****



**本 科 毕 业 论 文**

院 系 计算机科学与技术系

专 业 计算机科学与技术

题 目 面向API映射的在线查询

与管理平台的设计与实现

年 级 2016级 学 号 161220064

学生姓名 李宸玮

指导教师 张天 职 称 副教授

提交日期 2020年6月2日

**面向API映射的在线查询与管理平台的设计与实现**

**Design and Implementation of Online Query and Management Platform for API Mapping**

**南京大学本科生毕业论文（设计、作品）中文摘要**

题目：面向API映射的在线查询与管理平台的设计与实现

院系：计算机科学与技术系

专业：计算机科学与技术

本科生姓名：李宸玮

指导教师（姓名、职称）：张天 副教授

摘要：

为了适应跨平台、多语言的环境，软件项目通常需要发布不同的版本，而开发人员常常需要对项目进行迁移。为了使不同版本和语言间的项目迁移更为便捷高效，已经有了很多代码迁移和代码重构工具。开发人员在跨语言编程时，也已经具备了足够简洁明晰、规范完整的官方API文档可供查看。但为了进一步提高跨语言开发的效率，有许多的研究将目光投向了跨语言相似API的发掘和跨语言API映射的建立。

有研究提出了一种深度挖掘方法来广泛地探索API文档，改进了对API映射方法的建立，并将这种方法应用于Java到Swift两种语言的API一对一、一对多映射，最终得到了良好的API映射结果。但是这个结果仅仅被存储在文档之中，并没有被实际地进行应用，以方便开发人员的使用。

因此，本文基于现有的不同语言间的API映射关系，设计了在线查询和管理平台，并基于Spring MVC、HTML及MySQL数据库等开发框架与相关技术，进行了平台实现。目的是在多语言映射的数据库（Java/Swift）基础上，建立实现一种框架，帮助用户搜索查询API及其映射等多方面的信息。这种框架不仅可以帮助软件开发人员在跨语言编程时，快速了解和使用API，查找等效的API方法并选择调用，同时也可以帮助有关API映射的研究等相关工作的开展。本文亦实现了API映射的在线管理，有权限的人员通过登录后，可以直接在线录入API信息及其映射。此外，本文也对未来面向API映射的进一步研究，以及平台的进一步完善进行了展望，并指出了一个可行的方向。

关键词：API映射；在线查询与管理平台；Web应用的设计与实现

**南京大学本科生毕业论文（设计、作品）英文摘要**

THESIS: Design and Implementation of Online Query and Management Platform for

API Mapping

DEPARTMENT: Computer Science and Technology

SPECIALIZATION: Computer Science and Technology

UNDERGRADUATE: Chenwei Li

MENTOR: Associate Professor. Tian Zhang

ABSTRACT：

In order to adapt to a cross-platform, multi-language environment, software projects usually need to release different versions, and developers often need to migrate the project. In order to make project migration between different versions and languages ​​more convenient and efficient, there have been many code migration and code refactoring tools. Developers already have enough simple, clear, and complete official API documents for cross-language programming. However, in order to further improve the efficiency of cross-language development, many studies have turned their attention to the discovery of cross-language similar APIs and the establishment of cross-language API mapping.

Some studies have proposed a deep mining method to explore API documents extensively, improved the establishment of API mapping methods, and applied this method to Java to Swift APIs for one-to-one, one-to-many mapping, and finally Good API mapping results are obtained. But this result is only stored in the document, and has not been actually applied to facilitate the use of developers.

Therefore, based on the existing API mapping relationships between different languages, this article designs an online query and management platform, and implements the platform based on development frameworks and related technologies such as Spring MVC, HTML, and MySQL databases. The purpose is to build and implement a framework based on a multi-language mapping database (Java / Swift) to help users search and query API and its mapping and other aspects of information. This kind of framework can not only help software developers to quickly understand and use APIs during cross-language programming, find equivalent API methods and choose to call, but also help to carry out related work such as research on API mapping. This article also implements online management of API mapping. After login, authorized personnel can directly enter API information and its mapping online. In addition, this article also looks forward to further research on API mapping in the future and the further improvement of the platform, and points out a feasible direction.

KEY WORDS: API Mapping; Online Query and Management Platform; Design and Implementation of Web Application

**目录**

[1 引言 1](#_Toc42093433)

[1.1研究背景 1](#_Toc42093434)

[1.2研究意义 2](#_Toc42093435)

[1.3本文工作 3](#_Toc42093436)

[1.4论文的结构 3](#_Toc42093437)

[2 技术背景 5](#_Toc42093438)

[2.1 API（Java & Swift） 与API映射 5](#_Toc42093439)

[2.2 Spring MVC + MyBatis + Thymeleaf 6](#_Toc42093440)

[2.3 MySQL 8](#_Toc42093441)

[2.4 HTML + CSS 9](#_Toc42093442)

[3 需求分析 11](#_Toc42093443)

[3.1 功能性需求 11](#_Toc42093444)

[3.1.1 查询功能 12](#_Toc42093445)

[3.1.2 管理功能 12](#_Toc42093446)

[3.2 非功能性需求 13](#_Toc42093447)

[4 平台设计 15](#_Toc42093448)

[4.1 整体架构设计 15](#_Toc42093449)

[4.2 操作流程设计 16](#_Toc42093450)

[4.3 功能设计 17](#_Toc42093451)

[4.4 界面设计 18](#_Toc42093452)

[4.5 数据库设计 19](#_Toc42093453)

[4.6 难点分析 23](#_Toc42093454)

[5 平台的开发与实现 25](#_Toc42093455)

[5.1 开发框架环境搭建 25](#_Toc42093456)

[5.2 平台实现框架 26](#_Toc42093457)

[5.3 系统类结构 27](#_Toc42093458)

[5.4 后端：数据操作与请求响应 28](#_Toc42093459)

[5.5 前端：界面渲染 34](#_Toc42093460)

[6 效果展示实例 37](#_Toc42093461)

[6.1 主页 37](#_Toc42093462)

[6.2 API信息查询结果 37](#_Toc42093463)

[6.3 API映射索引 39](#_Toc42093464)

[6.4 管理API映射 40](#_Toc42093465)

[6.5 细节展示 42](#_Toc42093466)

[7 应用实例测试与评估 45](#_Toc42093467)

[7.1 功能测试 45](#_Toc42093468)

[7.2 响应时间与时间复杂度 46](#_Toc42093469)

[8 工作总结与展望 49](#_Toc42093470)

[8.1工作总结 49](#_Toc42093471)

[8.2平台优化与未来展望 49](#_Toc42093472)

[参考文献 51](#_Toc42093473)

[致谢 55](#_Toc42093474)

## 1 引言

### 1.1研究背景

为了适应跨平台、多语言的环境，软件项目通常需要发布不同的版本，开发人员常常需要对项目进行迁移。但这种迁移的工程量往往是巨大的。且对于开发人员来说，在迁移过程中可能会遇到相当多的问题：他们需要克服在不同语言、平台等之中过渡的不适应感，要去重新掌握、对接新的开发平台和开发语言， 从而完成高度差异化的开发工作。

API（Application Programming Interface），也即应用程序接口，提供了应用程序与开发人员访问一组例程的能力，这使得开发人员无需访问源码，也无需理解内部工作机制的细节，即可完成开发工作。无论开发人员使用的是Java、C++、Swift……还是任何一种编程语言，他们都一定避不开对该语言API的学习和使用。因而，在开发人员的编程工作中，API具有非常重要的作用和地位。

目前，为了使不同版本的项目迁移更为便捷高效，已经出现了很多代码迁移和代码重构工具[1][2][3]。开发人员在跨语言编程时，也已经具备了足够简洁明晰、规范完整的官方API文档可供查看[4]。同时亦有在线网站提供了在线搜索、查询功能，使得开发人员可以方便地查阅API文档，快速地找到所需要的API，进行查阅学习。但为了进一步方便项目的迁移，提升跨语言开发的效率，跨语言API映射的建立和研究是极为必要的。已经有许多研究[5][6][7][8][9]就挖掘并建立用于语言迁移,及平台迁移的API之间可能的映射，做出了许多重要的工作与探索。

有研究[10]通过仔细查阅例如Java，Swift等不同编程语言的API文档,发现文档作者皆详细描述了API的方法、参数和返回值等内容，但这些API文档中的丰富信息，却并没有在一些基于文档的API映射研究[6][7]得到充分利用。并由此考虑通过充分利用API文档的参数、返回值以及方法描述等，从而获得更好的API映射结果。在此研究中，作者通过使用深度挖掘方法来检查API文档，并实现建立了一对一或一对多的API映射。基于Java和Swift都是时下非常流行的编程语言，因而在这两种语言之间建立API映射是十分必要的。作者将这种方法应用于Java到Swift之间的一对一和一对多映射之中，在15个不同和被广泛使用的Java类上评估了此方法，并得到一对一映射和一对多映射的前10位精度分别为76%和50%，提升了API映射的结果。这不仅推动了关于API映射研究工作的进展，同时也为设计和实现一个方便开发人员使用、查阅的API映射索引平台提供了可能，以及数据支撑。

在项目的迁移，跨语言的过渡之中，面向API映射的索引平台仍然是欠缺的。不同语言的API文档各自独立呈现，当开发人员需要将一种语言的API映射到具有相似功能的另一种语言API，需要自己进行较为麻烦繁琐的查询、学习与筛选，耗费了大量的精力和时间。在这种情况下，一个面向API映射的索引查询平台的出现，将会大大减轻开发人员在此项工作中的压力，提升工作的效率。

基于开发人员对面向API映射的相关需求，以及现有的Java API到Swift

API的一对一和一对多映射关系的数据库支持，本文提出了面向API映射的在线查询与管理平台的设计与实现。

### 1.2研究意义

建立实现面向API映射的在线查询与管理平台，其目的就是在多语言映射的数据库（Java/Swift）基础上，建立实现一种框架，以帮助用户从两种语言之中轻松过渡：

1. 在线查询（Java/Swift）两种语言的API相关信息：两种语言中每一个类所包含的方法；每一种方法所涉及的包括apiSignature、描述、参数、返回值、示例代码等在内的具体介绍。
2. 在线索引Java和Swift语言的API映射（双向）。借此，开发人员可以快速查找两种语言等效的类和方法。

同时，本文也实现了API映射的在线管理平台。有权限的人员通过登录后，可以直接就平台在线管理录入API数据及API映射关系。

这种框架不仅可以帮助软件开发人员在跨语言编程时，快速了解使用API，查找等效的API方法并选择调用，在Java和Swift两种语言中轻松过渡。同时，也可以帮助到API映射的研究等相关工作的开展。

### 1.3本文工作

本文就面向API映射的在线查询管理平台的设计与实现，进行了细致全面的介绍。

首先，本文介绍了对本文研究工作有参考价值的相关知识与背景，以及工作中所使用到的技术和方法。

其次，介绍了本文工作的需求分析与组织架构、平台设计，实现过程以及最终成果的呈现。包括对API映射数据的收集、数据储存与呈现的格式，以及到最终整个平台的搭建。

最后，本文对工作进行了总结，也对未来展开了期待与展望，以期进一步深化研究成果，扩展系统功能，也能够更加丰富面向API映射的数据库建设。在展望中，本文也具体提出了后续的研究方向和改进计划。

### 1.4论文的结构

本文主要分为四个部分：

1. 第一部分引言介绍了本文的研究背景、研究意义、工作内容以及论文的整体结构。
2. 第二部分介绍了包括API与API映射、 Spring MVC、 MyBatis、 Thymeleaf、MySQL、以及涉及前端的HTML/CSS等在内的背景知识与相关技术，并对与本文工作相关的内容进行了梳理。
3. 第三、四、五、六、七部分是本文的主体部分。详细阐述了本文工作的前期需求分析与组织架构、设计与实现、最终的实现成果展示以及对应用实例的测试和评估工作。
4. 第八部分也即总结与展望部分。主要梳理了本工作的内容、分析了本工作的亮点与不足之处，也对接下来的研究、改进与拓展进行了展望。

## 2 技术背景

本章将对本项目所涉及到的相关技术和背景进行介绍，其中将着重介绍API映射、SpringMVC+MyBatis+Thymeleaf开发框架、MySQL数据库管理系统、以及关于前端页面设计的HTML+CSS框架。

### 2.1 API（Java & Swift） 与API映射

#### 2.1.1 API（Java & Swift）

本文中所指的API（Application Programming Interface），也即应用程序接口，限定为Java和Swift两种编程语言的API。

Java是一门面向对象编程的语言，具有功能强大和简单易用两个特征，是静态面向对象编程语言的代表。在Java之中，类（class）作为数据和操作数据的方法的集合，对程序进行了封装和组织。Java还包括一个类的扩展集合，也即程序包（package），方便了开发人员在自己的程序中使用。常用的Java API类包括ArrayList、Calendar、Class、File、HashSet、Integer、String等等[11]。

在运行Java程序时，虚拟机装载程序的class文件所使用的是Java API class文件。对于Java程序而言，无论平台内部如何，Java API都具有同样的表现和可以预测的行为，而Java程序自身就是一种平台无关性的程序。

基于Java从Version1.0到Java 12的不同版本，Java API也具有一定的区别。当然，开发人员无需特别记住所有的API，在需要时进行查阅即可。不过在查阅API文档时，应注意版本号的区别。以前文提及到的在线API查阅网站MaTools为例，它提供了多个不同版本的Java X手册。

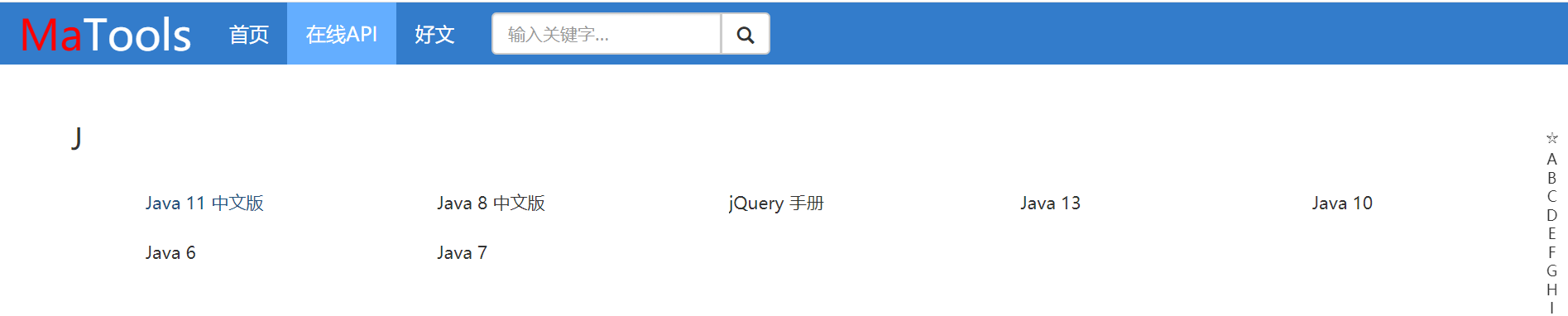


图2-1 在线API查阅网站MaTools界面

Swift则是一种强大直观的编程语言，它旨在提供资深开发者所需的自由空间和丰富功能，适用于macOS、iOS等的开发。Swift结合了C和Objective-C的优点但不受C兼容性的限制,支持过程式编程和面向对象编程。它的语法简洁，表现力却非常强。Swift通常以简洁的语法表示命名参数，使得开发者可以轻松地阅读和维护以Swift编写的API[12]。

#### 2.1.2 API映射

如前文所述，在跨语言、跨平台的开发过程和代码迁移过程中，对于API映射关系的建立是极为必要的。

Swift和Java的语法比较接近，基于相同（似）功能的实现效果，本文给出了Java到Swift的API映射。在本文当中，一个Java API可能会有多个Swift API映射。关于API及其映射的格式、详细数据等，将会在本文第三部分(面向API映射的数据存储与管理)进行详细介绍。

### 2.2 Spring MVC + MyBatis + Thymeleaf

#### 2.2.1 Spring MVC

Spring MVC[13]是一款基于MVC架构模式的轻量级Web框架，其目的是将Web开发模块化，对整体架构进行解耦，简化Web开发流程。Spring MVC基于请求驱动，即使用请求—响应模型。

