Mybatis

1.Mybatis入门

- 1.1 mybatis介绍
- 1.2 Jdbc编码的问题
- 1.3 mybatis架构
- 1.4 环境搭建
- 1.5 实现根据id查询用户信息
  - 1.5.1 准备主配置文件
  - 1.5.2 编写映射文件
  - 1.5.3 主配置文件加载映射文件
  - 1.5.4 编写Java代码
- 1.6 根据用户姓名进行模糊查询1
  - 1.6.1 编写sql
  - 1.6.2 编写映射文件
  - 1.6.3 测试代码
- 1.7 根据用户名进行模糊查询2
  - 1.7.1 编写映射文件
  - 1.7.2 测试
  - 1.7.3 原理
- 1.8 #{}和${}区别
- 1.9 selectOne和selectList区别?

2.crud

- 2.1 插入
  - 2.1.1 准备sql
  - 2.1.2 编写xml
  - 2.1.3 测试
- 2.2 修改
  - 2.2.1 编写sql
  - 2.2.2 编写xml

2.2.3 测试
2.3 删除
2.3.1 编写sql
2.3.2 编写xml
2.3.3 测试
2.4 mysql自增主键的返回(重点)
2.4.1 自增主键的原理
2.4.2 编写xml
2.4.3 测试
2.5 mysql自增主键的简化写法
2.6 使用mysql的uuid实现主键(了解)
3.原始dao开发
3.1 编写工具类
3.2 编写接口
3.3 编写dao实现
3.4 编写xml
3.5 测试
3.6 思考
4.Mapper动态代理开发
4.1 mapper动态代理开发规范
4.2 项目准备
4.3 编写mapper接口
4.4 编写mapper.xml
4.5 测试
5.主动配置文件
5.1 加载映射文件
5.1.1 直接写上xml名字
5.1.2 直接写上接口的全限定名
5.1.3 批量加载
5.2 mybatis别名
5.2.1 mybatis默认配置的别名
5.2.2 自定义别名

5.3 加载外部的properties文件
6.输入映射
6.1 输入映射的介绍
6.2 传递简单类型
6.3 传递pojo类型
6.4 传递包装的pojo类型
6.4.1 编写包装的pojo
6.4.2 编写接口
6.4.3 编写xml
6.4.4 测试
6.5 传递map集合
6.5.1 编写接口
6.5.2 编写xml
6.5.3 测试
6.6 通过@param注解来标记参数 **
7.输出映射
7.1 输出简单类型
7.1.1 编写接口
7.1.2 编写xml
7.1.3 测试
7.2 输出pojo类型
7.3 输出map集合(不推荐)
7.4 resultMap
7.4.1 订单表结构
7.4.2 查询订单列表
7.4.2.1 编写mapper接口
7.4.2.2 编写xml
7.4.2.3 测试
7.4.3 取别名(方式一)
7.4.4 resultMap(方式二)
7.4.5 配置自动转换（下划线自动转驼峰）方式三
8.动态sql

8.1 if标签
8.1.1 多条件查询
8.1.2 if标签改造
8.2 where标签
8.3 sql片段
8.4 foreach标签
9.关联查询
9.1 一对一查询
9.1.1 编写sql
9.1.2 编写vo接口结果
9.1.3 编写接口
9.1.4 编写xml
9.1.5 测试
9.2 一对一查询(方式二)
9.2.1 修改pojo
9.2.2 编写xml
9.3 一对多的查询
9.3.1 修改pojp
9.3.2 编写xml
9.3.3 测试
10.缓存(了解)
10.1 缓存的介绍
10.2 一级缓存
10.3 二级缓存(了解)
10.3.1 验证二级缓存默认是否存在
10.3.2 开启二级缓存
10.4 懒加载

1.Mybatis入门

1.1 mybatis介绍

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github。

MyBatis是一个优秀的持久层框架，它对jdbc的操作数据库的过程进行封装，使开发者只需要关注 SQL本身，而不需要花费精力去处理例如注册驱动、创建connection、创建statement、手动设置参数、结果集检索等jdbc繁杂的过程代码。

Mybatis通过xml或注解的方式将要执行的各种statement（statement、preparedStatemnt、CallableStatement）配置起来，并通过java对象和statement中的sql进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射成java对象并返回。

1.2 Jdbc编码的问题

1、 数据库连接创建、释放频繁造成系统资源浪费，从而影响系统性能。如果使用数据库连接池可解决此问题。

2、 Sql语句在代码中硬编码，造成代码不易维护，实际应用中sql变化的可能较大，sql变动需要改变java代码。

3、 使用preparedStatement向占有位符号传参数存在硬编码，因为sql语句的where条件不一定，可能多也可能少，修改sql还要修改代码，系统不易维护。

4、 对结果集解析存在硬编码（查询列名），sql变化导致解析代码变化，系统不易维护，如果能将数据库记录封装成pojo对象解析比较方便。

1 dbutils

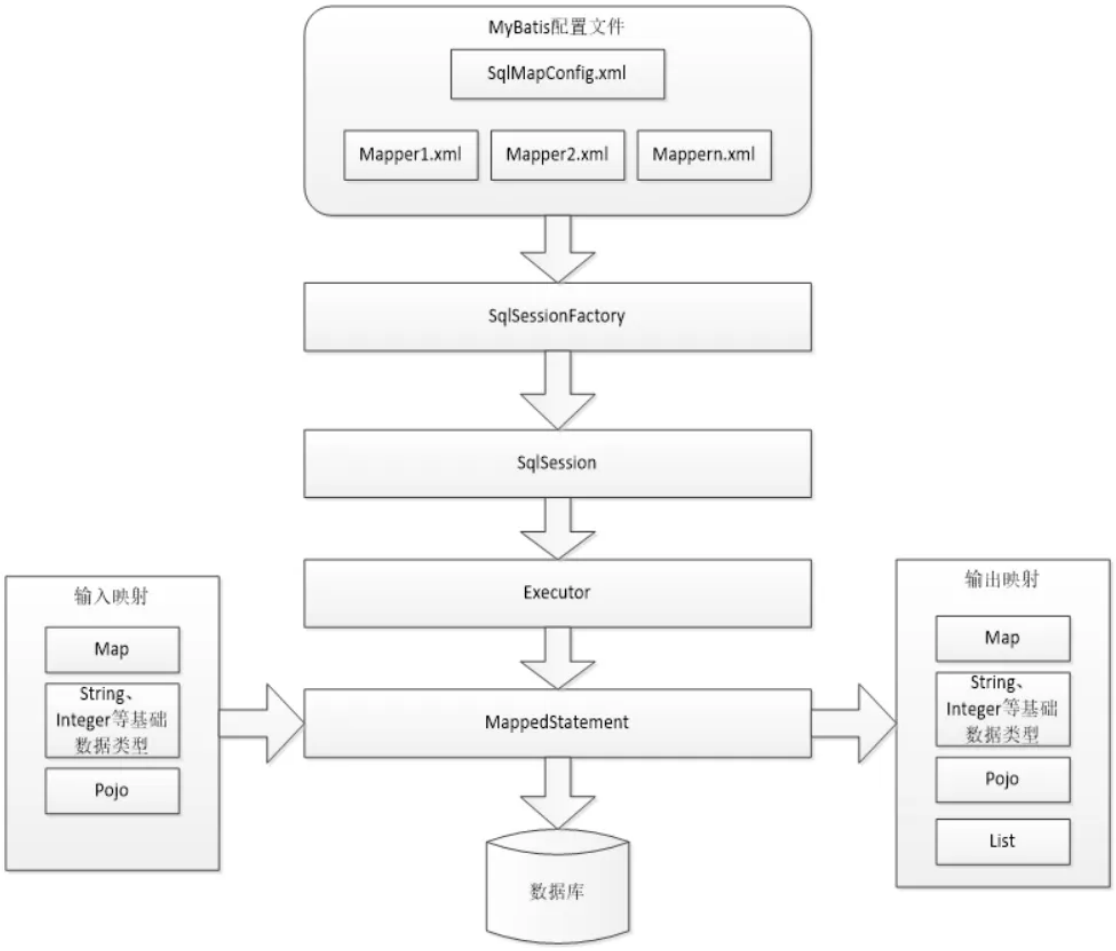
2　mybatis

3　hibernate

4　mybatis-plus　*mybatis通用mapper*　springdatajpa

orm框架　对象关系映射　javabean->数据库表映射起来

1.3 mybatis架构



1、 mybatis配置

SqlMapConfig.xml，此文件作为mybatis的全局配置文件，配置了mybatis的运行环境等信息。

mapper.xml文件即sql映射文件，文件中配置了操作数据库的sql语句。此文件需要在SqlMapConfig.xml中加载。

2、 通过mybatis环境等配置信息构造SqlSessionFactory即会话工厂

3、 由会话工厂创建sqlSession即会话，操作数据库需要通过sqlSession进行。

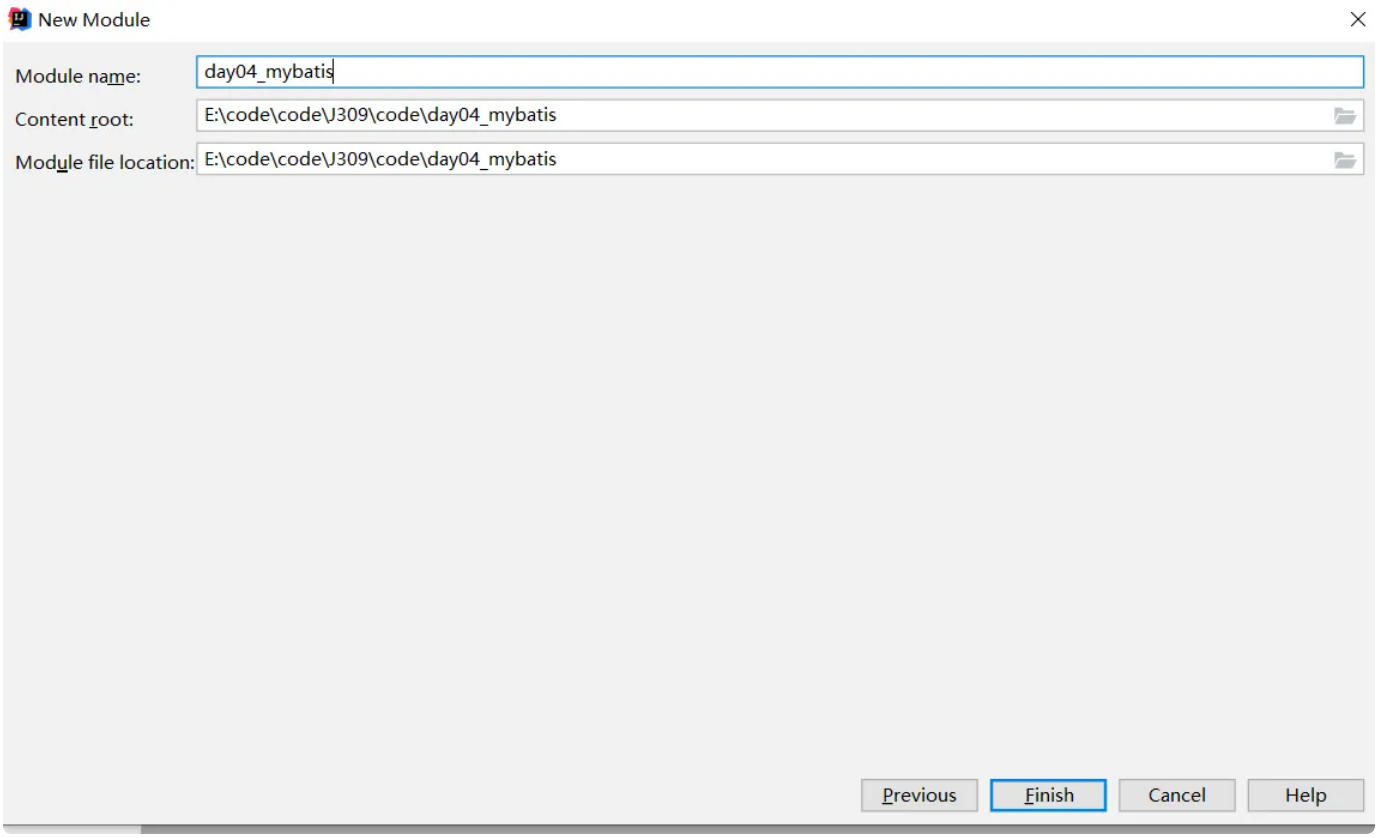
4、 mybatis底层自定义了Executor执行器接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。

5、 Mapped Statement也是mybatis一个底层封装对象，它包装了mybatis配置信息及sql映射信息等。mapper.xml文件中一个sql对应一个Mapped Statement对象，sql的id即是Mapped statement的id。

