# Mybatis学习笔记

# 什么是框架

框架（framework）是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法，另一种定义为：框架是可被应用开发者定制的应用骨架，前者是应用方面，而后者是从目的方面给出的定义。

简而言之，框架就是某种应用的半成品，就是一组组件，供你选用完成你自己的系统，简单的说就是使用别人搭好的舞台，你来表演，而且，框架一般是成熟的，不断升级的软件

它是我们软件开发中的一套解决方案，不同的框架解决的是不同的问题，

使用框架的好处：

框架封装了很多细节，使开发者使用极其简单的方式实现功能，大大提高开发的效率

# 框架要解决的问题

框架要解决的最重要的一个问题是技术整合的问题，在 J2EE 的 框架中，有着各种各样的技术，不同的软件企业需要从 J2EE 中选择不同的技术，这就使得软件企业最终的应用依赖于这些技术，技术自身的复杂性和技术的风险性将会直接对应用造成冲击。而应用是软件企业的核心，是竞争力的关键所在，因此应该将应用自身的设计和具体的实现技术解耦。这样，软件企业的研发将集中在应用的设计上，而不是具体的技术实现，技术实现是应用的底层支撑，它不应该直接对应用产生影响。

三层架构：

表现层：

是用于展示数据的

业务层：

是处理业务需求

持久层：

是和数据库做交互的

# 持久层技术解决方案

## JDBC技术：

Connection

PreparedStatement

ResultSet

## Spring的JDBCTemplate

Spring中提供的对jdbc的简单封装

## Apache的DbUtils

它和spring的jdbcTemplate很像，也就是对jdbc的简单封装

以上这些都不是框架，

Jdbc是规范

Spring的jdbcTemplate和apache的DBUtils都只是工具类

# Mybati框架概述

mybatis 是一个优秀的基于 java 的持久层框架，它内部封装了 jdbc，使开发者只需要关注 sql语句本身，而不需要花费精力去处理加载驱动、创建连接、创建 statement 等繁杂的过程。

mybatis 通过 xml或注解的方式将要执行的各种 statement 配置起来，并通过 java 对象和 statement 中 sql的动态参数进行映射生成最终执行的 sql语句，最后由 mybatis 框架执行 sql并将结果映射为 java 对象并返回。

采用 ORM 思想解决了实体和数据库映射的问题，对 jdbc 进行了封装，屏蔽了 jdbc api 底层访问细节，使我们不用与 jdbc api 打交道，就可以完成对数据库的持久化操作。

为了我们能够更好掌握框架运行的内部过程，并且有更好的体验，下面我们将从自定义 Mybatis 框架开始来学习框架。此时我们将会体验框架从无到有的过程体验，也能够很好的综合前面阶段所学的基础。

Mybatis是一个持久层框架，用Java语言编写，它封装了jdbc的很多细节，使开发者只需要关注sql语句本身，而无需关注注册驱动，创建连接等繁杂过程，它使用ORM思想实现了结果集的封装，

ORM：

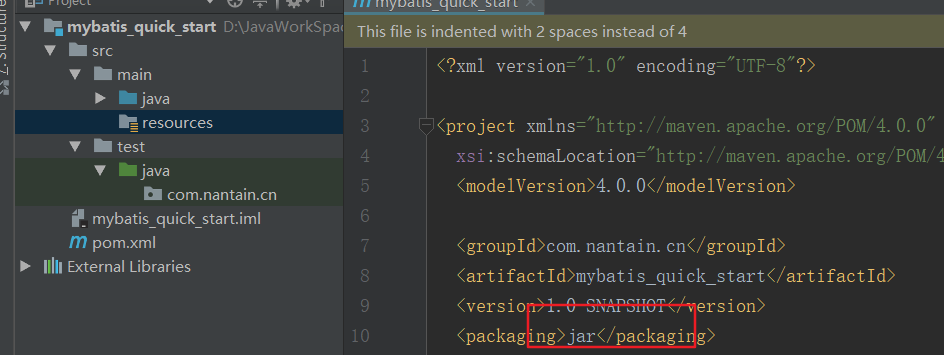
Object Relational Mapping对象关系映射

简单的说：

就是把数据库和实体类及实体类的属性对应起来，让我们可以操作实体类就实现操作数据库

# Mybatis入门案例

## 创建maven工程



## 导入mybatis相关jar包

<!--mybatis相关jar包-->  
<dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis</artifactId>  
 <version>3.4.5</version>  
</dependency>  
  
<!--书库据相关jar包-->  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.6</version>  
</dependency>  
  
<!--log4j日志jar包-->  
<dependency>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 <version>1.2.17</version>  
</dependency>  
  
<!--junit单元测试-->  
<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 <scope>test</scope>  
</dependency>

## 创建实体类、持久层接口

用户实体类：

public class User implements Serializable {  
  
 private Integer id;  
 private String username;  
 private String address;  
 private Date birthday;  
 private String sex;

持久层接口：

*/\*\*  
 \* 创建用户持久层接口  
 \*/*public interface IUserDao {  
  
 */\*\*  
 \* 提供查询所有的方法  
 \*/* public List<User> findAllUser();  
}

## 配置mybatis环境

### 创建SqlmapConfig.xml的配置文件【此为mybatis主配置文件】



### 在配置文件中添加mybatis的约束，及相关环境配置

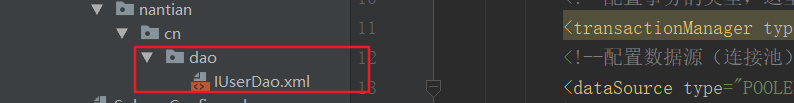
Mybatis主配置文件中的约束（详见mybatis官方文档入门项）

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

主配置文件环境配置

<configuration>  
 <!--配置环境-->  
 <environments default="mysql">  
 <!--配置mysql的环境-->  
 <environment id="mysql">  
 <!--配置事务的类型，这里一般使用JDBC-->  
 <transactionManager type="JDBC"></transactionManager>  
 <!--配置数据源（连接池）-->  
 <dataSource type="POOLED">  
 <!--配置连接数据库的四个基本信息-->  
 <property name="driver" value="com.mysql.jdbc.Driver"></property>  
 <property name="url" value="jdbc:mysql://localhost:3306/mybatisdb"></property>  
 <property name="username" value="root"></property>  
 <property name="password" value="root"></property>  
 </dataSource>  
 </environment>  
 </environments>  
  
 <!--指定映射配置文件的位置，映射配置文件指的是每个dao独立的配置文件-->  
 <mappers>  
 <mapper resource="com/nantian/cn/dao/IUserDao/xml"></mapper>  
 </mappers>  
  
</configuration>

### 创建IuserDao.xml配置文件【此为mybatis映射配置文件】



### 在Mybatis映射配置文件约束，及相关的映射配置

映射文件约束

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

映射文件配置：

<mapper namespace="com.nantian.cn.dao.IUserDao">  
  
 <!--配置查询所有-->  
 <select id="findAllUser">  
 select \* from user;  
 </select>  
</mapper>

### Mybatis的环境搭建总结：

1. 创建maven工程导入所依赖jar包的坐标
2. 创建实体类和dao的接口
3. 创建mybatis的主配置文件sqlmaoConfig.xml
4. 创建映射配置文件 IuserDao.xml

环境搭建的注意事项：

1. 创建IuserDao.xml和IuserDao.java时名称是为了和我们常规的开发保持一致。在mybats中它把持久层接口名称和映射配置文件也叫做：mapper

所以：IuserDao 和 IuserMapper是一样的

1. 在Idea中创建目录的时候，它和创建包是不一样的
2. Mybatis的映射配置文件位置必须和接口的包结构相同
3. 映射配置文件的mapper标签namespace属性的取值必须是接口的全限定类名
4. 映射配置文件的操作配置（select）,id属性的取值必须是持久层接口中的方法名

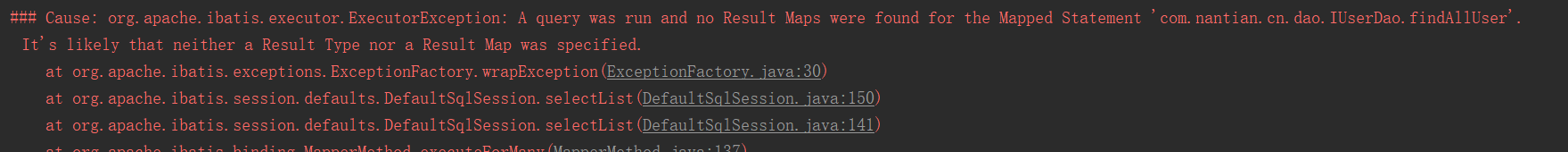
### Mybatis入门案例测试步骤：

public static void main(String[] args) throws Exception {  
  
 //1.读取配置文件  
 InputStream resourceAsStream = Resources.*getResourceAsStream*("SqlmapConfig.xml");  
 //2.创建sqlSessionFactory工厂  
 SqlSessionFactoryBuilder sqlSessionFactoryBuilder = new SqlSessionFactoryBuilder();  
 SqlSessionFactory factory = sqlSessionFactoryBuilder.build(resourceAsStream);  
 //3.使用工厂生产SqlSession对象  
 SqlSession sqlSession = factory.openSession();  
 //4.使用SqlSession对象创建持久层接口的代理对象  
 IUserDao mapper = sqlSession.getMapper(IUserDao.class);  
 //5.使用代理对象执行方法  
 List<User> users = mapper.findAllUser();  
 for (User user: users) {  
 System.*out*.println(user);  
 }  
 //6.释放资源  
 sqlSession.close();  
 resourceAsStream.close();  
}

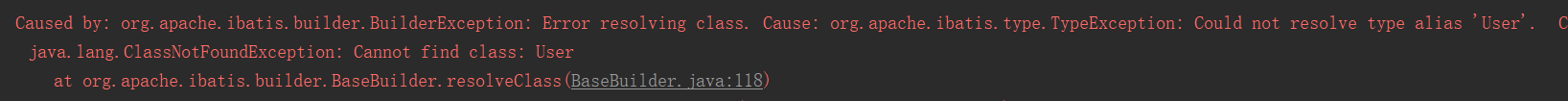
注意事项：不要忘记在映射配置文件中告知mybatis要把查询出来的结果数据封装到哪个实体类中，配置的方式就是指定实体类的全限定类名

### 可能出现的问题：

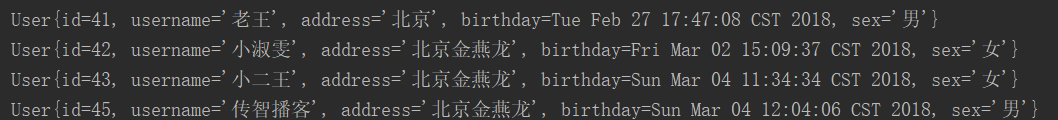
1. 因为mapper配置中select等标签中为配置resultType属性，框架不知道查询出来的数据要封装到哪里去



1. 配置了resultType后报错找不到类型，因为没有配置包扫描，所以resultType中配置的放回值类型，还不能使用别名，及不能直接写诸如：resultType = “User”,这时在没配置包扫描的情况下， 返回的类型应该写的是类的全限定类名



经过以上的修复，就能正确的查询出数据结果了：



# Mybatis注解开发入门

## 主配置文件不要动，映射配置文件则可以删除了

主配置文件不用动，是指在主配置文件中配置的数据库连接信息不用动，与使用xml映射配置文件不同的是，我们在主配置文件中的<mappers>标签中将使用class属性来指定被注解的dao的全限定类名，配置如下：

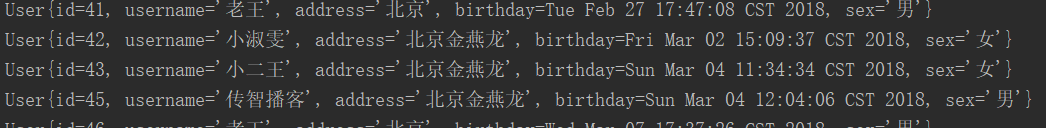
<!--指定映射配置文件的位置，映射配置文件指的是每个dao独立的配置文件  
 如果使用注解来配置的话，此处应该使用class属性来指定被注解的dao全限定类名  
-->  
<mappers>  
 <mapper class="com.nantian.cn.dao.IUserDao"></mapper>  
</mappers>

## 在dao接口的方法上使用@Select等注解

在注解的属性中来书写我们需要执行的sql语句

*/\*\*  
 \* 提供查询所有的方法  
 \*/*@Select("select \* from user")  
 List<User> findAllUser();

测试运行结果：



# Mybatis通过编写dao接口的实现类来实现持久层操作

这不是用的最多的开发，在开发中为了程序的简练，我们使用的最多的就是不编写dao实现类的方式来进行开发，但也有一些比较老，又奇葩的公司还是会使用编写dao实现的方式来进行开发，比如某某银行的某某项目