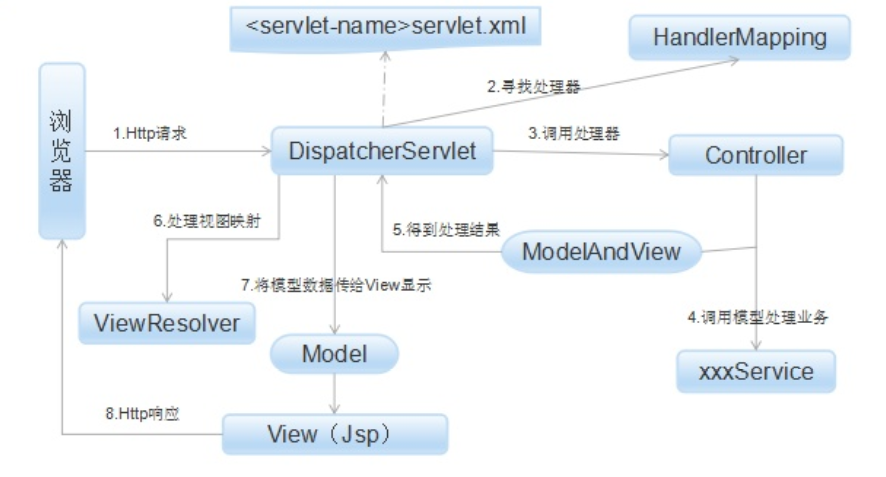


图2-2 Spring MVC整体请求流程（框架）

Spring MVC遵循MVC架构规范，因此分层开发的数据模型层（Model）、响应视图层（View）和控制层（Controller），方便了开发人员开发出结构规整的Web层。控制层（Controller）负责前端与后端的交互，数据模型层（Model）封装用户的输入/输出数据，视图层（View）选择恰当的视图来显示最终的执行结果，最终实现了一个完整规范的Web项目的请求与响应。

#### 2.2.2 MyBatis

数据的存储和处理往往离不开数据库和SQL语句，而在Web应用的开发之中，所需要做的就是连接数据库。

在底层连接数据库时往往使用的是JDBC技术，也即Java所提供的数据库连接和操作SQL的底层API。但随着原始的JDBC不再满足飞速发展的项目开发需求，MyBatis弥补了这一传统开发模式的不足，大大提升了项目的开发效率和程序的数据处理性能。

MyBatis是一款支持动态SQL语句的持久层框架，将SQL语句配置在XML文件中，避免了JDBC在Java类中添加SQL语句的硬编码问题，使得开发人员可以更多地将精力集中在SQL语句上。通过MyBatis的输出映射机制，将结果集的检索自动映射成Java对象。且有MyBatis还会创建自己的数据库连接池，对数据库连接数据进行管理[14][15]。

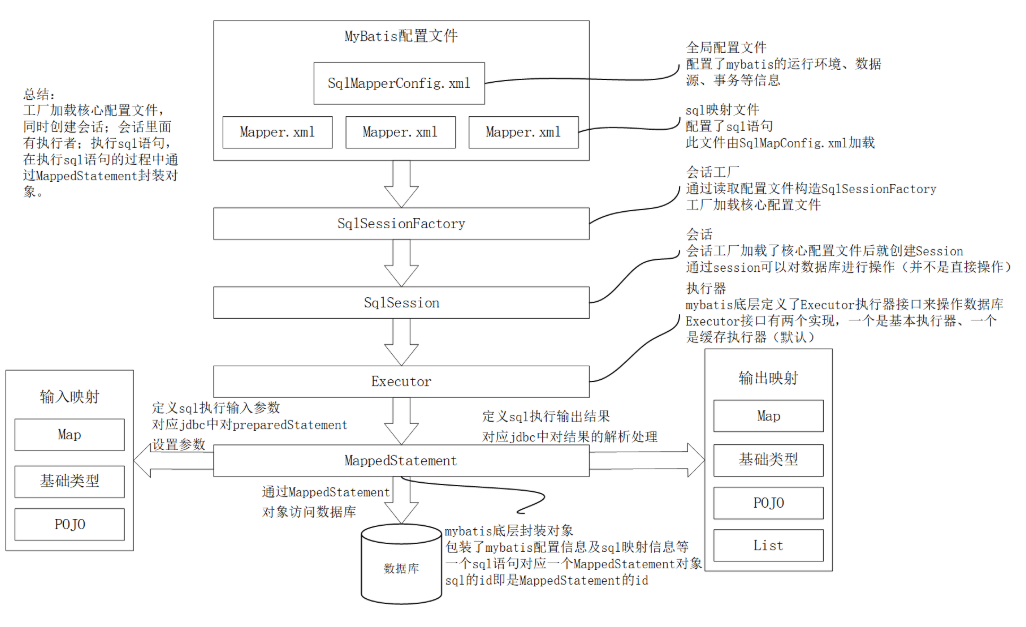


图2-3 MyBatis运行流程结构图

#### 2.2.3 Thymeleaf

Thymeleaf是用来开发Web和独立环境项目的服务器端的Java模板引擎，可用于HTML甚至是纯文本等内容的动态渲染，类似于JSP和Freemarker。它支持HTML原型，通过在HTML标签里增加额外的属性来达到模板+数据的展示方式。

Thymeleaf同时提供了一个与Spring MVC完美集成的可选模块，可以快速地实现表单绑定、属性编辑器等功能。Thymeleaf与Spring MVC的视图与自动化集成配置堪称完美，开发时仅需关注Thymeleaf的语法即可[16]。

故而，本文的设计与实现选择了：Spring MVC + MyBatis + Thymeleaf的技术框架，支撑了本次整合良好、结构规整的Web开发过程。

### 2.3 MySQL

MySQL是一个关系型数据库管理系统，它将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，如此提高了数据访问速度和灵活性。MySQL使用标准的SQL数据语言形式，与前文所提到的MyBatis进行连接，存储和管理着Web平台的所有API和API映射的相关数据。

为简化和帮助开发，本文亦使用了MySQL Workbench，这是一款专为MySQL设计的可视化数据库设计软件。利用它可以方便和快捷地设计和创建新的数据库图示，建立数据库文档，或进行复杂的MySQL迁移。它所提供的可视化操作，大大简化了SQL语句的编写过程，自动化实现了数据库表单的各种操作[17]。

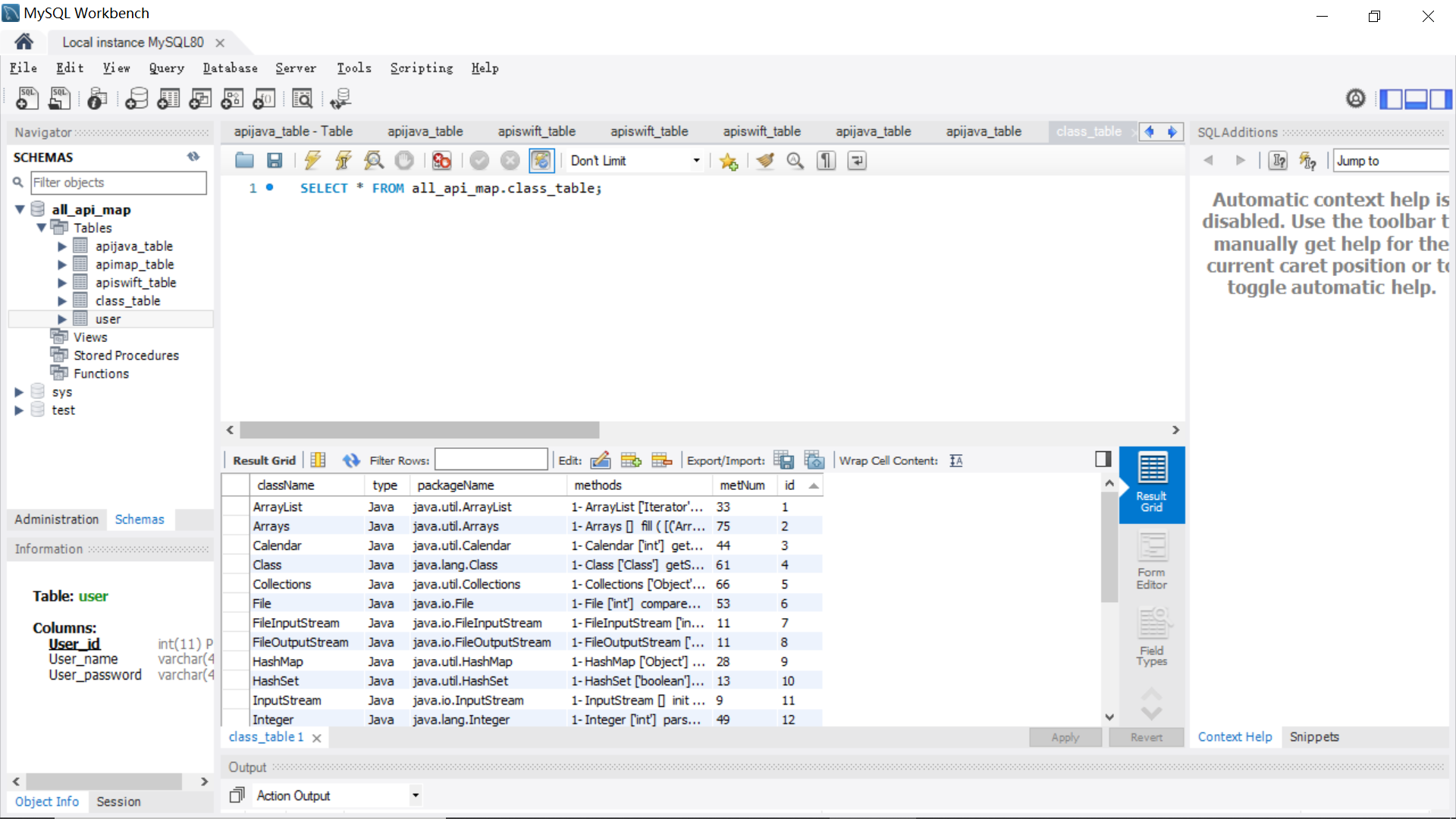


图2-4 MySQL Workbench操作界面

### 2.4 HTML + CSS

HTML的英文全称是Hyper Text Marked Language，也即超文本标记语言，其包含了一系列的标签，对网络上的文档格式进行了统一，使分散的网络资源成为了一个逻辑整体。在这里，用其作为Web平台搭建的前端网页编程语言[18]。

为开发出美观的、规范的页面，本文同时使用了CSS技术来静态地修饰网页，对网页中的各个元素进行排版与精确控制。

CSS，全称为Cascading Style Sheets，即层叠样式表。它为HTML提供了一种样式描述，定义了网页中元素的显示方式，包括字体、颜色、位置等各种样式。它丰富的样式定义，易于使用和修改的性质，可应用于多页面而实现的风格统一等等，给前端页面的设计和实现提供了极大的帮助。

CSS可通过以下三种方式添加到HTML当中[19]：

1. 内联样式，在HTML元素中使用“Style”属性
2. 内部样式表，在HTML文档头部<head>区域使用<style>元素
3. 外部引用，直接在头部引用CSS文件。

在本文当中，多使用第三种方式，来完成前端页面的布局设计。为了方便与简洁，也偶有使用第一种方式以直接约束元素样式。

## 3 需求分析

通过在线平台，用户可以直接搜索并查询有关Java和Swift两种语言的API相关信息及其映射。平台亦需要提供API信息及其映射的录入功能。除基本功能的实现外，平台还需要敏捷的用户交互反应能力，良好的操作界面和清晰的提示信息等，来引导和方便用户的操作与使用。

### 3.1 功能性需求

面向API映射的在线查询与管理平台的需求主要分为两大类：查询即通过输入API名称查询该API的具体信息及其API映射的信息。管理部分主要是是对API信息进行单条导入，及API映射关系的导入。其中，API及其映射的查询功能与管理功能相互独立，能够平行使用。

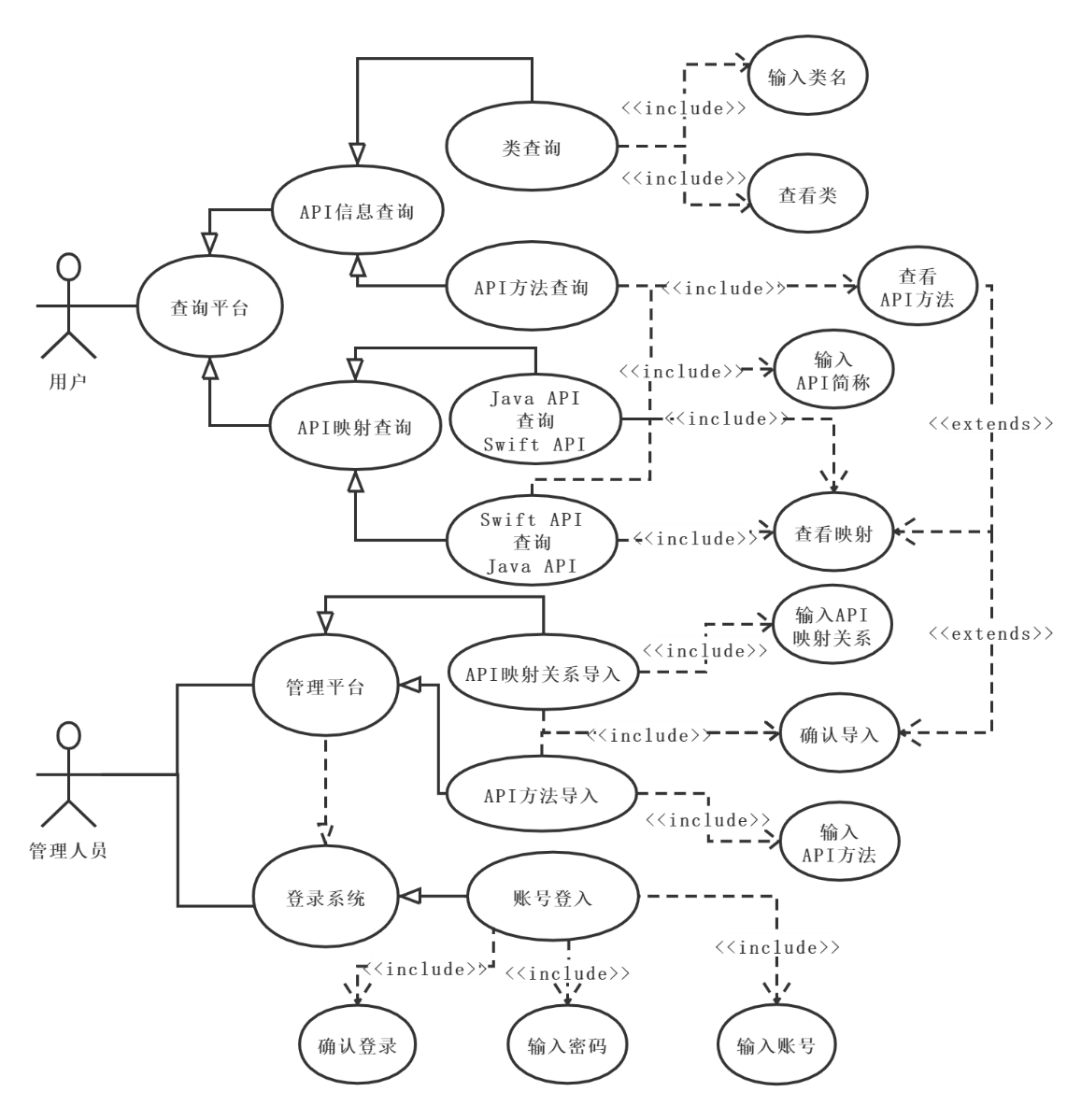


图3-1 UML用例图

### 3.1.1 查询功能

平台的核心数据是Java和Swift语言的许多条API, 这些API又可以被归属为不同的类。比如Java 的HashSet类就会有诸如HashSet\_add()等的13条API数据。为了方便开发人员等使用者，可以充分地利用这些API信息，本文又将查询功能细化为三部分：

·类查询

输入Java或Swift语言的类名，查询该类的语言类型、所包含的API数据条数、所有所包含的API的名称。当开发人员等使用者想不起来或者无法确定具体的API时，可以通过输入类名并查询其API，来进一步地找到自己所需要的具体方法。

