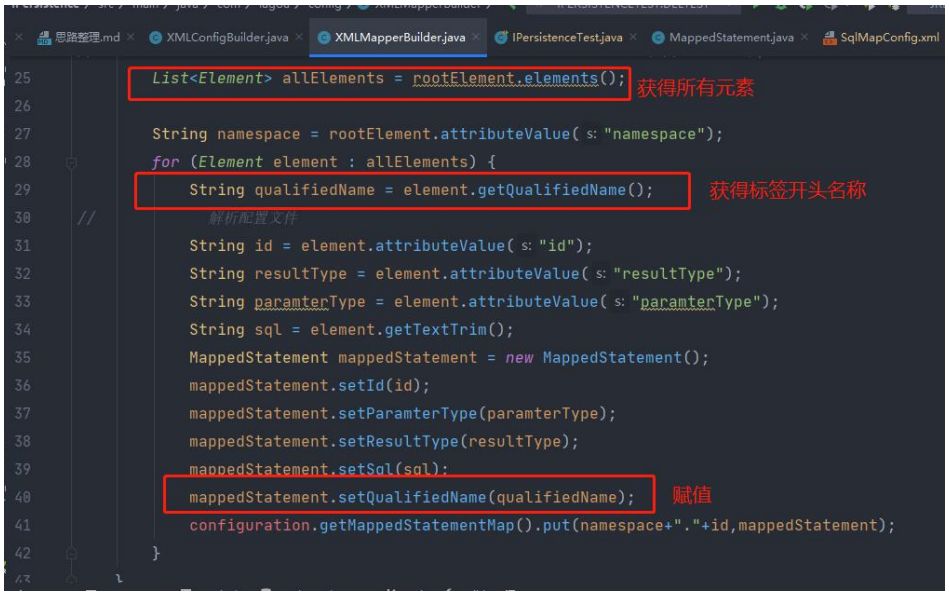
思路讲解

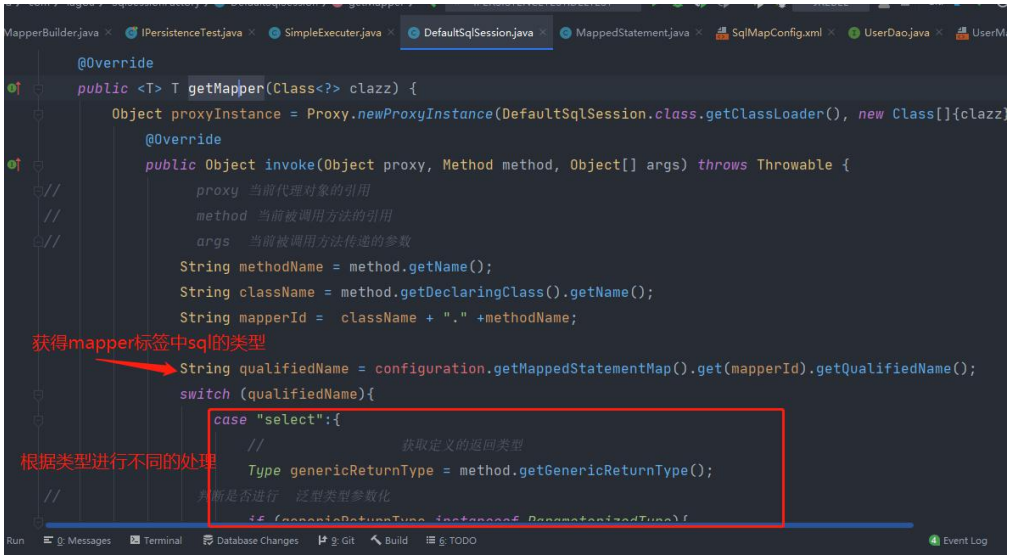
1. 为了便于和 select 语句进行区分，并且因为是数据的增删改操作，不需要返回具体的数据，所以为了方便您区分，建议和 select 语句标签进行区分。首先需要解析 insert、update、delete 语句 SQL，在 MappedStatement 中添加字段 qualifiedName（xml 中语句标签的开头，如<select>、<insert>）

