

知识点：

1、mybatis的基础知识

1.1、对原生态jdbc程序问题解决

1.2、框架原理

1.3、入门程序

1.4、用户的增删改查

1.5、mybatis开发dao的两种方法

1.5.1、原始dao的开发方法（程序中需要些dao接口和dao实现类）

1.5.2、mybatis的mapper代理开发

1.6、mybatis的配置文件 sqlMapConfig.xml

1.7、mybatis输入映射

1.8、mybatis输出映射

2、高级知识

2.1、订单商品模型分析

2.2、高级映射（一对一、一对多、多对多）

2.3、延迟加载

2.4、查询缓存

2.5、逆向工程

使用数据库脚本创建MySQL数据库，包含四张表：用户表、订单表、订单明细表、商品表。

1、基础知识

1.1、原生态jdbc问题总结

使用原生态的jdbc查询mysql数据库用户表的记录时的问题总结：

（1）数据库连接在使用时创建，不使用就立即释放，导致对数据库进行频繁的开启和关闭，造成资源浪费，影响数据库性能。

解决方法：使用数据库连接池来解决。

(2) 将SQL语句硬编码到Java代码中，SQL如果需要修改，程序员需要重新编译Java代码。

解决方法：将SQL配置到xml文件中。

(3) 在向statement设置参数时，占位符的位置和设置参数值硬编码在Java代码中，不利于维护。

解决方法：将SQL语句以及占位符和参数全部配置在xml中。

(4) 从resultset中遍历数据时存在硬编码。将我们要获取的字段进行硬编码，不利于系统维护。

解决方法：设想，自动映射为Java对象。

1.2、mybatis框架：

mybatis是一个持久性的框架，Apache下的顶级项目。mybatis 让程序员将主要精力放在sql上，通过提供的映射，自由灵活地满足SQL语句。mybatis可以向preparedStatement中输入参数自动输入映射，将查询结果集灵活地映射出Java对象（输出映射）。开始时，myBatis托管到Google code下，后来托管到GitHub上，下载框架地

址：<http://github.com/mybatis/mybatis-3/release>，sqlMapConfig.xml是核心配置文件，名称不固定，配置了数据源，事务等mybatis运行环境，mapper.xml配置映射关系（配置SQL语句），SqlSessionFactory（会话工厂）用来创建SqlSession

，SqlSession（会话）是面向用户的接口，作用是操作数据库（发出增删改查操作命令），Executor(执行器)是一个接口，基本执行器和缓存执行器。作用是执行数据库。

MapperStatement（底层封装对象），作用是对操作数据库存储封装，包括SQL语句、输出结果类型。

















1.3、入门程序：

需求：
























根据用户的ID来查询用户信息、根据用户名称实现模糊查询、添加用户、删除用户、更新用户。

环境：

引入项目需要的Jar 包：

-  ant-1.9.6.jar
-  ant-launcher-1.9.6.jar
-  asm-5.2.jar
-  cglib-3.2.5.jar
-  commons-logging-1.2.jar
-  ehcache-core-2.6.5.jar
-  javassist-3.22.0-GA.jar
-  log4j-1.2.17.jar
-  log4j-api-2.3.jar
-  log4j-core-2.3.jar
-  mybatis-3.4.6.jar
-  mybatis-ehcache-1.0.2.jar
-  mysql-connector-java-5.1.25.jar
-  ognl-3.1.16.jar
-  slf4j-api-1.7.25.jar
-  slf4j-log4j12-1.7.25.jar

3.3工程结构:

- ▼  mybatisPractice [mybatisPractice master]
 - ▼  src
 - >  com.cauchy.mybatis.dao
 - >  com.cauchy.mybatis.first
 - >  com.cauchy.mybatis.mapper
 - >  com.cauchy.mybatis.po
 - ▼  config
 - >  sqlmap
 -  db.properties
 -  ehcache.xml
 -  log4j.properties
 -  sqlMapConfig.xml
 - >  JRE System Library [jre1.8.0_191]
 - >  Apache Tomcat v7.0 [Apache Tomcat v7.0]
 - >  Web App Libraries
 - >  JUnit 4
 - >  Referenced Libraries
 -  build
 - ▼  WebContent
 - >  META-INF
 - ▼  WEB-INF
 - >  lib
 -  README.md

配置文件:

sqlMapConfig.xml 文件用于配置运行环境，数据源，事务等信息。

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```

2 <!DOCTYPE configuration
3 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6     <environments default="development">
7         <environment id="development">
8             <!-- 使用jdbc事务管理器 -->
9             <transactionManager type="JDBC"/>
10            <!-- 数据连接池 -->
11            <dataSource type = "POOLED">
12                <property name="driver"
13value="com.mysql.jdbc.Driver"/>
14                <property name="url"
15value="jdbc:mysql://localhost:3306/mybatis"></property>
16                <property name="username" value="root"></property>
17                <property name="password" value="password">
18</property>
19            </dataSource>
20        </environment>
21    </environments>
22</configuration>

```

log4j.properties文件，用于设置日志相关信息：

```

1 log4j.rootLogger=DEBUG,stdout
2 log4j.appender.stdout=org.apache.log4j.ConsoleAppender
3 log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
4 log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

```

根据用户的id来查询用户信息：要先配置映射文件，在映射文件中来配置SQL语句（User.xml，mapper代理的配置文件通常名称为XXXMapper.xml）

要创建一个实体类来接受查询结果：

```

1 package com.cauchy.mybatis.po;
2
3 import java.io.Serializable;
4 import java.util.Date;

```

```
5 import java.util.List;
6
7 public class User {
8     private int id;
9     private String userName;
10    private String sex;
11    private Date birthday;
12    private String address;
13    public int getId() {
14        return id;
15    }
16    public void setId(int id) {
17        this.id = id;
18    }
19    public String getUserName() {
20        return userName;
21    }
22    public void setUserName(String username) {
23        this.userName = username;
24    }
25    public String getSex() {
26        return sex;
27    }
28    public void setSex(String sex) {
29        this.sex = sex;
30    }
31    public Date getBirthday() {
32        return birthday;
33    }
34    public void setBirthday(Date birthday) {
35        this.birthday = birthday;
36    }
37    public String getAddress() {
38        return address;
39    }
40    public void setAddress(String address) {
41        this.address = address;
42    }
43    @Override
44    public String toString() {
45        return "User [id=" + id + ", username=" + userName + ", sex="
+ sex + ", birthday=" + birthday + ", address="
```

```

46         + address + "]);
47     }
48
49 }
50

```

在config/sqlMap下创建User.xml配置文件:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper
3  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <!-- 命名空间，作用为对SQL简单分类化管理，实现SQL隔离 -->
6  <mapper namespace = "test">
7      <!-- 在映射文件中配置SQL语句，通过select语句执行数据库查询，id：标识映
      射文件中的SQL称为statement的id
8          prarmeterType 指定输入参数的类型 ，#{ }表示占位符，#{id}表示接受输
      入的参数，如果输入参数为简单类型，
9              #{ }中的参数名可以任意。resultType为输出类型。
10         -->
11         <select id = "findUserById" parameterType = "int"
      resultType="com.cauchy.mybatis.po.User">
12             select * from user where id = #{id}
13         </select>
14 </mapper>

```

在SqlMapConfig.xml中来加载配置文件。

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE configuration
3  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4  "http://mybatis.org/dtd/mybatis-3-config.dtd">
5  <configuration>
6      <environments default="development">
7          <environment id="development">
8              <!-- 使用jdbc事务管理器 -->
9              <transactionManager type="JDBC"/>
10             <!-- 数据连接池 -->

```

```

11         <dataSource type = "POOLED">
12             <property name="driver"
value="com.mysql.jdbc.Driver"/>
13             <property name="url"
value="jdbc:mysql://localhost:3306/mybatis"></property>
14             <property name="username" value="root"></property>
15             <property name="password" value="password">
</property>
16         </dataSource>
17     </environment>
18 </environments>
19 <mappers>
20     <mapper resource = "sqlmap/UserMapper.xml"/>
21 </mappers>
22 </configuration>

```

MybatisFirst.java:

```

1 package com.cauchy.mybatis.first;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.util.Date;
6
7 import org.apache.ibatis.io.Resources;
8 import org.apache.ibatis.session.SqlSession;
9 import org.apache.ibatis.session.SqlSessionFactory;
10 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
11 import org.junit.Test;
12
13 import com.cauchy.mybatis.po.User;
14
15 public class MybatisFirst {
16     @Test
17     public void findUserById() throws IOException{
18         // 配置文件
19         String resource = "sqlMapConfig.xml";
20         // 得到配置文件流
21         InputStream inputStream =
Resources.getResourceAsStream(resource);

```

```

22      // 创建对话工厂：
23      SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
24      // 获取对话：
25      SqlSession sqlSession = sqlSessionFactory.openSession();
26      // 查询：
27      User user = sqlSession.selectOne("test.findUserById",1);
28      System.out.println(user);
29      sqlSession.close();
30  }
31 }
32

```

根据用户中文名称查询用户：

配置映射文件，向User.xml文件中添加内容。

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper
3  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <!-- 命名空间，作用为对SQL简单分类化管理，实现SQL隔离 -->
6  <mapper namespace = "test">
7      <!-- 在映射文件中配置SQL语句，通过select语句执行数据库查询，id：标识映
射文件中的SQL称为statement的id
8          prarmeterType 指定输入参数的类型 ，#{ }表示占位符，#{id}表示接受输
入的参数，如果输入参数为简单类型，
9              #{ }中的参数名可以任意。resultType为输出类型。
10         -->
11         <select id = "findUserById" parameterType = "int"
resultType="com.cauchy.mybatis.po.User">
12             select * from user where id = #{id}
13         </select>
14         <!-- 根据名称模糊查询可能返回多个结果，${ }表示拼接字符串，表示将接受的内
容不加任何修饰拼接到SQL中，
15             使用${ }可能会导致SQL注入，${ }中接收的参数内容如果传入的是简单类型，
那么{ }中只能使用value -->
16         <select id = "findUserByName" parameterType="java.lang.String"
resultType = "com.cauchy.mybatis.po.User">
17             select * from user where username like '%${value}%'
18         </select>

```



```
19 </mapper>
```

MybatisFirst.java:

```
1 package com.cauchy.mybatis.first;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.util.Date;
6 import java.util.List;
7
8 import org.apache.ibatis.io.Resources;
9 import org.apache.ibatis.session.SqlSession;
10 import org.apache.ibatis.session.SqlSessionFactory;
11 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
12 import org.junit.Test;
13
14 import com.cauchy.mybatis.po.User;
15
16 public class MybatisFirst {
17     @Test
18     public void findUserById() throws IOException{
19         // 配置文件
20         String resource = "sqlMapConfig.xml";
21         // 得到配置文件流
22         InputStream inputStream =
23             Resources.getResourceAsStream(resource);
24         // 创建对话工厂：
25         SqlSessionFactory sqlSessionFactory = new
26             SqlSessionFactoryBuilder().build(inputStream);
27         // 获取对话：
28         SqlSession sqlSession = sqlSessionFactory.openSession();
29         // 查询：
30         User user = sqlSession.selectOne("test.findUserById",1);
31         System.out.println(user);
32         sqlSession.close();
33     }
34     @Test
35     public void findUserByName() throws IOException {
36         // 配置文件
```

```

35     String resource = "SqlMapConfig.xml";
36     // 得到配置文件流
37     InputStream inputStream =
Resources.getResourceAsStream(resource);
38     //创建对话工厂
39     SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
40     // 获取对话
41     SqlSession sqlSession = sqlSessionFactory.openSession();
42     // 根据用户的名称来模糊查询
43     List<Object> list =
sqlSession.selectList("test.findUserByName", "朱");
44     for(Object user:list) {
45         System.out.println(user);
46     }
47     sqlSession.close();
48 }
49 }
50

```

总结:

parameterType作用是在映射文件中指定输入的类型；resultType作用是在映射文件中指定输出结果的类型；#{ }表示一个占位符\${ }表示拼接符号，会引起sql注入，不建议使用。selectOne表示查询出一条数据，如果使用selectOne可以实现，那么使用selectList也可以实现。selectList 表示查询出多条记录。

添加用户的需求:

向User.xml配置文件中添加内容:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper
3  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <!-- 命名空间，作用为对SQL简单分类化管理，实现SQL隔离 -->
6  <mapper namespace = "test">
7      <!-- 在映射文件中配置SQL语句，通过select语句执行数据库查询，id: 标识映
射文件中的SQL称为statement的id
8      parameterType 指定输入参数的类型 ， #{ }表示占位符，#{id}表示接受输
入的参数，如果输入参数为简单类型，

```

```

9      #{}中的参数名可以任意。resultType为输出类型。
10     -->
11     <select id = "findUserById" parameterType = "int"
resultType="com.cauchy.mybatis.po.User">
12         select * from user where id = #{id}
13     </select>
14     <!-- 根据名称模糊查询可能返回多个结果,${}表示拼接字符串,表示将接受的内
容不加任何修饰拼接到SQL中,
15         使用${}可能会导致SQL注入, ${}中接收的参数内容如果传入的是简单类型,
那么{}中只能使用value -->
16     <select id = "findUserByName" parameterType="java.lang.String"
resultType = "com.cauchy.mybatis.po.User">
17         select * from user where username like '%${value}%'
18     </select>
19     <!-- 添加用户 -->
20     <!-- parameterType 指定的参数类型为pojo, 包括用户信息, #{}中指定pojo的
属性名, 接受属性值 , mybatis通过ognl来获取对象的属性值-->
21     <insert id="insertUser" parameterType =
"com.cauchy.mybatis.po.User">
22         insert into user (id,username,birthday,sex,address)value(#
{id},#{userName},#{birthday},#{sex},#{address})
23     </insert>
24 </mapper>

```

MybatisFirst.java:

```

1 package com.cauchy.mybatis.first;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.util.Date;
6 import java.util.List;
7
8 import org.apache.ibatis.io.Resources;
9 import org.apache.ibatis.session.SqlSession;
10 import org.apache.ibatis.session.SqlSessionFactory;
11 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
12 import org.junit.Test;
13
14 import com.cauchy.mybatis.po.User;

```

```
15
16 public class MybatisFirst {
17     @Test
18     public void findUserById() throws IOException{
19         // 配置文件
20         String resource = "sqlMapConfig.xml";
21         // 得到配置文件流
22         InputStream inputStream =
Resources.getResourceAsStream(resource);
23         // 创建对话工厂：
24         SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
25         // 获取对话：
26         SqlSession sqlSession = sqlSessionFactory.openSession();
27         // 查询：
28         User user = sqlSession.selectOne("test.findUserById",1);
29         System.out.println(user);
30         sqlSession.close();
31     }
32     @Test
33     public void findUserByName() throws IOException {
34         // 配置文件
35         String resource = "SqlMapConfig.xml";
36         // 得到配置文件流
37         InputStream inputStream =
Resources.getResourceAsStream(resource);
38         //创建对话工厂
39         SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
40         // 获取对话
41         SqlSession sqlSession = sqlSessionFactory.openSession();
42         // 根据用户的名称来模糊查询
43         List<Object> list =
sqlSession.selectList("test.findUserByName","朱");
44         for(Object user:list) {
45             System.out.println(user);
46         }
47         sqlSession.close();
48     }
49     @Test
50     public void insertUser() throws IOException {
51         // 配置文件
```

