JPA笔记（2014）

————李耀

【JPA规范】

JPA(Java Persistence API)是SUN官方提出的java持久化规范，它为java开发人员提供了一种对象/关系映射工具来管理java应用中的关系数据。

它的出现主要是为了简化现有的持久化开发工作和整合ORM技术。结束现在Hibernate，Toplink，JDO等ORM框架各自为营的局面。

值得注意的是，JPA是在充分吸收了Hibernate，Toplink，JDO等ORM框架的基础上发展而来的，具有易于使用、伸缩性强等优点。从目前开发社区的响应上看，JPA受到l极大的支持和赞赏，其中就包括Spring和EJB3.0的开发团队，着眼未来几年的走向，JPA作为ORM领域标准化整合者的目标应该不难实现。

JPA的总体思想和现有的Hibernate，Toplink，JDO等ORM框架大体一致，总的来说，

JPA包括以下3方面的技术：

————>1.ORM映射元数据

JPA支持XML和JDK5.0的注释(注解)两种元数据形式，元数据描述对象和表之间的映射关系，框架据此将实体对象持久化到数据库表中。

————>2.java持久化API

用来操作实体对象，执行CRUD操作，框架在后台替我们完成所有的事情，开发者可以从繁琐的JDBC和SQL代码中解脱出来。

————>3.查询语言

这是持久化操作中很重要的一个方面，通过面向对象而非面向数据库的查询语言查询数据，避免程序的SQL语句紧密耦合。

【注意】JPA不是一种新的ORM框架，它只是用于规范现有的ORM技术，它不能取代现有的ORM框架，相反，在采用JPA开发的时候，我们仍将使用到这些ORM框架，只是此时开发出来的应用不再依赖于某个持久化提供商，应用可以在不修改代码的情况下可以在任何JPA环境中运行，真正做低耦合，可拓展的程序设计。

------------------------------------------------------------------------------

【使用Hibernate开发JPA的应用】

第一步. 导入开发JPA所依赖的jar文件：在D:\IdeProjectManager\Hibernate-JPA-jar\lib中。

第二步. 在src根目录下建立META-INF文件夹，并在其中写JPA的配置文件persistence.xml，

JPA规范要求在类路径的META-INF目录下放置配置文件persistence.xml。

而且文件的名称是固定的。

——————>建模思想

方式一，先建表，然后再根据表来配置文件和实体bean（受到了传统数据库建模的影响，用途比较多）

方式二，先编写配置文件和实体bean，然后再生成表（采用的是领域建模的思想，这种思想相对第一种来说更加的OOP面向对象）

第三步.编写实体和它的映射元数据等等。

------------------------------------------------------------------------------

**persistence.xml文件（通用的）**

<persistence xmlns="http://java.sun.com/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd"

version="1.0">

<!--persistence-unit称为持久化单元，即就是代表一堆实体bean的集合，并为其取一个名称-->

<!-- 事物的类型transaction-type，包括两种：本地事务和全局事务

【本地事务】只是针对同一个数据库操作而言的。

【全局事务】

而对于特殊情况，比如说转账业务，扣钱的动作实在mysql数据库进行的，加钱的动作是在oracal数据库中进行的，那么我们要保证这两个动作必须在同一个事务中完成，普通的JDBC是不可能完成的，这时我么就是用到了全局事务，这儿存在一个二次提交的过程，当扣钱语句和加钱语句分别完成之后，首先预提交到各自对应的数据库中，然后数据库也是预处理提交，并返回结果给两个connection连接，此时进行判断，如果两个预处理的结果都正确无误的话，才会真正进行二次提交事务，数据库中的数据才能够被真正的改变。这样就必须使用全局事务了。

注意：基本上的应用都是采用本地事务来完成的。

-->

<persistence-unit name="itcast" transaction-type="RESOURCE\_LOCAL">

<properties>

<!-- 指定方言 -->

<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>

<!-- create-drop代表在应用在启动时建表，在应用关闭时是删表，update代表按需更新 -->

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

<!-- 数据库配置 -->

<property name="hibernate.connection.driver\_class" value="org.gjt.mm.mysql.Driver"/>

