## 一：mybatis概述

Mybatis是一个持久层框架，采用的是Java语言写的，封装了很多细节，使开发者只需要关注sql语句本身就好了，而无需关注什么各种驱动注册，连接创建等过程！

Mybatis使用Orm思想解决了实体和数据库映射的问题，对jdbc进行了封装！

Orm: object relational mapping 即对象关系映射

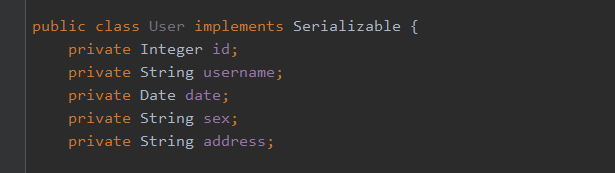
简单说就是将数据库中的字段和实体类中的属性对应起来！我们通过实体类就可以直接操作表！

## 二：mybatis入门案例（环境搭建！）

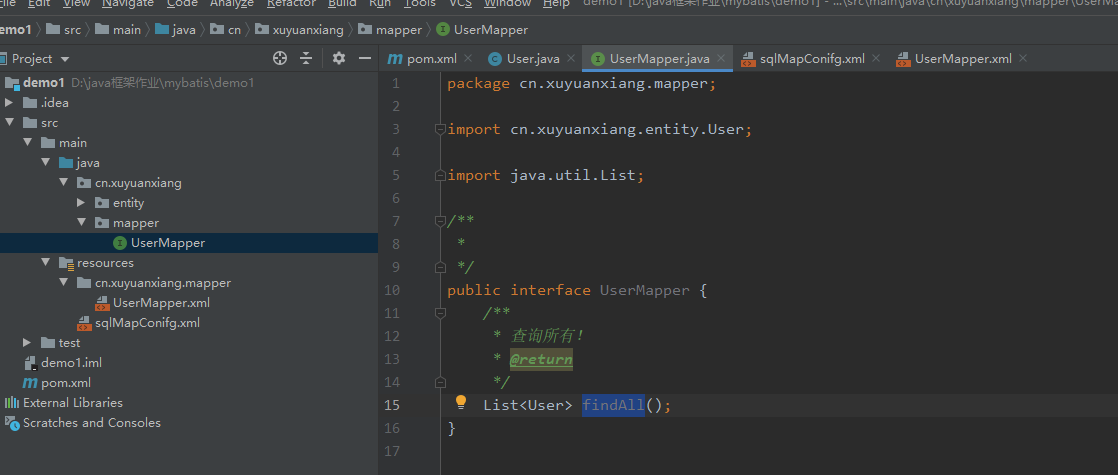
首先创建maven项目导入jar包的坐标！



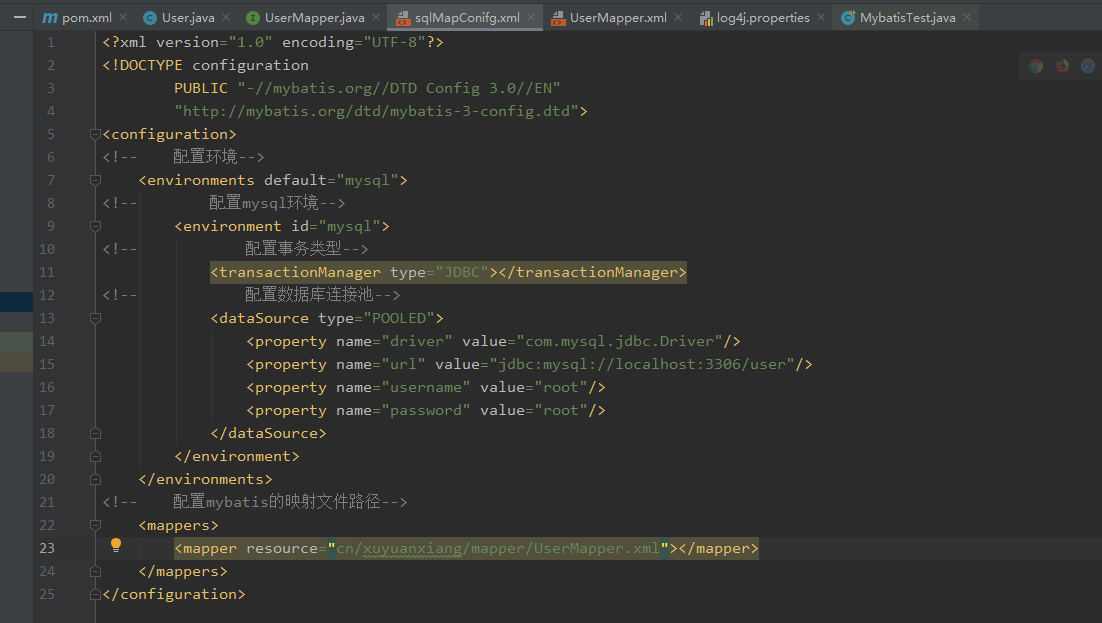
创建一个user实体类！用来封装数据库中的信息，这里的属性名字和数据库中表的字段名字相同！



创建usermapper的接口，内部写一个抽象方法查询所有



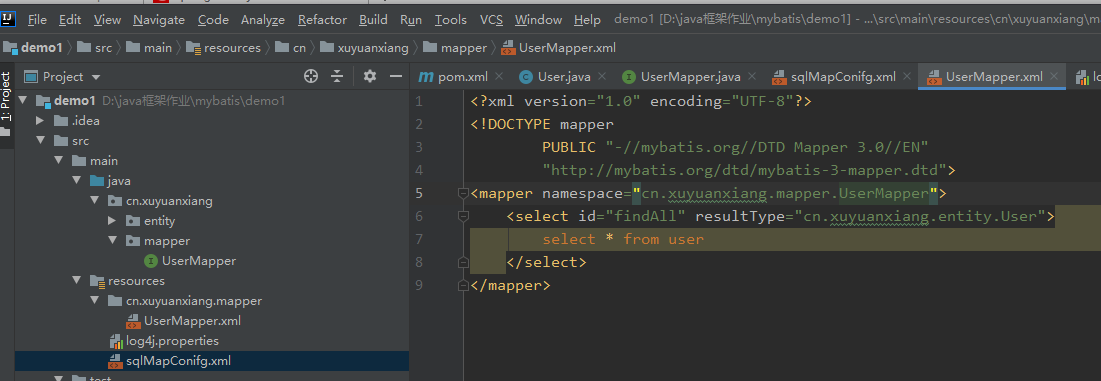
根目录下创建一个mybatis的配置文件，做出连接池，数据库类型，mapper映射文件等配置