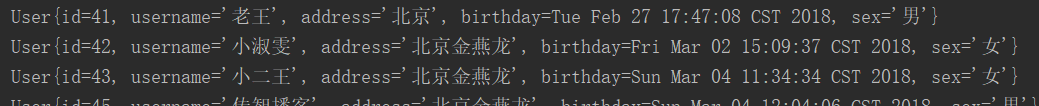
## 编写dao接口的实现

public class UserDaoImpl implements IUserDao {  
  
 private SqlSessionFactory factory;  
  
 public UserDaoImpl(SqlSessionFactory factory) {  
 this.factory = factory;  
 }  
  
 @Override  
 public List<User> findAllUser() {  
 //1.使用工厂创建sqlSession对象  
 SqlSession sqlSession = factory.openSession();  
 //2.使用sqlSession对象中的 selectList()方法 来执行查询所有的操作  
 List<User> users = sqlSession.selectList("com.nantian.cn.dao.IUserDao.findAllUser");  
 sqlSession.close();  
 //3.返回查询结果  
 return users;  
 }  
}

测试：

//1.读取配置文件  
InputStream resourceAsStream = Resources.*getResourceAsStream*("SqlmapConfig.xml");  
//2.创建sqlSessionFactory工厂  
SqlSessionFactoryBuilder sqlSessionFactoryBuilder = new SqlSessionFactoryBuilder();  
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(resourceAsStream);  
//3.使用工厂创建dao对象  
IUserDao userDao = new UserDaoImpl(factory);  
//4.调用dao接口中的方法，  
List<User> users = userDao.findAllUser();  
for (User user : users) {  
 System.*out*.println(user);  
}  
resourceAsStream.close();

运行结果：



## 总结：

编写dao接口实现的方式，和不编写dao接口实现的方式，主要区别在于，因为不编写dao接口，所以mybatis通过SqlSession对象，映射配置文件，给我们生成的dao的代理对象，也就是要对dao接口进行，通过映射配置文件中的具体持久层操作，增、删、改等进行增强，而在编写dao接口的实现类方式中，由于我们有了dao接口的具体的实现，通过sqlsession中的具体的方法来进行持久层操作，比如selectList()方法，这些SqlSession对象中替我们封装好的方法，需要一个statement，而这个statement就是映射配置文件中的namespace + 接口方法名，把这个作为statement参数，就可以找到我们在映射配置文件中书写的具体的要执行的sql语句了（也就是具体的持久层操作）

# Mybatis设计模式分析

//1.读取配置文件  
InputStream resourceAsStream = Resources.*getResourceAsStream*("SqlmapConfig.xml");  
//2.创建sqlSessionFactory工厂  
SqlSessionFactoryBuilder sqlSessionFactoryBuilder = new SqlSessionFactoryBuilder();  
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(resourceAsStream);  
//3.使用工厂生产SqlSession对象  
SqlSession sqlSession = factory.openSession();  
//4.使用SqlSession对象创建持久层接口的代理对象  
IUserDao mapper = sqlSession.getMapper(IUserDao.class);  
//5.使用代理对象执行方法  
List<User> users = mapper.findAllUser();  
for (User user: users) {  
 System.*out*.println(user);  
}  
//6.释放资源  
sqlSession.close();  
resourceAsStream.close();

## 读取配置文件

绝对路径：D:/xx/xxx.xml 局限性：如果电脑没有d盘的情况

相对路径：src/java/main/xxx.xml 局限性： 如果是个web项目我们的项目一部署src目录就没有了

所以这两种方式在开发中都很少会使用

都配置文件通常使用如下方式：

a.使用类加载器，它只能读取类路径的配置文件

b.使用ServletContext对象额getRealPath()方法，它能获取到当前项目部署的绝对路径，这个路径是项目运行在哪里，它就在哪，所以说，读取配置文件的时候我们只是用这种方式，其它的都有局限性，不太靠谱

## 创建工厂，mybatis使用了构建者模式

Builder就是构建者

构建者模式的优势：把对象的创建细节隐藏，使使用者直接调用方法即可拿到对象，屏蔽了那些繁琐的操作

## 创建SqlSession使用了工厂模式

工厂模式的优势：解耦（降低类之间的依赖关系），使我们每次不用去重新编译，重新部署

## 创建Dao接口实现类，使用了代理模式

代理模式的优势：在不修改源码的基础上，对已有的方法进行增强，也就是我们通过SqlSession获取Dao接口的实现类，获取到的是Dao接口的代理对象，从而实现了不编写Dao实现类也能够实现功能，完成持久层的操作

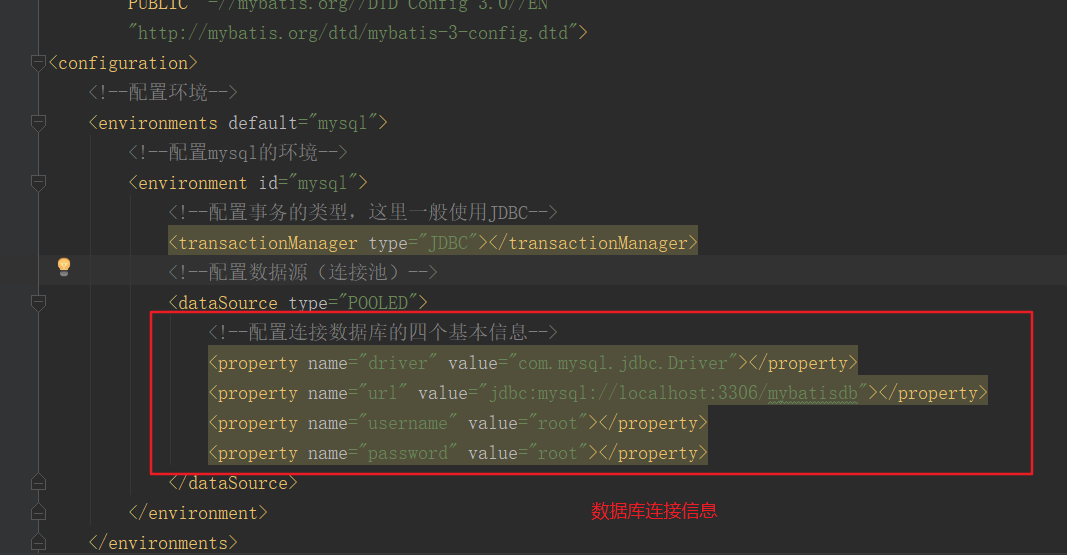
Mybatis在使用代理Dao接口的方式实现持久层操作(增删改查)时

做了两件事

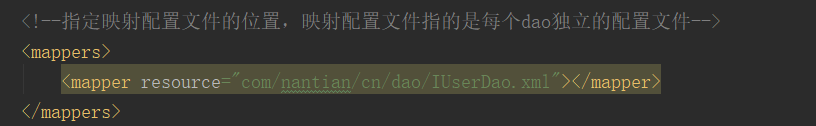
1. 创建代理对象
2. 在代理对象中调用SqlSesseion对象中的方法，诸如：selectList()

## 配置文件分析

### 主配置文件（mybatis-3-config.dtd）



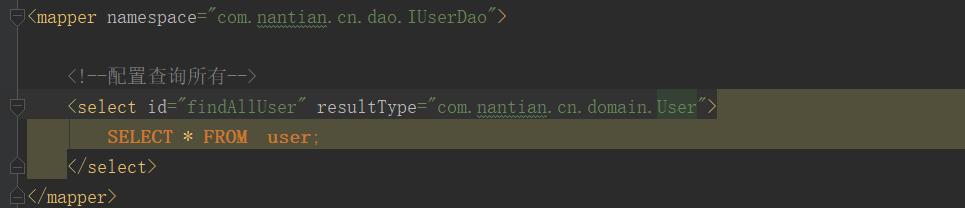
数据库连接的信息，有了它们就可以创建Connection对象：



Mapper映射位置信息：有了这个配置，就可以找到映射配置信息

### 映射配置文件(mybatis-3-mapper.dtd)

有了它就知道了映射配置信息



有了它就有了执行的sql语句，可以获取PreparedStatement预编译对象，同时此配置中还有封装的实体类的权限定类名，这时候，这两个就构成了我们的用什么语句执行，封装到哪里去，所以这些配置信息是我们上来首先要做的事情，也就是读取配置文件，读成一个流，解析配置文件，解析配置文件，用到的是dom4j来解析xml配置文件

## 解析完Xml配置文件后selectList()方法：

1. 根据配置文件的信息创建Connection对象

注册驱动，获取连接

1. 获取预处理对象，PreparedStatement

此时需要Sql语句

1. 执行持久层操作，比如查询

ResultSet resultSet = preparedStatement.execute(Query);

1. 遍历结果集用于封装

List<E> list = new ArrayList<>();

While(resultSet.next()) {

E element = xxxx;//这里的element可以根据映射配置文件中配置的实体类的权限定类名，通过反射的方式来获得【我们的实体类属性和数据库表中的列名是一致的，

于是我们就可以把表的列名看成是实体类的属性名称，就可以使用反射的方式根据名称获取每个属性，并把值赋进去】

进行封装，把每个result的内容添加到element中，把element加入到list中

List.add(element);

}

1. 返回list

Return list;

### 要想让selectList()方法执行

1. 连接信息
2. 映射信息

它包含了两个部分，

1. 执行的sql语句
2. 封装结果的实体类权限定类名

把这两个信息组合起来，定义一个对象，保存到map<Sttring,Mapper>集合中，以命名空间加方法名作为key,mapper对象作为value，mapper对象中，包含了执行的sql语句以及配置的实体类的全限定类名

|  |  |
| --- | --- |
| String | Mapper |
| com.nantian.cn.dao.IUserDao  .findAllUser | Mapper对象{String sql；  String domianClassPath} |

## 根据dao接口的字节码创建dao的代理对象

//4.使用SqlSession对象创建持久层接口的代理对象  
IUserDao mapper = sqlSession.getMapper(IUserDao.class);

Public <T> getMapper(Class<T> daoInterfaceClass) {

/\*\*

类加载器：它使用的和被代理对象相同的类加载器

代理对象实现的接口：和被代理对象实现相同的接口

如何代理：它就是增强的方法，我们需要自己来提供，此处是一个InvocationHandler的接口，我们需要写一个改接口的实现类，在实现类中调用selectList()方法

\*/

Proxy.newProxyInstance(类加载器，代理对象要实现的接口的字节码数组，如何代理)；

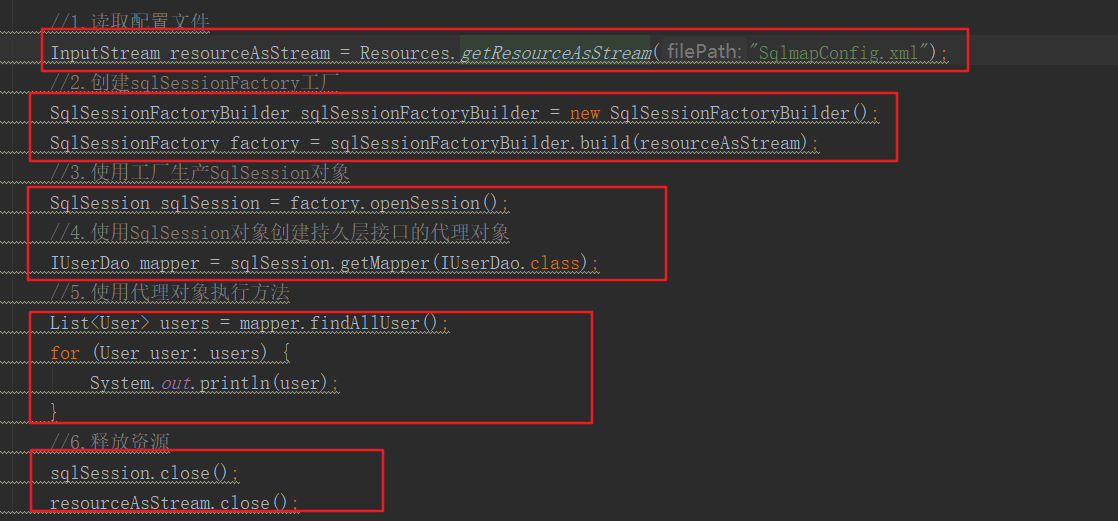
}

# 自定义mybatis框架

## 分析

通过入门案例中可以看到mybatis中能看到的类：

1. class Resources
2. class SqlSessionFactoryBuilder
3. interface SqlSessionFactory
4. interface SqlSession



## 步骤

### 通过类加载器读取配置文件

1.class Resources

getResourceAsStream(String filepath),根据传入的配置文件路径，使用累加器获取到一个字节输入流，并返回

*/\*\*  
 \* 使用类加载器读取配置文件的类  
 \*/*public class Resources {  
  
 */\*\*  
 \* 根据传入的参数获取一个字节输入流  
 \** ***@param*** *filePath  
 \** ***@return*** *\*/* public static InputStream getResourceAsStream(String filePath) {  
 return Resources.class.getClassLoader().getResourceAsStream(filePath);  
 }  
}

### 通过构建者模式，构建SqlSessionFactory工厂对象

class SqlSessionFactoryBuilder

该类中的build(InputStream resourceAsStream)方法，通过传入的字节输入流，使用XML解析工具类，得到一个Configuration对象，然后通过传入Configuration对象，创建一个interface SqlSessionFactory接口实现类的实例对象，也就是工厂对象

*/\*\*  
 \* 用于创建一个SqlSessionFactory对象  
 \*/*public class SqlSessionFactoryBuilder {  
 */\*\*  
 \* 根据参数的字节输入流，使用XML解析工具类得到一个Configuration对象，  
 \* 再通过Configuration对象创建一个工厂接口的实例对象  
 \** ***@param*** *resourceAsStream  
 \** ***@return*** *\*/* public SqlSessionFactory build(InputStream resourceAsStream) {  
 Configuration config = null;  
 try {  
 config = XMLConfigBuilder.*buildConfiguration*(resourceAsStream);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return new DefaultSqlSessionFactory(config);  
 }  
}