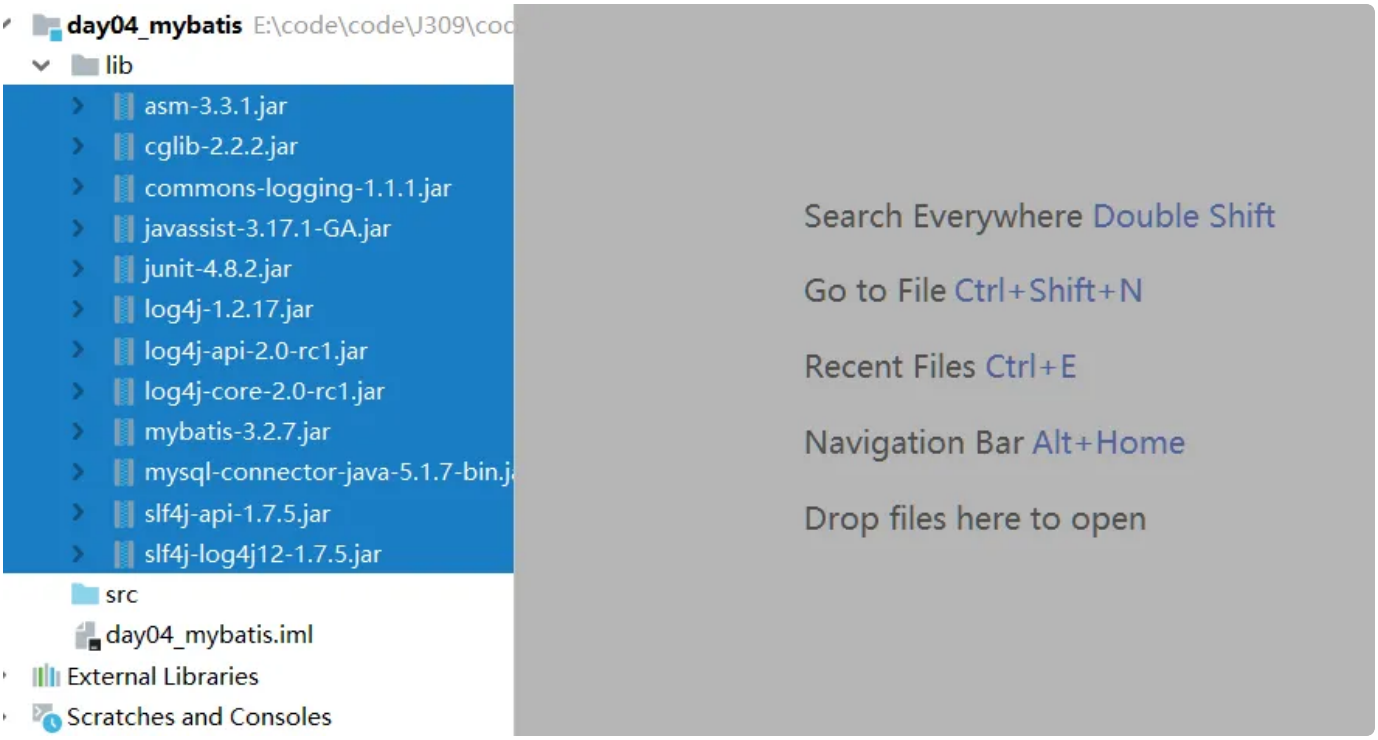
6、 Mapped Statement对sql执行输入参数进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql前将输入的java对象映射至sql中，输入参数映射就是jdbc编程中对preparedStatement设置参数。

7、 Mapped Statement对sql执行输出结果进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql后将输出结果映射至java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程。

1.4 环境搭建



导包


day04_mybatis E:\code\code\J309\coc
lib
asm-3.3.1.jar
cglib-2.2.2.jar
commons-logging-1.1.1.jar
javassist-3.17.1-GA.jar
junit-4.8.2.jar
log4j-1.2.17.jar
log4j-api-2.0-rc1.jar
log4j-core-2.0-rc1.jar
mybatis-3.2.7.jar
mysql-connector-java-5.1.7-bin.j
slf4j-api-1.7.5.jar
slf4j-log4j12-1.7.5.jar
src
day04_mybatis.iml
External Libraries
Scratches and Consoles
Search Everywhere Double Shift
Go to File Ctrl+Shift+N
Recent Files Ctrl+E
Navigation Bar Alt+Home
Drop files here to open

导入sql

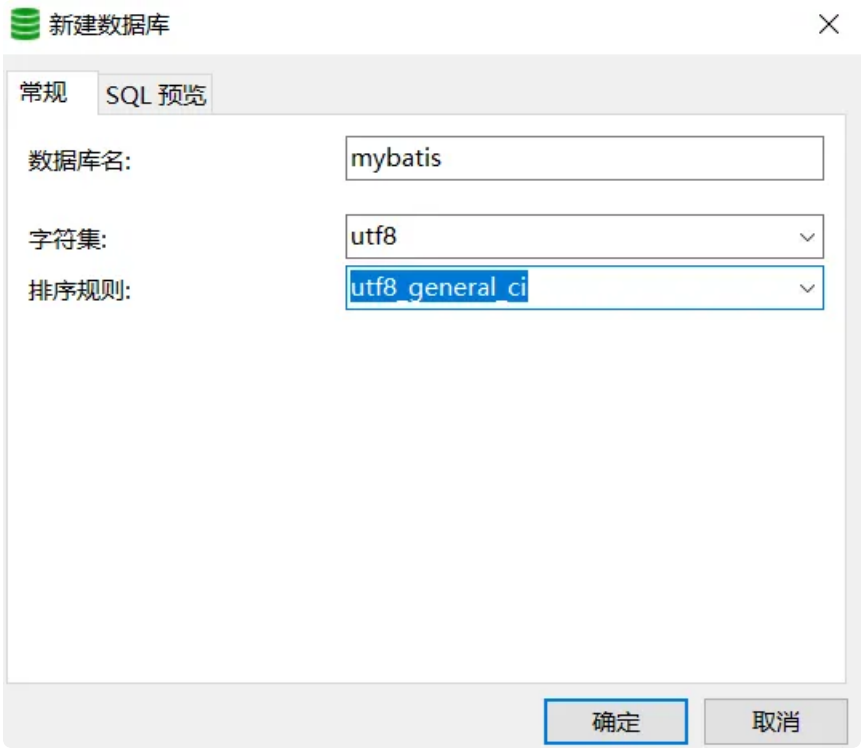

新建数据库
常规
SQL 预览
数据库名:
mybatis
字符集:
utf8
排序规则:
utf8_general_ci
确定
取消


表
gxa_category
gxa_goods
gxa_goods_desc
gxa_order
gxa_order_item
gxa_user

准备pojo

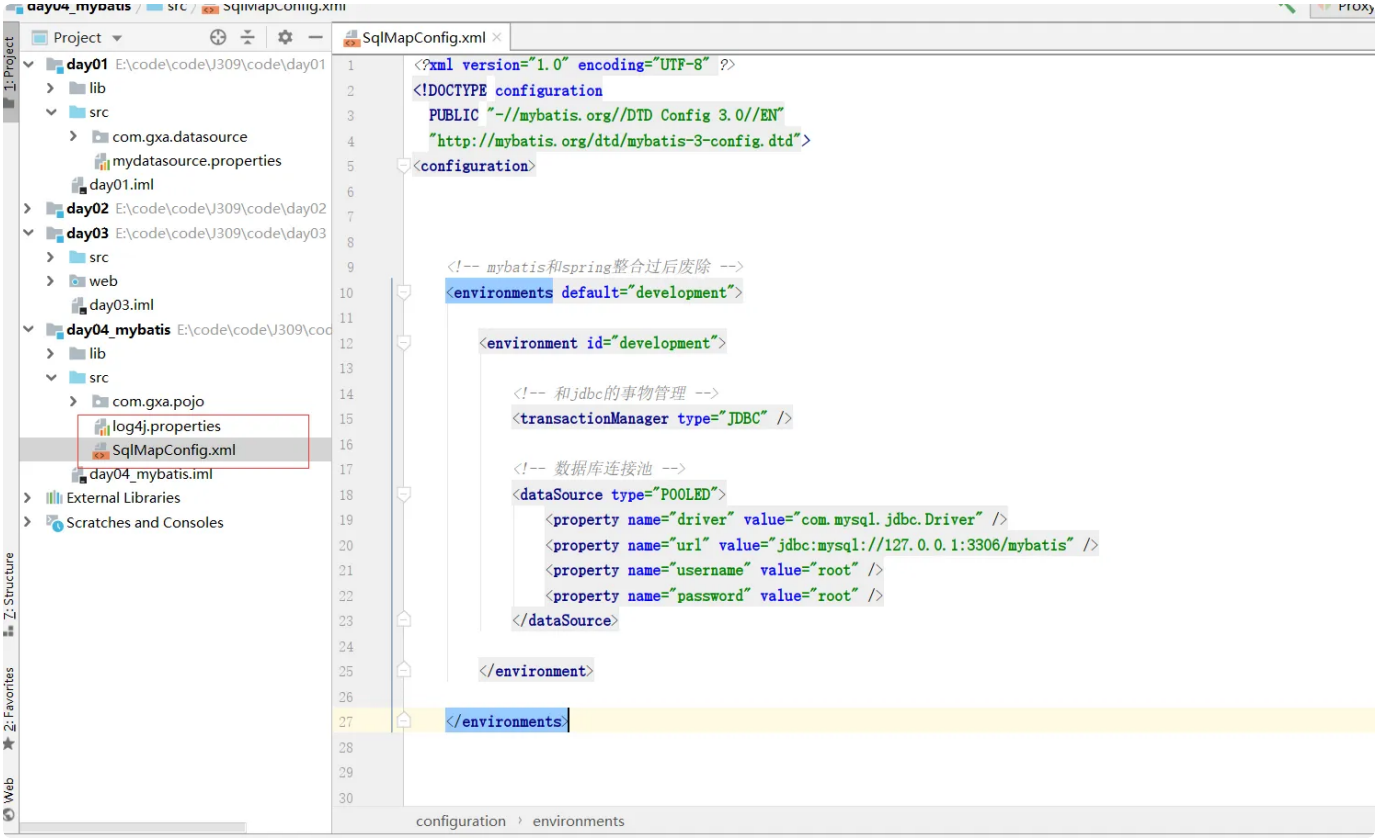
```java
/**
 * Created by zxd on 2022/10/18 9:37
 */
public class User implements Serializable {

    private Long id;

    private String username;

    private String password;

    private String salt;

    private String phone;

    private Date created;

    private Date lastLoginTime;

    private Integer status;

    public User(Long id, String username, String password, String salt, String phone, Date created, Date lastLoginTime, Integer status) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.salt = salt;
        this.phone = phone;
        this.created = created;
        this.lastLoginTime = lastLoginTime;
        this.status = status;
    }

    public User() {
    }

    public Long getId() {
        return id;
    }

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

    public String getUsername() {
```

```
45          return username;
46      }
47
48      public void setUsername(String username) {
49          this.username = username;
50      }
51
52      public String getPassword() {
53          return password;
54      }
55
56      public void setPassword(String password) {
57          this.password = password;
58      }
59
60      public String getSalt() {
61          return salt;
62      }
63
64      public void setSalt(String salt) {
65          this.salt = salt;
66      }
67
68      public String getPhone() {
69          return phone;
70      }
71
72      public void setPhone(String phone) {
73          this.phone = phone;
74      }
75
76      public Date getCreated() {
77          return created;
78      }
79
80      public void setCreated(Date created) {
81          this.created = created;
82      }
83
84      public Date getLastLoginTime() {
85          return lastLoginTime;
86      }
87
88      public void setLastLoginTime(Date lastLoginTime) {
89          this.lastLoginTime = lastLoginTime;
90      }
91
92      public Integer getStatus() {
```

```
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }


    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", salt='" + salt + '\'' +
                ", phone='" + phone + '\'' +
                ", created=" + created +
                ", lastLoginTime=" + lastLoginTime +
                ", status=" + status +
                '}';
    }
}
```

1.5 实现根据id查询用户信息

1.5.1 准备主配置文件


day04_mybatis
Project
day01 E:\code\code\J309\code\day01
lib
src
com.gxa.datasource
mydatasource.properties
day01.iml
day02 E:\code\code\J309\code\day02
day03 E:\code\code\J309\code\day03
src
web
day03.iml
day04_mybatis E:\code\code\J309\cod
lib
src
com.gxa.pojo
log4j.properties
SqlMapConfig.xml
day04_mybatis.iml
External Libraries
Scratches and Consoles
SqlMapConfig.xml
<?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>
<!-- mybatis和spring整合过后废除 -->
<environments default="development">
<environment id="development">
<!-- 和jdbc的事物管理 -->
<transactionManager type="JDBC" />
<!-- 数据库连接池 -->
<dataSource type="POOLED">
<property name="driver" value="com.mysql.jdbc.Driver" />
<property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis" />
<property name="username" value="root" />
<property name="password" value="root" />
</dataSource>
</environment>
</environments>
configuration › environments
1: Project
7: Structure
2: Favorites
Web

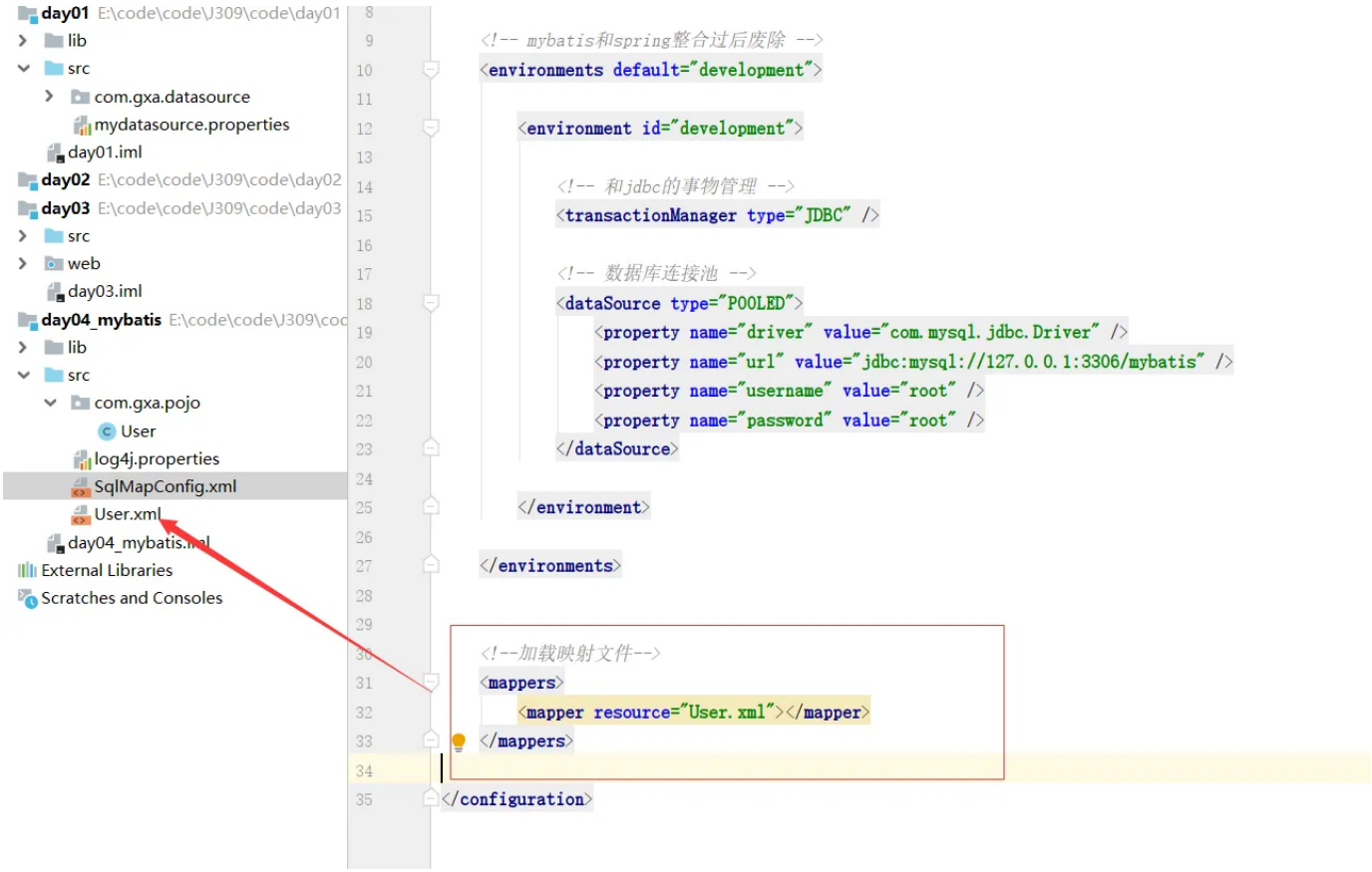
XML

```xml
<?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>



  <!-- mybatis和spring整合过后废除 -->
  <environments default="development">

    <environment id="development">

      <!-- 和jdbc的事物管理 -->
      <transactionManager type="JDBC" />

      <!-- 数据库连接池 -->
      <dataSource type="POOLED">
        <property name="driver" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://127.0.0.1:3306/mybatis" />
        <property name="username" value="root" />
        <property name="password" value="root" />
      </dataSource>

    </environment>

  </environments>




</configuration>
```

