

综合项目

1 学习目标

1. 利用二阶段所学知识,进行后台接口的开发
2. 熟练掌握动态SQL技术

2 前期准备

2.1 创建SpringBoot工程

①在 JSDSecondStage 项目下创建SpringBoot模块,名为 SmartBlog

- 项目名: SmartBlog
- Group名: cn.tedu

②在 pom.xml 中,统一修改项目版本为**2.5.4**

2.2 数据表准备

- 执行 sblog.sql ,生成对应sblog数据库

2.3 导入相关依赖

- 在pom.xml中的dependencies标签中,添加如下依赖(修改pom.xml文件,需要手动点击刷新按钮,使配置生效)

```
1 <!--添加MySQL数据库驱动依赖-->
2 <dependency>
3     <groupId>mysql</groupId>
4     <artifactId>mysql-connector-java</artifactId>
5     <scope>runtime</scope>
6 </dependency>
7 <!--添加mybatis启动依赖-->
8 <dependency>
9     <groupId>org.mybatis.spring.boot</groupId>
10    <artifactId>mybatis-spring-boot-starter</artifactId>
11    <version>2.2.0</version>
12 </dependency>
```

2.4 MyBatis框架简易配置

- 打开项目的application.yml文件,进行基础配置

```
1 #设置连接数据库的url、username、password,这三部分不能省略
2 spring:
3     datasource:
4         url: jdbc:mysql://localhost:3306/sblog?
serverTimezone=Asia/Shanghai&characterEncoding=utf8
5         username: root
6         password: root
7 #MyBatis开启驼峰映射,并且扫描xml文件
8 mybatis:
9     configuration:
10         map-underscore-to-camel-case: true
11         mapper-locations: classpath:/mapper/*.xml
```

```
12 #日志设置
13 logging:
14     level:
15         cn:
16             tedu: debug
```

2.5 测试代码实现

- 在 `src/test/java` 目录中添加测试类**MyBatisTest**，对MyBatis框架整合进行基本测试，代码如下：

```
1 package cn.tedu;
2
3 import org.apache.ibatis.session.SqlSession;
4 import org.junit.jupiter.api.Test;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.boot.test.context.SpringBootTest;
7
8 import java.sql.Connection;
9
10 /**
11  * 对mybatis框架整合进行基本测试
12  */
13 @SpringBootTest
14 public class MyBatisTests {
15     @Autowired
16     private SqlSession sqlSession;
17     @Test
18     public void testGetConnection() {
19         Connection conn=sqlSession.getConnection();
20         System.out.println("connection="+conn);
21     }
22 }
```

3 动态SQL

3.1 什么是动态SQL

- 动态 SQL是MyBatis强大特性之一。极大的简化我们拼装SQL的操作。
- SQL的内容是变化的, 可以根据条件获取到不同的sql语句.主要是where部分发生变化。
- 动态sql的实现, 使用的是mybatis提供的标签

3.2 常用动态SQL标签

3.2.1 where标签

- where和if一般结合使用
- 若where标签中的 if 条件都不满足，则where标签没有任何功能，即不会添加where关键字
- 若where标签中的 if 条件满足，则where标签会自动添加where关键字，并将条件最前方多余的and去掉
- 注意：where标签不能去掉条件最后多余的and

3.2.2 if标签

- if标签通常用于 WHERE 语句、UPDATE 语句、INSERT 语句中
- 语法格式: <if test="boolean判断结果"> SQL代码 </if>
- 如果布尔值是真的话,就会将这个SQL代码片段加入到SQL语句中;只有if标签,没有 else。可以有多个<if>标签并列。

3.2.3 set标签

- set和if一般结合使用
- 使用set标签,相当于update语句中的set字段
- 一般用于修改语句,如果传递的参数为null,那么就不会修改该列的值
- 会智能的去掉最后一个语句后面的逗号

3.2.4 foreach标签

- foreach适用于批量添加、删除和查询记录
- 语法格式:

```
1 <foreach collection="集合类型" open="开始的字符" close="结束的字符" item="集合中的成员" separator="集合成员之间的分割符">
2     #{item的值}
3 </foreach>
```

- **collection**:表示循环的对象是数组还是list集合。
如果dao方法的形参是数组, collection="array";如果dao方法形参是list, collection="list";
但是如果传入的是多个参数,则需要在collection中写传入的集合的参数名或者数据的参数名
- **open**:循环开始的字符。
- **close**:循环结束的字符。
- **item**:集合成员, 自定义的变量。
- **separator**:集合成员之间的分隔符。
- **#{item的值}**:获取集合成员的值。

3.2.5 choose(when、otherwise)标签

- choose 主要用于分支判断,类似于 java 中带了 break的 switch...case, 只会满足所有分支中的一个
- 语法格式:

```
1 <choose>
2     <when test="">通过test表达式拼接SQL</when>
3     <when test="">通过test表达式拼接SQL</when>
4     <otherwise>当when都不符合条件,就会选择otherwise拼接SQL</otherwise>
5 </choose>
```

3.2.6 sql标签

- sql标签是用于抽取可重用的 SQL 片段，将相同的，使用频繁的 SQL 片段抽取出来，单独定义，方便多次引用。
- sql抽取：经常将要查询的列名，或者插入用的列名抽取出来方便引用。
- include来引用已经抽取的sql。

4 整合MyBatis完成标签业务操作

4.1 相关设计

4.1.1 标签表设计

- `tb_Tag` 表示标签表,其中设计的字段内容如下

Field	Type	Comment
id	bigint(20) unsigned	
name	varchar(50)	标签名
remark	varchar(100)	评论
created_time	datetime	注册时间
modified_time	datetime	修改时间

4.1.2 POJO对象设计

- 创建 `pojo/Tag` 类，通过此类封装标签相关信息，对象的具体内容如下：

```
1 package cn.tedu.pojo;
2
3 import java.util.Date;
4
5 public class Tag {
6     private Long id;
7     private String name;
8     private String remark;
9     private Date createTime;
10    private Date modifiedTime;
11
12    public Long getId() {
13        return id;
14    }
15
16    public void setId(Long id) {
17        this.id = id;
18    }
19
20    public String getName() {
21        return name;
22    }
23
24    public void setName(String name) {
25        this.name = name;
26    }
27
28    public String getRemark() {
```

```

29         return remark;
30     }
31
32     public void setRemark(String remark) {
33         this.remark = remark;
34     }
35
36     public Date getCreateTime() {
37         return createTime;
38     }
39
40     public void setCreateTime(Date createTime) {
41         this.createTime = createTime;
42     }
43
44     public Date getModifiedTime() {
45         return modifiedTime;
46     }
47
48     public void setModifiedTime(Date modifiedTime) {
49         this.modifiedTime = modifiedTime;
50     }
51
52     @Override
53     public String toString() {
54         return "Tags{" +
55             "id=" + id +
56             ", name='" + name + '\'' +
57             ", remark='" + remark + '\'' +
58             ", createTime=" + createTime +
59             ", modifiedTime=" + modifiedTime +
60             '}';
61     }
62 }