```

52     String resource = "SqlMapConfig.xml";
53     // 得到配置文件流
54     InputStream inputStream =
Resources.getResourceAsStream(resource);
55     //创建对话工厂
56     SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
57     // 获取对话
58     SqlSession sqlSession = sqlSessionFactory.openSession();
59     // 执行插入
60     User user = new User();
61     user.setUsername("Linux");
62     user.setBirthday(new Date());
63     user.setSex("M");
64     user.setAddress("London");
65     sqlSession.insert("test.insertUser", user);
66     // 执行提交
67     sqlSession.commit();
68     sqlSession.close();
69 }
70 }
71

```

自增主键的返回：

MySQL自增主键，执行insert提交之前来自动生成一个自增主键，通过函数能够获取到刚刚插入数据的自增主键

SELECT LAST_INSERTED_ID是在insert操作之后来执行。

UserMapper.xml配置文件：

```

1 <insert id="insertUser" parameterType = "com.cauchy.mybatis.po.User">
2     <!-- 将insert 语句执行后的主键值返回 ,只适用于自增主键.keyProperty: 将
查询到的主键值设置到parameterType 指定的对象的id属性
3         order指定insert之前还是之后拿到值-->
4     <selectKey keyProperty="id" order="AFTER"
resultType="java.lang.Integer">
5         select last_insert_id()
6     </selectKey>
7     insert into user (username,birthday,sex,address)values(

```

```
8 {userName},{#{birthday}},{#{sex}},{#{address}})
  </insert>
```

非自增主键的返回：

使用uuid函数来生成主键，需要修改表中的id字段类型为String，长度设置为35位，执行的思路是通过UUID查询到主键，将主键输入到SQL语句中，执行UUID的执行顺序为insert之前执行。

使用MySQL的uuid：

```
1 <insert id="insertUser" parameterType = "com.cauchy.mybatis.po.User">
2     <!--执行过程，首先通过UUID得到主键，将主键值设置到id属性中-->
3     <selectKey keyProperty="id" order="BEFORE"
4     resultType="java.lang.Integer">
5         select uuid()
6     </selectKey>
7     insert into user (id,username,birthday,sex,address)value(#{
8     id},{#{userName}},{#{birthday}},{#{sex}},{#{address}})
9 </insert>
```

通过Oracle序列来生成主键：

```
1 <selectKey keyProperty="id" order="BEFORE"
2 resultType="java.lang.Integer">
3     select 序列名.next()
4 </selectKey>
```

删除用户：

User.xml配置文件添加删除SQL配置：

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <!-- 命名空间，作用为对SQL简单分类化管理，实现SQL隔离 -->
```

```

6 <mapper namespace = "test">
7     <!-- 在映射文件中配置SQL语句，通过select语句执行数据库查询，id: 标识映射文件中的SQL称为statement的id
8         prarmeterType 指定输入参数的类型 ，#{ }表示占位符，#{id}表示接受输入的参数，如果输入参数为简单类型，
9         #{ }中的参数名可以任意。resultType为输出类型。
10     -->
11     <select id = "findUserById" parameterType = "int"
resultType="com.cauchy.mybatis.po.User">
12         select * from user where id = #{id}
13     </select>
14     <!-- 根据名称模糊查询可能返回多个结果,${ }表示拼接字符串，表示将接受的内容不加任何修饰拼接到SQL中，
15         使用${ }可能会导致SQL注入，${ }中接收的参数内容如果传入的是简单类型，那么{ }中只能使用value -->
16     <select id = "findUserByName" parameterType="java.lang.String"
resultType = "com.cauchy.mybatis.po.User">
17         select * from user where username like '%${value}%'
18     </select>
19     <!-- 添加用户 -->
20     <!-- parameterType 指定的参数类型为pojo，包括用户信息，#{ }中指定pojo的属性名，接受属性值 ，mybatis通过ognl来获取对象的属性值-->
21     <insert id="insertUser" parameterType =
"com.cauchy.mybatis.po.User">
22         insert into user (id,username,birthday,sex,address)value(#{id},#{userName},#{birthday},#{sex},#{address})
23     </insert>
24     <delete id="deleteUser" parameterType = "java.lang.Integer">
25         delete from user where id = #{id}
26     </delete>
27 </mapper>

```

MybatisFirst.java:

```

1 package com.cauchy.mybatis.first;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.util.Date;
6 import java.util.List;

```

```
7
8 import org.apache.ibatis.io.Resources;
9 import org.apache.ibatis.session.SqlSession;
10 import org.apache.ibatis.session.SqlSessionFactory;
11 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
12 import org.junit.Test;
13
14 import com.cauchy.mybatis.po.User;
15
16 public class MybatisFirst {
17     @Test
18     public void findUserById() throws IOException{
19         // 配置文件
20         String resource = "sqlMapConfig.xml";
21         // 得到配置文件流
22         InputStream inputStream =
23             Resources.getResourceAsStream(resource);
24         // 创建对话工厂：
25         SqlSessionFactory sqlSessionFactory = new
26             SqlSessionFactoryBuilder().build(inputStream);
27         // 获取对话：
28         SqlSession sqlSession = sqlSessionFactory.openSession();
29         // 查询：
30         User user = sqlSession.selectOne("test.findUserById",1);
31         System.out.println(user);
32         sqlSession.close();
33     }
34     @Test
35     public void findUserByName() throws IOException {
36         // 配置文件
37         String resource = "SqlMapConfig.xml";
38         // 得到配置文件流
39         InputStream inputStream =
40             Resources.getResourceAsStream(resource);
41         //创建对话工厂
42         SqlSessionFactory sqlSessionFactory = new
43             SqlSessionFactoryBuilder().build(inputStream);
44         // 获取对话
45         SqlSession sqlSession = sqlSessionFactory.openSession();
46         // 根据用户的名称来模糊查询
47         List<Object> list =
48             sqlSession.selectList("test.findUserByName", "朱");
```



```
44     for(Object user:list) {
45         System.out.println(user);
46     }
47     sqlSession.close();
48 }
49 @Test
50 public void insertUser() throws IOException {
51     // 配置文件
52     String resource = "SqlMapConfig.xml";
53     // 得到配置文件流
54     InputStream inputStream =
Resources.getResourceAsStream(resource);
55     //创建对话工厂
56     SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
57     // 获取对话
58     SqlSession sqlSession = sqlSessionFactory.openSession();
59     // 执行插入
60     User user = new User();
61     user.setUserName("Linux");
62     user.setBirthday(new Date());
63     user.setSex("M");
64     user.setAddress("London");
65     sqlSession.insert("test.insertUser", user);
66     // 执行提交
67     sqlSession.commit();
68     sqlSession.close();
69 }
70 @Test
71 public void deleteUser() throws IOException {
72     // 配置文件
73     String resource = "SqlMapConfig.xml";
74     // 得到配置文件流
75     InputStream inputStream =
Resources.getResourceAsStream(resource);
76     //创建对话工厂
77     SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
78     // 获取对话
79     SqlSession sqlSession = sqlSessionFactory.openSession();
80     // 执行删除
81     sqlSession.delete("test.deleteUser", 12);
```

```

82         // 执行提交
83         sqlSession.commit();
84         sqlSession.close();
85     }
86 }
87

```

更新用户:

User.xml配置文件添加配置信息:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper
3  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <!-- 命名空间，作用为对SQL简单分类化管理，实现SQL隔离 -->
6  <mapper namespace = "test">
7      <!-- 在映射文件中配置SQL语句，通过select语句执行数据库查询，id: 标识映射文件中的SQL称为statement的id
8          prarmeterType 指定输入参数的类型 ，#{ }表示占位符，#{id}表示接受输入的参数，如果输入参数为简单类型，
9          #{ }中的参数名可以任意。resultType为输出类型。
10         -->
11         <select id = "findUserById" parameterType = "int"
12         resultType="com.cauchy.mybatis.po.User">
13             select * from user where id = #{id}
14         </select>
15         <!-- 根据名称模糊查询可能返回多个结果,${ }表示拼接字符串，表示将接受的内容不不加任何修饰拼接到SQL中，
16             使用${ }可能会导致SQL注入，${ }中接收的参数内容如果传入的是简单类型，那么{ }中只能使用value -->
17         <select id = "findUserByName" parameterType="java.lang.String"
18         resultType = "com.cauchy.mybatis.po.User">
19             select * from user where username like '%${value}%'
20         </select>
21         <!-- 添加用户 -->
22         <!-- parameterType 指定的参数类型为pojo，包括用户信息，#{ }中指定pojo的属性名，接受属性值 ，mybatis通过ognl来获取对象的属性值-->
23         <insert id="insertUser" parameterType =
24         "com.cauchy.mybatis.po.User">
25             insert into user (id,username,birthday,sex,address)value(#{

```

```

{id},{#userName},{#birthday},{#sex},{#address})
23     </insert>
24     <delete id="deleteUser" parameterType = "java.lang.Integer">
25         delete from user where id = #{id}
26     </delete>
27     <!-- parameter 指定parameterType 指定user对象包含id以及要更新的内容,
id必须存在-->
28     <update id="updateUser" parameterType =
"com.cauchy.mybatis.po.User">
29         update user set username = #{userName},birthday = #{birthday},
30         sex = #{sex},address = #{address} where id = #{id}
31     </update>
32 </mapper>

```

MybatisFirst.java:

```

1  package com.cauchy.mybatis.first;
2
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.util.Date;
6  import java.util.List;
7
8  import org.apache.ibatis.io.Resources;
9  import org.apache.ibatis.session.SqlSession;
10 import org.apache.ibatis.session.SqlSessionFactory;
11 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
12 import org.junit.Test;
13
14 import com.cauchy.mybatis.po.User;
15
16 public class MybatisFirst {
17     @Test
18     public void findUserById() throws IOException{
19         // 配置文件
20         String resource = "sqlMapConfig.xml";
21         // 得到配置文件流
22         InputStream inputStream =
Resources.getResourceAsStream(resource);
23         // 创建对话工厂:

```

```

24         SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
25         // 获取对话:
26         SqlSession sqlSession = sqlSessionFactory.openSession();
27         // 查询:
28         User user = sqlSession.selectOne("test.findUserById",1);
29         System.out.println(user);
30         sqlSession.close();
31     }
32     @Test
33     public void findUserByName() throws IOException {
34         // 配置文件
35         String resource = "sqlMapConfig.xml";
36         // 得到配置文件流
37         InputStream inputStream =
Resources.getResourceAsStream(resource);
38         //创建对话工厂
39         SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
40         // 获取对话
41         SqlSession sqlSession = sqlSessionFactory.openSession();
42         // 根据用户的名称来模糊查询
43         List<Object> list =
sqlSession.selectList("test.findUserByName","朱");
44         for(Object user:list) {
45             System.out.println(user);
46         }
47         sqlSession.close();
48     }
49     @Test
50     public void insertUser() throws IOException {
51         // 配置文件
52         String resource = "sqlMapConfig.xml";
53         // 得到配置文件流
54         InputStream inputStream =
Resources.getResourceAsStream(resource);
55         //创建对话工厂
56         SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
57         // 获取对话
58         SqlSession sqlSession = sqlSessionFactory.openSession();
59         // 执行插入

```

```
60     User user = new User();
61     user.setUserName("Linux");
62     user.setBirthday(new Date());
63     user.setSex("M");
64     user.setAddress("London");
65     sqlSession.insert("test.insertUser", user);
66     // 执行提交
67     sqlSession.commit();
68     sqlSession.close();
69 }
70 @Test
71 public void deleteUser() throws IOException {
72     // 配置文件
73     String resource = "sqlMapConfig.xml";
74     // 得到配置文件流
75     InputStream inputStream =
Resources.getResourceAsStream(resource);
76     //创建对话工厂
77     SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
78     // 获取对话
79     SqlSession sqlSession = sqlSessionFactory.openSession();
80     // 执行删除
81     sqlSession.delete("test.deleteUser", 12);
82     // 执行提交
83     sqlSession.commit();
84     sqlSession.close();
85 }
86 @Test
87 public void updateUser() throws IOException {
88     // 配置文件
89     String resource = "SqlMapConfig.xml";
90     // 得到配置文件流
91     InputStream inputStream =
Resources.getResourceAsStream(resource);
92     //创建对话工厂
93     SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
94     // 获取对话
95     SqlSession sqlSession = sqlSessionFactory.openSession();
96     User user = new User();
97     user.setUserName("LinDa");
```

```

98         user.setBirthday(new Date());
99         user.setSex("M");
100        user.setAddress("JiNan");
101        user.setId(9);
102        // 执行删除
103        sqlSession.update("test.updateUser", user);
104        // 执行提交
105        sqlSession.commit();
106        sqlSession.close();
107    }
108
109 }
110

```

Hibernate和mybatis的本质区别：

Hibernate是一个入门门槛较高的标准orm Dao层框架，本身不需要程序员来写SQL语句，由系统自动生成。对SQL语句进行优化比较困难，mybatis专注SQL本身，需要程序员自己编写SQL语句，SQL语句的修改和优化比较简单。mybatis是一个不完全的orm框架，虽然是程序员自己写SQL，但是也实现了映射。在应用场景上，Hibernate适用于需求变化不多的中小型项目，mybatis适用于需求变化比较到的项目，例如互联网项目，企业进行技术选型，以低成本高回报的选型原则，适用于项目本身的技术。

在开发中，通过SqlSessionFactoryBuilder来创建会话工厂，只需要当工具类来使用即可，不需要使用单例模式来创建，只需要创建一次即可。会话工厂类SqlSessionFactory创建SqlSession，使用单例模式管理sqlSessionFactory，一旦创建，只使用一个实例。

整合Spring后使用Spring来管理SqlSessionFactory。SqlSession是一个面向用户的接口，sqlSession中提供了很多操作数据库的方法，例如：selectOne返回单个对象，selectList返回多个对象。

SqlSession是线程不安全的，在SqlSession中除了有接口中的方法，还有数据域的属性，SqlSession最佳应用场合在方法体内，定义称为局部变量。

原始dao方法开发：

UserDao.java 接口代码：

```

1 package com.cauchy.mybatis.dao;
2
3 import com.cauchy.mybatis.po.User;

```

```
4
5 public interface UserDao {
6     // 根据id来查询用户信息
7     public User findUserById(int id) throws Exception;
8     // 添加用户的信息
9     public void insertUser(User user) throws Exception;
10    // 删除用户的信息
11    public void deleteUser(int id) throws Exception;
12 }
13
```

UserDao实现类代码:

```
1 package com.cauchy.mybatis.dao;
2
3 import org.apache.ibatis.session.SqlSession;
4 import org.apache.ibatis.session.SqlSessionFactory;
5
6 import com.cauchy.mybatis.po.User;
7
8 public class UserDaoImpl implements UserDao{
9     // 需要注入SqlSessionFactory
10    private SqlSessionFactory sqlSessionFactory;
11    public UserDaoImpl(SqlSessionFactory sqlSessionFactory) {
12        this.sqlSessionFactory = sqlSessionFactory;
13    }
14    @Override
15    public User findUserById(int id) throws Exception {
16        SqlSession sqlSession = sqlSessionFactory.openSession();
17        User user = sqlSession.selectOne("test.findUserById",id);
18        sqlSession.close();
19        return user;
20    }
21    @Override
22    public void insertUser(User user) throws Exception {
23        SqlSession sqlSession = sqlSessionFactory.openSession();
24        sqlSession.insert("test.insertUser",user);
25        sqlSession.commit();
26        sqlSession.close();
27    }
28 }
```

```

28     @Override
29     public void deleteUser(int id) throws Exception {
30         SqlSession sqlSession = sqlSessionFactory.openSession();
31         sqlSession.delete("test.deleteUser",id);
32         sqlSession.commit();
33         sqlSession.close();
34     }
35
36 }
37

```

测试类代码：

