- MappedStatement 类是 Mybatis 框架的核心类之一，它存储了一个 sql 对应的所有信息
- Mybatis 通过解析 XML 和 mapper 接口上的注解，生成 sql 对应的 MappedStatement 实例，并放入 SqlSessionTemplate 中 configuration 类属性中
- 正真执行 mapper 接口中的方法时，会从 configuration 中找到对应的 mappedStatement，然后进行后续的操作
- 本文将通过源码介绍 mappedStatement 生成的全过程

---

1 生成 MappedStatement 的入口

- 在 Springboot 模块分析 —— MybatisAutoConfiguration 解析 一文中，介绍了生成 mapper 接口对应 beanDefintion 的流程，并且可以看出 生成 mappedStatement 有 3 条路径
  - 如果定义了 mapperLocation，在初始化 SqlSessionFactoryBean 后，回去解析 mapperLocation 对应的 xml 文件，解析生成 xml 中每一个 sql 对应的 mappedStatement 实例，并加入 configuration 中
  - 如果没有定义 mapperLocation，当解析 mapper 接口对应的 beanDefinition 后，会调用 MapperFactoryBean#checkDaoConfig() 方法，通过接口全路径名(将.java变为.xml)，来找对应的 xml
  - 上一步解析完了 xml 后，会去遍历 mapper 接口中的方法，如果方法中有 sql 相关注解，就会生成 sqlSource 并覆盖上一步生成的 mappedStatement！
- 综上， mappedStatement 生成的入口有 3 个，优先级为：mapperLocation > 注解 sql > 接口对应 xml 路径

2 MappedStatement 生成源码分析

- 当实例化 mapper 接口对应的 beanDefintion 后，会调用 MapperFactoryBean#checkDaoConfig() 方法进行生成，本小节详细分析源码过程
- 最后通过一些参数初始化了 mappedStatement，观察这些参数是怎么来的，可以更好的做参数设置，此处留待以后分析
- 源码中也涉及到了各个注解的解析，观察相应代码可以清楚的理解各个注解的使用，，此处留待以后分析

2.1 checkDaoConfig() — 生成 MappedStatement 总流程

```
protected void checkDaoConfig() {
  super.checkDaoConfig();
  Configuration configuration = getSqlSession().getConfiguration();
  //如果定义了 mapperLocation，并且包含 mapperInterface 对应的 xml，则此时的
configuration.hasMapper() 返回 true，这样就避免了重复解析
  if (this.addToConfig && !configuration.hasMapper(this.mapperInterface)) {
      configuration.addMapper(this.mapperInterface);
```

```
  }
}
public <T> void addMapper(Class<T> type) {
    if (type.isInterface()) {
        MapperAnnotationBuilder parser = new MapperAnnotationBuilder(config, type);
        parser.parse();//
    }
}
public void parse() {
    String resource = type.toString();
    //又一次判断，避免重复解析
    if (!configuration.isResourceLoaded(resource)) {
        loadXmlResource();// xml 解析流程！解析 xml 中 sql 生成 mappedStatement
        configuration.addLoadedResource(resource);
        assistant.setCurrentNamespace(type.getName());
        parseCache();
        parseCacheRef();
        //遍历 mapper 接口中的所有方法
        Method[] methods = type.getMethods();
        for (Method method : methods) {
            parseStatement(method);
        }
    }
    parsePendingMethods();
}
```

2.2 loadXmlResource() — 解析 xml 生成 MappedStatement

- 关键-关键-关键：这里的 xml 路径是，接口的全名路径，并加后缀转为 .xml

```
private void loadXmlResource() {
  //又一次判断，避免重复解析
  if (!configuration.isResourceLoaded("namespace:" + type.getName())) {
    //关键-关键-关键：这里的 xml 路径是，接口的全名路径，并加后缀转为 .xml
    String xmlResource = type.getName().replace('.', '/') + ".xml";
    InputStream inputStream = type.getResourceAsStream("/" + xmlResource);
    if (inputStream == null) {
      // Search XML mapper that is not in the module but in the classpath.
        inputStream = Resources.getResourceAsStream(type.getClassLoader(),
xmlResource);
    }
    if (inputStream != null) {
      XMLMapperBuilder xmlParser = new XMLMapperBuilder(inputStream,
assistant.getConfiguration(), xmlResource, configuration.getSqlFragments(),
type.getName());
      xmlParser.parse();//开始解析
    }
  }
}
public void parse() {
    if (!configuration.isResourceLoaded(resource)) {
        configurationElement(parser.evalNode("/mapper"));
        configuration.addLoadedResource(resource);
```

```
            bindMapperForNamespace();
        }
        //后面 3 个不知道干啥的
    }
    private void configurationElement(XNode context) {
        String namespace = context.getStringAttribute("namespace");
        builderAssistant.setCurrentNamespace(namespace);
        cacheRefElement(context.evalNode("cache-ref"));
        cacheElement(context.evalNode("cache"));
        parameterMapElement(context.evalNodes("/mapper/parameterMap"));
        resultMapElements(context.evalNodes("/mapper/resultMap"));
        sqlElement(context.evalNodes("/mapper/sql"));
        buildStatementFromContext(context.evalNodes("select|insert|update|delete"));
    }
    private void buildStatementFromContext(List<XNode> list) {
        //databaseId 用于多数据处理？
        if (configuration.getDatabaseId() != null) {
            buildStatementFromContext(list, configuration.getDatabaseId());
        }
        buildStatementFromContext(list, null);
    }
    private void buildStatementFromContext(List<XNode> list, String requiredDatabaseId) {
        //遍历所有的 sql 节点，解析对应的 mappedStatement
        for (XNode context : list) {
            final XMLStatementBuilder statementParser = new
    XMLStatementBuilder(configuration, builderAssistant, context, requiredDatabaseId);
            try {
                statementParser.parseStatementNode();
            } catch (IncompleteElementException e) {
                configuration.addIncompleteStatement(statementParser);
            }
        }
    }
```