```

4.1.3 DAO接口设计

- 在dao包，基于MyBatis规范设计用户数据访问接口TagDao
- 并且在接口上用**@Mapper**注解修饰,该注解由MyBatis框架提供，用于描述数据层接口，告诉系统底层为此接口创建其实现类，在实现类中定义数据访问逻辑，执行与数据库的会话(交互)。

```

1 package cn.tedu.dao;
2
3 import org.apache.ibatis.annotations.Mapper;
4
5 @Mapper
6 public interface TagDao {
7 }

```

4.1.4 DAO单元测试类

- 在test/java/cn/tedu目录中添加测试类**TagDaoTest**
- 使用Spring注入TagDao实例

```

1 package cn.tedu;
2
3 import cn.tedu.dao.TagDao;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.boot.test.context.SpringBootTest;
6
7 @SpringBootTest
8 public class TagDaoTests {
9     @Autowired
10     private TagDao tagDao;
11 }

```

4.2 添加新的标签信息

4.2.1 DAO接口

- 在TagDao接口中,添加**insert**接口方法
- 参数: Tag Tag -- 新增的标签信息
- 返回值: int -- 返回插入的记录数

```

1 @Mapper
2 public interface TagDao {
3     @Insert("INSERT INTO tb_Tag(name,remark,created_time,modified_time)
4     VALUES (#{name},#{remark},#{createTime},#{modifiedTime})")
5     int insert(Tag tag);
6 }

```

4.2.2 DAO单元测试类实现

- 定义**insert**单元测试方法

```

1 @Test
2 void insert() {
3     Tag tag = new Tag();
4     tag.setName("mysql");
5     tag.setRemark("mysql..");
6     tag.setCreateTime(new Date());
7     tag.setModifiedTime(new Date());
8     tagDao.insert(tag);
9 }

```

4.2.3 执行测试

- 执行**insert**方法,执行成功,查看tb_Tag表中的记录是否发生变化

4.3 查询标签信息

4.3.1 DAO接口

- 在TagDao接口中,添加**list**接口方法
- 返回值: List<Tag> -- 返回多条标签信息

```

1 @Select("select * from tb_Tag")
2 List<Tag> list();

```

4.3.2 DAO单元测试类实现

- 定义list单元测试方法

```
1 @Test
2 public void testList() {
3     List<Tag> list = tagDao.list();
4     for (Tag tag : list) {
5         System.out.println(tag);
6     }
7 }
```

4.3.3 执行测试

- 执行testList方法,执行成功

4.4 更新标签信息

4.4.1 DAO接口

- 在TagDao接口中,添加update接口方法
- 参数: Tag tag -- 更新的标签信息
- 返回值: int -- 返回更新的记录数

```
1 @Update("UPDATE tb_Tag SET name=#{name},remark=#{remark},modified_time=#{modifiedTime} WHERE id=#{id}")
2 int update(Tag tag);
```

4.4.2 DAO单元测试类实现

- 定义update单元测试方法

```
1 @Test
2 public void update() {
3     Tag tag = new Tag();
4     tag.setId(1L);
5     tag.setName("Chinese");
6     tag.setRemark("这门语言简直牛炸天了");
7     tag.setModifiedTime(new Date());
8     System.out.println(rows>0 ? "修改成功!" : "修改失败!");
9 }
```

4.4.3 执行测试

- 执行update方法,执行成功

4.5 删除标签信息

4.5.1 Dao接口

- 在TagDao接口中,添加list接口方法
- 参数: Long id -- 删除的标签id
- 返回值: int -- 返回删除的记录数

```
1 @Delete("DELETE FROM tb_Tag where id=#{id}")
2 int deleteById(Long id);
```

4.5.2 Dao单元测试类实现

- 定义deleteById单元测试方法

```
1  @Test
2  public void deleteById() {
3      int rows = TagDao.deleteById(1L);
4      System.out.println(rows > 0 ? "删除成功!" : "删除失败!");
5  }
```

4.5.3 执行测试

- 执行deleteById方法,执行成功

5 整合MyBatis完成用户数据操作

5.1 相关设计

5.1.1 用户表设计

- tb_users表示用户表,其中设计的字段内容如下

Field	Type	Comment
id	int(11)	
username	varchar(50)	用户名
nickname	varchar(50)	昵称
password	varchar(255)	密码
mobile	varchar(20)	电话号码
status	tinyint(4)	账号是否被锁住, 0→禁用, 1→启用
created_time	datetime	注册时间
modified_time	datetime	修改时间

5.1.2 POJO对象设计

- 创建pojo包, 设计一个User对象, 通过此对象封装用户相关信息, 对象的具体内容如下:

```
1  package cn.tedu.pojo;
2
3  import java.util.Date;
4
5  public class User {
6      private Integer id;
7      private String username;
8      private String nickname;
9      private String password;
10     private String mobile;
11     private Integer status;
12     private Date createTime;
13     private Date modifiedTime;
14
15     public Integer getId() {
16         return id;
17     }
18 }
```



```
17     }
18
19     public void setId(Integer id) {
20         this.id = id;
21     }
22
23     public String getUsername() {
24         return username;
25     }
26
27     public void setUsername(String username) {
28         this.username = username;
29     }
30
31     public String getNickname() {
32         return nickname;
33     }
34
35     public void setNickname(String nickname) {
36         this.nickname = nickname;
37     }
38
39     public String getPassword() {
40         return password;
41     }
42
43     public void setPassword(String password) {
44         this.password = password;
45     }
46
47     public String getMobile() {
48         return mobile;
49     }
50
51     public void setMobile(String mobile) {
52         this.mobile = mobile;
53     }
54
55     public Integer getStatus() {
56         return status;
57     }
58
59     public void setStatus(Integer status) {
60         this.status = status;
61     }
62
63     public Date getCreatedTime() {
64         return createdTime;
65     }
66
67     public void setCreatedTime(Date createdTime) {
68         this.createdTime = createdTime;
69     }
70
71     public Date getModifiedTime() {
72         return modifiedTime;
73     }
74
75     public void setModifiedTime(Date modifiedTime) {
76         this.modifiedTime = modifiedTime;
77     }
78
79     @Override
```

```

80     public String toString() {
81         return "User{" +
82             "id=" + id +
83             ", username='" + username + '\'' +
84             ", nickname='" + nickname + '\'' +
85             ", password='" + password + '\'' +
86             ", mobile='" + mobile + '\'' +
87             ", status=" + status +
88             ", createTime=" + createTime +
89             ", modifiedTime=" + modifiedTime +
90             '}';
91     }
92 }

```

5.1.3 DAO接口设计

- 创建dao包，基于MyBatis规范设计用户数据访问接口 UserDao
- 并且在接口上用 **@Mapper** 注解修饰,该注解由MyBatis框架提供，用于描述数据层接口，告诉系统底层为此接口创建其实现类，在实现类中定义数据访问逻辑，执行与数据库的会话(交互)。

```

1  package cn.tedu.dao;
2
3  import org.apache.ibatis.annotations.Mapper;
4
5  @Mapper
6  public interface UserDao {
7  }

