Java 应用框架 Hibernate

实验指导书

软件学院

实验一 Hibernate 基础配置

实验目的

帮助学生理解掌握如何使用 Hibernate 进行持久层开发；
帮助学生了解 3 层架构的意义：逻辑层、数据持久层和数据库。

技术要点

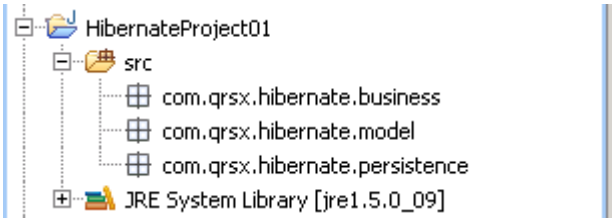
- JDBC 基础知识；
- ORM 基础知识。

实验说明

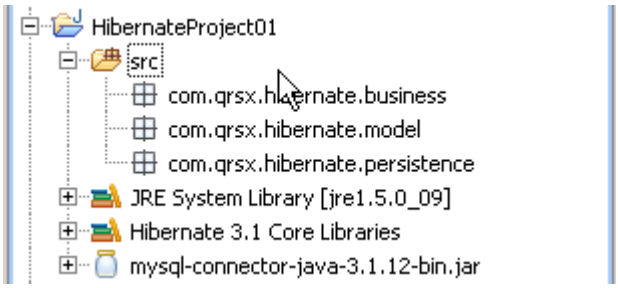
本例分为 3 层，分别是业务逻辑层、数据持久层和数据库。
业务逻辑层：处理业务逻辑，比如将一个学生对象增加、删除和修改。
数据持久层：屏蔽数据库操作，向业务逻辑层提供 API。对于业务逻辑层来说，看到的只是对象的概念，而诸如字段、数据表、主键等比较底层的东西都是透明的。
数据库：数据持久化的场所，长期存储数据。

实验步骤

1. 运行 MyEclipse5.0GA（或更高版本），新建一个“Java Project”，项目名称为“HibernateProject01”，添加源码包“src”，然后在 src 中加入类包（操作步骤详见《Eclipse 使用手册》），项目结构如下



2. 加入 Hibernate 类包的支持，将鼠标放到项目名“HibernateProject01”上，点击右键，选择“MyEclipse－>Add Hibernate Capabilities”加入 Hibernate 类包的支持。
3. 本实验及其以后的实验，都采用 MySql 数据库，因此需要 mysql-connector-java-3.1.12-bin.jar 的支持，将其放到 HibernateProject01 下面，然后将其设置为项目类库（操作步骤详见《Eclipse 使用手册》）。
4. 以上 3 部设定完毕后，项目结构如下：



5. 启动 MySql 服务，在 MySql 中创建数据库“HibernateProject01”，然后创建学生数据表 Student_table，表结构如下：

字段	数据类型	说明
id	Vacher（50）	主键
name	Varcher（100）	学生名
cardId	Varchar（15）	学号
age	Int	年龄

创建 Student 数据表的 sql 脚本如下：

Student.sql：

```
CREATE TABLE `student_table` (
  `id` varchar(255) NOT NULL,
  `name` varchar(255) default NULL,
  `cardId` varchar(255) default NULL,
  `age` tinyint(4) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

6. 在“com.qrsx.hibernate.model”中创建学生实体对象 Student.java，代码如下：

Student.java：

```
/*
 * @(#)Student.java Jan 25, 2007
 * Copyright 2007 qingdaosoftware, Inc. All rights reserved
 */
package com.qrsx.hibernate.model;

/**
 *
 * Company : 青软实训<br>
 * Author  : 王希涛<br>
 * Version : 1.0<br>
 * Date    : Jan 25, 2007<br>
 */
```

```java
public class Student {

    private String id;
    private String name;
    private String cardId;
    private Integer age;

    /**
     * @return the age
     */
    public Integer getAge() {
        return age;
    }
    /**
     * @param age the age to set
     */
    public void setAge(Integer age) {
        this.age = age;
    }
    /**
     * @return the cardId
     */
    public String getCardId() {
        return cardId;
    }
    /**
     * @param cardId the cardId to set
     */
    public void setCardId(String cardId) {
        this.cardId = cardId;
    }
    /**
     * @return the id
     */
    public String getId() {
        return id;
    }
    /**
     * @param id the id to set
     */
    public void setId(String id) {
        this.id = id;
    }
    /**
```

```
     * @return the name
     */
    public String getName() {
        return name;
    }
    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }
}
```

7. 在“com.qrsx.hibernate.persistence”中编写取得 Session 的类 HibernateUtil.java：

HibernateUtil.java：

```
package com.qrsx.hibernate.persistence;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

/**
 *
 * Company : 青软实训<br>
 * Author  : 王希涛<br>
 * Version : 1.0<br>
 * Date    : Jan 25, 2007<br>
 */
public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new Configuration().configure()
                    .buildSessionFactory();
        } catch (HibernateException ex) {
            ex.printStackTrace();
            throw new RuntimeException("Exception building
SessionFactory: "
                    + ex.getMessage(), ex);
        }
```

```
    }

    public static Session currentSession()
                        throws HibernateException {
        Session s = sessionFactory.openSession();
        return s;
    }

    public static void closeSession(Session s) {
        s.close();
    }
}
```

8. 在“com.qrsx.hibernate.persistence”类包中编写操作数据库的 Java 文件 StudentDAOImp.java，代码如下：

StudentDAOImp.java：

```
package com.qrsx.hibernate.persistence;

import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.qrsx.hibernate.model.Student;

public class StudentDAOImp {
    static Session session = null;

    /*------------创建新的学生对象-----------------*/
    public static void createStu(Student stu) {
        try {
            session = HibernateUtil.currentSession(); // 开启连接
            Transaction tx = session.beginTransaction(); // 开启事务
            session.save(stu);
            tx.commit();

        } catch (HibernateException e) { // 捕捉例外
            e.printStackTrace();
        } finally {
            HibernateUtil.closeSession(session);
        }
```

```
    }

    /*------------删除学生对象-----------------*/
    public static void delStu(String id) {
        try {
            session = HibernateUtil.currentSession(); // 开启连接
            Transaction tx = session.beginTransaction(); // 开启事务
            Student stu = (Student) session.get(Student.class, id);
            session.delete(stu);
            tx.commit();

        } catch (HibernateException e) { // 捕捉例外
            e.printStackTrace();
        } finally {
            HibernateUtil.closeSession(session);
        }
    }

    /*------------修改学生对象-----------------*/
    public static void mdfStu(Student stu) {
        try {
            session = HibernateUtil.currentSession(); // 开启连接
            Transaction tx = session.beginTransaction(); // 开启事务
            session.update(stu);
            tx.commit();

        } catch (HibernateException e) { // 捕捉例外
            e.printStackTrace();
        } finally {
            HibernateUtil.closeSession(session);
        }
    }

    /*------------取得所有的学生列表-----------------*/
    public static List getAllStu() {
        List list = null;
        try {
            session = HibernateUtil.currentSession(); // 开启连接
            Transaction tx = session.beginTransaction(); // 开启事务
            Query q = session.createQuery("from Student");
            list = q.list();
            tx.commit();

