第6章 初识 MyBatis

学习目标

- 了解 MyBatis 的基础知识
- 熟悉 MyBatis 的工作原理
- 掌握 MyBatis 入门程序的编写

MyBatis 是当前主流的 Java 持久层框架之一，它与 Hibernate 一样，也是一种 ORM 框架。因其性能优异，且具有高度的灵活性、可优化性和易于维护等特点，所以受到了广大互联网企业的青睐，是目前大型互联网项目的首选框架。从本章开始，我们将对 MyBatis 框架的相关知识进行详细讲解。

6.1 什么是 MyBatis

MyBatis（前身是 iBatis）是一个支持普通 SQL 查询、存储过程以及高级映射的持久层框架，它消除了几乎所有的 JDBC 代码和参数的手动设置以及对结果集的检索，并使用简单的 XML 或注解进行配置和原始映射，用以将接口和 Java 的 POJO（Plain Old Java Object，普通 Java 对象）映射成数据库中的记录，使得 Java 开发人员可以使用面向对象的编程思想来操作数据库。

MyBatis 框架也被称之为 ORM（Object/Relation Mapping，即对象关系映射）框架。所谓的 ORM 就是一种为了解决面向对象与关系型数据库中数据类型不匹配的技术，它通过描述 Java 对象与数据库表之间的映射关系，自动将 Java 应用程序中的对象持久化到关系型数据库的表中。ORM 框架的工作原理如图 6-1 所示。

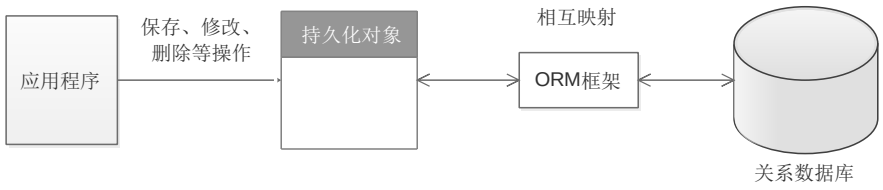

图6-1 ORM 框架的工作原理

从图 6-1 可以看出，使用 ORM 框架后，应用程序不再直接访问底层数据库，而是以面向对象的方式来操作持久化对象（PO，即 Persisent Object），而 ORM 框架则会通过映射关系将这些面向对象的操作转换成底层的 SQL 操作。

当前的 ORM 框架产品有很多，常见的 ORM 框架有 Hibernate 和 MyBatis。这两个框架的主要区别如下：

- Hibernate：是一个全表映射的框架。通常开发者只需定义好持久化对象到数据库表的映射关系，就可以通过 Hibernate 提供的方法完成持久层操作。开发者并不需要熟练的掌握 SQL 语句的编写，Hibernate 会根据制定的存储逻辑，自动的生成对应的 SQL，并调用 JDBC 接口来执行，所以其开发效率会高于 MyBatis。然而 Hibernate 自身也存在着一些缺点，例如它在多表关联时，对 SQL 查询的支持较差；更新数据时，需要发送所有字段；不支持存储过程；不能通过优化 SQL 来优化性能等。这些问题导致其只适合在场景不太复杂且对性能要求不高的项目中使用。
- MyBatis：是一个半自动映射的框架。这里所谓的“半自动”是相对于 Hibernate 全表映射而言的，MyBatis 需要手动匹配提供 POJO、SQL 和映射关系，而 Hibernate 只需提供 POJO 和映射关系即可。与 Hibernate 相比，虽然使用 MyBatis 手动编写 SQL 要比使用 Hibernate 的工作量大，但 MyBatis 可以配置动态 SQL 并优化 SQL，可以通过配置决定 SQL 的映射规则，它还支持存储过程等。对于一些复杂的和需要优化性能的项目来说，显然使用 MyBatis 更加合适。

6.2 MyBatis 的下载和使用

在本书编写时，MyBatis 的最新版本是 mybatis-3.4.2，本书所讲解的 MyBatis 框架就是基于此版本的，所以希望读者也下载同样的版本，以便于学习。

该版本的 MyBatis 可以通过网址“https://github.com/mybatis/mybatis-3/releases”下载得到。在浏览器中输入网址后，可以在页面中看到 MyBatis 对应的下载链接，如图 6-2 所示。

