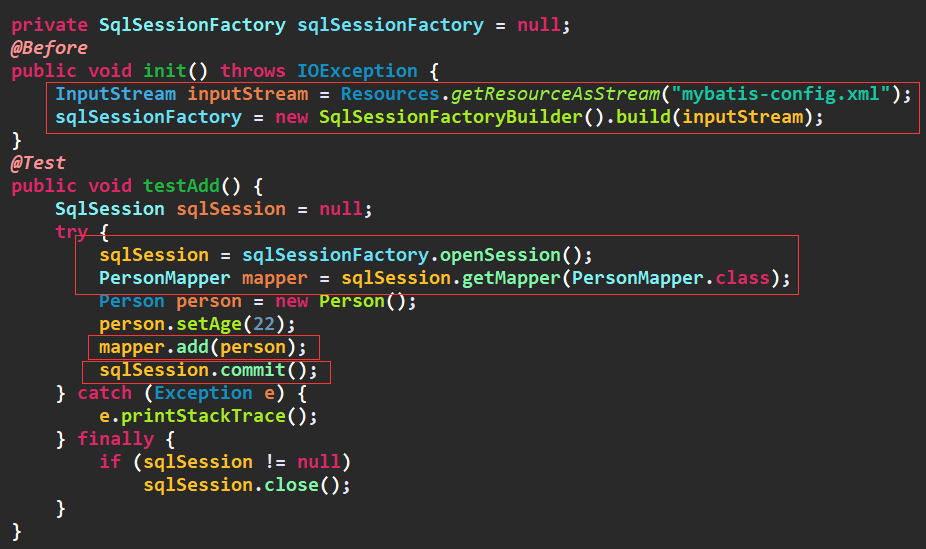
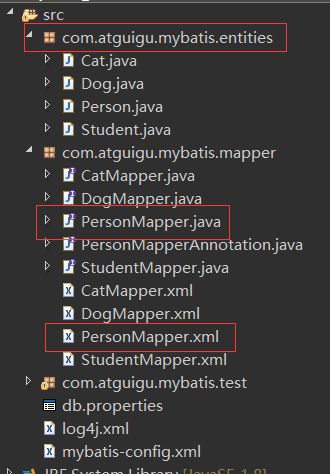
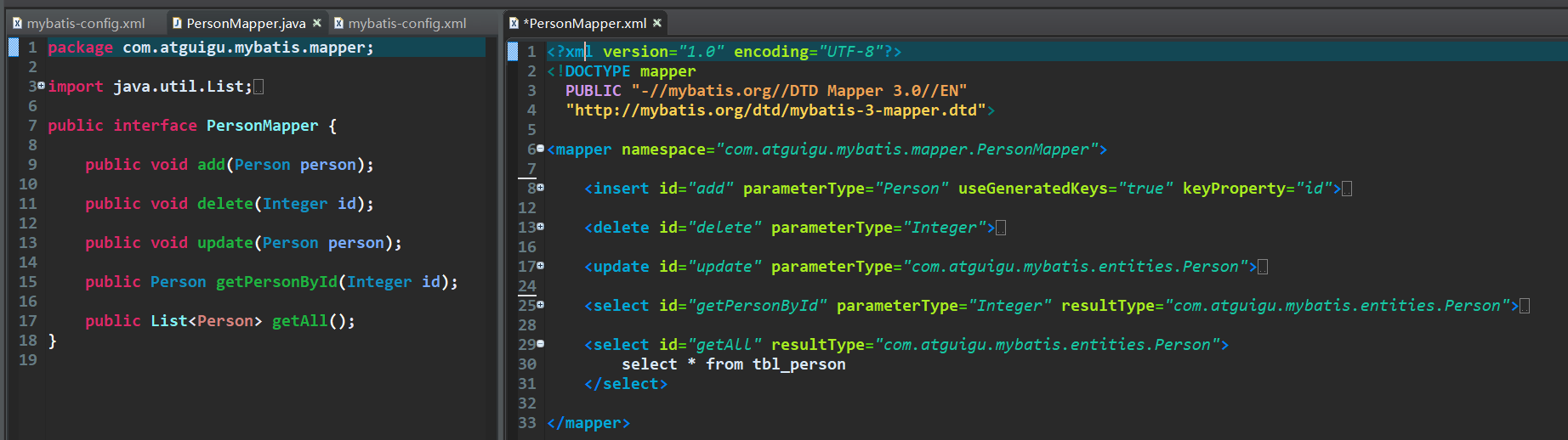
Mybatis和Hibernate一样也是一个ORM框架，Hibernate太智能化，当需要使用自定义的SQL语句时，太麻烦；而Mybatis是SQL和java代码分开的，灵活，SQL语句这种东西还是需要我们自己来写的

Hibernate太重，而Mybatis是轻量级的(轻量级在更多功能的实现上，是否需要阅读源码，轻量级提供了更多的实现方式)

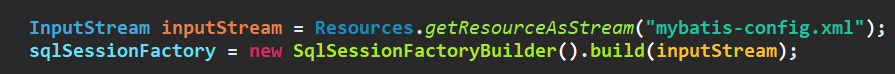
Mybatis的简单实现：

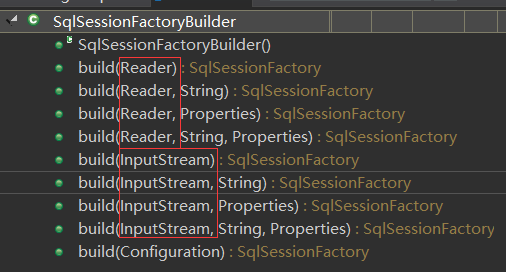




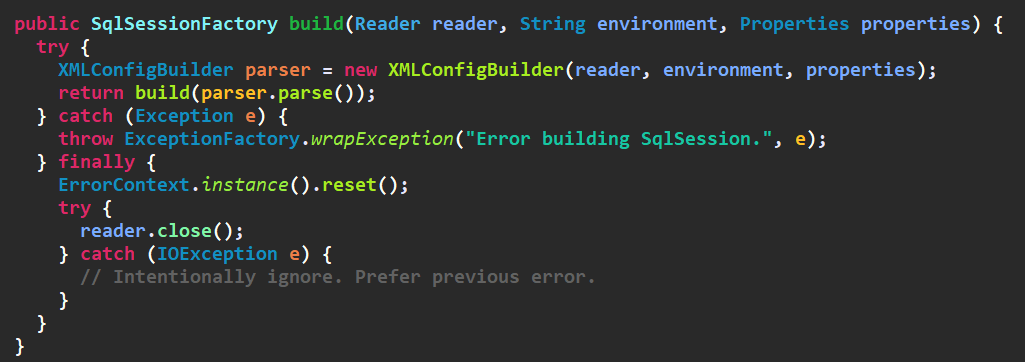
这里Mapper接口实际上就相当于dao，Mybatis实现了java代码和SQL语句的分离，将dao对应实现功能的SQL语句写在配置文件中

Mybatis加载配置文件：

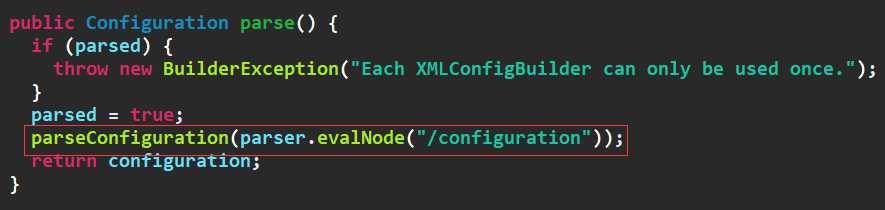


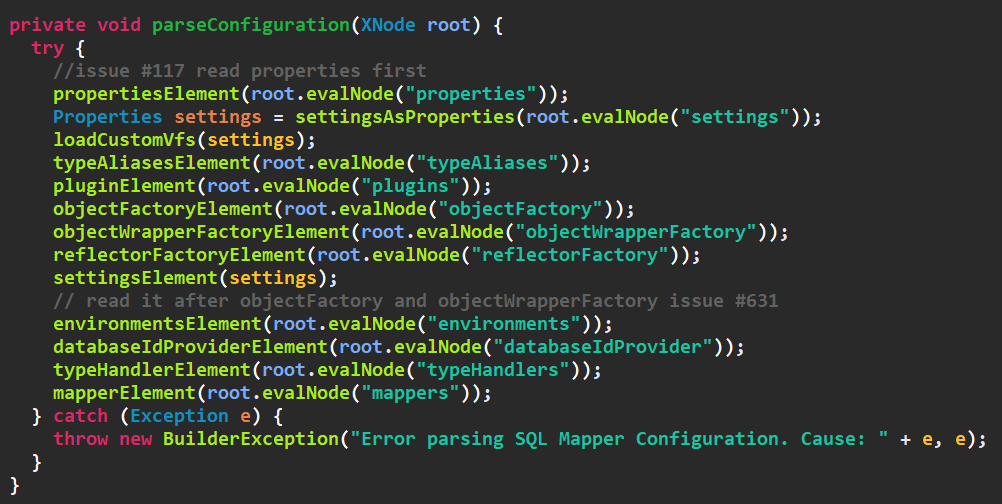


通过查看SqlSessionFactoryBuilder的方法可以看出，Mybatis提供了很多种读取配置文件的方法，但是最终都会调用build方法解析配置文件然后返回一个SqlSessionFactory对象



XMLConfigBuilder就是解析Mybatis配置文件的工具类，查看parser.parse()

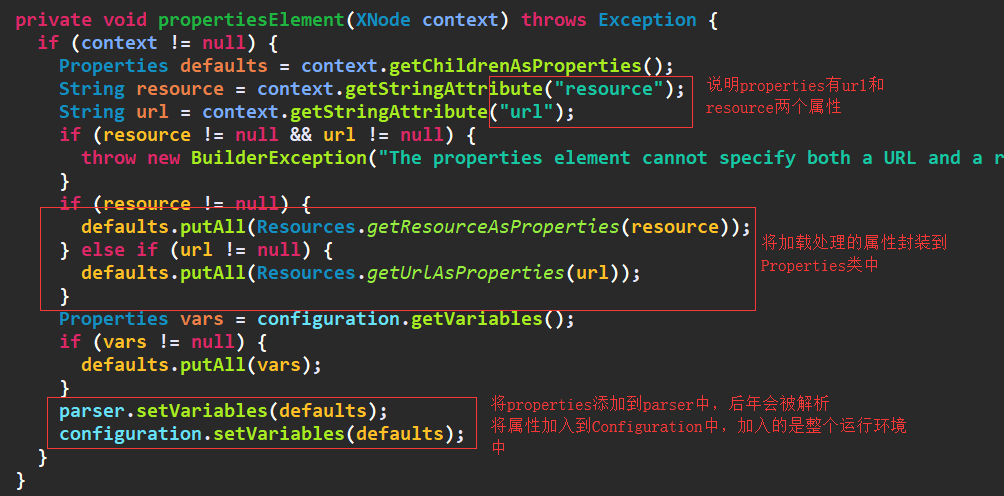




在配置文件标签的顺序也应该按此顺序写

Properties属性主要是加载外部资源文件





Environments用于定义Mybatis的运行环境，一个environments下可以有多个environment，但是在实际运行的时候取决于environments中的default属性，有多个environment，在开发和实际运行阶段就可以相互切换

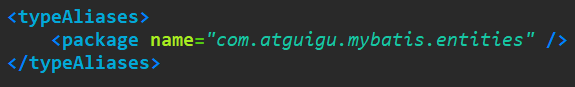
transactionManager：事务管理机制，Mybatis提供了两种事务管理机制，JDBC/MANAGED，使用JDBC时，Mybatis的事务管理由jdbcTransactionFactory提供，MANAGED表示托管的意思，一些服务器如WebLogic提供了事务管理的能力，因此可以使用MANAGED

dataSource的type表示使用数据库连接池，dataSource表示的就是数据库的连接



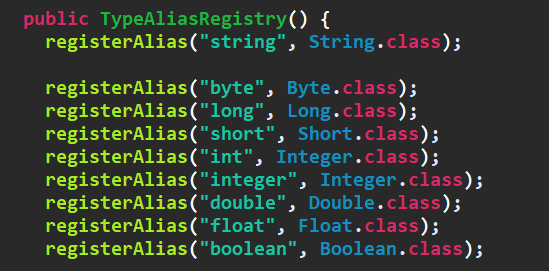


TypeAliase：设置别名，当Mybatis中需要使用类的全路径的地方，可以设置别名，直接使用类名即可





对别名了进行注册，而执行registerAlias方法是TypeAliasRegistry类，查看TypeAliasRegistry类的构造方法，可以发现TypeAliasRegistry对基本数据类型进制注册了，所以在Mybatis的很多地方我们都可以直接使用类名

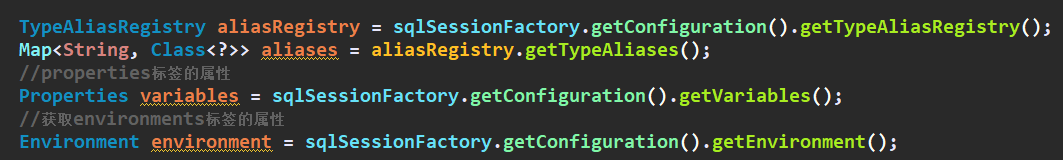






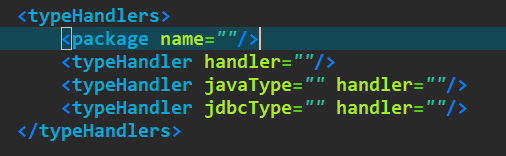
最终将别名和其对应的类的全路径保存到一个HashMap中

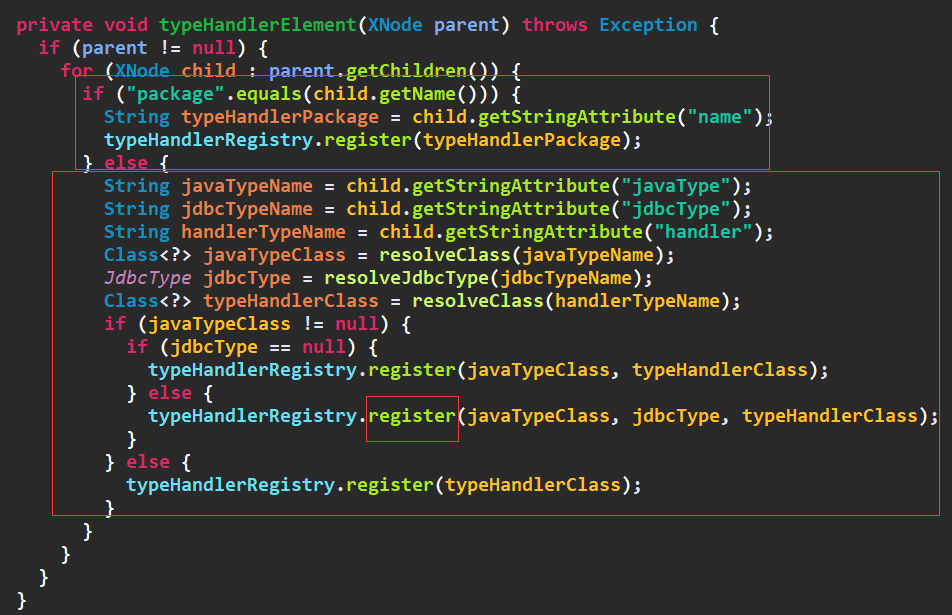
在很多地方我们都可以看到configuration的身影，其实Configuration中保存的就是配置文件的相关信息



