

## Hibernate 第五天 Hibernate结合JPA

Hibernate 第五天 Hibernate结合JPA

回顾

今天任务

教学目标

一. JPA简介

1. 什么是JPA?

2. JPA和Hibernate的关系!

二. JPA注解开发步骤

1. 为实体类添加注解

2. 修改Hibernate的核心配置文件

3. 测试

二. JPA的主键策略

1. JPA的主键策略

2. hibernate的主键策略

三. 关联关系

1. 一对多

1.1 创建订单实体

1.2 修改客户实体

1.3 修改Hibernate核心配置文件

1.4 测试

2. 多对多

2.1 创建User实体

2.2 创建Role实体

2.3 配置映射

2.4 测试

3. 一对一

3.1 创建Person实体

3.2 创建Card实体

3.3 Hibernate核心配置文件

3.4 测试

课前默写

作业

面试题

回顾

1. HQL查询
2. QBC查询
3. 本地SQL查询
4. 延迟加载策略
5. 抓取策略
6. 整合连接池
7. 二级缓存

## 今天任务

1. JPA简介和基础操作
2. JPA的主键策略
3. JPA的关联关系

## 教学目标

1. 掌握JPA简介和基础操作
2. 掌握JPA的主键策略
3. 掌握JPA的关联关系

## 一. JPA简介

JPA是Java Persistence API的简称,中文名Java持久层Api,是JDK1.5注解或者Xml描述对象-关系表的映射关系,并将运行期的实体类对象持久化Dao数据库中!注意的是,如果两种映射发生冲突的时候XML优先于注解的方式!

### 1. 什么是JPA?

JPA由EJB 3.0软件专家组开发,作为JSR-220实现的一部分。但它又不限于EJB 3.0,你可以在Web应用、甚至桌面应用中使用。JPA的宗旨是为POJO提供持久化标准规范,由此可见,经过这几年的实践探索,能够脱离容器独立运行,方便开发和测试的理念已经深入人心了。Hibernate3.2+、TopLink 10.1.3以及[OpenJPA](#)都提供了JPA的实现。

JPA的总体思想和现有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:

#### ORM映射元数据

JPA支持XML和[JDK5.0](#)注解两种元数据的形式,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;

#### API

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

#### 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。

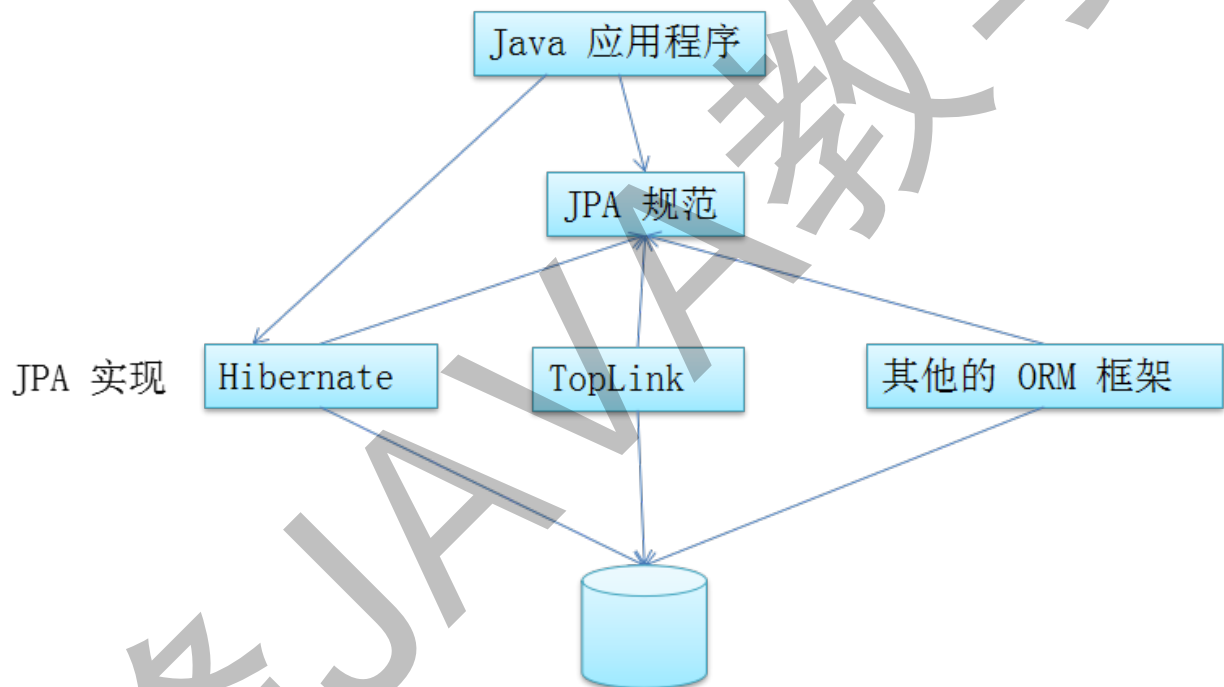
### 2. JPA和Hibernate的关系!

