# 第 3 章 MyBatis 的关联映射

本章学习目标

- 理解表与表之间的关系
- 掌握一对一关系的映射方法
- 掌握一对多关系的映射方法
- 掌握多对多关系的映射方法

在实际开发中，根据业务需要，数据表之间往往会存在某种关联关系，例如，一对一、一对多等。当程序操作数据库时，如果被操作的表与其他表相关联，那么处理这些表中数据时必须要考虑它们之间的关联关系。为此，作为流行的 ORM 框架，MyBatis 提供了映射表之间关联关系的功能，如此一来，使用 MyBatis 能够以简洁的方式操作多张数据表。接下来，本章将对 MyBatis 的关联映射做详细讲解。

## 3.1 表与表之间的关系

在学习 MyBatis 的关联映射前，首先要了解表与表之间的关系。表与表之间的关系主要包括一对一、一对多和多对多等，接下来本节将对这几种关系做详细讲解。

在一对一关系中，一方数据表中的一条记录最多可以和另一方数据表中的一条记录相关。例如，现实生活中学生与校园卡就属于一对一的关系，一个学生只能拥有一张校园卡，一张校园卡只能属于一个学生，如图 3.1 所示。

在一对多关系中，主键数据表中的一条记录可以和另外一张数据表的多条记录相关，例如，现实生活中班级与学生的关系就属于一对多的关系，一个班级可以有很多学生，一个学生只能属于一个班级，如图 3.2 所示。

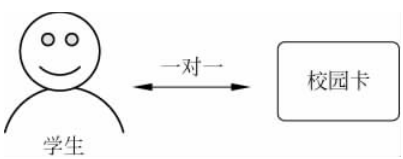



图 3.1 一对一关系

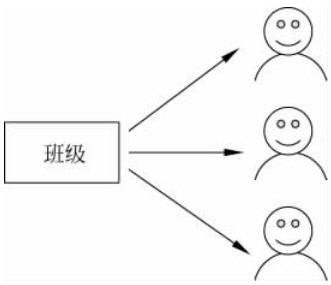



图 3.2 一对多关系

在多对多关系中，两个数据表里的每条记录都可以和另一张数据表里任意数量的记录相关，例如，现实生活中学生与教师就属于多对多的关系，一名学生可以由多名教师授课，一名教师可以为多名学生授课，具体如图 3.3 所示。

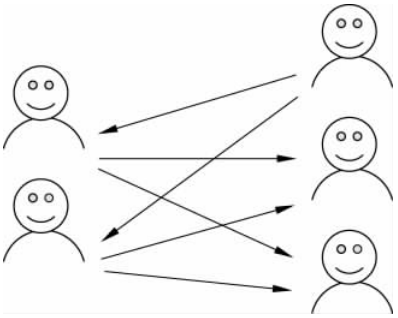


图 3.3　多对多关系

如果直接通过 SQL 语句维护数据表，在维护一对一的表关系时，通常采用在任意一方引入对方的主键作为外键的方式；在维护一对多的表关系时，通常采用在“多”方加入“一”方主键作为外键的方式；在维护多对多的表关系时，通常采用中间表的方式。

然而在企业项目开发中，程序通常是以操作 POJO 对象的形式来操作数据库的，为了满足程序需要，MyBatis 提供了关系映射的功能，这使得开发人员能够以操作 POJO 对象的形式处理表关系。

MyBatis 中支持一对一、一对多、多对多等多种映射方式，在实际应用中，MyBatis 通过映射文件中的< resultMap >元素实现关联映射，接下来本章将展开具体讲解。

## 3.2　一　对　一

在完成数据表设计之后，如果使用 MyBatis 处理一对一的表关系，需要在 MyBatis 的映射文件中添加< association >元素。

< association >元素是映射文件< resultMap >元素的子元素，其配置代码如下所示。

```
1  <resultMap type="" id="">
2          <id property="" column=""/>
3          <result property="" column=""/>
4          <association property="" column=""  javaType="" select="" >
5          </association>
6      </resultMap>
```

< association >元素提供了一系列属性用于维护数据表关系，具体如表 3.1 所示。

**表 3.1　< association >元素的属性**

| 属　性 | 说　明 |
| --- | --- |
| property | 用于指定所映射的类的属性 |
| column | 用于指定数据表中对应的字段，与 property 属性指定的类的属性相对应 |
| javaType | 用于指定所映射的类的属性的类型 |
| jdbcType | 用于指定数据表中对应的字段的类型 |
| fetchType | 用于指定在关联查询时是否延迟加载 |
| select | 用于指定引入嵌套查询的子 SQL 语句 |
| autoMapping | 用于指定是否自动映射 |
| typeHandler | 用于指定一个类型处理器 |