可以通过sqlSessionFactory获取Configuration，并且可以获取配置文件的属性

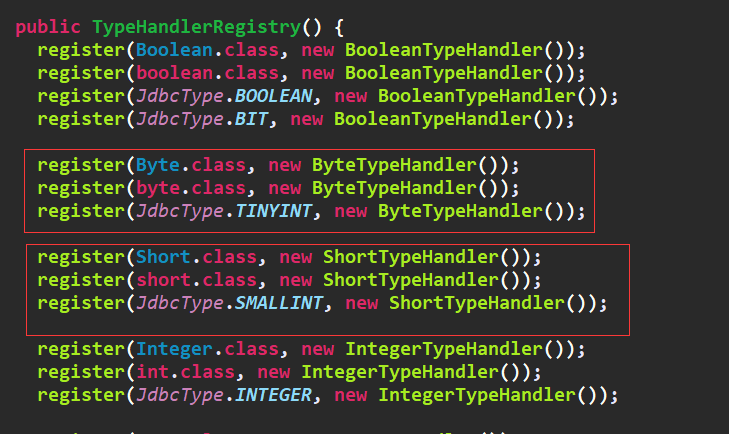
TypeHandlers：类型转换器，当Mybatis在预处理语句SQL(PrepareStatement)中设置一个参数时，还是从结果集取出值时，都需要一个合适的装换方式将结果转换为java类型，Mybatis默认了很多的TypeHandler，可以通过源码看看





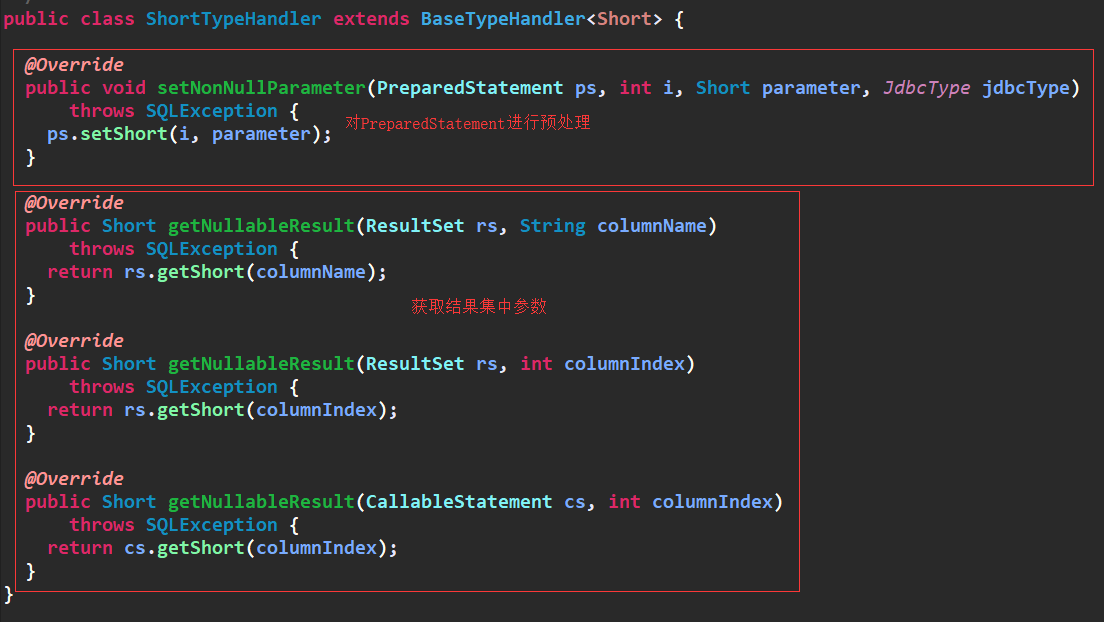
和设置别名类似，可以支持包和当个类型，，关键方法还是register

同样在register所在的类TypeHandlerRegistry的构造方法中声明了很多的默认构造器

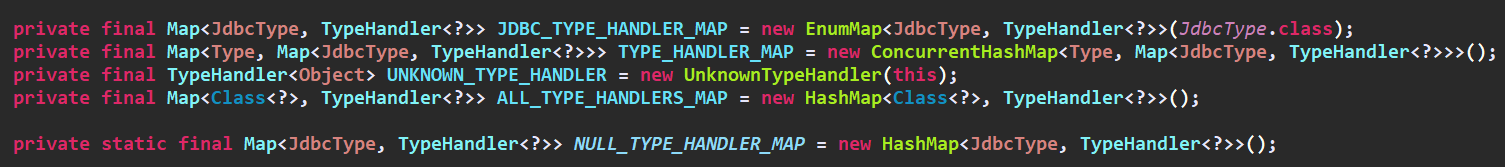


可以看出一个TypeHandler通常对应多个类型

一个是Mysql的数据类型，一个是java的基本类型及其对应的包装类，例如这里的ShortTypeHandler对应的是Java的short和数据库的SmallInt



可以看出，当需要自定义一个TypeHandler时，只需继承BaseTypeHandler即可，然后在配置文件中声明



不同类型的TypeHandler保存在不同Map中，有jdbc\_type\_handler\_map，type\_handler\_map，All\_type\_handler\_map等，主要是为了在查询的时候更加快捷

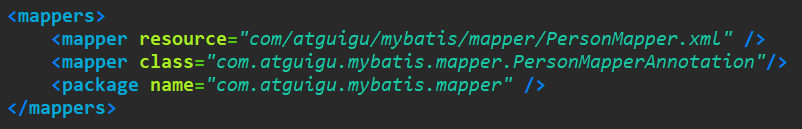


用于向MyBatis中定义一些插件，其实plugin就是Interceptor的作用，它可以拦截Executor、ParameterHandler 、ResultSetHandler 、StatementHandler的部分方法

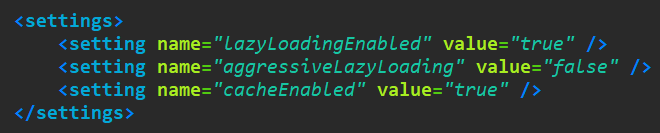
Executor就是真正执行SQL语句的东西，ParameterHandler是处理我们传入参数的，其实就是ParameterHandler在选择TypeHandler，ResultSetHandler就是处理返回的结果集

Mapper用于引用Mapper映射文件，就是让Mybatis用来建立数据表和JavaBean映射的一个桥梁，一个Mapper对应一个Dao

有如下三种配置方式，但是一般采用第三种方式



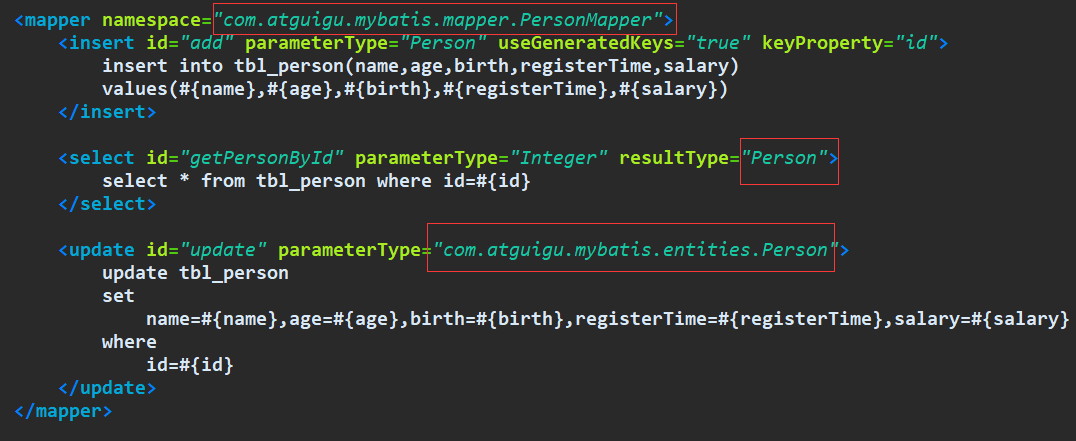
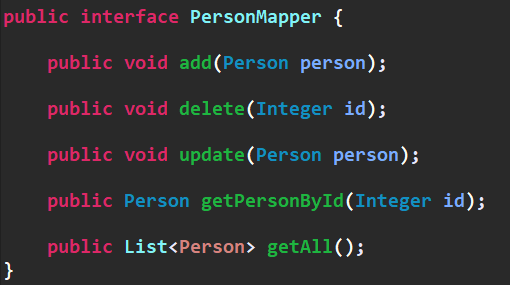
Settings属性用于定义全局变量



解析过程基本类似，最终都是将解析结果存放在Configuration中

更多全局变量：<https://www.cnblogs.com/lichangyun/p/8507900.html>

Mapper.xml映射文件：mappers作为根节点，有元素select、insert、update、delete、cache、cache-ref、resultMap

Mapper的namespace指定的是对应的Mapper(dao)，这里必须使用全类名，别名是只是针对mapper元素而言，因为要通过namespace创建Mapper的代理对象

先看看CRUD的操作：insert，update，delete

Id：对应的是dao的方法，必须有

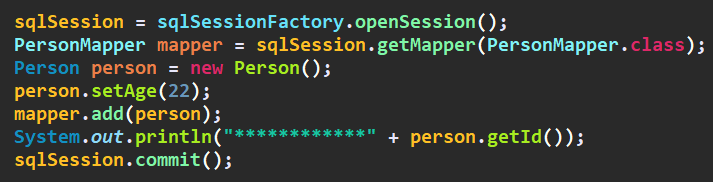
parameterType：对应dao方法的参数类型，可以是基本数据类型，也可以是JavaBean

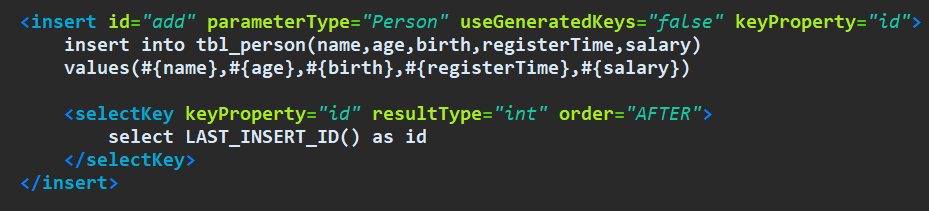
flushCache：执行当前SQL语句时，会导致本地缓存以及二级缓存刷新，默认为true，即将缓存中的数据进行同步

StatementType：PREPARED、STATEMENT、CALLABLE，默认值为PREPARED，让Mybatis分别使用Statement、PrepareStatement或

CallableStatement执行SQL语句

timeout：当前语句在多长时间内无结果就会抛出异常

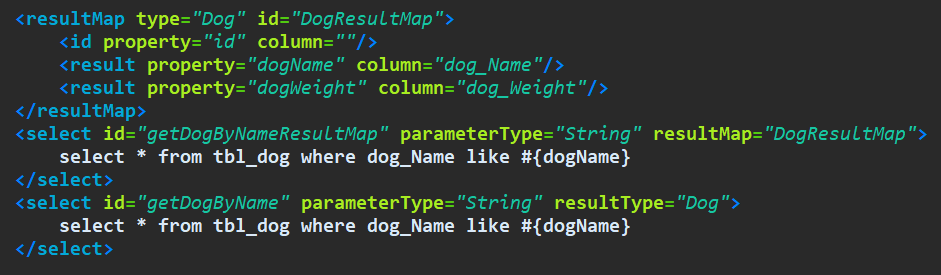




当在向数据库插入一条记录时，但是未插入主键，此时的主键是数据库自增的，比如这里只mapper.add(person)执行后数据库就对应着一条记录了，只是还未commit，如果此时想获取插入的主键，有两种方式获取：使用JDBC的getGeneratedKeys返回插入的主键或者在inser语句的selectKey获取；使用jdbc的getGeneratedKeys获取时，需要声明UseGeneratedKeys为true，并且

keyProperty为对应JavaBean的属性名，当使用selectKey时，需要添加SQL语句，order表示是在insert的语句执行前还是后执行selectKey中的语句

Select和resultMap：



Select用于查询操作，同样id是必须的属性

flushCache：结果同步到缓存，默认为false

useCache：是否使用缓存，查询结果将会被缓存，默认true

timeout：超时报错

fetchSize：每次批量返回结果的行数

statementType：让Mybatis分别使用Statement、PrepareStatement或CallableStatement执行SQL语句

ParameterType：方法出入参数的类型，可以是基本数据类型，也可以是JavaBean，也可以是HashMap

参数为基本数据类型表示查询条件单一

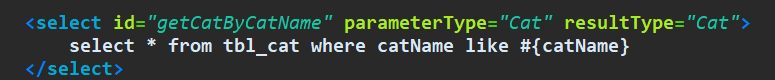
参数为JavaBean查询条件多，灵活

参数为HashMap实际上和JavaBean类似，HashMap<String, Object>()，String为属性名，而Object为对应的属性值

resultType：表示返回值的类型，一般都是对应的JavaBean

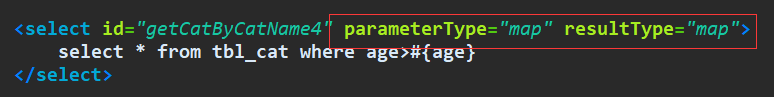
返回值：返回值可能有多个，在声明resultType的都声明成对应的JavaBean，返回值为多个时，需要声明List集合接收返回值，Mybatis会自动将其封装到List中，注意只能是List(有序可重复)



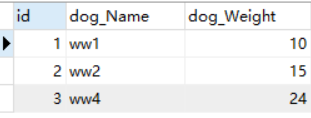
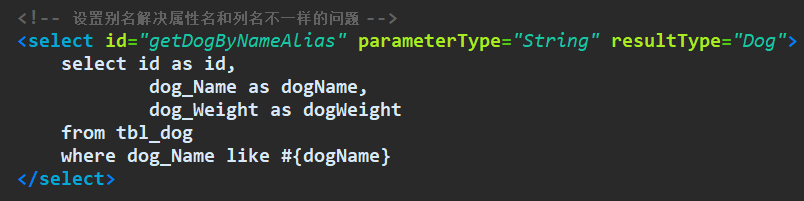


当ParameterType是一个Map时，resultType也应该声明为Map，出入参数是按属性名(String)：属性值(Object)查询的结果也不再是JavaBean，而是将JavaBean的属性和其对应属性值封装到Map中，多个JavaBean被封装到List中

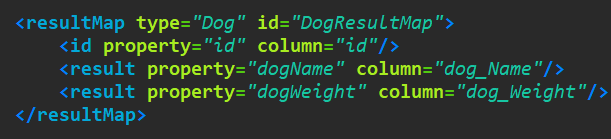


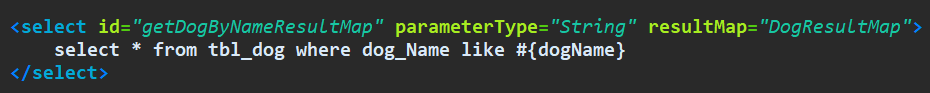


Mybatis是如何将查询结果封装到javaBean的：比如查询结果如下，Mybatis会调用resultType对应的JavaBean的set方法将属性注入到对象中，调用的set方法则是根据查询结果来找的，比如这里为dog赋值name属性时，会调用dog的setDog\_Name方法，但是如果Dog的属性名为name，那么就会导致查询结果不会封装到JavaBean中，此时可以使用AS，对查询结果名别名

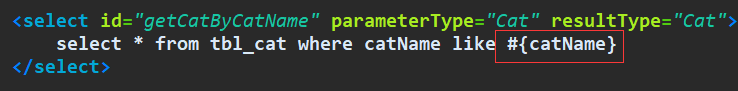
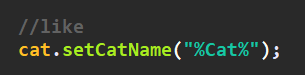
但是如果每次都这样，很麻烦，所以可以使用resultMap来实现声明一个结果集

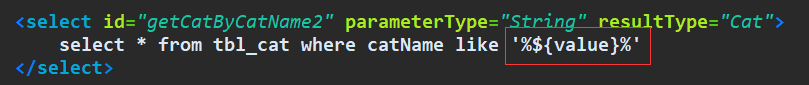




Type表示resultMap封装结果集的类型，那么就不能再定义select的返回结果为resultType，而是要定义resultMap，Mybatis会自动进行别名操作