        } catch (HibernateException e) { // 捕捉例外
```

```
            e.printStackTrace();
        } finally {
            HibernateUtil.closeSession(session);
        }
        return list;
    }
}
```

在 com.qrsx.hibernate.model 类包中，对 Student.java 文件创建一个 Hibernate 映射文件 Student.hbm.xml，代码如下：

Student.hbm.xml：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>

    <!--把类和数表关联起来-->
    <class name="com.qrsx.hibernate.model.Student"
           table="student_table">

    <!--id的产生方式是uuid.hex-->
        <id name="id" unsaved-value="null">
            <generator class="uuid.hex" />
        </id>

        <!--映射学号-->
        <property name="cardId"  type="string"  />

        <!--映射名字-->
        <property name="name"   type="string" />

        <!--映射学生岁数-->
        <property name="age" type="int" />

    </class>
</hibernate-mapping>
```

9. 配置 Hibernate 的描述文件。在 src 源码包中新建 hibernate.cfg.xml，加入如下配置代码：

Hibernate.cfg.xml：

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd
">

<!-- Generated by MyEclipse Hibernate Tools.                   -->
<hibernate-configuration>

<session-factory>
    <property name="connection.username">root</property>
    <property name="connection.password">browser</property>
    <property name="connection.url">
        jdbc:mysql://localhost:3306/HibernateProject01
    </property>
    <property name="dialect">
        org.hibernate.dialect.MySQLDialect
    </property>
    <property name="connection.driver_class">
        com.mysql.jdbc.Driver
    </property>
    <property name="show_sql">true</property>

    <mapping resource="com/qrsx/hibernate/model/Student.hbm.xml" />

</session-factory>

</hibernate-configuration>
```

10. 配置 log4j，辅助 Hibernate 的日志输出，在 src 源码包中新建 log4j.properties，加入以下代码：

Log4j.properties：

```
log4j.rootLogger=info,CONSOLE
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.Target=System.out
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern= %4p [%t] (%F:%L) -
%m%n
```

11. 在 com.qrsx.hibernate.business 类包中编写业务逻辑处理文件 StudentManager.java

StudentManager.java：

```java
/*
 * @(#)StudentManager.java Jan 25, 2007
 * Copyright 2007 qingdaosoftware, Inc. All rights reserved
 */
package com.qrsx.hibernate.business;

import java.util.List;

import com.qrsx.hibernate.model.Student;
import com.qrsx.hibernate.persistence.DAOImp;

/**
 *
 * Company : 青软实训<br>
 * Author  : 王希涛<br>
 * Version : 1.0<br>
 * Date    : Jan 25, 2007<br>
 */
public class StudentManager {

    public static void main(String[] args){
        StudentManager sm = new StudentManager();
        sm.create();
        sm.update();
        sm.delete();
    }

    /**
     * 创建新的学生对象
     */
    private void create(){
      Student stu = new Student();
      stu.setName("tomclus");
      stu.setCardId("200512345");
      stu.setAge(33);
      DAOImp.createStu(stu);
    }

    /**
     * 修改学生对象
     */
    private void update(){
      List list = DAOImp.getAllStu();
      Student stu = (Student) list.get(0);
```

```
        stu.setName("newName");
        stu.setCardId("123");
        //此为业务逻辑方法
        if (stu.getAge() < 18) {
            stu.setAge(18);
        }
        DAOImp.mdfStu(stu);
    }

    /**
     * 删除学生对象
     */
    private void delete(){
        List list = DAOImp.getAllStu() ;
        Student stu=(Student) list.get(0);
        DAOImp.delStu(stu.getId());
    }
}
```

12. 运行 StudentManager，Eclipse 的控制台中执行结果如下：

执行结果：

```
执行create():
Hibernate: insert into student_table (cardId, name, age, id) values
(?, ?, ?, ?)

执行update():
Hibernate: select student0_.id as id0_, student0_.cardId as cardId0_,
student0_.name as name0_, student0_.age as age0_ from student_table
student0_
Hibernate: update student_table set cardId=?, name=?, age=? where id=?

执行delete():
Hibernate: select student0_.id as id0_, student0_.cardId as cardId0_,
student0_.name as name0_, student0_.age as age0_ from student_table
student0_
Hibernate: select student0_.id as id0_0_, student0_.cardId as
cardId0_0_, student0_.name as name0_0_, student0_.age as age0_0_ from
student_table student0_ where student0_.id=?
Hibernate: delete from student_table where id=?
```

实验二：Hibernate 中的数据关联

实验案例 1：实体化对象状态

实验目的

1. 将持久状态对象转换成临时状态对象
2. 体验 flush（ ）
3. 体验 evit()方法
4. 将持久态对象转换成游离态对象

实验要点

1. 持久态对象转换临时态对象
2. flush（ ）方法
3. evit()方法
4. 持久态对象转换成游离态对象

实现效果

1. 持久态对象转换临时态对象运行结果：

   a) Ecplise 中运行结果：

```
Hibernate: select empl0_.id as id1_0_0_, empl0_.na
Empl [id=1, name=jack, age=38, dId=2]
Hibernate: delete from empl where id=?
Empl [id=1, name=jack, age=38, dId=2]
```

   b) 程序执行完数据库中查询结果：

ID	NAME	AGE	DEPTID
2	Tom	40	1
3	Mike	28	1

2. flush 方法：

   a) Ecplise 中运行结果：

```
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select seq_empl.nextval from dual
Hibernate: insert into empl (name, deptId, age, id) values (?, ?, ?, ?)
```

b) 程序执行完数据库中查询结果：

ID	NAME	AGE	DEPTID
2	Tom	40	1
3	Mike	28	1

3. evit 方法：

Ecplise 中运行结果：

```
2014-7-11 11:19:24 org.hibernate.hql.internal.ast.A
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select seq_empl.nextval from dual
```

4. clear 方法：

```
2014-7-11 11:23:07 org.hibernate.hql.internal.ast.A
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select seq_empl.nextval from dual
```

实现步骤

1. 搭建环境
   a) 新建 Empl（员工信息）表，（相关建表语句所在目录：”第二章 hibernate 操纵对象\第二节 实体对象状态\实验指导手册\环境工程包\empl 建表语句.txt”）（如本地数据库中已存在该表可直接拿来使用）：
      i. 设置 id 为主键。
      ii. 创建表时需新建一个序列，用于 id 字段自增长：
         1. 序列名：seq_empl，序列每次增加 1，最小为 1，最大为 999，并且不设置缓存。
      iii. Empl 表结构：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	28	1

2. 导入工程：
   a) 将“第二章 hibernate 操纵对象\第二节 实体对象状态\实验指导手册\环境工程包“目录下的 HibernateDemo10 文件导入到 ecplise 中。

3. 进行实验
   a) 持久态对象转变为临时态对象：
      i. 打开 test 包下的 Test 类。
      ii. 在 Session session=sf.openSession();下面一行调用 session.get()方法查询得到 id 为 1 的 Empl 对象，命名为 empl（此时的对象为持久态）。
      iii. 调用 empl 的 toString（ ）方法，并且打印输出，观察结果。
      iv. 调用 session 的 beginTransaction()方法，开启事务，返回一个 Transaction 对象，命名为 tran。
      v. 调用 session 的 delete（ ）方法删除之前所得到的持久状态的对象 empl。
      vi. 关闭 session。
      vii. 调用 empl 的 toString（ ）方法，并打印输出。观察结果。
      viii. 打开数据库，在数据库中查询 empl 表。
      结果说明：此时在 ecplise 中仍然可以输出对象属性，但是数据库中已不存在 id 为 1 的员工信息。此时的 empl 对象为临时态对象。
   b) flush 方法：
      i. 将 Session session=sf.openSession()之后的代码注释掉。
      ii. new 一个 Empl 对象。（对象属性为：Id 为 4，name 为 MAYYR,age 为 18，did 为 2）
      iii. 调用 session 的 save（ ）方法将 empl 对象纳入 session 管理。
      iv. 调用 session 的 flush（ ）方法，执行程序。
      v. 查看数据库，此时并未发现数据库新增数据。
      vi. 调用 Transaction 的 commit 方法。执行程序。再次查看数据库结果
   c) evit 方法
      i. 将 session.flush()注释掉，调用 session 的 evit（empl）方法。表示将 empl 逐出 session 缓存
      ii. 运行程序。
   d) 持久对象转换为游离对象。
      i. 将 evit 方法的这行代码注释。
      ii. 调用 session 的 clear（ ）方法。
      iii. 运行程序

实验案例 2：session 接口

目的

1 使用 merge 将临时态的对象插入到数据库

2 使用 replicate 进行相关操作

要点

1．merge 方法

2.replicate 方法

实现效果

5. merge 方法：

a) Ecplise 中运行结果：