表 3.1 列举了< association >元素的属性,开发人员可根据需要酌情使用。

前文中已经讲过,在处理表关系时,学生与校园卡是一对一的关联关系,接下来,本节将以此为例演示 MyBatis 实现一对一的关联映射。

**1. 数据准备**

(1) 在 MySQL 中创建数据库 chapter03,在数据库 chapter03 中创建数据表 stu_card、数据表 stu,其中,数据表 stu_card 用于映射校园卡信息,数据表 stu 用于映射学生信息,SQL 语句如下所示。

```
DROP DATABASE IF EXISTS chapter03;
CREATE DATABASE chapter03;
USE chapter03;
CREATE TABLE stu_card(
cid INT PRIMARY KEY AUTO_INCREMENT, #ID
balance DOUBLE          #卡内余额
);
CREATE TABLE stu(
sid INT PRIMARY KEY AUTO_INCREMENT, #ID
sname VARCHAR(20),      #学生姓名
age VARCHAR(20),        #学生年龄
course VARCHAR(20),     #科目
cardid INT UNIQUE,      #校园卡 ID
FOREIGN KEY(cardid) REFERENCES stu_card(cid)   #外键
);
```

(2) 向数据表 stu_card 中插入数据,SQL 语句如下所示。

```
INSERT INTO stu_card (balance) VALUES (1000.5);
INSERT INTO stu_card (balance) VALUES (5000.5);
```

(3) 向数据表 stu 中插入数据,SQL 语句如下所示。

```
INSERT INTO stu(sname,age,course,cardid)VALUES('ZhangSan','20','Java',1);
INSERT INTO stu(sname,age,course,cardid)VALUES('liSi','21','Java',2);
```

(4) 通过 SQL 语句测试数据是否插入成功,执行结果如下所示。

```
mysql> select * from stu_card;
+-----+----------+
| cid |  balance |
+-----+----------+
|   1 |   1000.5 |
|   2 |   5000.5 |
+-----+----------+
2 rows in set (0.00 sec)

mysql> select * from stu;
```

```
+-----+----------+------+--------+--------+
| sid | sname    | age  | course | cardid |
+-----+----------+------+--------+--------+
|   1 | ZhangSan |   20 | Java   |      1 |
|   2 | liSi     |   21 | Java   |      2 |
+-----+----------+------+--------+--------+
2 rows in set (0.00 sec)
```

从以上执行结果可以看出，数据插入成功。

**2. 创建工程**

在 Eclipse 中新建 Web 工程 chapter03，将 MyBatis 的驱动 jar 包（mybatis-3.4.6.jar 和 lib 目录下的 jar 包）复制到 lib 目录下，完成 jar 包的导入。

**3. 创建 POJO 类**

在工程 chapter03 的 src 目录下创建 com.qfedu.pojo 包，在 com.qfedu.pojo 包下分别创建两个 POJO 类：Stu 和 StuCard，如例 3-1 和例 3-2 所示。

**【例 3-1】 StuCard.java**

```
1  package com.qfedu.pojo;
2  public class StuCard {
3      private int cid;
4      private double balance;
5      public StuCard() {
6          super();
7      }
8      public StuCard(int cid, double balance) {
9          super();
10         this.cid = cid;
11         this.balance = balance;
12     }
13     public int getCid() {
14         return cid;
15     }
16     public void setCid(int cid) {
17         this.cid = cid;
18     }
19     public double getBalance() {
20         return balance;
21     }
22     public void setBalance(double balance) {
23         this.balance = balance;
24     }
25     @Override
26     public String toString() {
27         return "StuCard [cid=" + cid + ", balance=" + balance + "]";
28     }
29 }
```

**【例 3-2】 Stu.java**

```
1  package com.qfedu.pojo;
2  public class Stu {
3      private int sid;
4      private String sname;
5      private String age;
6      private String course;
7      private StuCard sc;
8      public int getSid() {
9          return sid;
10     }
11     public void setSid(int sid) {
12         this.sid = sid;
13     }
14     public String getSname() {
15         return sname;
16     }
17     public void setSname(String sname) {
18         this.sname = sname;
19     }
20     public String getAge() {
21         return age;
22     }
23     public void setAge(String age) {
24         this.age = age;
25     }
26     public String getCourse() {
27         return course;
28     }
29     public void setCourse(String course) {
30         this.course = course;
31     }
32     public StuCard getSc() {
33         return sc;
34     }
35     public void setSc(StuCard sc) {
36         this.sc = sc;
37     }
38     @Override
39     public String toString() {
40         return "Stu [sid=" + sid + ", sname=" + sname + ", age=" + age + ",
41                course=" + course + ", sc=" + sc + "]";
42     }
43 }
```

以上代码是两个 POJO 类，其中 Stu 类封装了学生信息，StuCard 类封装了校园卡信息，除了 getter 和 setter 方法之外，为方便输出信息，这两个类还提供了 toString 方法。

**4. 创建映射文件**

在工程 chapter03 的 src 目录下分别创建 com. qfedu. mapper 包，在 com. qfedu. mappr 包下创建两个映射文件：StuCardMapper. xml 和 StuMapper. xml，如例 3-3 和例 3-4 所示。

**【例 3-3】 StuCardMapper. xml**

```
<?xml version = "1.0" encoding = "UTF - 8" ?>
<! DOCTYPE mapper
PUBLIC " - //mybatis. org//DTD Mapper 3.0//EN"
"http://mybatis. org/dtd/mybatis - 3 - mapper. dtd">
< mapper namespace = "stucard">
    < select id = "findStuCardBycid" parameterType = "Integer"
         resultType = "com. qfedu. pojo. StuCard">
        select * from stu_card where cid = #{cid}
    </select >
</mapper >
```

