第3章 MyBatis进阶

学习目标


视频讲解

- 掌握MyBatis的映射文件的使用方法和规范，能够编写映射文件。
- 掌握MyBatis一对一关系的映射方法，能够完成一对一应用程序。
- 掌握MyBatis一对多关系的映射方法，能够完成一对多应用程序。
- 掌握MyBatis多对多关系的映射方法，能够完成多对多应用程序。

为了灵活运用MyBatis框架，开发人员需要掌握MyBatis映射文件的结构及使用方法。通过深入理解映射文件的组成和重要参数，开发人员能够根据项目需求对映射文件进行灵活配置，以达到最佳的性能和扩展性。本章将对MyBatis的映射文件和关联映射的内容进行讲解，同时通过一个实战演练——智慧农业果蔬系统普通用户的数据管理项目巩固Mapper和XML映射文件的相关知识。

3.1 映射文件

在2.4.8节中提到，MyBatis配置文件中的<mappers>元素用于引入映射文件。这些映射文件是MyBatis框架的核心，定义了SQL语句和数据库中的数据如何映射到Java对象，是MyBatis的“翻译器”，将数据库信息翻译成Java对象，使开发者能够更轻松地与数据库交互，而不必过分关注数据库连接和底层细节，从而大幅度降低编码的工作量。本节将对MyBatis的映射文件结构及元素用法进行讲解。

3.1.1 映射文件结构

通过映射文件，我们能够将数据从应用程序的POJO对象映射到持久化存储中，实现数据的创建、读取、更新和删除操作。映射文件以XML文件的形式存在，一般采用“POJO类的名称＋Mapper”的规则进行命名，例如EducationMapper.xml。MyBatis规定了其映射文件的层次结构，具体如下所示。

```
<?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 = "">
    <!-- 参数映射 -->
    < parameterMap id = "" type = "">
        < parameter property = "" jdbcType = ""/>
        < parameter property = "" jdbcType = ""/>
    </parameterMap >
```

```
    <!-- 返回值映射 -->
    <resultMap id="" type="">
        <id property="" column="" jdbcType=""/>
        <result property="" column="" jdbcType=""/>
        <result property="" column="" jdbcType=""/>
    </resultMap>
    <!-- SQL语句 -->
    <select id="" resultMap="">
    </select>
    <insert id="" parameterMap="">
    </insert>
    <update id="" parameterMap="">
    </update>
    <delete id="" parameterType="">
    </delete>
</mapper>
```

上述代码中列出了 MyBatis 映射文件的常见元素，例如<mapper>、<select>、<insert>、<update>和<delete>等。本书将带领读者学习 MyBatis 中常用元素的功能语法和使用方法。

3.1.2 <mapper>元素

<mapper>元素是 MyBatis 映射文件的根元素，它用于定义数据库操作的 SQL 语句以及数据库记录与 Java 对象的映射规则。整个映射文件的内容都必须包含在这个元素内部，示例配置如下。

```
<mapper namespace="com.qfedu.EducationMapper">
   <!-- 这里包含了SQL语句和映射关系的定义 -->
</mapper>
```

上述代码中，namespace 属性是<mapper>元素的必需属性，它用于定义映射文件的命名空间。通常，这个命名空间会关联到一个对应的数据访问层的 Java 接口，该接口定义了映射文件中的 SQL 操作方法，使得数据库操作可以通过接口方法进行调用。映射文件的命名空间应该与其关联的接口文件的包名和类名一致，以便正确关联映射文件和接口文件，此处为 com.qfedu.EducationMapper 接口。

<mapper>元素内部可以包含 SQL 语句的定义，包括查询、插入、更新和删除等操作。SQL 语句的定义通常使用<select>、<insert>、<update>和<delete>等元素来实现。此外，<mapper>元素还用于定义如何将数据库记录映射到 Java 对象的规则，通常通过<resultMap>元素来实现。后续将对上述元素进行详细讲解。

<mapper>元素还可以通过 resource、url、class 等属性引入其他映射文件，这有助于将映射配置分解成多个文件，使得映射文件更易于维护和组织。具体使用规则可参考 2.4.8 节。

3.1.3 <select>元素

<select>元素是 MyBatis 中常用的元素之一，主要用于定义数据库查询操作，它包含了 SQL 语句、参数映射、结果映射以及其他与数据库操作相关的细节。

为了更加灵活地映射查询语句，<select>元素中提供了一些属性，如表 3-1 所示。

表 3-1 <select>元素属性

属性名称	说明
id	必需属性,用于给查询 SQL 定义一个唯一的标识符
parameterType	指定传递给查询 SQL 的参数类型,默认为 unset
resultType	指定查询结果的返回类型,通常是一个 Java 类。不能与 resultMap 同时使用
resultMap	指定一个映射结果集的规则,不能与 resultType 同时使用
flushCache	控制是否刷新缓存,默认值为 false
useCache	控制是否使用二级缓存,默认值为 true
timeout	查询的超时时间,以秒为单位
fetchSize	指定数据库游标的数量
statementType	指定用于查询的 PreparedStatement 类型,可选 STATEMENT、PREPARED 或 CALLABLE,通常为 PREPARED,默认为 unset
resultSetType	指定返回的结果集类型,可选 FORWARD_ONLY、SCROLL_SENSITIVE、SCROLL_INSENSITIVE 或 DEFAULT,通常为 FORWARD_ONLY,默认为 unset
resultOrdered	控制多列结果集的顺序,通常为 false
resultSets	指定存储过程执行的多个结果集

表 3-1 列举出了<select>元素中的属性,每个属性都有其独特的作用,开发人员根据查询的要求选择适当的属性来定制查询操作。

接下来,通过<select>元素定义一条查询表 education 中所有记录的 SQL 语句,具体步骤如下。

(1) 在 IDEA 中新建 Web 项目 chapter03,并完成 MyBatis 框架的集成和数据库配置。本案例使用第 2 章的数据表 education。

(2) 在项目的 src 目录下创建 com.qfedu.pojo 包,在该包中新建类 Education,具体代码参考第 2 章的例 2-1。

(3) 在项目的 src 目录下创建包 com.qfedu.mapper,在该包中新建接口 EducationMapper,作为数据库访问层接口,具体代码如例 3-1 所示。

例 3-1 EducationMapper.java。

```
1  package com.qfedu.mapper;
2  import com.qfedu.pojo.Education;
3  import java.util.List;
4  public interface EducationMapper {
5      List<Education> findAllEducation();
6  }
```

(4) 在 com.qfedu.mapper 包中新建名为 EducationMapper 的 XML 文件,映射配置如例 3-2 所示。

例 3-2 EducationMapper.xml。

```
1  <mapper namespace="com.qfedu.mapper.EducationMapper">
2    <select id="findAllEducation" resultType="com.qfedu.pojo.Education">
3      select * from education
4    </select>
5  </mapper>
```

上述配置定义了一个名为 findAllEducation 的 MyBatis 查询操作,它的功能是从名为

education 的数据库表中检索所有记录，并将结果映射为 com. qfedu. pojo. Education 类型的 Java 对象列表。< select >元素的 id 属性的值与命名空间关联的接口文件中的方法名一致，此处为 com. qfedu. mapper. EducationMapper 接口中的 findAllEducation( )方法；resultType 属性用于指定返回结果的映射类型，此处为 com. qfedu. Education。

(5) 在项目的 src 目录下创建包 com. qfedu. test，在该包中新建类 TestEducationMapper，通过调用 EducationMapper 接口的方法，执行映射文件中的 SQL 操作。首先，在 TestEducationMapper 类中定义一个静态方法 testFindAllEducation()，然后，在 main()方法中调用该方法，具体代码如例 3-3 所示。

例 3-3 TestEducationMapper. java。

```
1  package com.qfedu.test;
2  //此处省略导包的代码
3  public class TestEducationMapper {
4      public static void main(String[] args) {
5          testFindAllEducation();
6      }
7      public static void testFindAllEducation(){
8          InputStream inputStream = null;
9          try {
10             inputStream =
11                     Resources.getResourceAsStream("mybatis-config.xml");
12         } catch (IOException e) {
13             e.printStackTrace();
14         }
15         SqlSessionFactory build =
16                 new SqlSessionFactoryBuilder().build(inputStream);
17         SqlSession sqlSession = build.openSession();
18         EducationMapper mapper = sqlSession.getMapper(
19                                                 EducationMapper.class);
20         List<Education> educations = mapper.findAllEducation();
21         for (Education education : educations) {
22             System.out.println(education.toString());
23         }
24         sqlSession.close();
25   }
26 }
```

