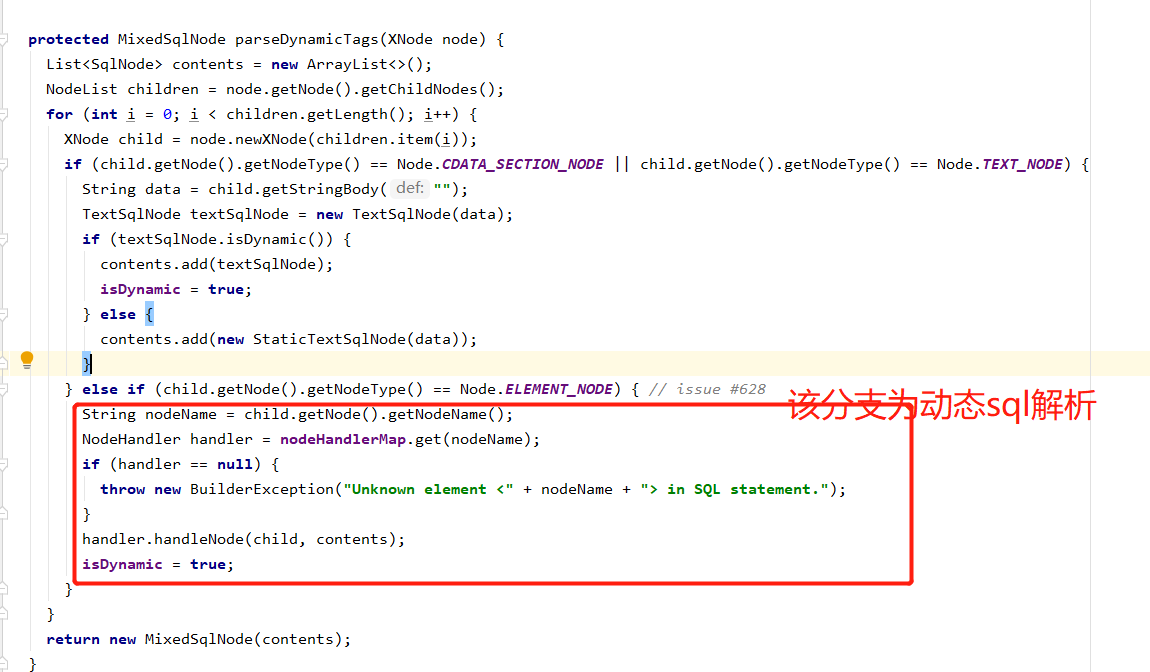
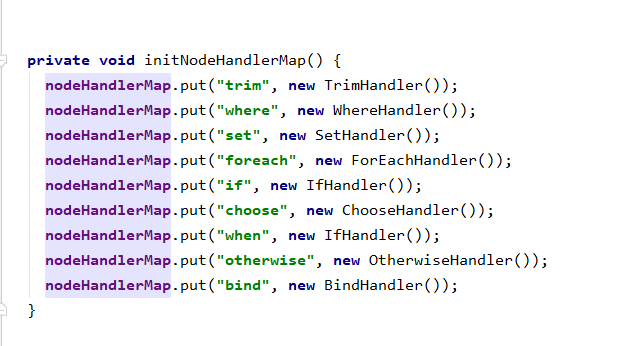
# Mybatis动态sql是做什么的？都有哪些动态sql？

1. 定义sql类型，主要标签有< select > <insert><delete><update>
2. 配置java 实体和数据库表字段的映射关系，主要有<resultMap><result>
3. 动态拼接sql,主要有<if><foreach><choose>
4. 格式化sql语句，主要有<where><set><trim>
5. 配置关联关系，如一对多，一对一，多对多<collection><association>