**【例 3-4】 StuMapper. xml**

```
<?xml version = "1.0" encoding = "UTF - 8" ?>
<! DOCTYPE mapper
PUBLIC " - //mybatis. org//DTD Mapper 3.0//EN"
"http://mybatis. org/dtd/mybatis - 3 - mapper. dtd">
< mapper namespace = "stu">
    < select id = "findStudentBySid" parameterType = "Integer"
         resultMap = "stuResultsMap">
        select s. * , c. balance from stu  s, stu_card c where s. cardid = c. cid
              and s. sid = #{sid}
    </select >
    < resultMap type = "stu" id = "stuResultsMap">
        < id column = "sid" property = "sid"/>
        < result column = "sname" property = "sname"/>
        < result column = "age" property = "age"/>
        < result column = "course" property = "course"/>
        < association property = "sc" javaType = "StuCard">
            < id column = "cid" property = "cid"/>
            < result column = "balance" property = "balance"/>
        </association >
    </resultMap >
</mapper >
```

在以上代码中，< select >元素用于映射查询，< resultMap >元素用于结果集处理，在 < resultMap >元素的子元素中，< id >元素用于映射主键，< result >元素用于映射其他字段，< association >元素用于映射 stu 表和 stu_card 表的关联关系。

**5. 创建配置文件**

在工程 chapter03 的 src 目录下创建配置文件 mybatis-config. xml，如例 3-5 所示。

【例 3-5】 mybatis-config.xml

```
<?xml version = "1.0" encoding = "UTF - 8" ?>
<! DOCTYPE configuration
PUBLIC " - //mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis - 3 - config.dtd">
< configuration >
    <! -- 配置别名  -- >
    < typeAliases >
        < package name = "com.qfedu.pojo"/>
    </typeAliases >
    <! -- 配置环境  -- >
    < environments default = "mysql">
        < environment id = "mysql">
            < transactionManager type = "JDBC"/>
            < dataSource type = "POOLED">
            < property name = "driver" value = "com.mysql.jdbc.Driver"/>
            < property name = "url"
                value = "jdbc:mysql://localhost:3306/chapter03"/>
            < property name = "username" value = "root"/>
            < property name = "password" value = "root"/>
            </dataSource >
        </environment >
    </environments >
    <! -- 引入映射文件 -- >
    < mappers >
        < mapper resource = "com/qfedu/mapper/StuMapper.xml"/>
        < mapper resource = "com/qfedu/mapper/StuCardMapper.xml"/>
    </mappers >
</configuration >
```

在以上配置文件中，< typeAliases >用于指定别名，本次采用的是扫描包的方法；< mappers >元素用于引入映射文件。

**6. 编写测试类**

在工程 chapter03 的 src 目录下创建 com.qfedu.test 包，在 com.qfedu.test 包下创建测试类 TestOneToOne，如例 3-6 所示。

【例 3-6】 TestOneToOne.java

```
package com.qfedu.test;
import java.io.InputStream;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session. * ;
import com.qfedu.pojo.Stu;
public class TestOneToOne {
    public static void main(String[ ] args) throws Exception {
        InputStream in =
            Resources.getResourceAsStream("mybatis - config.xml");
        SqlSessionFactory sessionFactory = new
```

```
11              SqlSessionFactoryBuilder().build(in);
12          SqlSession session = sessionFactory.openSession();
13          //使用 MyBatis 查询结果 sid 为 1 的学生信息(包括学生的校园卡信息)
14          Stu stu = session.selectOne("stu.findStudentBySid",1);
15          System.out.println(stu);
16          session.close();
17      }
18 }
```

在以上代码中,先通过 SqlSession 对象获取查询结果,此结果包含有处理表关系后的信息,然后将结果输出。

执行测试类 TestOneToOne,执行结果如图 3.4 所示。

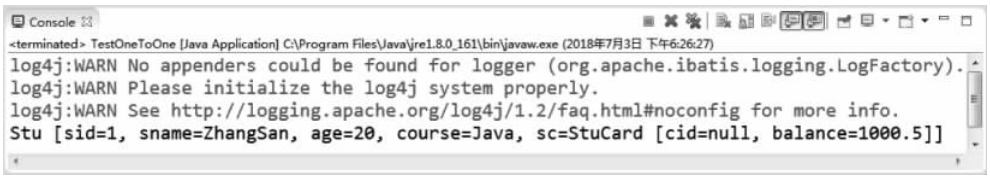



图 3.4　执行 TestOneToOne 类的结果

从图 3.4 可以看出,程序显示出包括校园卡在内的学生信息,由此可见,MyBatis 成功处理一对一关系映射。

## 3.3　一　对　多

相比于一对一,一对多的表关系在开发中更为常见,处理过程也略微复杂。通常情况下,如果使用 MyBatis 处理一对多的关系,需要在 MyBatis 的映射文件中添加< collection >元素。

< collection >元素是映射文件< resultMap >元素的子元素,其配置代码如下所示。

```
1  <resultMap type = "" id = "">
2          <id property = "" column = ""/>
3          <result property = "" column = ""/>
4          <collection property = "" column = "" javaType = "">
5              <id column = "" property = ""/>
6              <result column = "" property = "" />
7          </collection>
8  </resultMap>
```

