

# 第二章 MyBatis框架

-MyBatis持久层框架 2018/11/8 [泽林.王峰]

## 授课目标

- 1、上章回顾
- 2、MyBatis开发Dao的两种方式
- 3、一对一关联关系映射
- 4、一对多关联关系映射
- 5、动态SQL查询

## 授课内容

### 1、 上章回顾

### 2、 MyBatis开发Dao的两种方式

重点掌握：接口+xml配置的方式实现

### 3、 一对一关联关系映射

#### 3.1)方法一：使用自定义实体类实现(掌握)

##### 3.1.1)自定义UsersCustom.java类

```
1  Users.java类：
2  public class Users implements Serializable {
3      private int uid;
4      private String unname;
5      private String email;
6      private String birth;
7      private double balance;
8
9      public Users() {
10     }
11
12     public Users(String unname, String email, String birth, double balance) {
13         this.unname = unname;
14         this.email = email;
15         this.birth = birth;
16         this.balance = balance;
17     }
```

```
18
19     public Users(int uid, String uname, String email, String birth, double balance) {
20         this.uid = uid;
21         this.uname = uname;
22         this.email = email;
23         this.birth = birth;
24         this.balance = balance;
25     }
26
27     public int getUid() {
28         return uid;
29     }
30
31     public void setUid(int uid) {
32         this.uid = uid;
33     }
34
35     public String getUname() {
36         return uname;
37     }
38
39     public void setUname(String uname) {
40         this.uname = uname;
41     }
42
43     public String getEmail() {
44         return email;
45     }
46
47     public void setEmail(String email) {
48         this.email = email;
49     }
50
51     public String getBirth() {
52         return birth;
53     }
54
55     public void setBirth(String birth) {
56         this.birth = birth;
57     }
58
59     public double getBalance() {
60         return balance;
61     }
62
63     public void setBalance(double balance) {
64         this.balance = balance;
65     }
66
67     @Override
68     public String toString() {
69         return "Users{" +
70             "uid=" + uid +
```

```
71         ", uname='" + uname + '\'' +
72         ", email='" + email + '\'' +
73         ", birth='" + birth + '\'' +
74         ", balance=" + balance +
75         '}'';
76     }
77 }
78
```

```
1  IdCard类:
2  public class IdCard implements Serializable {
3      private int id;
4      private String cno;
5      private int uid;
6
7      public IdCard() {
8      }
9
10     public IdCard(String cno, int uid) {
11         this.cno = cno;
12         this.uid = uid;
13     }
14
15     public IdCard(int id, String cno, int uid) {
16         this.id = id;
17         this.cno = cno;
18         this.uid = uid;
19     }
20
21     public int getId() {
22         return id;
23     }
24
25     public void setId(int id) {
26         this.id = id;
27     }
28
29     public String getCno() {
30         return cno;
31     }
32
33     public void setCno(String cno) {
34         this.cno = cno;
35     }
36
37     public int getUid() {
38         return uid;
39     }
40
41     public void setUid(int uid) {
42         this.uid = uid;
43     }
44 }
```

```

44
45     @Override
46     public String toString() {
47         return "IdCard{" +
48             "id=" + id +
49             ", cno='" + cno + '\'' +
50             ", uid=" + uid +
51             '}';
52     }
53 }
54

```

```

1  /**
2   * 自定义实体类（完成一对一查询的第一种方法）
3   */
4  public class UsersCustom extends Users {
5      private int id;
6      private String cno;    //身份证号
7      private int userId;    //外键：users表的主键(代表当前身份证的用户Id)
8
9      public int getId() {
10         return id;
11     }
12
13     public void setId(int id) {
14         this.id = id;
15     }
16
17     public String getCno() {
18         return cno;
19     }
20
21     public void setCno(String cno) {
22         this.cno = cno;
23     }
24
25     public int getUserId() {
26         return userId;
27     }
28
29     public void setUserId(int userId) {
30         this.userId = userId;
31     }
32
33     @Override
34     public String toString() {
35         return "UsersCustom{" +
36             "id=" + id +
37             ", cno='" + cno + '\'' +
38             ", userId=" + userId +
39             '}' + super.toString();
40     }

```

```
41 }
42
```

### 3.1.2)定义接口UsersMapper.java

```
1 public interface UsersMapper {
2     public List<UsersCustom> findUsers() throws Exception;
3 }
```