创建mapper映射配置文件，这里的路径需要和java中usermapper接口路径对应！（这个设置再mybatis配置文件中可以取消！）

里面的<mapper>namespace指定了映射到usermapper接口，<select>中id指定了方法名称以及sql语句！

至此我们配置完成，以后无需再写usermapper的实现类！



## 三：mybatis入门案例（xml配置文件式）

在上一节的基础上创建一个测试类：

首先读取配置文件返回输入流：

然后创建一个工厂构建者：

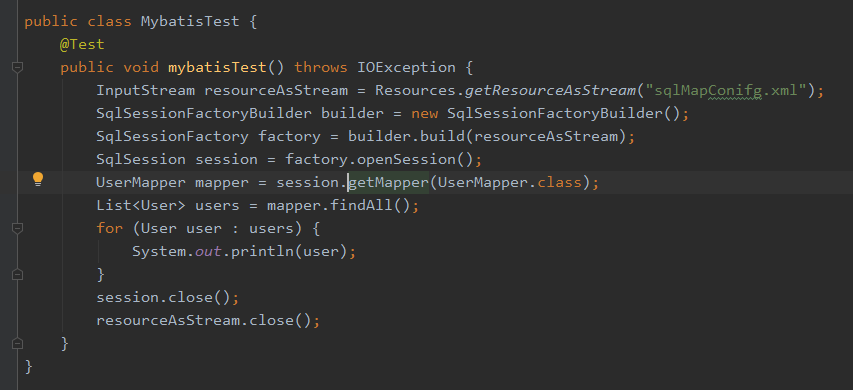
用工厂构建者带着我们的输入流创建出sqlsession工厂！

创建出的工厂可以开启一个session会话

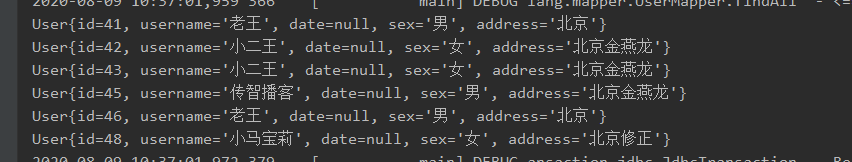
开启的session会话在获取对应的接口类class文件，通过反射加代理的方式进行代理对象的创建（这里就创建了类似接口实现类的代理对象，所以我们不需要再创建实现类了！）

创建出的代理对象完成我们的查询操作！

最后释放资源

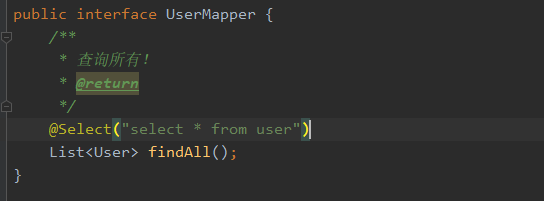


运行测试程序，结果正常！

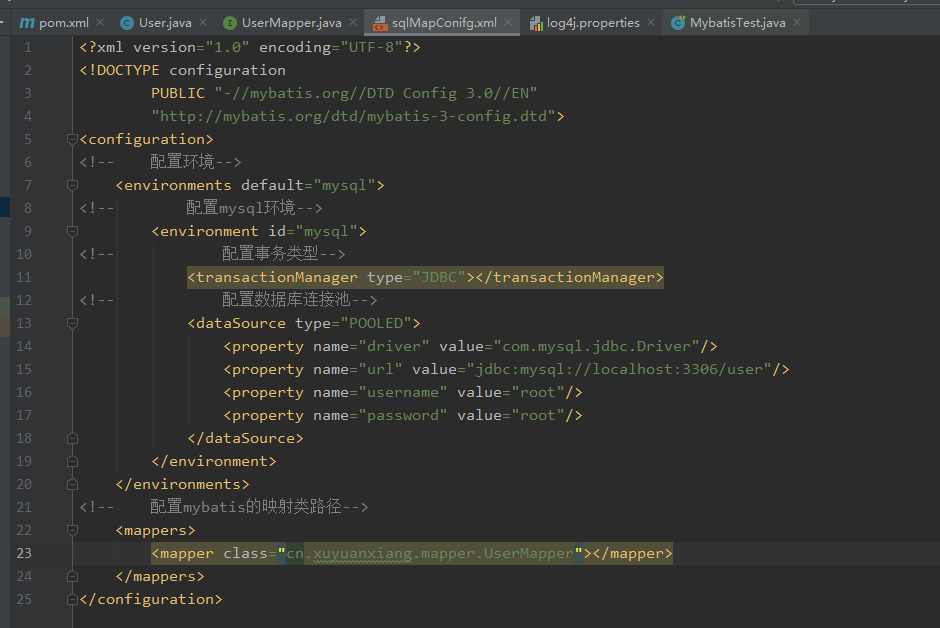


## 四：mybatis入门案例（注解式）

首先在mapper接口中给方法添加上@Select注解，内部value属性指定sql语句



修改mybatis配置文件，在<mapper>标签中将resource改成class，指定好我们的mapper接口路径！最后实现输出结果与原先一致！

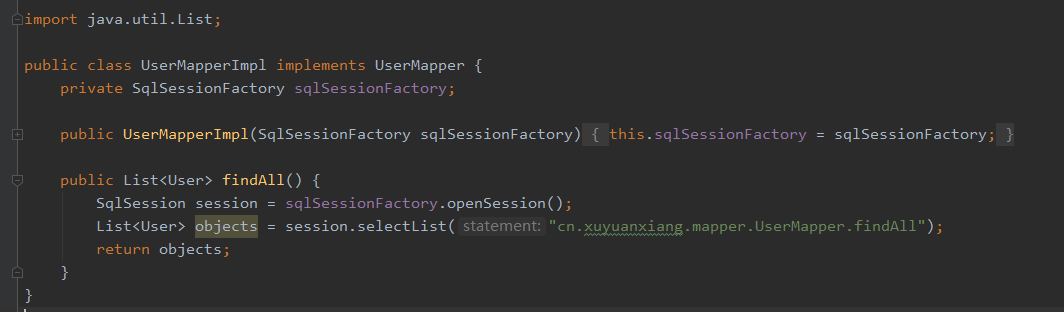


## 五：mybatis入门案例（接口实现类式）

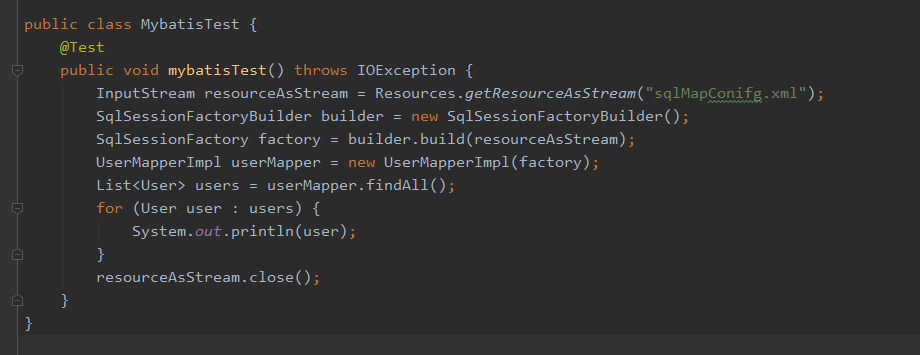
当然我们也可以不采用mybatis的动态代理生成dao实现类，而是自己编写实现类进行操作（方便理解源码，实际作用不大！）

首先创建一个mapper接口的实现类！里面写入我们需要的sqlsessionfactory。

在findall方法中选择selectlist方法进行查询（参数写入我们的接口方法名（在mapper.xmlp配置文件中已经将此路径录入了，这里直接引用就可以实现方法的调用。））

