分类号：TP311.5　　单位代码：10335

密　级：无　　学　　号：21451026

浙江大学

硕士学位论文

中文论文题目：基于 MyBatis 的自适应代码生成工具的研究与实现

英文论文题目：Research and implementation of a MyBatis-based tool to generate self-adapting code

申请人姓名：付荣

指导教师：干红华 副教授

合作导师：

专业学位类别：工程硕士

专业学位领域：软件工程

所在学院：软件学院

论文提交日期 2016 年 4 月 16 日

基于MyBatis的自适应代码生成工具的研究与实现

付荣 浙江大学

基于 MyBatis 的自适应代码生成工具的研究与实现

论文作者签名:_____

指导教师签名:_____

论文评阅人 1：_____

评阅人 2：_____

评阅人 3：_____

评阅人 4：_____

评阅人 5：_____

答辩委员会主席：_____

委员 1：_____

委员 2：_____

委员 3：_____

委员 4：_____

委员 5：_____

答辩日期：_____

Research and implementation of a MyBatis-based tool to generate self-adapting code

Author's signature:_____

Supervisor's signature:_____

Thesis reviewer 1：_____

Thesis reviewer 2：_____

Thesis reviewer 3：_____

Thesis reviewer 4：_____

Thesis reviewer 5：_____

Chair：_____
(Committee of oral defence)

Committeeman 1：_____

Committeeman 2：_____

Committeeman 3：_____

Committeeman 4：_____

Committeeman 5：_____

Date of oral defence：_____

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得 **浙江大学** 或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：　　签字日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解 **浙江大学** 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 **浙江大学** 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：　　导师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

摘要

现如今，MyBatis 框架由于它的灵活性，越来越受到人们的追捧，但是开发人员不得不面对 MyBatis 带来的诸多问题。领域特定语言逐渐被人们推广，模型驱动开发也逐渐成熟，代码生成技术也受到人们越来越多的关注。因此，希望能够借助领域特定语言、模型驱动开发、代码生成技术，为基于 MyBatis 框架的开发提供了全新的解决方案，实现大部分持久层代码的自动生成，以此来减少开发人员的工作量，降低编码的出错率，统一编程规则，提高开发效率，节约开发成本，提升程序质量。

本文提出了一个基于 MyBatis 的自适应代码生成工具，其中包括七大功能模块: 外部 DSL、语法树、语义模型、数据模型、解析器、代码模型和代码生成器。主要采用模型驱动的开发方式，多个功能模块之间利用模型的灵活性实现代码的生成。并且采用 JAVA 作为唯一开发语言，可以方便地应用到不同的平台之中。

该工具适用于所有采用 MyBatis 和 Spring 框架来开发的广大 JAVA 开发人员。开发人员仅仅需要编写关于自己业务逻辑的 DSL 脚本,便可以轻松地生成持久层的代码。它可以内省数据库的表，不仅能生成基础的增删改查代码，还能生成分页查询以及复杂的多表级联查询代码。此外该工具还能自适应数据库的快速修改，不仅支持数据库的替换，还支持数据库表字段的修改。因此开发人员可以将更多的精力投入到其他更为重要的工作之中。这仅仅是代码生成之路的一小步，未来可以做的还有更多。

关键词: 领域特定语言，模型驱动开发，MyBatis，代码生成

Abstract

Today, MyBatis is becoming more and more popular because of its flexibility. But developers have to face many problems which MyBatis brings. So we want to provide a new solution for MyBatis-based development with the help of Domain Specific Language, Model-Driven Development and Code Generation Technology. With the new solution, we can generate most persistence layer code of our application automatically, in order to reduce the workload of developers, reduce the probability of error, unify the rules of programming, improve the development efficiency, save development cost and improve the quality of applications.

A MyBatis-based tool to generate self-adapting code is proposed in this paper. This tool includes seven function modules: external DSL, syntax tree, semantic model, data model, parser, code model and code generator. Multiple modules work together, to generate code with the help of flexible model. Besides, this tool uses JAVA as the only programming language, so that can be easily applied to different platforms.

This tool is applicable to most JAVA developers which use MyBatis and Spring framework to develop their application. Developers just need to write some DSL scripts with their own business logic, then they can generate most persistence layer code easily. It can introspect database tables to generate code about simple CRUD operations, paging querying and complex multi-table join querying. Besides, it supports replacement of database and modification of columns. With the help of this tool, developers can pay more attention to other more important work. This tool is just a tiny step for Code Generation, we can do more in the future.

Key Words: Domain Specific Language, Model-Driven Development, MyBatis, Code Generation

目录

图目录

表目录

第 1 章 绪论

1.1 课题背景及研究的目的与意义

MyBatis 框架由于它的灵活性，广泛的应用在实际的开发之中。虽然与 JDBC 相比已经减少了开发人员大量的工作量，但是程序中整个底层数据库的 SQL 语句还是要开发人员自己来编写。对于数据的可靠性、完整性的瓶颈便更多地依赖于程序员对 SQL 的掌握程度。SQL 语句的可读性很低，调试也非常的困难。

通过对 MyBatis 官方的 MBP（MyBatis Generator）的分析，可以发现它采用了 DSL(Domain Specific Language，领域特定语言)作为支撑。对于 MBG 的 XML 配置，正是外部 DSL 的一种解决方案；而对于 Example 类，正是内部 DSL 的一种解决方案。

本文将提出一个为 MyBatis 以及 Spring 框架量身打造的 MyBatis 代码生成工具。开发人员只需要编写适应于自己业务逻辑的 DSL 脚本，便可以完成 MyBaits 相关的代码的全自动生成，包括 Dao 层、Model 层、Mapper 映射文件。不仅支持简单的 CRUD（插入、查询、更新、删除）操作，还支持分页查询、多表级联查询，甚至适应数据库的替换以及表结构的修改。解决了采用 MyBatis 开发所带来的诸多问题，开发人员可以在不知道 MyBatis 的使用细节的情况下，便可以参与项目开发。以此来减少开发人员的工作量，降低编码的出错率，提高开发效率，对于促进 MyBatis 的推广和使用，具有十分重要的意义。

1.2 MyBatis 的现状

MyBatis 是一个支持 SQL 语句、存储过程以及高级映射的优秀的数据持久化框架。MyBatis 主要通过 XML 配置文件或者注解的方式，将应用层中的接口与 POJO(Plain Old Java Object，普通的 Java 对象)映射成数据库中的记录。MyBatis 与其他 ORM 框架（Object Relational Mapping，对象关系映射）不同（比如：Hibernate），它并不会将 Java 对象和数据库中的表直接关联起来，而是把 Java 方法与 SQL 语句关联起来。MyBatis 将 JDBC 的访问接口进行了封装，因此之前调用 JDBC 接口所以带来的大量重复性工作都不需要再做，对于开发人员来说减少了大量代码，提高了开发的效率[1]。

如果采用 MyBatis 进行开发，那么对于数据的可靠性、完整性的瓶颈更多地依赖于程序员对 SQL 的掌握程度[2]。SQL 语句写在 XML 或者注解里，虽然方便

开发人员修改和浏览，但是可读性很低，调试也不方便。不能够像 JDBC 一样，直接在代码里根据业务逻辑进行复杂动态 SQL 的拼接。

虽然与 JDBC 相比已经极大地减少了开发人员的工作量，但是整个底层 SQL 语句还是需要开发人员自己来编写以及维护[3]。和 Hibernate 相比，开发人员的工作量其实还是很大的，尤其是数据库表中字段比较多，关联的表比较多的时候。SQL 语句完全依赖于数据库的特性，这导致 MyBatis 的数据库移植性会很差。MyBatis 不太适应快速数据修改，如果数据库的表结构发生了改变，底层的 SQL 语句也都可能要进行修改，工作量巨大无比。

1.3 MyBatis 相关代码生成工具的现状

目前关于 MyBatis 的代码生成工具很少，实用一点的只有 MyBatis 官方所提供的 MyBatis Generator。

MBG 作为代码生成工具，能内省数据库表，开发人员只需进行简单的 XML 配置，便可以自动生成部分代码，但仅仅是一些执行简单增删改查（CRUD）时所需的 MyBatis 相关的代码，包括 DAO 层、Model 层、Mapper 映射文件。这样一来，开发人员与数据库进行交互时，就不需要自己手动创建对象和配置文件。但是如果要使用存储过程、分页查询和多表级联查询，开发人员还是需要手写对象与 SQL 语句。

在默认情况下 MBG 会为配置的每个表生成实体类的同时，再生成一个对应的 Example 类。这些 Example 类可以用于构造复杂的筛选条件，以满足业务所需要的复杂查询[1]。

1.4 MyBatis 代码生成工具面临的问题

虽然 MyBatis 提供了一个功能强大的 MBG 代码生成工具，但是 MBG 仅仅解决了对数据库操作有最大影响的一些简单的 CRUD（插入、查询、更新、删除）操作。开发者仍然需要对分页查询和多表级联查询手写 SQL 和对象。而分页查询、多表级联查询，正是当今系统开发中不可避免的操作。如果开发人员自己手写这些代码，万一遇上需要修改表结构或者更换数据库，开发人员不得不重新生成一遍代码，那么这些手写的代码很有可能会被覆盖，开发人员不得不重新写一遍这些代码。这是所有开发人员都不愿意见到的，这为项目后期的维护和升级带来了巨大的障碍。所以开发人员需要的是一种更为完善的工具，令开发人员能够更加简单方便地进行开发。

1.5 本课题研究的主要内容

本课题将提出一个为 MyBatis 以及 Spring 框架量身打造的 MyBatis 代码生成工具。借助领域特定语言、模型驱动开发、代码生成技术，为基于 MyBatis 框架的开发提供了新的解决方案，实现 MyBatis 相关代码的全自动生成。解决了采用 MyBatis 开发所带来的诸多问题，开发人员可以完全不知道 MyBatis 的使用细节，便可以参与项目开发。以此来减少开发人员的工作量，降低编码的出错率，提高开发效率，对于促进 MyBatis 的推广和使用，具有十分重要的意义。

本课题为了满足广大开发人员的需求，具有如下几个创新点:

1.根据 MyBatis 的特点，定义适合于 MyBatis 的外部 DSL。开发人员只需要编写适应于自己业务逻辑的 DSL，便可全自动生成 MyBaits 相关的代码，DSL 起到了类似于 API 的作用。本系统采用开发人员所熟悉的 XML 作为外部 DSL 的载体。对于广大开发人员来说，降低了学习的门槛，容易掌握，容易使用，结构简单，但是又能清晰的表示业务逻辑关系。

2.内部通过构建语义模型来驱动。语义模型起到了桥梁的作用。当读取 DSL 脚本生成语法树后，遍历语法树生成语义模型，然后根据语义模型生成代码模型，最后根据代码模型来生成目标代码。语义模型和 DSL 语法树的分离，反映了领域对象和其表现的分离。模型和语言就可以独立演化，如果修改了模型，无须修改 DSL，只需给 DSL 添加必要的语言构造即可，为以后系统的升级与维护提供了便利。

3.该工具采用 Java 语言编写，可以跨平台使用。打成 Jar 包，可以方便地集成到任意 Java 语言开发的项目之中，降低了使用的门槛，提高了使用的效率。

1.6 全文组织结构

论文一开始介绍了 MyBatis 相关的知识，然后介绍此代码生成工具相关的技术，最后提出了基于 MyBaits 的代码生成工具。主要分为以下六个章节:

第一章 绪论。本章针对 MyBaits 持久层框架的现状，介绍了 MyBatis 以及 MyBatis 官方代码生成工具的背景，并阐述了本课题的研究意义。

第二章 相关研究工作。本章主要介绍了该代码生成工具用到的一些关键技术，为后面章节的介绍做好铺垫。

第三章 代码生成工具的需求分析。本章主要针对具体的需求进行分析，明确该工具需要实现的功能与预期的目标。此外还分析了实现中可能存在的关键问

题与难点。

第四章 总体架构设计与功能模块。主要介绍了该工具的总体架构设计以及具体功能模块的设计与实现，包括外部 DSL 脚本、语法树、语义模型、数据模型、解析器、代码模型、代码生成器。

第五章 代码生成工具的使用说明与效果展示。针对具体的案例，对工具的使用进行说明，包括如何配置、如果修改、如果使用接口、如果构建复杂查询等方面。

第六章 总结与展望。对该代码生成工具进行了系统的总结，对该工具存在的问题进行说明，同时提出对该工具改进的建议与今后的计划。

第 2 章 相关研究工作

2.1 MyBatis Generator

MyBatis Generator（MBG）是由 MyBatis 官方提供的一款代码生成工具。它能够内省数据库表，只需开发人员进行简单的 XML 配置，便会自动生成一些简单增删改查（CRUD）执行时所需要的相关代码，包括 DAO 层、Model 层、Mapper 映射文件。这样一来，与数据库进行交互时，开发人员就不需要自己手动地创建对象以及配置文件[4]。但是开发人员仍然需要对存储过程、分页查询和多表级联查询手写对应的对象与 SQL 语句。

MBG 采用了 DSL(Domain Specific Language，领域特定语言)作为支撑。对于 MBG 的配置，正是外部 DSL 的解决方案；而对于 Example 类，正是内部 DSL 的解决方案。因此，后面一章将介绍 DSL 的相关内容。

2.2 领域特定编程语言

DSL(Domain Specific Language，领域特定语言)是针对某一特定的领域，面向具体问题，具有受限表达性的一种计算机程序设计语言[5]。这一定义包含了四个关键的元素:

（1）**计算机程序设计语言**（computer programming language）：与其他大多数现代程序设计语言一样，DSL 的结构也应该被设计得比较容易被人们所理解。但是 DSL 应该还是可以由计算机来执行的，被人们用于控制计算机去做一些针对性的事情。

（2）**语言性**（language nature）：DSL 作为一种程序设计语言，必须具备连续的表达能力。

（3）**受限的表达性**（limited expressiveness）：DSL 只是针对特定领域来解决具体问题的一个很小的集合。DSL 所具有的表达能力仅仅限于某一特定领域，从而无法提供丰富的功能，但是可以有效得解决系统中某一具体问题。DSL 一般无法用于构建一个完整的系统，但是易学易懂易用。

（4）**针对领域**（domain focus）：由于能力的局限性，DSL 只能够被应用于某一明确的小领域之中，这也正是 DSL 的价值体现。

DSL 的核心价值在于，使用者们可以针对系统某个部分的意图进行清晰明确地沟通，可以极大地提高开发效率[5]。DSL 的受限表达性，使得它不容易犯错，

就算犯错了，也容易被发现。使用者仅仅需要关心真正用到的部分，这大大地降低了使用者学习的成本，提高开发效率[6]。在采用 MyBatis 的开发中，通过采用 DSL，可以封装使用者对 MyBatis 的操作，这样使用者无需关心 MyBatis 的使用细节，可以将更多的精力放在其他重要的地方，大大提供开发效率。

当一个软件规模比较大以及业务相对复杂时，客户、设计人员、开发人员、领域专家以及软件用户之间的交流会变得越来越困难，这也正是很多项目失败或者延期最常见的原因之一[7]。DSL 提供了一种清晰明确的语言，能够有效地增进沟通。针对数据库的增删改查操作，如果能够以一种简单清晰的语言表达出来，使用者们之间的沟通也会变得很方便。

当面对需要迁移执行环境时，宿主语言就显得很局限[8]。如果将事务以适当的模型清晰地构造出来，然后根据不同的执行环境，生成对应的代码，就可以应对这个问题[9]。正如当开发人员采用 MyBaits 时，一旦数据库替换了，那么之前的 SQL 语句可能就需要修改，工作量是巨大的。如果能够针对不同的数据库，重新生成相应的 SQL 语句，这是事半功倍的。

优秀的 DSL 具有小巧简单的特点，采用 DSL 不仅可以降低业务逻辑和应用程序之间的耦合度，而且还能有效地来处理业务需求和数据模型的变化，便于系统的升级与维护[10]。当 MyBatis 版本变化时，开发人员仅仅需要适当的修改数据模型以及生成规则，这不会影响到使用者的对工具的使用，因为使用者是可以不关心 MyBatis 的使用细节的。

2.3 模型驱动开发

MDD（Model-Driven Development，模型驱动开发）是一种以模型作为主要部件的高级别抽象的开发方法[11]。MDD 采用模型作为主要部件，是为了解决软件中复杂性和变更能力这两个基本危机[12]。

MDD 的基本思想是让开发从编程转移到更高级别的抽象中去，一般通过模型来生成代码或者其他部件来驱动自动化开发[13]。在分析、架构、设计、实现等不同的阶段都应该有不同的模型，同时要保证这些模型是可以持续更新的，来应对不断变化的环境，这需要在实际开发中根据自己的业务和技术需求去设计优秀的模型[14]。

实际的系统不可能永远只用一种技术来实现，随着复杂度的递增，每当有新技术产生的时候，为了应对需求的不断变化以及文档的迅速失效，开发人员又不得不做大量重复的工作，这些工作显得很有必要，但是又没有什么意义，开发人

员对此普遍抱有不满情绪，浪费了大量的时间[18]。

当然，MDD 不是一种放之四海而皆准的万能解决方案，它比较适合特定领域，因为它只是采用模型为特定情况增强了处理特定问题的能力[19]。面对各种各种的环境，开发人员总是希望能构建出一个万能的模型。但是开发人员必须意识到，只有解决方案空间比较狭窄时，才有可能创建一个高层次的抽象模型。因为随着解决方案空间的扩大，代码生成会越发困难[20]。正如 DSL 里面所强调的，针对具体领域更容易让开发人员从中获利。

2.4 代码生成技术

现如今，手动编写代码的方式之中存在大量重复的工作，会出现千篇一律的代码风格，还容易出错，而且不容易控制代码质量[21]。代码生成技术正是为了解决这些问题而提出的一种解决方案。

代码生成技术一般会提供自定义的配置接口，开发人员可以根据特定的业务需求进行配置，然后生成相应的代码。代码生成技术不仅能够提高开发效率，还能够保证代码风格的一致，极大地提到了代码的可维护性和可读性[22]。这部分生成的代码一般不需要开发人员投入过多的关注，能够让开发人员免于大量的重复性劳动，可以更加专注于上层架构的研发，提高整体系统的质量。

2.5 本章小结

本章主要介绍了本代码生成工具所涉及的一些相关技术，主要包括 Myabtis、MyBatis Generator、领域特定编程语言、模型驱动开发、代码生成技术。这些技术是实现该代码生成工具的基础。

第 3 章 代码生成工具的需求分析

3.1 业务需求

基于 MyBatis 的自适应代码生成工具是一种针对使用 MyBatis 开发的用户的辅助工具。开发人员只需要编写适应于自己业务逻辑的 DSL 脚本，便可以完成 MyBaits 相关的代码的全自动生成，包括 Dao 层、Model 层、Mapper 层映射文件。不仅支持简单的 CRUD（插入、查询、更新、删除）操作，还支持分页查询、多表级联查询，甚至适应数据库的替换以及表结构的修改。