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

<property name="hibernate.connection.password" value="624963768"/>

<property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/itcast?useUnicode=true&amp;characterEncoding=UTF-8"/>

</properties>

</persistence-unit>

</persistence>

**一．【JPA用法概念介绍】**

**1. Person.java(定义实体类的介绍)**

package cn.itcast.bean;

import java.util.Date;

import javax.persistence.Basic;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.EnumType;

import javax.persistence.Enumerated;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Lob;

import javax.persistence.Table;

import javax.persistence.Temporal;

import javax.persistence.TemporalType;

import javax.persistence.Transient;

//实体bean

@Entity

@Table(name="Person")

public class Person {

private Integer id;

private String name;

private Date birthday= new Date();//1999-08-13

private Gender gender = Gender.MAN;//枚举类型的（设置默认值）

private String info;//个人说明（说明文字比较多，255的长度存不下）

private byte[] file;//字节数组

private String imagePath;//不成为持久化字段==该属性不和数据库某个表中的属性映射

public Person() {}

public Person(String name) {

this.name = name;

}

/\*注解，@Id是主键标识 @GeneratedValue是主键生成策略(默认是AUTO，根据所选数据库方言来选择主键生成策略) \*/

@Id @GeneratedValue

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

@Column(length=10,nullable=false,name="personName")

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Temporal(TemporalType.DATE) //1999-08-13格式，要想加上时间，就要用到时间戳

@Column(nullable=false)

public Date getBirthday() {

return birthday;

}

public void setBirthday(Date birthday) {

this.birthday = birthday;

}

@Enumerated(EnumType.STRING) //【以字符串保存进数据库中】

@Column(length=5,nullable=false)

public Gender getGender() {

return gender;

}

public void setGender(Gender gender) {

this.gender = gender;

}

//这个注解声明了该属性对应在数据库中的字段是一个【大文本类型】

@Lob

public String getInfo() {

return info;

}

public void setInfo(String info) {

this.info = info;

}

//该注解是【字段延迟加载】的，当字段大小过大的时候，如果没有访问到该字段，该字段内容就不会被加载。节省了系统内存。

@Lob

@Basic(fetch=FetchType.LAZY)

public byte[] getFile() {

return file;

}

public void setFile(byte[] file) {

this.file = file;

}

//用来标注该【属性不作为持久化字段】，即就是不和数据库中某属性映射。

@Transient

public String getImagePath() {

return imagePath;

}

public void setImagePath(String imagePath) {

this.imagePath = imagePath;

}

}

**3. Gender.java(枚举性别)**

package cn.itcast.bean;

//性别(类型是枚举出来的)

public enum Gender {

MAN,WOMEN

}

**4. PersonTest.java(JUnit测试方法，主要介绍使用持久化API对实体bean实行CRUD的操作)**

package cn.JUnit.test;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import javax.persistence.Query;

import org.junit.BeforeClass;

import org.junit.Test;

import cn.itcast.bean.Person;

//使用持久化API对实体bean实行CRUD的操作

public class PersonTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {}

@Test //【保存实体】

public void save(){

//1.首先获取EntityManagerFactory对象（和hibernate的SessionFactory对象类似）

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开始事务

em.persist(new Person("lisi"));//————————>新建状态

em.getTransaction().commit();

em.close();

factory.close();

}

@Test //【获取实体Bean对象】——方式1

public void getPerson1(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开始事务

//参数1代表实体类，参数2代表实体标识符属性的值(Id的值，即就是第几条记)

Person person = em.find(Person.class, 1);//相当于hibernate中的get方法

/\*【refresh】方法的作用：当你最先从数据库中获取到一条记录的时候,之后由于又有一些业务处理,用了大约1分钟，在此期间该记录的某个属性值被修改了,此时如果再使用find方法来获取数据的话,获取到的数据并不是该记录的最新数据,因为它会从EntityManager的一级缓存中去获取刚刚(没有进行业务处理之前)获取到的该记录的对象,所以要想获取到该记录最新的数据,就必须使用refresh方法对该实体进行刷新操作！方才可以获得最新数据\*/

em.refresh(person);

System.out.println(person.getName());

em.getTransaction().commit();

em.close();

factory.close();

