本章学习目标

- 原始方式开发 Dao
- 动态代理方式开发 Dao 接口（mapper 接口）
- 输入映射-简单类型和 JavaBean 类型
- 输入映射-包装 JavaBean 类型
- 输出映射-简单类型和 JavaBean 类型
- 输出映射-ResultMap 类型
- 动态 sql-if 标签
- 动态 sql-where 标签
- 动态 sql-sql 片段
- 动态 sql-foreach 标签
- 关联查询-一对一自定义 JavaBean
- 关联查询-一对一使用 ResultMap
- 关联查询-一对多查询
- MyBatis 逆向工程

1. 动态代理方式开发 Dao 接口

1.1. 原始方式开发 Dao

Dao 接口：

```
public interface CustomerDao {


    public void saveCustomer(Customer customer);


    public void updateCustomer(Customer customer);


    public void deleteCustomer(Integer id);

```

```
    public List<Customer> queryAllCustomer();


    public Customer queryCustomerById(Integer id);


    public List<Customer> queryCustomeyByName(String name);
}
```

Dao 实现：

```
public class CustomerDaoImpl implements CustomerDao {


    @Override
    public void saveCustomer(Customer customer) {
        SqlSession sqlSession = null;
        try {
            sqlSession = SessionUtils.getSession();
            sqlSession.insert("insertCustomer", customer);
            sqlSession.commit();
        } catch (Exception e) {
            e.printStackTrace();
            sqlSession.rollback();
        } finally{
            sqlSession.close();
        }
    }


    @Override
    public void updateCustomer(Customer customer) {
        SqlSession sqlSession = null;
        try {
```

```
            sqlSession = SessionUtils.getSession();
            sqlSession.update("updateCustomer", customer);
            sqlSession.commit();
        } catch (Exception e) {
            e.printStackTrace();
            sqlSession.rollback();
        } finally{
            sqlSession.close();
        }
    }

    @Override
    public void deleteCustomer(Integer id) {
        SqlSession sqlSession = null;
        try {
            sqlSession = SessionUtils.getSession();
            sqlSession.delete("deleteCustomer",id);
            sqlSession.commit();
        } catch (Exception e) {
            e.printStackTrace();
            sqlSession.rollback();
        } finally{
            sqlSession.close();
        }
    }

    @Override
    public List<Customer> queryAllCustomer() {
        SqlSession sqlSession = null;
```

```
        try {
            sqlSession = SessionUtils.getSession();
            return sqlSession.selectList("queryAllCustomer");
        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            sqlSession.close();
        }
        return null;
    }

    @Override
    public Customer queryCustomerById(Integer id) {
        SqlSession sqlSession = null;
        try {
            sqlSession = SessionUtils.getSession();
            return sqlSession.selectOne("queryCustomerById",id);
        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            sqlSession.close();
        }
        return null;
    }

    @Override
    public List<Customer> queryCustomeyByName(String name) {
        SqlSession sqlSession = null;
        try {
```

```
            sqlSession = SessionUtils.getSession();
            return sqlSession.selectList("queryCustomerByName",name);
        } catch (Exception e) {
            e.printStackTrace();
        } finally{
            sqlSession.close();
        }
        return null;
    }

}
```

Customer.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">
<!-- 该文件存放 CRUD 的 sql 语句 -->
<mapper namespace="test">
    <!-- 添加 -->
    <insert id="insertCustomer" parameterType="customer">
        INSERT INTO t_customer(NAME,gender,telephone)
VALUES(#{name},#{gender},#{telephone})
    </insert>

    <!-- 修改 -->
    <update id="updateCustomer" parameterType="customer">
        UPDATE t_customer SET NAME = #{name} WHERE id = #{id}
    </update>
```

```
    <!-- 查询所有数据 -->
    <select id="queryAllCustomer" resultType="customer">
        SELECT * FROM t_customer
    </select>

    <!-- 根据 id 查询 -->
    <select id="queryCustomerById" parameterType="Integer"
resultType="customer">
        SELECT * FROM t_customer WHERE id=#{value}
    </select>

    <!-- 根据 name 模糊查询 -->
    <select id="queryCustomerByName"  parameterType="string"
resultType="customer">
        SELECT * FROM t_customer WHERE NAME LIKE #{value}
    </select>

