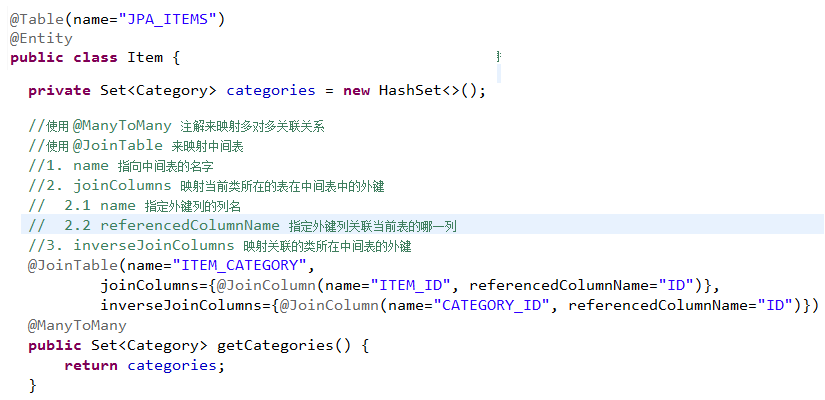
# 单向一一

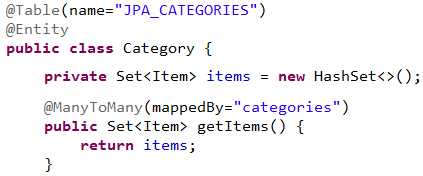
# 双向一一

## 概念

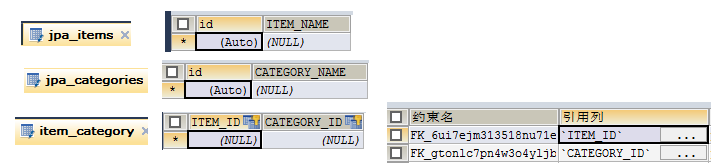
在双向多对多关系中，我们必须指定一个关系维护端(owner side),可以通过 @ManyToMany 注释中指定 mappedBy 属性来标识其为关系维护端。

## 配置（建议单向维护）





## 数据生成信息



## 测试准备

### Item

@Table(name="JPA\_ITEMS")

@Entity

**public** **class** Item {

**private** Integer id;

**private** String itemName;

**private** Set<Category> categories = **new** HashSet<>();

@GeneratedValue

@Id

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

@Column(name="ITEM\_NAME")

**public** String getItemName() {

**return** itemName;

}

**public** **void** setItemName(String itemName) {

**this**.itemName = itemName;

}

//使用 @ManyToMany 注解来映射多对多关联关系

//使用 @JoinTable 来映射中间表

//1. name 指向中间表的名字

//2. joinColumns 映射当前类所在的表在中间表中的外键

// 2.1 name 指定外键列的列名

// 2.2 referencedColumnName 指定外键列关联当前表的哪一列

//3. inverseJoinColumns 映射关联的类所在中间表的外键

@JoinTable(name="ITEM\_CATEGORY",

joinColumns={@JoinColumn(name="ITEM\_ID", referencedColumnName="ID")},

inverseJoinColumns={@JoinColumn(name="CATEGORY\_ID", referencedColumnName="ID")})

@ManyToMany

**public** Set<Category> getCategories() {

**return** categories;

}

**public** **void** setCategories(Set<Category> categories) {

**this**.categories = categories;

}

}

### Category

@Table(name="JPA\_CATEGORIES")

@Entity

**public** **class** Category {

**private** Integer id;

**private** String categoryName;

**private** Set<Item> items = **new** HashSet<>();

@GeneratedValue

@Id

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

@Column(name="CATEGORY\_NAME")

**public** String getCategoryName() {

**return** categoryName;

}

**public** **void** setCategoryName(String categoryName) {

**this**.categoryName = categoryName;

}

@ManyToMany(mappedBy="categories")

**public** Set<Item> getItems() {

**return** items;

}

**public** **void** setItems(Set<Item> items) {

**this**.items = items;

}

}

### 准备JPATestManyMany

**public** **class** JPATestManyMany {

**private** EntityManagerFactory entityManagerFactory;

**private** EntityManager entityManager;

**private** EntityTransaction transaction;

@Before

**public** **void** init(){

entityManagerFactory = Persistence.*createEntityManagerFactory*("jpa\_test1");

entityManager = entityManagerFactory.createEntityManager();

transaction = entityManager.getTransaction();

transaction.begin();