```

5.1.4 DAO接口映射文件

- 在项目的resources目录下创建mapper目录，并在目录下创建UserMapper.xml文件
- 并且在指定要映射的 **UserDao** 接口的全路径

```

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="cn.tedu.dao.UserDao">
6
7  </mapper>

```

5.1.5 Dao单元测试类

- 在src/test/java目录中添加测试类 **UserDaoTests**
- 使用Spring注入UserDao实例

```

1  package cn.tedu;
2
3  import cn.tedu.dao.UserDao;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.boot.test.context.SpringBootTest;
6
7  @SpringBootTest
8  public class UserDaoTests {
9      @Autowired
10     private UserDao userDao;
11 }

```

5.2 添加新的User信息

5.2.1 Dao接口

- 在UserDao接口中,添加**insert**接口方法
- 参数: User user -- 包含新增用户信息
- 返回值: int -- 添加的用户记录数

```
1 @Mapper
2 public interface UserDao {
3     /**添加新的User信息*/
4     int insert(User user);
5 }
```

5.2.2 Dao接口映射文件

- 在UserMapper.xml中添加新增User信息的SQL

```
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="cn.tedu.dao.UserDao">
6     <insert id="insert">
7         INSERT INTO tb_users
8         (username, nickname, password, mobile, STATUS, created_time,
9         modified_time)
10        VALUES (#{username}, #{nickname}, #{password}, #{mobile}, #
11        {status}, #{createdTime}, #{modifiedTime})
12    </insert>
13 </mapper>
```

5.2.3 Dao单元测试类实现

- 定义insert单元测试方法,插入一条新的用户记录

```
1 @SpringBootTest
2 public class UserDaoTests {
3     @Autowired
4     private UserDao userDao;
5     @Test
6     public void insert(){
7         User user=new User();
8         user.setUsername("Jack");
9         user.setPassword("123456");
10        user.setNickname("Jack");
11        user.setMobile("1391112121");
12        user.setStatus(1);
13        user.setCreatedTime(new Date());
14        user.setModifiedTime(new Date());
15        userDao.insert(user);
16    }
17 }
```

5.2.4 执行测试

- 执行insert方法,执行成功,查看tb_users表中是否出现一条记录

id	username	nickname	password	mobile	status	created_time	modified_time
2	Jack	Jack	e10adc3949ba59abbe56e057f20f883e	1391112121	1	2023-02-07 15:48:16	2023-02-07 15:48:16

5.3 查询指定时间范围注册的用户信息

5.3.1 DAO接口

- 在UserDao接口中,添加list接口方法
- 参数: String beginTime -- 起始时间
- 参数: String endTime -- 终止时间
- 返回值: List<User> -- 返回多条用户信息

```
1 List<User> list(String beginTime,String endTime);
```

5.3.2 DAO接口映射文件

- 在application.yml中指定扫描的包路径

```
1 #MyBatis开启驼峰映射,并且扫描xml文件
2 mybatis:
3   #指定entity扫描包类让mybatis自动扫描到自定义的包路径,这样在mapper.xml中就直接写类名即可
4   type-aliases-package: cn.tedu.pojo
```

- UserMapper.xml中定义list的SQL语句

```
1 <select id="list" resultType="User">
2   SELECT id, username, nickname, password, mobile, status, created_time,
3   modified_time
4   FROM tb_users
5   WHERE created_time > #{createdTime}
6 </select>
```

5.3.3 DAO单元测试类实现

- 定义list单元测试方法

```
1 @Test
2 public void list(){
3   List<User> list = userDao.list("2023-04-16 12:00:00","2023-04-17
4   23:59:59");
5   for(User user:list){
6     System.out.println(user);
7   }
8 }
```

5.3.4 执行测试

- 执行testList方法,执行成功,将查询结果打印在控制台

```
User{id=3, username='Jack', nickname='Jack', password='null', mobile='1391112121', status=1,
createdTime=Tue Feb 07 16:03:06 CST 2023, modifiedTime=Tue Feb 07 16:03:06 CST 2023}
```

5.3.5 分析问题

- 类似界面

- 查询情况:
 - 只输入起始时间,查询起始时间之后所有的注册信息
 - 只输入终止时间,查询终止时间之前的所有注册信息
 - 输入起始时间和终止时间,查询起始时间之后并且终止时间之前的所有注册信息
 - 起始时间和终止时间都不输入,则查询所有信息

5.3.6 使用动态SQL修改案例

- 将UserMapper.xml中id为list的SQL,修改如下:

```
1 <select id="list" resultType="User">
2     SELECT id, username, nickname, mobile, STATUS, created_time,
3     modified_time
4     FROM tb_users
5     <where>
6         <if test="beginTime!=null">
7             AND created_time >= #{beginTime}
8         </if>
9         <if test="endTime!=null">
10            AND created_time <= #{endTime}
11        </if>
12    </where>
13 </select>
```

5.3.7 进行测试

- 执行list方法测试方法,并且输入起始时间和终止时间,查看测试结果

```
1 @Test
2 public void list(){
3     List<User> list = userDao.list("2023-04-16 12:00:00","2023-04-16
4     23:59:59");
5     for(User user:list){
6         System.out.println(user);
7     }
8 }
```

- 执行list方法测试方法,并且只输入起始时间,测试结果

```
1 @Test
2 public void list(){
3     List<User> list = userDao.list("2023-04-16 12:00:00",null);
4     for(User user:list){
5         System.out.println(user);
6     }
7 }
```

- 执行list方法测试方法,并且只输入终止时间,测试结果

```

1  @Test
2  public void list(){
3      List<User> list = userDao.list(null,"2023-04-16 23:59:59");
4      for(User user:list){
5          System.out.println(user);
6      }
7  }

```

- 执行list方法测试方法,什么也不输入,测试结果

```

1  @Test
2  public void list(){
3      List<User> list = userDao.list(null,null);
4      for(User user:list){
5          System.out.println(user);
6      }
7  }

```

5.4 更新用户信息

5.4.1 Dao接口

- 在UserDao接口中,添加**update**接口方法
- 参数: User user -- 包含修改用户信息
- 返回值: int -- 返回修改的记录数

```

1  /**更新用户信息*/
2  int update(User user);

```

5.4.2 Dao接口映射文件

- 在UserMapper.xml中添加SQL

```

1  <update id="update">
2      UPDATE tb_users SET
3      username=#{username},
4      nickname=#{nickname},
5      mobile=#{mobile},
6      modified_time=#{modifiedTime}
7      WHERE id=#{id}
8  </update>

```

5.4.3 Dao单元测试类实现

- 定义**update**单元测试方法

```

1  @Test
2  public void update(){
3      User user=new User();
4      user.setId(3);
5      user.setUsername("Tom");
6      user.setNickname("Pony-001");
7      user.setMobile("1234569098");
8      user.setStatus(2);
9      user.setModifiedTime(new Date());
10     userDao.update(user);
11 }