    <!-- 删除 -->
    <delete id="deleteCustomer" parameterType="int">
        DELETE FROM t_customer WHERE id=#{value}
    </delete>
</mapper>
```

测试类：

```
public class Demo1 {

    @Test
    public void test1(){
        Customer c = new Customer();
```

```
        c.setName("陈六 222");
        c.setGender("男");
        c.setTelephone("13244445555");

        CustomerDao dao = new CustomerDaoImpl();
        dao.saveCustomer(c);
    }

    @Test
    public void test2(){
        Customer c = new Customer();
        c.setId(1);
        c.setName("李四");

        CustomerDao dao = new CustomerDaoImpl();
        dao.updateCustomer(c);
    }

    @Test
    public void test3(){
        CustomerDao dao = new CustomerDaoImpl();
        dao.deleteCustomer(8);
    }

    @Test
    public void test4(){
        CustomerDao dao = new CustomerDaoImpl();
        List<Customer> list = dao.queryAllCustomer();
        for (Customer customer : list) {
```

```
            System.out.println(customer);
        }
    }

    @Test
    public void test5(){
        CustomerDao dao = new CustomerDaoImpl();
        Customer customer = dao.queryCustomerById(1);
        System.out.println(customer);
    }

    @Test
    public void test6(){
        CustomerDao dao = new CustomerDaoImpl();
        List<Customer> list = dao.queryCustomeyByName("%陈%");
        for (Customer customer : list) {
            System.out.println(customer);
        }
    }
}
```

1.2. 动态代理方式开发 Dao 层（推荐使用）

好处：无需再去编写 Dao 层的实现类

Dao 接口：

```
public interface CustomerDao {

    public void saveCustomer(Customer customer);
```

```
    public void updateCustomer(Customer customer);

    public void deleteCustomer(Integer id);

    public List<Customer> queryAllCustomer();

    public Customer queryCustomerById(Integer id);

    public List<Customer> queryCustomeyByName(String name);
}
```

Customr.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">
<!--
    如果是动态代理方式
      1）namespace 必须和 Dao 接口的路径保持一致
      2）Dao 接口里面的方法和 sql 语句的 ID
      3) Dao 接口的方法的参数和返回值类型 和 映射文件的 parameterType 和
resultType
 -->
<mapper namespace="cn.sm1234.dao.CustomerDao">
    <!-- 添加 -->
    <insert id="saveCustomer" parameterType="customer">
        INSERT INTO t_customer(NAME,gender,telephone)
VALUES(#{name},#{gender},#{telephone})
    </insert>
```

```
    <!-- 修改 -->
    <update id="updateCustomer" parameterType="customer">
        UPDATE t_customer SET NAME = #{name} WHERE id = #{id}
    </update>

    <!-- 查询所有数据 -->
    <select id="queryAllCustomer" resultType="customer">
        SELECT * FROM t_customer
    </select>

    <!-- 根据 id 查询 -->
    <select id="queryCustomerById" parameterType="Integer"
resultType="customer">
        SELECT * FROM t_customer WHERE id=#{value}
    </select>

    <!-- 根据 name 模糊查询 -->
    <select id="queryCustomerByName"  parameterType="string"
resultType="customer">
        SELECT * FROM t_customer WHERE NAME LIKE #{value}
    </select>

    <!-- 删除 -->
    <delete id="deleteCustomer" parameterType="int">
        DELETE FROM t_customer WHERE id=#{value}
    </delete>
</mapper>
```

测试代码：

```
public class Demo2 {
```

```
@Test
public void test1(){
    Customer c = new Customer();
    c.setName("陈六 333");
    c.setGender("男");
    c.setTelephone("13244445555");

    SqlSession sqlSession = SessionUtils.getSession();
    //getMapper(): 返回指定接口的动态代理的实现类对象
    CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
    dao.saveCustomer(c);
    sqlSession.commit();
    sqlSession.close();
}

@Test
public void test2(){
    Customer c = new Customer();
    c.setId(1);
    c.setName("李四 222");

    SqlSession sqlSession = SessionUtils.getSession();
    //getMapper(): 返回指定接口的动态代理的实现类对象
    CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
    dao.updateCustomer(c);
    sqlSession.commit();
    sqlSession.close();
}
```

```
@Test
public void test3(){
    SqlSession sqlSession = SessionUtils.getSession();
    CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
    dao.deleteCustomer(15);
    sqlSession.commit();
    sqlSession.close();
}

@Test
public void test4(){
    SqlSession sqlSession = SessionUtils.getSession();
    CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
    List<Customer> list = dao.queryAllCustomer();
    for (Customer customer : list) {
        System.out.println(customer);
    }
}

@Test
public void test5(){
    SqlSession sqlSession = SessionUtils.getSession();
    CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
    Customer customer = dao.queryCustomerById(1);
    System.out.println(customer);
}

@Test
```

```
    public void test6(){
        SqlSession sqlSession = SessionUtils.getSession();
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
        List<Customer> list = dao.queryCustomerByName("%陈%");
        for (Customer customer : list) {
            System.out.println(customer);
        }
    }
}
```