}

@Test //【获取实体Bean对象】——方式2

public void getPerson2(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开始事务

/\*getReference方法相当于hibernate中的load方法，起到【延迟初始化】的作用。

当没有访问到实体Bean的属性内容的时候，数据库就不会真正的加载数据进来，返回的只是代理对象。只有当访问到了person.getName()的时候，数据才会被真的加载进来。

注意，必须在同一个事务中。参数1代表实体类，参数2代表实体标识符属性的值(Id的值，即就是第几条记录)\*/

Person person = em.getReference(Person.class, 1);

System.out.println(person.getName());

em.getTransaction().commit();

em.close();

factory.close();

}

@Test //【更新实体Bean对象】——方式1

public void updatePerson1(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开始事务

Person person = em.find(Person.class, 1);//————————>托管状态

person.setName("wangwu");//—————>托管状态，更新成功且会同步到数据库

em.getTransaction().commit();

em.close();

factory.close();

}

@Test //【更新实体Bean对象】——方式2

public void updatePerson2(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开始事务

Person person = em.find(Person.class, 1);//————————>托管状态

em.clear();//把目前处于实体管理器里所有的实体Bean变成游离状态(脱管态)

person.setName("liuliu");//—>游离状态，更新成功但不会同步到数据库，需要merge

em.merge(person);//该方法就是把游离状态执行的更新操作同步回数据库！！！

em.getTransaction().commit();

em.close();

factory.close();

}

@Test //【删除实体Bean对象】

public void deletePerson(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开始事务

Person person = em.find(Person.class, 1);//————————>托管状态

em.remove(person);//删除实体，实体必须是托管状态

em.getTransaction().commit();

em.close();

factory.close();

}

/\* JPA里面实体有4种状态：

1.新建状态

2.托管状态

3.游离状态（脱管）

4.删除状态\*/

@Test //【查询】

public void query(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

/\*在JPA规范里面，select o是必须要的，o代表对象，Person代表实体的名称，这个o是别名。o.id后面不能直接将参数赋上去，要防止SQL侵入攻击！ (所以要采用：1.命名参数查询——>:id即可/2.位置参数查询——>？/？1(告诉？从1开始)即可\*/

Query query = em.createQuery("select o from Person o where o.id=?1");//位置参数查询

query.setParameter(1,2);//第一个参数：从哪个开始，第二个参数：要查询第几条记录

//query.getResultList();类似于Hibernate中的query.list()得到结果列表一样的含义

List<Person> persons = query.getResultList();

for(Person person : persons){ //循环查询

System.out.println(person.getName());

}

/\*【注意】对于统计查询：select count(\*) from person而言：Hibernate采用的是query.unique();就可以得到单结果。JPA里面会采用query.getSingleResult();获得单结果。\*/

em.close();

factory.close();

}

@Test //【删除更新】（要开启事务）

public void deleteQuery(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开始事务

Query query = em.createQuery("delete from Person o where o.id=?1");//位置参数查询

query.setParameter(1,2);

query.executeUpdate();//执行更新

em.getTransaction().commit();

em.close();

factory.close();

}

@Test //【更新】（要开启事务）

public void updateQuery(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开始事务

//命名参数查询

Query query = em.createQuery("update Person o set o.name=:name where o.id=:id"); query.setParameter("name","wangmazi");

query.setParameter("id",3);

query.executeUpdate();//执行更新

em.getTransaction().commit();

em.close();

factory.close();

}

}

**二．【JPA中一对多双向关联与级联操作】**

**1. Order.Java(订单)**

package cn.itcast.bean;

import java.util.HashSet;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

//订单（强调一对多，多对一的关系）——————>双向关联

/\*在JPA当中，当出现一对多或是多对一的关联关系的时候，多的一方为关联关系的维护方。

关系的维护方即就是要负责【外键记录/字段的更新】，关系被维护方是没有权利更新外键记录/字段的。————>也就是说，订单项orderItem为关联关系的维护\*/

@Entity

@Table(name="orders") //默认表名和关键字order by冲突，所以重起表名

public class Order {

private String orderId;//订单号

private Float allPrice = 0f;//订单总价

//在订单Order中要关联各个订单项，我们采用集合的方式来存储—————【一对多】

private Set<OrderItem> items = new HashSet<OrderItem>();//泛型指明存储类型

//字符串类型要使用UUID增长,JPA暂时不支持

@Id @Column(length=12)

public String getOrderId() {

return orderId;