```

1  package com.cauchy.mybatis.dao;
2
3  import java.io.IOException;
4  import java.io.InputStream;
5
6  import org.apache.ibatis.io.Resources;
7  import org.apache.ibatis.session.SqlSessionFactory;
8  import org.apache.ibatis.session.SqlSessionFactoryBuilder;
9  import org.junit.Test;
10
11 import com.cauchy.mybatis.po.User;
12
13 public class UserDaoImplTest {
14     private SqlSessionFactory sqlSessionFactory;
15     public void setUp()throws IOException{
16         // 配置文件
17         String resource = "sqlMapConfig.xml";
18         // 得到配置文件流：
19         InputStream inputStream =
20             Resources.getResourceAsStream(resource);
21         // 创建对话工厂
22         sqlSessionFactory = new
23             SqlSessionFactoryBuilder().build(inputStream);
24     }
25     @Test
26     public void testFindUserById()throws Exception{
27         setUp();
28
29     }
30 }

```



```

26         UserDao userDao = new UserDaoImpl(sqlSessionFactory);
27         User user = userDao.findUserById(1);
28         System.out.println(user);
29     }
30 }

```

思路是程序员需要完成dao接口和dao实现类，需要向dao实现类中注入SqlSessionFactory，在方法体内来创建SqlSession

总结Dao方式的问题：Dao接口实现类存在大量的模板方法，设想可以将其提取出来，大大减轻程序员的工作量；调用session方法时，将statement的id硬编码，调用Sqlsession方法时，由于sqlsession方法使用泛型，即使变量类型传入错误，在编译阶段也不会报错，不利于程序开发。

mapper代理方式：

程序员需要编写mapper接口，mybatis就可以自动生成mapper就可以实现类。程序员还需要编写mapper.xml映射文件

注意事项：

在配置文件中的namespace有特殊作用，为mapper接口的地址，mapper接口中的方法名和mapper.xml中的statement中的id要一致；mapper方法输入参数的类型和mapper.xml文件statement中的parameterType指定的类型一致；mapper接口中的返回值类型和映射文件中的resultType类型一致。

以上的规范主要是对以下的代码进行统一的生成：

```

1 User user = sqlSession.selectOne("test.findUserById",id);
2 sqlSession.insert("test.insertUser",user);
3 .....

```

UserMapper.java：

```

1 package com.cauchy.mybatis.mapper;
2
3 import java.util.List;
4 import com.cauchy.mybatis.po.User;
5
6 public interface UserMapper {

```

```

7      // 根据id来查询用户信息
8      public User findUserById(int id) throws Exception;
9      // 根据用户名返回用户列表
10     public List<User> findUserByName(String name) throws Exception;
11     // 插入用户
12     public void insertUser(User user) throws Exception;
13     // 删除用户
14     public void deleteUser(int id) throws Exception;
15 }

```

UserMapper.xml:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper
3  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <!--使用mapper代理方法，namespace有特殊作用，namespace为mapper接口地址 -->
6  <mapper namespace="com.cauchy.mybatis.mapper.UserMapper">
7      <!-- 在映射文件中配置SQL语句 -->
8      <!-- 通过select执行数据库查询，id: 标识映射文件中的SQL称为statement的id -->
9      <!-- prarmeterType 指定输入参数的类型
10     #{ }表示占位符，#{id} 表示接受输入的参数如果输入的参数为简单类型，#{ }中的参数名可以任意
11     resultType为输出类型，
12     -->
13
14     <select id = "findUserById" parameterType = "int" resultType =
15     "com.cauchy.mybatis.po.User">
16         select * from user where id = #{id}
17     </select>
18     <!-- 根据名称模糊查询可能返回多个结果 -->
19     <!-- ${ }表示拼接字符串，表示将接受的内容不加任何修饰拼接到SQL中，使用
20     $可能会导致SQL注入，${ }中接受的参数内容如果传入的是简单类型
21     {}中只能使用value -->
22     <select id = "findUserByName" parameterType="java.lang.String"
23     resultType = "com.cauchy.mybatis.po.User">
24         select * from user where username like '%${value}%'
25     </select>

```

```

23      <!-- 添加用户 -->
24      <!-- parameterType 指定的参数类型为pojo，包括用户信息，#{ }中指定pojo的
      属性名，接受属性值，
25      mybatis通过ognl来获取对象的属性值-->
26      <insert id="insertUser" parameterType =
      "com.cauchy.mybatis.po.User">
27          <!-- 将insert 语句执行后的主键值返回，只适用于自增主键
28          keyProperty: 将查询到的主键值设置到parameterType 指定的对象的id属
      性，order指定insert之前还是之后拿到值-->
29          <selectKey keyProperty="id" order="AFTER"
      resultType="java.lang.Integer">
30              select last_insert_id()
31          </selectKey>
32          insert into user (username,birthday,sex,address)value(#{
      {username}},#{birthday}},#{sex}},#{address})
33      </insert>
34      <delete id="deleteUser" parameterType = "java.lang.Integer">
35          delete from user where id = #{id}
36      </delete>
37      <!-- parameter 指定parameterType 指定user对象包含id以及要更新的内容，
      id必须存在-->
38      <update id="updateUser" parameterType =
      "com.cauchy.mybatis.po.User">
39          update user set username = #{username},birthday = #{birthday},
40          sex = #{sex},address = #{address} where id = #{id}
41      </update>
42 </mapper>

```

在SqlMapConfig.xml 中添加配置信息：

```

1 <mappers>
2     <mapper resource = "sqlmap/User.xml"/>
3     <mapper resource = "com/cauchy/mybatis/mapper/UserMapper.xml"/>
4 </mappers>

```

UserMapperTest.java:

```

1 package com.cauchy.mybatis.mapper;

```

```
2
3 import static org.junit.Assert.*;
4
5 import java.io.IOException;
6 import java.io.InputStream;
7 import java.util.List;
8
9 import org.apache.ibatis.io.Resources;
10 import org.apache.ibatis.session.SqlSession;
11 import org.apache.ibatis.session.SqlSessionFactory;
12 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
13 import org.junit.Before;
14 import org.junit.Test;
15
16 import com.cauchy.mybatis.po.User;
17
18 public class UserMapperTest {
19     private SqlSessionFactory sqlSessionFactory;
20     @Before
21     public void setUp() throws IOException {
22         String resource = "SqlMapConfig.xml";
23         InputStream inputStream =
24             Resources.getResourceAsStream(resource);
25         sqlSessionFactory = new
26             SqlSessionFactoryBuilder().build(inputStream);
27     }
28     @Test
29     public void testFindUserById() throws Exception {
30
31         // 创建UserMapper对象
32         SqlSession sqlSession = sqlSessionFactory.openSession();
33         UserMapper userMapper =
34             sqlSession.getMapper(UserMapper.class);
35         // 调用UserMapper的方法:
36         User user = userMapper.findUserById(1);
37         System.out.println(user);
38     }
39     @Test
40     public void testFindUserByName() throws Exception {
41
42         // 创建UserMapper对象
43         SqlSession sqlSession = sqlSessionFactory.openSession();
```

```

41     UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
42     // 调用UserMapper的方法:
43     List<User> list = userMapper.findUserByName("朱");
44     System.out.println(list);
45 }
46
47 }
48

```

代理对象内部调用selectOne或者selectList:

如果mapper方法返回单个对象，代理对象内部就会通过selectOne查询，如果返回的是集合对象，则通过selectList来查询。

一些问题的总结：

mapper接口的参数只能有一个，系统是否不利于扩展维护，系统框架中，dao层代码是被业务层公用的，即使mapper接口的参数只有一个，可以使用pojo满足不同业务的需求。

注意：持久层中的参数可以使用包装类型，service方法建议不使用包装类型。

将数据库的连接属性参数单独配置在db.properties中，只需要在配置文件中加载此文件，在SQLMapConfig.xml

将数据库的配置参数放到单独的文件中是为了其他文件方便引用

在SqlMapConfig.xml文件中设置加载：

```

1  <!-- 加载数据库配置信息 -->
2  <properties resource="db.properties"></properties>

```

db.properties文件：

```

1  jdbc.driver=com.mysql.jdbc.Driver
2  jdbc.url=jdbc:mysql://localhost:3306/mybatis
3  jdbc.username=root
4  jdbc.password=password

```

SqlMapConfig.xml:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE configuration
3 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6     <properties resource="db.properties"/>
7     <environments default="development">
8         <environment id="development">
9             <!-- 使用jdbc事务管理器 -->
10            <transactionManager type="JDBC"/>
11            <!-- 数据连接池 -->
12            <dataSource type = "POOLED">
13                <property name="driver"
14value="com.mysql.jdbc.Driver"/>
15                <property name="url"
16value="jdbc:mysql://localhost:3306/mybatis"></property>
17                <property name="username" value="root"></property>
18                <property name="password" value="password">
19            </property>
20            </dataSource>
21        </environment>
22    </environments>
23    <mappers>
24        <mapper resource="sqlmap/User.xml"/>
25        <mapper resource="com/cauchy/mybatis/mapper/UserMapper.xml"/>
26    </mappers>
27 </configuration>
```

properties文件特性有以下特性：在properties元素体内定义的属性首先被读取，然后会读取properties元素中resource或者URL加载的属性，会覆盖已经读取的同名属性。最后读取parameterType传递的属性，他会覆盖同名属性。因此通过parameterType传递的属性有最高的优先级，resource或者URL次之，最低优先级是properties元素体内的属性。

mybatis在运行过程可以更改一些配置参数，全局参数会影响mybatis的运行行为：

typeAliases（别名），在mapper.xml中，定义了很多statement，statement需要parameterType来指定输入参数的类型，还需要指定输出结果的类型，如果在指定输入类型时，输入全路径，不方便进行开发，我们可以来设置定义一些别名，针对parameterType和

resultType指定的类型来定义一些别名，在mapper.xml中，mybatis默认的别名：
针对pojo类型的变量，我们可以自己在sqlMapConfing.xml中配置文件中定义别名：

```
1 <typeAliases>
2     <!-- 针对单个别名 -->
3     <typeAlias type="com.cauchy.mybatis.po.User" alias="user"/>
4 </typeAliases>
```

在UserMapper.xml中应用别名：

```
1 <!-- 此处使用了别名请注意！！ -->
2 <select id = "findUserById" parameterType = "int" resultType = "user">
3     select * from user where id = #{id}
4 </select>
```

批量别名的定义：

```
1 <typeAliases>
2     <!-- 批量别名的定义 别名就是类名，首字母大写或者小写都可以-->
3     <package name="com.cauchy.mybatis.po"/>
4 </typeAliases>
```

typeHandlers（类型处理器）

mybatis中是通过类型处理器来完成jdbc和Java类型的转换。一般不需要我们自定义。

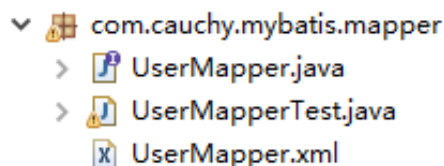
mapper（mapper配置）：

```
1 <mappers>
2     <mapper resource="sqlmap/User.xml"></mapper>
3     <mapper resource="mapper/UserMapper.xml"></mapper>
4 </mappers>
```

来加载单个映射文件

通过mapper接口来加载:

```
1 <mappers>
2     <!-- 通过mapper接口来加载映射文件
3     遵循一些规范: 需要将mapper接口的类名和mapper.xml映射文件名称保持一致, 并
4     且在一个目录
5     上边的规范的前提是使用mapper代理方式,
6     -->
7     <mapper class = "com.cauchy.mybatis.mapper.UserMapper"/>
8     <mapper resource="sqlmap/User.xml"></mapper>
9 </mappers>
```



```
▼ com.cauchy.mybatis.mapper
  > UserMapper.java
  > UserMapperTest.java
  UserMapper.xml
```

按照上面的规范完成代码, 执行测试方法, 可以查询出结果。

批量加载mapper(推荐使用)

```
1 <!-- 指定mapper接口的包名, 批量加载的规范: 需要将mapper接口的类名和
2 mapper.xml映射文件名称保持一致, 并且在一个目录上边的规范的前提是使用mapper
3 代理方式,
4 -->
5 <package name = "com.cauchy.mybatis.mapper"/>
```

输入映射

通过parameterType来指定参数类型类型可以是简单类型 HashMap pojo的包装类型

传递pojo的包装对象:

完成用户信息的综合查询, 需要传入查询条件 (包括用户信息、其他信息)

针对上面的需求, 建议使用自定义的包装类型的pojo

建立UserCustom类:

```
1 package com.cauchy.mybatis.po;
2
```



```

3 public class UserCustom extends User {
4
5 }

```

建立UserQueryVo包装类：

```

1 package com.cauchy.mybatis.po;
2 // 在这类中封装所需的查询条件
3 public class UserQueryVo {
4     // 用户的查询条件
5     private UserCustom userCustom;
6     public UserCustom getUserCustom() {
7         return userCustom;
8     }
9
10    public void setUserCustom(UserCustom userCustom) {
11        this.userCustom = userCustom;
12    }
13
14 }
15

```

在UserMapper.xml中定义用户信息综合查询（查询条件复杂，通过高级查询来进行复杂的查询）

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <!--使用mapper代理方法，namespace有特殊作用，namespace为mapper接口地址 -->
6 <mapper namespace="com.cauchy.mybatis.mapper.UserMapper">
7     <!-- 在映射文件中配置SQL语句，通过select执行数据库查询，id：标识映射文件中的SQL称为statement的id -->
8     <!-- prarmeterType 指定输入参数的类型，#{ }表示占位符，#{id} 表示接受输入的参数如果输入的参数为简单类型，#{ }中的参数名可以任意
9         resultType为输出类型-->
10    <select id = "findUserById" parameterType = "int" resultType =

```

```

"user">
11     select * from user where id = #{id}
12 </select>
13 <!-- 根据名称模糊查询可能返回多个结果 -->
14 <!-- ${}表示拼接字符串，表示将接受的内容不加任何修饰拼接到SQL中，使用
$可能会导致SQL注入，${}中接受的参数内容如果传入的是简单类型
15 {}中只能使用value -->
16 <select id = "findUserByName" parameterType="java.lang.String"
resultType = "com.cauchy.mybatis.po.User">
17     select * from user where username like '%${value}%'
18 </select>
19 <!-- 添加用户 -->
20 <!-- parameterType 指定的参数类型为pojo，包括用户信息，#{ }中指定pojo的
属性名，接受属性值，
21 mybatis通过ognl来获取对象的属性值-->
22 <insert id="insertUser" parameterType =
"com.cauchy.mybatis.po.User">
23     <!-- 将insert 语句执行后的主键值返回，只适用于自增主键
24     keyProperty: 将查询到的主键值设置到parameterType 指定的对象的id属
性，order指定insert之前还是之后拿到值-->
25     <selectKey keyProperty="id" order="AFTER"
resultType="java.lang.Integer">
26         select last_insert_id()
27     </selectKey>
28     insert into user (username,birthday,sex,address)value(#{
userName},#{birthday},#{sex},#{address})
29 </insert>
30 <delete id="deleteUser" parameterType = "java.lang.Integer">
31     delete from user where id = #{id}
32 </delete>
33 <!-- parameter 指定parameterType 指定user对象包含id以及要更新的内容，
id必须存在-->
34 <update id="updateUser" parameterType =
"com.cauchy.mybatis.po.User">
35     update user set username = #{userName},birthday = #{birthday},
36     sex = #{sex},address = #{address} where id = #{id}
37 </update>
38 <!-- 用户信息综合查询 -->
39 <!-- #{userCustom.sex}:取出pojo包装用户的性别 -->
40 <!-- #{userCustom.userName}:取出pojo包装用户的名称 -->
41 <select id="findUserList"
parameterType="com.cauchy.mybatis.po.UserQueryVo"