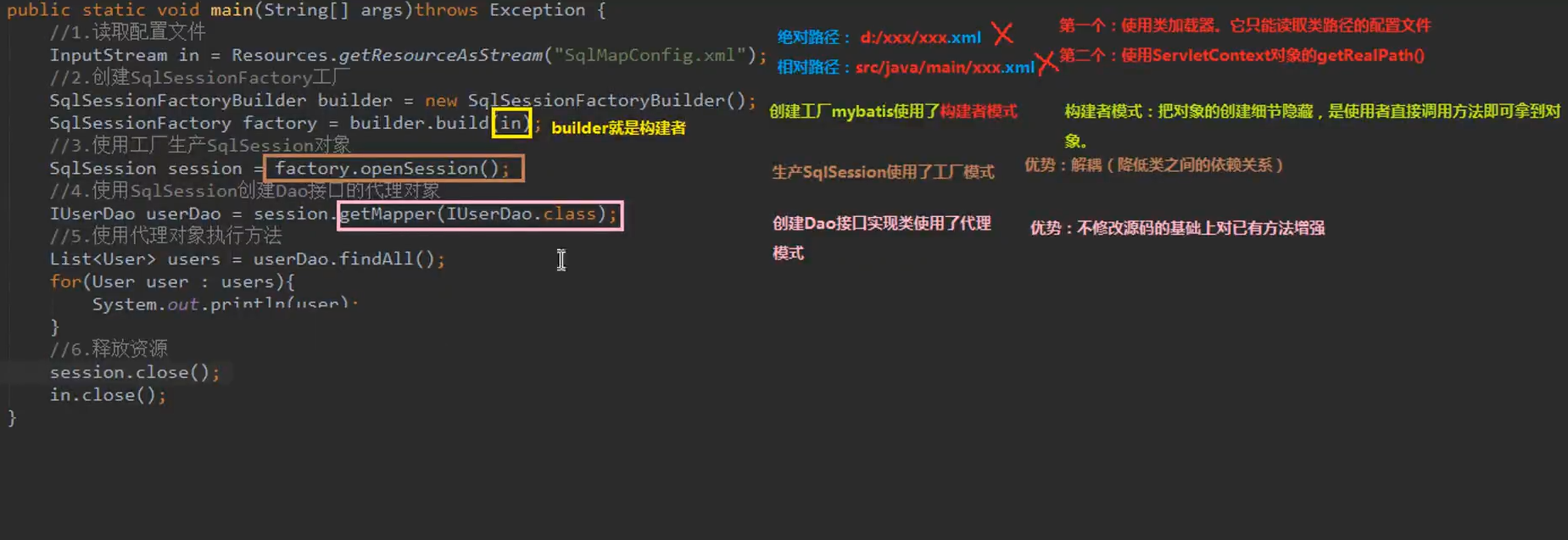


在测试类修改代码，我们不需要mybatis帮我们创建代理对象，所以直接创建实现类，返回查询结果。正确！



## 六：mybatis入门案例设计模式分析

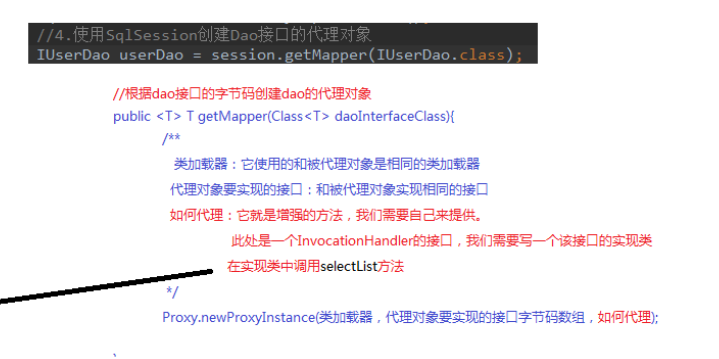
如下就是我们在完成mybatis中所涉及到的设计模式。之所以还有工厂，代理等配置，是为了提高灵活性。而在springboot等更高级的框架中就连这些也自动配置了！



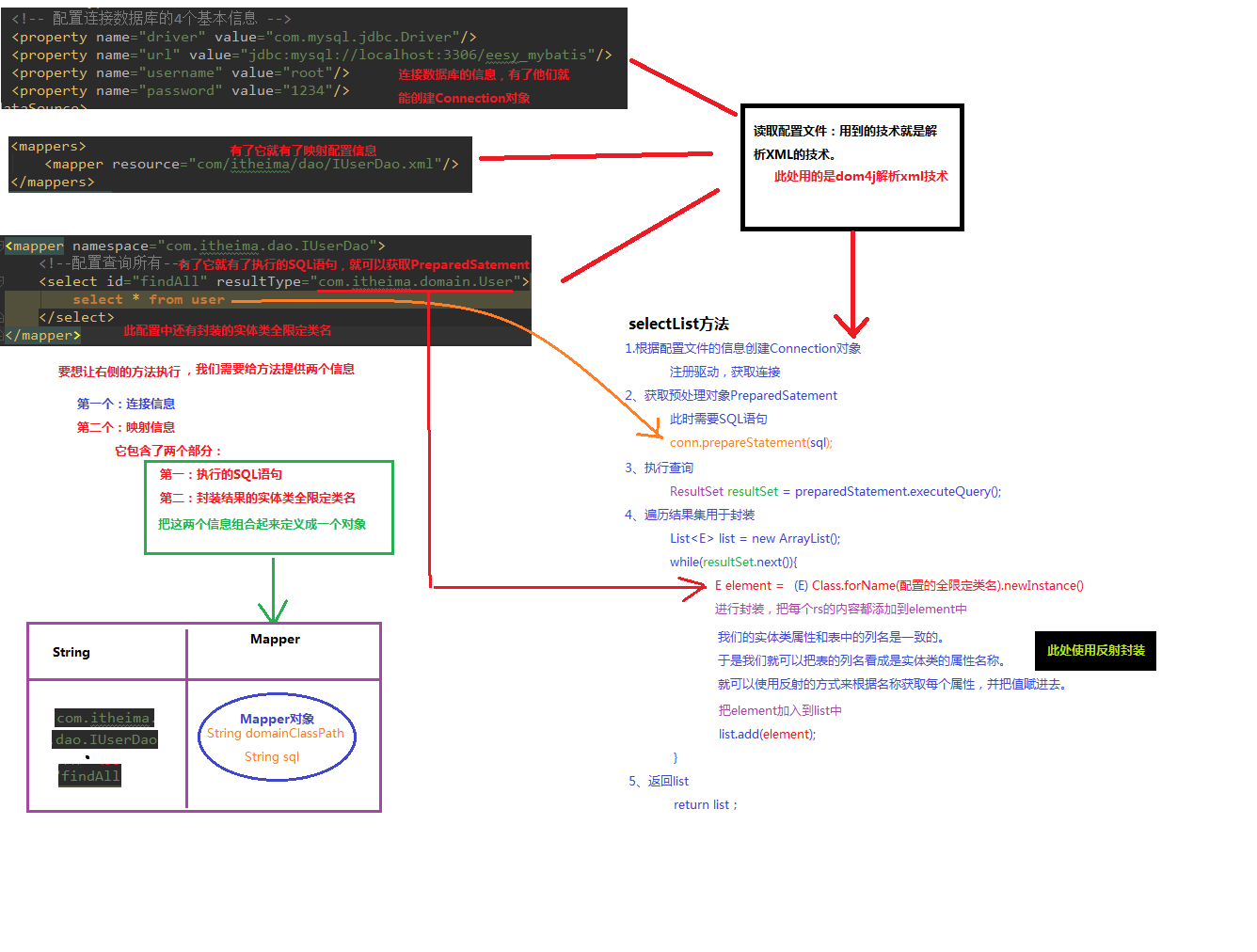
## 七：mybatis自定义分析

现在我们来对mybatis的动态代理以及执行sql语句进行分析：

根据以上的铺垫，我们可以预测，mybatis底层的动态代理其实就是将userdao接口产生代理对象，里面的方法进行了增强（调用了selectList方法！）

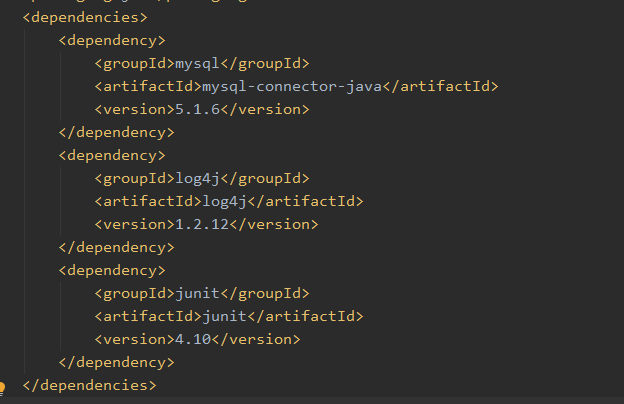


在我们调用的selectlist方法中，需要获取两个信息才能够完成查询操作，一个是连接信息（也就是connect，这里需要datdasource，已经在配置文件中有读取到）。另一个是映射信息（也就是sql语句以及返回值类型！），而映射信息在底层的封装原理是一个map集合，键名就是我们的全类名和方法名的组合，值就是mapper对象（里面封装了返回值类型和sql语句）。这样我们selectlist方法运行的时候就可以通过连接信息创建connection，通过映射信息获取sql语句执行，再获取返回值对象类型将查询到的数据进行封装！