}

public void setOrderId(String orderId) {

this.orderId = orderId;

}

@Column(nullable=false)

public Float getAllPrice() {

return allPrice;

}

public void setAllPrice(Float allPrice) {

this.allPrice = allPrice;

}

/\*【一对多的注解】————该注解是为了告诉JPA:订单order和订单项orderItem是一对多的关系

1.cascade是【级联】操作：

值PERSIST是【级联持久化/级联保存】(作用：当我们执行了级联保存操作的时候，Hibernate内部实际上执行了insert语句，将要保存的数据插入到数据库中，如果数据库中已经有该id的记录，那么就会插入失败，所以还需要绑定级联更新MERGE)

——————>一般只会在订单中执行该操作，因为保存订单顺便级联保存其订单项。

值MERGE是【级联更新/级联合并】(作用： 该操作控制着对父对象更新是否会波及其子对象的更新，当order父对象处于游离状态的时候，我们对其进行更新，其子对象orderItem中内容是否也会同时更新并同步回数据库中)

——————>只有在调用merge()方法的时候，该级联操作才会起到作用。

值REFRESH是【级联刷新】(作用：当获取到一条数据的时候，该数据之后又被其他操作修改了，所以或得数据有可能不是最新的，这时候就要调用实体管理器中的REFRESH方法来刷新该数据，REFRESH也就相当于重新获取数据，内部执行的是select语句。)

——————>只有在调用persist()方法的时候，该级联操作才会起到作用。

值REMOVE是【级联删除】(作用：如果要想删除父对象的同时，将其所有的子对象也一并删除了，就需要用到该级联操作)

——————>只有在调用remove()方法的时候，该级联操作才会起到作用。

注意：如果应用中该四项级联动作都用到了，可以直接使用CascadeType.ALL即可。

2.fetch是设置【是否延迟加载】操作：

——————>【只要是xxxToMany的关系映射，延迟属性默认都是延迟加载(高效)】

——————>【只要是xxxToOne的关系映射，延迟属性默认都是立即加载(高效快速)】

值EAGER是立即加载

应用场景：当得到父对象order的时候，想要立即得到其子对象orderItem里面的数据的话，就可以使用立即加载操作来完成。

值LAZY是延迟加载

应用场景：当得到父对象order的时候，并不希望立即得到其子对象orderItem里面的数据，就可以使用延迟加载操作来完成。

【注意】什么时候才会初始化该集合里面的数据呢？当我们第一次访问该属性并对该属性进行操作的时候，该属性的集合数据才会从数据库中被拿出来。一般对某属性使用了延迟加载操作，则称该属性为延迟属性，当我们要访问该延迟属性的时候，前提必须保证EntityManager对象处于打开状态。如果我们目前还没有获取到该延迟属性，而且同时EntityManager对象也已经关闭，那么必然就会出现延迟加载意外的问题！！！

3.mappedBy声明当前类为关系的被维护端，反过来，另一个类就是关系的维护方。它的值是关系维护方OrderItem类中维护关联关系的属性order属性(该属性并不是成员属性，而是其get方法的后缀)

mappedBy类似于Hibernate中的set集合中的inverse属性。\*/

@OneToMany(cascade={CascadeType.REFRESH,CascadeType.PERSIST,

CascadeType.MERGE,CascadeType.REMOVE},

fetch=FetchType.LAZY,

mappedBy="order")

public Set<OrderItem> getItems() {

return items;

}

public void setItems(Set<OrderItem> items) {

this.items = items;

}

/\* （好难理解！！！）

上面设置了级联保存，所以保存订单的时候，也会将其集合里的所有订单项保存。

往订单中放入订单项的时候，如果采用Set<OrderItem>集合的方式不是很方便，

所以为了添加方便，提供一个【增加订单项的方法】\*/

public void addOrderItem(OrderItem orderItem){

/\*因为订单项是关系的维护端，只有它才能够更新外键字段(往外键字段设值)，

所以必须通过订单项来设订单项跟订单之间的关系.this代表订单\*/

orderItem.setOrder(this);

