|  |
| --- |
| **REST风格Web API自动生成系统的设计与实现** |

|  |
| --- |
| Design and Implementation of REST-Style Web API Generation Tool |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生姓名 | 邓煜荣 | 学号 | | 201311701407 | |
| 所在学院 | 数学与计算机学院 | | 班级 | | 软件1134 |
| 所在专业 | 软件工程 | | | | |
| 申请学位 | 学士学位 | | | | |
| 指导教师 | 高升 | | 职称 | | 副教授 |
| 副指导教师 |  | | 职称 | |  |
| 答辩时间 | 年 月 日 | | | | |

**目 录**

[设计总说明 I](#_Toc481495400)

[**introduction** II](#_Toc481495401)

[1 绪论 1](#_Toc481495402)

[1.1 本设计的背景 1](#_Toc481495403)

[1.2 本设计的需求来源 2](#_Toc481495404)

[1.3 国内外的研究情况 2](#_Toc481495405)

[1.4 本设计要解决的问题及其意义 2](#_Toc481495406)

[1.5 本设计要达到的目标 3](#_Toc481495407)

[2 本设计的需求分析 3](#_Toc481495408)

[2.1 用例分析 3](#_Toc481495409)

[3 相关技术的概况 3](#_Toc481495410)

[3.1 REST 架构风格 3](#_Toc481495411)

[3.2 Spring Boot 4](#_Toc481495412)

[3.3 Spring Data JPA 4](#_Toc481495413)

[3.4 Spring HATEOAS 5](#_Toc481495414)

[3.5 Hibernate 5](#_Toc481495415)

[3.6 Swagger UI 5](#_Toc481495416)

[4 代码生成工具设计 5](#_Toc481495417)

[4.1 整体架构设计 5](#_Toc481495418)

[4.2 数据模型的建立 6](#_Toc481495419)

[4.2.1 实体信息数据模型 6](#_Toc481495420)

[4.2.2 元数据信息数据模型 7](#_Toc481495421)

[4.3 代码生成逻辑的设计 9](#_Toc481495422)

[4.3.1 实体信息到类元信息的转换逻辑 9](#_Toc481495423)

[4.3.2 类元信息到代码的转换逻辑 11](#_Toc481495424)

[4.4 用户界面的设计 11](#_Toc481495425)

[5 实现效果 11](#_Toc481495426)

[鸣 谢 12](#_Toc481495427)

[参考文献 13](#_Toc481495428)

# 设计总说明

Web服务在近年来得到了广泛的应用，但其复杂程度和实现难度被使用者所诟病。Web API是Web服务的一种轻量级实现，其性能和实现难度相对于Web服务均占有优势，被大量运用于移动设备和Web应用前后端交互，Web API快速开发需求随之出现。本设计的目的是解决Web API的快速开发的问题。虽然支持快速开发Web API的框架和工具数量大，但是仍然需要开发人员手工编写代码，过程中会出现大量重复且必须的代码，影响开发的进度和增加开发成本。本设计通过通用代码的自动生成，有助于开发人员将主要的精力集中于核心的业务开发中，缩短开发周期和开发的成本。

关键词：代码生成；Web API；REST风格

# **introduction**

Web Services has been widely used in recently years, but its complexity and difficulty has been criticized. Web API is a lightweight implementation of Web Services. Compared to Web Services, Web API has better performance and easier way to implement, which is widely used in mobile devices and interaction between front and back in Web Applications. There is a great requirement of Web API rapid development. The purpose of this code generation tool is to solve the problem of great Web API rapid development requirements. Existing frameworks and tools still require developers to write code manually, with a lot of duplicated code in the process, affecting development progress and increasing development costs. This code generation helps developer focus on the development of business instead of the duplicated but necessary code by means of generating that duplicated code, which will shorten the time of development and development costs.