在以上配置代码中,< id >元素用于映射主键,< column >元素用于映射普通字段,< collection >元素用于映射一对多的关系。在< collection >的属性中,property 属性用于指定 POJO 对象的成员属性,column 属性用于指定对应的字段。

前文中已经讲过,在处理表关系时,班级与学生是一对多的关联关系,接下来,本书将以此为例演示 MyBatis 实现一对多的关联映射。

**1. 数据准备**

(1) 在数据库 chapter03 中创建数据表 stu_class、数据表 stu_info,其中,数据表 stu_class 用于映射班级信息,数据表 stu_info 用于映射学生信息,SQL 语句如下所示。

```
USE chapter03;
CREATE TABLE stu_class(
cid INT PRIMARY KEY AUTO_INCREMENT, #ID
cname VARCHAR(20),        #班级名称
sum INT                   #总人数
);
CREATE TABLE stu_info(
sid INT PRIMARY KEY AUTO_INCREMENT, #ID
sname VARCHAR(20),        #学生姓名
age VARCHAR(20),          #学生年龄
course VARCHAR(20),       #科目
classid INT,              #班级 ID
FOREIGN KEY(classid) REFERENCES stu_class(cid)   #外键
);
```

(2) 向数据表 stu_class 中插入数据,SQL 语句如下所示。

```
INSERT INTO stu_class(cname,sum) VALUES ('Java1801',50);
INSERT INTO stu_class(cname,sum) VALUES ('Java1802',46);
```

(3) 向数据表 stu_info 中插入数据,SQL 语句如下所示。

```
INSERT INTO stu_info(sname,age,course,classid) VALUES
('ZhangSan','20','Java',1);
INSERT INTO stu_info(sname,age,course,classid) VALUES
('liSi','21','Java',2);
INSERT INTO stu_info(sname,age,course,classid) VALUES
('WangWu','20','Java',1);
INSERT INTO stu_info(sname,age,course,classid) VALUES
('ZhaoLiu','19','Java',1);
```

(4) 通过 SQL 语句测试数据是否插入成功,执行结果如下所示。

```
mysql> select * from stu_class;
+-----+-----------+------+
| cid | cname     | sum  |
+-----+-----------+------+
|   1 | Java1801  |   50 |
|   2 | Java1802  |   46 |
+-----+-----------+------+
2 rows in set (0.00 sec)

mysql> select * from stu_info;
```

```
+-----+----------+------+--------+---------+
| sid | sname    | age  | course | classid |
+-----+----------+------+--------+---------+
|   1 | ZhangSan |   20 | Java   |       1 |
|   2 | liSi     |   21 | Java   |       2 |
|   3 | WangWu   |   20 | Java   |       1 |
|   4 | ZhaoLiu  |   19 | Java   |       1 |
+-----+----------+------+--------+---------+
4 rows in set (0.00 sec)
```

从以上执行结果可以看出，数据插入成功。

**2. 创建 POJO 类**

在 src 目录下的 com.qfedu.pojo 包下分别创建两个 POJO 类：StuInfo 和 StuClass，如例 3-7 和例 3-8 所示。

**【例 3-7】 StuInfo.java**

```
1  package com.qfedu.pojo;
2  public class StuInfo {
3      private int sid;
4      private String sname;
5      private String age;
6      private String course;
7      public int getSid() {
8          return sid;
9      }
10     public void setSid(int sid) {
11         this.sid = sid;
12     }
13     public String getSname() {
14         return sname;
15     }
16     public void setSname(String sname) {
17         this.sname = sname;
18     }
19     public String getAge() {
20         return age;
21     }
22     public void setAge(String age) {
23         this.age = age;
24     }
25     public String getCourse() {
26         return course;
27     }
28     public void setCourse(String course) {
29         this.course = course;
30     }
31     @Override
32     public String toString() {
```

```
33        return "Stu [sid = " + sid + ", sname = " + sname + ", age = " + age + ","
34            + " course = " + course + "]";
35    }
36 }
```



**【例 3-8】 StuClass. java**

```
1  package com.qfedu.pojo;
2  import java.util.List;
3  public class StuClass {
4      private Integer cid;
5      private String cname;
6      private Integer sum;
7      private List<StuInfo> stuInfoList;
8      public Integer getCid() {
9          return cid;
10     }
11     public void setCid(Integer cid) {
12         this.cid = cid;
13     }
14     public String getCname() {
15         return cname;
16     }
17     public void setCname(String cname) {
18         this.cname = cname;
19     }
20     public Integer getSum() {
21         return sum;
22     }
23     public void setSum(Integer sum) {
24         this.sum = sum;
25     }
26     public List<StuInfo> getStuInfoList() {
27         return stuInfoList;
28     }
29     public void setStuInfoList(List<StuInfo> stuInfoList) {
30         this.stuInfoList = stuInfoList;
31     }
32 }
```

