

子月生

## Mybatis详解系列(五)--Mybatis Generator和全注解风格的MyBatis3DynamicSql

posted @ 2020-05-03 12:55 子月生 阅读(9062) 评论(10) 收藏

### 简介

Mybatis Generator (MBG) 是 Mybatis 官方提供的代码生成器，通过它可以在项目中自动生成简单的 CRUD 方法，甚至“无所不能”的高级条件查询（**MyBatis3DynamicSql**，有了它根本不需要 Mybatis Plus），让我们避免了进行数据库交互时需要手动创建对象和配置 Mybatis 映射等基础工作。

另外，MBG 有很好地扩展性，它提供了大量的接口和插件用来给我们自定义生成的代码应该是什么样子，例如我们可以自定义注释、代码格式化、添加 toString 方法等。本文将讲解如何使用这些接口。

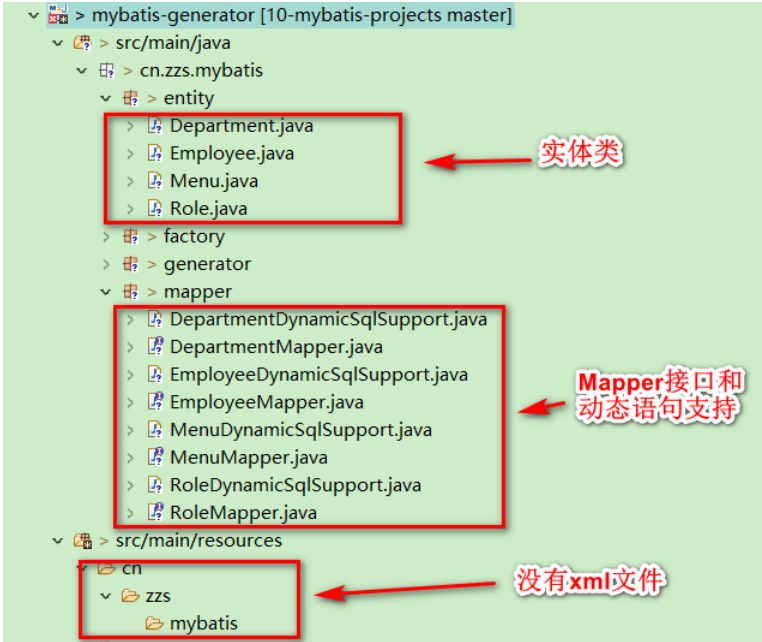
本文内容大致如下，由于篇幅较长，可选择阅读：

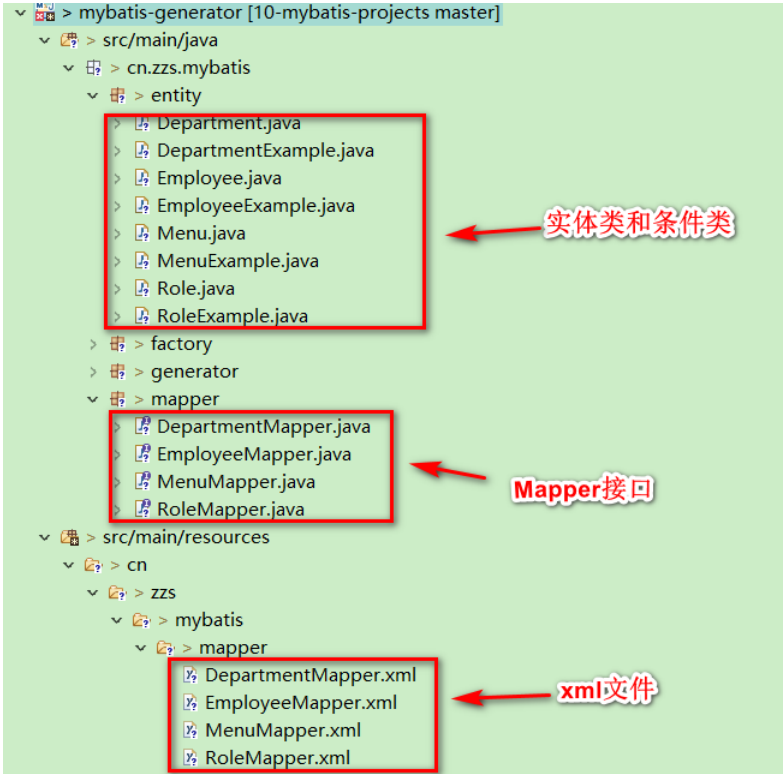
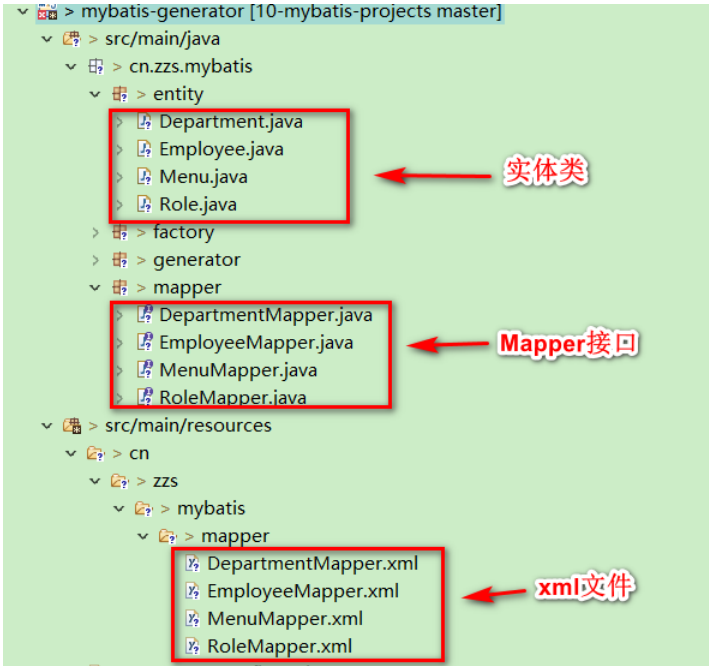
1. 如何使用 MBG 生成代码；
2. 详解 MBG 的配置，将配置使用自定义注释生成器、实体类中添加 toString/equals/hashCode 方法等。
3. MyBatis3DynamicSql 风格(无 XML) API 的使用。

通过本文的学习，你将能够通过简单改造 MBG 来生成自己想要的代码，另外，我们也将认识强大的 MyBatis3DynamicSql 风格（它提供的条件类使用 Lambda 解耦，全注解，支持单表查询、多表查询、分页、排序、分组等等）。