```

```

42         resultType="com.cauchy.mybatis.po.UserCustom">
43         select * from user where user.sex = #{userCustom.sex}
44         and user.username like '%${userCustom.userName}%'
45     </select>
46 </mapper>

```

UserMapper.java

```

1  package com.cauchy.mybatis.mapper;
2
3  import java.util.List;
4  import com.cauchy.mybatis.po.User;
5  import com.cauchy.mybatis.po.UserCustom;
6  import com.cauchy.mybatis.po.UserQueryVo;
7
8  public interface UserMapper {
9      // 根据id来查询用户信息
10     public User findById(int id) throws Exception;
11     // 根据用户名返回用户列表
12     public List<User> findUserByName(String name) throws Exception;
13     // 插入用户
14     public void insertUser(User user) throws Exception;
15     // 删除用户
16     public void deleteUser(int id) throws Exception;
17     // 用户信息的综合查询
18     public List<UserCustom> findUserList(UserQueryVo
19     userQueryVo) throws Exception;
20 }

```

UserMapperTest.java:

```

1  package com.cauchy.mybatis.mapper;
2
3  import static org.junit.Assert.*;
4
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.util.List;

```

```
8
9 import org.apache.ibatis.io.Resources;
10 import org.apache.ibatis.session.SqlSession;
11 import org.apache.ibatis.session.SqlSessionFactory;
12 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
13 import org.junit.Before;
14 import org.junit.Test;
15
16 import com.cauchy.mybatis.po.User;
17 import com.cauchy.mybatis.po.UserCustom;
18 import com.cauchy.mybatis.po.UserQueryVo;
19
20 public class UserMapperTest {
21     private SqlSessionFactory sqlSessionFactory;
22     @Before
23     public void setUp() throws IOException {
24         String resource = "SqlMapConfig.xml";
25         InputStream inputStream =
26             Resources.getResourceAsStream(resource);
27         sqlSessionFactory = new
28             SqlSessionFactoryBuilder().build(inputStream);
29     }
30     @Test
31     public void testFindUserById() throws Exception {
32
33         // 创建UserMapper对象
34         SqlSession sqlSession = sqlSessionFactory.openSession();
35         UserMapper userMapper =
36             sqlSession.getMapper(UserMapper.class);
37         // 调用UserMapper的方法:
38         User user = userMapper.findUserById(1);
39         System.out.println(user);
40     }
41     @Test
42     public void testFindUserByName() throws Exception {
43
44         // 创建UserMapper对象
45         SqlSession sqlSession = sqlSessionFactory.openSession();
46         UserMapper userMapper =
47             sqlSession.getMapper(UserMapper.class);
48         // 调用UserMapper的方法:
49         List<User> list = userMapper.findUserByName("朱");
```

```

46         System.out.println(list);
47     }
48     @Test
49     public void testFindUserList() throws Exception {
50         SqlSession sqlSession = sqlSessionFactory.openSession();
51         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
52         // 创建包装对象:
53         UserQueryVo userQueryVo = new UserQueryVo();
54         UserCustom userCustom = new UserCustom();
55         userCustom.setSex("M");
56         userQueryVo.setUserCustom(userCustom);
57         // 调用方法:
58         List<UserCustom> list = userMapper.findUserList(userQueryVo);
59         System.out.println(list);
60     }
61 }
62

```

输出映射:

resultType:

pojo类型: 使用resultType作为映射, 只有查询出来的列名称和pojo的属性名一致, 该列才可以映射成功。如果全部不一致, 没有创建pojo对象, 只要查询出来的列名和属性名有一个一致, 就会创建一个pojo对象。

简单类型:

用户信息的综合查询列表总数, 通过查询总数和上边用户综合查询列表才可以实现分页。

UserMapper.xml:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper
3  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <!--使用mapper代理方法, namespace有特殊作用, namespace为mapper接口地址 -->
6  <mapper namespace="com.cauchy.mybatis.mapper.UserMapper">
7      <!-- 在映射文件中配置SQL语句, 通过select执行数据库查询, id: 标识映射文件中的SQL称为statement的id -->

```

```

8      <!-- prarmeterType 指定输入参数的类型 ,#{ }表示占位符, #{id} 表示接受输入
      的参数如果输入的参数为简单类型, #{ }中的参数名可以任意
9      resultType为输出类型-->
10     <select id = "findUserById" parameterType = "int" resultType =
      "user">
11         select * from user where id = #{id}
12     </select>
13     <!-- 根据名称模糊查询可能返回多个结果 -->
14     <!-- ${ }表示拼接字符串, 表示将接受的内容不加任何修饰拼接到SQL中, 使用
      $可能会导致SQL注入, ${ }中接受的参数内容如果传入的是简单类型
15     {}中只能使用value -->
16     <select id = "findUserByName" parameterType="java.lang.String"
      resultType = "com.cauchy.mybatis.po.User">
17         select * from user where username like '%${value}%'
18     </select>
19     <!-- 添加用户 -->
20     <!-- parameterType 指定的参数类型为pojo, 包括用户信息, #{ }中指定pojo的
      属性名, 接受属性值 ,
21     mybatis通过ognl来获取对象的属性值-->
22     <insert id="insertUser" parameterType =
      "com.cauchy.mybatis.po.User">
23         <!-- 将insert 语句执行后的主键值返回 ,只适用于自增主键
24         keyProperty: 将查询到的主键值设置到parameterType 指定的对象的id属
      性, order指定insert之前还是之后拿到值-->
25         <selectKey keyProperty="id" order="AFTER"
      resultType="java.lang.Integer">
26             select last_insert_id()
27         </selectKey>
28         insert into user (username,birthday,sex,address)value(#{
      userName},#{birthday},#{sex},#{address})
29     </insert>
30     <delete id="deleteUser" parameterType = "java.lang.Integer">
31         delete from user where id = #{id}
32     </delete>
33     <!-- parameter 指定parameterType 指定user对象包含id以及要更新的内容,
      id必须存在-->
34     <update id="updateUser" parameterType =
      "com.cauchy.mybatis.po.User">
35         update user set username = #{userName},birthday = #{birthday},
36         sex = #{sex},address = #{address} where id = #{id}
37     </update>
38     <!-- 用户信息综合查询 -->

```

```

39      <!-- #{userCustom.sex}:取出pojo包装用户的性别 -->
40      <!-- #{userCustom.userName}:取出pojo包装用户的名称 -->
41      <select id="findUserList"
parameterType="com.cauchy.mybatis.po.UserQueryVo"
42          resultType="com.cauchy.mybatis.po.UserCustom">
43          select * from user where user.sex = #{userCustom.sex}
44          and user.username like '%${userCustom.userName}%'
45      </select>
46      <!-- 用户信息的综合查询总数: -->
47      <select id="findUserCount" parameterType =
"com.cauchy.mybatis.po.UserQueryVo" resultType = "int">
48          select count(*) from user where user.sex = #{userCustom.sex}
49          and user.username like '%${userCustom.userName}%'
50      </select>
51 </mapper>

```

UserMapper.java:

```

1  package com.cauchy.mybatis.mapper;
2
3  import java.util.List;
4  import com.cauchy.mybatis.po.User;
5  import com.cauchy.mybatis.po.UserCustom;
6  import com.cauchy.mybatis.po.UserQueryVo;
7
8  public interface UserMapper {
9      // 根据id来查询用户信息
10     public User findUserById(int id) throws Exception;
11     // 根据用户名返回用户列表
12     public List<User> findUserByName(String name) throws Exception;
13     // 插入用户
14     public void insertUser(User user) throws Exception;
15     // 删除用户
16     public void deleteUser(int id) throws Exception;
17     // 用户信息的综合查询
18     public List<UserCustom> findUserList(UserQueryVo
userQueryVo) throws Exception;
19     // 用户综合信息的总数查询:
20     public int findUserCount(UserQueryVo userQueryVo) throws Exception;
21 }

```

UserMapperTest.java:

```
1 package com.cauchy.mybatis.mapper;
2
3 import static org.junit.Assert.*;
4
5 import java.io.IOException;
6 import java.io.InputStream;
7 import java.util.List;
8
9 import org.apache.ibatis.io.Resources;
10 import org.apache.ibatis.session.SqlSession;
11 import org.apache.ibatis.session.SqlSessionFactory;
12 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
13 import org.junit.Before;
14 import org.junit.Test;
15
16 import com.cauchy.mybatis.po.User;
17 import com.cauchy.mybatis.po.UserCustom;
18 import com.cauchy.mybatis.po.UserQueryVo;
19
20 public class UserMapperTest {
21     private SqlSessionFactory sqlSessionFactory;
22     @Before
23     public void setUp() throws IOException {
24         String resource = "SqlMapConfig.xml";
25         InputStream inputStream =
26             Resources.getResourceAsStream(resource);
27         sqlSessionFactory = new
28             SqlSessionFactoryBuilder().build(inputStream);
29     }
30     @Test
31     public void testFindUserById() throws Exception {
32
33         // 创建UserMapper对象
34         SqlSession sqlSession = sqlSessionFactory.openSession();
35         UserMapper userMapper =
36             sqlSession.getMapper(UserMapper.class);
37         // 调用UserMapper的方法:
38         User user = userMapper.findUserById(1);
39     }
40 }
```



```
36         System.out.println(user);
37     }
38     @Test
39     public void testFindUserByName() throws Exception {
40
41         // 创建UserMapper对象
42         SqlSession sqlSession = sqlSessionFactory.openSession();
43         UserMapper userMapper =
44         sqlSession.getMapper(UserMapper.class);
45         // 调用UserMapper的方法:
46         List<User> list = userMapper.findUserByName("朱");
47         System.out.println(list);
48     }
49     @Test
50     public void testFindUserList()throws Exception{
51         SqlSession sqlSession = sqlSessionFactory.openSession();
52         UserMapper userMapper =
53         sqlSession.getMapper(UserMapper.class);
54         // 创建包装对象:
55         UserQueryVo userQueryVo = new UserQueryVo();
56         UserCustom userCustom = new UserCustom();
57         userCustom.setSex("M");
58         userQueryVo.setUserCustom(userCustom);
59         // 调用方法:
60         List<UserCustom> list = userMapper.findUserList(userQueryVo);
61         System.out.println(list);
62     }
63     @Test
64     public void testFindUserCount() throws Exception{
65         SqlSession sqlSession = sqlSessionFactory.openSession();
66         UserMapper userMapper =
67         sqlSession.getMapper(UserMapper.class);
68         // 创建包装对象:
69         UserQueryVo userQueryVo = new UserQueryVo();
70         UserCustom userCustom = new UserCustom();
71         userCustom.setSex("M");
72         userCustom.setUserName("朱");
73         userQueryVo.setUserCustom(userCustom);
74         // 调用方法:
75         int count = userMapper.findUserCount(userQueryVo);
76         System.out.println("count is :"+ count);
77     }
78 }
```

```
75 }  
76
```

查询出来的结果集如果只有一行一列，可以使用简单类型的输出映射。

输出pojo对象和pojo列表：

不管输出的是一个对象还是一个列表，在mapper.xml中的resultType指定的类型时一样的，在mapper.java返回值不同。resultMap来完成高级输出结果映射，如果查询出来的列名和pojo的属性名不一致，通过resultMap对列名和属性名进行映射。

定义resultMap：

userCustomer类中属性名和查询的列名不一致，要定义resultMap：

```
1 <resultMap type="com.cauchy.mybatis.po.User" id="userResultMap">  
2     <!-- 标识查询结果集的唯一标识 -->  
3     <id column="id_" property="id"/>  
4     <!-- result: 普通列的标识 -->  
5     <result column="username_" property="userName"/>  
6 </resultMap>  
7 <!-- 使用resultMap进行输出的映射 -->  
8 <select id="findUserByResultMap" parameterType="int"  
9     resultMap = "userResultMap">  
10     select id id_,username username_ from user where id = #{value}  
11 </select>
```

UserMapper.java:

```
1 package com.cauchy.mybatis.mapper;  
2  
3 import java.util.List;  
4 import com.cauchy.mybatis.po.User;  
5 import com.cauchy.mybatis.po.UserCustom;  
6 import com.cauchy.mybatis.po.UserQueryVo;  
7  
8 public interface UserMapper {  
9     // 根据id来查询用户信息  
10     public User findUserById(int id) throws Exception;  
11     // 根据用户名返回用户列表
```

```

12     public List<User> findUserByName(String name) throws Exception;
13     // 插入用户
14     public void insertUser(User user) throws Exception;
15     // 删除用户
16     public void deleteUser(int id)throws Exception;
17     // 用户信息的综合查询
18     public List<UserCustom> findUserList(UserQueryVo
19     userQueryVo)throws Exception;
20     // 用户综合信息的总数查询:
21     public int findUserCount(UserQueryVo userQueryVo)throws Exception;
22     // 使用resultMap查询
23     public User findUserByIdResultMap(int id) throws Exception;
24 }

```

UserMapperTest.java:

```

1 package com.cauchy.mybatis.mapper;
2
3 import java.util.List;
4 import com.cauchy.mybatis.po.User;
5 import com.cauchy.mybatis.po.UserCustom;
6 import com.cauchy.mybatis.po.UserQueryVo;
7
8 public interface UserMapper {
9     // 根据id来查询用户信息
10    public User findUserById(int id) throws Exception;
11    // 根据用户名返回用户列表
12    public List<User> findUserByName(String name) throws Exception;
13    // 插入用户
14    public void insertUser(User user) throws Exception;
15    // 删除用户
16    public void deleteUser(int id)throws Exception;
17    // 用户信息的综合查询
18    public List<UserCustom> findUserList(UserQueryVo
19    userQueryVo)throws Exception;
20    // 用户综合信息的总数查询:
21    public int findUserCount(UserQueryVo userQueryVo)throws Exception;
22    // 使用resultMap查询
23    public User findUserByIdResultMap(int id) throws Exception;
24 }

```

动态SQL

mybatis的核心就是对SQL语句进行灵活的操作，通过表达式进行判断，对表达式进行灵活的拼接组装等等，也就是生成SQL的过程。

需求：用户信息综合查询列表和用户信息查询列表总数这两个statement的定义使用动态SQL

```
1 <!-- 用户信息综合查询 -->
2 <!-- #{userCustom.sex}:取出pojo包装用户的性别 -->
3 <!-- #{userCustom.userName}:取出pojo包装用户的名称 -->
4 <select id="findUserList"
  parameterType="com.cauchy.mybatis.po.UserQueryVo"
 5     resultType="com.cauchy.mybatis.po.UserCustom">
6     select * from user where user.sex = #{userCustom.sex}
7     and user.username like '%${userCustom.userName}%'
8 </select>
```

对查询条件进行判断只有查询结果不为空，才进行查询。

```
1 <select id="findUserList"
  parameterType="com.cauchy.mybatis.po.UserQueryVo"
2     resultType="com.cauchy.mybatis.po.UserCustomer">
3     select * from user
4     <where>
5         <if test="userCustomer != null">
6             <if test="userCustomer.sex != null and
7 userCustomer.sex != ''">
8                 user.sex = #{userCustomer.sex} and
9             </if>
10            <if test="userCustomer.username != null and
11 userCustomer.username != ''">
12                user.username like '%${userCustomer.username}%'
13            </if>
14        </if>
    </where>
</select>
```

sql 片段：

将上面实现的SQL判断代码块抽取出来，组成一个SQL片段，其他的statement就是引用这个SQL片段：