### 使用工厂模式创建SqlSession对象

interface SqlSessionFactory 其中的openSession()方法，来获取SqlSession对象，使用了改接口的实现类，DefaultSqlSessionFactory，实现类中的openSession()方法，通过传入Configuration对象，通过interface SqlSession接口实现类的有参构造方法 ，创建了一个SqlSessin实例并返回；

*/\*\*  
 \* SqlsessionFactory接口的实现类  
 \*/*public class DefaultSqlSessionFactory implements SqlSessionFactory {  
  
 private Configuration config;  
  
 public DefaultSqlSessionFactory(Configuration config) {  
 this.config = config;  
 }  
 */\*\*  
 \* 用于创建一个操作数据库的核心对象  
 \** ***@return*** *\*/* @Override  
 public SqlSession openSession() {  
 return new DefaultSqlSession(config);  
 }  
}

### 使用代理模式创建Dao（mapper）接口的代理对象

interface SqlSession在其实现类中的getMapper(Class<T> daoInterfaceClass)方法，通过传入的Dao接口的字节码，使用代理模式，最终返回一个Dao接口的代理对象

*/\*\*  
 \* 用于创建代理对象  
 \** ***@param*** *daoInterfaceClass dao接口的字节码  
 \** ***@param*** <*T*>  
 *\** ***@return*** *使用动态代理的时候，通过Proxy.newProxyInstance()获取到的就是最后的代理对象；  
 \*/*@Override  
public <T> T getMapper(Class<T> daoInterfaceClass) {  
  
 Object proxyMapper = Proxy.*newProxyInstance*(daoInterfaceClass.getClassLoader(), new Class[]{daoInterfaceClass},  
 new InvocationHandler() {  
 @Override  
 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
 //1.获取方法名  
 String methodName = method.getName();  
 //2.获取方法所在类的名称  
 String className = method.getDeclaringClass().getName();  
 //3.组合key  
 String key = className + "." + methodName;  
 Map<String, Mapper> mappers = config.getMappers();  
 Mapper mapper = mappers.get(key);  
 if (mapper == null) {  
 throw new IllegalArgumentException("传入的参数有误");  
 }  
  
 //6.调用工具类，执行查询所有  
 return new Executor(config).executeQuery(mapper.getQuerySql(), mapper.getResultType());  
 }  
 });  
 return (T) proxyMapper; //返回Dao接口的代理对象  
}

详见mybatis案例：cstom\_mybatis

# Crud细节

## Mybatis中使用#{}，${}的区别

${}使用的是PrepareStatement参数字符串的拼接，#{}使用的是PrepareStatement的参数占位符，

## 新增数据 id 的返回值

Select last\_insert\_id();

<!--保存用户-->  
<insert id="saveUser" parameterType="com.nantian.cn.pojo.User" >  
 <!-- 配置插入操作后，获取插入数据的id-->  
 <selectKey keyProperty="id" keyColumn="id" resultType="int" order="AFTER">  
 SELECT last\_insert\_id();  
 </selectKey>  
 INSERT INTO user (username,birthday,sex,address) VALUES (#{username},#{birthday},#{sex},#{address});  
</insert>

在<insert标签内部使用<selectKey标签，标签中的属性keyProperty对应Java实体类中的id属性名称，keyColumn属性对应数据库表中id的列名

## OGNL表达式

Object Graphic Navigation Language

对象图导航语言

它是通过对象的取值方法来获取数据，在写法上把get给省略了

比如：我们获取用户名称

类中的写法：user.getUserName();

OGNL表达式写法：user.username

Mybatis中为什么能直接写username,而不用user呢？因为在parameterType中已经提供了属性所属的类，所以此时不需要写对象名

## 实体类中的属性与数据库中列名不一致

此时如果不做任何处理的话，查询出来的数据将无法封装到对应的实体类中，这时有两种解决方案：

1. 就是在写SQL语句额时候，对查询的列起别名，把查询结果的列名起一个Java实体类属性对应的别名，这种方法的执行效率也是最高的，因为它是在SQL语句的层面上解决的问题
2. 使用resultMap，这也是实际开发中用的

使用resultMap 的时候，<reslut标签中的property属性中是对应Java中的实体类中的属性，是严格区分大小写的，Column属性对应MySQL数据库中的列名，MySQL数据库在windows系统中不区分大小写，但在Linux系统中严格区分大小写，由于我们的项目最终都会部署到Linux服务器中，所以都严格区分大小写即可

<resultMap id="resultMap" type="com.nantian.cn.pojo.User">  
 <id property="id" column="id" ></id>  
 <result property="username" column="username"></result>  
 <result property="userBirthday" column="birthday"></result>  
 <result property="userSex" column="sex"></result>  
 <result property="userAddress" column="address"></result>  
</resultMap>

Id属性，是resultMap的唯一标识，type属性为要封装的实体类，<id标签一般用来标识该属性对应数据库中的主键列

<result标签，用来配置数据库实体类中的属性和数据库表中的列名对应关系，property属性为Java实体类中的属性名称，严格区分大小写，Column属性为数据库表中列名，MySQL在Windows系统中不区分大小写

ResultMap执行效率没有起别名高，但是开发效率更高，因为我们开发中可能有很多种查询，如果起别名的话，每条SQL语句都要起别名，而使用resultMap定义好返回值的封装以后，一个mapper映射文件中的所有查询操作都可以使用该resultMap，利于复用，开发效率高

# Mybatis中参数的深入

## parameterType(输入类型)

### 传递简单类型

（1）我们传的int【比如根据id查询，或者根据主键删除】

（2）还有就是模糊查询的时候传递的String 【比如根据输入某个用户名包含的字符】

### 传递pojo对象

Mybatis使用OGNL表达式，来解析字段值，#{}或者${}，花括号中的值为pojo属性名称

### 传递pojo包装对象

开发中通过 pojo 传递查询条件 ，查询条件是综合的查询条件，不仅包括用户查询条件还包括 其它的查询条件（比如将用户购买商品信息也作为查询条件），这时可以使用包装对象传递输入参数。

Pojo 类中包含 pojo。

需求：根据用户名查询用户信息，查询条件放到 QueryVo 的 user 属性中。

## resultType(输出类型)

### 输出简单类型

1. 输出int类型，比如查询总记录条数，返回一个int值

### 输出pojo对象

比如查询一条数据，一般是根据id(主键)查询一条记录

### 输出pojo列表

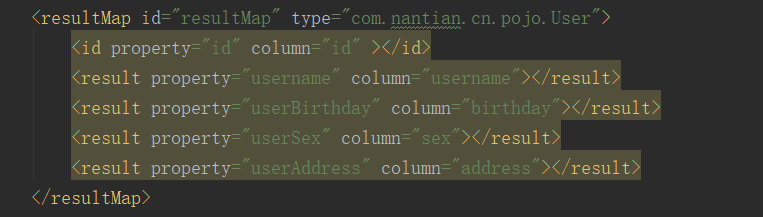
比如，查询所有，注意：虽然说是返回pojo列表，但是resultType返回类型并不需要写List

而是仍然写pojo类型，比如查询所有，将返回一个保存者pojo对象的list集合

## resultMap 结果类型

resultType 可以指定 pojo 将查询结果映射为 pojo，但需要 pojo 的属性名和 sql 查询的列名一致 方可映射成功。

如果 sql 查询字段名和 pojo 的属性名不一致，可以通过 resultMap 将字段名和属性名作一个对应 关系 ，resultMap 实质上还需要将查询结果映射到 pojo 对象中【type属性，就是将查询结果映射到pojo对象中】。



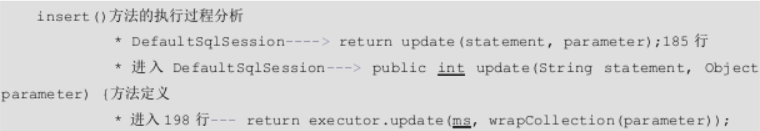
resultMap 可以实现将查询结果映射为复杂类型的 pojo，比如在查询结果映射对象中包括 pojo 和 list 实现一对一查询和一对多查询。

# Mybatis执行流程分析

## 编写Dao接口实现类的方式执行流程

现在我们可以通过调用 SqlSession 接口中的方法实现 CRUD 操作了，那么 SqlSession 接口中的方法是 如何实现的。

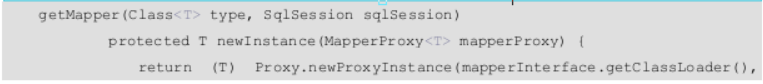
下面一起来分析 sqlSession 接口中的 insert()方法的执行过程，我们进行源码跟踪发现了 Mybatis 只是 对于 jdbc 的封装。

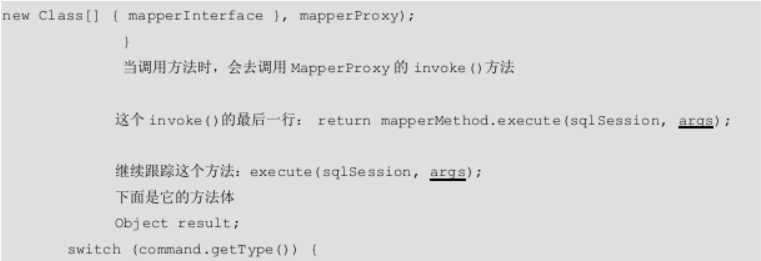




## 使用代理Dao方法的执行流程

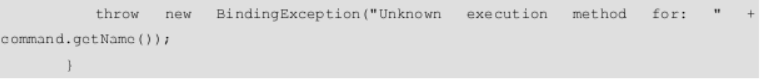
通过对源代码进行跟踪，我们会发现在调用 SqlSession 接口的 getMapper()方法时，它的底层进一步 调用了 JDK 动态代理方式来生成代理子类对象的，这个过程如下：











## PreparedStatement对象有三种执行方法：

1. execute：它能执行CRUD中的任意一种语句，他的返回值是一个Boolean类型，白哦是是否有结果集，有结果集返回true，没有结果集返回false
2. executeUpdate：他只能执行CUD操作，查询语句无法执行，它的返回值是影响数据库记录的行数，
3. executeQuery：他只能执行select语句，无法执行增删改，执行结果为封装的结果集ResultSet对象

## 总结：

## 传统的开发模式编写Dao的实现类的执行流程

Mybatis的执行流程，分为了两类操作一类是CUD(增删改)操作，一类是R(查询)操作，增删改操作，首相都会进入到DefaultSqlSession中，根据传入语句、语句的参数不同，调用不同的insert、delete、update方法，但是insert、delete、update最终都会调用update方法，在update方法中继续调用Executor中的update方法，Executor是一个接口，所以首先会进入到其实现类CachingExecutor中的update方法，然后在CachingExecutor的update方法中又调用了delegate.update()的方法，该方法又是Executor接口中的方法，最终会调用到BaseExecutor中的update方法，在该方法中又会去调用一个叫doUpdate()的方法，该方法是抽象方法，找到起子类SimpleExecutor，最终调用的是其子类SimpleExecutor中的doUpdate()方法，在该方法中又会去调用handler.update()的方法，handler是PreparedStatementHandler对象，里面是抽象，要找到其子类RoutingStatementHandler,执行RoutingStatementHandler中的update方法，在该方法中最终调用的是PreparedStatementHandler中的update()方法；PreparedStatementHandler类中的update方法执行的就是传统的jdbc的操作了

查询操作，大致流程和增删改一样，所不同的是由最初调用的query方法，最终调用到doQuery()方法，

## 使用代理Dao的模式开发

该中开发模式和传统的最大区别就是使用了动态代理，通过传入Dao接口的字节码，通过getMapper()方法，使用动态代理，在代理的过程中，也是调用mybatis增删改查的那一套操作流程，所以我们不需要写Dao的接口的实现，通过代理Dao的模式，也能完成编写Dao实现类的模式所能实现的功能

# Mybatis主配置文件SqlMapConfig.xml深入配置

配置内容SqlMapConfig.xml 中配置的内容和顺序如下：

properties（属性）

settings（全局配置参数）

typeAliases（类型别名）

typeHandlers（类型处理器）

objectFactory（对象工厂）

plugins（插件）

environments（环境集合属性对象）

environment（环境子属性对象）

transactionManager（事务管理）

dataSource（数据源）

mappers（映射器）

## properties（属性）

SqlMapConfig.xml 可以引用 java 属性文件中的配置信息如下：

在 classpath 下定义 db.properties 文件:



SqlMapConfig.xml 引用如下：

可以在<properties标签内部使用<property标签来配置数据库的连接信息，也可以在<properties 标签中使用resource属性来引入外部配置文件

常用属性：

1. Resource属性：用于指定配置文件的位置，是按照类路径的写法来写，并且必须存在于类路径下，
2. url属性：是要求按照url的写法来写地址，

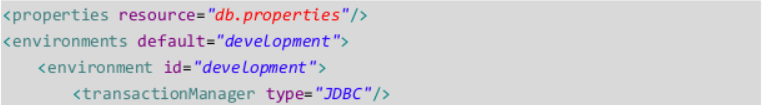
url: Uniform Resource Locator 统一资源定位符，它是可以唯一标识一个资源的位置

