# Mybatis简介

* 首先我们知道传统的开发中持久层的操作有七个步骤：

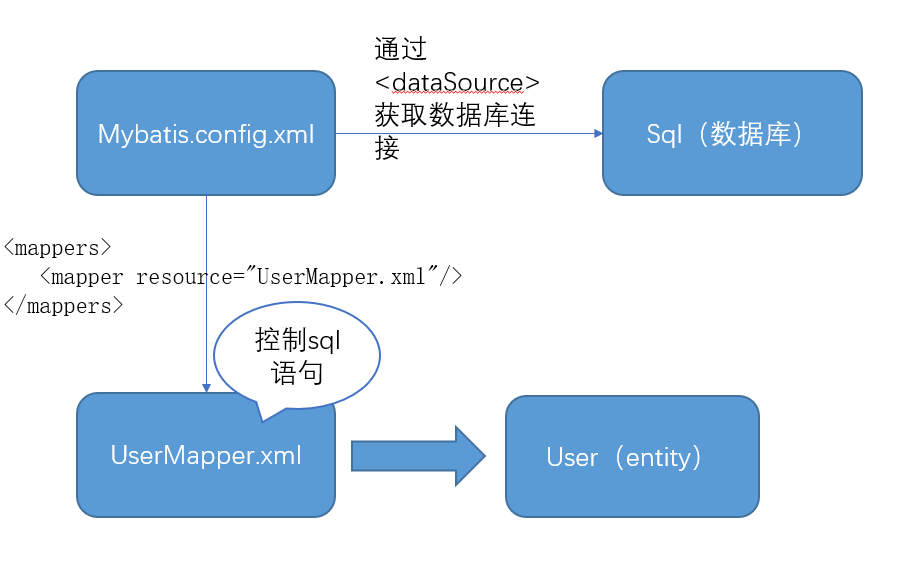
1. 加载驱动
2. 获取连接
3. 编写sql语句
4. 取得statement对象
5. 用statement执行
6. 处理执行完的结果
7. 释放资源

* 是一个给予Java持久层的框架

1. 持久化:数据从瞬时状态变为持久状态。
2. 持久层:完成持久化工作的代码块。 --dao层
3. Mybatis就是帮助程序员将数据存入到数据库中，和从数据库中取数据。
4. 传统的jdbc 操作: 有很多重复的代码。比如: 数据取出时的封装数据库的建立连接等等。通过框架提高开发效率。
5. Mybatis是一个半自动化的ORM框架O-object --R--relationship M-mapping

## 1.Mybatis配置文件说明

* 我们首先需要清楚Mybatis的映射关系



* Mybatis.config.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <environments default="development">  <environment id="development">  <transactionManager type="JDBC"/>  <dataSource type="POOLED">  <property name="driver" value="com.mysql.cj.jdbc.Driver"/>  <property name="url" value="jdbc:mysql://localhost:3306/test?serverTimezone=UTC"/>  <property name="username" value="root"/>  <property name="password" value="root"/>  </dataSource>  </environment>  </environments>  <mappers>  <mapper resource="UserMapper.xml"/>  </mappers>  </configuration> |

* + 在此文档中

<mapper resource="UserMapper.xml"/>是用来映射sql控制文件

* UserMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="cn.gsdx.day01.dao.IUserDao">  <select id="queryUser" resultType="cn.gsdx.day01.entity.User">  select \* from user where id = #{id}  </select>  </mapper> |

* + namespace：如果不是面对接口编程，那么namespace可以随便写，如果是面对接口编程，那么namespace则需要指定为：接口的全限定类名；
  + id : 为接口中指定的方法名称；
  + resultType：为指定的返回值类型

## 2.Mybatis.config.xml的标签定义

### properties

* mybatis可以使用properties来引入外部properties配置文件
  + resource：用来引入类路径下的文件
  + url：用来引入网路上或磁盘上的文件
* 例如：<properties resource=”dbconfig.properties”></properties>

Setting

### Settings

* 包含很多重要的设置项
  + Settings用来设置每一个设置项
    - name ： 设置项名
    - value ： 设置项取值

### typeAlias：为某个Java类型取别名

* type指定别名的类型全类名
* alais：指定新的别名（默认为类名小写）
* 在全局配置（mybatis.config.xml）中配置之后，也可以在映射文件（UserMapper.xml）中使用
  + eg: <typeAliases>

<typealiastype type=“com.gsdx…“ alais=”user” />

</typeAliases>

### Package:批量指定别名

* package:为某个包下的类批量起别名（别名不区分大小写）
  + name：指定包名
  + （当包下的别名冲突时，可以使用注解@Alias（）来指定别名）

## 3.Mybatis处理参数

* 单个参数时不做特殊处理
* 多个参数下mybatis会把参数封装成map，使用key-value的形式，而我们需要的值为value，mybatis默认的key为param1，param2或者0，1（从零开始的索引）；
* eg : 在UserDao接口中有一个多参的方法:
  + public void queryUser(String name, Integer id);

此时我们需要在sql语句中使用：

select \* from where name=#{param1} , id=#{param2}

* 我们也可以指定特定的key
  + 在接口中使用@Param注解
  + public void queryUser(@Param(“name”) String name, @Param(“id”) Integer id);
  + 此时sql语句就可以使用#{id} ，#{name}来获取参数；
* 如果多个参数正好是我们业务逻辑数据，那我们可以直接传入pojo
  + #{属性名}：取出传入的pojo的属性值
* 如果多个数据不是我们业务逻辑的数据，并且不经常使用的话，我们也可以直接传入一个map；
  + #{key}：此时可以直接取出map中key的值；
* 如果多个数据不是我们的业务逻辑数据，并且经常使用的话，推荐写一个TO（transfer Object）数据传输对象
* 如果参数是Collection类型或是数组
  + 也会特殊处理把传入的list或数组封装到map中
    - List封装的key—>list/collection
    - 数组—>array
  + public void queryUser(List<Integer> list);
    - #{list[0]} : 取出数组第一个元素

## 4.${}与#{}取值的区别

* #{}是以一种预编译的处理方式，将参数设置到sql语句
  + select \* from user where id = ？ //可以防止sql注入
* ${}是直接将参数拼接到sql语句后面
  + select \* from user where id = 1; //有安全问题
* 因此我们只有在jdbc占位符不支持的地方我们可以使用${}
  + Select \* from ${year}\_salary where user=#{user};
  + 此时我们可以通过传参动态的获取年份: 2012\_salary 等2013\_salary
  + select \* from ${year}\_salary order by ${salary} ${order}
  + 在排序的地方原生jdbc也是不支持占位符的，因此我们可以直接拼接；
    - 总结：只要不是在关键和敏感的字段，我们都可以进行拼接;

## 5.#{}更丰富的用法

* jdbcType通常需要在某种情况下设置：在数据为null的时候，有些数据库不能识别mybatis对null的处理，比如oracle：当字段为null时会报错；（jdbcType OTHER ：无效类型）；
* 这是因为mybatis对null数据都是映射的jdbc的other类型
* 此时我们可以指定他为null类型；（#{name, jdbcType=null}）

