

Spring Data JPA @Entity之间的关联关系注解如何正确使用？

 juejin.cn/post/7143411144764424199

September 15, 2022

本文已参与「新人创作礼」活动，一起开启掘金创作之路。

首先,实体与实体之间的关联关系一共分为四种,分别为OneToOne、OneToMany、ManyToOne和ManyToMany;而实体之间的关联关系又分为双向和单向。实体之间的关联关系是在JPA使用中最容易发生问题的地方。

1、OneToOne关联关系


@OneToOne一般表示对象之间一对一的关联关系，它可以放在field上面，也可以放在get/set方法上面。其中JPA协议有规定，如果配置双向关联，维护关联关系的是拥有外键的一方，而另一方必须配置mappedBy;如果是单项关联，直接配置在拥有外键的一方即可。

举例说明：

user表是用户的主信息，user_info是用户的拓展信息，两者之间是一一对一的关系。user_info表里面有一个user_id作为关联关系的外键,如果是单项关联，我们的写法如下：



less


 代码解读

复制代码

```
@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String name;
    private String email;
    private String sex;
    private String address;
}
```

我们只需要在拥有外键的一方配置@OneToOne注解就可以了

▼
less

 代码解读


复制代码

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString(exclude = "user")
public class UserInfo {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    private Integer ages;
    private String telephone;
    @OneToOne
    private User user;
}
```

这就是单向关联关系,那么如何设置双向关联关系呢？我们保持UserInfo不变，在User实体对象里面添加一段代码即可



less

 代码解读

复制代码

```
@OneToOne(mappedBy = "user")
private UserInfo userInfo;

@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    private String name;
    private String email;
    private String sex;
    private String address;
    @OneToOne(mappedBy = "user")
    private UserInfo userInfo;
}
```

1.1 解读OneToOne源码

▼

csharp

 代码解读

复制代码

```
public @interface OneToOne {  
  
    Class targetEntity() default void.class;  
  
    CascadeType[] cascade() default {};  
  
    FetchType fetch() default EAGER;  
  
    boolean optional() default true;  
  
    String mappedBy() default "";  
  
    boolean orphanRemoval() default false;  
}
```

targetEntity：作为关联目标的实体类。

cascade：级联操作策略，就是我们常说的级联操作。

fetch：数据获取方式EAGER(立即加载)/LAZY(延迟加载) **optional**：表示关联的实体是否存在null值 **mappedBy**：关联关系被谁维护的一方对象里面的属性名字,双向关联的时候必填。

1.2 mappedBy 注意事项

- 只有关联关系的维护方才能操作两个实体之间外键的关系。被维护方即使设置维护方属性进行存储也不会更新外键关联
- mappedBy不能与@JoinColumn或者@JoinTable同时使用，因为没有任何意义，关联关系不在这里面维护。
- mappedBy的值是指另一方的实体里面属性的字段，而不是数据库字段，也不是实体的对象的名字。也就是维护关联关系的一方属性字段名称，或者加了@JoinColumn 或 @JoinTable 注解的属性字段名称。如上面的User例子user里面的mappedBy的值，就是userinfo里面的user字段的名称。

1.3 CascadeType 用法

在CascadeType的用法中，CascadeType的枚举值只有5个，分别如下：


- CascadeType.PERSIST 级联新建
- CascadeType.REMOVE 级联删除
- CascadeType.PEFRESH 级联刷新
- CascadeType.MERGE 级联更新
- CascadeType.ALL 四项全选

测试级联新建和级联删除:

第一步：在@OneToOne上面添加 cascade = {CascadeType.PERSIST,CascadeType.REMOVE}，代码如下所示：

▼

less


 代码解读

复制代码

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString(exclude = "user")
public class UserInfo {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    private Integer ages;
    private String telephone;
    @OneToOne(cascade = {CascadeType.PERSIST, CascadeType.REMOVE})
    private User user;
}
```