1.5.2 编写映射文件

```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">
<!-- 命名空间，用于隔离sql，后面还有一个很重要的作用 -->
<mapper namespace="test">


        <!--查询标签-->
        <!--id sql唯一的标记 statement的id或者sql的id,在一个命名空间中唯一-->
        <!--parameterType: 输入参数类型-->
        <select id="findUserById" parameterType="long" resultType="com.gxa.pojo.User">
            select * from gxa_user where id=#{id}
        </select>



</mapper>
```

1.5.3 主配置文件加载映射文件



1.5.4 编写Java代码

```java
/**
* Created by zxd on 2022/10/18 9:57
*/
public class TestUserDao {


    @Test
    public void helloMybatis() throws IOException {

        //1.加载主配置文件

        //SqlSessionFactoryBuilder创建 SqlSessionFactory
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFactoryBuilder();

        //2.加载主配置文件
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

        //3.解析xml的内容，创建sqlSessionFactory对象
        SqlSessionFactory sessionFactory = sqlSessionFactoryBuilder.build(in);

        //4.通过sqlSessionFactory获取SqlSession
        SqlSession sqlSession = sessionFactory.openSession();

        //5.执行sql返回结果
        User user = sqlSession.selectOne("test.findUserById", 3L);


        System.out.println(user);

    }

}

```

1.6 根据用户姓名进行模糊查询1

1.6.1 编写sql

```sql
select * from gxa_user where username like '%强%'
```

1.6.2 编写映射文件

```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">
<!-- 命名空间，用于隔离sql，后面还有一个很重要的作用 -->
<mapper namespace="test">



        <!--如果有多条记录，mybatis会把每一条记录的结果信息封装到pojo中，然后把pojo放入list集合-->
        <select id="findUserByUsername1" parameterType="string" resultType="com.gxa.pojo.User">
            select * from gxa_user where username like #{username}
        </select>



</mapper>
```

底层原理

```java
String sql="select * from gxa_user where username like ?";
st=conn.prepareStatement(sql);
st.setString(1,"%哥%")

```

1.6.3 测试代码

```java
@Test
public void findUserName() throws IOException {

    //1.加载主配置文件

    //SqlSessionFactoryBuilder创建 SqlSessionFactory
    SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFactoryBuilder();

    //2.加载主配置文件
    InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

    //3.解析xml的内容，创建sqlSessionFactory对象
    SqlSessionFactory sessionFactory = sqlSessionFactoryBuilder.build(in);

    //4.通过sqlSessionFactory获取SqlSession
    SqlSession sqlSession = sessionFactory.openSession();

    //5.执行sql返回结果
    List<User> users = sqlSession.selectList("test.findUserByUsername1","%哥%");


    System.out.println(users);

    }
```

1.7 根据用户名进行模糊查询2

1.7.1 编写映射文件

```xml
        <select id="findUserByUsername2" parameterType="string" resultType="com.gxa.pojo.User">
                select * from gxa_user where username like '%${value}%'
         </select>
```

1.7.2 测试

```java
  @Test
    public void findUserName2() throws IOException {

        //1.加载主配置文件

        //SqlSessionFactoryBuilder创建 SqlSessionFactory
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFactoryBuilder();

        //2.加载主配置文件
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

        //3.解析xml的内容，创建sqlSessionFactory对象
        SqlSessionFactory sessionFactory = sqlSessionFactoryBuilder.build(in);

        //4.通过sqlSessionFactory获取SqlSession
        SqlSession sqlSession = sessionFactory.openSession();

        //5.执行sql返回结果
        List<User> users = sqlSession.selectList("test.findUserByUsername2", "哥");


        System.out.println(users);

    }
```

1.7.3 原理

```java
String value="哥";

String sql="select * from gxa_user where username like '%"+value+"%'";

st=conn.createStatement();

st.executeQuery(sql);
```

面临sql注入问题

1.8 #{}和${}区别

#{}表示一个占位符,没有sql注入问题,通过#{}可以实现向preparedStatement占位符?中设置参数。#{}可以防止sql注入问题 #{}可以接收简单类型或者pojo的属性值。如果是传递的简单数据类型,#{}中的名字随便写，最好还是见名之意 底层就是往第一个?设置参数

${} 表示拼接$sql$, sql注入问题, ${}可以接收简单类型或者pojo的属性值。如果是传递的简单数据类型,${}中的名字只能写value 底层是拼接的sql

parameterType：指定输入参数的类型，mybatis通过ognl表达式获取参数的值设置到sql中

resultType: 指定输出结果的类型,mybatis将sql查询结果的一行记录映射为resultType指定的对象中。如果有多条记录，分别映射，并且把对象放入list集合中。

1.9 selectOne和selectList区别?

selectOne查询一条记录，如果使用selctOne查询的sql返回多条记录就会抛异常

selectList 查询0条或者多条都行

selectOne底层是selectList来执行，并且正常情况是获取第一个元素

Java

```java
public <T> T selectOne(String statement, Object parameter) {
        List<T> list = this.selectList(statement, parameter);
        if (list.size() == 1) {
            return list.get(0);
        } else if (list.size() > 1) {
            throw new TooManyResultsException("Expected one result (or null) to be returned by selectOne(), but found: " + list.size());
        } else {
            return null;
        }
    }
```

2.crud

2.1 插入

2.1.1 准备sql

```xml
insert into gxa_user(username,password,salt,phone,created,last_login_time,`status`) VALUES ()
```

2.1.2 编写xml

```xml
  <insert id="insertUser" parameterType="com.gxa.pojo.User">
                insert into gxa_user(username,password,salt,phone,created,last_login_time,`status`) VALUES (#{username},#{password},#{salt},#{phone},#{created},#{lastLoginTime},#{status})
        </insert>
```

#{}取的是pojo的属性值

2.1.3 测试

XML

```
@Test
    public void insertUser() throws IOException {

        //1.加载主配置文件

        //SqlSessionFactoryBuilder创建 SqlSessionFactory
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFa
ctoryBuilder();

        //2.加载主配置文件
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xm
l");

        //3.解析xml的内容，创建sqlSessionFactory对象
        SqlSessionFactory sessionFactory = sqlSessionFactoryBuilder.build
(in);

        //4.通过sqlSessionFactory获取SqlSession
        SqlSession sqlSession = sessionFactory.openSession();

        User user=new User(null,"坤坤","111","111","11",new Date(),new Dat
e(),1);

        int result = sqlSession.insert("test.insertUser", user);

        System.out.println(result);


        //提交事务
        sqlSession.commit();

    }
```

2.2 修改

2.2.1 编写sql

```xml
update gxa_user set username=#{username},password=#{password},phone=#{phone} where id=#{id}
```

2.2.2 编写xml

```xml
  <update id="updateUser" parameterType="com.gxa.pojo.User">
              update gxa_user set username=#{username},password=#{password},phone=#{phone} where id=#{id}
        </update>

```

2.2.3 测试

```XML
 @Test
    public void updateUser() throws IOException {

        //1.加载主配置文件

        //SqlSessionFactoryBuilder创建 SqlSessionFactory
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFactoryBuilder();

        //2.加载主配置文件
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

        //3.解析xml的内容，创建sqlSessionFactory对象
        SqlSessionFactory sessionFactory = sqlSessionFactoryBuilder.build(in);

        //4.通过sqlSessionFactory获取SqlSession
        SqlSession sqlSession = sessionFactory.openSession();

        User user=new User(5L,"龙龙","222","111","333",new Date(),new Date(),1);

        int result = sqlSession.update("test.updateUser", user);

        System.out.println(result);


        //提交事务
        sqlSession.commit();

    }
```

2.3 删除

2.3.1 编写sql

```SQL
delete from gxa_user where id=#{id}
```

2.3.2 编写xml

```Java
    <delete id="deleteUserById" parameterType="long">
            delete from gxa_user where id=#{id}
        </delete>
```

2.3.3 测试

```SQL
  @Test
    public void deleteUser() throws IOException {

        //1.加载主配置文件

        //SqlSessionFactoryBuilder创建 SqlSessionFactory
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFactoryBuilder();

        //2.加载主配置文件
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

        //3.解析xml的内容，创建sqlSessionFactory对象
        SqlSessionFactory sessionFactory = sqlSessionFactoryBuilder.build(in);

        //4.通过sqlSessionFactory获取SqlSession
        SqlSession sqlSession = sessionFactory.openSession();


        int result = sqlSession.delete("test.deleteUserById", 5L);

        System.out.println(result);


        //提交事务
        sqlSession.commit();

    }
```

2.4 mysql自增主键的返回(重点)

2.4.1 自增主键的原理

```java
insert into gxa_user(username,password,salt,phone,created,last_login_time,`status`) VALUES ('aa','xxx','xxx','XX','2022-03-01','2022-03-01',1);

select LAST_INSERT_ID()

```

2.4.2 编写xml

```xml
      <insert id="insertUser" parameterType="com.gxa.pojo.User">
                <!--selectKey: 实现主键的返回-->
                <!--keyColumn:主键对应数据库的哪一列-->
                <!--keyProperty:主键对应pojo的哪一属性-->
                <!--配置在插入语句之前执行还是之后执行-->
                <!--resultType:主键的类型-->
                <selectKey keyColumn="id" keyProperty="id" order="AFTER" resultType="long">
                    select LAST_INSERT_ID()
                </selectKey>
                insert into gxa_user(username,password,salt,phone,created,last_login_time,`status`) VALUES
                 (#{username},#{password},#{salt},#{phone},#{created},#{lastLoginTime},#{status})
        </insert>
```

2.4.3 测试

```java
    @Test
     public void insertUser() throws IOException {

         //1.加载主配置文件

         //SqlSessionFactoryBuilder创建 SqlSessionFactory
         SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFactoryBuilder();

         //2.加载主配置文件
         InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

         //3.解析xml的内容，创建sqlSessionFactory对象
         SqlSessionFactory sessionFactory = sqlSessionFactoryBuilder.build(in);

         //4.通过sqlSessionFactory获取SqlSession
         SqlSession sqlSession = sessionFactory.openSession();

         User user=new User(null,"坤坤","111","111","11",new Date(),new Date(),1);

         int result = sqlSession.insert("test.insertUser", user);

         System.out.println(user);


         //提交事务
         sqlSession.commit();

     }

```

2.5 mysql自增主键的简化写法

```xml
    <insert id="insertUser" useGeneratedKeys="true" keyProperty="id"  parameterType="com.gxa.pojo.User">
        insert into gxa_user(username,password,salt,phone,created,last_login_time,`status`) VALUES
        (#{username},#{password},#{salt},#{phone},#{created},#{lastLoginTime},#{status})
    </insert>

```

2.6 使用mysql的uuid实现主键(了解)

复制表

```sql
CREATE TABLE `gxa_user2`  (
  `id` varchar(255) NOT NULL ,
  `username` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `password` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `salt` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `phone` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT NULL,
  `created` datetime NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
  `last_login_time` datetime NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
  `status` int(11) NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE = InnoDB AUTO_INCREMENT = 10 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Compact;
```

新建pojo

Java

```java
/**
* Created by zxd on 2022/10/18 9:37
*/
public class User2 implements Serializable {

    private String id;

    private String username;

    private String password;

    private String salt;

    private String phone;

    private Date created;

    private Date lastLoginTime;

    private Integer status;

    public User2(String id, String username, String password, String salt, String phone, Date created, Date lastLoginTime, Integer status) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.salt = salt;
        this.phone = phone;
        this.created = created;
        this.lastLoginTime = lastLoginTime;
        this.status = status;
    }

    public User2() {
    }

    public String getId() {
        return id;
    }

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

    public String getUsername() {
```

```
45          return username;
46      }
47
48      public void setUsername(String username) {
49          this.username = username;
50      }
51
52      public String getPassword() {
53          return password;
54      }
55
56      public void setPassword(String password) {
57          this.password = password;
58      }
59
60      public String getSalt() {
61          return salt;
62      }
63
64      public void setSalt(String salt) {
65          this.salt = salt;
66      }
67
68      public String getPhone() {
69          return phone;
70      }
71
72      public void setPhone(String phone) {
73          this.phone = phone;
74      }
75
76      public Date getCreated() {
77          return created;
78      }
79
80      public void setCreated(Date created) {
81          this.created = created;
82      }
83
84      public Date getLastLoginTime() {
85          return lastLoginTime;
86      }
87
88      public void setLastLoginTime(Date lastLoginTime) {
89          this.lastLoginTime = lastLoginTime;
90      }
91
92      public Integer getStatus() {
```

```
93          return status;
94      }
95
96
97      public void setStatus(Integer status) {
98          this.status = status;
99      }
100
101
102     @Override
103     public String toString() {
104         return "User{" +
105             "id=" + id +
106             ", username='" + username + '\'' +
107             ", password='" + password + '\'' +
108             ", salt='" + salt + '\'' +
109             ", phone='" + phone + '\'' +
110             ", created=" + created +
111             ", lastLoginTime=" + lastLoginTime +
112             ", status=" + status +
113             '}';
114     }
115 }
```