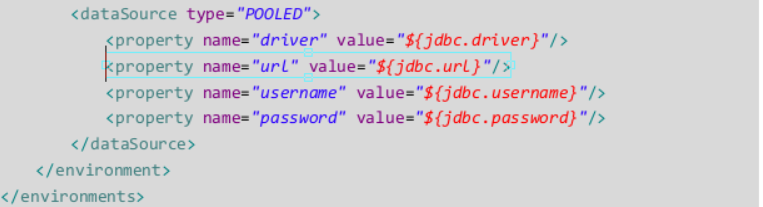
它的写法：协议 主机 端口 URI

如：<http://localhost:8080/mybatisService/demoServlet>

URI:Uniform Resource Identifier 统一资源标识符，它是在应用中可以唯一定位一个资源的

使用url属性的时候，一定要使用file协议，如：file：///c:/db.properties





注意： MyBatis 将按照下面的顺序来加载属性：

 使用 properties 元素加载的外部属性文件优先级最高。

 然后会读取 properties 元素中 resource 加载的属性，它会覆盖已读取的同名属性。

## typeAliases（类型别名）

在 SqlMapConfig.xml 中配置：



<!--使用typeAliases配置别名，它只能配置domain中类的别名，-->  
<typeAliases>  
 <!--typeAlias用于配置别名，type属性，用于指定的是实体类的全限定名，alias属性，指定别名，  
 当指定了别名就不在区分大小写了-->  
 <typeAlias type="com.nantian.cn.pojo.User" alias="user"></typeAlias>  
  
 <!--用于指定要配置别名的包，当指定之后，该包下的实体类就会注册别名，并且类名就是别名，不再区分大小写-->  
 <package name="com.nantian.cn.pojo"></package>  
</typeAliases>

## mappers（映射器）

(1) 使用相对于类路径的资源

<mapper resource="com/nantian/cn/dao/IUserDao.xml"></mapper>

(2) 使用 mapper 接口类路径,通常使用注解开发的时候会使用class属性

如：<mapper class="com.nantian.cn.dao.IUserDao"/>

注意：此种方法要求 mapper 接口名称和 mapper 映射文件名称相同，且放在同一个目录中

（3）注册指定包下的所有 mapper 接口

如：<package name="com.nantian.cn.dao"></package>

此种方法要求 mapper 接口名称和 mapper 映射文件名称相同，且放在同一个目录中。

# Mybatis连接池

## 连接池

连接池就是用于存储连接的一个容器，该容器其实就是一个List集合对象，该集合必须是线程安全的，不能两个线程同时拿到同一个连接，而且，该集合还必须实现队列的特性，也就是先进先出

我们在实际开发中都会使用连接池，因为它可以减少我们获取连接所消耗的时间

## Mybatis中的连接池

Mybatis中的连接池提供了三种方式的配置：

配置的位置：

主配置文件中SqlMapConfig.xml中的dataSource标签，type属性就是表示采用了何种连接池

Type属性的取值：

1. POOLED 采用传统的javax.sql.DataSource规范中的连接池，mybatis中有针对规范的实现
2. UNPOOLED采用传统的获取连接的方式，虽然也是javax.sql.DataSource接口，但是并没有使用池的思想。
3. JNDI 采用服务器提供的JNDI技术实现，来获取DataSource对象，不同的服务器所能拿到的DataSource对象是不一样的，

注意：如果不是web项目或者maven项目的war工程，是不能使用的，我们使用tomcat服务器的时候，采用的连接池就是dbcp连接池

相应地，MyBatis 内部分别定义了实现了 java.sql.DataSource 接口的 UnpooledDataSource，

PooledDataSource 类来表示UNPOOLED、POOLED 类型的数据源。

MyBatis 在初始化时，解析此文件，根据<dataSource>的 type 属性来创建相应类型的的数据源

DataSource，即：

type=”POOLED”：MyBatis 会创建 PooledDataSource实例，从连接池中获取连接

type=”UNPOOLED” ： MyBatis 会创建UnpooledDataSource 实例，而且是每次都会创建一个新的连接

type=”JNDI”：MyBatis 会从JNDI 服务上查找 DataSource 实例，然后返回使用

使用POOLED连接池的执行分析：

1. 首先，!state.idelConnections.isEmpty();会判断是否还有空闲的连接，如果还有空闲的连接，通过conn = idelConnections.remove(0);【idelConnections就是一个list集合】从集合/连接池中拿一个连接来用
2. a.如果没有空闲的连接，首先通过state.sactiveConnections.size() < pooleMaxmumActiveConnections判断活动的连接池数量是否小于设定的最大活动连接池的数量；如果小于，就会创建一个新的连接来使用conn = new PooledConnection(args XXX);

b.如果活动连接池的数量不小于设定的最大活动连接池数量， 也就是说连接池满了，没地放新的连接了，会通过PooledConnection oldestActiveConnection = state.activeConnections.get(0);也就是获取活动连接池中最老的一个连接出来使用，然后进行一些相关的设置和清理操作，

总结:首先就是连接池分为两部分的空间，一部分是空闲池，一部分是活动池，当来了一个线程需要获取连接的时候，（1）首先会去空闲池获取，如果空闲池中还有连接的话，直接拿出一个来用，（2）如果空闲池中已经没有连接了，会判断活动连接池中是否已经到了最大容量，会获取最老的连接来使用，如果活动连接池中还有空间，就会创建一个新的连接来使用。

配置：在mybatis主配置文件中通过<dataSource 标签中的type属性来进行配置，而<dataSource 标签中的<property标签配置的就是连接数据库的四大信息

<!--配置连接池-->  
<dataSource type="POOLED">  
 <property name="driver" value="com.mysql.jdbc.Driver"></property>  
 <property name="url" value="jdbc:mysql://localhost:3306/mybatisdb"></property>  
 <property name="username" value="root"></property>  
 <property name="password" value="root"></property>  
</dataSource>

# Mybatis中的事务

什么是事务

事务的四大特性ACID

不考虑隔离性会产生的三个问题

解决方法：

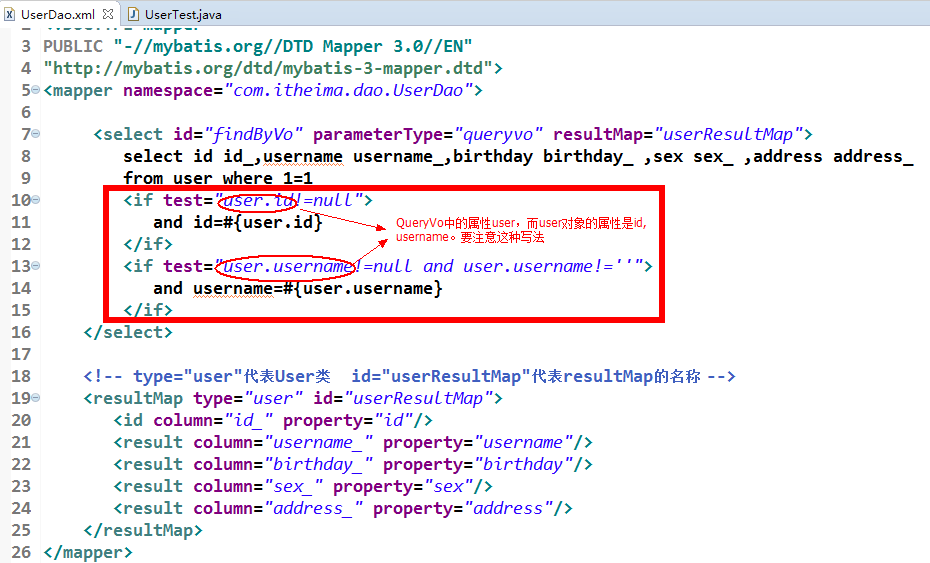
# Mybatis中的动态SQL

## 动态 SQL之<if>

我们根据实体类的不同取值，使用不同的 SQL语句来进行查询。比如在 id如果不为空时可以根据 id查询，如果 username 不同空时还要加入用户名作为条件。这种情况在我们的多条件组合查询中经常会碰到。

注意：<if>标签的 test属性中写的是对象的属性名，如果是包装类的对象要使用 OGNL表达式的写法。

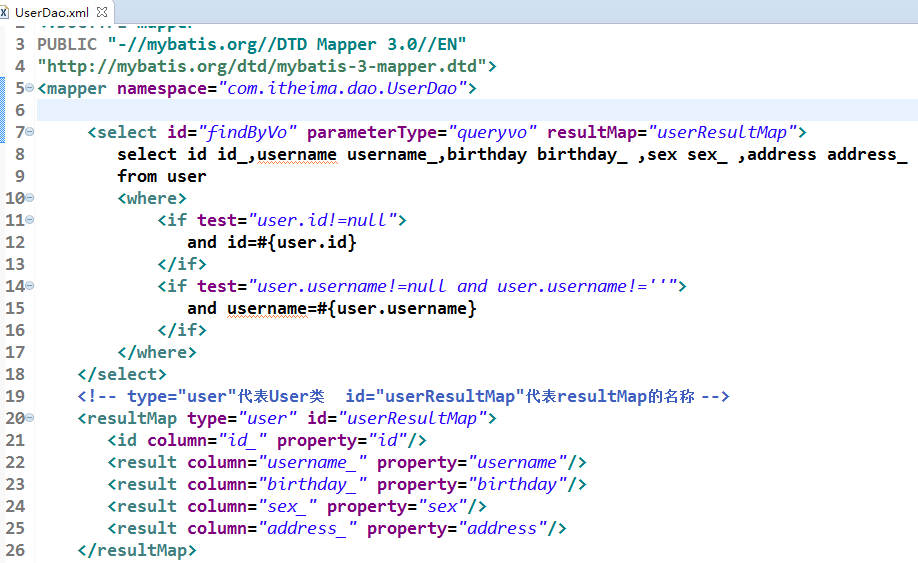
另外要注意where 1=1 的作用~！



## 动态 SQL之<where>

为了简化上面 where 1=1 的条件拼装，我们可以采用<where>标签来简化开发。

修改 UserDao.xml 映射文件如下：



## 动态标签之<foreach> 标签

需求：

传入多个 id 查询用户信息，用下边两个 sql 实现：

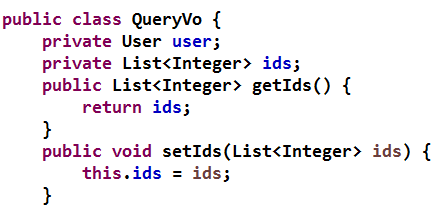
SELECT \* FROM USERS WHERE username LIKE '%张%' AND (id =10 OR id =89 OR id=16)

SELECT \* FROM USERS WHERE username LIKE '%张%' AND id IN (10,89,16)

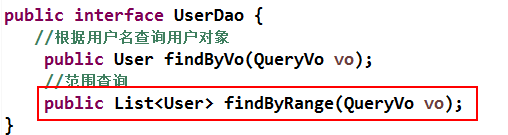
这样我们在进行范围查询时，就要将一个集合中的值，作为参数动态添加进来。

这样我们将如何进行参数的传递？

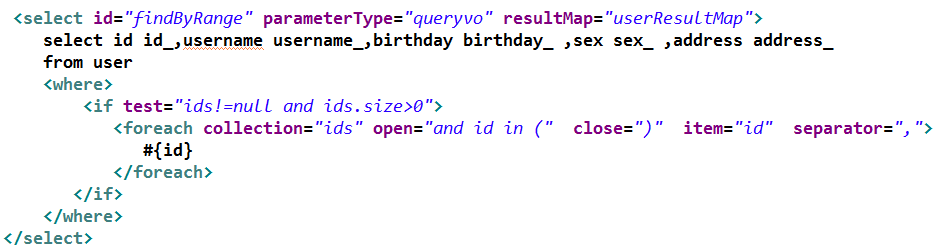
在在 QueryVo 中加入一个 List集合用于封装参数



实现范围查询，在 UserDao 接口中添加方法



在 在 UserDao.xml 文件添加范围查询的配置



SQL 语句： select 字段from user where id in (?)