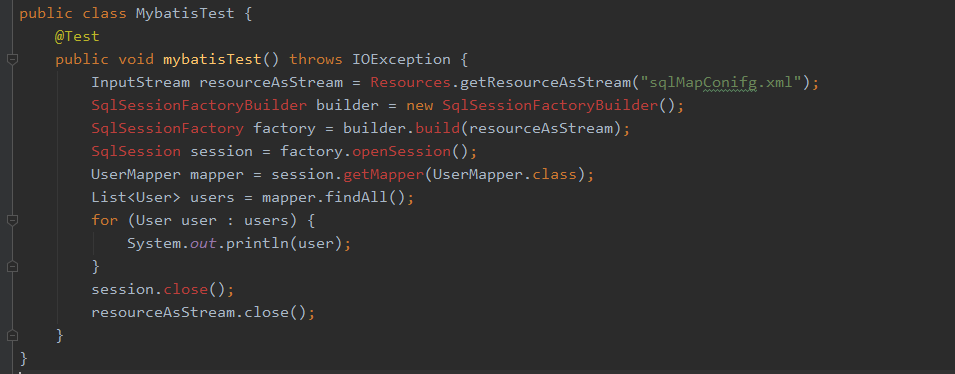


## 八：自定义mybatis编码（移除mybatis依赖，根据测试类，创建缺少的接口和类）

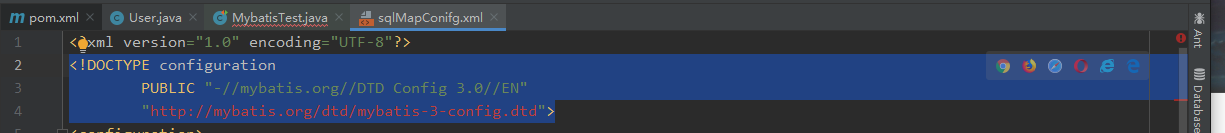
创建一个项目，其他与之前相同，只不过是我们删除对于mybatis的依赖！



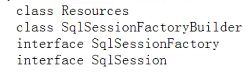
删除依赖以后，我们发现缺失了如下类和接口



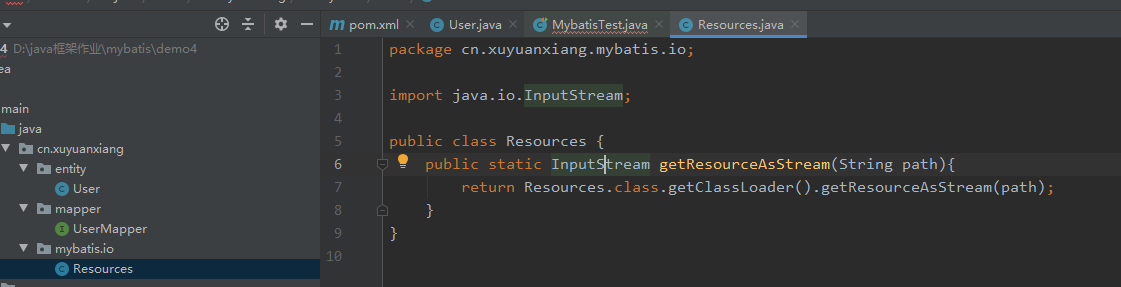
同时，配置文件中引入的mybatis相关也失效了，删除掉！



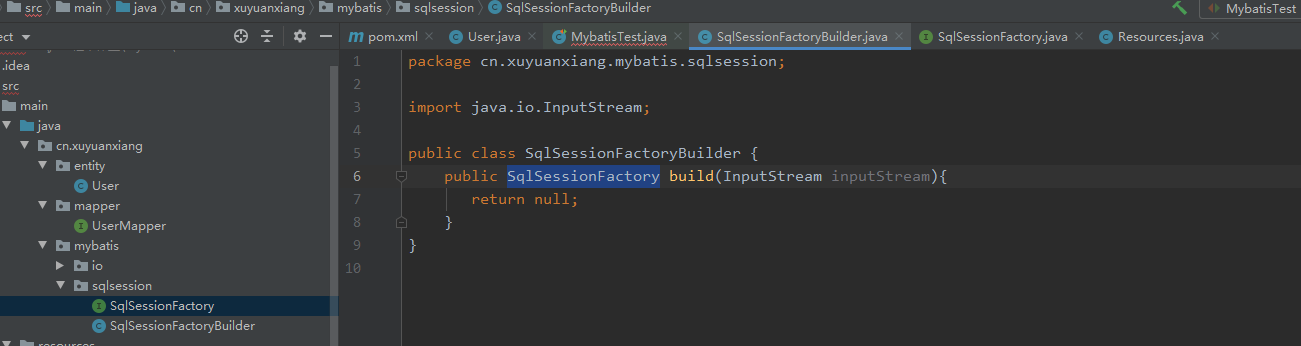
可以看到，如果需要我们的测试方法正常运行，必须创建如下类和接口！



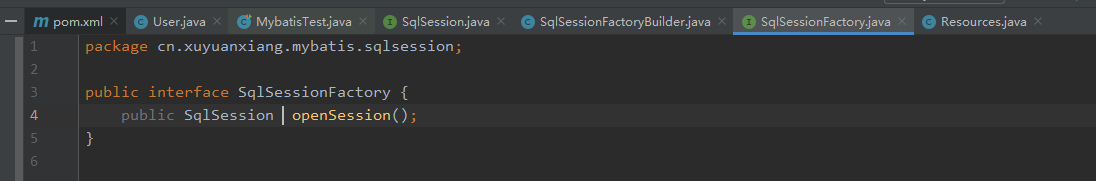
首先创建我们的Resources类，创建一个静态方法用来读取配置文件同时返回一个输入流！



创建一个SqlSessionFactoryBuilder类，里面包涵一个bulid方法，参数为inputstream流，返回值是sqlsessionfactory接口！里面应该包涵解析输入流的过程，这里暂时忽略！