编写xml

```java
 <insert id="insertUser2"  parameterType="com.gxa.pojo.User2">

                        <!--selectKey: 实现主键的返回-->
                        <!--keyColumn:主键对应数据库的哪一列-->
                        <!--keyProperty:主键对应pojo的哪一属性-->
                        <!--配置在插入语句之前执行还是之后执行-->
                        <!--resultType:主键的类型-->
                        <selectKey keyColumn="id" keyProperty="id" order="BEFORE" resultType="string">
                            select uuid()
                        </selectKey>

        insert into gxa_user2(id,username,password,salt,phone,created,last_login_time,`status`) VALUES
        (#{id},#{username},#{password},#{salt},#{phone},#{created},#{lastLoginTime},#{status})
    </insert>

```

测试代码

```java
    @Test
    public void insertUser2() throws IOException {

        //1.加载主配置文件

        //SqlSessionFactoryBuilder创建 SqlSessionFactory
        SqlSessionFactoryBuilder sqlSessionFactoryBuilder=new SqlSessionFactoryBuilder();

        //2.加载主配置文件
        InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

        //3.解析xml的内容，创建sqlSessionFactory对象
        SqlSessionFactory sessionFactory = sqlSessionFactoryBuilder.build(in);

        //4.通过sqlSessionFactory获取SqlSession
        SqlSession sqlSession = sessionFactory.openSession();

        User2 user=new User2(null,"坤坤","111","111","11",new Date(),new Date(),1);

        int result = sqlSession.insert("test.insertUser2", user);

        System.out.println(user);


        //提交事务
        sqlSession.commit();

    }
```

3.原始dao开发

3.1 编写工具类

```java
/**
 * Created by zxd on 2022/10/18 15:16
 */
public class MybatisUtils {

    private  static SqlSessionFactory sqlSessionFactory;


    static {

        try {

            //1.加载主配置文件
            SqlSessionFactoryBuilder sqlSessionFactoryBuilder = new SqlSessionFactoryBuilder();

            //2.加载主配置文件
            InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");

            //3.解析xml的内容，创建sqlSessionFactory对象
            sqlSessionFactory = sqlSessionFactoryBuilder.build(in);

        }catch (Exception e){
            e.printStackTrace();
            throw new RuntimeException("配置文件加载失败!");
        }

    }


    /**
     * 获取sqlSession
     * @return
     */
    public static SqlSession getSqlSession(){
        return sqlSessionFactory.openSession();
    }

}

```

3.2 编写接口

```java
/**
 * Created by zxd on 2022/10/18 15:19
 */
public interface UserDao {


    User findUserById(Long id);

    List<User> findUserByUsername(String username);

    void insertUser(User user);

    void updateUser(User user);

    void deleteUserById(Long id);

}
```

3.3 编写dao实现

```java
/**
 * Created by zxd on 2022/10/18 15:22
 */
public class UserDaoImpl implements UserDao {

    @Override
    public User findUserById(Long id) {
        SqlSession sqlSession = MybatisUtils.getSqlSession();
        return sqlSession.selectOne("test.findUserById",id);
    }

    @Override
    public List<User> findUserByUsername(String username) {
        SqlSession sqlSession = MybatisUtils.getSqlSession();
        return sqlSession.selectList("test.findUserByUsername",username);
    }

    @Override
    public void insertUser(User user) {
        SqlSession sqlSession = MybatisUtils.getSqlSession();
        sqlSession.insert("test.insertUser",user);
        sqlSession.commit();
    }

       @Override
    public void updateUser(User user) {
        SqlSession sqlSession = MybatisUtils.getSqlSession();
        sqlSession.update("test.updateUser",user);
        sqlSession.commit();
    }

      @Override
    public void deleteUserById(Long id) {
        SqlSession sqlSession = MybatisUtils.getSqlSession();
        sqlSession.delete("test.deleteUserById",id);
        sqlSession.commit();
    }
}

```

3.4 编写xml

XML

```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">
<!-- 命名空间，用于隔离sql，后面还有一个很重要的作用 -->
<mapper namespace="test">


        <!--查询标签-->
        <!--id sql唯一的标记 statement的id或者sql的id,在一个命名空间中唯一-->
        <!--parameterType: 输入参数类型-->
        <select id="findUserById" parameterType="long" resultType="com.gxa.pojo.User">
            select * from gxa_user where id=#{id}
        </select>



        <select id="findUserByUsername" parameterType="string" resultType="com.gxa.pojo.User">
              select * from gxa_user where username like '%${value}%'
        </select>


    <insert id="insertUser" useGeneratedKeys="true" keyProperty="id"  parameterType="com.gxa.pojo.User">
        insert into gxa_user(username,password,salt,phone,created,last_login_time,`status`) VALUES
        (#{username},#{password},#{salt},#{phone},#{created},#{lastLoginTime},#{status})
    </insert>


       <update id="updateUser" parameterType="com.gxa.pojo.User">
              update gxa_user set username=#{username},password=#{password},phone=#{phone} where id=#{id}
       </update>

       <delete id="deleteUserById" parameterType="long">
           delete from gxa_user where id=#{id}
       </delete>


</mapper>

```

```
40
41
42
43
```

3.5 测试

Java

```java
/**
 * Created by zxd on 2022/10/18 15:25
 */
public class TestUserDao2 {

    private UserDao userDao=new UserDaoImpl();

    @Test
    public void testFind(){
        User user = userDao.findUserById(3L);
        System.out.println(user);
    }

    @Test
    public void testFindUsername(){

        List<User> userList = userDao.findUserByUsername("强");
        System.out.println(userList);

    }

    @Test
    public void insert(){

        User user=new User(null,"龙龙","111","111","11",new Date(),new Date(),1);
        userDao.insertUser(user);
    }

    @Test
    public void update(){

        User user=new User(10L,"龙龙2","22","111","22",new Date(),new Date(),1);
        userDao.updateUser(user);
    }


    @Test
    public void delete(){
        userDao.deleteUserById(11L);
    }



```

```
44  }
45
```

3.6 思考

dao实现类存在大量的重复代码

1.接口的名字和sql的id一致

2.接口的参数类型和paramterType类型一致

3.接口返回值类型和resultType的类型一致

mybatis可不可以帮我们去生成实现类的代码!!! 可以

4.Mapper动态代理开发

dao存在大量重复代码，mybatis底层使用动态代理帮我们自动生成实现类代码

4.1 mapper动态代理开发规范

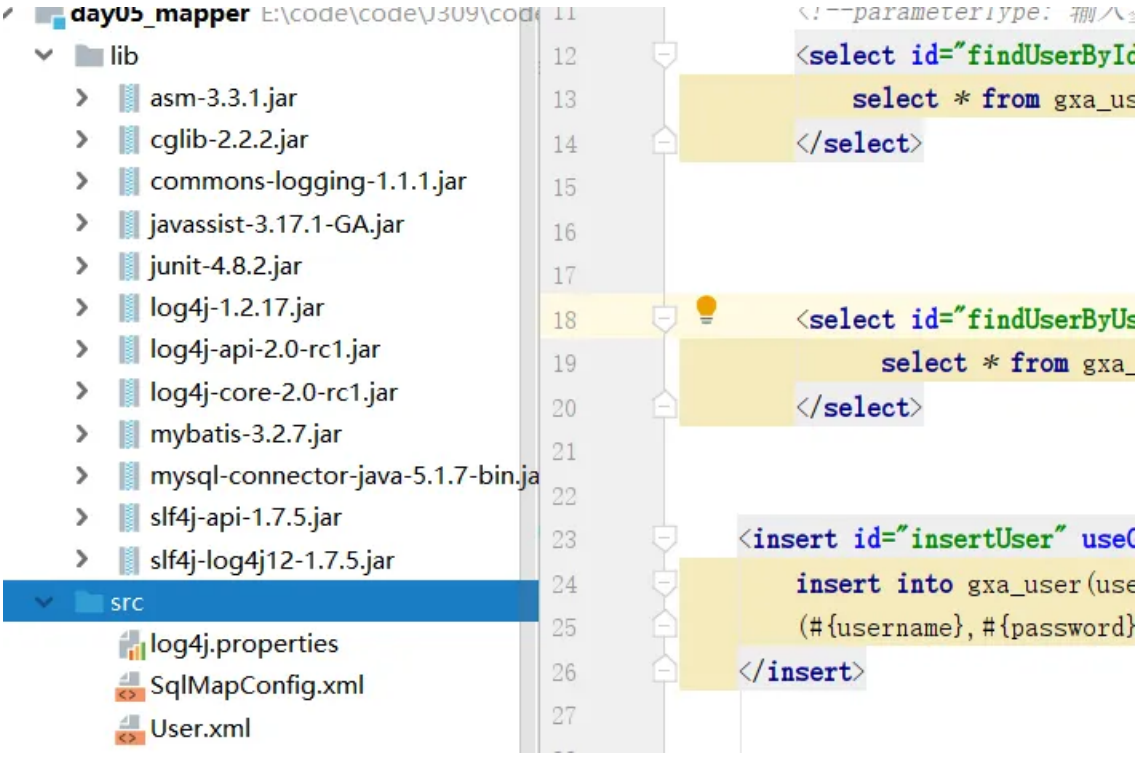
1.Mapper.xml文件的namespace和mapper接口的全限定名一致

2.Mapper接口的方法名和mapper.xml中定义的每个sql的id要一致

3.mapper接口方法的参数类型要和mapper.xml中的parameterType的类型要一致

4.mapper接口的输出参数类型要和mapper.xml中定义的每个sql的resultTyp的类型要一致

4.2 项目准备



4.3 编写mapper接口

```java
/**
* Created by zxd on 2022/10/18 15:19
*/
public interface UserMapper {


    User findUserById(Long id);

    List<User> findUserByUsername(String username);

    void insertUser(User user);

    void updateUser(User user);

    void deleteUserById(Long id);

}

```

4.4 编写mapper.xml

XML

```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">
<!-- 命名空间，用于隔离sql，后面还有一个很重要的作用 -->
<mapper namespace="com.gxa.mapper.UserMapper">


        <!--查询标签-->
        <!--id sql唯一的标记 statement的id或者sql的id,在一个命名空间中唯一-->
        <!--parameterType: 输入参数类型-->
        <select id="findUserById" parameterType="long" resultType="com.gxa.pojo.User">
            select * from gxa_user where id=#{id}
        </select>



        <select id="findUserByUsername" parameterType="string" resultType="com.gxa.pojo.User">
              select * from gxa_user where username like '%${value}%'
        </select>


    <insert id="insertUser" useGeneratedKeys="true" keyProperty="id"  parameterType="com.gxa.pojo.User">
        insert into gxa_user(username,password,salt,phone,created,last_login_time,`status`) VALUES
        (#{username},#{password},#{salt},#{phone},#{created},#{lastLoginTime},#{status})
    </insert>


       <update id="updateUser" parameterType="com.gxa.pojo.User">
              update gxa_user set username=#{username},password=#{password},phone=#{phone} where id=#{id}
       </update>

       <delete id="deleteUserById" parameterType="long">
           delete from gxa_user where id=#{id}
       </delete>


</mapper>

```

```
40
41
42
43
```

4.5 测试

Java

```java
/**
* Created by zxd on 2022/10/18 16:49
*/
public class TestUserMapper {



    @Test
    public void testFind(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);


        User user = userMapper.findUserById(3L);

        System.out.println(user);

    }

}

```

5.主动配置文件

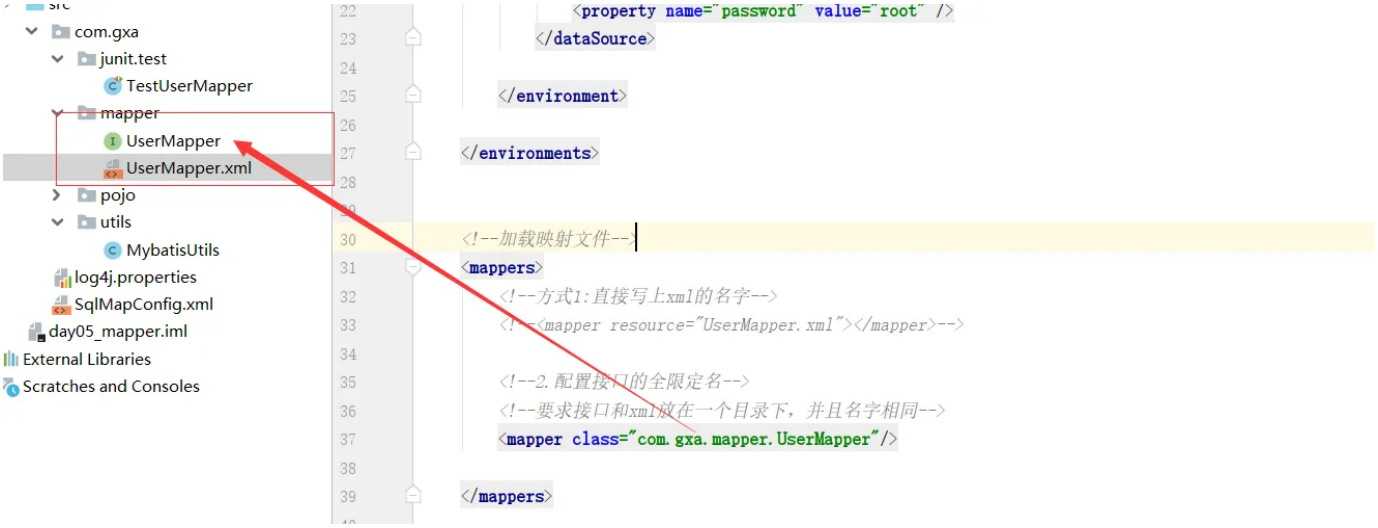
5.1 加载映射文件

5.1.1 直接写上xml名字

```xml
<!--加载映射文件-->
<mappers>
  <!--方式1:直接写上xml的名字-->
  <mapper resource="User.xml"></mapper>
</mappers>

```

5.1.2 直接写上接口的全限定名

```xml
<!--加载映射文件-->
<mappers>
  <!--方式1:直接写上xml的名字-->
      <!--<mapper resource="UserMapper.xml"></mapper>-->

  <!--2.配置接口的全限定名-->
  <!--要求接口和xml放在一个目录下，并且名字相同-->
  <mapper class="com.gxa.mapper.UserMapper"/>

</mappers>
```



5.1.3 批量加载

XML

```xml
    <!--加载映射文件-->
    <mappers>
      <!--方式1:直接写上xml的名字-->
          <!--<mapper resource="UserMapper.xml"></mapper>-->

      <!--2.配置接口的全限定名-->
      <!--要求接口和xml放在一个目录下，并且名字相同-->
<!--    <mapper class="com.gxa.mapper.UserMapper"/>-->

      <!--3.批量加载mapper接口-->
      <!--要求接口和xml放在一个目录下，并且名字相同-->
       <package name="com.gxa.mapper"/>

    </mappers>
```