(6) 执行例 3-3 中的 main()方法，对< select >元素定义的数据库查询操作进行测试，结果如图 3-1 所示。

从图 3-1 中可以看出，通过映射文件中的< mapper >元素成功将映射文件 EducationMapper. xml 与数据库访问接口 EducationMapper 建立了关联，并通过< select >元素成功执行查询表 education 中所有记录的 SQL 语句。

3.1.4 < insert >元素、< delete >元素、< update >元素

< insert >元素用于映射数据库插入操作的 SQL 语句，< update >元素用于映射数据库更新操作的 SQL 语句，而< delete >元素则用于映射数据库删除操作的 SQL 语句。与< select >元素相似，这 3 个元素在结构上都包含了 SQL 语句及参数类型等相关信息，接下

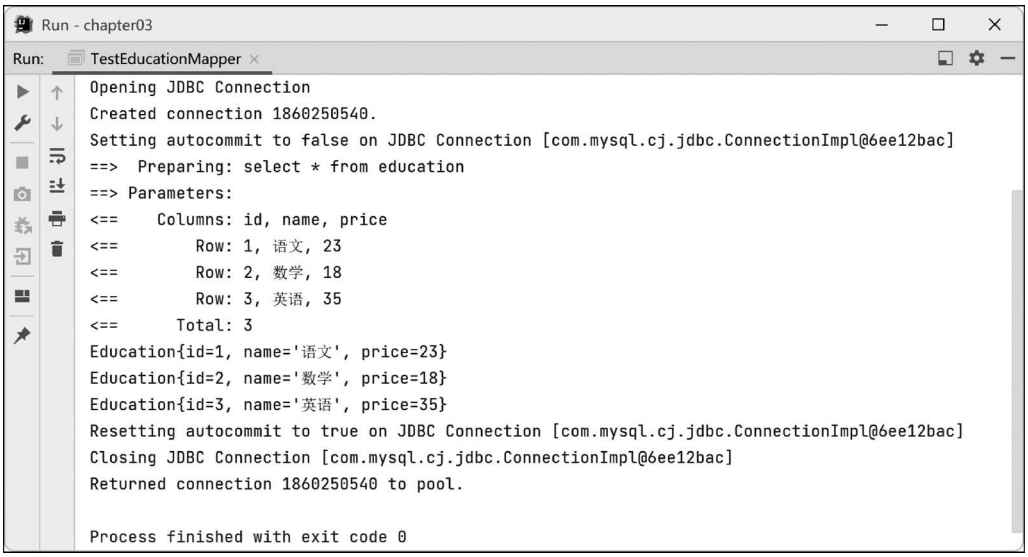

图 3-1　测试< select >元素查询操作的结果

来分别进行演示。

1. < insert >元素

通过< insert >元素定义一条向表 education 中插入数据的 SQL 语句,具体步骤如下。

(1) 在 EducationMapper. xml 文件中添加如下配置。

```
< insert id = "addEducation" parameterType = "com. qfedu. pojo. Education">
    insert into education(name,price) values( #{name}, #{price})
</insert >
```

上述配置定义了一个名为 addEducation 的插入操作,该操作使用参数占位符 #{},将 com. qfedu. pojo. Education 对象的 name 属性和 price 属性的值嵌入到 SQL 语句中,从而实现将它们的值分别插入数据库表 education 中的 name 和 price 列。开发人员在 Java 代码中需要使用 addEducation()方法执行插入操作,并通过传递 Education 对象指定要插入的数据。

(2) 在 EducationMapper 接口中声明方法 addEducation(),具体代码如下。

```
int addEducation(Education education);
```

(3) 在 TestEducationMapper 类中定义静态方法 testAddEducation(),具体代码如下。

```
public static void testAddEducation(){
    InputStream inputStream = null;
    try {
        inputStream =
                Resources. getResourceAsStream("mybatis - config. xml");
    } catch (IOException e) {
        e. printStackTrace();
    }
    SqlSessionFactory build =
                  new SqlSessionFactoryBuilder(). build(inputStream);
    SqlSession sqlSession = build. openSession();
```

```
        EducationMapper mapper = sqlSession.getMapper(EducationMapper.class);
        Education education = new Education("Java",100);
        //插入名称为 Java,价格为 100 的记录
        int num = mapper.addEducation(education);
        System.out.println(num);
        sqlSession.commit();                    //DML 操作需要提交事务
        sqlSession.close();
    }
```

(4) 在 TestEducationMapper 类的 main()方法中调用 testAddEducation()方法,并执行 main()方法,对< insert >元素定义的数据库插入操作进行测试,结果如下。

```
1
```

由测试结果可以看出,返回受影响的行数为 1,通过< insert >元素成功执行向表 education 中插入数据的 SQL 语句。此时,调用 testFindAllEducation()方法查询表 education 中的所有数据,结果如下。

```
Education{id = 1, name = '语文', price = 23}
Education{id = 2, name = '数学', price = 18}
Education{id = 3, name = '英语', price = 35}
Education{id = 6, name = 'Java', price = 100}
```

2. <delete>元素

通过< delete >元素定义一条删除表 education 中指定数据的 SQL 语句,具体步骤如下。

(1) 在 EducationMapper. xml 文件中添加如下配置。

```
<delete id = "deleteEducation" parameterType = "Integer">
   delete from education where id = #{id}
</delete>
```

上述配置定义了一个名为 deleteEducation 的删除操作,该操作将通过一个整数参数 id 来指定要删除的数据标识。通过在 Java 代码中调用 deleteEducation()方法和传递合适的参数,可以执行表 education 中的数据删除操作。

(2) 在 EducationMapper 接口中声明方法 deleteEducation(),具体代码如下。

```
int deleteEducation(int id);
```

(3) 在 TestEducationMapper 类中定义静态方法 testDeleteEducation(),关键代码如下。

```
EducationMapper mapper = sqlSession.getMapper(EducationMapper.class);
int num = mapper.deleteEducation(1);              //删除 id 为 1 的记录
System.out.println(num);
```

(4) 在 TestEducationMapper 类的 main()方法中调用 testdeleteEducation()方法,并执行 main()方法,对< delete >元素定义的数据库删除操作进行测试,结果如下。

```
1
```

由测试结果可以看出,返回受影响的行数为 1,通过< delete >元素成功执行删除表 education 中数据的 SQL 语句。此时,调用 testFindAllEducation()方法查询表 education

中的所有数据,结果如下。

```
Education{id=2, name='数学', price=18}
Education{id=3, name='英语', price=35}
Education{id=6, name='Java', price=100}
```


3. <update>元素

通过<update>元素定义一条更新表 education 中指定数据的 SQL 语句,具体步骤如下。

(1) 在 EducationMapper.xml 文件中添加如下配置。

```
<update id="updateEducation" parameterType="com.qfedu.pojo.Education">
    update education set price = #{price} where id = #{id}
</update>
```

这个配置定义了一个名为 updateEducation 的更新操作,该操作将通过 com.qfedu.pojo.Education 对象的 id 属性和 price 属性的值来指定要更新的数据的条件和新值。通过在 Java 代码中调用 updateEducation()方法和传递合适的参数,可以执行表 education 中的数据更新操作。

(2) 在 EducationMapper 接口中声明方法 updateEducation(),具体代码如下。

```
int updateEducation(int id);
```

(3) 在 TestEducationMapper 类中定义静态方法 testUpdateEducation(),关键代码如下。

```
EducationMapper mapper = sqlSession.getMapper(EducationMapper.class);
//将 id 为 2 的记录的 price 值更新为 111
int num = mapper.updateEducation(2,111);
System.out.println(num);
```

(4) 在 TestEducationMapper 类的 main()方法中调用 testUpdateEducation()方法,并执行 main()方法,对<update>元素定义的更新数据库的操作进行测试,结果如下。

```
1
```