**Keywords: code generation; Web api; REST-Style**

**REST风格Web API自动生成系统的设计与实现**

软件工程，201311701407，邓煜荣

指导教师：高升

毕业设计说明书

# 绪论

## 本设计的背景

随着互联网的发展，Web服务在近年来得到了广泛的应用。Web服务在正确地部署到服务器以后，可以通过互联网进行访问。在这个过程当中，采用标准化的XML格式等多种平台无关的技术，大幅度提高了软件的重用率和互操作性。SOA（Service-Oriented Architecture，面向服务架构）让Web服务得到了很大的发展，相关的标准和技术不断地出现，对Web服务的每个细节都做出了相应的规定，使其更加标准化，进一步降低了异构平台的访问成本，其中对Web服务影响较大的协议有SOAP（Simple Object Access Protocol，简单对象访问协议）和XML-RPC（XML Remote Procedure Call，XML远程过程调用），但在标准化的过程中带来臃肿和复杂逐渐被使用者和开发人员诟病。

2000年，Roy Fielding在其论文《Architectural Styles and the Design of Network-based Software Architectures》中提出了REST（Representational State Transfer，表述性状态转移）软件架构风格，提倡无状态、使用HTTP方法等简易的措施代替复杂的SOAP访问。

受到REST架构风格的启发，Web API（Web Application Programming Interface，Web应用程序编程接口）开始被广泛使用。Web API是Web服务的一种轻量级实现，Web API通常采用JSON（JavaScript Object Notation，JavaScript对象标记语言）作为数据交换格式，通过URI携带参数等等的更为简洁的方式实现Web服务的访问。相比起传统的Web服务，Web API的实现难度低，性能开销小，而且JSON格式的数据浏览器可以以较低的性能开销处理，这些的优势都让Web API成为当今开发者欢迎的技术。

近年来移动互联网的发展，移动设备的处理速度和储存容量使其难以离开云端的服务器完成计算任务。在资源受限设备上，Web API能够更好地完成移动设备与云端之间的数据交互。另外，在Web应用功能越来越多，架构越来越复杂时，Web API是Web应用的前后端分离可行的解决方案。无论是互联网行业还是传统的软件都有着较大的Web API开发需求。

## 本设计的需求来源

Web API的需求随着互联网发展的不断发展而日渐增加。采用Web API实现前后端分离的Web应用逐渐增多，根据现有的代码分析，大部分的Web API仅仅是实现简单的数据查询、封装等简单操作，这类简单操作的实现代码极为类似，这类的代码采用人工编写的方式会浪费大量时间。在总结代码库当中所有项目的数据访问、数据封装等功能的重复代码以后，得出一个Web API实现的模板，并以此为蓝本生成Web API的项目。

这个代码生成工具是一家创业公司用于快速开发外包项目的解决方案，可以生成用于评估实体设计是否合理的原型以及在此之上进行开发的工程框架。

## 国内外的研究情况

代码生成工具是根据用户输入的领域模型以及其他信息生成相应的代码。

国外的软件公司有着自己的代码生成解决方案。一种较为常见的解决方案是把代码生成功能整合在建模工具里面，例如蓝色巨人国际商业机器公司收购的Rational公司的产品Rose在代码生成方面比较成熟。用户在Rose软件上完成建模以后，可以生成相应的模型代码。与Rose软件相似的PowerDesigner，同样可以根据用户在其之上建立的模型生成相应的模型代码。另外一种较为常见解决方案是是将代码生成的功能整合到IDE（Integrated Development Environment，集成开发环境）当中，这种解决方案做得比较好的产品有微软公司旗下的开发工具Visual Studio。Visual Studio通过一系列的向导收集用户的需求信息，向导完成以后IDE会自动根据信息生成项目的基本框架，开发人员便可在此之上进行高层业务逻辑的开发。

