

# MyBatis解析全局配置文件

有道云链接: [http://note.youdao.com/noteshare?](http://note.youdao.com/noteshare?id=afc77bcd736625615de69179b2fac56f&sub=EB1083FF31A64AAC93C2AEA47C1A3A63)

[id=afc77bcd736625615de69179b2fac56f&sub=EB1083FF31A64AAC93C2AEA47C1A3A63](http://note.youdao.com/noteshare?id=afc77bcd736625615de69179b2fac56f&sub=EB1083FF31A64AAC93C2AEA47C1A3A63)

MyBatis解析全局配置文件

MyBatis介绍

传统JDBC和Mybatis相比的弊病

MyBatis 源码编译

启动流程分析

简单总结

文档: MyBatis的二级缓存原理分析 (请暂时关注解析缓存内容)

文档: Mybatis解析动态sql原理分析

文档: Mybatis-设计模式总结.note

## MyBatis介绍

MyBatis是一个持久层的ORM框架, 使用简单, 学习成本较低。可以执行自己手写的SQL语句, 比较灵活。但是MyBatis的自动化程度不高, 移植性也不高, 有时从一个数据库迁移到另外一个数据库的时候需要自己修改配置, 所以称只为准自动ORM框架

MyBatis基础应用:

文档: MyBatis

链接: <http://note.youdao.com/noteshare?id=5d41fd41d970f1af9185ea2ec0647b64>

## 传统JDBC和Mybatis相比的弊病

传统JDBC

```
1 @Test
2 public void test() throws SQLException {
```

```
3 Connection conn=null;
4 PreparedStatement pstmt=null;
5 try {
6     // 1.加载驱动
7     Class.forName("com.mysql.jdbc.Driver");
8
9     // 2.创建连接
10    conn= DriverManager.
11        getConnection("jdbc:mysql://localhost:3306/mybatis_example", "root",
12            "123456");
13
14    // SQL语句
15    String sql="select id,user_name,create_time from t_user where id=?";
16
17    // 获得sql执行者
18    pstmt=conn.prepareStatement(sql);
19    pstmt.setInt(1,1);
20
21    // 执行查询
22    //ResultSet rs= pstmt.executeQuery();
23    pstmt.execute();
24    ResultSet rs= pstmt.getResultSet();
25
26    rs.next();
27    User user =new User();
28    user.setId(rs.getLong("id"));
29    user.setUserName(rs.getString("user_name"));
30    user.setCreateTime(rs.getDate("create_time"));
31    System.out.println(user.toString());
32 } catch (Exception e) {
33     e.printStackTrace();
34 }
35 finally{
36     // 关闭资源
37     try {
38         if(conn!=null){
39             conn.close();
```

```
40 }
41 if(pstmt!=null){
42     pstmt.close();
43 }
44 } catch (SQLException e) {
45     e.printStackTrace();
46 }
47 }
48 }
```

### 传统JDBC的问题如下:

- 1.数据库连接创建，释放频繁造成系统资源的浪费，从而影响系统性能，使用数据库连接池可以解决问题。
- 2.sql语句在代码中硬编码，造成代码的不易维护，实际应用中sql的变化可能较大，sql代码和java代码没有分离开来维护不方便。
- 3.使用preparedStatement向有占位符传递参数存在硬编码问题因为sql中的where子句的条件不确定，同样是修改不方便/
- 4.对结果集中解析存在硬编码问题，sql的变化导致解析代码的变化，系统维护不方便。