模糊查询：Mybatis提供了两种实现模糊查询的方式#{}/${}

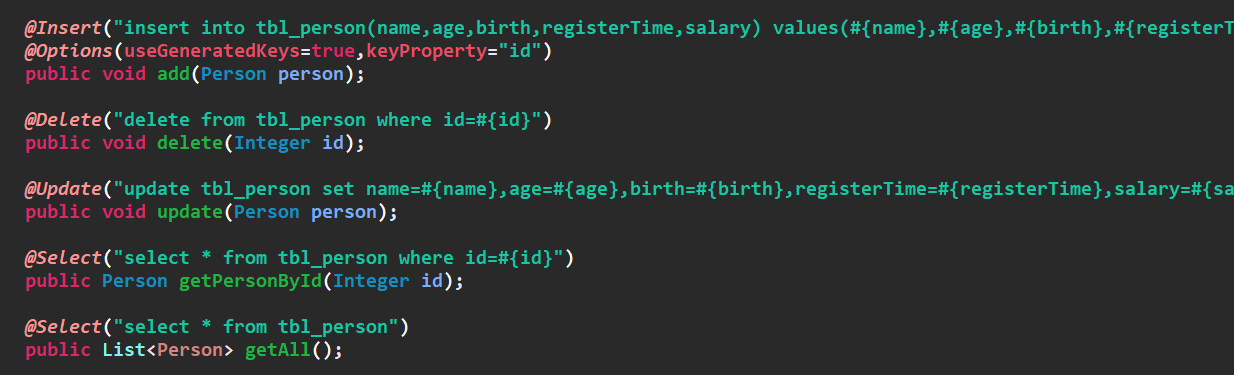
 



建议采用第一种，安全，传入的值直接就是%cat%

like CONCAT('%', #{consignee,jdbcType=VARCHAR},'%')：推荐使用这种

注解声明方式：除了通过在Mapper.xml中声明SQL语句外，Mybatis还提供了注解的方式



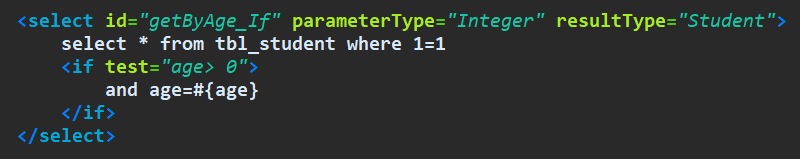
但是需要在Mybatis的配置文件中声明

<mapper class=*"com.atguigu.mybatis.mapper.PersonMapperAnnotation"*/>

其他使用方式基本一样，但是还是xml的方式好点

动态SQL语句：

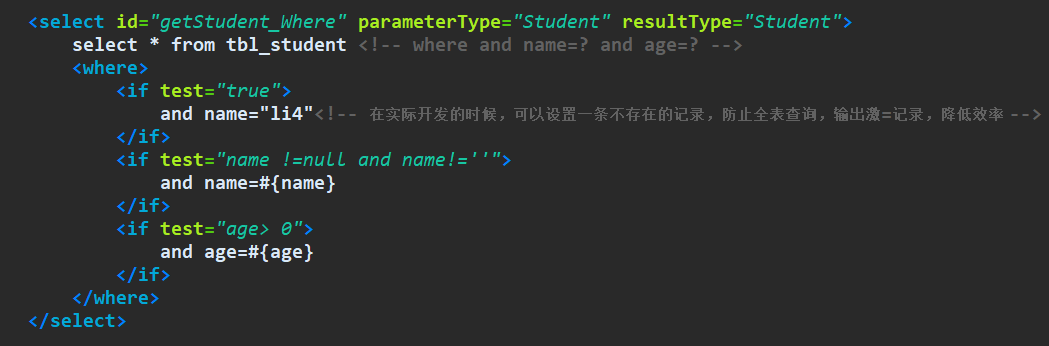
If：





注意：这里需要声明1=1或者1=0，因为当条件不满足时，就会导致SQL语句不完整而报错，所以添加1=1或1=0,1=1返回所有结果集，1=0返回为空，也可以不写1=1，但是这样会执行select \* from进行全表扫描，降低效率，其实1=1也会进行全表扫描，所以一般会设置一条不存在的记录

Where：

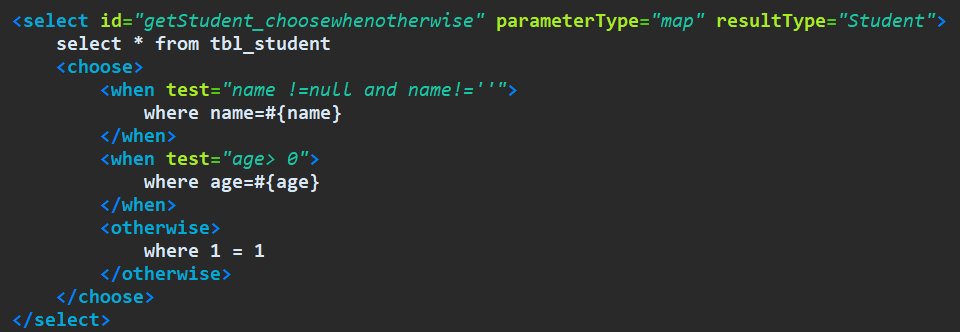




Where会自动添加where查询条件，并且将第一个if中的and/or给省略，test中的name为传入参数Student中的属性

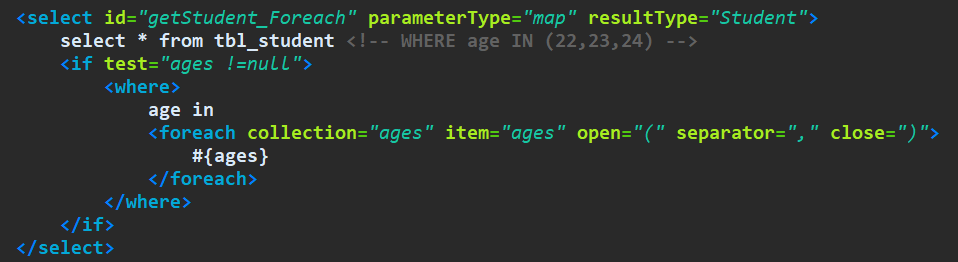
只有当条件满足的时候菜户添加where标签，where标签其实就是为了解决当条件不满足时，会导致SQL语句中有其他关键字，导致SQL语句出错，所以引入了where标签，，通过像set标签也是这样的

choose/when/otherwise：类似switch



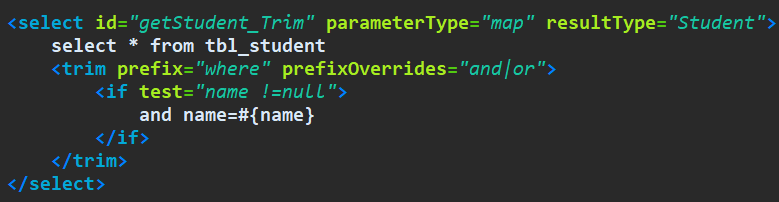
只有满足条件执行，不再判断下一条

Foreach：类似SQL中的in子句



这里不是遍历的意思，而是执行IN中所有条件

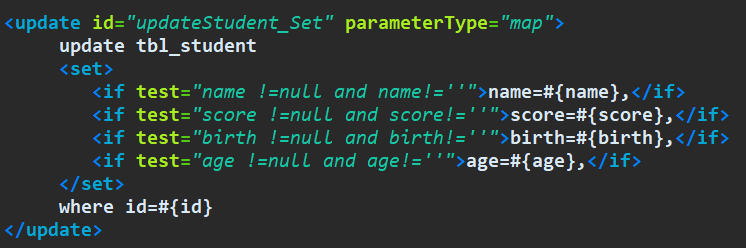
Trim：trim+if能代替where

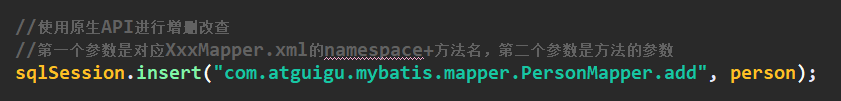


这句话的意思是if为真，那么and或者or将会被where覆盖掉：select \* from tbl where name=

Trim很好用

Set：set集合，执行多态判断

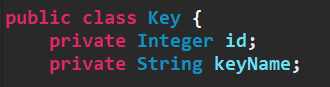
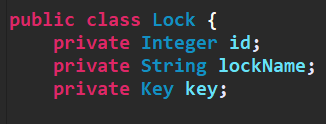




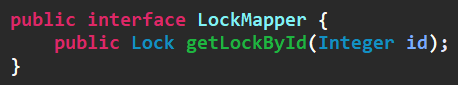
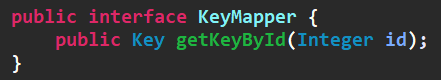
用得贼少，但是这种速度快

关联查询：

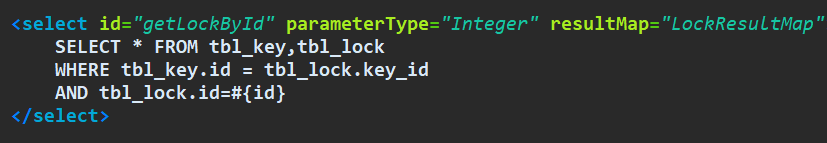
一对一

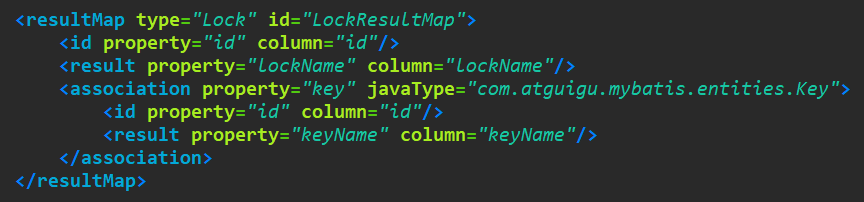


Lock可以找对对应的唯一的一把锁



通过Lock的id获取Lock以及其对应的Key

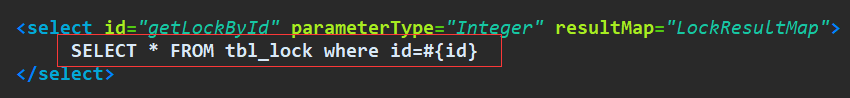


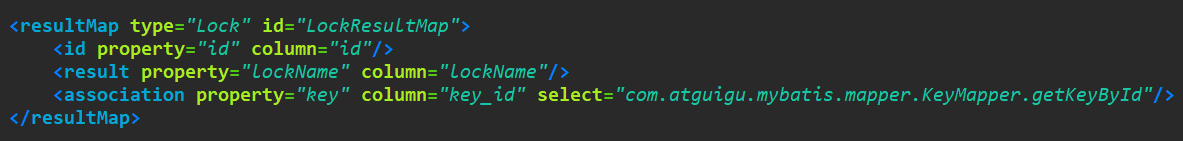


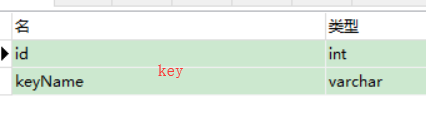
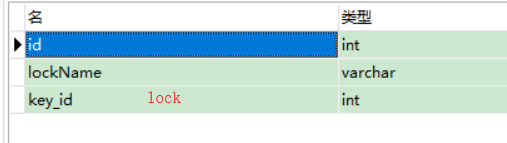
都存在关联查询了肯定就不能再使用resultType了，而要用resultMap

这里是同association去查询对应的锁

Association用于1:1关联查询，但是这种方式用得少，而使用select

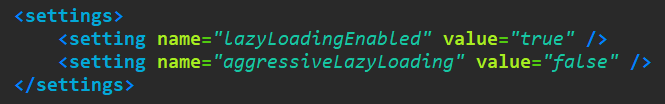






Lock表的key\_id是外键，对应key表的主键，第一种查询的SQL语句本身就进行了关联，实际本质上没有用到外键的思想，只发送了一条SQL，而第二种方式的SQL语句和一般的SQL一样，是通过外键去关联Key的，发送了两条SQL

延迟加载：Mybatis设置延迟加载



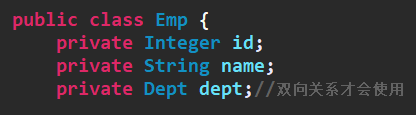
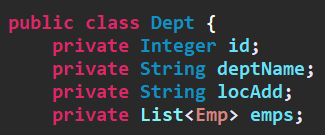
lazyLoadingEnabled：是否启动延迟加载，默认为true

aggressiveLazyLoading：当启动延迟加载时，如果aggressiveLazyLoading为true时，在加载延迟对象的某一属性时，会将整个对象加载出来，如果为false会按需加载，默认为true

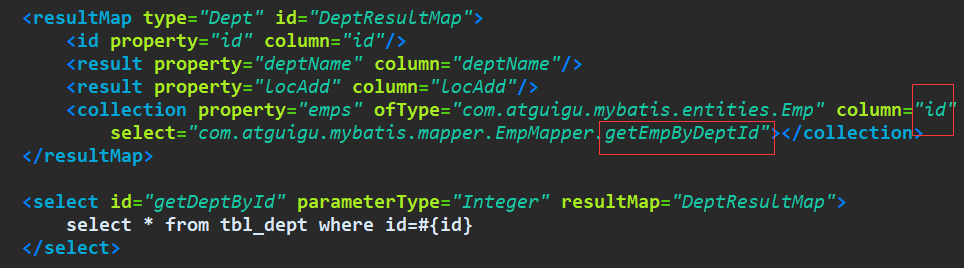
同样也可以采用注解的方式

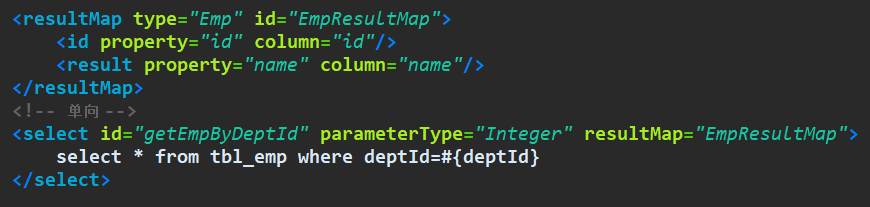
一对多：

单向一对多：获取一个Dept下的所有Emp



一个Dept有多个Emp





单向查询过程：先在Dept表中查询到Dept相应的信息，然后根据查询的Dept的id到Emp表中查询所有deptId为Dept的id的所有Emp，然后将结果封装为List

所以collection中的clumn为id(Dpet的id)

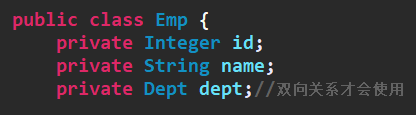
Collection表示一对多关联查询

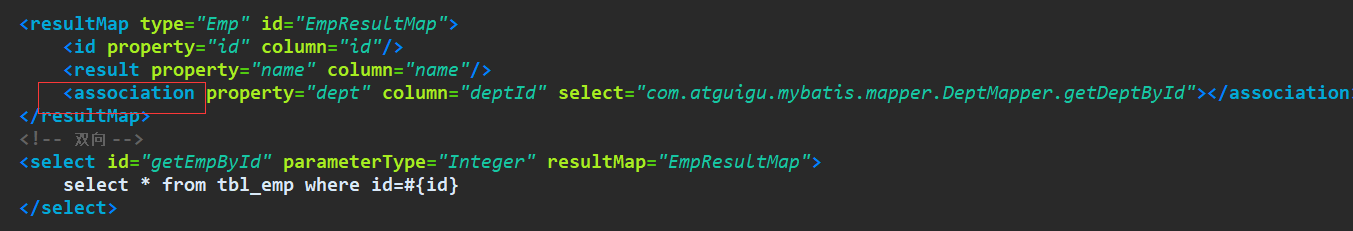
ofType：结果集中的对象类型

select执行查询的方法

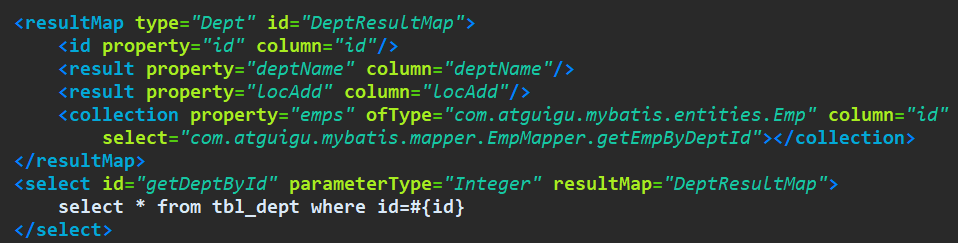
双向一对多：

现在要通过Emp找到其对应的Dept





根据Emp的id在Emp表中找到对应的记录后，然后根据deptId到Dept的表中查询Dept，这里在执行DeptMapper的getDeptById的时候，又会去执行Dept对应的所有Emp，按理是这样，但是这样会照成死循环，所有这里必须是association，一对一