//将订单项加入到订单中。

this.items.add(orderItem);

}

}

**2. OrderItem.java(订单项)**

package cn.itcast.bean;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.ManyToOne;

//订单项（强调一对多，多对一的关系)——————>双向关联

/\* 在JPA当中，当出现一对多或是多对一的关联关系的时候，多的一方为关联关系的维护方。关系的维护方即就是要负责【外键记录/字段的更新】，关系被维护方是没有权利更新外键记录/字段的。————>也就是说，订单项orderItem为关联关系的维护方\*/

@Entity

public class OrderItem {

private Integer id;//商品id

private String productName;//商品名称

private Float sellPrice=0f;//商品价钱

//在订单项OrderItem中要关联其对应的订单——————【多对一】

private Order order;

@Id @GeneratedValue

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

@Column(length=40,nullable=false)

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

@Column(nullable=false)

public Float getSellPrice() {

return sellPrice;

}

public void setSellPrice(Float sellPrice) {

this.sellPrice = sellPrice;

}

/\*【多对一的注解】————该注解是为了告诉JPA:订单项orderItem和订单order是多对一的关系

【注意】

——————>级联策略cascade

1.在订单项中不需要执行级联持久化/级联保存PERSIST动作，因为该动作一般只会在订单中执行，因为保存订单顺便级联保存其订单项，而不是保存订单项去级联保存期父对象。

2.同时在订单项中也不需要定义级联删除REMOVE动作，因为我们不可能因为删除了某一条订单项记录，而还要顺带级联删除其整个订单，只是不符合常理的。

3.级联刷新REFRESH和级联更新MERGE动作要用到，因为当某一条订单项记录被修改了，我们很有必要对整个订单进行刷新和更新动作。

——————>加载策略fetch

【只要是xxxToOne的关系映射，延迟属性默认都是立即加载(高效快速)】

——————>是否可选策略optional

值为true，代表该对象是可选的，也就是说可以没有该对象，即就是该对象存放的值可以为null(空)。

值为false，代表该对象是必须要有的，即就是该对象在数据库中存储的值不可以为空。

@JoinColumn是【在维护端定义外键】，类似于hibernate中多对一的时候设置的column=外键的名称。\*/

@ManyToOne(cascade={CascadeType.REFRESH,CascadeType.MERGE},

fetch=FetchType.EAGER,

optional=false)

@JoinColumn(name="order\_id")

public Order getOrder() {

return order;

}

public void setOrder(Order order) {

this.order = order;

}

}

**3. OneToManyTest.java(多对一的单元测试)**

package cn.JUnit.Test;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import org.junit.BeforeClass;

import org.junit.Test;

import cn.itcast.bean.Order;

import cn.itcast.bean.OrderItem;

//多对一的单元测试

public class OneToManyTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {}

//第一步：通过创建factory对象来导致表的生成。从表的情况来测试数据实体定义是否是成功的。

@Test

public void save(){

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();//开启事务

Order order = new Order();

order.setAllPrice(34f);

order.setOrderId("999");//主键

OrderItem orderItem1 = new OrderItem();

orderItem1.setProductName("zuqiu");

orderItem1.setSellPrice(90f);

OrderItem orderItem2 = new OrderItem();

orderItem2.setProductName("lanqiu");

orderItem2.setSellPrice(150f);

//往订单里面分别添加订单项

order.addOrderItem(orderItem1);

order.addOrderItem(orderItem2);

em.persist(order);//保存订单

em.getTransaction().commit();//提交事务

em.close();

factory.close();

}

}

**三．【JPA中一对一双向关联与级联操作】**

**1. IdCard.java(身份证实体类)**

package cn.itcast.bean;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.OneToOne;

//身份证实体类————>人员和身份证之间是【一对一】的关联关系，需要在每个实体中都定义另外一个实体

//【注意】：当关联关系是一对一的时候，JPA规范并没有指定谁为关系维护端，我们定义Person即可。

@Entity

public class IdCard {

private Integer id;

private String cardno;

private Person person;

public IdCard() {}

public IdCard(String cardno) {

this.cardno = cardno;