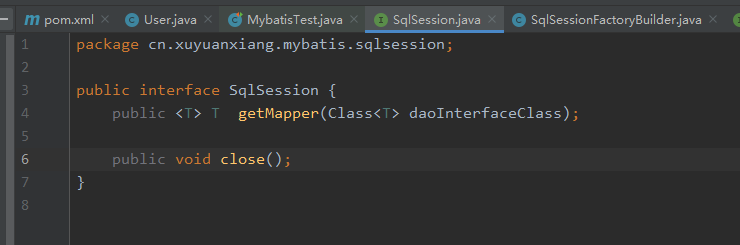


创建sqlsessionfactory接口，包涵一个openssion抽象方法！返回sqlsession接口

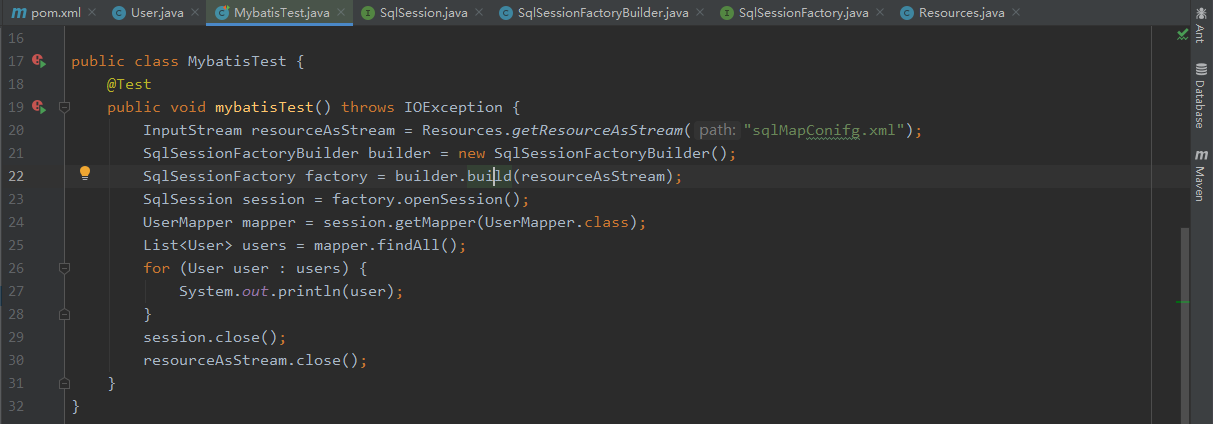


创建sqlsession接口，里面包涵一个getmapper方法，参数需要Class文件（由于未知类型，所以采用<T>泛型），返回值也是T泛型。里面将来实现动态代理，将我们的usermapper接口生成代理对象。

再添加一个close抽象方法。

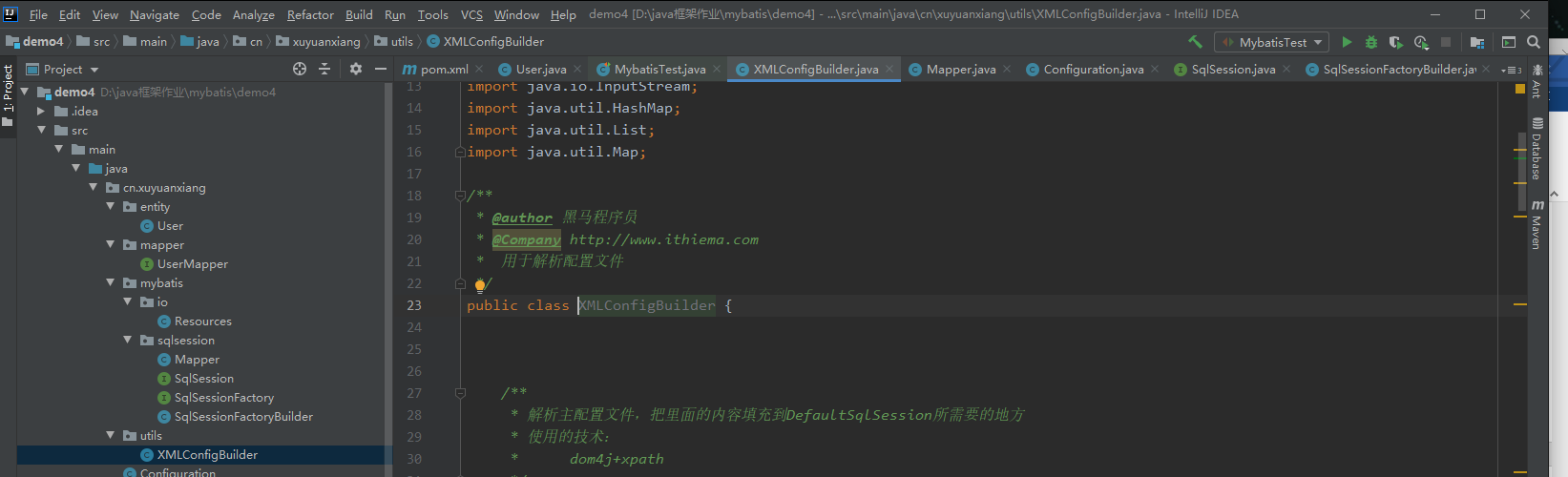


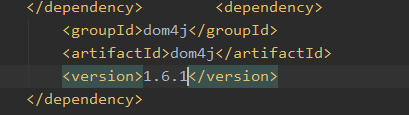
此刻发现，我们的Test方法已经不再编译报错了，当然它依然不能运行。我们后面对其中的方法进行完善！



## 九：自定义mybatis编码（解析xml的工具类介绍）

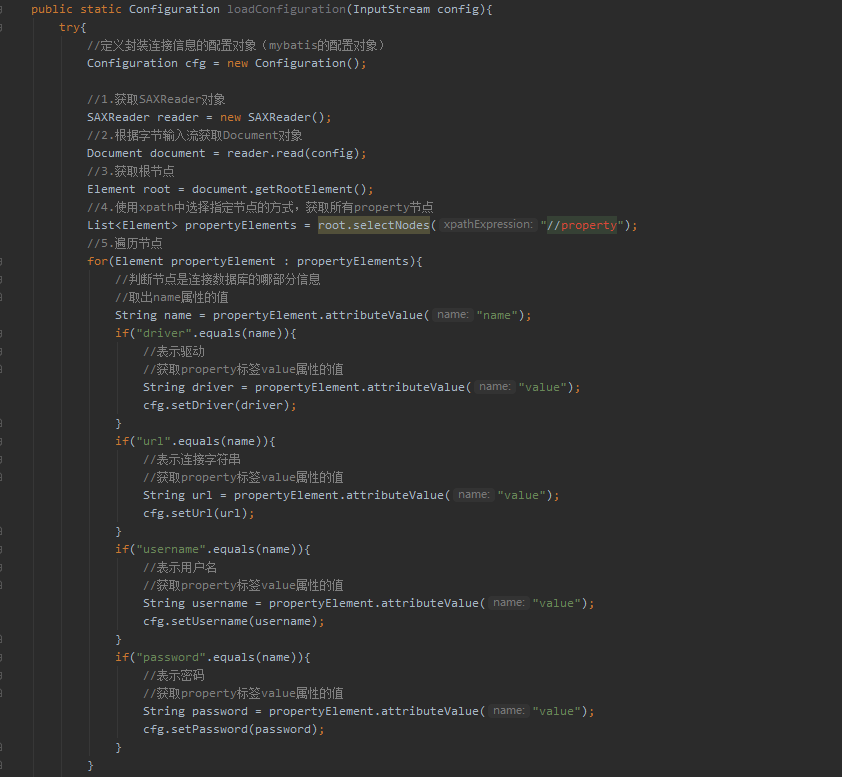
将我们的dom4j解析xml的java类引入（这里直接用别人的，这个不是我们研究的重点）