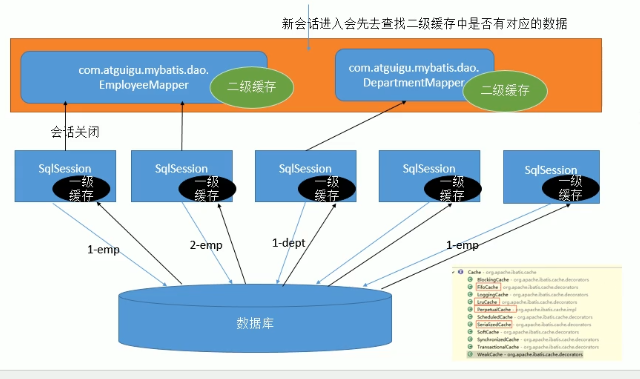
## 6.缓存机制

Mybatis系统默认有两级缓存，一级缓存和二级缓存；

* 一级缓存（本地缓存）
  + 一级缓存是一直开启的，mybatis将数据取出后封装到sqlSession的一个map中，我们每次取值都会先从map中查询，然后去数据库查询；与数据库同一次会话期间查询到的数据会放到本地缓存之中，以后想要获取相同的数据，直接从缓存中拿。
  + 当sqlSession不同时
    - 直接从数据库中拿数据
  + 当sqlSession相同时：
    - 会从sqlSession中拿数据
  + 当sqlSession相同，但查询条件不同时
    - 如果sqlSession中没有这条数据，则从数据库中查询
  + 当sqlSession相同，两次查询之间执行了增删改操作
    - 从数据库中拿值（因为增删改可能对数据进行了修改）
  + 当sqlSession相同，当手动清空了缓存
    - 从数据库拿值
* 二级缓存（全局缓存）

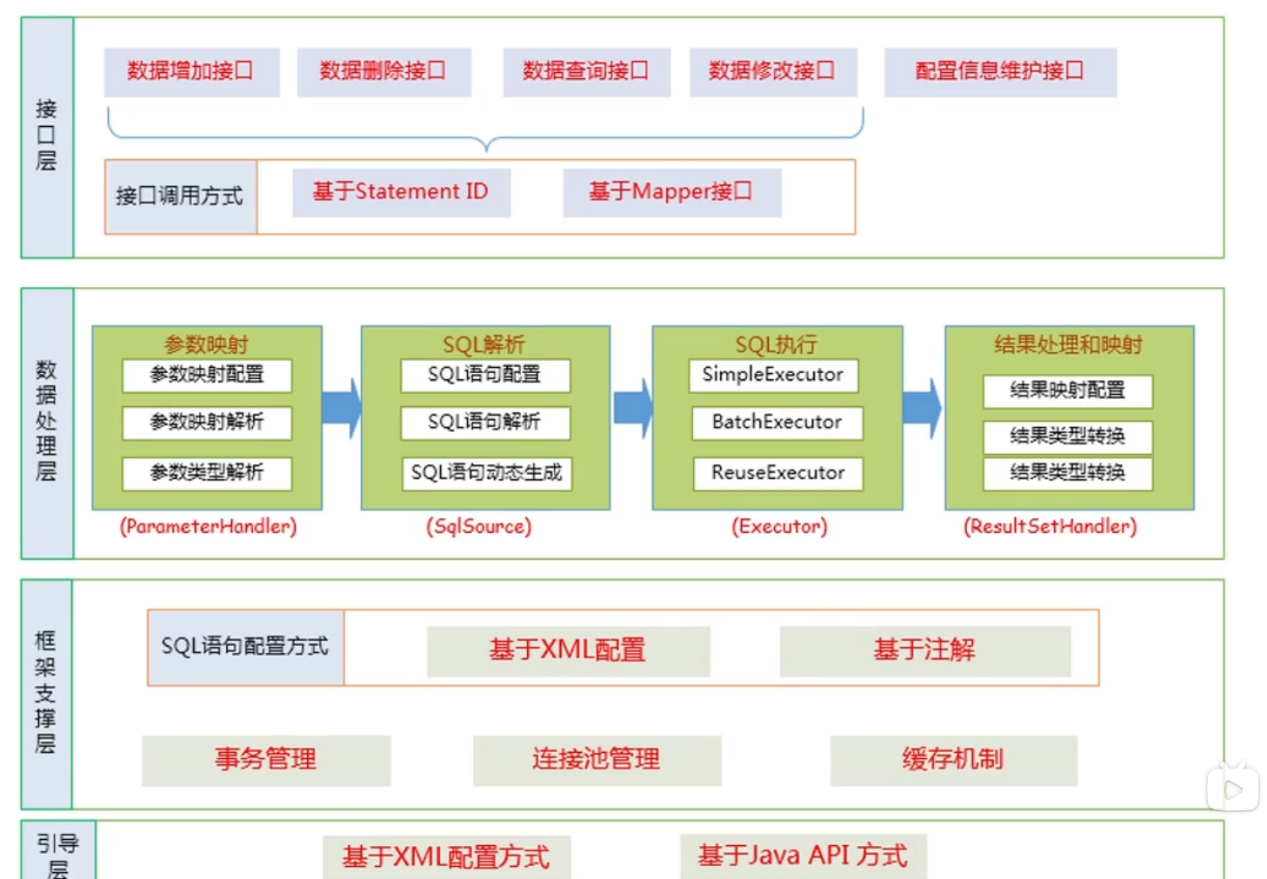
基于namespace级别的缓存，一个namespace对应一个二级缓存

* + 工作机制
    - 一个会话，查询一条数据，这个数据就会被放到当前会话的一级缓存之中；
    - 如果会话关闭，一级缓存中的数据会被保存到二级缓存之中，新的会话查询信息，就可以参照二级缓存
  + 使用步骤：
    - 开启二级缓存(全局配置中)
      * <setting name=”cacheEnable” value=”true”></setting>
    - 去mapper.xml中配置二级缓存
      * <cache eviction="FIFO" flushInterval="60000" readOnly="false">
        + eviction：缓存的回收策略
        + flushInterval: 刷新间隔，单位毫秒
        + size：引用数目
        + readOnly：只读 true/false
    - 因为我们的二级缓存会实现序列化和反序列化，因此我们的pojo需要实现序列化接口
    - Type：指定自定义的缓存的全类名
      * 自定义接口需要实现Cache接口
* 和缓存有关的设置/配置
  + cacheEnable：true/false（关闭的是二级，而没有关闭一级缓存）
  + 每个select标签都有 useCache属性true/false
    - 当为false时： 一级缓存依然可以使用，但是二级缓存不能使用
  + 每个增删改标签都有一个 flushCache：清空缓存
    - 默认为true，一级缓存和二级缓存都会清空
    - 查询标签默认为false
  + openSession.clearCache()，会清空一级缓存
  + localCacheScope：本地缓存作用域（默认为SESSION,当前会话所有数据保存在会话缓存中，STATEMENT 禁用一级缓存）
* 缓存的使用顺序