```
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select seq_empl.nextval from dual
Empl [ name=john, age=23, dId=1]
Hibernate: insert into empl (name, deptId, age, id) values (?, ?, ?, ?)
```

b) 程序执行完数据库中查询结果：

10	TERRY	28	2
2	Tom	35	1
3	Mike	28	1
9	DEMO	19	1
13	john	23	1

6. replicate 的应用：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	48	1

实现步骤

4. 搭建环境
   a) 新建 Empl（员工信息）表，（相关建表语句所在目录：第二章 hibernate 操纵对象\第二节 实体对象状态\实验指导手册\环境工程包\实验 002 session 接口方法\empl 建表语句.txt）（如本地数据库中已存在该表可直接拿来使用）：
      i. 设置 id 为主键。
      ii. 创建表时序新建一个序列，用于 id 字段自增长：
         1. 序列名：seq_empl，序列每次增加 1，最小为 1，最大为 999，并且不设置缓存。
      iii. Empl 表结构：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	28	1

5. 导入工程：
   a) 将“第二章 hibernate 操纵对象\第二节 实体对象状态\实验指导手册\环境工程包\实验案例 002 session 接口“目录下的 HibernateDemo12 文件导入到 ecplise 中。
6. 进行实验
   a) merge 方法：
      i. 打开 test 包下的 Test 类。在 Session session=sf.openSession()下一行调用 session 的 beginTransaction()方法开启事务，返回一个 Transaction 对象 tran。
      ii. 创建一个临时态的 Empl 对象 empl：new Empl("JOHN",23,1);
      iii. 使用 session 的 merge 方法 ,将创建的 Empl 临时态对象插入数据库，返回一个新的 Empl 对象 newEmpl。
      iv. 调用 newEmpl 的 toString（）方法，并作为输出结果打印输出。提交事务并关闭 session。
      v. 运行程序，观察控制台结果，并查询数据库中数据。
   b) replicate：
      i. 创建一个新的数据库 mrdb1，在该数据库中新建 Empl 表。（相关建表语句所在目录：第二章 hibernate 操纵对象\第二节 实体对象状态\实验指导手册\资源\实验 002 session 接口方法\empl 建表语句.txt）：
         1. 设置 id 为主键。
         2. 创建表时序新建一个序列，用于 id 字段自增长：

a) 序列名：seq_empl，序列每次增加 1，最小为 1，最大为 999，并且不设置缓存。

3. Empl 表结构：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	28	1

ii. 在 src 目录下添加一个配置文件：hibernateD.cfg.xml,将 url 这一属性中数据库名改为：mrdb1。其余配置与 hibernate.cfg.xml 一致，从 hibernate.cfg.xml 里复制过来即可。

iii. 找到并打开 test 包下的 Test1 类，在 Transaction tr1=session1.beginTransaction() 下一行调用 session1 的 get（Empl.class,3）方法，获得 id 为 3 的 Empl 对象，命名为 empl。

iv. 调用 empl 的 setAge（int arg）方法，将该对象的 age 改为 48。

v. 调用 tr1 的 commit（ ）方法提交事务。

vi. 调用 session1 的 close（ ）方法，关闭 session1。

vii. 创建第二个数据源，如下所示：

```
Configuration con2 = new Configuration();
con2.configure("/hibernateD.cfg.xml");
SessionFactory sf2=con2.buildSessionFactory();
Session session2 = sf2.openSession();
```

viii. 在 Session session2 = sf2.openSession()下一行调用 session2 的 beginTransaction（ ）方法，开启事务，返回 Transaction 的对象，命名为 tr2。

ix. 调用 session2 的 replicate（Object arg，ReplicationMode arg1）方法将 mrdb 数据库中的 empl 对象复制到 mrdb1 数据库中。参数说明：

1. arg：empl（需要复制的对象）
2. arg1：ReplicationMode.LATEST_VERSION（复制的模式）

x. 调用 tr2 的 commit（ ）方法提交事务。

xi. 调用 session2 的 close（ ）方法，关闭 session2。

xii. 运行 Test1，并打开 mrdb1 数据库，查看 empl 表的数据

实验三： Hibernate 的高级特性

映射值类型集合(Set 元素)

实验目的

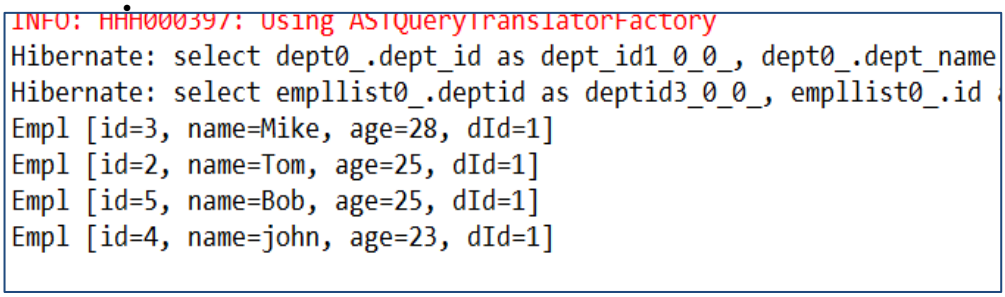
1.添加 set 元素使程序正常运行

实验要点

1. 映射 set 的应用

实现效果

a) 运行结果：



实现步骤

2 搭建环境

c) 新建 Empl（员工信息）表以及 Dept（部门信息）表（相关建表语句所在目录：第三章 映射值类型集合及实体关联关系\第一节 映射类型集合\实验指导手册\环境工程包\建表语句.txt）：

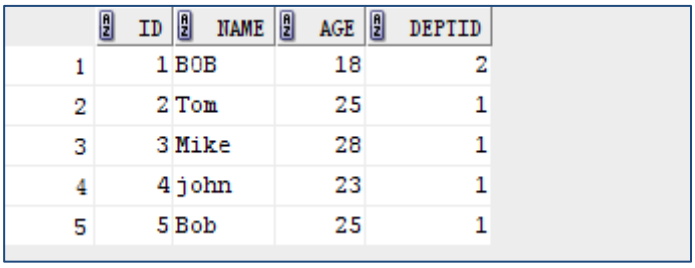
i. 设置 id 为主键。

ii. 创建表时序新建序列，用于 id 字段自增长：

1. 序列名：seq_empl，序列每次增加 1，最小为 1，最大为 999，并且不设置缓存。
2. 序列名：seq_dept，序列每次增加 1，最小为 1，最大为 999，并且不设置缓存。

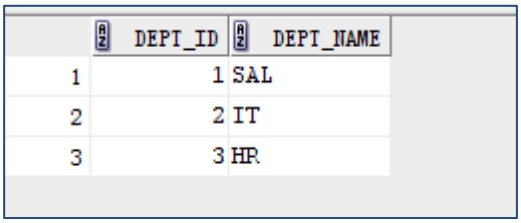
iii. 相关表结构

1. Empl 表结构：



	ID	NAME	AGE	DEPTID
1	1	BOB	18	2
2	2	Tom	25	1
3	3	Mike	28	1
4	4	john	23	1
5	5	Bob	25	1

2. Dept 表结构：



	DEPT_ID	DEPT_NAME
1	1	SAL
2	2	IT
3	3	HR

d) 导入工程：

i. 将“第三章　映射值类型集合及实体关联关系\第一节　映射类型集合\实验指导手册\环境工程包“目录下的 HibernateDemo14 文件导入到 ecplise 中。

3 进行实验

e) 映射 Set：

i. 找到并打开 Dept 实体类的映射文件 Dept.hbm.xml ,在 class 节点内添加 set 元素。

1. Set 元素的结构如下：