```

5.4.4 执行测试

- 执行testUpdate方法,执行成功,查看tb_users表中的记录是否发生变化

id	username	nickname	password	mobile	status	created_time	modified_time
3	Tom	Pony-001	e10adc3949ba59abbe56e057f20f883e	1234569098	1	2023-02-07 16:03:06	2023-02-08 14:05:01
4	Jack	Jack	e10adc3949ba59abbe56e057f20f883e	1391112121	1	2023-02-08 13:28:28	2023-02-08 13:28:28

5.4.5 分析问题

- 在上述的更新操作中,我们需要填写要更新的用户信息,如下图所示

用户名:	<input type="text"/>	昵称:	<input type="text"/>
电话:	<input type="text"/>	状态:	<input type="text"/>

- 但是我们正常的使用时,如果不输入要修改的某一个或某几个信息,那么就应该不会修改该字段的值
- 所以我们此处为了能实现只修改填写的字段的内容,我们可以继续通过动态SQL技术,来使这类问题得到更为简单的实现

5.4.6 使用动态SQL修改案例

- 将UserMapper.xml中id为update的SQL,修改如下:

```
1 <update id="update">
2     UPDATE tb_users
3     <set>
4         <if test="username!=null and username!=''">username=#{username},
5     </if>
6         <if test="nickname!=null and nickname!=''">nickname=#{nickname},
7     </if>
8         <if test="mobile!=null and mobile!=''">mobile=#{mobile},</if>
9         <if test="modifiedTime!=null">modified_time=#{modifiedTime}</if>
10    </set>
11    WHERE id=#{id}
12 </update>
```

5.5 批量修改用户状态

5.5.1 Dao接口

- 在UserDao接口中,添加validById接口方法
- 参数: Integer[] ids -- 需要修改的多个用户id
- 参数: Integer status -- 修改为该状态值
- 返回值: int -- 返回修改的记录数

```
1 /**修改用户状态*/
2 int validById(Integer[] ids, Integer status);
```

5.5.2 Dao接口映射文件

- 在UserMapper.xml中添加SQL,但是目前我们SQL怎么写呢?此处我们传入了多个id值,所以可以使用动态SQL中的foreach标签

```
1 <update id="validById">
2     UPDATE tb_users
3     SET status=#{status}
4     WHERE
5     id IN
6     <foreach collection="ids" open="(" close=")" separator="," item="id">
7         #{id}
8     </foreach>
9 </update>
```

5.5.3 Dao单元测试类实现

- 定义validById单元测试方法

```
1 @Test
2 public void validById() {
3     Integer[] ids = new Integer[]{3, 4};
4     int rows = userDao.validById(ids, 0);
5     System.out.println(rows > 0 ? "修改成功" + rows + "条!" : "修改失败!");
6 }
```

5.5.4 执行测试

- 执行testValidById方法,执行成功,查看tb_users表中的记录是否发生变化

id	username	nickname	password	mobile	status	created_time	modified_time
3	Lucy	Pony-001	e10adc3949ba59abbe56e057f20f883e	1234569098	0	2023-02-07 16:03:06	2023-02-08 14:21:30
4	Jack	Jack	e10adc3949ba59abbe56e057f20f883e	1391112121	0	2023-02-08 13:28:28	2023-02-08 13:28:28

5.5.6 分析问题

- 在上述的修改操作中,我们也可以选择 not填写修改的id,那么就会报错,那这种是我们不希望看到的,所以我们希望用户如果不填写id,那么就不会修改任何用户的记录

5.5.7 使用动态SQL修改案例

- 将UserMapper.xml中id为validById的SQL,修改如下:

```
1 <update id="validById">
2     UPDATE tb_users
3     set status=#{status}
4     <where>
5         <choose>
6             <when test="ids!=null and ids.length>0">
7                 id in <!--(1,2,3,4,5)-->
8                 <foreach collection="ids" open="(" close=")" separator=","
9                 item="id">
10                     #{id}
11                 </foreach>
12             </when>
13             <otherwise>
14                 1=2
15             </otherwise>
16         </choose>
17     </where>
```


6 整合MyBatis完成文章业务操作

6.1 相关设计

6.1.1 文章表设计

- `tb_articles` 表示 文章表 ,其中设计的字段内容如下

Field	Type	Comment
id	bigint(20)	ID
title	varchar(50)	标题
type	char(1)	类型 (1 原创 2 转载 3翻译)
content	varchar(500)	文章内容
status	char(1)	状态 (1审核 2通过 3关闭)
user_id	bigint(20) unsigned	用户id
created_time	datetime	创建时间
modified_time	datetime	更新时间

6.1.2 文章标签表

- `tb_articles_tags` 表表示 文章标签表 ,基于业务创建文章、标签关系表（文章和标签是一种多对多的关系，这种关系需要一张关系表。）,其中设计的字段内容如下

Field	Type	Comment
id	bigint(20)	ID
article_id	bigint(20)	文章 ID
tag_id	bigint(20)	标签 ID
created_time	datetime	创建时间
modified_time	datetime	更新时间

6.1.3 POJO对象设计

```
1 package cn.tedu.pojo;
2
3 import java.util.Arrays;
4 import java.util.Date;
5 import java.util.List;
6
7 public class Article {
8     private Long id;
9     private String title;
10    private String type;
11    private String status;
12    private String content;
13    private Long userId;
```

```
14     private Date createTime;
15     private Date modifiedTime;
16
17     public Long getId() {
18         return id;
19     }
20
21     public void setId(Long id) {
22         this.id = id;
23     }
24
25     public String getTitle() {
26         return title;
27     }
28
29     public void setTitle(String title) {
30         this.title = title;
31     }
32
33     public String getType() {
34         return type;
35     }
36
37     public void setType(String type) {
38         this.type = type;
39     }
40
41     public String getStatus() {
42         return status;
43     }
44
45     public void setStatus(String status) {
46         this.status = status;
47     }
48
49     public String getContent() {
50         return content;
51     }
52
53     public void setContent(String content) {
54         this.content = content;
55     }
56
57     public Long getUserId() {
58         return userId;
59     }
60
61     public void setUserId(Long userId) {
62         this.userId = userId;
63     }
64
65     public Date getCreateTime() {
66         return createTime;
67     }
68
69     public void setCreateTime(Date createTime) {
70         this.createTime = createTime;
71     }
72
73     public Date getModifiedTime() {
74         return modifiedTime;
75     }
76
```

```

77     public void setModifiedTime(Date modifiedTime) {
78         this.modifiedTime = modifiedTime;
79     }
80
81     @Override
82     public String toString() { //alt+insert
83         return "Article{" +
84             "id=" + id +
85             ", title='" + title + '\'' +
86             ", type='" + type + '\'' +
87             ", status='" + status + '\'' +
88             ", content='" + content + '\'' +
89             ", userId=" + userId +
90             ", createdTime=" + createdTime +
91             ", modifiedTime=" + modifiedTime +
92             '}';
93     }
94 }