2. 输入映射

输入映射支持的类型：

1）基本的类型，int，String，double 等（*）

2）JavaBean 类型（*）

3）包装 JavaBean 类型（对象里面包含另一个对象）

2.1. 基本类型

```
<insert id="testParameterType" parameterType="string">
        INSERT INTO t_customer(NAME) VALUES(#{name})
    </insert>
```

```
//输入映射
    public void testParameterType(String name);
}
```

```
/**
     * 输入映射-基本类型
     */
    @Test
    public void test1(){
```

```
        SqlSession sqlSession = SessionUtils.getSession();
        //getMapper(): 返回指定接口的动态代理的实现类对象
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
        dao.testParameterType("张三");
        sqlSession.commit();
        sqlSession.close();
    }
```

2.2. JavaBean 类型

```
<insert id="testParameterType" parameterType="customer">
        INSERT INTO t_customer(NAME,gender,telephone)
VALUES(#{name},#{gender},#{telephone})
    </insert>
```

```
public void testParameterType(Customer c);
```

```
/**
     * 输入映射-基本类型
     */
    @Test
    public void test1(){
        SqlSession sqlSession = SessionUtils.getSession();
        //getMapper(): 返回指定接口的动态代理的实现类对象
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
        Customer c = new Customer();
        c.setName("张三 2222");
        c.setGender("男");
        c.setTelephone("13211112222");
        dao.testParameterType(c);
        sqlSession.commit();
        sqlSession.close();
```

```
    }
```

2.3. 包装 JavaBean 类型

一个对象里面包含另一个对象

```
<insert id="testParameterType"
parameterType="cn.sm1234.domain.CustomerVo">
        INSERT INTO t_customer(NAME,gender,telephone)
VALUES(#{customer.name},#{customer.gender},#{customer.telephone})
    </insert>
```

```
public void testParameterType(CustomerVo c);
```

```
public class CustomerVo {
    private Customer customer;
    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

}
```

```
@Test
    public void test1(){
        SqlSession sqlSession = SessionUtils.getSession();
        //getMapper(): 返回指定接口的动态代理的实现类对象
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);

        CustomerVo vo = new CustomerVo();
        Customer c = new Customer();
```

```
        c.setName("张三 333");
        c.setGender("男");
        c.setTelephone("13211112222");
        vo.setCustomer(c);

        dao.testParameterType(vo);

        sqlSession.commit();
        sqlSession.close();
    }
```

3. 输出映射

3.1. 基本类型

```
<!-- 统计记录数 -->
    <select id="queryTotalCount" resultType="Long">
        SELECT COUNT(*) FROM t_customer
    </select>
```

```
public Long queryTotalCount();
```

```
/**
     * 输出映射
     */
    @Test
    public void test2(){
        SqlSession sqlSession = SessionUtils.getSession();
        //getMapper(): 返回指定接口的动态代理的实现类对象
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);
```

```
        Long count = dao.queryTotalCount();
        System.out.println(count);

        sqlSession.commit();
        sqlSession.close();
    }
```

3.2. JavaBean 类型（*）

```
<select id="queryCustomer" parameterType="int" resultType="customer">
        SELECT * FROM t_customer WHERE id=#{value}
    </select>
```

```
public Customer queryCustomer(Integer id);
```

```
    /**
     * 输出映射
     */
    @Test
    public void test2(){
        SqlSession sqlSession = SessionUtils.getSession();
        //getMapper(): 返回指定接口的动态代理的实现类对象
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);

        /*Long count = dao.queryTotalCount();
        System.out.println(count);*/

        Customer c = dao.queryCustomer(1);
        System.out.println(c);
```

```
        sqlSession.commit();
        sqlSession.close();
    }
```