查询缓存：

一级缓存：

默认情况下，Mybatis使用一级缓存，是sqlSession级别，在同一个sqlSession下，只要sqlSession不flush，不close执行相同的select语句都是从缓存里获取

Mybatis内部缓存是使用了一个HashMap，map数据结构需要key/value

key:hashCode+查询的SqlId+编写的sql查询语句

value：执行上面的查询所获得java对象

缓存有效的前提条件：

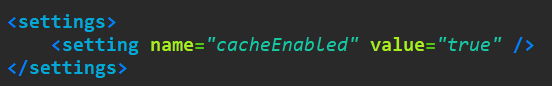
查询条件必须一样

sqlSession不能flush、不能close、不能clearCache

执行insert、update、delete操作后，也不会从缓存中获取，会出现幻读

执行insert、update、delete操作后会刷新缓存

二级缓存：



二级缓存需要开启



使用二级缓存的JavaBean必须实现Serializable接口，Mybatis在关闭sqlSession时，会将 sqlSession缓存的对象序列化到硬盘上



在XxxMapper.xml中声明哪些需要被缓存

创建FIFO缓存策略，每一分钟刷新一次，最大可以存512个对象，且只读

缓存策略就是当缓存对象超过size后，需要对缓存中的对象进行移出

FIFO：先进先出，即按缓存对象的顺序移出他们

LRU：最近最少使用(默认)

SOFT：软引用，移出软引用对象

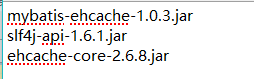
WEAK：弱引用

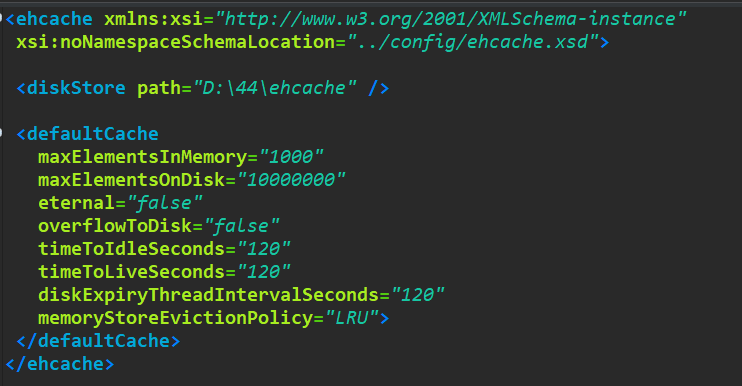
在执行完commit操作之后，都会刷新缓存，否则会出现脏读的情况

第三方缓存：

Mybatis有自带的一二级缓存，但是性能不好，所以可以使用第三方缓存

EhCache可以用来代替Mybatis的二级缓存

添加Jar包



配置ehcache.xml



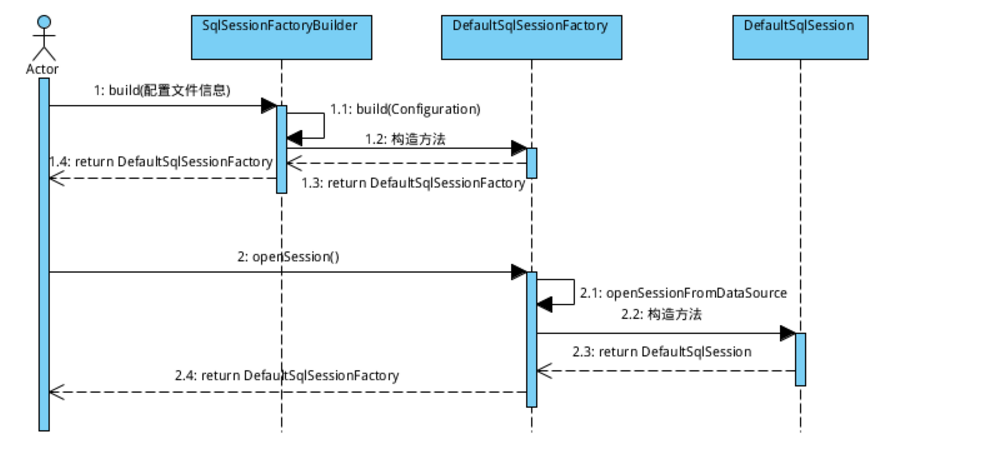
配置属性说明

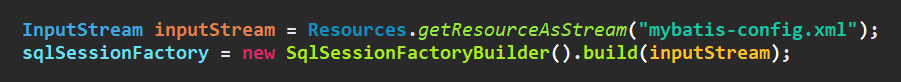


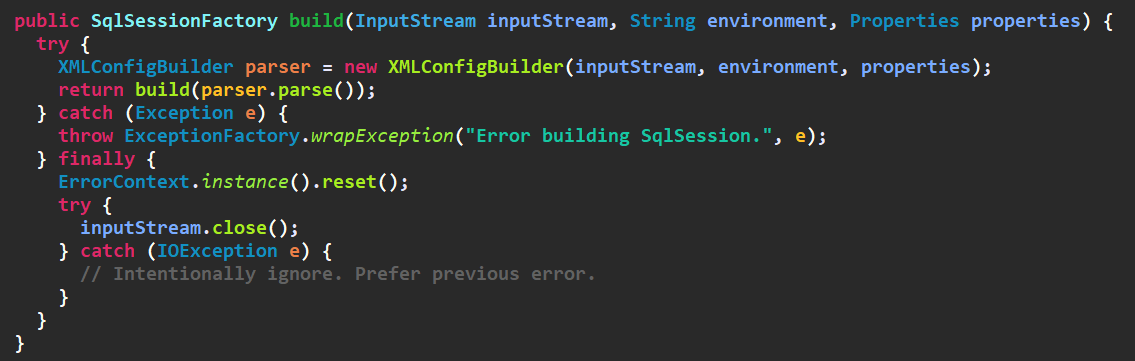
在XxxMapper中声明第三方缓存

Executor是在创建SqlSession时创建的，一级缓存BaseExecutor和二级缓存CachingExecutor的创建都是在SqlSession创建时创建的，但是一级缓存BaseExecutor是基于SqlSession的，而CachingExecutor内部有一个Cache，每次在查询的时候都会先获取Cache，而Cache的创建是在sqlSessionFactory中创建的

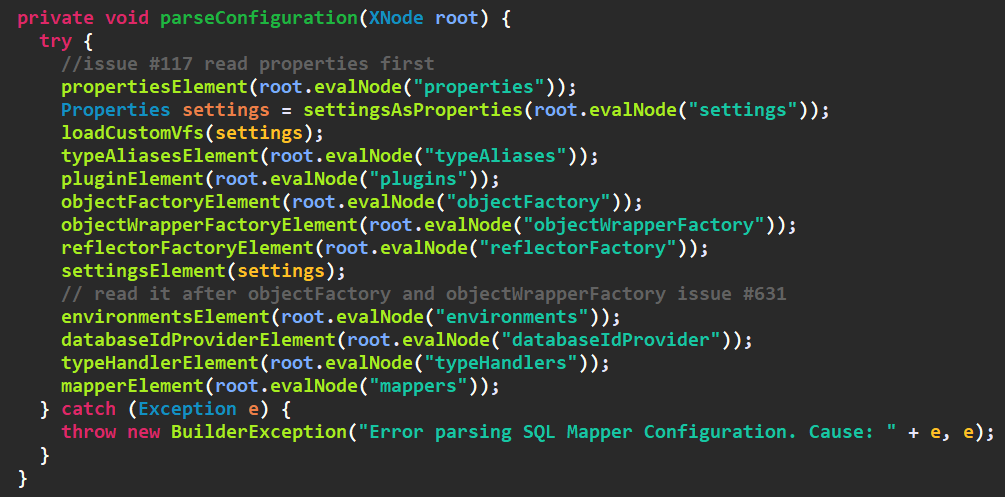
Mybatis运行大体流程：



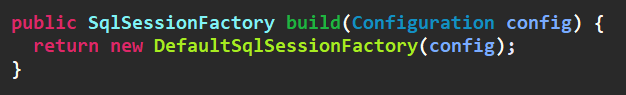




Build的作用是创建一个sqlSessionFactory，parser.parse的作用是加载配置文件



Parse中调用parseConfiguration解析Mybatis.xml中声明的标签(properties,typeAliases,mappers等)，往下解析会发现加载后信息会被封装到一个Configuration中(XMLConfiguration用于解析配置文件，Configuration保存配置文件信息)



加载完配置信息后，就可以获取sqlSessionFactory，这里返回的是DefaultSqlSessionFactory

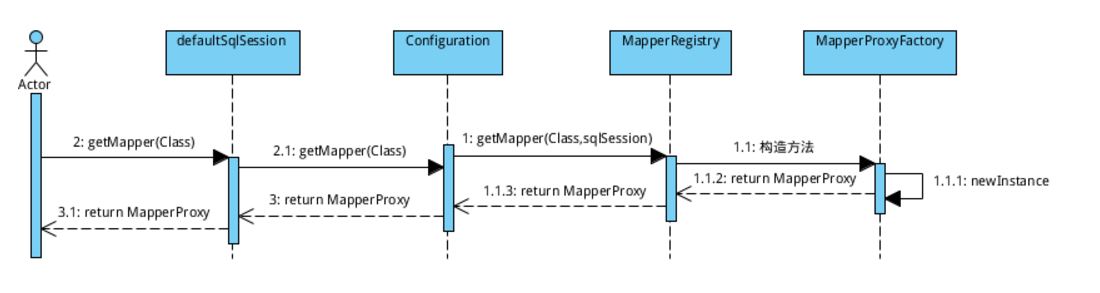


获取到sqlSessionFactory后，就可以获取sqlSession



在sqlSession中包含了Configuration配置信息和executor(**Executor是全局的[在setting中指定]，在创建SqlSession的时候就确定，Statement是针对具体方法，在标签中指定**)

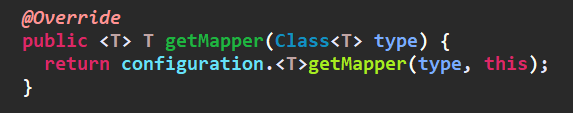
**Executor在获取SqlSession的时候创建**



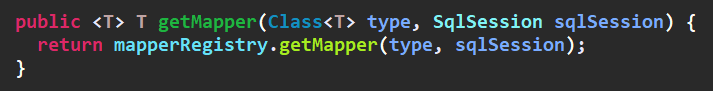
获取到sqlSession后，就是执行CRUD操作

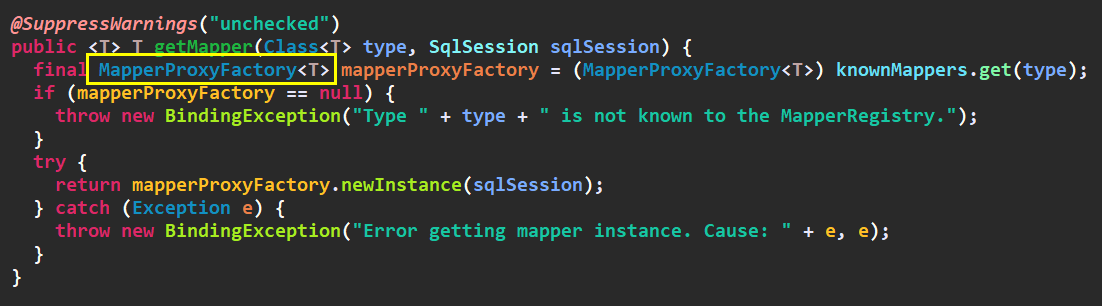
在Mybatis中实际使用通过MapperProxy动态代理创建dao(JDK动态代理，接口)



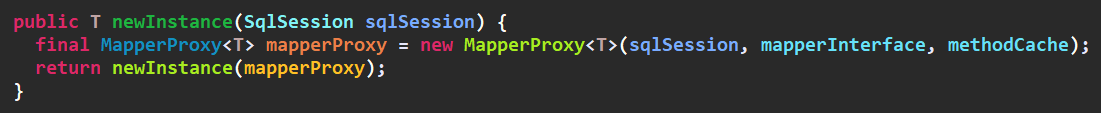


可以看出使用从configuration中获取的





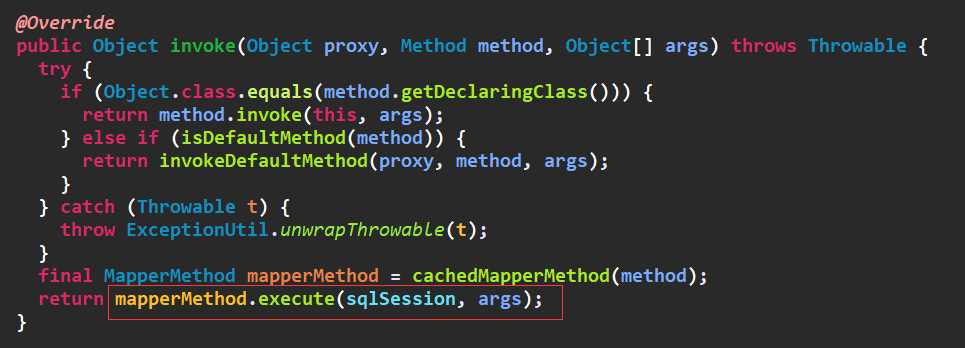
这里创建的使用一个代理的MapperProxyFactory，听过factory获取MapperProxy



创建好代理对象后，就只执行方法



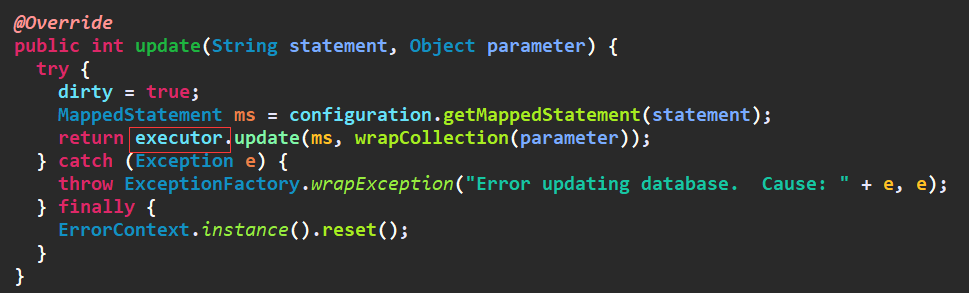




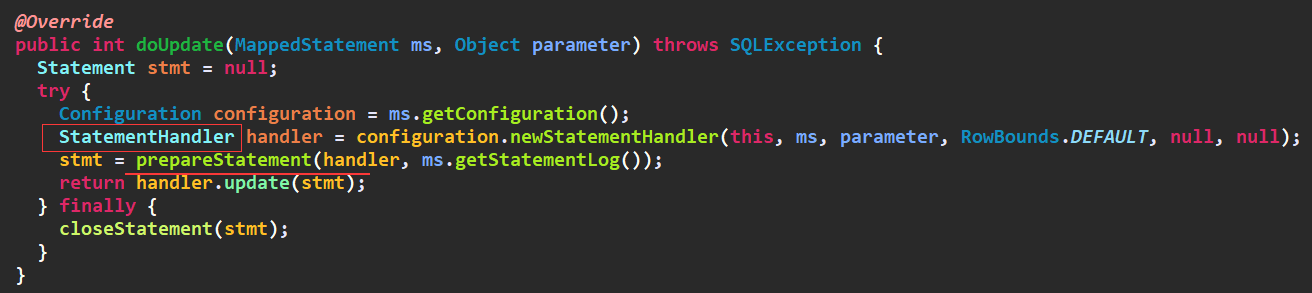
既然是代理对象，在执行方法的时候，肯定执行的是代理对象的invoke方法，这里有一个缓存方法的操作

