MyBatis框架

1. MyBatis框架简介

1.1 框架概念

> 框架，就是软件的半成品，完成了软件开发过程中的通用操作，使用很少的代码就能够完成特定的功能，从而简化开发人员的开发步骤，提高开发效率。

1.2 常用的框架

- 持久层框架：完成数据库操作的框架
  - MyBatis
- MVC框架： 简化了servlet的开发步骤
  - SpringMVC
- 胶水框架: 主要与其他框架进行整合
  - Spring

`简称： SSM框架-----简化jdbc+servlet+jsp`

- 简化SSM框架
  - Springboot框架

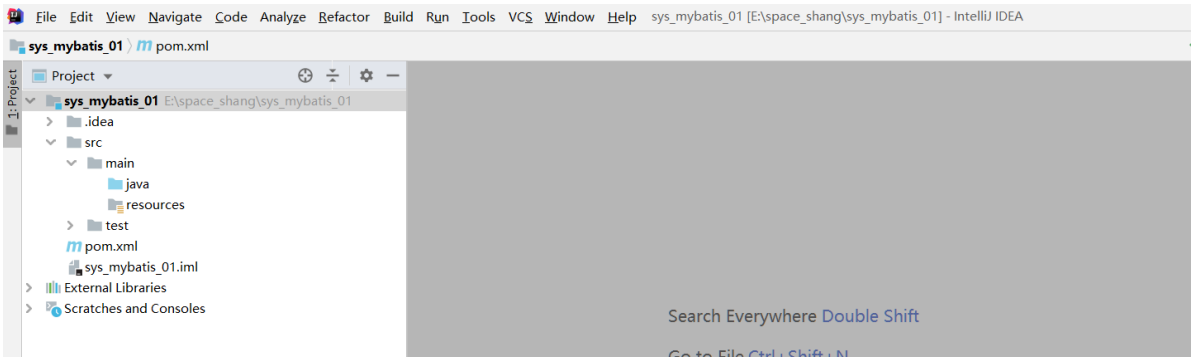
1.3 MyBatis框架

> MyBatis是一个 `半自动` 的 `ORM` 框架
>
> ORM(Object Relational Mapping) 对象关系映射： 将Java中的一个对象与数据库表的一行记录一一对应。
>
> ORM框架提供了实体类与数据表的映射关系，通过映射文件的配置，实现对象的持久化。

- MyBatis的前身是IBatis,是Apache提供的一个开源的项目
- MyBatis的特点:
  - 支持自定义SQL、存储过程
  - 对原有的JDBC进行了封装，消除了所有的JDBC的代码
  - 支持XML和注解配置方式完成ORM操作，实现映射结果

2. MyBatis框架部署

2.1 创建maven项目



2.2 导入依赖

```
  <dependencies>
     <!--    mybatis依赖   -->
         <dependency>
             <groupId>org.mybatis</groupId>
             <artifactId>mybatis</artifactId>
             <version>3.5.10</version>
         </dependency>
     <!---mysql依赖-->
         <dependency>
             <groupId>mysql</groupId>
             <artifactId>mysql-connector-java</artifactId>
             <version>8.0.29</version>
         </dependency>

         <dependency>
             <groupId>org.projectlombok</groupId>
             <artifactId>lombok</artifactId>
             <version>1.18.24</version>
         </dependency>

           <dependency>
             <groupId>junit</groupId>
             <artifactId>junit</artifactId>
             <version>4.12</version>
         </dependency>

     </dependencies>
```