3.3. ResultMap 类型

用于解决表的字段名称和实体类的属性名称不一致的情况。

```
<!-- 定义 ResultMap -->
    <resultMap type="cn.sm1234.domain.CustomerRM" id="customerResultMap">
        <!-- id:映射主键 -->
        <id column="id" property="custId"/>
        <result column="name" property="custName"/>
        <result column="gender" property="custGender"/>
        <result column="telephone" property="custTelephone"/>
    </resultMap>

    <select id="queryCustomerResultMap" parameterType="int"
resultMap="customerResultMap">
        SELECT * FROM t_customer WHERE id=#{value}
    </select>
```

```
public CustomerRM queryCustomerResultMap(Integer id);
```

```
/**
     * 输出映射
     */
    @Test
    public void test2(){
        SqlSession sqlSession = SessionUtils.getSession();
```

```
        //getMapper(): 返回指定接口的动态代理的实现类对象
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);

        /*Long count = dao.queryTotalCount();
        System.out.println(count);*/

    /*  Customer c = dao.queryCustomer(1);
        System.out.println(c);*/

        CustomerRM c = dao.queryCustomerResultMap(1);
        System.out.println(c);

        sqlSession.commit();
        sqlSession.close();
    }
```

4. 动态 sql

4.1. if 标签

```
<select id="queryByNameAndTelephone" parameterType="customer"
resultType="customer">
        SELECT * FROM t_customer
        WHERE 1=1
        <if test="name!=null and name!='' ">
            AND NAME LIKE #{name}
        </if>
```

```
        <if test="telephone!=null and telephone!='' ">
            AND telephone LIKE #{telephone}
        </if>
    </select>
```

```
public List<Customer> queryByNameAndTelephone(Customer customer);
```

```
/**
     * if 标签
     */
    @Test
    public void test1(){
        SqlSession sqlSession = SessionUtils.getSession();
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);

        Customer c = new Customer();
        //c.setName("%张%");
        c.setTelephone("%33%");

        List<Customer> list = dao.queryByNameAndTelephone(c);
        for (Customer customer : list) {
            System.out.println(customer);
        }

        sqlSession.commit();
        sqlSession.close();
    }
```

4.2. where 标签

```
<select id="queryByNameAndTelephone" parameterType="customer"
```

```
resultType="customer">
        SELECT * FROM t_customer
        <!-- <where>: where 条件，自动把第一个条件的 and 去掉 -->
        <where>
            <if test="name!=null and name!='' ">
                AND NAME LIKE #{name}
            </if>
            <if test="telephone!=null and telephone!='' ">
                AND telephone LIKE #{telephone}
            </if>
        </where>
    </select>
```

4.3. sql 片段

作用：把相同的 sql 片段抽取出来。

```
<!-- sql 片段 -->
    <sql id="customerField">
        id,name,gender,telephone
    </sql>
```

```
<select id="queryByNameAndTelephone" parameterType="customer"
resultType="customer">
        SELECT
        <include refid="customerField"/>
         FROM t_customer
        <!-- <where>: where 条件，自动把第一个条件的 and 去掉 -->
        <where>
```

```
            <if test="name!=null and name!='' ">
                AND NAME LIKE #{name}
            </if>
            <if test="telephone!=null and telephone!='' ">
                AND telephone LIKE #{telephone}
            </if>
        </where>
    </select>
```

4.4. foreach 标签

```
<delete id="deleteCustomerByIn" parameterType="customer">
        DELETE FROM t_customer WHERE
        <!--
            collection: 需要遍历的属性
            item: 遍历的变量
            open: 循环前面的 sql 语句
            close:  循环后面的 sql 语句
            separator: 分隔符

            id IN(13,16,12)
         -->
        <foreach collection="ids" item="id" open="id IN(" close=")"
separator=",">
            #{id}
        </foreach>
    </delete>
```

```
public void deleteCustomerByIn(Customer customer);
```

```
/**
```

```
     * foreach 标签
     */
    @Test
    public void test2(){
        SqlSession sqlSession = SessionUtils.getSession();
        CustomerDao dao = sqlSession.getMapper(CustomerDao.class);

        Customer c = new Customer();
        Integer[] ids = {12,13,16};
        c.setIds(ids);

        dao.deleteCustomerByIn(c);

        sqlSession.commit();
        sqlSession.close();
    }
```