### 3.1.3)定义UsersMapper.xml的映射文件：

```
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
6 <mapper namespace="com.zelin.mapper.UsersMapper">
7     <!--一对一查询的方法一（使用自定义类实现）-->
8     <!--1.查询所有的用户（两张表）-->
9     <select id="findUsers" resultType="UsersCustom">
10         select u.uid,uname,email,birth,balance,cno,id
11         from users u,idcard i
12         where u.uid=i.uid
13     </select>
14 </mapper>
```

### 3.1.4)进行单元测试：

```
1 @RunWith(SpringJUnit4ClassRunner.class)
2 @ContextConfiguration("classpath*:spring/applicationContext*.xml")
3 public class TestOneToOne {
4     //1.从spring容器中注入sqlSessionFactoryBean
5     @Autowired
6     private SqlSessionFactoryBean sqlSessionFactoryBean;
7
8     /**
9      * 一对一加载的第一种实现方式
10     * @throws Exception
11     */
12     @Test
13     public void test01() throws Exception{
14         //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
15         SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
16         //3.根据工厂对象得到SqlSession对象
17         SqlSession sqlSession = factory.openSession();
18         UsersMapper mapper = sqlSession.getMapper(UsersMapper.class);
19         List<UsersCustom> users = mapper.findUsers();
20         for(UsersCustom custom : users){
21             System.out.println(custom);
22         }
23     }
```

```

23     }
24 }
25

```

### 3.1.5)运行结果如下：

```

UsersCustom{id=1, cno=' 423423423423423890', userId=0}Users{uid=1, uname=' 张三', email=' zhangsan@localhost', birth=' 1990-09-10', balance=0.0}
UsersCustom{id=2, cno=' 424242424234234143', userId=0}Users{uid=2, uname=' 李四', email=' lisi@localhost', birth=' 1992-05-16', balance=0.0}
UsersCustom{id=3, cno=' 524324242342342342', userId=0}Users{uid=3, uname=' 王五', email=' wangwu@localhost', birth=' 1995-05-16', balance=0.0}
UsersCustom{id=4, cno=' 233242542423423424', userId=0}Users{uid=4, uname=' 赵六', email=' zhaoliu@localhost', birth=' 1995-05-18', balance=0.0}
UsersCustom{id=5, cno=' 121343423423423423', userId=0}Users{uid=6, uname=' 李小明', email=' xiaomingli@qq.com', birth=' 1998-03-30', balance=0.0}
UsersCustom{id=6, cno=' 44444444444', userId=0}Users{uid=9, uname=' 成龙', email=' chenlong@163.com', birth=' 1954-09-21', balance=0.0}

```

## 3.2)方法二：使用自定义结果集的映射来实现

### 3.2.1 ) 修改UsersMapper.xml文件

```

1  <!--一对一查询的方法二（使用自定义结果集映射实现）-->
2  <resultMap id="usersMap" type="users">
3      <id property="uid" column="uid"/>
4      <result property="email" column="email"/>
5      <result property="uname" column="uname"/>
6      <result property="birth" column="birth"/>
7      <result property="balance" column="balance"/>
8      <association property="idCard" javaType="idCard" resultMap="idCardMap"/>
9  </resultMap>
10 <resultMap id="idCardMap" type="idCard">
11     <id column="id" property="id"/>
12     <result column="cno" property="cno"/>
13     <result column="uid" property="uid"/>
14 </resultMap>
15 <select id="findUsers2" resultMap="usersMap">
16     select
17         u.uid,uname,email,birth,i.id as id,cno,i.uid as uid
18     from users u,idcard i
19     where i.uid=u.uid
20 </select>

```

### 3.2.2 ) 在UserMapper接口中添加方法findUsers2

```

1  public List<Users> findUsers2() throws Exception;

```

### 3.2.3 ) 测试

```

1  /**
2   * 一对一加载的第二种实现方式(使用结果集映射来实现)
3   * @throws Exception
4   */
5  @Test
6  public void test02() throws Exception{
7      //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象

```

```

8      SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
9      //3.根据工厂对象得到SqlSession对象
10     SqlSession sqlSession = factory.openSession();
11     UsersMapper mapper = sqlSession.getMapper(UsersMapper.class);
12     List<Users> users = mapper.findUsers2();
13     for(Users user: users){
14         System.out.println(user + "--->" + user.getIdCard());
15     }
16 }

