mybatis 加强深入

接口式编程

```
messageList = sqlSession.selectList("Message.queryMessageList", message);
```

在这句中 存在着各种风险，其实各种需要规避的地方，比如namespace与id号的问题，指定类型的问题

规范了形式，与spring结合会很有用
这是在MessageDao层的

```
IMessage imessage=sqlSession.getMapper(IMessage.class);
        messageList=imessage.queryMessageList(message);
```

这是我们在接口定义的规范的

```
import java.util.List;

import com.imooc.bean.Message;

/**
 * 与messag相对应的接口
 * @author Administrator
 *
 */
public interface IMessage {

 public List<Message> queryMessageList(Message message);
}
```

这是我们在映射配置xml文件中修改的

```
<mapper namespace="com.imooc.dao.IMessage">
```

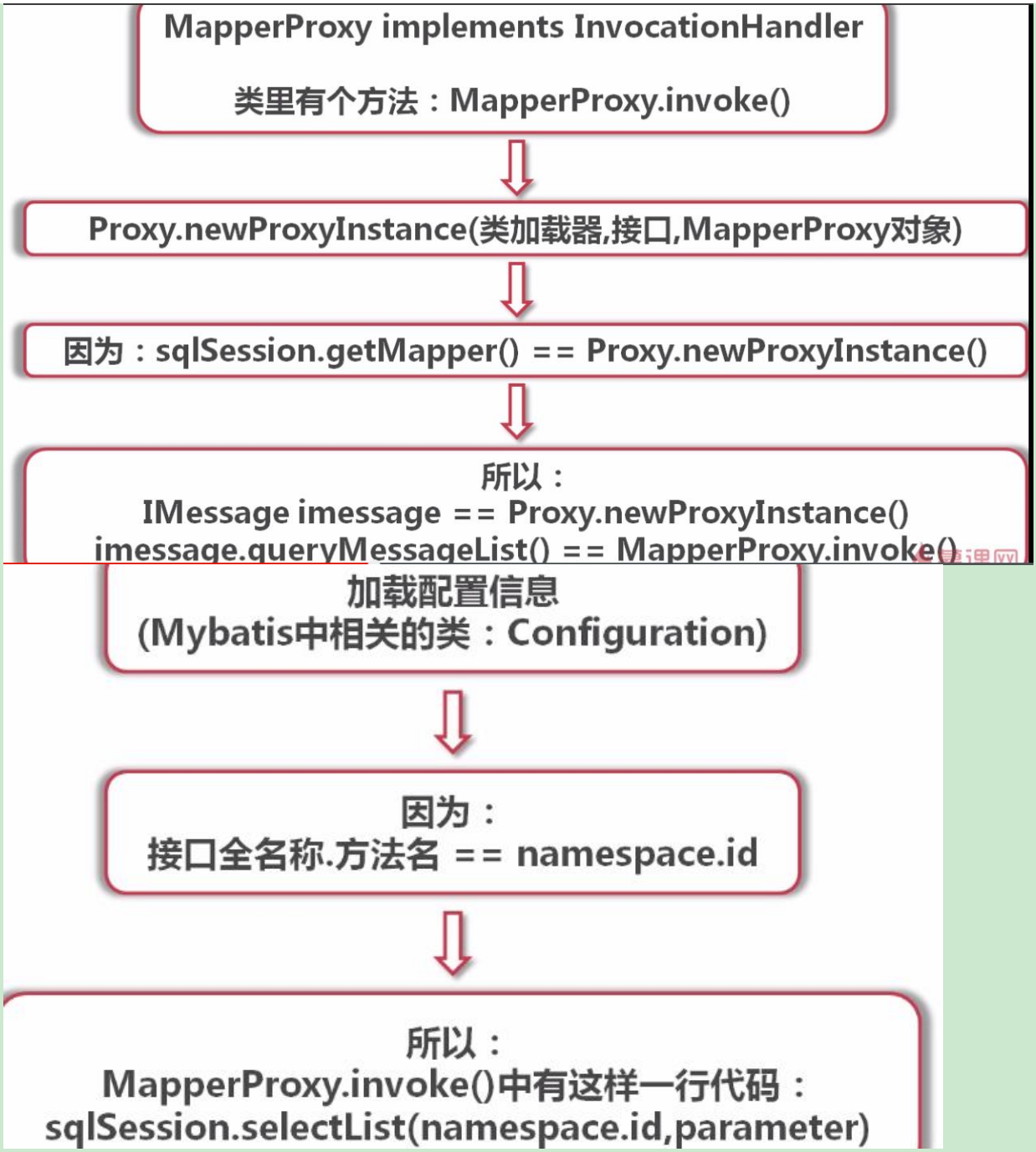
接口式编程mybatis

配置文件的namespace对应接口的包的名称+接口名称
配置文件中对应的sql的id对应接口的方法名