```
<set name="XXX(映射的set对象)" table="XXX（对象相关数据表名）"
lazy="false" inverse="true" cascade="all">
<key column="XXX(关联字段)"/>
 <one-to-many class="XXX(一对多，对应多的一方的实体类)"/></set>
```

2. Set 元素内的属性说明：

a) name: emplList。

b) table: empl。

c) key column: deptid。

d) one-to-many class：com.oracle.entity.Empl。

4 运行程序并观察结果。

实验案例 2：映射一对多的应用

实验目的

1 使用 set 元素进行单向一对多关联查询员工信息

实验要点

1. 实体类之间的关联属性。
2．配置映射一对多的关联。
3．使用 set 元素。

实现效果

```
Hibernate: select dept0_.dept_id as dept_id1_0_0_, dept
Hibernate: select empllist0_.deptid as deptid3_0_0_, em
Empl [id=3, name=Mike, age=28, dId=1]
Empl [id=2, name=Tom, age=25, dId=1]
Empl [id=5, name=Bob, age=25, dId=1]
Empl [id=4, name=john, age=23, dId=1]
```

4.搭建环境

f) 新建 Empl（员工信息）表以及 Dept（部门信息）表，（相关建表语句所在目录：第三章　映射值类型集合及实体关联关系\第一节　映射类型集合\实验指导手册\环境工程包\建表语句.txt）：

i. 设置 id 为主键。

ii. 创建表时序新建序列，用于 id 字段自增长：

1. 序列名：seq_empl，序列每次增加 1，最小为 1，最大为 999，并且不设置缓存。
2. 序列名：seq_dept，序列每次增加 1，最小为 1，最大为 999，并且不设置缓存。

iii. 相关表结构

1. Empl 表结构：

	ID	NAME	AGE	DEPTID
1	1	BOB	18	2
2	2	Tom	25	1
3	3	Mike	28	1
4	4	john	23	1
5	5	Bob	25	1

2. Dept 表结构

	DEPT_ID	DEPT_NAME
1	1	SAL
2	2	IT
3	3	HR

g) 导入工程：

i. 将“第三章　映射值类型集合及实体关联关系\第二节 映射实体关联关系\实验指导手册\环境工程包“目录下的 HibernateDemo16 文件导入到 ecplise 中。

5.进行实验

h) 实体类：

i. 找到并打开 entity 包下的 Dept 类。

ii. 在 Dept 中添加一个 Empl 类型的 HashSet 的集合属性，添加 set、get 方法

i) 配置文件：

i. 找到并打开 Dept.hbm.xml，在 class 节点内添加 set 元素。

1. set 元素内结构如下：

```
<set name="XXX(映射的set对象)"  table="XXX（对象相关数据表名）"
lazy="false"  inverse="true"  cascade="all">
<key column="XXX(关联字段)"/>
 <one-to-many class="XXX(一对多，对应多的一方的实体类)"/>
  </set>
```

2. set 元素内的属性说明：

a) name: emplList。

b) table: empl。

c) key column: deptid。

d) one-to-many class：com.oracle.entity.Empl。

j) 实现方法：

i. 找到并打开 dao 包下的 DeptImpl 类，在 Session session=sf.openSession()下一行调用 session 的 get(clz, deptId)方法的到一个 Dept 类的对象，命名为 dept。

ii. 调用 dept 的 getEmplList()方法获取一个 Empl 的 List 集合对象，命名为 emplList，将其作为返回结果。

k) 测试结果：

i. 找到 test 包下的 Test1 类，使用 DeptImpl 实例化对象。

ii. 使用实例化得到的对象调用 queryEmplByDept 方法。（方法中的参数为：Dept.class　1），得到 Empl 类的集合对象。

iii. 遍历输出集合对象

iv. 运行程序观察结果

实验案例 3：映射单向多对多的应用

实验目的

1 通过使用多对多关系映射，查询学生信息。

实验要点

1 实体类之间的关联属性。

2 配置映射多对多的关联。

实现效果

```
Hibernate: select course0_.id as id1_0_0_, course0_.coursename as coursena2_0
Hibernate: select studentlis0_.course_id as course_i1_0_0_, studentlis0_.stud
Student [id=20140601, name=Mary]
Student [id=20140101, name=Tom]
```

实现步骤

1 搭建实验环境：

l) 创建 student 表和 course 表以及 studentandcourse 表。其中 studentandcourse 表中的 student_id 字段与 student 中的 id 字段为外键约束；studentandcourse 表中的 course_id 字段与 course 中的 id 字段为外键约束（相关建表语句在“第三章　映射值类型集合及实体关联关系\第二节 映射实体关联关系\实验指导手册\环境工程包”内）。

说明：一个课程可以有多个学生选择，一个学生也可以选择多个课程

m) 导入工程：

i. 将“第三章　映射值类型集合及实体关联关系\第二节 映射实体关联关系\实验指导手册\环境工程包“目录下的 HibernateDemo17 文件导入到 ecplise 中。

2 进行实验：

n) 找到 entity 包下的 Course 实体类，在 Course 类下添加一个 Student 的 Set 集合,添加 set，get 方法

o) 找到 entity 包下的配置文件 course.hbm.xml（注意因为是 Course 对 Student 的多对多关联，所以需要在 course.hbm.xml 中添加 set 元素，并在里面添加 many-to-many 元素）。

i. set 元素内结构如下：