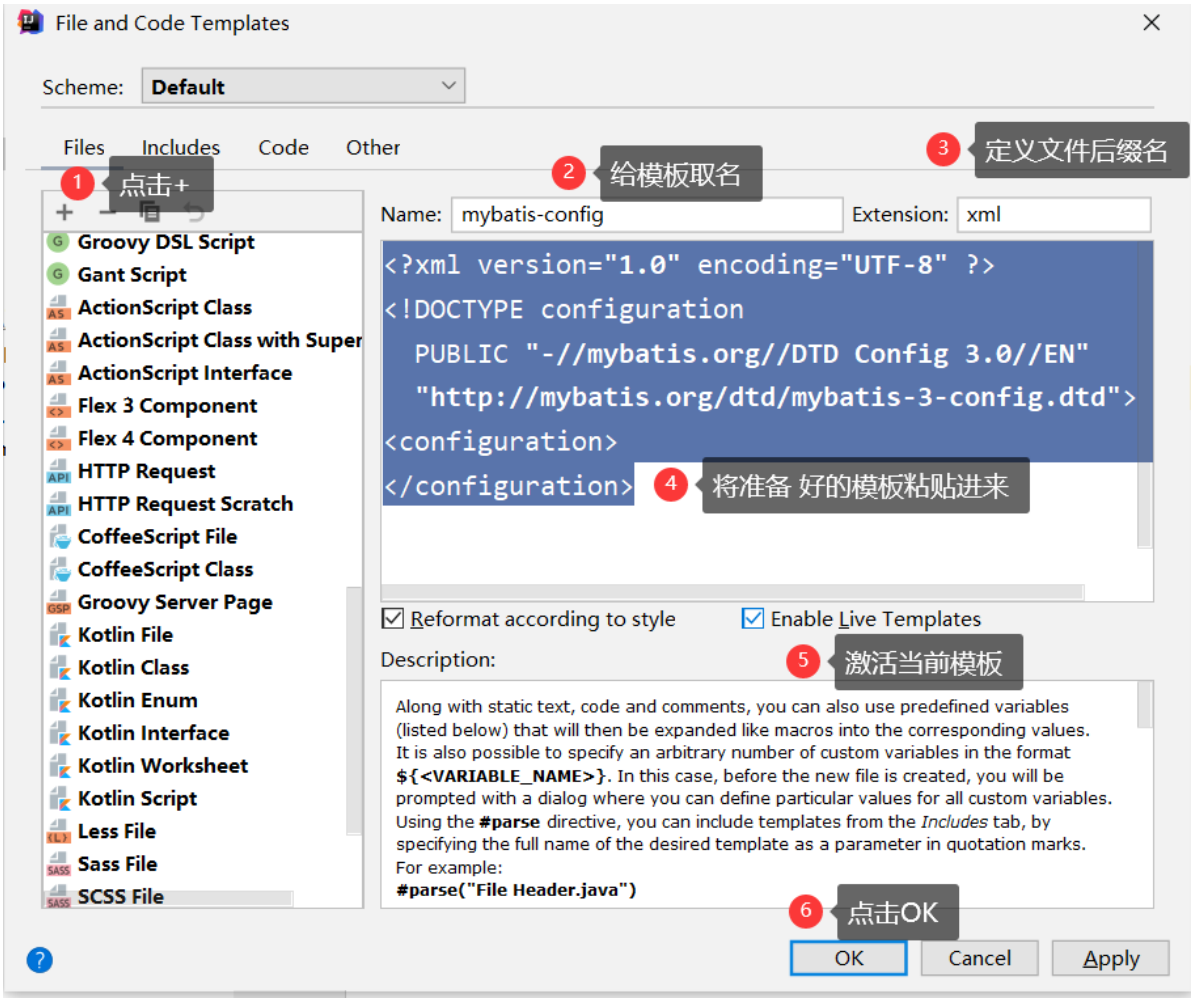
2.3 创建实体类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * 实体类:  orm框架映射关系
 */
@Data
@AllArgsConstructor  //有参构造
@NoArgsConstructor  //无参构造
```

```
public class Users {

    private int id;

    private String username;

    private String password;


}
```

2.4 创建mybatis的主配置文件

- 创建自定义模板： 选择resources----右键new-----Edit File Templates



- 在resources中创建名为mybatis-config.xml的文件
- 在mybatis-config.xml文件配置数据库的相关信息

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

    <!--配置数据库连接信息-->
    <!--
            在environments标签中可以定义多个environment标签，每个environment标签可以定义一套连接配置
      -->
```

```
    <environments default="mysql">

        <environment id="mysql">
            <!--
                transactionManager标签用于配置数据库管理方式
            -->
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/db?
serverTimezone=UTC"/>
                <property name="username" value="root"/>
                <property name="password" value="123456"/>
            </dataSource>
        </environment>

    </environments>


</configuration>
```