```

6.1.4 DAO接口设计

① ArticleDao

```

1  package cn.tedu.dao;
2
3  import org.apache.ibatis.annotations.Mapper;
4
5  @Mapper
6  public interface ArticleDao {
7
8  }

```

② ArticleTagDao

```

1  package cn.tedu.dao;
2
3  import org.apache.ibatis.annotations.Mapper;
4
5  @Mapper
6  public interface ArticleTagDao {
7  }

```

6.1.5 Dao接口映射文件

① ArticleMapper.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="cn.tedu.dao.ArticleDao">
6
7  </mapper>

```

② ArticleTagMapper.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="cn.tedu.dao.ArticleTagDao">
6
7  </mapper>

```

6.1.6 Dao单元测试类

① ArticleDaoTest

```

1  package cn.tedu;
2
3  import org.springframework.boot.test.context.SpringBootTest;
4
5  @SpringBootTest
6  public class ArticleDaoTest {
7      @Autowired
8      private ArticleDao articleDao;
9  }

```

6.2 添加新的文章信息

6.2.1 Dao接口

- 在ArticleDao接口中,添加**insert**接口方法
- 参数: Article article -- 包含新增文章信息
- 返回值: int -- 添加的文章记录数

```

1  int insert(Article article);

```

6.2.2 Dao接口映射文件

- ArticleMapper.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="cn.tedu.dao.ArticleDao">
6      <insert id="insert">
7          INSERT INTO tb_articles
8              (title, type, content, status, user_id, created_time,
9              modified_time)
10             VALUES (#{title}, #{type}, #{content}, #{status}, #{userId}, now(),
11                     now())
12      </insert>
13  </mapper>

```

6.2.3 Dao单元测试类实现

① ArticleDaoTest

```

1  @SpringBootTest
2  public class ArticleDaoTest {
3      @Autowired
4      private ArticleDao articleDao;

```

```

5
6     @Test
7     public void insert() {
8         Article article = new Article();
9         article.setTitle("Spring Boot");
10        article.setContent("Very Good");
11        article.setType("1");
12        article.setStatus("1");
13        article.setUserId(1L);
14        article.setCreateTime(new Date());
15        article.setModifiedTime(new Date());
16        articleDao.insert(article);
17        System.out.println(rows > 0 ? "新增成功!" : "新增失败!");
18    }
19 }

```

6.2.4 执行测试

- 执行insert方法,执行成功,查看tb_articles表中是否出现一条记录

id	title	...	content	STATUS	user_id	created_time	modified_time
1	Spring Boot	1	Very Good	1	1	2023-02-10 12:42:30	2023-02-10 12:42:30

6.3 添加新的文章信息(改)

6.3.1 分析问题

- 现在我们插入一条文章记录是没有问题的,但是文章表和文章标签表是关联的,所以我们应该在插入文章时,也要将文章和标签的相关信息一同插入到文章标签表中,所以也需要在当前测试方法中,执行插入文章标签表记录的功能

6.3.2 DAO接口

- 在ArticleTagDao中定义插入文章和标签关系数据的接口方法
- 参数: Long articleId -- 对应的是新增的文章id
- 参数: Long[] tagIds -- 由于文章和标签是一对多的关系,文章可能有多个标签,所以使用数组
- 返回值: int -- 返回插入的记录数

```

1  /**将文章和标签关系数据写入数据库*/
2  int insert(Long articleId,Long[] tagIds);

```

6.3.3 DAO接口映射文件

① ArticleTagMapper.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="cn.tedu.dao.ArticleTagDao">
6      <insert id="insert">
7          INSERT INTO tb_article_tags
8          (article_id,tag_id,modified_time,created_time)
9          VALUES
10             <foreach collection="tagIds" separator="," item="tagId">
11                 (#{articleId},#{tagId},NOW(),NOW())
12             </foreach>
13      </insert>

```

6.3.4 DAO单元测试类实现

① ArticleDaoTests

```

1  package cn.tedu;
2
3  import cn.tedu.dao.ArticleDao;
4  import cn.tedu.dao.ArticleTagDao;
5  import cn.tedu.pojo.Article;
6  import org.junit.jupiter.api.Test;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.boot.test.context.SpringBootTest;
9  import org.springframework.transaction.annotation.Transactional;
10
11 import java.util.Date;
12
13 @SpringBootTest
14 public class ArticleDaoTest {
15     @Autowired
16     private ArticleDao articleDao;
17     @Autowired
18     private ArticleTagDao articleTagDao;
19
20     @Test
21     public void insert() {
22         Article article = new Article();
23         article.setTitle("Spring Boot");
24         article.setContent("Very Good");
25         article.setType("1");
26         article.setStatus("1");
27         article.setUserId(1L);
28         article.setCreateTime(new Date());
29         article.setModifiedTime(new Date());
30         //将文章自身信息写入到数据库
31         int rows = articleDao.insert(article);
32         System.out.println(rows > 0 ? "新增文章成功!" : "新增文章失败!");
33         //将文章和标签关系数据写入到数据库
34         rows = articleTagDao.insert(article.getId(), new Long[]{1L, 3L});
35         System.out.println(rows > 0 ? "新增文章标签关系成功!" : "新增文章标签失
36         败!");
37     }
38 }

```

6.3.5 执行测试

- 执行insert方法,执行成功,查看tb_articles表中是否出现一条记录

id	title	TYPE	content	STATUS	user_id	created_time	modified_time
1	Spring Boot	1	Very Good	1	1	2023-02-10 12:42:30	2023-02-10 12:42:30
8	Spring Boo...	1	Very Good2	1	1	2023-02-10 13:10:21	2023-02-10 13:10:21

- 查看tb_articles_tag表中是否出现两条记录

id	article_id	tag_id	created_time	modified_time
5	<null>	1	2023-02-10 13:10:21	2023-02-10 13:10:21
6	<null>	3	2023-02-10 13:10:21	2023-02-10 13:10:21