·API信息查询

输入Java或Swift语言的API名称，查询包括API Signature、具体描述、参数、返回值等在内的关于该条API的具体信息，以掌握API的具体用途功能，及其使用方法。

·API映射索引

输入Java或Swift语言的API名称，查询其对应的另一种语言的API映射。平台输出显示具体的API映射关系对，方便开发人员等用户就API映射建立双向索引，并帮助他们的代码迁移工作。

### 3.1.2 管理功能

管理部分指API及其映射的导入功能。这一部分需要设置管理员权限，只有通过平台的登录验证了管理员身份之后，才可以进一步操作。

·权限验证

如果需要进入管理界面，则必定先会跳转至登录界面。拥有管理权限的管理人员会拥有对应的账号和密码，账号和密码会被预存在数据库表单之中。当输入账号、密码正确，则会正确地跳转至管理界面。否则，平台会一直弹窗并提示“权限验证失败”，并停留在登录界面。

·API导入

平台直接在界面上提供所需要添加的API映射的具体信息，管理人员通过页面提示“填空”完成。平台接收被添加的API数据，并进行去重和审核后，将API添加到数据库的表单之中。

·API映射导入

API映射导入实际上是对数据库内API映射关系的建立。管理人员分别输入Java API和Swift API的名称。如果两个API都存在，且确认API映射关系无误，并进行去重与审核后，在API映射关系表单中导入并存储该API映射对。

### 3.2 非功能性需求

作为一个面向用户的在线查询与管理平台，需要充分考虑到用户的使用体验和感受。本文在此提出平台需要满足的非功能特性：

·实用性和易用性

平台实际上是一个web网站，用户可直接通过浏览器访问平台主页，并通过点击菜单栏选项、在搜索框等里输入各类信息以完成对应的操作。平台的各部分通过页面顶部的菜单栏进行组织连接，用户可通过点击实现各功能页面的相互跳转。平台的使用和操作逻辑，应符合用户习惯的一般标准，并应设置有良好的界面引导并弹窗提示，方便用户能够直接上手和使用。

·时效性和时滞性

作为一个在线平台，当用户在使用时，应具有较快的处理速度与响应速度，尽量减少用户的等待时间。且有平台与后端数据库实时连接，数据库的数据更新应当没有时滞。也即，在API及其映射添加导入成功后，就可以立刻在查询平台搜索并给出对应的信息呈现。

·健壮性和稳定性

对于用户输入，平台应进行输入预处理，需要尽可能考虑到用户的各种操作，及输入情况，并针对各种情况进行边界处理。平台应能够接收与识别用户的错误输入或不合规的操作，以给出良好的响应与用户下一步的操作提示。

·可维护性和可扩展性

平台应该留有接口，以方便后续的功能增加与API语言类型的扩展。当后续数据集的规模增加，运算强度升高，平台应当仍然具备相应的支撑能力。

## 4 平台设计

本章将基于前文的需求分析，从平台的整体架构、功能、界面以及数据库三个方面，详细介绍前期的设计部分及细节。并针对设计部分，进一步地展开了难点分析。

### 4.1 整体架构设计

平台是作为一个整体的系统，供用户直接在线操作使用。平台后端直接与存有Java API和Swift API及其映射的数据库相连，提供与前端用户交互的数据支撑。平台的主要功能分为查询和导入两部分，这两部分功能相互独立，又彼此链接，可以通过菜单栏相互访问。

查询面向所有用户使用，在平台首页可以直接操作。而查询则又分为信息查询，与映射索引两个部分：在信息查询功能下，可以通过对Java和Swift的类的查询，从整体上把握API信息，或对具体的一条API方法信息直接查询；在API映射索引部分，则实现了对Java API与Swift API的双向索引。导入部分则需要管理员权限，需通过网站提供的登录系统访问。它支持对Java API和Swift API的在线导入，以及对API映射关系的在线导入。同时，对API映射的索引和对导入结果的反馈等，都可以通过在页面点击API名称直接查看API具体结果。

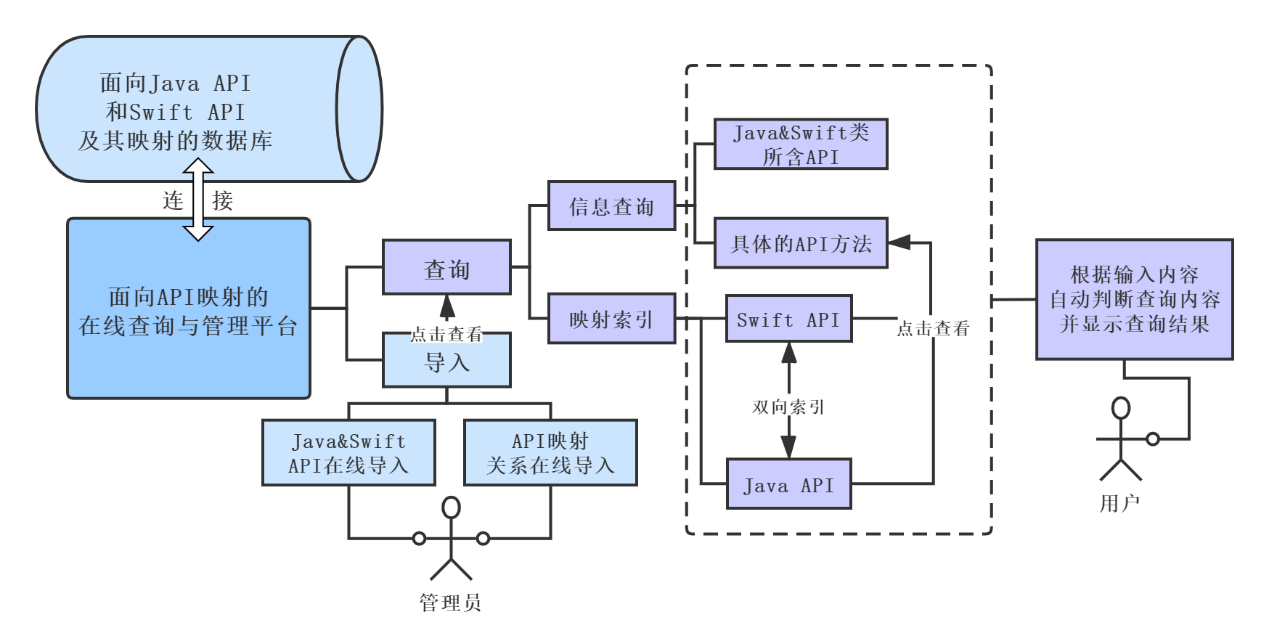


图4-1 平台整体架构图

### 4.2 操作流程设计

面向API映射的在线查询与管理平台应没有特定的使用流程逻辑，查询与管理功能部分各自相互独立，可以平行操作。各页面彼此连接，可以直接相互跳转，用户操作比较自由。但为了梳理细分平台的各个功能，及其一般性使用，本文给出了平台的一般性操作流程设计。

1. 进入主界面：输入格式为Class的类名，直接查询两种语言的类所包含的方法等相关信息；输入API简称，当只找到一个API时，直接查询对应的API具体内容，若找到多个同名API（API的简称相同，但由于参数、返回值等的不同仍被视为不同API），则显示所有API至页面，供用户点击查询。
2. 点击顶部菜单栏API映射索引，跳转至对应功能页面查询映射。直接输入API简称，若找到多个同名API（同上），则显示所有API并供用户点击索引其映射，否则直接显示API映射关系（Java API全称&Swift API全称）。同样地，单击页面API名称可跳转至API信息查询界面，显示该条API所有具体信息。
3. 单击顶部菜单栏API映射管理，在浏览器弹出新窗口并跳转至登录界面。输入用户名和方法（预先存储在数据库表单），验证管理员权限成功后跳转至管理界面。验证失败则以弹框为方式进行提示，并保持在登录界面，方便用户的二次或多次权限验证尝试。
4. 在登录成功后，页面跳转至管理界面。管理界面默认首页为添加Java/Swift API信息功能，首先需要在单选框里选中API类型，并根据提示填写对应的API内容，最后点击最下方按钮提及即可。如果API添加成功，则会跳转至API信息查询结果页面，显示所添加的API信息。
5. 点击页面顶部菜单栏添加API映射，则切换至映射添加功能页面。此时需要分别输入Java & Swift映射对的API全称，并直接点击提交即可。平台会首先对输入进行判断，如果确定所输入的两条API皆存在，则添加API映射对至对应表单。当用户添加成功后，界面会跳转至所添加的API映射结果页面，显示对应的API映射关系。同样地，点击API名称会直接跳转显示对应的API具体信息。
6. 在每一个页面的顶部，都设有菜单栏选项。用户可以通过点击对应功能，反复多次地进行搜索查询、管理添加等相关操作。

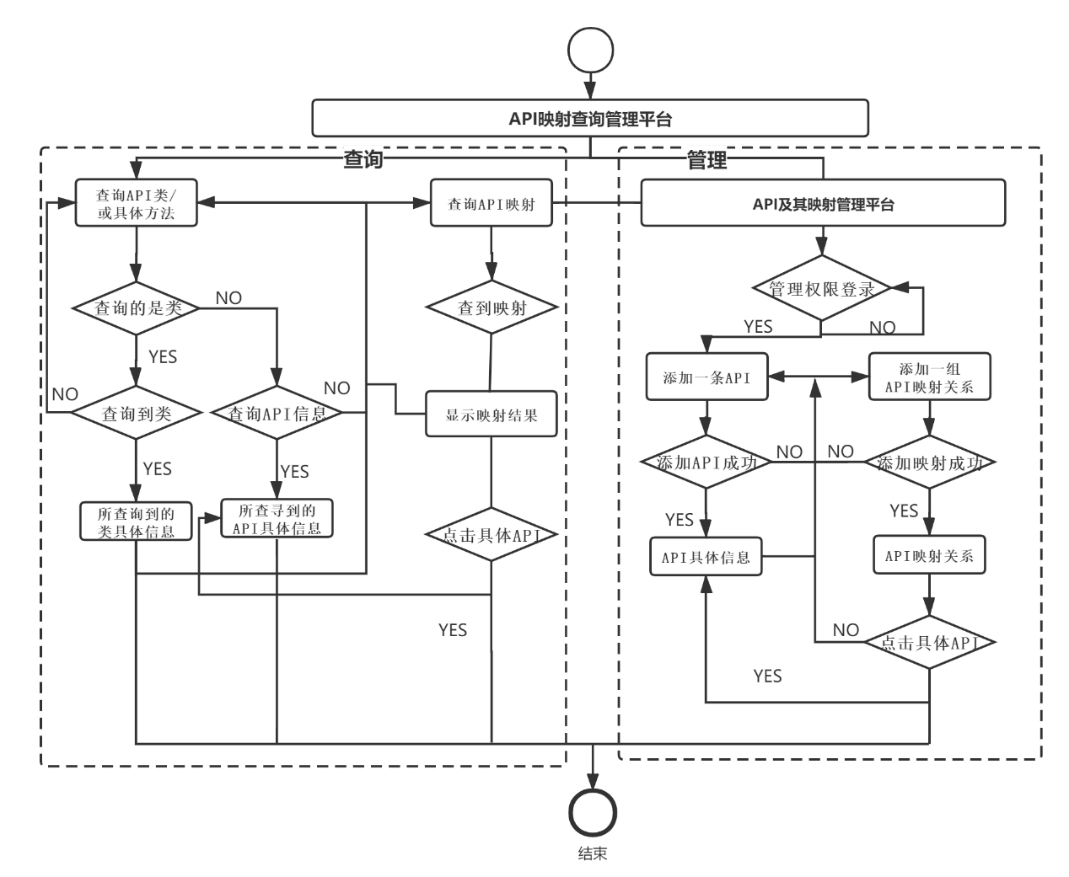


图4-2 平台操作流程图

### 4.3 功能设计

基于前文的需求分析，平台的功能被分为了查询与管理两个部分，而每一个具体的功能细分到每一个页面。这里以平台的基本使用逻辑：输入—输出，对平台功能的细节设计进行说明。

·查询

如前文第三章需求分析，查询包括对类的查询、对API方法的查询、及对API映射的查询。基于一般的查询逻辑，本文使用所查询的类及API的名称作为查询索引。由于Java和Swift语言没有同名类和API，因此无需在输入上对语言类型进行区分。由于一条含参并包括返回值信息的API全称较为繁琐，在输入部分对API名称进行简化，即包括了类名和方法名的class\_method()格式，实现API及其映射的模糊搜索。输出则结合所在页面对应的功能，与输入的字符串内容，在数据库中查询对应表单，并返回显示具体的信息。

·管理

管理部分是对API及其映射的导入功能的实现。导入页面给出数据信息表格和空白输入框，管理人员根据页面上的提示，填写API及其映射的数据，并将整条信息作为一个表单进行提交。数据库一一对应传参，录入数据。同时也会反馈录入结果，以供管理人员查看。也即在录入成功后，跳转至API查询结果页面，显示API的具体信息。

### 4.4 界面设计

通过前文的需求分析，在界面设计部分需要充分考虑与用户的交互。本文给出如下界面设计原则：

* 平台所有界面风格应该统一，采用相同的背景图、色调、字体格式等。为了界面的美观性，平台采用时下流行的深色模式。
* 平台的菜单栏位于界面顶部居中，组织起平台的所有功能逻辑。
* 当鼠标悬置于菜单栏选项，或其他需要选中的选项时，选项字体阴影颜色加深，给出操作响应反馈。点击实现对应跳转。
* 搜索框、输入框等应在页面上下、左右居中显示。大小应合理，并尽可能容纳对应的输入文本内容。
* 基于所需要输出显示的API及其映射信息量较多的特性，输出文本应根据对应的类型和内容合理布局。使得所有信息量既能够被全面完整地展示，也需要使得界面清楚美观，方便阅读。

### 4.5 数据库设计

#### 4.5.1数据库：数据表集合

在application.properties里的配置中，连接本地数据库all\_api\_map：

**spring.datasource.url**=**jdbc:mysql://localhost:3306/all\_api\_map?use**

**Unicode=true&characterEncoding=gbk&serverTimezone=GMT****spring.datasource.username**=**root  
spring.datasource.password**=**password  
spring.datasource.driver-class-name**=**com.mysql.cj.jdbc.Driver  
logging.config**=**classpath:log4j2.xml**

在数据库all\_api\_map中共有5个表单，分别存放了：

1. class\_table：Java和Swift两种语言的所有类（Class）的具体信息
2. apijava\_table：所有Java API具体信息的集合
3. apiswift\_table：所有Swift API具体信息的集合
4. apimap\_table：Java—Swift一对一映射关系
5. user：具有平台管理权限的用户信息

这5个表单集合了本平台所有的数据信息，是本平台的信息查询、映射搜索以及管理等功能的基础与关键。

#### 4.5.2类表

在本平台中，一条API数据即为一个类\_具体方法。在类表中，存储了Java和Swift两种语言API之中的类，及其包含的具体方法。

这里设计类表的初衷是，当用户在查询API信息时，如果出于对API的不熟悉和不了解，而无法获得准确的输入关键词，可以先去类的集合中阅览，再进一步地了解、查看每一条API。这样，可以更好地熟悉平台，快捷而迅速地得到所需要查询的API信息，或是映射。

一条类数据共包含：类名、类的语言类型、该类所包含的所有方法名称、类所包含的方法个数，以及对应的编号。

在class\_table中共有6个表字段：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| className | type | packageName | methods | metNum | Id |
| 类名 | 该类属于Java/Swift | 包名  （只有Java有） | 该类所包含的所有方法名 | 方法  个数 | 序号 |

此表共包含Java类17个，Swift类若干。

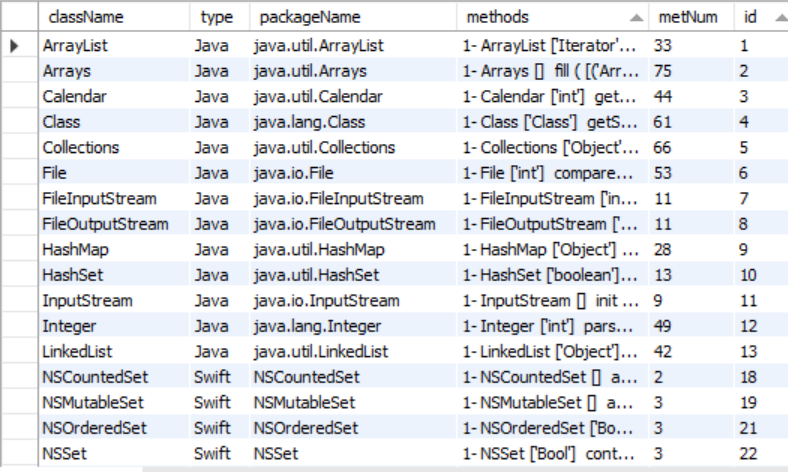


图4-3 Workbench可视化管理class\_table

#### 4.5.3 Java API表(apijava\_table)

此表所存储的也即本平台的核心数据：API。

每一条API即为形如className [return] methodName(parameter)的一条方法，例如：HashSet ['boolean'] add ( [('Object', ['e'])] )。