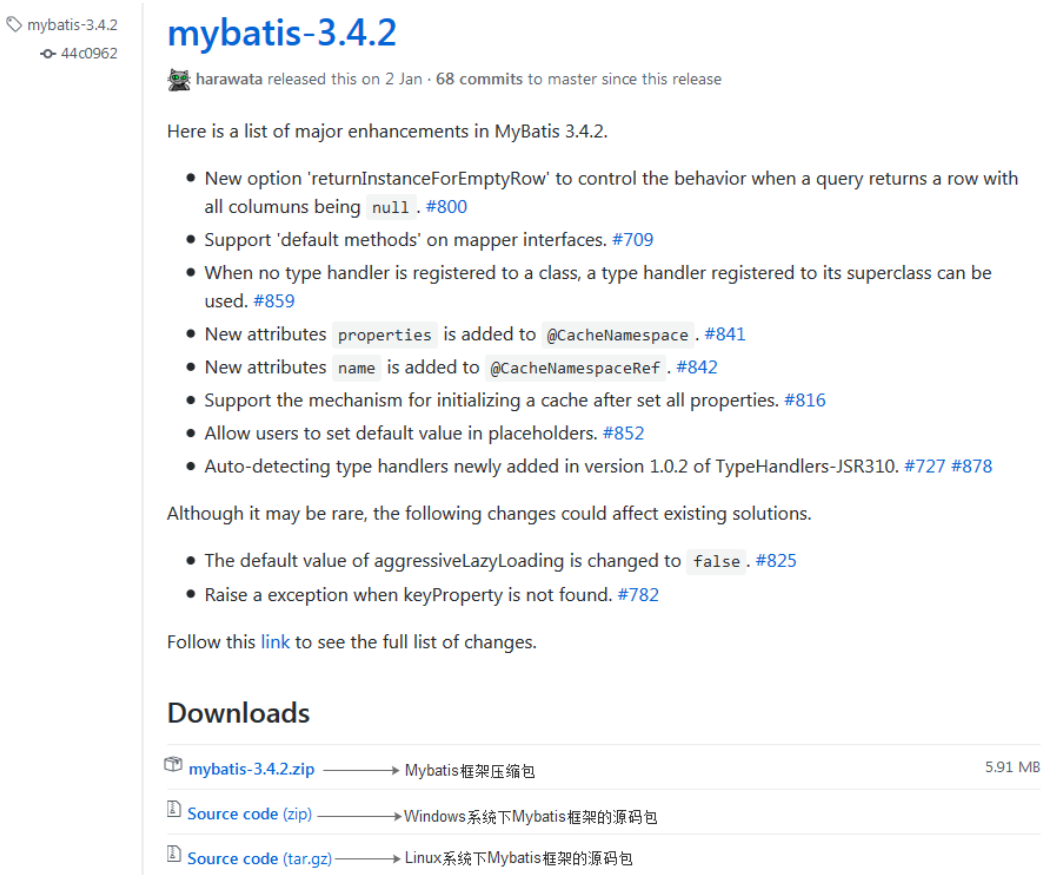

图6-2　MyBatis 的下载

从图 6-2 可以看出，页面中包含了三个下载链接，这里我们只需下载 mybatis-3.4.2.zip 即可。下载并解压 mybatis-3.4.2.zip 压缩包，会得到一个名为 mybatis-3.4.2 的文件夹，该文件夹下包含的文件如图 6-3 所示。

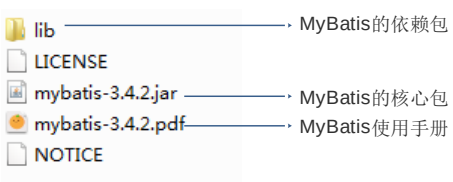

图6-3　MyBatis 压缩包内的文件结构

使用 MyBatis 框架非常简单，只需在应用程序中引入 MyBatis 的核心包和 lib 目录中的依赖包即可。

注意：

如果底层采用的是 MySQL 数据库，那么还需要将 MySQL 数据库的驱动 JAR 包添加到应用程序的类路径中；如果采用其他类型的数据库，则同样需要将对应类型的数据库驱动包添加到应用程序的类路径中。

6.3 MyBatis 的工作原理

为了使读者能够更加清晰的理解 MyBatis 程序，在正式讲解 MyBatis 入门案例之前，先来了解一下 MyBatis 程序的工作原理，如图 6-4 所示。

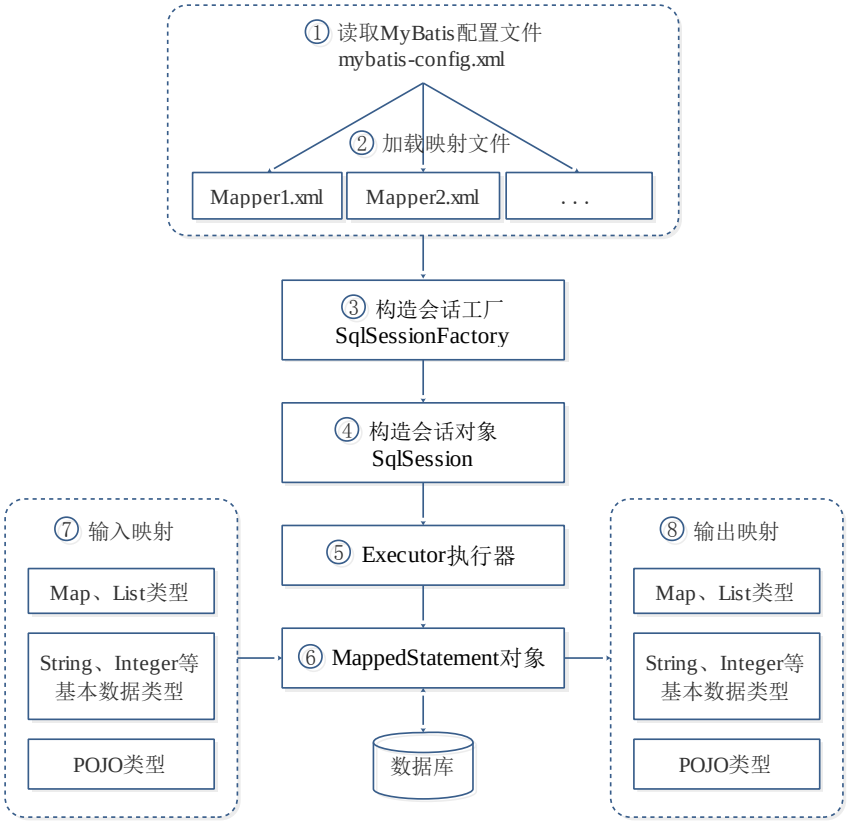

图6-4　MyBatis 框架执行流程图

从图 6-4 可以看出，MyBatis 框架在操作数据库时，大体经过了 8 个步骤。下面就对图 6-4 中的每一步流程进行详细讲解，具体如下。

（1）读取 MyBatis 配置文件 mybatis-config.xml。mybatis-config.xml 作为 MyBatis 的全局配置文件，配置了 MyBatis 的运行环境等信息，其中主要内容是获取数据库连接。

（2）加载映射文件 Mapper.xml。Mapper.xml 文件即 SQL 映射文件，该文件中配置了操作数据库的 SQL 语句，需要在 mybatis-config.xml 中加载才能执行。mybatis-config.xml 可以加载多个配置文件，每个配置文件对应数据库中的一张表。