调用：通过sqlSession.getMapper(接口.class)调用

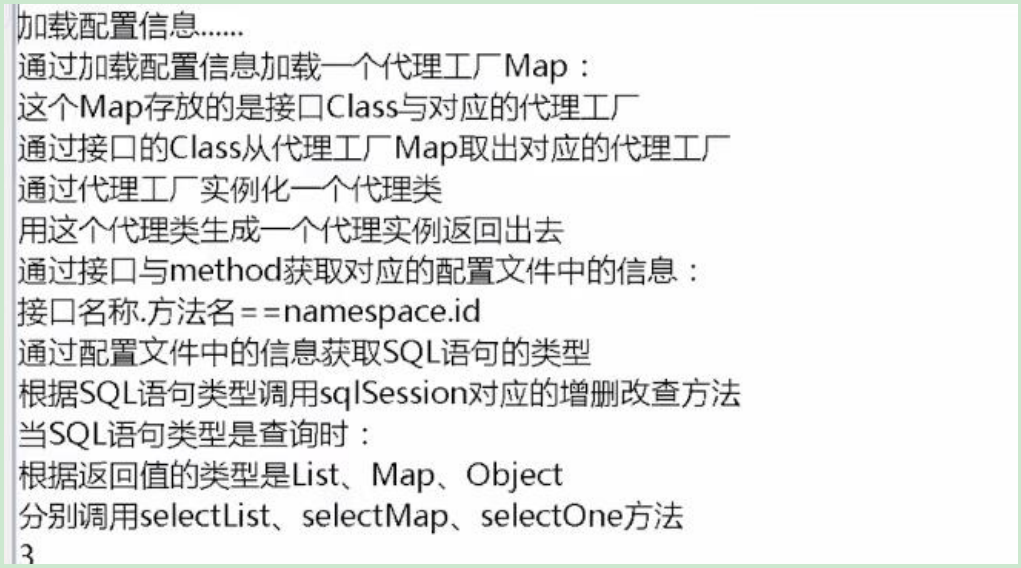
/2、遇到spring时{1、总配置文件中的数据源配置托管给spring管理2、db层（getsqlsession）会消失3、组织对象代码移交给service层（即传入的参数）4、sql执行代码由spring实现5、dao层就剩接口文件（小三上位）与配置文件}

上面这个接口式编程为什么能够实现呢？是因为动态代理，

```
/*动态代理,接口没有实现类.Mybatis为接口提供实现类,即用Proxy.newProxyInstance()创建代理实例,返回类型为Object,利用泛型强制转换
*/
IMessage imessage = sqlSession.getMapper(IMessage.class);
/*代理实例调用接口方法时,并不会执行,而是触发 MapperProxy.invoke(),其中包含sqlSession.selectList(namespace.id,parameter)*/
/*至于为什么会包含,因为接口方法与(加载Mybatis的)配置信息对应得上,即 接口名.方法=namespace.id*/
messageList = imessage.queryMessageList(message);
```



imessage.queryMessageList(parameter) ==sqlSession.selectList(namespace.id,parameter)
关于接口式编程，mybatis内部实现的流程