出于对映射关系的考虑，将Java API与Swift API分置两表，各自存储。一条Java API包含以下字段：

1. fullName：即该API的全称（每一条API互不相同），信息量包含类名、方法名、返回值和形参。例如：HashSet ['boolean'] add ( [('Object', ['e'])] )
2. apiNameForSearch：即用于搜索该API的关键词。出于对API全称fullName复杂度的考虑，发现API全称fullName并不适用于作为搜索关键词。因此，平台统一了API搜索格式：类名\_方法名()。例如：HashSet\_add()。

\*需要注意的是，用于搜索的这个简称有可能是重复的。在实际API的统计与存储之中，不同的参数和返回值（即使功能相同）算作不同的API。而在简称之后，有可能出现多个同名API，需要进一步查询具体API。

1. packageName\className\methodName：对包名、类名、方法名做了进一步的细分。
2. apiSignature：API格式统一的特定标识（包名$方法调用）

例如：java.util.Hashset$public boolean add(object e)

1. description：对API功能与实现的具体描述。例如：HashSet add()的功能描述是Adds the specified element to this set if it is not already present。
2. formParameter/returnParam：该方法参数/返回值的具体描述。包括参数类型与其实际的数据意义描述。
3. codeSampele：代码样例。\*不是所有的API都有，部分API此字段为null。
4. apiId：数据类型为int。并非仅仅是序列号，而是每一条API的专属标识ID，在映射联表中作为Java与Swift各自的API标识信息，组成映射对。

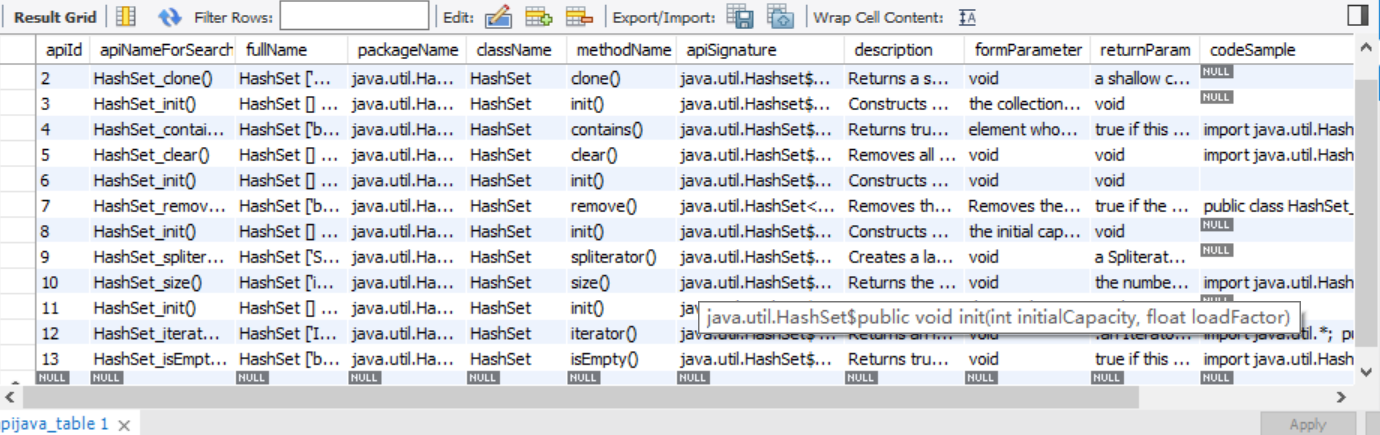


图4-4 Workbench可视化管理apijava\_table

数据库共存储有Java API数据674条。

#### 4.5.4 Swift API表(apiswift\_table)

Swift API表(apiswift\_table )基本上与Java API表相同，存储的是Swift API包括apiSignature、description等在内的全部信息。

需要注意的是，Swift并不像Java有包package，但为了格式的统一与输入输出的方便，Swift API仍保留了packageName包名这一字段：这里一律填写为Swift API的类名以作替代。

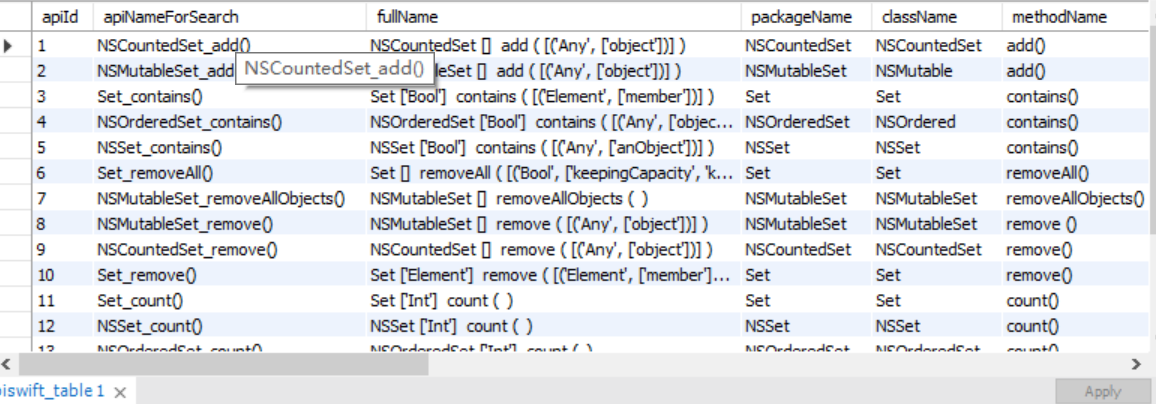


图4-5 Workbench可视化管理apiswift\_table

#### 4.5.5 API映射联表（apimap\_table）

API映射，指给定一个Java API，找出在Swift中实现了与之相同功能的API序列T，可能存在一对一，也可能存在一（Java）对多（Swift）的映射（当然，也可能不存在映射）。对于存在的API映射关系，同时也给出了在相同的测试用例下编写的可运行代码。

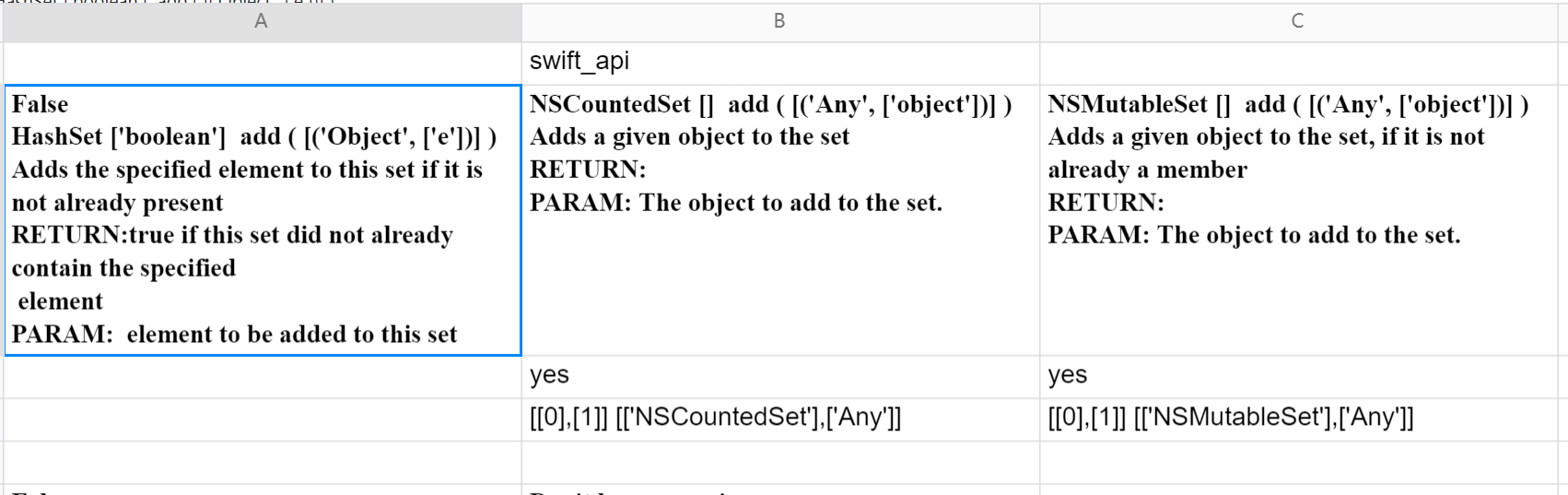


图4-6 API映射的一组原始数据

鉴于前文中，已将每一条API数据提炼出来，并分别存储在apijava\_table和apiswift\_table当中。当用前文中的apiId也即API标识，对这组原始数据进行再次加工后可得到映射：

|  |  |  |
| --- | --- | --- |
| Java API apiId | Swift API apiId | Swift API apiId |
| 1 | 1 | 2 |

出于对API映射双向（Java—Swift）索引功能实现的考虑，将上面这个一（Java）对多（Swift）的映射再次拆分为：<1，1> | <1，2> 共两组一一对应的<Java，Swift>映射。

将以上一一映射的数据，存储在数据表apimap\_table当中，apijava\_table与apiswift\_table中的API映射关系就此建立。根据API唯一标识apiId即可访问具体映射。由此，Java API与Swift API实现了双向索引。

#### 4.5.6 用户表单

基于API映射管理功能的需求特殊性，平台要求具有管理员权限，才可以进行管理相关功能的操作。因而，除了API及其映射的相关信息以外，数据库还另外建立了用户数据表。

一条用户信息包含三个字段：

1. User\_id 用户id序列
2. User\_name 用户名称
3. User\_password 用户密码

在表中，直接预存了管理员账户User\_id 1，其用户名称为njucs，用户密码

为lovenjucs。当需要访问平台的管理功能时，需要先登录、匹配以取得管理员权限进行后续操作。

当然，User用户表单还可以存储更多的用户。但是平台并未提供注册功能，需要具有后台权限，直接在数据库all\_api\_map.user中添加。

### 4.6 难点分析

基于以上对于平台的界面设计，功能设计，以及数据库设计的介绍，本文将以Spring MVC为框架，进行Web平台的开发与实现。

在实际的开发与实现工作中，需要解决如下难点：

1. 接受识别用户的多种输入

除了点击操作外，平台使用的核心基于用户的输入操作。但是由于用户的输入具有不可预测性，需要对用户的输入进行一定的规范和引导。首先，应当设置便于用户理解和输入的规定格式，其次要对用户的所有输入进行预处理，最后要根据用户输入给出对应的反馈，对于非常规输入也应进行判断和处理，以避免应用的崩溃。

1. 良好的UI界面的设计

作为一个面向用户的在线查询与管理平台，需要带给用户操作逻辑清晰，界面美观舒适的使用体验。更是需要适当的操作提示与引导，方便新用户的上手。在呈现提供给用户所搜索查询到的数据时，也需要有良好的布局排版，全面清楚的呈现。由于平台功能细分较多，还需要具有统一主题的设计风格，维护平台的统一性。一个良好的UI界面设计非常重要，更是需要精选的设计和统筹的考虑。

1. 合理高效的算法，对数据进行操作

用户通过Web在线平台所进行的操作，本质上是对数据库的增、查等操作。用户从输入数据到得到反馈的响应时间，应当尽量缩短。一个合理的数据库表单设计，高效的增、查算法是平台实现的关键。

1. 平台的整体组织架构与前后端交互

尽管本次工作所使用的技术框架逻辑层次鲜明，极大地方便了前后端的交互处理，和代码项目的组织管理。但是在整个开发工作中，仍需要宏观把握平台的整体组织架构与开发项目的工程代码管理，具有一定的要求与难度。

## 5 平台的开发与实现

本章介绍了本次工作的开发框架与环境搭建。并分别详细介绍了前端和后端的实际开发过程以及关键代码。

### 5.1 开发框架环境搭建

·硬件设备：Dell xps13笔记本电脑



图5-1 系统信息

·IDE：Intellij Idea

·数据库：MySQL5.7 \ MySQL Workbench

·Java version：1.8.0\_201

Java(TM) SE Runtime Environment (build 1.8.0\_201-b09)

Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)

在一般的开发过程中 ，是直接将jar包放置在工程的lib文件下的。但由于其依赖的jar包版本的不同，不正确的引入可能导致版本兼容的问题。因此，这里使用Maven来管理依赖资源。

将依赖配置写入工程的pom文件中，并修改application.properties文件中相应的数据库连接配置后，即完成了整个开发框架环境的搭建。由于SpingBoot内置了Tomcat服务器，点击运行程序，即可在任意通用浏览器通过http://localhost:8080进行页面的访问。

### 5.2 平台实现框架

本平台的开发与实现综合利用了多种技术与框架：HTML与CSS实现了前端页面的内容显示与布局修饰，Thymeleaf实现了前后端交互和后端数据的动态渲染，Spring MVC为本次开发框架的核心，实现了后端的数据存储与对应功能，并通过MyBatis技术与SQL语句建立映射，实现了对数据库的连接和相应的数据库操作。本文亦使用了可视化数据库管理软件MySQL workbench对数据库进行管理。在4.3节与4.4节将以后端与前端两部分，分别对平台的具体实现过程进行详细介绍。