由测试结果可以看出,返回受影响的行数为 1,通过<update>元素成功执行更新表 education 中数据的 SQL 语句。此时,调用 testFindAllEducation()方法查询表 education 中的所有数据,结果如下。

```
Education{id=2, name='数学', price=111}
Education{id=3, name='英语', price=35}
Education{id=6, name='Java', price=100}
```

3.1.5 <resultMap>元素

<resultMap>元素是 MyBatis 映射文件中用于定义结果集映射关系的重要元素。通过使用<resultMap>元素,可以灵活地配置映射规则,使得查询结果的列能够正确地映射到 Java 对象的属性,从而处理更加复杂的映射情况,例如 3.2 节中讲解的一对多关联映射和多对多关联映射。MyBatis 映射文件中<resultMap>元素的完整结构代码如下所示。

```
<resultMap id="resultMapId" type="com.example.Person">
  <!-- 主键映射,使用 <id> 元素 -->
  <id property="id" column="person_id" />
  <!-- 普通字段映射,使用 <result> 元素 -->
  <result property="name" column="person_name" />
  <result property="age" column="person_age" />
  <!-- 一对一关联映射,使用 <association> 元素 -->
  <association property="address" javaType="com.example.Address">
    <result property="street" column="street_name" />
    <result property="city" column="city_name" />
  </association>
  <!-- 一对多关联映射,使用 <collection> 元素 -->
  <collection property="phoneNumbers" ofType="com.example.PhoneNumber">
    <id property="id" column="phone_id" />
    <result property="number" column="phone_number" />
  </collection>
</resultMap>
```

上述代码的<resultMap>元素的子元素及子元素属性的说明如下所示。

- <resultMap>元素的 id 属性用于标志这个映射规则,通常是一个唯一的名称。
- <id>元素用于定义主键字段的映射规则,其中,property 属性指定了 Java 属性名,column 属性指定了数据库列名。
- <result>元素用于定义普通字段的映射规则,与<id>类似但适用于非主键字段。
- <association>元素用于定义一对一关联映射,其中 property 属性指定关联对象的 Java 属性,javaType 属性指定关联对象的 Java 类型。内部使用<result>元素定义关联对象的字段映射。3.2.1 节将使用案例进行演示。
- <collection>元素用于定义一对多关联映射,其中 property 属性指定集合属性,ofType 属性指定集合元素的类型。同样,内部使用<result>元素定义集合元素的字段映射。3.2.2 节将使用案例进行演示。

3.1.6 <sql>元素

<sql>元素用于定义可重用的 SQL 代码片段。在 MyBatis 应用程序开发中,经常需要编写多条 SQL 语句以满足各种业务需求,而这些 SQL 语句可能包含相同的代码段。为了提高代码的可维护性和重用性,可以使用<sql>元素将这些共享的代码片段提取出来并进行定义,从而避免了重复编写相同的代码,使代码更加整洁和易于管理。

通过<sql>元素定义代码片段,具体示例如下所示。

```
<sql id="educationCols">
   id,name,price
</sql>
```

上述代码中,id 属性用于指定该代码片段在命名空间的唯一标志,此处为 educationCols。当完成上述代码片段的定义后,可以在 MyBatis 映射文件中使用这个代码片段,以避免在多个地方重复编写相同的列名,示例代码如下。

```
<select id="selectEducation" resultType="com.qfedu.Education">
   select <include refid="educationCols"/> from education
```

```
</select>
<insert id = "insertEducation" parameterType = "com.qfedu.Education">
   insert into education(<include refid = "educationCols"/>) values(#{id},
#{name},#{price})
</insert>
```

上述代码中,<include>元素用于包含<sql>元素定义的SQL代码片段,其refid属性匹配<sql>元素的id属性。

3.2 关联映射

MyBatis的关联映射可以帮助开发人员在数据库中进行复杂的查询操作,避免手动拼接SQL语句。它可以将多个表之间的关系映射到Java对象之间的关系,从而简化了复杂的多表查询工作。关联映射支持三种主要类型的映射:一对一、一对多、多对多。本节将对MyBatis关联映射的功能和语法进行讲解。

3.2.1 一对一关联映射

1. 设计背景

一对一关联映射是指一个表中的一条记录对应着另一个表中的一条记录。在数据库模型中,这种关系通常表示两个实体之间的一对一关系。例如,一个人只有一个身份证号,一个身份证号也只对应一个人,这就是一对一关联。

在一个OA管理系统中,涉及两个实体:员工和工号。每个员工都会对应一个唯一的工号。在前面讲解的<resultMap>元素中包含了<association>子元素,MyBatis通过该元素来处理一对一关联关系。

2. 创建实体类和数据库

在chapter03项目的com.qfedu.pojo包下新建Employee类和Card类,分别表示员工类和工号类,具体代码如例3-4与例3-5所示。

例3-4 Employee.java。

```
1  public class Employee {
2      private int id;                          //主键
3      private String name;                     //姓名
4      private int cid;                         //工号表外键
5      private int did;                         //部门表外键
6      private Card card;                       //工号表实体类
7      //此处省略构造方法、Getter()、Setter()和toString()方法
8  }
```

例3-5 Card.java。

```
1  public class Card {
2      private int id;
3      private int number;
4      private String introduce;
5      //此处省略构造方法、Getter()、Setter()和toString()方法
6  }
```

(1) 在数据库 student 中创建员工表 employee,SQL 语句如下所示。

```
DROP TABLE IF EXISTS 'employee';
CREATE TABLE 'employee' (
  'id' int(0) NOT NULL AUTO_INCREMENT,
  'name' varchar(255) NULL DEFAULT NULL,
  'cid' int(0) NULL DEFAULT NULL,
  PRIMARY KEY ('id') USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci
ROW_FORMAT = Dynamic;
```

(2) 向 employee 表中插入数据,SQL 语句如下所示。

```
    INSERT INTO 'employee' VALUES (1, '余 * 兴', 1);
    INSERT INTO 'employee' VALUES (2, '谭 * 端', 2);
    INSERT INTO 'employee' VALUES (3, '宋 * 桥', 3);
```

(3) 在数据库 student 中创建工号表 card,SQL 语句如下所示。

```
DROP TABLE IF EXISTS 'card';
CREATE TABLE 'card' (
  'id' int(0) NOT NULL AUTO_INCREMENT,
  'number' int(0) NULL DEFAULT NULL,
  'introduce' varchar(255) NULL DEFAULT NULL,
  PRIMARY KEY ('id') USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci
  ROW_FORMAT = Dynamic;
```

(4) 向 card 表中插入数据,SQL 语句如下所示。

```
INSERT INTO 'card' VALUES (1, 77000, '组长');
INSERT INTO 'card' VALUES (2, 66789, '组员');
INSERT INTO 'card' VALUES (3, 22960, '组员');
```

将 card 表的 id 和 employee 表的 cid 进行关联,即每个员工记录在 employee 表中有一个 cid 字段,用于指示员工持有的工号。

3. 编写 MyBatis 的配置文件和映射文件

(1) 在 chapter03 项目下创建 resource 文件夹,将其标注为资源文件夹。在 resource 文件夹下新建 MyBatis 的配置文件,具体代码如例 3-6 所示。

例 3-6 mybatis-config.xml。

```
1  <?xml version = "1.0" encoding = "UTF - 8" ?>
2  <! DOCTYPE configuration PUBLIC " - //mybatis.org//DTD Config 3.0//EN"
3      "http://mybatis.org/dtd/mybatis - 3 - config.dtd">
4  < configuration >
5      <!-- 引入配置文件 -- >
6      < properties resource = "db.properties"/>
7      <!-- 设置 -- >
8      < settings >
9          <!-- 开启数据库日志检测 -- >
10         < setting name = "logImpl" value = "STDOUT_LOGGiNG"/>
11     </settings >
12     <!-- 包名简化缩写 -- >
13     < typeAliases >
14         < package name = "com.qfedu.pojo"/>
```

```
    </typeAliases>
    <!-- 配置环境 -->
    <environments default="dev">
        <!-- 配置 mysql 环境 -->
        <environment id="dev">
            <!-- 配置事务管理器 -->
            <transactionManager type="JDBC"/>
            <!-- 配置数据库连接 -->
            <dataSource type="POOLED">
                <!-- 配置数据库连接驱动 -->
                <property name="driver" value="${jdbc.myDriver}"/>
                <!-- 配置数据库连接地址 -->
                <property name="url" value="${jdbc.myUrl}"/>
                <!-- 配置用户名 -->
                <property name="username" value="${jdbc.myUsername}"/>
                <!-- 配置密码 -->
                <property name="password" value="${jdbc.myPassword}"/>
            </dataSource>
        </environment>
    </environments>
    <!-- 配置 mapper 映射文件 -->
    <mappers>
    <!-- 将 com.mapper 包下的所有 mapper 接口引入 -->
    <package name="com.qfedu.mapper" />
    </mappers>
</configuration>
```

