# 1.接着1.1中的东西我们继续，现在我们希望可以修改数据库Employee中的东西，

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

*addEmployee*();

//现在开始修改数据库中的表的数据,和下面的东西是一样的但是这个需要占用很多的内存

//所以我们将这个SessionFactory制作成单态的,通过制作一个工具类

}

**private** **static** **void** addEmployee() {

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

SessionFactory sf=cf.buildSessionFactory();

Session session=sf.openSession();

Transaction ts=session.beginTransaction();

Employee u=**new** Employee();

u.setEmail("1318830916@qq.com");

u.setHiredate(**new** Date());

u.setName("zhangyujin");

//insert

//保存，

//这个就是持久化对象了

session.save(u);//添加该对象到数据库 //等于==inset into 。。被hibernate封装，将来不管用什么数据库都是可以试实现的

ts.commit();//提交事务

session.close();

}

}

# 2.下面的操作就是 使用一个hibernate 保证只有一个SessionFactory，如果实在是项目很庞大，数据库很多，可以使用很多，但是太多项目会撑不住的，一个数据库对应一个SessionFactory即可

**package** com.hlj.util;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.cfg.Configuration;

//使用final 不能被实例化了

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

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

**private** **static** Configuration *configuration*=**null**;

**static**{

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

*sessionFactory*=*configuration*.buildSessionFactory();

}

**private** MySessionFactory(){};

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

**return** *sessionFactory*;

}

}

# 3.然后我们就开始修改用户了

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

*addEmployee*();

//现在开始修改数据库中的表的数据,和下面的东西是一样的但是这个需要占用很多的内存

//所以我们将这个SessionFactory制作成单态的,通过制作一个工具类

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

//修改用户Employee.java先获取用户，然后修改

//load是可以通过主键来获取表的记录

Employee employee =(Employee)session.load(Employee.**class**, 3);

}

# 4.为了让我我们的Employee//这个pojo，domain,javabean按照标准序列应该序列化，目的是可以唯一标识改对象，同时可以在网络和文件传输

//就是让他继承implements Serializable

**public** **class** Employee **implements** Serializable{

/\*\*

\*

\*/

**private** **static** **final** **long** *serialVersionUID* = 1L;

# 5.然后到了TestMain 中进行修改

**public** **class** TestMain {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

*addEmployee*();

//现在开始修改数据库中的表的数据,和下面的东西是一样的但是这个需要占用很多的内存

//所以我们将这个SessionFactory制作成单态的,通过制作一个工具类

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

Transaction ts=session.beginTransaction();

//修改用户Employee.java先获取用户，然后修改

//load是可以通过主键来获取表的记录

Employee employee =(Employee)session.load(Employee.**class**, 3);

//下面这个就是相当于更新了

employee.setEmail("13151515@qq.com");

ts.commit();

session.close();

}

# 6.然后运行之后就会产生下面的东西了

1106 [main] INFO org.hibernate.cfg.SettingsFactory - Default entity-mode: pojo

1106 [main] INFO org.hibernate.cfg.SettingsFactory - Named query checking : enabled

1111 [main] INFO org.hibernate.impl.SessionFactoryImpl - building session factory

1114 [main] INFO org.hibernate.impl.SessionFactoryObjectFactory - Not binding factory to JNDI, no JNDI name configured

Hibernate: select employee0\_.id as id2\_0\_, employee0\_.name as name2\_0\_, employee0\_.email as email2\_0\_, employee0\_.hiredate as hiredate2\_0\_ from employee employee0\_ where employee0\_.id=?

Hibernate: update employee set name=?, email=?, hiredate=? where id=?

# 7.这样我们就是完成了更新了，下面开始删除

**public** **class** TestMain **implements** updateEmployee {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

/\* addEmployee();

updateEmployee();\*/

//然后我们开始删除的操作了

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

Transaction ts=session.beginTransaction();

Employee employee = (Employee)session.load(Employee.**class**, 2);

session.delete(employee);

ts.commit();

session.close();

}

# 8.删除完成之后出现下面的东西

731 [main] INFO org.hibernate.cfg.SettingsFactory - Deleted entity synthetic identifier rollback: disabled

731 [main] INFO org.hibernate.cfg.SettingsFactory - Default entity-mode: pojo

731 [main] INFO org.hibernate.cfg.SettingsFactory - Named query checking : enabled

767 [main] INFO org.hibernate.impl.SessionFactoryImpl - building session factory

951 [main] INFO org.hibernate.impl.SessionFactoryObjectFactory - Not binding factory to JNDI, no JNDI name configured

Hibernate: select employee0\_.id as id0\_0\_, employee0\_.name as name0\_0\_, employee0\_.email as email0\_0\_, employee0\_.hiredate as hiredate0\_0\_ from employee employee0\_ where employee0\_.id=?

Hibernate: delete from employee where id=?

# 9. 数据库也是可以的

select employee0\_.id as id0\_0\_, employee0\_.name as name0\_0\_, employee0\_.email as email0\_0\_, employee0\_.hiredate as hiredate0\_0\_ from employee employee0\_ where employee0\_.id=1;

+--------+------------+-------------------+--------------+

| id0\_0\_ | name0\_0\_ | email0\_0\_ | hiredate0\_0\_ |

+--------+------------+-------------------+--------------+

| 1 | zhangyujin | 1318830916@qq.com | 2015-04-18 |

+--------+------------+-------------------+--------------+

1 row in set

mysql>

# 10以上是根据id来查询的，但是上面是不能的，这里不做解释