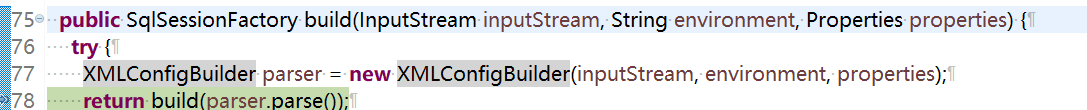
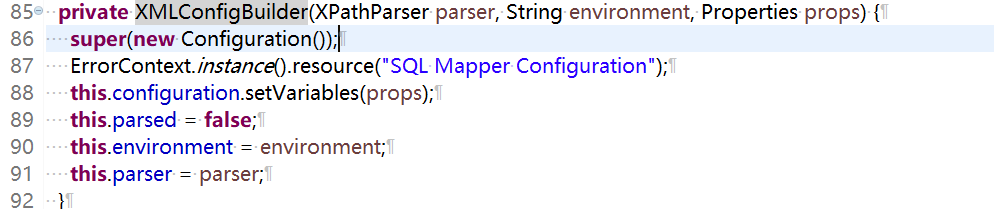
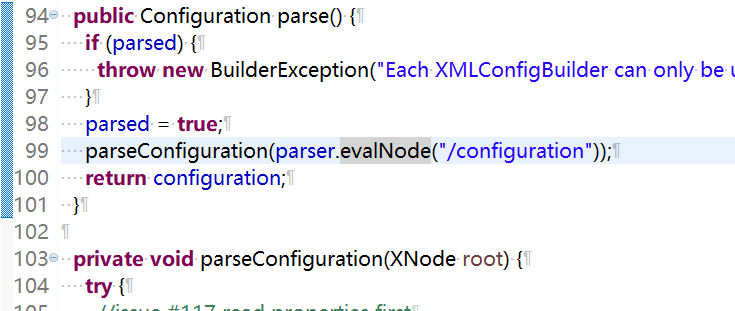
1. **初始化**
2. 初始化SqlSessionFactory，初始化SqlSessionFactory时会先初始化Configuration.java 这个核心配置。