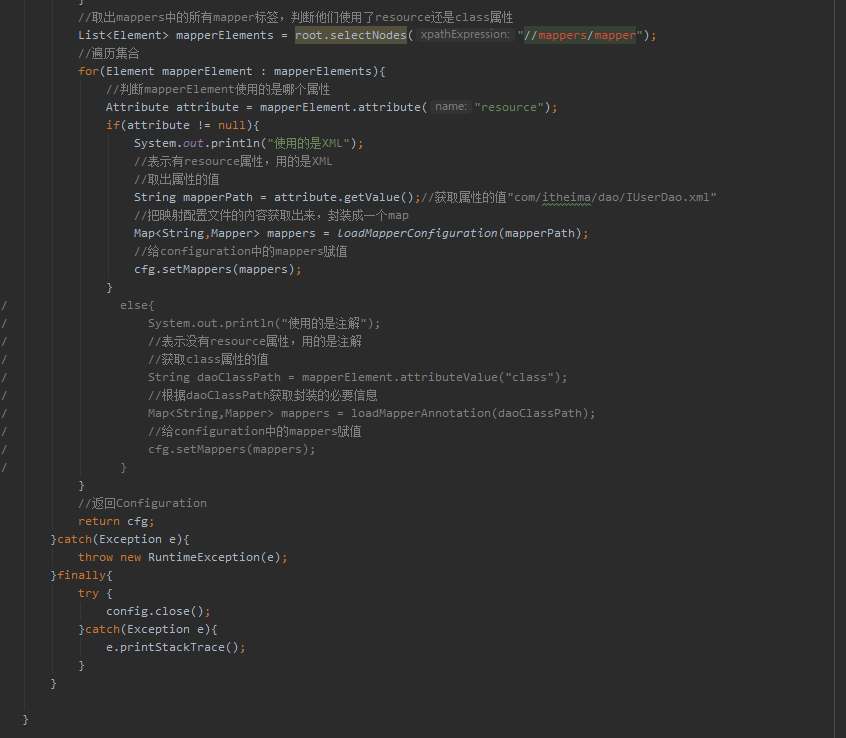


引入dom4j的坐标！

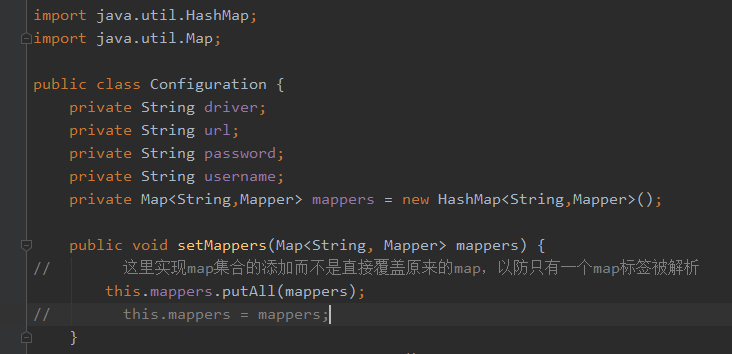
这里的loadconfiguration方法解析我们的输入流：

其中创建了一个configuration的封装连接池信息的实体类，里面也封装了我们的map集合，存放usermapper里面的方法名字（键名）和mapper封装类对象（值，用来存放此usermapper需要用到的sql语句以及返回的对象的类型！）



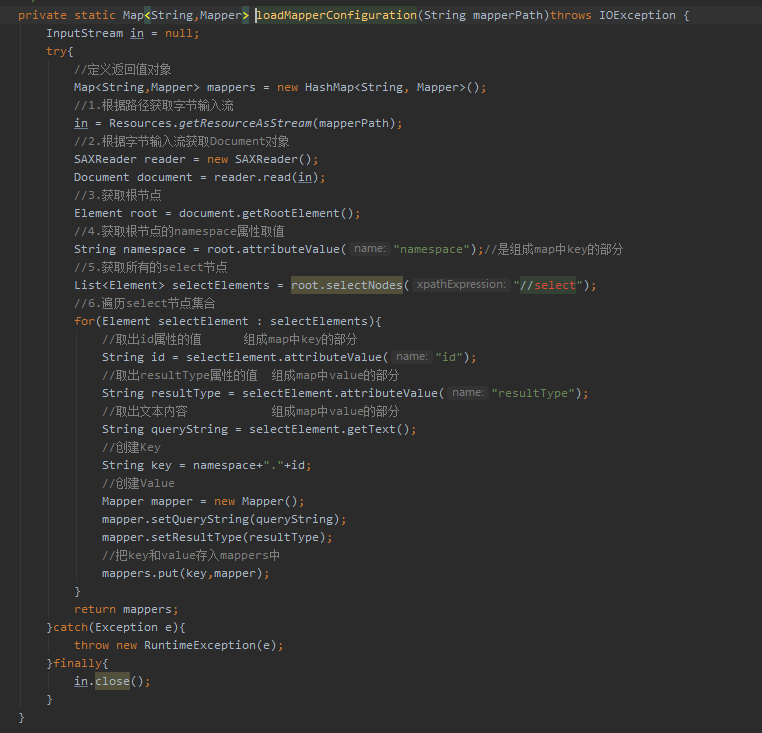


这就是创建的configuration封装类：



可以发现在loadconfiguration方法里面又进行了loadmapperconfiguration方法的调用，用来解析对用的mapper.xml映射文件。

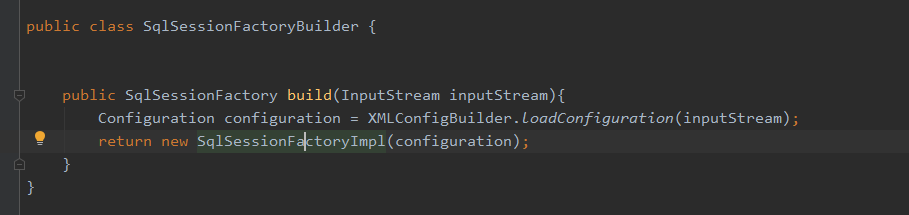
loadmapperconfiguration方法里面将解析到的sql语句以及返回类型封装到了一个map集合中，最后将这个map返回给loadconfiguration方法，然后将map赋值给configuration



以上全部是在xml配置文件的基础上进行解析的，涉及到的注解解析我们全部注释掉了！

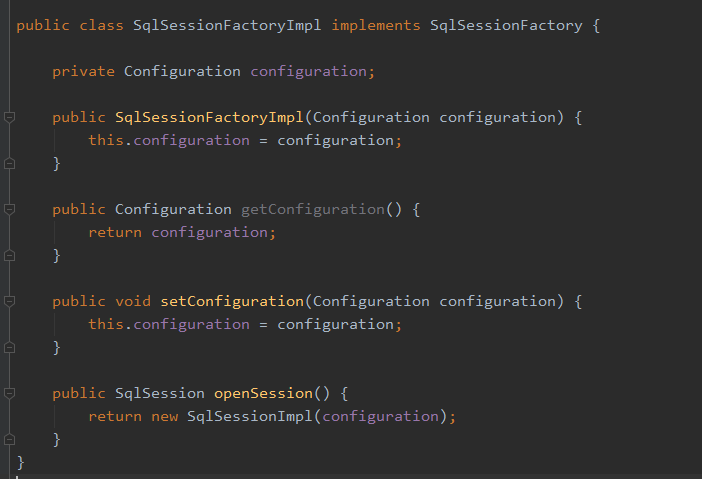
## 十：自定义mybatis编码（创建两个默认实现类并分析两个实现类中的关系）