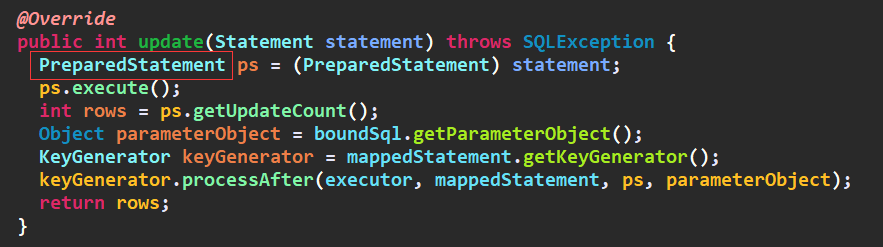


分类执行CRUD操作，最终又回到了操作sqlSession上



这里以update为例，最终执行update操作的是executor



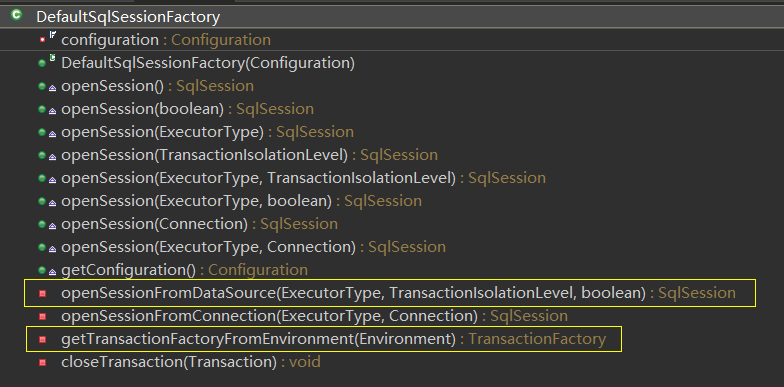


从PrepareStatement可以看出，最终还是调用的使用jdbc的相关操作，execute实际上执行的是jdbc的操作

源码分析：

SqlSessionFactory

sqlSessionFactory用于获取sqlSession，其常用实现类是DefaultSqlSessionFactory，获取sqlSession的方法是调用DefaultSqlSessionFactory的openSession方法



openSession方法最终都会调用openSessionFromDataSource方法获取SqlSession，参数说明

ExecutorType：Executor的类型，用于执行SQL语句，ExecutorType是一个枚举类型，有3个值(SIMPLE，BATCH，REUSE)，分别对应SimpleExecutor，BatchExecutor，ReuseExecutor

SimpleExecutor：简单的执行类，不会任何处理就执行sql

BatchExecutor：批量执行，和JDBC中addBatch一样，BatchExecutor底层实际调用的就是addBatch

ReuseExecutor：重复使用执行，其定义了一个Map<String, Statement>，将执行的sql作为key，将执行的Statement作为value保存，这样执行相同的sql时就可以使用已经存在的Statement，就不需要新创建了

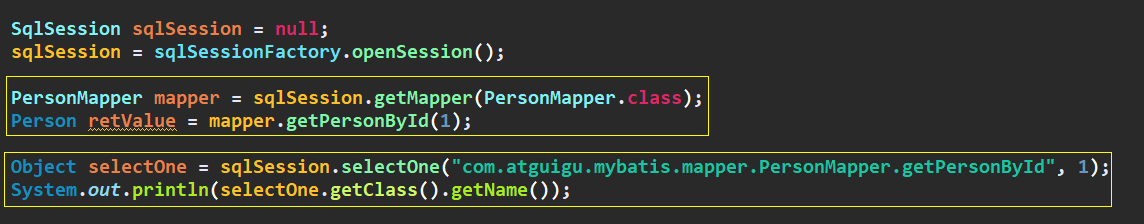
Mybatis还有一种CachingExecutor，首先会去缓存中获取，没有再去执行一般的Executor(SIMPLE，BATCH，REUSE)，通过openSession获取的SqlSession只能是一般的Executor

TransactionIsolationLevel：这也是一个枚举类型，声明的事务的隔离级别

Boolean：是否自动提交(autoCommit)

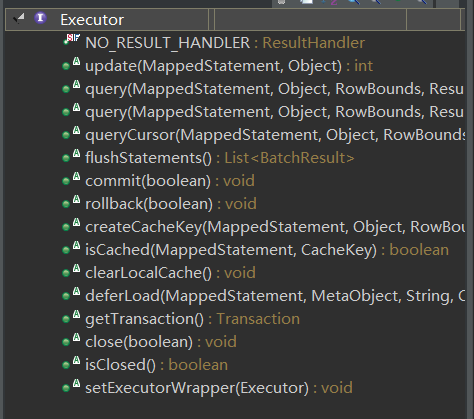
getTransactionFactoryFromEnvironment方法用户获取事务管理：从配置文件中获取，事务的管理由谁执行，JDBC/MANGED(服务器自带)/不支持事务

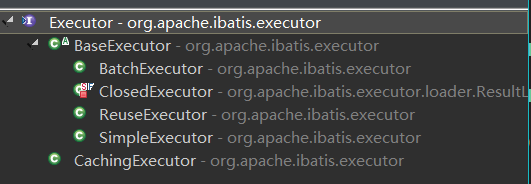
SqlSession中包含了多执行CRUD的方法，最终都是执行的是这些CRUD方法



getMapper的作用实际上就是创建一个MapperProxy代理对象，目的就是为了会去getPersonById方法的路径，然后找到对应的Mapper.xml的SQL语句。而在最终都是会调用SqlSession的<T> T selectOne(String statement, Object parameter)方法(这是针对getPersonById而言)(所以方式2的速度更快)，PersonMapper.getPersonById只是为了获取SQL

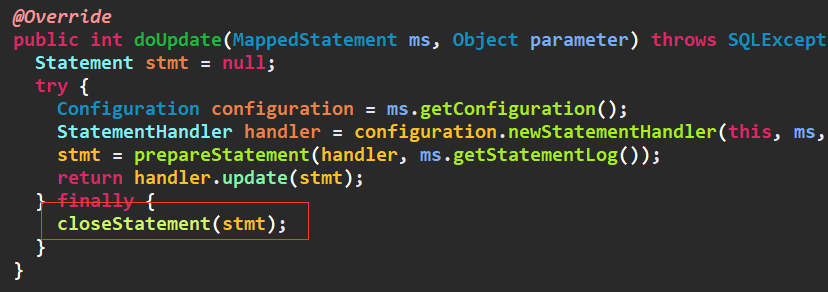
Executor：用于执行SQL语句，主要提供了update，query方法以及事务相关操作





Executor公有如下几个实现类，BaseExecutor对Executor中的大部分方法进行了实现，doUpdate，doQuery等方法在各自的实现类中实现

SimpleExecutor：一个简单的执行类，并不会做任何处理就执行SQL，每次都创建新的Statement，用完就关闭

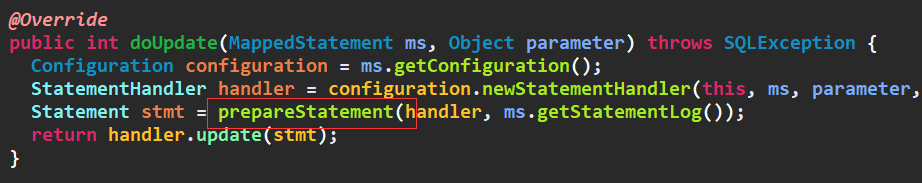


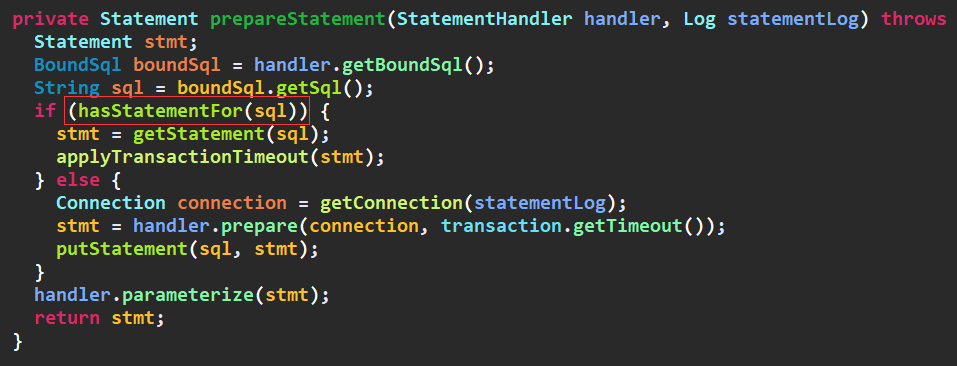
BatchExecutor：批量操作，通过批量操作提高性能，就是JDBC的addBatch方法

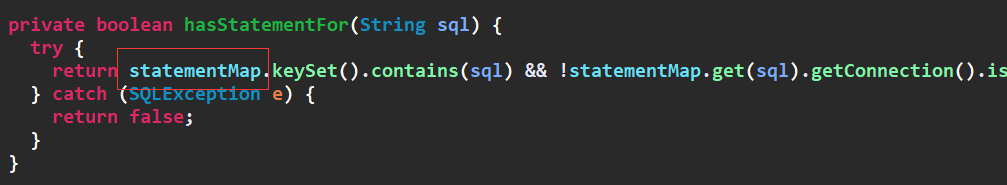


ReuseExecutor：顾名思义就是重复使用执行，其定义了一个Map<String, Statement>，将执行的sql作为key，将执行的Statement作为value保存，这样执行相同的sql时就可以使用已经存在的Statement，就不需要新创建了

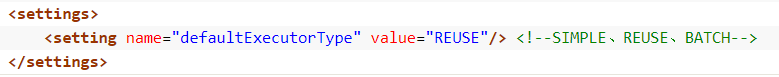






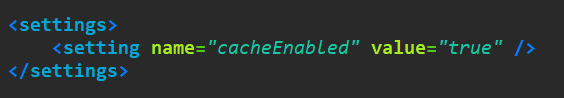


可以在Mybatis的配置文件中声明使用哪种Executor(默认是simple)



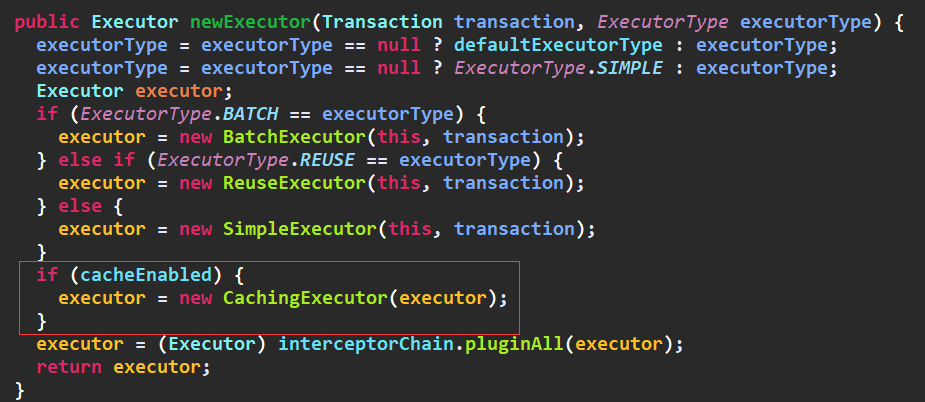
如何查看defaultExecutorType的默认值呢： Configuration

CachingExecutor：直接实现接口中的方法，其实这个就是二级缓存，而BaseExecutor是一级缓存

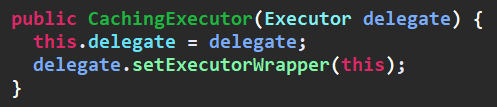


可以在配置文件中声明是否启动，默认启动

默认开启二级缓存，又有一个defaultExecutorType，到底用那一个？



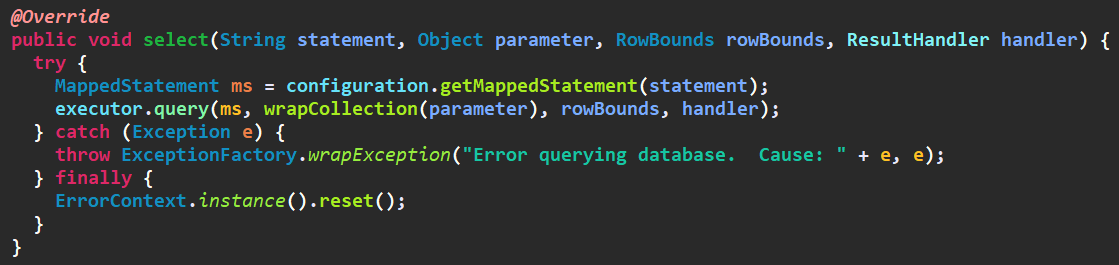
看看Configuration中创建Executor的方法，先根据defaultExecutorType创建一个Executor，然后判断是否开启二级缓存



所以在CachingExecutor有两个Executor delegate

在来看看查询和更新操作

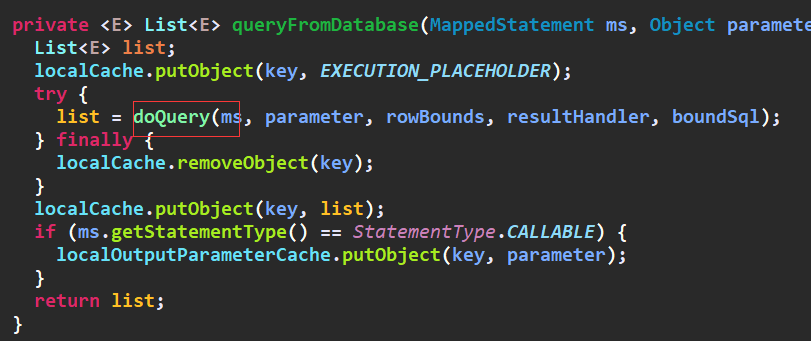
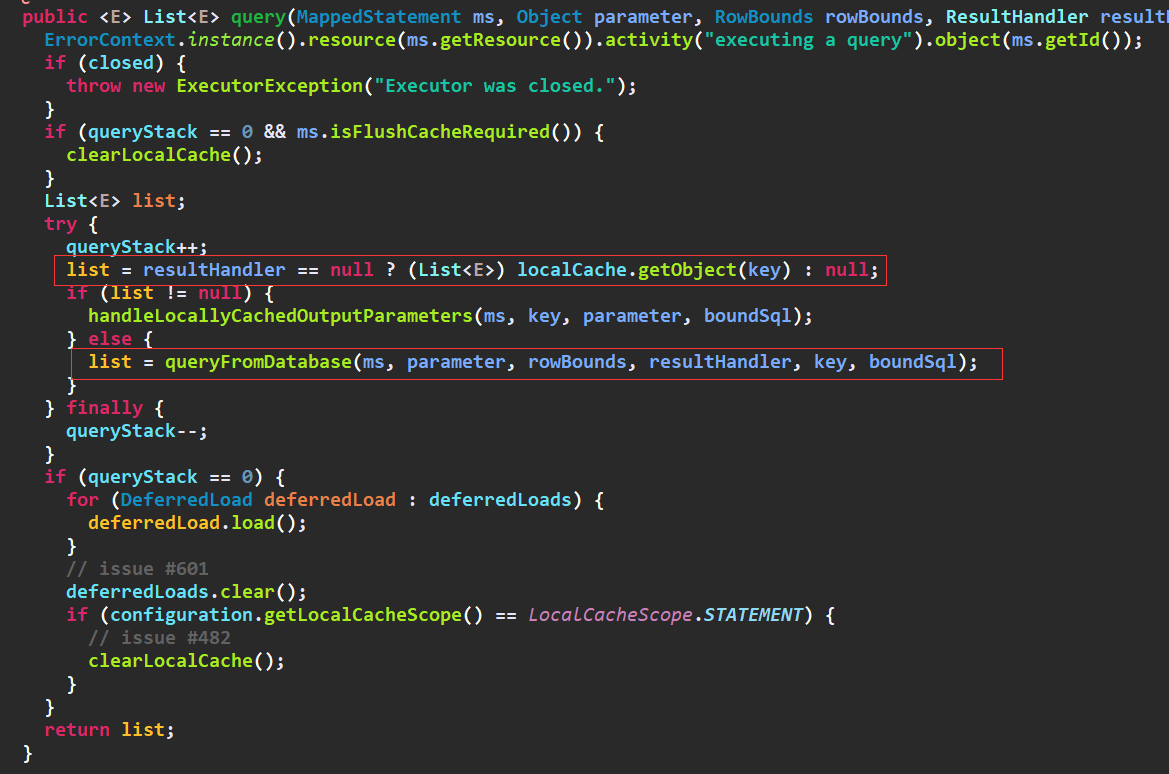
查询：缓存只会存在查询操作中