新增测试方法：

SCSS

 代码解读

复制代码

```
@Test
public void testPersistAndRemove(){
    User user = User.builder()
        .name("jackxx")
        .email("123456@126.com")
        .build();
    UserInfo userInfo = UserInfo.builder()
        .ages(12)
        .user(user)
        .telephone("12345678")
        .build();
    // 新建UserInfo,级联新建User
    userInfoRepo.save(userInfo);
    // 删除UserInfo,级联删除User
    userInfoRepo.delete(userInfo);
}
```

执行SQL如下所示：

```
Hibernate: insert into user (address, email, name, sex) values (?, ?, ?, ?)
Hibernate: insert into user_info (ages, telephone, user_id, id) values (?, ?, ?, ?)
Hibernate: select userinfo0_.id as id1_4_0_, userinfo0_.ages as ages2_4_0_, userinfo0_.telephone as telephon3_4_0_, userinfo0_.user_id as user_id4_4_0_,
        user1_.id as id1_3_1_, user1_.address as address2_3_1_, user1_.email as email3_3_1_, user1_.name as name4_3_1_, user1_.sex as sex5_3_1_ from user_info
        userinfo0_ left outer join user user1_ on userinfo0_.user_id=user1_.id where userinfo0_.id=?
Hibernate: delete from user_info where id=?
Hibernate: delete from user where id=?
```

@稀土掘金技术社区

从上面运行结果中可以看到，执行insert的时候，会先插入user表，再插入user_info表。执行delete的时候，先删除user_info表中数据，再删除user表中的数据。

上面只是讲述级联删除的场景，下面我们再说一下[关联关系的删除场景](#)该怎么做？


1.4 orphanRemoval属性用法

orphanRemoval表示当关联关系被删除的时候，是否应用级联删除。

首先我们，沿用上面的例子，[当我们删除userinfo的时候，把user置空](#)



ini

 代码解读


复制代码

```
userInfo.setUser(null);  
userInfoRepo.delete(userInfo);
```

再看运行结果



sql

 代码解读


复制代码

```
Hibernate: delete from user_info where id=?
```

我们只删除了UserInfo的数据,没有删除user的数据,说明没有进行级联删除,我们将orphanRemoval属性设置为true



ini

 代码解读


复制代码

```
@OneToOne(cascade = {CascadeType.PERSIST}, orphanRemoval = true)  
private User user;
```


测试代码：



SCSS

 代码解读

复制代码

```
@Test
public void testRemove(){
    User user = User.builder()
        .name("jackxx")
        .email("123456@126.com")
        .build();
    UserInfo userInfo = UserInfo.builder()
        .ages(12)
        .user(user)
        .telephone("12345678")
        .build();
    // 新建UserInfo,级联新建User
    userInfoRepo.save(userInfo);
    userInfo.setUser(null);
    // 删除UserInfo,级联删除User
    userInfoRepo.delete(userInfo);
}
```

执行结果如下所示：

```
Hibernate: update user_info set ages=?, telephone=?, user_id=? where id=?
Hibernate: delete from user where id=?
Hibernate: delete from user_info where id=?
```

@稀土掘金技术社区

在执行结果中多了一条update语句，是因为去掉了CascadeType.REMOVE，这个时候不会进行级联删除了。当我们把user对象更新为null的时候，就会执行一个update语句把关联关系去掉。

1.5 orphanRemoval 和 CascadeType.REMOVE的区别

- CascadeType.REMOVE 级联删除，先删除user表的数据，再删除user_info表的数据。（因为存在外键关联，无法先删除user_info表的数据）
- orphanRemoval = true 先将user_info表中的数据外键user_id 更新为 null，然后删除user_info表的数据,再删除user表的数据。


