1，Mybatis动态sql是做什么的？都有一些动态sql？简述一下动态sql的执行原理？

MyBatis动态sql可以在xml文件内，以标签的形式编写动态sql，具体有set,foreach,if,where,choose,when,otherwise,bind,trim。执行原理：是根据表达式的值完成逻辑判断并动态拼接sql。

2，Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

支持延迟加载。实现原理是：使用CGLIB创建目标的代理对象，当调用目标方法时，进入拦截器方法。比如调用a.getB().getName(),拦截器invoke()方法发现a.getB()是null值，那么久会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用a.setB(b),于是a的对象b属性就有值，接着完成a.getB().getName()方法的调用。

3，Mybatis有什么执行器执行器？它们之间的区别是什么？

Executor的实现对象有3个：SimpleExecutor,ReuseExecutor,BatchExecutor。
1.SimpleExecutor:每执行一次update或者select,就会开启一个Statement对象，用完立刻关闭。
2.ReuseExecutor:执行update或者select，以sql作为key查找Statement对象，存在就使用，不存在就创建。用完后，不关闭Statement对象，而是放置于Map内，供下一次使用。
3.BatchExecutor：执行update(没有select)，将所有sql都添加到批处理中，等待统一执行，它缓存了多个Statement对象，每个Statement对象都是adddBatch()完毕后，执行executeBatch()批处理，

4，简述下Mybatis的一级，二级缓存（分别从存储结构，范围，重复场景。三个方面来作答）？

一级缓存：MyBatis的一级缓存是指SqlSession级别的，作用域是SqlSession，MyBatis默认开启一级缓存，在同一个SqlSession中，相同的Sql查询的时候，第一次查询的时候，就会从缓存中去，如果发现没有数据，那么就从数据库中查询出来，并且缓存到HashMap中，如果下次还是相同的查询，就直接缓存中查询，就不在去查询数据库，对应的就不在去执行SQL语句，当查询到的数据进行增删改操作的时候，缓存就会失效，在spirng容器管理中每次查询都是创建一个新的SqlSession，所以在分布式环境中不会出现数据不一致的情况

二级缓存：二级缓存是mapper级别的缓存，多个sqlSession去操作同一个mapper的sql语句，多个sqlSession可以共用二级缓存，二级缓存是跨SqlSession。第一次调用mapper下的sql是查询信息，查询到的信息会存放到mapper对应的二级缓存区域，第二次调用nameSpace下的mapper映射文件中，相同的SQL去拆线呢，会去对应的二级缓存中去结果，使用值需要开启cache标签，在select标签上添加useCache属性为true，在更信和删除时候需要手动卡其flushCache刷新缓存。

5，简述Mybatis的插件运行原理，以及如何编写一个插件？

```
package com.lo.plugin;

import org.apache.ibatis.executor.statement.StatementHandler;
import org.apache.ibatis.plugin.*;

import java.lang.Object;
import java.sql.Connection;
import java.util.Properties;

/**
 * @Author chen_jie
 * @Date 2020/7/19 17:34
 * @Version 1.0
 * @Description 预处理mybatis的插件或者说是拦截器
 * @Param
 * @return
 **/

@Intercepts({
        @Signature(type = StatementHandler.class,
        method = "prepare",
        args = {Connection.class,Integer.class})
})
public class MyPlugin implements Interceptor {
```

```
@Override
    public Object intercept(Invocation invocation) throws Throwable {
        System.out.println("对方法进行了增强。。。。。。。。。。");
        return invocation.proceed();
    }

    @Override
    public Object plugin(Object target) {
       return Plugin.wrap(target,this);
    }

    @Override
    public void setProperties(Properties properties) {
        System.out.println("获取到的配置文件的参数是："+properties);
    }
}
```