5.2 mybatis别名

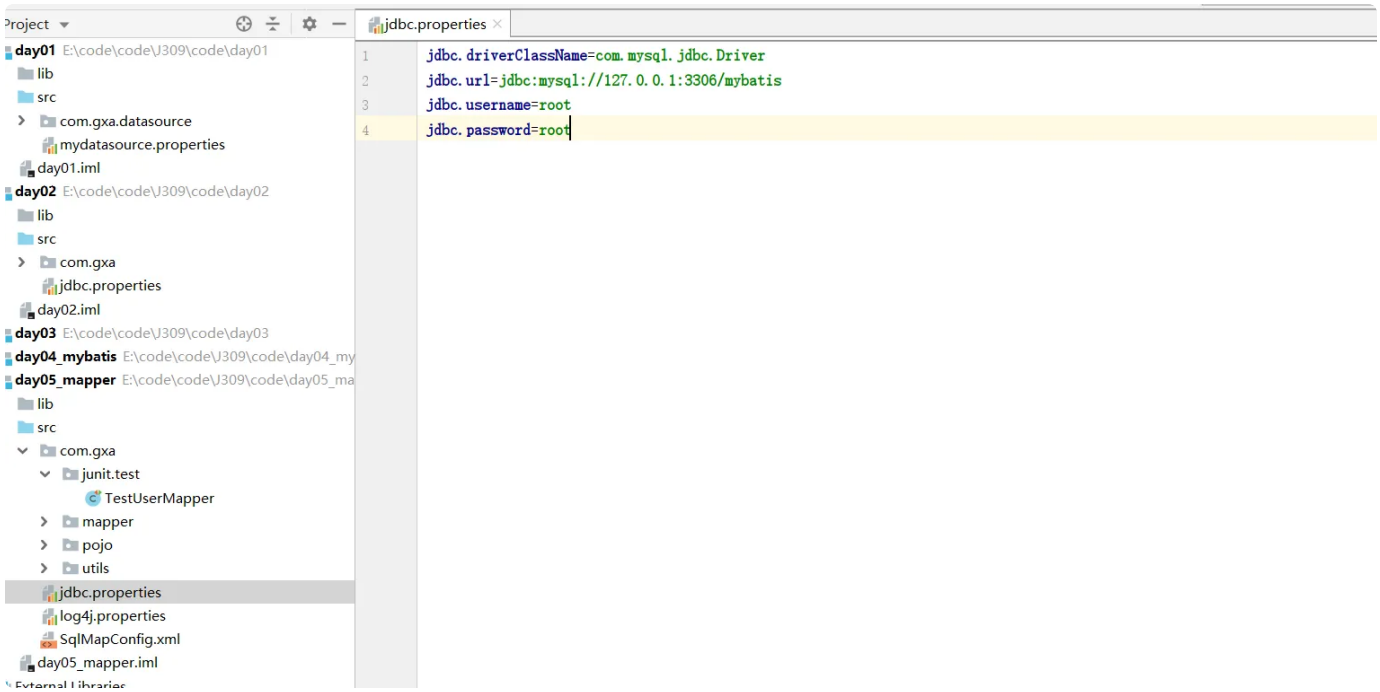
5.2.1 mybatis默认配置的别名

别名	映射的类型
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer
double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
map	Map

5.2.2 自定义别名

```xml
    <!--自定义别名-->
    <typeAliases>

        <!--配置单个别名 忽略大小写-->
    <!--    <typeAlias type="com.gxa.pojo.User"  alias="user"/>-->

        <!--批量定义别名-->
        <package name="com.gxa.pojo"/>

    </typeAliases>
```

5.3 加载外部的properties文件



XML

```xml
  <!--加载外部的properties文件-->
  <properties resource="jdbc.properties">
    <!--如果外部也有该属性外部的覆盖内部的-->
        <property name="jdbc.username" value="xxx"/>
  </properties>


  <!--自定义别名-->
  <typeAliases>

    <!--配置单个别名 忽略大小写-->
<!--    <typeAlias type="com.gxa.pojo.User"  alias="user"/>-->

    <!--批量定义别名-->
    <package name="com.gxa.pojo"/>

  </typeAliases>



  <!-- mybatis和spring整合过后废除 -->
  <environments default="development">

    <environment id="development">

      <!-- 和jdbc的事物管理 -->
      <transactionManager type="JDBC" />

      <!-- 数据库连接池 -->
      <dataSource type="POOLED">
        <property name="driver" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
      </dataSource>

    </environment>

  </environments>

```

6.输入映射

6.1 输入映射的介绍

parameterType

1.传递简单类型

int long string ...

使用#{}或者${}都能获取，#{}名字可以任意,${}名字只能写成value

2.传递pojo类型 ***

#{}或者${}获取pojo的属性名

3.传递包装的pojo类型 **

4.传递map集合(不推荐)

5.通过@Param注解来传递参数 **

6.2 传递简单类型

传递简单类型

int long string ...

使用#{}或者${}都能获取，#{}名字可以任意,${}名字只能写成value

接口

```java
User findUserById(Long id);

```

xml

```java
<select id="findUserById" parameterType="long" resultType="user">
          select * from gxa_user where id=#{id}
</select>
```

6.3 传递pojo类型

#{}或者${}获取pojo的属性名

parameterType写pojo的全限定名，如果配置了别名也可以写别名

接口

```java
void insertUser(User user);
```

xml

```Java
    <insert id="insertUser" useGeneratedKeys="true" keyProperty="id"  parameterType="com.gxa.pojo.User">
        insert into gxa_user(username,password,salt,phone,created,last_login_time,`status`) VALUES
        (#{username},#{password},#{salt},#{phone},#{created},#{lastLoginTime},#{status})
    </insert>
```

6.4 传递包装的pojo类型

DTO 数据传输对象

6.4.1 编写包装的pojo

```Java
/**
 * Created by zxd on 2022/10/19 10:35
 */
public class UserDto {

    private User user;

    //其它条件

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}

```

6.4.2 编写接口

```java
List<User>  findUserByUserDto(UserDto userDto);
```

6.4.3 编写xml

```java
    <select id="findUserByUserDto" parameterType="userDto" resultType="com.gxa.pojo.User">
                select * from gxa_user where username like #{user.username}
      </select>

```

6.4.4 测试

```java
    @Test
     public void testFindUserDto(){

         SqlSession sqlSession = MybatisUtils.getSqlSession();

         UserMapper userMapper = sqlSession.getMapper(UserMapper.class);


         UserDto userDto=new UserDto();
         User user=new User();
         user.setUsername("%哥%");
         userDto.setUser(user);

         List<User> userList = userMapper.findUserByUserDto(userDto);

         System.out.println(userList);

     }
```

6.5 传递map集合

6.5.1 编写接口

```java
   void insertUser2(Map map);
```

6.5.2 编写xml

```Java
    <insert id="insertUser2" useGeneratedKeys="true" keyProperty="id"  parameterType="map">
        insert into gxa_user(username,password,phone) VALUES (#{username},#{password},#{phone})
    </insert>
```

6.5.3 测试

```Java
    @Test
    public void testInsert(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

        Map map=new HashMap<>();
        map.put("username","banban");
        map.put("password","xxx");
        map.put("phone","xxx");

        userMapper.insertUser2(map);

        sqlSession.commit();

    }
```

不方便进行维护，不推荐

6.6 通过@param注解来标记参数 **

需求: 根据用户名和密码查询一个用户的信息

编写接口

```java
    //用@Param注解标记的参数，此时mybatis底层会把该参数放入map集合,用注解配置的名字作为key，传递的值作为value
    User findUserByUsernameAndPassword(@Param("username") String username,@Param("password")  String password);

```

编写xml

```java
   <select id="findUserByUsernameAndPassword" parameterType="map" resultType="user">
            select * from gxa_user where username =#{username} and password=#{password}
    </select>
```

测试

```java
   @Test
     public void testFindUsername(){

         SqlSession sqlSession = MybatisUtils.getSqlSession();

         UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

         User user = userMapper.findUserByUsernameAndPassword("强哥", "RbWEEsXVVxiR765qwusODQ==");

         System.out.println(user);


     }
```

7.输出映射

7.1 输出简单类型

7.1.1 编写接口

```Java
Long findUserCount();
```

7.1.2 编写xml

```Java
    <select id="findUserCount" resultType="long">
          select count(*) from gxa_user
    </select>

```

7.1.3 测试

```java
    @Test
    public void testCount(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

        Long count = userMapper.findUserCount();

        System.out.println(count);

    }
```

7.2 输出pojo类型

resultType配置的是pojo的全限定名，如果配置了别名也可以使用别名。如果查询的记录有多条，会把多条记录分别映射到pojo当中，然后把pojo放入list集合中。

```java
    <select id="findUserByUsername" parameterType="string" resultType="com.gxa.pojo.User">
            select * from gxa_user where username like '%${value}%'
        </select>
```

7.3 输出map集合(不推荐)

一个map对象相当于一个pojo对象，一个map集合封装了一条结果。如果有条结果把map放入list集合中

编写接口

```java
Map findUserById2(Long id);
```

编写xml

```java
<select id="findUserById2" parameterType="long" resultType="map">
            select * from gxa_user where id=#{id}
        </select>
```

测试

```java
    @Test
    public void testFind2(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);


        Map map = userMapper.findUserById2(3L);

        Object username = map.get("username");

        System.out.println(username);

        System.out.println(map);

    }
```

7.4 resultMap

7.4.1 订单表结构

订单列表:

订单详情:

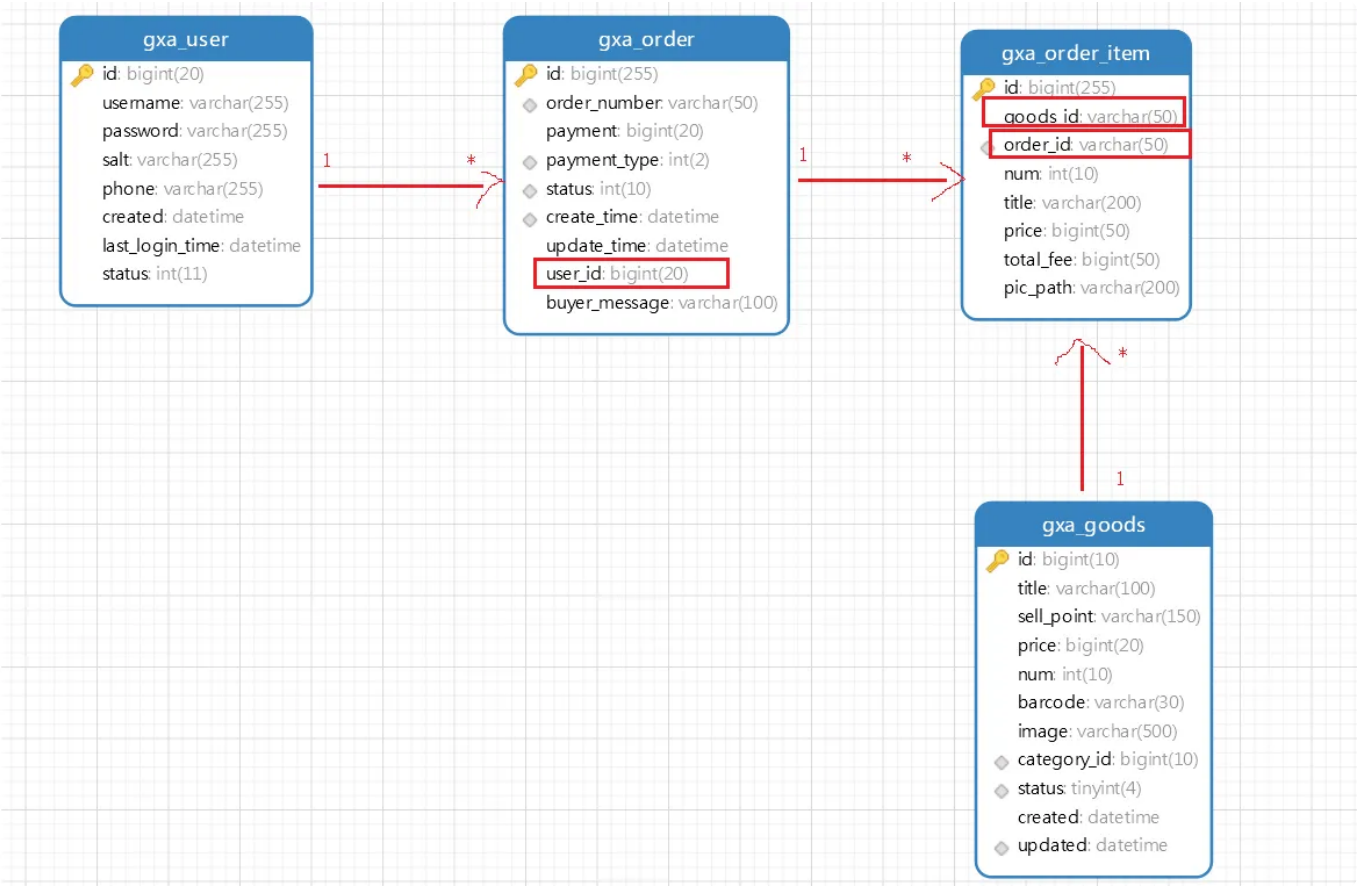
如何存储下单的商品信息?