```
<set name="XXX(映射的set对象)"  table="XXX（第三方表名）"
  lazy="false"  cascade="all"  inverse="true" >
        <key column="XXX(关联字段)"
        foreign-key="XXX(当前表与第三方表的外键)"/>
    <many-to-many  column="XXX(第三方表中对应多的一方的
  外键)"class="XXX(对应多方的实体类类)"/>
</set>
```

ii. Set 元素内的属性说明：

1. name: studentList。
2. table: studentandcourse。
3. key column: course_id。
4. foreign-key：fk_course。
5. many-to-many class：student_id。
6. class：com.oracle.entity.Student

p) 在 Session session=sf.openSession()下一行调用 session 的 get(clz, courseId)方法得到 Course 类型的持久化对象，命名为 course。

q) 调用 course 的 getStudentList（ ）方法获得一个 Student 类型的 Set 集合，命名为 stuList，关闭 session，并将 stuList 作为返回结果。

3 测试程序：

r) 找到并且打开 ManyToMany 类

s) 使用 CourseImpl 实例化对象

t) 调用 queryStudentByCourse 方法通过对 id 为 1001 的 Course 对象查询，得到 Student 类型的 Set 集合

u) 使用增强型 for 循环遍历输出集合对象。

v) 运行程序观察结

实验四 Hibernate 配置实验

实验目的

掌握 Hibernate 的映射机制，并熟悉 Hibernate 的查询操作。

实验要点

搭建 hiberate 环境及所需要的数据库；
用 Hibernate 方式实现通过学校代码查询一个学校信息的功能。
实验步骤：
1、创建 web 工程；
2、添加 Struts2 框架所必须的 jar 包；
3、添加 Struts2 的核心配置文件 struts.xml；
4、配置 web.xml 文件；
5、创建数据源；
6、使用向导添加 Hibernate 功能；
7、映射数据表；
8、编写查询页面与结果页面；
9、编写数据存取类，实现数据查询；
10、编写业务逻辑类及 Action 类；
11、配置 stuts.xml；
12、部署试运行。

实验报告：

(1) 提交相关配置文件：

hibernate.cfg.xml：

```
<hibernate-mapping>
    <class name="com.wr.po.School" table="school" catalog="exp5db">
        <id name="schno" type="java.lang.String">
            <column name="schno" length="11" />
            <generator class="assigned" />
        </id>
        <property name="schname" type="java.lang.String">
            <column name="schname" length="100" not-null="true" />
        </property>
        <property name="schcity" type="java.lang.String">
            <column name="schcity" length="50" />
        </property>
        <property name="schage" type="java.lang.Integer">
```

```
            <column name="schage" />
        </property>
    </class>
</hibernate-mapping>
```

School.hbm.xml:

```
<hibernate-mapping>
    <class name="com.wr.po.School" table="school" catalog="exp5db">
        <id name="schno" type="java.lang.String">
            <column name="schno" length="11" />
            <generator class="assigned" />
        </id>
        <property name="schname" type="java.lang.String">
            <column name="schname" length="100" not-null="true" />
        </property>
        <property name="schcity" type="java.lang.String">
            <column name="schcity" length="50" />
        </property>
        <property name="schage" type="java.lang.Integer">
            <column name="schage" />
        </property>
    </class>
     </hibernate-mapping>
```

Struts.xml:

```
<struts>

    <package name="exp5" extends="struts-default" >
        <action name="findSchool" class="com.wr.action.SchoolAction"
method="findSchool">
            <result name="success">school.jsp</result>
            <result name="error">index.jsp</result>
        </action>
    </package>

     </struts>
```

Web.xml:

```
<filter>
        <filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFi
lter</filter-class>
    </filter>
```

```
    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
         </welcome-file-list>
```

程序代码：

School.java

```
package com.wr.po;

/**
 * School entity. @author MyEclipse Persistence Tools
 */

public class School implements java.io.Serializable {

    // Fields

    private String schno;
    private String schname;
    private String schcity;
    private Integer schage;

    // Constructors

    /** default constructor */
    public School() {
    }

    /** minimal constructor */
    public School(String schno, String schname) {
        this.schno = schno;
        this.schname = schname;
    }

    /** full constructor */
    public School(String schno, String schname, String schcity, Integer schage) {
        this.schno = schno;
        this.schname = schname;
        this.schcity = schcity;
```

```
        this.schage = schage;
    }

    // Property accessors

    public String getSchno() {
        return this.schno;
    }

    public void setSchno(String schno) {
        this.schno = schno;
    }

    public String getSchname() {
        return this.schname;
    }

    public void setSchname(String schname) {
        this.schname = schname;
    }

    public String getSchcity() {
        return this.schcity;
    }

    public void setSchcity(String schcity) {
        this.schcity = schcity;
    }

    public Integer getSchage() {
        return this.schage;
    }

    public void setSchage(Integer schage) {
        this.schage = schage;
    }
}
```

HibernateSessionFactory.java

略

SchoolDao.java:

```
package com.wr.dao;

import org.hibernate.Session;

import com.wr.po.School;
import com.wr.util.HibernateSessionFactory;

public class SchoolDao {

        public School findByNo(String schoolno){
            Session session = HibernateSessionFactory.getSession();
            return (School)session.get(School.class, schoolno);
        }
    }
```

SchoolService.java:

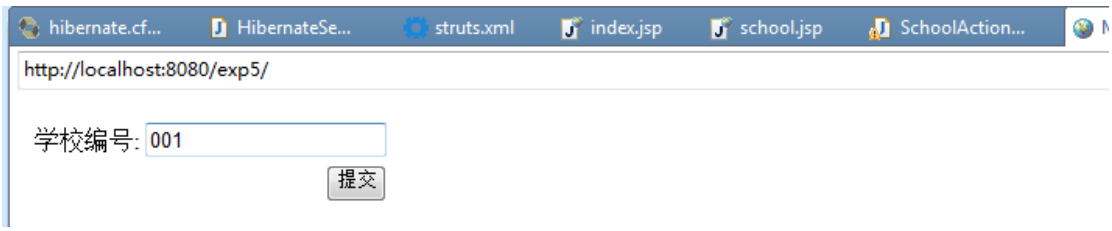
```
package com.wr.service;

import com.wr.dao.SchoolDao;
import com.wr.po.School;

public class SchoolService {

        private SchoolDao schoolDao;

        public SchoolService(){
            this.schoolDao=new SchoolDao();
        }

        public School find(String schoolno){
            return schoolDao.findByNo(schoolno);
        }
}
```

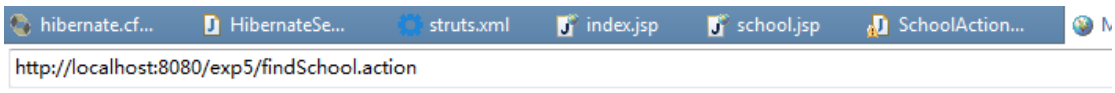
SchoolAction.java

```
package com.wr.action;

import com.opensymphony.xwork2.ActionSupport;
import com.wr.po.School;
import com.wr.service.SchoolService;
```

```
public class SchoolAction extends ActionSupport{

    private String schoolno;
    private School school;

    public School getSchool() {
        return school;
    }

    public void setSchool(School school) {
        this.school = school;
    }

    public String getSchoolno(){
        return schoolno;
    }

    public void setSchoolno(String schoolno) {
        this.schoolno = schoolno;
    }

    public String findSchool(){
        SchoolService schoolService=new SchoolService();

        school=schoolService.find(schoolno);

        if (school!=null) {
            return SUCCESS;
        }else{
            return ERROR;
        }
    }