在DefaultSqlSession中查找任何一个查询方法

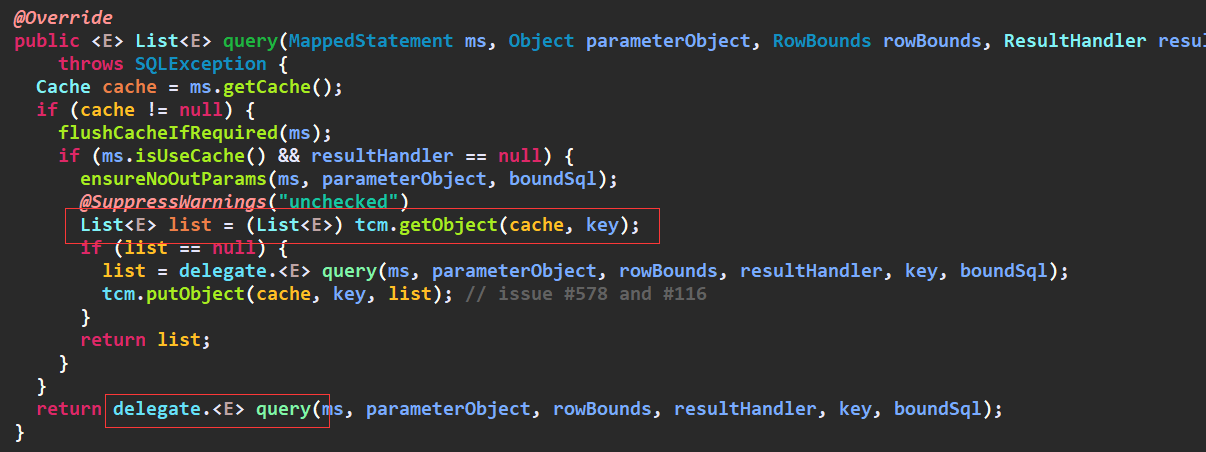
Executor.query只会出现在BaseExecutor和CachingExecutor中

如果没开启二级缓存：



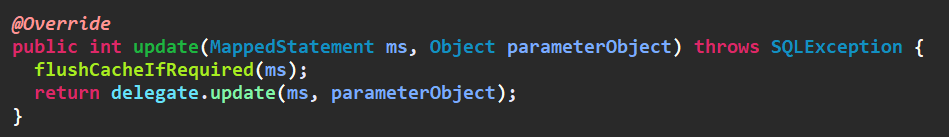
先会到缓存中找，这里的query方法是BaseExecutor的，所以一级缓存是BaseExecutor的，其子类都有一级缓存，如果缓存中没有，在调用子类的的doQuery方法

如果开启二级缓存：



会先到二级缓存中找，如果没找到再到delegate[委派](Executor)中查询，而这里的delegate就是BaseExecutor的子类，有会到了上面的操作

更新操作：

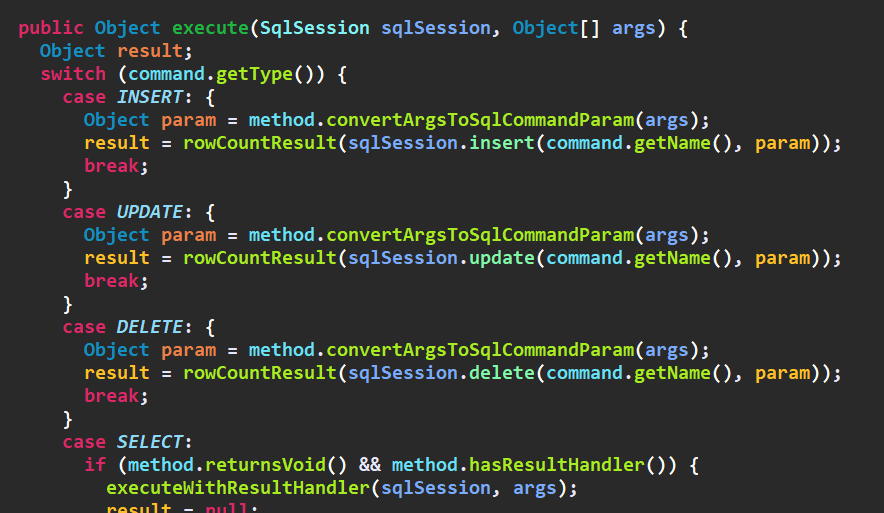


CachingExecutor的更新操作直接调用的是BaseExecutor的方法，往下继续走，会发现更新操作具体执行者是BaseExecutor的子类(因为只有查询操作才会到缓存中查询)

BaseExecutor的doQuery，doUpdate，doXxx等方法的实现都是在子类中，而update，query等方法在BaseExecutor中实现

<https://blog.csdn.net/shewmi/article/details/80283094>

MapperMethod：封装了SqlSession中的操作，当执行代理对象的方法时，实际上执行的是这里的execute方法



MapperMethod的execute方法是关键，选择SqlSession中合适的方法执行操作



Select操作情况比较多，根据返回类型和个数执行SqlSession的不同方法

MapperProxyFactory，MapperProxy，MapperMethod，MapperRegister

MapperProxyFactory用于创建MapperProxy

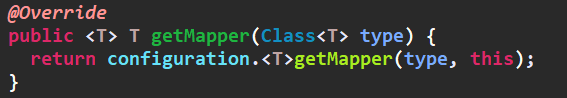
MapperProxy为sqlSession.getMapper(mapper.class)mapper的代理对象，通过此代理对象可制作SQL操作

MapperMethod对sqlSession中的方法进行封装，关键方法为execute，根据标签类型(insert、select)执行相应的方法，然后在相应的方法里通过Executor执行对应Executor的方法

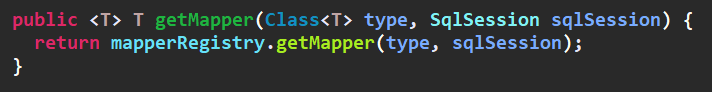
MapperRegister用于注册Mapper接口

获取MapperProxy的过程：

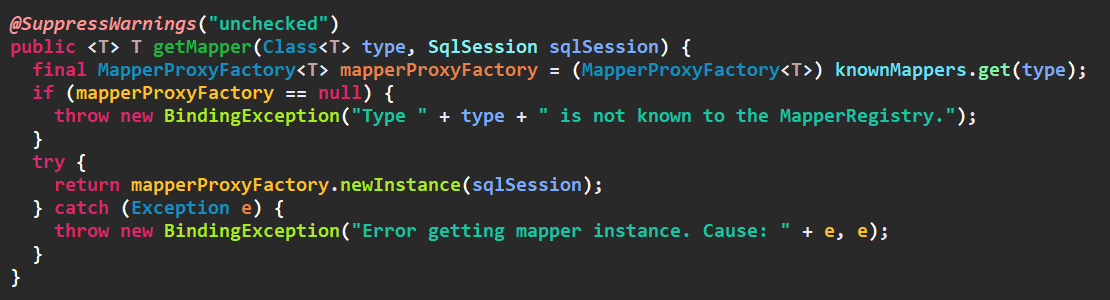
先调用sqlSession的getMapper



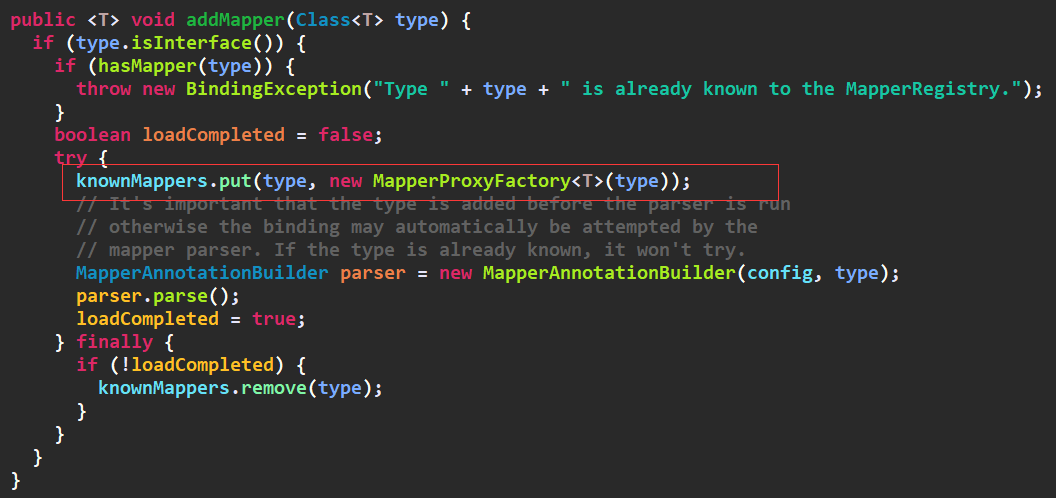
在configuration中获取



通过MapperRegister获取



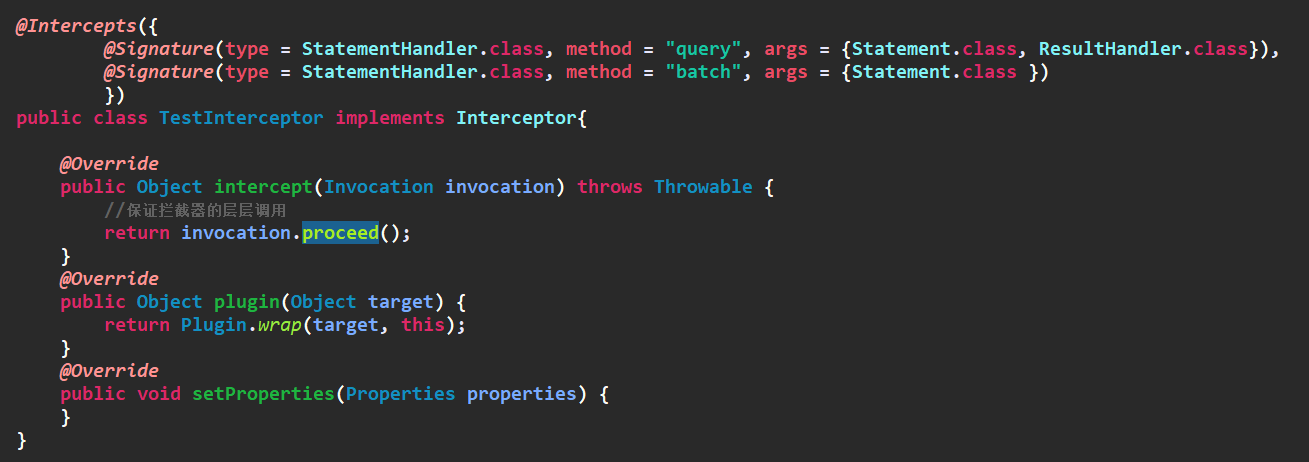
先从MapperRegister的knownMappers中获取MapperFactoryProxy，然后通过MapperFactoryProxy创建MapperProxy



在MapperRegister中有一个addMapper法，在加载配置文件的时候，回去加载mapper.xml，此时每个接口都对应一个MapperProxyFactory保存在knownMapper中，type是接口的全类名，和其对应的MapperProxyFactory(knownMapper是一个HashMap结构)

Mybatis自定义拦截器：Mybatis中拦截器就是plugin

实现例子：

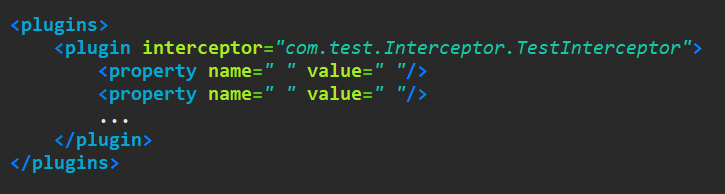


必须声明@Interceptors和@Signature注解，因为Plugin的wrap方法会读取注解里的参数，一个Interceptors里可以有多个Signature

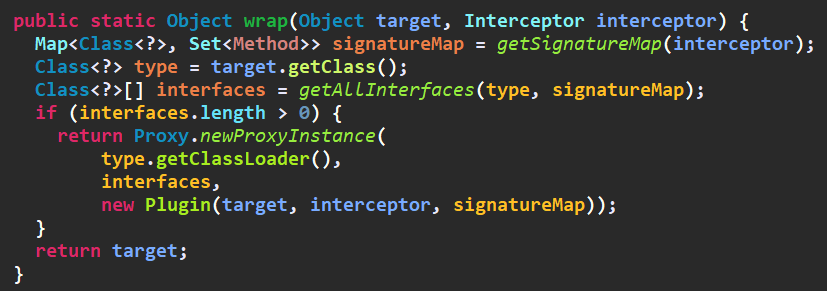
@Interceptors表示这是一个拦截器，@Signature声明的是什么时候会执行此拦截器