<foreach>标签用于遍历集合，它的属性：

collection:代表要遍历的集合元素，注意编写时不要写#{}

open:代表语句的开始部分

close:代表结束部分

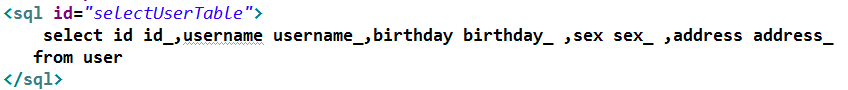
item:代表遍历集合的每个元素，生成的变量名

sperator:代表分隔符

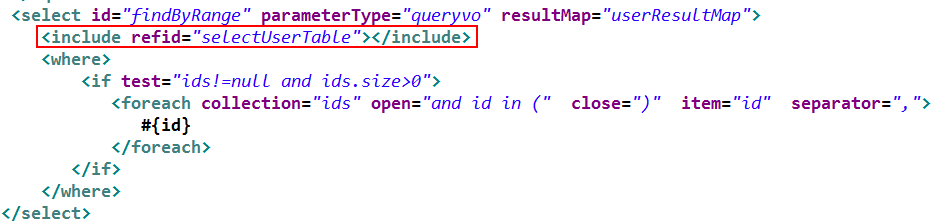
## Mybatis中简化编写的SQL片段

Sql 中可将重复的 sql 提取出来，使用时用 include 引用即可，最终达到 sql 重用的目的。

我们先到 UserDao.xml 文件中使用<sql>标签，定义出公共部分，如下：



然后在UserDao.xml 文件中用<include>标签再引用上面的 id。



其中<include>标签的 refid 属性的值就是<sql> 标签定义 id的取值。

注意：如果引用其它 mapper.xml 的 sql 片段，则在引用时需要加上 namespace，如下：

<include refid="namespace.sql 片段”/>

# Mybatis多表关联查询

## 一对一查询

这种情况也是多对一,账户和用户，就是多个账户可以属于一个用户，是多对一，但是细化到每一个账户的话，其实还是一对一，并且这种情况在mybatis中也就是当一对一来做的，mybatis中并没有严格的多对一

案例：账户----用户（账户对用户是是多对一，多个账户可属于一个用户）

<!--定义封装Account和User的resultMap-->  
<resultMap id="accountUserMap" type="account" >  
 <!--我们在配置column属性的视乎，不一定对应的数据库表中的属性名，其实它对应的是最终SQL执行后的结果的列名，  
 比如我们起了别名，它对应的是起了别名的列的名称-->  
 <id property="id" column="aid"></id>  
 <result property="uid" column="uid"></result>  
 <result property="money" column="money"></result>  
  
 <!--一对一的关系映射，配置封装user的内容  
 association标签中的column属性，表示我们用哪个字段来获取一对一中一所对应的实体类  
 这里是用过账户的uid来获取对应的用户  
 -->  
 <association property="user" column="uid">  
 <id property="id" column="id" ></id>  
 <result property="username" column="username"></result>  
 <result property="birthday" column="birthday"></result>  
 <result property="address" column="address"></result>  
 <result property="sex" column="sex"></result>  
 </association>  
</resultMap>

查询所有为例:要求查询所有账户，并包含账户所属的用户信息

<!--查询所有-->  
<select id="findAll" resultMap="accountUserMap">  
 SELECT u.\* ,a.id as aid ,a.uid,a.money FROM account a ,user u where u.id = a.uid;  
</select>

## 插播：

select \* from 表a,表b where 条件这种就是数据库中的内连接查询，这种查询方式要通过条件来避免产生笛卡尔积，这里语句中吧inner join给省略了

对于下面的一对多案例，我们的需求是要查询出所有的用户信息以及用户下的账户信息，不管有没有账户都必须查出所有的用户信息，所以这时我们需要使用外连接查询

## 一对多查询

案例：用户----账户（用户对账户是一对多，一个用户可以有几个账户）

<!--定义User的resultMap-->  
<resultMap id="userAccountMap" type="user">  
 <id property="id" column="id"></id>  
 <result property="username" column="username"></result>  
 <result property="birthday" column="birthday"></result>  
 <result property="address" column="address"></result>  
 <result property="sex" column="sex"></result>  
  
 <!--配置user对象中account集合的映射 ofType属性对应实体类-->  
 <collection property="accounts" ofType="account">  
 <id property="id" column="aid"></id>  
 <result property="uid" column="uid"></result>  
 <result property="money" column="money"></result>  
 </collection>  
</resultMap>

查询所有用户为例：查询所有的用户，并包含用户所有的账户信息

<!--查询所有用户-->  
<select id="findAllUsers" resultMap="userAccountMap">  
 SELECT u.\* ,a.id as aid, a.uid ,a.money FROM USER u LEFT OUTER JOIN account a ON u.id = a.UID;  
</select>

## 多对多

示例:用户和角色

一个用户可以有多个角色

一个角色可以赋予多个用户

步骤：

1. 建立两张表：用户表和角色表

让用户表和角色表具有多对多的关系，需要使用中间表，中间表包含各自的主键，在中间表中是外键

1. 建立两个实体类：用户实体类和角色实体类，

让用户和角色的实体类实现出来多对多的关系

各自包含对象一个集合引用

1. 建立两个配置文件

用户的配置文件

角色的配置文件

1. 实现配置

当我们查询用户时，可以同时得到用户所包含的角色信息

当我们查询角色时，可以同时得到角色所赋予的用户信息

案例：角色----用户（角色和用户为多对多，一个角色可以赋予多个用户，并且一个用户可以拥有多个角色）

<!--定义User的resultMap-->  
<resultMap id="roleMap" type="role">  
 <id property="roleId" column="rid"></id>  
 <result property="roleName" column="ROLE\_NAME"></result>  
 <result property="roleDesc" column="ROLE\_DESC"></result>  
  
 <!--配置多对多的关系映射  
 ofType属性对应实体类，这里起别名了，不区分大小写  
 -->  
 <collection property="users" ofType="user">  
 <id property="id" column="id"></id>  
 <result property="username" column="username"></result>  
 <result property="birthday" column="birthday"></result>  
 <result property="address" column="address"></result>  
 <result property="sex" column="sex"></result>  
 </collection>  
</resultMap>

查询所有角色为例：可以同时得到角色所赋予的用户信息

<!--查询所有角色-->  
<select id="findAll" resultMap="roleMap">  
 SELECT r.id AS rid,r.ROLE\_NAME,r.ROLE\_DESC,u.\* FROM role r  
 LEFT OUTER JOIN user\_role ur ON r.ID = ur.RID LEFT OUTER JOIN USER u ON u.id = ur.UID;  
</select>

根据用户查询角色就是把SQL语句反过来就好，角色查用户左外连接查询，角色表为左表，查询角色表中的所有数据，以及用户表中相关连的数据

用户查角色，只需反过来，让用户表为左表/或者使用右外连接查询即可

<!--定义User的resultMap-->  
<resultMap id="userMap" type="user">  
 <id property="id" column="id"></id>  
 <result property="username" column="username"></result>  
 <result property="birthday" column="birthday"></result>  
 <result property="address" column="address"></result>  
 <result property="sex" column="sex"></result>  
  
 <!--配置多对多关系映射-->  
 <collection property="roles" ofType="role">  
 <id property="roleId" column="rid"></id>  
 <result property="roleName" column="ROLE\_NAME"></result>  
 <result property="roleDesc" column="ROLE\_DESC"></result>  
 </collection>  
</resultMap>

查询语句

<!--查询所有用户-->  
<select id="findAllUsers" resultMap="userMap">  
 SELECT u.\*, r.id AS rid,r.ROLE\_NAME,r.ROLE\_DESC FROM USER u  
 LEFT OUTER JOIN user\_role ur ON u.id = ur.UID LEFT OUTER JOIN role r ON ur.RID = r.ID;  
</select>

# Mybatis中的缓存

## 延迟加载：

延迟加载：就是在需要用到数据时才进行加载，不需要用到数据时就不加载数据。延迟加载也称懒加载.

好处：先从单表查询，需要时再从关联表去关联查询，大大提高数据库性能，因为查询单表要比关联查询多张表速

度要快。。

坏处：因为只有当需要用到数据时，才会进行数据库查询，这样在大批量数据查询时，因为查询工作也要消耗时

间，所以可能造成用户等待时间变长，造成用户体验下降。

### 问题：

在一对多中，比如：当我们有一个用户，他有100个账户，在查询用户的时候，要不要把关联额账户查出来？

在查询账户的时候，要不要把关联的用户查出来？

在查询用户时，用户下的账户信息应该是，什么时候使用，什么时候查询

在查询账户时，账户的所属用户信息应该随着账户查询时一同被查询出来

什么时延迟加载：

在真正使用数据的时候才发起查询，不用的时候不查询，按需加载也叫懒加载

什么时立即加载：

不管用不用，只要调用方法，马上发起查询

在对应的四种表关系中：一对多、多对一【这里就是列出所有的表关系，事实上mybatis中并没有多对一的关系】、一对一、多对多，看关联的关系是多还是一

一对多、多对多 ：关联关系对应的是多，通常情况下采用的是延迟加载

多对一、一对一 ：关联关系对应的是一，通常情况下我们采用立即加载

### 一对一/多对一关系实现延迟加载

账户查询关联的用户就是多对一，实际也是一对一{多个账户可以同时属于一个用户}

步骤一：

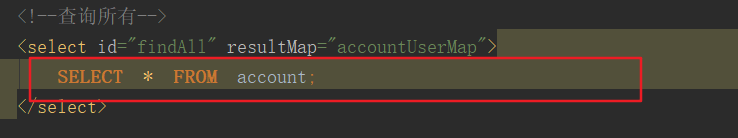
<association

标签内部，不需要在映射实体类和数据库列名，而是使用了一个select属性，属性指定根据id查询user对象的方法：



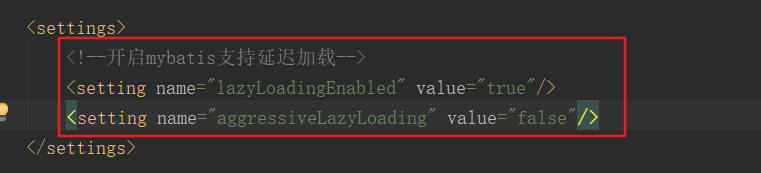
步骤二：

查询语句中不需要再写关联插叙，只需要单表查询语句，因为如果用连接查询的话很明显的就把两张表都查过了，就不能实现延迟加载

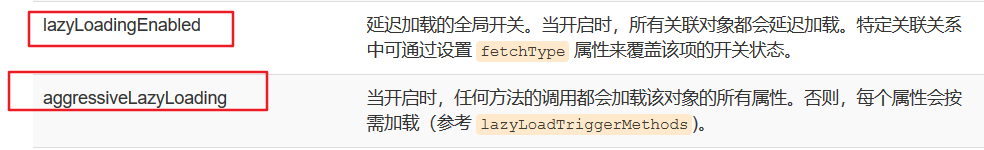


步骤三：

在mybatis主配置文件中开启支持延迟加载lazyLoadingEnabled，和aggressiveLazyLoading当开启时，任何方法的调用都会加载该对象的所有属性。 否则，每个属性会按需加载（参考 lazyLoadTriggerMethods)。这两个属性在mybatis官方网站，settings中查找



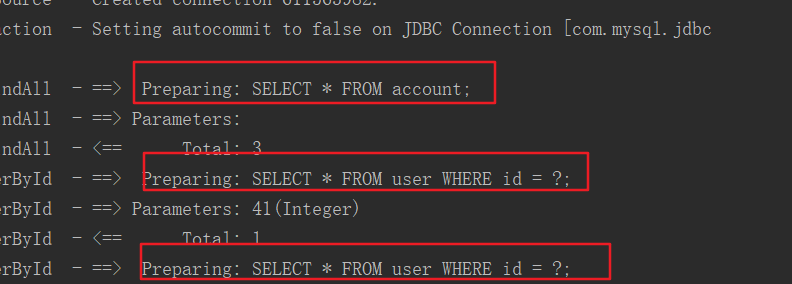




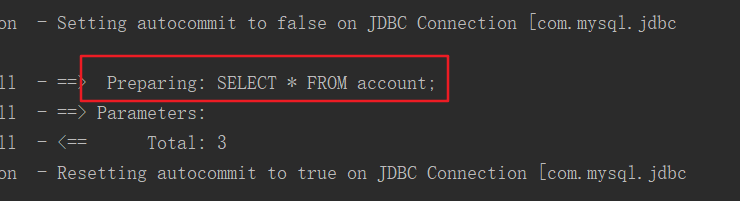
效果：

不配置延迟加载的情况：只要查询账户，就会执行查询关联的用户，进观我们已经没有再用连接查询的SQL语句，但是如下，我们在<association标签中使用了 select属性，指定了查询用户的方法，如果这个也没指定的话就不会查询用户信息，用户信息将为null但这并不是我们想要的结果





配置延迟加载后，则会按需加载，只有当使用到用户的时候，才会查询关联的用户，否则不查询



### 一对多关系实现延迟加载

根据用户查询用户下的账户---一对多关系{一个用户可以同时拥有多个账户}

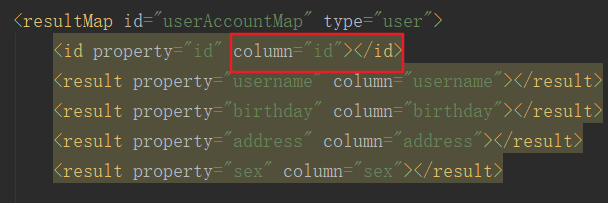
步骤一：

<collection 标签的配置，我们的多对一，一对一，配置一我们采用的是<association标签;

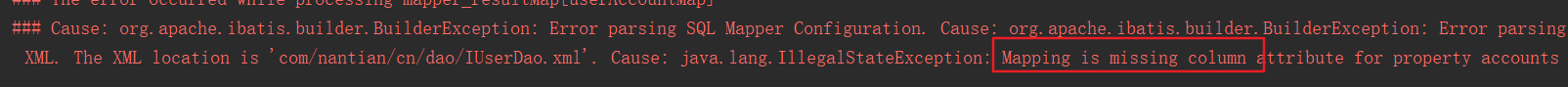
一对多，配置多我们采用的是<collection标签，与多对一，一对一的延迟加载一样，在标签内部不需要再配置数据库列名和实体类属性的关系映射，同样将采用select属性来指定通过哪个方法来查询关联的“多”的数据