在SqlSessionFactoryBuilder类中，通过构建XMLConfigBuilder在初始化Configuration和解析mybatis配置文件。

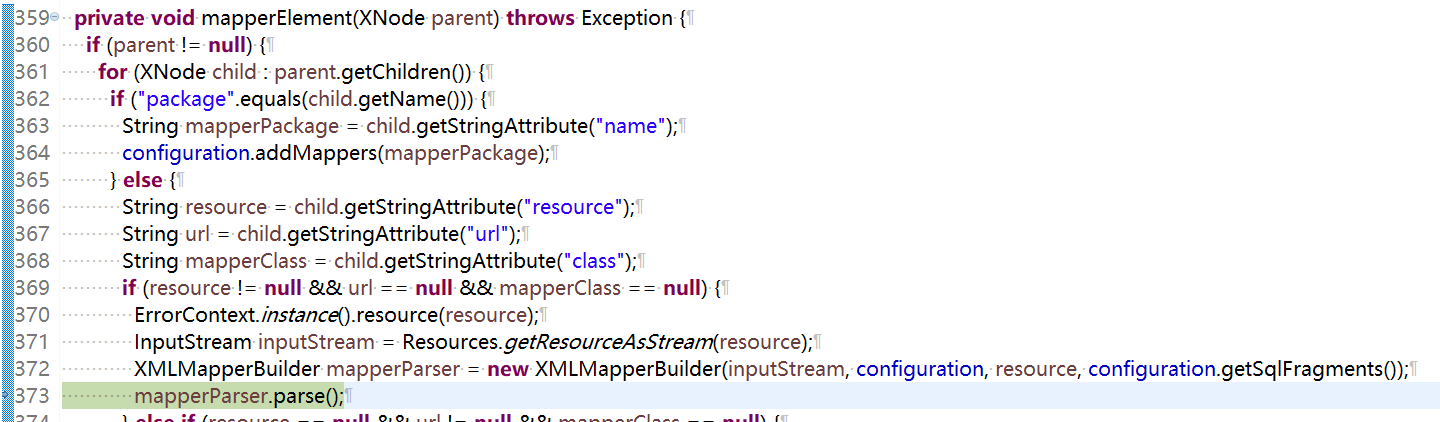




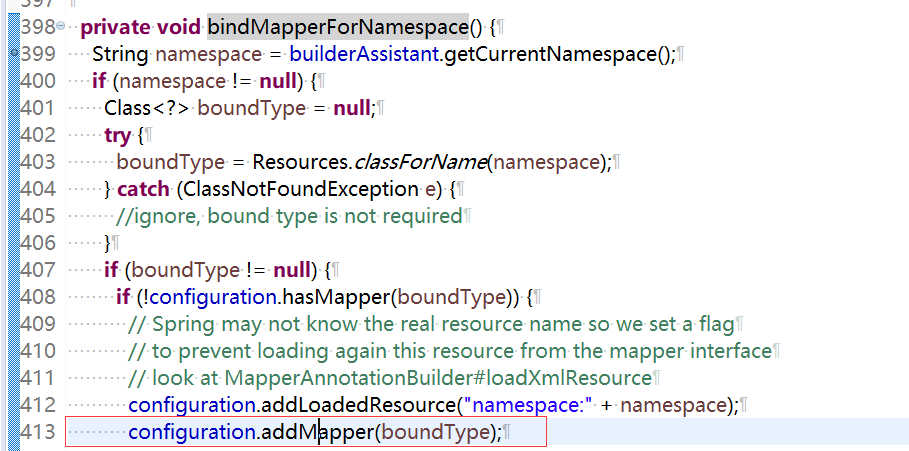
解析配置文件方法是在XMLConfigBuilder类中的parse方法，



1. 解析mapper.xml文件，是在XMLConfigBuilder类中mapperElement方法，解析xml核心类是XMLMapperBuilder类中parse方法负责解析.

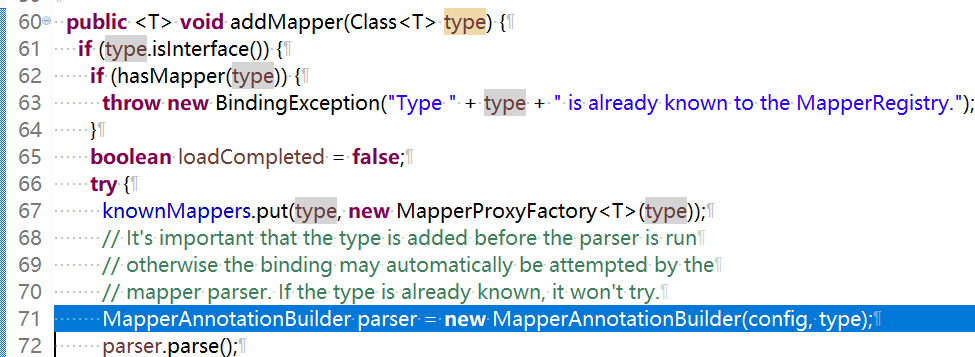


在解析mapper.xml文件时,就会根据xml文件中 namespace (<mapper namespace=*"com.chirq.study.mybatis.mapper.UserMapper"* >)节点对应的接口,创建代理工厂类,详细查看XMLMapperBuilder.Java中parse方法执行的bindMapperForNamespace();方法:



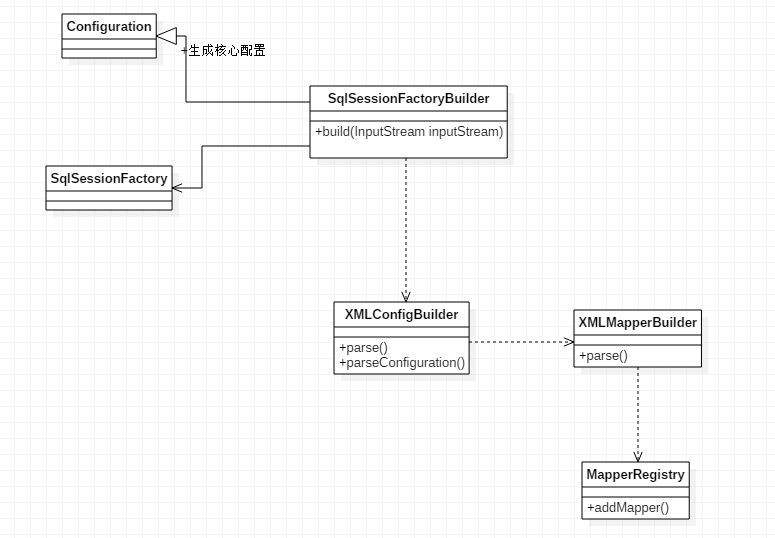
configuration.addMapper(boundType);方法将创建的代理工厂对象存入MapperRegistry类中的MAP缓存中.

在创建代理工厂的同时会解析mapper接口中带注解的方法,方法通过注解实现的则通过MapperAnnotationBuilder类来负责解析.

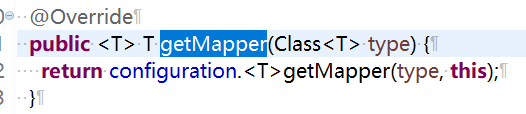


解析出来的方法都存放于Configuration类中的Map<String, MappedStatement> mappedStatements;缓存MAP中.