# 简述一下动态sql的执行原理？



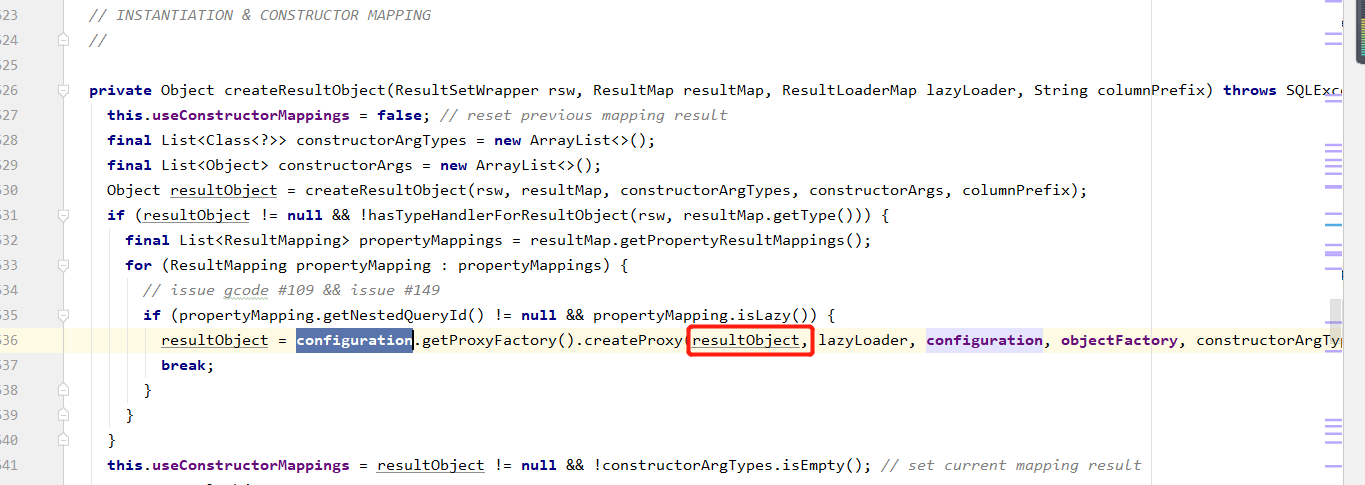
下图中的map对象存放了所有动态标签的解析器



# 2、Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

支持

实现原理：在DefaultResultSetHanlder的createResultObject方法中，映射一条记录到对象中的时候，会判断对象是否开启了延迟加载，如果开启的话，会为对象创建一个代理对象，当操作对象的相关方法时，会触发代理对象去执行查询子对象的sql.

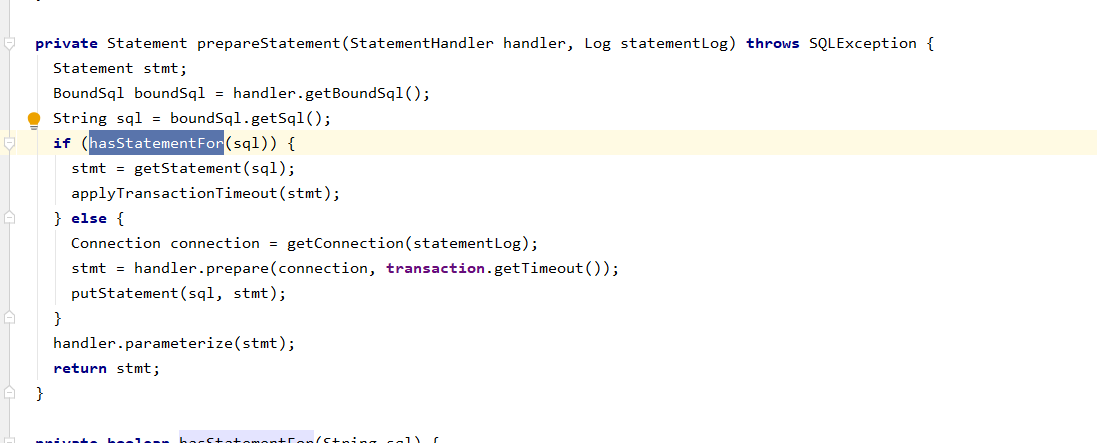


# 3、Mybatis都有哪些Executor执行器？它们之间的区别是什么？

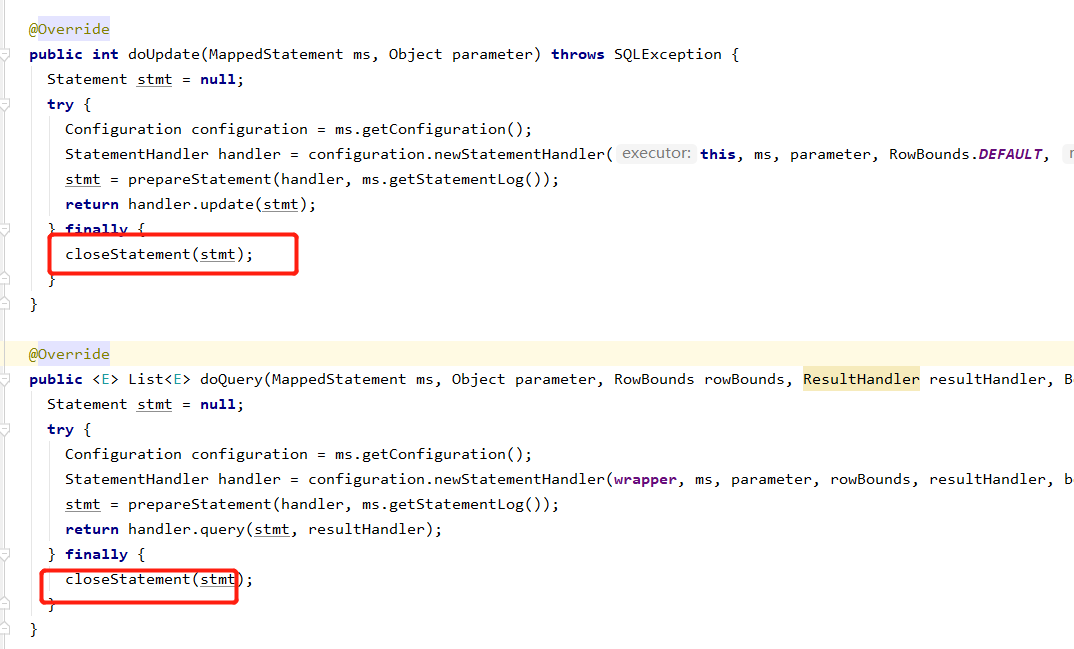
#### 主要有SimpleExecutor、ReuseExecutor、BatchExecutor