}
```

(2) 页面效果:





学校编号001 学校名字mmc 学校城市maoming 学校年龄60

实验五 Hibernate 中高级检索

实验案例 1：使用 setResultTransformer 实现 QBC 检索

实验目的

使用 setResultTransformer 实现 QBC 检索。

实验要点

setResultTransformer 的应用

实现效果

1. 使用 setResultTransformer 实现 QBC 检索（age>=25 的 员工 name）。

```
Hibernate: select this_.dept_id as d
Hibernate: select empllist0_.deptid
Tom
Mike
Bob
```

实现步骤

1 搭建环境：

w) 新建相关数据表：

（Dept 对于 Empl 是一对多关系。一个部门可以有多个员工。）

i. 在 oracle 数据库中新建 Empl 表,（相关 sql 语句在实验指导手册环境工程包目录下）

1. 相关表结构如下：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	28	1
4	Marry	26	3
5	Bob	25	1

ii. 在 oracle 数据库中新建 Dept 表，（相关 sql 语句在实验指导手册环境工程包目录下）

1. 相关表结构如下：

	DEPT_ID	DEPT_NAME
1	1	SAL
2	2	IT
3	3	HR

x) 导入工程，将“第五章 hibernate 高级检索\第一节 连接查询\实验指导手册\环境工程包”目录下的 HibernateDemo38 文件导入到 ecplise 中。

2 进行实验：

(Dept 和 Empl 的实体类以及配置文件已帮学员配置完成，并且在 Dept 实体类中已添加 Empl 的 Set 集合对象：emplList，

i. 找到并打开 DeptImpl，在 queryDeptByQBCCollection(int age)这个方法内，找到 criteria.add(Restrictions.ge("E.age",age));这一行代码。

ii. 在 criteria.add(Restrictions.ge("E.age",age))下一行，调用 criteria 的 setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP)方法 。

iii. 调用 criteria 的 list()方法，返回 List 的集合对象，命名为 list。

iv. 调用 list 的 iterator（ ）方法，返回一个 Iterator 的对象，命名为 it。

v. 使用 while 语句 ,使用 hasNext()方法判断 it 对象是否有下一条数据。

vi. 如果有下一条记录。使用 Map 的 map 对象接收这条记录。即：Map map=(Map) it.next();

vii. 调用 map 的 get(Criteria.ROOT_ALIAS)方法，获得一个 Dept 类型的对象，命名为 dept。

viii. 调用 map 的 get（“E”）方法，得到一个 Empl 对象，命名为 empl。

ix. 调用 empl 的 getName（ ）方法，并将其打印输出。

x. 找到 test 包下的 Test1 类，相关测试代码已为学员完成，请学员直接运行测试类。观察结果。

实验案例 2：迫切内连接、隐式内连接应用

实验目的

HQL 迫切内连接的应用

HQL 隐式内连接的应用。

实验要点

HQL 迫切内连接

HQL 隐式内连接

实现效果

1HQL 迫切内连接：

```
Hibernate: select dept0_.dept_id as dept_i
Dept [deptId=1, deptName=SAL ]
Tom
Mike
john
Bob
Dept [deptId=2, deptName=IT ]
BOB
```

HQL 隐式内连接（查询部门 id>1 的员工信息）：

```
Hibernate: select empl0_.id as id1_1_, empl0_.name as name2_
Hibernate: select dept0_.dept_id as dept_id1_0_0_, dept0_.de
Hibernate: select empllist0_.deptid as deptid3_0_0_, empllis
BOB
```

实现步骤

1 搭建环境：

y) 新建相关数据表：

（Dept 对于 Empl 是一对多关系。一个部门可以有多个员工。员工对于部门是多对一关系）

i. 在 oracle 数据库中新建 Empl 表,（相关 sql 语句在实验指导手册环境工程包目录下）

1. 相关表结构如下：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	28	1
4	Marry	26	3
5	Bob	25	1

ii. 在 oracle 数据库中新建 Dept 表，（相关 sql 语句在实验指导手册环境工程包目录下）

1. 相关表结构如下：

	DEPT_ID	DEPT_NAME
1	1	SAL
2	2	IT
3	3	HR

z) 导入工程，将“第五章 hibernate 高级检索\第一节 连接查询\实验指导手册\环境工程包”目录下的 HibernateDemo39 文件导入到 ecplise 中。

2 进行实验：

(Dept 和 Empl 的实体类以及配置文件已帮学员配置完成，并且已完成 empl 和 dept 的多对一以及一对多关联设置。)

aa) 迫切内连接：

i. 找到并打开 test 包下的 Test1 测试类。在 IEmplDao deptdao = new EmplImpl();下一行，编写一条 sql 语句：from Dept d inner join fetch d.emplList。（inner join fetch 表示迫切内连接）

ii. 运行测试类。观察控制台。

bb) 隐式内连接：

i. 找到并打开 test 包下的 Test2，找到 IEmplDao empldao = new EmplImpl();这一行代码。

ii. 在 IEmplDao empldao = new EmplImpl();下一行编写一条 sql 语句：from Empl e where e.dept.deptId>1。（通过隐式内连接查询部门 id>1 的员工信息。）。

iii. 运行测试类，观察控制台。

实验案例 3：动态查询

实验目的

使用 Restrictions 接口动态查询检索数据。

使用 QBE 模式动态检索数据

实验要点

Restrictions 动态查询。

QBE 动态查询。

实现效果

1. 动态查询：

a) Restrictions 动态查询：

```
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select this_.id as id1_0_0_, this_.name as
Empl [id=2, name=Tom, age=25, dId=1]
Empl [id=4, name=john, age=23, dId=1]
Empl [id=5, name=Bob, age=25, dId=1]
```

b) QBE 动态查询

```
Hibernate: select this_.id as id1_0_0_, this_.name a
Empl [id=1, name=BOB, age=38, dId=3]
Empl [id=2, name=Tom, age=25, dId=1]
Empl [id=4, name=john, age=23, dId=1]
Empl [id=5, name=Bob, age=25, dId=1]
Empl [id=11, name=BOB, age=29, dId=2]
```

实现步骤

搭建环境：

cc) 新建相关数据表：

i. 在 oracle 数据库中新建 Empl 表, (相关 sql 语句在实验指导手册环境工程包目录下)

1. 相关表结构如下：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	28	1
4	Marry	26	3
5	Bob	25	1

dd) 导入工程，将“第五章 hibernate 高级检索\第三节 高级查询\实验指导手册\环境工程包”目录下的 HibernateDemo42 文件导入到 ecplise 中。

进行实验：

ee) 动态查询：

i.Restrictions 动态查询：

1. 找到并打开 dao 包下的 emplImpl 类，在 queryEmplQBC()方法体内找到 Criteria criteria=session.createCriteria(Empl.class);。
2. 在该代码下一行调用 Restrictions 的 like(String propertyName，String value,MatchMode matchmode)。参数说明：
   a) propertyName：name
   b) value：“o”。
   c) matchmode：MatchMode.ANYWHERE。
3. 调用 criteria 的 list( )方法。返回 List 的集合对象，命名为 list。
4. 将 list 作为方法的返回值。
5. 找到并打开 test 包下的 Test1，相关测试代码已提供，直接运行 Test1.查看结果。

ii. QBE 动态查询

1. 找到并打开 dao 包下的 emplImpl 类，在 queryEmplQBE(Empl empl) 方法体内找到 Criteria criteria=session.createCriteria(Empl.class); 。
2. 调用 Example 的 create（Object arg）方法，将 empl 对象作为参数传入，返回 Example 类型的对象命名为 example。
3. 调用 example 的 enableLike（MatchMode.ANYWHERE）的方法。（表示允许进行任意位置模糊查询）
4. 调用 example 的 ignoreCase( )。（表示忽略大小写）
5. 调用 criteria 的 add（example）方法。
6. 调用 criteria 的 list( )方法。返回 List 的集合对象，命名为 list。
7. 将 list 作为方法的返回值。
8. 找到并打开 test 包下的 Test2 测试类。在 IEmplDao empldao = new EmplImpl();下一行创建一个 empl 对象，如下所示：