与Spring结合的时候，Dao层会消失的，我们应该把具体参数放在service层

简单的说DAO层是跟数据库打交道的，service层是处理一些业务流程的
@SQL语句里的limit使用方法

SELECT * FROM table LIMIT [offset,] rows | rows OFFSET offset
在我们使用查询语句的时候，经常要返回前几条或者中间某几行数据，这个时候怎么办呢？不用担心，mysql已经为我们提供了上面这样一个功能。

LIMIT 子句可以被用于强制 SELECT 语句返回指定的记录数。LIMIT 接受一个或两个数字参数。参数必须是一个整数常量。如果给定两个参数，第一个参数指定第一个返回记录行的偏移量，第二个参数指定返回记录行的最大数目。初始记录行的偏移量是 0(而不是 1)： 为了与 PostgreSQL 兼容，MySQL 也支持句法： LIMIT # OFFSET #。

mysql> SELECT * FROM table LIMIT 5,10; //检索记录行6-15

//为了检索从某一个偏移量到记录集的结束所有的记录行，可以指定第二个参数为 -1：
mysql> SELECT * FROM table LIMIT 95,-1; // 检索记录行 96-last.

//如果只给定一个参数，它表示返回最大的记录行数目：
mysql> SELECT * FROM table LIMIT 5; //检索前 5 个记录行
//换句话说，LIMIT n 等价于 LIMIT 0,n。

案例：
SELECT * FROM table LIMIT 0,5;//0、1、2、3、4
SELECT * FROM table LIMIT 5,5;//5、6、7、8、9
SELECT * FROM table LIMIT 10,5;//10、11、12、13、14

分页功能实现的时候需要每次都要实现一次，这个分页的代码，大部分时候都需要分页功能，但是我们这个时候就是还需要再实现一遍分页功能的话，这样式重复码代码是不好的，我们来看哪些代码是需要分页的代码

拦截器：
1 要做什么事
2 拦截下来做什么事
3 事情完成后交回主权
注意：
1 要拦截器什么样的对象
2 拦截对象的什么样的行为
3 什么时候拦截

拦截器就是在mybatis不改变的源码的情况下，实现改变mybatis的功能，我们需要在这里进行改变，首先我们需要mybatis执行sql语句的时候是什么样子的（源码在此的过程 ）

拦截器是什么样的呢
需要继承一个接口 implements Interceptor
@Intercepts({@Signature(type=StatementHandler.class,method="prepare",args={Connection.class})})**另外需要在这里说明，我们拦截的内容**

实现这个接口，需要实现三个方法 其中plugin(Object target)方法，target是被拦住的对象，如果不需要就放掉，如果需要，就需要返回代理类，
return Plugin.wrap(target, this); this表示这个例子的实例
就是拦截对象的地方，我们通过**this这个传递我们的拦截器的注释信息**，然后**通过注释信息**，**就能获取到**我们需要拦截的具体内容，如果通过这个拦截器的对象与我们拦截的不匹配的话，就放行，**如果是**我们**目标的类**，就会经过**被拦截**，就是经过Plugin内部的实现方法，就是一个动态代理的方法
intercept方法中就是我们拦截下来对象的具体实现方法， **invocation**这个对象中**就是我们拦截的对象**，就是我们上面设置了注释的，是符合注释的具体参数的，我们需要的对象

如果我们的**注释具体参数写错了** ，在plugin方法中，**返回的是原对象本身，继续它原本的行为**，我们拦截不下来，如果使**我们需要的对象**，就会**生成一个代理类，然后再执行intercept方法**
需要**通过具体的判断**来判断我们**拦截的住的对象是不是想要实现分页的功能**，这个需要判断，列如通过名字的判断：
queryMessageList**ByPage在后面的字母中，有需要我们设置分页的说明，我们就依靠这个来进行判断** ，在判断的时候，。只需要以ByPage结尾的都是我们需要的对象，在真实的项目中有所变化，只需要项目组的统一，明白就好
我们需要获得MapperStatement的里面的id值，来进行匹配，但是这个id值，**我们需要匹配的id值**，就是**select id**的这个值是**受保护的protectd**，**无法拿到** ，我们只能需要通过反射来加载它，**而mybatis帮我们封装好了一个方法就是 MetaObject这个方法** ，用这个方法
MetaObject metaObject = MetaObject.forObject(statementHandler, SystemMetaObject.DEFAULT_OBJECT_FACTORY, SystemMetaObject.DEFAULT_OBJECT_WRAPPER_FACTORY);
这个时候我们获取到的metaobject，就是可以认为是statementHandler,只不过他提供了一个方法，可以很方便的获得属性， 这个方法就是getvalue