（3）构建会话工厂。通过 MyBatis 的环境等配置信息构建会话工厂 SqlSessionFactory。

（4）创建 SqlSession 对象。由会话工厂创建 SqlSession 对象，该对象中包含了执行 SQL 的所有方法。

（5）MyBatis 底层定义了一个 Executor 接口来操作数据库，它会根据 SqlSession 传递的参数动态的生成需要执行的 SQL 语句，同时负责查询缓存的维护。

（6）在 Executor 接口的执行方法中，包含一个 MappedStatement 类型的参数，该参数

是对映射信息的封装，用来存储要映射的 SQL 语句的 id、参数等。Mapper.xml 文件中一个 SQL 对应一个 MappedStatement 对象，SQL 的 id 即是 MappedStatement 的 id。

（7）输入参数映射。在执行方法时，MappedStatement 对象会对用户执行 SQL 语句的输入参数进行定义（可以定义为 Map、List 类型、基本类型和 POJO 类型），Executor 执行器会通过 MappedStatement 对象在执行 SQL 前，将输入的 Java 对象映射到 SQL 语句中。这里对输入参数的映射过程就类似于 JDBC 编程中对 preparedStatement 对象设置参数的过程。

（8）输出结果映射。在数据库中执行完 SQL 语句后，MappedStatement 对象会对 SQL 执行输出的结果进行定义（可以定义为 Map 和 List 类型、基本类型、POJO 类型），Executor 执行器会通过 MappedStatement 对象在执行 SQL 语句后，将输出结果映射至 Java 对象中。这种将输出结果映射到 Java 对象的过程就类似于 JDBC 编程中对结果的解析处理过程。

通过上面对 MyBatis 框架执行流程的讲解，相信读者对 MyBatis 框架已经有了一个初步的了解。对于初学者来说，上面所讲解的内容可能不会完全理解，现阶段也不要求读者能完全理解，这里讲解 MyBatis 框架的执行过程是为了方便后面程序的学习。在学习完 MyBatis 框架后，读者自然就会明白上面所讲解的内容了。

6.4 MyBatis 入门程序

通过前几个小节的学习，相信读者对 MyBatis 框架已经有了一个初步的了解。接下来的几个小节中，将通过一个客户信息管理的入门案例来讲解下 MyBatis 框架的基本使用。

6.4.1 查询客户

在实际开发中，查询操作通常都会涉及到单条数据的精确查询，以及多条数据的模糊查询。那么使用 MyBatis 框架是如何进行这两种查询的呢？接下来，本小节将讲解下如何使用 MyBatis 框架根据客户编号查询客户信息，以及根据客户名模糊查询客户信息。

1．根据客户编号查询客户信息

根据客户编号查询客户信息主要是通过查询客户表中的主键（这里表示唯一的客户编号）来实现的，其具体实现步骤如下：

（1）在 MySQL 数据库中，创建一个名为 mybatis 的数据库，在此数据库中创建一个 t_customer 表，同时预先插入几条数据。此操作所执行的 SQL 语句如下所示。

```
# 创建一个名称为 mybatis 的数据库
CREATE DATABASE mybatis;
# 使用 mybatis 数据库
USE mybatis;
# 创建一个名称为 t_customer 的表
CREATE TABLE t_customer (
```

```
    id int(32) PRIMARY KEY AUTO_INCREMENT,
    username varchar(50),
    jobs varchar(50),
    phone varchar(16)
);
# 插入 3 条数据
INSERT INTO t_customer VALUES ('1', 'joy', 'doctor', '13745874578');
INSERT INTO t_customer VALUES ('2', 'jack', 'teacher', '13521210112');
INSERT INTO t_customer VALUES ('3', 'tom', 'worker', '15179405961');
```

完成上述操作后，数据库 t_customer 表中的数据如图 6-5 所示。

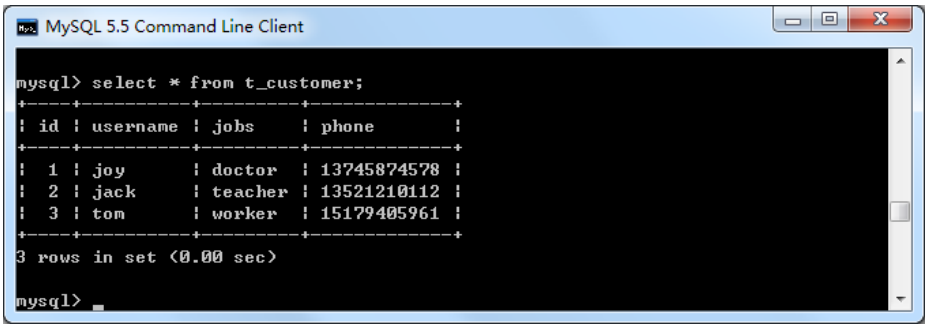

图6-5 t_customer 表

（2）在 Eclipse 中，创建一个名为 chapter06 的 Web 项目，将 MyBatis 的核心 JAR 包、lib 目录中的依赖 JAR 包，以及 MySQL 数据库的驱动 JAR 包一同添加到项目的 lib 目录下，并发布到类路径中。添加后的 lib 目录如图 6-6 所示。

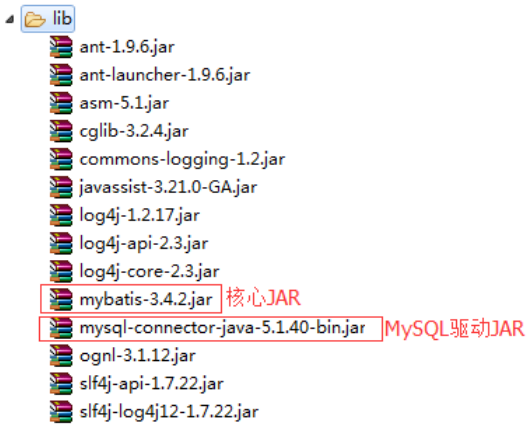

图6-6 MyBatis 相关 JAR 包

（3）由于 MyBatis 默认使用 log4j 输出日志信息，所以如果要查看控制台的输出 SQL 语句，那么就需要在 classpath 路径下配置其日志文件。在项目的 src 目录下创建 log4j.properties 文件，编辑后的内容如文件 6-1 所示。