2.5 创建DAO接口

> 一个接口中可以定义多个方法，每个方法 就是对应一个功能。

```
package org.qf.dao;

import org.qf.entity.Users;

/**
 * dao接口
 */
public interface UsersMapper {

    /**
     * 登录功能
     * @param username
     * @return
     */
    public Users login(String username);

    /**
     * 注册功能
     * @param username
     * @param password
     * @return
     */
    public int register(String username,String password);


}
```

2.6 创建Dao接口的映射文件

> mybatis的映射文件

- 在 resources目录下，创建文件夹mappers,在文件夹下创建名为:UsersMapper.xml
- 在映射文件中实现dao接口定义的方法操作

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace: 指向的是对应接口的全限定名（从包名开始找对应的类名）-->
<mapper namespace="org.qf.dao.UsersMapper">

     <!-- 登录-->
<!--
    id:指向的是接口的对应的方法名称
    resultType:当返回值是对象或者集合时，要添加resultType属性，属性值为该对象的全限定名
    当方法中传入有参数，参数是基本数据类型（int String） 不需要 添加parameterType属性，
    如果传入的是对象，必须要添加parameterType,属性值为该对象的全限定名
-->
    <select id="login" resultType="org.qf.entity.Users">
        select * from users where username=#{username}
    </select>

<!--

        当返回值是int、String等基本数据类型，不需要添加resultType
-->
    <insert id="register">
        insert into users(username,password)values(#{username},#{password})
    </insert>



</mapper>
```

2.7 将映射文件添加到主配置文件中

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

    <!--配置数据库连接信息-->
    <!--
            在environments标签中可以定义多个environment标签，每个environment标签可以定义一套连接配置
      -->
    <environments default="mysql"...>

<!--    调用映射文件-->
    <mappers>
        <mapper resource="mappers/UsersMapper.xml"></mapper>
    </mappers>


</configuration>
```

3. 单元测试

3.1 添加单元测试依赖

```
 <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
```

3.2 测试代码

```
package org.qf.userTest;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Test;
import org.qf.dao.UsersMapper;
import org.qf.entity.Users;

import java.io.IOException;
import java.io.InputStream;

public class UserTest {

    @Test
    public void  show(){
        //1.加载主配置文件
        try {
            InputStream is= Resources.getResourceAsStream("mybatis-config.xml");

            //2.创建会话工厂
            SqlSessionFactory factory=new SqlSessionFactoryBuilder().build(is);

            //3.创建会话
            SqlSession sqlSession = factory.openSession();

            //4. 获取dao接口
            UsersMapper mapper = sqlSession.getMapper(UsersMapper.class);

            int lisi = mapper.register("bb", "123");
            if(lisi>0){
                System.out.println("注册成功");
            }else{
                System.out.println("注册失败");
            }


//
//            Users admin = mapper.login("admin");
//
//            if(admin != null){
//                if(admin.getPassword().equals(("123456"))){
```

```
//                    System.out.println("登录成功");
//                }else{
//                    System.out.println("登录失败");
//                }
//            }



        } catch (IOException e) {
            e.printStackTrace();
        }
    }


}
```

4. MyBatis的CRUD操作

4.1 查询操作-模糊查询

- 在usersMapper接口中定义方法

```
package org.qf.dao;

import lombok.experimental.PackagePrivate;
import org.apache.ibatis.annotations.Param;
import org.qf.entity.Users;

import java.util.List;

/**
 * dao接口
 */
public interface UsersMapper {

      /**
     * 模糊查询所有
     * @return
     */
    public List<Users> showList(Users users);


}
```

- 在usersMapper映射文件 中 定义 实现

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<!--namespace: 指向的是对应接口的全限定名（从包名开始找对应的类名）-->
<mapper namespace="org.qf.dao.UsersMapper">