}

@Id

@GeneratedValue

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

@Column(length=18,nullable=false)

public String getCardno() {

return cardno;

}

public void setCardno(String cardno) {

this.cardno = cardno;

}

/\*mappedBy说明【IdCard是关联关系的被维护端】

cascade定义级联策略：不能有级联删除策略(身份证号没了，不能人就没了)，要根据业务需求而定。

optional代表是否是可选的，当前应用的意思是有了身份证，是否一定要有这个人，那当然了！

fetch代表加载策略，默认立即加载。\*/

@OneToOne(cascade={CascadeType.PERSIST,CascadeType.MERGE,

CascadeType.REFRESH},mappedBy="idcard",

optional=false,fetch=FetchType.EAGER)

public Person getPerson() {

return person;

}

public void setPerson(Person person) {

this.person = person;

}

}

**2. Person.java(人员实体类)**

package cn.itcast.bean;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.OneToOne;

//人员实体类————>人员和身份证之间是【一对一】的关联关系，需要在每个实体中都定义另外一个实体

//【注意】：当关联关系是一对一的时候，JPA规范并没有指定谁为关系维护端，我们定义Person即可。

@Entity

public class Person {

private Integer id;

private String name;

private IdCard idcard;

public Person() {}

public Person(String name) {

this.name = name;

}

@Id

@GeneratedValue

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

@Column(length=10,nullable=false)

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

//在关联关系的维护端定义外键

@OneToOne(optional=false,cascade={CascadeType.ALL})

@JoinColumn(name="idcard\_id")

public IdCard getIdcard() {

return idcard;

}

public void setIdcard(IdCard idcard) {

this.idcard = idcard;

}

}

**3. OneToOneTest.java（一对一测试）**

package cn.Junit.Test;

import static org.junit.Assert.\*;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import org.junit.BeforeClass;

import org.junit.Test;

import cn.itcast.bean.IdCard;

import cn.itcast.bean.Person;

public class OneToOneTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

@Test

public void save() {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();

Person person = new Person("zhangsan");

IdCard idcard = new IdCard("62052319901221517x");

idcard.setPerson(person);

person.setIdcard(idcard);

em.persist(person);//保存实体

em.getTransaction().commit();

em.close();

factory.close();

}

}

**四．【JPA中多对多双向关联与级联操作】**

**1. Student.java(学生实体)**

package cn.itcast.bean;

import java.util.HashSet;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.JoinTable;

import javax.persistence.ManyToMany;

//学生实体——————>【多对多关联关系】

/\*【多对多关联关系】的关联关系维护方是谁，JPA也没有明确指出，我们可以定义Student作为关联关系的维护方。

多对多关联关系，是由中间表(关联表)来完成关联动作的！

比如建立关系：student.getTeacher().add(teacher);体现在中间表就相当于insert into ...语句。

比如解除关系：student.getTeacher().remove(teacher);体现在中间表就相当于delete from ...语句。这些操作都是在关联关系的维护端完成的。\*/

@Entity

public class Student {

private Integer id;

private String name;

//关联教师实体

private Set<Teacher> teacher = new HashSet<Teacher>();

public Student() {}

public Student(String name) {

this.name = name;

}

@Id @GeneratedValue

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

@Column(length=10,nullable=false)

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

/\*在多对多关联关系中，关系的维护方定义关系的外键等操作，包括【中间表的定义】，也必须在维护端定义。

关联表的定义由@JoinTable完成。

joinColumns:代表中间表与关联关系维护方之间的外键定义,即中间表中各外键的属性名

inverseJoinColumns：代表中间表与关联关系的被维护方之间的外键定义。\*/

@ManyToMany(cascade=CascadeType.REFRESH)

@JoinTable(name="student\_teacher",joinColumns=@JoinColumn(name="student\_id"),

inverseJoinColumns=@JoinColumn(name="teacher\_id"))

public Set<Teacher> getTeacher() {

return teacher;

}

public void setTeacher(Set<Teacher> teacher) {

this.teacher = teacher;