以上代码是两个 POJO 类，其中 StuInfo 类封装了学生信息，StuClass 类封装了班级信息。

**3. 创建映射文件**

在 com. qfedu. mappr 包下创建映射文件 StuClassMapper. xml，如例 3-9 所示。

**【例 3-9】 StuClassMapper. xml**

```
1  <?xml version = "1.0" encoding = "UTF - 8" ?>
2  <! DOCTYPE mapper
```

```
3  PUBLIC " - //mybatis.org//DTD Mapper 3.0//EN"
4  "http://mybatis.org/dtd/mybatis - 3 - mapper.dtd">
5  <mapper namespace = "stuClass">
6      <select id = "findStuClassByCid" parameterType = "Integer"
7           resultMap = "stuClassResultsMap">
8          select c. * ,s.sid,s.sname,s.age,s.course from stu_class c , stu_info
9               s where s.classid = c.cid and c.cid = #{cid}
10     </select>
11     <resultMap type = "stuClass" id = "stuClassResultsMap">
12         <id column = "cid" property = "cid"/>
13         <result column = "cname" property = "cname"/>
14         <result column = "sum" property = "sum"/>
15         <collection property = "stuInfoList" ofType = "stuInfo">
16             <id column = "sid" property = "sid"/>
17             <result column = "sname" property = "sname"/>
18             <result column = "age" property = "age"/>
19             <result column = "course" property = "course"/>
20         </collection>
21     </resultMap>
22 </mapper>
```

在以上代码中，< select >元素用于映射查询，< resultMap >元素用于结果集处理，在< resultMap >元素的子元素中，< id >元素用于映射主键，< result >元素用于映射其他字段，< collection >元素用于映射 stu_class 表和 stu_info 表的关联关系。

**4. 在配置文件中引入最新的映射文件**

打开 src 目录下的 mybatis-config. xml 文件，修改该文件的< mappers >元素，将映射文件 StuClassMapper. xml 引入到配置文件，修改后的代码如下所示。

```
1  <mappers>
2          <mapper resource = "com/qfedu/mapper/StuMapper.xml"/>
3          <mapper resource = "com/qfedu/mapper/StuCardMapper.xml"/>
4          <mapper resource = "com/qfedu/mapper/StuClassMapper.xml"/>
5  </mappers>
```

**5. 编写测试类**

在工程 chapter03 的 src 目录下创建 com. qfedu. test 包，在 com. qfedu. test 包下创建测试类 TestOneToMany，如例 3-10 所示。

**【例 3-10】 TestOneToMany. java**

```
1  package com.qfedu.test;
2  import java.io.InputStream;
3  import java.util.List;
4  import org.apache.ibatis.io.Resources;
5  import org.apache.ibatis.session. * ;
6  import com.qfedu.pojo. * ;
7  public class TestOneToMany {
```

```
8       public static void main(String[] args) throws Exception {
9           InputStream in =
10                Resources.getResourceAsStream("mybatis-config.xml");
11          SqlSessionFactory sessionFactory = new
12                SqlSessionFactoryBuilder().build(in);
13          SqlSession session = sessionFactory.openSession();
14          //使用 MyBatis 查询 cid 为 1 的班级信息(包括该班级的学生信息)
15          StuClass stuClass =
16                session.selectOne("stuClass.findStuClassByCid",1);
17          System.out.println("班级 ID: " + stuClass.getCid() + "\n 班级名称: " +
18              stuClass.getCname() + "\n 班级人数: " + stuClass.getSum() + "\n 学生信息: ");
19          List<StuInfo> stuInfoList = stuClass.getStuInfoList();
20          for (StuInfo stuInfo : stuInfoList) {
21              System.out.println(stuInfo);
22          }
23          session.close();
24      }
25  }
```

在以上代码中,先通过 SqlSession 对象获取查询结果,此结果包含有处理表关系后的信息,然后将结果输出。

执行测试类 TestOneToMany,执行结果如图 3.5 所示。




图 3.5　执行 TestOneToMany 类的结果

从图 3.5 中可以看出,程序显示出包括学生信息在内的班级信息,由此可见,MyBatis 成功实现一对多关系映射。

## 3.4　多　对　多

通常情况下,多对多表关系要转化为一对多的形式进行处理,这种转化是通过一张中间表来实现的。在使用 MyBatis 处理多对多关系时,需要先将多对多关系转化为一对多关系,然后使用< collection >元素完成映射。

3.3 节中已经讲过,在处理表关系时,学生与教师是多对多的关联关系,接下来,本节将以此为例演示 MyBatis 实现多对多的关联映射。

**1. 数据准备**

(1) 在数据库 chapter03 中创建数据表 class_info、数据表 teach_info,其中,数据表 class_info 用于映射班级信息,数据表 teach_info 用于映射教师信息,SQL 语句如下所示。

```
1  USE chapter03;
2  CREATE TABLE class_info(
3  cid INT PRIMARY KEY AUTO_INCREMENT, #ID
4  cname VARCHAR(20),      #班级名称
5  sum INT                 #总人数
6  );
7  CREATE TABLE teach_info(
8  tid INT PRIMARY KEY AUTO_INCREMENT, #ID
9  tname VARCHAR(20),      #教师姓名
10 age VARCHAR(20),        #教师年龄
11 course VARCHAR(20)      #课程
12 );
```