    <select id="showList" parameterType="org.qf.entity.Users"
 resultType="org.qf.entity.Users">
        select * from users where username like concat('%',#{username},'%') and
password=#{password}
    </select>


</mapper>
```

4.2 查询操作-条件查询

- 在usersMapper接口中定义方法

```
/**
     * 条件查询
     * @param id
     * @return
     */
    public Users  showUserById(int id);
```

- 在usersMapper映射文件 中 定义 实现

```
  <select id="showUserById" resultType="org.qf.entity.Users">
        select * from users where id=#{id}
    </select>
```

4.3 删除操作-通过主键删除

- 在usersMapper接口中定义方法

```
 /**
     * 通过id删除
     * @param id
     * @return
     */
    public int deleteUser(int id);
```

- 在usersMapper映射文件 中 定义 实现

```
  <delete id="deleteUser">
        delete from users where id=#{id}
  </delete>
```

5. 关联映射

5.1 实体关系

实体----数据实体　实体关系指的是数据与数据之间的关系
例如：　用户和角色　　房屋和楼栋　　订单和商品

5.2 实体关系分类

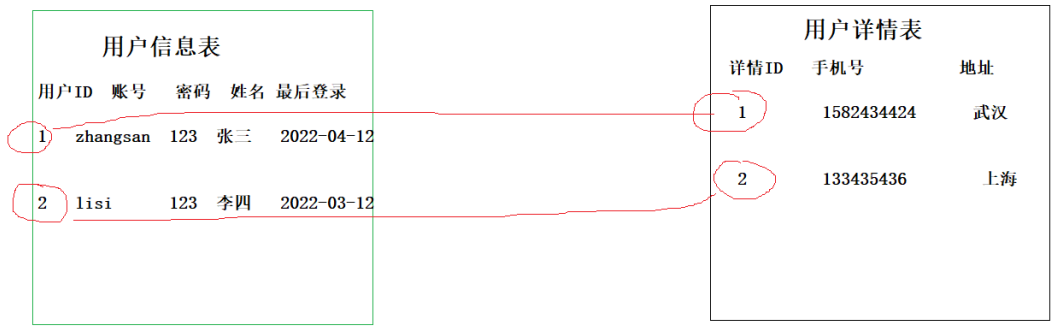
1. 一对一的关系
2. 一对多的关系
3. 多对一的关系
4. 多对多的关系

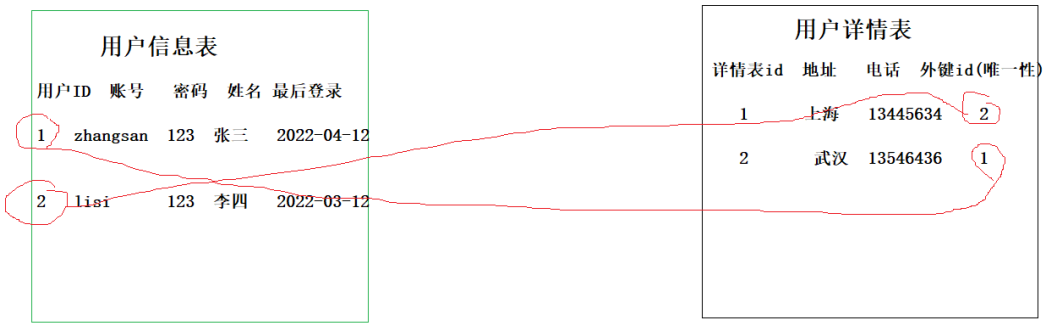
5.3 一对一的关系

案例：　用户信息表和用户详情表

数据表关系：

- 用户表主键和详情表主键相同时，表示是匹配的数据
- 唯一 的外键





5.3.1 案例

用户信息表------用户详情表

在查询用户的同时关联查询出与之对应的详情

1. 创建数据表

----------　用户信息表----------