订单主表　订单编号，总金额，状态

必须要去记住下单的商品

订单项表

每个订单项记住下单的每个商品信息



7.4.2 查询订单列表

7.4.2.1 编写mapper接口

```java
/**
 * Created by zxd on 2022/10/19 14:14
 */
public interface OrderMapper {

    /**
     * 查询所有的订单
     * @return
     */
    List<Order> findOrderAll();

}
```

7.4.2.2 编写xml

```java
<?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">
<!-- 命名空间，用于隔离sql，后面还有一个很重要的作用 -->
<mapper namespace="com.gxa.mapper.OrderMapper">


        <select id="findOrderAll" resultType="order">
            select * from gxa_order
        </select>




</mapper>
```

7.4.2.3 测试

```java
@Test
public void testFind(){

    SqlSession sqlSession = MybatisUtils.getSqlSession();

    OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);


    List<Order> orderList = orderMapper.findOrderAll();

    System.out.println(orderList);

}
```

```
14:20:03,185 DEBUG findOrderAll:139 - ==> Parameters:
14:20:03,205 DEBUG findOrderAll:139 - <==      Total: 3
[Order{id=1, orderNumber='null', payment=1, paymentType=null, status=1, createTime=null, updateTime=null, userId=null, buyerMessage='null'}, Order{id=2, orderI

Process finished with exit code 0
```

id	order_number	payment	payment_type	status	create_
1	xxx001	1	1	1	2020-1
2	xxx002	1	1	1	2020-1
3	xxx003	1	1	1	2020-1

```
14:20:03,205 DEBUG findOrderAll:139 - <==      Total: 3
[Order{id=1, orderNumber='null', payment=1, paymentType=null, status=1, createTime=null, updateTime=null, u

Process finished with exit code 0
```

mybatis在查询到数据后，把数据库中的每一行数据要通过resultType指定的pojo进行封装，拿着表中的字段名去pojo中找到对应的属性进行设置，如果不一致不管

7.4.3 取别名(方式一)

```java
        <select id="findOrderAll" resultType="order">
            select id,order_number as orderNumber,payment,payment_type as paymentType,status,create_time
            as createTime,update_time as updateTime,user_id as userId,buyer_message as buyerMessage from gxa_order
        </select>

```

7.4.4 resultMap(方式二)

resultMap手动处理字段和属性的关系

```Java
    <!--定义自定义的结果集映射-->
    <resultMap id="orderResultMap" type="com.gxa.pojo.Order">

        <!--映射主键 column主键在数据库的列名  property主键在pojo的属性名-->
        <!--jdbcType jdbc写法字段类型 javaType java写法字段的类型 不写自动识别-->
        <id column="id" property="id"/>

        <!--映射普通属性-->
        <result column="order_number" property="orderNumber" />
        <result column="payment" property="payment" />
        <result column="payment_type" property="paymentType" />
        <result column="status" property="status" />
        <result column="create_time" property="createTime" />
        <result column="update_time" property="updateTime" />
        <result column="user_id" property="userId" />
        <result column="buyer_message" property="buyerMessage" />

    </resultMap>


    <select id="findOrderAll" resultMap="orderResultMap">
            select * from gxa_order
    </select>

```

7.4.5 配置自动转换（下划线自动转驼峰）方式三

```java
<!--加载外部的properties文件-->
    <properties resource="jdbc.properties">
        <!--如果外部也有该属性外部的覆盖内部的-->
        <property name="jdbc.username" value="xxx"/>
    </properties>


    <!--配置下划线自动转换驼峰-->
    <settings>
        <setting name="mapUnderscoreToCamelCase" value="true"/>
    </settings>

    <!--自定义别名-->
    <typeAliases>

        <!--配置单个别名 忽略大小写-->
<!--        <typeAlias type="com.gxa.pojo.User"  alias="user"/>-->

        <!--批量定义别名-->
        <package name="com.gxa.pojo"/>

    </typeAliases>
```

8.动态sql

8.1 if标签

8.1.1 多条件查询

需求:

根据用户的状态和用户名进行模糊查询

编写sql

```Java
select * from gxa_user where `status`=1 and username like '%强%'
```

编写DTO

```Java
/**
 * Created by zxd on 2022/10/19 15:20
 */
public class UserQueryDto implements Serializable {

    private String username;

    private Integer status;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public Integer getStatus() {
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }
}

```

编写接口

```java
List<User> findUserByQueryDto(UserQueryDto userQueryDto);
```

编写xml

```java
  <select id="findUserByQueryDto" parameterType="userQueryDto" resultType="user">
            select * from gxa_user where `status`=#{status} and username like '%${username}%'
        </select>

```

测试

```java
  @Test
    public void testFind(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);


        UserQueryDto userQueryDto=new UserQueryDto();
        userQueryDto.setUsername("强");
        userQueryDto.setStatus(1);

        List<User> userList = userMapper.findUserByQueryDto(userQueryDto);


        System.out.println(userList);


    }

```

需求变更:

必须跟两个条件

根据状态查询用户信息

根据用户名模糊查询

根据用户的状态和用户名进行模糊查询

不跟条件

8.1.2 if标签改造

```Java
        <select id="findUserByQueryDto" parameterType="userQueryDto" result
Type="user">

            select * from gxa_user
            where 1=1
            <if test="status !=null ">
                and `status`=#{status}
            </if>
            <if test="username !=null and username !=''">
                and  username like '%${username}%'
            </if>

        </select>
```

where后面不能直接跟上and

8.2 where标签

where标签可以自动生成where，并且自动去除where后的第一个and

```java
        <select id="findUserByQueryDto" parameterType="userQueryDto" result
Type="user">

            select * from gxa_user
           <where>
            <if test="status !=null ">
                and `status`=#{status}
            </if>
            <if test="username !=null and username !=''">
                and  username like '%${username}%'
            </if>
           </where>

        </select>
```

8.3 sql片段

sql语句有公共的语句，可以把公共的语句抽取出来

include标签来引入

```Java
    <sql id="selectUserFrom">
        select * from gxa_user
    </sql>

        <!--查询标签-->
        <!--id sql唯一的标记 statement的id或者sql的id,在一个命名空间中唯一-->
        <!--parameterType: 输入参数类型-->
        <select id="findUserById" parameterType="long" resultType="user">
             <include refid="selectUserFrom"/>
             where id=#{id}
        </select>
```

8.4 foreach标签

准备sql

```Java
select * from gxa_user where id in(1,2,3)
```

改造dto

```java
/**
 * Created by zxd on 2022/10/19 15:20
 */
public class UserQueryDto implements Serializable {

    private String username;

    private Integer status;

    private List<Long> ids;

    public List<Long> getIds() {
        return ids;
    }

    public void setIds(List<Long> ids) {
        this.ids = ids;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public Integer getStatus() {
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }
}

```

编写接口

```java
List<User> findUserByIds(UserQueryDto userQueryDto);
```

编写xml

```java
        <select id="findUserByIds" parameterType="userQueryDto" resultType="user">
            select * from gxa_user
            <!-- where id in(1,2,3) -->
            <where>

                <!--collection: 需要跌到集合-->
                 <!--item: 代表迭代出来的每一项-->
                 <!--open 在迭代之前添加的sql-->
                 <!--separator 指定分隔符-->
                 <!--close在迭代后添加的sql-->
                 <foreach collection="ids" item="id" open="id in(" separator="," close=")">
                     #{id}
                 </foreach>

            </where>
        </select>
```

测试

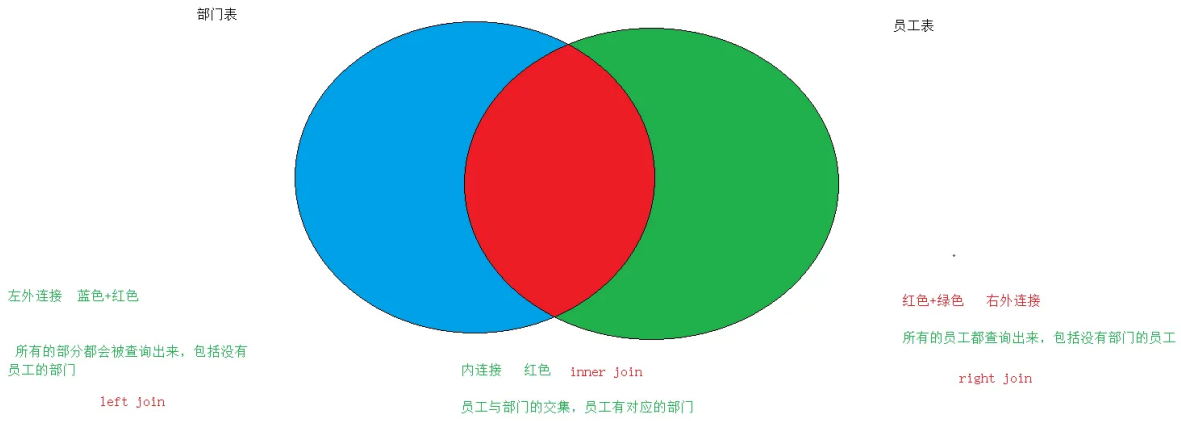
```java
@Test
  public void testFind(){

      SqlSession sqlSession = MybatisUtils.getSqlSession();

      UserMapper userMapper = sqlSession.getMapper(UserMapper.class);


      UserQueryDto userQueryDto=new UserQueryDto();
      userQueryDto.setIds(Arrays.asList(1L,2L,3L));


      List<User> userList = userMapper.findUserByIds(userQueryDto);

      System.out.println(userList);


  }

```

9.关联查询

9.1 一对一查询

需求: 查询所有订单，关联查询订单下面所有用户

一个订单记录会对应一个用户信息



9.1.1 编写sql

Java

```
SELECT
    o.id,
    o.order_number,
    o.payment,
    u.username,
    u.phone
FROM
    gxa_order o,
    gxa_user u
WHERE
    o.user_id = u.id
```

信息 结果 1 剖析 状态

id	order_number	payment	username	phone
1	xxx001	1	吴优哥雅	sada@126.com
2	xxx002	1	林老师	sada@126.com
3	xxx003	1	强哥	sada@126.com

9.1.2 编写vo接口结果

Java

```java
/**
 * Created by zxd on 2022/10/19 16:52
 */
public class OrderUserVo implements Serializable {


    private Integer id;

    private String orderNumber;

    private Integer payment;

    private String username;

    private String phone;

    public Integer getId() {
        return id;
    }

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

    public String getOrderNumber() {
        return orderNumber;
    }

    public void setOrderNumber(String orderNumber) {
        this.orderNumber = orderNumber;
    }

    public Integer getPayment() {
        return payment;
    }

    public void setPayment(Integer payment) {
        this.payment = payment;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
```

```
        this.username = username;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public String toString() {
        return "OrderUserVo{" +
                "id=" + id +
                ", orderNumber='" + orderNumber + '\'' +
                ", payment=" + payment +
                ", username='" + username + '\'' +
                ", phone='" + phone + '\'' +
                '}';
    }
}
```

9.1.3 编写接口

```java
/**
 * Created by zxd on 2022/10/19 14:14
 */
public interface OrderMapper {

    /**
     * 查询所有的订单
     * @return
     */
    List<Order> findOrderAll();


    /**
     * 查询所有订单关联查询用户信息
     * @return
     */
    List<OrderUserVo> findOrderUserAll();

}

```

9.1.4 编写xml

```java

        <select id="findOrderUserAll" resultType="orderUserVo">
                            SELECT
                    o.id,
                    o.order_number,
                    o.payment,
                    u.username,
                    u.phone
                FROM
                    gxa_order o,
                    gxa_user u
                WHERE
                    o.user_id = u.id
        </select>


```

9.1.5 测试

```java
/**
 * Created by zxd on 2022/10/18 16:49
 */
public class TestOrderMapper {



    @Test
    public void testFind(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);


        List<OrderUserVo> orderUserAll = orderMapper.findOrderUserAll();

        System.out.println(orderUserAll);

    }



}
```

9.2 一对一查询(方式二)

使用resultMap

使用面向对象的方式来实现多表查询的结果

9.2.1 修改pojo

```java
/**
 * Created by zxd on 2022/10/19 14:18
 */
public class Order implements Serializable {


    private Long id;

     //...


    //记住用户信息
    private User user;

    //get set

}
```

9.2.2 编写xml

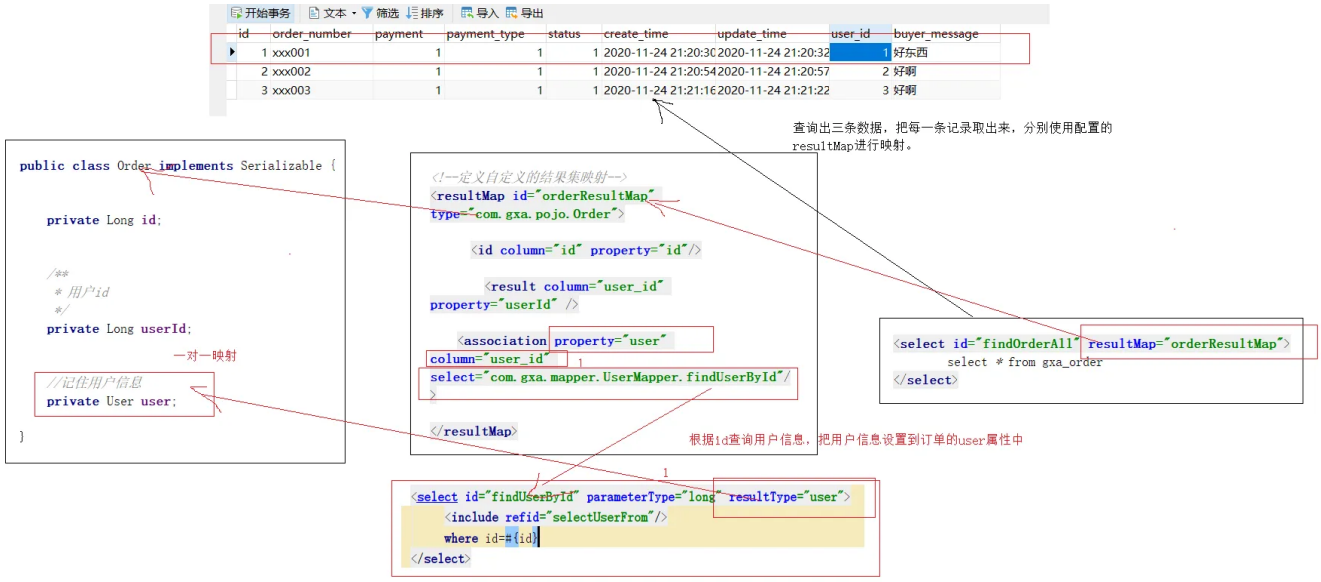
```Java
    <!--定义自定义的结果集映射-->
    <resultMap id="orderResultMap" type="com.gxa.pojo.Order">

        <!--映射主键 column主键在数据库的列名  property主键在pojo的属性名-->
        <!--jdbcType jdbc写法字段类型 javaType java写法字段的类型 不写自动识别-->
        <id column="id" property="id"/>

        <!--映射普通属性-->
        <result column="order_number" property="orderNumber" />
        <result column="payment" property="payment" />
        <result column="payment_type" property="paymentType" />
        <result column="status" property="status" />
        <result column="create_time" property="createTime" />
        <result column="update_time" property="updateTime" />
        <result column="user_id" property="userId" />
        <result column="buyer_message" property="buyerMessage" />

        <!--一对一的关系映射 一个订单只有一个用户信息-->
        <!--property: 需要映射的属性-->
        <!--column: 根据当前表的哪一列作为查询条件-->
        <!--select找对应的sql来执行-->
        <association property="user" column="user_id" select="com.gxa.mapper.UserMapper.findUserById"/>

    </resultMap>


    <select id="findOrderAll" resultMap="orderResultMap">
            select * from gxa_order
    </select>
```

```java

    @Test
    public void testFind(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        OrderMapper orderMapper = sqlSession.getMapper(OrderMapper.class);

        List<Order> orderList = orderMapper.findOrderAll();


        System.out.println(orderList);

    }

