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

```
Users.java类：
public class Users implements Serializable {
    private int uid;
    private String uname;
    private String email;
    private String birth;
    private double balance;

    public Users() {
    }

    public Users(String uname, String email, String birth, double balance) {
        this.uname = uname;
        this.email = email;
        this.birth = birth;
        this.balance = balance;

    }
```

```java

    public Users(int uid, String uname, String email, String birth, double balance) {
        this.uid = uid;
        this.uname = uname;
        this.email = email;
        this.birth = birth;
        this.balance = balance;
    }

    public int getUid() {
        return uid;
    }

    public void setUid(int uid) {
        this.uid = uid;
    }

    public String getUname() {
        return uname;
    }

    public void setUname(String uname) {
        this.uname = uname;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getBirth() {
        return birth;
    }

    public void setBirth(String birth) {
        this.birth = birth;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    @Override
    public String toString() {
        return "Users{" +
                "uid=" + uid +
                ", uname='" + uname + '\'' +
                ", email='" + email + '\'' +
                ", birth='" + birth + '\'' +
                ", balance=" + balance +
                '}';
    }
}

```

```java
IdCard类：
public class IdCard implements Serializable {
    private int id;
    private String cno;
    private int uid;

    public IdCard() {
    }

    public IdCard(String cno, int uid) {
        this.cno = cno;
        this.uid = uid;
    }

    public IdCard(int id, String cno, int uid) {
        this.id = id;
        this.cno = cno;
        this.uid = uid;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCno() {
        return cno;
    }

    public void setCno(String cno) {
        this.cno = cno;
    }

    public int getUid() {
        return uid;
    }

    public void setUid(int uid) {
        this.uid = uid;

    }

    @Override
    public String toString() {
        return "IdCard{" +
                "id=" + id +
                ", cno='" + cno + '\'' +
                ", uid=" + uid +
                '}';
    }
}

```

```java
/**
 * 自定义实体类（完成一对一查询的第一种方法）
 */
public class UsersCustom extends  Users {
    private int id;
    private String cno;     //身份证号
    private int userId;     //外键：users表的主键(代表当前身份证的用户Id)

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCno() {
        return cno;
    }

    public void setCno(String cno) {
        this.cno = cno;
    }

    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    @Override
    public String toString() {
        return "UsersCustom{" +
                "id=" + id +
                ", cno='" + cno + '\'' +
                ", userId=" + userId +
                '}' + super.toString();

    }
}

```

3.1.2)定义接口UsersMapper.java

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

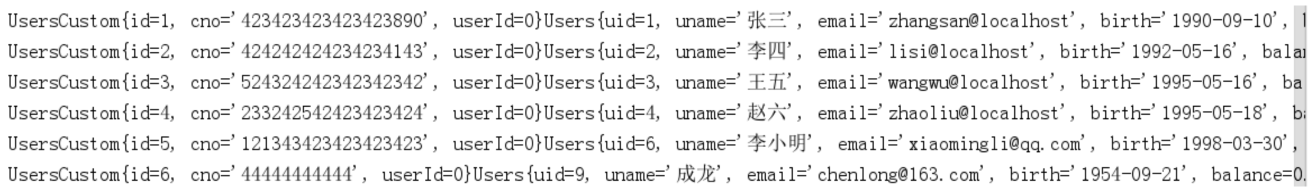
3.1.3)定义UsersMapper.xml的映射文件：

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

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

3.1.4)进行单元测试：

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath*:spring/applicationContext*.xml")
public class TestOneToOne {
    //1.从spring容器中注入sqlSessionFactoryBean
    @Autowired
    private SqlSessionFactoryBean sqlSessionFactoryBean;

    /**
     * 一对一加载的第一种实现方式
     * @throws Exception
     */
    @Test
    public void test01() throws  Exception{
        //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
        SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
        //3.根据工厂对象得到SqlSession对象
        SqlSession sqlSession = factory.openSession();
        UsersMapper mapper = sqlSession.getMapper(UsersMapper.class);
        List<UsersCustom> users = mapper.findUsers();
        for(UsersCustom custom : users){
            System.out.println(custom);

        }
    }
}

```

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

```xml
 <!--一对一查询的方法二（使用自定义结果集映射实现）-->
   <resultMap id="usersMap" type="users">
       <id property="uid" column="uid"/>
       <result property="email" column="email"/>
       <result property="uname" column="uname"/>
       <result property="birth" column="birth"/>
       <result property="balance" column="balance"/>
       <association property="idCard" javaType="idCard" resultMap="idCardMap"/>
   </resultMap>
   <resultMap id="idCardMap" type="idCard">
       <id column="id" property="id"/>
       <result column="cno" property="cno"/>
       <result column="uid" property="uid"/>
   </resultMap>
   <select id="findUsers2" resultMap="usersMap">
       select
       u.uid,uname,email,birth,i.id as id,cno,i.uid as uid
       from users u,idcard i
       where i.uid=u.uid
   </select>
```

3.2.2）在UserMapper接口中添加方法findUsers2