JPA 的目标之一是制定一个可以由很多供应商实现的API，并且开发人员可以编码来实现该API，而不是使用私有供应商特有的API。因此开发人员只需使用供应商特有的API来获得JPA规范没有解决但应用程序中需要的功能。尽可能地使用JPA API，但是当需要供应商公开但是规范中没有提供的功能时，则使用供应商特有的API。

JPA是需要Provider来实现其功能的，Hibernate就是JPA Provider中很强的一个，应该说无人能出其右。从功能上来说，JPA就是Hibernate功能的一个子集。Hibernate 从3.2开始，就开始兼容JPA。Hibernate3.2获得了Sun TCK的JPA(Java Persistence API) 兼容认证。

只要熟悉Hibernate或者其他ORM框架，在使用JPA时会发现其实非常容易上手。例如实体对象的状态，在Hibernate有自由、持久、游离三种，JPA里有new, managed, detached, removed, 明眼人一看就知道，这些状态都是一一对应的。再如flush方法，都是对应的，而其他的再如说Query query = manager.createQuery(sql)，它在Hibernate里写法上是session，而在JPA中变成了manager，所以从Hibernate到JPA的代价应该是非常小的

同样，JDO，也开始兼容JPA。在ORM的领域中，看来JPA已经是王道，规范就是规范。在各大厂商的支持下，JPA的使用开始变得广泛。



## 二. JPA注解开发步骤

### 1. 为实体类添加注解

使用Hibernate的方法操作数据,我们利用xml配置文件将持久化类映射到数据库表,通过面向对象的思想,操作对象间接的操作数据库,但是xml配置写起来比较繁琐,那么我们可以使用Hibernate配置JPA注解的方式进行开发,这样简化开发步骤!

```
@Entity
@Table(name="t_customer3")
public class Customer{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
```

```

private Long id;

@Column(name="name")
private String name;

@Column(name="gender")
private Character gender;

@Column(name="age")
private Integer age;

@Column(name="level")
private String level;
}

```

注解名称	注解含义
@Entity	表明是一个JPA实体,可以使用JPA注解
@Table	table表明实体类对应一个表,name属性对应表名
@Id	指定改变量为主键
@GeneratedValue	主键生成策略
@Column	Column将变量对应数据库列!name指定对应列名

补充: @Column指定实体类变量对应数据库列, name属性指定对应列名,除此之外,在介绍下@Column的其他属性!

属性名称	属性介绍
nullable	boolean类型!此字段是否可以为null
length	指定列长度
precision	指定数字类型位数
scale	指定小数点位数

补充其他属性:

- 临时字段

@Transient 注解, 用于给实体类添加临时属性 (临时属性不需要反映到数据库中)

```

@Transient // 临时字段, 不反映到数据库中
private Boolean married;
public Boolean getMarried() {
    return married;
}
public void setMarried(Boolean married) {
    this.married = married;
}

```

- 默认值字段

默认值字段!创建列时添加default修饰

```

@ColumnDefault("默认值!默认值都是字符串类型");
private Integer age;

```

## 2. 修改Hibernate的核心配置文件

添加JPA注解的实体类,需要将类全路径配置到核心配置文件上!

```

<mapping class="类的全路径"></mapping>

```

## 3. 测试

```

public class AnnotationTest {
    /**
     * 使用注解
     */
    @Test
    public void testAnnotation(){
        Session session = HibernateUtil.openSession();
        Transaction tx = session.beginTransaction();

        Customer cust = new Customer();
        cust.setName("王五2222");
        session.save(cust);

        tx.commit();
    }
}