### mybatis对传统的JDBC的解决方案

- 1、数据库连接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库连接池可解决此问题。

解决：在SqlMapConfig.xml中配置数据库连接池，使用连接池管理数据库链接。

- 2、Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

解决：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。

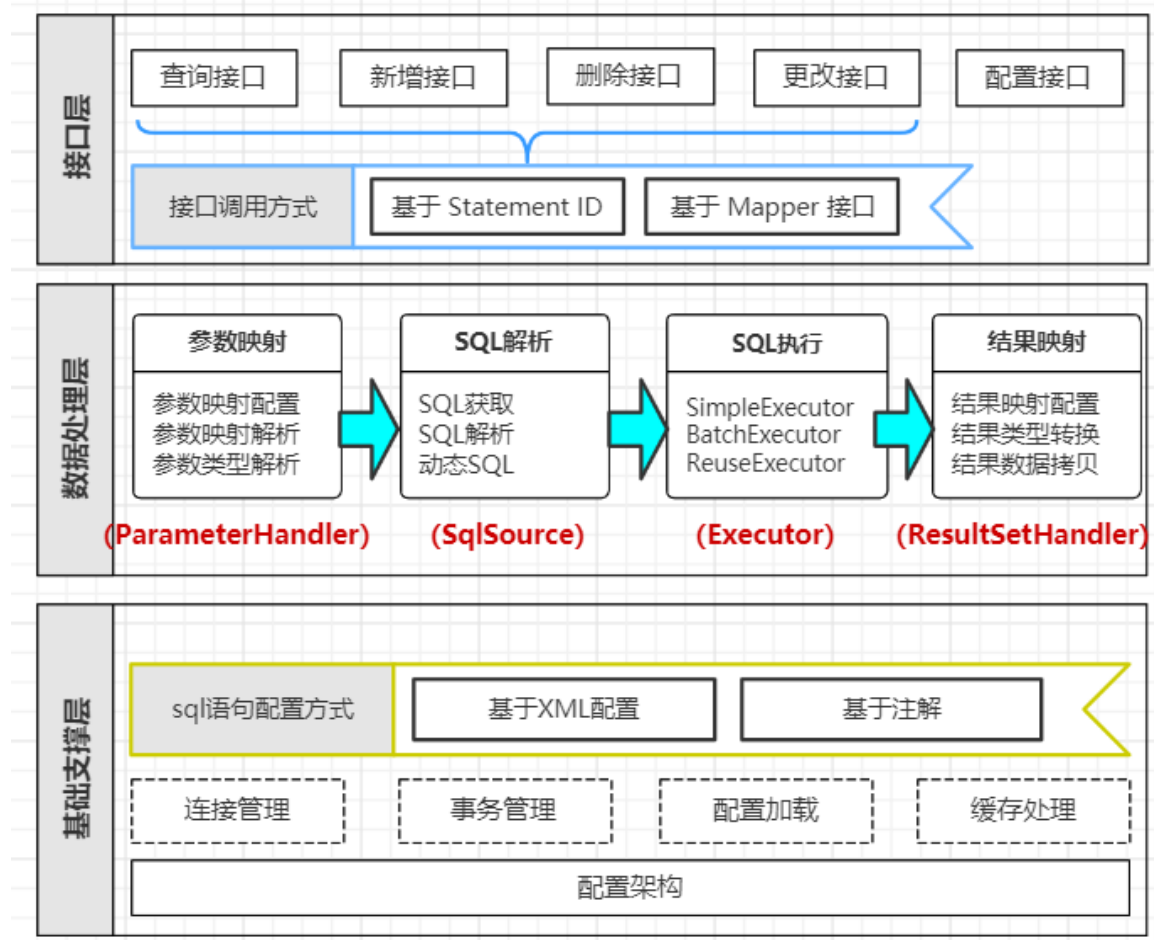
- 3、向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。

解决：Mybatis自动将java对象映射至sql语句，通过statement中的parameterType定义输入参数的类型。

4、对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。

解决：Mybatis自动将sql执行结果映射至java对象，通过statement中的resultType定义输出结果的类型。

## Mybaitis整体体系图



我们把Mybatis的功能架构分为三层：

- API接口层：提供给外部使用的接口API，开发人员通过这些本地API来操纵数据库。接口层一接收到调用请求就会调用数据处理层来完成具体的数据处理。
- 数据处理层：负责具体的SQL查找、SQL解析、SQL执行和执行结果映射处理等。它主要的目的是根据调用的请求完成一次数据库操作。
- 基础支撑层：负责最基础的功能支撑，包括连接管理、事务管理、配置加载和缓存处理，这些都是共用的东西，将他们抽取出来作为最基础的组件。为上层的数据处理层提供最基础的支撑。