文件6-1 log4j.properties

```
# Global logging configuration
```

```
log4j.rootLogger=ERROR, stdout
# MyBatis logging configuration...
log4j.logger.com.itheima=DEBUG
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

在文件 16-1 中，包含了全局的日志配置、MyBatis 的日志配置和控制台输出，其中 MyBatis 的日志配置用于将 com.itheima 包下所有类的日志记录级别设置为 DEBUG。

由于 log4j 文件中的具体内容已经超出了本书范围，所以这里不过多讲解，读者可自行查找资料学习。上述配置文件代码也不需要读者全部手写，在 MyBatis 使用手册中的 Logging 小节，可以找到如图 6-7 所示的配置信息，只需将其复制到项目的 log4j 配置文件中，并对 MyBatis 的日志配置信息进行简单修改即可使用。

Create a file called `log4j.properties` as shown below and place it in your classpath:

```
# Global logging configuration
log4j.rootLogger=ERROR, stdout
# MyBatis logging configuration...
log4j.logger.org.mybatis.example.BlogMapper=TRACE
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

图6-7　MyBatis 使用手册中的 Logging 配置

（4）在 src 目录下，创建一个 com.itheima.po 包，在该包下创建持久化类 Customer，并在类中声明 id、username、jobs 和 phone 属性，及其对应的 getter/setter 方法，如文件 6-2 所示。

文件6-2　Customer.java

```
1  package com.itheima.po;
2  /**
3   * 客户持久化类
4   */
5  public class Customer {
6      private Integer id;       // 主键 id
7      private String username; // 客户名称
8      private String jobs;      // 职业
9      private String phone;     // 电话
10     public Integer getId() {
11         return id;
12     }
13     public void setId(Integer id) {
14         this.id = id;
```

```
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getJobs() {
        return jobs;
    }
    public void setJobs(String jobs) {
        this.jobs = jobs;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    @Override
    public String toString() {
        return "Customer [id=" + id + ", username=" + username +
                        ", jobs=" + jobs + ", phone=" + phone + "]";
    }
}
```

从上述代码可以看出，持久化类 Customer 与普通的 JavaBean 并没有什么区别，只是其属性字段与数据库中的表字段相对应。实际上，Customer 就是一个 POJO（普通 Java 对象），MyBatis 就是采用 POJO 作为持久化类来完成对数据库操作的。

（5）在 src 目录下，创建一个 com.itheima.mapper 包，并在包中创建映射文件 CustomerMapper.xml，编辑后如文件 6-3 所示。

文件6-3 CustomerMapper.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">
<!-- namespace 表示命名空间 -->
<mapper namespace="com.itheima.mapper.CustomerMapper">
    <!--根据客户编号获取客户信息 -->
    <select id="findCustomerById" parameterType="Integer"
        resultType="com.itheima.po.Customer">
```

```
9          select * from t_customer where id = #{id}
10     </select>
11 </mapper>
```

在文件 6-3 中，第 2~3 行是 MyBatis 的约束配置，第 5~11 行是需要程序员编写的映射信息。其中，<mapper>元素是配置文件的根元素，它包含一个 namespace 属性，该属性为这个<mapper>指定了唯一的命名空间，通常会设置成“包名+SQL 映射文件名”的形式。子元素<select>中的信息是用于执行查询操作的配置，其 id 属性是<select>元素在映射文件中的唯一标识；parameterType 属性用于指定传入参数的类型，这里表示传递给执行 SQL 的是一个 Integer 类型的参数；resultType 属性用于指定返回结果的类型，这里表示返回的数据是 Customer 类型。在定义的查询 SQL 语句中，“#{}”用来表示一个占位符，相当于“?”，而“#{id}”表示该占位符待接收参数的名称为 id。

多学一招： 快速获取配置文件的约束信息

在 MyBatis 的映射文件中，包含了一些约束信息，初学者如果自己动手去编写，不但浪费时间，还容易出错。其实，在 MyBatis 使用手册中，就可以找到这些约束信息，具体的获取方法如下：

打开 MyBatis 的使用手册 mybatis-3.4.2.pdf，在第 2 小节 Getting started（入门指南）下的 2.1.5 小节 Exploring Mapped SQL Statements 中，即可找到映射文件的约束信息。如图 6-8 所示。

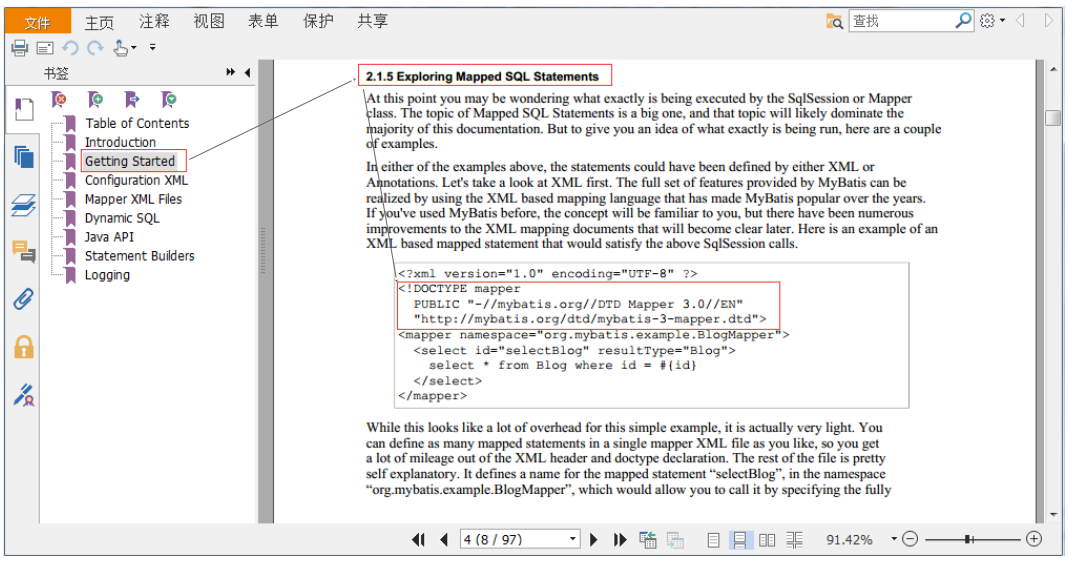

图6-8　映射文件的约束信息

在图 6-8 中，方框处标注的文件信息就是 MyBatis 映射文件的约束信息。初学者只需将信息复制到项目创建的 XML 文件中即可。

（6）在 src 目录下，创建 MyBatis 的核心配置文件 mybatis-config.xml，编辑后如文件 6-4 所示。

文件6-4　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     <!--1.配置环境 ，默认的环境 id 为 mysql-->
6     <environments default="mysql">
7        <!--1.2.配置 id 为 mysql 的数据库环境 -->
8        <environment id="mysql">
9           <!-- 使用 JDBC 的事务管理 -->
10          <transactionManager type="JDBC" />
11          <!--数据库连接池 -->
12          <dataSource type="POOLED">
13             <property name="driver" value="com.mysql.jdbc.Driver" />
14             <property name="url"
15                       value="jdbc:mysql://localhost:3306/mybatis" />
16             <property name="username" value="root" />
17             <property name="password" value="root" />
18          </dataSource>
19       </environment>
20    </environments>
21    <!--2.配置 Mapper 的位置 -->
22    <mappers>
23        <mapper resource="com/itheima/mapper/CustomerMapper.xml" />
24    </mappers>
25 </configuration>
```