在例 3-6 中,第 6 行代码表示引用外部配置文件 db.properties,其具体代码如例 3-7 所示。

例 3-7 db.properties。

```
jdbc.Driver = com.mysql.cj.jdbc.Driver
jdbc.Url = jdbc:mysql://localhost:3306/textbook
jdbc.Username = root
 jdbc.Password = root
```

需要注意的是,db.properties 文件必须存放于 resource 目录下,否则,mybatis-config.xml 文件无法读取 db.properties 中的内容。

(2) 在 chapter03 项目的 com.qfedu.mapper 包中新建 EmployeeMapper 接口,并在该接口中声明查询所有员工的方法,具体代码如例 3-8 所示。

例 3-8 EmployeeMapper.java。

```
public interface EmployeeMapper {
    List<Employee> findAllEmployee();
}
```

(3) 在 chapter03 项目的 com.qfedu.mapper 包中新建名为 EmployeeMapper 的 XML 文件,作为 EmployeeMapper 接口的映射文件,具体代码如例 3-9 所示。

例 3-9 EmployeeMapper.xml。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
```

```
3           "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4   <mapper namespace="com.qfedu.mapper.EmployeeMapper">
5       <resultMap id="" type="employee">
6           <id property="id" column="id"/>
7           <result property="name" column="name"/>
8           <result property="cid" column="cid"/>
9           <association property="card" javaType="card">
10              <id property="id" column="id"/>
11              <result property="number" column="number"/>
12              <result property="introduce" column="introduce"/>
13          </association>
14      </resultMap>
15      <select id="findAllEmployee" resultMap="employeeMap">
16          select * from employee e left join card c on e.cid = c.id
17      </select>
18  </mapper>
```

在例 3-9 中，定义了一个名为 employeeMap 的映射规则，用<association>元素将 employee 表的数据映射到 employee 对象和关联的 card 对象中。同时，本例中还定义了一个查询操作 findAllEmployee，用于查询员工信息，并在查询结果中包含了与员工关联的工号信息。

4. 编写测试类

（1）在 chapter03 项目的 com.qfedu.test 包中创建 TestFindAllEmployee 类，用于验证一对一关联映射，具体代码如例 3-10 所示。

例 3-10 TestFindAllEmployee.java。

```
1   public class TestFindAllEmployee {
2       public static void main(String[] args) {
3           /*创建输入流*/
4           InputStream inputStream = null;
5           /*将 MyBatis 配置文件转化为输入流*/
6           try {
7                 inputStream =
8                         Resources.getResourceAsStream("mybatis-config.xml");
9           } catch (IOException e) {
10              e.printStackTrace();
11          }
12          /*通过 SqlSessionFactoryBuilder()创建 SqlSessionFactory 对象*/
13          SqlSessionFactory build =
14                    new SqlSessionFactoryBuilder().build(inputStream);
15          /*通过 SqlSessionFactory 对象创建 SqlSession 对象*/
16          SqlSession sqlSession = build.openSession();
17          EmployeeMapper mapper =
18          sqlSession.getMapper(EmployeeMapper.class);
19          List<Employee> allEmployee = mapper.findAllEmployee();
20          for (Employee employee : allEmployee) {
21              System.out.println(employee.toString());
22          }
23          /*关闭事务*/
24          sqlSession.close();
25      }
26  }
```

(2) 执行 TestFindAllEmployee 类的 main()方法。测试一对一关联映射的结果如图 3-2 所示。

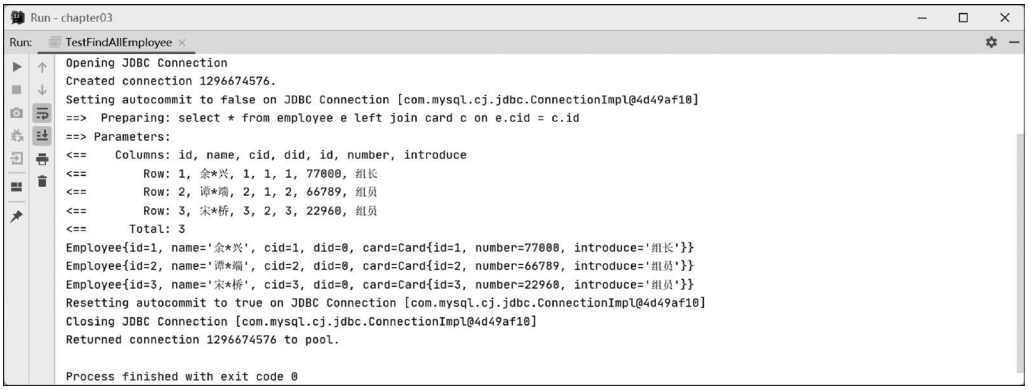

图 3-2 测试一对一关联映射的结果

从图 3-2 中可以看出,employee 对象中的 card 属性一对一映射成功。did 字段此处不做处理,3.2.3 节中将使用该字段对一对多关联映射进行详细讲解。

3.2.2 一对多关联映射

1. 设计背景

一对多关联映射是指一个表中的一条记录对应着另一个表中的多条记录。在本节中,员工表包含了一个部门外键 did,其中每位员工归属于一个部门,但一个部门可以包含多位员工。为了实现一对多关联映射,可以借助< collection >元素来完成这样的映射。

2. 创建实体类和数据库

(1) 员工类 Employee 已存在,在 chapter03 项目的 com.qfedu.pojo 包中新建部门实体类 Department,具体代码如例 3-11 所示。

例 3-11 Department.java。

```
public class Department {
    private int id;                              //部门表 id
    private String name;                         //部门名称
    private List < Employee > employees;         //员工集合
    //此处省略构造方法、Getter()、Setter()和 toString 方法
}
```

(2) 在数据库 student 中创建部门表 department,SQL 语句如下所示。

```
DROP TABLE IF EXISTS 'department';
CREATE TABLE 'department' (
  'id' int(0) NOT NULL AUTO_INCREMENT,
  'name' varchar(255) NULL DEFAULT NULL,
  PRIMARY KEY('id') USING BTREE
  )ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE = utf8mb4_0900_ai_ci
  ROW_FORMAT = Dynamic;
```

(3) 向部门表插入数据,SQL 语句如下所示。

```
INSERT INTO 'department' VALUES (1, '宣传部');
INSERT INTO 'department' VALUES (2, '行政部');
```

3. 编写映射文件

(1) 在 chapter03 项目的 com. qfedu. mapper 包中新建 DepartmentMapper 接口,具体代码如例 3-12 所示。

例 3-12 DepartmentMapper. java。

```
1  public interface DepartmentMapper{
2      List < Department > findAllDepartment();
3  }
```

(2) 在 chapter03 项目的 com. qfedu. mapper 包中新建名为 DepartmentMapper 的 XML 文件,作为 DepartmentMapper 接口的映射文件,具体代码如例 3-13 所示。

例 3-13 DepartmentMapper. xml。