```sql
CREATE TABLE `users` (
  `user_id` int NOT NULL AUTO_INCREMENT,
  `user_name` varchar(255) DEFAULT NULL,
  `user_pwd` varchar(255) DEFAULT NULL,
  `user_realname` varchar(255) DEFAULT NULL,
  `user_img` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;


-----------------用户详情表-----------------
CREATE TABLE `details` (
  `detail_id` int NOT NULL AUTO_INCREMENT,
  `user_addr` varchar(255) DEFAULT NULL,
  `user_tel` varchar(255) DEFAULT NULL,
  `user_desc` varchar(255) DEFAULT NULL,
  `uid` int NOT NULL,
  PRIMARY KEY (`detail_id`),
  UNIQUE KEY `u_ids` (`uid`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

2. 创建实体类

- Users

```java
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * 实体类： orm框架映射关系
 */
@Data
@AllArgsConstructor  //有参构造
@NoArgsConstructor  //无参构造
public class Users {

   private  int userId;

   private String userName;

   private String  userPwd;

   private String userRealname;

   private String userImg;

   private Details details;  //一对一的关系

}
```

- Details

```java
package org.qf.entity;
```

```java
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Details {

    private int detailId;

    private String userAddr;

    private String userTel;

    private String userDesc;

    private int userId;  //外键


}
```

3. 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">
<!--namespace: 指向的是对应接口的全限定名（从包名开始找对应的类名）-->
<mapper namespace="org.qf.dao.UsersMapper">

    <resultMap id="userMap" type="org.qf.entity.Users">
        <id column="user_id" property="userId"></id>
        <result column="user_name" property="userName"></result>
        <result column="user_pwd" property="userPwd"></result>
        <result column="user_realname" property="userRealname"></result>
        <!-- 一对一的关联  -->
        <result column="detail_id" property="details.detailId"></result>
        <result column="user_addr" property="details.userAddr"></result>
        <result column="user_tel" property="details.userTel"></result>
        <result column="user_desc" property="details.userDesc"></result>
    </resultMap>

   <select id="showList" resultMap="userMap">
    select * from users u inner join details d on u.user_id=d.uid where u.user_name=#
{userName}
   </select>

</mapper>
```

5.4 一对多和多对一的关系

> 案例：
>
> - 一对多：班级和学生 类别和商品 楼栋和房屋
> - 多对一： 学生和班级 商品和类别
>
> 数据表关系：
>
> - 在多的一端添加外键和一的一端进行关联。

- 数据库表设计

> 班级表与学生表

```
--------------------班级表-----------------------------------
CREATE TABLE `classes` (
  `cid` int NOT NULL AUTO_INCREMENT,
  `cname` varchar(255) DEFAULT NULL,
  `cdesc` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`cid`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

------------------学生表-------------------------------------
CREATE TABLE `student` (
  `sid` int NOT NULL AUTO_INCREMENT,
  `sname` varchar(255) DEFAULT NULL,
  `sage` varchar(255) DEFAULT NULL,
  `scid` int NOT NULL,
  PRIMARY KEY (`sid`),
  KEY `s_id` (`scid`),
  CONSTRAINT `s_id` FOREIGN KEY (`scid`) REFERENCES `classes` (`cid`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

5.4.1 一对多的案例

> 查询班级下对应的学生信息
>
> - 查询1班对应的学生信息

1. 实体类

- 班级类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Classes {
```

```
    private int cid;

    private String cname;

    private String cdesc;

    private List<Student> stus;  //存储当前班级下的学生信息


}
```

- 学生类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    private int sid;

    private String sname;

    private int sage;

    private int scid;
}
```

2. 映射文件

```
<?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">
<mapper namespace="org.qf.dao.ClassesMapper">

    <resultMap id="cMap" type="org.qf.entity.Classes">
        <id column="cid" property="cid"></id>
        <result column="cname" property="cname"></result>
        <result column="cdesc" property="cdesc"></result>
       <!-- 一对多的关联关系进行关联 -->
        <collection property="stus" ofType="org.qf.entity.Student">
            <id property="sid" column="sid"></id>
            <result property="sname" column="sname"></result>
            <result property="sage" column="sage"></result>
        </collection>
    </resultMap>

    <select id="showClass" resultMap="cMap">
        select * from classes c INNER JOIN student s on c.cid=s.scid where c.cname=#
{cname}
```

```
    </select>

</mapper>
```