6.3.6 分析问题

- id不能获取

6.3.7 解决问题

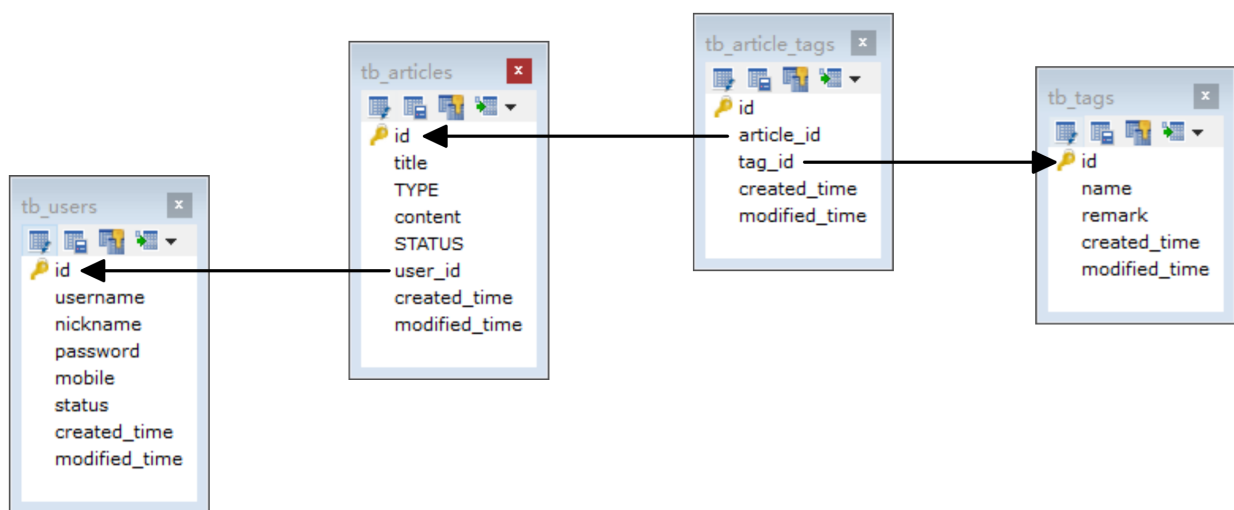
- 在MyBatis的XML中的CRUD的标签中,两个属性:
 - useGeneratedKeys="true"表示使用自增主键值
 - keyProperty="id" 这里表示将主键值赋值给参数中的id属性

① ArticleMapper.xml

```
1 <mapper namespace="cn.tedu.dao.ArticleDao">
2   <insert id="insert" useGeneratedKeys="true" keyProperty="id">
3     INSERT INTO tb_articles
4       (title, type, content, status, user_id, created_time,
5        modified_time)
6     VALUES (#{title}, #{type}, #{content}, #{status}, #{userId}, now(),
7             now())
8   </insert>
9 </mapper>
```

6.4 基于文章id查询文章内容以及文章对应的标签信息

6.4.1 ER图



- 用户表和文章表是一对多的关系
- 文章表和标签表是多对多的关系
- 一个用户和一个文章的关系是一对一的关系
- 一个文章和多个标签的关系是一对多的关系

6.4.2 POJO对象设计

① Article

```
1 package cn.tedu.pojo;
2
3 import java.util.Date;
4 import java.util.List;
5
6 public class Article {
```

```
7     private Long id;
8     private String title;
9     private String type;
10    private String status;
11    private String content;
12    private Long userId;
13    private Date createdTime;
14    private Date modifiedTime;
15    private List<Tag> Tag;
16    private User author;
17
18    public Long getId() {
19        return id;
20    }
21
22    public void setId(Long id) {
23        this.id = id;
24    }
25
26    public String getTitle() {
27        return title;
28    }
29
30    public void setTitle(String title) {
31        this.title = title;
32    }
33
34    public String getType() {
35        return type;
36    }
37
38    public void setType(String type) {
39        this.type = type;
40    }
41
42    public String getStatus() {
43        return status;
44    }
45
46    public void setStatus(String status) {
47        this.status = status;
48    }
49
50    public String getContent() {
51        return content;
52    }
53
54    public void setContent(String content) {
55        this.content = content;
56    }
57
58    public Long getUserId() {
59        return userId;
60    }
61
62    public void setUserId(Long userId) {
63        this.userId = userId;
64    }
65
66    public Date getCreatedTime() {
67        return createdTime;
68    }
69
```



```

70     public void setCreateTime(Date createTime) {
71         this.createTime = createTime;
72     }
73
74     public Date getModifiedTime() {
75         return modifiedTime;
76     }
77
78     public void setModifiedTime(Date modifiedTime) {
79         this.modifiedTime = modifiedTime;
80     }
81     public List<Tag> getTag() {
82         return Tag;
83     }
84
85     public void setTag(List<Tag> Tag) {
86         this.Tag = Tag;
87     }
88     public User getAuthor() {
89         return author;
90     }
91
92     public void setAuthor(User author) {
93         this.author = author;
94     }
95     @Override
96     public String toString() {
97         return "Article{" +
98             "id=" + id +
99             ", title='" + title + '\'' +
100             ", type='" + type + '\'' +
101             ", status='" + status + '\'' +
102             ", content='" + content + '\'' +
103             ", userId=" + userId +
104             ", author=" + author +
105             ", createTime=" + createTime +
106             ", modifiedTime=" + modifiedTime +
107             ", Tag=" + Tag +
108             '}';
109     }
110 }

```

6.4.3 DAO接口

- 在ArticleDao中定义基于文章id查询文章内容以及文章对应的标签信息的接口方法
- 参数: Long id -- 对应的是查询的文章id
- 返回值: Article -- 返回文章信息

```

1 Article selectById(Long id);

```

6.4.4 DAO接口映射文件

① ArticleMapper.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="cn.tedu.dao.ArticleDao">

```

```

6      <insert id="insert" useGeneratedKeys="true" keyProperty="id">
7          INSERT INTO tb_articles
8              (title, type, content, status, user_id, created_time,
modified_time)
9              VALUES (#{title}, #{type}, #{content}, #{status}, #{userId}, now(),
now())
10     </insert>
11     <!--基于文章id查询文章信息以及文章对应的tag信息-->
12     <select id="selectById" resultMap="articleTag">
13         SELECT ar.id,ar.title,ar.type,ar.content,ar.status,ar.user_id,
14             tag.id tagId,tag.name tagName,u.username,u.nickname
15         FROM tb_articles ar
16             JOIN tb_users u ON ar.user_id=u.id
17             LEFT JOIN tb_article_tags art ON ar.id=art.article_id
18             LEFT JOIN tb_tags tag ON art.tag_id =tag.id
19         where ar.id=#{id}
20     </select>
21     <!--ResultMap是mybatis中用于实现高级映射的元素-->
22     <resultMap id="articleTag" type="cn.tedu.pojo.Article">
23         <id property="id" column="id"></id>
24         <result property="title" column="title"></result>
25         <result property="type" column="type"></result>
26         <result property="content" column="content"></result>
27         <result property="status" column="status"></result>
28         <result property="userId" column="user_id"></result>
29         <result property="createdTime" column="created_time"></result>
30         <result property="modifiedTime" column="modified_time"></result>
31         <association property="author" javaType="cn.tedu.pojo.User">
32             <id property="id" column="user_id"></id>
33             <result property="username" column="username"></result>
34             <result property="nickname" column="nickname"></result>
35         </association>
36         <!--Collection对应one2many关系映射-->
37         <collection property="Tag" ofType="cn.tedu.pojo.Tag">
38             <id property="id" column="tagId"></id>
39             <result property="name" column="tagName"></result>
40         </collection>
41     </resultMap>
42 </mapper>

```

6.4.5 Dao单元测试类实现

① ArticleDaoTest

```

1  package cn.tedu;
2
3  import cn.tedu.dao.ArticleDao;
4  import cn.tedu.dao.ArticleTagDao;
5  import cn.tedu.pojo.Article;
6  import cn.tedu.pojo.Tag;
7  import org.junit.jupiter.api.Test;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.transaction.annotation.Transactional;
11
12 import java.util.Date;
13 import java.util.List;
14
15 /**
16  * @author 老安
17  * @data 2023-04-17 19:20

```