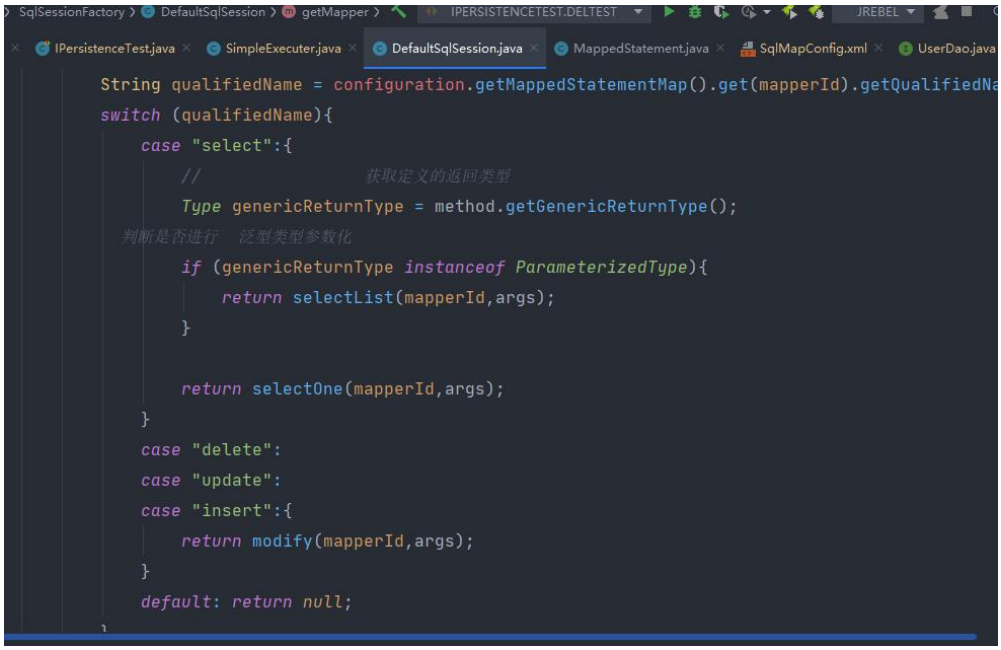
2. 解析 mapper 文件时，将只解析 select 标签改为解析所有标签，并且将标签的类型存入到 MappedStatement 中



3. 修改 com.lagou.SqlSessionFactory.DefaultSqlSession#getMapper 方法，添加对增删改操作的处理，因为增删改操作不需要返回被操作的数据，所以可以共用相同的代码。

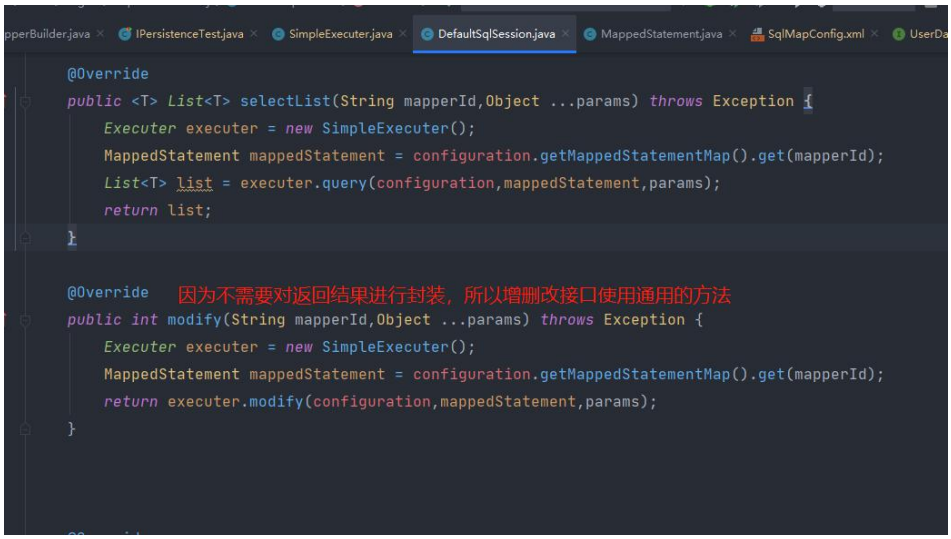
增删改操作和查询操作最主要的区别在于 SQL 执行后对于结果的处理，查询操作需要封装查询结果并且返回，而增删改可以直接返回 preparedStatement.executeUpdate()方法执行的结果。





```
String qualifiedName = configuration.getMappedStatementMap().get(mapperId).getQualifiedNa
switch (qualifiedName){
    case "select":{
        //                获取定义的返回类型
        Type genericReturnType = method.getGenericReturnType();
     判断是否进行   泛型类型参数化
        if (genericReturnType instanceof ParameterizedType){
            return selectList(mapperId,args);
        }

        return selectOne(mapperId,args);
    }
    case "delete":
    case "update":
    case "insert":{
        return modify(mapperId,args);
    }
    default: return null;
```



```
@Override
public <T> List<T> selectList(String mapperId,Object ...params) throws Exception {
    Executer executer = new SimpleExecuter();
    MappedStatement mappedStatement = configuration.getMappedStatementMap().get(mapperId);
    List<T> list = executer.query(configuration,mappedStatement,params);
    return list;
}

@Override    因为不需要对返回结果进行封装，所以增删改接口使用通用的方法
public int modify(String mapperId,Object ...params) throws Exception {
    Executer executer = new SimpleExecuter();
    MappedStatement mappedStatement = configuration.getMappedStatementMap().get(mapperId);
    return executer.modify(configuration,mappedStatement,params);
}
```