2.2.1 遍历所有的 sql 节点，解析对应的 mappedStatement

- 关键-关键-关键：从这里可以看出，xml sql 节点的 id 必须和方法名一致才有效

```
    public void parseStatementNode() {
      String id = context.getStringAttribute("id");
      String databaseId = context.getStringAttribute("databaseId");
      if (!databaseIdMatchesCurrent(id, databaseId, this.requiredDatabaseId)) {
        return;//多数据源处理？
      }
      //省略所有生成以下方法中参数的代码，通过这些参数最终初始化 mappedStatement
      //观察这些参数是怎么来的，可以更好的做参数设置,此处省略
      builderAssistant.addMappedStatement(id, sqlSource, statementType, sqlCommandType,
          fetchSize, timeout, parameterMap, parameterTypeClass, resultMap, resultTypeClass,
          resultSetTypeEnum, flushCache, useCache, resultOrdered,
          keyGenerator, keyProperty, keyColumn, databaseId, langDriver, resultSets);
    }
    public MappedStatement addMappedStatement() {
        //这里的 id 还只是到 接口的路径，需要将 xml sql 节点的 id 加到后面，形成完整的 id
```

```
    //关键-关键-关键：从这里可以看出，xml sql 节点的 id 必须和方法名一致才有效
    id = applyCurrentNamespace(id, false);
    boolean isSelect = sqlCommandType == SqlCommandType.SELECT;
    MappedStatement.Builder statementBuilder = 构建器;
    //获取 parameterMap
    ParameterMap statementParameterMap = getStatementParameterMap(parameterMap,
parameterType, id);
    if (statementParameterMap != null) {
      statementBuilder.parameterMap(statementParameterMap);
    }
    //创建 mappedStatement
    MappedStatement statement = statementBuilder.build();
    //加入 configuration
    configuration.addMappedStatement(statement);
    return statement;
}
```

2.3 parseStatement() — 解析接口方法注解生成 MappedStatement

- 关键-关键-关键：如果方法中有 sql 相关注解，则会覆盖 xml 中的 mappedStatement

```
void parseStatement(Method method) {
  //生成方法参数的类型，如果有多个参数，则为 MapperMethod$ParamMap 类型
  Class<?> parameterTypeClass = getParameterType(method);
  //方法中的 @Lang 注解，默认为 XMLLanguageDriver
  LanguageDriver languageDriver = getLanguageDriver(method);
  //方法中 sql 相关注解，如果有则生成对应 sqlSource
  SqlSource sqlSource = getSqlSourceFromAnnotations(method, parameterTypeClass,
languageDriver);
  if (sqlSource != null) {
    //关键-关键-关键：如果方法中有 sql 相关注解，则会覆盖 xml 中的 mappedStatement
    Options options = method.getAnnotation(Options.class);
    final String mappedStatementId = type.getName() + "." + method.getName();
    //省略所有生成以下方法中参数的代码，通过这些参数最终初始化 mappedStatement
    //观察这些参数是怎么来的，可以更好的做参数设置,此处省略

    //对比解析 xml 中的该方法，可以看出，这里的 parameterMapID 和 databaseID 没有
    //说明注解 sql，这两个参数没有用
    assistant.addMappedStatement(mappedStatementId,sqlSource,statementType,
        sqlCommandType,fetchSize,timeout,
        // ParameterMapID
        null,
        parameterTypeClass,resultMapId,getReturnType(method),resultSetType,flushCache,
        useCache,
        // TODO gcode issue #577
        false,keyGenerator,keyProperty,keyColumn,
        // DatabaseID
        null,languageDriver,
        // ResultSets
        options != null ? nullOrEmpty(options.resultSets()) : null);
  }
}
```

3 MappedStatement 的作用

- MappedStatement 的作用可参考 mapper 接口方法运行流程