(2) 向数据表 class_info 中插入数据,SQL 语句如下所示。

```
1  INSERT INTO class_info(cname,sum) VALUES ('Java1801',50);
2  INSERT INTO class_info(cname,sum) VALUES ('Java1802',46);
```

(3) 向数据表 teach_info 中插入数据,SQL 语句如下所示。

```
1  INSERT INTO teach_info(tname,age,course) VALUES('ZhangSan','31','JavaSE');
2  INSERT INTO teach_info(tname,age,course) VALUES('LiSi','33','JavaWeb');
```

(4) 建立一张中间表 class_teach 并向表中插入数据,SQL 语句如下所示。

```
1  CREATE TABLE class_teach(
2  id INT PRIMARY KEY AUTO_INCREMENT, #ID
3  class_id  INT, #班级 ID
4  teach_id  INT, #教师 ID
5  FOREIGN KEY(class_id) REFERENCES class_info(cid),   #外键
6  FOREIGN KEY(teach_id) REFERENCES teach_info(tid)    #外键
7  );
8  INSERT INTO class_teach(class_id,teach_id)VALUES(1,1);
9  INSERT INTO class_teach(class_id,teach_id)VALUES(2,1);
10 INSERT INTO class_teach(class_id,teach_id)VALUES(1,2);
```

(5) 通过 SQL 语句测试数据是否插入成功,执行结果如下所示。

```
mysql> select * from class_info;
+-----+----------+------+
| cid | cname    | sum  |
+-----+----------+------+
|   1 | Java1801 |   50 |
|   2 | Java1802 |   46 |
+-----+----------+------+
2 rows in set (0.00 sec)

mysql> select * from teach_info;
```

```
+-----+----------+------+---------+
| tid | tname    | age  | course  |
+-----+----------+------+---------+
|  1  | ZhangSan |  31  | JavaSE  |
|  2  | LiSi     |  33  | JavaWeb |
+-----+----------+------+---------+
2 rows in set (0.00 sec)

mysql> select * from class_teach;
+----+----------+----------+
| id | class_id | teach_id |
+----+----------+----------+
|  1 |        1 |        1 |
|  2 |        2 |        1 |
|  3 |        1 |        2 |
+----+----------+----------+
3 rows in set (0.00 sec)
```

从以上执行结果可以看出,数据插入成功。

**2. 创建 POJO 类**

在 src 目录的 com.qfedu.pojo 包下分别创建两个 POJO 类：TeachInfo 和 ClassInfo，如例 3-11 和例 3-12 所示。

**【例 3-11】 TeachInfo.java**

```
1  package com.qfedu.pojo;
2  import java.util.List;
3  public class TeachInfo {
4      private Integer tid;
5      private String tname;
6      private String age;
7      private String course;
8      private List<ClassInfo> classInfoList;
9      public Integer getTid() {
10         return tid;
11     }
12     public void setTid(Integer tid) {
13         this.tid = tid;
14     }
15     public String getTname() {
16         return tname;
17     }
18     public void setTname(String tname) {
19         this.tname = tname;
20     }
21     public String getAge() {
22         return age;
23     }
24     public void setAge(String age) {
```

```
25          this.age = age;
26      }
27      public String getCourse() {
28          return course;
29      }
30      public void setCourse(String course) {
31          this.course = course;
32      }
33      public List<ClassInfo> getClassInfoList() {
34          return classInfoList;
35      }
36      public void setClassInfoList(List<ClassInfo> classInfoList) {
37          this.classInfoList = classInfoList;
38      }
39      @Override
40      public String toString() {
41          return "TeachInfo [tid=" + tid + ", tname=" + tname + ", age=" + age
42           + ", course=" + course + ", classInfoList=" + classInfoList + "]";
43      }
44  }
```

**【例 3-12】 ClassInfo. java**

```
1   package com.qfedu.pojo;
2   import java.util.List;
3   public class ClassInfo {
4       private Integer cid;
5       private String cname;
6       private Integer sum;
7       private List<TeachInfo> teachInfoList;
8       public Integer getCid() {
9           return cid;
10      }
11      public void setCid(Integer cid) {
12          this.cid = cid;
13      }
14      public String getCname() {
15          return cname;
16      }
17      public void setCname(String cname) {
18          this.cname = cname;
19      }
20      public Integer getSum() {
21          return sum;
22      }
23      public void setSum(Integer sum) {
24          this.sum = sum;
25      }
26      public List<TeachInfo> getTeachInfoList() {
```

```
27          return teachInfoList;
28      }
29      public void setTeachInfoList(List<TeachInfo> teachInfoList) {
30          this.teachInfoList = teachInfoList;
31      }
32      @Override
33      public String toString() {
34          return "ClassInfo [cid=" + cid + ", cname=" + cname + ", sum=" + sum
35                  + ", teachInfoList=" + teachInfoList + "]";
36      }
37  }
```

以上代码是两个 POJO 类，其中 ClassInfo 类封装了班级信息，TeachInfo 类封装了教师信息，除了 getter 和 setter 方法之外，为方便输出信息，这两个类还提供了 toString()方法。

