本科毕业论文（设计）

论文题目：API 相关概念的解释提取

姓　　名：刘辉　　学　　号：16302010048

院　　系：软件学院

专　　业：软件工程

指导教师：彭鑫　　职　　称：教授

单　　位：复旦大学软件学院

完成日期：2020　年　6 月　4 日

目录

摘要.....1
ABSTRACT.....2
第一章 引言.....3
1.1 背景.....3
1.2 主要工作.....3
1.2.1 数据准备模块.....4
1.2.2 词频统计模块.....4
1.2.3 分类器模块.....4
1.2.4 工具实现.....5
1.3 论文结构.....5
第二章 相关技术.....6
2.1 哈希算法.....6
2.1.1 哈希表.....6
2.1.2 布隆过滤器.....6
2.2 文本相似度.....7
2.3 卷积神经网.....8
2.4 相关开发框架和工具.....8
2.4.1 Scrapy.....8
2.4.2 spaCy.....9
2.4.3 Keras.....9
2.4.4 Vue.js.....9
2.5 本章小结.....10
第三章 设计与实现.....11
3.1 方法概览.....11
3.2 数据准备模块.....11
3.2.1 数据库表设计.....12
3.2.2 URL 去重.....13
3.2.3 网络爬虫.....15

3.2.4 Scrapy 管道.....16
3.3 词频统计模块.....18
3.4 分类器模块.....19
3.4.1 数据集构建.....19
3.4.2 数据集介绍.....21
3.4.3 模型搭建.....21
3.5 工具实现.....22
3.5.1 系统架构.....22
3.5.2 前端.....23
3.5.3 后端.....24
3.6 本章小结.....24
第四章 实验评估.....26
4.1 分类器评估.....26
4.2 系统评估.....28
4.3 本章小结.....29
第五章 总结与展望.....30
5.1 本文总结.....30
5.2 未来展望.....31
参考文献.....32
致谢.....33

摘要

开发者基于 API（Application Programming Interface）进行软件开发。当开发受阻时，开发者需要查询 API 文档，而文档中那些陌生的概念又会成为新的阻碍。此时，开发者需要花费大量时间从各种平台上搜索并检索概念的相关解释，以理解清楚概念。

本文将实现一套 API 相关概念解释提取系统，旨在节约开发者的时间，提高开发者的开发效率。该系统从各个平台上提取 API 相关概念的解释，并对这些数据进行处理、过滤以及提纯，最终以 Web 应用的形式为开发者提供查询接口。

为了实现上述目标，本文将所做的工作分成数据准备模块、词频统计模块、分类器模块以及工具实现这四个部分。其中数据准备模块以及分类器模块是系统的核心模块。数据准备模块由网络爬虫和 Scrapy 管道组成，网络爬虫负责从各平台提取概念解释，Scrapy 管道负责对爬虫提取的数据进行处理。数据准备模块中的一些关键问题在实现时有多种解决方案，本文选择基于 MySQL 存储的方式对 URL（Uniform Resource Locator）去重，并在爬取维基百科页面时选择深度优先搜索策略。分类器模块使用深度学习技术实现，该模块能够识别解释性文本和非解释性文本，用于提纯已收集的数据。

在解决了关键问题之后，本文描述了该系统的整体架构，给出了该系统各个部分详细的设计与实现，最后对整个系统进行评估。评估结果表明该系统适合于查询 Java 领域内的概念解释，不适于其他领域。

关键词：概念解释，API，爬虫，分类器

ABSTRACT

Developers develop software based on API. They need to browse API documents when development is blocked, but the new concepts in these documents will be new obstacles. In this case, they need to spend a lot of time searching and checking relevant explanations of concepts from other platforms to understand clearly.

In order to save developers' tome and improve their development efficiency, this thesis will implement a system that used to extract the explanations of concepts in API documents. This system extract explanations from various platforms, processes, filters and purifies these data. Finally, it provides a query interface from developers in the form of Web application.

To achieve the above goals, this thesis divides the work into four parts: data preparation module, word frequency statistics module, classifier module and tool implementation. data preparation module and classifier module are the core modules. The data preparation module is composed of web crawlers and Scrapy pipelines. Web crawlers are responsible for extracting the concept explanation from each platform, and the Scrapy pipelines are responsible for processing the data extracted by crawlers. There are several solutions to some key problems in the data preparation module. This thesis filters duplicate URL based on MySQL storage, and choose depth first search strategy when crawling the pages of Wikipedia. The classifier module is implemented by deep learning technology, which can recognize explanatory text and non-explanatory text, and is used to purify the collected data.

After solving the key problems, this thesis describes the architecture of the system, then gives the detailed design and implementation of each part of the system, and finally evaluates the whole system. The evaluation results show that the system is suitable for querying concept explanations in Java domain, not for other domains.

KEY WORDS: concept explanation, API, crawler, classifier

第一章 引言

1.1 背景

当一个组织或者个人向其他开发者提供开发工具包时，往往会为开发工具包中的 API 添加一定量的注释，以向开发者说明 API 实现的功能以及如何调用 API。除了给 API 添加注释外，开发工具包的提供者还可能提供 API 文档。而且对于一些工具包，互联网上也会有相应的使用教程。

正常情况下，开发者能够在开发工具包的源文件中找到想要的 API 注释。比如 JDK（Java Development Kit）中就包含有大量 Javadoc 格式的注释，开发者只要熟悉 Javadoc 常见标记的含义便能够比较快速且清楚地了解 API 实现的功能以及 API 的使用方法。当然有些开发工具包的源文件中可能没有注释，此时开发者就需要去其官网查看 API 文档或者搜索相应的教程。

不过当注释、文档或者教程中出现了一些新概念时，开发过程便会遇到阻碍。例如使用 Thread 类进行 Java 多线程实践，如果对线程相关原理和概念不熟悉，那么便需要了解 Thread 类中 daemon 属性的具体含义；使用 HashMap 类进行软件开发，如果对迭代器不了解，那么便需要了解 HashMap 类中 EntryIterator 内部类的具体作用。此时开发者就需要通过各种途径来搜寻与这个概念相关的解释，如 API 文档本身、维基百科以及搜索引擎搜索结果等。虽然开发者可以在这些平台上找到较为满意的解释，但是需要在各个平台之间进行切换，需要过滤掉那些质量低的搜索结果，还需要判断搜索结果是这个概念的解释还是这个概念的应用，这是一系列繁琐且耗时的操作。

1.2 主要工作

为了解决开发者在搜索 API 相关概念的解释时操作繁琐的问题，本文将实现一个 API 相关概念解释提取系统。并且为了提高系统针对性，本文将提取的概念解释限定在 Java 领域内。在开发之前，先要确定从哪些平台上提取概念解释。包含概念解释的平台众多，本文选择从权威的 Java 官方文档、数据组织良好的维基百科以及数据质量参差不齐的 Stack Overflow 这三个平台上提取与 Java 相关的概念解释。整个系统的开发分为两个阶段：第一阶段的主要开发任务为数据准备、词频统计以及数据提纯；第二阶段的主要开发任务为编写前端页面及实现后端 Restful 接口。