5. 关系查询-一对一查询

用户和订单的需求

通过查询订单，查询用户，就是一对一查询。

5.1. 自定义 JavaBean（*）

```
<select id="queryOrderUser" resultType="cn.sm1234.domain.OrderUser">
        SELECT o.id,o.user_id,u.name,productname
            FROM t_order o
            LEFT JOIN t_user u ON o.user_id=u.id
```

```
    </select>
```

```
public interface OrderDao {


    /**
     * 查询订单，查询用户查询
     */
    public List<OrderUser> queryOrderUser();
}
```

```
public class OrderUser extends Order {
    private String name;


    public String getName() {
        return name;
    }


    public void setName(String name) {
        this.name = name;
    }


}
```

```
@Test
    public void test1(){
        SqlSession sqlSession = SessionUtils.getSession();
        OrderDao dao = sqlSession.getMapper(OrderDao.class);


        List<OrderUser> list = dao.queryOrderUser();
        for (OrderUser ou : list) {
            System.out.println(ou.getName()+"-"+ou.getProductname());
        }
```

```
        sqlSession.commit();
        sqlSession.close();
    }
```

5.2. ResultMap 封装

```
<resultMap type="cn.sm1234.domain.Order" id="OrderUserResultMap">
        <id column="id" property="id"/>
        <result column="productname" property="productname"/>
        <result column="orderno" property="orderno"/>
        <!-- 关联属性
            property: 关联属性名称
            javaType: 类型
        -->
        <association property="user" javaType="cn.sm1234.domain.User">
            <id column="user_id" property="id"/>
            <result column="name" property="name"/>
            <result column="password" property="password"/>
        </association>
    </resultMap>

    <select id="queryOrderUserResultMap" resultMap="OrderUserResultMap">
        SELECT o.id,o.user_id,u.name,productname
            FROM t_order o
            LEFT JOIN t_user u ON o.user_id=u.id
    </select>
```

```
public class Order {
    private Integer id;
```

```
    private String productname;
    private String orderno;
    private Integer userId;

    private User user;
```

```
/**
     * 查询订单，查询用户(ResultMap)
     * @return
     */
    public List<Order> queryOrderUserResultMap();
```

```
@Test
    public void test2(){
        SqlSession sqlSession = SessionUtils.getSession();
        OrderDao dao = sqlSession.getMapper(OrderDao.class);

        List<Order> list = dao.queryOrderUserResultMap();
        for (Order o: list) {

    System.out.println(o.getProductname()+"-"+o.getUser().getName());
        }

        sqlSession.commit();
        sqlSession.close();
    }
```

6. 关联查询-一对多查询

```
<resultMap type="cn.sm1234.domain.User" id="UserOrderResultMap">
        <id column="id" property="id"/>
        <result column="name" property="name"/>
        <!--
            collection:封装集合
               property：关联属性名
         -->
        <collection property="orders" javaType="cn.sm1234.domain.Order">
            <id column="order_id" property="id"/>
            <result column="orderno" property="orderno"/>
            <result column="productname" property="productname"/>
        </collection>
    </resultMap>


    <select id="queryUserOrder" resultMap="UserOrderResultMap">
        SELECT u.id,u.name,o.orderno,o.id order_id,o.productname FROM
t_user u LEFT JOIN t_order o ON o.user_id=u.id
    </select>
```

```
public class User {
    private Integer id;
    private String name;
    private String password;

    private List<Order> orders = new ArrayList<Order>();
```

```
public interface UserDao {
```

```
    public List<User> queryUserOrder();
}
```

```
@Test
    public void test3(){
        SqlSession sqlSession = SessionUtils.getSession();
        UserDao dao = sqlSession.getMapper(UserDao.class);

        List<User> userList = dao.queryUserOrder();
        for (User user : userList) {
            System.out.println("用户: "+user.getId()+"-"+user.getName());
            for (Order o:user.getOrders()) {
                System.out.println("订单信息:
"+o.getOrderno()+"-"+o.getProductname());
            }
        }

        sqlSession.commit();
        sqlSession.close();
    }
```