```

## 二. JPA的主键策略

### 1. JPA的主键策略

JPA 主键策略，没有 Hibernate 主键策略丰富，例如：

```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
@Column(name="id")
private Long id;
```

其他的策略：

- IDENTITY：利用数据库的自增长的能力。适合 mysql
- SEQUENCE：利用数据库的序列机制，适合 Oracle
- TABLE：通过表产生主键，框架借由表模拟序列产生主键，使用该策略可以使应用更易于数据库移植。不同的JPA实现商生成的表名是不同的，如 OpenJPA生成openjpa\_sequence\_table表，Hibernate生成一个hibernate\_sequences表，而TopLink则生成sequence表。这些表都具有一个序列名和对应值两个字段，如SEQ\_NAME和SEQ\_COUNT
- AUTO：自动选择一个最适合底层数据库的主键生成策略。这个是默认选项，即如果只写@GeneratedValue，等价于@GeneratedValue(strategy=GenerationType.AUTO)。

### 2. hibernate的主键策略

如果使用Hibernate对JPA的实现，可以使用Hibernate对主键生成策略的扩展，通过Hibernate的@GenericGenerator实现。

注意：id的数据类型改成String

```
@Id
//声明一个策略通用生成器，name为"system-uuid",策略strategy为"uuid"。
@GenericGenerator(name="system-uuid", strategy="uuid")
@Column(name="id")
private String id;
```

测试:

```
/**
 * UUID主键策略
 */
@Test
public void testUUID(){
    Session session = HibernateUtil.openSession();
    Transaction tx = session.beginTransaction();
```

```

        Customer cust = new Customer();
        cust.setName("王五2222");
        cust.setId(UUID.randomUUID().toString());
        session.save(cust);

        tx.commit();
    }

```

也可以按照如下配置，在注解中指定要使用的策略生成器，这样测试用例中就不用人工生成uuid了。

```

@Id
//声明一个策略通用生成器，name为"system-uuid"，策略strategy为"uuid"。
@GenericGenerator(name="system-uuid", strategy="uuid")
//用generator属性指定要使用的策略生成器。
@GeneratedValue(generator="system-uuid")
@Column(name="id")
private String id;

```

### 三. 关联关系

#### 1. 一对多

##### 1.1 创建订单实体

```

/**
 * 订单（多方）
 */
@Entity
@Table(name="t_order4")
public class Order {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name="orderno")
    private String orderno;

    //关联客户
    @ManyToOne
    @JoinColumn(name="cust_id")
    private Customer customer;
}

```

## 1.2 修改客户实体

添加关联订单,这里使用了级联配置

```
//关联订单
@OneToMany(mappedBy="customer", cascade={CascadeType.ALL})
private Set<Order> orders = new HashSet<Order>();
```

CascadeType.PERSIST 级联持久化（保存）操作（持久保存拥有方实体时，也会持久保存该实体的所有相关数据。）

CascadeType.REMOVE 级联删除操作（删除一个实体时，也会删除该实体的所有相关数据。）

CascadeType.MERGE 级联更新（合并）操作（将分离的实体重新合并到活动的持久性上下文时，也会合并该实体的所有相关数据。）

CascadeType.REFRESH 级联刷新操作（只会查询获取操作）

CascadeType.ALL 包含以上全部级联操作

注意：以上配置需要执行相应的级联操作的特定方法，扩展阅读：<http://sefcertyu.iteye.com/blog/475237>

## 1.3 修改Hibernate核心配置文件

hibernate.hbm.xml中添加Order的映射

```
<mapping class="com.qfedu.hibernate.pojo.Order" />
```

## 1.4 测试

```
/**
 * 测试一对多注解
 */
@Test
public void testOneToMany(){
    Session session = HibernateUtil.openSession();
    Transaction tx = session.beginTransaction();

    Customer cust = new Customer();
    cust.setName("王五3");
    session.save(cust);

    Order o1 = new Order();
    o1.setOrderno("201709003");

    cust.getOrders().add(o1);
```



```
        session.save(cust);
        //session.save(o1);

        tx.commit();
    }
}
```

## 2. 多对多

用户和角色是多对多关系!