本文的主要工作为四个部分，分别是数据准备模块、词频统计模块、分类器模块以及工具实现。其中数据准备模块主要用于从 Java 官方文档、Stack Overflow 以及维基百科这三个平台上提取和处理概念的解释，在提取维基百科上的概念解释时需要使用到相似度管道。除此之外，此模块还用于提取维基百科页面的 URL 以及提取解释性和非解释性的文本。词频统计模块统计从 Java 官方文档提取的解释以及从 Stack Overflow 上提取的回答的词频，这些词频用于支持相似度管道的实现。分类器模块能够识别解释性的文本和非解释性的文本，用于判定从 Stack Overflow 上提取的回答是否为解释性的文本。工具实现部分以 Web 的形式提供了一个 API 相关概念的解释查询工具。

1.2.1 数据准备模块

数据准备模块是 API 相关概念解释提取系统的核心模块，此模块基于 Scrapy 框架实现，主要负责从 Java 官方文档、Stack Overflow 以及维基百科上提取并处理概念的解释。对解释的处理通过 Scrapy 的管道功能实现，此模块实现的各种管道分别用于过滤掉文本中连续的换行符、去掉文本首部及尾部的空白字符、过滤掉文本中连续的空格、对文本进行语法分析并去掉质量低的文本、将初步处理后的文本存储到 MySQL 数据库中、对维基百科上提取的文本进行相似度计算并丢弃相似度低的文本。

1.2.2 词频统计模块

词频统计模块是为了支持数据准备模块中的相似度管道而设计和实现的。该模块读取从 Java 官方文档提取的解释以及从 Stack Overflow 上提取的问题回答并进行词频统计，最后将统计结果存储到 MySQL 数据库中。词频高的词被视作为 Java 领域内的高频词，基于这些高频词就可以在提取维基百科时计算文本相似度以确定当前概念的解释是否属于 Java 领域。

1.2.3 分类器模块

分类器模块用于进一步处理从 Stack Overflow 上提取的问题回答。由于 Stack Overflow 上的问题回答大多是世界各地的开发者对于某个问题的理解，而且是在业余时间的讨论与交流，所以这些回答的绝大部分并不是精心组织过的，因而设计并实现分类器模块来去掉非解释性的回答。

1.2.4 工具实现

工具实现部分为开发者提供一个 API 相关概念的解释查询工具，这部分的主要任务为实现前端及后端。前端的开发工作主要包括：作为网站门户的主页面、显示用户搜索结果的搜索页面、展示概念解释详细信息的描述页面。其中主页面由网站 Logo 以及搜索框组成，搜索页面由网站 Logo、搜索框、显示搜索结果的分页组件以及热词组成，描述页面由网站 Logo 以及详细的概念解释组成。后端提供了三个 Restful 风格的接口，这些接口的作用分别是：为前端返回与某个概念相关的所有解释、返回热度前五的搜索词和增加某一个搜索词的热度。

1.3 论文结构

本文组织如下：

第一章引言，介绍本文的背景，就本文的主要工作及各个模块实现的功能进行了说明。

第二章相关技术，介绍本文涉及的编程框架和相关算法。

第三章设计与实现，就 API 相关概念解释提取系统各部分的整体设计和具体实现进行了详细介绍。

第四章实验评估，就分类器及整个系统的效果进行评估。

第五章总结与展望。

第二章 相关技术

2.1 哈希算法

哈希算法是计算机领域十分常见的一类算法，此类算法将任意输入转换成数值输出。本文涉及哈希表和布隆过滤器这两种应用哈希算法的数据结构。

2.1.1 哈希表

哈希表[1]计算输入元素对应的存储位置，并在该位置上插入、删除或者修改元素。现在简单估计哈希表所占内存空间的大小。假定哈希表中总共存储了 10000 个 URL，容量为 16384，存储 32 位的 int 数据，URL 的平均长度为 50，且不存在哈希冲突的情况。哈希表所占内存空间分为两个部分，第一部分为预分配的 int 数组所占的 64 KB 内存，第二部分为 URL 所占的 488.28 KB 内存，因此整个哈希表所占的内存空间约为 0.54MB 内存。

2.1.2 布隆过滤器

布隆过滤器实际上是一个比特数组，通过多个哈希函数分别对输入数据计算哈希值，然后执行相应的插入及查询操作。现在简单估计布隆过滤器所占内存空间的大小。假定布隆过滤器总共存储了 10000 个 URL，容量为 65536。由于布隆过滤器并没有存储 URL，只是通过相应的比特代表 URL 是否存在，因此其所占内存空间为 65536 bits，即 8 KB，相较于哈希表来说内存极大减少。布隆过滤器支持插入及查询操作，其插入操作如图 2-1 所示。

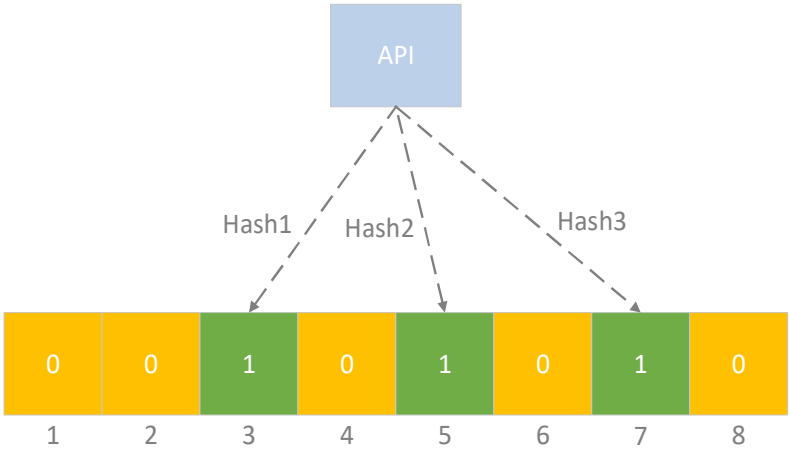

图 2-1 布隆过滤器插入操作

如图 2-1 所示，当插入字符串“API”时，三个哈希函数计算的哈希值分别为 3、5、7，且这三个位置均为 0，那么将这些值改为 1，插入成功。接着插入第二个字符串“explanation”，假设“explanation”对应的三个哈希值分别为 2、5、8，虽然位置 5 上的比特已经为 1，但是 2 和 8 这两个位置上的比特为 0，说明“explanation”不存在，可以安全插入。

接下来对布隆过滤器的查询操作进行简要说明。当由一个输入计算而来的三个哈希值对应的比特中有一个为 0 时，那么说明此输入不存在。如果由一个输入计算而来的三个哈希值对应的比特均为 1，由于存在哈希冲突问题，此时并不能说明该输入存在于布隆过滤器之中。

2.2 文本相似度

本文需要从维基百科上提取 Java 领域相关概念的解释，但维基百科收录的页面众多，包含了各个领域的知识，不可能将维基百科上所有的页面都爬取下来并进行处理，再组织成其他的形式进行展示。因而识别从这些页面中提取的与 Java 领域相关的文本很有必要。本文的解决方案是通过计算文本相似度识别 Java 领域内的概念解释。

文本相似度计算的方案有很多，本文主要探讨了编辑距离、LCS（Longest Common Substring）、余弦相似度、基于神经网络模型这四种解决方案，详细介绍如表 2-1 所示。

表 2-1 文本相似度计算方案

类型	方案	基本思想	优缺点
基于字符	编辑距离[2]	从 S_1 到 S_2，需要删除、插入、替换操作的最少次数。	计算准确，但是十分耗时。
	LCS[2]	最长公共子串。	原理简单，实现容易，但只适用于派生词等短文本，不适用于长文本。
基于词语	余弦相似度[2]	$\frac{\overrightarrow{S_1}\cdot\overrightarrow{S_2}}{\|\overrightarrow{S_1}\|*\|\overrightarrow{S_2}\|}$ （2-1）	实现简单，解释性强，不适用于语义相似度。
基于神经网络模型	Word2Vec[3]等	搭建深度学习模型，并对模型进行训练，利用训练好的模型计算文本相似度。	效果好，但要准备数据集以及相应的知识储备。