5.4.2 多对一的案例

> 案例： 学生---班级
>
> - 当查询一个学生的时候，关联查询这个学生所在的班级信息

1. 实体类

- 班级类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Classes {

    private int cid;

    private String cname;

    private String cdesc;

}
```

- 学生类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    private int sid;

    private String sname;

    private int sage;

    private int scid;

    private Classes classes;  //学生所在的班级
}
```

2. 映射文件

```
<?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">
<mapper namespace="org.qf.dao.StudentMapper">

    <resultMap id="stuMap" type="org.qf.entity.Student">
        <id column="sid" property="sid"></id>
        <result column="sname" property="sname"></result>
        <result column="sage" property="sage"></result>
        <result column="cname" property="classes.cname"></result>
        <result column="cdesc" property="classes.cdesc"></result>
    </resultMap>

    <select id="showStudent" resultMap="stuMap">
        select * from student s INNER JOIN classes c on s.scid=c.cid where s.sname=#
{sname}
    </select>
</mapper>
```

5.5 多对多的关系

> 案例：
>
> 用户和角色 角色和权限 订单和商品
>
> 数据表关系：
>
> - 建立第三张中间关系表添加 两个外键分别与两张表主键进行关联

- 数据库表的设计

```
--------------------------学生表----------------------------------
CREATE TABLE `student` (
  `sid` int NOT NULL AUTO_INCREMENT,
  `sname` varchar(255) DEFAULT NULL,
  `sage` int DEFAULT NULL,
  PRIMARY KEY (`sid`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-----------------------课程表-----------------------------------
CREATE TABLE `course` (
  `course_id` int NOT NULL AUTO_INCREMENT,
  `course_name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`course_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

----------------------中间表-------------------------------------------
CREATE TABLE `stu_course` (
  `id` int NOT NULL AUTO_INCREMENT,
  `sid` int DEFAULT NULL,
  `cid` int DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

> 学生----------课程
>
> - 查询学生时，同时查询该学生选择的课程

1. 实体类

- 学生类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    private int sid;

    private String sname;

    private int sage;

    private List<Course> courses;  //学生选择的课程

}
```

- 课程类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Course {

    private int courseId;

    private String courseName;
}
```

2. 映射文件

```
<?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">
<mapper namespace="org.qf.dao.StudentMapper">

    <resultMap id="stuMap" type="org.qf.entity.Student">
```

```
        <id column="sid" property="sid"></id>
        <result column="sname" property="sname"></result>
        <result column="sage" property="sage"></result>
        <collection property="courses" ofType="org.qf.entity.Course">
            <result column="course_name" property="courseName"></result>
        </collection>
    </resultMap>

    <select id="showStudent" resultMap="stuMap">
        select * from student s INNER JOIN stu_course t INNER JOIN course c
on s.sid=t.sid and t.cid=c.course_id where s.sname=#{sname}
    </select>
</mapper>
```

- 根据课程名称查询课程信息时，同时查询选择了该门课程的学生

1. 学生类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    private int sid;

    private String sname;

    private int sage;

}
```

2. 课程类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Course {

    private int courseId;

    private String courseName;
```

```
    private List<Student> stus;  //该课程下选修的学生
}
```

2. 映射文件

```
<?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">
<mapper namespace="org.qf.dao.CourseMapper">

    <resultMap id="sMap" type="org.qf.entity.Course">
        <id column="course_id" property="courseId"></id>
        <result column="course_name" property="courseName"></result>
        <collection property="stus" ofType="org.qf.entity.Student">
            <id column="sid" property="sid"></id>
            <result column="sname" property="sname"></result>
            <result column="sage" property="sage"></result>
        </collection>
    </resultMap>