在文件 6-3 中，第 2~3 行是 MyBatis 的配置文件的约束信息，下面<configuration>元素中的内容就是开发人员需要编写的配置信息。这里按照<configuration>子元素的功能不同，将配置分为了两个步骤：第 1 步配置了环境，第 2 步配置了 Mapper 的位置。关于上述代码中各个元素的详细配置信息将在下一章进行详细讲解，此案例中读者只需要按照上述代码配置即可。

小提示：

上述配置文件同样不需要读者完全手动编写。在 MyBatis 使用手册 mybatis-3.4.2.pdf 的 2.1.2 小节中，已经给出了配置模板（包含约束信息），读者只需要复制过来，依照自己的项目需求修改即可。

（7）在 src 目录下，创建一个 com.itheima.test 包，在该包下创建测试类 MybatisTest，并在类中编写测试方法 findCustomerByIdTest()，如文件 6-5 所示。

文件6-5　MybatisTest.java

```
1  package com.itheima.test;
```

```
import java.io.InputStream;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Test;
import com.itheima.po.Customer;
/**
 * 入门程序测试类
 */
public class MybatisTest {
    /**
     * 根据客户编号查询客户信息
     */
    @Test
    public void findCustomerByIdTest() throws Exception {
        // 1、读取配置文件
        String resource = "mybatis-config.xml";
        InputStream inputStream =
                Resources.getResourceAsStream(resource);
        // 2、根据配置文件构建 SqlSessionFactory
        SqlSessionFactory sqlSessionFactory =
                new SqlSessionFactoryBuilder().build(inputStream);
        // 3、通过 SqlSessionFactory 创建 SqlSession
        SqlSession sqlSession = sqlSessionFactory.openSession();
        // 4、SqlSession 执行映射文件中定义的 SQL，并返回映射结果
        Customer customer = sqlSession.selectOne("com.itheima.mapper"
                + ".CustomerMapper.findCustomerById", 1);
        // 打印输出结果
        System.out.println(customer.toString());
        // 5、关闭 SqlSession
        sqlSession.close();
    }
}
```

在文件 6-5 的 findCustomerByIdTest()方法中，首先通过输入流读取了配置文件，然后根据配置文件构建了 SqlSessionFactory 对象。接下来通过 SqlSessionFactory 对象又创建了 SqlSession 对象，并通过 SqlSession 对象的 selectOne()方法执行查询操作。selectOne()方法的第 1 个参数表示映射 SQL 的标识字符串，它由 CustomerMapper.xml 中<mapper>元素的 namespace 属性值+<select>元素的 id 属性值组成；第 2 个参数表示查询所需的参数，这里

查询的是客户表中 id 为 1 的客户。为了查看查询结果，这里使用了输出语句输出查询结果信息。最后，程序执行完毕时，关闭了 SqlSession。

使用 JUnit4 测试执行 findCustomerByIdTest()方法后，控制台的输出结果如图 6-9 所示。

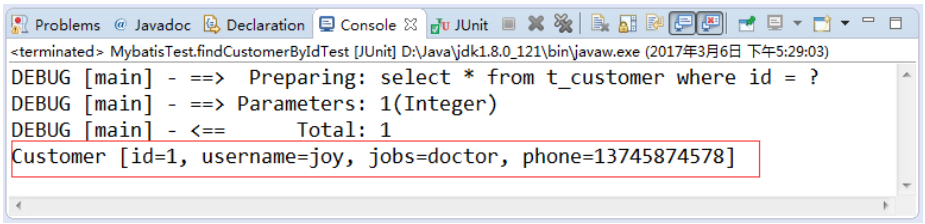

图6-9 运行结果

从图 6-9 可以看出，使用 MyBatis 框架已经成功查询出了 id 为 1 的客户信息。

2．根据客户名模糊查询客户信息

了解了如何使用 MyBatis 根据客户编号查询客户信息后，接下来讲解下如何根据客户的名称来模糊查询相关的客户信息。

模糊查询的实现非常简单，只需在映射文件中通过<select>元素编写相应的 SQL 语句，并通过 sqlSession 的查询方法执行该 SQL 即可。其具体实现步骤如下：