### 2.1 创建User实体

```
/**
 * 用户（多方）
 */
@Entity
@Table(name="t_user1")
public class User{
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private Integer id;

    @Column(name="user_name")
    private String name;

    //关联角色
    @ManyToMany(cascade={CascadeType.ALL})
    // @JoinTable: 用于映射中间表
    // joinColumns: 当前方在中间表的外键字段名称
    // inverseJoinColumns: 对方在中间表的外键字段名称
    @JoinTable(
        name="t_user_role1",
        joinColumns=@JoinColumn(name="user_id"),
        inverseJoinColumns=@JoinColumn(name="role_id"))
    private Set<Role> roles = new HashSet<Role>();
}
```

### 2.2 创建Role实体

```
/**
 * 角色（多方）
 */
@Entity
```

```

@Table(name="t_role1")
public class Role{

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private Integer id;
    @Column(name="role_name")
    private String name;

    //关联用户
    @ManyToMany(mappedBy="roles")
    private Set<User> users = new HashSet<User>();
}

```

### 2.3 配置映射

```

<mapping class="com.qfedu.hibernate.pojo.User" />
<mapping class="com.qfedu.hibernate.pojo.Role" />

```

### 2.4 测试

```

/**
 * 测试多对多注解
 */
@Test
public void testManyToMany(){
    Session session = HibernateUtil.openSession();
    Transaction tx = session.beginTransaction();

    User u1 = new User();
    u1.setName("Helen");

    Role r1 = new Role();
    r1.setName("VIP");

    u1.getRoles().add(r1);

    session.save(u1);
    //session.save(r1);

    tx.commit();
}

```

### 3. 一对一

#### 3.1 创建Person实体

```
@Entity
@Table(name="t_person1")
public class Person{

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private Integer id;

    @Column(name="name")
    private String name;

    //关联身份证
    @OneToOne(mappedBy="person", cascade={CascadeType.ALL})
    private Card card;
}
```

#### 3.2 创建Card实体

```
@Entity
@Table(name="t_card1")
public class Card{

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private Integer id;

    @Column(name="card_no")
    private String cardno;

    //关联公民
    @OneToOne
    @JoinColumn(name="person_id")
    private Person person;
}
```

#### 3.3 Hibernate核心配置文件

```
<mapping class="com.qfedu.hibernate.pojo.Person" />
<mapping class="com.qfedu.hibernate.pojo.Card" />
```

### 3.4 测试

```
@Test
public void testOneToOne(){

    Session session = HibernateUtil.openSession();
    Transaction tx = session.beginTransaction();

    Person p = new Person();
    p.setName("老王");

    Card c = new Card();
    c.setCardno("44333222");

    p.setCard(c);

    session.save(p);
    //session.save(c);
    tx.commit();
}
```

#### 总结

一对一为例：针对 mappedBy、@JoinColumn、cascade 的总结

#### 外键由谁来维护

- 1、当关联关系的双方都不配置mappedBy 属性时，那么双方会互相生成外键，并且执行三条sql  
两条插入sql，一条额外的维护外键的sql
- 2、如果有一方配置了mappedBy 属性，那么对方会生成外键
- 3、mappedBy 和 @JoinColumn 不能配置在同一方中
- 4、如果配置在同一方中，以mappedBy为准，@JoinColumn失效
- 5、只能有一方配置 mappedBy

#### 级联操作

- 1、在A设置了级联操作，A就应该被session操作
- 2、在A方设置了级联操作，B就应该被设置为A的属性
- 3、如果A中有外键，那么B应该被设置为A的属性，外键才能被填充

#### 课前默写

1. 使用Hibernate的HQL的过程
2. 使用Hibernate的QBC完成查询的过程
3. 使用Hibernate的本地SQL完成查询的过程

## 作业

1. 使用JPA的方式完成Hibernate课程第三天的员工部门作业

## 面试题

1. JPA与Hibernate的联系和区别
2. JPA的基础操作
3. JPA的关联关系
4. JPA的缓存