```

### 3.3)方法三：使用懒加载实现（掌握）

#### 3.3.1)修改UsersMapper.xml配置文件

```

1  <!--一对一查询的方法三（使用自定义结果集映射+懒加载实现）-->
2  <resultMap id="usersMap2" type="users">
3      <id property="uid" column="uid"/>
4      <result property="email" column="email"/>
5      <result property="uname" column="uname"/>
6      <result property="birth" column="birth"/>
7      <result property="balance" column="balance"/>
8      <!--下面的关联查询代表当查询idCard这个属性时，会执行findIdCardByUid子查询-->
9      <association property="idCard" javaType="idCard" column="uid"
10         select="findIdCardByUid" />
11  </resultMap>
12  <select id="findIdCardByUid" parameterType="int" resultType="idCard">
13      select * from idcard where uid = #{value}
14  </select>
15  <select id="findUsers3" resultMap="usersMap2">
16      select * from users
17  </select>

```

#### 3.3.2)在UserMapper.java接口中添加方法：

```

1  public List<Users> findUsers3() throws Exception;

```

#### 3.3.3)单元测试

```

1  /**
2   * 一对一加载的第三种实现方式(使用懒加载查询来实现)
3   * @throws Exception
4   */
5  @Test
6  public void test03() throws Exception{
7      //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
8      SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
9      //3.根据工厂对象得到SqlSession对象
10     SqlSession sqlSession = factory.openSession();
11     UsersMapper mapper = sqlSession.getMapper(UsersMapper.class);
12     List<Users> users = mapper.findUsers3();

```

```

13         for(Users user: users){
14             System.out.println(user + "--->" + user.getIdCard());
15         }
16     }

```

## 3.4)方法四：使用注解完成

### 3.4.1)修改IdCardMapper.java接口

```

1  public interface IdCardMapper {
2      //根据主键查询idcard时将其关联的Users查询出来
3      @Results(id = "idCardResult",value={
4          @Result(column = "id",property = "id",id = true),
5          @Result(column = "cno",property = "cno"),
6          @Result(column = "uid",property = "users",
7              one = @One(select =
8                  "com.zelin.mapper.UsersMapper.findUserByUid",fetchType= FetchType.LAZY))
9      })
10     @Select("select * from idCard where id=#{value}")
11     public IdCard findIdCardById(int id) throws Exception;
12 }

```

### 3.4.2)修改UsersMapper.java接口：

```

1  /**
2      * 根据用户id查询用户对象
3      * @param uid
4      * @return
5      * @throws Exception
6      */
7  @Select("select * from users where uid=#{value}")
8  public Users findUserByUid(int uid) throws Exception;

```

### 3.4.3)测试：

```

1  /**
2      * 一对一加载的第四种实现方式(使用注解查询来实现)
3      * @throws Exception
4      */
5  @Test
6  public void test04() throws Exception{
7      //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
8      SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
9      //3.根据工厂对象得到SqlSession对象
10     SqlSession sqlSession = factory.openSession();
11     IdCardMapper mapper = sqlSession.getMapper(IdCardMapper.class);
12     IdCard idCard = mapper.findIdCardById(1);
13     System.out.println(idCard + "-->" + idCard.getUsers());
14 }

```



## 4、 一对多关联关系映射(两种都掌握)

### 4.1)第一种方式，使用自定义ResultMap完成

```
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="com.zelin.mapper.CategoryMapper">
6     <!--1.一对多查询-->
7     <resultMap id="categoryMap" type="category">
8         <id column="cid" property="cid"/>
9         <result column="cname" property="cname"/>
10    <!-- 查询在类别中的所有的图书 -->
11    <collection property="books" ofType="book" javaType="list">
12        <id column="bid" property="bid"/>
13        <result column="bname" property="bname"/>
14        <result column="bauthor" property="bauthor"/>
15        <result column="publisher" property="publisher"/>
16        <result column="cid" property="cid"/>
17        <result column="imgpath" property="imgpath"/>
18    </collection>
19 </resultMap>
20 <!--1.查询所有的图书类别-->
21 <select id="findAll" resultMap="categoryMap">
22     select b.*,cname from category c,book b
23     where b.cid=c.cid
24 </select>
25 </mapper>
```

### 4.2)定义CategoryMapper.java这个接口

```
1 public interface CategoryMapper {
2     public List<Category> findAll() throws Exception;
3 }
```

### 4.3)测试

```
1 @RunWith(SpringJUnit4ClassRunner.class)
2 @ContextConfiguration("classpath*:spring/applicationContext*.xml")
3 public class TestOneToMany {
4
5     //1.从spring容器中注入sqlSessionFactoryBean
6     @Autowired
7     private SqlSessionFactoryBean sqlSessionFactoryBean;
8     @Test
9     public void test01() throws Exception{
10
11         //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
```

```

11     SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
12     //3.根据工厂对象得到SqlSession对象
13     SqlSession sqlSession = factory.openSession();
14     CategoryMapper mapper = sqlSession.getMapper(CategoryMapper.class);
15     List<Category> cates = mapper.findAll();
16     for(Category category : cates){
17         System.out.println(category + "-->" + category.getBooks());
18     }
19 }
20 }