3.2 用户需求

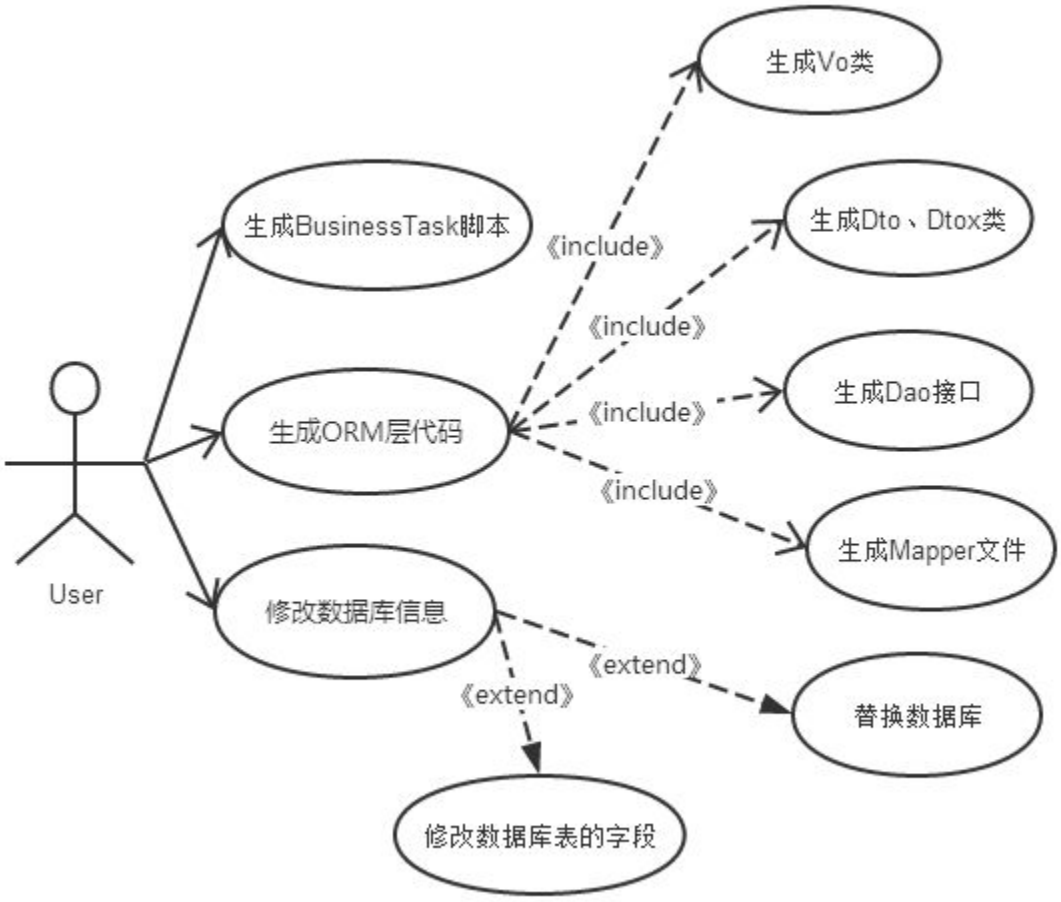

图 3.1 用例图

如图 3.1 所示，开发人员作为该工具的主要用户，可以借助该工具生成 ORM 层相关代码。

生成 Business Task 脚本：通过内部 DSL 的方式，开发人员编写 Java 代码，调用该代码生成工具提供的接口，传入脚本所需要的相关参数，便可以生成对应的

Business Task 脚本。

生成 ORM 层代码：通过外部 DSL 的方式，开发人员需要编写 XML 脚本，包括 Init Task 全局初始化脚本和 Business Task 业务脚本。再利用 Ant 工具来生成 ORM 层相关的代码，包括 Vo 类、Dto 类、Dtox 类、Dao 接口和 Mapper 文件。Vo 类是与数据库表所对应的实体类；Dto 类是根据业务脚本中配置的查询条件所封装出来的传输对象类；Dtox 类继承自 Dto 类，用于方便开发人员的扩展修改；Dao 接口是开发人员访问数据库的接口；Mapper 文件是 MyBatis 的映射文件。

修改数据库信息：开发人员可以根据自己实际开发中的情况，修改数据库的信息，包括数据库的替换以及数据库表字段的修改。

3.3 功能需求

其实该代码生成工具的功能需求十分简单，仅仅是生成代码。该工具的核心功能便是能够自动生成持久化层相关的代码，减少开发人员的工作量。主要包括 Dao 层代码、Model 层代码、Mapper 映射文件。不仅支持简单的 CRUD（插入、查询、更新、删除）操作，还支持分页查询、多表级联查询，甚至适应数据库的替换以及表结构的修改。

3.4 非功能需求

该代码生成工具提供的 DSL 描述规则要简单清晰，开发人员能够轻松掌握，降低学习时间，减少开发成本。

该代码生成工具在生成代码的过程中需要输出关键的日志信息，如果生成代码的过程中发生了错误，要输出友好的提示信息，告知错误发生的位置以及具体的原因，方便开发人员的使用。

该代码生成工具能够生成的代码需要具有统一的代码风格，并且能够将生成的代码和手写的代码区分开，方便开发人员查看以及修改。

3.5 关键问题

难点 1：关于本工具中 DSL 的定义，需要对 MyBatis 进行深入的研究，以及参考其他优秀 DSL 的设计，DSL 是一切的开始，是用户使用的接口。这里设计两个 DSL，一个是用于生成业务脚本的内部 DSL，还有一个是用于配置生成规则的外部 DSL。

难点 2：模型的构建，这里涉及了很多模型，主要包括数据模型、语义模型、代码模型。在分析、架构、设计、实现等不同的阶段都应该有不同的模型，同时要保证这些模型是可以持续更新的，来应对不断变化的环境。对于模型的制定涉及到

许多方面知识和技巧。

难点 3：代码生成时如何保证代码本身的正确性、权威性。如果允许开发人员手写代码，那么又要如何区分生成的代码和手写的代码。

一旦上述这些问题解决之后，便可以顺利的实现一个快捷高效的 MyBatis 代码生成工具。

3.6 本章小结

本章主要对该代码生成工具进行了简单的需求分析，阐述了完成该工具需要实现的功能以及在实现过程中可能会遇到的一些主要问题。

第 4 章 总体架构设计与功能模块

本章将对整体框架进行设计以及对各个功能模块进行详细的描述。本系统设计分为七大模块：外部 DSL、语法树、语义模型、数据模型、解析器、代码模型和代码生成器。在确定各个模块之后，将对每个模块进行详细说明，以及介绍各个模块的实现方式。

4.1 总架构设计

通过对 MyBatis 的学习和 DSL 的研究，针对 MyBatis 动态 SQL 构建一个基于 XML 的简单实用的外部 DSL。然后通过相应的 XML 工具，读取 MXL 并且保证得到正确的语法树。随后进行语义模型的构建，将读取 DSL 得到的语法树转换成系统内部的语义模型，随后将语义模型转换成相应的代码模型，最后通过代码生成器生成对应平台的目标代码。具体设计方案如图 4.1 所示。

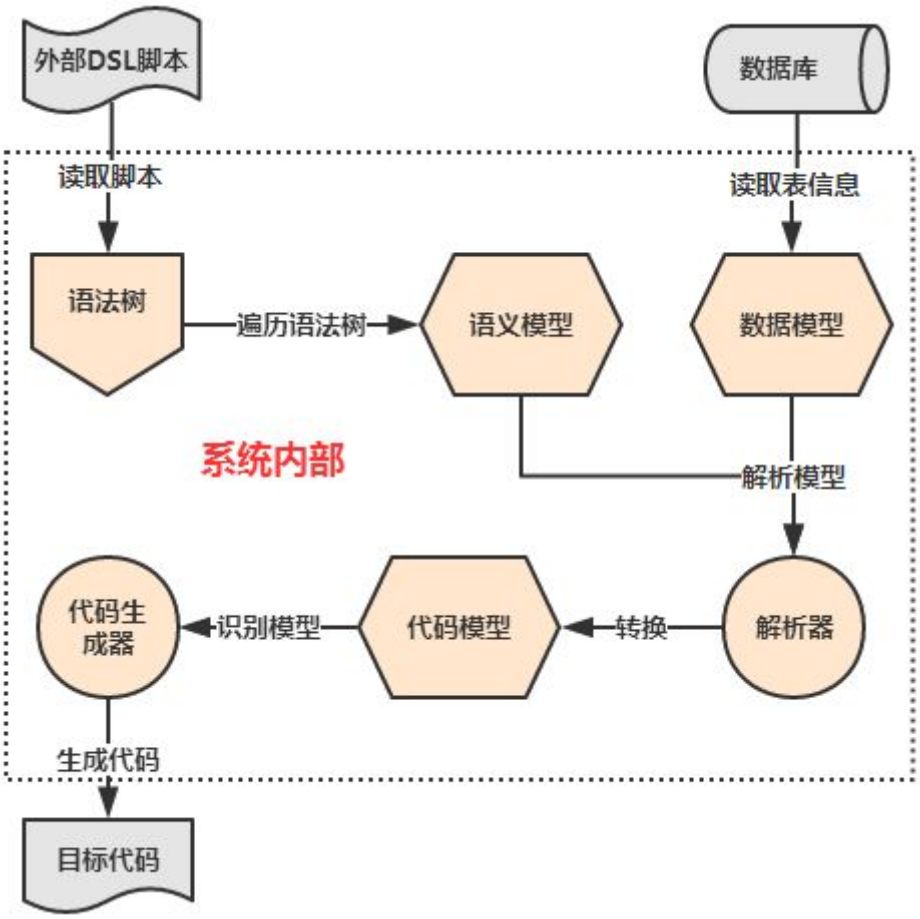

图 4.1 系统基本架构图

4.2 外部 DSL 脚本的设计

DSL 脚本的目的是用一种特定语言来表达用户所要表达的业务逻辑，最后通过系统内部程序来生成最终的代码，也就是说用一种更为简单的模型来描述复杂的模型。

本系统主要采用 XML 作为载体，为 MyBatis 相关代码制定特定的表达方式，主要包括数据库信息、项目信息和多表查询的逻辑等。这里将采用开发人员所熟悉的 XML 作为外部 DSL 脚本的载体，这样可以让开发人员避免学习新的语法，从而能更快的使用 DSL。因为如果开发人员一下子接触太多的新技术，很容易产生抵触心理。并且 XML 相关的工具有很多，可以通过这些工具直接读取 XML，自动解析成语法树，可以省去大量的时间，还不容易出错。

外部 DSL 脚本一共分为两部分: 一个是全局信息（称之为“全局配置脚本”），而其他都是针对数据库中每一个表编写的局部信息以及具体业务逻辑（称之为“业务逻辑脚本”）。

在全局任务脚本中，包含了数据源信息和项目信息，如图 4.2 所示。

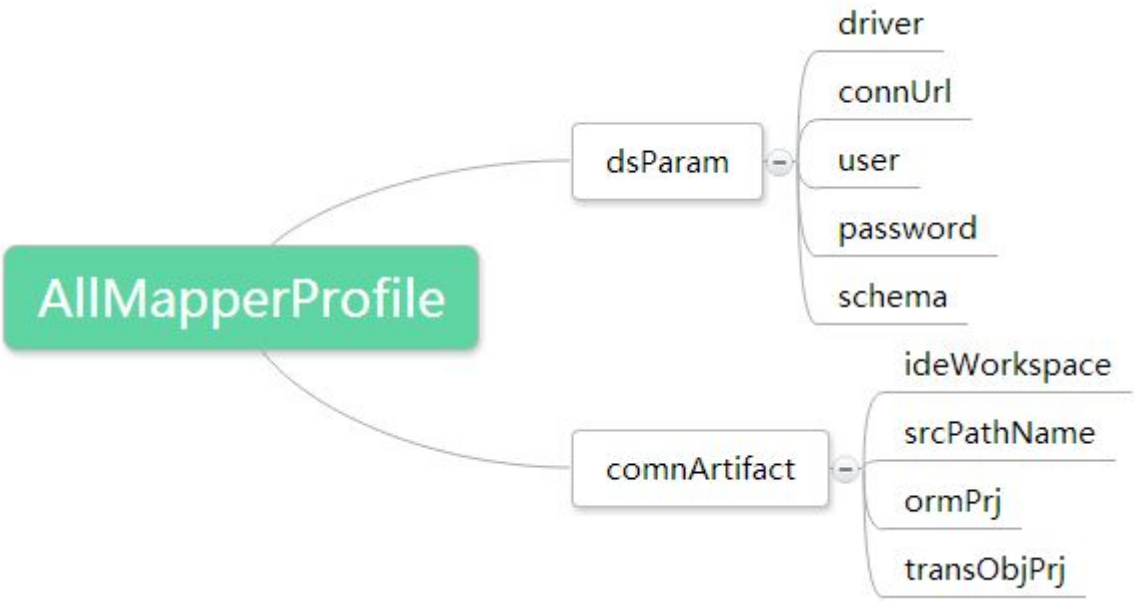

图 4.2 全局配置脚本模型

其中 AllMapperProfile 包含了 dsParam 和 ComnArtifact 两个子元素，分别代表数据源信息和项目信息。dsParam 中的 driver 代表用于访问数据库的 JDBC 驱动程序的完全限定类名称，connUrl 代表用于访问数据库 JDBC 连接的 URL，user

代表访问数据库的用户帐号，password 代表访问数据库的密码，schema 代表访问数据库的 schema。根据 dsParam 里的这些信息，就可以连接到用户所需要的数据库，并且访问里面的数据。comnArtifact 中 ideWorkspace 代表开发中工作空间的全路径，ormPrj 代表 ORM 项目的名字，transObj 代表 DTO 数据传输对象所在项目的名称，srcPathName 代表项目中源码的相对路径。

在业务逻辑脚本中，包含了具体的业务逻辑，如图 4.3 和图 4.4 所示。

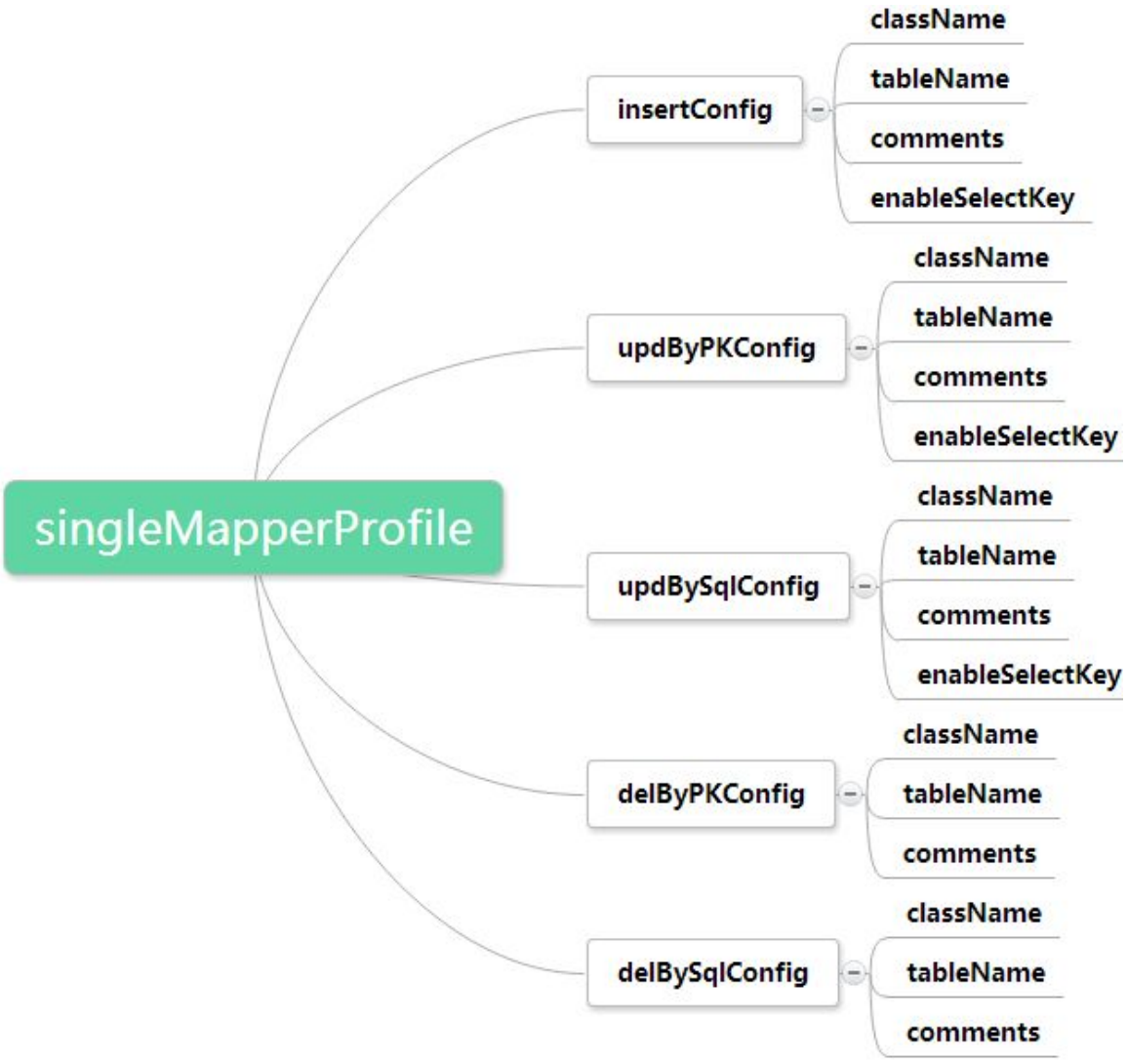

图 4.3 业务逻辑脚本部分模型

其中 singleMapperProfile 包含了 mapperArtifact、resultMapConfig、insertConfig、updByPkConfig、updBySqlConfig、delByPkConfig 和 delBySqlConfig。mapperArtifact 中的 mapperNs 表示对应 MyBatis 中 mapper 文件的命名空间。

resultMapConfig 用于表示查询操作，其中的 SelectStmt 表示 mapper 文件中 select 标签的 id 属性以及 Dao 文件中的方法名，className 表示查询结果 Dto 的全类名，tableName 表示查询的表名称，dftOrderBy 表示默认排序的列，comments 表示 select 查询的注释信息。filterConfig 表示查询 SQL 语句最后的过滤条件，其中 attribute 表示属性名，comparator 表示比较的规则，value 表示比较的值。oneToOne 表示两个表级联查询时,一对一的关系；oneToMany 表示两个表级联查询时，一对多的关系。由于可能会关联过的表，所以它们可以和 resultMapConfig 嵌套使用。