### 关于 MBG 生成代码的风格

MBG 支持生成不同风格、不同语言的代码，例如，MBG 能够生成 Java 或 Kotlin 代码。另外，MBG 支持生成旧版的 MyBatis3 风格（**我们常用的 xml 配置属于 MyBatis3 风格，官方认为这种风格已经过时**），也支持新版的 MyBatis3DynamicSql 的风格（**MyBatis3DynamicSql 风格为官方推荐**）。几种风格的对比如下：

代码风格	描述
MyBatis3DynamicSql	<p>默认风格，官方推荐</p> <p>Java代码</p> <p>全注解，不生成 XML 文件</p> <p>生成的高级条件查询灵活性较大，使用 lambda 表达式避免条件对象渗透到上一层</p> <p>一个表生成一个实体类</p>  <p>实体类</p> <p>Mapper接口和动态语句支持</p> <p>没有xml文件</p>
MyBatis3Kotlin	Kotlin 代码，本文不涉及
MyBatis3	<p>早期风格</p> <p>Java代码</p> <p>能够生成 MyBatis3 兼容的 xml 或 全注解</p> <p>生成的高级条件查询灵活性较小，条件类渗透到上一层，而且 sql 和代码耦合度较高</p>

代码风格	描述
	<div><p>一个表除了生成基本类，可能还会生成主键类和BLOB类（如果指定的话）</p></div>
MyBatis3Simple	<div><p>MyBatis3 的简易版</p><p>Java代码</p><p>能够生成 MyBatis3 兼容的 xml 或 全注解</p><p>不生成 "by example" 或 "selective" 的方法</p><p>一个表生成一个实体类</p></div>

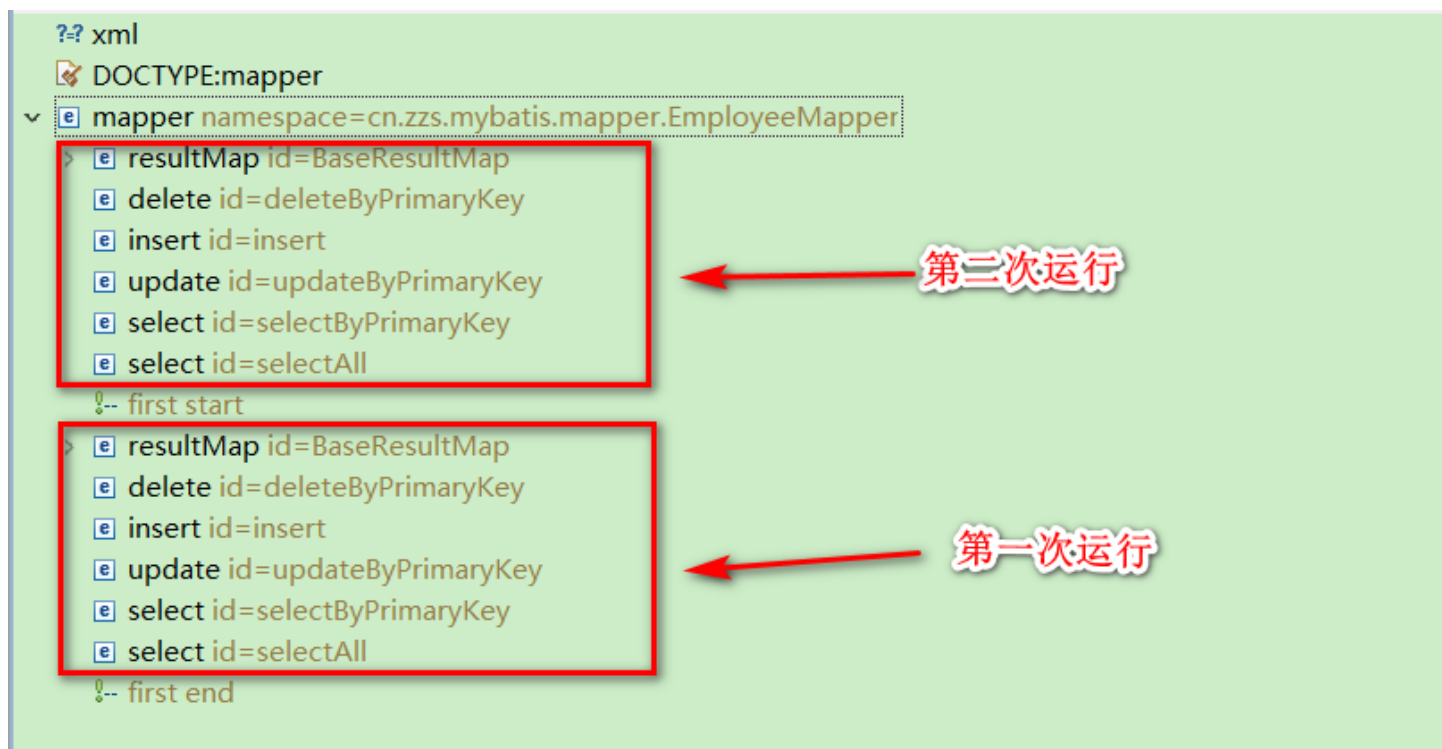
由于 MyBatis3 风格生成的 Example 类存在的问题，实际项目中建议使用 **MyBatis3Simple** 风格或官方推荐的 **MyBatis3DynamicSql** 风格。

# 关于 MBG 文件覆盖的问题

当我们在迭代开发环境中使用 MBG，需要注意文件覆盖的问题，默认情况下，文件覆盖规则如下：

1. 如果 XML 已经存在，MBG 会采用文件合并的方式。

它不会修改你自定义的节点，但是会更新原来生成的 CRUD 节点（如果表发生变化）。**文件合并有个前提，就是原来生成的 CRUD 节点必须包含 @mbg.generated 的默认注释。**否则，当再次运行 MBG 时，它将无法识别哪些是它生成过的节点，于是会出现下图的情况，即 CRUD 节点被重复插入。



2. 如果 Java 或 Kotlin 文件已经存在，MBG 可以覆盖现有文件或使用其他唯一名称保存新生成的文件，这取决于你如何配置 `<overwrite>>false</overwrite>`。