https://www.cnblogs.com/ymy/p/11111111.html

一个Mybatis最简单的使用列子如下：

```

1
2  /**
3   * @Author 徐庶 QQ:1092002729
4   * @Slogan 致敬大师，致敬未来的你
5   */
6  public class App {
7      public static void main(String[] args) {
8          String resource = "mybatis-config.xml";
9          Reader reader;
10         try {
11             //将XML配置文件构建为Configuration配置类
12             reader = Resources.getResourceAsReader(resource);
13             // 通过加载配置文件流构建一个SqlSessionFactory DefaultSqlSessionFactory
14             SqlSessionFactory sqlMapper = new SqlSessionFactoryBuilder().build(reader);
15             // 数据源 执行器 DefaultSqlSession
16             SqlSession session = sqlMapper.openSession();
17             try {
18                 // 执行查询 底层执行jdbc
19                 //User user = (User)session.selectOne("com.tuling.mapper.selectById", 1);
20
21                 UserMapper mapper = session.getMapper(UserMapper.class);
22                 System.out.println(mapper.getClass());
23                 User user = mapper.selectById(1L);
24                 System.out.println(user.getUserName());
25             } catch (Exception e) {
26                 e.printStackTrace();
27             }finally {
28                 session.close();
29             }
30             } catch (IOException e) {
31                 e.printStackTrace();
32             }
33         }
34     }

```

总结下就是分为下面四个步骤：

- 从配置文件（通常是XML文件）得到SessionFactory;
- 从SessionFactory得到SqlSession;
- 通过SqlSession进行CRUD和事务的操作;
- 执行完相关操作之后关闭Session。

## MyBatis 源码编译

MyBatis的源码编译比较简单，随便在网上找一篇博客即可，在这里不多说

<https://www.cnblogs.com/mokingone/p/9108999.html>

如果希望在spring源码中引入你自己的这份源码，可以做如下操作

1. 修改你mybatis源码的pom的<version> 这样可以和官方的区分开来

```
1 <version>3.5.3-xs1s</version>
```

2. 这样你在spring源码中就可以引入这份mybatis源码了.

a. 如果引入mybatis-spring 同样需要做1、3步骤

```
1 compile("org.mybatis:mybatis-spring:2.0.3-xs1s")
2 compile("org.mybatis:mybatis:3.5.3-xs1s")
```

3. 当然，如果你想在spring这边看到你mybatis源码相关的注释，还得在mybatis源码的pom里面加入plugin,使它生成 jar 的同时 生成 sources 包

```
1 <plugin>
2   <artifactId>maven-source-plugin</artifactId>
3   <version>3.0.1</version>
4   <configuration>
5     <attach>true</attach>
6   </configuration>
7   <executions>
8     <execution>
9       <phase>compile</phase>
10      <goals>
11        <goal>jar</goal>
12      </goals>
13    </execution>
14  </executions>
15 </plugin>
```

## 启动流程分析

```
1 String resource = "mybatis-config.xml";
```

```

2 //将XML配置文件构建为Configuration配置类
3 reader = Resources.getResourceAsReader(resource);
4 // 通过加载配置文件流构建一个SqlSessionFactory DefaultSqlSessionFactory
5 SqlSessionFactory sqlMapper = new SqlSessionFactoryBuilder().build(reader);

