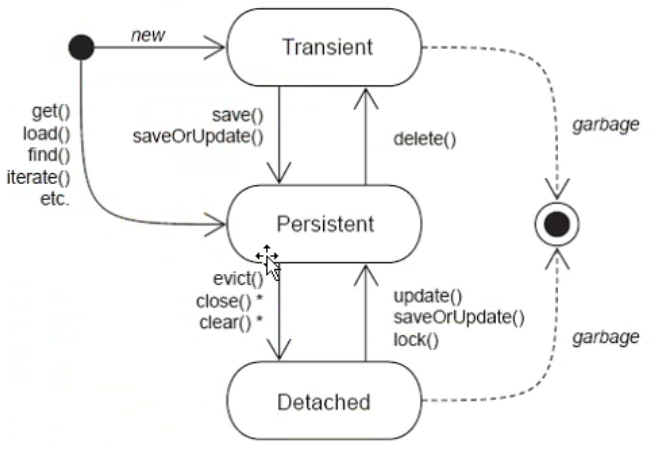
# Hibernate数据状态

根据数据是否在数据库中存在以及是否处于session管理之下，数据对象的状态分为三种：瞬时态、持久态、脱管态（游离态）。

瞬时态transient：不处于session管理下，在数据库中没有对应的数据。超过作用域会被gc，一般是new出来且没与session关联的对象。

持久态persistent：处于session管理下，在数据库中有对应的记录。操作并commit后修改数据库。

脱管态detached：不处于session管理下，在数据库中有相应的记录。



数据状态转移图

# Hibernate缓存

Hibernate有一级缓存（session级缓存）与二级缓存（SessionFactory共享缓存）。

一级缓存就是不需要配置，使用get、save等方法的时候会自动放入缓存，但是缓存空间的大小没有保护机制，需要程序员自己使用session.clear()、evict等方法清除session缓存防止内存溢出。

一级缓存可以使用hashmap来模拟缓存实现的机制：将key与Object放到Map中，查数据库的时候先查看Map中是否存在，没有则查数据库并添加到map。

二级缓存需要配置，hibernate本身的二级缓存机制没有第三方的Hashtable、EHCache、OSCache等缓存机制做的好。Hashtable属于内存缓存，EHCache、OSCache会使用内存兼硬盘进行缓存。比如使用OSCache，需要在hibernate.cfg.xml中配置启用相应的OSCache，设置默认的OSCache.xml内容，在hibernate中可以通过sessionFactory获取其statistics统计对象信息等。

# Eclipse的hibernate开发使用

安装hibernate支持插件，配置相应的包名等，即可自动创建hibernate的配置文件、Pojo等代码。如新建一个maven项目，选择webapp模板，并设置默认jar修复文件默认目录，然后创建hibernate ORM框架。

对应的jsp创建也可以使用插件或者框架实现，如struts框架插件。一般来说经常使用创建一个web工程，插入struts框架（或者jsp+servlet框架），再插入hibernate框架，再通过引入spring框架进行全局管理。