那么，下面开始详细介绍如何使用 MBG。

## 项目环境的说明

### 工程环境

JDK: 1.8.0\_231 (要求 JDK8 及以上)

maven: 3.6.1

IDE: Spring Tool Suites4 for Eclipse 4.12

mysql: 5.7.28

### 数据库脚本

具体的 sql 脚本也提供好了（脚本路径）。

# maven配置

## pom.xml配置

MBG 支持使用**Java 代码**、**maven 插件**、**Eclipse 插件**等方式运行，本文使用 maven 插件方式，所以需要 在 build/plugins 节点引入 MBG 插件，并加入其它依赖项，例如 mybatis、mysql 驱动等。另外，因为本文也需要用到 Java 程序来运行 MBG（当使用自定义类时），所以把插件的依赖项从 build/plugins/plugin 节点中单独取出来。

```
<dependencies>
    <!-- mybatis-generator -->
    <dependency>
        <groupId>org.mybatis.generator</groupId>
        <artifactId>mybatis-generator-core</artifactId>
        <version>1.4.0</version>
    </dependency>
    <!-- mybatis -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.7</version>
    </dependency>
    <!-- dynamic-sql 用于测试MyBatis3DynamicSql生成的代码-->
    <dependency>
        <groupId>org.mybatis.dynamic-sql</groupId>
        <artifactId>mybatis-dynamic-sql</artifactId>
        <version>1.3.0</version>
    </dependency>
    <!-- jdbc驱动 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.15</version>
    </dependency>
    <!-- logback -->
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-core</artifactId>
        <version>1.2.3</version>
        <type>jar</type>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.2.3</version>
        <type>jar</type>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.mybatis.generator</groupId>
            <artifactId>mybatis-generator-maven-plugin</artifactId>
            <version>1.4.0</version>
            <executions>
```

```
        <execution>
            <id>Generate MyBatis Artifacts</id>
            <phase>package</phase>
            <goals>
                <goal>generate</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <verbose>true</verbose>
        <contexts>default</contexts>
        <overwrite>false</overwrite>

    <configurationFile>${basedir}/src/main/resources/generatorConfig.xml</configurationFile>
        <includeCompileDependencies>true</includeCompileDependencies>
    </configuration>
</plugin>
</plugins>
</build>
```

插件参数详解

plugin/configuration 节点可以配置影响 MBG 行为的参数，如下：

参 数	表达式
co nfi gu rat io nF ile	<i>mybatis.generator.configurationFile java.io. File </i> 配置文件路径。 < br/ > 默认为 {basedir}/src/main/resources/generatorConfig.xml
co nt ex ts	\${mybatis.generator.contexts}

参 数	表达式
jd bc Dri ve r	<code>\${mybatis.generator.jdbcDriver}</code>
jd bc Pa ss w or d	<code>\${mybatis.generator.jdbcPassword}</code>
jd bc U RL	<code>\${mybatis.generator.jdbcURL}</code>
jd bc Us erI d	<code>\${mybatis.generator.jdbcUserId}</code>

参 数	表达式
ou tp ut Dir ec tor y	<i>mybatis.generator.outputDirectory</i>   <i>java.io.File</i>   <i>MBG</i> 文件输出路径。只有在配置文 小写 ) , 才会使用这个路径。 < br/ > 默认为 {project.build.directory}/generated-sources/mybatis-generator
ov er wri te	\${mybatis.generator.overwrite}



参 数	表达式

参 数	表达式
sq l s c r i p t	<code>\${mybatis.generator.sqlScript}</code>

参 数	表达式
ta bl eN a m es	<code>\${mybatis.generator.tableNames}</code>
ve rb os e	<code>\${mybatis.generator.verbose}</code>

参 数	表达式
in cl ud eC o m pil eD ep en de nc ies	<code>\${mybatis.generator.includeCompileDependencies}</code>

参 数	表达式
in cl ud eA llD ep en de nc ies	<code>\${mybatis.generator.includeAllDependencies}</code>

参 数	表达式
ski p	<code>\${mybatis.generator.skip}</code>

## 代码生成规则配置

使用 maven 插件的方式不需要编写代码，只要将规则配置到 generatorConfig.xml 文件就行，配置内容主要为：

- 1. 如何连接到数据库
- 2. 生成什么对象，以及如何生成它们
- 3. 哪些表将用于对象生成

下面先给一个简单版的，后面再具体讲解这些参数的意义：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration
    PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
    "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>
    <!--导入配置 -->
```

```
<properties resource="jdbc.properties"></properties>

<!-- context 一般是一个数据源一个context -->
<context id="default" targetRuntime="MyBatis3Simple" defaultModelType="flat">
    <!-- 插件 -->
    <plugin type="org.mybatis.generator.plugins.EqualsHashCodePlugin">
        <property name="useEqualsHashCodeFromRoot" value="true"/>
    </plugin>
    <plugin type="org.mybatis.generator.plugins.ToStringPlugin">
        <property name="useToStringFromRoot" value="true"/>
    </plugin>

    <!-- 注释 -->
    <commentGenerator type="cn.zzs.mybatis.generator.MyCommentGenerator">
        <property name="addRemarkComments" value="true"/>
        <property name="dateFormat" value="yyyy-MM-dd HH:mm:ss"/>
    </commentGenerator>

    <!--jdbc的数据库连接 -->
    <jdbcConnection
        driverClass="${jdbc.driver}"
        connectionURL="${jdbc.url}"
        userId="${jdbc.username}"
        password="${jdbc.password}">
    </jdbcConnection>

    <!-- 类型解析器 -->
    <javaTypeResolver>
        <property name="forceBigDecimals" value="true"/>
    </javaTypeResolver>

    <!-- 实体类 -->
    <javaModelGenerator
        targetPackage="cn.zzs.mybatis.entity"
        targetProject=".\\src\\main\\java">
        <!-- <property name="rootClass" value="cn.zzs.mybatis.entity.EntityClass"/> -->
        <property name="trimStrings" value="true"/>
    </javaModelGenerator>

    <!-- xml -->
    <sqlMapGenerator
        targetPackage="cn.zzs.mybatis.mapper"
        targetProject=".\\src\\main\\resources">
    </sqlMapGenerator>

    <!-- Mapper接口 -->
    <javaClientGenerator type="XMLMAPPER"
        targetPackage="cn.zzs.mybatis.mapper"
        targetProject=".\\src\\main\\java">
        <!-- <property name="rootInterface" value="cn.zzs.mybatis.entity.BaseMapper"/> -->
    </javaClientGenerator>