（1）在映射文件 CustomerMapper.xml 中，添加根据客户名模糊查询客户信息列表的 SQL 语句，具体实现代码如下。

```
<!--根据客户名模糊查询客户信息列表-->
<select id="findCustomerByName" parameterType="String"
    resultType="com.itheima.po.Customer">
    select * from t_customer where username like '%${value}%'
</select>
```

与根据客户编号查询相比，上述配置代码中的属性 id、parameterType 和 SQL 语句都发生相应变化。其中，SQL 语句中的“${}”用来表示拼接 SQL 的字符串，即不加解释的原样输出。“${value}”表示要拼接的是简单类型参数。

脚下留心：

在使用“${}”进行 SQL 字符串拼接时，无法防止 SQL 注入问题。所以想要既能实现模糊查询，又要防止 SQL 注入，可以对上述映射文件 CustomerMapper.xml 中模糊查询的 select 语句进行修改，使用 MySQL 中的 concat()函数进行字符串拼接。具体修改示例如下所示。

```
select * from t_customer where username like concat('%',#{value},'%')
```

（2）在测试类 MybatisTest 中，添加一个测试方法 findCustomerByNameTest()，其代码如下所示。

```
/**
 * 根据用户名称来模糊查询用户信息列表
 */
```

```
@Test
public void findCustomerByNameTest() throws Exception{
    // 1、读取配置文件
    String resource = "mybatis-config.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    // 2、根据配置文件构建 SqlSessionFactory
    SqlSessionFactory sqlSessionFactory =
              new SqlSessionFactoryBuilder().build(inputStream);
    // 3、通过 SqlSessionFactory 创建 SqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 4、SqlSession 执行映射文件中定义的 SQL，并返回映射结果
    List<Customer> customers = sqlSession.selectList("com.itheima.mapper"
                + ".CustomerMapper.findCustomerByName", "j");
    for (Customer customer : customers) {
        //打印输出结果集
        System.out.println(customer);
    }
    // 5、关闭 SqlSession
    sqlSession.close();
}
```

从上述代码可以看出，findCustomerByNameTest()方法只是在第 4 步时与根据客户编号查询的测试方法有所不同，其他步骤都一致。在第 4 步时，由于可能查询出的是多条数据，所以调用的是 SqlSession 的 selectList()方法来查询返回结果的集合对象，并使用 for 循环输出结果集对象。

使用 JUnit4 执行 findCustomerByNameTest()方法后，控制台的输出结果如图 6-10 所示。

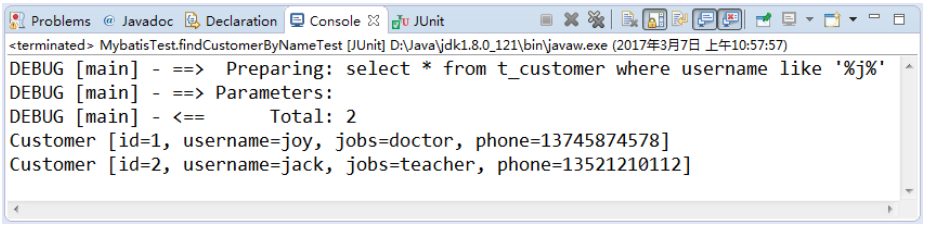

图6-10　运行结果

从图 6-10 可以看出，使用 MyBatis 框架已成功查询出了客户表中客户名称中带有“j”的 2 条客户信息。

至此，MyBatis 入门程序的查询功能就已经讲解完成。从上面两个查询方法中可以发现，MyBatis 的操作大致可分为以下几个步骤：

1) 读取配置文件。
2) 根据配置文件构建 SqlSessionFactory。
3) 通过 SqlSessionFactory 创建 SqlSession。

4) 使用 SqlSession 对象操作数据库（包括查询、添加、修改、删除，以及提交事务等）。
5) 关闭 SqlSession。

6.4.2 添加客户

在 MyBatis 的映射文件中，添加操作是通过<insert>元素来实现的。例如，向数据库中的 t_customer 表中插入一条数据可以通过如下配置来实现。

```
<!-- 添加客户信息 -->
<insert id="addCustomer" parameterType="com.itheima.po.Customer">
    insert into t_customer(username,jobs,phone)
    values(#{username},#{jobs},#{phone})
</insert>
```

在上述配置代码中，传入的参数是一个 Customer 类型，该类型的参数对象被传递到语句中时，#{username}会查找参数对象 Customer 的 username 属性（#{jobs}和#{phone}也是一样），并将其的属性值传入到 SQL 语句中。为了验证上述配置是否正确，下面编写一个测试方法来执行添加操作。

在测试类 MybatisTest 中，添加测试方法 addCustomerTest()，其代码如下所示。

```
/**
 * 添加客户
 */
@Test
public void addCustomerTest() throws Exception{
    // 1、读取配置文件
    String resource = "mybatis-config.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    // 2、根据配置文件构建 SqlSessionFactory
    SqlSessionFactory sqlSessionFactory =
              new SqlSessionFactoryBuilder().build(inputStream);
    // 3、通过 SqlSessionFactory 创建 SqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 4、SqlSession 执行添加操作
    // 4.1 创建 Customer 对象，并向对象中添加数据
    Customer customer = new Customer();
    customer.setUsername("rose");
    customer.setJobs("student");
    customer.setPhone("13333533092");
    // 4.2 执行 SqlSession 的插入方法，返回的是 SQL 语句影响的行数
```

```
        int rows = sqlSession.insert("com.itheima.mapper"
                + ".CustomerMapper.addCustomer", customer);
        // 4.3 通过返回结果判断插入操作是否执行成功
        if(rows > 0){
            System.out.println("您成功插入了"+rows+"条数据！");
        }else{
            System.out.println("执行插入操作失败!!!");
        }
        // 4.4 提交事务
        sqlSession.commit();
        // 5、关闭 SqlSession
        sqlSession.close();
    }
```