```
1  <!-- 定义一个SQL片段id 为唯一标识,根据经验,SQL片段基于单表定义,在我们的
    SQL片段中,不要包括where -->
2  <sql id="query_user_where">
3      <if test="userCustom != null">
4          <if test="userCustom.sex != null and userCustom.sex != ''">
5              user.sex = #{userCustom.sex}
6          </if>
7          <if test="userCustom.userName != null and userCustom.userName
    != ''">
8              user.username like '%${userCustom.userName}%'
9          </if>
10     </if>
11 </sql>
```

引用SQL片段：

```
1  <select id="findUserList"
    parameterType="com.cauchy.mybatis.po.UserQueryVo"
2      resultType="com.cauchy.mybatis.po.UserCustomer">
3      select * from user
4      <where>
5          <!-- 引用SQL片段 -->
6          <include refid="query_user_where"></include>
7      </where>
8  </select>
```

foreach

向SQL中传递了一个list，mybatis使用foreach解析，如下：

在用户查询列表和用户查询总数的statement中，要增加多个id的输入查询，两种方法，

```
1  select * from user where id = 1 or id = 10 or id = 16
```

使用In

```
1 select * from user where id in (1,10,16)
```

在输入的参数类型中添加List<Integer>属性：

```
1 package com.cauchy.mybatis.po;
2
3 import java.util.List;
4
5 // 在这类中封装所需的查询条件
6 public class UserQueryVo {
7     // 用户的查询条件
8     private UserCustom userCustom;
9     private List<Integer> ids;
10    public List<Integer> getIds() {
11        return ids;
12    }
13
14    public void setIds(List<Integer> ids) {
15        this.ids = ids;
16    }
17
18    public UserCustom getUserCustom() {
19        return userCustom;
20    }
21
22    public void setUserCustom(UserCustom userCustom) {
23        this.userCustom = userCustom;
24    }
25
26 }
27
```

修改UserMapper.xml

在查询条件中，查询条件定义称为一个片段，需要修改SQL片段：

```

1 <sql id="query_user_where">
2     <if test="userCustom != null">
3         <if test="userCustom.sex != null and userCustom.sex !=
4         ''">
5             user.sex = #{userCustom.sex}
6         </if>
7         <if test="userCustom.userName != null and
8         userCustom.userName != ''">
9             user.username like '%${userCustom.userName}%'
10        </if>
11        <if test="ids!=null">
12            <!-- 使用foreach来遍历传入的ids,collection是用来指定
13            集合属性 item用来遍历生成对象中 , open开始遍历时要拼接的串,
14            close使用下边的SQL进行拼接: and (id = 1 or id =
15            10 or id = 16) separator为两个对象需要拼接的串-->
16            <foreach collection="ids" item="user_id"
17            open="and(" close = ")" separator="or">
18                id = #{user_id}
19            </foreach>
20        </if>
21    </if>
22 </sql>

```

UserMapperTest.java:

```

1 package com.cauchy.mybatis.mapper;
2
3 import static org.junit.Assert.*;
4
5 import java.io.IOException;
6 import java.io.InputStream;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 import org.apache.ibatis.io.Resources;
11 import org.apache.ibatis.session.SqlSession;
12 import org.apache.ibatis.session.SqlSessionFactory;
13 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
14 import org.junit.Before;

```

```
15 import org.junit.Test;
16
17 import com.cauchy.mybatis.po.User;
18 import com.cauchy.mybatis.po.UserCustom;
19 import com.cauchy.mybatis.po.UserQueryVo;
20
21 public class UserMapperTest {
22     private SqlSessionFactory sqlSessionFactory;
23     @Before
24     public void setUp() throws IOException {
25         String resource = "SqlMapConfig.xml";
26         InputStream inputStream =
Resources.getResourceAsStream(resource);
27         sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
28     }
29     @Test
30     public void testFindUserById() throws Exception {
31
32         // 创建UserMapper对象
33         SqlSession sqlSession = sqlSessionFactory.openSession();
34         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
35         // 调用UserMapper的方法:
36         User user = userMapper.findUserById(1);
37         System.out.println(user);
38     }
39     @Test
40     public void testFindUserByName() throws Exception {
41
42         // 创建UserMapper对象
43         SqlSession sqlSession = sqlSessionFactory.openSession();
44         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
45         // 调用UserMapper的方法:
46         List<User> list = userMapper.findUserByName("朱");
47         System.out.println(list);
48     }
49     @Test
50     public void testFindUserList()throws Exception{
51         SqlSession sqlSession = sqlSessionFactory.openSession();
52         UserMapper userMapper =
```



```
sqlSession.getMapper(UserMapper.class);
53      // 创建包装对象:
54      UserQueryVo userQueryVo = new UserQueryVo();
55      UserCustom userCustom = new UserCustom();
56      userCustom.setSex("M");
57      userQueryVo.setUserCustom(userCustom);
58      // 调用方法:
59      List<UserCustom> list = userMapper.findUserList(userQueryVo);
60      System.out.println(list);
61  }
62  @Test
63  public void testFindUserCount() throws Exception{
64      SqlSession sqlSession = sqlSessionFactory.openSession();
65      UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
66      // 创建包装对象:
67      UserQueryVo userQueryVo = new UserQueryVo();
68      UserCustom userCustom = new UserCustom();
69      userCustom.setSex("M");
70      userCustom.setUserName("朱");
71      userQueryVo.setUserCustom(userCustom);
72      // 调用方法:
73      int count = userMapper.findUserCount(userQueryVo);
74      System.out.println("count is :"+ count);
75  }
76  @Test
77  public void testFindUserByIdResultMap() throws Exception{
78      SqlSession sqlSession = sqlSessionFactory.openSession();
79      UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
80      // 调用方法:
81      User user = userMapper.findUserByIdResultMap(3);
82      System.out.println(user);
83  }
84  @Test
85  public void testFindUserList2()throws Exception{
86      SqlSession sqlSession = sqlSessionFactory.openSession();
87      UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
88      // 创建包装对象:
89      UserQueryVo userQueryVo = new UserQueryVo();
90      UserCustom userCustom = new UserCustom();
```

```

91         userCustom.setSex("M");
92         // 要传入多个id
93         List<Integer> ids = new ArrayList<Integer>();
94         ids.add(1);
95         ids.add(6);
96         ids.add(10);
97         // 将IDS传入
98         userQueryVo.setIds(ids);
99         userQueryVo.setUserCustom(userCustom);
100        // 调用方法:
101        List<UserCustom> list = userMapper.findUserList(userQueryVo);
102        System.out.println(list);
103    }
104 }
105

```

订单商品数据模型:

数据模型分析思路，要明确每张表记录的是什么，分模块地进行对每张表的内容进行熟悉，相当于你学习系统的需求的过程。记录每张表的重要字段设置，例如非空字段、外键字段。以及数据库级别表与表之间的关系、表与表之间的业务关系，建立在某个业务意义之上来分析。在分析表与表之间的业务关系是需要建立在某个业务基础上面去分析，先分析数据库级别之间有关系的表业务关系。

user和order 一个用户创建多个订单，一对多。

orders和user 一对一，一个订单只能由一个用户创建。

order和orderdetail，一个订单可以包括多个订单明细，因为一个订单可以购买多个商品。一对多。

orderdetail和order:一对一。

orderdetail和item:一个订单的明细只对应一个商品，一对一

item和orderdetail :一个商品可以包含在多个订单明细中，一对多

orders和item :二者之间通过detail表建立联系。一对多

items和orders :一对多。

orders和item是多对多关系。

user和item是多对多的关系。

一对一的查询:

需求：查询订单信息，关联查询创建订单的用户信息：

resultType实现：

SQL语句：

确定查询的主表：订单表

确定查询的关联表：用户表

关联查询使用内连接还是外连接？

通过外键查询用户表只能查询到一条记录，可以使用内连接。

```
1 SELECT orders.*,user.username,user.sex,user.address FROM orders,user
   WHERE orders.user_id = user.id
```

创建pojo：

```
1 package com.cauchy.mybatis.po;
2
3 import java.util.Date;
4 import java.util.List;
5
6 public class Orders {
7     private Integer id;
8     private Integer userId;
9     private String number;
10    private Date createtime;
11    private String note;
12
13    // 用户信息：
14    private User user;
15    public User getUser() {
16        return user;
17    }
18    private List<Orderdetail> orderdetails;
19
20
21    public List<Orderdetail> getOrderdetails() {
22        return orderdetails;
23    }
}
```

```
24     public void setOrderdetail(List<Orderdetail> orderdetails) {
25         this.orderdetails = orderdetails;
26     }
27     public void setUser(User user) {
28         this.user = user;
29     }
30     public Integer getId() {
31         return id;
32     }
33     public void setId(Integer id) {
34         this.id = id;
35     }
36     public Integer getUserId() {
37         return userId;
38     }
39     public void setUserId(Integer userId) {
40         this.userId = userId;
41     }
42     public String getNumber() {
43         return number;
44     }
45     public void setNumber(String number) {
46         this.number = number;
47     }
48     public Date getCreatetime() {
49         return createtime;
50     }
51     public void setCreatetime(Date createtime) {
52         this.createtime = createtime;
53     }
54     public String getNote() {
55         return note;
56     }
57     public void setNote(String note) {
58         this.note = note;
59     }
60     @Override
61     public String toString() {
62         return "Orders [id=" + id + ", userId=" + userId + ", number="
+ number + ", createtime=" + createtime
63             + ", note=" + note + "]\n";
64     }
```

```
65 }  
66
```

将SQL查询出来的结果集映射到pojo中来：

pojo中要包括所有的查询列表：

原始的orders不能映射全部字段，需要创建一个pojo继承包括查询字段较多的类：

```
1 package com.cauchy.mybatis.po;  
2  
3 // 通过此类来映射订单和用户查询的结果：  
4 public class OrdersCustom extends Orders{  
5     // 添加用户的一些信息：  
6     private String username;  
7     private String sex;  
8     private String address;  
9     public String getUsername() {  
10         return username;  
11     }  
12     public void setUsername(String username) {  
13         this.username = username;  
14     }  
15     public String getSex() {  
16         return sex;  
17     }  
18     public void setSex(String sex) {  
19         this.sex = sex;  
20     }  
21     public String getAddress() {  
22         return address;  
23     }  
24     public void setAddress(String address) {  
25         this.address = address;  
26     }  
27  
28 }  
29
```

OrderCustomMapper.xml:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace = "com.cauchy.mybatis.mapper.OrderCustomMapper">
6     <!-- 查询订单关联用户信息 -->
7     <select id = "findOrderUser" resultType =
8         "com.cauchy.mybatis.po.OrderCustom">
9         select orders.*,user.username,user.sex,user.address from
10         orders,user
11         where orders.user_id = user.id;
12     </select>
13 </mapper>

```

OrderCustomMapper.java:

```

1 package com.cauchy.mybatis.mapper;
2
3 import java.util.List;
4
5 import com.cauchy.mybatis.po.OrderCustom;
6
7 public interface OrderCustomMapper {
8     // 查询订单以及用户
9     public List<OrderCustom> findOrderUser()throws Exception;
10 }
11

```

测试方法OrderCustomMapperTest.java:

```

1 package com.cauchy.mybatis.mapper;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.util.List;
6
7 import org.apache.ibatis.io.Resources;

```

```

8  import org.apache.ibatis.session.SqlSession;
9  import org.apache.ibatis.session.SqlSessionFactory;
10 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
11 import org.junit.Before;
12 import org.junit.Test;
13
14 import com.cauchy.mybatis.po.OrderCustom;
15
16 public class OrderCustomMapperTest {
17     private SqlSessionFactory sqlSessionFactory;
18     @Before
19     public void setUp() throws IOException {
20         String resource = "sqlMapConfig.xml";
21         InputStream inputStream =
Resources.getResourceAsStream(resource);
22         sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
23     }
24     @Test
25     public void testFindOrderUsers() throws Exception {
26         SqlSession sqlSession = sqlSessionFactory.openSession();
27         // 创建代理对象
28         OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
29         // 调用方法:
30         List <OrderCustom> list = orderCustomMapper.findOrderUser();
31         System.out.println(list);
32         sqlSession.close();
33     }
34 }

```

resultMap实现:

SQL语句:

同上。

使用resultMap将查询的结果映射到Order对象中，在orders类中添加属性，将关联出来的用户信息映射到order对象中的user属性中。需要在orders类中添加user属性:

定义 OrderCustomMapper.xml:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace = "com.cauchy.mybatis.mapper.OrderCustomMapper">
6     <!-- 查询订单关联用户信息 -->
7     <select id = "findOrderUser" resultType =
8         "com.cauchy.mybatis.po.OrderCustom">
9         select orders.* ,user.username,user.sex,user.address from
10 orders,user where
11         orders.user_id = user.id
12     </select>
13     <resultMap type="com.cauchy.mybatis.po.OrderCustom"
14 id="orderUserResultMap">
15         <!-- 映射订单信息 -->
16         <!-- id : 指定查询列中订单信息中的唯一标识，如果有多个列组成唯一标
17 识，则需要多个id -->
18         <id column="id" property="id"/>
19         <result column="user_id" property="userId"/>
20         <result column="number" property="number"/>
21         <result column="createtime" property="createTime"/>
22         <result column="note" property="note"/>
23         <!-- 映射用户信息 -->
24         <!-- association 用于映射关联单个对象的信息，property要将关联信息
25 映射到order的哪一个属性 -->
26         <association property="user" javaType =
27         "com.cauchy.mybatis.po.User">
28             <!-- 关联用户的唯一标识，用户标识用户信息的列-->
29             <id column="user_id" property="id"/>
30             <result column="username" property="userName"/>
31             <result column="sex" property="sex"/>
32             <result column="address" property="address"/>
33         </association>
34     </resultMap>
35     <select id="findOrderUserResultMap"
36 resultMap="orderUserResultMap">
37         select orders.* , user.username,user.sex,user.address from
38 orders, user where orders.user_id = user.id
39     </select>
40 </mapper>
```


定义OrderCustomMapper.java:

```
1 package com.cauchy.mybatis.mapper;
2
3 import java.util.List;
4
5 import com.cauchy.mybatis.po.OrderCustom;
6 import com.cauchy.mybatis.po.Order;
7
8 public interface OrderCustomMapper {
9
10     public List<OrderCustom> findOrderUser()throws Exception;
11
12     public List<Order> findOrderUserResultMap()throws Exception;
13 }
14
```

OrderCustomMapperTest.java:

```
1 package com.cauchy.mybatis.mapper;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.util.List;
6
7 import org.apache.ibatis.io.Resources;
8 import org.apache.ibatis.session.SqlSession;
9 import org.apache.ibatis.session.SqlSessionFactory;
10 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
11 import org.junit.Before;
12 import org.junit.Test;
13
14 import com.cauchy.mybatis.po.Order;
15 import com.cauchy.mybatis.po.OrderCustom;
16
17 public class OrderCustomMapperTest {
18     private SqlSessionFactory sqlSessionFactory;
19     @Before
20     public void setUp()throws IOException{

```

```

21     String resource = "sqlMapConfig.xml";
22     InputStream inputStream =
Resources.getResourceAsStream(resource);
23     sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
24 }
25 @Test
26 public void testFindOrderUsers() throws Exception{
27     SqlSession sqlSession = sqlSessionFactory.openSession();
28     // 创建代理对象
29     OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
30     // 调用方法:
31     List <OrderCustom> list = orderCustomMapper.findOrderUser();
32     System.out.println(list);
33     sqlSession.close();
34 }
35 @Test
36 public void testFindOrderUsersResultMap()throws Exception{
37     SqlSession sqlSession = sqlSessionFactory.openSession();
38     // 创建代理对象:
39     OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
40     //调用方法
41     List<Order> list = orderCustomMapper.findOrderUserResultMap();
42     System.out.println(list);
43     sqlSession.close();
44 }
45 }

```

小结:

实现一对一查询resultType使用实现较为简单，如果pojo没有包括查询出来的列名，需要增加列名对应的属性即可查询，即如果没有查询结果的特殊要求，使用resultType

resultMap 实现比较复杂，如果对查询结果又特殊的要求，使用resultMap可以完成将关联查询映射到pojo的属性中。resultMap可以实现延迟加载。

一对多:

Items.java:

```
1 package com.cauchy.mybatis.po;
2
3 import java.util.Date;
4
5 public class Items {
6     private Integer id;
7     private String name;
8     private Float price;
9     private String detail;
10    private String pic;
11    private Date createTime;
12    public Integer getId() {
13        return id;
14    }
15    public void setId(Integer id) {
16        this.id = id;
17    }
18    public String getName() {
19        return name;
20    }
21    public void setName(String name) {
22        this.name = name;
23    }
24    public Float getPrice() {
25        return price;
26    }
27    public void setPrice(Float price) {
28        this.price = price;
29    }
30    public String getDetail() {
31        return detail;
32    }
33    public void setDetail(String detail) {
34        this.detail = detail;
35    }
36    public String getPic() {
37        return pic;
38    }
39    public void setPic(String pic) {
40        this.pic = pic;
41    }
```

```
42     public Date getCreateTime() {
43         return createTime;
44     }
45     public void setCreateTime(Date createTime) {
46         this.createTime = createTime;
47     }
48
49 }
50
```

OrderDetail.java:

```
1  package com.cauchy.mybatis.po;
2
3  public class OrderDetail {
4      private Integer id;
5      private Integer ordersId;
6      private Integer itemsId;
7      private Integer itemNum;
8      private Items items;
9      public Integer getId() {
10         return id;
11     }
12     public void setId(Integer id) {
13         this.id = id;
14     }
15     public Integer getOrdersId() {
16         return ordersId;
17     }
18     public void setOrdersId(Integer ordersId) {
19         this.ordersId = ordersId;
20     }
21     public Integer getItemsId() {
22         return itemsId;
23     }
24     public void setItemsId(Integer itemsId) {
25         this.itemsId = itemsId;
26     }
27     public Integer getItemNum() {
28         return itemNum;

```

```

29     }
30     public void setItemNum(Integer itemNum) {
31         this.itemNum = itemNum;
32     }
33     public Items getItems() {
34         return items;
35     }
36     public void setItems(Items items) {
37         this.items = items;
38     }
39 }
40

```

需求分析：

查询订单以及订单明细：

sql语句：

主查询表：订单表；

确定关联查询表：订单明细表；

在一对多查询基础上添加订单明细表关联即可；

使用resultType来将上面的查询结果集映射到pojo中，订单信息就会重复，要求是对order查询的映射不能出现重复数据，在order类中增加List<Orderdetail>属性，最终将会将订单信息映射到order中，订单所对应的订单明细会映射到order中的orderdetail中最终映射的order的记录数为不重复的。每个order中的detail属性，存储了该订单的订单明细。

在Orders.java 中添加属性

```

1  private List<Orderdetail> orderdetails;
2  public List<Orderdetail> getOrderdetails() {
3      return orderdetails;
4  }
5  public void setOrderdetail(List<Orderdetail> orderdetails) {
6      this.orderdetails = orderdetails;
7  }

```

OrderCustomMapper.xml:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace = "com.cauchy.mybatis.mapper.OrderCustomMapper">
6     <!-- 查询订单关联用户信息 -->
7     <select id = "findOrderUser" resultType =
8         "com.cauchy.mybatis.po.OrderCustom">
9         select orders.*,user.username,user.sex,user.address from
10 orders,user
11         where orders.user_id = user.id;
12     </select>
13     <resultMap type="com.cauchy.mybatis.po.OrderCustom"
14 id="orderUserResultMap">
15         <!-- 映射订单信息 -->
16         <!-- id : 指定查询列中订单信息中的唯一标识，如果有多个列组成唯一标
17 识，则需要多个id -->
18         <id column="id" property="id"/>
19         <result column="user_id" property="userId"/>
20         <result column="number" property="number"/>
21         <result column="createtime" property="createTime"/>
22         <result column="note" property="note"/>
23         <!-- 映射用户信息 -->
24         <!-- association 用于映射关联单个对象的信息，property要将关联信息
25 映射到order的哪一个属性 -->
26         <association property="user" javaType =
27         "com.cauchy.mybatis.po.User">
28             <!-- 关联用户的唯一标识，用户标识用户信息的列-->
29             <id column="user_id" property="id"/>
30             <result column="username" property="userName"/>
31             <result column="sex" property="sex"/>
32             <result column="address" property="address"/>
33         </association>
34     </resultMap>
35     <select id="findOrderUserResultMap"
36 resultMap="orderUserResultMap">
37         select orders.* , user.username,user.sex,user.address from
38 orders, user where orders.user_id = user.id
39     </select>
40     <resultMap type="com.cauchy.mybatis.po.Order"
41 id="orderAndOrderDetailResultMap" extends="orderUserResultMap">
42         <!-- 映射订单信息，使用继承标签可以控制冗余 -->

```

```

34      <!-- 订单明细信息，每个订单有多个明细信息，要使用collection来处理
ofType:指定要映射到集合pojo的属性名 -->
35      <collection property="orderDetailList" ofType =
"com.cauchy.mybatis.po.OrderDetail">
36          <!-- 关联明细的唯一标识，property: 要将对象的唯一标识对应的
OrderDetail对象的属性 -->
37          <id column="orderdetail_id" property = "id"/>
38          <result column="item_id" property="itemId"/>
39          <result column="item_num" property = "itemNum"/>
40          <result column="order_id" property = "orderId"/>
41      </collection>
42  </resultMap>
43  <select id="findOrderAndOrderDetailResultMap" resultMap =
"orderAndOrderDetailResultMap">
44      select
orders.*,user.username,user.sex,user.address,orderdetail.id
orderdetail_id,orderdetail.item_id,
45      orderdetail.item_num,orderdetail.order_id from
orders,user,orderdetail where orders.user_id = user.id and
46      orderdetail.order_id = orders.id
47  </select>
48 </mapper>

```

OrderCustomMapper.java:

```

1  package com.cauchy.mybatis.mapper;
2
3  import java.util.List;
4
5  import com.cauchy.mybatis.po.Order;
6  import com.cauchy.mybatis.po.OrderCustom;
7
8  public interface OrderCustomMapper {
9      // 查询订单以及用户
10     public List<OrderCustom> findOrderUser()throws Exception;
11     // resultMap实现上一方法:
12     public List<Order> findOrderUserResultMap()throws Exception;
13     // 订单信息:
14     public List<Order> findOrderAndOrderDetailResultMap() throws
Exception;

```

```
15 }  
16
```

OrderCustomMapperTest.java:

```
1  package com.cauchy.mybatis.mapper;  
2  
3  import java.io.IOException;  
4  import java.io.InputStream;  
5  import java.util.List;  
6  
7  import org.apache.ibatis.io.Resources;  
8  import org.apache.ibatis.session.SqlSession;  
9  import org.apache.ibatis.session.SqlSessionFactory;  
10 import org.apache.ibatis.session.SqlSessionFactoryBuilder;  
11 import org.junit.Before;  
12 import org.junit.Test;  
13  
14 import com.cauchy.mybatis.po.Order;  
15 import com.cauchy.mybatis.po.OrderCustom;  
16  
17 public class OrderCustomMapperTest {  
18     private SqlSessionFactory sqlSessionFactory;  
19     @Before  
20     public void setUp() throws IOException {  
21         String resource = "sqlMapConfig.xml";  
22         InputStream inputStream =  
Resources.getResourceAsStream(resource);  
23         sqlSessionFactory = new  
SqlSessionFactoryBuilder().build(inputStream);  
24     }  
25     @Test  
26     public void testFindOrderUsers() throws Exception {  
27         SqlSession sqlSession = sqlSessionFactory.openSession();  
28         // 创建代理对象  
29         OrderCustomMapper orderCustomMapper =  
sqlSession.getMapper(OrderCustomMapper.class);  
30         // 调用方法:  
31         List <OrderCustom> list = orderCustomMapper.findOrderUser();  
32         System.out.println(list);
```



```

33         sqlSession.close();
34     }
35     @Test
36     public void testFindOrderUsersResultMap()throws Exception{
37         SqlSession sqlSession = sqlSessionFactory.openSession();
38         // 创建代理对象:
39         OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
40         //调用方法
41         List<Order> list = orderCustomMapper.findOrderUserResultMap();
42         System.out.println(list);
43         sqlSession.close();
44     }
45     @Test
46     public void testFindOrderandOrderDetailResultMap()throws
Exception{
47         SqlSession sqlSession = sqlSessionFactory.openSession();
48         // 创建代理对象:
49         OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
50         // 调用方法
51         List<Order> list =
orderCustomMapper.findOrderAndOrderDetailResultMap();
52         System.out.println(list);
53         sqlSession.close();
54     }
55 }

```

小结:

mybatis使用collection对关联查询的多条记录映射到一个list集合中，使用resultType实现将订单order中的orderdetail中需要自己处理，使用双重循环遍历，来去掉重复记录，将订单明细存储在List中。

多对多:

需求: 查询用户及用户购买的商品信息，分析:

查询主表: 用户表

关联表: 用户和商品没有直接关联，所以关联:

orders 、 orderdetail、 items:

```

1 select orders.*,user.username,user.sex,user.address,orderdetail.id
   orderdetail_id,orderdetail.items_id,
2 orderdetail.item_num,orderdetail.orders_id,items.name
   items_name,items.detail items_detail ,items.price
3 items_price from orders,user,orderdetail,items WHERE orders.user_id =
   user.id and orderdetail.orders_id
4 = orders.id and orderdetail.items_id = items.id

```

映射思路：

将用户信息映射到user在user中添加订单列表属性List<Orders> orderlist

在Orders中添加订单明细属性List<Orderdetail>orderdetails属性

在Orderdetails中添加items属性，以及get和set方法。

mapper.xml:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace = "com.cauchy.mybatis.mapper.OrderCustomMapper">
6     <!-- 查询订单关联用户信息 -->
7     <select id = "findOrderUser" resultType =
8         "com.cauchy.mybatis.po.OrderCustom">
9         select orders.*,user.username,user.sex,user.address from
10 orders,user
11         where orders.user_id = user.id;
12     </select>
13     <resultMap type="com.cauchy.mybatis.po.OrderCustom"
14 id="orderUserResultMap">
15         <!-- 映射订单信息 -->
16         <!-- id : 指定查询列中订单信息中的唯一标识，如果有多个列组成唯一标
17 识，则需要多个id -->
18         <id column="id" property="id"/>
19         <result column="user_id" property="userId"/>
20         <result column="number" property="number"/>
21         <result column="createtime" property="createTime"/>
22         <result column="note" property="note"/>
23     </resultMap>
24     <!-- 映射用户信息 -->

```

```

20      <!-- association 用于映射关联单个对象的信息，property要将关联信息
映射到order的哪一个属性 -->
21      <association property="user" javaType =
"com.cauchy.mybatis.po.User">
22          <!-- 关联用户的唯一标识，用户标识用户信息的列-->
23          <id column="user_id" property="id"/>
24          <result column="username" property="userName"/>
25          <result column="sex" property="sex"/>
26          <result column="address" property="address"/>
27      </association>
28  </resultMap>
29  <select id="findOrderUserResultMap"
resultMap="orderUserResultMap">
30      select orders.* , user.username,user.sex,user.address from
orders, user where orders.user_id = user.id
31  </select>
32  <resultMap type="com.cauchy.mybatis.po.Order"
id="orderAndOrderDetailResultMap" extends="orderUserResultMap">
33      <!-- 映射订单信息，使用继承标签可以控制冗余 -->
34      <!-- 订单明细信息，每个订单有多个明细信息，要使用collection来处理
ofType:指定要映射到集合pojo的属性名 -->
35      <collection property="orderDetailList" ofType =
"com.cauchy.mybatis.po.OrderDetail">
36          <!-- 关联明细的唯一标识，property: 要将对象的唯一标识对应的
OrderDetail对象的属性 -->
37          <id column="orderdetail_id" property = "id"/>
38          <result column="item_id" property="itemId"/>
39          <result column="item_num" property = "itemNum"/>
40          <result column="order_id" property = "orderId"/>
41      </collection>
42  </resultMap>
43  <select id="findOrderAndOrderDetailResultMap" resultMap =
"orderAndOrderDetailResultMap">
44      select
orders.*,user.username,user.sex,user.address,orderdetail.id
orderdetail_id,orderdetail.item_id,
45      orderdetail.item_num,orderdetail.order_id from
orders,user,orderdetail where orders.user_id = user.id and
46      orderdetail.order_id = orders.id
47  </select>
48  <resultMap type="com.cauchy.mybatis.po.User"
id="userAndItemsResultMap">

```

```

49      <!-- 用户信息 -->
50      <id column = "user_id" property = "id"/>
51      <result column = "username" property="userName"/>
52      <result column = "sex" property = "sex"/>
53      <result column = "address" property = "address"/>
54      <!-- 订单信息，每个用户对应多个订单 -->
55      <collection property="orderList" ofType =
"com.cauchy.mybatis.po.Order">
56          <id column="id" property="id"/>
57          <result column="number" property="number"/>
58          <result column="createtime" property="createTime"/>
59          <result column = "note" property = "note"/>
60          <!-- 订单明细 -->
61          <collection property="orderDetailList" ofType =
"com.cauchy.mybatis.po.OrderDetail">
62              <id column="orderdetail_id" property="id"/>
63              <result column="item_id" property = "itemId"/>
64              <result column="item_num" property="itemNum"/>
65              <result column="order_id" property="orderId"/>
66              <!-- 每个商品对应一个 明细 -->
67              <association property="item" javaType =
"com.cauchy.mybatis.po.Item">
68                  <id column = "item_id" property = "id"/>
69                  <result column="item_name" property="name"/>
70                  <result column="item_detail" property="detail"/>
71                  <result column="item_price" property = "price"/>
72              </association>
73          </collection>
74      </collection>
75  </resultMap>
76  <!-- 查询用户以及购买的商品信息 -->
77  <select id="findUserAndItemsResultMap" resultMap =
"userAndItemsResultMap">
78      select
orders.*,user.username,user.sex,user.address,orderdetail.id
orderdetail_id,orderdetail.item_id,
79      orderdetail.item_num,orderdetail.order_id,item.name
item_name,item.detail item_detail,item.price
80      item_price from orders,user,orderdetail,item where
orders.user_id = user.id and orderdetail.order_id
81      = orders.id and orderdetail.item_id = item.id
82  </select>

```

```
83 </mapper>
```

OrderCustomMapper.java:

```
1 package com.cauchy.mybatis.mapper;
2
3 import java.util.List;
4
5 import com.cauchy.mybatis.po.Order;
6 import com.cauchy.mybatis.po.OrderCustom;
7 import com.cauchy.mybatis.po.User;
8
9 public interface OrderCustomMapper {
10     // 查询订单以及用户
11     public List<OrderCustom> findOrderUser()throws Exception;
12     // resultMap实现上一方法:
13     public List<Order> findOrderUserResultMap()throws Exception;
14     // 订单信息:
15     public List<Order> findOrderAndOrderDetailResultMap() throws
Exception;
16     // 查询用户购买的商品信息:
17     public List<User> findUserAndItemsResultMap()throws Exception;
18 }
19
```

测试:

```
1 package com.cauchy.mybatis.mapper;
2
3 import java.io.IOException;
4 import java.io.InputStream;
5 import java.util.List;
6
7 import org.apache.ibatis.io.Resources;
8 import org.apache.ibatis.session.SqlSession;
9 import org.apache.ibatis.session.SqlSessionFactory;
10 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
11 import org.junit.Before;
```

```
12 import org.junit.Test;
13
14 import com.cauchy.mybatis.po.Order;
15 import com.cauchy.mybatis.po.OrderCustom;
16 import com.cauchy.mybatis.po.User;
17
18 public class OrderCustomMapperTest {
19     private SqlSessionFactory sqlSessionFactory;
20     @Before
21     public void setUp()throws IOException{
22         String resource = "sqlMapConfig.xml";
23         InputStream inputStream =
24             Resources.getResourceAsStream(resource);
25         sqlSessionFactory = new
26             SqlSessionFactoryBuilder().build(inputStream);
27     }
28     @Test
29     public void testFindOrderUsers() throws Exception{
30         SqlSession sqlSession = sqlSessionFactory.openSession();
31         // 创建代理对象
32         OrderCustomMapper orderCustomMapper =
33             sqlSession.getMapper(OrderCustomMapper.class);
34         // 调用方法:
35         List <OrderCustom> list = orderCustomMapper.findOrderUser();
36         System.out.println(list);
37         sqlSession.close();
38     }
39     @Test
40     public void testFindOrderUsersResultMap()throws Exception{
41         SqlSession sqlSession = sqlSessionFactory.openSession();
42         // 创建代理对象:
43         OrderCustomMapper orderCustomMapper =
44             sqlSession.getMapper(OrderCustomMapper.class);
45         //调用方法
46         List<Order> list = orderCustomMapper.findOrderUserResultMap();
47         System.out.println(list);
48         sqlSession.close();
49     }
50     @Test
51     public void testFindOrderandOrderDetailResultMap()throws
52         Exception{
53         SqlSession sqlSession = sqlSessionFactory.openSession();
```

```

49      // 创建代理对象:
50      OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
51      // 调用方法
52      List<Order> list =
orderCustomMapper.findOrderAndOrderDetailResultMap();
53      System.out.println(list);
54      sqlSession.close();
55  }
56  @Test
57  public void testFindUserAndItemsResultMap()throws Exception{
58      SqlSession sqlSession = sqlSessionFactory.openSession();
59      // 创建代理对象:
60      OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
61      // 调用方法
62      List<User>list =
orderCustomMapper.findUserAndItemsResultMap();
63      System.out.println(list);
64      sqlSession.close();
65  }
66  }

```

多对多查询总结:

将查询用户购买商品的信息明细清单，包括用户名，用户地址，购买商品名称，购买商品时间，购买商品数量

针对上面的需求，只需使用resultType将查询到的记录，映射到一个扩展的pojo类型中，很简单的实现我们的需求

延迟加载:

什么是延迟加载:

resultMap: 可以实现高级映射（使用association、collection来实现一对一、一对多的映射、二者具备延迟加载的功能

如果查询订单、关联查询用户信息，如果先查询订单信息，就可以满足要求，当我们需要查询用户信息时，再查询用户信息，对用户信息的按需查询就称为延迟加载。延迟加载：先从

单表查询，需要时再从关联表去查询，就可以大大提高数据库的性能，因为查询单表要比关联查询多张表效率要快。

使用association实现延迟加载：

查询订单，并关联查询用户：

OrderCustomMapper.xml中需要定义两个mapper方法对应的statement

1、只查询订单信息：

```
1 select * from orders;
```

在查询订单信息的statement中使用association去延迟加载下面的statement

2、查询用户信息：

通过查询到的订单信息，关联查询用户信息：

```
1 select orders.* user.username,user.sex,user.address from orders,user
   where orders.user_id = user.id;
```

OrderCustomMapper.xml:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <mapper namespace = "com.cauchy.mybatis.mapper.OrderCustomMapper">
6     <!-- 查询订单关联用户信息 -->
7     <select id = "findOrderUser" resultType =
8         "com.cauchy.mybatis.po.OrderCustom">
9         select orders.*,user.username,user.sex,user.address from
10         orders,user
11         where orders.user_id = user.id;
12     </select>
13     <resultMap type="com.cauchy.mybatis.po.OrderCustom"
14         id="orderUserResultMap">
15         <!-- 映射订单信息 -->
16         <!-- id : 指定查询列中订单信息中的唯一标识，如果有多个列组成唯一标
```


识，则需要多个id -->

```
14     <id column="id" property="id"/>
15     <result column="user_id" property="userId"/>
16     <result column="number" property="number"/>
17     <result column="createtime" property="createTime"/>
18     <result column="note" property="note"/>
19     <!-- 映射用户信息 -->
20     <!-- association 用于映射关联单个对象的信息，property要将关联信息
映射到order的哪一个属性 -->
21     <association property="user" javaType =
"com.cauchy.mybatis.po.User">
22         <!-- 关联用户的唯一标识，用户标识用户信息的列-->
23         <id column="user_id" property="id"/>
24         <result column="username" property="userName"/>
25         <result column="sex" property="sex"/>
26         <result column="address" property="address"/>
27     </association>
28 </resultMap>
29 <select id="findOrderUserResultMap"
resultMap="orderUserResultMap">
30     select orders.* , user.username,user.sex,user.address from
orders, user where orders.user_id = user.id
31 </select>
32 <resultMap type="com.cauchy.mybatis.po.Order"
id="orderAndOrderDetailResultMap" extends="orderUserResultMap">
33     <!-- 映射订单信息，使用继承标签可以控制冗余 -->
34     <!-- 订单明细信息，每个订单有多个明细信息，要使用collection来处理
ofType:指定要映射到集合pojo的属性名 -->
35     <collection property="orderDetailList" ofType =
"com.cauchy.mybatis.po.OrderDetail">
36         <!-- 关联明细的唯一标识，property: 要将对象的唯一标识对应的
OrderDetail对象的属性 -->
37         <id column="orderdetail_id" property = "id"/>
38         <result column="item_id" property="itemId"/>
39         <result column="item_num" property = "itemNum"/>
40         <result column="order_id" property = "orderId"/>
41     </collection>
42 </resultMap>
43 <select id="findOrderAndOrderDetailResultMap" resultMap =
"orderAndOrderDetailResultMap">
44     select
orders.*,user.username,user.sex,user.address,orderdetail.id
```

```

orderdetail_id,orderdetail.item_id,
45     orderdetail.item_num,orderdetail.order_id from
orders,user,orderdetail where orders.user_id = user.id and
46     orderdetail.order_id = orders.id
47 </select>
48 <resultMap type="com.cauchy.mybatis.po.User"
id="userAndItemsResultMap">
49     <!-- 用户信息 -->
50     <id column = "user_id" property = "id"/>
51     <result column = "username" property="userName"/>
52     <result column = "sex" property = "sex"/>
53     <result column = "address" property = "address"/>
54     <!-- 订单信息，每个用户对多个订单 -->
55     <collection property="orderList" ofType =
"com.cauchy.mybatis.po.Order">
56         <id column="id" property="id"/>
57         <result column="number" property="number"/>
58         <result column="createtime" property="createTime"/>
59         <result column = "note" property = "note"/>
60         <!-- 订单明细 -->
61         <collection property="orderDetailList" ofType =
"com.cauchy.mybatis.po.OrderDetail">
62             <id column="orderdetail_id" property="id"/>
63             <result column="item_id" property = "itemId"/>
64             <result column="item_num" property="itemNum"/>
65             <result column="order_id" property="orderId"/>
66             <!-- 每个商品对应一个 明细 -->
67             <association property="item" javaType =
"com.cauchy.mybatis.po.Item">
68                 <id column = "item_id" property = "id"/>
69                 <result column="item_name" property="name"/>
70                 <result column="item_detail" property="detail"/>
71                 <result column="item_price" property = "price"/>
72             </association>
73         </collection>
74     </collection>
75 </resultMap>
76 <!-- 查询用户以及购买的商品信息 -->
77 <select id="findUserAndItemsResultMap" resultMap =
"userAndItemsResultMap">
78     select
orders.*,user.username,user.sex,user.address,orderdetail.id

```

```

orderdetail_id,orderdetail.item_id,
79      orderdetail.item_num,orderdetail.order_id,item.name
item_name,item.detail item_detail,item.price
80      item_price from orders,user,orderdetail,item where
orders.user_id = user.id and orderdetail.order_id
81      = orders.id and orderdetail.item_id = item.id
82      </select>
83      <!-- 延迟加载的resultMap -->
84      <resultMap type="com.cauchy.mybatis.po.Order"
id="orderUserLazyLoading">
85          <!-- 要实现对用户信息进行延迟加载 -->
86          <!-- 对订单信息进行配置 -->
87          <id column="id" property = "id"/>
88          <result column="user_id" property="userId"/>
89          <result column="number" property="number"/>
90          <result column="createtime" property="createTime"/>
91          <result column="note" property="note"/>
92          <!-- 实现用户信息的延迟加载 select指定需要延迟
93          加载的statement（根据user_id 去查询用户信息的statement
column 是订单信息中关联用户信息的列） -->
94          <association property="user"
javaType="com.cauchy.mybatis.po.User"
95          select="com.cauchy.mybatis.mapper.UserMapper.findUserById"
column="user_id">
96              </association>
97          </resultMap>
98          <!-- 查询订单关联查询用户，用户信息需要延迟加载-->
99          <select id="findOrderUserLazyLoading" resultMap =
"orderUserLazyLoading">
100              select * from orders
101          </select>
102      </mapper>

```

OrderCustomMapper.java

```

1 package com.cauchy.mybatis.mapper;
2
3 import java.util.List;
4
5 import com.cauchy.mybatis.po.Order;

```

```

6  import com.cauchy.mybatis.po.OrderCustom;
7  import com.cauchy.mybatis.po.User;
8
9  public interface OrderCustomMapper {
10     // 查询订单以及用户
11     public List<OrderCustom> findOrderUser()throws Exception;
12     // resultMap实现上一方法:
13     public List<Order> findOrderUserResultMap()throws Exception;
14     // 订单信息:
15     public List<Order> findOrderAndOrderDetailResultMap() throws
Exception;
16     // 查询用户购买的商品信息:
17     public List<User> findUserAndItemsResultMap()throws Exception;
18     // 延迟加载用户信息
19     public List<Order> findOrderUserLazyLoading()throws Exception;
20 }
21

```

测试思路:

执行上面的mapper方法, findOrdersUserLazyLoading内部去调用查询order信息的statement, 获取了一个List, 在程序中的第二步, 在程序中去遍历List过程中, 当我们调用order中的getUser时 就开始延迟加载, 去调用UserMapper.xml中的findUserById 方法, 获取用户信息, mybatis默认是没有开启延迟加载的, 需要在SQLMapperConfig中去配置:

```

1  <settings>
2      <!-- 打开延迟加载的开关 -->
3      <setting name="lazyLoadingEnabled" value="true"/>
4      <!-- 将积极加载改为消极加载 -->
5      <setting name="aggressiveLazyLoading" value="false"/>
6  </settings>

```

OrderCustomMapperTest.java:

```

1  package com.cauchy.mybatis.mapper;
2
3  import java.io.IOException;
4  import java.io.InputStream;

```

```
5  import java.util.List;
6
7  import org.apache.ibatis.io.Resources;
8  import org.apache.ibatis.session.SqlSession;
9  import org.apache.ibatis.session.SqlSessionFactory;
10 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
11 import org.junit.Before;
12 import org.junit.Test;
13
14 import com.cauchy.mybatis.po.Order;
15 import com.cauchy.mybatis.po.OrderCustom;
16 import com.cauchy.mybatis.po.User;
17
18 public class OrderCustomMapperTest {
19     private SqlSessionFactory sqlSessionFactory;
20     @Before
21     public void setUp()throws IOException{
22         String resource = "sqlMapConfig.xml";
23         InputStream inputStream =
24             Resources.getResourceAsStream(resource);
25         sqlSessionFactory = new
26             SqlSessionFactoryBuilder().build(inputStream);
27     }
28     @Test
29     public void testFindOrderUsers() throws Exception{
30         SqlSession sqlSession = sqlSessionFactory.openSession();
31         // 创建代理对象
32         OrderCustomMapper orderCustomMapper =
33             sqlSession.getMapper(OrderCustomMapper.class);
34         // 调用方法:
35         List <OrderCustom> list = orderCustomMapper.findOrderUser();
36         System.out.println(list);
37         sqlSession.close();
38     }
39     @Test
40     public void testFindOrderUsersResultMap()throws Exception{
41         SqlSession sqlSession = sqlSessionFactory.openSession();
42         // 创建代理对象:
43         OrderCustomMapper orderCustomMapper =
44             sqlSession.getMapper(OrderCustomMapper.class);
45         //调用方法
46         List<Order> list = orderCustomMapper.findOrderUserResultMap();
```

```
43         System.out.println(list);
44         sqlSession.close();
45     }
46     @Test
47     public void testFindOrderandOrderDetailResultMap()throws
Exception{
48         SqlSession sqlSession = sqlSessionFactory.openSession();
49         // 创建代理对象:
50         OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
51         // 调用方法
52         List<Order> list =
orderCustomMapper.findOrderAndOrderDetailResultMap();
53         System.out.println(list);
54         sqlSession.close();
55     }
56     @Test
57     public void testFindUserAndItemsResultMap()throws Exception{
58         SqlSession sqlSession = sqlSessionFactory.openSession();
59         // 创建代理对象:
60         OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
61         // 调用方法
62         List<User>list =
orderCustomMapper.findUserAndItemsResultMap();
63         System.out.println(list);
64         sqlSession.close();
65     }
66     @Test
67     public void testFindOrderUserLazyLoading() throws Exception {
68         SqlSession sqlSession = sqlSessionFactory.openSession();
69         // 创建代理对象:
70         OrderCustomMapper orderCustomMapper =
sqlSession.getMapper(OrderCustomMapper.class);
71         // 调用方法:
72         List<Order> list =
orderCustomMapper.findOrderUserLazyLoading();
73         // 遍历订单列表:
74         for (Order order : list) {
75             // 执行getUser 来执行延迟加载:
76             User user = order.getUser();
77             System.out.println(user);
```

```
78     }
79     sqlSession.close();
80 }
81 }
```

通过debug模式，可以看到执行，可以看到在执行循环过程中，将用户信息查询出来。如果不使用mybatis提供的association和collection中的延迟加载，实现方法为先查询订单列表，再根据用户id查询用户信息。先去查询第一个mapper方法，获取订单列表，在程序中按需去调取第二个mapper方法来查询用户信息。即，使用延迟加载的方法，先去查询简单的SQL，再去按照需求加载关联查询的信息。

查询缓存：

mybatis提供的用户减轻数据库的压力，提供了查询缓存，以提高数据库的性能。在操作数据库时，需要构造sqlSession对象，在对象中有一个数据结构（HashMap）用于存储缓存数据，不同的sqlSession之间的缓存区域相互之间不影响，二级缓存是mapper级别的缓存，多个sqlSession对象去操作同一个mapper的SQL语句，多个sqlSession对象去操作数据库对象得到的数据会存在域二级缓存区域。二级缓存区域是跨sqlSession,就是说多个sqlSession可以共用二级缓存。在查询过程中，如果缓存中有数据，就不用从数据库中提取数据，可以提高系统的性能。

