**Hibernate，JPA 对象关系映射之关联关系映射策略**

**关联关系映射**

关联关系映射，是映射关系中比较复杂的一种映射关系，总的说来有一对一、一对多和多对多几种关系。细分起来他们又有单向和双向之分。

3 [评论：](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#icomments)



[王 汉敏](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#authorN1001C), 软件工程师, IBM

2013 年 6 月 17 日

* **内容**



在 IBM Bluemix 云平台上开发并部署您的下一个应用。

[**开始您的试用**](https://developer.ibm.com/sso/bmregistration?lang=zh_CN&ca=dwchina-_-bluemix-_-j-lo-jparelated-_-sidebar)

**关联关系映射**

关联关系映射，是映射关系中比较复杂的一种映射关系，总的说来有一对一、一对多和多对多几种关系。细分起来他们又有单向和双向之分。下面我们逐一介绍一下。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#ibm-pcon)

**单向 OneToOne**

单向一对一是关联关系映射中最简单的一种，简单地说就是可以从关联的一方去查询另一方，却不能反向查询。我们用下面的例子来举例说明，清单 1 中的 Person 实体类和清单 2 中的 Address 类就是这种单向的一对一关系，我们可以查询一个 Person 的对应的 Address 的内容，但是我们却不能由一个 Address 的值去查询这个值对应的 Person。

**清单 1. 单向一对一关系的拥有端**

@Entity

public class Person implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private int age;

@OneToOne

private Address address;

// 　 Getters & Setters

}

**清单 2. 单向一对一关系的反端**

@Entity

public class Address implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String street;

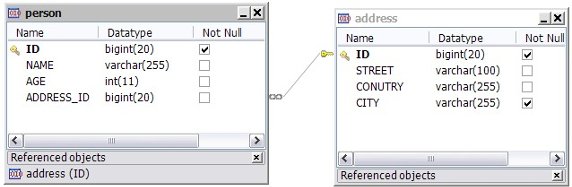
private String city;

private String country;

// Gettes& Setters

}

**图 1. 单向一对一关系对应的 ER 图**



从图 1 他们的 ER 图上可以看出，这种单向的一对一关系在数据库中是以外键的形式被映射的。其中关系的发出端存储一个指向关系的接收端的一个外键。在这个例子中 Person 表中的 ADDRESS\_ID 就是指向 address 标的一个外键，缺省情况下这个外键的字段名称，是以它指向的表的名称加下划线“\_”加“ID”组成的。当然我们也可以根据我们的喜好来修改这个字段，修改的办法就是使用 @JoinColumn 这个注解。在这个例子中我们可以将这个注解注释在 Person 类中的 Address 属性上去。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#ibm-pcon)

**双向 OneToOne**

双向关系有一方为关系的拥有端，另一方是关系的反端，也就是“Inverse”端。在这里例子中 Person 拥有这个关系，而 Address 就是关系的“Inverse”端。Address 中我们定义了一个 person 属性，在这个属性上我们使用了 @OneToOne 注解并且定义了他的“mappedBy”属性，这个在双向关系的“Inverse”端是必需的，在下面将要介绍的双向关系中也要用到这个属性。

**清单 3. 双向一对一关系中的接受端**

@Entity

public class Address implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String street;

private String city;

private String country;

@OneToOne(mappedBy = "address")

private Person person;

// Gettes& Setters

}

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#ibm-pcon)

**单向 OneToMany**

单向关系的一对多我们可以用清单 4 和清单 5 来说明。在关系的发出端 Person 中我们使用 OneToMany 这个注解对 cellPhones 这个属性进行了注释，cellPhones 中存储的是 CellPhone 的一个 List 对象，JPA 就是用这种方式实现一对多的。

**清单 4. 单向一对多关系的发出端**

public class Person implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private int age;

@OneToMany

private List<CellPhone> cellPhones;

// Getters and Setters

}

**清单 5. 单向一对多关系的接收端**

@Entity

public class CellPhone implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String manufacture;

private String color;

private Long phoneNo;