    <select id="showCourse" resultMap="sMap">
        select * from course c INNER JOIN stu_course  t INNER JOIN student s
on c.course_id=t.cid and t.sid=s.sid where c.course_name=#{courseName}
    </select>
</mapper>
```

6. 动态SQL

6.1 动态SQL

> 根据查询条件动态完成SQL的拼接

6.2 案例

1. 创建数据库表

```
CREATE TABLE `members` (
  `member_id` int NOT NULL AUTO_INCREMENT,
  `member_nick` varchar(255) DEFAULT NULL,
  `member_gender` varchar(255) DEFAULT NULL,
  `member_age` int DEFAULT NULL,
  `member_city` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`member_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

2. 创建实体类

```
package org.qf.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Members {

    private int memberId;

    private String memberNick;

    private String  memberGender;

    private int memberAge;

    private String memberCity;
}
```

3. 创建dao接口

> 在DAO接口中定义一个多条件查询的方法

```
package org.qf.dao;

import org.qf.entity.Members;

import java.util.List;
import java.util.Map;

public interface MembersMapper {

   //多条件查询中，如果查询条件不确定，可以直接使用map（hashmap）作为参数
    //优点：无需单独传递查询条件的类
    //缺点：当向map中存放参数时，key必须与动态SQL保持一致

    public List<Members> showList(Map<String,Object> map);

}
```

6.3 if

- 映射文件

```
<?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">
<mapper namespace="org.qf.dao.MembersMapper">

    <resultMap id="memberMap" type="org.qf.entity.Members">
        <!--
            column:对应的是数据表中的字段名
            property：实体类 的属性名
        -->
        <id  column="member_id" property="memberId"></id>
        <result column="member_nick" property="memberNick"></result>
        <result column="member_gender" property="memberGender"></result>
        <result column="member_age" property="memberAge"></result>
```

```
        <result column="member_city" property="memberCity"></result>
    </resultMap>

    <select id="showList" resultMap="memberMap">
        select * from members where 1=1
        <if test="id != null">  <!-- id 就是参数对象的属性/参数map的key值-->
            and member_id=#{id}
        </if>
        <if test="nick != null">
            and member_nick=#{nick}
        </if>
        <if test="age != null">
            and member_age=#{age}
        </if>
        <if test="gender != null">
            and member_gender=#{gender}
        </if>
    </select>

</mapper>
```

- 测试

```
InputStream resourceAsStream = Resources.getResourceAsStream("mybatis-config.xml");

SqlSessionFactory factory=new SqlSessionFactoryBuilder().build(resourceAsStream);

SqlSession sqlSession=factory.openSession();

MembersMapper mapper = sqlSession.getMapper(MembersMapper.class);

Map<String,Object> map=new HashMap<String, Object>();
map.put("gender","男");

List<Members> members = mapper.showList(map);

for(Members m:members){
    System.out.println(m.toString());
}
```

6.4 where

```
<?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="org.qf.dao.MembersMapper">

    <resultMap id="memberMap" type="org.qf.entity.Members">
        <!--
            column:对应的是数据表中的字段名
            property: 实体类 的属性名
        -->
        <id  column="member_id" property="memberId"></id>
        <result column="member_nick" property="memberNick"></result>
        <result column="member_gender" property="memberGender"></result>
        <result column="member_age" property="memberAge"></result>
```

```
        <result column="member_city" property="memberCity"></result>
    </resultMap>

    <select id="showList" resultMap="memberMap">
        select * from members
        <where>
            <if test="id != null">  <!-- id 就是参数对象的属性/参数map的key值-->
                and member_id=#{id}
            </if>
            <if test="nick != null">
                and member_nick=#{nick}
            </if>
            <if test="age != null">
                and member_age=#{age}
            </if>
            <if test="gender != null">
                and member_gender=#{gender}
            </if>
        </where>
        order by member_age desc
    </select>

</mapper>
```