声明了拦截器后，还需要在Mybatis的配置文件中声明，Property表示此拦截器需要一些参数声明

**TYPE指定拦截的对象**

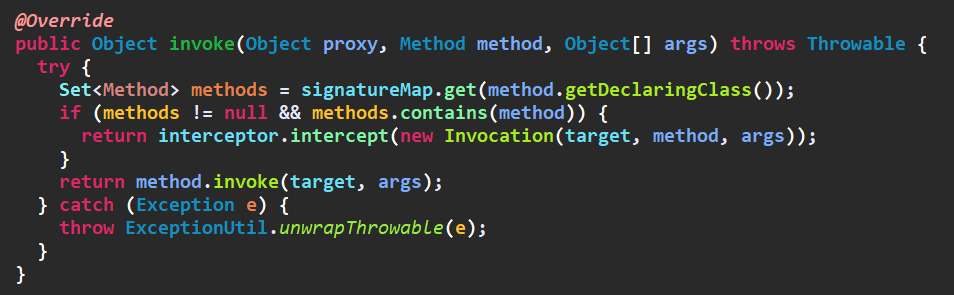


Plugin.wrap(target, this)方法：创建代理对象

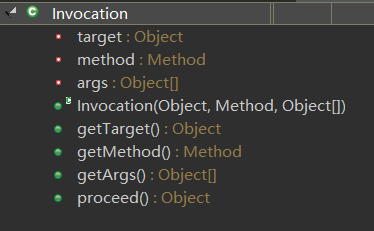


Target是目标对象，interceptor为拦截器，signatureMap记录需要被拦截的类与方法，生成的代理对象的InvocationHandler为

new Plugin，所以在执行代理对象的方法时，执行的是Plugin的invoke

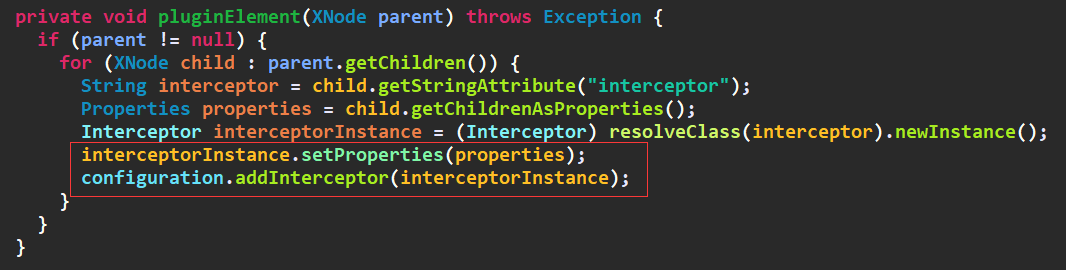


intercept(Invocation invocation)方法：拦截器的核心方法，拦截器的逻辑实现

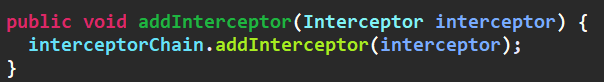


Invocation的target用于获取目标对象(比如这里的StatementHandler)，method为目标方法，args为方法参数

在Mybatis中只会对Executor、ParameterHandler、ResultSetHandler和StatementHandler的实现类进行拦截

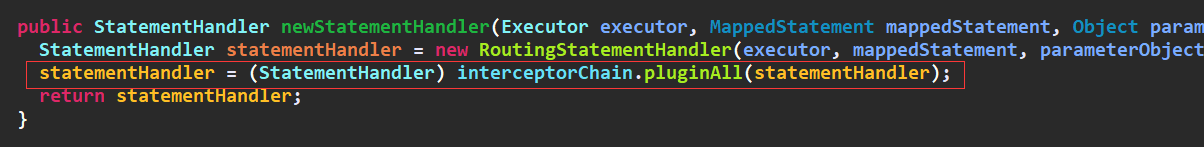


在加载Mybatis的配置文件时，会将声明的Interceptor加入到configuration中

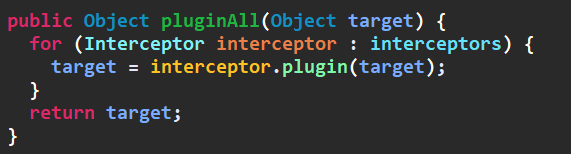


而configuration又将Interceptor保存在一个InterceptorChain中

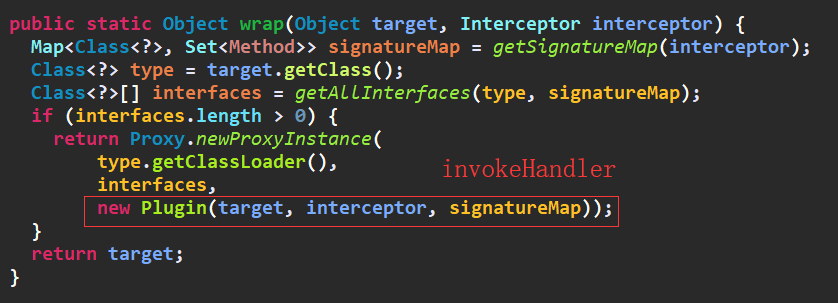
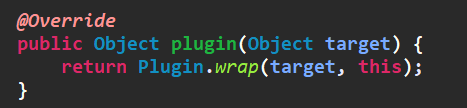
那为什么Mybatis只会对4个类进行拦截



StatementHandler等4个类的初始化是在Configuration中进行初始化的，在初始化的时候，会去判断是否在此类上加载拦截器



此时会调用InterceptorChain的pluginAll方法去为target(StatementHandler)创建代理对象，此时的Interceptor为自定义的TestInterceptor，在执行interceptor.plugin(target)实际是执行自己的plugin

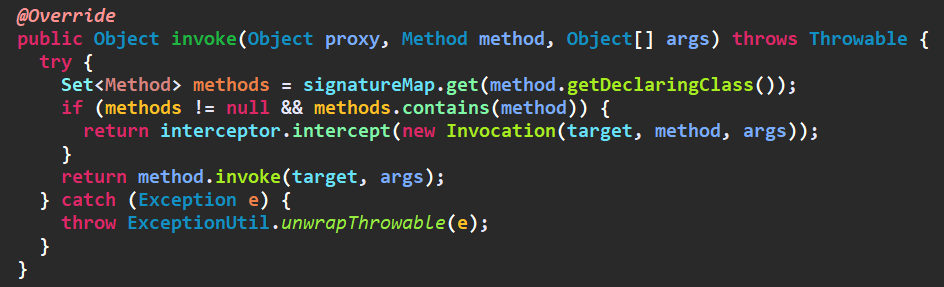


通过Plugin的wrap创建代理对象，所以返回的Object是StatementHandler的代理对象，代理对象的invokeHandler为Plugin

而只有Executor、ParameterHandler、ResultSetHandler和StatementHandler的实现类在初始化的时候会去加载InterceptorChain，所以只能对这4个类进行拦截

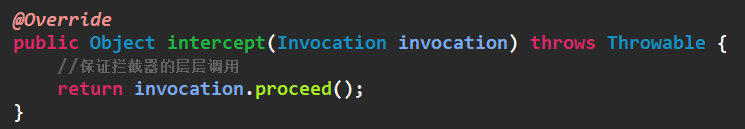
**拦截器创建的是代理对象**

运行过程：当在执行StatementHandler的query方法时，此时的StatementHandler对象已经是一个代理对象了，所以执行的是代理对象invoke方法(Plugin的invoke方法，Plugin为invokeHandler)



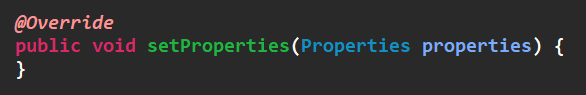
先会判断执行方法方法是否被拦截如果没被拦截，直接执行目标方法

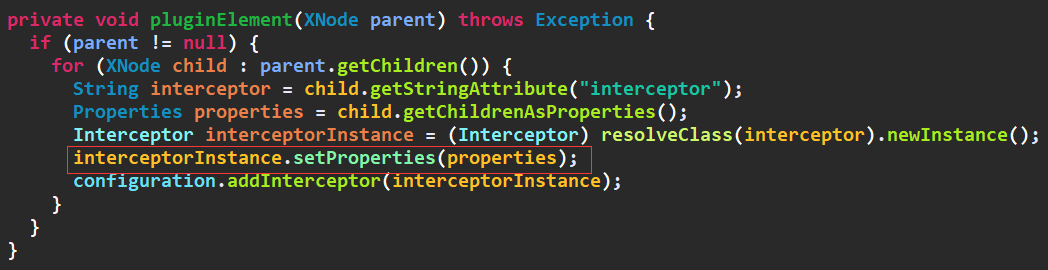
如果被拦截，会执行interceptor的intercept方法

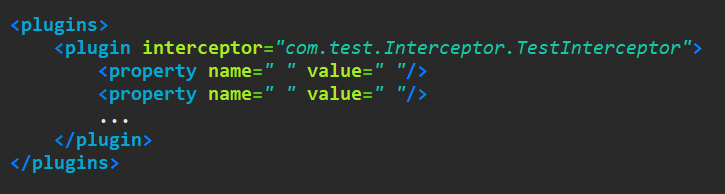


这个方法中实现拦截器里逻辑，Invocation可以获取目标对象，方法和方法参数

这里**返回值必须是invocation.proceed，保证拦截器的层层调用**(执行了代理对象的方法后就不会在执行原来的方法)，多个拦截器就会在代理对象的基础上再创建代理对象



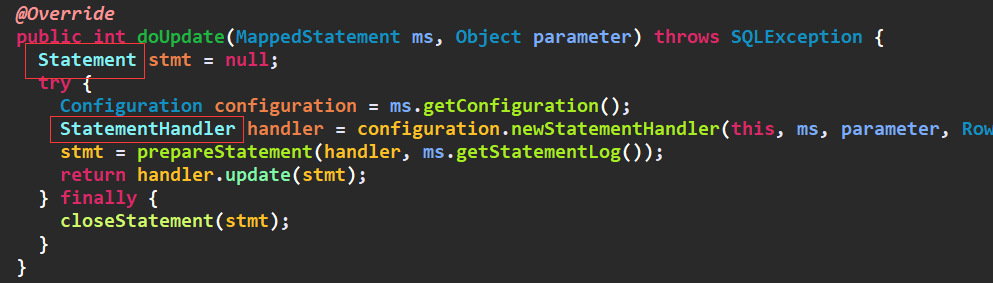


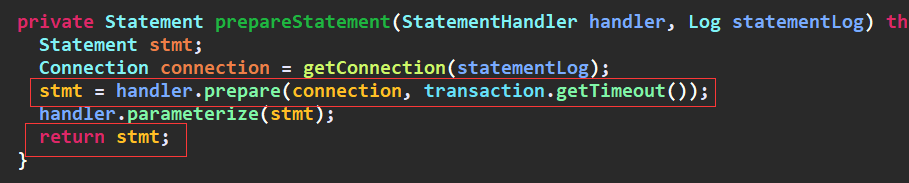


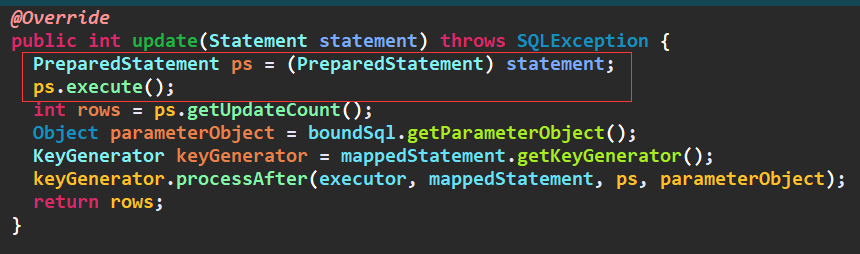
Interceptor的setProperties方法在可以获取plugin声明参数

自定义拦截器需了解Mybatis的运行流程以及对源码非常了解

StatementHandler：StatementHandler的创建是在**Executor的方法**里创建的



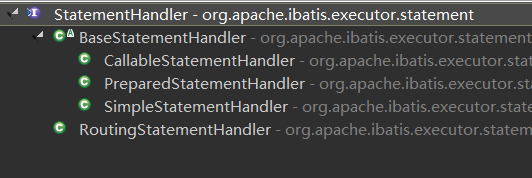




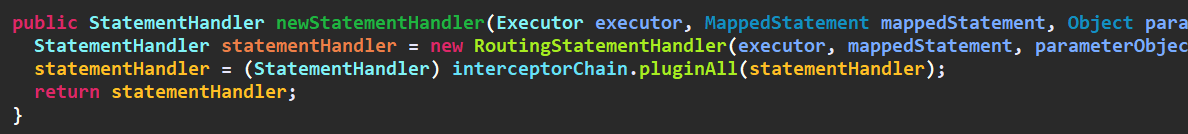
StatementHandler的作用是为实际执行SQL操作的Statement作一些初始化操作，如connection，参数准备parameterize，这些操作是在BaseStatementHandler中进行的，将准备好的结果封装到Statement中，就等执行execute，StatementHandler中还包含了操作具体Statement的方法

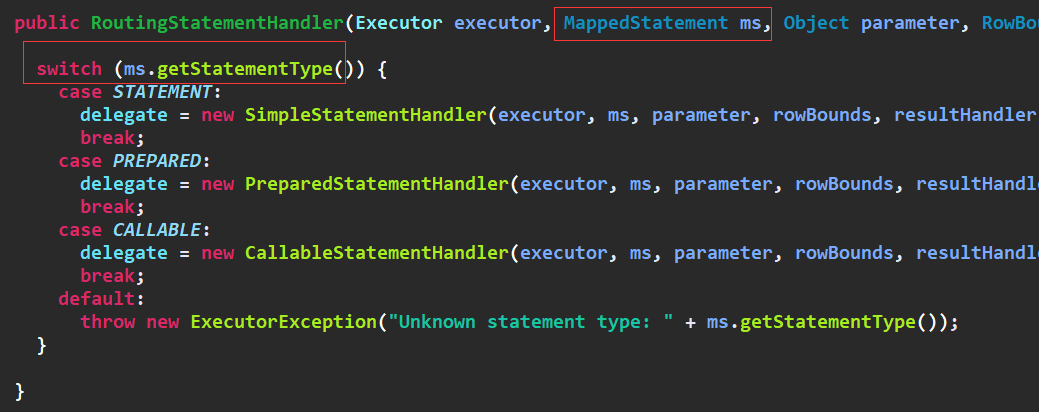
StatementHandler的作用就是为执行Statement(jdbc)的execute做初始化操作

在jdbc中有Statement，prepareStatement，CallableStatement当然在Mybatis中有其对应的Handler

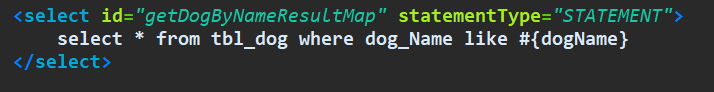


RoutingSatementHandler的作用是根据情况选择对应的StatementHandler





MappedStatement中保存的是Mapper.xml中对应的insert，delete，update标签对应的属性



可以在标准声明采用哪种Statement执行此SQL，默认为prepare，所以StatementHandler的创建是在执行方法时创建的(Executor执行之后)

Executor，StatementHandler，Statement之间的区别？

Executor是对操作的Statement进行管理，比如SimpleExecutor每次执行完就被摧毁，而ReuseExecutor的Statement首先会到缓存中查询，StatementHandler是对执行的操作进行一些处理，如建立连接，sql语句参数的完善等，完后将处理的结果直接交给Statement(JDBC的)进行处理

StatementHandler是Mybatis的接口，是一个父接口，而Statement是JDBC的接口

Executor管理的是Statement(JDBC)接口

Executor：Simple(每次创建新的Statement)，Batch(批处理Statement)，Reuse(可从缓存中获取Statement)

StatementHandler：Simple(Statement)，Callable(Callable)，prepared(prepared)[默认]

Executor对Statement进行管理，而StatementHandler为Statement做一些初始化的操作(比如数据库连接，SQL语句参数)，但是这些操作都是在BaseStatementHandler中做的，而具体的StatementHandler是执行具体Statement的方法