```

好处: 配置了一次过后，后面查询订单相关的sql只用考虑订单表的sql，然后配置之前的resultMap，他会根据查询出来的用户id去关联查询用户信息。

```java
    <select id="findOrderById" resultMap="orderResultMap">
          select * from gxa_order where id=#{id}
    </select>
```

resultMap适合于单条数据查询然后关联进行多表查询,如果使用resultMap关联查询列表性能不太好



9.3 一对多的查询

查询用户信息，关联查询用户的订单信息

9.3.1 修改pojp

Java

```java
public class User implements Serializable {

    private Long id;

    //...

    //封装订单信息
    private List<Order> orders;

}
```

9.3.2 编写xml

UserMapper.xml Java

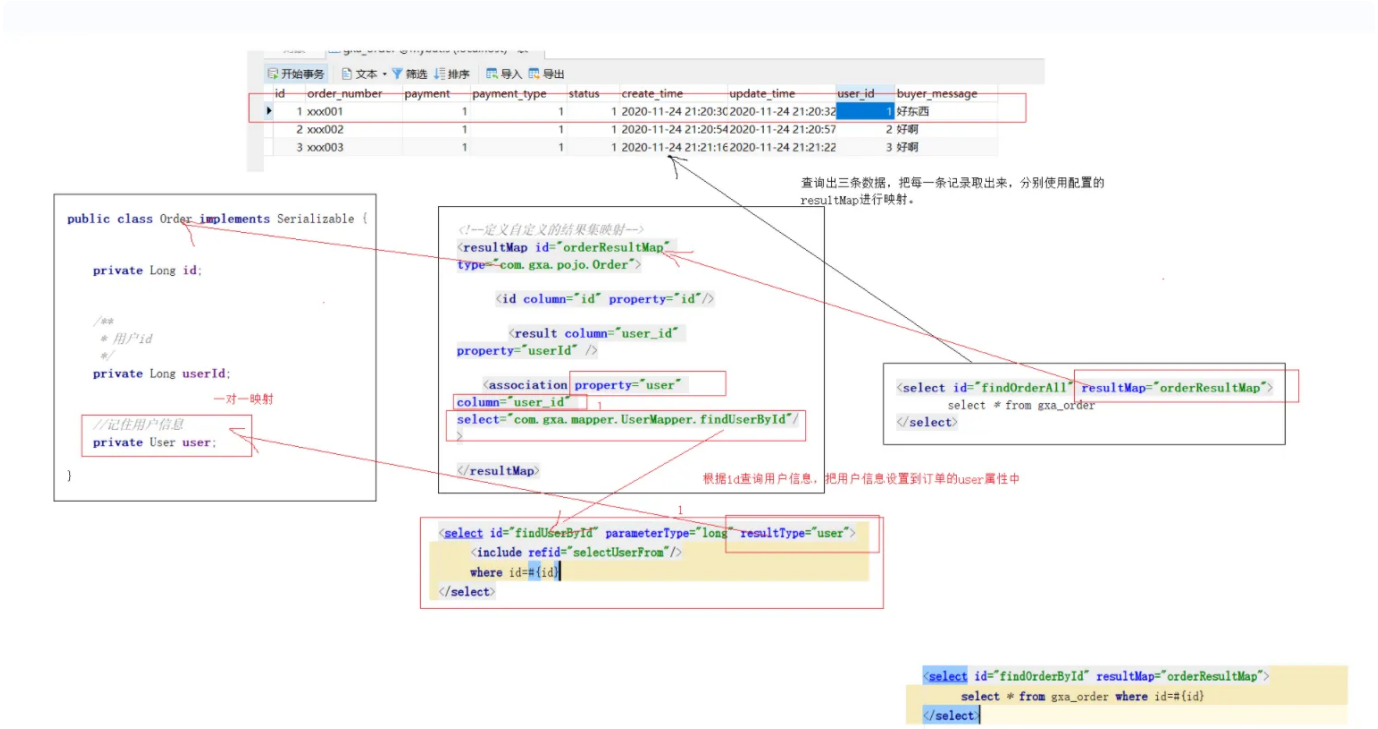
```
 <resultMap id="userResultMap" type="com.gxa.pojo.User">

        <id column="id" property="id"/>

        <!--映射普通属性-->
        <result column="username" property="username" />
        <result column="phone" property="phone" />

        <!--collection 映射一对多的关系-->
        <!--property 需要映射的集合-->
        <!--column 查询条件-->
        <!--select 需要执行的sql 根据id查询这个用户对应的订单信息-->
        <collection property="orders" column="id" select="com.gxa.mapper.OrderMapper.findOrderByUserId"/>

    </resultMap>


    <!--查询标签-->
    <!--id sql唯一的标记 statement的id或者sql的id,在一个命名空间中唯一-->
    <!--parameterType: 输入参数类型-->
    <select id="findUserById" parameterType="long" resultMap="userResultMap">
            select * from gxa_user where id=#{id}
    </select>

```

OrderMapper.xml Java

```
    <!--定义自定义的结果集映射-->
    <resultMap id="orderResultMap" type="com.gxa.pojo.Order">

        <!--映射主键 column主键在数据库的列名  property主键在pojo的属性名-->
        <!--jdbcType jdbc写法字段类型 javaType java写法字段的类型 不写自动识别-->
        <id column="id" property="id"/>

        <!--映射普通属性-->
        <result column="order_number" property="orderNumber" />
        <result column="payment" property="payment" />
        <result column="payment_type" property="paymentType" />
        <result column="status" property="status" />
        <result column="create_time" property="createTime" />
        <result column="update_time" property="updateTime" />
        <result column="user_id" property="userId" />
        <result column="buyer_message" property="buyerMessage" />

        <!--一对一的关系映射 一个订单只有一个用户信息-->
        <!--property: 需要映射的属性-->
        <!--column: 根据当前表的哪一列作为查询条件-->
        <!--select找对应的sql来执行-->
        <association property="user" column="user_id" select="com.gxa.mapper.UserMapper.findUserById"/>

    </resultMap>


    <!--如果一个sql只是内部使用，接口可以不写-->
    <select id="findOrderByUserId" parameterType="long" resultMap="orderResultMap">
            select * from gxa_order where user_id=#{userId}
    </select>
```

9.3.3 测试

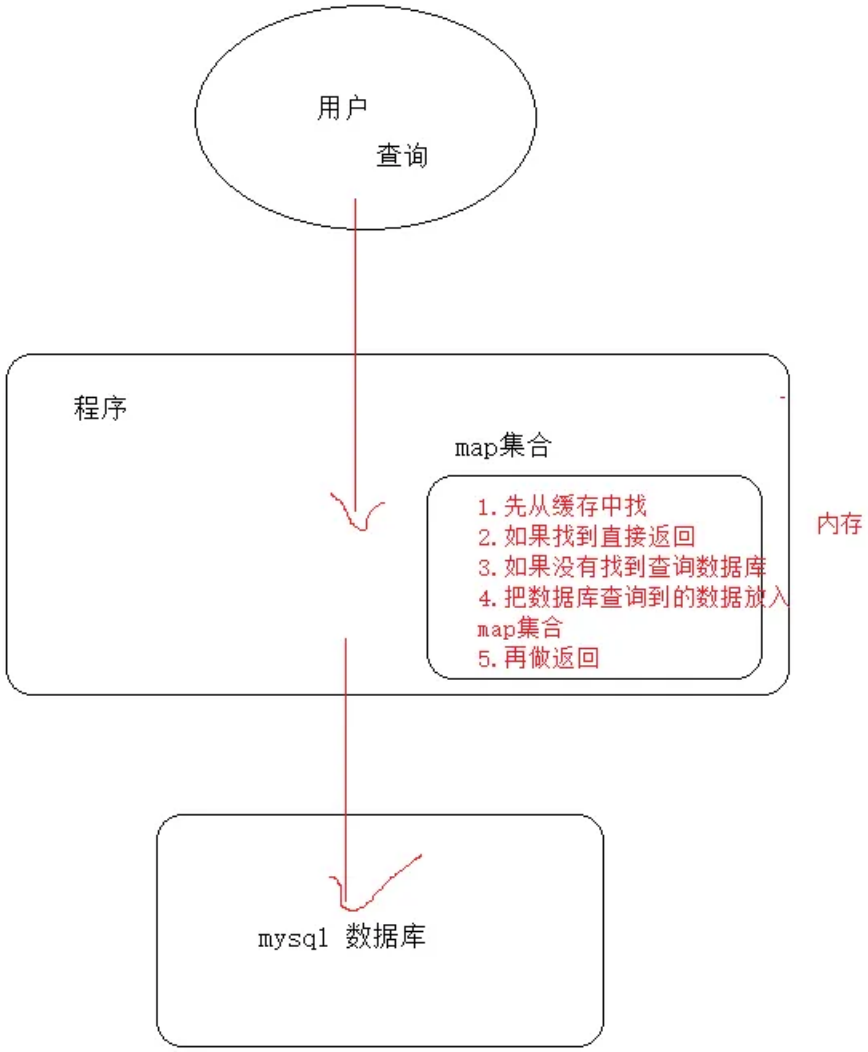
```java
    @Test
    public void testFind(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

        User user = userMapper.findUserById(1L);

        System.out.println(user);

    }

```

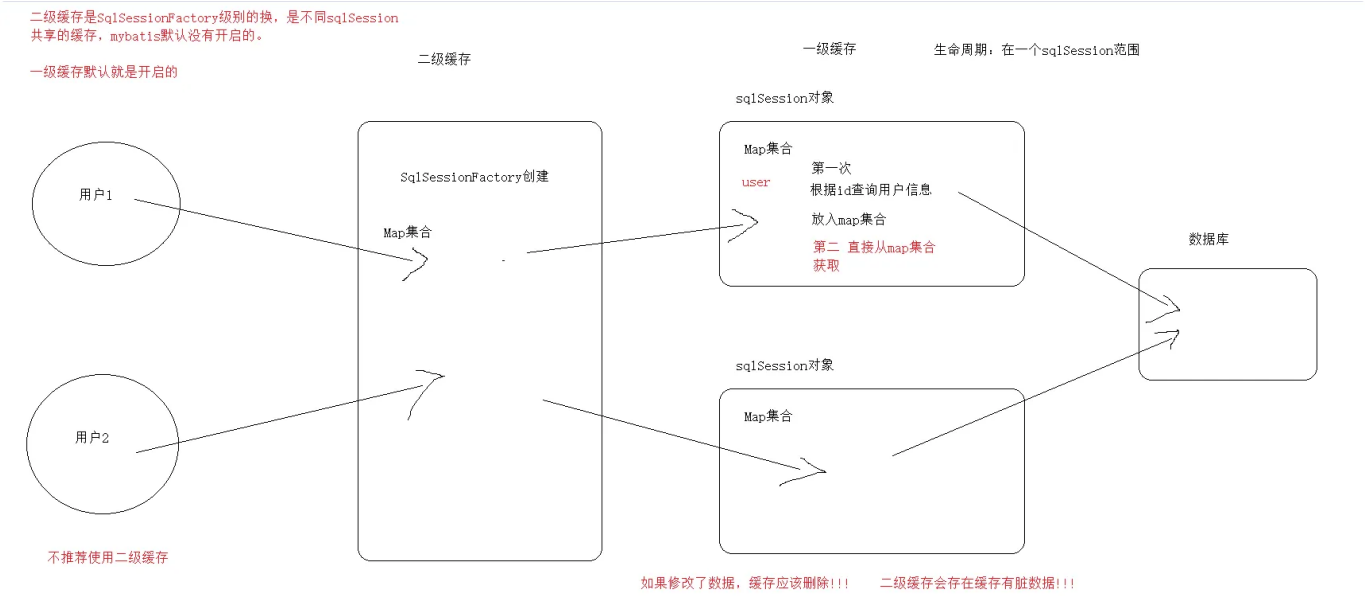


10.缓存(了解)