首先在sqlsessionFactoryBuilder中对文件输入流进行读取！



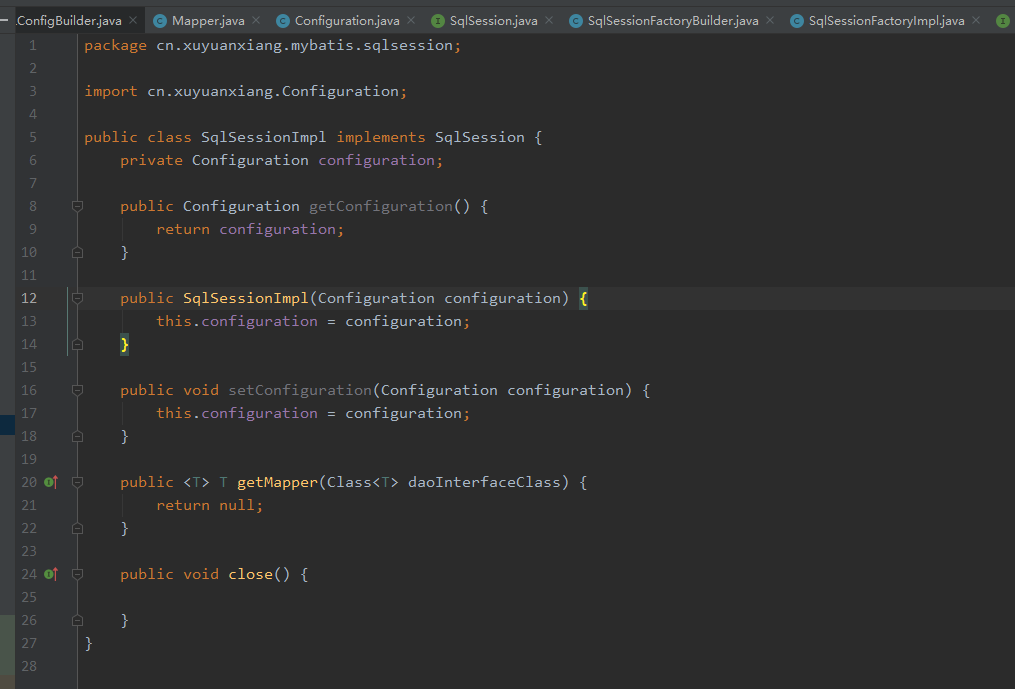
由于要返回一个sqlsessionFactory，我们在这里进行创建他的实现类！

这里需要用到Configuration中的信息（连接池信息以及sql语句和实体返回类），所以我们创建一个configuration存放根据输入流读取数据后得到的configuration类。



SqlsessionFactoryimpl中的opensession又要返回一个sqlsession类，我们需要创建他的实体类！

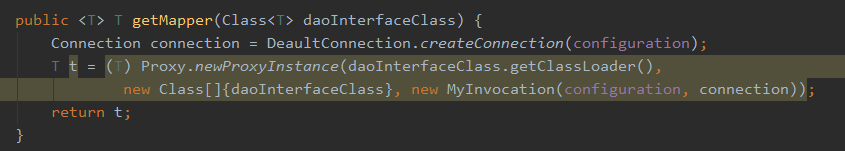
同样的道理，sqlsessionimpl以后要进行查询，也需要用到configuration中的信息（连接池信息以及sql语句和实体返回类），所以里面也包涵了一个configuration属性对上面传过来的configuration进行封装！同时这里需要重写编写getmapper，进行动态代理，返回我们需要的usermapper接口的代理对象（代理对象中实现了查询功能（用上了configuration中的所有信息！））。



## 十一：自定义mybatis编码（实现基于xml查询的所有操作！）

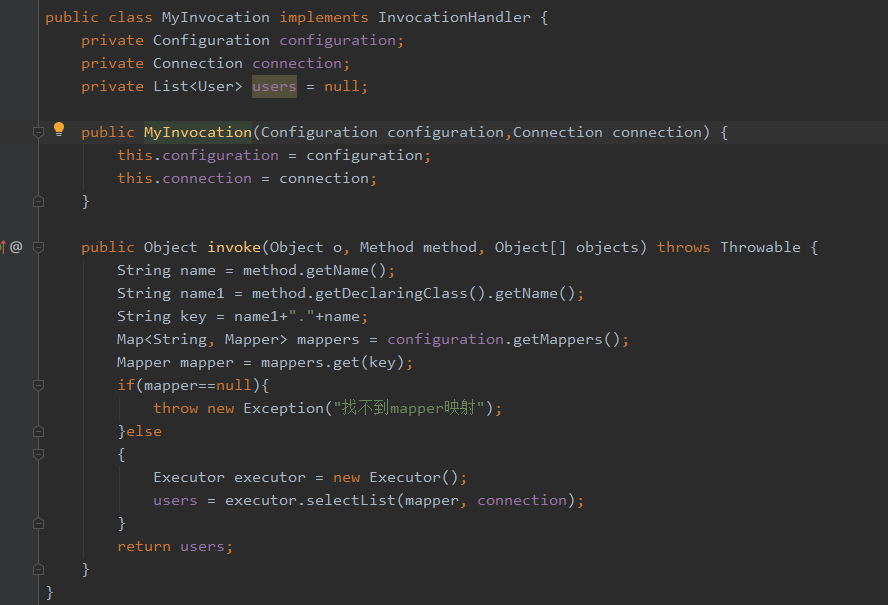
首先我们需要在getmapper中完成代理对象的创建：

需要注意的是，第一个参数位置，我们创建谁的代理对象就用谁的类加载器。同时这里传入的参数就已经是一个接口啦，所以第二个参数位置直接放入我们的接口就好了，第三个参数非常重要，它实现我们的代理逻辑！

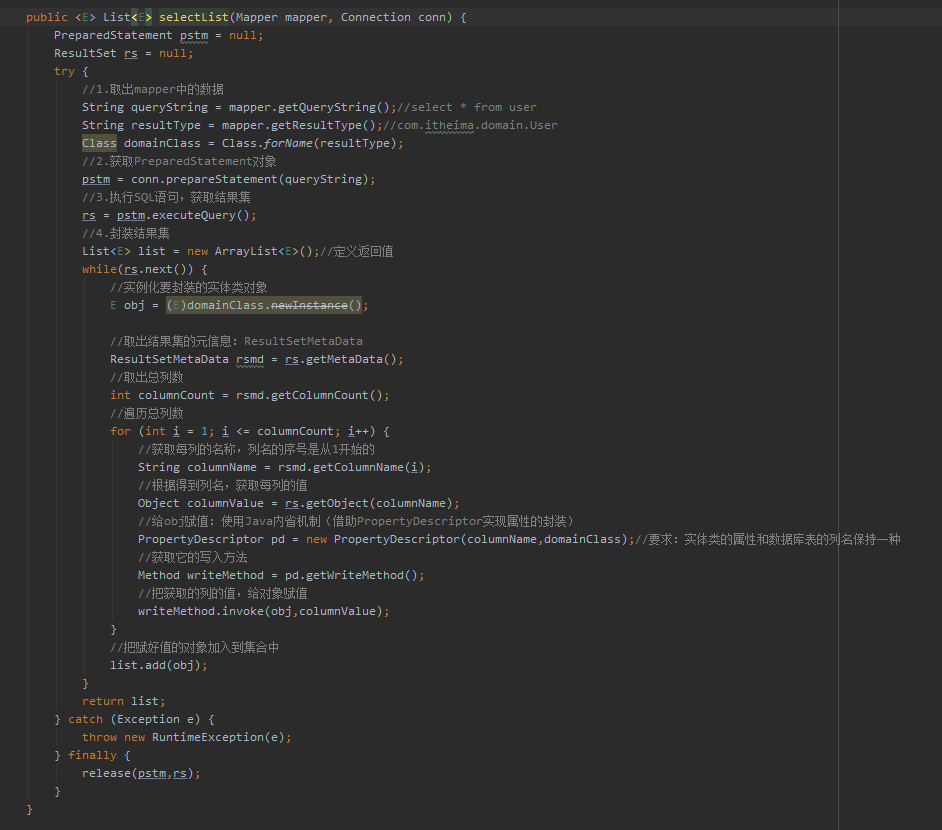


首先创建一个myInvocation对象，继承invocationHandler接口，实现其中的invoke方法！

在invoke方法中我们通过过去方法参数名称以及方法所在类名称拼接出一个map对象的key值，通过key值就可已从map集合中取出mapper对象，进而取出我们的sql语句和返回值类型！这些逻辑在executor中完成！我们需要做的就是将一个connection的连接和一个mapper对象传入，供方法使用！