```
1  <?xml version = "1.0" encoding = "UTF - 8"?>
2  <! DOCTYPE mapper PUBLIC " - //mybatis.org//DTD Mapper 3.0//EN"
3          "http://mybatis.org/dtd/mybatis - 3 - mapper.dtd">
4  < mapper namespace = "com.qfedu.mapper.DepartmentMapper">
5      < resultMap id = "departmentMap" type = "department">
6          < id property = "id" column = "did"/>
7          < result property = "name" column = "name"/>
8          < collection property = "employees" ofType = "employee">
9              < id property = "eid" column = "id"/>
10             < result property = "name" column = "name"/>
11         </collection >
12     </resultMap >
13     < select id = "findAllDepartment" resultMap = "departmentMap">
14         select d.id did, d.name, e.name, e.id eid from department d
15         left join employee e on e.did = d.id
16     </select >
17 </mapper >
```

在例 3-13 中,定义了一个名为 departmentMap 的映射规则,用< collection >元素将 department 表的数据映射到 department 对象和关联的 employees 对象中。同时,本示例中还定义了一个查询操作 employees,用于查询部门信息,并在查询结果中包含了与部门关联的员工信息。

4. 编写测试类

(1) 在 chapter03 项目的 com. qfedu. test 包中创建 TestFindAllDepartment 类,用于验证一对多关联映射,具体代码如例 3-14 所示。

例 3-14 TestFindAllDepartment. java。

```
1  public class TestFindAllDepartment {
2      public static void main(String[] args) {
3          /* 创建输入流 */
4          InputStream inputStream = null;
5          /* 将 MyBatis 配置文件转化为输入流 */
6          try {
7              inputStream =
8                      Resources.getResourceAsStream("mybatis - config.xml");
9          } catch (IOException e) {
10             e.printStackTrace();
```

```
        }
        /* 通过 SqlSessionFactoryBuilder()创建 SqlSessionFactory 对象 */
        SqlSessionFactory build =
                new SqlSessionFactoryBuilder().build(inputStream);
        /* 通过 SqlSessionFactory 对象创建 SqlSession 对象 */
        SqlSession sqlSession = build.openSession();
        DepartmentMapper mapper =
        sqlSession.getMapper(DepartmentMapper.class);
        List<Department> departments = mapper.findAllDepartment();
        for (Department department : departments) {
            System.out.println(department.toString());
        }
        /* 关闭事务 */
        sqlSession.close();
    }
}
```

(2) 执行例 3-14 中的 main()方法。测试一对多关联映射的结果如图 3-3 所示。

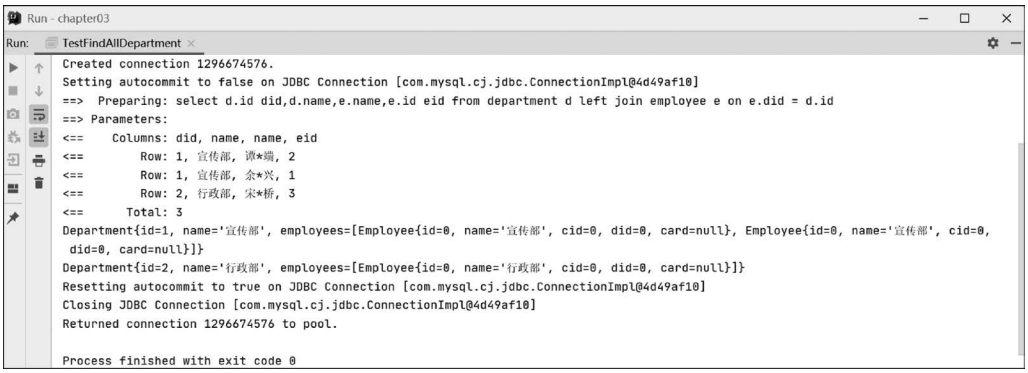

图 3-3 测试一对多关联映射的结果

从图 3-3 可以看出,Department 类中的集合对象 employees 一对多映射成功,输出该部门员工的详细信息。

3.2.3 多对多关联映射

1. 设计背景

多对多关联映射是指两个实体之间存在多对多的关系,通常需要中间表来建立它们之间的连接。

在一个 OA 管理系统中,涉及两个实体:员工和技能培训课程。由于培训需求中,每个员工可以选择参加多门技能培训课程,同时每门技能培训课程也可以有多名员工参加。每个员工有对应的工号。为了有效地处理这种多对多的关系,可以通过 MyBatis 的 <collection>元素实现这种关联映射。

2. 创建实体类和数据库

(1) 在 chapter03 项目的 com.qfedu.pojo 包中新建技能培训课程的实体类 Course,具体代码如例 3-15 所示。

例 3-15 Course.java。

```
public class Course {
    Integer id;
    String courseName;
    List<Employee> employees;
    此处省略 Getter()、Setter()、toString()和构造方法
}
```

(2) 员工类中需要课程类的实体类集合属性,在 Employee 类中添加集合对象 courses,具体代码如例 3-16 所示。

例 3-16 Employee.java。

```
public class Employee {
    private int id;
    private String name;
    private List<Course> courses;
}
```

(3) 创建技能培训课程表 course,SQL 语句如下所示。

```
DROP TABLE IF EXISTS 'course';
CREATE TABLE 'course' (
  'id' int(0) NOT NULL AUTO_INCREMENT,
  'course_name' varchar(255) CHARACTER NOT NULL,
  PRIMARY KEY('id') USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_0900_ai_ci ROW_FORMAT = Dynamic;
```

(4) 向技能培训课程表中插入数据,SQL 语句如下所示。

```
INSERT INTO 'course' VALUES (1, '礼仪培训');
INSERT INTO 'course' VALUES (2, '话术培训');
```

(5) 多对多关联映射需要中间表做关联,创建中间表 employee_course 的 SQL 语句如下所示。

```
DROP TABLE IF EXISTS 'employee_course';
CREATE TABLE 'employee_course' (
  'id' int(0) NOT NULL AUTO_INCREMENT,
  'e_id' int(0) NOT NULL,
  'c_id' int(0) NOT NULL,
  PRIMARY KEY('id') USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_0900_ai_ci ROW_FORMAT = Dynamic;
```

中间表 employee_course 包括 3 个字段:主键 id、表示员工 id 的 e_id 和表示课程 id 的 c_id。

(6) 向中间表中插入数据,SQL 语句如下所示。

```
INSERT INTO 'employee_course' VALUES (1, 1, 1);
INSERT INTO 'employee_course' VALUES (2, 2, 1);
INSERT INTO 'employee_course' VALUES (3, 3, 2);
```

3. 创建接口文件和映射文件

(1) 在 chapter03 项目的 EmployeeMapper 接口中新增查询员工的技能培训课程的方

法,具体代码如例3-17所示。

例3-17 EmployeeMapper.java。

```
public interface EmployeeMapper {
    List<Employee> findEmployeeCourse();
}
```

(2) 在chapter03项目的EmployeeMapper.xml文件中创建多对多映射关系,具体代码如例3-18所示。

例3-18 EmployeeMapper.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="com.qfedu.mapper.EmployeeMapper">
    <resultMap id="courseMap" type="employee">
        <id column="eid" property="id" />
        <result column="ename" property="name"/>
        <collection property="courses" ofType="com.qfedu.pojo.Course">
            <id column="id" property="cid"/>
            <result column="course_name" property="courseName"></result>
        </collection>
    </resultMap>
    <select id="findEmployeeCourse" resultMap="courseMap">
        select e.id eid,e.name ename,c.id cid,c.course_name courseName
        from employee e
        left join employee_course ec
        on e.id = ec.e_id
        left join
        course c
        on
        ec.c_id = c.id
    </select>
</mapper>
```

在例3-18中,定义了一个名为courseMap的映射规则,用<collection>元素将员工与其参与的课程相关联。同时,本例中还定义了一个查询操作findEmployeeCourse,从数据库中选择员工信息以及他们参与的课程,并将结果映射到Java对象中,以便获取员工与课程的多对多关联信息。

4. 编写测试类

(1) 在chapter03项目的com.qfedu.test包中新建TestFindAEmployeeCourse类,用于验证多对多关联映射,具体代码如例3-19所示。