Mybatis中当有新的会话进来时会先去二级缓存中查找，再去一级缓存中查询，然后去数据库中查询；

* Mybatis对于缓存留有Cache接口用给给我们自定义缓存的策略。

# Mybatis工作原理

* 分层框架图

## 1.四大对象：

### Executor

* sqlSession 首先会调用executor.query方法执行查询。
* Executor有以下四种(均继承了BaseExecutor，可在spring文件bean中配置)：
  + SimpleExecutor-简单的Executor，所有的数据库操作均委托给StatementHandler处理
  + BatchExecutor - 批量更新的Executor，使用jdbc的batchupdate方法，使用此Executor，需要手动执行SqlSession的flushStatements，立即执行sql。
  + ReuseExecutor-大体上与SimpleExecutor相同，只是根据sql缓存了jdbc的statement，遇到相同的sql时，省去了statement的创建，提高效率，但需要手动执行SqlSession的flushStatements，清除缓存
  + CachingExecutor - 在mapper中配置cache，对查询结果做缓存，其中缓存的key根据以下代码存数生成，建议最好不要用mybatis这个层次的缓存，一般业务层的缓存更实用，而且更容易控制

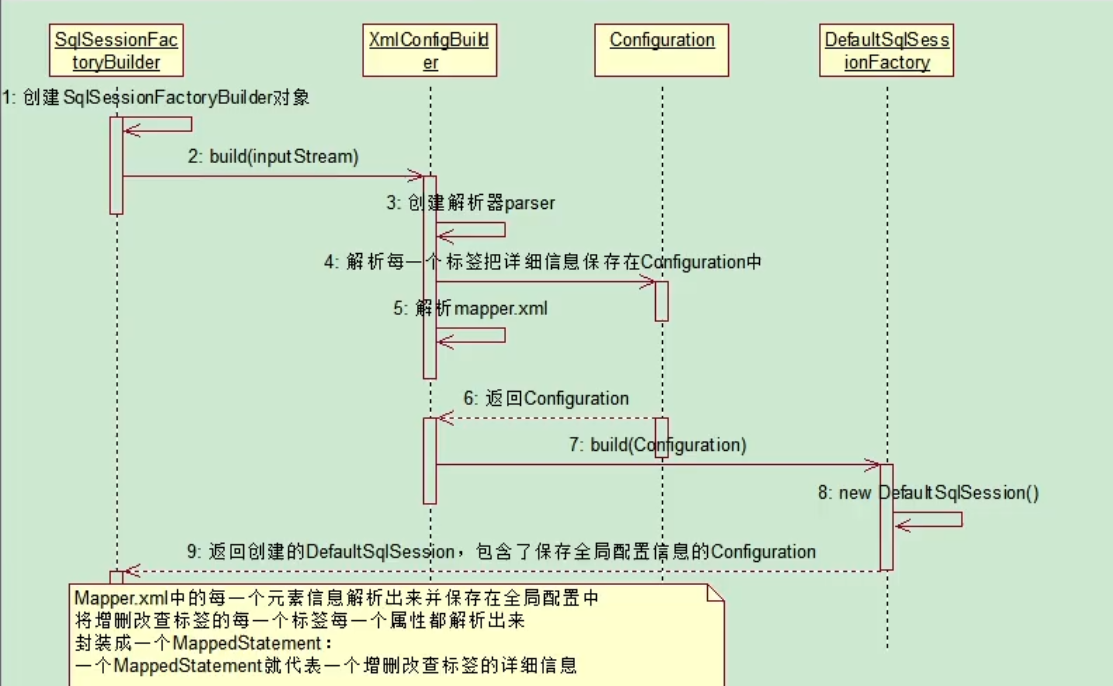
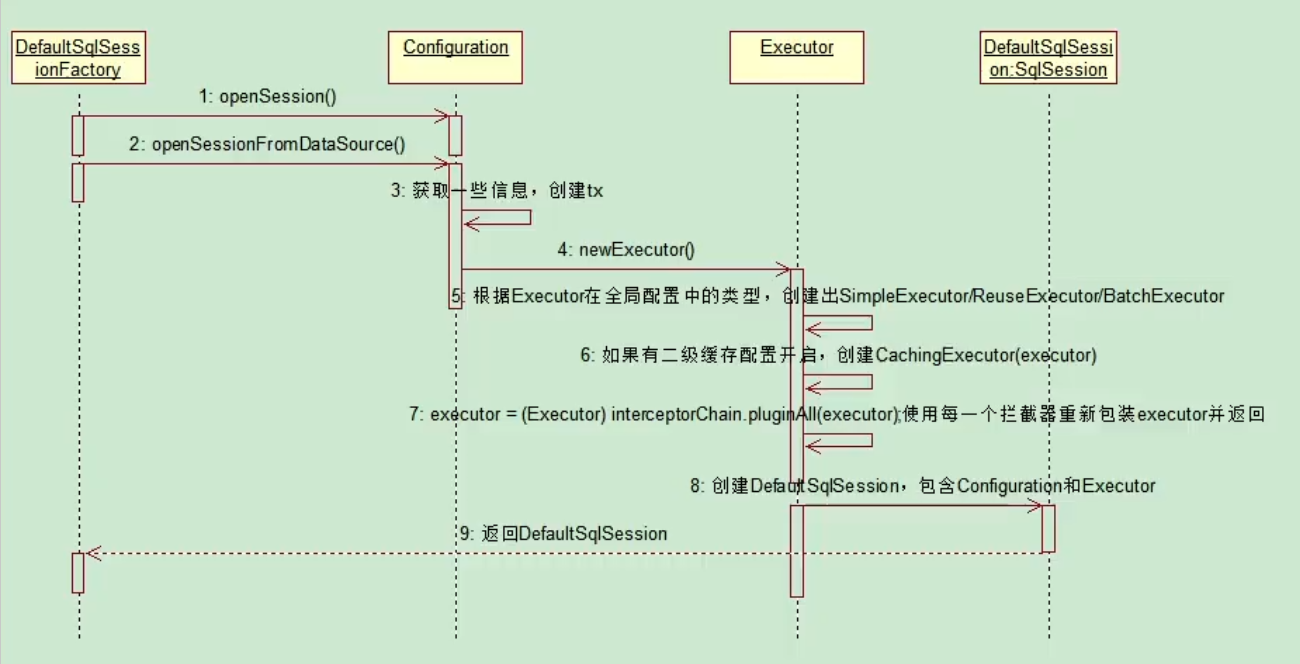
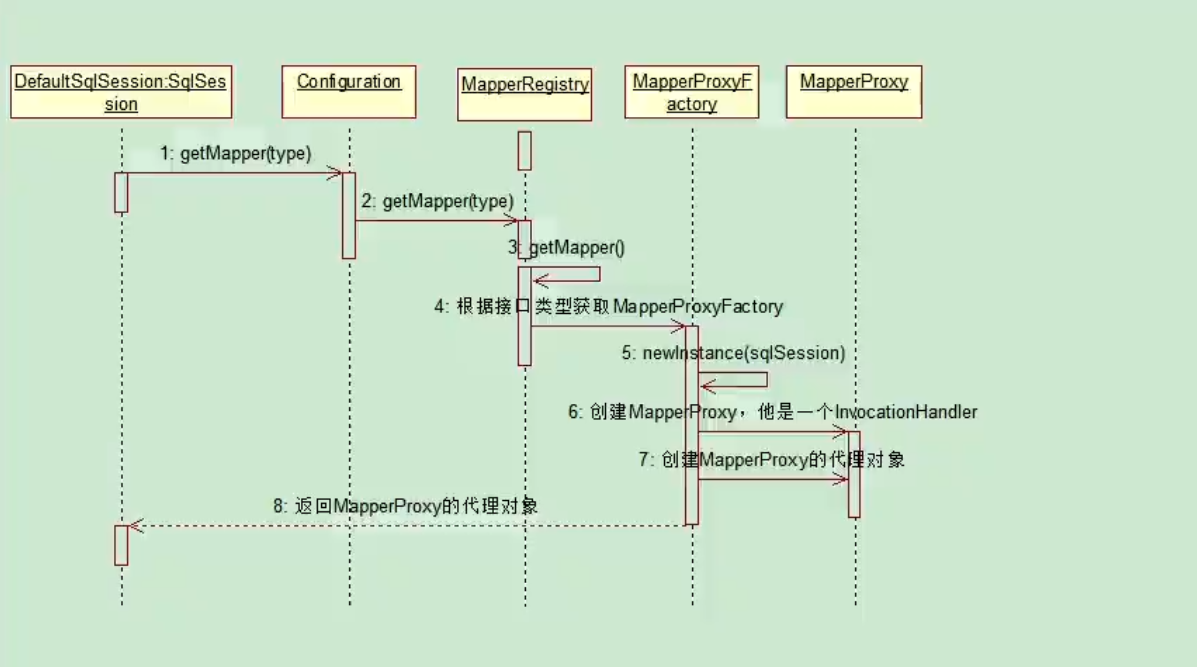
### StatementHandler