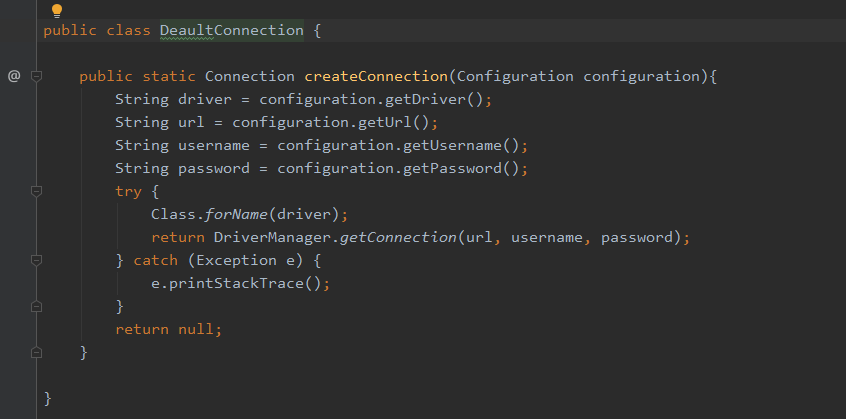
这便是executor中的selectlist的逻辑：



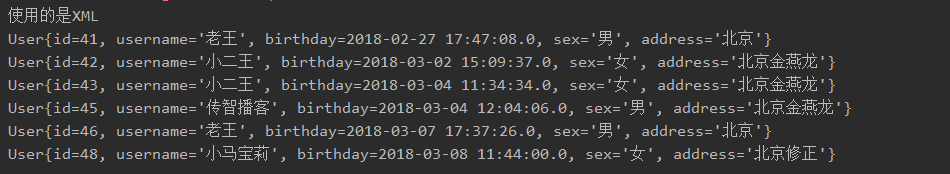
而我们需要传入的mapper参数在代理增强方法中已经获取到了，另外的一个参数connection我们在getmapper这个获取代理对象的方法中通过定义工具类中静态方法的方式获取：

如下：我们在方法中传入configuration参数，便可以获取到其中的数据库连接信息！返回给getmapper方法一个connection，供最后executor中的selectlist方法调用！

至于selectlist查询后如何将返回值封装至对象我们就不深入研究了！

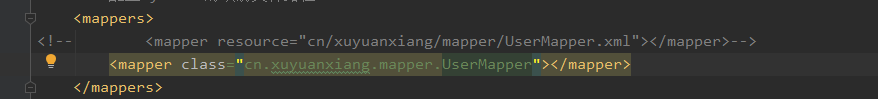


最后测试结果：成功呢

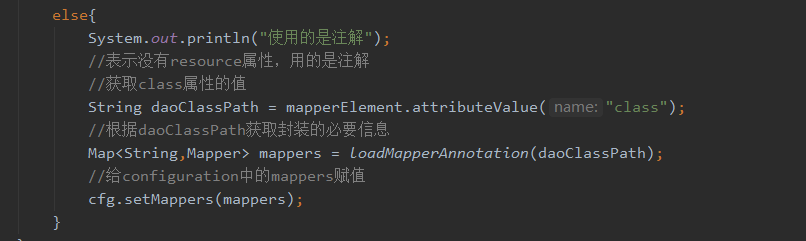


## 十二：自定义mybatis编码（实现基于注解查询的所有操作！）

首先更改我们的配置文件，指定mapper映射到接口

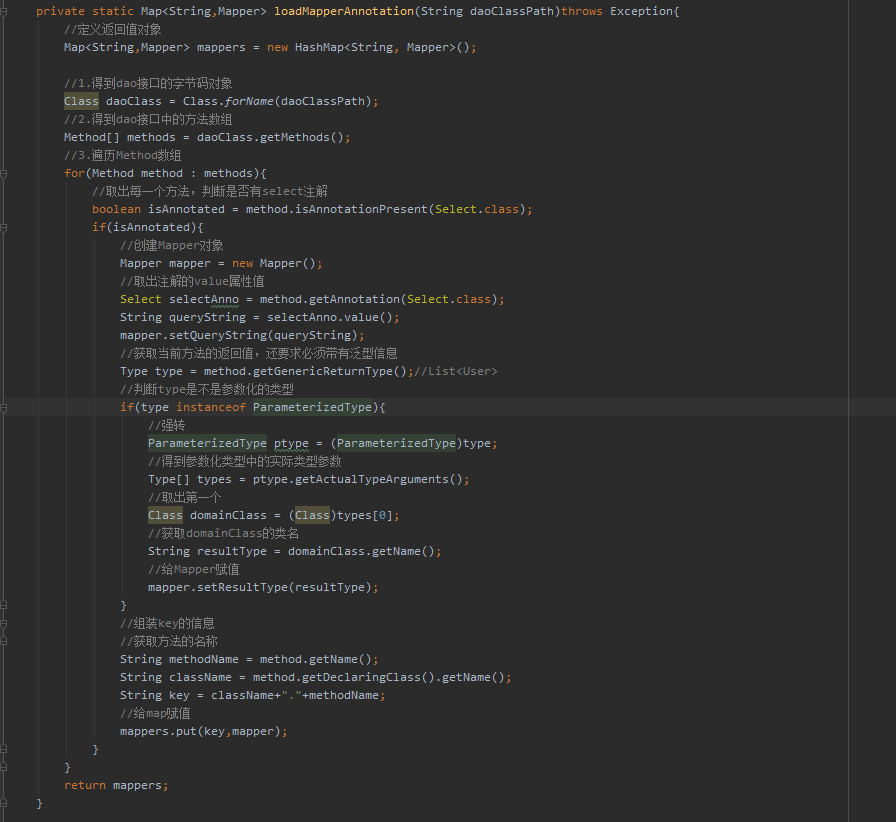


打开前面注释掉的注解解析部分！

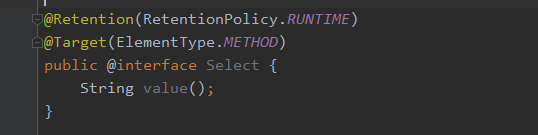


主要是里面的loadMapperAnnotation方法：采用了反射获取接口的class类，进而获取其中select注解里的sql语句。再通过固定方法获取返回值类型。

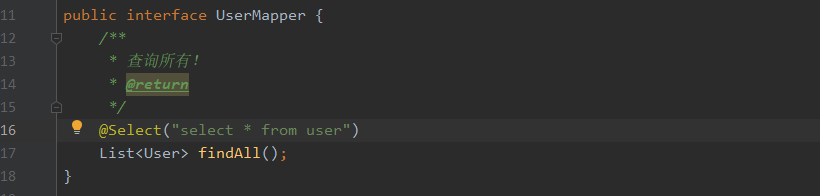
最后一样返回mappers类供其他方法调用！



在这里定义一个select注解



在我们的方法上添加自己的注解！



运行结果：成功！