```

18  */
19  @SpringBootTest
20  public class ArticleDaoTest {
21      @Autowired
22      private ArticleDao articleDao;
23      @Autowired
24      private ArticleTagDao articleTagDao;
25
26      @Test
27      public void insert() {
28          Article article = new Article();
29          article.setTitle("Spring Boot");
30          article.setContent("Very Good");
31          article.setType("1");
32          article.setStatus("1");
33          article.setUserId(1L);
34          article.setCreateTime(new Date());
35          article.setModifiedTime(new Date());
36          //将文章自身信息写入到数据库
37          System.out.println("插入文章前的id:" + article.getId());
38          int rows = articleDao.insert(article);
39          System.out.println("插入文章后的id:" + article.getId());
40          System.out.println(rows > 0 ? "新增文章成功!" : "新增文章失败!");
41          //将文章和标签关系数据写入到数据
42          rows = articleTagDao.insert(article.getId(), new Long[]{1L, 3L});
43          System.out.println(rows > 0 ? "新增文章标签关系成功!" : "新增文章标签失
败!");
44      }
45
46      @Test
47      public void selectById() {
48          Article article = articleDao.selectById(3L);
49          System.out.println("文章信息:" + article);
50          System.out.println("文章所属用户信息:" + article.getAuthor());
51          List<Tag> tags = article.getTag();
52          for (Tag tag : tags) {
53              System.out.println("文章所属标签:" + tag);
54          }
55      }
56  }

```

6.4.6 执行测试

- 执行**selectById**方法,执行成功,将查询结果打印在控制台

```

Article{id=10, title='Spring Boot1', type='1', status='1', content='Very Good2', userId=1, author=User{id=1, username='Jack',
nickname='Jack', password='null', mobile='null', status=null, createTime=null, modifiedTime=null}, createTime=Fri Feb 10
13:16:28 CST 2023, modifiedTime=Fri Feb 10 13:16:28 CST 2023, tags=[Tag{id=1, name='TiDB', remark='null', createTime=null,
modifiedTime=null}]}

```

6.5 查询所有文章

6.5.1 DAO接口

- 在ArticleDao中定义查询所有文章的接口方法
- 返回值: List<Article> -- 返回多条文章信息

```

1  List<Article> list();

```

① ArticleMapper.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="cn.tedu.dao.ArticleDao">
6      <insert id="insert" useGeneratedKeys="true" keyProperty="id">
7          INSERT INTO tb_articles
8              (title, type, content, status, user_id, created_time,
modified_time)
9              VALUES (#{title}, #{type}, #{content}, #{status}, #{userId}, now(),
now())
10     </insert>
11     <!--基于文章id查询文章信息以及文章对应的tag信息-->
12     <select id="selectById" resultMap="articleTag">
13         SELECT ar.id,ar.title,ar.type,ar.content,ar.status,ar.user_id,
14             tag.id tagId,tag.name tagName,u.username,u.nickname
15         FROM tb_articles ar
16             JOIN tb_users u ON ar.user_id=u.id
17             LEFT JOIN tb_article_tags art ON ar.id=art.article_id
18             LEFT JOIN tb_tags tag ON art.tag_id =tag.id
19         where ar.id=#{id}
20     </select>
21     <!--基于文章id查询文章信息以及文章对应的tag信息-->
22     <select id="list" resultMap="articleTag">
23         SELECT ar.id,ar.title,ar.type,ar.content,ar.status,ar.user_id,
24             tag.id tagId,tag.name tagName,u.username,u.nickname
25         FROM tb_articles ar
26             JOIN tb_users u ON ar.user_id=u.id
27             LEFT JOIN tb_article_tags art ON ar.id=art.article_id
28             LEFT JOIN tb_tags tag ON art.tag_id =tag.id
29     </select>
30     <!--ResultMap是mybatis中用于实现高级映射的元素-->
31     <resultMap id="articleTag" type="cn.tedu.pojo.Article">
32         <id property="id" column="id"></id>
33         <result property="title" column="title"></result>
34         <result property="type" column="type"></result>
35         <result property="content" column="content"></result>
36         <result property="status" column="status"></result>
37         <result property="userId" column="user_id"></result>
38         <result property="createdTime" column="created_time"></result>
39         <result property="modifiedTime" column="modified_time"></result>
40         <association property="author" javaType="cn.tedu.pojo.User">
41             <id property="id" column="user_id"></id>
42             <result property="username" column="username"></result>
43             <result property="nickname" column="nickname"></result>
44         </association>
45         <!--Collection对应one2many关系映射-->
46         <collection property="Tag" ofType="cn.tedu.pojo.Tag">
47             <id property="id" column="tagId"></id>
48             <result property="name" column="tagName"></result>
49         </collection>
50     </resultMap>
51 </mapper>

```

6.5.3 DAO单元测试类实现

① ArticleDaoTest

```
1 package cn.tedu;
2
3 import cn.tedu.dao.ArticleDao;
4 import cn.tedu.dao.ArticleTagDao;
5 import cn.tedu.pojo.Article;
6 import cn.tedu.pojo.Tag;
7 import org.junit.jupiter.api.Test;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.context.SpringBootTest;
10 import org.springframework.transaction.annotation.Transactional;
11
12 import java.util.Date;
13 import java.util.List;
14
15 /**
16  * @author 老安
17  * @data 2023-04-17 19:20
18  */
19 @SpringBootTest
20 public class ArticleDaoTest {
21     @Autowired
22     private ArticleDao articleDao;
23     @Autowired
24     private ArticleTagDao articleTagDao;
25
26     @Test
27     public void insert() {
28         Article article = new Article();
29         article.setTitle("Spring Boot");
30         article.setContent("Very Good");
31         article.setType("1");
32         article.setStatus("1");
33         article.setUserId(1L);
34         article.setCreateTime(new Date());
35         article.setModifiedTime(new Date());
36         //将文章自身信息写入到数据库
37         System.out.println("插入文章前的id:" + article.getId());
38         int rows = articleDao.insert(article);
39         System.out.println("插入文章后的id:" + article.getId());
40         System.out.println(rows > 0 ? "新增文章成功!" : "新增文章失败!");
41         //将文章和标签关系数据写入到数据
42         rows = articleTagDao.insert(article.getId(), new Long[]{1L, 3L});
43         System.out.println(rows > 0 ? "新增文章标签关系成功!" : "新增文章标签失
败!");
44     }
45
46     @Test
47     public void selectById() {
48         Article article = articleDao.selectById(3L);
49         System.out.println("文章信息:" + article);
50         System.out.println("文章所属用户信息:" + article.getAuthor());
51         List<Tag> tags = article.getTag();
52         for (Tag tag : tags) {
53             System.out.println("文章所属标签:" + tag);
54         }
55     }
56     @Test
57     public void list(){
```

```

58         List<Article> list = articleDao.list();
59         for (Article article : list) {
60             System.out.println(article);
61         }
62     }
63 }