```
Empl empl=new Empl();
```

9. 调用 empl 的 setName(String arg)方法。参数 arg 为“O”。（表示查询 name 中有“O”的员工信息，忽略大小写）。
10. 运行 Test2.观察结果。此时发现控制台只有 sql 语句。没有输出结果。接下来我们需要对实体类进行修改。
11. 找到并打开 entity 包下的 Empl 实体类。
12. 将属性 id、age、dId 的类型改为分装类。即 Integer。同时修改 set、get 方法以及构造器。
13. 再次运行 Test2 测试类。发现控制台有相关数据。

实验案例 4：子查询

实验目的

子查询的应用。

实验要点

嵌套子查询。

关键字应用

a) exists

b) elements

操纵函数应用

实现效果

1. 嵌套子查询。

```
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select dept0_.dept_id as dept_id1_0_, dept0_.dept_name as
Hibernate: select empllist0_.deptid as deptid3_0_0_, empllist0_.id as
Dept [deptId=1, deptName=SAL ]
```

2. 关键字查询：

a) exists

```
Hibernate: select dept0_.dept_id as dept_id
Hibernate: select empllist0_.deptid as dept
Hibernate: select empllist0_.deptid as dept
Dept [deptId=1, deptName=SAL ]
Dept [deptId=2, deptName=IT ]
```

b) elements

3. 操纵函数应用

```
Hibernate: select dept0_.dept_id as dept_id1_0_, dept0_.dept_name as
Hibernate: select empllist0_.deptid as deptid3_0_0_, empllist0_.id a
Dept [deptId=2, deptName=IT ]
```

实现步骤

```
Hibernate: select dept0_.dept_id as dept_id1_0_
Hibernate: select empllist0_.deptid as deptid3_
Dept [deptId=1, deptName=SAL ]
```

搭建环境：

ff) 新建相关数据表：

（Dept 对于 Empl 是一对多关系。一个部门可以有多个员工。）

i. 在 oracle 数据库中新建 Empl 表,（相关 sql 语句在实验指导手册环境工程包目录下）

1. 相关表结构如下：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	28	1
4	Marry	26	3
5	Bob	25	1

ii. 在 oracle 数据库中新建 Dept 表，（相关 sql 语句在实验指导手册环境工程包目录下）

1. 相关表结构如下：

	DEPT_ID	DEPT_NAME
1	1	SAL
2	2	IT
3	3	HR

gg) 导入工程，将“第五章 hibernate 高级检索\第三节 高级查询\实验指导手册\环境工程包”目录下的 HibernateDemo44 文件导入到 ecplise 中。

进行实验：

hh) 嵌套子查询（查询员工数量大于 1 的部门信息）：

i. 找到并打开 test 包下的 Test1 测试类。在 String sql=“”内填入相关 sql 语句。,此处 sql 语句为： from Dept d where (select count(e) from d.emplList e)>1。

ii. 对于结果的输出处理已为学员完成，直接运行程序观察结果。

ii) 关键字查询：

i. 使用 exists 关键字，查询有员工的部门信息：

1. 找到并打开 test 包下的 Test1 测试类。将 sql 字符串改为：from Dept d where exists(from d.emplList e)。
2. 对于结果的输出处理已为学员完成，直接运行程序观察结果。

ii. 使用 elements 关键字，查询给定的对象是否在部门的员工集合里：

1. 找到并打开 test 包下的 Test1 测试类。将 sql 语句改为 :from Dept d where:empl in elements(d.emplList)。(empl 为自定义字符串)
2. 找到并打开到包下的 EmplImpl 类，在 Query query=session.createQuery(sql); 下一行调用 query 的 setEntity(String arg0,Object arg1);参数说明：

a) arg0：自定义字符串。即上一步在 sql 语句中自定义的字符串：empl。
b) arg1：创建一个临时态的对象：new Empl(1,"BOB",18,2)

3. 运行 test 包下的 Test1 测试类。

jj) 操纵函数应用（使用 size 函数查询员工数量大于 1 的部门信息）：

i. 打开并找到 dao 包下的 EmplImpl 类。将 query.setEntity("empl", new Empl(1,"BOB",18,2));这一行代码注释掉或者删除。

ii. 找到并打开 test 包下的 Test1 测试类。。,将 sql 语句改为 :from Dept d where d.emplList.size>1。

iii. 运行 Test1 测试类。观察结果。

实验六 事务并发与缓存管理

实验案例 1：事务

实验目的

使用 Hibernate API 申明 JDBC 事务。

实验要点

使用 Hibernate API 申明 JDBC 事务。

实现效果

使用 Hibernate API 申明 JDBC 事务：

	ID	NAME	AGE	DEPTID
1	1	BOB	18	3
2	2	Tom	25	1
3	3	Mike	28	1
4	4	john	23	1
5	5	Bob	25	1

实现步骤

搭建环境：

kk) 新建相关数据表：

i. 在 oracle 数据库中新建 Empl 表, (相关 sql 语句在实验指导手册环境工程包目录下)

1. 相关表结构如下：

ID	NAME	AGE	DEPTID
1	Jack	23	2
2	Tom	25	1
3	Mike	28	1
4	Marry	26	3
5	Bob	25	1

ll) 导入工程，将“第六章 事务并发与缓存管理\第三节 事务\实验指导手册\环境工程包”目录下的 HibernateDemo51 文件导入到 ecplise 中。

进行实验：

mm) 找到并打开 hibernate.cfg.xml 配置文件。在 session-factory 节点内添加 property 元素。语法结构如下：（用于确认实现 JDBC 的事务）

i. <property name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>

nn) 找到并打开 dao 包下的 emplImpl 类，在 Session session = sf.openSession(); 下一行调用 session 的 beginTransaction()方法，开启事务。返回一个 Transaction 对象，命名为 tr。

oo) 使用 session 的 get（ ）方法，获得 id 为 1 的持久化 Empl 对象 empl。

pp) 调用 empl 的 setdId（int arg）方法，将 empl 的 dId 改为 3。

qq) 调用 tr 的 commit（ ）方法。

rr) 将 empl 对象作为方法的返回值。

ss) 找到并打开 test 包下的 Test1 测试类。相关测试代码已提供，学员直接运行程序，然后从 sqldeveloper 中查看 Empl 表。观察 id 为 1 的员工的 dId 的值。

实验案例 2：并发处理

实验目的

对并发事件进行处理。

实验要点

处理并发事件。

实现效果

处理并发事件：