代码如下：

```
    @Override
    public int modify(Configuration configuration, MappedStatement mappedStatement, Object... params) throws Exception {
        List<Object> execute = execute(configuration, mappedStatement, params);

        int updateCount = (Integer) execute.get(0);
        return updateCount;
    }
```

```
  private <T> List<T> execute(Configuration configuration, MappedStatement mappedStatement, Object[] params) throws Exception {
        //          注册驱动
        Connection connection = configuration.getDateSource().getConnection();
//          获得 sql
        String sql = mappedStatement.getSql();
```

```
//            解析 sql
            BoundSql boundSql= getBoundSql(sql);
            String paramterType = mappedStatement.getParamterType();
            Class<?> paramterTypeClass =    getTypeClass(paramterType);
            PreparedStatement                    preparedStatement                    =
connection.prepareStatement(boundSql.getSql());
            for (int i = 0; i < boundSql.getParameterMappings().size(); i++) {
                Object param = params[0];
                ParameterMapping                    parameterMapping                    =
boundSql.getParameterMappings().get(i);
//                使用反射获得参数对应字段
                Field                    declaredField                    =
paramterTypeClass.getDeclaredField(parameterMapping.getContent());
//                暴力破解
                declaredField.setAccessible(true);
//                开始设置参数
//                  todo 注意点：list 角标从 0 开始，但是 preparedStatement 角标从 1 开始，
所以需要使 list 角标 加一
                preparedStatement.setObject(i+1,declaredField.get(param));
            }
//            执行 sql
            List result = null;
            switch (mappedStatement.getQualifiedName()){
                case "select": {
                    preparedStatement.execute();
                    ResultSet resultSet = preparedStatement.getResultSet();
                    ArrayList<Object> results = new ArrayList<>();
                    String resultType = mappedStatement.getResultType();
//                封装返回结果集
                    while (resultSet.next()){
                        Class<?> typeClass = getTypeClass(resultType);
                        Object o = typeClass.newInstance();

//                获取源
                        ResultSetMetaData metaData = resultSet.getMetaData();
                        for (int i = 1; i <= metaData.getColumnCount(); i++) {

                            Object value = resultSet.getObject(metaData.getColumnName(i));
//                          todo 使用内省进行赋值
                            PropertyDescriptor          propertyDescriptor          =          new
PropertyDescriptor(metaData.getColumnName(i), typeClass);
                            Method writeMethod = propertyDescriptor.getWriteMethod();
                            writeMethod.invoke(o,value);
                        }
```

```
                        results.add(o);
                    }
                    result=results;
                    break;
                }
// 增删改操作代码
                case "insert":
                case "update":
                case "delete":{
                    int update = preparedStatement.executeUpdate();
                    result= new ArrayList<>();
                    result.add(update);
                    break;
                }
                default: ;
            }
            return result;
        }
```