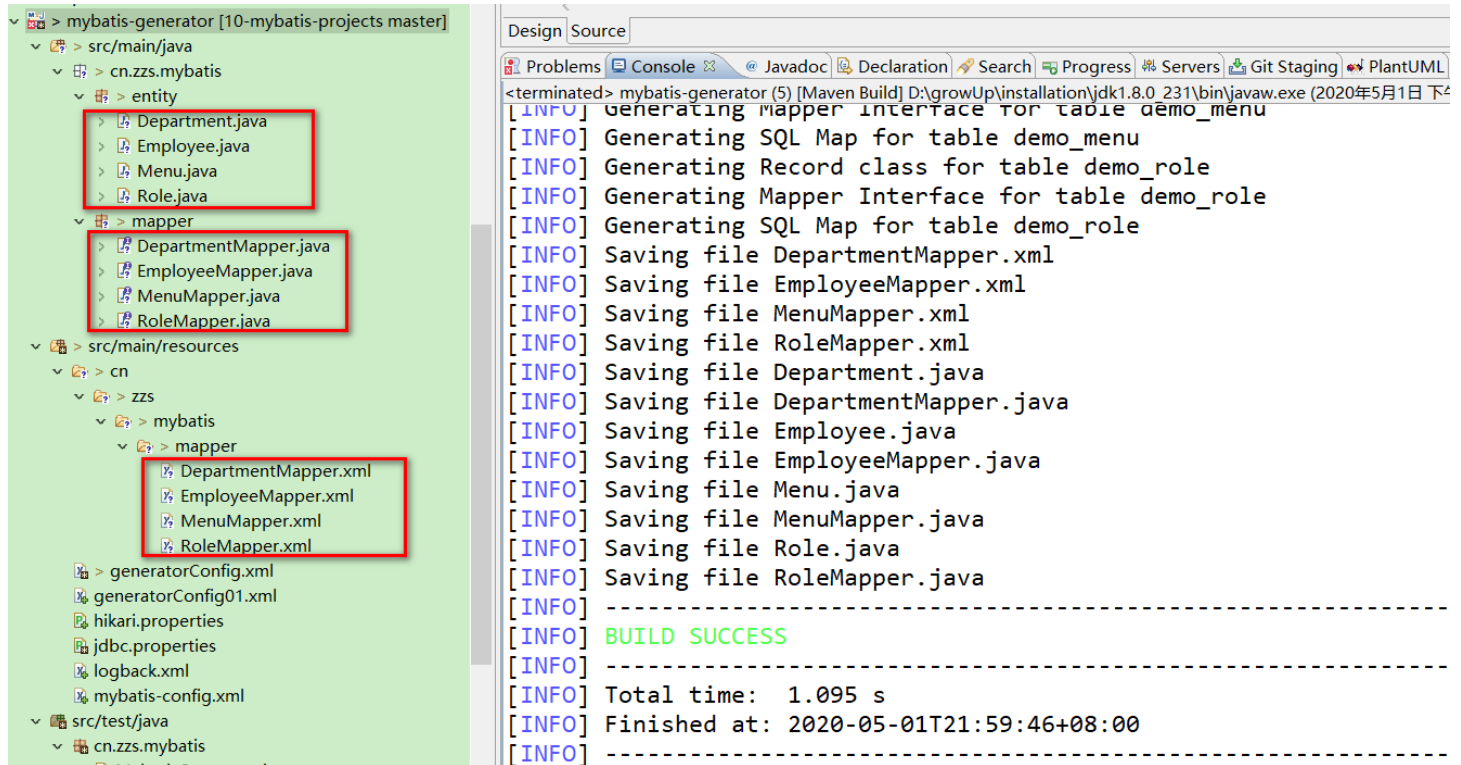
    <!-- 指定数据库表 -->
    <table tableName="demo_%" >
        <domainObjectRenamingRule searchString="^Demo" replaceString=""/>
    </table>
```

```
</context>
</generatorConfiguration>
```

## 命令执行

### maven 插件的运行方式

maven build, 输入 `mybatis-generator:generate` , 生成成功。



### Java程序的运行方式

通常情况下, 我们都是使用 maven 插件的方式, 但是, 当我们在 MBG 中指定了自定义的实现, 使用 maven 插件可能会报错, 这个时候就需要通过 Java 程序的方式运行 MBG, 具体方法如下:

```
public static void main(String[] args) throws Exception {

    // LoggerFactory.forceSlf4jLogging();
    // System.setProperty("user.name", "zzs");
    // 这个集合记录着生成、合并、覆盖文件的信息
    List<String> warnings = new ArrayList<String>();
    InputStream in =
MybatisGenerator.class.getClassLoader().getResourceAsStream("generatorConfig.xml");
    ConfigurationParser cp = new ConfigurationParser(warnings);
    Configuration config = cp.parseConfiguration(in);

    // 不覆盖 Java 文件
    boolean overwrite = false;
    DefaultShellCallback callback = new DefaultShellCallback(overwrite);
    MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config, callback, warnings);