* Executor在doQuery时生成StatementHandler对象
* 在MyBatis实现了statementHandler的有四个类：
  + RoutingStatementHandler，这是一个封装类，它不提供具体的实现，只是根据Executor的类型，创建不同的类型StatementHandler。
  + SimpleStatementHandler，这个类对应于JDBC的Statement对象，用于没有预编译参数的SQL的运行。
  + PreparedStatementHandler 这个用于预编译参数SQL的运行。
  + CallableStatementHandler 它将实存储过程的调度。
* statementHandler的主要方法：
  + prepare：编译sql
  + parameterize：prepare后设置参数
  + query/update：sql执行
* 基于StatementHandler对象定义的插件：
  + 修改sql语句则在预编译SQL(prepare方法前）操作。
  + 修改参数则在调用parameterize方法前修改逻辑。或者使用ParameterHandler来改造设置参数。
  + 组装结果集则在query方法前后加入逻辑，或者使用ResultHandler来改造组装结果。

### parameterHandler和resultHandler对象

* parameterHandler是参数处理器
* resultHandler返回集合处理器

## 2.运行流程图

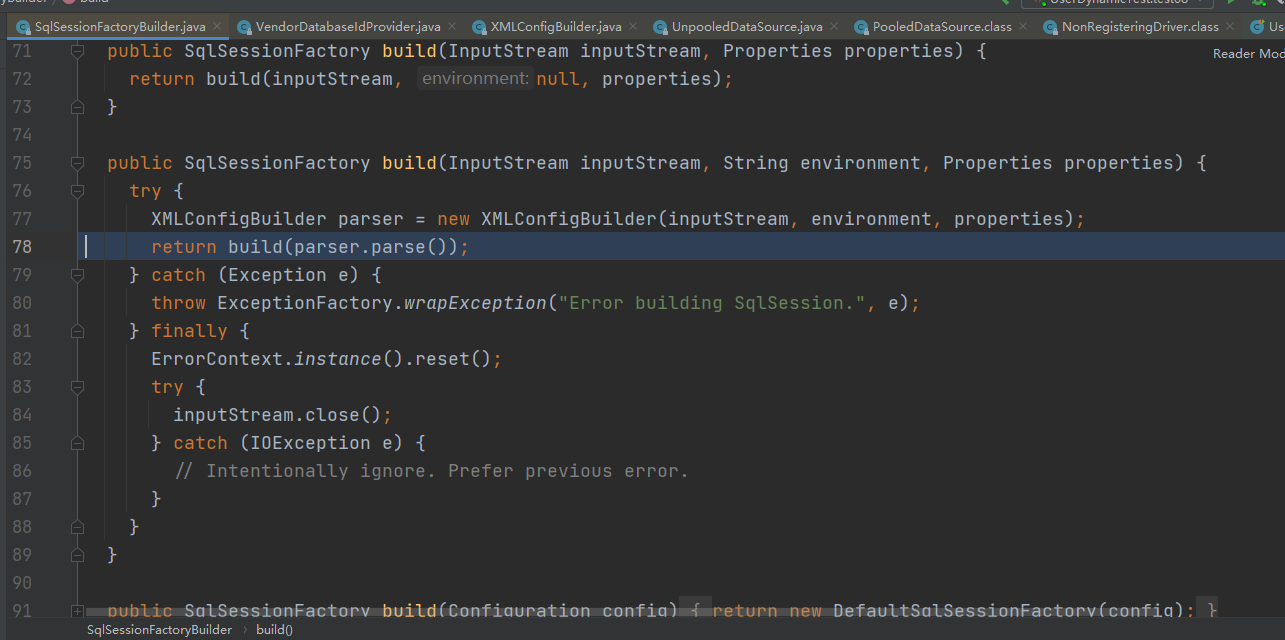
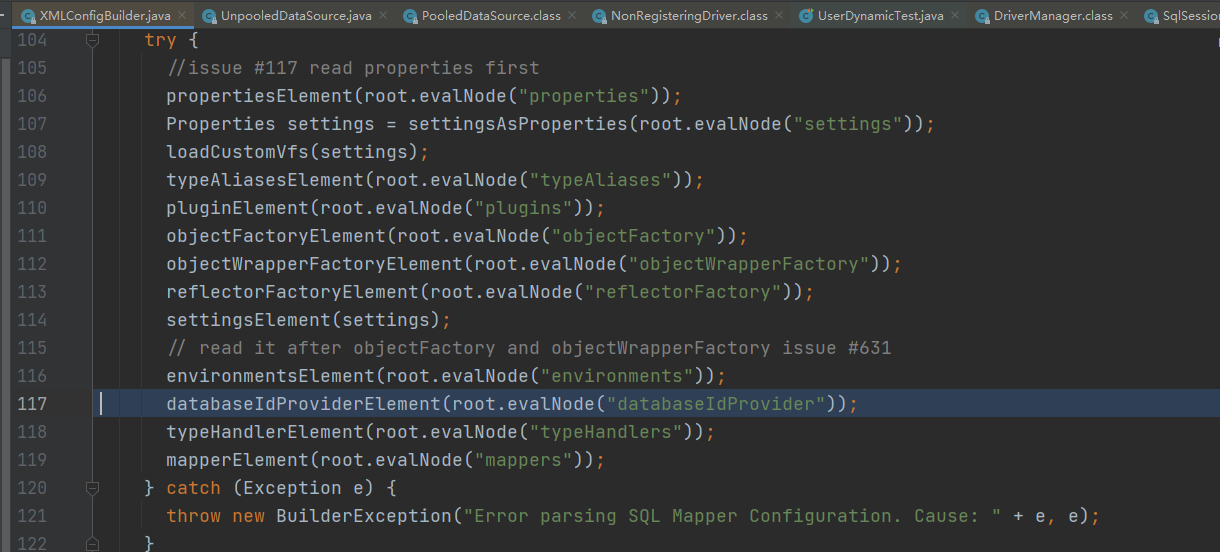
* + 1、根据配置文件创建sqlSessionFactory 实际上返回的是Default SqlSessionFactory（Configuration封装了所有配置详情）
  + 总结: SqlSessionFactoryBuilder().build(resources)把配置文件的信息全部解析并保存到Configuration对象中，返回包含了Configuration的DefaultSqlSession对象
* 2. sqlSessionFactory.openSession();
  + 返回DefaultSqlSession，里面包含了Executor和Configuration
  + Execute会在这一步被创建
* 3. 获取接口的实现类对象（代理对象）
* 总结：
  + 1、根据配置文件（初始化一个）configuration对象
  + 2、创建一个DefaultSqlSession对象
    - 它里面包含了Configuration以及Executor（根据全局配置文件中的defaultExecutorType创建出对应的Executor）
  + 3、DefaultSqlSession.getMapper()，获取映射接口的代理对象，MapperProxy(),里面有DefaultSqlSession；
  + 5、增删改查方法：
    - * 调用DefaultSqlSession的增删改查
      * 会创建一个StatementHandler对象。（同时也会创建出ParamentHandler和ResultSetHandler 对象）
      * 调用StatementHandler的预编译和设置参数
      * 调用StatementHandler的增删改查方法
      * 使用ResultSetHandler进行封装结果
  + 注意四大对象：