2.3 卷积神经网

卷积神经网 CNN（Convolutional Natural Network）[4]是深度学习领域中最基础、最重要的几个模型之一，其结构主要由输入层、卷积层、池化层、全连接层及输出层构成[5]。CNN 在处理如图像和文本等网格化的数据时有着突出的效果，图 2-2 为 CNN 图像识别模型的基本结构。

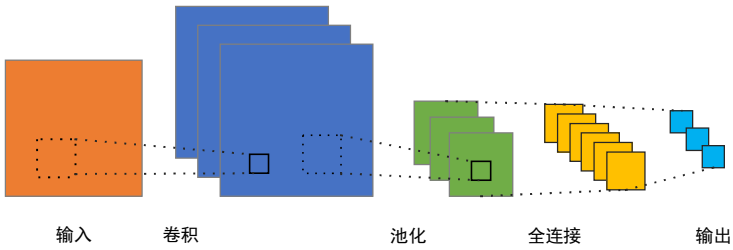

图 2-2 CNN 图像识别模型结构

本文运用 CNN 的卷积层、池化层以及全连接层构建深度学习模型，训练构建好的模型并对文本数据进行分类。

2.4 相关开发框架和工具

2.4.1 Scrapy

Scrapy 是一个高级的 Web 抓取框架，可以用于数据挖掘、监控以及自动化测试等诸多领域[6]。Scrapy 为开发者提供了非常多可供选择的特性，比如自定义数据类、自定义数据处理管道、自定义中间件、自定义数据输出格式等，开发者可以选择使用这些特性中的一个或几个来完成相应的开发任务。Scrapy 允许开发者配置代理、并发度、下载延迟、请求头等一系列参数，这些参数对于爬虫的性能有着显著的影响。图 2-3 为 Scrapy 架构图。

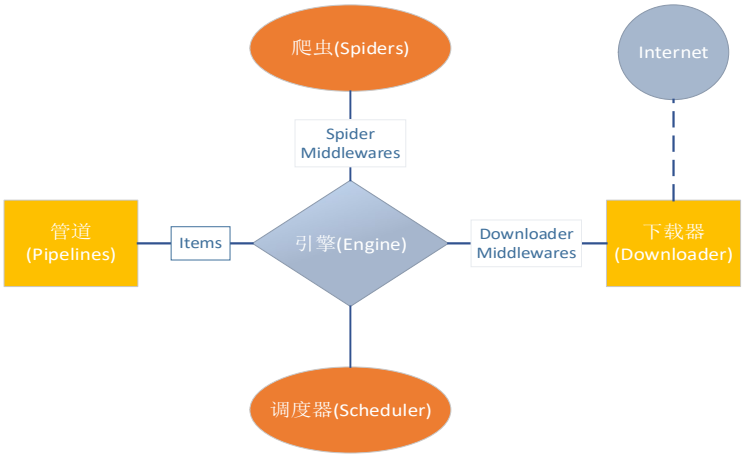

图 2-3 Scrapy 架构图

在整个 Scrapy 架构中，引擎是核心部分，负责在爬虫、下载器、调度器、管道之间传递数据，协调各部分之间的工作。爬虫是开发者需要实现的部分，用于提取开发者指定网站上的数据。对于爬虫框架的初级使用者来说，实现这部分即可。下载器向服务器发送 HTTP 请求，待服务器正确响应后，下载相应的网页。调度器接收由引擎转交的爬虫任务，并对这些任务进行处理，如过滤掉重复的任务、对任务重排序等。管道用于存储引擎传过来的 Item 形式的数据，存储的形式可以是文本文件，也可以存储到数据库中，具体的存储形式由开发者自己决定。

2.4.2 spaCy

spaCy 是一个用于自然语言处理的开源软件库[7]。在自然语言处理方面，spaCy 是一个开发者不得不去了解和学习的工具。它提供了诸多强大功能，如词性标注、命名实体识别等。合理使用 spaCy 提供的功能，开发的速度和效率将显著提高。

2.4.3 Keras

Keras 是一个用 Python 编写并着眼于快速实现原型的深度学习框架[8][8]。Keras 的核心数据结构是模型和层，其中层用于设计和实现神经网络中的各个层次，模型用于组织神经网络中的层。Keras 为开发者提供了 Sequential 模型及 Model 模型。Sequential 模型通过线性的方式组织神经网络中的层，各层之间只能是前后相连的紧邻关系。而 Model 模型则提供了更为复杂的组织方式，比如将定义好的几层拼接成为一层。Keras 中的层有多种，如接收数据的 Input 层、产生结果的 Output 层、设置激活函数的 Activation 层、汇聚局部信息的 Convolution 层等。

2.4.4 Vue.js

Vue.js 是用于构建用户界面的前端框架[9]。Vue 并不是一个臃肿的大型框架，其核心关注点在视图层，如果开发者想要更多的功能支持，可以通过 npm 安装第三方库对 Vue 项目进行扩展。Vue 有两大特性，分别是响应式及组件化。响应式是指当开发者修改数据时，与之绑定的页面元素也会改变。图 2-4 展示了 Vue 的响应式原理。

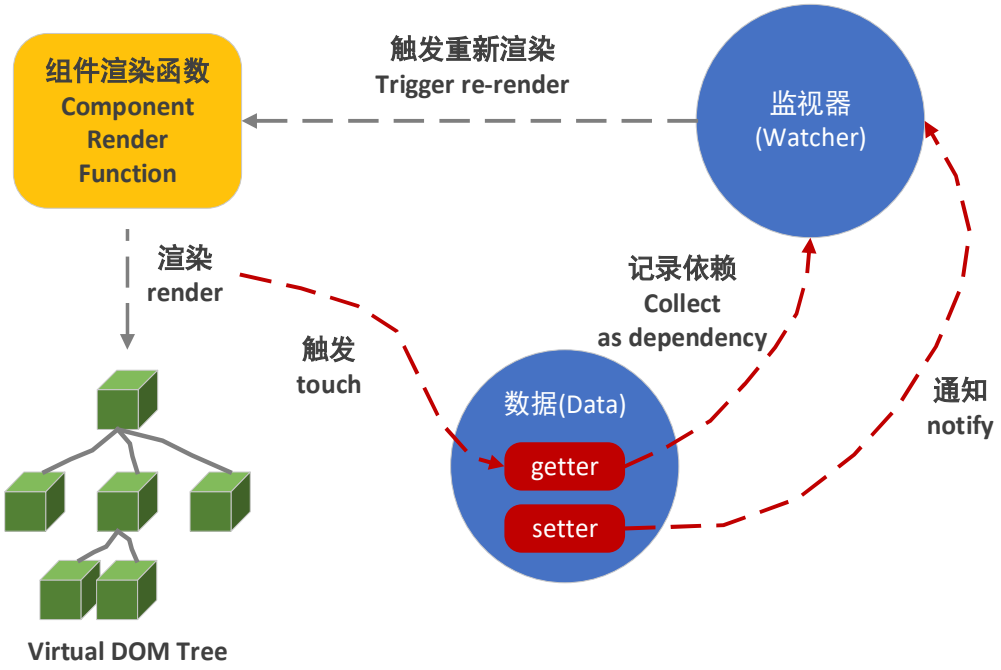

图 2-4 Vue 响应式原理[9]