在上述代码的第 4 步操作中，首先创建了 Customer 对象，并向 Customer 对象中添加了属性值；然后通过 SqlSession 对象的 insert()方法执行插入操作，并通过该操作返回的数据来判断插入操作是否执行成功；最后通过 SqlSesseion 的 commit()方法提交了事务，并通过 close()方法关闭了 SqlSession。

使用 JUnit4 执行 addCustomerTest()方法后，控制台的输出结果如图 6-11 所示。

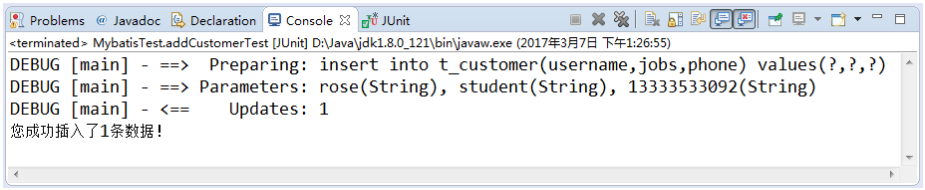

图6-11 运行结果

从图 6-11 可以看到，已经成功插入了 1 条数据。为了验证是否真的插入成功，此时查询数据库中的 t_customer 表，如图 6-12 所示。

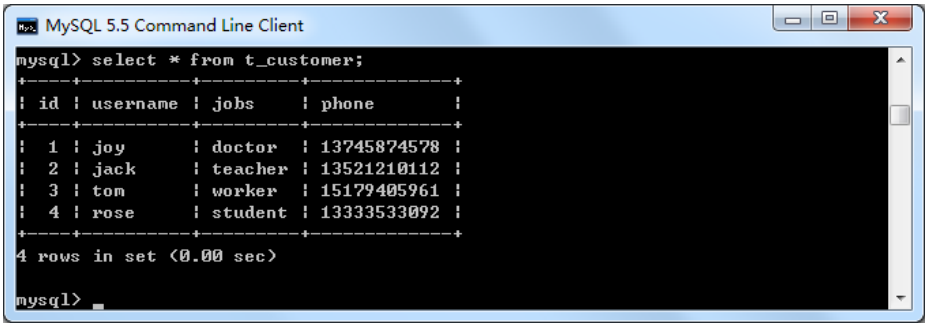

图6-12 t_customer 表

从图 6-12 可以看出，使用 MyBatis 框架已成功新增了一条 id 为 4 的客户信息。

6.4.3 更新客户

MyBatis 的更新操作在映射文件中是通过配置<update>元素来实现的。如果需要更新用户数据，可以通过如下代码配置来实现。

```
<!-- 更新客户信息 -->
<update id="updateCustomer" parameterType="com.itheima.po.Customer">
    update t_customer set
    username=#{username},jobs=#{jobs},phone=#{phone}
    where id=#{id}
</update>
```

与插入数据的配置相比，更新操作配置中的元素与 SQL 语句都发生了相应变化，但其属性名却没有变。为了验证配置是否正确，下面以上一小节中新插入的数据为例，来进行更新客户测试。

在测试类 MybatisTest 中，添加测试方法 updateCustomerTest()，将 id 为 4 的用户职业修改为“programmer”，电话修改为“13311111111”，其代码如下所示。

```
/**
 * 更新客户
 */
@Test
public void updateCustomerTest() throws Exception{
    // 1、读取配置文件
    String resource = "mybatis-config.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    // 2、根据配置文件构建 SqlSessionFactory
    SqlSessionFactory sqlSessionFactory =
             new SqlSessionFactoryBuilder().build(inputStream);
    // 3、通过 SqlSessionFactory 创建 SqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 4、SqlSession 执行更新操作
    // 4.1 创建 Customer 对象，对对象中的数据进行模拟更新
    Customer customer = new Customer();
    customer.setId(4);
    customer.setUsername("rose");
    customer.setJobs("programmer");
    customer.setPhone("13311111111");
    // 4.2 执行 SqlSession 的更新方法，返回的是 SQL 语句影响的行数
    int rows = sqlSession.update("com.itheima.mapper"
           + ".CustomerMapper.updateCustomer", customer);
```

```
        // 4.3 通过返回结果判断更新操作是否执行成功
        if(rows > 0){
            System.out.println("您成功修改了"+rows+"条数据！");
        }else{
            System.out.println("执行修改操作失败!!!");
        }
        // 4.4 提交事务
        sqlSession.commit();
        // 5、关闭 SqlSession
        sqlSession.close();
    }
```

与添加客户的方法相比，更新操作的代码增加了 id 属性值的设置，并调用 SqlSession 的 update()方法对 id 为 1 客户的职业和电话进行修改。