1. 四大对象每一个在创建时：都有一个interceptorChain.pluginAll(parameterHandler);

## 3.Mybatis数据库连接源码分析

* 我们首先在SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(resources);创建sqlSessionFactory；

在sqlSessionFactory的创建中我们首先进入dom4J的技术进行配置文件的解析，拿到全局配置文件的信息；

* 我们首先进入的是, SqlSessionFactory工厂里的parser.parser()方法进行配置文件的解析；
* 在parser.parser();方法里，我们可以看到XMLConfiuration中我们通过配置文件中的标签，进行全局配置信息的获取，我们直接跳到databaseIdProviderElement(root.evalNode("databaseIdProvider"));
* 在XMLConfiguration中进入databaseIdProvider配置信息的获取，同样是在XMLConfiguration文件中；

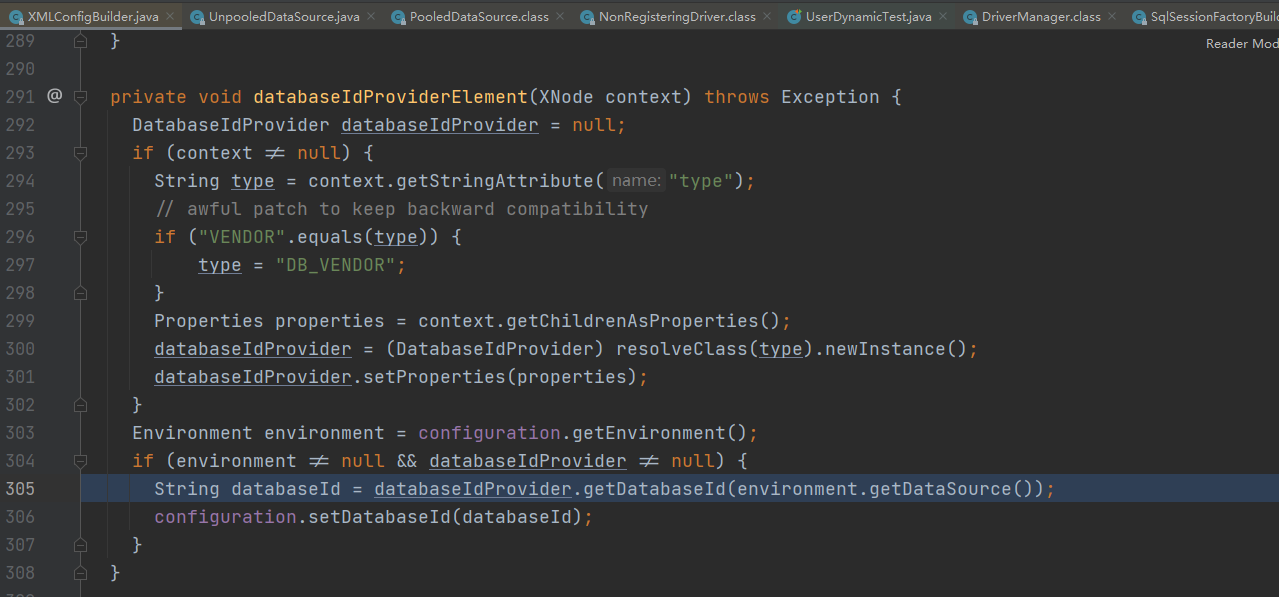
databaseIdProvider是不同数据库厂商提供的ID，我们可以在全局配置中进行配置：