**3. 创建映射文件**

在 com.qfedu.mappr 包下创建映射文件 ClassInfoMapper.xml，如例 3-13 所示。

**【例 3-13】 ClassInfoMapper.xml**

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="classInfo">
6      <select id="findClassInfoByCid" parameterType="Integer"
7           resultMap="classInfoResultsMap">
8          select c.*,t.* from class_info  c ,teach_info t, class_teach ct where
9              c.cid=ct.class_id
10         and ct.teach_id = t.tid and c.cid = #{cid}
11     </select>
12     <resultMap type="classInfo" id="classInfoResultsMap">
13         <id column="cid" property="cid"/>
14         <result column="cname" property="cname"/>
15         <result column="sum" property="sum"/>
16         <collection property="teachInfoList" ofType="teachInfo">
17             <id column="tid" property="tid"/>
18             <result column="tname" property="tname"/>
19             <result column="age" property="age" />
20             <result column="course"  property="course"/>
21         </collection>
22     </resultMap>
23 </mapper>
```

在以上代码中，<select>元素用于映射查询，<resultMap>元素用于结果集处理，在<resultMap>元素的子元素中，<id>元素用于映射主键，<result>元素用于映射其他字段，<collection>元素用于映射 class_info 表和 teach_info 表的关联关系。

**4. 在配置文件中引入最新的映射文件**

打开 src 目录下的 mybatis-config.xml 文件，修改该文件的<mappers>元素，将映射文

件 ClassInfoMapper.xml 引入到配置文件，修改后的代码如下所示。

```
<mappers>
        <mapper resource="com/qfedu/mapper/StuMapper.xml"/>
        <mapper resource="com/qfedu/mapper/StuCardMapper.xml"/>
        <mapper resource="com/qfedu/mapper/StuClassMapper.xml"/>
        <mapper resource="com/qfedu/mapper/ClassInfoMapper.xml"/>
    </mappers>
```

**5. 编写测试类**

在 com.qfedu.test 包下创建测试类 TestManyToMany，如例 3-14 所示。

**【例 3-14】 TestManyToMany.java**

```
package com.qfedu.test;
import java.io.InputStream;
import java.util.List;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.*;
import com.qfedu.pojo.*;
public class TestManyToMany {
    public static void main(String[] args) throws Exception {
        InputStream in =
            Resources.getResourceAsStream("mybatis-config.xml");
        SqlSessionFactory sessionFactory = new
            SqlSessionFactoryBuilder().build(in);
        SqlSession session = sessionFactory.openSession();
        //使用 MyBatis 查询 cid 为 1 的班级信息(包括该班级的教师信息)
        ClassInfo  classInfo =
            session.selectOne("classInfo.findClassInfoByCid",1);
        System.out.println("班级 ID: " + classInfo.getCid() + "\n 班级名称: " +
       classInfo.getCname() + "\n 班级人数: " + classInfo.getSum() + "\n 教师信息: ");
        List<TeachInfo> teachInfoList = classInfo.getTeachInfoList();
        for (TeachInfo teachInfo : teachInfoList) {
            System.out.println(teachInfo.toString());
        }
        session.close();
    }
}
```

在以上代码中，先通过 SqlSession 对象获取查询结果，此结果包含有处理表关系后的信息，然后将结果输出。

执行测试类 TestManyToMany，执行结果如图 3.6 所示。

从图 3.6 中可以看出，程序显示出包括学生信息在内的班级信息，由此可见，MyBatis 成功实现关系映射。




图 3.6　执行 TestManyToMany 类的结果

## 3.5　主 键 映 射

数据表的主键用于标识该表中每条记录的唯一性，使用 MyBatis 操作数据库应考虑到多表关联下的主键映射问题。

在实际开发中，当往数据表中插入数据时，如果对于插入数据的主键没有特殊要求，那么可以采用不返回主键值的方式配置插入语句，这样能够避免额外的 SQL 开销。但是，实际开发中常遇到一些多表关联下的操作，例如，在一次操作中插入具有一对多关系的表数据，当插入多方的数据时，需要获取刚刚插入的一方的主键值，此时就要采用插入后获取主键值的方式配置。

获取主键值可以通过配置映射文件中的< insert >元素完成，< insert >元素的 useGeneratedKeys 属性用于获取数据库内部产生的主键值，keyProperty 属性用于指定主键，

此处需要注意的是，如果操作的数据库不支持主键自增功能时，要使用< insert >元素的 < selectKey >子元素获取主键值。

接下来，本节以一对多条件下的数据表 stu_class 为例演示主键映射。

**1. 修改映射文件**

修改映射文件 StuClassMapper. xml，添加< insert >元素，具体代码如下所示。

```
1  < insert id = "addStuClass01" parameterType = "stuClass"
2  useGeneratedKeys = "true"   keyProperty = "cid">
3          insert into stu_class(cname,sum) values ( #{cname}, #{sum})
4      </insert >
```

在以上代码中，useGeneratedKeys 属性用于获取数据库内部产生的主键值，keyProperty 属性用于指定主键。

**2. 编写测试类**

在 com. qfedu. test 包下创建测试类 TestAddStuClass，如例 3-15 所示。

**【例 3-15】　TestAddStuClass. java**

```
1  package com.qfedu.test;
2  import java.io.InputStream;
3  import org.apache.ibatis.io.Resources;
```