```
开始事务1：
开始事务2：
Hibernate: select account0_.id as id1_0_0_, account0_.name as name2_0_0_, account0_.
Hibernate: select account0_.id as id1_0_0_, account0_.name as name2_0_0_, account0_.
查询账户余额为：1200.0
取出100元，账户剩余：1100.0
提交事务
Hibernate: update account set name=?, balance=? where id=?
查询账户余额为：1100.0
存入100元，账户剩余：1200.0
提交事务
Hibernate: update account set name=?, balance=? where id=?
```

实现步骤

搭建环境：

tt) 新建相关数据表：

i. 在 oracle 数据库中新建 Account 表,（相关 sql 语句在实验指导手册环境工程包目录下）

1. 相关表结构如下：

ID	NAME	BALANCE
1	Tom	1000

（本次实验只插入一条数据，仅做实验使用。）

uu) 导入工程 ,将“第六章 事务并发与缓存管理\第四节 并发处理\实验指导手册\环境工程包”目录下的 HibernateDemo53 文件导入到 ecplise 中。

进行实验：

项目说明：

在 HibernateDemo53 中的 dao 包下有两个线程，分别是 BussinessCheckService 和 BussinessService，BussinessCheckService 对应的是事务一，主要业务为取出 100 元。BussinessService 对应的事事务二 ,主要业务为存入 100 元。在 test 包下的 Test1 测试类中，通过调用其 start（）方法使这两个线程并发运行。请学员观察运行结果，并对出现的并发情况进行改进。

vv) 打开并找到 test 包下的 Test1，运行该测试类。运行结果如下：

```
开始事务1：
开始事务2：
Hibernate: select account0_.id as id1_0_0_, account0_.name as name2_0_0_, a
Hibernate: select account0_.id as id1_0_0_, account0_.name as name2_0_0_, a
查询账户余额为：1000.0
查询账户余额为：1000.0
存入100元，账户剩余：1100.0
提交事务
取出100元，账户剩余：900.0
提交事务
Hibernate: update account set name=?, balance=? where id=?
Hibernate: update account set name=?, balance=? where id=?
```

结果分析：

当事务一存入 100 元后，账户剩余 1100，正确，随后当事务二取出 100 元后，账户剩余为 900，出现问题（应该此时为 1000）。这种情况即为并发。

ww) 找到并打开 dao 包下的 BussinessCheckService 类，将 Account account=(Account) session.get(Account.class, 1); 这一行的 get(Account.class,1) 改为：session.get(Account.class, 1,LockMode.UPGRADE);“ LockMode.UPGRADE”表示采用悲观锁。

xx) 类似的，找到 BussinessService 类，将 Account account=(Account) session.get(Account.class, 1); 这一行的 get(Account.class,1) 改为：session.get(Account.class, 1,LockMode.UPGRADE);

yy) 在运行 Test1 测试类，观察结果，发现此时的结果为正确结果。已经解决了并发的问题。

实验案例 3：hibernate 缓存

实验目的

Hibernate 的一级缓存的应用。

Hibernate 的二级缓存的应用

实验要点

Hibernate 的一级缓存：

Hibernate 的二级缓存

实现效果

Hibernate 的一级缓存：

c) ecplise 控制台输出结果：

```
false
false
true
Hibernate: select account0_.id as id1_0_0_, account0_.name as name2_0_0_, acc
```

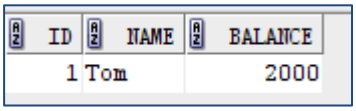
Hibernate 二级缓存：

ID	NAME	BALANCE
1	Tom	1300

e) Ecplise 控制台输出信息：

```
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Hibernate: select account0_.id as id0_0_, account0_.name as name0_0_, account0_.balance as balance0_0_ from account account0_
Hibernate: update account set name=?, balance=? where id=?
2014-8-12 13:46:00 org.hibernate.service.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
```

f) 数据库中查询结果：



ID	NAME	BALANCE
1	Tom	2000

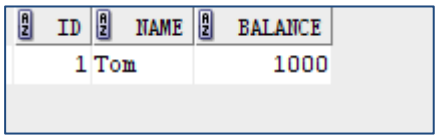
实现步骤

搭建环境：

zz) 新建相关数据表：

i. 在 oracle 数据库中新建 Account 表,（相关 sql 语句在实验指导手册环境工程包目录下）

1. 相关表结构如下：



ID	NAME	BALANCE
1	Tom	1000

（本次实验只插入一条数据，仅做实验使用。）

aaa) 导入工程 ,将“第六章 事务并发与缓存管理\第五节 Hibernate 缓存\实验指导手册\环境工程包”目录下的 HibernateDemo54 文件导入到 ecplise 中。

进行实验：

bbb) Hibernate 的一级缓存：

i. 找到并打开 test 包下的 firstCache 测试类。在 Account account1=(Account) session.load(Account.class, 1);下一行调用 session 的 evict(Object arg);方法 ,将 account1 作为参数传入。,表示把 account1 对象从缓存中清除。

ii. 在调用 session 的 load 方法获得一个 id 为 1 的 Account 对象 , 命名为 account2。

iii. 打印输出 account1==account2 的结果。（此时结果为 false）。

iv. 调用 session 的 contains ( ) 方法，分别将 account1 和 account2 两个对象作为参数传入。分别打印输出结果。（contains 方法是用来判断对象是否在缓存中）。

v. 调用 account1 的 setBalance(int arg);方法，将 account1 对象的 balance 属性改为 1200.

vi. 运行程序。发现控制台提示 no session 错误。因为此时的 account1 对象为临时态对象，并且已被清理出 session 缓存。所以会报 no session 异常。

vii. 将 account1 的 setBalance 方法注释掉，调用 account2 的 setBalance(int arg);方法，将 account1 对象的 balance 属性改为 1300.

viii. 运行程序。并从数据库中查询数据。此时 id 为 1 的对象 balance 属性已改为 1300.

ccc) Hibernate 二级缓存：

i. 找到资源文件夹 ehcache jar 包文件夹，将里面的 ehcache-core-2.4.3.jar 、 hibernate-ehcache-4.1.7.Final.jar 、 slf4j-api-1.6.6.jar 加入到当前应用的 classpath 中。

ii. 将资源文件夹内的 ehcache.xml 文件复制黏贴到 src 目录下（ehcache，xml 我们使用默认配置即可，故本次实验无需修改）。

iii. 找到 hibernate.cfg.xml 文件，在 session-factory 节点内添加如下元素：

```
<property name="hibernate.cache.use_second_level_cache">true</property>
<property name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
<property name="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
```

iv. 找到 entity 包下的 Account.hbm.xml 文件，在 Account.hbm.xml 内 class 节点内的 id 属性上方添加如下属性：

```
<cache usage="read-write"/>
```

v. 打开并找到 test 包下的 secondCache，在 session.close()的下一行再次使用 session 对象调用 sf 的 openSession，再次开启 session。

vi. 调用 session 的 beginTransaction()方法开启事务 ,返回 Transaction 类型的对象，命名为 tr。

vii. 再一次调用 session 的 get(Account.class, 1);方法，返回 id 为 1 的 Account 对象，命名为 account2。

viii. 调用 account2 的 setBalance（int arg）方法将该对象的 balance 改为 2000。

ix. 调用 tr 的 commit（ ）方法。提交事务

x. 关闭 session。

xi. 运行 secondCache，发现控制台只有一条查询的 sql 语句，说明使用了二级缓存。

xii. 查看数据库中该对象的 balnance 属性的值。