```java
public List<Users> findUsers2() throws Exception;
```

3.2.3）测试

```java
/**
    * 一对一加载的第二种实现方式(使用结果集映射来实现)
    * @throws Exception
    */
   @Test
   public void test02() throws  Exception{

       //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
```

```java
        SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
        //3.根据工厂对象得到SqlSession对象
        SqlSession sqlSession = factory.openSession();
        UsersMapper mapper = sqlSession.getMapper(UsersMapper.class);
        List<Users> users = mapper.findUsers2();
        for(Users user: users){
            System.out.println(user + "--->" + user.getIdCard());
        }
    }
```

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

```xml
<!--一对一查询的方法三（使用自定义结果集映射+懒加载实现）-->
    <resultMap id="usersMap2" type="users">
        <id property="uid" column="uid"/>
        <result property="email" column="email"/>
        <result property="uname" column="uname"/>
        <result property="birth" column="birth"/>
        <result property="balance" column="balance"/>
        <!--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-->
        <association property="idCard" javaType="idCard" column="uid"
                     select="findIdCardByUid" />
    </resultMap>
    <select id="findIdCardByUid" parameterType="int" resultType="idCard">
        select * from idcard where uid = #{value}
    </select>
    <select id="findUsers3" resultMap="usersMap2">
        select * from users
    </select>
```

3.3.2)在UserMapper.java接口中添加方法：

```java
public List<Users> findUsers3() throws Exception;
```

3.3.3)单元测试

```java
/**
     * 一对一加载的第三种实现方式(使用懒加载查询来实现)
     * @throws Exception
     */
    @Test
    public void test03() throws  Exception{
        //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
        SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
        //3.根据工厂对象得到SqlSession对象
        SqlSession sqlSession = factory.openSession();
        UsersMapper mapper = sqlSession.getMapper(UsersMapper.class);
        List<Users> users = mapper.findUsers3();
        for(Users user: users){
            System.out.println(user + "--->" + user.getIdCard());
        }
    }
```

3.4)方法四：使用注解完成

3.4.1)修改IdCardMapper.java接口

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

```

3.4.2)修改UsersMapper.java接口：

```
/**
     * 根据用户id查询用户对象
     * @param uid
     * @return
     * @throws Exception
     */
    @Select("select * from users where uid=#{value}")
    public Users findUserByUid(int uid) throws Exception;
```

3.4.3)测试：

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

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

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.zelin.mapper.CategoryMapper">
    <!--1.一对多查询-->
   <resultMap id="categoryMap" type="category">
       <id column="cid" property="cid"/>
       <result column="cname" property="cname"/>
       <!--查询在类别中的所有的图书-->
       <collection property="books" ofType="book" javaType="list">
           <id column="bid" property="bid"/>
           <result column="bname" property="bname"/>
           <result column="bauthor" property="bauthor"/>
           <result column="publisher" property="publisher"/>
           <result column="cid" property="cid"/>
           <result column="imgpath" property="imgpath"/>
       </collection>
   </resultMap>
    <!--1.查询所有的图书类别-->
    <select id="findAll" resultMap="categoryMap">
        select b.*,cname from category c,book b
        where b.cid=c.cid
    </select>
</mapper>
```

4.2)定义CategoryMapper.java这个接口

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

4.3)测试

```java
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath*:spring/applicationContext*.xml")
public class TestOneToMany {

    //1.从spring容器中注入sqlSessionFactoryBean
    @Autowired
    private SqlSessionFactoryBean sqlSessionFactoryBean;
    @Test
    public void test01() throws  Exception{
        //2.根据sqlSessionFactoryBean得到sqlSessionFactory对象
        SqlSessionFactory factory = sqlSessionFactoryBean.getObject();
        //3.根据工厂对象得到SqlSession对象
        SqlSession sqlSession = factory.openSession();
        CategoryMapper mapper = sqlSession.getMapper(CategoryMapper.class);
        List<Category> cates = mapper.findAll();
        for(Category category : cates){
            System.out.println(category + "-->" + category.getBooks());
        }
    }
}
```

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