使用 JUnit4 执行 updateCustomerTest()方法后，控制台的输出结果如图 6-13 所示。

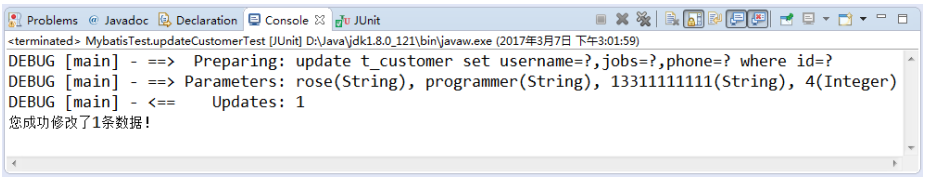

图6-13 运行结果

此时 t_customer 表中的数据如图 6-14 所示。

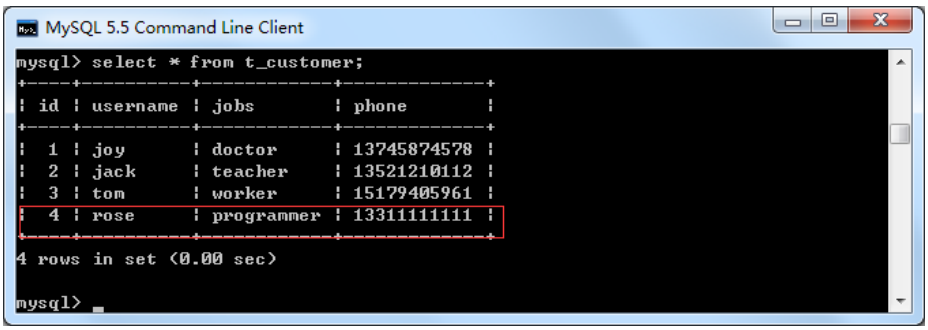

图6-14 t_customer 表

从图 6-14 可以看出，使用 MyBatis 框架已经成功更新了 id 为 4 的客户信息。

6.4.4 删除客户

MyBatis 的删除操作在映射文件中是通过配置<delete>元素来实现的。在映射文件 CustomerMapper.xml 中添加删除客户信息的 SQL 语句，其示例代码如下：

```
    <!-- 删除客户信息 -->
    <delete id="deleteCustomer" parameterType="Integer">
```

```
    delete from t_customer where id=#{id}
</delete>
```

从上述配置的 SQL 语句中可以看出，我们只需要传递一个 id 值就可以将数据表中相应的数据删除。

要测试删除操作的配置十分简单，只需使用 SqlSession 对象的 delete()方法传入需要删除数据的 id 值即可。在测试类 MybatisTest 中，添加测试方法 deleteCustomerTest()，该方法用于将 id 为 4 的客户信息删除，其代码如下所示。

```
/**
 * 删除客户
 */
@Test
public void deleteCustomerTest() throws Exception{
    // 1、读取配置文件
    String resource = "mybatis-config.xml";
    InputStream inputStream = Resources.getResourceAsStream(resource);
    // 2、根据配置文件构建 SqlSessionFactory
    SqlSessionFactory sqlSessionFactory =
           new SqlSessionFactoryBuilder().build(inputStream);
    // 3、通过 SqlSessionFactory 创建 SqlSession
    SqlSession sqlSession = sqlSessionFactory.openSession();
    // 4、SqlSession 执行删除操作
    // 4.1 执行 SqlSession 的删除方法，返回的是 SQL 语句影响的行数
    int rows = sqlSession.delete("com.itheima.mapper"
           + ".CustomerMapper.deleteCustomer", 4);
    // 4.2 通过返回结果判断删除操作是否执行成功
    if(rows > 0){
        System.out.println("您成功删除了"+rows+"条数据！");
    }else{
        System.out.println("执行删除操作失败!!!");
    }
    // 4.3 提交事务
    sqlSession.commit();
    // 5、关闭 SqlSession
    sqlSession.close();
}
```

使用 JUnit4 执行 deleteCustomerTest()方法后，控制台的输出结果如图 6-15 所示。

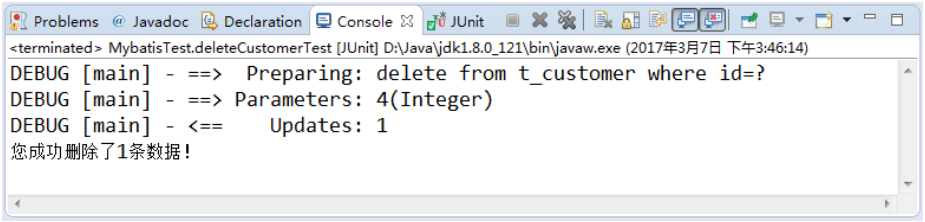

图6-15 运行结果

此时，再次查看表 t_customer 中的数据信息时，其结果如图 6-16 所示。

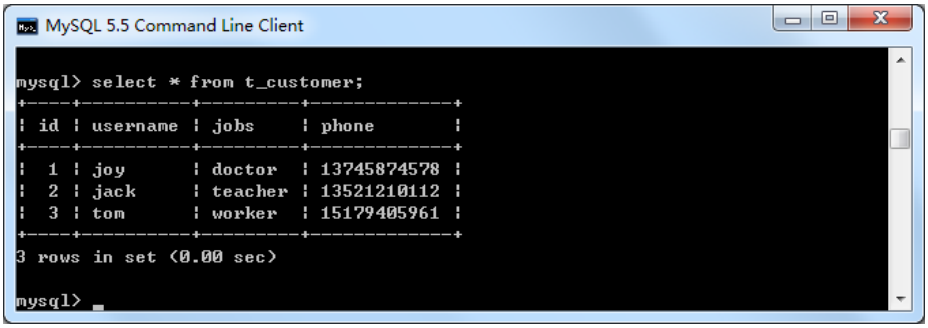

图6-16 t_customer 表

从图 6-16 可以看出，使用 MyBatis 框架已经成功删除了 id 为 4 的客户信息。

至此，MyBatis 入门程序的增删改查操作已经讲解完成。关于程序中映射文件和配置文件中的元素信息，将在下一章进行详细讲解，本章入门程序读者只需了解所使用的元素即可。

6.5 本章小结

本章首先对 MyBatis 框架的概念、特点和下载使用进行了讲解，然后对 MyBatis 框架的工作原理进行了流程分析，最后通过一个简单的增删改查案例来演示 MyBatis 框架的基本使用。通过本章的学习，读者可以了解 MyBatis 的概念和作用，熟悉 MyBatis 的工作原理，并能够使用 MyBatis 框架完成基本的数据库操作。

【思考题】

1、请简述 MyBatis 框架与 Hibernate 框架的区别。

2、请简述 MyBatis 的工作执行流程。