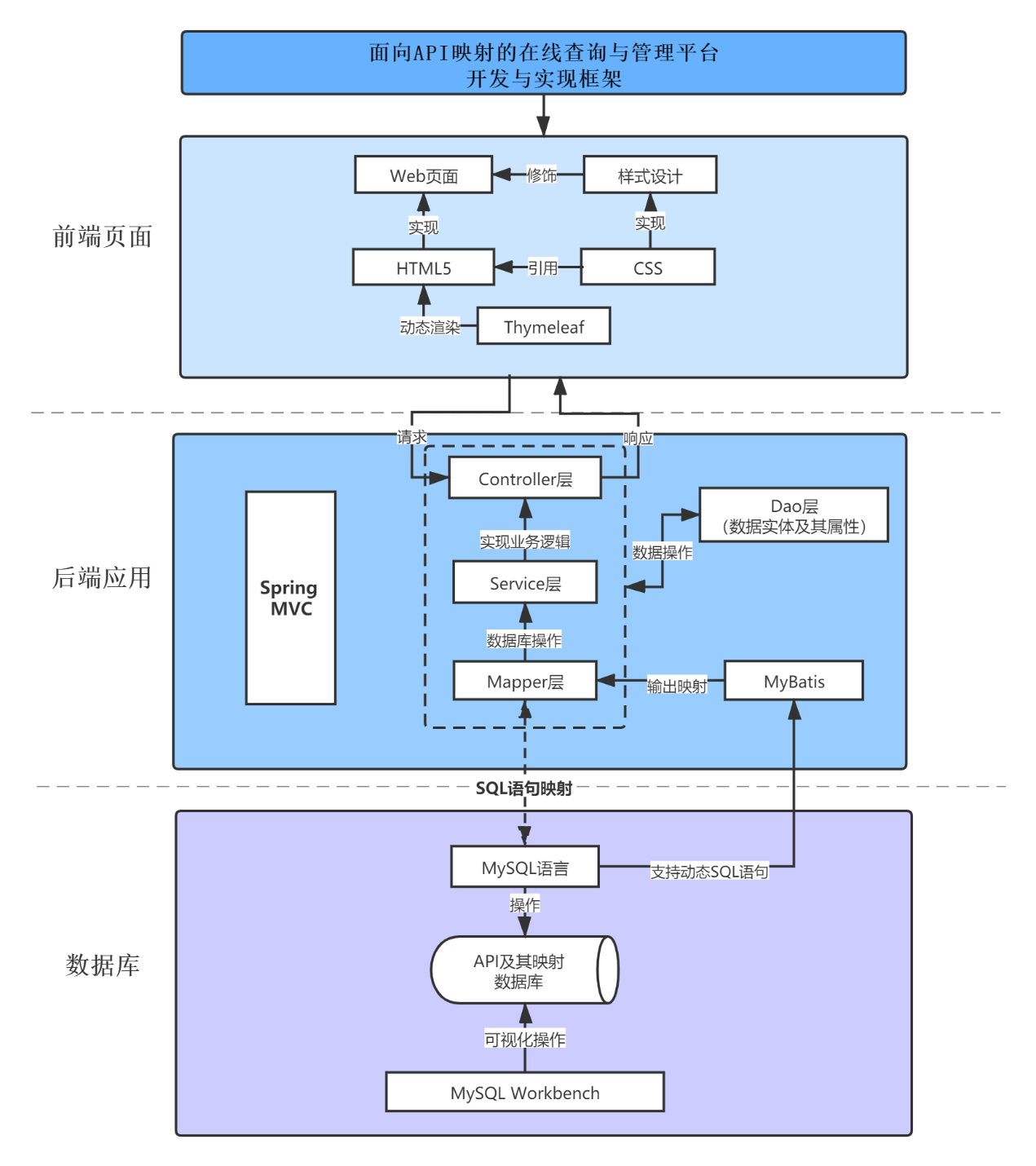


图5-2 平台实现框架

### 5.3 系统类结构

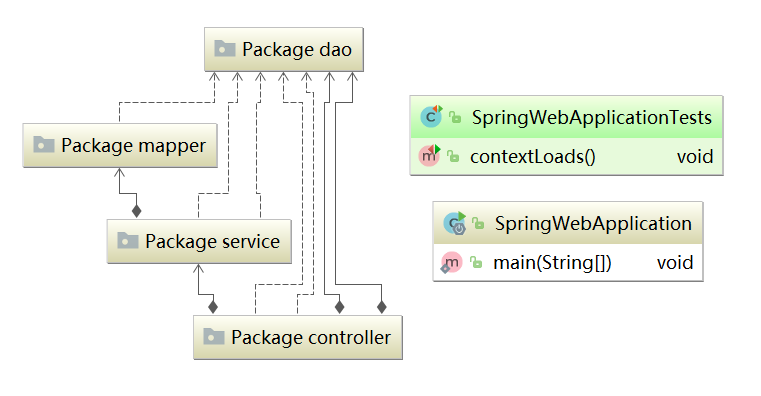


图5-3 系统类结构图

平台的后端开发为面向对象式编程，整个开发系统主要是构造、实现和调用一个个类。本文主要使用Spring MVC的架构，在此之中这些类主要分为四层，也即Dao层、Mapper层、Service层、Controller层，在代码项目中分别被归在对应的四个包里。被归在同一个包中的类为平行关系，各个类之间相互独立，分别实现了平台不同部分的功能。但在这四个包的类之间则有着紧密的依赖关系和组合关系，以此共同实现了平台的对应功能与跳转。

本文采用xxController、xxMapper、xxService、XX(Dao层定义实体及其属性)的命名方式构建类，对应实现平台的某一主体功能。以查询功能为例[图5-4]，Dao层包括ClassName、JavaSwiftAPI、ASetOfMap三个类，分别定义类和API这两个实体，以及一组API映射关系；Controller层包括SearchAPIController、MapAPIController两个类，Service层包括SearchService和MapService，Mapper层包括SearchAPIMapper和MapAPIMapper。Controller层、Service层和Mapper层的类都依赖于Dao层的三个实体类。Controller层的类与Service层的对应类为组合关系和依赖关系，Service层的类又组合并依赖于Mapper层的对应类。

整个系统的类结构以Spring MVC框架分层，逻辑关系清楚明晰。通过UML类图[图5-3][图 5-4]的描述，可以清楚地呈现整个系统，全面地表示类、接口与他们之间的协作关系。

关于Dao层、Mapper层、Service层和Controller层的介绍与其具体实现，将会在5.4节进一步展开介绍。

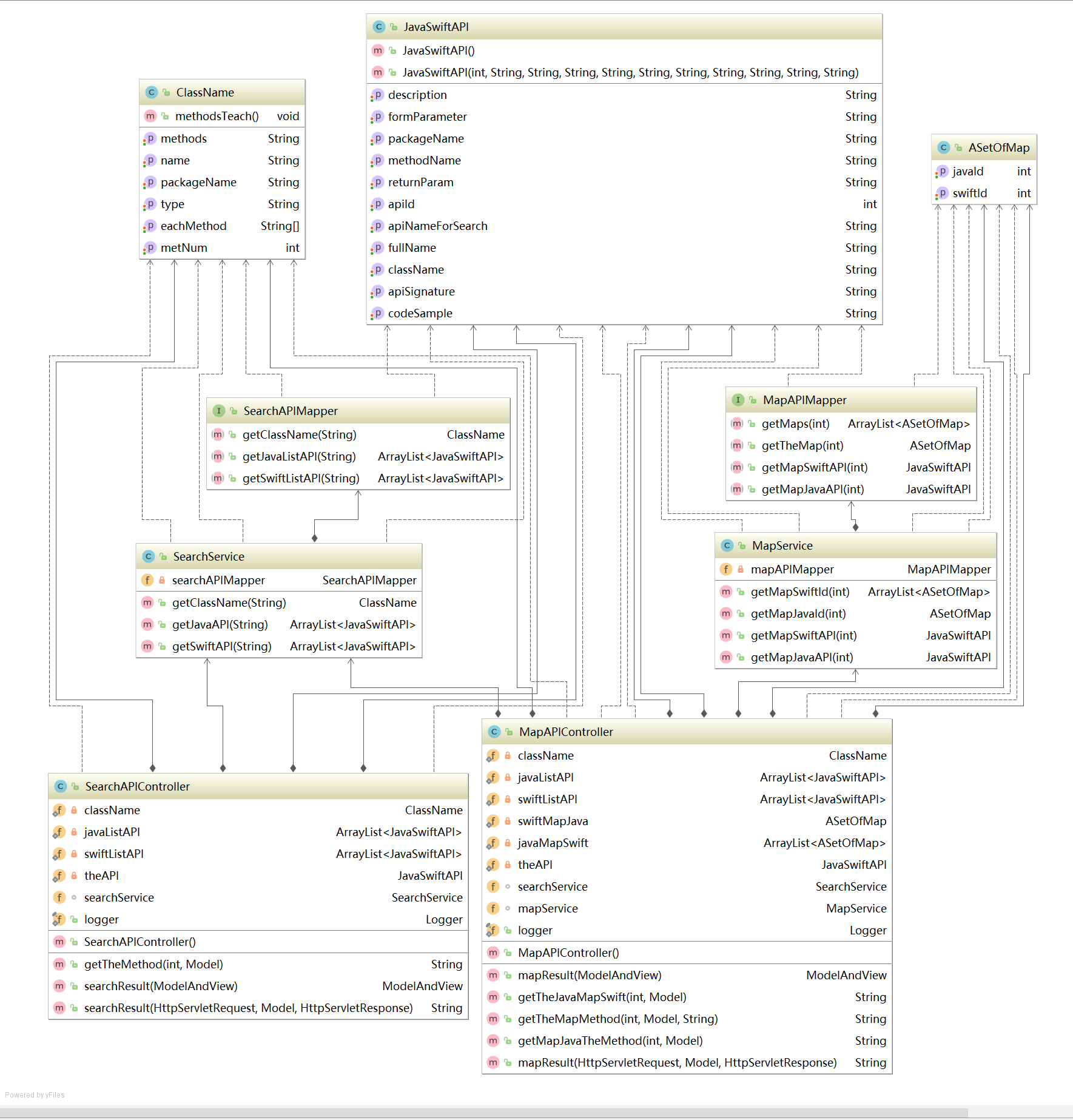


图5-4 系统类结构-查询功能部分实现细节图

### 5.4 后端：数据操作与请求响应

#### 5.4.1 Dao

Dao层即为Data Access Object层，也就是数据库交互层。基于MVC架构，Dao层的作用是封装对数据库的访问，只是达到按某个条件获得指定数据的要求。在这里定义了平台所有基础的数据实体及其属性。

应前文数据库表单的存储管理设计，这里主要分为：

1. AsetOfMap : 存一组API映射关系，也即一组<JavaApiId,SwiftApiId>的映射对，数据类型皆为int。

2. ClassName：class类的实体（名称、类型、包名、所包含的方法、所包含的方法个数）及对应的getX()和setX()的数据操作方法等。

3. JavaSwiftAPI ：Java和Swift的API实体（方法名称、类型、参数、返回值、apiSignature、描述、样例代码、apiId标识等）及对应的getX()和setX()的数据操作方法等。

4. HelloUser：具有管理权限的用户实体（id，用户名，密码）及对应的getX()和setX()的数据操作方法等。



图5-5 Dao层UML类图

#### 5.4.2 Mapper

在开发过程中，开发人员很容易发现原生MyBatis有许多痛点：比如在mapper.xml文件中有大量的SQL，如果数据库表字段变动，那么配置文件就需要修改了。而且SQL分页也需要自己手动实现，需要大量的传参。且有数据库的可移植性很差，要更新数据库的话，mapper.xml中的SQL往往要重新写。在这样一个冗余的开发模式下，往往又会产生大量的代码。

因而为了提高开发效率，简化开发流程，本文使用了只需要写好实体类，就可以支持相应的增、删、改、查方法的Mapper。

顾名思义，Mapper即“映射”。在MyBatis中会有一个SQL语句的配置文件，在运行时会加载SQL语句并映射相应的参数：SQL语句查询后返回的结果，会映射到配置标签的输出映射属性对应的Java类型。

在增、删、改、查配置标签中，有许多SQL配置是需要传递参数的，而通过MyBatis所封装好的接口，可以极为简单高效地将后端数据与对应操作，与SQL连接起来。比如，#{parameter}对应的就是后端函数的参数，它通过MyBatis注解，嵌入SQL语句，可以直接实现在数据库中的各种带参数的操作。

@Select(**"select \* from apijava\_table where apiNameForSearch = #{name}"**)  
@Results({  
 @Result(property = **"apiId"**,column = **"apiId"**),  
 @Result(property = **"apiNameForSearch"**,column = **"apiNameForSearch"**),  
 @Result(property = **"fullName"**,column = **"fullName"**),  
 @Result(property = **"packageName"**,column = **"packageName"**),  
 @Result(property = **"className"**,column = **"className"**),  
 @Result(property = **"methodName"**,column = **"methodName"**),  
 @Result(property = **"apiSignature"**,column = **"apiSignature"**),  
 @Result(property = **"description"**,column = **"description"**),  
 @Result(property = **"formParameter"**,column = **"formParameter"**),  
 @Result(property = **"returnParam"**,column = **"returnParam"**),  
 @Result(property = **"codeSample"**,column = **"codeSample"**)  
})  
ArrayList<JavaSwiftAPI> getJavaListAPI(String name);

以上面这段代码为例，后端需要通过API的名称去数据库中查找对应的API，构造了函数getJavaListAPI（name），而后通过@Select和@Result注解，来映射查询结果集到实体类ArrayList<JavaSwiftAPI>。

这里还需要注意的是，如果需要传递多个参数，则需要@Param注解，而非简单地直接用#{}取参。因为当只有一个参数时，传进去一个值也只有一个参数可以匹配。当存在多个参数时，传进去的值无法进行区分，则会出现：映射文件无法获得传入的参数的情况。

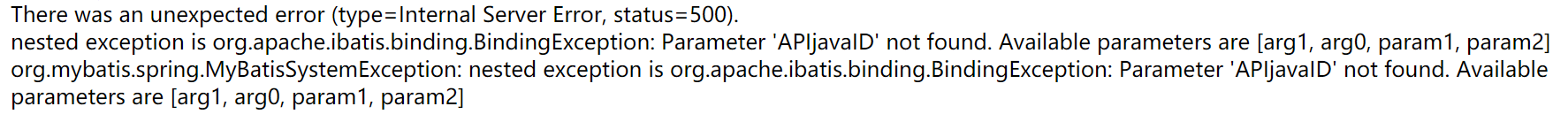


图5-6 报错：无法解析参数

如此，后端数据与数据库数据可以以非常简单的方式进行交互。本文中所有的数据增加、删除、修改、查询等操作都在Mapper中编写，实现了后端对数据库的直接控制与操作。

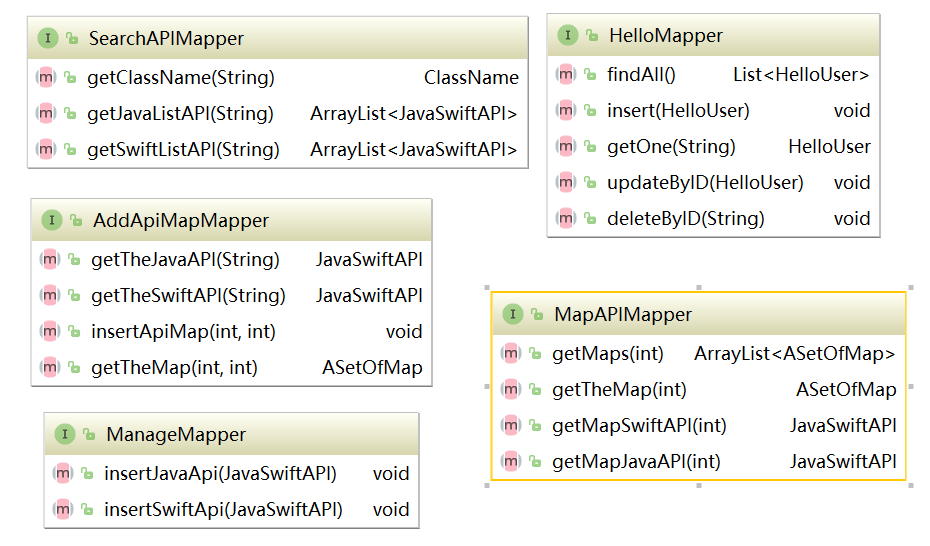


图5-7 Mapper层UML类图

#### 5.4.3 Service

Service层是业务层，实现了接受请求后的业务逻辑，其本质上是业务逻辑粘连了Mapper层和Controller层，是从二者之中解耦出来的独立层。它封装了一些通用的业务逻辑和操作（比如一些数据的检验等）；当然，它也可以与数据层进行交互，或者处理一些其他比如第三方API的请求等等。

在本文中，Service作为业务逻辑的关键一层是不可或缺的，但相关操作比较简单，属于过渡层。

以信息查询功能部分的Service层代码为例：

在Service层当中，对Mapper中实例化的相关数据库操作函数，进行了二次封装，实现了与Controller层的对接和数据交互。在Controller层调用该层的函数，即可对数据库直接进行增、删、改、查等相关操作，通过函数参数传递实现数据的交互。

@Service  
**public class** SearchService {

@Resource  
 **private** SearchAPIMapper;  
 **public** ClassName getClassName(String name){  
 ClassName classNameResult=**new** ClassName();  
 classNameResult = **searchAPIMapper**.getClassName(name);  
 **if** (classNameResult!=**null**){  
 System.***out***.println(**"getClass!:"**+classNameResult.getName());  
 }  
 **return** classNameResult;  
 }  
  
 **public** ArrayList<JavaSwiftAPI> getJavaAPI(String apiNameForSearch){  
 ArrayList<JavaSwiftAPI> javaListAPI=**searchAPIMapper**.getJavaListAPI(apiNameForSearch);  
 **return** javaListAPI;  
 }  
  
 **public** ArrayList<JavaSwiftAPI> getSwiftAPI(String apiNameForSearch){  
 ArrayList<JavaSwiftAPI> swiftListAPI=**searchAPIMapper**.getSwiftListAPI(apiNameForSearch);  
 **return** swiftListAPI;  
 }}