6.5 trim

```
<?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="org.qf.dao.MembersMapper">

    <resultMap id="memberMap" type="org.qf.entity.Members">
        <!--
            column:对应的是数据表中的字段名
            property: 实体类 的属性名
        -->
        <id  column="member_id" property="memberId"></id>
        <result column="member_nick" property="memberNick"></result>
        <result column="member_gender" property="memberGender"></result>
        <result column="member_age" property="memberAge"></result>
        <result column="member_city" property="memberCity"></result>
    </resultMap>

    <select id="showList" resultMap="memberMap">
        select * from members
        <trim prefix="where" prefixOverrides="and | or" suffix="order by member_age">

            <if test="id != null">  <!-- id 就是参数对象的属性/参数map的key值-->
                and member_id=#{id}
            </if>
            <if test="nick != null">
                and member_nick=#{nick}
            </if>
            <if test="age != null">
                and member_age=#{age}
            </if>
```

```
            <if test="gender != null">
                and member_gender=#{gender}
            </if>

        </trim>
    </select>

</mapper>
```

- 修改操作

```
<?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="org.qf.dao.MembersMapper">

    <resultMap id="memberMap" type="org.qf.entity.Members">
        <!--
            column:对应的是数据表中的字段名
            property: 实体类 的属性名
        -->
        <id  column="member_id" property="memberId"></id>
        <result column="member_nick" property="memberNick"></result>
        <result column="member_gender" property="memberGender"></result>
        <result column="member_age" property="memberAge"></result>
        <result column="member_city" property="memberCity"></result>
    </resultMap>

    <select id="showList" resultMap="memberMap">
        select * from members
        <trim prefix="where" prefixOverrides="and | or" suffix="order by member_age">

            <if test="id != null">  <!-- id 就是参数对象的属性/参数map的key值-->
                and member_id=#{id}
            </if>
            <if test="nick != null">
                and member_nick=#{nick}
            </if>
            <if test="age != null">
                and member_age=#{age}
            </if>
            <if test="gender != null">
                and member_gender=#{gender}
            </if>

        </trim>
    </select>


    <update id="updateMember" parameterType="org.qf.entity.Members">
        update members
        <trim prefix="set" suffixOverrides=",">
                <if test="memberNick != null">
                    member_nick=#{memberNick},
                </if>
                <if test="memberGender != null">
                    member_gender=#{memberGender},
```

```
                </if>
        </trim>
        <where>
            member_id=#{memberId}
        </where>
    </update>

</mapper>
```

6.6 set

```
<?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="org.qf.dao.MembersMapper">

    <resultMap id="memberMap" type="org.qf.entity.Members">
        <!--
            column:对应的是数据表中的字段名
            property: 实体类 的属性名
        -->
        <id  column="member_id" property="memberId"></id>
        <result column="member_nick" property="memberNick"></result>
        <result column="member_gender" property="memberGender"></result>
        <result column="member_age" property="memberAge"></result>
        <result column="member_city" property="memberCity"></result>
    </resultMap>

    <select id="showList" resultMap="memberMap">
        select * from members
        <trim prefix="where" prefixOverrides="and | or" suffix="order by member_age">

            <if test="id != null">  <!-- id 就是参数对象的属性/参数map的key值-->
                and member_id=#{id}
            </if>
            <if test="nick != null">
                and member_nick=#{nick}
            </if>
            <if test="age != null">
                and member_age=#{age}
            </if>
            <if test="gender != null">
                and member_gender=#{gender}
            </if>

        </trim>
    </select>


    <update id="updateMember" parameterType="org.qf.entity.Members">
        update members
        <set>
                <if test="memberNick != null">
                    member_nick=#{memberNick},
                </if>
                <if test="memberGender != null">
```

```
                    member_gender=#{memberGender}
                </if>
            </set>
            <where>
                member_id=#{memberId}
            </where>
        </update>

</mapper>
```

7.日志

> 便于查看SQL语句等很快定位到错误 信息

7.1导入依赖

```
<!--日志包-->
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>1.2.17</version>
        </dependency>

        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.25</version>
        </dependency>
```

7.2 导入log4j.properties

一般放在resources下