国内在代码生成方面的研究相对较少。国内一些中间件厂商会有相关的一些代码生成工具，但是其作用仅仅是限于该中间件相关的代码，适用范围窄，应用场合单一等等的缺点。

## 本设计要解决的问题及其意义

REST风格的Web API概念自提出以来，产生了不少相关的开发框架和支撑技术。比如Spring MVC、RESTLet等，开发人员利用这些框架可以快速地构建和实现REST风格的Web API。但在实践当中，使用这些框架和工具的过程中难以避免涉及到大量数据库操作和文档撰写等机械而重复的工作。REST风格的Web API的操作已经被限定使用HTTP协议的GET、POST、DELETE、PUT等，对于不同对象的相同操作，无论是文档还是实现代码都有很高的相似程度，但是开发人员却不得不在这些重复的机械劳动上花费时间和精力，最终影响到了整个项目的开发周期和开发成本。

本设计就是为了解决REST风格Web API实现过程当中重复劳动的问题。开发人员只需要在开发过程中定义好系统的实体、关联关系等信息，生成工具会根据这些信息生成数据库操作、接口文档等文档。开发人员只需要编写整个系统当中核心的业务逻辑，这样可以把开发人员的精力集中于核心的业务逻辑开发当中。

## 本设计要达到的目标

本设计要达到的目的是一个能够生成基于Spring Boot、Hibernate、Swagger技术的带文档说明的层次化的Web API实现。生成的Web API项目按层次可以分为了数据访问层、业务逻辑层和数据表示层三层的结构。

数据访问层封装了数据库访问逻辑，往上层暴露相应的接口，在数据访问层之上的业务逻辑层无需了解数据访问的细节和具体实现。在生成的目标系统当中，数据访问层是通过Spring Data JPA和Hibernate两者共同实现的。

业务逻辑层是整个系统的核心部分，不同的目标系统处理的业务逻辑不一样，所以业务逻辑是留空由开发人员进行实现的。生成的系统当中，业务逻辑层包含有最基本的级联保存和删除的代码。

数据表示层主要由各种的Controller类组成，Controller则是响应用户的Web API调用的组件。这些组件通过Spring MVC的数据绑定机制获得用户调用Web API传递过来的参数，经过简单的预处理逻辑以后传递到业务逻辑层，业务逻辑层执行业务逻辑以后，将返回的数据转换层Spring HATEOAS的Resource类，传递到Spring MVC并自动将其序列化成JSON响应并输出返回给调用者，完成整个API的调用流程。

# 本设计的需求分析

本设计的核心需求是不同用途的REST风格Web API的快速开发以及快速构建原型，主要面向的用户是软件的设计人员和开发人员。

## 用例分析

功能性需求

非功能性需求

# 相关技术的概况

本设计生成的Web API采用REST架构风格。在实现上，采用了Spring Boot、Spring HATEOAS、Spring Data JPA以及Hibernate技术，Web API的文档和相应的展示页面采用Swagger UI技术、Web API工具的用户界面采用Java Swing实现。

## REST 架构风格

REST架构风格是Roy Thomas Fielding所提出的一种软件架构风格。这种的架构风格提倡将超媒体的分布式系统做到从无到有、利用客户端-服务器方式、无状态会话、缓存、统一接口、分层以及按需编码等等原则，通过资源、资源标识符、表示、表示元数据、资源元数据以及控制数据等概念对软件系统当中涉及的数据进行抽象。

上面这些原则在实践当中，将由系统分析得出的实体信息、元信息等抽象为资源，由服务器为这些资源分配一个唯一的标识符，具体实现为URI。把业务的逻辑抽象成为对服务器上不同资源的操作，操作采用HTTP协议中定义的方法来制定对资源进行什么样的操作，采用HTTP的状态码标识操作的执行结果。另外，对于一些变化较少的资源信息，可以同时给出缓存相应的数据，由客户端对这些资源进行缓存，降低了网络的访问次数，继而提高服务的效率。不难看出，REST风格的软件架构并不是什么新的概念，只是对HTTP协议当中的概念进行了回归。