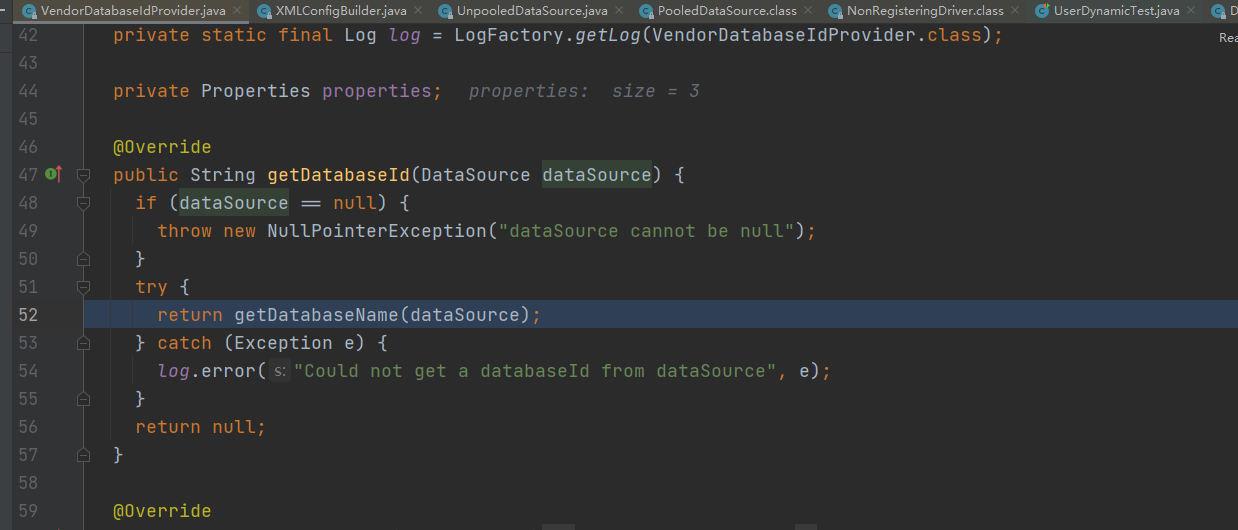
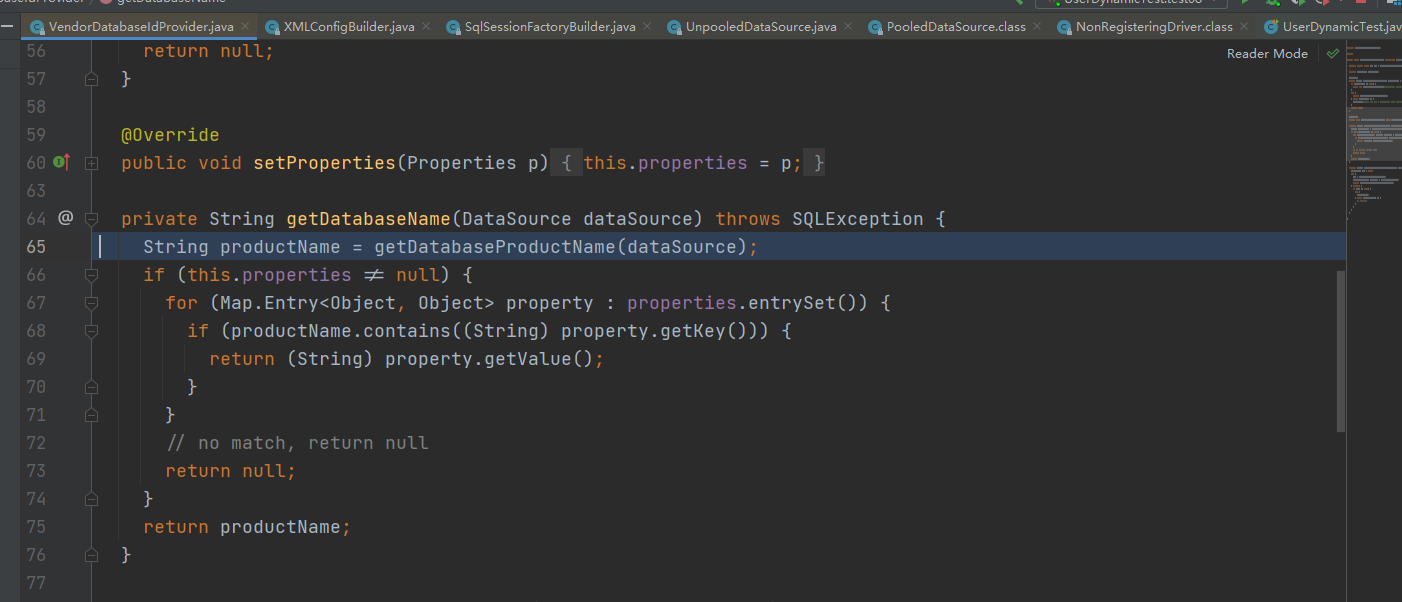
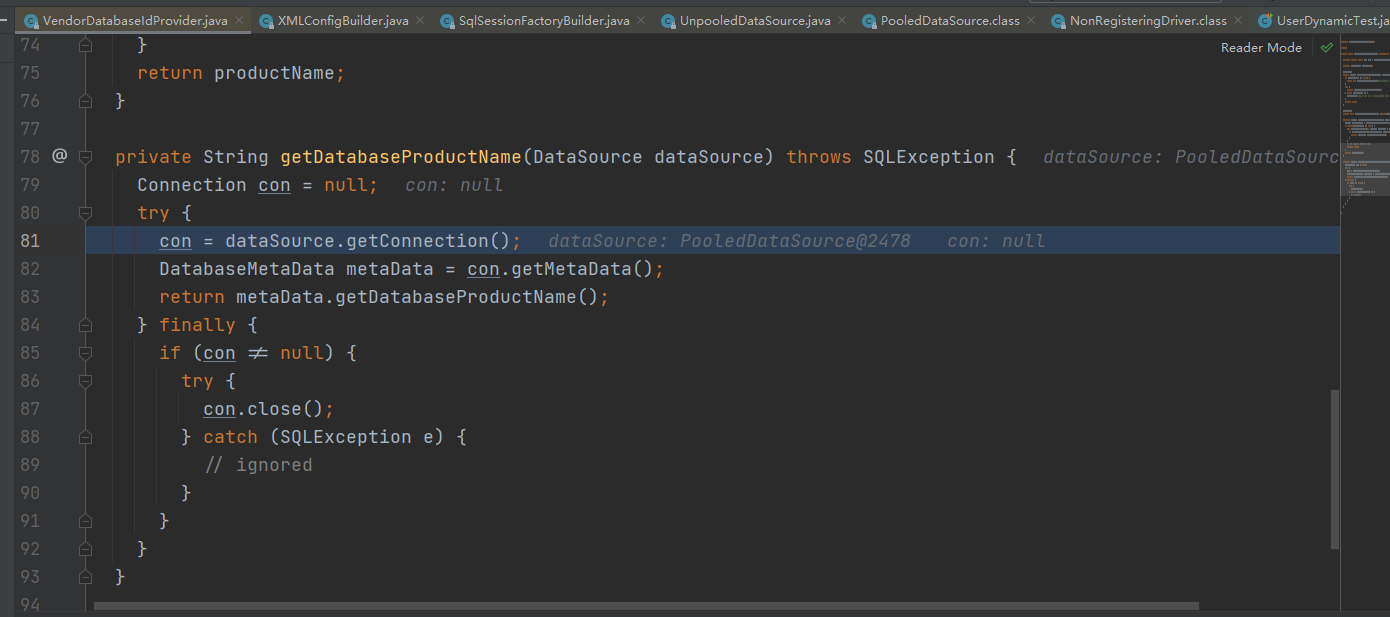
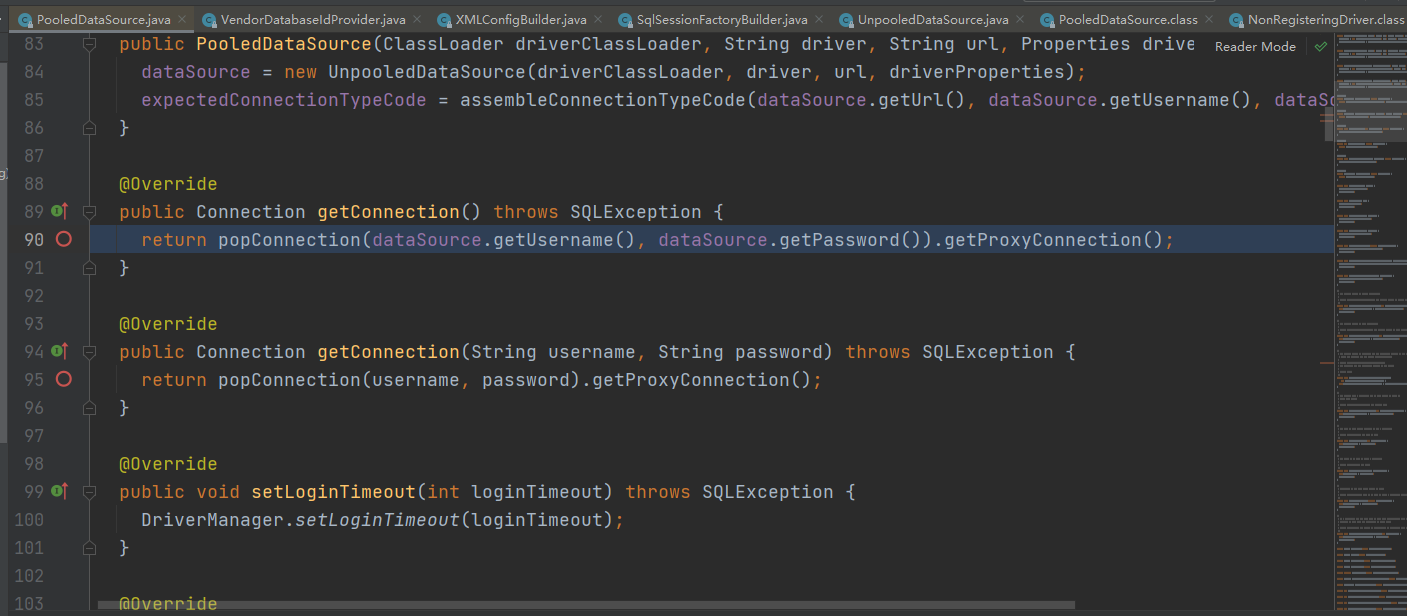