区别主要体现在statement的管理方式上

下图为ReuseExecutor，可以看到每次query或者update都会到map中查询是否有改sql对应的statement,如果有拿来用，如果没有才创建并放到map中缓冲起来。



SimpleExecutor是每次都会创建，使用完之后立即关闭如下图所示



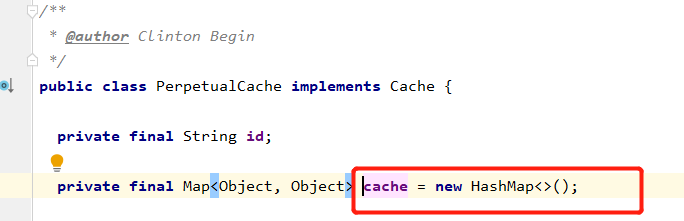
BatchExecutor和SimpleExecutor类似，主要不同体现在update方法上，batchExcutor可以支持批量执行多条sql。

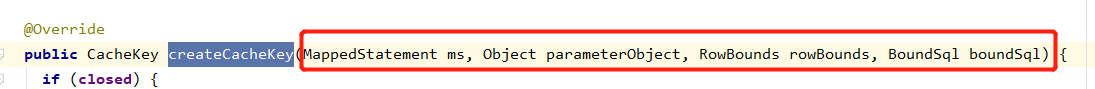


# 4简述下Mybatis的一级、二级缓存（分别从存储结构、范围、失效场景。三个方面来作答）？

**一级缓冲：**

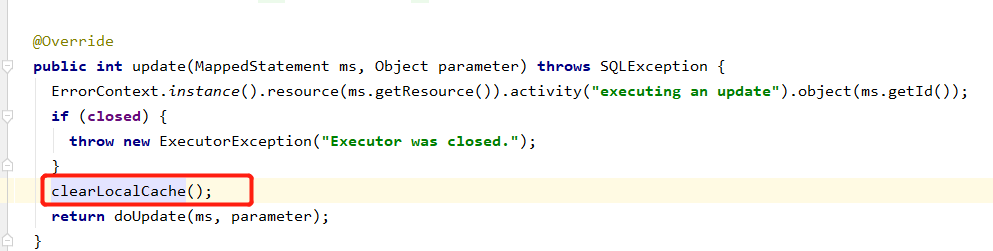
存储结构：Map 如下图所示，key由MappedStatement,参数,分页信息，sql信息组成





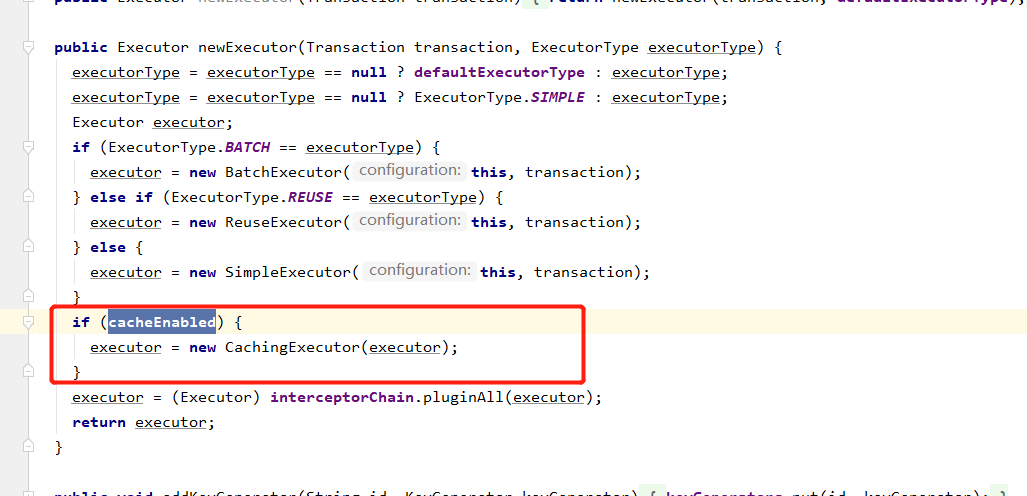
范围：Sqlsession

失效场景:当执行update方法时，会清空缓冲，如下图



二级缓冲：

当我们配置了cacheEnable属性后，mybatis会使用cacheExcutor执行我们的语句如下图，cacheExcutor使用装饰器模式，对其他excutor做了装饰。