```

通过上面代码发现，创建SqlSessionFactory的代码在SqlSessionFactoryBuilder中，进去一探究竟：

```

1 //整个过程就是将配置文件解析成Configuration对象，然后创建SqlSessionFactory的过程
2 //Configuration是SqlSessionFactory的一个内部属性
3 public SqlSessionFactory build(InputStream inputStream, String environment, Properties properties) {
4     try {
5         XMLConfigBuilder parser = new XMLConfigBuilder(inputStream, environment, properties);
6         return build(parser.parse());
7     } catch (Exception e) {
8         throw ExceptionFactory.wrapException("Error building SqlSession.", e);
9     } finally {
10         ErrorContext.instance().reset();
11         try {
12             inputStream.close();
13         } catch (IOException e) {
14             // Intentionally ignore. Prefer previous error.
15         }
16     }
17 }
18
19 public SqlSessionFactory build(Configuration config) {
20     return new DefaultSqlSessionFactory(config);
21 }

```

下面我们看下解析配置文件过程中的一些细节。

先给出一个配置文件的例子：

Copy

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3     PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

```

```
4  "http://mybatis.org/dtd/mybatis-3-config.dtd">
5  <configuration>
6  <!--SqlSessionFactoryBuilder中配置的配置文件的优先级最高；config.properties配置文件的优先级次之；properties标签中的配置优先级最低 -->
7  <properties resource="org/mybatis/example/config.properties">
8  <property name="username" value="dev_user"/>
9  <property name="password" value="F2Fa3!33TYyg"/>
10 </properties>
11
12 <!--一些重要的全局配置-->
13 <settings>
14 <setting name="cacheEnabled" value="true"/>
15 <!--<setting name="lazyLoadingEnabled" value="true"/>-->
16 <!--<setting name="multipleResultSetsEnabled" value="true"/>-->
17 <!--<setting name="useColumnLabel" value="true"/>-->
18 <!--<setting name="useGeneratedKeys" value="false"/>-->
19 <!--<setting name="autoMappingBehavior" value="PARTIAL"/>-->
20 <!--<setting name="autoMappingUnknownColumnBehavior"
    value="WARNING"/>-->
21 <!--<setting name="defaultExecutorType" value="SIMPLE"/>-->
22 <!--<setting name="defaultStatementTimeout" value="25"/>-->
23 <!--<setting name="defaultFetchSize" value="100"/>-->
24 <!--<setting name="safeRowBoundsEnabled" value="false"/>-->
25 <!--<setting name="mapUnderscoreToCamelCase" value="false"/>-->
26 <!--<setting name="localCacheScope" value="STATEMENT"/>-->
27 <!--<setting name="jdbcTypeForNull" value="OTHER"/>-->
28 <!--<setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode,toString"/>-->
29 <!--<setting name="logImpl" value="STDOUT_LOGGING" />-->
30 </settings>
31
32 <typeAliases>
33
34 </typeAliases>
35
36 <plugins>
37 <plugin interceptor="com.github.pagehelper.PageInterceptor">
38 <!--默认值为 false，当该参数设置为 true 时，如果 pageSize=0 或者 RowBounds.limit = 0 就会查询出全部的结果-->
```



```

39  <!--如果某些查询数据量非常大，不应该允许查出所有数据-->
40  <property name="pageSizeZero" value="true"/>
41  </plugin>
42  </plugins>
43
44  <environments default="development">
45    <environment id="development">
46      <transactionManager type="JDBC"/>
47      <dataSource type="POOLED">
48        <property name="driver" value="com.mysql.jdbc.Driver"/>
49        <property name="url" value="jdbc:mysql://10.59.97.10:3308/windty"/>
50        <property name="username" value="windty_opr"/>
51        <property name="password" value="windty!234"/>
52      </dataSource>
53    </environment>
54  </environments>
55
56  <databaseIdProvider type="DB_VENDOR">
57    <property name="MySQL" value="mysql" />
58    <property name="Oracle" value="oracle" />
59  </databaseIdProvider>
60
61  <mappers>
62    <!--这边可以使用package和resource两种方式加载mapper-->
63    <!--<package name="包名"/>-->
64    <!--<mapper resource="./mappers/SysUserMapper.xml"/>-->
65    <mapper resource="./mappers/CbondissuerMapper.xml"/>
66  </mappers>
67
68  </configuration>

```

下面是解析配置文件的核心方法：

Copy