```xml
<!--2.一对多查询(第二种方式：使用关联查询完成)-->
    <resultMap id="categoryMap2" type="category">
        <id column="cid" property="cid"/>
        <result column="cname" property="cname"/>
        <collection property="books" javaType="list" ofType="book"
                    select="findBooksByCid" column="cid"/>
    </resultMap>
    <select id="findBooksByCid" parameterType="int" resultType="book">
        select * from book where cid=#{value}
    </select>
    <select id="findAll2" resultMap="categoryMap2">
        select * from category
    </select>
```

4.2.2)修改CategoryMapper.java接口

```java
public List<Category> findAll2() throws  Exception;
```

4.2.3)测试：

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

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

```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">
<!--动态SQL查询-->
<mapper namespace="com.zelin.mapper.BookMapper">
   <!--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。-->
    <!--1.1)定义进行条件查询的语句-->
    <sql id="dynamicSQL" >
        select * from book
        <where>
            <if test="bname != null and bname != ''">
               and bname like '%${bname}%'
            </if>
            <if test="bauthor != null and bauthor != ''">
               and  bauthor like '%${bauthor}%'
            </if>
            <if test="publisher != null and publisher != ''">
               and publisher like '%${publisher}%'
            </if>
            <if test="cid != 0">
              and cid = #{cid}
            </if>
        </where>

    </sql>
    <!--条件查询-->
    <select id="findBooksByKeywords" parameterType="bookVo" resultType="book">
      <!-- 将动态sql语句包含进来-->
        <include refid="dynamicSQL"/>

    </select>
```

```
</mapper>
```

5.1.2）定义BookMapper.java接口

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

5.1.3）测试：

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

}
```

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

```xml
<!--条件查询二：根据多个图书编号查询图书列表-->
    <sql id="dynamicSQL2">
        select * from book
        <where>
            bid in
            <foreach collection="ids" item="id" open="(" close=")" separator=",">
                #{id}
            </foreach>
        </where>
    </sql>
    <!--条件查询二：根据多个id查询图书-->
    <select id="findBooksByIds" parameterType="bookVo" resultType="book">
```

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

5.3.2) 修改BookMapper.java接口

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

5.3.3) 测试：

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

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

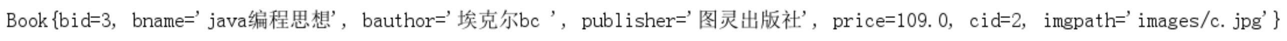
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--动态SQL查询-->
<mapper namespace="com.zelin.mapper.BookMapper">
   <!--1.功能一：根据图书名称、图书作者、出版社及图书类别编号查询图书。-->
    <!--1.1)定义进行条件查询的语句-->
    <sql id="dynamicSQL" >
        <where>
            <if test="bname != null and bname != ''">
                and bname like '%${bname}%'
            </if>
            <if test="bauthor != null and bauthor != ''">
                and  bauthor like '%${bauthor}%'
            </if>
            <if test="publisher != null and publisher != ''">
                and publisher like '%${publisher}%'

            </if>
```

```xml
            <if test="cid != 0">
                and cid = #{cid}
            </if>
        </where>

    </sql>
    <!--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。-->
    <select id="findBooksByKeywords" parameterType="bookVo" resultType="book">
        select * from book
      <!-- 将动态sql语句包含进来-->
        <include refid="dynamicSQL"/>
    </select>

    <!--条件查询二：根据多个图书编号查询图书列表-->
    <sql id="dynamicSQL2">
        <where>
            bid in
            <foreach collection="ids" item="id" open="(" close=")" separator=",">
                #{id}
            </foreach>
        </where>

    </sql>
    <!--条件查询二：根据多个id查询图书-->
    <select id="findBooksByIds" parameterType="bookVo" resultType="book">
        select * from book
        <!-- 将动态sql语句包含进来-->
        <include refid="dynamicSQL2"/>
    </select>

    <sql id="dynamicSQL3">
        <where>
            <if test="bname != null and bname != ''">
                and bname like '%${bname}%'
            </if>
            <if test="bauthor != null and bauthor != ''">
                and  bauthor like '%${bauthor}%'
            </if>
            <if test="publisher != null and publisher != ''">
                and publisher like '%${publisher}%'
            </if>
            <if test="cid != 0">
                and cid = #{cid}
            </if>
            <if test="ids != null and ids.size() > 0">
                and bid in
                <foreach collection="ids" item="id" open="(" close=")" separator=",">
                    #{id}
                </foreach>
            </if>
        </where>
    </sql>


    <!--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。-->
    <select id="findBooksByCond" parameterType="bookVo" resultType="book">
            select * from book
            <include refid="dynamicSQL3"/>
    </select>
</mapper>
```

5.4.2）修改BookMapper.java接口

```java
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'}
```