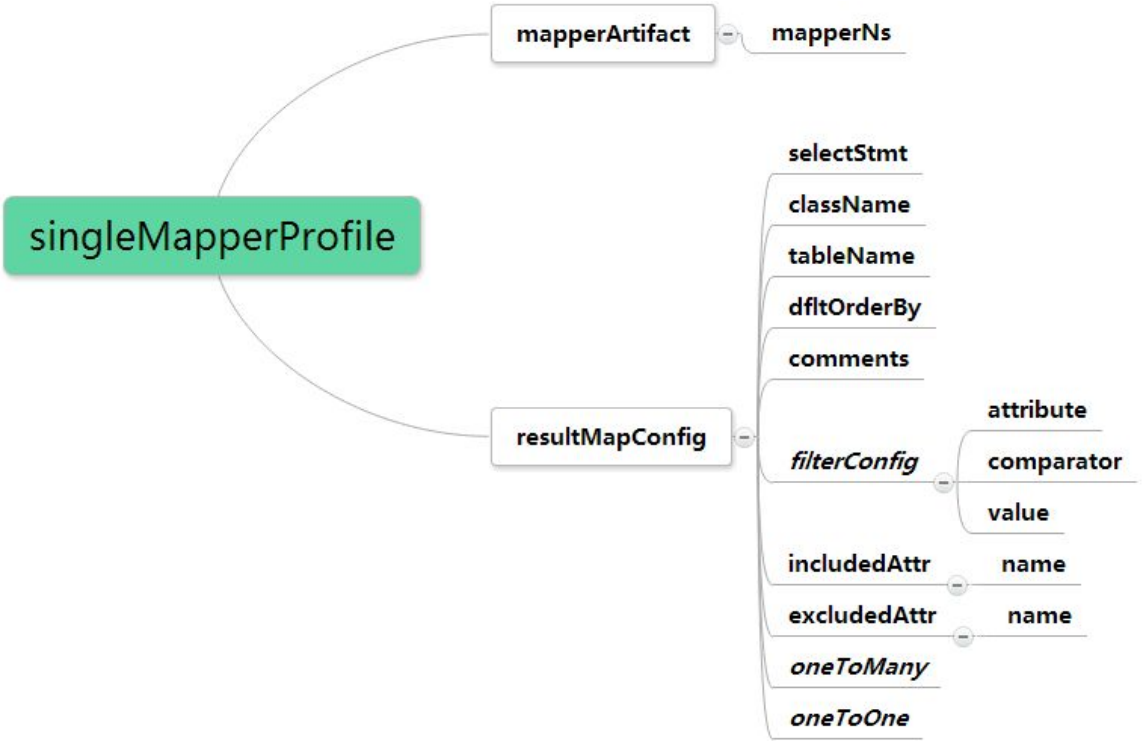

图 4.4 业务逻辑脚本部分模型

如图 4.5 所示，OneToOne 和 OneToMany 也都包含了 resultMapConfig。OneToMany 中 listOfSon 表示所关联的表在当前传输对象 Dto 中对应的属性（通常为列表对象），fatherAttr 表示用于被关联表的外键所对应的实体类中的属性，sonAttr 表示关联表的主键所对应实体类中的属性，joinType 表示 join 的类型。OneToOne 中唯一不同的就是 refToson，refToSon 表示所关联的表在当前传输对象 Dto 中对应的属性，其他跟 OneToMany 都是一样的。

singleMapperProfile 中的 insertConfig 表示插入操作，updByPKConfig 表示根据主键的更新操作，updBySqlConfig 表示根据 SQL 的更新操作，delByPKConfig

表示根据主键的删除操作，delBySqlConfig 表示根据 SQL 的删除操作。

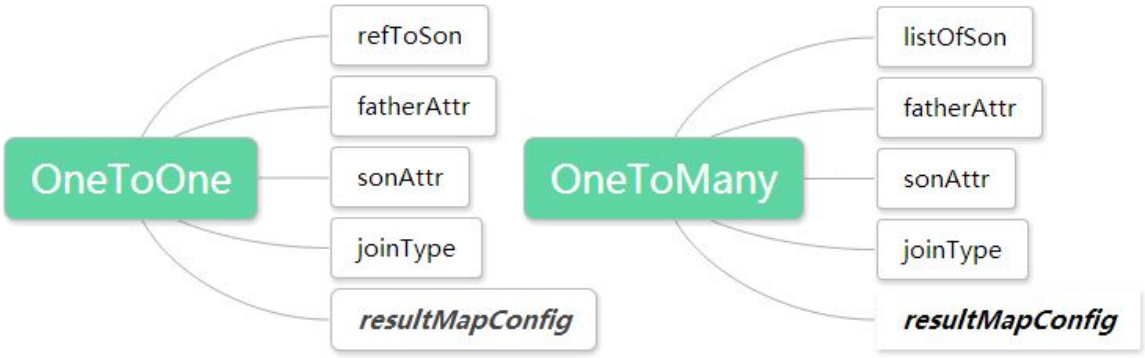

图 4.5 级联查询相关模型

其中除了 className 代表的有些不同，其他都是一样的，tableName 表示当前表的名称，comments 表示当前插入操作的注释，enableSelectKey 表示对于非空属性是否需要进行操作。insertConfig、updByPKConfig 和 delByPKConfig 中的 className 表示当前表所对应的实体类 VO 的全类名，updBySqlConfig 和 delBySqlConfig 中的 className 表示当前传输对象 DTO 的全类名。

4.3 语法树和语义模型的设计与实现

在上一小节中，已经详细阐述了外部 DSL 脚本所对应的 XML Schema 的设计。语义模型用于将用户所写的 DSL 脚本以结构化的形式存储在系统内部。语义模型作为 DSL 和解析器之间的桥梁，让结构以及转变过程更为清晰。

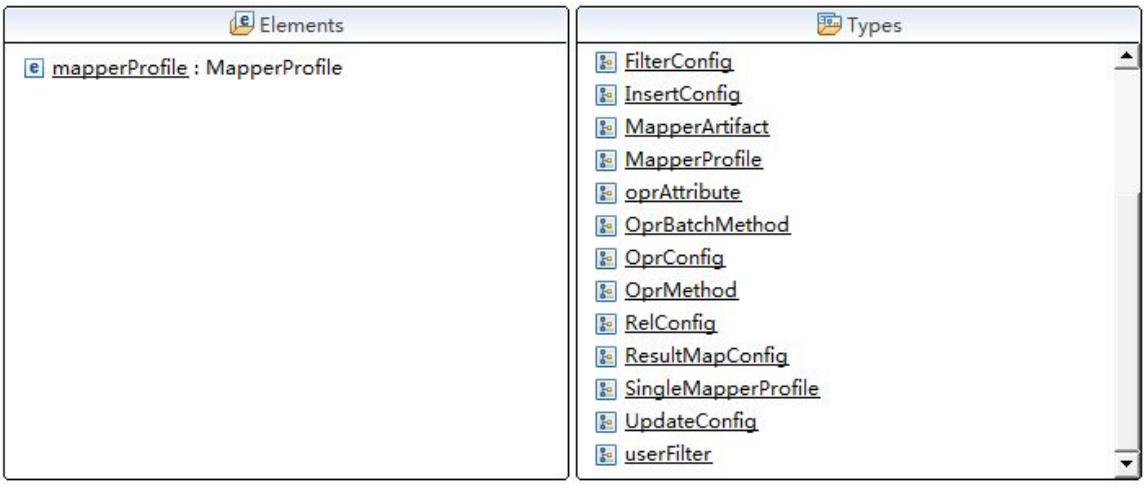

图 4.6 DSL XML Schema

将 XML Schema 在 Eclipse 中以 xsd 文件描述出来，这样可以方便查看与修改，

如图 4.6 所示。然后用使用 Eclipse 直接把 Schema 转换成对应的 JAXB 类。

这样一来，语义模型就出来了。当用户写好外部 DSL 脚本时，便可以读取脚本，创建对应的模型结构进行存储。由于外部 DSL 采用 XML 作为载体，现在有很多 XML 相关的工具可以使用，本系统采用 Ant 来实现 XML 文件的读取。Ant 具有跨平台性，同样采用纯 Java 语言来编写。Ant 通过调用 target 树，就可以执行各种 task。Ant 可以简单方便地集成到系统的开发环境之中，作为一个功能强大的自动化工具十分适合本系统。可以将连接数据源以及初始化工作参数的工作定义成 Ant 中的一个初始化 Init Task。然后将其他每个业务操作工作定义成单独的 Business Task，而这些 Business Task 依赖于前面的 Init Task，简直不谋而合，如图 4.7 所示。

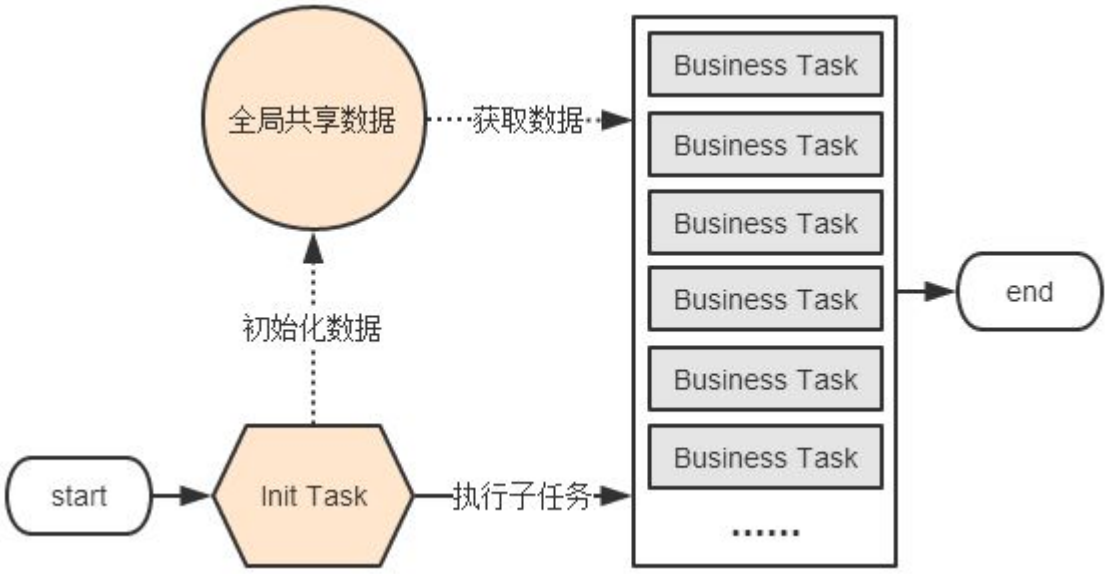

图 4.7 Ant Task 执行顺序图

当 Init Task 执行完以后，所有 Business Task 都可以使用这些初始化数据。Business。Task 再通过对应的语义模型来生成相应的代码。

4.4 数据模型的设计与实现

系统根据用户在外部 DSL 脚本中所配置的数据源信息，读取数据库中所有相关的数据库表，生成具有数据库表信息的数据模型。这个任务将依赖前面提到的 Init Task。数据模型应该包括两个部分，一个是所有表的信息，另一个是表里所有列的信息。需要定义一个实体类，用于存储列信息，包括列的名称、列的注释、列的默认值、是否为主键，列对应的 JDBC 类型。还需要定义个实体类，用于存

储表信息，包括表的名称、表的注释以及所有列对象的集合，大致的类图如图 4.8 所示。

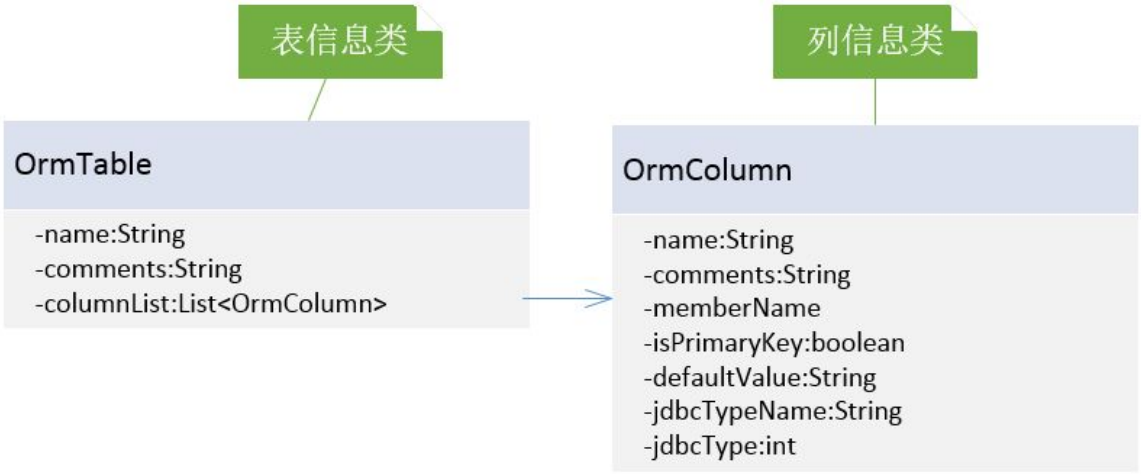

图 4.8 表信息和列信息类图

这里要注意的是不同的数据库中类型会有所差异，这里就需要考虑到以后的扩展以及维护方面的需求，可以采用工厂模式，将识别列这种会变化的操作封装起来，方便以后针对不同类型数据库做相应的定制操作，如图 4.9 所示。

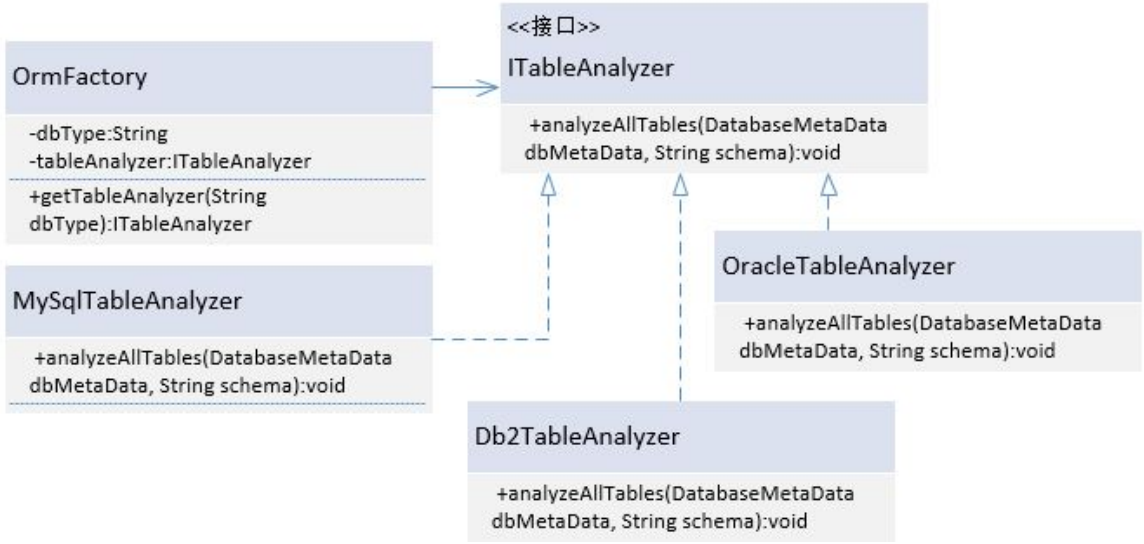

图 4.9 数据库表格分析相关类图

所有的数据库分析类都由 OrmFactory 类提供的 getTableAnalyzer 方法来构建，并且通过传入的参数 dbType 来区分所构建数据库分析类的类别，其中类别包括 MYSQL、DB2、ORACLE 以及 OTHER。方法返回的结果是一个 ITableAnalyzer 的接口，采用了面向接口编程的原则。MySqlTable、DBTableAnalyzer 以及

OracleTableAnalyzer 分别针对 mysql、db2、oracle 实现了 ITableAnalyzer 接口中的 analyzerAllTables 方法，将对数据库的所有表进行分析并且存储到数据模型之中。

4.5 解析器的设计与实现

解析器主要是通过解析语义模型和数据模型，生成代码模型。这部分将是系统的“心脏”，一切的成败都在于此。本系统中解析器要做的是遍历语义模型中所有的 SingleMapperProfile，构建所有的增删改查。由于需要构建的东西较多，所以可以将职责划分的更为明确一点，为增删改查分别创建单独的构建类，大致设计的类图如图 4.10 所示。

为查询操作相关代码的构建设计一个名为 ResultMapBuilder 的类；为插入操作相关代码的构建设计一个名为 InsertBuilder 的类；为更新操作相关代码的构建设计一个名为 UpdateBuilder 的类；为删除操作相关代码的构建设计一个名为 DeleteBuilder 的类。

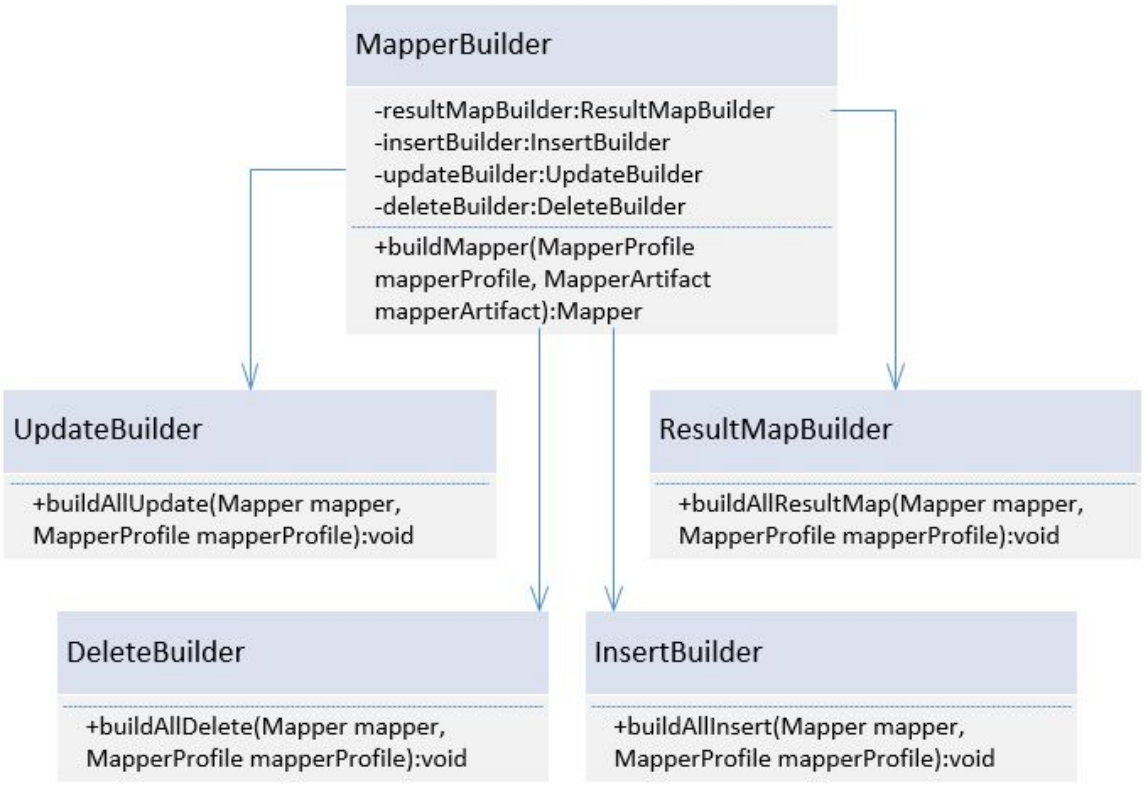

图 4.10 解析器类图

在 ResultMapBulder 中，需要将其所对应的数据库表转换成相应的实体类 Vo，如果缓存中已经存在对应实体类 Vo，便不会重新构建实体类 Vo。其次需要构建