2、@JoinColumn & @JoinColumn

这两个注解是集合关系，他们可以同时使用，@JoinColumn表示单字段，@JoinColumns表示多个@JoinColumn

@JoinColumn源码

▼

java

 代码解读

复制代码

```
public @interface JoinColumn {

    String name() default "";

    String referencedColumnName() default "";

    boolean unique() default false;

    boolean nullable() default true;

    boolean insertable() default true;

    boolean updatable() default true;

    String columnDefinition() default "";


    String table() default "";

    ForeignKey foreignKey() default @ForeignKey(PROVIDER_DEFAULT);
}
```

- **name**：代表外键的字段名。
- **referencedColumnName**：关联表对应的字段，如果不注明，默认就是关联表的主键
- **unique**：外键字段是否唯一
- **nullable**：外键字段是否允许为空
- **insertable**：是否跟随一起新增
- **updateable**：是否跟随一起更新
- **columnDefinition**：为列生成DDL时使用的SQL片段
- **foreignKey**：外键策略



arduino

 代码解读

复制代码


```
// 外键策略
public enum ConstraintMode {

    // 创建外键约束
    CONSTRAINT,
    // 不创建外键约束
    NO_CONSTRAINT,
    // 采用默认行为
    PROVIDER_DEFAULT
}
```

foreignKey的用法：



less


 代码解读

复制代码

```
@OneToOne(cascade = {CascadeType.PERSIST}, orphanRemoval = true)
@JoinColumn(foreignKey = @ForeignKey(ConstraintMode.NO_CONSTRAINT), name = "user_id")
private User user;
```

JoinColumns的用法：

less

 代码解读


复制代码

```
@OneToOne(cascade = {CascadeType.PERSIST}, orphanRemoval = true)
@JoinColumns({
    @JoinColumn(name = "user_id", referencedColumnName = "ID"),
    @JoinColumn(name = "user_ZIP", referencedColumnName = "ZIP")
})
private User user;
```

3、@ManyToOne & @OneToMany

@ManyToOne代表多对一的关联关系，而@OneToMany代表一对多，一般两个成对使用表示双向关联关系。在JPA协议中也是明确规定：**维护关联关系的是拥有外键的一方，而另一方必须配置mappedBy**

▼
csharp

 代码解读

复制代码

```
public @interface OneToMany {  
  
    Class targetEntity() default void.class;  
  
    CascadeType[] cascade() default {};  
  
    FetchType fetch() default LAZY;  
  
    String mappedBy() default "";  
  
    boolean orphanRemoval() default false;  
}
```

```
public @interface ManyToOne {  
  
    Class targetEntity() default void.class;  
  
    CascadeType[] cascade() default {};  
  
    FetchType fetch() default EAGER;  
  
    boolean optional() default true;  
}
```

使用这两个字段，**需要注意以下几点**：


- @ManyToOne 一定是维护外键关系的一方，所以没有mappedBy字段；
- @ManyToOne 删除的时候**一定不能把One的一方删除了**，所以也没有orphanRemoval选项；
- @ManyToOne 的Lazy效果和 @OneToOne 的一样，所以和上面的用法基本一致；
- @OneToMany 的Lazy是有效果的；

3.1 Lazy机制

举例说明：假设User有多个地址Address



less

 代码解读


复制代码

```
@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    private String name;
    private String email;
    private String sex;
    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY)
    private List<UserAddress> address;
}
```

@OneToMany 双向关联并且采用LAZY的机制



less

 代码解读


复制代码

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class UserAddress {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String address;

    @ManyToOne(cascade = CascadeType.ALL)
    private User user;
}
```

测试代码：

SCSS

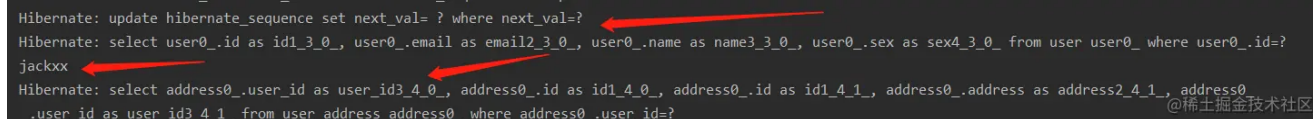
 代码解读

复制代码