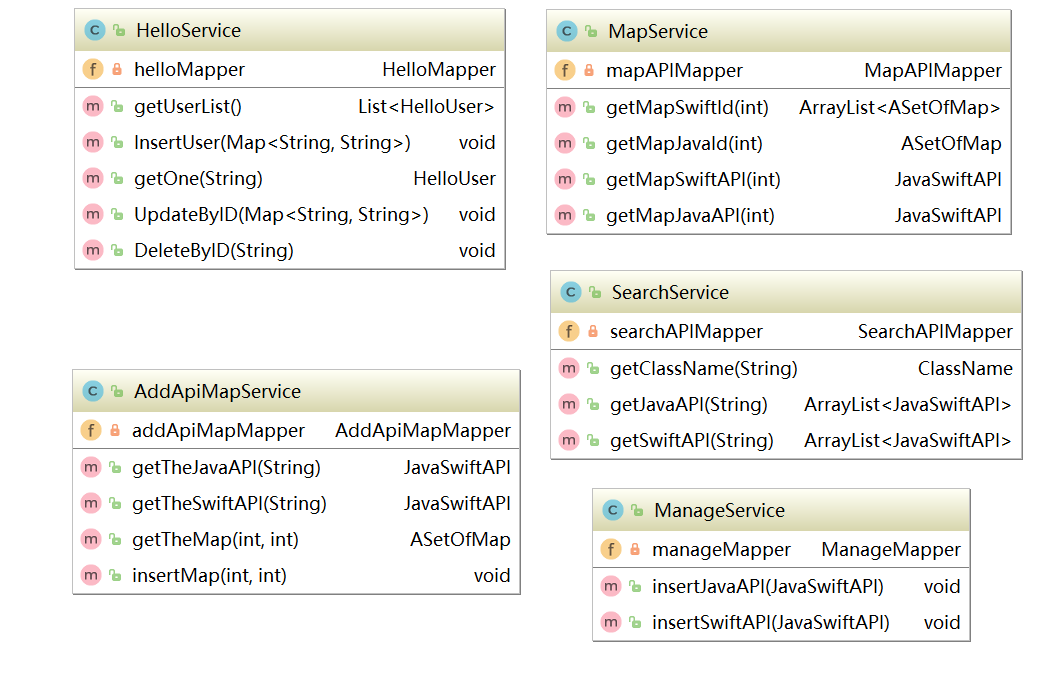


图5-8 Service层UML类图

#### 5.4.4 Controller

Controller控制器是本文编程框架中最重要的一层，它负责Web应用开发中最重要的请求处理与结果返回。Controller处理由DispatcherServlet分发的请求，它把用户请求的数据经过业务处理层处理之后封装成一个Model，然后再把该Model返回给对应的View进行展示。Spring MVC提供的Controller方法非常简单，无需继承特定的类或实现特定的接口，只需要使用@Controller注解标记类，就可以使得Controller被外界访问到。

当使用@Controller注解去标识一个类时，其实就是在告诉Spring MVC该类是一个Handler控制器类。当Spring初始化Bean信息时，会扫描到所有标注了@Controller注解的类，并将其作为Handler来加载。

在类中必须实现处理请求逻辑的方法，并且使用注解标注它们处理的URL路径。比如@RequestMapping和@RequestParam等，表明该方法是一个处理前端请求的方法。

@RequestMapping为控制器指定可以处理的URL请求，它不仅可以放置在方法上，也可以放置在类上。且有它还可以限定请求方法，请求参数，请求头。其method属性可以指定“GET”或“POST”请求类型。作为HTTP请求的两种基本方法，GET是从指定的资源请求数据；而POST是向指定的资源提交要被处理的数据。

当用户在页面触发某种请求时，一般会将一些参数带到后端，此时则可以通过参数绑定，将客户端请求的数据绑定到Controller方法的形参上。

@RequestMapping(value = **"/result"**,method = RequestMethod.***POST***)  
**public** String searchResult(HttpServletRequest request, Model model, HttpServletResponse response) **throws** IOException {  
 String searchText=request.getParameter(**"searchText"**);

以API信息查询功能的跳转逻辑为例，当用户在搜索框输入关键词，并提交表单后，通过request的getParameter( )方法得到输入信息，并将其作为后续搜索，显示API信息具体结果等系列操作的线索词。

在Controller中可以调用Service封装好的数据库操作方法。在对输入文本做简单处理后，即可通过调用Service中的函数，对数据库进行查询等相关的操作，得到所需要的数据信息。

model.addAttribute(**"fullName"**,*javaListAPI*.get(0).getFullName());  
model.addAttribute(**"packageName"**,*javaListAPI*.get(0).getPackageName());  
model.addAttribute(**"apiType"**,**"Java"**);  
model.addAttribute(**"Signature"**,*javaListAPI*.get(0).getApiSignature());  
model.addAttribute(**"description"**,*javaListAPI*.get(0).getDescription());  
model.addAttribute(**"codeSample"**,*javaListAPI*.get(0).getCodeSample());  
model.addAttribute(**"returnParam"**,*javaListAPI*.get(0).getReturnParam());  
model.addAttribute(**"formParameter"**,*javaListAPI*.get(0).getFormParameter());  
**return "methodresult"**;

最后，得到所需要的所有后端数据，并通过model.addAttribute( )方法将数据传递到前端，并跳转至结果页面。所有被传递的数据将通过Thymeleaf被动态地渲染，并显示在该页面。如此，一次搜索查询过程完成。



图5-9 Controller层UML类图

### 5.5 前端：界面渲染

前文中已经详述过本文所采用的前端界面渲染技术：HTML+CSS+Thymeleaf。

在本文中的设计与实现当中，基本采取一个页面对应一个HTML文件的映射关系，并引入css样式以使得界面更为美观。由于Spring MVC框架已经配置好，只需要在resources资源文件夹目录下的static和templates目录下，分别添加静态资源文件和HTML页面，项目则会自动进行加载。

HTML通过成对的标签如<head></head>等进行描述，支持HTML的浏览器会自动对HTML中的元素进行处理，并渲染页面。本文主要使用了HTML的<table></table>表格和<form></form>表单。

<form>标签用于为用户输入创建HTML表单，可直接向服务器传输数据。一个表单包含input元素（包括文本字段、密码字段、复选框、单选框、提交按钮等），也可以包含textarea、label等元素。textarea是用于多行文本输入控制的文本域，当所显示的或需要输入的内容过多，则需要使用textarea进行文本框扩展。HTML5表单提供了多种输入类型，具有更好的输入控制和验证，本平台的搜索查询、登录、管理等功能皆使用此标签实现。

当表单的input元素类型为submit时，如果用户单击确认按钮，表单的内容则会被传递到由表单的动作属性定义的目的文件，并由这个文件对接收到的输入数据进行相关的处理。

一个简单的HTML表格由<table>元素以及多个th表格头、tr表格行和td表格单元组成。这使得平台在查询搜索并反馈API信息结果时，可以清晰且有条理地呈现一条API数据的所有有关信息。当然，为了页面的美观性，平台也对页面的布局与着色排版等进行了设计和约束。

<**div class="mytable"**>  
 <**h1**>已添加API映射对如下：</**h1**>  
 <**table id="table-1"**>  
 <**thead**><**th**>Java API</**th**></**thead**>  
 <**tbody**>  
 <**td**><**a th:href="@{/addedmapresult(type=${'java'})}" style="color**: **white" th:text****="${JavaAPI}"**></**a**> </**td**> </**tbody**></**table**>  
 <**table id="table-2"**>  
 <**thead**><**th**>Swift API </**th**></**thead**>  
 <**tbody**>  
 <**td**><**a th:href="@{/addedmapresult(type=${'swift'})}" style="color**: **white" th:text="${SwiftAPI}"**></**a**></**td**></**tbody**></**table**></**div**>

<**form action="/result" method="POST"**>  
 <**div class="search-box"**>  
 <**input class="search-txt" type="text" name="searchText" placeholder="Type to search"**>  
 <**input class="search-btn" type="submit" name="serchButton" value=""**>  
 <**i class="fas fa-search"**></**i**>  
 </**div**>  
</**form**>

通过HTML标签约束，所有需要反馈的内容都会在页面上以表格形式清楚呈现，界面整齐美观。需要注意的是，这里采用了Thymeleaf动态渲染，需要通过th:text=="${Content}"对后端model.addAttribute( )的文本内容取参并进行动态渲染。此外，对于如查询到多个API，或是查询API映射关系这种仅会单单显示API名称的页面，本文亦使用了<a>标签，将此页面链接到其所点击API的具体信息页面，以供用户进一步查询。同样地，需要通过@{}和${}来进行加载和取参，以获取到用户点击确定的需所要查询的API信息。@{}是Thymeleaf中动态获取到静态资源文件的方法。

## 6 效果展示实例

本章以一个实际使用实例，展示了本平台的使用方式，各功能实现效果，页面显示及平台的一些细节。

### 6.1 主页

直接在搜索栏输入localhost:8080/homepage即可跳转到平台的主页，界面非常简单：给出了操作提示信息（输入格式），鼠标悬浮在问号图标引出搜索框，根据提示输入要查询的API类或方法即可。

为区分查询功能，在页面顶部菜单栏设置查询模块，可根据需求选择。选中某一特定功能时，面板下方显示红色线条选中标识。

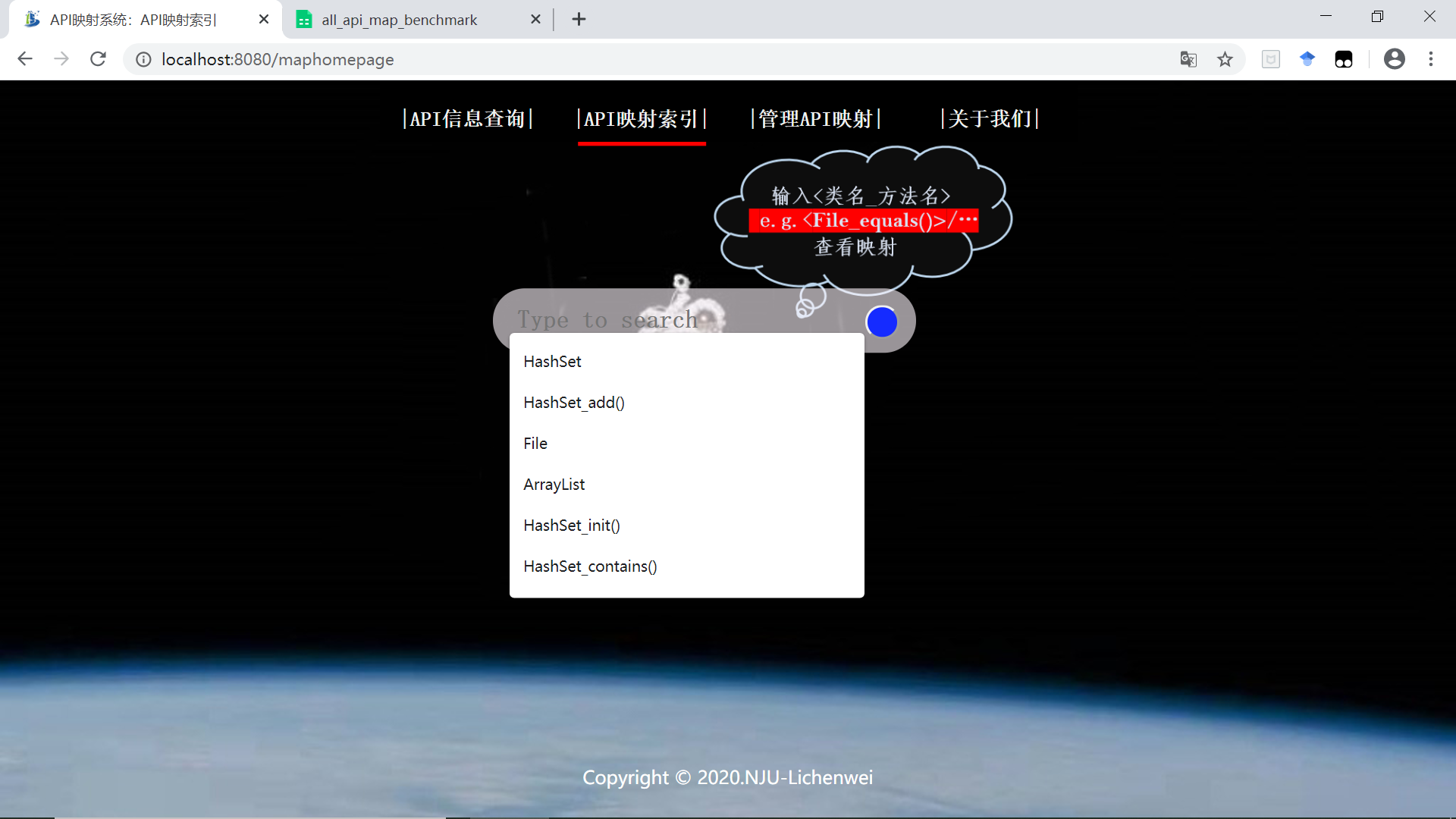


图6-1 主页2：API映射索引

### 6.2 API信息查询结果

API信息查询包括两种：

1. 输入类名，查看其所包含的各种方法。
2. 输入类名\_方法名，查看API的具体信息。

综合数据结果考虑采取，API信息查询结果可能内容较多，无法全部显示在页面，故而采取<table>和<textarea>结合的方式，布局页面。



图6-2 API类查询结果

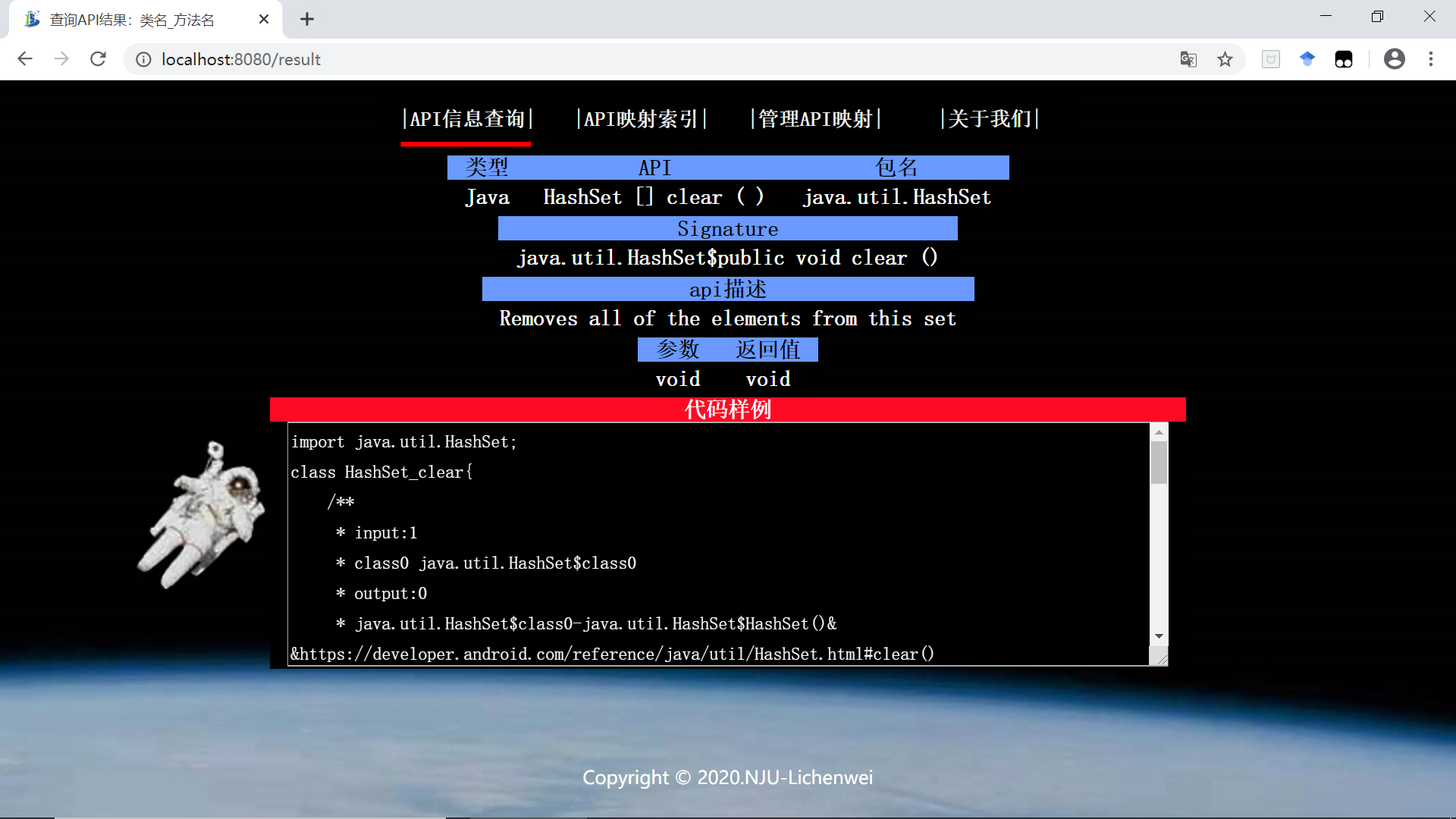


图6-3 API具体方法查询结果

前文曾提及到，基于输入便利性而设置的class\_method输入格式，可能会导致搜索多个同名结果的情况。以下图显示的HashSet\_init()为例，不同参数和返回值的API是独立存在的，故而需要进一步点击显示在页面的API名称，来查看具体的API方法。点击名称后，同上文的API查询一样，页面显示API的具体信息[图6-3]