例3-19 TestFindAEmployeeCourse.java。

```
public class TestFindAEmployeeCourse {
    public static void main(String[] args) {
        /*创建输入流*/
        InputStream inputStream = null;
        /*将MyBatis配置文件转化为输入流*/
        try {
            inputStream =
                    Resources.getResourceAsStream("mybatis-config.xml");
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
        /* 通过 SqlSessionFactoryBuilder()创建 SqlSessionFactory 对象 */
        SqlSessionFactory build =
                new SqlSessionFactoryBuilder().build(inputStream);
        /* 通过 SqlSessionFactory 对象创建 SqlSession 对象 */
        SqlSession sqlSession = build.openSession();
        EmployeeMapper mapper = sqlSession.getMapper(
                                                EmployeeMapper.class);
        List<Employee> employeeCourse = mapper.findEmployeeCourse();
        for (Employee employee : employeeCourse) {
            System.out.println(employee);
        }
        /* 关闭事务 */
        sqlSession.close();
    }
}
```

(2) 执行例 3-19 中的 main()方法,测试多对多关联映射的结果如图 3-4 所示。

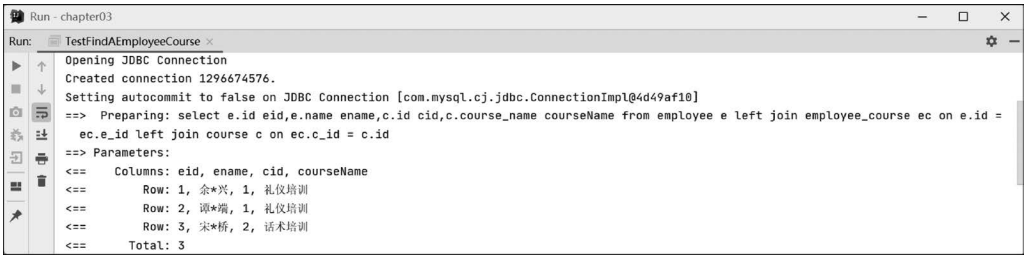

图 3-4 测试多对多关联映射的结果

从图 3-4 可以看出,Employee 类中的集合对象 course 多对多映射成功。

MyBatis 的关联映射可以提高查询效率、避免冗余数据、简化代码、提高可读性、支持高级查询等,对于开发大型数据库应用程序具有重要意义。

3.3 实战演练:智慧农业果蔬系统中普通用户的数据管理

为了加深对 MyBatis 核心组件相关编程知识的理解,本节以智慧农业果蔬系统为例,开发该系统中普通用户的管理模块。通过对该模块的实现,读者可以掌握 Mapper 和 XML 映射文件的相关知识。本实战演练的实战描述、实战分析和实现步骤如下所示。

【实战描述】

使用 IDEA 软件搭建一个 Web 项目 chapter03,通过 MyBatis 框架的 SqlSession 语法完成对 MySQL 数据库 chapter03 下普通用户表 user 的增、删、改、查操作,并在控制台输出日志信息。

【实战分析】

(1) 创建一个名为 chapter03 的数据库,并在该数据库下创建数据表 user。

(2) 向表中插入测试数据。

(3) 在 IDEA 软件中创建一个名为 chapter03 的项目,并引入 MyBatis 的相关 JAR 包。

(4) 在 chapter03 项目下创建对应的 POJO 类、接口、映射文件、配置文件和测试类。

(5) 编写和执行测试类,验证数据库中的数据表信息是否同步,并查看控制台的打印日志是否正确。

【实现步骤】

1. 搭建开发环境

(1) 在 MySQL 中创建数据库 chapter03 和数据表 user,SQL 语句如例 3-20 所示。

例 3-20 user.sql。

```
DROP TABLE IF EXISTS 'user';
CREATE TABLE 'user' (
  'id' int(0) NOT NULL AUTO_INCREMENT COMMENT '注解 ID',
  'userName' varchar(255) CHARACTER SET DEFAULT NULL COMMENT '用户名',
  'passWord' varchar(255) CHARACTER SET DEFAULT NULL COMMENT '密码',
  'phone' varchar(255) CHARACTER SET DEFAULT NULL COMMENT '手机号',
  'realName' varchar(255) CHARACTER DEFAULT NULL COMMENT '真实姓名',
  'sex' varchar(255) CHARACTER SET DEFAULT NULL COMMENT '性别',
  'address' varchar(255) CHARACTER SET DEFAULT NULL COMMENT '地址',
  'email' varchar(255) CHARACTER SET DEFAULT NULL COMMENT '邮箱',
  PRIMARY KEY('id') USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 9 CHARACTER SET = utf8 COLLATE =
utf8_general_ci ROW_FORMAT = Dynamic;
```

(2) 向数据表 user 中插入数据,SQL 语句如下所示。

```
INSERT INTO 'user' VALUES (1, '曾*梁', '2', '138****6907', '曾*梁', '男', '北京市昌平区', '138
****6907@163.com');
INSERT INTO 'user' VALUES (2, 'wu', 'dd', '156****1543', '吴*英', '男', '北京市海淀区', '156*
***1543@163.com');
INSERT INTO 'user' VALUES (3, '吴*德', '111111', '192****9012', '吴*德', '女', '北京市丰台区',
'192****9012@163.com');
INSERT INTO 'user' VALUES (6, 'wang', '123456', '155****2130', '王*强', '女', '北京市房山区',
'155****2130@163.com');
INSERT INTO 'user' VALUES (7, 'fang', '123456', '170****1239', '方*智', '女', '北京市通州区',
'170****1239@163.com');
INSERT INTO 'user' VALUES (8, 'jian', '11', '166****8613', '*坚', '男', '北京市密云区', '166*
***8613@163.com');
```

(3) 在 Windows 的命令提示符窗口中输入查询 user 表数据的 SQL 语句,具体语句如下所示。

```
select * from user;
```

(4) 执行查询 user 表数据的 SQL 语句,查询结果如图 3-5 所示。

从图 3-5 中可以看出,user 表数据添加成功。

2. 创建项目

(1) 在 IDEA 中新建 Web 项目 chapter03,将 MyBatis 的 JAR 包 mybatis-3.5.6.jar 复制到 WEB-INF 下的 lib 文件夹中,完成 JAR 包的导入。

(2) 在 chapter03 项目的 src 目录下创建 com.qfedu.pojo 包,并在该包下新建 User 类,具体代码如例 3-21 所示。

```
mysql> select * from user;
+----+----------+----------+-------------+----------+------+--------------+---------------------+
| id | userName | passWord | phone       | realName | sex  | address      | email               |
+----+----------+----------+-------------+----------+------+--------------+---------------------+
|  1 | 曾*梁    | 2        | 138****6907 | 曾*梁    | 男   | 北京市昌平区 | 138****6907@163.com |
|  2 | wu       | dd       | 156****1543 | 吴*英    | 男   | 北京市海淀区 | 156****1543@163.com |
|  3 | 吴*德    | 111111   | 192****9012 | 吴*德    | 女   | 北京市丰台区 | 192****9012@163.com |
|  6 | wang     | 123456   | 155****2130 | 王*强    | 女   | 北京市房山区 | 155****2130@163.com |
|  7 | fang     | 123456   | 170****1239 | 方*智    | 女   | 北京市通州区 | 170****1239@163.com |
|  8 | jian     | 11       | 166****8613 | *坚      | 男   | 北京市密云区 | 166****8613@163.com |
+----+----------+----------+-------------+----------+------+--------------+---------------------+
6 rows in set (0.01 sec)
```

图 3-5　user 表查询结果

例 3-21　User.java。

```
1  public class User {
2      private int id;
3      private String userName;
4      private String passWord;
5      private String phone;
6      private String realName;
7      private String sex;
8      private String address;
9      private String email;
10     //此处省略 Getter/Setter、toString()和构造方法
11 }
```