当在 Vue 实例中的 data 选项中传入普通的 JavaScript 对象时，Vue 会在这个实例对应的 Watcher 实例中记录此数据项的依赖。当数据项改变时，会调用该数据项的 setter 方法，而该方法会通知 Watcher，进而触发重新渲染。

组件化是指开发者可以编写可复用的组件，在需要使用的地方直接引入，而不是通过复制粘贴这种低效的方式。每个组件需要定义唯一的组件名，组件文件以.vue 结尾，文件中同时包含 HTML、JavaScript 以及 CSS 文本。其中 HTML 文本用于组织页面结构，JavaScript 文本提供动态交互效果，CSS 文本丰富页面样式[10][11][12]。之所以通过一个文件而不是多个文件定义一个 Vue 组件，是因而这样可以减少开发者在各文件之间切换的时间，并且可以降低组件的维护成本。

2.5 本章小结

本章对论文涉及的哈希算法、文本相似度计算方案、神经网络模型以及相关开发框架和工具进行了介绍。其中就 Scrapy 的架构及 Vue 的两大特性进行了详细说明，并探究了 Vue 响应式特性的实现原理。

第三章 设计与实现

3.1 方法概览

API 相关概念解释提取系统核心功能的实现分为数据准备、词频统计以及数据提纯这三个步骤，图 3-1 展示了该系统整体的方法流程。

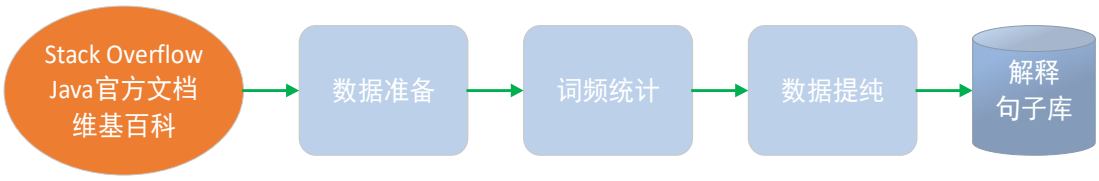

图 3-1 方法流程

上述步骤的具体实现分别对应于数据准备模块、词频统计模块以及分类器模块，接下来介绍这三个模块的设计与实现细节。

3.2 数据准备模块

数据准备模块负责 Java 官方文档、Stack Overflow 以及维基百科这些平台上提取数据，该模块的主要开发工作为设计并实现网络爬虫以及 Scrapy 管道。图 3-2 展示了数据准备模块与数据库的交互。

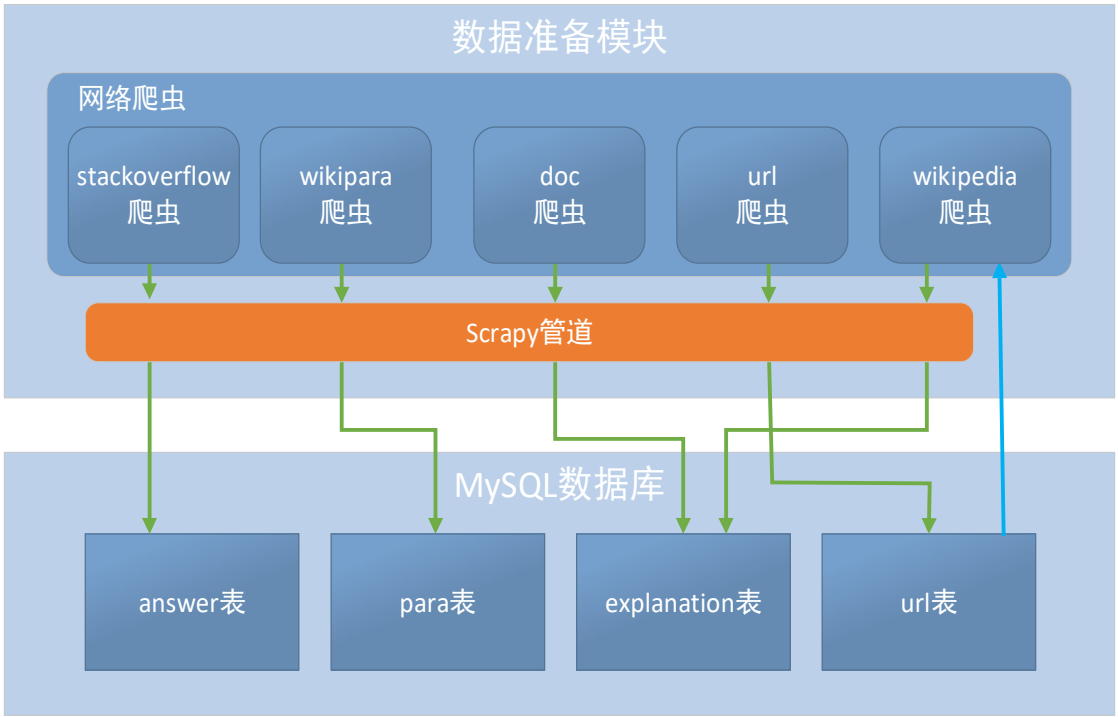

图 3-2 数据准备模块与数据库的交互

如图 3-2 所示，数据准备模块会将提取到的概念解释通过 Scrapy 管道存储到 MySQL 数据库相应的表中。接下来详细介绍支持数据准备模块数据库表的设计、URL 去重方案、网络爬虫以及 Scrapy 管道的设计与实现。

3.2.1 数据库表设计

在数据准备模块中，各个爬虫之间主要通过共享的数据进行协作。并且由于此模块需要处理的任务较多，所以需要设计多个数据库表以支持不同的需求。接下来详细介绍各个数据库表的设计，并说明这些表中各个字段的作用以及在使用这些表时的注意事项。

1）回答表

该表是用来存储从 Stack Overflow 上提取的问题回答的表，其具体说明如表 3-1 所示。

表 3-1 Stack Overflow 上问题回答 answer 表结构

字段	字段类型	字段描述
id	INT	主键，唯一标识
is_explanation	TINYINT	标识该回答是否为解释性的文本
answer	VARCHAR	回答的文本内容

表中 is_explanation 字段用于标识该回答是否为概念解释，由于该字段只存在两个可能的取值，所以将其类型设计为 TINYINT。回答最开始被存储到 MySQL 数据库中时，is_explanation 字段被设置为 0。之后再用分类器将这些回答从数据库中读出并进行预测，将那些被判定为解释性文本的回答的 is_explanation 字段更新为 1。

2）段落表

段落表用于存储从维基百科上提取的解释性文本和非解释性文本，这些解释性文本和非解释性文本构成一个数据集，为分类器的实现提供数据支持。表 3-2 为段落表的表结构。

表 3-2 段落 para 表结构

字段	字段类型	字段描述
id	INT	主键，唯一标识
is_explanation	TINYINT	标识是否为解释性文本
text	TEXT	文本的内容

3）解释表

该表存储那些经过一系列处理后最终被判定为 API 相关概念解释的文本，并为后端提供数据支持。表 3-3 为解释表的表结构说明。

表 3-3 解释 explanation 表结构

字段	字段类型	字段描述
id	INT	主键，唯一标识
keyword	VARCHAR	概括解释内容的关键字
description	TEXT	解释的文本内容

表中 keyword 字段用于概括解释内容，该字段主要用于加速后端对用户输入的模糊查询过程。

4）URL 表

该表用于存储爬虫提取的维基百科页面的 URL。这些 URL 在后续作为爬虫的起始地址，用于提取维基百科上 API 相关概念的解释。表 3-4 为 URL 表的表结构。