在内部呢，其实我们第一次访问的是RoutingStatementHandler，而后RoutingStatementHandler这个类内部有个对象的属性delegate找 这个delegate是statementHandler是实现的BaseStatementHandler的接口，然后再去访问到BaseStatementHandler；所以我们写getvalue的时候，必须是这样写才能访问到 MappedStatement mappedStatement = (MappedStatement)metaObject.getValue("delegate.mappedStatement");

在这个代码中，非常容易出现**强转（强制类型转换）**的情况，我们必须对这个类型情况了解的比较清楚才行，不然容易出错

然后通过我们拿到的mappedStatement

顺利的获得了id
String id = mappedStatement.getId();

java.lang.String.matches() 方法告诉这个字符串是否匹配给定的正则表达式
if(id.matches(".+ByPage$"))**判断是否已ByPage结尾的**

分页拦截器写起来还是非常的复杂，所以说要抓住核心，因为复杂，所以写的情况大不相同，但是目的都是相同的，根据目的来看代码
sql语句在prepareStatement中的 一个BoundSql属性里面
这个BoundSql是可以直接获取的，BoundSql boundSql = statementHandler.getBoundSql();所以我们直接获取了
下面我们需要些改造后的SQL语句，带分页的
在前面的代码中，**我们已经将page分页，传到了Dap层的参数中，只不过我们拿到后没有使用而已**
而这个BoundSql不仅可以拿到sql语句，**还可以拿到别人传给sql的参数，我们从来这里来取出page**
Map<String, Object>parmeter=(Map<String, Object>) boundSql.getParameterObject();
Page page=(Page)parmeter.get("page");
现在我们已经获得了page
这是我们改变好的sql语句
改造后带分页sql查询的语句
String pagesql=sql+"limit"+page.getDbIndex()+","+page.getDbNumber();
那如何将这条sql语句换回去呢
mybatis提供了一种方法，修改我们原本不能修改的值
metaObject.setValue("delegate.boundSql", pagesql);
这个代码的值就是修改原来的sql，为我们写的sql（偷天换日）
这个就是修改sql值，**将原来默认的sql，改为我们默认的sql**
然后我们需要将下面的步骤，交还给mybatis，不然的话，一直是被阻止的情况
return invocation.proceed();
我们只需要在**返回值这里写上这么一句代码**就可以了
在这个代码这个内部，**其实是一个反射，用反射来调用被你拦截的住的方法的**
我们**需要根据当前的SQL语句查询总条数的SQL**，否则的话，**Page参数是残缺的**，传回到页面上**这个总页数是没有的，因为没有总条数的话**
是算不出总页数的
但是这条sql语句是String countSql = "select count(*) from (" + sql + ")a";
我们拿到当前的sql语句当一次子查询，
当前我们拦截到的statment，是用来拦截实现分页的，不能用于其他查询，所以这个查询的话，我们**需要自己配置JDBC，查询总条数**
我们当前拦截的**这个方法的参数 Connection**，可以直接用的，**在innvocation中，当然是有参数**的
Connection connection = (Connection)invocation.getArgs()[0];
由于我们知道在这个拦截中**只有一个参数，所以第一个**就可以了
然后再做骚转
PreparedStatement countStatement = connection.prepareStatement(countSql);
然后就到了这一步，
在mybatis中，这条sql，当有检索条件的时候，**sql语句是有参数的**，而statment**没有为这些参数赋值之前是不能执行**的，**所有#{}在mybatis中被解析为？号**了，这些东西我们是没有的，但是在mybatis中是有记录这些#{}对应的关系位置的
这里插一句：我们不要怕源码，都是用java代码实现的，对任何开源框架的解析只要找对方向，有足够的基础，都是可以被解析的
在BaseStatementHandler中是有各种具体的信息，其中就有我们需要的PrepareStatementHandler
在这个PrepareStatementHandler中就有我们需要的东西，这个类中实现了一个setParamters方法，这个方法就是它掌握的信息为我们的prepare来赋值
在原来的语句中，我们虽然对原本的语句进行了包装，但是没有改变了结构，其中包括？的个数等等，**这个是可以为prepareStatment来赋值的**
ParameterHandler parameterHandler = (ParameterHandler)metaObject.getValue("delegate.parameterHandler");
这个是来获取parameterHandler
这个类提供了一个方法是来获取的，：
parameterHandler.setParameters(countStatement);
ResultSet rs = countStatement.executeQuery();
加上这个语句，这个方法就是可以执行的了来获取结果ResultSet

```
    if(rs.next()) {
            page1.setTotalNumber(rs.getInt(1));
        }
```