基本查询操作，包括 ResultMap 结果集，根据主键查询基本表，根据 SQL 语句 where 条件查询基本表，以及相应的 COUNT 查询。最后是构建复杂的业务查询，主要是根据 OneToOne 和 OneToMany 的关系来构建。

在 InsertBuilder 中，同样需要先将其所对应的数据库表转换成相应的实体类 Vo，如果缓存中已经存在对应实体类 Vo，便不会重新构建实体类 Vo。然后是构建所有列的插入，选择性的插入（看值是否为 NULL），批量插入，没有主键的批量插入。

在 UpdateBuilder 中，同样需要先将其所对应的数据库表转换成相应的实体类 Vo，如果缓存中已经存在对应实体类 Vo，便不会重新构建实体类 Vo。然后是构建根据主键的更新操作，根据主键的更新分为两种，一种是有时间锁，一种是没有时间锁。接着是构建根据主键选择性更新操作，同样分为两种，一种是有时间锁，一种是没有时间锁。最后是构建根据 SQL 的 where 语句来更新的操作，一种所有列都更新，一种是根据值来判断选择性更新。

在 DeleteBuilder 中，同样需要先将其所对应的数据库表转换成相应的实体类 Vo，如果缓存中已经存在对应实体类 Vo，便不会重新构建实体类 Vo。然后是构建根据主键删除的操作，包括带有时间锁的删除、没有时间锁的删除、批量删除。最后构建根据 SQL 语句的 where 条件来删除。

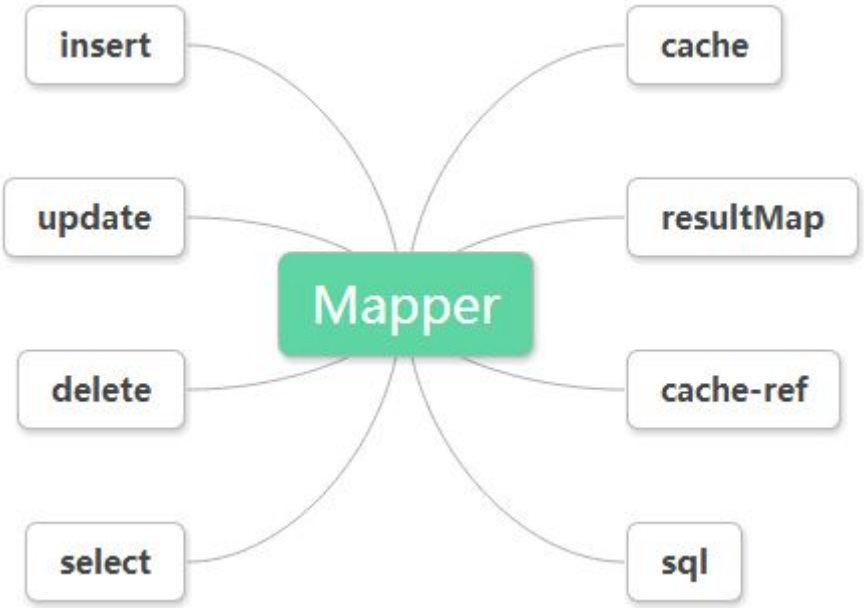

图 4.11 Mapper 模型图

4.6 代码模型的设计与实现

代码模型主要包含了 MyBatis 相关代码的信息，包括实体类、Dao 层代码、Mapper 映射文件等等。用户完全不用关心这部分，他们只要关心这个模型生成的代码是不是他们所期望的就行了。

对于 Mapper 映射文件的代码模型，主要依据 MyBatis3 官方所提供的配置文件参数格式进行制定，Mapper 映射文件顶层元素有九个，其中 parameterMap 已经废弃，其他分别是 cache、cache-ref、resultMap、sql、insert、update、delete、select，如图 4.11 所示。后面将从 SQL 语句本身开始介绍 Mapper 映射文件模型中每个元素的细节。

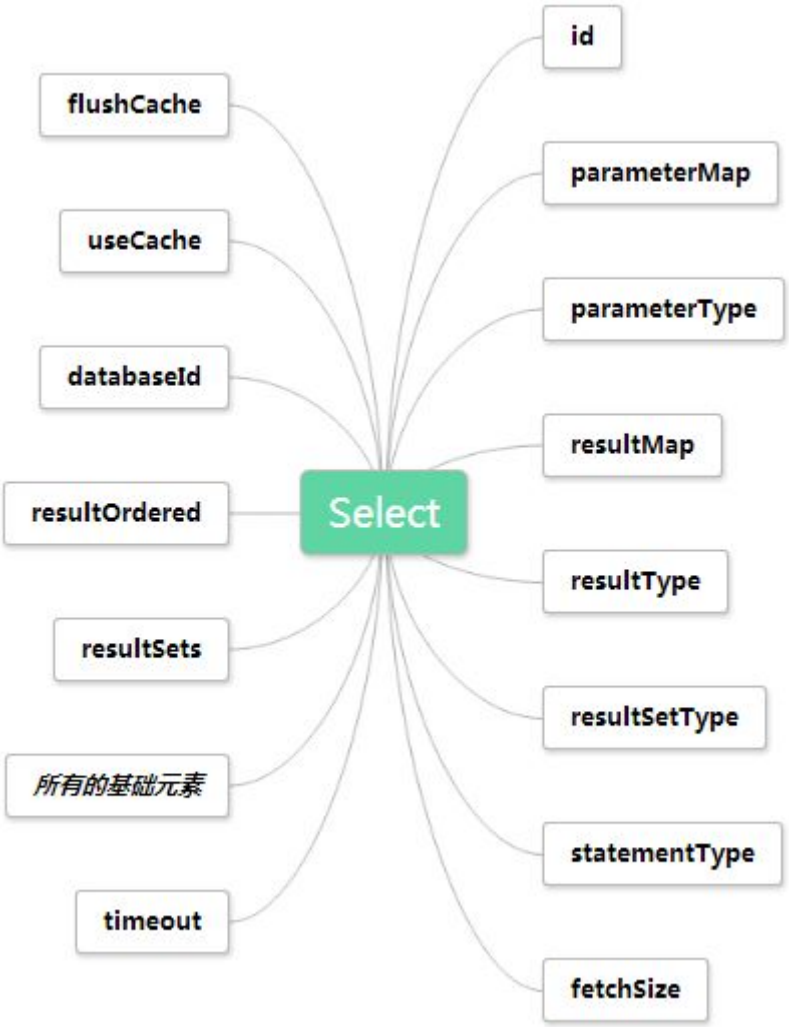

图 4.12 Select 模型图

select 查询语句是 MyBatis 映射文件中最常用的元素之一，如图 4.12 所示。id 是在命名空间中唯一的标识符，可以被用来引用这条语句。parameterType 是将会传入这条语句的参数类的完全限定名或别名。resultType 表示从这条语句中返回的期望类型的类的完全限定名或者别名，如果是集合，那么应该是集合所包含额类型，而不是集合本身。resultMap 是外部 resultMap 的命名引用，resultMap 和 resultType 不能同时使用。flushCache 代表缓存是否需要清空，如果设置为 true，任何时候只要语句被调用，都会导致本地缓存和二级缓存都会被清空，默认值是 false，也就是不清空。useCahce 表示是否要进行二级缓存，如果设置为 true，所执行语句的结果会被进行二级缓存，默认值是 true。timeOut 表示在抛出异常之前，驱动程序等待数据库返回请求结果的秒数。fetchSize 代表驱动程序每次批量返回结果的行数。statementType 的值有三种，STATEMENT、PREPARED 或者 CALLABLE，会使得 MyBatis 分别使用 Statemet、PreparedStatement 或 CallableStatement，默认值是 PREPARED。resultSetType 的值有三种，FORWARD_ONLY、SCROLL_SENSITIVE 或 SCORLL_INSENSITIVE。databaseId 表示如果配置了 databaseIdProvider，Mybatis 会加载所有不带 databaseId 或者匹配当前 databaseId 的语句，如果这两种语句都有，则不带 databaseId 的语句会被忽略。resultOrdered 这个设置仅仅适用于嵌套结果 select 语句，如果设置为 true，就是假设包含了嵌套结果集，这样一来，当返回一个主结果行时，就不会发生有对前面结果集的应用情况了，这使得在获取嵌套结果集的时候不至于导致内存不够用，默认值是 false。resultSets 这个设置仅适用于多结果集的情况，它将列出语句执行后返回的结果集并且每个结果集都会给予一个名称，名称会以逗号隔开。

“所有基础元素”包含了很多地方都会用到的元素，所以在这里归类在一起，以免重复介绍，也方便画图，令图更加简单明了。正因为很多地方都会用到这些元素，所以称之为“基础元素”，如图 4.13 所示。

MyBatis 的动态 SQL 就是建立在这些基础元素之上的，其中的 include 用于对其他元素的引用，通过设置 refid 的值，来引用对应 id 的元素，id 值在命名空间中具有唯一标识的特性。其中的 cDataBegin 和 cDataEnd 是成对出现的，生成代码是会替换成 XML 语法中的 CDATA，表示里面的所有内容都将被解析器忽略。其中 trim 元素的主要功能是可以在自己所包含的内容前面加上某些前缀，同样也可以在它后面加上某些后缀，与之对应的属性是 prefix 和 suffix。利用 trim 可以把包含内容的首部或者尾部的某些内容覆盖掉，对应的属性是 prefixOverrides 和

suffixOverrides，也可以用于替代 where 元素的功能。

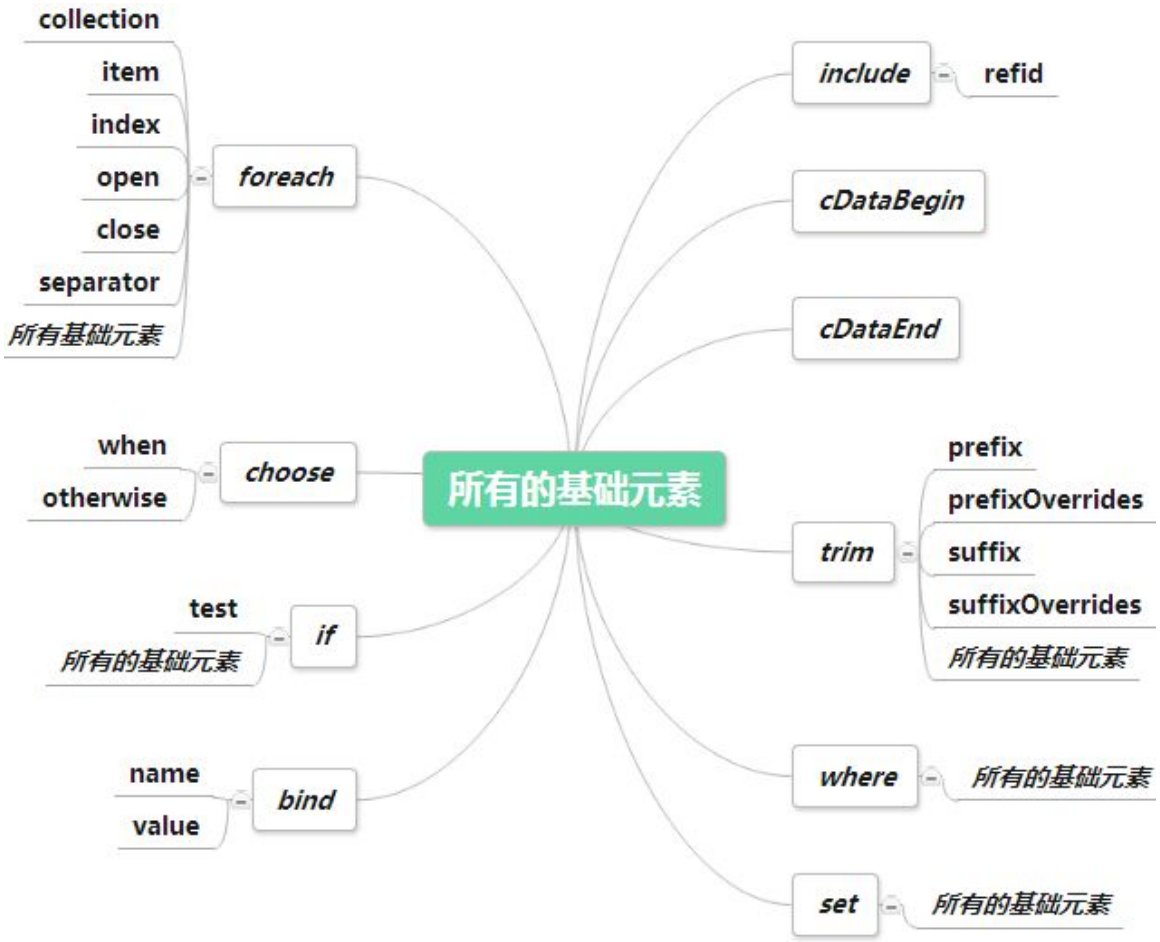

图 4.13 所有基础元素图

其中 where 语句的作用主要是简化 SQL 语句中 where 里的条件判断。where 元素的作用是将会在写入 where 元素的地方输出“where”，它的好处是不需要考虑 where 元素里条件输出的样子，MyBatis 会自动帮开发人员处理这些琐事，会把开头多余的 and 或者 or 忽略掉。此外，在 where 元素里，开发人员不需要过多的关心空格的问题，MyBatis 同样会智能地加上。其中 foreach 的主要作用是在构建 in 条件的时候，它可以在 SQL 语句中进行集合迭代操作。foreach 元素的主要有 collection、item、index、open、close、separator 六个属性。collection 表示集合，是 foreach 中最为关键的，也是最容易出错的，该属性必须被指定，有 list、array、map 三种。item 表示集合内每一个元素进行迭代的时候的别名，index 用于表示迭代过程中，每次迭代到的位置。open 表示该语句以什么作为开头。separator 表示在每次进行迭代的时候，中间以什么符号作为分隔符。close 代表以什么作为结

束。choose 元素的作用类似于 JAVA 中的 switch 语句，基本上跟 JSTL 中的 choose 用法是一样的，通常与 where 和 otherwise 搭配使用。when 元素是按照条件顺序来进行条件判断，如果条件满足，就会输出其中的内容，并且跳出 choose。当所有的条件都不不能匹配满足的话，就会输出 otherwise 中的内容，就类似于 JAVA 中 switch 里的 default。还有 if 元素就是简单的条件判断，使用 if 语句，可以实现一些简单的条件选择。set 标签提出追加到条件末尾的任何不想关的逗号，当在 update 语句中使用 if 标签时，如果前面的 if 没有执行，则可能导致出现逗号多余的错误。最后一个是 bind 元素，这个元素只有 name 和 value 两个属性，它作用就是把 value 里的值和 name 绑定，也就是把 value 赋值给 name。这样就可以在需要用到 value 里的值的时候，采用 name 来替换，将会变化的部分放在 value 里，以后仅仅需要修改 value 里的值便可以。

insert、update 和 delete 都差不多，只有一点点的区别。insert 的具体模型如图 4.14 所示，一部分跟前面介绍的 select 是一样的。其中 id 跟 select 一样是命名空间中的唯一标识符，用于代表这条语句。parameterType 是将要传入到语句中的参数的完全限定类名或者别人。flushCache 同样代表本地缓存和二级缓存是否需要清空。timeout 同样是驱动程序将等待数据库返回结果的最长秒数。databaseId 跟 select 中一样，跟配置的 databaseIdProvider 一起产生作用。statementType 也是一样的，分为 STATEMENT、PREPARED、CALLABLE 三种。useGeneratedKey 仅仅对 insert 和 update 有用，代表是否使用 JDBC 的 getGeneratedKeys 方法来获取由数据库内部自动生成的主键。keyCloumn 同样仅仅适用于 insert 和 update 之中，通过生成的键值设置表的列名。keyProperty 仅仅适用于 insert 和 update，唯一标记一个属性，MyBatis 会通过 getGeneratedKeys 的返回值或者通过 insert 语句的 selectKey 子元素来设置它的键值。selectkey 元素里的 keyProperty 是 selectKey 语句结果应该被设置的目标属性；keyCoumn 匹配属性的返回结果集中的列名称，如果想要得到多个生成的列，用逗号分割就行。resultType 代表结果的类型，通常 MyBatis 自己可以推算出来，但是自己填写，可以更加明确。Order 可以被设置为 BEFORE 或者 AFTER。如果设置成 BEFORE，那么它会首先选择主键，设置 keyProperty，然后执行 insert 语句。如果设置成 AFTER，就会先执行插入语句，然后才是 selectKey 元素。statementType 支持 STATEMENT、PREPARED 和 CALLABLE 语句的三种映射类型，它们分别代表 Statemet、PreparedStatement 和 CallableStatement 类型。