图6-4 查询API信息：多个API同名方法

### 6.3 API映射索引

鉴于Java和Swift API名称没有重复的，因此，平台没有特别要求用户在输入时规定语言类型。与API信息查询的操作模式相同，在输入简易化的API名称class\_mothod()后，页面会以表格的形式呈现对应的：搜索API名称、映射的API名称。

在输入关键词搜索后，平台首先会就API进行类名搜索归类，自动判定输入的API类型，并去对应表单查询API方法及其映射，由此实现了API映射的双向索引。

同上文所提交到的，出于搜索关键词的简化而导致的：查询到多个同名API的问题，在这里仍然得到了解决。平台会首先跳转到同名API集合页面，并交由用户自行选择点击目标API，以进一步的查询到其对应的API映射。

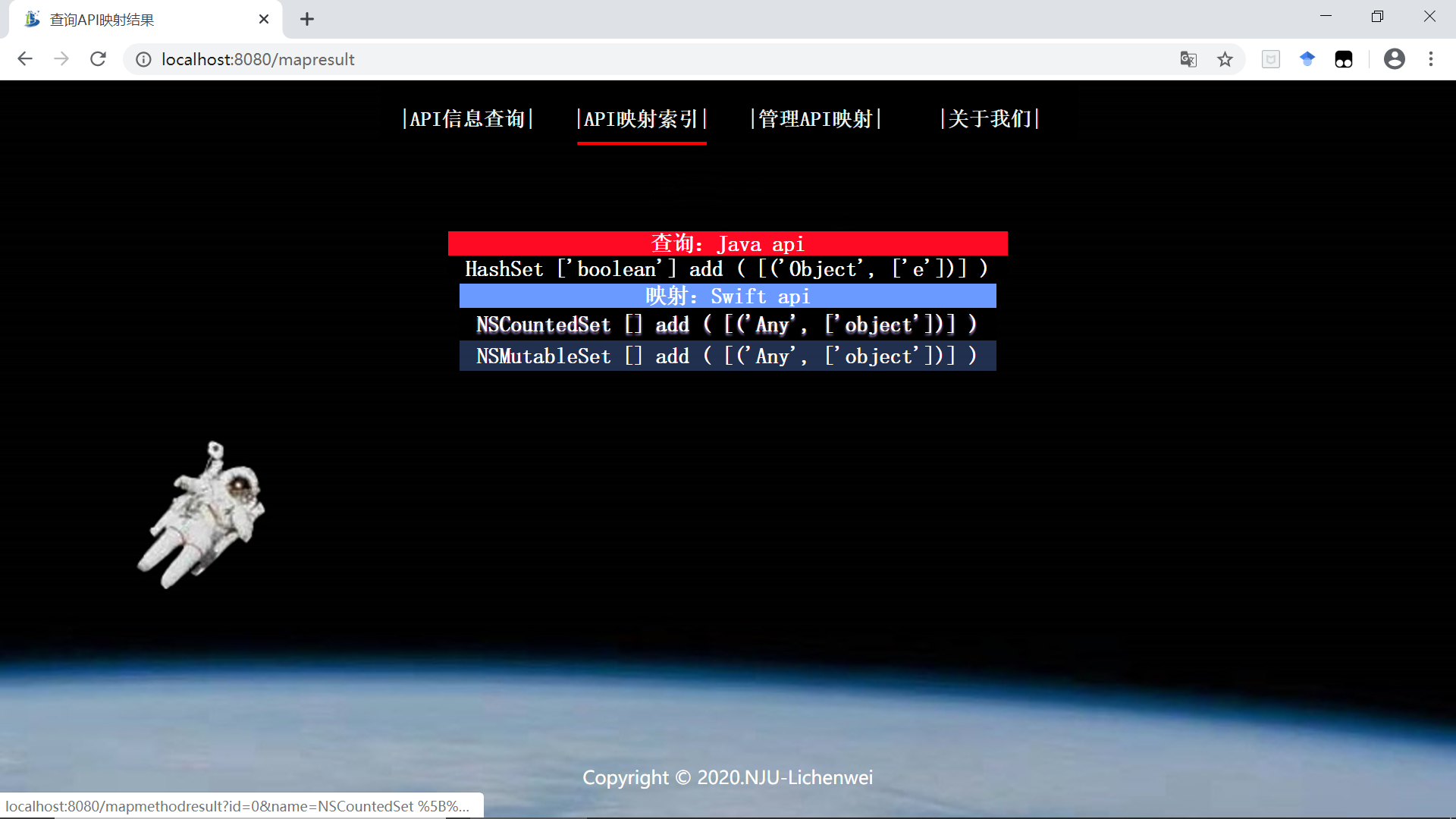


图6-5 Java API映射Swift API索引

如果仅仅是只显示API映射各自的方法名称，所带来的帮助性，以及便捷程度仍然是有所欠缺的。

故而，平台设计提供了点击API名称，进一步查询API具体信息的功能。界面会直接跳转到API信息查询结果页面[图6-3]。由此，用户可以仅需要：1. 输入API简化关键词；2. 鼠标点击查看阅览，共两个操作步骤，就可以快捷、简便地了解和熟悉API，以及API映射的所有内容和有效信息。

### 6.4 管理API映射

本平台提供了简单的API管理功能。但为了保证API及其映射数据查询的准确性，对该功能设置了管理员权限。也即只有通过登录，才能够进入API映射管理平台，并进行相关操作。

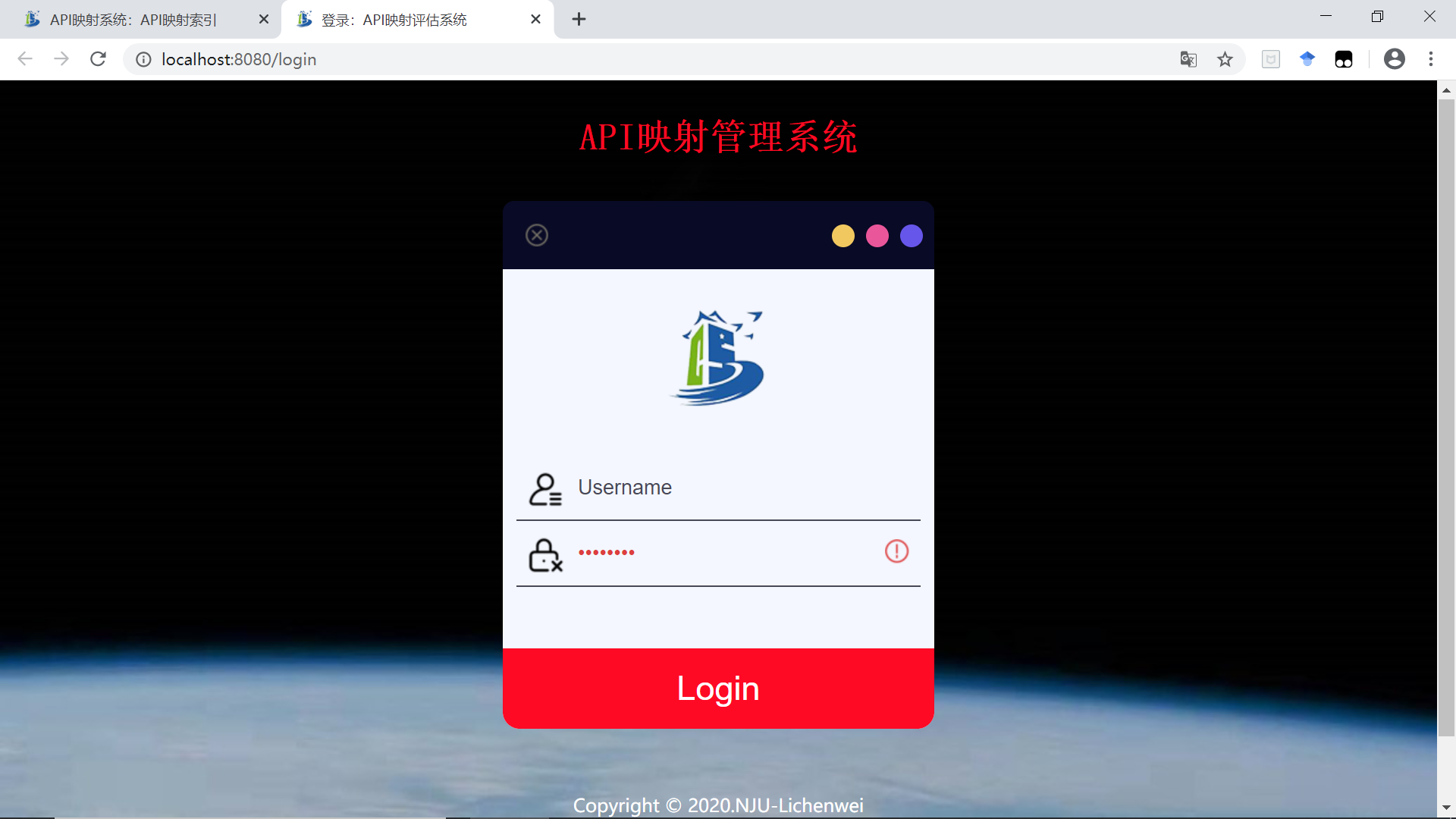


图6-6 API映射管理系统：登录界面

当用户名与密码验证正确后，将进入API管理界面，出于对API映射数据库稳定性的考虑，暂未提供删除和修改功能。即删除和修改需要更高的权限，直接通过MySQL Workbench在后台进行管理。

但为了方便API及其映射的添加，管理系统设定了API数据添加和API映射关系添加两种功能。当验证用户成功，直接跳转进入API数据添加页面：

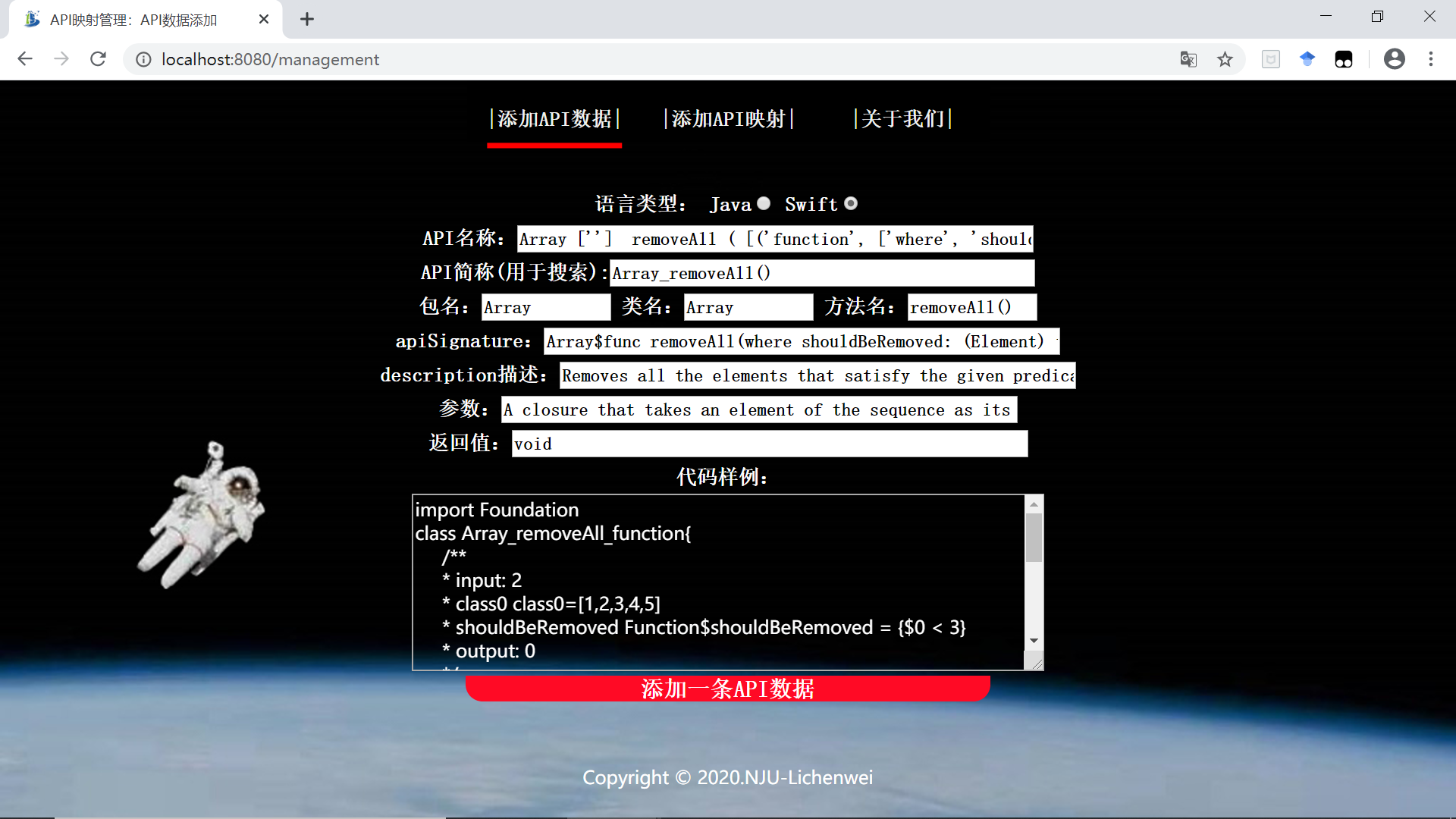


图6-7 API映射管理系统：添加API数据

目前API数据添加，仅限Java和Swift两种语言，用户需要进行勾选。当信息填入完毕，点击提交按钮，即可直接添加API数据至对应语言数据表。在确定提交前，会弹出确认框，让用户再次进行确认，以保证数据录入的正确性。录入成功后，数据库各数据表录入新数据，页面显示所录入的API数据信息[图6-3]。

同时，平台亦提供了API映射添加功能。为了维护数据库的稳定性与准确性，平台对API映射的输入做了严格的规范：输入为Java API和Swift API的准确全称（非查询时，为了方便而简化了返回值和参数信息的简称class\_method()）。

平台首先会就输入的API进行搜索判断，唯有当两种API都在数据库中时，才会添加映射关系。当然，平台也会对映射进行判断，如果该映射已在数据库中，则直接跳转页面显示API映射关系；否则，在数据库apimap\_table数据表中插入映射关系（也即所输入的两个API的apiId对），并反馈API映射结果。