```

1  private void parseConfiguration(XNode root) {
2    try {
3      //issue #117 read properties first
4      //解析properties标签，并set到Configuration对象中
5      //在properties配置属性后，在Mybatis的配置文件中就可以使用${key}的形式使用
      了。

```

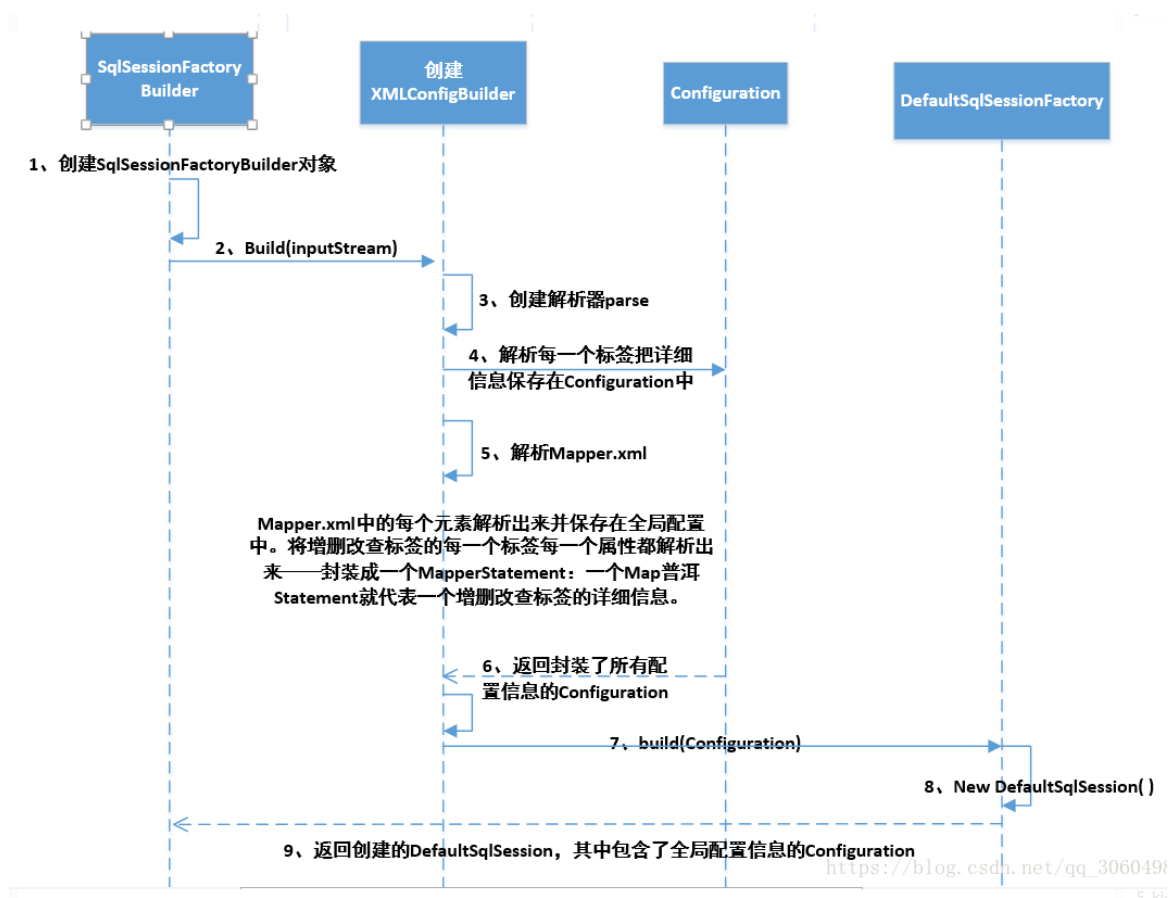
```