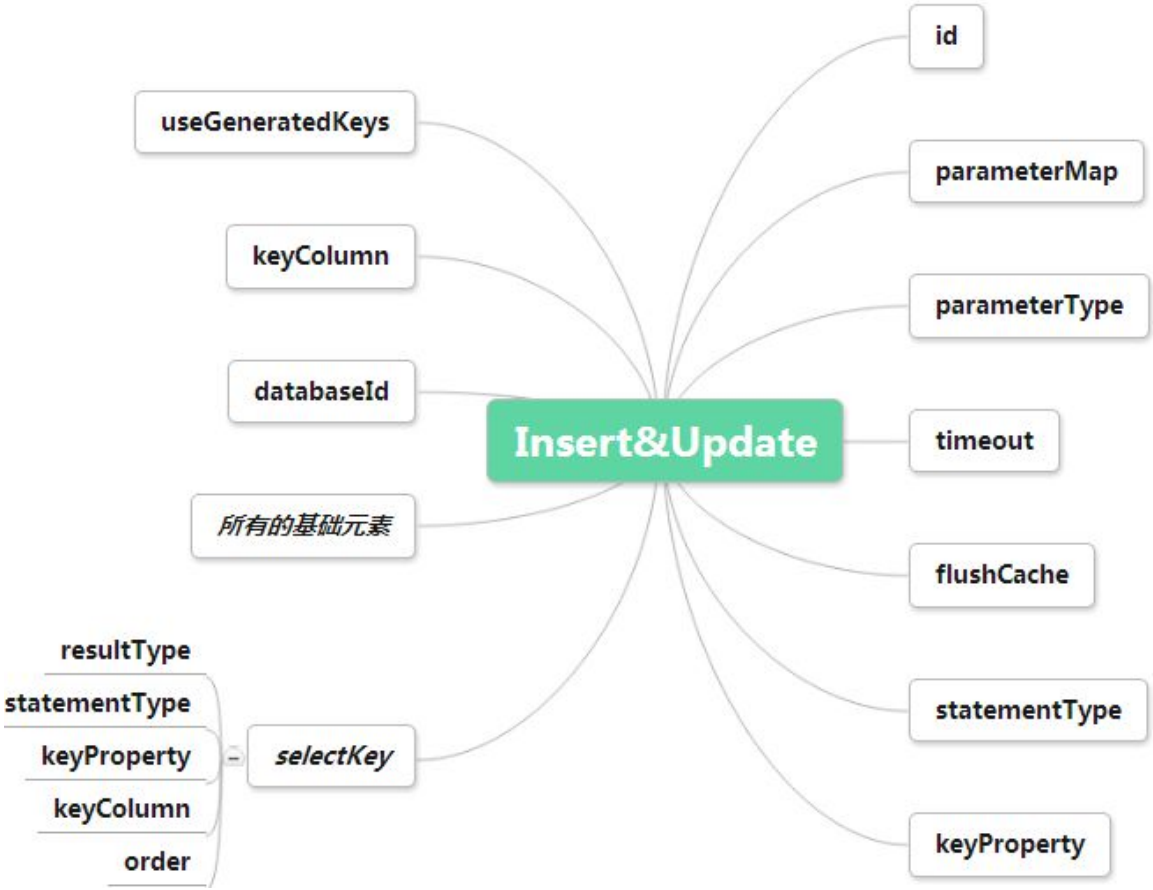

图 4.14 Insert&Update 模型图

Sql 元素用于定义可复用的 SQL 代码段，可以包含在其它语句中。通过 refid 可以包含对应 id 的 SQL 代码段。

表 4.1 支持的 JDBC 类型

BIT	FLOAT	CHAR	TIMESTAMP	OTHER	UNDEFINED
TINYINT	REAL	VARCHAR	BINARY	BLOG	NVARCHAR
SMALLINT	DOUBLE	LONGVARCHAR	VARBINARY	CLOB	NCHAR
INTEGER	NUMERIC	DATE	LONGVARBINARY	BOOLEAN	NCLOB
BIGINT	DECIMAL	TIME	NULL	CURSOR	ARRAY

resultMap 元素是 MyBatis 中最重要并且最强大的元素，其代码模型如图 4.15 所示。resultMap 元素有很多属性和子元素。属性有三个，分别是 id、type、autoMapping。id 是命名空间中的唯一标识，可以被引用到别的结果集中；type

是完全限定类名或者别名；autoMapping 代表是否自动映射结果集，如果启动自动映射，MyBatis 会自动查找与字段名小写同名的属性名，并且调用 setter 方法。

接下来详细说明 resultMap 里的每一个元素。idType 和 result 都是用于映射一个单独列的值到简单数据类型的单独属性或者字段。它们两者之间唯一的不同点是，id 表示的结果是标识属性，也就是说用于设置主键字段与领域模型属性的映射关系；而 result 用于设置普通字段与领域模型属性的映射关系。idType 和 result 中的 property 代表映射到列结果的字段或者属性；column 代表从数据库中得到的列名或者列名的重命名标签；javaType 是 JAVA 类的完全限定名或者类型别名；jdbcType 是当前数据库表所支持的 JDBC 类型列表中的类型，JDBC 类型仅仅需要对插入、更新和删除操作中可能为空的列进行处理；typeHandler 用于设置一个类型处理器，可以覆盖默认的类处理器，该属性的值为类的完全限定名或者一个类型处理器的实现或者别名，所有支持的 JDBC 类型如表 4.1 所示。

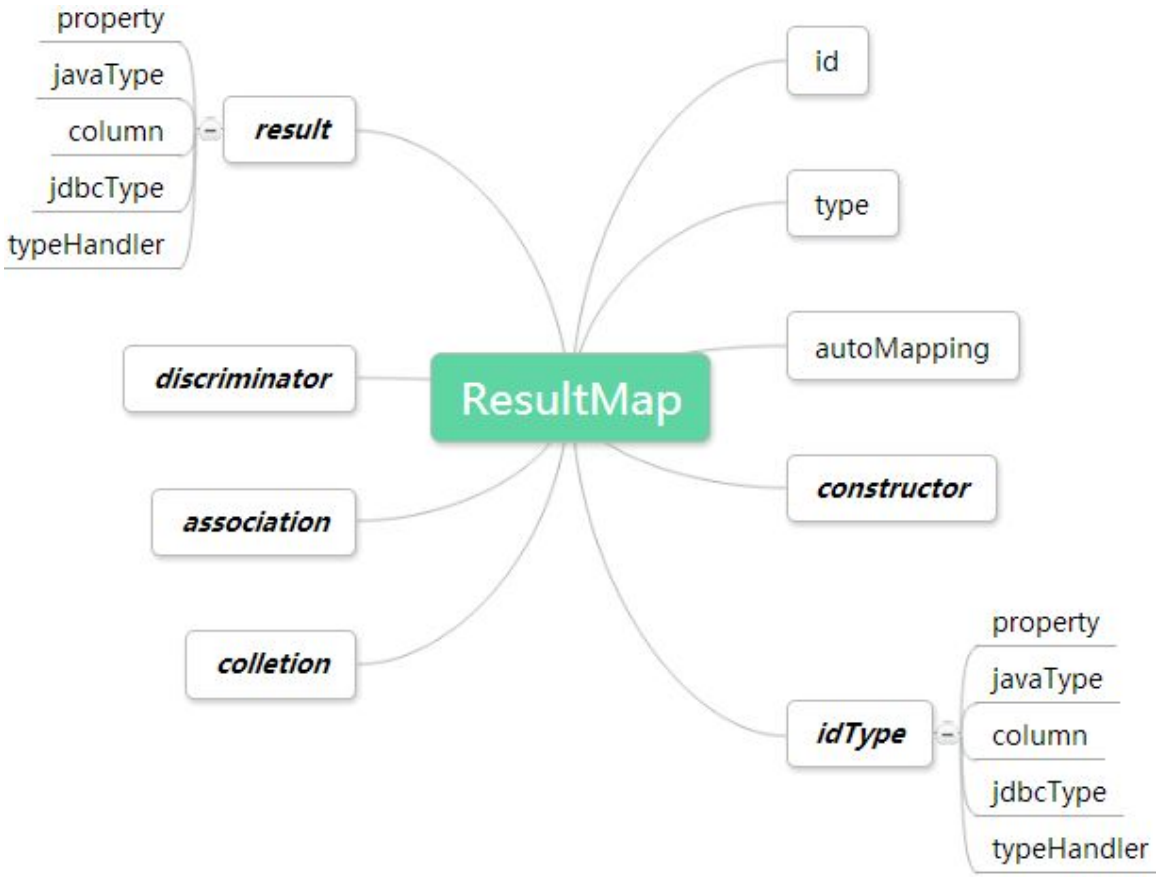

图 4.15 ResultMap 模型图

resultMap 中的 constructor 元素用于使用参数列表的构造函数来实例化领域模型，需要注意的是，它的子元素顺序必须与参数列表顺序对应。其中 idArg 子元素表示该入参为主键，arg 子元素表似乎该入参为普通字段，当然主键使用 arg 来设置也是可以的。idArg 和 arg 的属性都是一样的：column 来自数据库的列名或者别名；javaType 是一个 JAVA 类的完全限定名或者类型别名；jdbcType 表示当前数据库表格所支持的 JDBC 类型列表中的类型，JDBC 类型仅仅需要对插入、更新和删除操作中可能为空的列进行处理；typeHandler 是类型处理器，可是是类的完全限定名、一个类型处理器的实现或者类型别名；select 表示其他映射语句的 ID；resultMap 表示其他 ResultMap 的 ID。

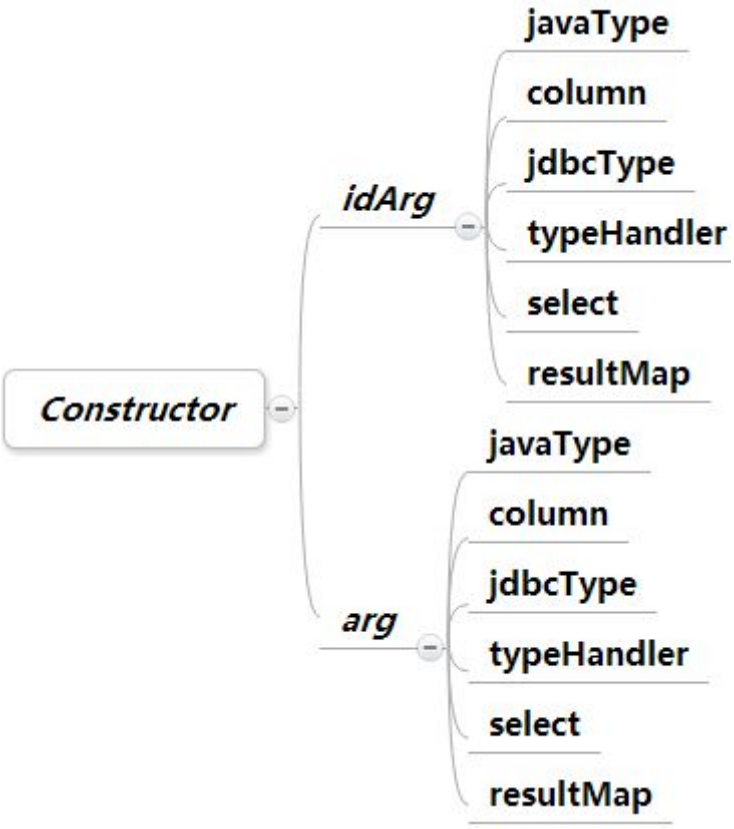

图 4.16 Constructor 模型图

resultMap 中的 association 元素用于处理 “has-one” 类型的关系，有两种方式可以告诉 MyBatis 如何加载关联，嵌套查询和嵌套结果。嵌套查询方式是通过执行另一个 SQL 映射语句来返回预期的复杂类型；嵌套结果方式使用嵌套结果映射来处理重复的联合结果的子集。association 里的 property 是映射到列结果的字段或者属性；javaType 是一个 JAVA 类的完全限定名或者一个类型别名；jdbcType

表示当前数据库表格所支持的 JDBC 类型列表中的类型，JDBC 类型仅仅需要对插入、更新和删除操作中可能为空的列进行处理；typeHandler 用于指定类型处理器。

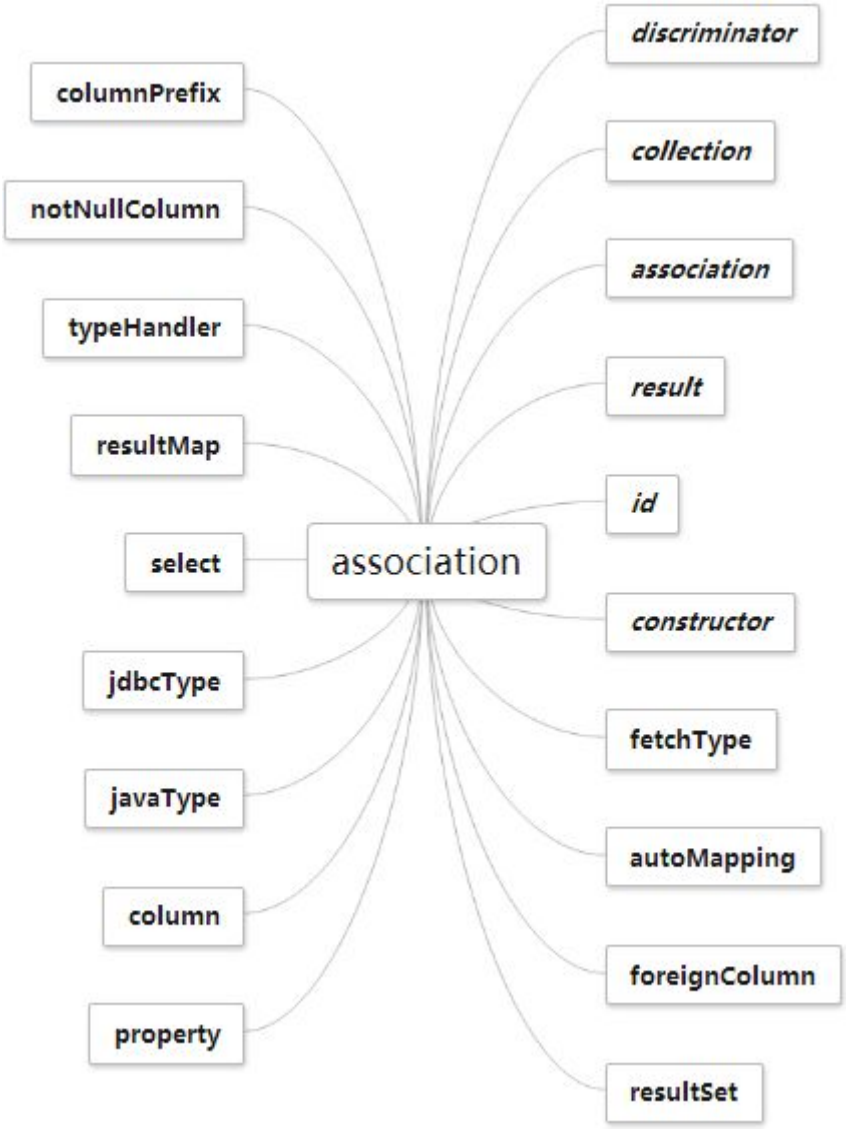

图 4.17 Association 模型图

与嵌套查询方式相关的属性有 column、select、fetchType：column 表示来自数据库的列名或者别名；select 是另一个映射语句的 ID，可以加载这个属性需要的复杂类型；fetchType 用于指定 lazy 加载还是 eager 加载。与嵌套结果方式相关的属性有 resultMap、columnPredix、notNullColumn、autoMapping：resultMap 是 ResultMap 的 ID，可以映射关联的嵌套结果到一个合适的对象图中，可以来调用

另一个查询语句，这允许联合多个表合成到一个结果集中；columnPrefix 用于指定列的前缀，当需要级联多个表时，为了防止出现列名冲突的现象，不得不使用别名；notNullColumn 用于指定子对象在对应列为空的时候是否需要创建对象，默认情况下，仅仅在不为空的时候才会创建；autoMapping 用于指定是否需要开启自动映射功能。

resultMap 中的 collection 的作用几乎和 association 是相同的，从代码模型来看，读了一个“ofType”属性，这个属性对于用来区分 JavaBean（或者字段）属性类型和集合所包含的类型来说是很重要的。

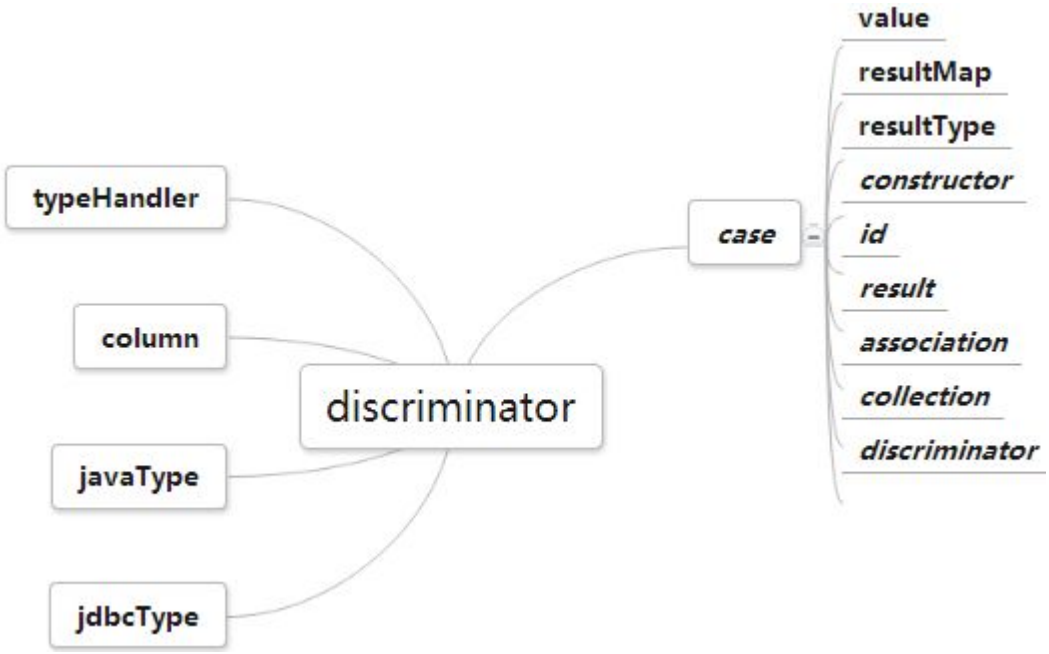

图 4.18 Discriminator 模型图

resultMap 中的 discriminator 作为鉴别器，可以实现动态映射关系信息的设置。有时候一个条数据库查询可能会返回包含了很多不同的数据类型的结果集，discriminator 元素就是来处理这个情况的，还包括的其他像类的继承层次结构的情况。它很像 JAVA 语言中的 switch 语句，具体的代码模型如图 4.18 所示。

discriminator 包含了四个属性和一个子元素。其中 column 属性指定了 SQL 查询结果的字段名或者字段别名，将用于 JDBC 的 resultSet.getString(columnName)；javaType 属性是一个 JAVA 类的完全限定名或者一个类型别名；jdbcType 属性表示当前数据库表格所支持的 JDBC 类型列表中的类型，JDBC 类型仅仅需要对插入、更新和删除操作中可能为空的列进行处理；typeHandler 属性用于指定类型处理器的全限定类名或者类型别名。discriminator

中的 case 子元素又包含了三个属性和六个子元素，六个子元素前面也都有介绍，这里就不重复了，三个属性分别是 value、resultMap、resultType，其中 value 值就是用来进行匹配的，具有排他性。