随着REST架构风格的广泛应用，Leonard Richardson提出了一个用于衡量Web API以及Web服务的成熟度的模型。这个成熟度模型根据特定的标准将成熟等级分成等级0到等级3共四个等级，采用HTTP协议作为传输协议即满足等级0的标准；等级1在等级0的基础上采用了分治法把复杂的系统分解成资源，所有的数据和服务都抽象成为资源，采用资源的唯一标识符进行访问；等级2在等级1的基础之上引入了采用HTTP动词的约束，对所有资源的操作进行了统一，避免了不必要的复杂性；等级3引入了HATEOAS（Hypermedia As The Engine Of Application State，超媒体作为应用状态引擎）的约束，满足该约束的Web服务仅仅需要暴露一个入口的接口，该接口的返回值当中含有其他的资源的URI，通过这种方式，为客户端提供了自发现和自我描述的能力，客户端便可以如同人类访问网站自如般使用Web服务。

## Spring Boot

Spring Boot是由著名的Java开发框架Spring Framework的管理者Pivotal软件公司提供的一种用于快速开发Spring Framework程序的解决方案。Spring Framework强大的功能受到不少Java开发人员的热爱。但随着其发展已经功能的不断增加，Spring Framework的配置难度和繁琐程度也随之增大，其组件化的设计在依赖管理方面增加了相应的成本。Spring Boot的出现就是为了这些问题。Spring Boot大量采用约定优于配置的原则，内置大量预置的参数，开发人员只需要进行极少量的配置即可以完成Spring Framework的IoC（Inversion of Control，控制反转）和AOP（Aspect Oriented Programming，面向切面编程）功能以及Spring MVC的配置，接近“开箱即用”的效果。另外，在依赖管理方面，Spring Boot将复杂的Maven依赖封装成了一个又一个的Starter，开发人员只需要根据所需的功能引入相应的Starter即可，在很大程度上降低了配置的难度和复杂程度。

## Spring Data JPA

Spring Data同样是由Pivotal软件公司为了简化Spring Framework数据访问的技术，提供了包括传统的关系型数据库和非关系型数据库的访问接口。Spring Data JPA（Java Persistence API，Java持久化应用程序编程接口）是Spring Data项目下的子项目，通过JPA元数据定义数据操作过程持久化的行为，常常被用于访问关系型数据库。使用Spring Data，开发人员要进行数据访问，所需的操作只是按照一定的规则对DAO（Data Access Object，数据访问对象）的方法进行命名，Spring Data会解析方法的名称并转换成相应的数据访问操作，免去了数据访问代码的编写，进一步提高了软件系统的开发效率。Spring Data项目本身仅仅提供的是一种编码模式，不提供数据访问的实现，数据访问的实现需要由第三方的ORM（Object/Relationship Mapping，对象关系映射）框架提供。

## Spring HATEOAS

Spring HATEOAS是由Pivotal软件公司提供，用于Spring Framework快速开发成熟度较高的Web API和Web 服务的一组工具，通过这个工具可以快速满足HATEOAS当中资源links列表的构建。

## Hibernate

Hibernate是一个成熟的开源全自动ORM框架。使用Hibernate，开发人员可以无需对数据访问的底层细节进行关注，无需编写SQL（Structured Query Language，结构化查询语言）以及数据记录与对象之间转换的代码即可完成数据库的访问。在使用Spring Data JPA以后，无需进行任何配置即可与Spring Data JPA进行整合。

Hibernate高度的自动化，难以避免会带来一定的性能开销，而且Hibernate生成的SQL语句晦涩难懂。所以Hibernate优化技能学习曲线极其陡峭，这在一定程度上制约了Hibernate的实践当中的应用。本设计在实现过程中会预留ORM框架更换的条件，可供更换使用其他的ORM框架。