    // 生成文件
    myBatisGenerator.generate(null);
    // 打印信息
```



```
warnings.forEach(System.err::println);  
}
```

# 生成规则详解

generatorConfig.xml 的顶层结构如下：

- generatorConfiguration（配置）
  - classPathEntry（JDBC驱动路径）
  - properties（properties文件路径）
  - context（生成对象的环境）
    - property（Context作用域参数）
    - jdbcConnection（JDBC连接）
    - connectionFactory（JDBC连接工厂）
    - commentGenerator（注释生成器）
    - javaModelGenerator（实体对象生成器）
    - javaClientGenerator（Mapper 接口或实现类生成器）
    - sqlMapGenerator（xml 生成器）
    - table（用于生成对象的表）
    - javaTypeResolver（Java 类型处理器）
    - plugin（插件）

下面选择部分节点展开分析：

## context\*

`<context>` 节点用于指定生成一系列对象的环境。我们可以在配置文件中配置多个 `<context>` 节点来实现从**不同数据源**或采用不同生成参数生成对象。

### 属性

这里最重要的属性是**targetRuntime**，它直接决定该环境下生成的代码风格，常用的风格为MyBatis3DynamicSql 和 MyBatis3Simple。

属性	描述
id	用于唯一标识指定环境。必选属性
defaultModelType	<p>指定实体对象的类型。包括三种：</p> <p><b>flat</b>：一个表生成一个实体类；</p> <p><b>conditional</b>：和 hierarchical 差不多，区别在于如果表只有一个主键，不会单独去生成一个主键类；</p> <p><b>hierarchical</b>：除了生成基本类，如果表中包含主键或 BLOB 列，都会单独再生成一个主键类或BLOB类</p> <p><b>默认为 conditional</b>，如果 targetRuntime 为 "MyBatis3Simple"、"MyBatis3DynamicSql"或"MyBatis3Kotlin"，则该属性忽略</p>

属性	描述
targetRuntime	用于指定生成代码的风格。默认为 MyBatis3DynamicSql
introspectedColumnImpl	指定org.mybatis.generator.api.IntrospectedColumn实现类。该类可以看成是某一系列的所有信息

子节点

<context> 包含以下子节点：

- <property> (0..N)
- <plugin> (0..N)
- <commentGenerator> (0 or 1)
- <connectionFactory>/<jdbcConnection> (1 Required)
- <javaTypeResolver> (0 or 1)
- <javaModelGenerator> (1 Required)
- <sqlMapGenerator> (0 or 1)
- <javaClientGenerator> (0 or 1)
- <table> (1..N)

其中， <property> 支持以下参数。其中，针对 mysql 数据库，可以将定界符修改为反单引号。如果想要使用自定义的代码格式或 XML 格式，可以配置自定义实现。

参数名	Property Values
autoDelimitKeywords	如果数据库关键字被作为列名使用，MBG 是否需要对其进行定界。默认为false
beginningDelimiter	指定定界符的开头，默认为"
endingDelimiter	指定定界符的结尾，默认为"
javaFileEncoding	指定处理 Java 文件使用的编码
javaFormatter	指定生成 Java 代码使用的格式化器。 如果自定义的话需要实现org.mybatis.generator.api.JavaFormatter 接口并提供无参构造，默认为org.mybatis.generator.api.dom.DefaultJavaFormatter
targetJava8	指定生成 Java 代码使用的 JDK 版本
kotlinFileE	不涉及

参数名	Property Values
ncoding	
kotlinForm atter	不涉及
xmlFormat ter	指定生成 XML 文件使用的格式化器。如果自定义的话需要实现 org.mybatis.generator.api.XmlFormatter 接口并提供无参构造，默认为 org.mybatis.generator.api.dom.DefaultXmlFormatter

jdbcConnection

<jdbcConnection> 节点用于指定 MBG 进行数据库交互所用的 JDBC 连接。注意， <jdbcConnection> 和 <jdbcConnection> 节点只要一个就行了。

属性

属性	描述
driverClass	JDBC 驱动。必选属性
connectionURL	JDBC URL。必选属性
userId	用户名
password	密码

子节点

<property> (0..N)：用于定义 JDBC 驱动所需的一些参数，较少用到。

connectionFactory

<jdbcConnection> 节点用于指定和配置 MBG 进行数据库交互时获取 JDBC 连接的工厂。注意， <jdbcConnection> 和 <jdbcConnection> 节点只要一个就行了。

属性

属性	描述
typ e	用于指定 连接工厂的类。必选属性 如果自定义的话需要实现org.mybatis.generator.api.ConnectionFactory 接口并提供无参构造，默认为org.mybatis.generator.internal.JDBCConnectionFactory

子节点

<jdbcConnection> 的子节点为 <property> (0..N)，参数如下表。如果是使用默认的连接工厂，这几个参数必须按照下表的提供，但是，如果是自定义的连接工厂，需要什么参数由你自己决定。

参数	描述
driverClass	JDBC 驱动
connectionURL	JDBC URL
userId	用户名
password	密码

## 如何使用HikariCP作为 MBG 的连接工厂

这里我简单的实现了一个连接工厂：

```
public class HikariConnectionFactory implements ConnectionFactory {

    private DataSource dataSource;

    public HikariConnectionFactory() {
        super();
        HikariConfig config = new HikariConfig("/hikari.properties");
        dataSource = new HikariDataSource(config);
    }

    @Override
    public Connection getConnection() throws SQLException {
        return dataSource.getConnection();
    }

    @Override
    public void addConfigurationProperties(Properties properties) {
        // do nothing
    }
}
```

将它配置到 XML 文件中，这时我们不需要添加任何的 <property> 子节点。

```
<connectionFactory type="cn.zzs.mybatis.factory.HikariConnectionFactory"/>
```

使用 Java 程序运行 MBG，可以看到代码可以正常生成。

## commentGenerator\*

<commentGenerator> 节点用于指定和配置 Java 代码或 XML 文件中使用的注释生成器。**默认的注释生成器比较鸡肋，一般我们都会考虑自己实现。**

属性

属性	描述
type	用于指定注释生成器的类。 如果自定义的话需要实现org.mybatis.generator.api.CommentGenerator接口并提供无参构造，默认为org.mybatis.generator.internal.DefaultCommentGenerator

子节点

<commentGenerator> 的子节点为 <property> (0..N)，参数如下表。注意，这些参数是针对默认注释生成器的，如果是自定义的，需要什么参数由你自己决定。

Property Name	Property Values
suppressAllComments	是否不生成任何注解。 默认为false。注意，前面提到过，MBG 通过注释中的@mbg.generated来判断某个元素是不是通过 MBG 生成，如果不生成注解的话，XML 的文件覆盖功能会受到影响。
suppressDate	是否在注释中不包含时间。 默认为false。
addRemarkComments	是否在注释中包含列的注释 默认为false
dateFormat	指定时间格式 例如：yyyy-MM-dd HH:mm:ss

自定义注释生成器

想要自定义注释生成器需要实现 org.mybatis.generator.api.CommentGenerator 接口并提供无参构造，在本项目中，我在 DefaultCommentGenerator 的基础上改造了一个注释生成器，感兴趣的可以移步到项目源码。

注意，编写自定义注解生成器时应该考虑在xml节点的注释中加入@mbg.generated来维持 MBG 文件合并的功能。

下面简单举个例子，MBG 生成的实体类属性的注解是这样的：

```
/**
 * Database Column Remarks:
 *   员工id
 *
 * This field was generated by MyBatis Generator.
 * This field corresponds to the database column demo_employee.id
 *
 * @mbg.generated Sat May 02 12:52:28 CST 2020
```

```
 */
private String id;
```

但是，我不想要这种注释，这时，我们可以改造以下方法：

```
@Override
public void addFieldComment(Field field, IntrospectedTable introspectedTable,
    IntrospectedColumn introspectedColumn) {
    if (suppressAllComments) {
        return;
    }

    field.addJavaDocLine("/**");

    // 获取列注释并加入到注解中
    String remarks = introspectedColumn.getRemarks();
    if (addRemarkComments && StringUtility.stringHasValue(remarks)) {
        String[] remarkLines = remarks.split(System.getProperty("line.separator"));
        for (String remarkLine : remarkLines) {
            field.addJavaDocLine(" * <p>" + remarkLine + "</p>");
        }
    }

    field.addJavaDocLine(" */");
}
```

并且在 XML 中进行如下配置：

```
<!-- 注释 -->
<commentGenerator type="cn.zzs.mybatis.generator.MyCommentGenerator">
    <property name="addRemarkComments" value="true"/>
</commentGenerator>
```

使用 Java 程序运行 MBG，可以看到实体类属性的注释变成我们想要的样子。