}

//提供添加teacher的方法

public void addTeacher(Teacher teacher){

this.getTeacher().add(teacher);

}

//提供删除teacher的方法

public void removeTeacher(Teacher teacher){

/\*删除某个指定的teacher对象，要判断teacher集合中是否包含要删除的teacher，、如果包含，就将其删除。那么如何判断teacher存在于该teacher集合中呢？就需要通过teacher的id来判断了！所以在teacher类中需要重写hashCode()和equals(Object obj)方法来判断了\*/

if(this.teacher.contains(teacher)){ //contains底层会自动调用hashCode方法来判断。

this.getTeacher().remove(teacher);

}

}

}

**2. Teacher.java(教师实体)**

package cn.itcast.bean;

import java.util.HashSet;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

//教师实体——————>【多对多关联关系】

@Entity

public class Teacher {

private Integer id;

private String name;

//关联学生实体

private Set<Student> student = new HashSet<Student>();

public Teacher() {}

public Teacher(String name) {

this.name = name;

}

@Id @GeneratedValue

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

@Column(length=10,nullable=false)

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

/\*当前类是关联关系的【被维护端】

级联策略：根据应用需求，它没有级联保存和级联更新，基本上也用不到级联删除！

声明mappedBy和默认本身就是延迟加载。\*/

@ManyToMany(cascade=(CascadeType.REFRESH),mappedBy="teacher")

public Set<Student> getStudent() {

return student;

}

public void setStudent(Set<Student> student) {

this.student = student;

}

/\* 在关系的维护方student中，要执行删除某个指定的teacher对象的动作，首先就需要判断teacher集合中是否包含要删除的teacher，如果包含，就将其删除。那么如何判断teacher存在于该teacher集合中呢？就需要通过teacher的id来判断了！所以在teacher类中需要重写hashCode()和equals(Object obj)方法用来判断了。\*/

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((id == null) ? 0 : id.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Teacher other = (Teacher) obj;

if (id == null) {

if (other.id != null)

return false;

} else if (!id.equals(other.id))

return false;

return true;

}

}

**3. ManyToManyTest.java(多对多关联关系)**

package cn.JUint.Test;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import org.junit.BeforeClass;

import org.junit.Test;

import cn.itcast.bean.Student;

import cn.itcast.bean.Teacher;

public class ManyToManyTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

//保存动作

@Test

public void save() {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();

//开始保存学生和老师对象

em.persist(new Student("zhangsan"));

em.persist(new Teacher("lisi"));

em.getTransaction().commit();

em.close();

factory.close();

}

//【建立】老师和学生之间的关联关系====添加一条记录进中间表

//注意：当实体处于事务当中并且处于托管态的时候，可以直接运行该方法即可执行更新。

@Test

public void buildTS() {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();

//1.首先要得到关系维护端student，因为维护端才能建立关联关系。

Student student = em.find(Student.class,1);

//2.其次要得到被维护端teacher，以便被维护端建立关联关系。

Teacher teacher = em.getReference(Teacher.class,1);

//3.建立和老师teacher的关系==添加一条记录进中间表

student.addTeacher(teacher);

em.getTransaction().commit();

em.close();

factory.close();

}

//【解除】老师和学生之间的关联关系====从中间表中删除一条记录

@Test

public void deleteTS() {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();

//关联关系的解除

Student student = em.find(Student.class,1);

//getReference代表不用从数据库中装载数据，直接从关联外键中取得。当然也可以使用find方法

Teacher teacher = em.getReference(Teacher.class,1);

student.removeTeacher(teacher);//解除关系

em.getTransaction().commit();

em.close();

factory.close();

}

/\*【删除老师】————>删除关系被维护方。步骤：1.解除关系2.删除

注意，直接删除是不行的，因为中间表中的外键还在，被维护方是没有权限更新中间表的。

要想执行删除动作，必须首先解除关系，再进行删除方可！\*/

@Test

public void deleteTeacher() {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();

Student student = em.find(Student.class,1);

Teacher teacher = em.getReference(Teacher.class,1);

student.removeTeacher(teacher);//首先解除关系

em.remove(teacher);//再执行删除动作

em.getTransaction().commit();

em.close();

factory.close();