## Swagger UI

Swagger UI是一个用于生成Web API可视化和交互界面的一个工具。Swagger工具的工作原理是通过在控制器方法以及模型对象至上增加Swagger的注解信息，描述这个方法返回值、参数等信息，Swagger UI会自动根据这些注解信息生成相应的说明和交互界面，开发人员不需要为此编写一句的代码和一个的HTML页面。

# 代码生成工具设计

## 整体架构设计

代码生成工具采用模块化的设计，用户界面和主要的生成逻辑均为单独的模块。通过与不同的用户界面整合，可以把代码生成工具实现为普通的桌面应用程序、Web应用程序甚至是IDE插件形式。

本代码生成工具采用的不是以模板引擎为基础的代码生成策略，而是基于代码结构进行代码生成。所以，代码生成工具必须同时具备构造目标系统代码结构和根据目标系统代码结构生成的能力。根据这两个目标，把代码生成的逻辑拆分成为了生成目标系统代码结构的逻辑与根据目标系统代码结构生成代码两个主要的逻辑。根据这两个主要功能，把整个代码生成生成的模块继续细分为完成构造目标系统代码结构的数据信息分析模块以及将系统代码结构生成相应代码的模块的代码生成模块。另外，在完成代码的生成以后，可以根据用户的选择，决定是否对生成的代码进行编译打包，编译打包的工作由编译打包模块完成。

三个模块在整个逻辑当中的顺序如图4-1所示。

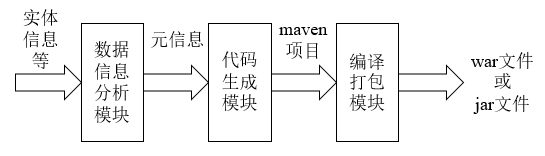


图4-1 系统工作流程

数据信息分析模块直接接收来自于用户界面的数据输入，这些数据包含有工程基本信息、实体类以及实体类之间关联关系的数据，这些数据在数据信息分析模块会被转换成相应的元信息，这些信息包含有Java类信息和工程的配置信息。

代码生成模块接收来自数据信息分析模块的元数据，根据特定的规则基于元信息生成相应的配置文件和代码文件。

编译打包模块则将会把代码生成模块生成的工程通过maven工具将其打包成为一个内嵌有Servlet容器的jar文件或者是用于部署到Servlet容器的war包文件。

系统整体的架构如图4-2所示。

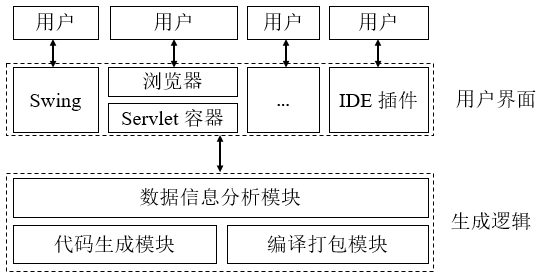


图4-2 系统整体架构示意图

## 数据模型的建立

无论是数据信息分析模块还是代码生成模块，其输入和输出的数据都需要按照一定的结构进行组织。在面向对象的编程语言当中，这些数据的组织通常采用类的形式进行组织，本设计采用Java语言实现，所以本设计同样也以类为单位对数据进行组织。