```
/**
 * <p>员工id</p>
 */
private String id;
```

## javaTypeResolver

`<javaTypeResolver>` 节点用于指定和配置 Java 类型解析器。默认的解析器可能会将数据库类型 decimal 或 numeric 解析为 `Short`、`Integer`、`Long` 等 Java 类型，如果我们不希望这样解析，就需要使用到这个节点。

Java 类型解析器使用默认的进行，一般不会去重写它。

### 属性

Attribute	Description
type	用于指定Java 类型解析器的类。 如果自定义的话需要实现org.mybatis.generator.api.JavaTypeResolver并提供无参构造，默认为org.mybatis.generator.internal.types.JavaTypeResolverDefaultImpl，它会将数据库类型 decimal 或 numeric 解析为 Integer 的 Java 类型

子节点

<javaTypeResolver> 的子节点为 <property> (0..N)，参数如下表。注意，这些参数是针对默认注释生成器的，如果是自定义的，需要提供什么参数由你自己决定。

Property Name	Property Values
forceBigDecimals	是否强制将数据库类型 decimal 或 numeric 解析为 BigDecimal 类型。 默认为false，会根据数据的小数点位数和长度来决定使用 Short、Integer、Long 或 BigDecimal。
useJSR310Types	是否不强制将数据库类型 date, time 和 timestamp 解析为 Date 默认为false，如果为true，解析规则将变成：date -> LocalDate, time -> LocalTime, timestamp -> LocalDateTime

javaModelGenerator

<javaModelGenerator> 节点用于配置实体类生成器。实体类生成器不支持自定义。

属性

Attribute	Description
targetPackage	指定存放生成类的包路径。必选属性
targetProject	指定 targetPackage 的源文件夹。必选属性 注意，如果该文件夹不存在，将会报错

子节点

<javaModelGenerator> 的子节点为 <property> (0..N)，参数如下表。用的比较多的是 rootClass 和 trimStrings。

Property Name	Property Values
constructorBased	是否生成包含全部参数的构造方法。 默认为false。当为true时，除了生成指定构造，还会生成对应的 resultMap。

Property Name	Property Values
enableSubPackages	是否在targetPackage基础上生成子包。 默认为false。当为true时，会将表所在 schema 名作为子包名
exampleTargetPackage	指定 Example 条件类的生成路径 默认和 targetPackage 相同
exampleTargetProject	指定 exampleTargetPackage 的源文件夹 默认和 targetProject 相同
immutable	如果为 true，生成的实体类将不包含 setter 方法，但会提供包含所有参数的构造。 默认为 false
rootClass	指定实体类需要继承的父类。
trimStrings	在setter方法中是否对传入字符串进行 trim 操作 默认为 false。

javaClientGenerator

<javaClientGenerator> 节点用于指定和配置客户端类生成器。这个节点是可选的，如果不指定，则不会生成客户端类。客户端类一般指的是 Mapper 接口及 Mapper 接口的一些辅助类，例如 SqlProvider。

属性

下面的 type 属性仅针对 MyBatis3Simple 和 MyBatis3 风格生效，而且 MyBatis3Simple 风格不支持 MIXEDMAPPER。

属性	描述
type	用于指定使用客户端类生成器的类。 如果自定义的话需要实现 org.mybatis.generator.codegen.AbstractJavaClientGenerator并提供无参构造，MBG 为我们提供了以下几种： 1. <b>ANNOTATEDMAPPER</b> ：包含 Mapper 接口和 SqlProvider 辅助类，全注解，不包含 XML 文件； 2. <b>XMLMAPPER</b> ：包含 Mapper 接口和 XML 文件，不包含注解； 3. <b>MIXEDMAPPER</b> ：包含 Mapper 接口和 XML 文件，简单的 CRUD 使用注解，高级条件查询使用 XML 文件。
targetPackage	指定存放生成类的包路径。必选属性



属性	描述
targetProject	指定存放生成类的包路径。必选属性

子节点

`<javaClientGenerator>` 的子节点为 `<property>` (0..N)，参数如下表。注意，这些参数是针对默认注释生成器的，如果是自定义的，需要什么参数由你自己决定。

Property Name	Property Values
enableSubPackages	是否在 targetPackage 基础上生成子包。 默认为 false。当为 true 时，会将表所在 schema 名作为子包名
rootInterface	指定 Mapper 接口需要实现的父接口。
useLegacyBuilder	是否使用过时的SqlBuilder来构造动态语句。 默认为 false

sqlMapGenerator

`<sqlMapGenerator>` 节点用于配置 XML 生成器，不支持自定义。

属性

Attribute	Description
targetPackage	指定存放生成 XML 的包路径。必选属性
targetProject	指定存放生成 XML 的包路径。必选属性

子节点

`<sqlMapGenerator>` 的子节点为 `<property>` (0..N)，参数如下表。

Property Name	Property Values
enableSubPackages	是否在targetPackage基础上生成子包。 默认为false。当为true时，会将表所在 schema 名作为子包名

table\*

`<table>` 节点用于指定需要用于生成对象的表以及配置某些生成规则。这个节点相比前面提到的，要更加复杂一些。

属性

`<table>` 节点支持的属性很多，一般保持默认就可以了。

Attribute	Description
tableName	需要用于生成对象的表名。必选属性 允许使用 SQL 通配符，例如：demo_ %
schema	指定数据库 schema 允许使用 SQL 通配符
catalog	指定数据库 catalog
alias	指定查询时字段别名前缀。 如果指定，生成的 select 语句将使用alias_actualColumnName的别名
domainObjectName	指定实体类的类名。 默认情况下使用驼峰命名规则
mapperName	指定 Mapper 接口名 默认为实体类名+Mapper
sqlProviderName	指定 SqlProvider 接口名 默认为实体类名+SqlProvider
enableInsert enableSelectByPrimaryK ey enableSelectByExample enableUpdateByPrimary Key enableDeleteByPrimaryK ey enableDeleteByExample enableCountByExample enableUpdateByExamp le	是否生成指定语句 默认为true
selectByPrimaryKeyQuer yId selectByExampleQueryI d	如果指定，在select 语句中将添加 'value' as QUERYID
modelType	指定实体对象的类型。context节点中已介绍过了
escapeWildcards	当 schema 或 tableName 包含 SQL 通配符时，在搜索列时是否对其进行转义

Attribute	Description
delimitIdentifiers	是否在 SQL 中对表名使用定界符并且使用确定的表名大小写 默认为false
delimitAllColumns	是否在 SQL 中对所有列名都添加定界符 默认为false

子节点

<table> 的子节点如下：