}

@After

**public** **void** destroy(){

transaction.commit();

entityManager.close();

entityManagerFactory.close();

}

}

# 操作

## 保存（双向维护）

//多对所的保存

@Test

**public** **void** testManyToManyPersist(){

Item i1 = **new** Item();

i1.setItemName("i-1");

Item i2 = **new** Item();

i2.setItemName("i-2");

Category c1 = **new** Category();

c1.setCategoryName("C-1");

Category c2 = **new** Category();

c2.setCategoryName("C-2");

//设置关联关系

i1.getCategories().add(c1);

i1.getCategories().add(c2);

i2.getCategories().add(c1);

i2.getCategories().add(c2);

c1.getItems().add(i1);

c1.getItems().add(i2);

c2.getItems().add(i1);

c2.getItems().add(i2);

//执行保存

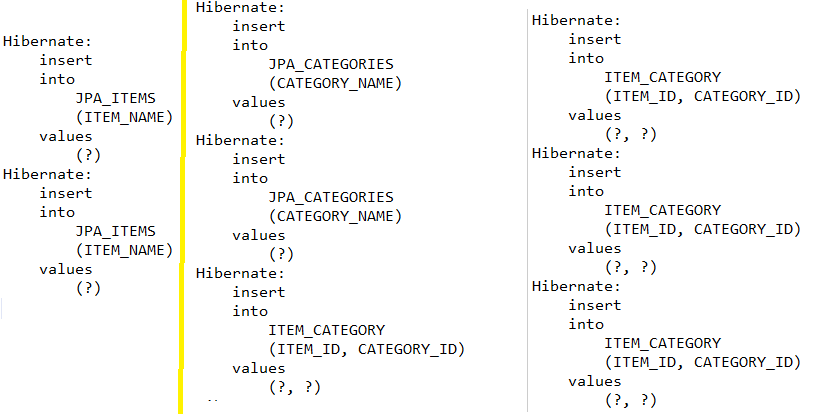
entityManager.persist(i1);

entityManager.persist(i2);

entityManager.persist(c1);

entityManager.persist(c2);

}



## 查询（oid）

//对于关联的集合对象, 默认使用懒加载的策略.

//使用维护关联关系的一方获取, 还是使用不维护关联关系的一方获取, SQL 语句相同.

//对于关联的集合对象, 默认使用懒加载的策略.

//使用维护关联关系的一方获取, 还是使用不维护关联关系的一方获取, SQL 语句相同.

@Test

**public** **void** testManyToManyFind(){

// Item item = entityManager.find(Item.class, 5);

// System.out.println(item.getItemName());

//

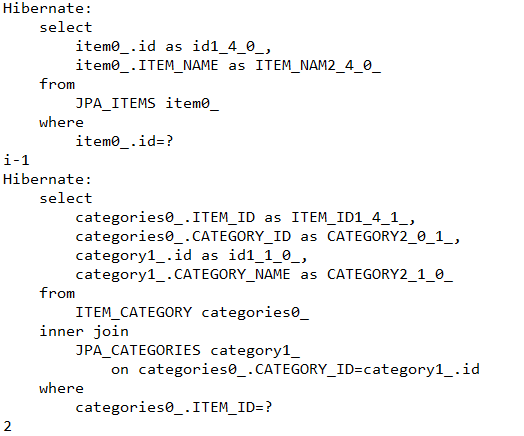
// System.out.println(item.getCategories().size());

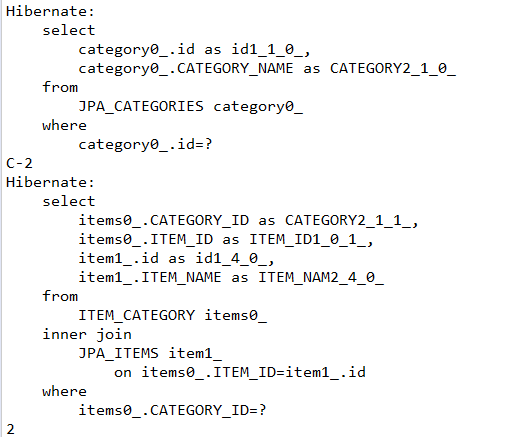
Category category = entityManager.find(Category.**class**, 3);

System.***out***.println(category.getCategoryName());

System.***out***.println(category.getItems().size());

}





## 删除（一方）

### 外键置空，只删一方

### 外键置空，都删

## 更新