```

#### 4.4)运行效果如下：

Category{cid=1, cname=' 小说类' }-->[Book{bid=1, bname=' 三国演义', bauthor=' 罗贯中', publisher=' 中国青年出版社', price=0.0, cid=1, Category{cid=2, cname=' 程序开发类' }-->[Book{bid=3, bname=' java编程思想', bauthor=' 埃克尔bc ', publisher=' 图灵出版社', price=0.0,

## 4.2)第二种方式，使用关联查询完成

### 4.2.1)修改CategoryMapper.xml文件

```

1  <!--2.一对多查询(第二种方式：使用关联查询完成)-->
2  <resultMap id="categoryMap2" type="category">
3      <id column="cid" property="cid"/>
4      <result column="cname" property="cname"/>
5      <collection property="books" javaType="list" ofType="book"
6          select="findBooksByCid" column="cid"/>
7  </resultMap>
8  <select id="findBooksByCid" parameterType="int" resultType="book">
9      select * from book where cid=#{value}
10 </select>
11 <select id="findAll2" resultMap="categoryMap2">
12     select * from category
13 </select>

```

### 4.2.2)修改CategoryMapper.java接口

```

1 public List<Category> findAll2() throws Exception;

```

### 4.2.3)测试：

```

1 //一对多查询的第二种方式：使用结果集关联查询完成
2 @Test
3 public void test02() throws Exception{
4     //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
5     SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
6     //3.根据工厂对象得到SqlSession对象
7     SqlSession sqlSession = factory.openSession();
8     CategoryMapper mapper = sqlSession.getMapper(CategoryMapper.class);
9     List<Category> cates = mapper.findAll2();
10    for(Category category : cates){
11        System.out.println(category + "-->" + category.getBooks());
12    }
13 }

```

## 5、 动态SQL查询

### 5.1)需求一：根据图书名称、图书作者、出版社及图书类别编号查询图书。

#### 5.1.1 ) 定义BookMapper.xml文件

```

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 <!--动态SQL查询-->
6 <mapper namespace="com.zelin.mapper.BookMapper">
7     <!--1.功能一：根据图书名称、图书作者、出版社及图书类别编号查询图书。-->
8     <!--1.1)定义进行条件查询的语句-->
9     <sql id="dynamicSQL" >
10         select * from book
11         <where>
12             <if test="bname != null and bname != ''">
13                 and bname like '%${bname}%'
14             </if>
15             <if test="bauthor != null and bauthor != ''">
16                 and bauthor like '%${bauthor}%'
17             </if>
18             <if test="publisher != null and publisher != ''">
19                 and publisher like '%${publisher}%'
20             </if>
21             <if test="cid != 0">
22                 and cid = #{cid}
23             </if>
24         </where>
25     </sql>
26     <!--条件查询-->
27     <select id="findBooksByKeywords" parameterType="bookVo" resultType="book">
28         <!-- 将动态sql语句包含进来-->
29         <include refid="dynamicSQL"/>
30     </select>

```

```
32 </mapper>
```

### 5.1.2 ) 定义BookMapper.java接口

```
1 public interface BookMapper {
2     public List<Book> findBooksByKeywords(BookVo bookVo) throws Exception ;
3 }
```

### 5.1.3 ) 测试 :

```
1 @RunWith(SpringJUnit4ClassRunner.class)
2 @ContextConfiguration("classpath*:spring/applicationContext*.xml")
3 public class TestDynamicSQL {
4     //1.从spring容器中注入sqlSessionFactoryBean
5     @Autowired
6     private SqlSessionFactoryBean sqlSessionFactoryBean;
7     //一对多查询的第一种方式：使用结果集映射完成
8     @Test
9     public void test01() throws Exception{
10         //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
11         SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
12         //3.根据工厂对象得到SqlSession对象
13         SqlSession sqlSession = factory.openSession();
14         BookMapper mapper = sqlSession.getMapper(BookMapper.class);
15         BookVo bookVo = new BookVo();
16         bookVo.setBauthor("b");
17         bookVo.setPublisher("治");
18         List<Book> books = mapper.findBooksByKeywords(bookVo);
19         for(Book book : books){
20             System.out.println(book);
21         }
22     }
23
24 }
```