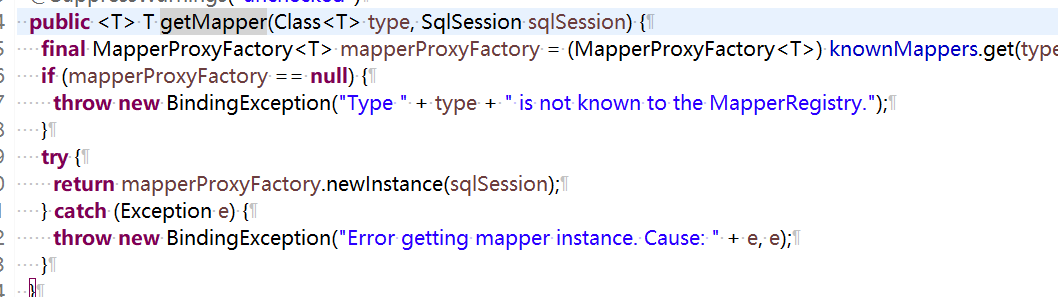
1. 类图



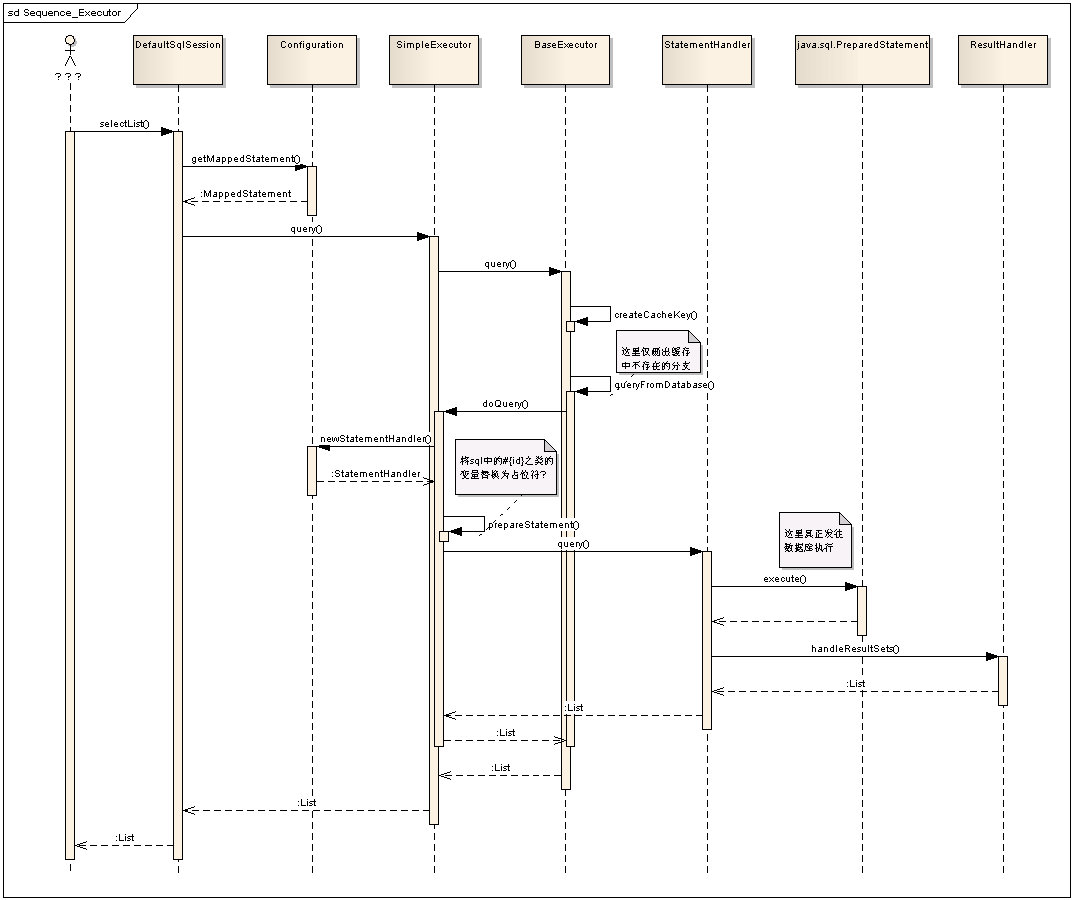
1. **获取SqlSession**
2. SqlSession session = sqlSessionFactory.openSession();获取SqlSession过程中,会先获取数据库连接,开启事务,创建Executor(执行者).
3. UserMapper userMapper = session.getMapper(UserMapper.**class**);获取具体sql执行者,这里就用到了在初始化过程中创建的代理工厂类



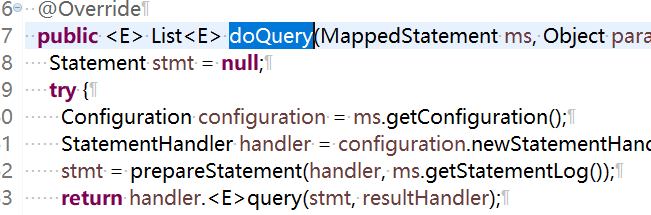
从MapperRegistry类中的MAP缓存中获取事先创建的代理工厂类,然后通过工厂类创建具体的代理MapperProxy:



1. **执行SQL**
2. User user = userMapper.selectByName("张三");
3. 具体执行操作的时序图



1. 查看具体怎么执行的需要从代理类入手MapperProxy类中的invoke方法看起.
2. **参数映射**
3. 在执行器SimpleExecutor执行doQuery方法时,先生成查询预编译处理者StatementHandler类

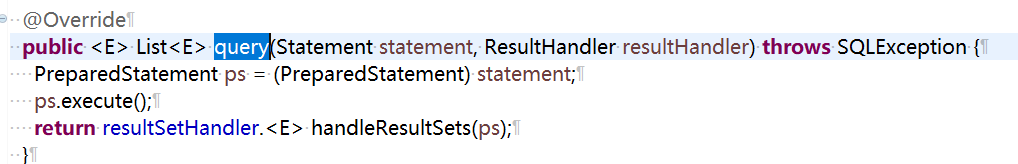


如PreparedStatementHandler类,然后由PreparedStatementHandler调用参数绑定类

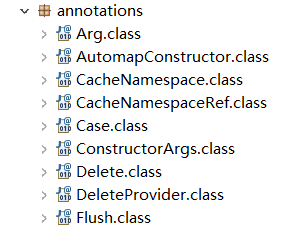
DefaultParameterHandler,参数绑定类会执行setParameters方法进行参数绑定.