需要注意的是，在上述代码中，User 类必须提供 Setter 方法，这样 MyBatis 框架才能通过配置文件映射 User 类和数据表 user 的关系。

(3) 在 resource 目录下新建 MyBatis 的配置文件 mybatis-config.xml，具体代码如例 3-22 所示。

例 3-22　mybatis-config.xml。

```
1  <?xml version = "1.0" encoding = "UTF - 8" ?>
2  <! DOCTYPE configuration
3          PUBLIC " - //mybatis.org//DTD Config 3.0//EN"
4          "http://mybatis.org/dtd/mybatis - 3 - config.dtd">
5  < configuration >
6     <!-- 设置 -- >
7     < settings >
8       <!-- 开启数据库日志检测 -- >
9       < setting name = "logImpl" value = "STDOUT_LOGGiNG"/>
10    </settings >
11    <!-- 包名简化缩进 -- >
12    < typeAliases >
13      <!-- typeAlias 方式 -- >
14      < package name = "com.qfedu.pojo"/>
15    </typeAliases >
16    <!-- 配置环境 -- >
17    < environments default = "dev">
18      <!-- 配置 MySQL 环境 -- >
19      < environment id = "dev">
20          <!-- 配置事务管理器 -- >
21          < transactionManager type = "JDBC"/>
22          <!-- 配置数据库连接 -- >
23          < dataSource type = "POOLED">
```

```
24              <!-- 配置数据库连接驱动 -->
25              <property name = "driver" value = "com.mysqljdbc.Driver"/>
26              <!-- 配置数据库连接地址 -->
27              <property name = "url" value = "localhost:3306/chapter02"/>
28              <!-- 配置用户名 -->
29              <property name = "username" value = "root"/>
30              <!-- 配置密码 -->
31              <property name = "password" value = "root"/>
32            </dataSource>
33        </environment>
34      </environments>
35      <mappers>
36        <package name = "com.qfedu.mapper"/>
37      </mappers>
38  </configuration>
```

在例 3-22 中,第 23～32 行代码用于配置数据库的连接信息,其中< dataSource >元素的 4 个属性分别配置数据库的驱动、URL、用户名和密码;第 35～37 行代码通知 MyBatis 在包 com.qfedu.mapper 下寻找并加载映射文件。

(4) 在 src 目录下创建 com.qfedu.mapper 包,在该包下新建 UserMapper 接口,用于声明查询、新增、修改和删除的方法,具体代码如例 3-23 所示。

例 3-23 UserMapper.java。

```
1  public interface UserMapper {
2      //查询
3      List < User > findAllUser();
4      //新增
5      Integer insertUser(User User);
6      //修改
7      Integer updateUser(User User);
8      //删除
9      Integer deleteUser(User User);
10 }
```

(5) 在 com.qfedu.mapper 包下新建 UserMapper 接口对应的映射文件 UserMapper.xml,具体代码如例 3-24 所示。

例 3-24 UserMapper.xml。

```
1  <?xml version = "1.0" encoding = "UTF - 8"?>
2  <!DOCTYPE mapper PUBLIC " - //mybatis.org//DTD Mapper 3.0//EN"
3          "http://mybatis.org/dtd/mybatis - 3 - mapper.dtd">
4  < mapper namespace = "com.qfeud.mapper.UserMapper">
5    < select id = "findAllUser" resultType = "com.qfedu.pojo.User">
6          select * from User
7    </select>
8    < insert id = "insertUser" parameterType = "com.qfedu.pojo.User">
9    insert into user(userName,passWord,phone,relName,sex,address,email)
10   values(#{userName},#{passWord},#{phone},#{relName},#{sex},
11   #{address},#{email})
12   </insert>
13   < update id = "updateUser" parameterType = "com.qfedu.pojo.User">
14     update User set passWord = #{passWord} where id = #{id}
```

```
15   </update>
16   <delete id="deleteUser">
17     delete from User where id = #{id}
18   </delete>
19 </mapper>
```

在例 3-24 中，第 5 行代码< select >元素中的 id 属性值 findAllUser 用于映射 UserMapper 接口中的 findAllUser()方法；第 8 行代码< insert >元素中的 id 属性值 insertUser 用于映射 UserMapper 接口中的 insertUser()方法；第 13 行代码< update >元素中的 id 属性值 updateUser 用于映射 UserMapper 接口中的 updateUser()方法；第 16 行代码< delete >元素中的 id 属性值 deleteUser 用于映射 UserMapper 接口中的 deleteUser()方法。

3. 编写测试类

(1) 在 src 目录下创建 com.qfedu.test 包，在该包下新建 TestFindAllUser 类，该类测试查询 user 表中所有普通用户的数据信息操作，具体代码如例 3-25 所示。

例 3-25 TestFindAllUser.java。

```
1  public class TestFindAllUser {
2      public static void main(String[] args) {
3          /*创建输入流*/
4          InputStream inputStream = null;
5          /*将 MyBatis 配置文件转化为输入流*/
6          try {
7              inputStream =
8                      Resources.getResourceAsStream("mybatis-config.xml");
9          } catch (IOException e) {
10             e.printStackTrace();
11         }
12         /*通过 SqlSessionFactoryBuilder()创建 SqlSessionFactory 对象*/
13         SqlSessionFactory build =
14                 new SqlSessionFactoryBuilder().build(inputStream);
15         /*通过 SqlSessionFactory 对象创建 SqlSession 对象*/
16         SqlSession sqlSession = build.openSession();
17         List<User> Users = sqlSession.selectList
18         ("com.qfedu.mapper.UserMapper.findAllUser");
19         for (User User : Users) {
20               System.out.println(User);
21         }
22         /*关闭事务*/
23         sqlSession.close();
24     }
25 }
```

(2) 执行 TestFindAllUser 类的 main()方法，查询智慧农业果蔬系统中普通用户信息的结果如图 3-6 所示。

从图 3-6 可以看出，控制台输出智慧农业果蔬系统中普通用户的 6 条数据信息，查询普通人员的操作执行成功。

(3) 在 com.qfedu.test 包下新建 TestInsertUser 类，该类用于实现向 user 表中新增一条记录，用户名为“周*扬”，密码为“24”，地址为“北京”，具体代码如例 3-26 所示。

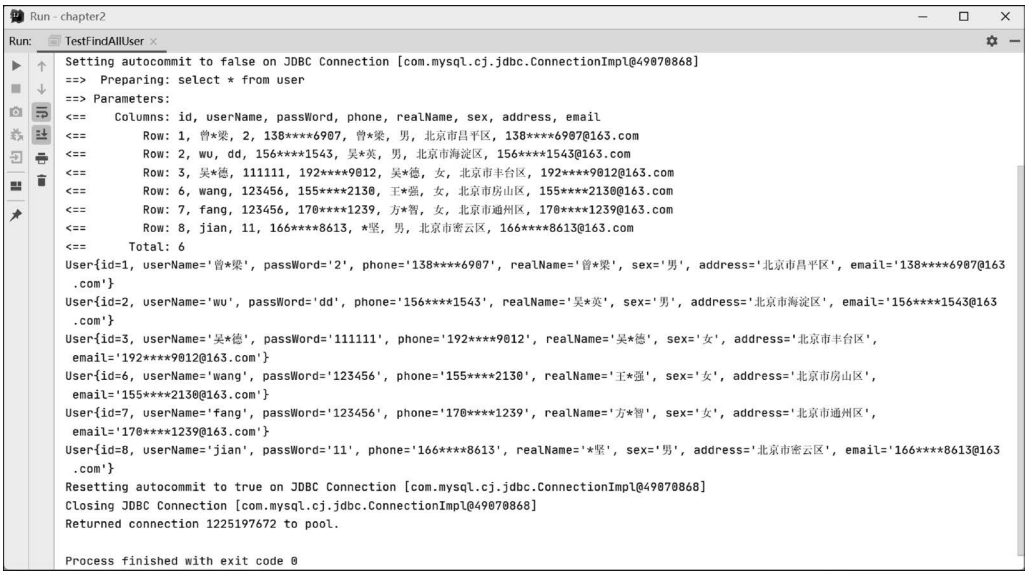

图 3-6　查询智慧农业果蔬系统中普通用户信息的结果

例 3-26　TestInsertUser.java。

```
1  public class TestInsertUser {
2      public static void main(String[] args) {
3          /*创建输入流*/
4          InputStream inputStream = null;
5          /*将 MyBatis 配置文件转化为输入流*/
6          try {
7              inputStream =
8                      Resources.getResourceAsStream("mybatis-config.xml");
9          } catch (IOException e) {
10             e.printStackTrace();
11         }
12         /*通过 SqlSessionFactoryBuilder()创建 SqlSessionFactory 对象*/
13         SqlSessionFactory build =
14                 new SqlSessionFactoryBuilder().build(inputStream);
15         /*通过 SqlSessionFactory 对象创建 SqlSession 对象*/
16         SqlSession sqlSession = build.openSession(true);
17         User User = new User();
18         User.setUserName("周*扬");
19         User.setPassWord("24");
20         User.setAddress("北京");
21       sqlSession.insert("com.qfedu.mapper.UserMapper.insertUser",User);
22         /*关闭事务*/
23         sqlSession.close();
24     }
25 }
```