这个语句是放到获得到page对象之后的，这点要注意
终于来获取到了值，直接保存到page里面
到了这一步，然后到了最后一步，也是非常重要的一步还没有完成，我们需要注册，来让mybatis来相信我们
在**configuration.xml中加上，如果我们有一个拦截器，就需要在这里，加上这么一个标签，这是我们必须实现的地方，不然的话，是实现不了的，这个应该在开头第一步就应该想到了，写到了这里，但是千万不能忘记**

```
 <plugins>
    <plugin interceptor="com.imooc.interceptor.PageInterceptor">
       <property name="test" value="abc"/>
    </plugin>
 </plugins>
```

在拦截器，我们还没有方法没有实现的一个方法就是setProperties
这个方法就是为了获取我们在configuration.xml
```
    <property name="test" value="abc"/>
    </plugin>
```
中设置的参数，
在setProperties这个方法来通过以下方法获取到我们在configuration.xml定义的property变量
```
properties.getProperty("test");
```
我们在类中声明一个变量
```
private String test;
```
然后在底下进行赋值
```
this.test = properties.getProperty("test");
```

我们这个拦截器的执行顺序是，先通过setProperties这个方法拿到我们在configuration.xml中拿到属性值，然后我们执行plugin获取到拦截的对象
intercept来处理了我们拦截的对象的要改变的方法
```
Map<String, Object>parmeter=(Map<String, Object>) boundSql.getParameterObject();
```
这里会出现一个警告，我们只需要改变为这样的就好
```
Map<?,?> parameter = (Map<?,?>)boundSql.getParameterObject();
```

摘抄于别人笔记
mybatis的拦截器实现分页（动态代理）
拦截sql语句来实现分页
1.拦截什么样的对象（以page作为参数传入；page对象）
2.拦截对象什么行为
3.什么时候拦截 （在prepareStatement的时候拦截）
（源码）

1.RoutingStatementHandler
2.通过RoutingStatementHandler对象的属性delegate找到statement实现类BaseStatementHandler
3.通过BaseStatementHandler类的反射得到对象的MappedStatement对象
4.通过MappedStatement的属性getID得到配置文件sql语句的ID
5.通过BaseStatementHandler属性的到原始sql语句
6.拼接分页sql（
1.需要查询总数的sql
2.通过拦截Connection对象得到PrepareStatement对象
3.得到对应的参数
4.把参数设到prepareStatement对象里的？（该？号在配置文件以#{}形式存在，mybatis会把它转为？号）
5.执行噶sql语句
6.得到总数
）
7.把属性值为新的sql

mybatis的拦截器实现分页（动态代理）
拦截sql语句来实现分页
1.拦截什么样的对象（1.RoutingStatementHandler;2.以page作为参数传入；page对象）
2.拦截对象什么行为
3.什么时候拦截 （在prepareStatement的时候拦截）
（源码）