// Getters and Setters

}

**图 2. 单向一对多关系对应的 ER 图**



在一对多关联关系映射中，默认是以中间表的方式来映射这种关系的。如在本例中，中间表为 person\_cellphone，表的名称为关系的拥有端和 Inverse 端中间用下划线连接。中间表的字两个字段分别为两张表的得表名加下划线“\_”加 ID 组成。当然我们也可以改表这种默认的中间表的映射方式，我们可以在关系的拥有端使用 @JoinClolum 来使用外键的方式映射这个关系。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#ibm-pcon)

**双向 OneToMany**

**清单 6. 双向一对多关系的接受端**

@Entity

public class Person implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private int age;

@OneToMany(mappedBy = "person")

private List<CellPhone> cellPhones;

// Getters and Setters

}

**清单 7. 双向一对多关系的发出端**

@Entity

public class CellPhone implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String manufacture;

private String color;

private Long phoneNo;

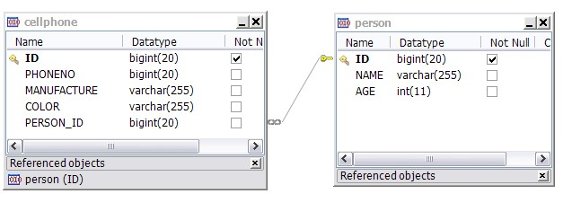
@ManyToOne

private Person person;

// Getters and Setters

}

**图 3. 双向一对多关系对应的 ER 图**



[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#ibm-pcon)

**单向 ManyToMany**

多对多关联关系中只能通过中间表的方式进行映射。本例的单向多对多关系如下所示。

在清单 8 中我们使用了 ManyToMany 这个注解来对 Teacher 中的 Students 进行注释，其中 Teacher 就是关系的发出端。而在 Student 中我们并没有作任何定义，这是单向多对多的所要求的。

**清单 8. 单向多对多关系的发出端**

@Entity

public class Teacher implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private Boolean gender;

private int age;

private int height;

@ManyToMany

private List<Student> students;

// Getters and Setters

}

**清单 9. 单向多对多关系的反端**

@Entity

public class Student implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private Boolean gender;

private int age;

private int height;

//Getters and Setters

}

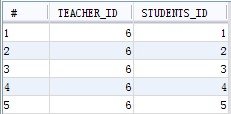
**图 4. Teacher 对应得数据库表**



**图 5. Students 对应得数据库表**



**图 6. 中间生成表**



[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#ibm-pcon)

**双向 ManyToMany**

**清单 10. 双向多对多关系的拥有端**

@Entity

public class Teacher implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private Boolean gender;

private int age;

private int height;

@ManyToMany

private List<Student> students;

// Getters and Setters

}

**清单 11. 双向多对多关系的反端**

@Entity

public class Student implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

private Boolean gender;

private int age;

private int height;

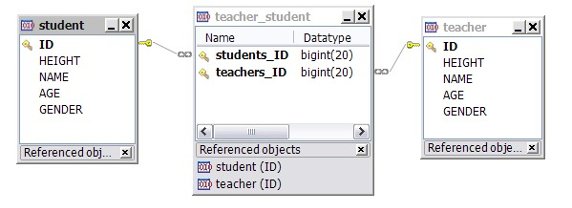
@ManyToMany(mappedBy = "students")

private List<Teacher> teachers;

//Getters and Setters

}

**图 7. 双向多对多关系对应的 ER 图**



[**回页首**](http://www.ibm.com/developerworks/cn/java/j-lo-jparelated/#ibm-pcon)

**总结**

关联关系映射，是对象映射关系中相对复杂的一种，但也是用处最多的一种，因为数据中的表不可能都是单独存在，彼此之间必定存在千丝万缕的联系，这也是关系型数据库的特征所在。同样关联关系的映射，也是对象关系映射中的难点，重点，仔细揣摩，也还是很容易理解掌握的。