表 3-4 URL 表结构

字段	字段类型	字段描述
id	INT	主键，唯一标识
url	VARCHAR	URL 的值
url_crc	INT	哈希值，通过 CRC32 函数计算得来

URL 的表结构除了基本的 id 字段以及 url 字段之外，还有一个字段 url_crc，该字段是 url 通过 CRC（Cyclic Redundancy Check）函数计算得来，可以加速查询 URL 的过程。具体加速查询的方法为：先基于 url_crc 字段创建 BTREE 索引，并在查询时添加上对于该字段的相等判断，这样 MySQL 便能够利用 B 树对查询进行加速。

3.2.2 URL 去重

数据准备模块需要设计并编写爬虫对各个平台上的资源进行提取，因此必定会涉及网站的网址（即 URL）。由于需要提取的数据量达到十万数量级，不可能在短时间内完成数据提取工作，往往需要在夜间保持爬虫的运行。在爬虫运行期间，会出现种种状况，比如网络中断、连接超时、计算机关闭，此时爬虫就会停止运行。当爬虫重新启动之后，要么从上次的断点恢复，要么从起始网站重新开始。这些情况都会导致 URL 重复，此时就需要对 URL 进行去重。对 URL 进行

去重主要有两点好处：第一点是去除重复数据；第二点是提高爬虫性能。

至于如何对 URL 进行去重，此论文探讨的解决方案主要有以下两类。第一类为无重复方案，第二类为基于 Hash 的解决方案。接下来详细介绍这两类解决方案的使用场景以及具体的实施步骤。

1）无重复方案

在多线程编程中，常使用不可变的数据或者对象、互斥机制、非阻塞机制等进行同步。除了这几种解决方案之外，还有一种常常被开发者忽略的方案，那便是无同步方案。无同步方案指如果代码本身不涉及共享数据，那么自然无须任何同步措施。栈封闭、线程本地存储（Thread Local）、可重入代码等就是无同步方案的具体实例。类比多线程同步中的无同步方案，如果爬虫在重启之后无需对 URL 进行去重就可以直接执行爬取页面的任务，本文便将其称为无重复方案。无重复方案并不意味着不需要额外的操作，该方案一般也需要额外的操作来保证爬虫能够从上次的断点恢复，否则爬虫重启之后还是从起始网址重新开始的。而且无重复方案只有在特定场景下才能应用，本文主要涉及三种场景，分别为页面较少、 URL 已知、URL 符合某一规则。

由于 Java 官方文档中类的总数在一万以内，这属于页面较少的场景。并且由于官方文档首页包含所有 Java 类对应页面的 URL，所以可以直接编写相应的解析函数将这些 URL 提取出来，然后继续请求这些新的页面并提取新页面中 Java 相关概念的解释。在 Srcapy 配置的并发请求数为 16，下载延迟为 0，网络状况良好时，爬虫能在 30 分钟内将 Java 官方文档上的所有页面爬取完。这种情况下，最简单的解决方案就是不提供任何恢复措施，爬虫每次都从起始网站重新开始。

如果页面的 URL 能够以文本文件提供或者从数据库中读取到，那么便是 URL 已知的场景。以 URL 由文本文件提供为例，爬虫在执行时，每次只需要从文本文件中读取一个 URL 并请求相应的页面即可。这种情况下，让爬虫从上一次的断点恢复是一项十分简单的任务，只需要在爬虫每处理完一个页面后便在某个特定的文件中写入当前的标号，这样当爬虫重启时便可以从该文件中读取标号，由此得知该从哪个位置恢复。上文提到的页面较少场景下的解决方案没有恢复措施，可以结合 URL 已知场景下的解决方案进行改进。首先将 Java 官方文档首页中所有 Java 类对应页面的 URL 提取出来，以每个 URL 占据一行的方式存储到文本文件中，这样便转换成了 URL 已知的场景，也就能够添加恢复措施。

在 Stack Overflow 平台上，与 Java 相关的问题按页显示，每一页显示 50 个问题，并且包含每个问题详细页面的 URL。当提取完一页的数据时，可以请求下一页数据。这些 URL 除了编号不同以外，其余都完全一致，这属于 URL 符合某

一规则的场景。这样，爬虫就能轻松地构建 URL，只需要在整个执行过程中记录最新的页面编号即可。重启时，通过读取文本文件中存储的编号即可以从上次的断点恢复。

2）基于 Hash 的解决方案

本文主要探讨了哈希表、布隆过滤器、MySQL 存储这三种基于 Hash 的 URL 去重方案，下面详细讨论这几种方案。

本文涉及的数据量在十万数量级，也就是说使用的哈希表仅占 5.4MB 内存，因而哈希表这一方案能够满足本文的需求。但是如果 URL 的数量达到了亿级，此时哈希表所占的内存将达到 5.27GB 甚至数十 GB，这种情况下，哈希表就不是一种高效的解决方案。

由于布隆过滤器并不能判断一个输入存在，因此可能丢弃一些尚未处理的 URL。但这并不是说明布隆过滤器没有用处，如果并不要求对所有的页面都进行爬取，那么一个设计良好（哈希冲突发生概率很低）的布隆过滤器可以快速地将那些重复的 URL 去掉，并且占用的内存空间相较于哈希表而言极少，这显然是一个高效的解决方案。

上文详细介绍了基于哈希表以及基于布隆过滤器的 URL 去重方案，但是这两种方案均存在一定的缺陷。基于哈希表的去重方案会消耗大量内存空间，而基于布隆过滤器的去重方案虽然消耗的内存极少，但是存在误判的情况。现在探讨是否存在既不会误判，消耗内存又少的解决方案。如果不想出现误判，那么必须将处理过的 URL 都存储下来，而且想要消耗的内存少，那么需要将这些 URL 存储到数据库中。

3.2.3 网络爬虫

在确定了不同场景下 URL 去重方案后，数据准备模块针对各种需求设计了五个网络爬虫（docs、stackoverflow、url、wikipedia、wikipara）。分别用于提取 Java 官方文档中各个 Java 类的解释、Stack Overflow 上的问题回答、维基百科页面中包含的 URL、维基百科上 API 相关概念的解释以及维基百科上解释性文本和非解释性文本。

1）docs

docs 爬虫用于提取 Java 官方文档中各个 Java 类的解释，执行逻辑主要有三个步骤。第一步为请求设定的起始网页；第二步为从起始网页中提取出所有 Java 类的详细介绍页面的 URL；第三步为通过提取到的 URL，分别请求这些 Java 类的详细介绍页面并提取出概念的解释，最后将这些解释交给 Scrapy 管道处理。

2）stackoverflow

stackoverflow 爬虫用于提取 Stack Overflow 上问题的回答，执行逻辑主要有两个步骤。第一步读取配置文件中的索引并根据读取到的索引构建起始网页的 URL；第二步为请求起始网页，提取问题的回答并交由 Scrapy 管道处理，最后构建下一页的 URL，将此 URL 存储到配置文件后再对其进行请求。