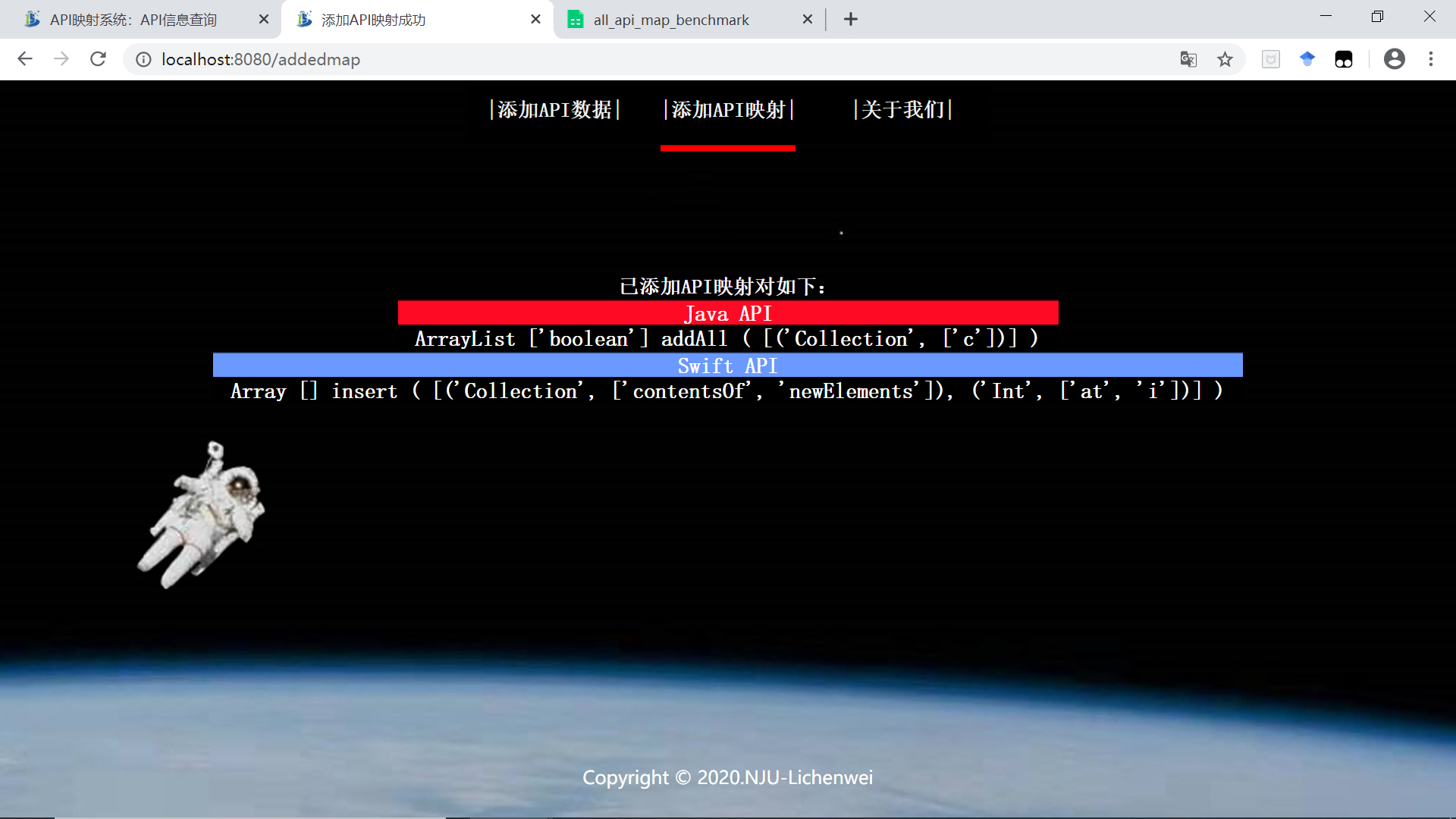


图6-8 API映射添加成功

与前文中API及其映射查询的功能逻辑相似，点击API名称仍然可以直接跳转到API具体信息页面，方便用户进一步了解所需要的API。

### 6.5 细节展示

平台采用了时下流行的深色模式，界面简单清爽，功能导航分类明晰。力求以最简洁和直观的方式，引导用户进行相关操作。

本文在此将例举、详述平台的一些细节设计：

·登录界面的视觉中心装饰，以及网页窗口icon借用南京大学计算机系Logo，加入了校园情怀。[图6-6]

·首页搜索框化用为宇航员头盔，以问号图案作为引导，界面协调有设计感。当鼠标悬置于搜索框上方后，弹出完整搜索框，提供给用户进行输入查询。

·当跳转至管理平台登录界面时，重新加载新的窗口。查询功能和管理平台各自独立，平行使用，互不影响。

·为保持界面统一的简约设计风格，所有用户提示信息皆以弹窗方式呈现。

## 7 应用实例测试与评估

本章将对面向API映射的在线查询与管理平台进行功能测试，获取它的程序执行时间，并对对它的时间复杂度进行估计和计算，综合地对本次开发工作进行评估。

### 7.1 功能测试

1. 页面跳转响应测试

这里主要测试各界面控件操作所连接的功能，以及页面间的跳转是否正常。操作方式：依次操作全部控件，看控件是否可被操作，操作结果是否正常；依次点击菜单栏的所有选项，看是否能正常跳转至对应页面，以及是否能够正确加载；同时打开多个页面，看每一个页面是否能正常进行操作；多次回退页面，看页面加载是否正常，有无异常情况。

最终测试结果皆为正常。没有遇到页面无法加载，报错等情况。

2. 输入接收处理测试

这里主要测试对输入数据的处理和反馈，分为两部分：

a.输入按照平台要求的格式规范的查询信息，看页面是否正确地响应跳转。以查询功能为例，查看所输入的类、API以及API映射能否被正确地查询到；如果格式正确，但并没有对应的查询结果，则看平台是否给出对应的提示，以及能否正确跳转以供用户继续查询。

b.输入多个格式规范外的查询信息，看后台程序是否会因为数据解析错误，而出现死循环、界面错误跳转等严重问题。仍以查询功能为例，平台要求格式为输入class或class\_method()以查询对应的类，或者API方法。这里特别设计如：空输入\ efe\_dfa\_\_\_ \ \_cefa&&W@ \ fdaef\_\_defrr \ dfafda()\_等多种测试样例。

最终测试结果均为正确。所有的数据格式平台均能接收并处理，给出了正确响应，符合平台设计与实现时的输入格式全覆盖预期。

3. 结果输出正确性与页面布局测试

这里主要测试输出数据结果能否被正确地渲染与显示。

在平台的设计与实现中，无论是关于类具体信息、API方法具体信息、还是API映射结果的输出都以HTML中的表格作为布局。这里需要对输出的两个方面进行测试：a.由于数据是通过Thymeleaf动态渲染的，需要判断关于输出结果的每一项内容是否正确；b.因为各项数据是归纳在表格中显示的，不同API的数据量可能不同，因此需要测试当输出数据长度或内容超出文本域时，能否被全部地加载显示，以及表格布局是否清楚合理，有无乱码等情况的出现。

经过大量且反复多次的测试，得到：输出结果均为正确，页面的布局和呈现清楚整齐，测试结果较为理想。

### 7.2 响应时间与时间复杂度

在平台的实际操作与使用过程当中，几乎感受不到操作时延。但为了对平台进一步地进行评估，本文采用Java语言所提供的API‘System.currentTimeMillis();’获取系统时间，并以此计算了程序执行时间。



图7-1 查询功能部分响应时间

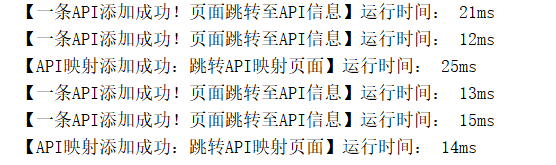


图7-2 添加功能部分响应时间

经过多次的实际操作并计算对应的程序响应时间[图7-1][图7-2]后，得出如下结论：

1. 当每次程序刚开始运行时，会需要一定的加载时间。这个时候无论进行任何操作，任何功能，响应时间都比较长，达到了700ms以上。

2. 由点击API名称查询具体API、Java映射Swift：API映射的具体方法，这两项的程序运行时间皆为0ms，可以得出：当没有直接对数据库进行操作（查、改）时，程序执行与页面响应时间是0时延的。（点击名称查看具体情况，仅涉及进一步地渲染，关于数据内容已经从数据库中找到并存在后端）

3. 查询功能部分响应时间基本上都在20ms以下；添加功能部分响应时间基本在10ms-30ms之间。整体来说添加功能部分耗时大于查询功能部分。经分析，这可能是出于添加时仍需要对数据库进行查找，以判断所添加的API是否已经在库中，避免重复添加而导致的。

整体来说，除了在程序刚开始运行时会出现一定的卡顿外，所有功能及操作响应时间皆为毫秒级。在实际使用时，无论是对用户的具体操作，还是相应结果反馈，平台的响应速度都非常快，几乎让人感觉不到时延。整个在线查询与管理平台运行流畅顺滑，程序及其效果评估为良好。

考虑到响应时间与数据库规模有关，数据库规模的扩大会导致对数据库相应操作的响应时间增加。基于对数据库及其表单的良好设计，目前各种操作的响应时间差别不大，基本在毫秒级。但由于本文的查询与管理算法，是在对输入语句进行解析后，直接用SQL语言进行对应操作，故而可以得出本平台的时间复杂度为O(n)，响应时间会随着数据集规模的增长而增长。目前数据库共存储API数据600余条，仅能提供一个较小规模的响应时间计算实例。后期随着数据库规模的不断增长，也可以继续就平台的响应时间与时间复杂度等相关内容，完成进一步的研究与探索。

## 8 工作总结与展望

### 8.1工作总结

本文详细介绍了面向API映射的在线查询与管理平台的具体设计与实现。

首先本文分析了当前多版本、多语言开发所面临的一些问题，分析了本文研究工作的背景情况，提出了搭建面向API映射的在线查询与管理平台的必要性，阐释了本文的研究与工作意义。

针对平台所使用的Spring MVC+ MyBatis +Thymeleaf等框架，MySQL数据库与前端文本标记语言HTML+CSS等有关技术，本文就开发所涉及到的所有方面给出了详尽的介绍。

本文亦从项目的需求分析、整体架构和平台设计，到最终的开发实现，与效果呈现，流程化地按序进行了详尽的说明与阐释。还给出了对应用实例的测试与评估。本文从宏观上呈现了相关研究与开发工作，也在细节上进行了全面细致地介绍。

最终，本文得到了一个面向Java—Swift语言API映射的在线查询与管理平台，能够实际地帮助到开发人员轻松在两种语言之间进行过渡，并对API映射的研究等相关工作的开展提供了一定的帮助与贡献。

### 8.2平台优化与未来展望

面向API映射的在线查询管理平台的设计与实现，使得需要进行代码迁移和多语言版本项目开发的用户，能够无需完全掌握各种语言及其API，即可轻松方便地查阅和了解API信息及使用方法，对两种语言功能或实现效果相同的API建立映射关系，轻松完成开发工作。

但是平台仍然有一些改进的空间：

一是目前仅支持Java&Swift两种语言的API及其映射。未来希望能够集成一个数据库强大的，能够支持所有主流语言的API及其中任意两种语言的API映射查询与管理。

二是目前的API映射关系仅仅建立在两种API方法的简单对应关系上。未来希望能够就具体使用时的细节进一步地细化。

三是为维护已建立的API数据库，平台仅提供了添加API功能，而不能对已有的API数据进行修改。未来应该进一步地加入第三人审阅系统，在二重校正与验证之下，对数据库的表单任意数据都可以进行各种操作与管理。

当然，面向API映射的在线查询管理平台的设计与实现，也为未来有关API映射的研究工作提供了新思路。它是数据研究内容的实际应用化，研究者们可以就API映射进行更深更广的探索，并应用于实际。也可以借用本平台，来作为日后研究工作的数据支撑。

希望未来的研究方向将是，平台开源化，平台数据库开放化。精通掌握着各种语言的开发人员与专业学者，都可以直接使用平台为自己的开发和研究工作提供帮助。同时，亦可以在使用平台的时候提供反馈，修正API映射数据的错漏，并提供更多建设性的建议，共同维护和建设一个庞大的面向API映射的数据库，维护一个公共的面向API的映射查询与管理在线平台。

## 参考文献

1. Kijin An, Na Meng, and Eli Tilevich. 2018. Automatic inference of Java-to-swift translation rules for porting mobile applications. In Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, MO-BILESoft@ICSE 2018, Gothenburg, Sweden, May 27 - 28, 2018. 180–190. https://doi . org/10 . 1145/3197231 . 3197240.
2. Alexandre FAU and Christian Mauceri.2019.Java 2 CSharp Translator for Eclipse.http://sourceforge .net/projects/j2cstranslator/. Accessed Nov. 4, 2019.
3. AhmedE.HassanandRichardC.Holt.2005. A light weight approach for migrating web frameworks. Information & Software Technology 47, 8 (2005), 521–532. https://doi . org/10 . 1016/j . infsof . 2004 . 10. 002.
4. 李正, 吴敬征, 李明树. API使用的关键问题研究[J]. 软件学报, 2018, 29(6): 1716-1738.http://www.jos.org.cn/1000-9825/5541.htm.
5. Chen, Chunyang , et al. "Mining Likely Analogical APIs across Third-Party Libraries via Large-Scale Unsupervised API Semantics Embedding." *IEEE Transactions on Software Engineering* PP.99(2019):1-1.
6. Pandita R , Jetley R P , Sudarsan S D , et al. Discovering likely mappings between APIs using text mining[C]// IEEE International Working Conference on Source Code Analysis & Manipulation.IEEE, 2015.
7. Lu Y , Li G , Zhao Z , et al. Learning to Infer API Mappings from API Documents[C]// International Conference on Knowledge Science, Engineering and Management. 2017.
8. Padmanaban, Yogesh. Learning API mappings for programming platforms[J]. Dissertations & Theses Gradworks, 2013.
9. Zhong, Hao & Thummalapenta, Suresh & Xie, Tao & Zhang, Lu & Wang, Qing. (2010). Mining API mapping for language migration. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10. 1. 195-204. 10.1145/1806799.1806831.
10. Zejun Zhang，Minxue Pan，Tian Zhang，Xinyu Zhou，Xuandong Li . Deep-Diving into Documentation to Develop Improved Java-to-Swift API Mapping. In 28th International Conference on Program Comprehension (ICPC ’20), October 5–6, 2020, Seoul, Republic of Korea. ACM, New York, NY, USA, 11 pages. Accepted.
11. 黄俊爽, 李聪, 李相俭,等. 浅谈Java面向对象程序设计[J]. 科技信息, 2010(13):469+471.
12. 杨雯. Swift语言研究[J]. 电子技术与软件工程, 2015, 000(011):P.254-255.
13. 舒礼莲. 基于Spring MVC的Web应用开发[J]. 计算机与现代化, 2013(11):171-172+177.
14. 徐雯, 高建华. 基于Spring MVC及MyBatis的Web应用框架研究[J]. 微型电脑应用, 2012(07):5-8+14.
15. Zhang D , Wei Z , Yang Y . Research on Lightweight MVC Framework Based on Spring MVC and Mybatis[C]// Proceedings of the 2013 Sixth International Symposium on Computational Intelligence and Design - Volume 01. IEEE, 2013.
16. Wei T , Center I , University L . Implement of Data Persistence Solution and Paging by Spring Data,MongoDB and Thymeleaf[J]. journal of longdong university, 2017.
17. 张诗雨, 陈汶滨. 基于索引的SQL语句优化方法探讨[J]. 计算机与现代化, 2009(03):138-140.
18. 崔秋丽. 基于HTML语言的网页制作方法[J]. 民营科技,2010,000 (012):P.66-66.
19. 沈茹. CSS样式表在网页制作中的应用[J]. 信息与电脑:理论版, 2011, 000(001):P.131-131.

**致谢**

随着这份毕业设计的完成，我的四年大学生活就此落下了帷幕。

在此，首先要特别感谢张天副教授。我曾经上过张天老师的《软件工程》《计算机程序语言》等多门课程，学到了许多专业的知识，并在实操中拥有了软工项目的经验，收获十分丰富。张天老师亦是我的毕业设计指导老师，他在我的整个毕业设计工作中，一直提供着无微不至的帮助与鼓励。是他的耐心引导和严格要求，让我顺利地完成了论文工作。从整体框架到语言和细节，张天老师都给了我详细的指导意见，使我的论文臻于完善。

其次要感谢母校南京大学计算机科学与技术系的培养。四年间我得到了各位老师的悉心教导，享受了学校和院系所提供的软硬件等各方面的资源与平台，并迅速地成长了起来。

还要感谢本系硕士研究生张则君学姐的帮助。她在硕士阶段对有关API映射的课题进行了深刻研究，提供了本平台所使用的数据基础。当我在做毕设的过程中遇到困难和瓶颈时，她也积极与我进行探讨，并给出了许多宝贵建议。

最后，感谢我的家人与朋友们一直以来的陪伴与鼓励，这对我来说至关重要。亲人的关心使我始终有一个温暖的避风港，让我能够无惧风雨地前行。感谢大学期间认识的室友们，好朋友史岳兰和崔凡荻，是她们的陪伴让我的大学生活更为丰富精彩，友谊是我一生的财富。