```
<property> (0..N)
<generatedKey> (0 or 1)
<domainObjectRenamingRule> (0 or 1)
<columnRenamingRule> (0 or 1)
<columnOverride> (0..N)
<ignoreColumn> (0..N)
<ignoreColumnsByRegex> (0..N)
```

这几个子节点中，常用到的是 <domainObjectRenamingRule> ，其他的用的比较少。下面挑几个来分析下：

property

<property> 配置的大部分参数都是为了覆盖全局的配置，并不常用。

Property Name	Property Values
constructorBased	是否生成包含全部参数的构造方法。 默认为false。当为true时，除了生成指定构造，还会生成对应的 resultMap。
ignoreQualifiersAtRuntime	在生成的 SQL 中，表名前是否不添加 schema 或 catalog 默认为 false
immutable	如果为 true，生成的实体类将不包含 setter 方法，但会提供包含所有参数的构造。 默认为 false
modelOnly	是否只生成实体类 默认为 false
rootClass	指定实体类需要继承的父类。
rootInterface	指定 Mapper 接口需要实现的父接口。
runtimeCatalog	指定 SQL 中使用的 catalog

Property Name	Property Values
runtimeSchema	指定 SQL 中使用的 schema
runtimeTableName	指定 SQL 中使用的 tableName
selectAllOrderByClause	指定selectAll方法中加入order by 'value'
trimStrings	在实体类的 setter 方法中是否对传入字符串进行 trim 操作 默认为 false。
useActualColumnNames	是否直接使用表名作为实体类类名。 默认为false
useColumnIndexes	是否在 resultMap 中使用索引而不使用列名进行映射
useCompoundPropertyNames	是否将“列名+列注释”作为实体类的属性名

domainObjectRenamingRule/columnRenamingRule

<domainObjectRenamingRule> 节点一般用于重命名实体类类名。

例如，在没有指定 domainObjectName 的情况下，demo\_employee 的表将生成实体类 DemoEmployee，但我希望去掉前面的 Demo 前缀，则可以这样处理：

```
<table tableName="demo_%" >
  <!-- replaceString属性可以省略 -->
  <domainObjectRenamingRule searchString="^Demo" replaceString="" />
</table>
```

另一个子节点 <columnRenamingRule> 也是相同的用法。

```
<table tableName="demo_%" >
  <columnRenamingRule searchString="^Employee_" replaceString="" />
</table>
```

ignoreColumnsByRegex/ignoreColumn

<ignoreColumnsByRegex> 和 <ignoreColumn> 节点用于告诉 MBG 生成代码时忽略某些列。使用方法如下。

```
<table tableName="Foo">
  <ignoreColumnsByRegex pattern="(?!i) col.*">
    <except column="col01"/>
    <except column="col13"/>
  </ignoreColumnsByRegex>
</table>
```

plugin\*

<plugin> 节点用于定义和配置插件。这个节点只有一个 type 属性，用于指定使用哪个插件，并通过子节点 property 来为这个插件设置参数。

### 官方提供的插件

MBG 为我们提供了许多好用的插件，如下：

插件	描述
org.mybatis.generator.plugins.SerializablePlugin	用于在实体类中实现java.io.Serializable接口
<b>org.mybatis.generator.plugins.ToStringPlugin</b>	用于在实体类中添加 toString 方法
<b>org.mybatis.generator.plugins.EqualsAndHashCodePlugin</b>	用于在实体类中添加 equals 和 hashCode 方法。
org.mybatis.generator.plugins.UnmergeableXmlMappersPlugin	用于指定不合并 XML 文件，这时 MBG 将 采用处理 Java 文件的方式来处理 XML 文件
org.mybatis.generator.plugins.CachePlugin	用于在 xml 文件中加入cache节点
org.mybatis.generator.plugins.CaseInsensitiveLikePlugin	用于在 Example 类中生成不区分大小写的Like
org.mybatis.generator.plugins.dsql.DisableDeletePlugin	禁用 MyBatisDynamicSQLV2 风格的所有删除方法
org.mybatis.generator.plugins.dsql.DisableInsertPlugin	禁用 MyBatisDynamicSQLV2 风格的所有插入方法
org.mybatis.generator.plugins.dsql.DisableUpdatePlugin	禁用 MyBatisDynamicSQLV2 风格的所有更新方法
org.mybatis.generator.plugins.FluentBuilderMethodsPlugin	用于在实体类中添加MyDomainClass withValue(Object v)方法。通过它可以实现如下赋值方式： new Employee().withAddress("北京").withDeleted(false).withName("zzs001");
<b>org.mybatis.generator.plugins.MapperAnnotationPlugin</b>	用于在 Mapper 接口中添加@Mapper接口
org.mybatis.generator.plugins.MapperConfigPlugin	用于生成 Mybatis 的主配置文件 MapperConfig

插件	描述
org.mybatis.generator.plugins.RenameExampleClassPlugin	用于重命名 Example 类
org.mybatis.generator.plugins.RowBoundsPlugin	用于在 Mapper 接口的 selectByExample 方法参数中加入 RowBounds 参数，用于支持分页
org.mybatis.generator.plugins.VirtualPrimaryKeyPlugin	用于指定表的主键

插件的配置方式非常简单，如下：

```
<!-- 插件 -->
<plugin type="org.mybatis.generator.plugins.EqualsHashCodePlugin">
  <property name="useEqualsHashCodeFromRoot" value="true"/>
</plugin>
<plugin type="org.mybatis.generator.plugins.ToStringPlugin">
  <property name="useToStringFromRoot" value="true"/>
</plugin>
```

运行 Java 程序，可以看到实体类中生成类 toString 、 hashCode 和 equals 方法。

自定义插件

如果自定义的话，需要实现 org.mybatis.generator.api.Plugin 接口，并提供无参构造，当然，这里推荐继承 org.mybatis.generator.api.PluginAdapter 来避免重写太多方法。下面举个例子来讲解自定义插件的使用。

在上面的例子中，我生成的实体名字是 Menu，如果我想命名为 MenuDomain，而生成的 Mapper 命名为 MenuDAO。为了实现这种需求，我们可以先定义一个插件，如下：

```
public class RenameFilePlugin extends PluginAdapter {

    @Override
    public boolean validate(List<String> warnings) {
        return true;
    }