3）url

url 爬虫采用深度优先搜索策略，用于提取维基百科页面中包含的 URL，执行逻辑主要有两个步骤。第一步为请求起始网页，并将深度设置为 1；第二步为提取网页中所有的 URL，并将那些不合规范的 URL 过滤掉，然后将提取到的 URL 交由 Scrapy 管道处理，最后请求这些新的 URL 并将深度加 1。为了避免 url 爬虫陷入死循环，在第二步中添加了对深度的判断逻辑，当深度大于设定的最大深度时，从当前函数返回。

4）wikipedia

wikipedia 爬虫用于提取维基百科上 API 相关概念的解释，执行逻辑主要有两个步骤。第一步为从 MySQL 中每次读取一定数量的 url，并分别进行请求；第二步为从请求到的网页中提取概念的解释，并交由 Scrapy 管道处理。需要注意的是，在运行 wikipedia 爬虫时，应在 Scrapy 的配置文件中开启相似度管道。因为从维基百科上提取的概念的解释并不一定是 Java 领域的，需要对其进行相似度计算。当计算得来的相似度大于设定的阈值时，才能将其保留，否则应当将其丢弃。

5）wikipara

wikipara 爬虫采用深度优先搜索策略，用于提取维基百科上解释性文本和非解释性文本。wikipara 爬虫的执行逻辑与 url 爬虫的执行逻辑基本相同，不同之处在于 wikipara 爬虫并不是将页面中所有段落都提取出来，而是提取页面的第一段作为解释性文本，提取其他任意一段作为非解释性文本。

3.2.4 Scrapy 管道

Scrapy 管道用于处理网络爬虫提取的以字典形式组织的数据，可以对字典的属性执行访问、修改等一系列操作。当一条数据被判定为不符合某项规则时，Scrapy 管道可以丢弃这条数据。本文设计并实现的 Scrapy 管道可以分为三大类，第一类为数据修改管道，第二类为数据判定管道，第三类为数据存储管道。整个 Scrapy 管道对于数据的处理流程如图 3-3 所示。

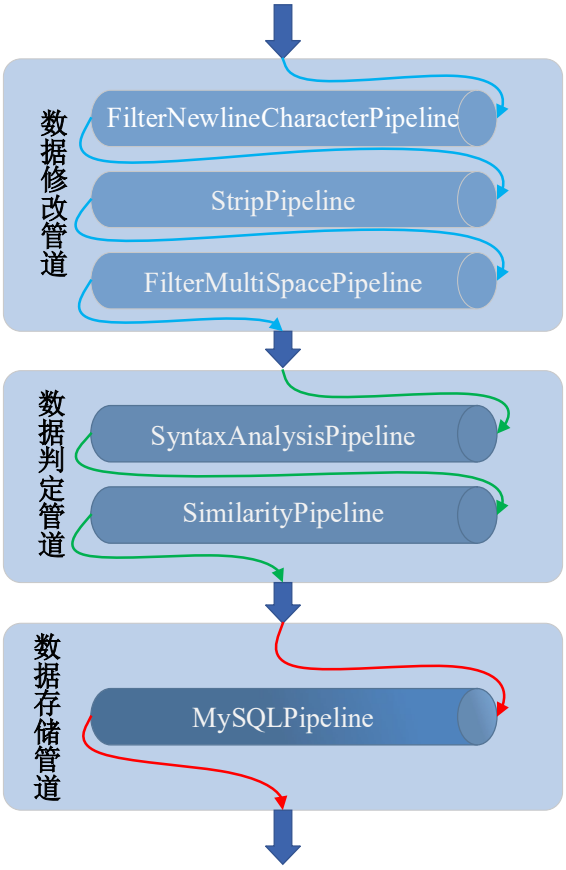

图 3-3 Scrapy 管道的处理流程

1）数据修改管道

数据修改管道包括 FilterNewlineCharacterPipeline、StripPipeline、FilterMultiSpacePipeline 这三个管道，分别用于过滤数据中的换行符、去掉数据头部及尾部的空白符、将数据中的连续空格替换成一个空格。前两个管道基于 Python 字符串的 replace 及 strip 便可以实现，最后一个管道基于 re 模块提供的正则表达式功能实现。由于这三个管道实现均较为简单，在此就不进行详细描述。

2）数据判定管道

这类管道用于判定输入的文本数据是否满足定义的规则，如果不满足，则丢弃。数据判定管道包括 SyntaxAnalysisPipeline 和 SimilarityPipeline 这两个管道，分别用于判定文本数据是否满足主谓宾结构、文本数据与 Java 领域概念的解释中高频词的相似度是否大于阈值。

语法分析管道在实现时使用到了 spaCy，其实现主要分为三个步骤：第一步为使用 spaCy 将文本数据分成一个个句子；第二步为使用 spaCy 将第一步得到的句子进一步分割为一个个单词；第三步为使用 spaCy 对这些单词进行依存分析，并丢弃那些不满足主谓宾结构的文本数据。

本文采用余弦相似度方案实现相似度管道，主要步骤有分词、统计词频、构建词向量以及计算相似度。由于本文提取的都是英文资源，所以分词这项任务十

分简单。可以自定义分词器，将输入的英文文本按照空格分隔的方式将各个单词分开，并且去除掉单词首部和尾部的标点符号。也可以使用 NLP（Natural Language Processing）工具 spaCy 进行分词。分词时，除了将各单词分开，还需要去掉停用词。停用词一般是功能性的词，比如限定词、介词等，这些词与其他词相比提供的信息很少，所以在处理文本时去掉。可以基于已有的停用词表来去掉停用词并在任务执行过程中动态添加或者删除停用词表中的词，也可以基于 spaCy 中 token 类的 is_stop 属性来去掉停用词。

在统计词频时，首先需要将分词得来的单词转化为小写形式，然后查询数据库的词频表中是否存在该单词。如果词频表中不存在该单词，那么就插入该单词并将词频设置为 1，否则更新该单词的词频为原来的词频加 1。统计词频可以整合到数据准备模块中，也就是每当提取到一个新的概念解释时就执行一次上述插入或者更新词频操作。由于统计词频与爬虫的联系并不紧密，并且为了提高爬虫的响应度和稳定性，本文决定将其作为一个单独的模块来实现。

数据库的词频表中存储的词至少有上万个，难以将这些词都读取出来以构建词向量。一个原因是这些词中，绝大部分词的词频低于 10，不具有代表性。另一个原因是构建一个高维度的向量会增加计算时间，降低系统的运行速度。所以合理的解决方案是，从数据库的词频表中选择词频前 K 的词构建词向量 V_1，再根据提取的概念解释构建相应的词向量 V_2。此时还不能基于 V_1 和 V_2 计算相似度，因为它们各个分量的含义不一致。需要对 V_1 和 V_2 进行修改，使得它们的各个分量代表的词完全一致时，才能用于计算相似度。如果 V_1 和 V_2 的相似度大于阈值，那么说明当前概念解释属于 Java 领域，应将该解释保存到数据库中，否则将该解释丢弃。

3）数据存储管道

经过修改且最终判定合法的数据将由数据存储管道存储到 MySQL 中。该管道在爬虫启动时建立与 MySQL 数据库的连接，每当有数据输入时便通过 DB-API（Database-Application Programming Interface）执行相应的 SQL（Structured Query Language）语句以存储数据。

3.3 词频统计模块

词频统计模块统计从 Java 官方文档提取的概念解释以及从 Stack Overflow 上提取的问题回答中个英文单词的词频，用于支持数据准备模块中相似度管道的实现。该模块在与数据库建立连接后，每次从数据库中读取出 100 条概念解释或者问题回答，并使用 spaCy 对这些数据进行分词并统计词频。分词时，基于

token.is_stop 去掉停用词。统计词频时，第一次出现的新词被插入到数据库中并且词频设置为 1，已出现过词的词频更新为原词频加 1。词频存储在数据库的词频表中，表 3-5 为词频表的表结构。