}

/\*【删除学生】————>删除关系维护方。可以直接删除，因为它有权限操作中间表\*/

@Test

public void deleteStudent() {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();

Student student = em.getReference(Student.class,1);

em.remove(student);//可以直接删除

em.getTransaction().commit();

em.close();

factory.close();

}

}

**五．【JPA中的联合主键】**

**1. AirLinePK.java(联合主键类)**

package cn.itcast.bean;

import java.io.Serializable;

import javax.persistence.Column;

import javax.persistence.Embeddable;

/\* 联合主键类————>只需要定义作为联合主键的字段的相关定义和描述（起点和终点）

联合主键：由两个或者多个字段组成的主键称之为联合主键

怎么样定义联合主键呢？

我们将组成联合主键的这些字段看成是一个整体，那么我们就可以定义一个主键类来描述这些复合字段。

主键类的定义：例如一条航线的起点和终点就可以作为该条航线的联合主键，即就是AirLinePK类。

——————>【联合主键类】必须要遵守的JPA规范：

要求： 1.必须提供一个public的无参的构造函数.

2.必须要实现序列化接口Serializable.

3.必须要重写hashCode和equals这两个方法

注意： 在航线实体类中只是用到了主键类AirLinePK的两个属性startCity和endCity，

所以要通过注解@Embeddable来告诉JPA的实现产品：当前类AirLinePK用

在实体类AirLine当中，只是使用了自己类(AirLinePK)中定义的属性而已。

也就是说，实体类只是使用主键类中定义的属性作为这个实体的持久化字段！

\*/

@Embeddable

public class AirLinePK implements Serializable{

private String startCity;//起点（航线要求用3个字符来代表一个城市）

private String endCity;//终点

//构造函数

public AirLinePK(){}

public AirLinePK(String startCity, String endCity) {

this.startCity = startCity;

this.endCity = endCity;

}

@Column(length=3)

public String getStartCity() {

return startCity;

}

public void setStartCity(String startCity) {

this.startCity = startCity;

}

@Column(length=3)

public String getEndCity() {

return endCity;

}

public void setEndCity(String endCity) {

this.endCity = endCity;

}

//重写hashCode和equals方法

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + ((endCity == null) ? 0 : endCity.hashCode());

result = prime \* result

+ ((startCity == null) ? 0 : startCity.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

AirLinePK other = (AirLinePK) obj;

if (endCity == null) {

if (other.endCity != null)

return false;

} else if (!endCity.equals(other.endCity))

return false;

if (startCity == null) {

if (other.startCity != null)

return false;

} else if (!startCity.equals(other.startCity))

return false;

return true;

}

}

**2. AirLine.java(航线实体类)**

package cn.itcast.bean;

import javax.persistence.Column;

import javax.persistence.EmbeddedId;

import javax.persistence.Entity;

//【航线实体类】,该实体中需要用到复合主键类AirLinePK

@Entity

public class AirLine {

private AirLinePK id;//复合主键类作为航线类实体的标识符。

private String name;

//构造函数

public AirLine() {}

public AirLine(AirLinePK id) {

this.id = id;

}

public AirLine(String startCity,String endCity,String name) {

this.id = new AirLinePK(startCity, endCity);

this.name = name;

}

//由于该实体的实体标识符是一个可嵌入的类(复合主键类),所以要使用@EmbeddedId对其进行id标识。

@EmbeddedId //等同于@Id，但不能使用@Id

public AirLinePK getId() {

return id;

}

public void setId(AirLinePK id) {

this.id = id;

}

@Column(length=20)

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

**3. AirLineTest.java(联合主键测试)**

package cn.JUnit.Test;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

import org.junit.BeforeClass;

import org.junit.Test;

import cn.itcast.bean.AirLine;

public class AirLineTest {

@BeforeClass

public static void setUpBeforeClass() throws Exception {

}

@Test

public void save() {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("itcast");

EntityManager em = factory.createEntityManager();

em.getTransaction().begin();

//保存记录

em.persist(new AirLine("PEK","SHK","beijing to shanghai"));

em.getTransaction().commit();

em.close();

factory.close();

}

}