```

6.5.4 执行测试

- 执行list方法,执行成功,查看控制台查询的结果

```

Article{id=10, title='Spring Boot1', type='1', status='1', content='Very Good2', userId=1, author=User{id=1, username='Jack',
nickname='Jack', password='null', mobile='null', status=null, createTime=null, modifiedTime=null}, createTime=Fri Feb 10
13:16:28 CST 2023, modifiedTime=Fri Feb 10 13:16:28 CST 2023, tags=[Tag{id=1, name='TiDB', remark='null', createTime=null,
modifiedTime=null}]}
Article{id=11, title='Spring Boot1', type='1', status='1', content='Very Good2', userId=1, author=User{id=1, username='Jack',
nickname='Jack', password='null', mobile='null', status=null, createTime=null, modifiedTime=null}, createTime=Tue Feb 14
13:55:35 CST 2023, modifiedTime=Tue Feb 14 13:55:35 CST 2023, tags=[Tag{id=1, name='TiDB', remark='null', createTime=null,
modifiedTime=null}]}
Article{id=1, title='Spring Boot1', type='1', status='1', content='Very Good', userId=1, author=User{id=1, username='Jack',
nickname='Jack', password='null', mobile='null', status=null, createTime=null, modifiedTime=null}, createTime=Fri Feb 10
12:42:30 CST 2023, modifiedTime=Fri Feb 10 12:42:30 CST 2023, tags=[]}
Article{id=8, title='Spring Boot1', type='1', status='1', content='Very Good2', userId=1, author=User{id=1, username='Jack',
nickname='Jack', password='null', mobile='null', status=null, createTime=null, modifiedTime=null}, createTime=Fri Feb 10
13:10:21 CST 2023, modifiedTime=Fri Feb 10 13:10:21 CST 2023, tags=[]}
Article{id=9, title='Spring Boot1', type='1', status='1', content='Very Good2', userId=1, author=User{id=1, username='Jack',
nickname='Jack', password='null', mobile='null', status=null, createTime=null, modifiedTime=null}, createTime=Fri Feb 10
13:12:46 CST 2023, modifiedTime=Fri Feb 10 13:12:46 CST 2023, tags=[]}

```

6.5.5 分析问题

- 在上述的两个案例中,查询的SQL是比较复杂的,并且两条SQL是有相同的部分的,所以我们可以使用动态SQL将重复部分提取为SQL片段,这样在后面使用时,就会轻松很多,并且结构也更清晰

```
<select id="selectById" resultMap="articleTag">
```

```

SELECT ar.id,ar.title,ar.type,ar.content,ar.status,ar.user_id,ar.created_time,ar.modified_time,
tag.id tagId,tag.name tagName,u.username,u.nickname
FROM tb_articles ar
JOIN tb_users u ON ar.user_id=u.id
LEFT JOIN tb_article_tags art ON ar.id=art.article_id
LEFT JOIN tb_tags tag ON art.tag_id =tag.id

```

```
where ar.id=#{id}
```

```
</select>
```

```
<select id="selectById" resultMap="articleTag">
```

```

SELECT ar.id,ar.title,ar.type,ar.content,ar.status,ar.user_id,ar.created_time,ar.modified_time,
tag.id tagId,tag.name tagName,u.username,u.nickname
FROM tb_articles ar
JOIN tb_users u ON ar.user_id=u.id
LEFT JOIN tb_article_tags art ON ar.id=art.article_id
LEFT JOIN tb_tags tag ON art.tag_id =tag.id

```

```
</select>
```

相同SQL

6.5.6 使用动态SQL修改案例

① ArticleMapper.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="cn.tedu.dao.ArticleDao">
6      <insert id="insert" useGeneratedKeys="true" keyProperty="id">
7          INSERT INTO tb_articles
8          (title, type, content, status, user_id, created_time,
modified_time)

```

```

9      VALUES ({title}, {type}, {content}, {status}, {userId}, now(),
now())
10    </insert>
11    <!--基于文章id查询文章信息以及文章对应的tag信息-->
12    <select id="selectById" resultMap="articleTag">
13      <include refid="selectArt"/>
14      where ar.id=#{id}
15    </select>
16    <!--基于文章id查询文章信息以及文章对应的tag信息-->
17    <select id="list" resultMap="articleTag">
18      <include refid="selectArt"/>
19    </select>
20    <!--ResultMap是mybatis中用于实现高级映射的元素-->
21    <resultMap id="articleTag" type="cn.tedu.pojo.Article">
22      <id property="id" column="id"></id>
23      <result property="title" column="title"></result>
24      <result property="type" column="type"></result>
25      <result property="content" column="content"></result>
26      <result property="status" column="status"></result>
27      <result property="userId" column="user_id"></result>
28      <result property="createdTime" column="created_time"></result>
29      <result property="modifiedTime" column="modified_time"></result>
30      <association property="author" javaType="cn.tedu.pojo.User">
31        <id property="id" column="user_id"></id>
32        <result property="username" column="username"></result>
33        <result property="nickname" column="nickname"></result>
34      </association>
35      <!--Collection对应one2many关系映射-->
36      <collection property="Tag" ofType="cn.tedu.pojo.Tag">
37        <id property="id" column="tagId"></id>
38        <result property="name" column="tagName"></result>
39      </collection>
40    </resultMap>
41    <sql id="selectArt">
42      SELECT ar.id,ar.title,ar.type,ar.content,ar.status,ar.user_id,
43        tag.id tagId,tag.name tagName,u.username,u.nickname
44      FROM tb_articles ar
45        JOIN tb_users u ON ar.user_id=u.id
46        LEFT JOIN tb_article_tags art ON ar.id=art.article_id
47        LEFT JOIN tb_tags tag ON art.tag_id =tag.id
48    </sql>
49  </mapper>

```

6.5.7 进行测试

- 重新执行**selectById**方法和**list**方法,查看查询结果是否和之前一致

7 表关系

6.1 一对一关系

- 一张表的记录对应另外一张表的一条记录



丈夫表

id	name
1	蝎子精
2	奥特之父
3	许仙
3	唐僧

妻子表

id	name
1	奥特之母
2	白素贞
3	蛇精

- 在实际的开发中应用不多.因为一对一可以创建成一张表

配偶表

id	husbandName	wifeName
1	蝎子精	蛇精
2	奥特之父	奥特之母
3	许仙	白素贞
3	唐僧	

6.2 一对多（多对一）

- 记录一关系数据的表称为主表,而记录多关系数据的表称为从表

员工表

eid	did	name
1	2	唐僧
2	3	玉皇大帝
3	2	孙悟空
4	1	牛魔王
5	2	王母娘娘
6	3	猪八戒
7	1	九头蛇
8	2	沙僧

部门表

id	name
1	妖精组
2	取经大队
3	天庭

6.3 多对多关系

- 一张关联表记录两张表的关联关系