6  propertiesElement(root.evalNode("properties"));
7
8  //解析setting标签的配置
9  Properties settings =
  settingsAsProperties(root.evalNode("settings"));
10 //添加vfs的自定义实现，这个功能不怎么用
11 loadCustomVfs(settings);
12
13 //配置类的别名，配置后就可以用别名来替代全限定名
14 //mybatis默认设置了很多别名，参考附录部分
15 typeAliasesElement(root.evalNode("typeAliases"));
16
17 //解析拦截器和拦截器的属性，set到Configuration的interceptorChain中
18 //MyBatis 允许你在已映射语句执行过程中的某一点进行拦截调用。默认情况下，MyBatis 允许使用插件来拦截的方法调用包括：
19 //Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
20 //ParameterHandler (getParameterObject, setParameters)
21 //ResultSetHandler (handleResultSets, handleOutputParameters)
22 //StatementHandler (prepare, parameterize, batch, update, query)
23 pluginElement(root.evalNode("plugins"));
24
25 //Mybatis创建对象是会使用objectFactory来创建对象，一般情况下不会自己配置这个objectFactory，使用系统默认的objectFactory就好了
26 objectFactoryElement(root.evalNode("objectFactory"));
27 objectWrapperFactoryElement(root.evalNode("objectWrapperFactory"));
28 reflectorFactoryElement(root.evalNode("reflectorFactory"));
29
30 //设置在setting标签中配置的配置
31 settingsElement(settings);
32
33 //解析环境信息，包括事物管理器和数据源，SqlSessionFactoryBuilder在解析时需要指定环境id，如果不指定的话，会选择默认的环境；
34 //最后将这些信息set到Configuration的Environment属性里面
35 environmentsElement(root.evalNode("environments"));
36
37 //
38 databaseIdProviderElement(root.evalNode("databaseIdProvider"));
39
40 //无论是 MyBatis 在预处理语句（PreparedStatement）中设置一个参数时，还是从结果集中取出一个值时，都会用类型处理器将获取的值以合适的方式转换成 Java 类

```

型。解析typeHandler。

```
41 typeHandlerElement(root.evalNode("typeHandlers"));
42 //解析Mapper
43 mapperElement(root.evalNode("mappers"));
44 } catch (Exception e) {
45     throw new BuilderException("Error parsing SQL Mapper Configuration.
Cause: " + e, e);
46 }
47 }
```



上面解析流程结束后会生成一个Configuration对象，包含所有配置信息，然后会创建一个SqlSessionFactory对象，这个对象包含了Configuration对象。

### 简单总结

对于MyBatis启动的流程（获取SqlSession的过程）这边简单总结下：

- SqlSessionFactoryBuilder解析配置文件，包括属性配置、别名配置、拦截器配置、环境（数据源和事务管理器）、Mapper配置等；解析完这些配置后会生成一个Configuration对象，这个对象中包含了MyBatis需要的所

有配置，然后用这个Configuration对象创建一个SqlSessionFactory对象，这个对象中包含了Configuration对象；

这里解析的东西比较多，大致概况：会把所有的信息都解析到Configuration对象中，比较简单不多介绍。简单介绍一下：

**文档：MyBatis的二级缓存原理分析（请暂时关注解析缓存内容）**

链接：<http://note.youdao.com/noteshare?>

[id=b97d399a65e31008fef704164d24c784&sub=wcp1596698664968348](http://note.youdao.com/noteshare?id=b97d399a65e31008fef704164d24c784&sub=wcp1596698664968348)

**文档：Mybatis解析动态sql原理分析**

链接：<http://note.youdao.com/noteshare?>

[id=8b076e28061434f6693d99401d3ed400&sub=D948A34AA06045AEB7A01733A98A4C06](http://note.youdao.com/noteshare?id=8b076e28061434f6693d99401d3ed400&sub=D948A34AA06045AEB7A01733A98A4C06)

**文档：Mybatis-设计模式总结.note**

链接：<http://note.youdao.com/noteshare?>

[id=4975cd9e83f1e73e14a369598a232abe&sub=5D52C27921074712B1AB91C9A72455C0](http://note.youdao.com/noteshare?id=4975cd9e83f1e73e14a369598a232abe&sub=5D52C27921074712B1AB91C9A72455C0)