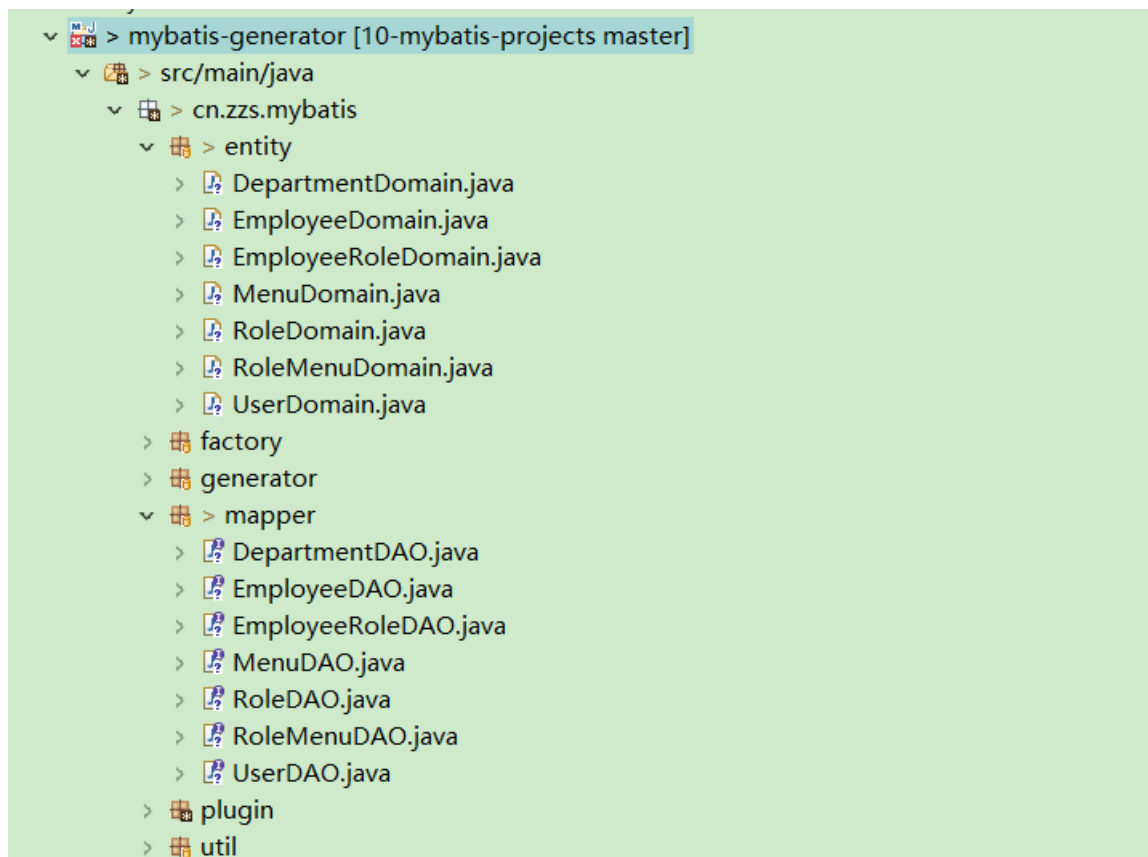
    @Override
    public void initialized(IntrospectedTable introspectedTable) {
        // 更改实体类名称，例如: cn.zzs.mybatis.entity.Menu => cn.zzs.mybatis.entity.MenuDomain
        String oldType = introspectedTable.getBaseRecordType();
        introspectedTable.setBaseRecordType(oldType + "Domain");

        // 更改Mapper名称
        String mapperType = introspectedTable.getMyBatis3JavaMapperType();
        introspectedTable.setMyBatis3JavaMapperType(mapperType.replace("Mapper", "DAO"));
    }
}
```

然后将插件配置：

```
<plugin type="cn.zzs.mybatis.plugin.RenameFilePlugin"></plugin>
```

生成文件如下：



以上，基本讲完 MBG 的使用方法，涉及到的内容可以满足实际使用需求。

## MyBatis3DynamicSql 风格 API 的使用

在研究 MBG 之前，其实我并没有听说过 MyBatis3DynamicSql 风格，因为项目里一直使用的是 Mybatis3Simple，网上也很少人提起。

Mybatis3Simple 可以生成简单的 CRUD，但是针对高级条件查询就无能为力了，实际项目中，我们有时必须手动地去编写高级查询的代码，通常情况下，我们用不同的方案来处理：

1. 自定义代码生成器来生成高级条件查询的代码。针对复杂场景时，我们还是得手动改造。
2. 使用 Mybatis Plus 的条件构造器。在 MP 3.0 之前，条件构造器会造成 sql 和代码的严重耦合。

现在，Mybatis 官方为我们提供了新的方案，那就是 MyBatis3DynamicSql。MyBatis3DynamicSql 风格丢弃了 XML 文件，使用全注解形式并搭配几个条件辅助类，刚接触时，我还是比较抗拒，因为前面 **MyBatis3** 中也尝试过全注解和条件类，当遇到某些复杂场景时，还是需要 **XML**，而且生成器提供的条件类会渗透到服务层。

直到开始使用 MyBatis3DynamicSql，我才发现它的强大。它可以做到：

1. **单表高级查询**。包括 Equal/NotEqual、Like/NotLike、In/NotIn、Between/NotBetween、IsNull/IsNotNull 等等，而且还可以判空设置条件。
2. **联表查询**。你可以像给单表一样给关联表设置条件。
3. **分组、排序、分页**。

4. 只返回你要的字段。
5. 多表结果集映射。
6. 子查询。

下面这个例子，涉及到了关联查询、排序、分页，而 MyBatis3DynamicSql 都能帮我们处理，并且它利用 **Lambda**表达式来解耦条件类。

```
@Test
public void testSelect() {
    // 注意，当查询结果多于1时会报错
    List<Employee> lsit = baseMapper.select(c ->
        c.leftJoin(DepartmentDynamicSqlSupport.department)
            .on(departmentId, new
                EqualsTo(DepartmentDynamicSqlSupport.id))
            .where(name, isLikeWhenPresent("zsz%"), or(name,
                isLikeWhenPresent("zzf%")))
            .and(status, isEqualTo((byte)1))
            .and(address, isIn("北京", "广东"))
            .and(DepartmentDynamicSqlSupport.name,
                isEqualToWhenPresent("质控部"))
            .orderBy(gmtCreate.descending())
            .limit(3)
            .offset(1)
    );
    lsit.forEach(System.err::println);
}
```

我相信，MyBatis3DynamicSql 风格会被更多开发者使用，这里就不长篇大论的讲解如何使用它，因为它的 API 并不难操作。感兴趣的朋友可以移步到项目源码阅读：测试例子

## 参考资料

MyBatis Generator官方文档

相关源码请移步：mybatis--generator

本文为原创文章，转载请附上原文出处链接：

<https://www.cnblogs.com/ZhangZiSheng001/p/12820344.html>

分层，抽象，高内聚，低耦合

分类: 第三方类库或工具，Mybatis

标签: java，mybatis，使用教程

好文要顶

关注我

收藏该文







子月生  
粉丝 - 57 关注 - 11

+加关注

1

0

« 上一篇: Mybatis源码详解系列(四)--你不知道的Mybatis用法和细节  
» 下一篇: Spring源码系列(一)--详细介绍bean组件

[刷新评论](#) [刷新页面](#) [返回顶部](#)

登录后才能查看或发表评论, 立即 [登录](#) 或者 [逛逛](#) [博客园首页](#)