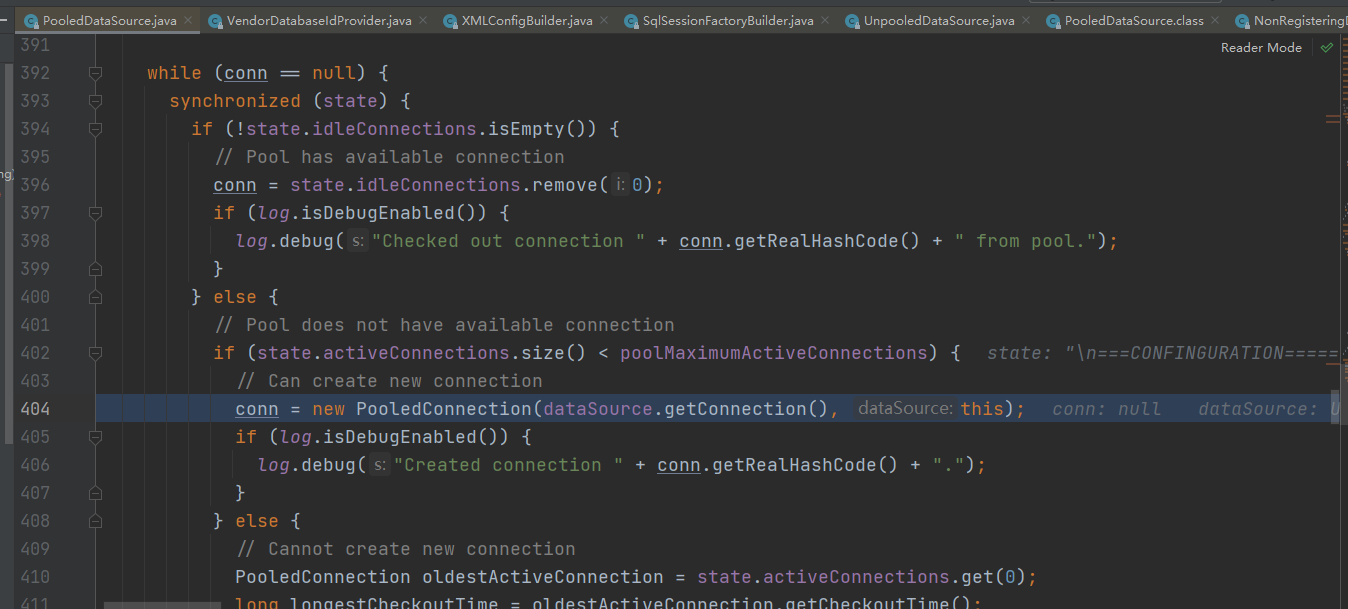
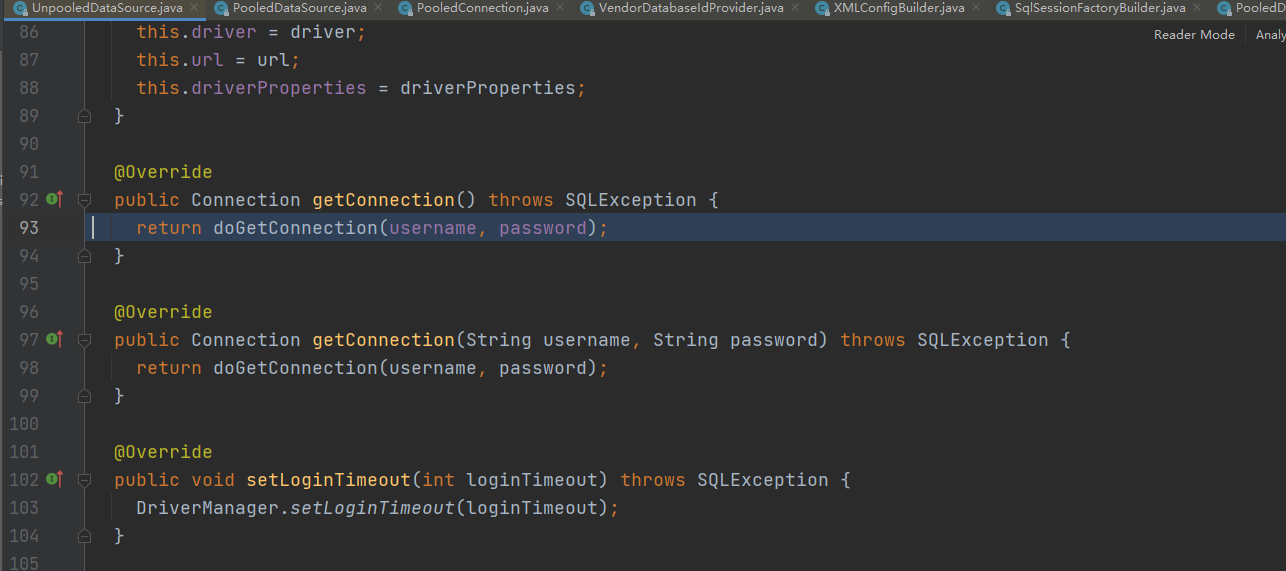
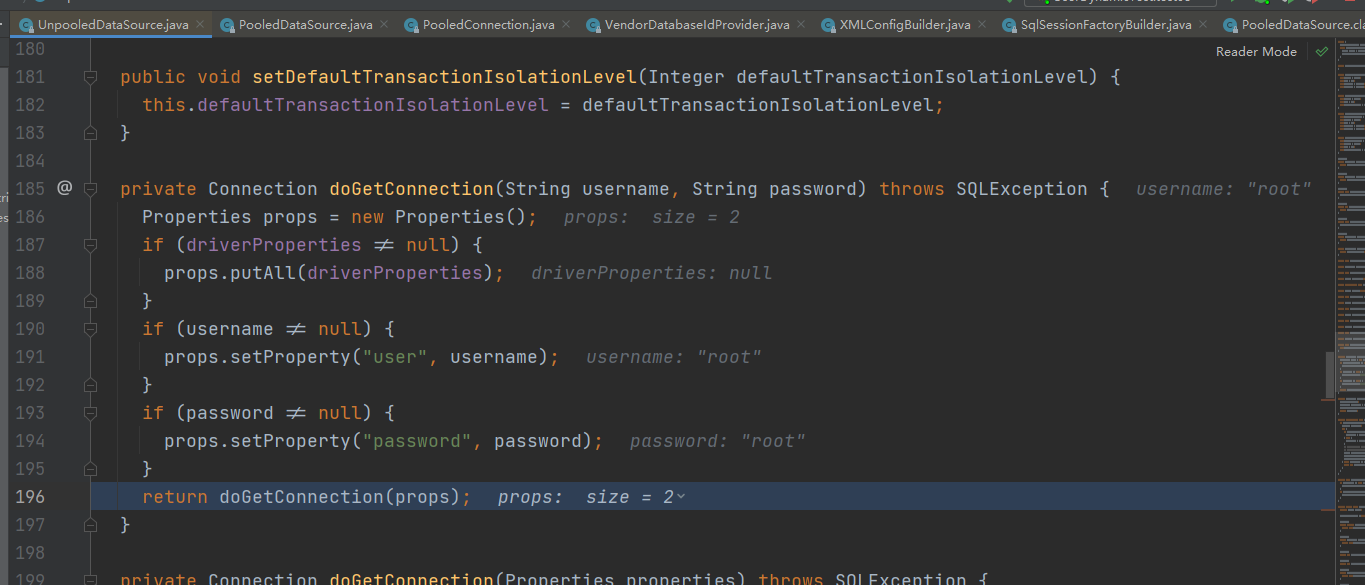
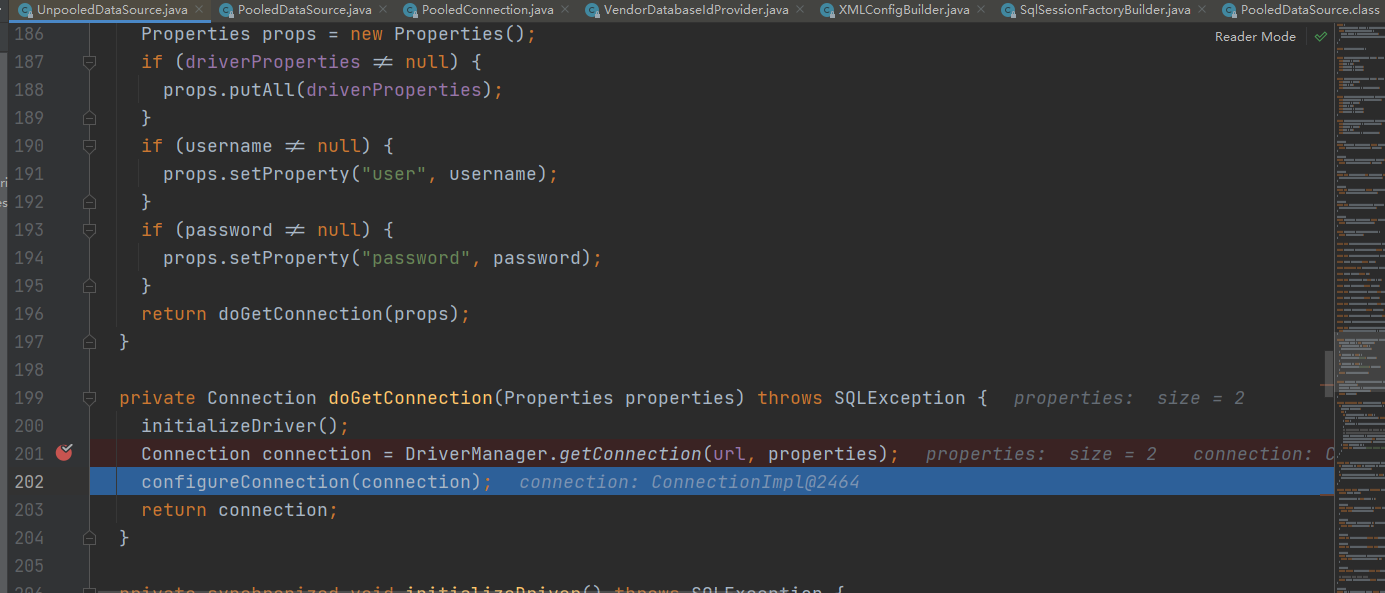
<databaseIdProvider type="DB\_VENDOR">