表 3-5 词频 frequency 表结构

字段	字段类型	字段描述
word	VARCHAR	主键，英文单词
frequency	INT	该词在 answer 表及 explanation 表中出现的次数

3.4 分类器模块

分类器模块能够识别解释性文本和非解释性文本，用于对数据进行提纯。该模块使用 Keras 框架搭建深度学习模型，基于爬虫收集的数据集进行训练。当训练完成时，该模块保存模型并通过文件名标识该模型的主要指标。在进行分类时，该模块加载保存的模型，读取数据库中的概念解释，并将那些预测为非解释性文本的记录的 is_explanation 字段设置为 0。

3.4.1 数据集构建

爬虫从 Java 官方文档、维基百科、Stack Overflow 这些平台上提取的文本并不一定是解释性的文本。其中 Java 官方文档中的文本以及维基百科页面的第一段都是组织良好的解释性文本，但是 Stack Overflow 上的文本质量较差，很多都不是解释性的文本。为了使得最后提取的数据都是解释性的文本，必须实现一个文本分类器对数据进行提纯。

在传统的机器学习领域，文本分类一般采用基于 TF-IDF（Term Frequency-Inverse Document Frequency）的特征提取方法，后来便开始使用各种分类器，如朴素贝叶斯、SVM（Support Vector Machine）等[13][14]。传统的机器学习方法对文本特征的表达能力很弱，随着深度学习的兴起，CNN 等深度学习技术也迅速被应用于文本分类。Yoon Kim 于 2014 年发表了基于 CNN 的文本分类模型 TextCNN，该模型在文本分类任务中取得了不错的效果[15][15]。有鉴于此，本文设计了一个基于 TextCNN 的文本分类器，用于识别解释性文本和非解释文本。因此，本文需要构建一个包含大量解释性文本以及非解释性文本的数据集，用于训练模型以及评估模型的性能。构建数据集包括数据收集和数据标注这两个步骤。

数据收集是通过编写爬虫，提取、处理并存储维基百科上页面的文本来实现

的。爬虫从一个起始页面开始爬取，处理该页面上的数据并提取链接，最后从提取出来的链接中选择一个并向第三方平台的服务器发出获取新页面的请求。上述逻辑比较清晰明了，关键点在于如何从众多链接中选择出一个作为新的页面，主要有以下两种策略：

1）广度优先搜索

在爬虫启动时，将起始页面的 URL 入队。之后每次从队列头部取出一个 URL，提取对应页面中所有 URL 并依次入队，如此循环直到队列为空。该策略有许多优点，比如相关度较大或者说重要性更高的最先爬取；多爬虫进行协作时，可以套用生产者-消费者模式，有利于爬虫之间的合作。但是该策略也存在着占用内存较大的缺点。假设一个页面平均包含 200 个 URL，URL 的平均长度为 50，第一层只有起始页面这一个页面节点。那么第四层时队列中将有近 800 万个 URL，约占用 381.5 MB 内存。当爬虫爬取到第五层时，队列中最多可能存储 16 亿个 URL，约占用 74.5 GB 内存。就本文而言，页面数量在十万量级，所以使用广度优先搜索策略并不会导致内存溢出，因此该策略具备可行性。

2）深度优先搜索

深度优先搜索在爬虫选择了一个指向其他页面的链接之后，便会请求这个新的页面，循环执行上述操作直到搜索结构的叶子节点。该策略实现简单并且占用内存极少，在本文中具备可行性。但该策略存在着一定的缺陷，当出现页面循环引用的情况时，使用深度优先搜索策略可能会使爬虫陷入死循环。可以结合 URL 去重机制，或者设置最大深度来修复此缺陷。

数据标注可以在数据收集完成之后进行，也可以整合到数据收集这个步骤中，具体取决于选用哪种数据标注方法。本文探讨的数据标注方法有人工标注和自动标注。

1）人工标注

在条件允许的情况下，人工标注是准确度最高的标注方法，而且在图像识别、语音识别等领域只能使用人工标注。但是人工标注费用极高，假设数据集含有 10 万条文本数据，标注每条数据需要花费 1 分钟，那么标注完这个数据集大约将花费 1666 小时，约 69 天。这远远超出了本文的承受范围，因此不具备可行性。

2）自动标注

自动标注一般是针对于那些组织良好，并且带有分类标签的数据。这样在提取这些数据时，就能根据这些标签将数据标注好。本文需要的数据是解释性的文本及非解释性的文本，不幸的是没有平台提供这些数据。在观察了维基百科上的页面内容之后，本文发现这些页面的第一段文本都是解释性的文本，而其他段落一般都是非解释性的文本。基于此发现，本文设计了相应的数据集自动标注方案，

即将维基百科页面上提取的第一段文本标注为解释性的文本，任选另一段文本并标注为非解释性的文本。

3.4.2 数据集介绍

本文准备的数据集有三个，分别是 para_small、para_mid 以及 para_large。这些数据集的详细信息如表 3-6 所示。

表 3-6 数据集详细信息

编号	名称	数据量	大小
1	para_small	1 万	4.1 MB
2	para_mid	10 万	41.8 MB
3	para_large	50 万	209 MB

数据集中的样本需要经过一定的处理后，才能作为模型的输入。首先，样本中包含 id，这是训练模型时不需要用到的数据，所以需要去掉 id。其次，由于训练集与测试集由同一个数据集提供，所以事先需要将数据集划分成训练集和测试集。

3.4.3 模型搭建

本文参考 Yoon Kim 于 2014 年提出的 TextCNN，并结合预训练的 Glove 词向量，搭建了用于识别解释性文本及非解释性文本的分类器模型[16][17]。该模型的具体结构如图 3-4 所示。

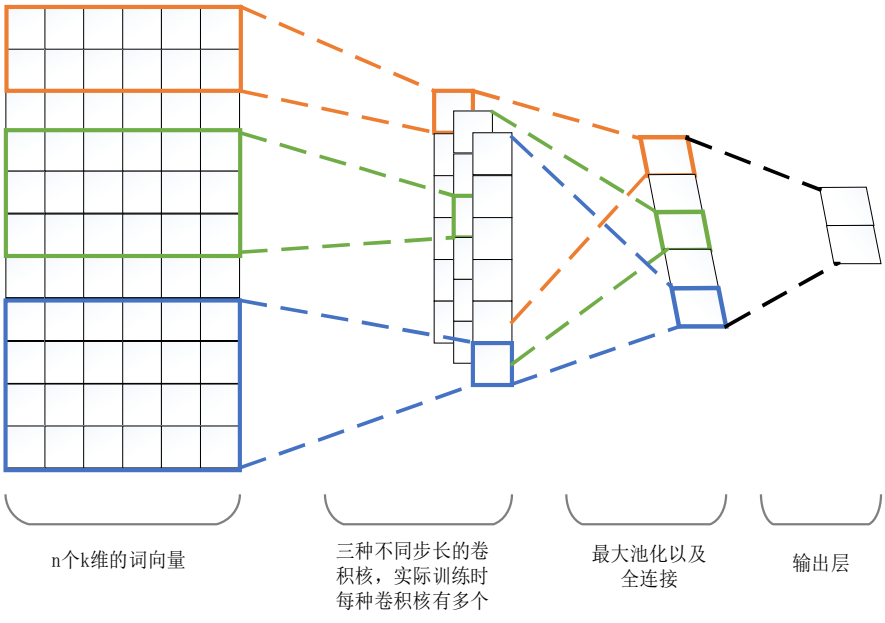

图 3-4 分类器模型

模型的输入为数据集中样本对应的词向量，输入经过三种不同步长的卷积核后进入全连接层，最后抵达输出层。该模型的输出值只有两个，其中 0 代表非解释性文本，1 代表解释性文本。