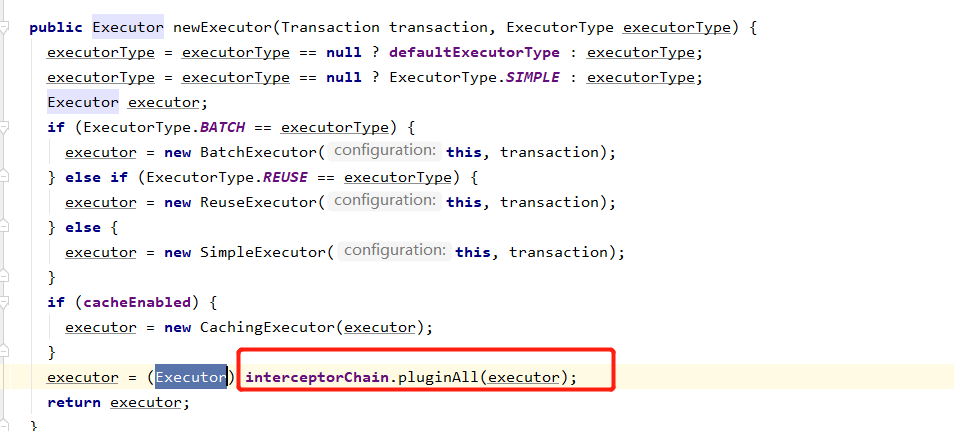
存储机构：可以支持外部存储（如redis,[*Ehcache*](https://link.jianshu.com/?t=http://www.ehcache.org/)），实现cache接口即可

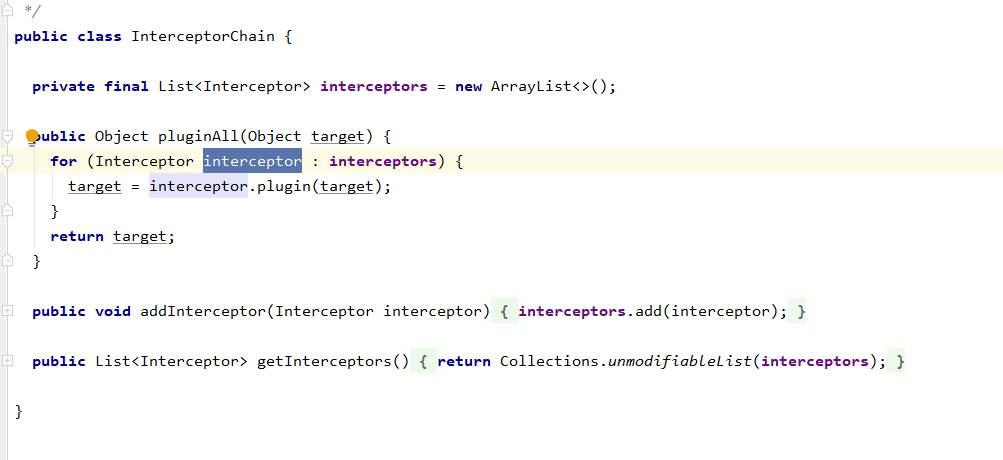
范围：namespace范围

失效：当在指定namespace下调用doupdate方法时失效。

# 5简述Mybatis的插件运行原理，以及如何编写一个插件？

创建Executor, ParameterHandler,ResultSetHandler,StatementHandler时，会调用inteceptorChain.pluginAll方法，如下图，inteceptorChain存放了所有实现了Interceptor接口的对象。并挨个调用拦截器对象的plugin方法，plugin方法中又调用了Plugin.*wrap*(target, **this**)方法，该wrap方法将会创建target的动态代理对象，并且Plugin实现了InvocationHandler接口。







实现自定义拦截器

@Intercepts({@Signature( type= Executor.class, method = "update", args ={MappedStatement.class,Object.class})})

public class ExamplePlugin implements Interceptor {

public Object intercept(Invocation invocation) throws Throwable {

System.out.println(“执行之前干你想干的”)

return invocation.proceed();

System.out.println(“执行之后干你想干的”)

}

public Object plugin(Object target) {

return Plugin.wrap(target, this);

}

public void setProperties(Properties properties) {

}

}