图 4.19 Cache&Cache-ref 模型图

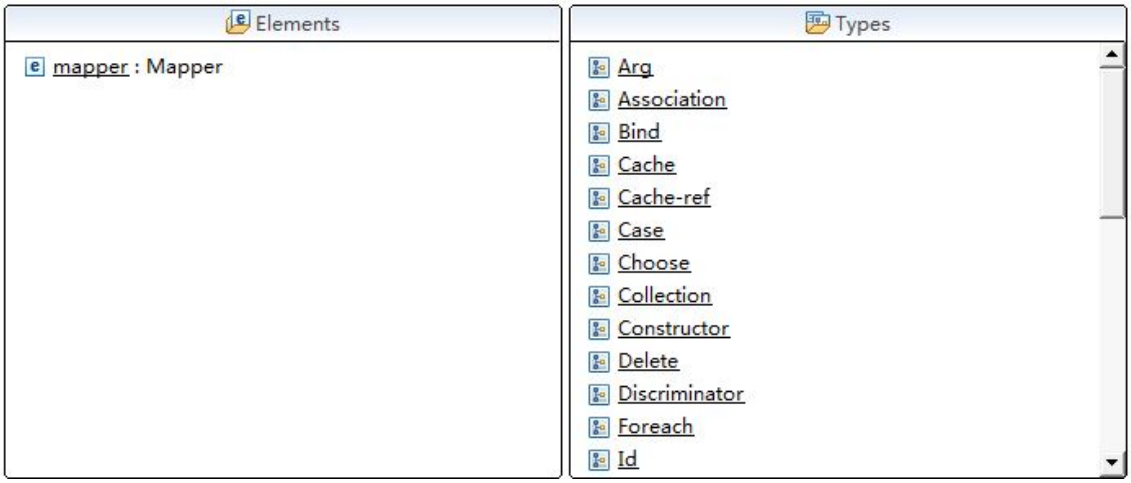

图 4.20 Mapper XML Schema

Mapper 映射文件顶层元素 cache 和 cache-ref 与缓存配置相关，具体模型如图 4.19 所示。cache 元素中 type 用于指定自定义的缓存方式。eviction 元素用于指定收回策略，有 LRU、FIFO、SOFT、WEAK 四种策略，默认是 LRU。LRU（最近最少使用）是移除最长时间未被使用的对象；FIFO（先进先出）是按照对象进入缓存的顺序来移除；SOFT（软引用）是基于垃圾回收器状态和软引用规则的移除策略；WEAK（弱引用）是一种相对于 SOFT 而言更为积极的一种移除策略。flushInterval 表示刷新间隔，单位为毫秒。size 表示引用数目，默认值是 1024。

readOnly 表示是否只读，只读缓存会带来性能上的优势，可读写缓存更为安全，因此默认是 false，也就是可读写缓存。cache-ref 元素用于引用另一个命名空间中的缓存，它只有一个 namespace 属性，就是用于指定命名空间的。

将代码模型的 XML Schema 在 Eclipse 中以 xsd 文件描述出来，这样可以方便查看与修改，如图 4.20 所示。然后用使用 Eclipse 直接把 Schema 转换成对应的 JAXB 类。

4.7 代码生成器的设计与实现

代码生成器主要通过识别代码模型中所包含的信息，来生成对应的目标代码。虽然，有时在目标环境中所有的代码都可以生成出来，但是更为常见的情况是，需要混合生成的代码和手写的代码。这使得代码生成技术不得不面临的一个难点就是：对于生成的代码和手写的代码需要进行区别对待。对于这个问题，使用代码生成技术最好能够遵循两个通用原则：不要修改生成的代码；将生成的代码与手写的代码严格分开[23]。

对于代码生成技术，必须保证它的权威性。如果手动修改代码生成的结果，当重新生成时，那些修改就会被覆盖。这样会给生成过程带来额外的工作，这样代码生成的许多好处就丧失殆尽了。当然如果使用代码生成技术仅仅是为了生成支架性质的代码，而不需要重复生成的话，也可以不考虑这个问题。还有一个例外，有时为了调试方便，插入一些跟踪语句，再次生成的时候，可以去除这些无相关代码。但是在其他情况下，最好还是不要修改任何生成的代码。这就有一个前提，就是生成的代码能够足够被使用者所信任，也就是保证它的权威性[24]。

将生成的代码和手写的代码分开的做法是很有意义的，可以将文件清楚的分为“全生成”和“全手写”两个部分，将生成的代码保存在源码树的单独分支中。生成的代码甚至可以不提交到源码库中，因为这些代码都是可以在构建过程中重新生成的。然而，并且不是所有的平台环境都支持这么做[24]。对于 Java 而言，如果想把一个类拆分成多个文件不是一件容易的事。

有一种做法，就是在类中划分区域，标记为生成或者手写，如果采用 Java 的话，还可以用注解来标记。但是这种方法会误导人们去编辑生成的代码，并且生成的代码也必须提交，这很有可能会弄乱版本控制的历史。

对于这个问题，有一个很不错的解决方案，就是采用继承来分离生成的代码和手写的代码。通过代码生成技术来生成一些父类，作为“核心类”，然后开发人员可以手写子类去继承它们，在子类中可以调用、增强或者重写生成的方法。

这样做的前提是需要放宽访问控制的权限规则，原本一些因为表示为私有的方法可能需要变成保护的，这样才能支持重写以及在子类中调用。但是这种方式所带来的灵活性是有目共睹的，放宽规则所需要付出的代价也很小[25]。

对于代码生成而言，还有一个不容忽视的问题，就是生成的代码可读性到底要怎么样，以及结构应该有多好？我与 Martin Fowler(《Domain-Specific Language》这本书的作者）的观点一致，应该让生成的代码如手写代码一样好：拥有清晰的变量名、良好的结构，以及遵循大多数应该遵循的好习惯[1]。当然也有例外，比如：如果需要花费很多时间才能拥有一个良好的结构，那么可以考虑把这些时间省下来，来做其他更重要的事；适当地存在重复的代码，但是可读性很好，也不需要过多的纠结于此[26]。

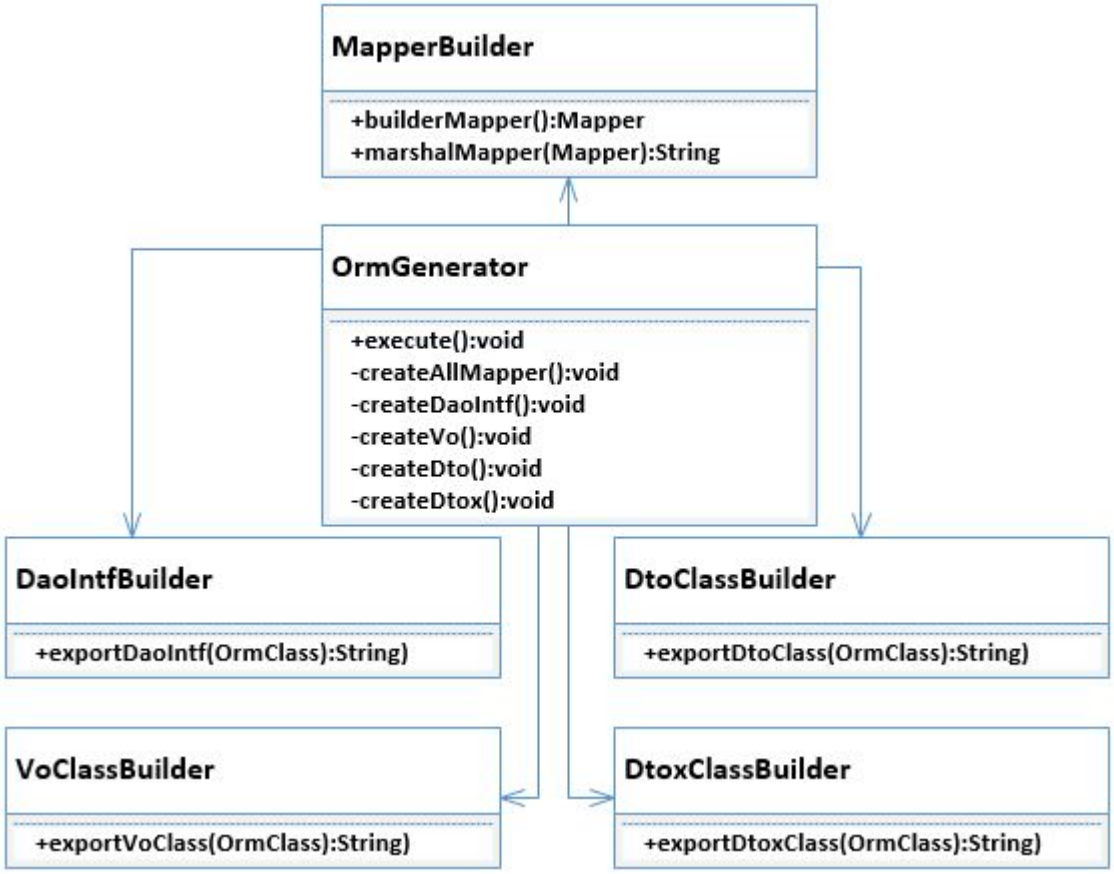

图 4.21 生成器相关类类图

本系统中设计了 MapperBuilder 类用于生成 MyBatis 的 Mapper 文件，DtoIntfBuilder 类用于生成 Dao 文件，VoClassBuilder 用于生成实体类 Vo 文件，DtoClassBuilder 用于生成传输对象类 Dto 文件，DtoxClassBuilder 用于生成 Dto 扩

展类文件，大致类图如图 4.21 所示。

只需要执行 OrmGenerator 类中的 execute 方法，execute 方法中会调用相应的 createXX 方法，然后 createXX 方法中会实例化相应的 XXBuilder 对象，最后执行相应的 exportXX 方法便可以生成需要的所有代码。其中就 MapperBuilder 特别一点，通过 JAXB 工具类中提供的 Marshaller 类来生成 Mapper 文件。

MapperBuilder 和 DaoIntfBuilder 生成的 Mapper 文件和 Dao 文件是成对出现的，文件名和类名的后缀都是 Dao。基于前面提到的原则，不建议用户修改。因为下次刷新代码的时候，修改的部分都会消失，需要用户自己重新修改代码，这跟不方便，也容易出错。如果用户想通过手写的方式扩展这部分代码，推荐通过增加 Dao 和 Mapper 的扩展文件方式来实现自定义的修改，推荐文件名以及类名后缀采用 Daox 的规则来命名。这样不仅将生成的代码和手写的代码进行了严格的分开，避免了很多麻烦。

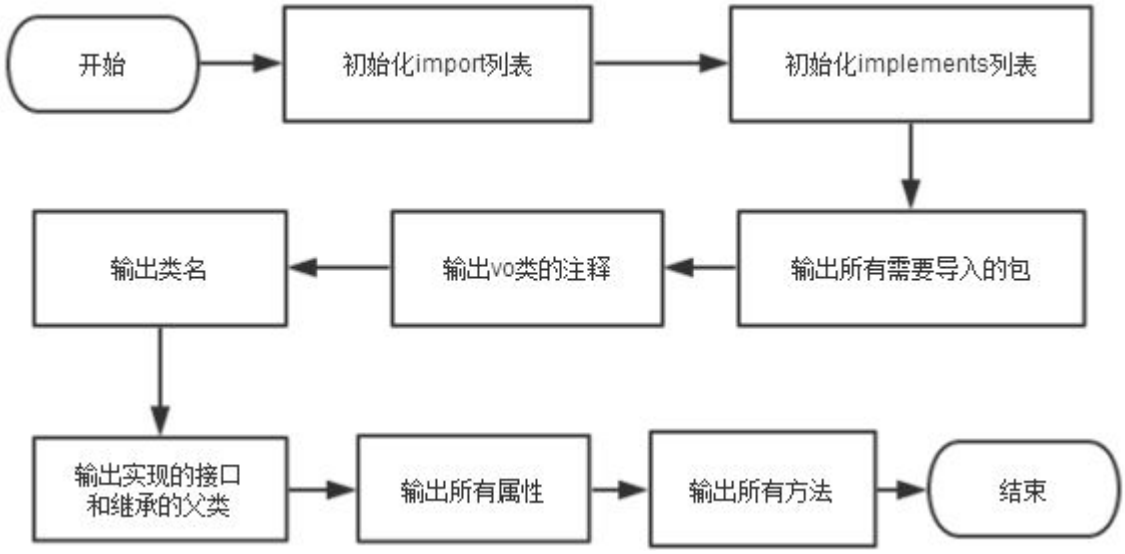

图 4.22 Vo 生成的过程图

VoClassBulider 生成的实体类 Vo 是完全根据数据表来生成的，内部属性和表中的列是一一对应的，同样不建议用户修改。VoClassBuilder 生成 Vo 的过程如图 4.22 所示。首先是初始化需要导入的包名的列表，这里只需要额外更加一个支持序列化的包 java.io.Serializable。然后是初始化需要实现的接口，这里需要 Cloneable 和 Serializable。接着就是输出代码，具体步骤已经在流程图中描述出来。其中属性不仅包括对应表中的所有列，还包括 Log4J 中的 Logger 和 Serializable 中的 SerializableUID。其中方法不仅包括属性的 getter 和 setter 方法，还增加了 clone

和 getEmptyCopy 方法。clone 用于完整的拷贝对象；getEmptyCopy 方法用于拷贝一个空对象，该空对象仅仅对主属性（对应于数据库表中为主键的属性，这里称之为主属性）的值进行拷贝。

传输对象 Dto 主要是依据用户在外部 DSL 脚本中描述的业务逻辑来生成的，OneToOne 的关联关系生成的属性就是 Vo，OneToMany 的关联关系生成的属性就是 Dtox 的 List 列表，所以同样不建议用户修改。DtoClassBuidler 类具体生成过程也跟 VoClassBuilder 类似，这里就不重复描述了。当业务复杂的时候，传输对象中，可能需要添加额外的属性或者方法。所以系统中所有关于传输对象的操作都是基于 Dtox 的，Dtox 继承自 Dto，Dtox 只会在没有的情况下生成，所以用户可以任意修改这个类，不用担心被覆盖掉。当然，用户想要增加实现的接口也是完全没有问题的。唯一需要注意的就是它是继承对应的 Dto 类的，这个不能修改，不然会出错，因为操作 Dtox 的时候很多属性和方法都是继承自 Dto 的。

4.8 本章小结

主要介绍了基于 MyBatis 的自适应代码生成工具的总体架构设计以及具体功能模块的设计与实现，包括 DSL 脚本、语法树、语义模型、数据模型、解析器、代码模型、代码生成器。

第 5 章 代码生成工具的使用说明与效果展示

本章开始介绍这个代码生成工具是如何使用的，以及如何高效地使用这个工具。现已经将代码传到 GitHub 上，项目命名为 MyBatisPioneer，简称 MBP，链接为：https://github.com/ourbeehive/MyBatisPioneer。所以后续中，为了方便起见，采用 MyBatisPioneer 来称呼该工具。下面将通过一个简单的案例来展示该工具。

5.1 案例描述

5.1.1 案例环境

该案例环境的系统为 Windows8 64 位，JDK 版本为 1.7 64 位，IDE 为 Eclipse Luna 4.4.1，Spring 框架版本为 3.2.1，MyBatis 框架版本为 3.2.8，DB2 数据库版本为 9.7.9，MySQL 数据库版本为 5.5.25，JUnit 版本为 4.11，Log4J 版本为 1.2.17，dbcp 版本为 1.4。

5.1.2 案例说明的范围

范围包括 Business Task 脚本的自动生成、实体类 Vo 的生成、传输对象类 Dto 的生成、传输对象扩展类 Dtox 的生成、数据访问接口 Dao 的生成、关系映射文件 Mapper 的生成、Dao 接口的使用。

表 5.1 数据库表信息表

表名	类型	列名	列类型	描述
T_USER	bigint(20)	USER_ID	用户的 ID	主键
	varchar(64)	NAME	用户姓名	
	varchar(256)	DESC	用户描述	
	timestamp	CRT_TM	创建时间	
	timestamp	UDP_TIME	更新时间	
T_ROLE	bigint(20)	ROLE_ID	角色的 ID	主键
	varchar(64)	NAME	角色名称	
	varchar(256)	DESC	角色描述	
	timestamp	CRT_TM	创建时间	
	timestamp	UPD_TM	更新时间	
T_USER_ROLE	bigint(20)	USER_ROLE_ID	用户角色关系 ID	主键
	bigint(20)	ROLE_ID	角色的 ID	外键
	bigint(20)	ROLE_ID	角色的 ID	外键
	timestamp	CRT_TM	创建时间	
	timestamp	UPD_TM	更新时间	

5.2 配置文件以及代码

在具体介绍工具使用之前，先在数据库中创建好要用到的数据库以及相应的表。数据库名为 MBP，MBP 中创建了 3 个表，分别是 T_USER、T_ROLE 和 T_USER_ROLE，表具体信息如表 5.1 所示。

5.2.1 DSL 脚本基本使用

正如前面所介绍的，用户是通过自定义的 DSL 脚本来使用 MyBatisPioneer 生成代码的。所以一开始当然是写 DSL 脚本。首先最好创建一个 properties 文件，用户存储数据库以及项目信息。当然直接写到脚本中也是可以的，但是看起来不直观，并且修改起来不方便。其中 datasource 开头的都是数据源相关的信息。ideWorkspace 是 IDE 的 Workspace 目录。artifact 开头的都是跟项目相关的信息：ormPrj 表示 ORM 项目名，这里将生成 Dao 以及 Mapper 文件；transObjPrj 表示 Data 层项目名，这里将生成 Vo、Dto 以及 Dtox 文件；baseDto 用于指定 Dto 类所需要继承的类。最下面的都是一些包名的指定，包括 Dao、Dto、Dtox 以及 Vo。这些信息都是根据自己实际开发情况进行修改，这里需要先创建好 ORM 层项目，取名为 MBP_ORM_JAR，以及 Data 层项目，取名为 MBP_TransObj_JAR。属性文件 MBP.AllTask.properties 如下所示：

```
datasource.driver=com.mysql.jdbc.Driver
datasource.connUrl=jdbc:mysql://localhost:3306/
datasource.user=root
datasource.password=123456
datasource.schema=mbp
ideWorkspace=E:/workplace_smart_luna/
artifact.srcPathName=/src/main/java/
artifact.ormPrj=MBP_ORM_JAR
artifact.transObjPrj=MBP_TransObj_JAR
artifact.comnLibPrj=MyBatisPioneer
artifact.baseDto=org.mbp.test.model.base.BaseDto
daoPkgPrefix=org.mbp.test.orm.dao.base
dtoxPkgPrefix=org.mbp.test.model.dto.ext
voPkgPrefix=org.mbp.test.model.vo
```