```
@Test
@Transactional
public void testUserAddress(){
    User user = User.builder()
        .name("jackxx")
        .email("123456@126.com")
        .build();
    UserAddress userAddress = UserAddress.builder()
        .address("shanghai1")
        .user(user)
        .build();
    UserAddress userAddress1 = UserAddress.builder()
        .address("shanghai2")
        .user(user)
        .build();
    addressRepo.saveAll(Lists.newArrayList(userAddress,userAddress1));

    User u = userRepo.findById(1).get();
    System.out.println(u.getName());
    System.out.println(u.getAddress());
}
```

运行结果如下所示：



```
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: select user0_.id as id1_3_0_, user0_.email as email2_3_0_, user0_.name as name3_3_0_, user0_.sex as sex4_3_0_ from user user0_ where user0_.id=?
jackxx
Hibernate: select address0_.user_id as user_id3_4_0_, address0_.id as id1_4_0_, address0_.id as id1_4_1_, address0_.address as address2_4_1_, address0_.user_id as user_id3_4_1_ from user_address address0_ where address0_.user_id=?
```

可以看到当我们想要输出Address信息的时候，才会加载Address的信息


4、ManyToMany

@ManyToMany代表多对多的关联关系、这种关联关系任何一方都可以维护关联关系。

我们假设user表和room表是多对多的关系，如下所示：




less

 代码解读

复制代码

```
@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    private String name;
    @ManyToMany(mappedBy = "users")
    private List<Room> rooms;
}
```

▼
less

 代码解读

复制代码

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class Room {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;


    private String title;

    @ManyToMany
    private List<User> users;
}
```

这种方法实不可取，当用到@ManyToMany的时候一定是三张表，不要想着建两张表，两张表肯定是违背表的原则

改进方法：创建中间表 修改Room里面的内容

less

 代码解读

复制代码

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class Room {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String title;

    @ManyToMany
    @JoinTable(name = "user_room",
        joinColumns = @JoinColumn(name = "room_id"),
        inverseJoinColumns = @JoinColumn(name = "user_id"))
    private List<User> users;
}
```

可以看到我们通过@JoinTable注解创建一张中间表，并且添加了两个设定的外键，我们来看看@JoinTable的源码：

SCSS

 代码解读

复制代码

```
public @interface JoinTable {

    String name() default "";

    String catalog() default "";

    String schema() default "";

    JoinColumn[] joinColumns() default {};

    JoinColumn[] inverseJoinColumns() default {};

    ForeignKey foreignKey() default @ForeignKey(PROVIDER_DEFAULT);

    ForeignKey inverseForeignKey() default @ForeignKey(PROVIDER_DEFAULT);

    UniqueConstraint[] uniqueConstraints() default {};

    Index[] indexes() default {};
}
```

- **name** : 中间表名称
- **joinColumns** : 维护关联关系一方的外键字段的名字
- **inverseJoinColumns** : 另一方表的外键字段的名字

在现实开发中，@ManyToMany注解用的比较少,一般都会使用成对的@ManyToOne 和 @OneToMany代替，因为我们的中间表可能还有一些约定的公共字段，如 ID, update_time, create_time等其他字段


4.1 利用@ManyToOne 和 @OneToMany表达多对多的关联关系

在上面的Demo中，我们稍作修改，新建一张user_room 中间表来存储双方的关联关系和额外字段

如下所示： user_room中间表



less

 代码解读

复制代码

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class user_room {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date createTime;
    private Date updateTime;
    @ManyToOne
    private User user;
    @ManyToOne
    private Room room;
}
```

user表



less

 代码解读


复制代码

```
@Data
@Entity
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;
    @OneToMany(mappedBy = "user")
    private List<user_room> userRoomList;
}
```

room表



less

 代码解读

复制代码

```
@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class Room {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @OneToMany(mappedBy = "room")
    private List<user_room> roomList;
}
```

好了本次就介绍到这！！！！