### 实体信息数据模型

实体信息数据模型主要包含实体信息、实体属性信息以及实体关系等信息。该模型如图3-3所示。

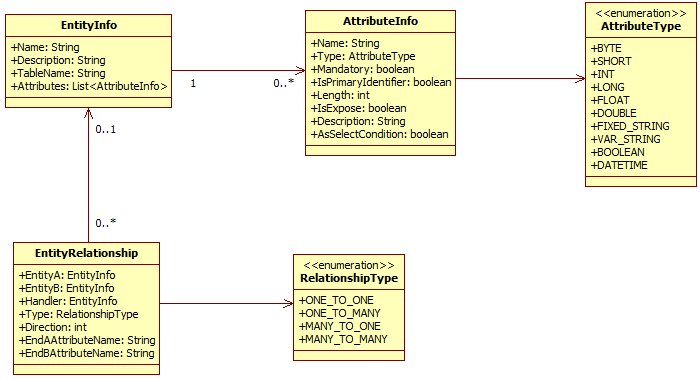


图3-3 实体信息数据模型

实体信息模型主要是根据实体设计过程进行抽象得到的结果。该模型通过实体信息（对应图中EntityInfo）、实体属性（AttributeInfo）和实体关联（EntityRelationship）三个实体以及由属性数据类型（AttributeType）和关系类型（RelationshipType）两个枚举组成。

### 元数据信息数据模型

元数据信息模型主要用于组织类的元信息数据。这些元数据主要是用于描述类、类当中字段、方法、注解等等Java元素的数据。元数据信息模型是根据Java语言的语法要素抽象得到的，元数据信息模型当中有10个与Java语法相关实体信息，这10个的实体分别是类信息（ClassInfo）、访问修饰符信息（ModifierInfo）、字段信息（FieldInfo）、方法信息（MethodInfo）、注解信息（AnnotationInfo）、类型信息（TypeInfo）、注解参数信息（AnnotationParameter）、参数信息（Parameter）指令信息（Instruction）以及值表达式信息（ValueExpression）。这10个实体类的关系如图3-4所示。

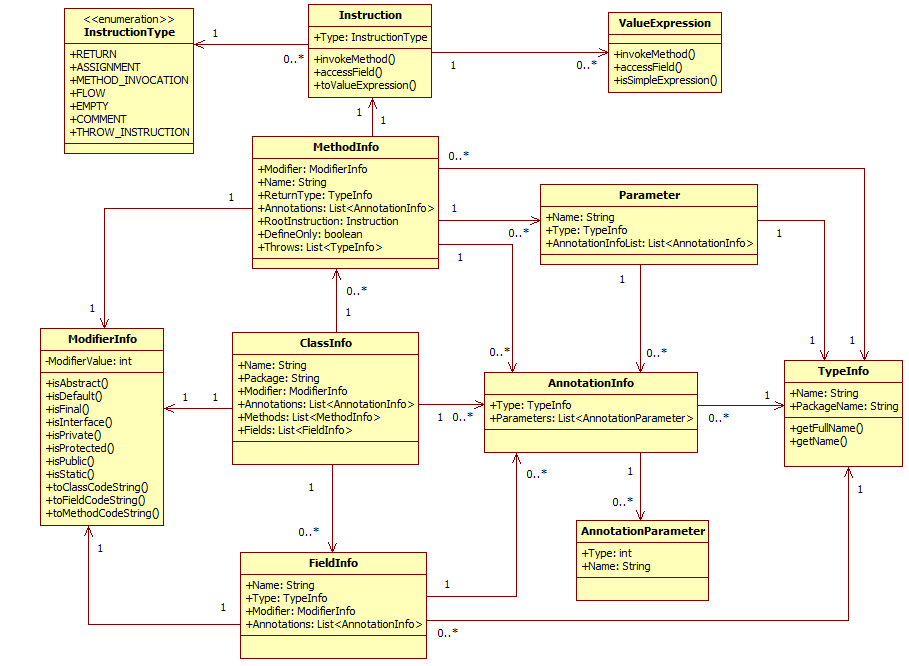


图3-4 元数据信息模型

在Java语言当中，所有的方法、字段等信息不能脱离类而存在，所以在抽象得到的元数据模型中，ClassInfo实体位于模型的中间位置，方法信息和字段信息类实例在类信息当中以字段的形式存在，并以此反映了方法、类字段和类之间的关系。位于左右两边外部的访问修饰符信息和类型信息会被多个不同的实体所使用。具体体现在Java当中类、字段和方法均有相关的访问修饰符。