然后是全局的初始化脚本 Init Task，这个文件大多数都不需要修改，只需要在最下面的 target 中添加用户自定义的 Business Task 便可，其中 property file 就是用于引入属性文件的，全局初始化脚本 MBP.AllTasks.xml 如下所示:

```
<project name="allTasks" default="theTarget">
<property file="MBP.AllTask.properties" />
<target name="init">
<echo message="Run the target init..." />
<taskdef name="initCtx" classname="org.ourbeehive.mbp.task.InitCtx"></taskdef>
<initCtx>
<mapperProfile>
<allMapperProfile>
<dsParam driver="${datasource.driver}"
        connUrl="${datasource.connUrl}"
        user="${datasource.user}"
        password="${datasource.password}"
        schema="${datasource.schema}" />
<comnArtifact ideWorkspace="${ideWorkspace}"
              srcPathName="${artifact.srcPathName}"
              ormPrj="${artifact.ormPrj}"
              transObjPrj="${artifact.transObjPrj}"
              comnSrcPrj="${artifact.comnSrcPrj}"
              extendsBaseDto="${artifact.baseDto}"
              classConst="${artifact.classConstant}"
              attrConst="${artifact.fieldConstant}" />
</allMapperProfile>
</mapperProfile>
</initCtx>
</target>
<target name="theTarget" depends="init">
</target>
</project>
```

然后是最重要的业务脚本 Business Task，该脚本可以用 MyBatisPioneer 提供的工具来生成。在 org.ourbeehive.mbp.script 中有个 ScriptGenerator 类。只需要在 main 中编写简单的代码便可以自动生成 Business Task 脚本。这里有三个数据库表，所以需要实例化三个 BusinessScriptProperty 对象。然后分别设置参数信息，包括：文件名、表的前缀、表的基本名、对应实体类名、对应属性名、中文名、实体类上一级包名。ScriptGenerator 的 exportScript 是用于生成 Business Script 的，第一个参数就是 BusinessScriptProperty 的 List 对象，第二个参数表示是否需要覆盖原有的脚本文件，这里是 false，也就是说如果已经存在，就不在做任何操作。而 showCopyCodeInfo 是用于在控制台中打印需要拷贝到初始化脚本 Init Task 中的代码片段信息，具体位置上面已经说明，当然用户也可以自己填写这些代码。生成的 Business Task 部分代码如下所示：

```
BusinessScriptProperty roleProperty = new BusinessScriptProperty();
roleProperty.setFileName("MBP.T_ROLE.XML");
roleProperty.setPrefixOfTable("MBP.T_");
roleProperty.setBaseTable("ROLE");
roleProperty.setBaseClass("Role");
roleProperty.setBaseObj("role");
roleProperty.setCnName("角色");
roleProperty.setPkgUpperNode("user");
BusinessScriptProperty userProperty = new BusinessScriptProperty();
userProperty.setFileName("MBP.T_USER.XML");
userProperty.setPrefixOfTable("MBP.T_");
userProperty.setBaseTable("USER");
userProperty.setBaseClass("User");
userProperty.setBaseObj("user");
userProperty.setCnName("用户");
userProperty.setPkgUpperNode("user");
BusinessScriptProperty userRoleProperty = new BusinessScriptProperty();
userRoleProperty.setFileName("MBP.T_USER_ROLE.XML");
userRoleProperty.setPrefixOfTable("MBP.T_");
userRoleProperty.setBaseTable("USER_ROLE");
```

```
userRoleProperty.setBaseClass("UserRole");
userRoleProperty.setBaseObj("userRole");
userRoleProperty.setCnName("用户角色关系");
userRoleProperty.setPkgUpperNode("user");
List<BusinessScriptProperty> scriptProperties = new ArrayList<>();
scriptProperties.add(userProperty);
scriptProperties.add(roleProperty);
scriptProperties.add(userRoleProperty);
ScriptGenerator gen =new ScriptGenerator();
gen.exportScripts(scriptProperties, false);
gen.showCopyCodeInfo(scriptProperties);
```

之前实例化了三个BusinessScriptProperty对象，所以会生成三个Business Task脚本，三个脚本都差不多，分别对应三个不同的数据库表。其中针对T_USER表生成的MBP.T_USER.XML脚本如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="genOrm" default="theTarget">
<!-- Attributes used by this task -->
      <property name="baseClass" value="User"/>
      <property name="baseObj" value="user"/>
      <property name="baseTable" value="USER"/>
      <property name="cnName" value="用户"/>
      <property name="prefixOfTable" value="MBP.T_"/>
      <property name="pkgUpperNode" value="user"/>
<property name="daoIntfName"
value="${daoPkgPrefix}.${pkgUpperNode}.I${baseClass}Dao" />
<property name="dtoxClassName"
value="${dtoxPkgPrefix}.${pkgUpperNode}.${baseClass}Dtox" />
<property name="voClassName"
value="${voPkgPrefix}.${pkgUpperNode}.${baseClass}Vo" />
<property name="rootTableName" value="${prefixOfTable}${baseTable}" />
<!-- The task definition -->
```

```
<target name="theTarget">
<taskdef name="genOrm" classname="org.ourbeehive.mbp.task.GenOrm">
<classpath refid="dependencyFinder" />
</taskdef>
<genOrm>
<mapperProfile>
<singleMapperProfile>
<mapperArtifact mapperNs="${daoIntfName}">
</mapperArtifact>
<insertConfig className="${voClassName}" tableName="${rootTableName}"
comments="新增${cnName}" enableSelectKey="true" />
<updByPKConfig className="${voClassName}" tableName="${rootTableName}"
comments="更新${cnName}" />
<updBySqlConfig className="${dtoxClassName}"
tableName="${rootTableName}" comments="更新${cnName}" />
<delByPKConfig className="${voClassName}" tableName="${rootTableName}"
comments="删除${cnName}" />
<delBySqlConfig className="${dtoxClassName}"
tableName="${rootTableName}" comments="删除${cnName}" />
<resultMapConfig className="${dtoxClassName}"
tableName="${rootTableName}" dfltOrderBy="updTm" comments="查询
${cnName}">
</resultMapConfig>
<!-- TODO: Add your business rules. -->
</singleMapperProfile>
</mapperProfile>
</genOrm>
</target>
</project>
```

接下来就可以生成代码了，在 Eclipse 中，选择菜单栏上面的 Run 中的 Run Configurations，New 一个 Java Application，取名为 Pioneer，Project 中填写

MyBatisPioneer，Main Class 中填写 Ant 工具中的 Main 类 org.apache.tool.ant.Main，Arguments 中填写初始化脚本 MBP.AllTasks.xml 的路径 -f "E:\git_workspace\MyBatisPioneer\test\org\ourbeehive\mbp\script\MBP.AllTasks.xml"，最后点击 Run 即可。

5.2.2 实体类 Vo

MBP_TransObj_JAR 项目中一共会生成三个 Vo 实体类，包括 UserVo、RoleVo 以及 UserRoleVo。这里以 UserVo 为例，具体见表 5.7 中示例代码。其中可以看到 UserVo 实现了 Cloneable 以及 Serializable 接口，里面的各个属性都是对应于数据库表格中的字段的。这里的代码如果修改了，下次重新生成代码的时候就会将自己覆盖掉，所以不推荐修改。因为所有附加的操作都是可以在 Dto 以及 Dtox 中进行的，所以这个文件也没有修改的必要。生成的 UserVo 类代码如下所示:

```
package org.mbp.test.model.vo.user;
import java.sql.Timestamp;
import java.io.Serializable;
public class UserVo implements Cloneable, Serializable {
    private static final long serialVersionUID = 1L;
    private Long userId = null;
    private String name = null;
    private String desc = null;
    private Timestamp crtTm = null;
    private Timestamp updTm = null;
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
    public UserVo getEmptyCopy() {
        UserVo newUserVo = new UserVo();
        newUserVo.setUserId(this.userId);
        return  newUserVo;
    }
    public void setUserId(Long userId) {
        this.userId  =  userId;
```

```
    }
    public Long getUserId() {
        return userId;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setDesc(String desc) {
        this.desc = desc;
    }
    public String getDesc() {
        return desc;
    }
    public void setCrtTm(Timestamp crtTm) {
        this.crtTm = crtTm;
    }
    public Timestamp getCrtTm() {
        return crtTm;
    }
    public void setUpdTm(Timestamp updTm) {
        this.updTm = updTm;
    }
    public Timestamp getUpdTm() {
        return updTm;
    }
}
```

5.2.3 传输对象 Dto 以及 Dtox

同样 MBP_TransObj_JAR 项目中也会生成三个 Dto 以及三个 Dtox 文件，

UserDto.java、RoleDto.java、UserRoleDto.java、UserDtox.java、RoleDtox.java 以及 UserRoleDtox.java。这里以 UserDto 以及 UserDtox 为例，具体见表 5.8 以及 5.9 中示例代码。UserDto 会继承之前在脚本中所指定的基类，这里是 BaseDto。Dtox 仅仅是继承了 Dto，用户可以根据自己的需要修改 Dtox 类。Dto 文件最好也不好修改，因为这里的属性是根据 Business Task 文件中配置信息生成的。如果修改了 Dto，MyBatisPioneer 下次生成 Dto 的时候也不会全部覆盖，而是采用 append 的方式进行添加的。不过更为推荐的方式是在 Dtox 中做一些人工操作，以免将人工手写的代码与生成的代码混乱，造成不必要的麻烦。生成的 UserDao 类代码如下所示:

```
package org.mbp.test.model.dto.base.user;
import org.mbp.test.model.base.BaseDto;
import org.mbp.test.model.vo.user.UserVo;
public class UserDto extends BaseDto {
	private static final long serialVersionUID = 1L;
	protected UserVo userVo = null;
	public void setUserVo(UserVo userVo) {
		this.userVo = userVo;
	}
	public UserVo getUserVo() {
		return userVo;
	}
}
```

5.2.4 Dao 层和 Mapper 关系映射文件

MBP_ORM_JAR 项目中会生成三个 Dao 以及三个 Mapper 文件，包括：IUserDao.java、IRoleDao.java、IUserRoleDao.java、IUserDao.xml、IRoleDao.xml 以及 IUserRoleDao.xml。。所有的基础代码都以继承生成，包括基础的增删改查、批量删除、批量插入、计数查询以及批量查询等。为了方便查看，代码中的注释已经被我删掉。Mapper 关系映射文件的内容比较长，这也就不贴出来了，生成的 IUserDao 接口代码如下所示:

```
package org.mbp.test.orm.dao.base.user;
```

```
import java.util.List;
import java.util.Map;
import org.mbp.test.model.base.MapperSqlClause;
import org.mbp.test.model.dto.ext.user.UserDtox;
import org.mbp.test.model.vo.user.UserVo;
public interface IUserDao {
    public Integer countUserBySql(MapperSqlClause param0);
    public UserDtox selectUserByPKLong(Long param0);
    public UserDtox selectUserByPKDtox(UserDtox param0);
    public List<UserDtox> selectUserBySql(MapperSqlClause param0);
    public int insertUserAll(UserVo param0);
    public int insertUserSelective(UserVo param0);
    public int insertUserAllBatch(List<UserVo> param0);
    public int insertUserAllBatchWithoutPk(List<UserVo> param0);
    public int updateUserAllByPK(UserVo param0);
    public int updateUserAllByPKTmLck(Map<String, UserVo> param0);
    public int updateUserSelectiveByPK(UserVo param0);
    public int updateUserSelectiveByPKTmLck(Map<String, UserVo> param0);
    public int updateUserAllBySql(UserDtox param0);
    public int updateUserSelectiveBySql(UserDtox param0);
    public int deleteUserByPKLong(Long param0);
    public int deleteUserByPKVo(UserVo param0);
    public int deleteUserByPKTmLck(UserVo param0);
    public int deleteUserByPKBatch(List<Long> param0);
    public int deleteUserBySql(MapperSqlClause param0);
}
```

5.3 MapperSqlClause 详解

分页查询比较特殊，不同的数据库分页查询的 SQL 语句也不一样。DB2 中分页查询需要两个参数，一个表示开始行，另一个表示结束行；MySQL 中分页查询也需要两个参数，一个表示开始行，另一个表示需要查询的总行数。这使得

关于分页查询的代码会显得略微复杂一点。

由于上面这个原因，MyBatisPioneer 增加了一个名为 MapperSqlClause 的类，用户需要自己将这个类拷贝到项目之中。在 MapperSqlClause 中；startRow 表示开始 DB2 的开始行；endRow 表示 DB2 的结束行；limitBegRow 表示 MySQL 的开始行，limitRowCnt 表示 MySQL 查询的行数；pageNbr 表示当前页数；cntInPage 表示每页行数；dfltMaxRow 表示最大行数，默认是 1000 行。针对 DB2 和 MySQL，MyBatisPioneer 将采取不同的解决方案。

MyBatisPioneer 支持在代码中拼接 where 子句，所以在 MapperSqlClause 类中添加了 String Buffer 类型的 whereCluase 属性，开发人员将 where 子句赋值到 whereCluase 属性中，MyBatisPionneer 将会自动将这部分代码添加到 Mapper 映射文件之中。

MyBatisPioneer 还支持用户在代码中配置查询时排序的 SQL 子句，所以在 MapperSqlClause 类中提供了 StringBuffer 类型的 orderByClause 属性，开发人员可以利用这个属性来配置列表查询时排序的规则。

之前提到开发人员在配置文件中可以指定 Dto 的父类，但是在这里要进行补充说明，为了能够使用分页查询、where 子句拼接、order 子句拼接，Dto 的父类中一定要有 MapperSqlClause 的属性，而且必须取名为 mapperSqlClause。本来想采用配置的方式，另一块更加灵活，但是会增加配置的复杂度。

MapperSqlClause 的出现增加了使用 MyBatis 的复杂度，但是同时带来的灵活性也是极高的。开发人员可以根据自己的需求来修改 MapperSqlClause，但是建议采取继承的方式进行修改，避免影响其他地方。比如用户想要在 MapperSqlClause 中添加用户信息的属性，在拼接 SQL 语句的时候，可能会用到用户的信息，这完全是可以的。

5.4 Dao 层接口使用

上面已经生成了所有底层的代码，现在需要做的就是调用他们。这里通过 SpringJUnit4 来描述调用的过程，同样可以用于测试代码的正确性，验证生成的代码能否正常使用。基础的代码以及配置这里就不展示了，不然又是一大堆表格或者图片，会显得过于冗长。并且这就是使用 MyBatis 的过程，这也不是本文所要描述的，应该将更多的注意力放在 MyBatisPioneer 上。

对于插入操作，这里生成了很多接口，比如调用 insertAll 接口。那可以采用如下代码来进行插入操作，运行结果这里就不截图了。这个操作代表在 User 表中

插入一个名字为 Name999，描述信息为 Desc999，创建时间和更新时间都是当前时间的行。

```
UserVo userVo = new UserVo();
userVo.setName("Name999");
userVo.setDesc("Desc999");
userVo.setCrtTm(DateUtils.currentTimestamp());
userVo.setUpdTm(DateUtils.currentTimestamp());
userDao.insertUserAll(userVo);
```

对于删除操作，这里同样提供了多种删除接口。比如要根据主键来删除用户，那么就可以采用如下代码进行删除操作，其中传入的参数就是主键的值。这里的意思就是从 User 表中删除主键为 5 的行。

```
userDao.deleteUserByPKLong(new Long(5));
```

对于更新操作，比如要根据主键更新用户信息，那么就可以采用下面的代码进行更新，其中传入的参数就是修改过的 User 对象。这个操作将把 User 表中主键为“1”的行的描述信息改成“Update by Tester”，更新时间字段的值改为当前时间。

```
UserVo userVo = new UserVo();
userVo.setUserId(new Long(1));
userVo.setDesc("Update by Tester");
userVo.setUpdTm(DateUtils.currentTimestamp());
userDao.updateUserAllByPK(userVo);
```

对于查询操作，比如说要根据 NAME 字段来查询 User 表，那么就需要根据拼接 where 语句的方式来查询，所以就可以采用如下代码进行查询。这个操作将查询出 User 表中所有 NAME 字段是以“Name”开头的行。

```
MapperSqlClause sqlClause = new MapperSqlClause();
StringBuffer whereClause = new StringBuffer();
whereClause.append("MBP__T_USER.NAME = 'Name%'");
sqlClause.setWhereClause(whereClause);
List<UserDtox> userDtox = userDao.selectUserBySql(sqlClause);
```

```
int count = userDao.countUserBySql(sqlClause);
```

上面这种在代码中拼接 SQL 的方式,似乎不是很友好,但是功能也十分强大,为数据库操作提供了更大的灵活性。然而一切复杂的查询,如果也采用这种方式,就真的不友好了,所以 MyBatisPioneer 提供了另一种更为友好的方式来执行这些复杂操作。

5.5 DSL 脚本复杂查询

正如第 4 章所描述的 DSL 脚本,MyBatisPioneer 的外部 DSL 脚本添加了大量的语法,足够广大开发人员用于日常开发。

对于 User 表,如果说现在需要查询 User 信息列表,但是创建日期早于 2016 年 3 月 26 号的用户都采取过滤操作,那么同样可以采用上面提到的拼接 SQL 语句的方式来操作。如果对于分页查询、根据主键查询都要进行这个过滤操作,那么就需要同时修改多个地方的代码。并且如果这些过滤条件一多,那么代码就会显得混乱不抗,难以看懂。可以采取一种更为优雅的方式来进行过滤,如下所创建一个查询操作。

```
<resultMapConfig selectStmt="get${baseClass}ListFilterCrtTime"
className="${dtoxClassName}" tableName="${rootTableName}"
comments="查询${cnName}过滤无效时间">
  <filterConfig attribute="crtTm" comparator="${greater}" value="'2016-3-25'" />
</resultMapConfig>
```

然后重新运行 MyBataisPioneer 去生成代码,这时会发现在 IUserDao 中多了四个方法。

```
public UserDtox getUserListFilterCrtTimeByPKLong(Long param0);
public UserDtox getUserListFilterCrtTimeByPKDtox(UserDtox param0);
public List<UserDtox> getUserListFilterCrtTimeBySql(MapperSqlClause param0);
public Integer countGetUserListFilterCrtTimeBySql(MapperSqlClause param0);
```

如果需要查询所有 User 以及数量,就可以采用如下的代码进行操作,同时还能采用拼接 SQL 的方式,添加其他过滤条件。

```
List<UserDtox> userDtoxs = userDao.getUserListFilterCrtTimeBySql(sqlClause);
int count = userDao.countGetUserListFilterCrtTimeBySql(sqlClause);
```

在实际项目中，常常会给数据库表添加一些特别的字段，比如表示数据否有效，表示类别等等。如果说有个字段 flag，如果它的值为 0 表示无效，那么业务里很多的查询中，都不应该出现这个字段所在的行数据；如果说还有一个字段 category，表示类别信息，如果只需查询类别是 1 的行，则必须在 where 语句中拼接这些过滤条件。当这些条件出现在代码中的时候，就会很不友好，显得冗余笨重。将它们直接在 Business Task 脚本中添加一个查询操作便可。