一级缓存的工作原理：

第一次查询id为1的用户时，sqlSession写入一级缓存的内存区域，第二次查询可以直接从缓存区域来读取数据。如果第一次写入之后，有增删改以及提交的操作行为，则将一级缓存清空。

一级缓存是默认开启的，不需要再配置文件中配置。在正式开发中，是将mybatis和spring进行整合开发，事务控制在service中，一个service方法包括很多mapper方法。

service：

在开始执行时，开启事务，创建sqlSession对象，第一次调用mapper的方法findUserById（1），第二次调用mapper方法findUserById（2），方法结束，SqlSession关闭。如果是执行两次service调用来查询用户信息，不走一级缓存，因为session方法结束，SqlSession就关闭了，一级缓存就清空了。

二级缓存：

SqlSession1去查询id为1的用户信息，查询到用户信息将会将查询到的数据存储到二级缓

存中，SqlSession2去查询id为1的用户信息，去缓存中查找是否存在，如果存在，直接从缓存中取出数据，二级缓存与一级缓存相比较二级缓存范围更大，是多个SqlSession共享一块UserMapper的内存区域，UserMapper有一个二级缓存区域，其他的mapper也有一块二级缓存区域。所谓的内存区域是按照namespace划分。两个mapper的namespace如果相同，这两个mapper执行SQL查询到的数据将存储到一个相同的二级缓存区域中。如果SqlSession3区执行相同mapper下的commit操作，则二级缓存将会清空。mybatis的二级缓存是mapper级别，除了SQLMapConfig.xml中设置二级缓存的总开关以外，还要再具体的mapper.xml中开启二级缓存。

在SQLMapConfig.xml中：

```
1 <!-- 开启二级缓存 -->
2 <setting name="cacheEnabled" value="true"/>
```

在UserMapper.xml中开启二级缓存，UserMapper.xml下的SQL执行完成会存储到它的缓存区域中去（HashMap）。

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mapper
3 PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <!--使用mapper代理方法，namespace有特殊作用，namespace为mapper接口地址 -->
6 <mapper namespace="com.cauchy.mybatis.mapper.UserMapper">
7     <!-- 在映射文件中配置SQL语句，通过select执行数据库查询，id：标识映射文件中的SQL称为statement的id -->
8     <!-- prarmeterType 指定输入参数的类型，#{ }表示占位符，#{id} 表示接受输入
9     的参数如果输入的参数为简单类型，#{ }中的参数名可以任意
10     resultType为输出类型-->
11     <select id = "findUserById" parameterType = "int" resultType =
12     "user">
13         select * from user where id = #{id}
14     </select>
15     <!-- 根据名称模糊查询可能返回多个结果 -->
16     <!-- ${ }表示拼接字符串，表示将接受的内容不加任何修饰拼接到SQL中，使用
17     $可能会导致SQL注入，${ }中接受的参数内容如果传入的是简单类型
18     {}中只能使用value -->
19     <select id = "findUserByName" parameterType="java.lang.String"
```



```

resultType = "com.cauchy.mybatis.po.User">
17     select * from user where username like '%${value}%'
18     </select>
19     <!-- 定义一个SQL片段id 为唯一标识,根据经验,SQL片段基于单表定义,在我们的SQL片段中,不要包括where -->
20     <sql id="query_user_where">
21         <if test="userCustom != null">
22             <if test="userCustom.sex != null and userCustom.sex !=
23             ''">
24                 user.sex = #{userCustom.sex}
25             </if>
26             <if test="userCustom.userName != null and
27             userCustom.userName != ''">
28                 user.username like '%${userCustom.userName}%'
29             </if>
30             <if test="ids!=null">
31                 <!-- 使用foreach来遍历传入的ids,collection是用来指定
32                 集合属性 item用来遍历生成对象中 , open开始遍历时要拼接的串,
33                 close使用下边的SQL进行拼接: and (id = 1 or id =
34                 10 or id = 16) separator为两个对象需要拼接的串-->
35                 <foreach collection="ids" item="user_id"
36                 open="and(" close = ")" separator="or">
37                     id = #{user_id}
38                 </foreach>
39             </if>
40         </if>
41     </sql>
42     <!-- 添加用户 -->
43     <!-- parameterType 指定的参数类型为pojo, 包括用户信息, #{ }中指定pojo的
44     属性名, 接受属性值 ,
45     mybatis通过ognl来获取对象的属性值-->
46     <insert id="insertUser" parameterType =
47     "com.cauchy.mybatis.po.User">
48         <!-- 将insert 语句执行后的主键值返回 ,只适用于自增主键
49         keyProperty: 将查询到的主键值设置到parameterType 指定的对象的id属
50         性, order指定insert之前还是之后拿到值-->
51         <selectKey keyProperty="id" order="AFTER"
52         resultType="java.lang.Integer">
53             select last_insert_id()
54         </selectKey>
55         insert into user (username,birthday,sex,address)value(#{
56         userName},#{birthday},#{sex},#{address})

```

```

47     </insert>
48     <delete id="deleteUser" parameterType = "java.lang.Integer">
49         delete from user where id = #{id}
50     </delete>
51     <!-- parameter 指定parameterType 指定user对象包含id以及要更新的内容,
id必须存在-->
52     <update id="updateUser" parameterType =
"com.cauchy.mybatis.po.User">
53         update user set username = #{userName},birthday = #{birthday},
54         sex = #{sex},address = #{address} where id = #{id}
55     </update>
56     <!-- 用户信息综合查询 -->
57     <!-- #{userCustom.sex}:取出pojo包装用户的性别 -->
58     <!-- #{userCustom.userName}:取出pojo包装用户的名称 -->
59     <select id="findUserList"
parameterType="com.cauchy.mybatis.po.UserQueryVo"
60         resultType="com.cauchy.mybatis.po.UserCustom">
61         select * from user
62         <where>
63             <!-- 引用SQL片段 -->
64             <include refid="query_user_where"></include>
65         </where>
66     </select>
67     <!-- 用户信息的综合查询总数: -->
68     <select id="findUserCount" parameterType =
"com.cauchy.mybatis.po.UserQueryVo" resultType = "int">
69         select count(*) from user where user.sex = #{userCustom.sex}
70         and user.username like '%${userCustom.userName}%'
71     </select>
72     <resultMap type="com.cauchy.mybatis.po.User" id="userResultMap">
73         <!-- 标识查询结果集的唯一标识 -->
74         <id column="id_" property="id"/>
75         <!-- result: 普通列的标识 -->
76         <result column="username_" property="userName"/>
77     </resultMap>
78     <!-- 使用resultMap进行输出的映射 -->
79     <select id="findUserByResultMap" parameterType="int"
80         resultMap = "userResultMap">
81         select id id_,username username_ from user where id = #{value}
82     </select>
83
84 </mapper>

```

对调用pojo类来实现序列化接口:

```
1 package com.cauchy.mybatis.pojo;
2
3 import java.io.Serializable;
4 import java.util.Date;
5 import java.util.List;
6
7 public class User implements Serializable{
8     private int id;
9     private String userName;
10    private String sex;
11    private Date birthday;
12    private String address;
13    private List<Order> orderList;
14
15    public int getId() {
16        return id;
17    }
18    public void setId(int id) {
19        this.id = id;
20    }
21    public String getUserName() {
22        return userName;
23    }
24    public void setUserName(String userName) {
25        this.userName = userName;
26    }
27    public String getSex() {
28        return sex;
29    }
30    public void setSex(String sex) {
31        this.sex = sex;
32    }
33    public Date getBirthday() {
34        return birthday;
35    }
36    public void setBirthday(Date birthday) {
37        this.birthday = birthday;
38    }
```

```

39     public String getAddress() {
40         return address;
41     }
42     public void setAddress(String address) {
43         this.address = address;
44     }
45
46     public List<Order> getOrderList() {
47         return orderList;
48     }
49     public void setOrderList(List<Order> orderList) {
50         this.orderList = orderList;
51     }
52     @Override
53     public String toString() {
54         return "User [id=" + id + ", userName=" + userName + ", sex="
+ sex + ", birthday=" + birthday + ", address="
55             + address + "]";
56     }
57
58 }
59

```

实现接口是为了将缓存数据取出，执行反序列化操作，因为二级缓存的存储介质多种多样，不一定在内存中。

UserMapperTest.java:

```

1  package com.cauchy.mybatis.mapper;
2
3  import static org.junit.Assert.*;
4
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.util.ArrayList;
8  import java.util.List;
9
10 import org.apache.ibatis.io.Resources;
11 import org.apache.ibatis.session.SqlSession;
12 import org.apache.ibatis.session.SqlSessionFactory;

```

```
13 import org.apache.ibatis.session.SqlSessionFactoryBuilder;
14 import org.junit.Before;
15 import org.junit.Test;
16
17 import com.cauchy.mybatis.po.User;
18 import com.cauchy.mybatis.po.UserCustom;
19 import com.cauchy.mybatis.po.UserQueryVo;
20
21 public class UserMapperTest {
22     private SqlSessionFactory sqlSessionFactory;
23     @Before
24     public void setUp() throws IOException {
25         String resource = "SqlMapConfig.xml";
26         InputStream inputStream =
Resources.getResourceAsStream(resource);
27         sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
28     }
29     @Test
30     public void testFindUserById() throws Exception {
31
32         // 创建UserMapper对象
33         SqlSession sqlSession = sqlSessionFactory.openSession();
34         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
35         // 调用UserMapper的方法:
36         User user = userMapper.findUserById(1);
37         System.out.println(user);
38     }
39     @Test
40     public void testFindUserByName() throws Exception {
41
42         // 创建UserMapper对象
43         SqlSession sqlSession = sqlSessionFactory.openSession();
44         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
45         // 调用UserMapper的方法:
46         List<User> list = userMapper.findUserByName("朱");
47         System.out.println(list);
48     }
49     @Test
50     public void testFindUserList()throws Exception{
```

```
51         SqlSession sqlSession = sqlSessionFactory.openSession();
52         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
53         // 创建包装对象:
54         UserQueryVo userQueryVo = new UserQueryVo();
55         UserCustom userCustom = new UserCustom();
56         userCustom.setSex("M");
57         userQueryVo.setUserCustom(userCustom);
58         // 调用方法:
59         List<UserCustom> list = userMapper.findUserList(userQueryVo);
60         System.out.println(list);
61     }
62     @Test
63     public void testFindUserCount() throws Exception{
64         SqlSession sqlSession = sqlSessionFactory.openSession();
65         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
66         // 创建包装对象:
67         UserQueryVo userQueryVo = new UserQueryVo();
68         UserCustom userCustom = new UserCustom();
69         userCustom.setSex("M");
70         userCustom.setUserName("朱");
71         userQueryVo.setUserCustom(userCustom);
72         // 调用方法:
73         int count = userMapper.findUserCount(userQueryVo);
74         System.out.println("count is :"+ count);
75     }
76     @Test
77     public void testFindUserByIdResultMap() throws Exception{
78         SqlSession sqlSession = sqlSessionFactory.openSession();
79         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
80         // 调用方法:
81         User user = userMapper.findUserByIdResultMap(3);
82         System.out.println(user);
83     }
84     @Test
85     public void testFindUserList2()throws Exception{
86         SqlSession sqlSession = sqlSessionFactory.openSession();
87         UserMapper userMapper =
sqlSession.getMapper(UserMapper.class);
88         // 创建包装对象:
```

```

89     UserQueryVo userQueryVo = new UserQueryVo();
90     UserCustom userCustom = new UserCustom();
91     userCustom.setSex("M");
92     // 要传入多个id
93     List<Integer> ids = new ArrayList<Integer>();
94     ids.add(1);
95     ids.add(6);
96     ids.add(10);
97     // 将IDS传入
98     userQueryVo.setIds(ids);
99     userQueryVo.setUserCustom(userCustom);
100    // 调用方法:
101    List<UserCustom> list = userMapper.findUserList(userQueryVo);
102    System.out.println(list);
103 }
104 @Test
105 public void testCache2() throws Exception{
106     SqlSession sqlSession1 = sqlSessionFactory.openSession();
107     SqlSession sqlSession2 = sqlSessionFactory.openSession();
108     SqlSession sqlSession3 = sqlSessionFactory.openSession();
109     UserMapper userMapper1 =
110     sqlSession1.getMapper(UserMapper.class);
111     UserMapper userMapper2 =
112     sqlSession2.getMapper(UserMapper.class);
113     UserMapper userMapper3 =
114     sqlSession3.getMapper(UserMapper.class);
115     // 第一次发起查询请求, 查询id为1的用户
116     User user1 = userMapper1.findUserById(2);
117     System.out.println(user1);
118     // 关闭操作, 才将数据写到二级缓存中
119     sqlSession1.close();
120     // 第二次发起请求, 查询id为1用户
121     User user2 = userMapper2.findUserById(2);
122     System.out.println(user2);
123     sqlSession2.close();
124 }
125 }

```

userCache配置:

在statement中设置userCache=false可以禁用二级缓存, 即每次查询都会发出SQL去查询,

默认情况是true，即该SQL使用二级缓存。

例如：

```
1 <select id = "findUserById" resultMap = "orderUserMap" userCache=
  "false">
```

对禁止使用缓存的statement每次查询都会从数据库中来提取数据。针对每次查询都需要最新数据的，要设置成为useCache="false"

刷新缓存（清空缓存）：

在mapper中的同一个namespace中，如果有insert，update，delete操作数据后，需要刷新缓存，如果不执行刷新缓存，就会出现脏读的情况。设置statement配置中的flushCache=true属性，默认情况下为true记为刷新缓存，如果改为false则不会刷新缓存，使用缓存时，如果手动修改数据库表中的查询数据会出现脏读的情况出现。如下：

```
1 <insert id = "insertUser" parameterType = "User" flushCache = "true">
```

总结：一般执行完commit操作都需要刷新缓存，flushCache= true表示刷新缓存，这样可以避免数据库脏读的情况出现。

ehcache：

ehcache是一个纯Java的进程缓存框架，是一种广泛使用的开源Java分布式缓存，具有快速，精干等特点，是hibernate中默认的cacheprovider。

分布式缓存：

我们的系统为了提高并发，性能，都需要对系统进行分布式部署，不使用分布式缓存，缓存的数据在各个服务器单独存储，对缓存数据进行集中管理，使用分布式的缓存框架，服务器从集中处来获取缓存。

mybatis无法实现分布式缓存，需要和其他的分布式缓存进行整合。mybatis提供了一个cache接口，如果要想实现自己的缓存逻辑，实现cache接口即可。mybatis和ehcache整合，mybatis和ehcache整合包中提供而来cache的实现类，在<cache>标签中，type指定cache接口的实现类类型。mybatis默认为Perpetualcache要和ehcache整合，只需配置type为ehcache实现接口的类即可，配置mapper.xml中的type为ehcache实现cache接口的类型。


```
1 <cache type="org.mybatis.caches.ehcache.EhcacheCache"/>
```

ehcache配置文件:

```
1 <ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
2   xsi:noNamespaceSchemaLocation="../config/ehcache.xsd">  
3   <diskStore path="F:\develop\ehcache" />  
4   <defaultCache  
5       maxElementsInMemory="1000"  
6       maxElementsOnDisk="1000000"  
7       eternal="false"  
8       overflowToDisk="false"  
9       timeToIdleSeconds="120"  
10      timeToLiveSeconds="120"  
11      diskExpiryThreadIntervalSeconds="120"  
12      memoryStoreEvictionPolicy="LRU">  
13   </defaultCache>  
14 </ehcache>
```

二级缓存的应用场景:

对访问请求较多的且用户对查询结果的实时性不高的, 此时采用二级缓存。

二级缓存的局限性:

对细粒度的数据缓存实现效果不好。