<!--配置user对象中account集合的映射-->  
<collection property="accounts" ofType="account" column="id" select="com.nantian.cn.dao.IAccountDao.findAccountByUid">  
</collection>

注意：别忘记Colum属性的配置，这个是用来指定查询参数的，和我们（如下图中的配置有关）我们想通过用户id列来查询用户下的账户，所以在<collection标签中的Colum属性中配置，把它当作参数，根据用户用户id去查询账户



如果忘记配置Colum属性，就会出现如下的错误信息



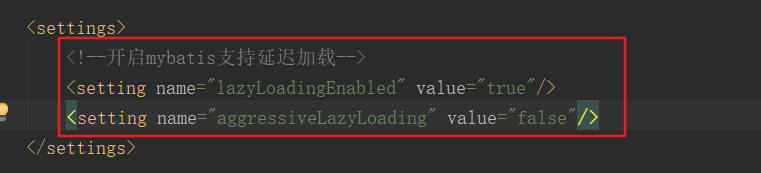
步骤二：

查询语句中不需要再写关联查询，只需要单表查询语句，因为如果用连接查询的话很明显的就把两张表都查过了，就不能实现延迟加载

<!--查询所有用户-->  
<select id="findAllUsers" resultMap="userAccountMap">  
 SELECT \* FROM USER;  
</select>

步骤三：

在mybatis主配置文件中开启支持延迟加载lazyLoadingEnabled，和aggressiveLazyLoading当开启时，任何方法的调用都会加载该对象的所有属性。 否则，每个属性会按需加载（参考 lazyLoadTriggerMethods)。这两个属性在mybatis官方网站，settings中查找



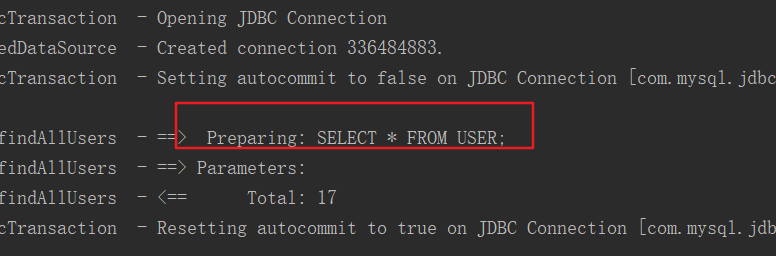
步骤四：

在AccountDao中添加根据用户id查询账户的方法，并在mapper映射文件中配置映射

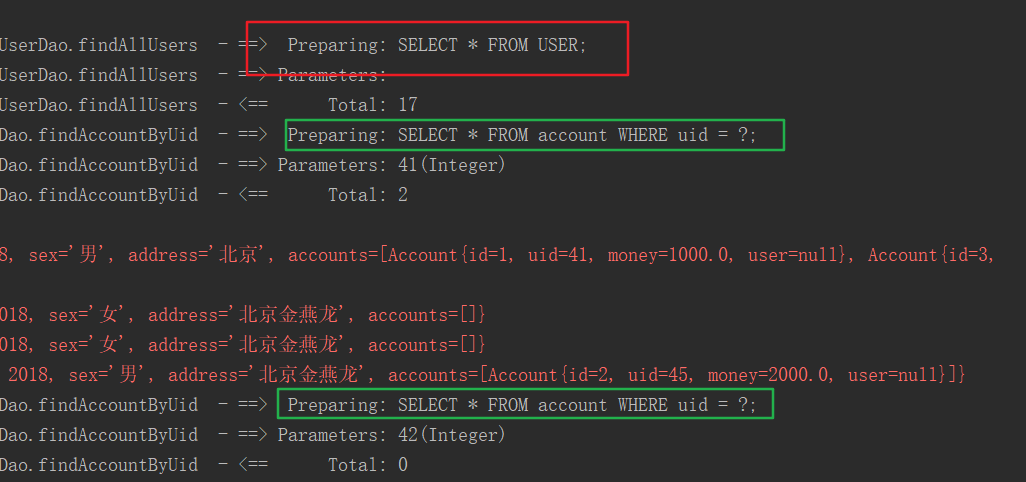
*/\*\*  
 \* 根据用户id查询账户  
 \** ***@param*** *uid  
 \** ***@return*** *\*/*List<Account> findAccountByUid(Integer uid);

<!--根据用户id查询账户列表-->  
<select id="findAccountByUid" resultType="account">  
 SELECT \* FROM account WHERE uid = #{uid};  
</select>

效果：当没有用到账户信息（比如遍历打印）的时候，就不会去查询账户信息



使用延迟加载后：



总结：延迟加载的思想都是，在用的时候去调用对方配置文件中的配置来实现延迟加载的功能

## 缓存

### 缓存的概念

存在于内存中的临时数据，比如我们在第一次查询出结果后，就可以把数据保存到缓存中去，下一次查询的时候就可以直接到缓存中获取，减少与数据库的交互并能快速的获取到数据

### 为什么使用缓存

减少和数据库的交互次数，提高执行效率

### 什么样的数据能使用缓存

适用于缓存的数据：

经常查询并且不经常改变的，数据的正确与否对最终结果影响不大的

不适用于缓存的数据：

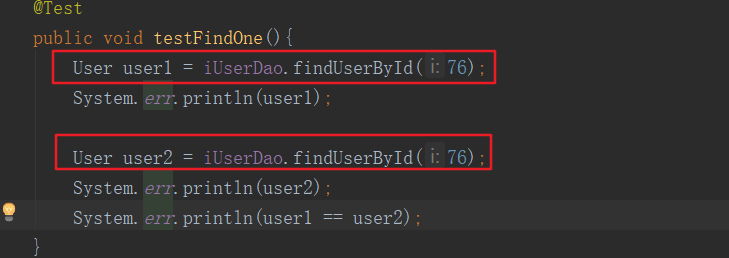
经常改变的数据，数据的正确与否对最终结果影响很大的

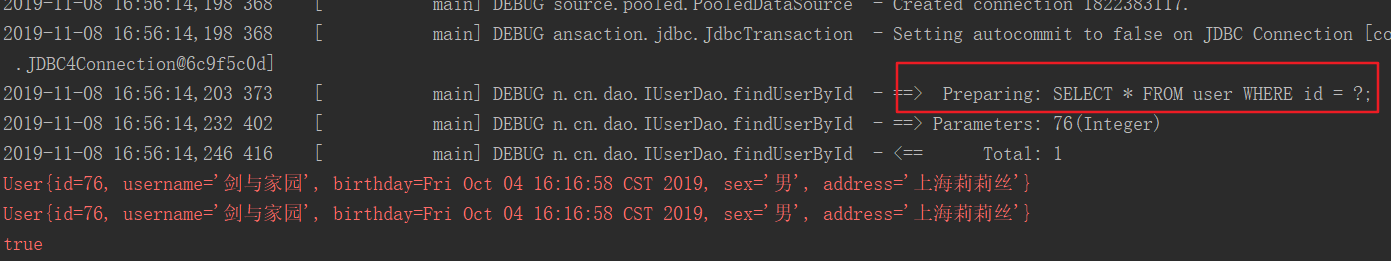
例如：商品的库存、银行的汇率、股市的牌价

### Mybatis中的一级缓存：

它指的是mybatis中SqlSession对象的缓存，当我们执行查询之后，查询结果会同时存入到SqlSession为我们提供的一块区域中，该区域的结构是一个map，当我们再次查询同样的数据，mybatis会先去SqlSession中查询是否有，有的话直接拿出来用，当SqlSession对象消失时，mybatis的一级缓存也就消失了

#### 测试一级缓存



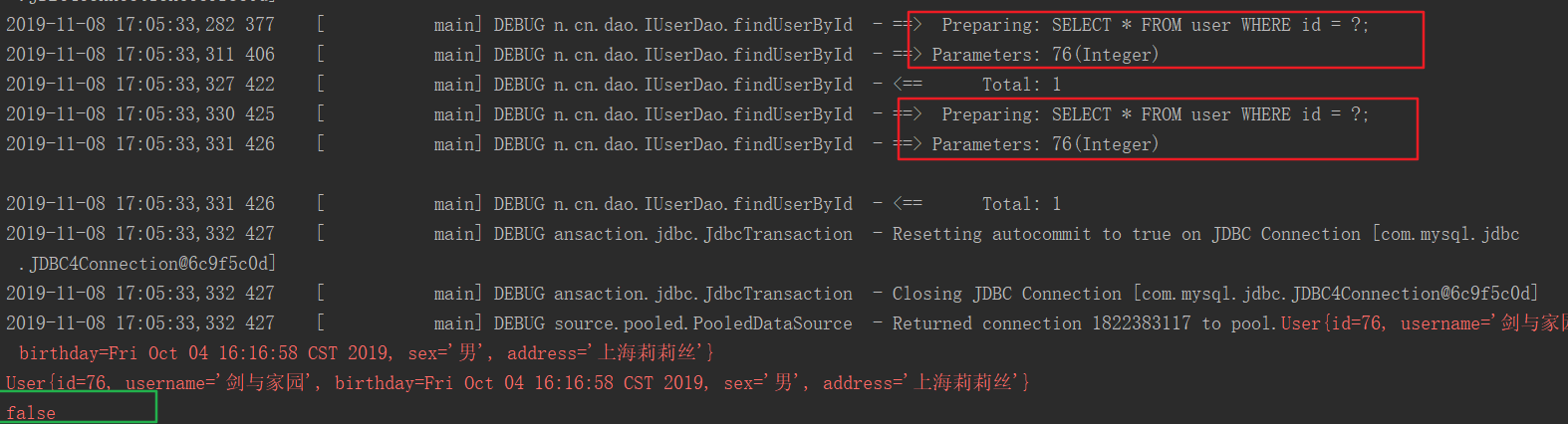


我们可以看到查询同一条数据，孙然调用了两次方法，但是指执行了一条是SQL语句，也就是说只想数据库中做了一次查询，而且两次查询结果是同一个对象

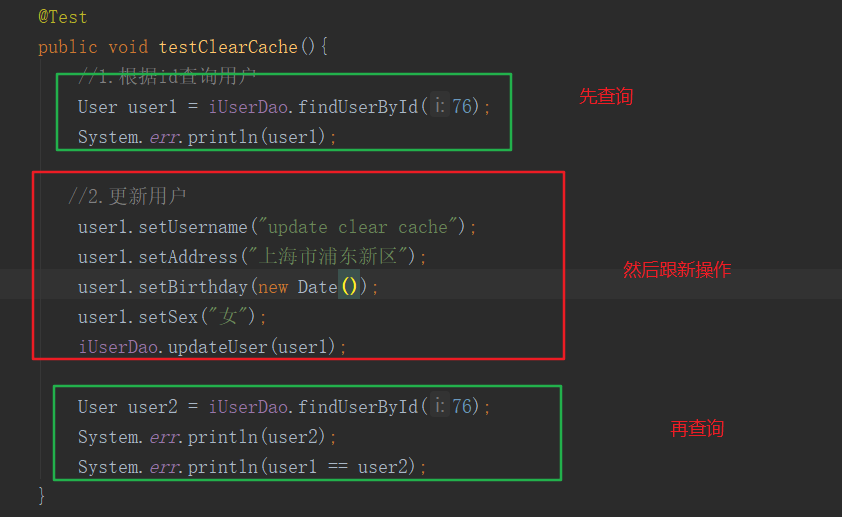
通过关闭SqlSession重新开启，或者清空SqlSession缓存来验证mybatis的一级缓存是SqlSession级别的缓存



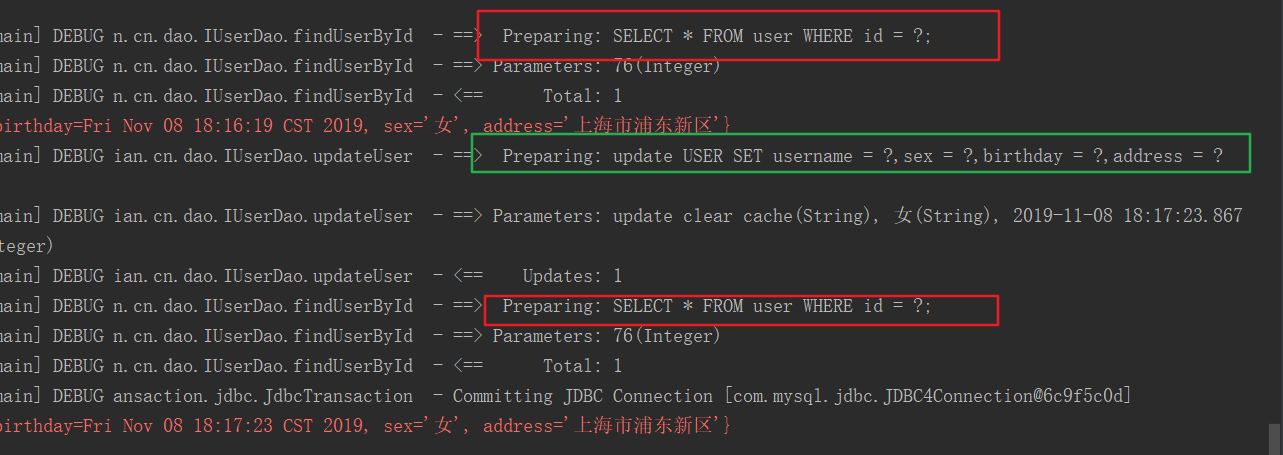
可以看到，当关闭SqlSessin重新开启，或者清空SqlSession中的缓存后，同一条查询执行了两次Sql语句，也就是向数据库中查询了两次，也就说明SqlSession关闭后，Mybatis的一级缓存也就消失了