1. 预编译处理者PreparedStatementHandler类执行query方法.

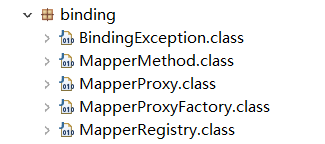
1. PreparedStatementHandler执行SQL,然后将结果集交给结果映射处理类.



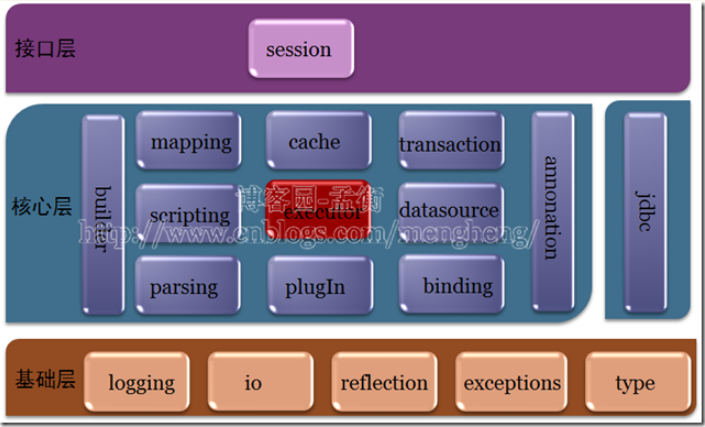
1. **结果映射**
2. 结果集处理类DefaultResultSetHandler执行handleResultSets方法.
3. 结果处理类遍历结果,将查询数据和字段类型转换成ResultSetWrapper类.
4. 然后从xml文件中获取ResultMap配置,即数据库字段到实体配置.
5. 然后执行DefaultResultSetHandler类的handleResultSet方法,该方法将查询结果转换成Java类.
6. 核心方法getRowValue,从查询结果集中获取一行记录并转换成响应JavaBean.
7. 核心方法createResultObject,该方法通过反射生成查询结果要转换成的对象,方法内部调用DefaultObjectFactory类的create方法反射生成对象.
8. 核心方法getPropertyMappingValue,该方法只要功能时从查询结果ResultSet中获取想要的值.
9. 核心方法applyPropertyMappings,该方法是将通过getPropertyMappingValue获取的值,赋值到createResultObject生成的对象中.
10. **包结构分析**
11. org.apache.ibatis.annotations,注解



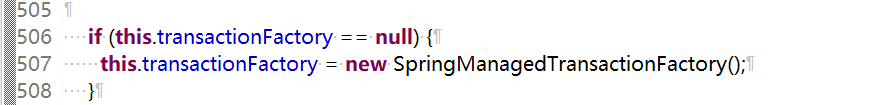
1. binding包,是动态代理相关操作类



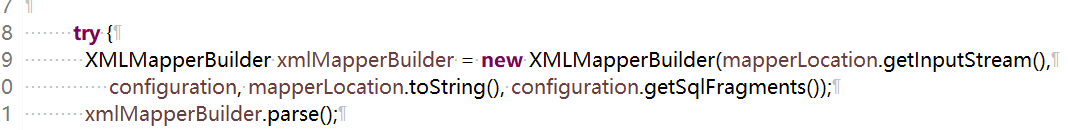
1. builder包,是解析xml配置文件,注解
2. cache缓存所在
3. datasource数据源,连接池
4. executor执行器
5. io文件流
6. mapping包,映射器所在包,从配置文件注解中解析出来的映射器类
7. ognl包,ognl表达式
8. parsing,解析xml工具类
9. plugn,插件
10. Reflection,Java反射,将查询结果动态赋值给java实体等
11. scription,动态SQL脚本处理,例如<set><where>脚本解析
12. session,核心包,启动入口,核心SqlSession,SqlSessionFactiry类等
13. transaction,事务
14. type,强大的类型映射类,从Java到PreparedStatement,从ResultSet到Java