## 5.3)需求二：根据多个图书编号查询图书列表。

### 5.3.1 ) 修改BookMapper.xml文件

```
1 <!--条件查询二：根据多个图书编号查询图书列表-->
2 <sql id="dynamicSQL2">
3     select * from book
4     <where>
5         bid in
6         <foreach collection="ids" item="id" open="(" close=")" separator=",">
7             #{id}
8         </foreach>
9     </where>
10 </sql>
11 <!--条件查询二：根据多个id查询图书-->
12 <select id="findBooksByIds" parameterType="bookVo" resultType="book">
```

```

13      <!-- 将动态sql语句包含进来-->
14      <include refid="dynamicSQL2"/>
15  </select>

```

### 5.3.2) 修改BookMapper.java接口

```

1  public List<Book> findBooksByIds(BookVo bookVo) throws Exception;

```

### 5.3.3) 测试：

```

1  //动态SQL查询二：根据多个图书id查询图书列表
2  @Test
3  public void test02() throws Exception{
4      //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
5      SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
6      //3.根据工厂对象得到SqlSession对象
7      SqlSession sqlSession = factory.openSession();
8      BookMapper mapper = sqlSession.getMapper(BookMapper.class);
9      BookVo bookVo = new BookVo();
10     bookVo.getIds().add(1);
11     bookVo.getIds().add(3);
12     bookVo.getIds().add(5);
13     List<Book> books = mapper.findBooksByIds(bookVo);
14     for(Book book : books){
15         System.out.println(book);
16     }
17 }

```

## 5.4)需求三：根据多个图书编号及图书关键字查询图书列表。

### 5.4.1) 修改BookMapper.xml文件

```

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  <!--动态SQL查询-->
6  <mapper namespace="com.zelin.mapper.BookMapper">
7      <!--1.功能一：根据图书名称、图书作者、出版社及图书类别编号查询图书。-->
8      <!--1.1)定义进行条件查询的语句-->
9      <sql id="dynamicSQL" >
10         <where>
11             <if test="bname != null and bname != ''">
12                 and bname like '%${bname}%'
13             </if>
14             <if test="bauthor != null and bauthor != ''">
15                 and bauthor like '%${bauthor}%'
16             </if>
17             <if test="publisher != null and publisher != ''">
18                 and publisher like '%${publisher}%'
19             </if>

```

```

20         <if test="cid != 0">
21             and cid = #{cid}
22         </if>
23     </where>
24
25 </sql>
26 <!--条件查询一：根据图书名称、图书作者、出版社及图书类别编号查询图书。-->
27 <select id="findBooksByKeywords" parameterType="bookVo" resultType="book">
28     select * from book
29     <!-- 将动态sql语句包含进来-->
30     <include refid="dynamicSQL"/>
31 </select>
32
33 <!--条件查询二：根据多个图书编号查询图书列表-->
34 <sql id="dynamicSQL2">
35     <where>
36         bid in
37         <foreach collection="ids" item="id" open="(" close=")" separator=",">
38             #{id}
39         </foreach>
40     </where>
41
42 </sql>
43 <!--条件查询二：根据多个id查询图书-->
44 <select id="findBooksByIds" parameterType="bookVo" resultType="book">
45     select * from book
46     <!-- 将动态sql语句包含进来-->
47     <include refid="dynamicSQL2"/>
48 </select>
49
50 <sql id="dynamicSQL3">
51     <where>
52         <if test="bname != null and bname != ''">
53             and bname like '%${bname}%'
54         </if>
55         <if test="bauthor != null and bauthor != ''">
56             and bauthor like '%${bauthor}%'
57         </if>
58         <if test="publisher != null and publisher != ''">
59             and publisher like '%${publisher}%'
60         </if>
61         <if test="cid != 0">
62             and cid = #{cid}
63         </if>
64         <if test="ids != null and ids.size() > 0">
65             and bid in
66             <foreach collection="ids" item="id" open="(" close=")" separator=",">
67                 #{id}
68             </foreach>
69         </if>
70     </where>
71 </sql>
72

```

```
73      <!--条件查询三：根据多个图书编号及图书关键字查询图书列表。-->
74      <select id="findBooksByCond" parameterType="bookVo" resultType="book">
75          select * from book
76          <include refid="dynamicSQL3"/>
77      </select>
78  </mapper>
```

#### 5.4.2 ) 修改BookMapper.java接口

```
1  public List<Book> findBooksByCond(BookVo bookVo) throws Exception;
```

#### 5.4.3 ) 测试

Book {bid=3, bname=' java编程思想', bauthor=' 埃克尔bc ', publisher=' 图灵出版社', price=109.0, cid=2, imgpath=' images/c. jpg' }