#### 触发清空一级缓存的情况



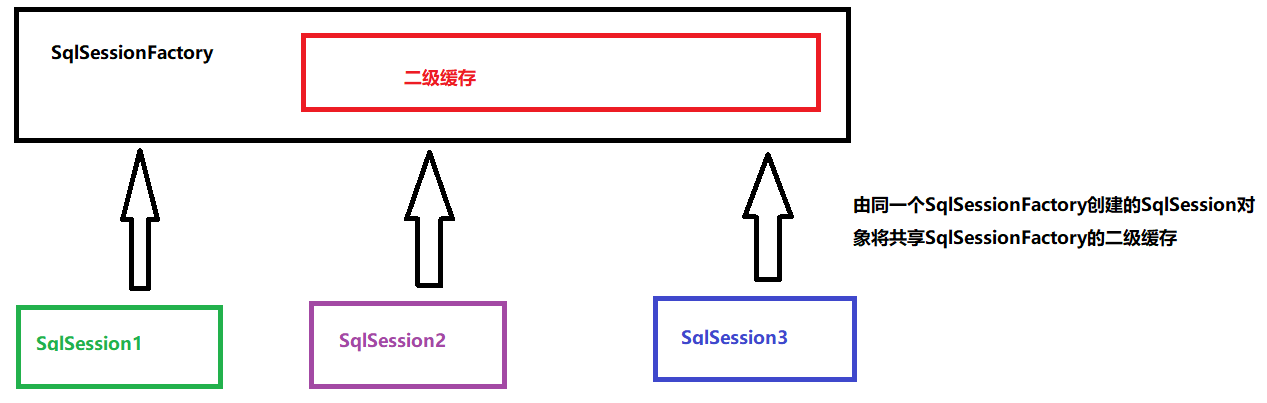
通过查看日志，可以看到，当通过id查询一条数据后，进行了更新操作，再用同一个id查询同一条数据，执行了两次查询，说明此时的缓存已经失效了



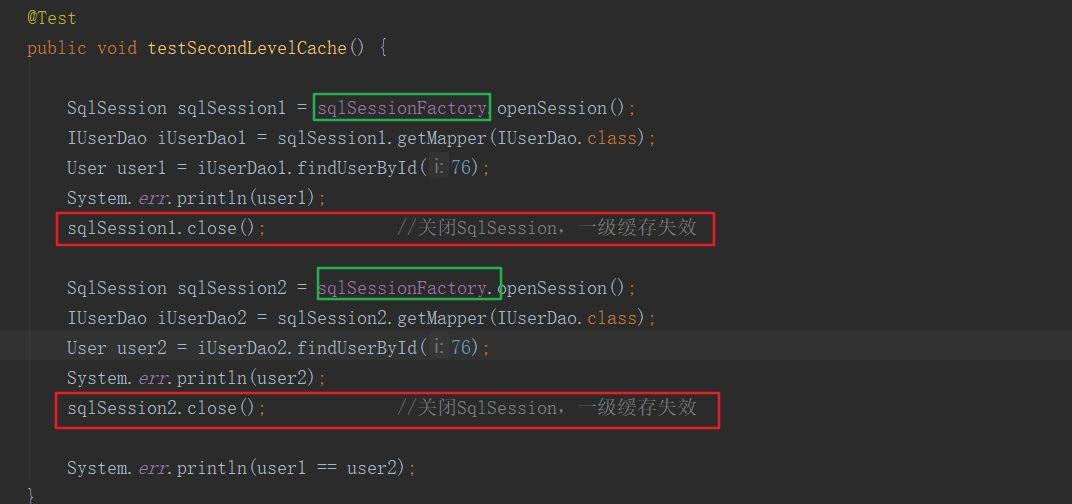
一级缓存是SqlSession范围（级别）的缓存，当调用SqlSession的修改、添加、删除、commit()、close()（为什么说是SqlSession的了，因为mybatis真正操作数据库就是通过SqlSession中的方法来操作数据的）等方法时，就会清空一级缓存，这样做的目的是为了让缓存中存储的是最新的数据，避免脏读；还有就是mybatis在整合spring使用的时候，一级缓存也会失效，【单独使用mybatis时在sqlsession范围内，一级缓存是生效的】，因为正式开发中，是将mybatis和spring整合开发的，事务控制在service中，一个service方法中包括很多mapper方法调用，如执行用两次service调用查询相同的数据信息，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

### Mybatis中的二级缓存

它指的是mybais中SqlSessionFactory对象的缓存，由同一个SqlSessionFactory对象创建的SqlSession共享其缓存

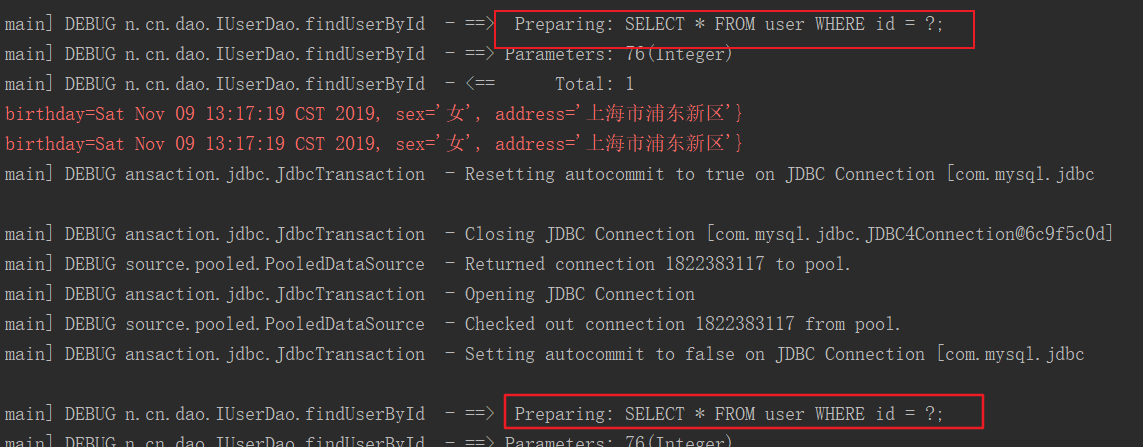


#### 测试二级缓存



创建二级缓存的场景，由同一个SqlSessionFactory对象创建SqlSession对象，

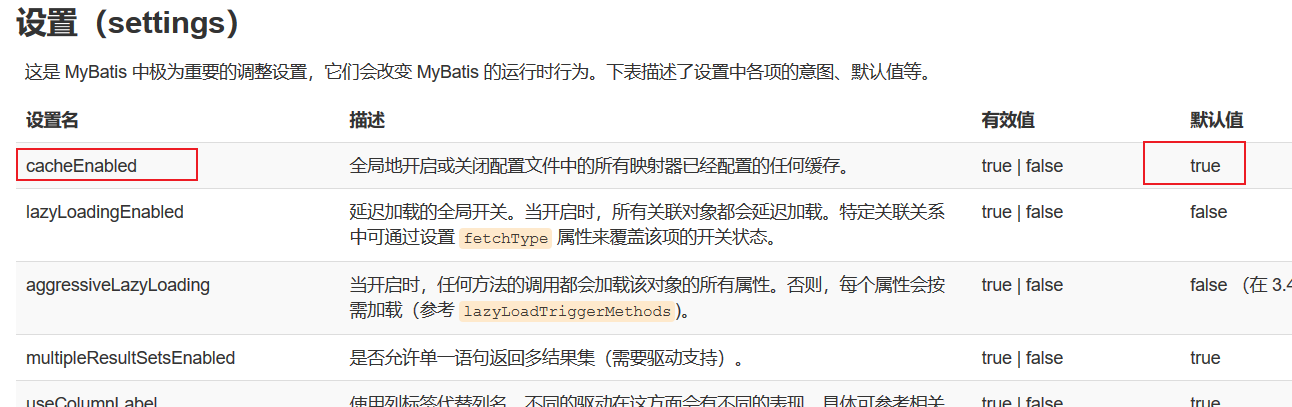
效果，执行结果如下：发现，该查询几次，还是查询几次，就说明，根本没用二级缓存，如果需要使用mybatis的二级缓存，我们要通过配置二级缓存的支持



#### 二级缓存的使用步骤

第一步：让Mybatis框架支持而建缓存（在mybatis主配置文件SqlMapConfig.xml中配置）

Mybatis所有的xml配置都可以到官方文档中找到,要学会使用官方文档，spring也是一样



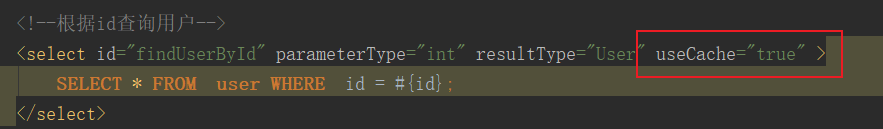
如下配置，这一步是在主配置文件中配置的，这一步不配置也可以，因为默认就是true

<settings>  
 <setting name="cacheEnabled" value="true"/>  
</settings>

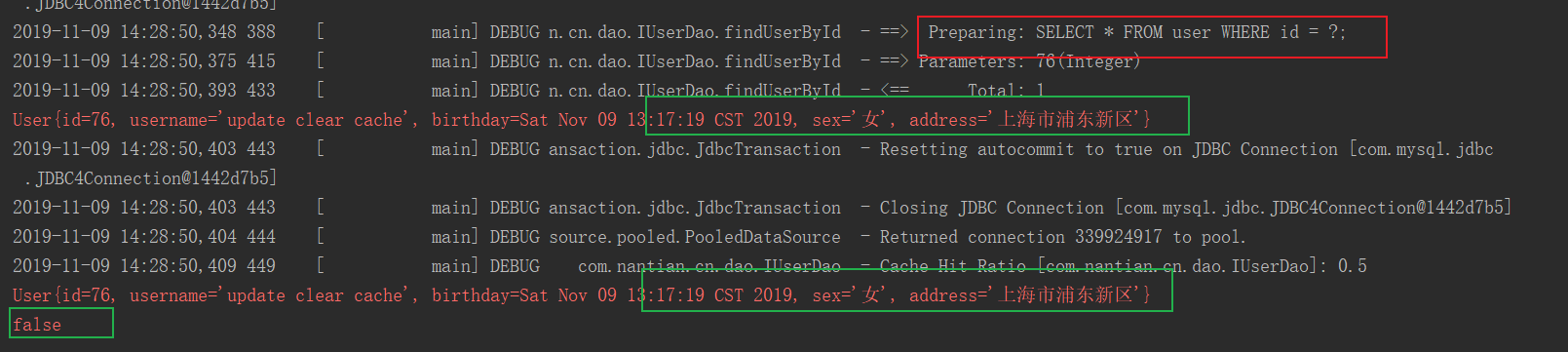
第二步：让当前的映射配置文件支持二级缓存（在映射配置文件中配置）

<!--开启user支持二级缓存-->  
<cache/>

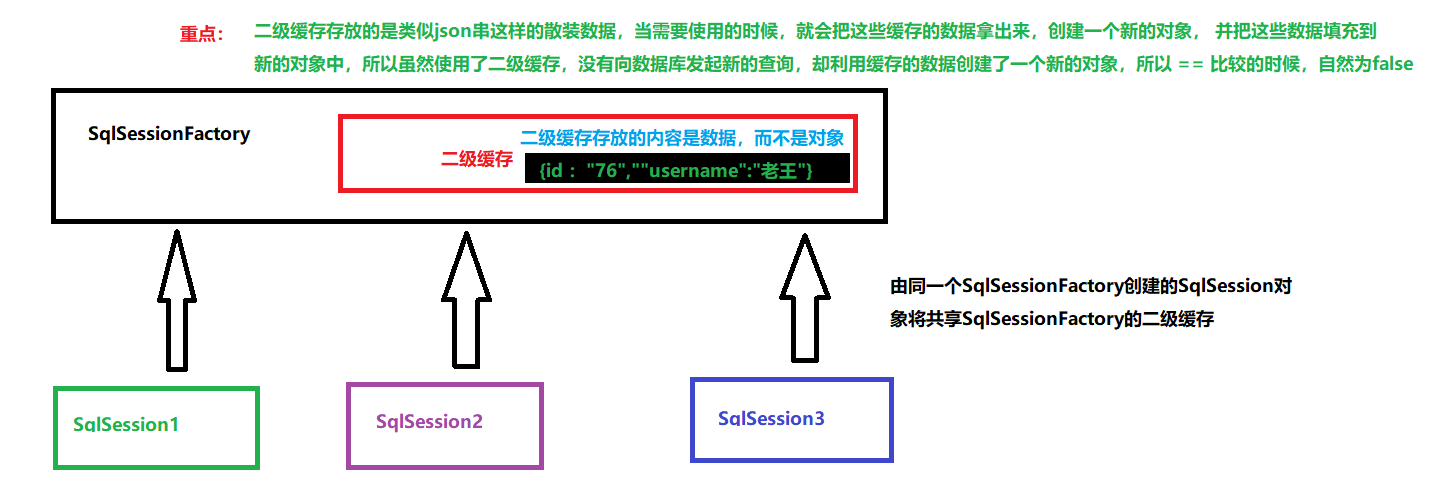
第三步：让当前操作支持二级缓存（在<select 标签中进行配置）



配置完成后的执行效果：可以看到查询同一个id，索然查询了两次，但是只执行了一条sql语句，也就说明只向数据库中查询了一次，第二次就是从缓存中获取的



值得注意的是：在一级缓存中，我们第二次查询，从缓存中获取到的数据和与第一次从数据库中查询出来的就是一个对象，所以用 == 号比较的时候，其结构为true而二级缓存，其结果为false，是因为，二级缓存存放的内容是数据，而不是对象,