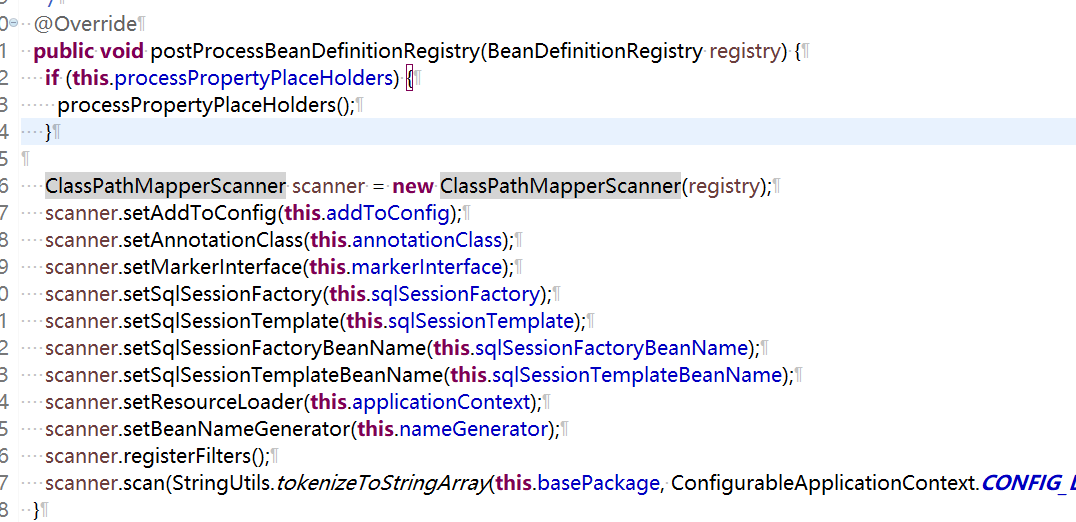
1. **mybatis-spring源码赏析**
2. Mybatis与spring集成，提供SqlSessionFactoryBean类来初始化SqlSessionFactory，在buildSqlSessionFactory方法中，由spring来接管事务



解析mapper.xml



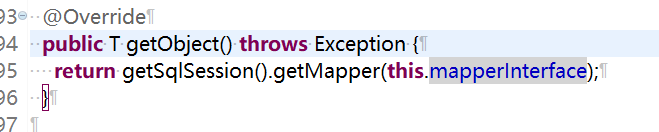
1. MapperScannerConfigurer来创建mapping bean实例，



MapperScannerConfigurer实现了postProcessBeanDefinitionRegistry方法，将已经注册好的Bean重新替换.

ClassPathMapperScanner类作用：

ClassPathMapperScanner的doScan方法,会将mapper包中的类替换成MapperFactoryBean，MapperFactoryBean实现了FactoryBean接口，每次生成对应Bean的时候，其实调用的FactoryBean的getObject方法。也就是说我们在注入mappler的时候，其实调用的是MapperFactoryBean的getObject方法，而这个方法会生成mappler接口的代理类MapperProxy的实例。



此处需要注意的是**public** **class** MapperFactoryBean<T> **extends** SqlSessionDaoSupport **implements** FactoryBean<T> ,MapperFactoryBean还继承了SqlSessionDaoSupport，该类偷梁换柱的把SqlSession替换成SqlSessionTemplate，详细查看getSqlSession方法。

1. SqlSessionTemplate作用

该类改用动态代理，在具体生成mybatis的DefaultSqlSession，执行mybatis操作前获取数据库连接和事务，在mybatis操作后执行资源释放，就不用再代码层面手动执行session.close();