注解是JDK1.5版本以后引入的用于在代码当中添加额外信息的特殊标记。注解信息使用在类、字段、方法以及方法的参数之中，所以模型当中，注解信息的位置同样是相对中心的位置。注解同样有类型和参数信息，所以注解信息对注解参数以及类型信息有着一定的关联关系。

在模型顶端的Instruction类和ValueExpression类分别表示一条Java指令以及一个就有表达值功能的表达式。在本工程当中，指令有着较为广泛的含义，一方面可以指一条具体的操作指令，也可以指通过某种流程控制结构组织起来的多条程序指令。指令具体的实现则通过对Instruction类的继承，正因为如此，在方法的指令属性上面不需要采用Instruction的集合类来存放多个的指令实例。ValueExpression与Instruction类似，其作用仅仅用于表示一个值，这个值可以应用在很多地方，比如用作函数参数，用作返回值等等。正如Java支持链式的方法调用，值表达式对象同样支持在表达式的基础上调用方法以及访问字段等操作，操作的返回值仍然是一个值对象。

在实际的代码生成过程当中，上面模型当中所有的实体会根据Java的语法规则组织成树状的结构，接近Java程序当中的语法元素的组织。

## 代码生成逻辑的设计

### 实体信息到类元信息的转换逻辑

实体信息到类元信息转换的逻辑，输入数据是工程信息（包含有工程的开发者信息、工程名称、版本、项目包名等信息）以及这个工程当中的实体信息和实体关系信息。输出信息是这个

从实体信息到类元信息转换的逻辑会根据类的类型进行拆分。本设计工具生成的类根据其作用可以分成十类，按照生成步骤的分别是Spring Boot的主类、Swagger UI的配置类、自定义异常类、所有组件都会依赖的公用类、实体类、DAO接口、Service类、Controller类、HATEOAS资源类、HATEOAS资源装配器。但是，Controller类与HATEOAS资源装配类之间存在相互依赖的关系，所以在生成相应的Controller和HATEOAS资源装配类时没有创建相关的字段，需要两个类均生成完毕以后添加彼此之间的依赖关系。所以从实体信息到整个系统的类元信息转换一共被拆分成了十一个步骤。

#### 内容固定的类元信息的生成逻辑

主流的开源框架当中，搭建开发环境所必须的配置和代码是高度接近甚至是一样的。在本代码生成工具所生成的目标系统当中同样存在着类似的大量重复代码，Spring Boot的主类、Swagger UI的配置类、自定义异常类、所有组件依赖的公用类都属于这类代码。

这一类的代码的生成逻辑较为简单而且固定。其基本的逻辑如下所示：

(1) 创建一个类元信息对象，根据其作用命名并指定所在的包名。

(2) 根据类中代码的结构，采用编码的方式构建类的代码结构。

(3) 保存数据。

#### 实体类元信息的生成逻辑

相比内容较为固定的元信息，实体类元信息的转换较为复杂。首先，用户输入的实体信息没有固定格式和规范，只要用户所输入的信息不违反强制性的约束条件，实体类转换逻辑都必须对实体信息做出最合适的处理。

实体信息到实体类元信息的转换有三个要点。第一是实体信息当中的属性名称和属性类型恰当地转换成相应的字段名称和字段类型，对于一些与编程语言保留字冲突的属性名称，转换逻辑会进行相应的判断，并通过添加一定的前缀后缀避免与语言的关键字冲突；第二是正确处理实体之间关联关系相应字段。实体之间的关联关系粗略可以分为一对一、一对多、多对一和多对多四种。转换逻辑需要做到正确创建关联关系对应字段的字段类型，创建的是对方实体类的实例或者是对方实体类的实例的集合；第三是恰当地根据实体信息以及实体关联信息创建用于ORM框架的注解信息。这些信息控制ORM框架进行数据库操作时的行为，是一项很重要的数据。这些数据的处理难点是处理关联关系维护方的配置。这些配置会影响到关联关系双方实体类在执行保存、更新、删除等操作的行为，这些信息生成不当会造成数据误删等重大损失。