(4) 执行 TestInsertUser 类的 main()方法,控制台的新增日志如图 3-7 所示。

从图 3-7 中可以看出,控制台的日志输出一条用户名为“周*扬”、密码为“24”、地址为“北京”的插入语句,并返回受影响的行数为 1,新增普通用户的操作执行成功。

(5) 在 com.qfedu.test 包下新建 TestUpdateUser 类,该类用于实现修改 id 为 10 的记

图 3-7　控制台的新增日志

录,具体代码如例 3-27 所示。

例 3-27　TestUpdateUser.java。

```
1  public class TestUpdateUser {
2      public static void main(String[] args) {
3          /*创建输入流*/
4          InputStream inputStream = null;
5          /*将 MyBatis 配置文件转化为输入流*/
6          try {
7              inputStream =
8                      Resources.getResourceAsStream("mybatis-config.xml");
9          } catch (IOException e) {
10             e.printStackTrace();
11         }
12         /*通过 SqlSessionFactoryBuilder()创建 SqlSessionFactory 对象*/
13         SqlSessionFactory build =
14                 new SqlSessionFactoryBuilder().build(inputStream);
15         /*通过 SqlSessionFactory 对象创建 SqlSession 对象*/
16         SqlSession sqlSession = build.openSession(true);
17         User User = new User();
18         User.setId(10);
19         User.setPassWord("123456");
20         sqlSession.update("com.qfedu.mapper.UserMapper.updateUser",User);
21         /*关闭事务*/
22         sqlSession.close();
23     }
24 }
```

(6) 执行 TestUpdateUser 类的 main()方法,控制台的修改日志如图 3-8 所示。

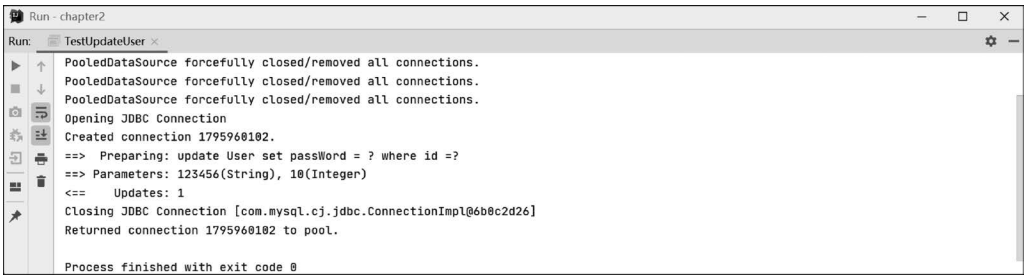

图 3-8　控制台的修改日志

从图 3-8 中可以看出,控制台的日志输出一条 id 为 10、账号密码为 123456 的修改语句,修改普通用户的操作执行成功。

(7) 在 com.qfedu.test 包下新建 TestDeleteUser 类,用于测试删除 id 为 10 的记录的操作,具体代码如例 3-28 所示。

例 3-28 TestDeleteUser.java。

```
public class TestDeleteUser {
    public static void main(String[] args) {
        /*创建输入流*/
        InputStream inputStream = null;
        /*将 MyBatis 配置文件转化为输入流*/
        try {
            inputStream =
                    Resources.getResourceAsStream("mybatis-config.xml");
        } catch (IOException e) {
            e.printStackTrace();
        }
        /*通过 SqlSessionFactoryBuilder()创建 SqlSessionFactory 对象*/
        SqlSessionFactory build =
                new SqlSessionFactoryBuilder().build(inputStream);
        /*通过 SqlSessionFactory 对象创建 SqlSession 对象*/
        SqlSession sqlSession = build.openSession(true);
        sqlSession.delete("com.qfedu.mapper.UserMapper.deleteUser",10);
        /*关闭事务*/
        sqlSession.close();
    }
}
```

(8) 执行 TestDeleteUser 类的 main()方法,控制台输出的删除普通用户的日志如图 3-9 所示。

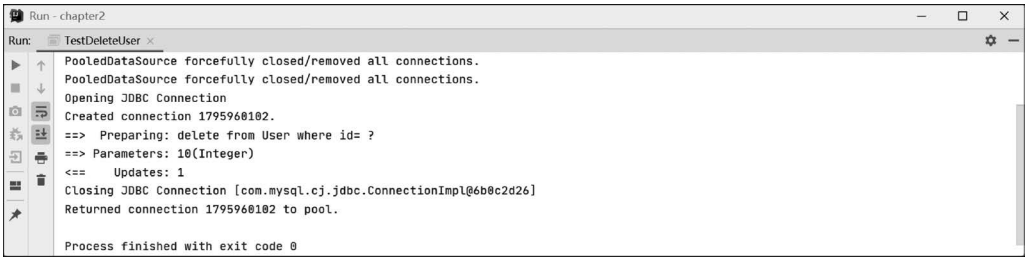

图 3-9 控制台输出的删除普通用户的日志

从图 3-9 中可以看出,控制台的日志输出一条 id 为 10 的删除语句,并返回受影响的行数 1,删除普通用户的操作执行成功。

本节使用 MyBatis 的基础知识实现了智慧农业果蔬系统中普通用户的数据管理模块,要求读者掌握 MyBatis 的映射文件、关联映射、基本语法和使用规范。

3.4 本章小结

本章首先介绍了 MyBatis 的映射文件结构和元素的使用方法,然后讲解了 MyBatis 的关联关系映射,最后通过一个实战演练帮助读者巩固 MyBatis 的映射文件、关联映射、基本语法和使用规范的灵活运用。通过对本章的学习,读者可以更好地理解和应用 MyBatis 的映射文件,实现高效、灵活的数据库操作,了解如何处理复杂的关联映射数据。

3.5 习　　题

一、填空题

1. MyBatis 的映射文件名一般为_____。

2. 在 MyBatis 的映射文件中，用于映射查询语句的元素是_____。

3. 在 MyBatis 的映射文件中，用于定义可重用 SQL 代码片段的元素是_____。

4. 在 MyBatis 中，如果关联关系的 Java 对象名称与数据库表字段名称不一致，可以使用_____标签对它们进行映射。

5. MyBatis 的 3 种关联关系映射分别是_____、_____和_____。

二、选择题

1. 在 MyBatis 中，以下哪个标签用于定义关联关系映射？（　　）

A. <select>　　B. <insert>　　C. <update>　　D. <association>

2. 在 MyBatis 的关联关系映射中，以下哪个标签用于定义一对一关联关系？（　　）

A. <association>　　B. <collection>

C. <resultMap>　　D. <parameterMap>

3. 在 MyBatis 的关联关系映射中，以下哪个标签用于定义一对多关联关系？（　　）

A. <association>　　B. <collection>

C. <resultMap>　　D. <parameterMap>

4. 关于 MyBatis 的映射文件，下列描述错误的是（　　）。

A. 一个 MyBatis 配置文件中可引入多个映射文件

B. 在编写 MyBatis 的映射文件时，开发人员无须关心元素顺序

C. <mappers>元素是 MyBatis 映射文件的根元素

D. 在编写 MyBatis 映射文件时，不是所有的元素都必须配置

5. 在下列选项中，不属于<select>元素的属性是（　　）。

A. id　　B. resultType　　C. resultMap　　D. value

三、简答题

1. 简述常见的 MyBatis 映射文件元素的功能和用法。

2. 简述一对一表关系的处理过程。

3. 简述一对多表关系的处理过程。

四、操作题

请编写一个程序，实现使用 MyBatis 标签对 Dog 类进行增、删、改、查操作，具体步骤可参考下方内容。

（1）创建 Dog 类，在该类中添加 name、age 属性。

（2）搭建 MyBatis 框架，使用 MyBatis 标签对 Dog 类进行增、删、改、查操作。