```
4  import org.apache.ibatis.session.*;
5  import com.qfedu.pojo.StuClass;
6  public class TestAddStuClass {
7      public static void main(String[] args) throws Exception {
8          InputStream in =
9              Resources.getResourceAsStream("mybatis-config.xml");
10         SqlSessionFactory sessionFactory = new
11             SqlSessionFactoryBuilder().build(in);
12         SqlSession session = sessionFactory.openSession();
13         StuClass stuClass = new StuClass();
14         stuClass.setCname("Java1803");
15         stuClass.setSum(50);
16         //使用 MyBatis 插入一条班级信息
17         int result = session.insert("stuClass.addStuClass01",stuClass );
18         if (result>0) {
19             System.out.println("成功插入" + result + "条数据");
20             System.out.println("插入数据的主键 cid 为:" + stuClass.getCid());
21         }else{
22             System.out.println("插入操作失败");
23         }
24         //提交事务
25         session.commit();
26         //关闭 SqlSession
27         session.close();
28     }
29 }
```

在以上代码中,程序向数据库中插入一条班级信息,由于映射文件中配置了< insert >元素的 useGeneratedKeys 属性,程序可直接通过 stuClass 对象的 getCid()方法获取新添加记录的主键 cid 值。

执行测试类 TestAddStuClass,执行结果如图 3.7 所示。




图 3.7　执行 TestAddStuClass 类的结果

从图 3.7 中可以看出,控制台输出成功插入数据的提示信息以及新添加记录的主键 cid 值,由此可见,MyBatis 成功实现多表关联场景下的主键映射。

除此之外,还可使用< insert >元素的< selectKey >子元素获取主键值,这种方式不但适用于数据库支持主键自增的场景,而且适用于数据库不支持主键自增的场景,接下来通过实例具体演示< selectKey >子元素的使用。

**1. 修改映射文件**

修改映射文件 StuClassMapper.xml,添加< insert >元素,具体代码如下所示。

```
1  <insert id="addStuClass02" parameterType="stuClass">
2          insert into stu_class(cname,sum) values (#{cname},#{sum})
3          <selectKey keyColumn="cid" keyProperty="cid" resultType="Integer"
4                order="AFTER">
5              select LAST_INSERT_ID()
6          </selectKey>
7      </insert>
```

在以上代码中，<selectKey>元素的 keyProperty 属性用于指定数据表主键，keyColumn 属性用于指定 POJO 对象的成员属性，resultType 属性用于指定返回值类型，order 属性的设定和使用的数据库有关，如果使用 MySQL 数据库，则 order 属性值是 AFTER。

**2. 修改测试类**

修改测试类 TestAddStuClass，将其第 17 行代码修改为如下所示。

```
int result = session.insert("stuClass.addStuClass02",stuClass);
```

执行测试类 TestAddStuClass，执行结果如图 3.8 所示。




图 3.8　重新执行 TestAddStuClass 类的结果

从图 3.8 中可以看出，控制台输出成功插入数据的提示信息以及新添加记录的主键 cid 值，由此可见，通过<selectKey>也可实现返回主键值。

## 3.6　本章小结

本章首先介绍了表与表之间的关系，其中包括一对一、一对多、多对多等；然后分别讲解了如何使用 MyBatis 实现不同表关系的关联映射，最后详细讲解了如何使用 MyBatis 实现主键映射。通过本章知识的学习，大家应该理解表与表之间的关联关系，掌握处理表关系的思路和方法，能够使用 MyBatis 框架完成关联状态下的多表操作。

## 3.7　习　　题

**1. 填空题**

(1) 表与表之间的关系包括_____、_____、_____等。

(2) 在一对一表关系中，每张数据表的关键字在对应的关系表中至多存在_____条记录。

(3) <association>元素的_____属性用于指定数据表中对应字段的类型。

(4) < collection >元素的_____子元素用于映射代表多方的数据表的主键。

(5) 在处理多对多表关系时,通常要将其转化成_____的形式进行处理。

**2. 选择题**

(1) 关于数据表的关联关系,下列选项错误的是(　　)。

A. MyBatis 支持一对一的表关系映射

B. 班级和学生的关系属于一对多的关系

C. 在一对多表关系中,主键数据表中的一个记录可与其关系表中的多个记录相关

D. MyBatis 无法支持多对多的表关系映射

(2) 在< association >元素的属性中,负责指定是否自动映射的是(　　)。

A. fetchType　　B. select

C. autoMapping　　D. typeHandler

(3) 在 MyBatis 的映射文件中,用于映射一对一关系的元素是(　　)。

A. < association >　　B. < collection >

C. < typeAlias >　　D. < discriminator >

(4) 在 MyBatis 的映射文件中,用于映射一对多关系的元素是(　　)。

A. < association >　　B. < collection >

C. < typeAlias >　　D. < discriminator >

(5) 在< selectKey >元素的属性中,用于指定数据表主键字段的是(　　)。

A. keyColumn　　B. keyProperty

C. resultType　　D. order

**3. 思考题**



(1) 简述一对多表关系的处理过程。

(2) 简述多对多表关系的处理过程。

**4. 编程题**

创建一张订单表,创建一张商品表,通过 MyBatis 查询 id 为 1 的订单表中的所有商品信息,注意多表关联。