由于进行实体间关系转换时的逻辑需要满足相关实体类元信息必须存在的先决条件，所以完整的实体类元信息生成也会分成三个步骤。第一步骤是生成实体类本身以及实体本身的属性对应的字段信息，忽略实体关系相关的字段。第二个步骤是在第一个步骤所生成的实体元信息基础上，通过解析关系信息，在实体类元信息当中添加相应的字段信息。第三个步骤是在前两个步骤生成的字段基础之上，生成相应的get/set方法，使得生成的实体类满足JavaBean的规范。

第一个步骤的逻辑描述如下：

对于工程当中的每一个实体信息E，都会进行下面的操作：

(1) 创建一个对应的实体类的元信息C，如果实体信息E指定的实体名称没有与编程语言的保留字冲突，则直接采用实体信息E的名称作为实体类C的名称；否则在E的名称后面加上“Entity”尾缀作为实体类C的类名，并将其包名设置为项目当中专门用于存放PO类包的包名。

(2) 创建Table注解。如果实体信息E当中指定了这个实体对应的表名，则直接使用实体信息E当中指定的表名；如果没有在实体信息E当中指定表名，如果实体的名称没有与数据库当中的保留字冲突，则直接使用实体类作为表名；如果冲突，则在实体名称后面增加“Table”尾缀作为表名。将确定下来的表名以“name”为名的参数添加到Table注解当中。

(3) 创建Entity注解，并将这个注解添加到类元信息C上。

(4) 对实体信息E当中的所有属性进行遍历。对每一个属性A。根据属性的类型转换成相应的字段类型，因为属性类型与字段类型关系属于一一对应的关系，所以字段类型的生成过程可以认为是直接的替换。属性的名称与实体名称类似，会先判断属性名称是否与编程语言的保留关键字冲突。如果属性名称与编程语言之中的保留关键字冲突，则在属性名称基础上增加“Field”后缀，作为字段的名称；否则直接使用属性名称作为字段的名称。利用转换得到的字段类型和字段名称创建相应的字段信息。如果这个属性被标记用作实体的唯一标识符，则在字段之上增加Id注解，否则增加Column注解。

(5) 对生成的字段进行遍历。根据实体类当中使用Id注解的字段数量的不同执行不同的逻辑。

① 如果实体类当中没有任何使用Id注解的字段，则说明这个实体类当中不存在任何的唯一标识符，则自动添加一个类型为long，名为id的字段，在这个字段上添加Id和GeneratedValue注解，用作业务无关的自增长标识符。

② 如果实体类当中只有一个使用Id注解的字段，则在这个字段增加一个GeneratedValue注解，为其赋予自增长标示符的角色。

③ 如果实体类当中有多个使用Id注解的字段，则在该实体类

### 类元信息到代码的转换逻辑

## 用户界面的设计

本代码生成工具的定位是软件开发的辅助工具，采用传统的桌面应用程序较Web应用程序更为合适。为了降低用户界面的复杂程度，用户界面采用Java的Swing进行用户界面开发。

### 主界面布局的设计

大部分的工具

# 实现效果

# 鸣 谢

本毕业设计得以顺利完成，首先要感谢导师高升老师在代码生成工具的设计、实现以及毕业设计说明书的撰写过程中给予帮助。

# 参考文献

（宋体小5号字，行距为固定值16磅）（参考文献用“[”，“]”括起，具体写法参照《广东海洋大学本科生毕业论文设计撰写规范》（校教务〔2007〕年122号）的要求）

附 录

（宋体五号字，行距为固定值20磅）（要求：附录的有无根据说明书（设计）情况而定，内容一般包括正文内不便列出的冗长公式推导、符号说明（含缩写）、计算机程序等。附录中有程序源代码的因篇幅限制可酌情考虑内容的字号。）