```
<filterConfig attribute="editFlag" comparator="${less}${greater}" value="'0'" />
<filterConfig begin="AND" attribute="catCd" comparator="=" value="'1'" />
```

但是上面描述的一些案例是不能够满足实际开发中的需求的，因为在实际开发中常常会用到大量的多表级联查询。后面将举例说明如何运用 MyBatisPioneer 来实现多表级联查询代码的生成。

比如说想查询用户以及他所具有的角色，那么这就是一个普遍的涉及三个表的级联查询。可以通过 USER 表中 USER_ID 来匹配 USER_ROLE 表中的 USER_ID，来找到对应的 ROLE_ID，再通过 ROLE_ID 找到 ROLE 表对应的行。那么就可在 Business Task 脚本中添加一份查询操作。

```
<resultMapConfig selectStmt="get${baseClass}WithRole"
className="${dtoxClassName}" tableName="${rootTableName}"
comments="查询${cnName}列表和角色信息">
    <oneToMany listOfSon="userRoleList" fatherAttr="userId" sonAttr="userId">
     <resultMapConfig className="org.mbp.test.model.dto.ext.user.UserRoleDtox"
tableName="${prefixOfTable}USER_ROLE">
                <oneToOne refToSon="role" fatherAttr="roleId" sonAttr="roleId">
                    <resultMapConfig tableName="${prefixOfTable}ROLE">
                    </resultMapConfig>
                </oneToOne>
            </resultMapConfig>
     </oneToMany>
</resultMapConfig>
```

这里在一个 resultMapConfig 中嵌套了一个 oneToMany 来查询 USER_ROLE 表，接着在 oneToMany 中嵌套一个 ontToOne 来查询 ROLE 表，基于对象的操作，

不是生硬的 SQL 语句，与 SQL 语句相比逻辑十分清晰，维护起来也很方便。如果需要加入 flag 的过滤条件也是可以的，加入到 resultMapConfig 中即可。

```
<filterConfig attribute="flag" comparator="${less}${greater}" value="'0'" />
```

如果说不需要 User 的描述信息，那么可以不查询出 DESC 字段的信息，在 resultMapConfig 中通过 excludedAttr 属性来进行排除配置。

```
<excludedAttr name="desc" />
```

如果说只需要 Role 的名称信息，其他的描述信息和时间信息都不需要，那么可以利用 includedAttr 属性来配置。当 includedAttr 和 excludedAttr 同时作用于某个字段（属性）时，以 includedAttr 为主。

```
<includedAttr name="name" />
```

这样，生成的 UserDto 中会增加一个属性 userRoleList，它是一个 UserRoleDtox 的 List 对象，以及生成对应的 get 和 set 方法。UserDtox 是不会变化的，因为 UserDtox 只有在没有生成的情况下，才会生成。

```
protected List<UserRoleDtox> userRoleList = null;
```

最后要调用 Dao 接口的时候，就更简单了。查询 ID 为 2 的用户信息，同时也查询出了用户具有的角色信息。

```
UserDtox userDtox = userDao.getUserWithRoleByPKLong(new Long(2));
```

如果想获取角色列表，可以调用 userDtox 的 getUserRoleList()方法获取 UserRoleDtox 的列表，然后调用 getRole 过去对应的角色。如果角色这么操作过于复杂，可以在 UserDtox 中写一个 getRoles 方法来封装获取角色列表的操作细节。如果频繁调用，甚至可以利用缓存，这都是用户自己可以选择性的实现的。

5.6 自适应性

MyBatisPioneer 能够适应数据库的替换以及表结构的修改。对于适应表结构的这点，很好理解。因为实体类 Vo 是根据数据库表的信息生成的，一旦数据库发生了变化，重新刷新实体类的时候，也会做出相应的改变，mapper 文件中 resultMap 也会最出相应的调整。只要不是在条件判断时，涉及到之前表结构修改的地方，开发人员是不需要做其他什么的。尽管在条件判断时，要用到，也只要修改 Business Task 中的业务逻辑即可，这也是很方便的事。

不同数据库的 SQL 或多或少都会有一些区别，尽管区别不是很大。但是如果数据库改变，那么要将这些有区别的地方，逐个修改过来，这工作量是巨大的，

也很容易出错。MyBatisPionner 根据 Init Task 中开发人员所填写的数据源信息，自动识别数据库，然后生成对应的 SQL。由于时间有效，目前 MyBatisPionner 仅仅支持 DB2 和 MySQL。

下面针对前面的案例，就将 MySQL 数据库修改成 DB2。首先在 DB2 上创建一个名为 SQLDB 的数据库，然后跟上面一样分别创建 T_USER、T_ROLE、T_USER_ROLE 三个数据库表。

现在开始改写 Init Task 的数据源信息配置，访问不同类型数据库的 JDBC 连接 URL 和驱动肯定不一样。这里我使用 Bluemix 上的 DB2，分配给我的数据库名是 SQLDB，而 schema 是 USER05755，具体配置如表 5.2 所示。MySQL 中的 shcema 和 database 是一个概念，所以使用 MyBatisPioneer 时，datasource.connUrl 就不指定 database（或者 schema），而在 datasource.schema 中指定，这样才能同时访问一个连接下的多个 database（或者 schema）。而 DB2 中不仅有 database，还有 schema，一个 database 里面有多个 schema。所以在连接 DB2 的时候，一般在 datasource.connUrl 中指定 database，在 datasource.schema 中指定 schema，datasouce.schema 是可以同时指定多个 schema 的。

表 5.2 数据源信息配

采用 DB2 时，数据源配置信息
datasource.driver=com.ibm.db2.jcc.DB2Driver datasource.connUrl=jdbc:db2://5.10.125.192:50000/SQLDB datasource.user=user05755 datasource.password=xEiQpRNiugEo datasource.schema=USER05755
采用 MySQL 时，数据源配置信息
datasource.driver=com.mysql.jdbc.Driver datasource.connUrl=jdbc:mysql://localhost:3306/ datasource.user=root datasource.password=123456 datasource.schema=MBP

因为 schema 发生了改变，所以 Business Task 中的数据库表的 schema 也要相应的修改。因为案例中统一采用 prefixOfTable 来表示数据库表的前缀，所以将 USER 表、ROLE 表、USER_ROLE 表中先的 “MBP._T” 修改为 “USER05755.T_”。

重新运行 MyBatisPioneer，前面已经介绍过如何运行。

这时生成的 Vo、Dto、Dtox、Dto 都是一样的，不会有变化，唯一有变化的就是 mapper 文件。比如原先的 MBP 都会替换成 USER05755; 分页查询的时候语句也不一样，如图 5.3 所示；插入后返回主键值的语句也不一样，如表 5.4 所示。

表 5.3 分页查询部分代码

MySQL 版本中，分页查询部分代码
<choose> <when test="limitBegRow == -1 or limitRowCnt == -1"> <![CDATA[limit ${dfltMaxRow}]]> </when> <otherwise> <![CDATA[limit #{limitBegRow},#{limitRowCnt}]]> </otherwise> </choose>
DB2 版本中，分页查询部分代码
<choose> <when test="startRow == -1 or endRow == -1"> <![CDATA[fetch first ${dfltMaxRow} rows only]]> </when> <otherwise> <![CDATA[ ) where RN between #{startRow} and #{endRow}]]> </otherwise> </choose>

表 5.4 插入操作部分代码

MySQL 版本中，插入操作部分代码
<insert id="insertUserAll" parameterType="org.mbp.test.mysql.model.vo.user.UserVo"> <selectKey resultType="java.lang.Long" keyProperty="userId" order="AFTER">

续表 5.4

MySQL 版本中，插入操作部分代码
<![CDATA[SELECT LAST_INSERT_ID()]]> </selectKey> <include refid="sql:insertUserAll"/> </insert>
DB2 版本中，插入操作部分代码
<insert id="insertUserAll" parameterType="org.mbp.test.db2.model.vo.user.UserVo"> <selectKey resultType="java.lang.Long" keyProperty="userId" order="AFTER"> <![CDATA[values (identity_val_local())]]> </selectKey> <include refid="sql:insertUserAll"/> </insert>

可以通过调用 Dao 接口测试效果，MyBatisPioneer 底层封装了这些不同的地方，这对于开发人员来说是透明的，开发人员无需关心这些变化。如果数据库的信息没有变化，仅仅是换了一个数据库而已，那么只要修改数据源信息就行了，这是相当方便的。

5.7 本章小结

本章针对具体的案例，对该工具的使用进行简单的说明，包括如何配置、如果修改、如果使用接口、如果构建复杂查询等方面。本章可以作为该工具的快速入门教程，读者可以从中认识到该工具的强大之处。

第 6 章 总结与展望

6.1 工作总结

为了解决 MyBatis 开发中带来的诸多不便，本文通过研究领域特定语言、模型驱动开发、代码生成等技术，为开发人员提出一个全新的解决方案，一种基于 MyBatis 的自适应代码生成工具 MyBatisPioneer。该工具目的是降低开发门槛，加快开发速度，统一编程规则，节约开发成本，提升程序质量。MyBatisPioneer 适用于所有采用 MyBatis 和 Spring 框架来开发的广大开发人员。开发人员仅仅需要编写关于自己业务逻辑的 DSL 脚本，便可以轻松地生成 ORM 层的代码。包括 Dao 层、Model 层、Mapper 映射文件。不仅支持简单的 CRUD（插入、查询、更新、删除）操作，还支持分页查询、多表级联查询，甚至适应数据库的替换以及表结构的修改。解决了采用 MyBatis 开发所带来的诸多问题，开发人员可以完全不知道 MyBatis 的使用细节，便可以参与项目开发。大量的代码都已经自动生成，这极大地提高了开发的效率。因此开发人员不用过度地关心底层的细节，将更多的精力放在如何解决业务问题上，以满足用户的需求和提高用户体验作为主要关注点。这样使得系统更加完善，更好地符合用户需求。

论文通过分析领域特定语言、模型驱动开发、代码生成技术等方面，证实了开发 MyBatisPioneer 的可行性，并且通过具体的需求分析明确了工具的需求。随后论文对总体框架设计以及各个功能模块的设计与实现进行了详细的描述，展示了该工具的运作方式以及原理。最后通过一个简单的案例展示了该工具的使用方式和使用效果，并且强调了使用中应该注意的问题。

MyBatisPioneer 仅仅是代码生成工具的一小步，这也就是取名为 Pioneer 的原因。现在仍然存在一些问题需要解决，可以做得事情还有很多。开发人员不应该完全依赖 MyBatisPioneer 来进行开发，毕竟 MyBatisPioneer 的目的是减少开发人员的工作量。针对特定的业务需求，MyBatisPioneer 生成的 SQL 语句不一定适合。开发人员需要灵活使用 MyBatisPioneer，并且结合 MyBatis 的特点来进行高效的开发。

6.2 今后展望

现在生成的代码中，采用 SQL 语句拼接的方式查询，仅仅支持在最后的 where 语句中拼接，而不支持在 join 时的条件拼接。这个问题目前可以通过重写 Dao、

Mapper 文件以及扩展 MapperSqlClause 类得到解决。关于数据库，现在仅仅支持 MySQL 以及 DB2，今后还需要针对 Oracle、PostGre、Sqlite 等关系型数据库进行整合。MyBatisPioneer 目前的扩展性不好，想扩展或者修改功能只能修改源码。比如用户无法定制自己的输出格式，无法定制自己的代码风格，无法定制代码中的注释信息以及无法根据需求替换一些列名等等。今后可以在 MyBatisPioneer 中预留一些接口，令用户自己去实现这些接口，在配置文件中指定接口的实现类，来完成定制操作。

通过 MyBatisPioneer 的成功实现，证明了通过底层数据出发，生成代码的可行性。现在各种应用程序层出不穷，很多公司都想做自己的网站以及手机 APP，然而缺乏专业知识，必须支付昂贵的开发费用给其他公司，或者其他外包的方式。通过简单的分析就可以发现，大多数的应用程序很多地方都是相似的，很多基础增删改查的操作都是可以自动生成的，根本不需要花费过多的时间成本和劳动成本。比如基本表的增删改查、用户管理、权限管理、登录注销操作和文件上传下载等等都是可以生成的，只要能提供开发人员一些方便的修改方式便可。对于一些复杂的业务逻辑，可以效仿 MyBatisPioneer 的外部 DSL，采用类似于 Business Task 的方式来让开发人员自己描述这些业务逻辑，从而自动生成代码。

因此，在今后的工作和研究中，我会继续完善 MyBatisPioneer，投入更多的时间与精力到代码生成工具之中，为人们提供一种更为简单的开发方式。

参考文献

[1] MyBatis Organization.MyBatis Introduction[EB/OL].http://mybatis.org/mybatis-3,2016.03.29.

[2] Liu Bin,Wang Zui.Application of Office Automation Based on SSH Framework[J].Computer Technology and Development,2010(1):39-40.

[3] Zhang Dandan,Wei Zhiqiang,Yang Yongquan.Research on Lightweight MVC Framework Based on Spring MVC and Mybatis[C].Hangzhou:Computational Intelligence and Design,2013:350-353.

[4] MyBatis.MBG[EB/OL].http://mybatis.org/generator/index.html,2014.01.21.

[5] Martin Fowler.Domain Specific Language[M].Chicago:Addison-Wesley Professional,2010:27-143.

[6] 赵卫东,刘永红.一种领域特定语言的研究与实现[J].程度大学学报,2013,32(2):142-144.

[7] 郁天宇.面向最终用户的领域特定语言的研究[D].博士学位论文,上海交通大学,2014.

[8] Xavier Amatriain,Pau Arumi.Frameworks Generate Domain-Specific Languages: A Case Study in the Multimedia Domain[J].IEEE Transactions on Software Engineering,2011,37(4):544-558.

[9] Dimitar Asenov,Peter Muller.Customizing the visualization and interaction for embedded domain-specific languages in a structured editor[C].San Jose:IVisual Languages and Human-Centric Computing,2013:127-130.

[10] 张迎春,黄林鹏.upDSL:一种描述动态更新策略的领域特定语言[J].微电子学与计算机,2008,25(10):34-36.

[11] Fabian Gilson,Vincent Englebert.A domain specific language for stepwise design of software architectures[C].Lisbon:Model-Driven Engineering and Software Development,2014:67-68.

[12] Zeki Bozkus,Taner Arsan.Big Data Platform Development with a Domain Specific Language for Telecom Industries[C].Magosa:High Capacity Optical Networks and Enabling Technologies,2013:116-120.

[13] Viana M C, Penteado R A D.Domain-Specific Modeling Languages to improve framework instantiation[J].Journal of Systems and Software,2013,86(12):3123-3139.

[14] Visic N,Fill H G, Buchmann R A,*et al*.A domain-specific language for modeling method definition From requirements to grammar[C].Athens:Research Challenges in Information Science,2015:286-297.

[15] Kyle L,Jeffrey S.Aspen:A domain specific language for performance modeling[C]. Salt Lake City:High Performance Computing, 2012:1-11.

[16] 孙宏伟,张树生,周竞涛,等.基于模型驱动的 XML 与数据库双向映射技术[J]. 计算机工程与应用, 2004,38(4):25-27.

[17] 刘美健.基于模型驱动的海洋环境数据平台研究与应用[J]. 海洋通报, 2014(2):193-198.

[18] 王海林.模型驱动下的数据库自动生成[J].计算机技术与发展,2011,21(8): 173-176.

[19] 黄钦.基于开源框架的通用代码生成引擎设计与实现[D].博士学位论文,电子科技大学, 2007.

[20] 张静,孔芳,杨季文.一个数据模型驱动的代码生成工具的设计与实现[J].计算机应用与软件,2011,27(11):151-153.

[21] 范玥,王淑玲.一种动态软件体系结构下的代码生成方法[J].小型微型计算机系统,2013,34(3):515-519.

[22] Timothy Richards, Edward K,Palmer T,*et al*.Towards Universal Code Generator Generation[C]. Massachusetts:Parallel and Distributed Processing,2008:1-8.

[23] Markus Voelter.A Catalog of Patterns for Program Generation[EB/OL]. http://www.voelter.de/data/pub/ProgramGeneration.pdf, 2003.03.22.

[24] Brunie N,Dinechin F D,Kupriianova O,*et al*.Code generators for mathematical functions[C].Lyon:Computer Arithmetic, 2015: 22-24.

[25] Jack Herrington.Code Generation in Action[M].Greenwich:Manning Publications, 2003:23-54.

[26] Senthil J,Arumugam S.Automatic Code Generation for Recurring Code Patterns in Web Based Applications and Increasing Efficiency of Data Access Code[J]. International Journal of Computer Science Issues,2012,9(3):473-476.

作者简历

教育经历:

2010 年 9 月至 2014 年 6 月，本科生阶段，就读于浙江工业大学，软件学院，软件工程专业。

2014 年 9 月至今，硕士研究生阶段，就读于浙江大学，软件学院，软件工程专业。

工作经历:

2015 年 4 月至 2015 年 7 月，宁波智慧物流科技有限公司，软件开发实习生。

2015 年 7 月至今，IBM，软件开发实习生。

攻读学位期间发表的论文和完成的工作简历:

2015 年 4 月至 2015 年 7 月，在智慧物流科技有限公司参与魔杖调度系统 Wand 的研发和交接包物流应用 App 的后台研发。

2015 年 7 月至 2015 年 8 月，在 IBM 参与自定义 SQL 语句验证工具 SSV 的设计与研发。

2015 年 8 月至 2016 年 1 月，在 IBM 参与移动支付网关 MPG 的设计与研发。

致谢

转眼之间两年的研究生时光就要结束了。由于家就在学校边上，所以我没有选择住宿，而是每天回家。这也使得自己缺少了与同学们交流的时间。然而我与同学们之间的友谊并没有因此而受到阻碍，大家都深深的感受到了这个学校浓厚的学习氛围，大家都积极参加到实验室的项目之中。学校提供了专业书籍丰富的图书馆，让我们能更全面的接触知识，及时对大脑进行充电。在论文撰写过程之中，得到了很多人的帮助。再次，我由衷地对他们表示最真诚的感谢。

首先要感谢干红华老师对我不厌其烦的教会，在我需要帮助的时候总是能指引我。还要感谢刘二腾老师，感谢他时常伸出援手，为我提供宝贵的意见，提高论文质量。同样感谢学校里的其他老师和同学，有了他们的陪伴与帮助，才有了我学习的动力。

感谢宁波智慧物流以 IBM 的同事们，是你们在我实际开发中提供了不断的帮助，给了我许多理论和实际开发上的指导，并能耐心讲解项目中的技术问题，对我能力的提升提供了极大的帮助。

还要感谢所有我所参考过的文献著作的作者们，前人的经验为我提供了极大的帮助。

最后，再次向所有帮助过、关心过我的人，说一声谢谢。祝大家身体健康，万事如意！

付荣

于浙江大学软件学院

2016 年 4 月 1 日