<property name="SQL Server" value="sqlserver"/>

<property name="DB2" value="db2"/>

<property name="Oracle" value="oracle" />

</databaseIdProvider>

* 这里dataSource，是UnpooledDataSource，在之前的源码中我们能够得出，这里就不继续深究。
* 我们继续深入，getDataBaseName()方法
* 进入该方法，注意这里是重点（连接数据库的方法我们可以看到了）我们进入该方法！
* 此时我们可以看到此时我们进入连接池的连接数据库方法！
* 我们进入该方法中，因为dataSource是UnpooledDataSource实例化的方法，此时我们可以联想到PooledDataSource也是依赖UnpooledDataSource，我们进入getConnection()方法中
* 我们继续深入
* 我们进入doGetConnection（）中，注意这是重点
* 从这里我们看到connection 的连接，DriverManager.getConnection(url, properties);看到这里我们就已经看到了原生的数据连接了，同时在UnPooledDataSource中定义了connection，以后的连接池调用可以直接调用这个参数，而不需要getConnection；

3.Mybatis插件开发

* 1.每个四大对象创建的时候不是直接返回，而是

InterceptorChain.pluginAll(parameterHander);

* 2获取所有Interceptor拦截器（所有插件都需要实现的接口）

调用InterceptorChain.pluginAll(targe);返回target包装后的对象

* 3.插件机制，我们可以使用插件为目标对象进行一个代理，AOP（面向切面编程）

我们的插件可以为四大对象创建出代理对象

代理对象可以拦截四大对象的每一个执行；