1.RoutingStatementHandler
2.通过RoutingStatementHandler对象的属性delegate找到statement实现类BaseStatementHandler
3.通过BaseStatementHandler类的反射得到里面的属性对象（proptect)的MappedStatement对象
4.通过MappedStatement的属性getID得到配置文件sql语句的ID
5.通过BaseStatementHandler属性的到原始sql语句
6.拼接分页sql（
1.需要查询总数的sql
2.通过拦截Connection对象得到PrepareStatement对象
3.得到对应的参数
4.把参数设到prepareStatement对象里的？（该？号在配置文件以#{}形式存在，mybatis会把它转为？号）
5.执行该sql语句
6.得到总数
）
7.把属性值为新的sql

拦截器总结
首先我们进行总结，首先，我们要实现一个分页拦截器，继承Interceptor这个接口，分页拦截器的类，再说一次，最好创建好了就注册，首先我们要确认，是否配置过property标签，如果有就获取到（setProperties方法）　，然后我们首先要确定好，我们要拦截的对象
```
@Intercepts({@Signature(type=StatementHandler.class,method="prepare",args={Connection.class})}
```
，通过注解来分析，然后再通过plugin方法来获取，如果使的我们就获取，如果不是就放行。到这一步然后我们就需要实现具体步骤了，intercept方法了，到这一步的时候，我们还需要过滤一下，因为这不一定是我们需要的，是bypage结尾的，然后我们处理具体步骤，将原来的Sql语句换成我们需要的pageSql
最后有重要的一步，就是将其返回出去，
```
return invocation.proceed();
```

批量插入数据
在JDBC中，我们用的是在JDBC中取到数据，然后在while中组装数据，这样是一条一条执行的，这样的浪费时间

在JDBC中，我们只需要，实现

```
for(CommandContent content : contentList) {
            statement.setString(1, content.getContent());
            statement.setString(2, content.getCommandId());
            statement.executeUpdate();
        }
```

这样的代码，我们将它改变一下，不立即执行

而是

```
for(CommandContent content : contentList) {
            statement.setString(1, content.getContent());
            statement.setString(2, content.getCommandId());
            statement.executeUpdate();
        }
```

这样的代码，我们不立即执行，我们需要加一下，然后在循环外执行

```
for(CommandContent content : contentList) {
            statement.setString(1, content.getContent());
            statement.setString(2, content.getCommandId());
           statement.addBatch();
        }
  statement.executeUpdate();
```

不同的数据库在批量新增的sql语句是不同的，这次是用mysql的方言写的，这是在xml中具体配置的

这里的，号只能在俩次圆括号的中间出现，这个时候我们需要一个separator=","来实现

```
 <insert id="insertBatch" parameterType="java.util.List">
    insert into COMMAND_CONTENT(CONTENT,COMMAND_ID) values
    <foreach collection="list" item="item" separator=",">
      (#{item.content},#{item.commandId})
    </foreach>
 </insert>
```

批量插入，jdbc本身已经提供了api。而mybatis是要自己在配置文件配置sql语句，所以根据mysql数据库的insert语句拼接出相应的sql

insert into 表名 values（数据1，数据2...），（数据1，数据2...）,...即可，所以传入的参数是list给<insert>标签即可，最后forEach标签循环出来数据拼接sql语句

数组参数引发的错误

1 mybatis是如何加载一个数组的Class

2 如何填写属性值表示数组类型

我们在Dao有一个优化的问题，就是

1 加载时机的不对问题，

2 就是反复加载

比如在Dao层的DbAcess的反复加载问题

每一个方法都会调用一次DbAcess类，

关于配置文件的解析过程，是采用Dom的方法进行解析XML的

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);