10.1 缓存的介绍

为什么要使用缓存，缓存可以提高查询速度，减少对数据库的访问，减轻服务器的压力。



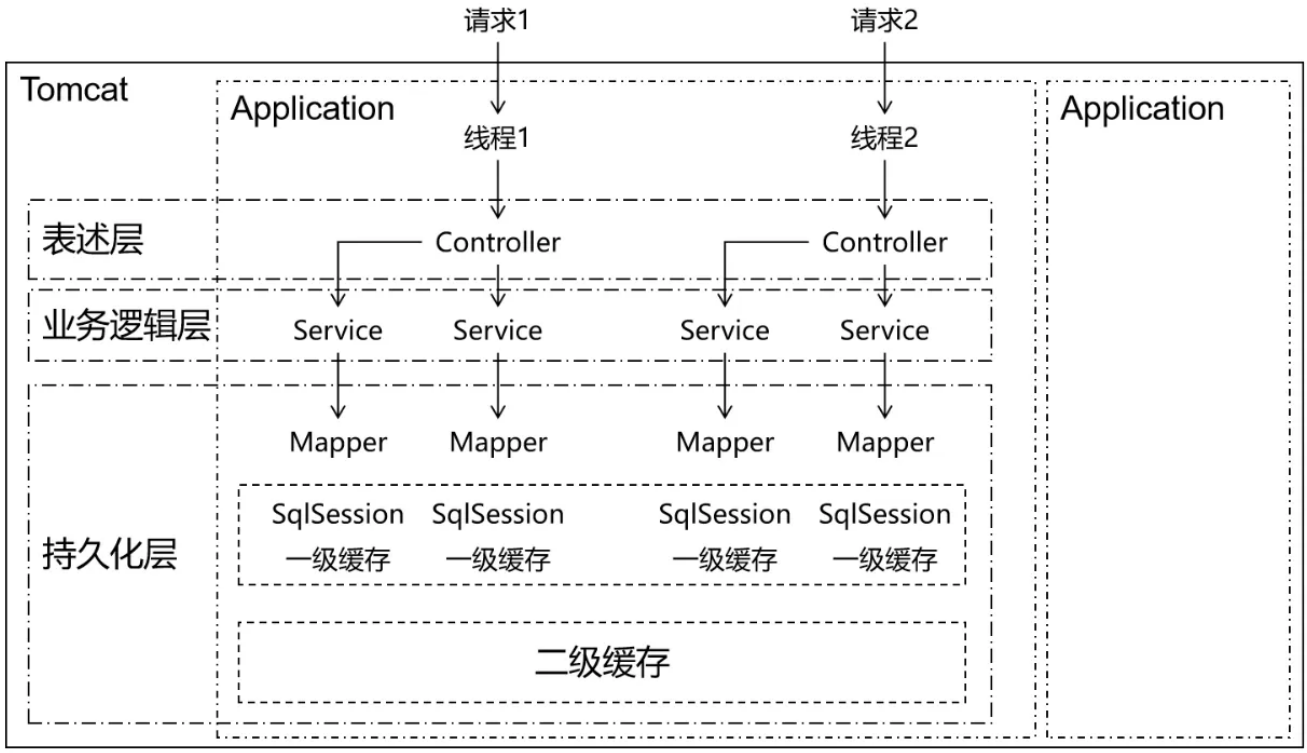


查询的顺序是：

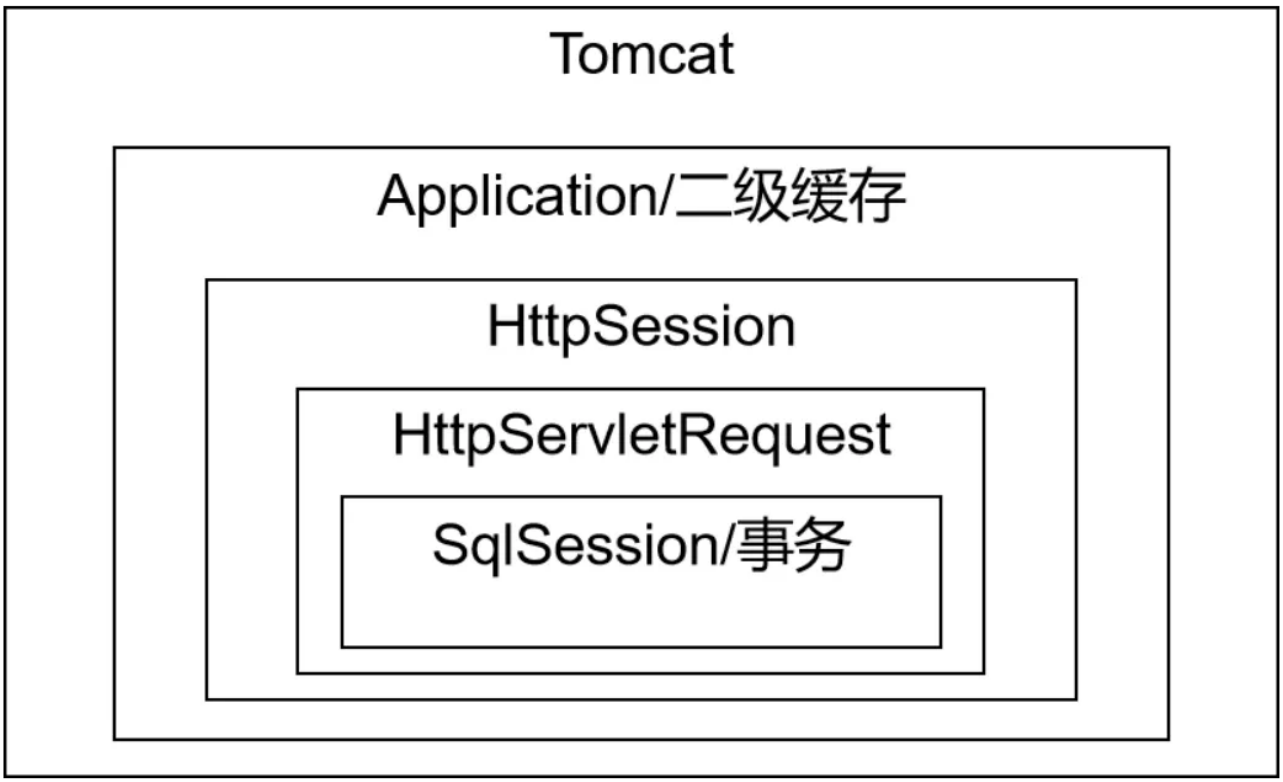
- 先查询二级缓存，因为二级缓存中可能会有其他程序已经查出来的数据，可以拿来直接使用。
- 如果二级缓存没有命中，再查询一级缓存
- 如果一级缓存也没有命中，则查询数据库
- SqlSession关闭之前，一级缓存中的数据会写入二级缓存

②效用范围

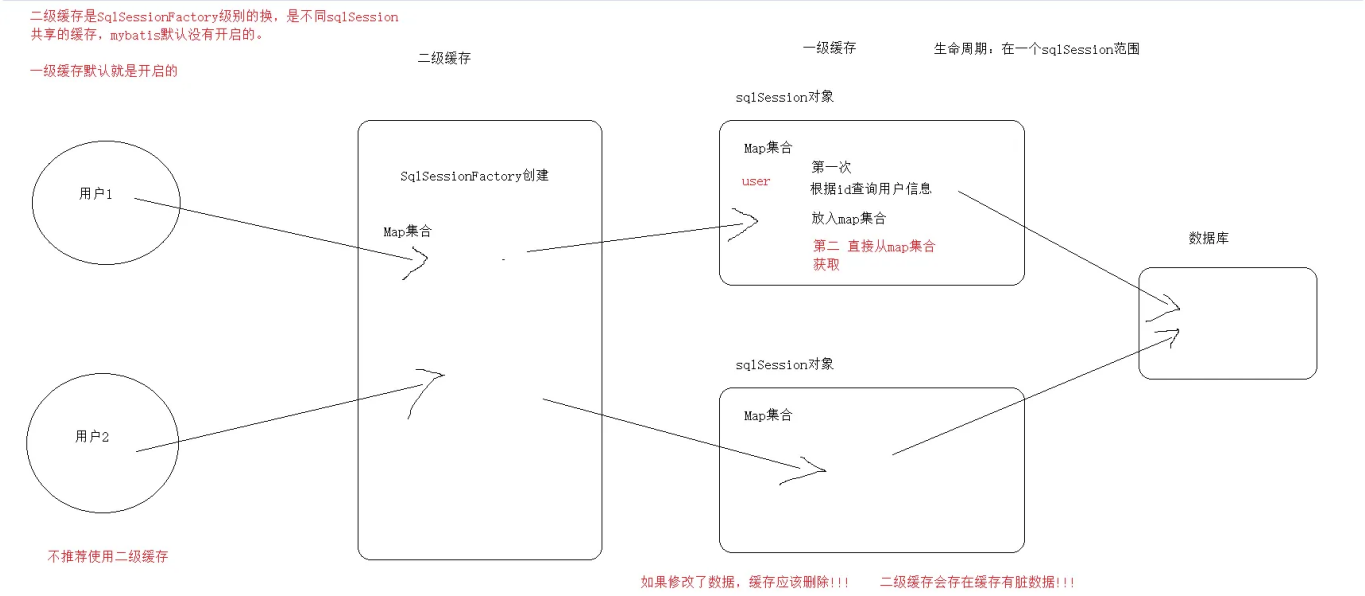
- 一级缓存：SqlSession级别
- 二级缓存：SqlSessionFactory级别



它们之间范围的大小参考下面图：



10.2 一级缓存



Java

```java
@Test
public void testFind(){

    SqlSession sqlSession = MybatisUtils.getSqlSession();

    UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

    User user = userMapper.findUserById(1L);

    System.out.println(user);


    User user2 = userMapper.findUserById(1L);
    System.out.println(user2);


}
```

10.3 二级缓存(了解)

10.3.1 验证二级缓存默认是否存在

```java
@Test
 public void testFind(){

     SqlSession sqlSession = MybatisUtils.getSqlSession();

     UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

     User user = userMapper.findUserById(1L);

     System.out.println(user);

     sqlSession.close();


     SqlSession sqlSession2 = MybatisUtils.getSqlSession();

     UserMapper userMapper2 = sqlSession2.getMapper(UserMapper.class);

     User user2 = userMapper2.findUserById(1L);

     System.out.println(user2);

     sqlSession.close();

 }
```

10.3.2 开启二级缓存

1.在SqlMapConfig.xml开启二级缓存的总开关

```java
<!--配置下划线自动转换驼峰-->
<settings>
    <setting name="mapUnderscoreToCamelCase" value="true"/>
    <!--开启mybatis二级缓存-->
    <setting name="cacheEnabled" value="true"></setting>
</settings>
```

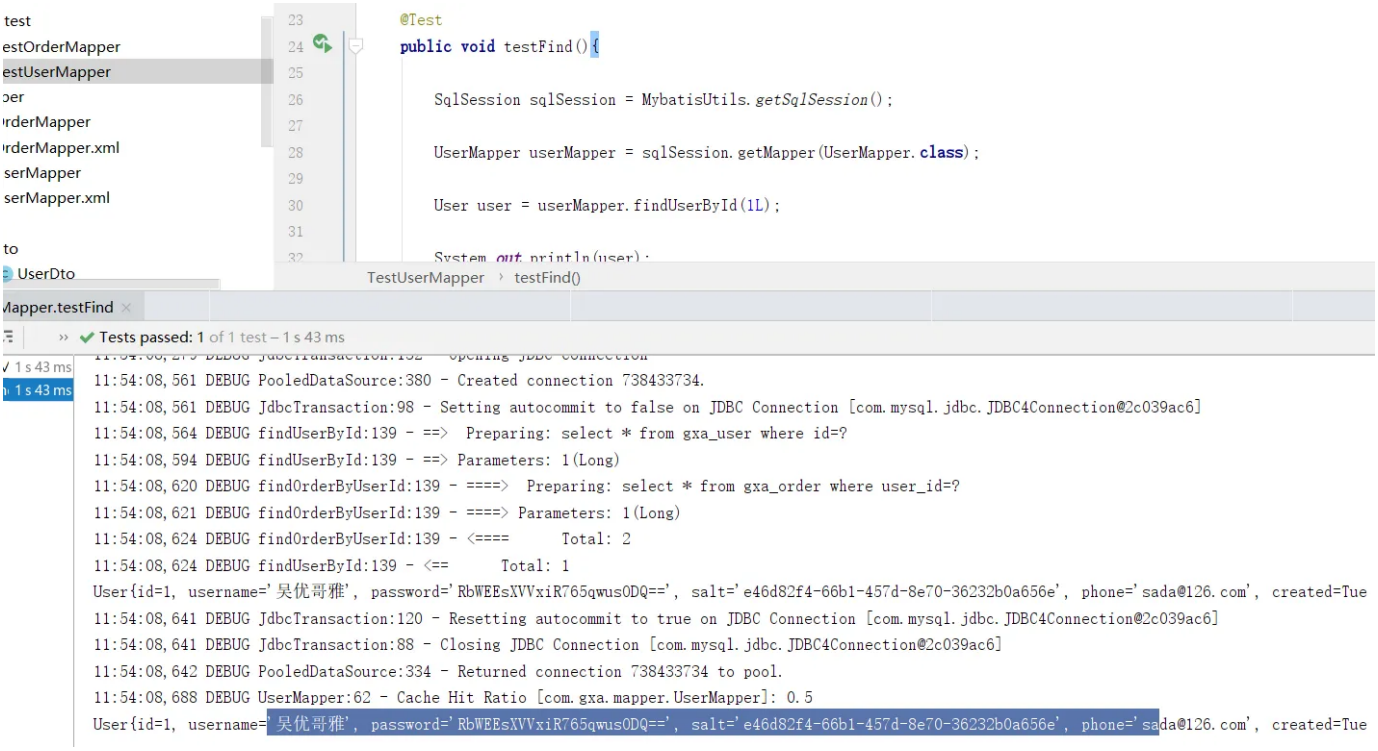
2.在需要使用二级缓存的mapper文件中开启分开关

UserMapper.xml Java

```
1    <!--开启二级缓存的分开关-->
2        <cache></cache>
```

3.pojo必须实现序列化接口

User Order 需要实现序列化接口



二级缓存相关配置

在Mapper配置文件中添加的cache标签可以设置一些属性：

- eviction属性：缓存回收策略LRU（Least Recently Used） — 最近最少使用的：移除最长时间不被使用的对象。FIFO（First in First out） — 先进先出：按对象进入缓存的顺序来移除它们。SOFT — 软引用：移除基于垃圾回收器状态和软引用规则的对象。WEAK — 弱引用：更积极地移除基于垃圾收集器状态和弱引用规则的对象。默认的是 LRU。
- flushInterval属性：刷新间隔，单位毫秒默认情况是不设置，也就是没有刷新间隔，缓存仅仅调用语句时刷新（修删）
- size属性：引用数目，正整数代表缓存最多可以存储多少个对象，太大容易导致内存溢出
- readOnly属性：只读，true/falsetrue：只读缓存；会给所有调用者返回缓存对象的相同实例。因此这些对象不能被修改。这提供了很重要的性能优势。false：读写缓存；会返回缓存对象的拷贝（通过序列化）。这会慢一些，但是安全，因此默认是 false。

10.4 懒加载

全局开启 SqlMapConfig.xml 开启懒加载

Java

```
    <!--配置下划线自动转换驼峰-->
    <settings>
        <setting name="mapUnderscoreToCamelCase" value="true"/>
        <!--开启mybatis二级缓存-->
        <setting name="cacheEnabled" value="true"></setting>


        <!--开启懒加载 lazyLoadingEnable-->
        <setting name="lazyLoadingEnabled" value="true"/>
        <setting name="aggressiveLazyLoading" value="false"/>
        <setting name="lazyLoadTriggerMethods" value=""/>

    </settings>
```

```java
    @Test
    public void testFind(){

        SqlSession sqlSession = MybatisUtils.getSqlSession();

        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

        User user = userMapper.findUserById(1L);

        System.out.println(user);

        System.out.println(user.getOrders());
    }

```

局部配置

```java
 <collection property="orders" column="id" select="com.gxa.mapper.OrderMapper.findOrderByUserId" fetchType="lazy"/>
```

mybatis注解