Executor的类型在Mybatis的配置文件中声明，而StatementHandler在insert标签中声明

Executor在获取SqlSession的时候就指定了

Statement和StatementHandler有联系又没有联系，没联系是指两个类之间不存在继承关系，有联系是指StatementHandler对应Statement

paramStatementHandler：对参数处理

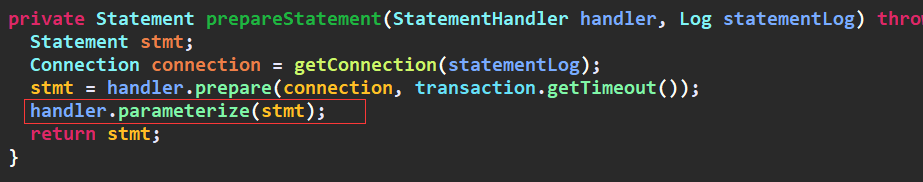
Executor：对执行的sql处理

StatementHandler：执行sql前的处理，对sql进行初始化操作

ResultSetHandler：返回结果处理

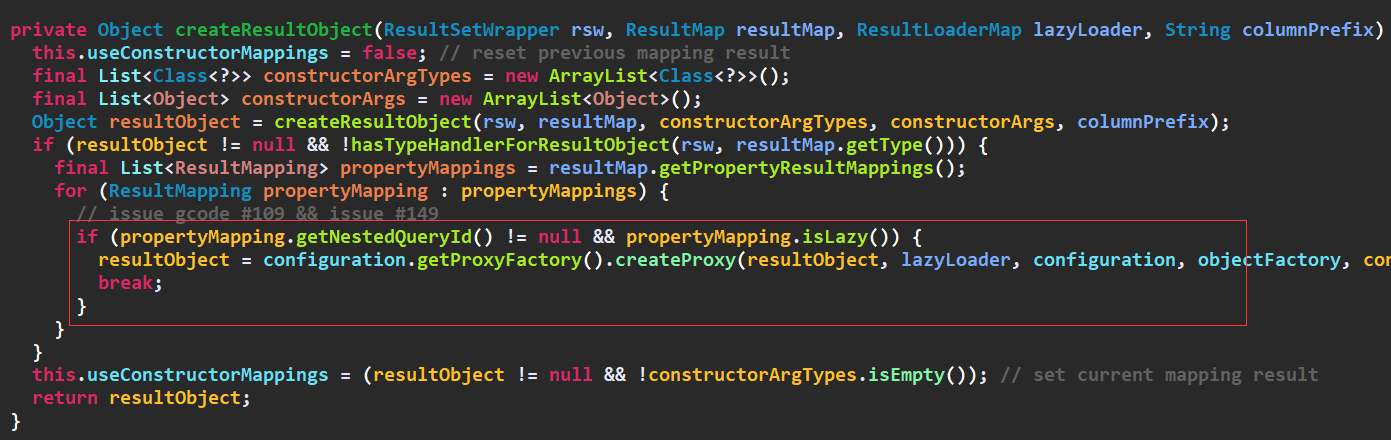
ResultSetHandler接口的实现类DefaultResultSetHandler主要对查询借结果根据ResultMap/ResultType封装为List集合

ParameterHandler接口的实现类DefaultParameterHandler在Statement为PrepareStatement时，在执行SQL操作之前，为SQL赋值参数



**延迟加载的原理：**

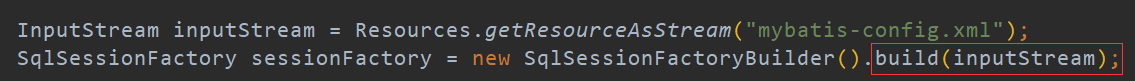
在DefaultResultSetHandler中有一个createResultObject方法



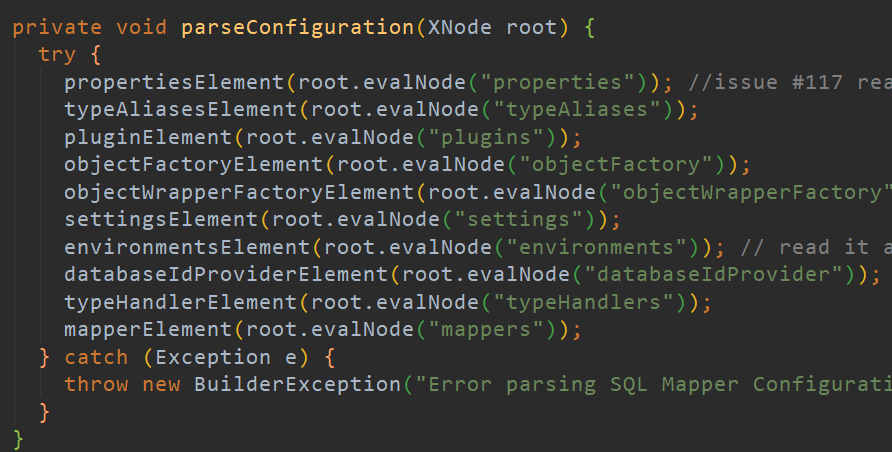
如果使用了延迟加载，会**创建一个代理对象**，代理对象中包含了返回的参数和目标的handler，这里使用的代理是

JavassistProxyFactory，如果没有启动延迟加载就返回原对象

加载过程：

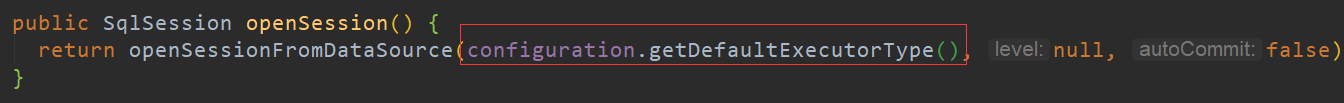


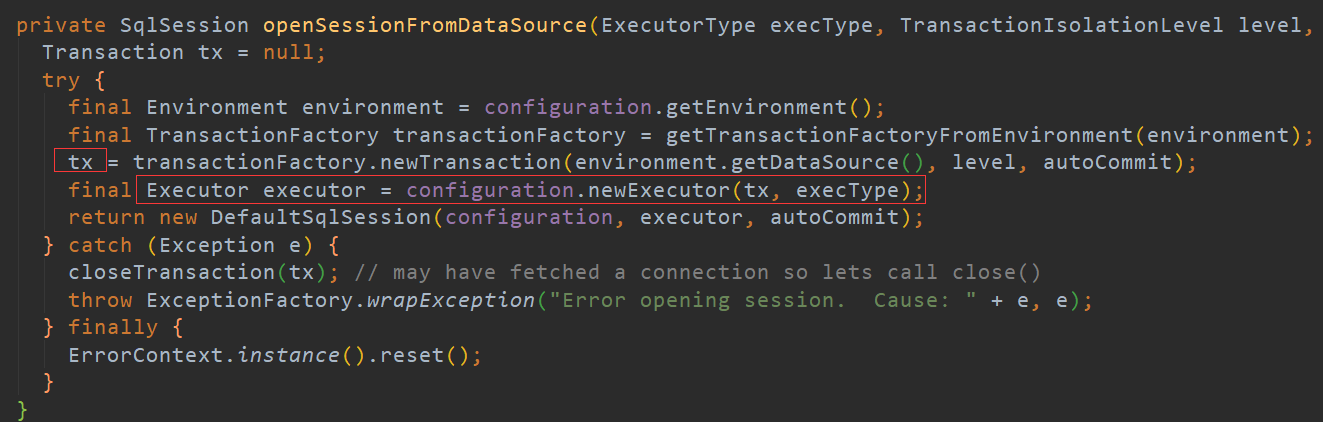
Build方法会调用parseConfiguration方法解析Node(标签)，将解析结果存放到Configuration中，然后返回一个DefaultSqlSessionFactory对象



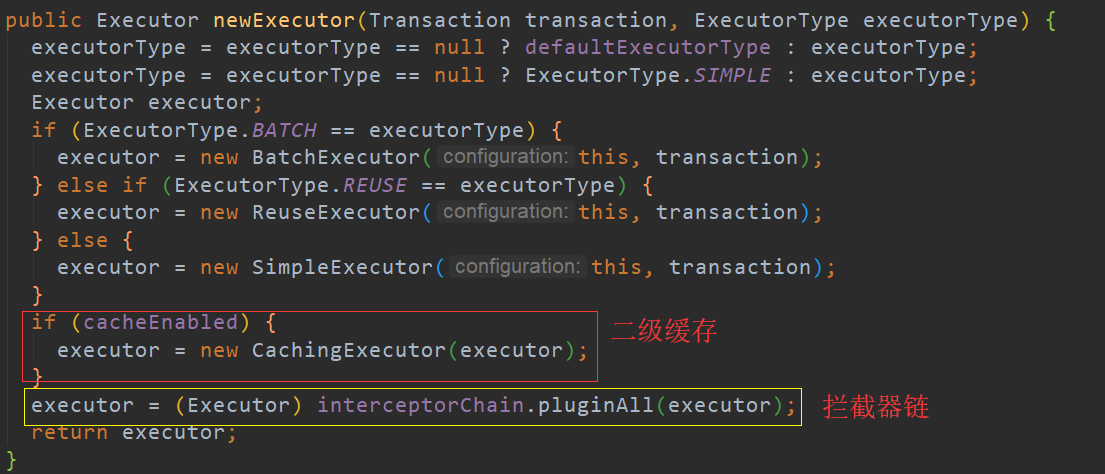
parseConfiguration方法依次解析标签

创建SqlSession：

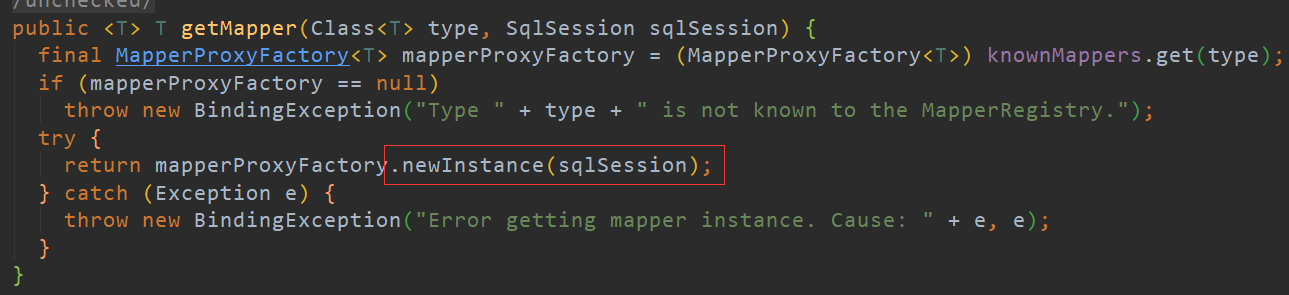




创建Executor后，创建DefaultSqlSession并返回

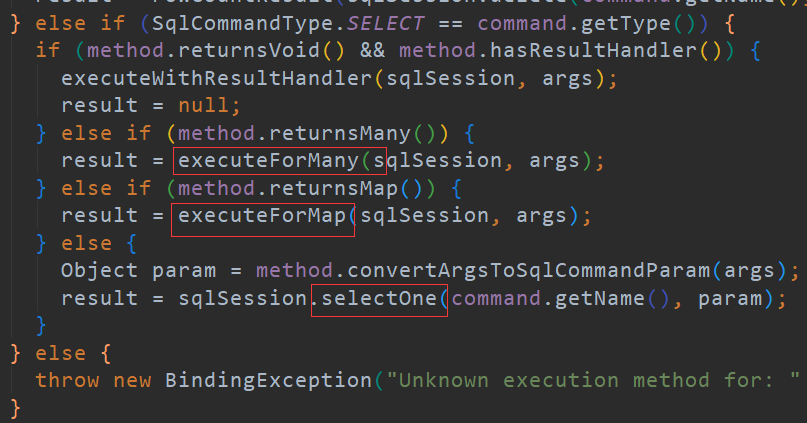


获取Mapper：获取Mapper的代理对象(MapperProxy)



执行代理对象的invoke方法



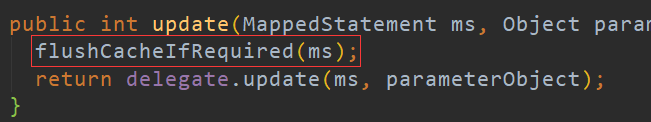


根据执行的操作，调用具体的方法，在select中，根据返回的值又调用不同类型的方法(convertArgsToSqlCommandParam对请求参数进行转换，转换为Map类型的集合)，最终方法的执行，还是回到了SqlSession上，DefaultSqlSession上具体实现了CRUD的方法，但是最终的执行有Executor操作

Update：insert和delete类似



这里的Executor为CachingExecutor(默认开启二级缓存)



执行update之前先刷新缓存，再执行update方法(delegate委派的Executor的方法)

Select：

SelectOne(DefaultSqlSession)：调用SelectList，然后获取第一个值

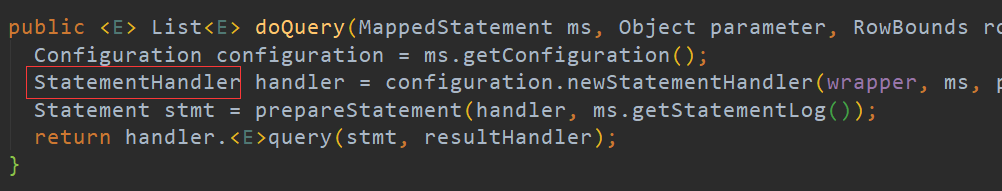
SelectMap：调用的是SelectList，然后将查询的结果封装为Map

SelectList：调用的Executor的query方法

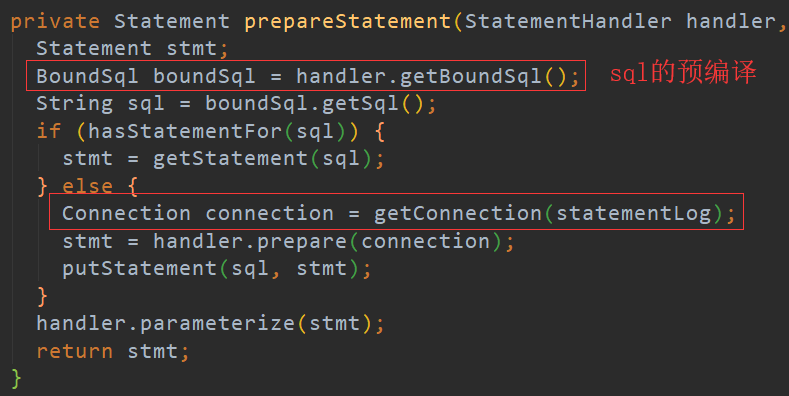
如果开启二级缓存执行的是CachingExecutor：先获取缓存，如果没有，再调用delegate的Executor(创建SqlSession时决定)的查询方法

如果未开始二级缓存，直接执行Executor的方法query方法(会查一级缓存)

doQuery：







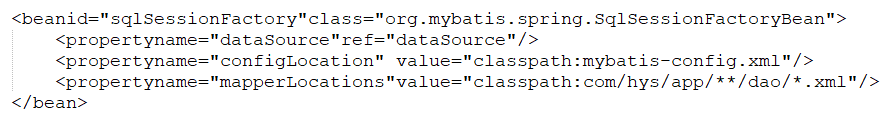
PrepareStatement中有一个BoundSql就是SQL的预编译结果



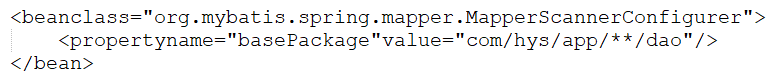
ResultSetHandler对结果进行处理并返回

Spring和Mybatis整合：

SqlSessionFactoryBean的创建：



SqlSessionFactoryBean会对mybatis-config.xml和mapperLocations的配置文件进行解析，将结果存放到Configuration中



配置MapperScannerConfigurer，为Mapper创建代理对象

当Mapper.class和Mapper.xml在同一目录时，可以不用写mapperLocations，在为Mapper创建代理对象时，会加载目录下的xml文件(mapper.xml文件的加载在MapperScannerConfigurer中，如果生命mapperLocations也会加载吧)

每次再调用Service方法中的mapper时，都会创建一个新的SqlSession(在SqlSessionTemplate执行方法时，会调用一个sqlSessionProxy的方法，而这个sqlSessionProxy会重新创建一个SqlSession作为Executor的参数，然后在Executor中调用Statement执行查询操作)，所以，spring和Mybatis整合后，一级缓存失效

这里的SqlSessionTemplate是单例的，创建Mapper的代理对象时会使用SqlSession，就是这个SqlSessionTemplate

查询过程：代理对象----SqlSession----Executor----Statement

根据返回类型，个数，调用selectOne，selectMany，selectList，但最终都调用的是selectList

创建过程：SqlSession(同时创建Executor)----代理对象----Statement

Mybatis和Hibernate的区别：

Hibernate对SQL进行封装，重写SQL麻烦，Mybatis可以对SQL进行优化

Hibernate具有数据库无关性，跨数据很好，Mybatis比Hibernate容易入门

Hibernate是完整的ORM框架，而Mybatis是半自动的