# Mybatis注解开发

Mybatis在注解开发中，针对crud提供了四个注解，

@Select

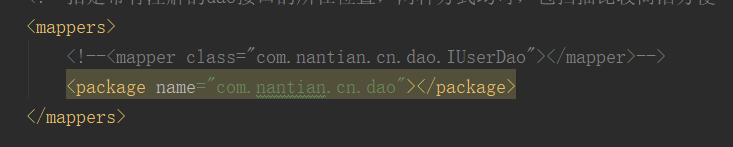
@Insert

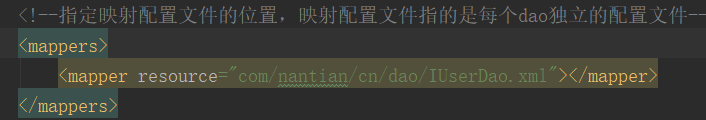
@Update

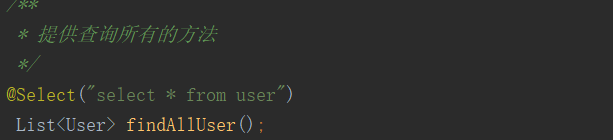
@Delete

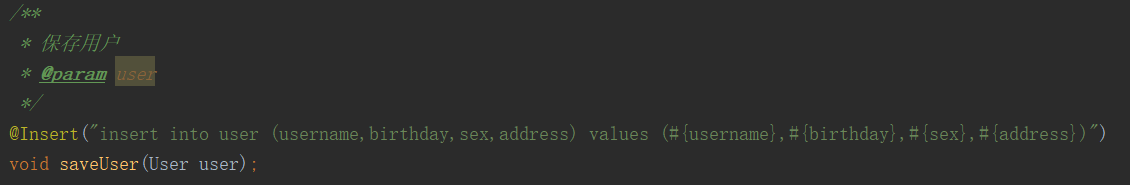
注解中的value属性，写的就是我们的SQL语句

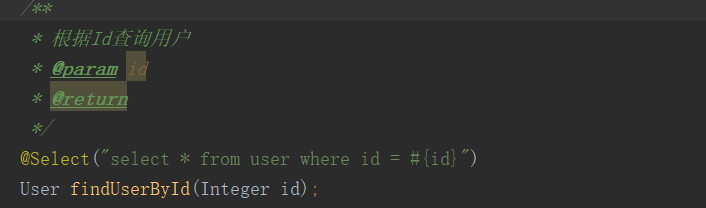
细节：我们在注解开发的时候，就不能有mapper映射配置文件，如果有映射配置文件，就算我们使用如下所示，<package标签引入dao,而不是使用<resource标签引入映射配置文件，程序也无法正常运行，也就是当使用注解开发的时候，就会不能再有mapper映射配置文件了。

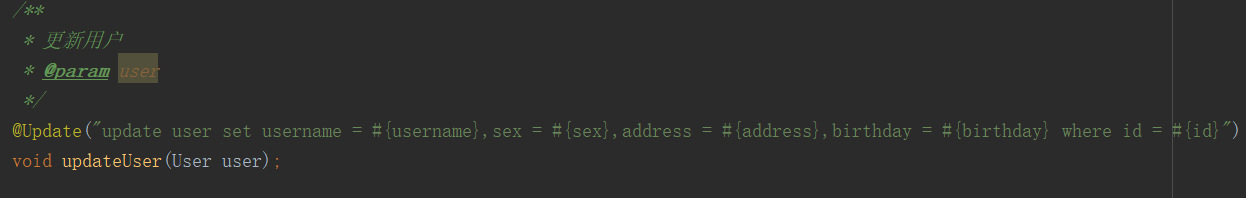


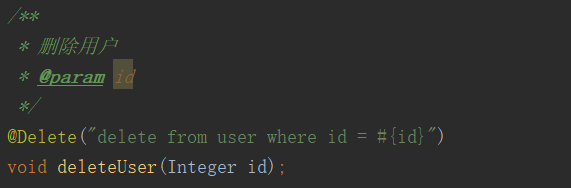






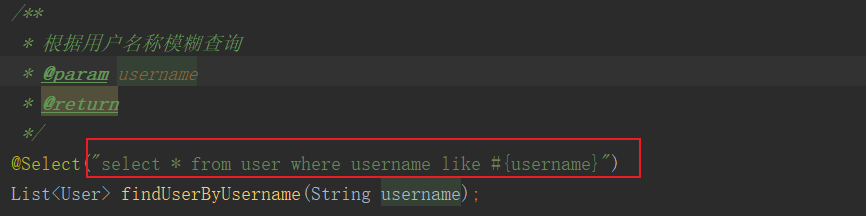


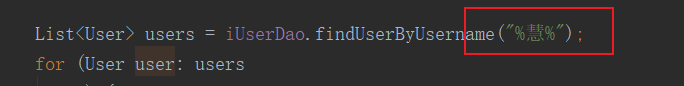




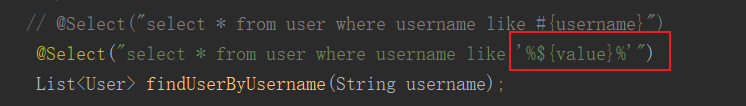
根据名称模糊查询：

第一种情况：在Sql语句中使用#占位符，没有百分号，此时传递参数的时候，需要添加%%通配符





第二种情况：在Sql语句中使用'%${value}%',value属性是固定写法，这两种写法的区别就在于，#{}采用的是参数占位符，'%${value}%'采用的是字符串拼接，占位符是可以防止sql注入的，而字符串拼接则不能防止SQL注入



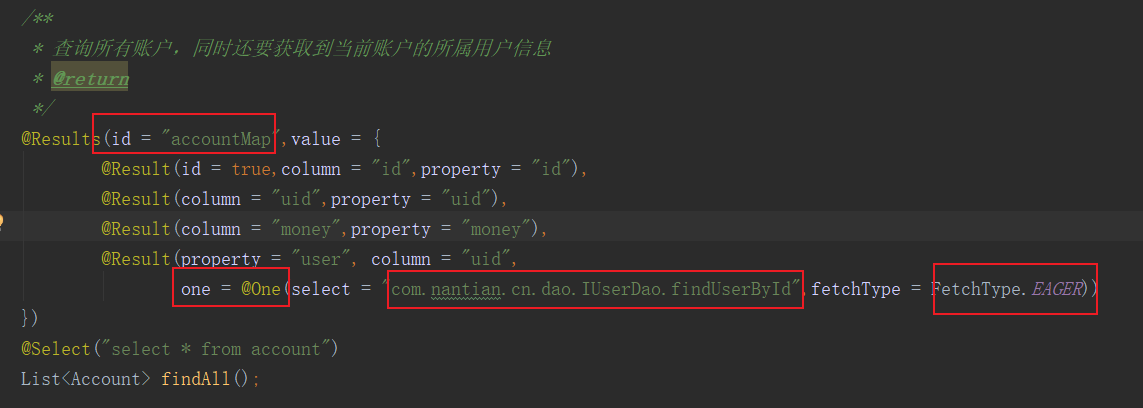
传递参数的时候就不需要在传递%%通配符了



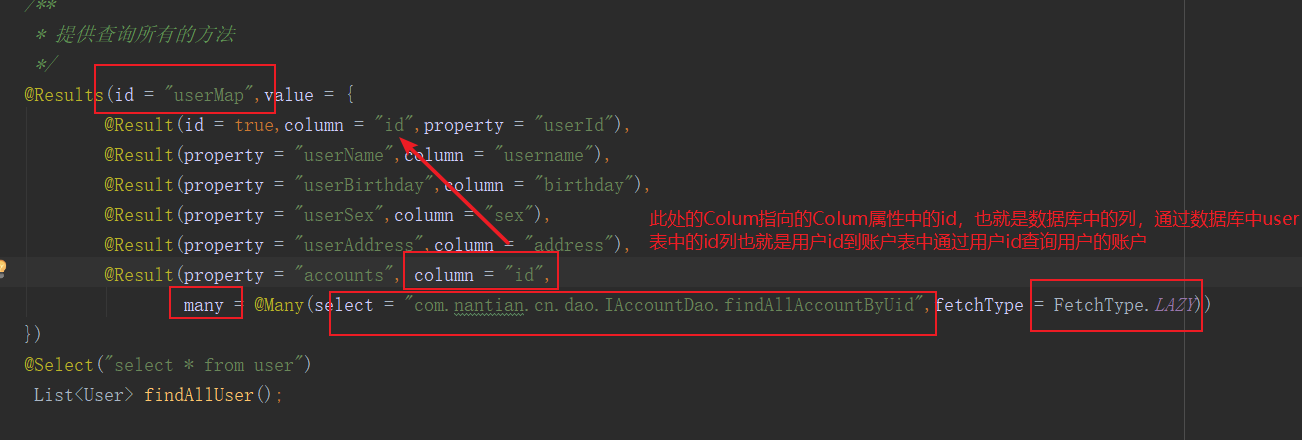
## 注解开发多对一/一对一关系查询

账户和用户，根据用户，查询账户关联的用户，就是多对一/一对一

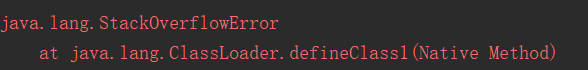
如下图的配置其实使用注解开发和使用xml开发原理基本是一样的，@Results注解中的id属性，给@Resluts起个名字是个唯一标识，相当于xml配置中resultMap的id，@Result注解，就相当于xml配置中的<result标签，里面也有id属性，如果当前列是id，给其为true值，one属性，就相当于xml配置中<association标签来配置一，此时Colum表示根据这个参数来查象关联的数据，和xml配置是一样的道理，one属性等于一个@One注解，@One注解中有select属性，这是查询关联数据方法的全限定名，fetchType属性用于指定加载的实际，有三种取值，default默认EAGER立即加载， 一般多对一或者一对一通常采用立即加载，LAZY延迟加载，一般一对多通常采用延迟加载



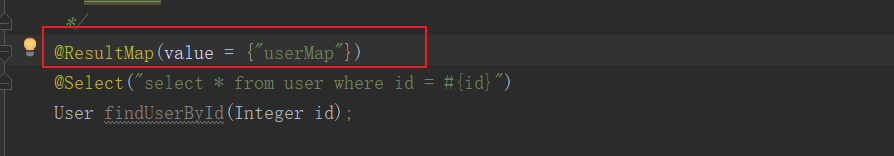
## 注解开发一对多关系查询

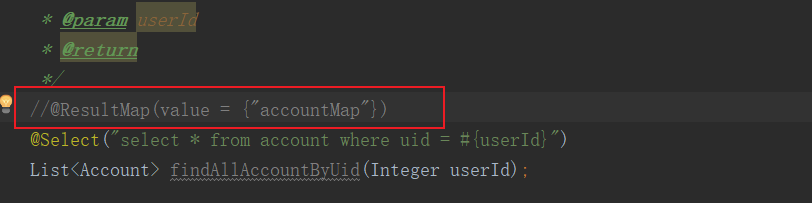


指的注意的是accountMap中也配置了关联查询的情况下，这里就不能在使用@ResultMap,不然，又会去查询用户，就进入死循环，导致stack内存溢出，



分析：





指定resultMap不是必须的，查询user必须指定是因为实体类与数据库列名不一致，不指定无法封装实体类，但accout实体类与数据库列名对应，可以不指定，如果两个Dao的Resultmap中都有彼此，一对多或者多对一的关联查询，数据库列名与实体类又不一致的情况，只能分开来测试，也就是测试一对多的时候，账户Dao中，不要配置one属性的关联查询，测试多对一的时候，用户Dao中不要又many属性的关联配置，就不会出现stack内存溢出了，还有就是IDEA中通过鼠标右键点击方法copyReference复制方法的全限定名的时候，报名和方法名之间用的是#号连接，一定要改成”.”

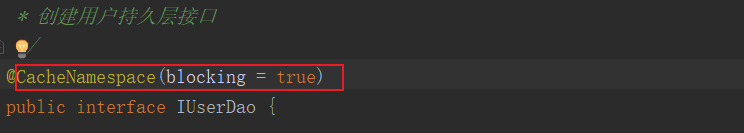
## Mybatis注解开发使用二级缓存

第一步：和xml配置一样

如下配置，这一步是在主配置文件中配置的，这一步不配置也可以，因为默认就是true

<settings>  
 <setting name="cacheEnabled" value="true"/>  
</settings>

第二步：在Dao接口上使用@CacheNamespace注解，里面有一个属性blocking默认为false表示不使用二级缓存，把值设置为true，就可以使用二级缓存了



***完结End...***