在build方法中，我们**进行解析**，然后又一个简单的调用，再到下一层，其中，**就有我们需要的XMLConfigBuilder类**，将**当前我们需要的参数**传入对象中，**我们需要返回调用这个对象内部的parse方法**，在这个对象内部，就是XMLConfigBuilder内部，这个parse方法，具体在XMLConfigBuilder类中，这个**parse在内部只能被调用一次**，这**里面还有一个关于parse的类这是XPathParser，这个XPathParser类中就一个document对象**，这个就是解析xml的对象，在总配置文件中**，我们配置的映射文件加载的时候，是能够加载到的，只要**我们**进行引入**的，其中映射文件的内容**parameterType属性也是可以被解析**的，**这个时候我们需要采用xml解析了，mybatis采用解析的是用DOM**，在这个**XPathParser类中，**有一个构造方法，式传进来的参数，其中就有一个Reader ，这个Reader，就是**总的配置文件的流** ，这个document就是一种存储形式，很方便的形式，

这里还有一个关键的概念，就是XPath

Xpth就是路径语言 Xpath就是可以用**路径表达式来表达XML这些节点啊，节点集**

XPathParser类，中有个方法evalNode方法有个参数，就是我们的路径表达式

这个evalute是个处理的类，接受到我们需要的参数，表达式，document，等参数，然后继续往下，到了**Xpath类（JDK中）**

到evalNode这个类中，用Node这个来接，这个Node就是表示节点的

然后返回一个Xnode，然后再一层一层返回返回回去

XMLConfigBuilder类

到这个类的时候，我们再去处理我们获取的节点到了parseConfiguration这个方法，这个方法获取到的是Xnode这个参数

然后我们需要parseConfiguration这个类中，我们获取到了Xnode中具体传过来的xml具体参数，Xnode这个类，里面都是很多都是基于Node这个类，

其中我们需要分析mappers这个，关键的映射文件，然后我们到了mapperElement这个方法中，这个Xnode是mybatis自己内部实现的，而node是jdk内部的，这个mapperElement这个方法，在这里，我们实现了获取到具体参数，我们将具体参数给获取到，将其获取到，保存为输入流，然后我们存在一个对象，XMLMapperBuilder这个类对应的对象，并且把这个流转换为document，这个流代表的就是我们配置的映射文件
这个方法对应的解析方法就是parse方法，这个parse方法来处理我们获取到的mapper

看源码就是从思路上下手，一步步，的寻找目标，理解其中的具体定义的变量的意义，然后一直往下寻找，在此找到自己的不足，我们再进行补充。

总结：这种框架实现的方法很多
JDBC mybatis，hibernate，这些主要的。。还有一些不常用的spring JDBC.....

mybatis展现的优点：**SQL语句与代码分离**
优点：便于管理与维护
缺点：不便于调试，需要借助日志工具获得信息
2 **用标签控制动态sql的拼接**
优点：用标签代替编写逻辑代码
缺点：拼接复杂的SQL语句时，没有代码灵活，比较复杂
3 **结果集与java对象的自动映射**
优点：保证名称相同就可以进行映射
缺点：对开发人员所写的SQL依赖性很强
配置映射关系自动完成
不配置映射关系
列名=字段名
4 编写原生SQL语句
优势：接近JDBC，比较灵活，
劣势：对SQL语句依赖程度高 ，半自动，数据库移植不方便

有俩个问题在上面尚未解决：一个是配置文件加载时机的问题 ，要等到调用sql语句的时候才加载， 我们可以自己写一个监听器，容器启动的时候加载配置文件
第二个问题：**一个配置文件重复加载的问题，每次调用sql都会去加载，内容其实是相同的，** 可以通过单列模式存放监听器加载的配置内容

1 我们最重要的有看源码的能力，不要害怕源码，源码也是java代码，在看的时候我们能了解到自己的不足，去补充，加强，而且我们看源码要有目标，这是看的时候最重要的，这样的话，我们就能提高自身的代码能力，并且学会别人开源框架大神们解决问题的思维与常用手段。框架的设计思想很重要。

在这里，我们主要涉及了反射，注释，动态代理，设计模式，