3.5 工具实现

数据准备模块、词频统计模块及分类器模块将第三方平台上的概念解释提取并存储，但开发者无法获取这些概念解释。为了解决该问题，在工具实现部分，本文以 Web 应用的形式提供了一个解释查询工具。

3.5.1 系统架构

API 相关概念解释提取系统由表示层、服务层以及存储层组成，图 3-5 展示了该系统的架构。

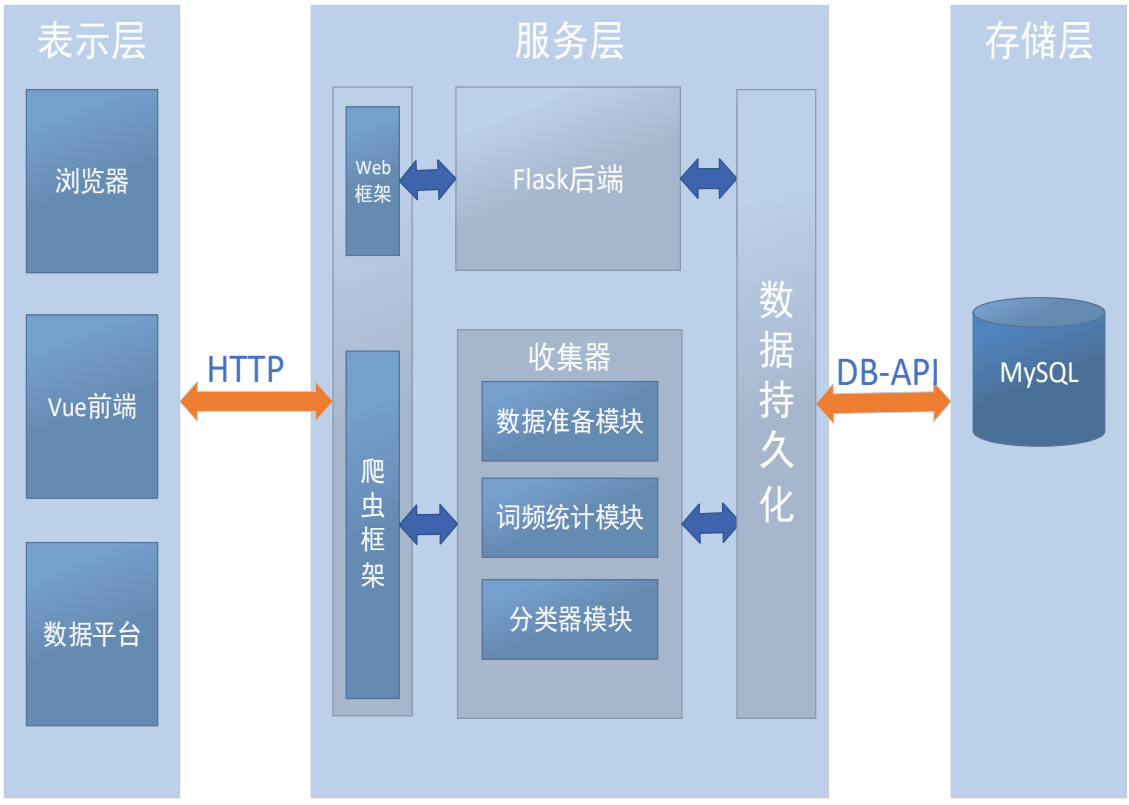

图 3-5 API 相关概念解释提取系统架构

如图 3-5 所示，表示层包括用于访问系统的浏览器，展示概念解释的 Vue 前端以及数据平台。服务层包括 Flask 后端以及收集器，前者用于向用户提供概念解释的查询服务，后者用于从数据平台上提取及收集概念解释。存储层使用的是高效稳定的免费数据库 MySQL。表示层与服务层之间通过 HTTP（Hyper Text

Transfer Protocol）协议[18]进行通信，并且服务层中使用了 Web 框架以及爬虫框架来处理或者发送相应的 HTTP 请求。由于服务层是使用 Python 开发的，因此服务层与存储层之间自然是通过 DB-API 进行通信。收集器中三个模块的设计与实现在前几节已经给出，不再赘述，本节就前端与后端的设计与实现进行说明。

3.5.2 前端

前端使用 Vue.js 框架开发，主要包括三个页面，主页面、搜索页面以及描述页面。主页面由网站 Logo 和搜索框组成，网站 Logo 并不是使用绘图软件制作的图片，而是通过 p 标签并设置相应的样式实现。图 3-6 为主页面。

图 3-6 主页面

搜索页面由网站 Logo、顶部搜索框、展示概念解释的分页组件以及热词部分组成，分页组件基于高质量的 UI（User Interface）组件库 View UI[19]实现，热词部分则通过无序列表实现。当由主页面跳转到搜索页面时，前端会向后端请求搜索结果以及热度前五的热词。请求结果以 JSON（JavaScript Object Notation）形式返回，前端解析完 JSON 数据后使用 Vue.js 框架的 v-for 语法将数组渲染成一个列表。图 3-7 为搜索页面。

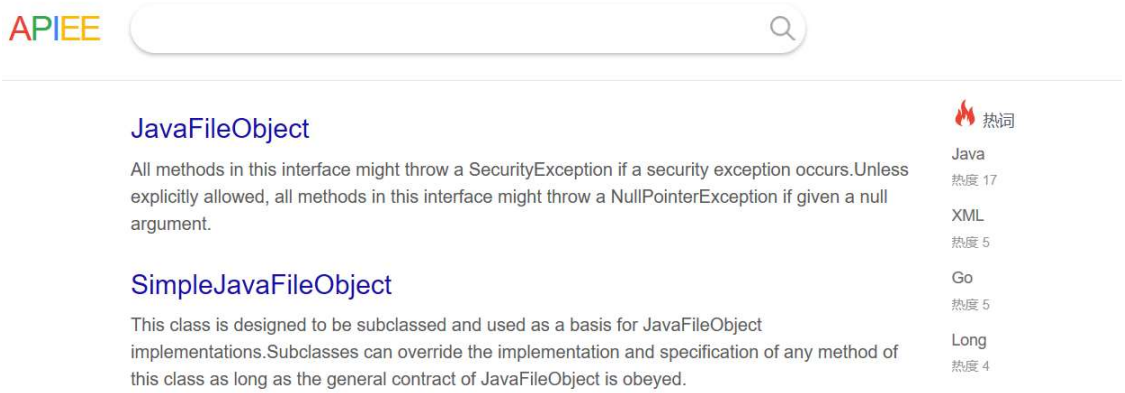

图 3-7 搜索页面

描述页面由网站 Logo 以及显示概念解释的卡片组成，其中卡片基于 View UI 的卡片组件实现。当由搜索页面跳转到描述页面时，搜索页面将概念解释传入路由参数中，描述页面在初始化时从路由参数中将概念解释取出并进行声明式渲染。图 3-8 为描述页面。

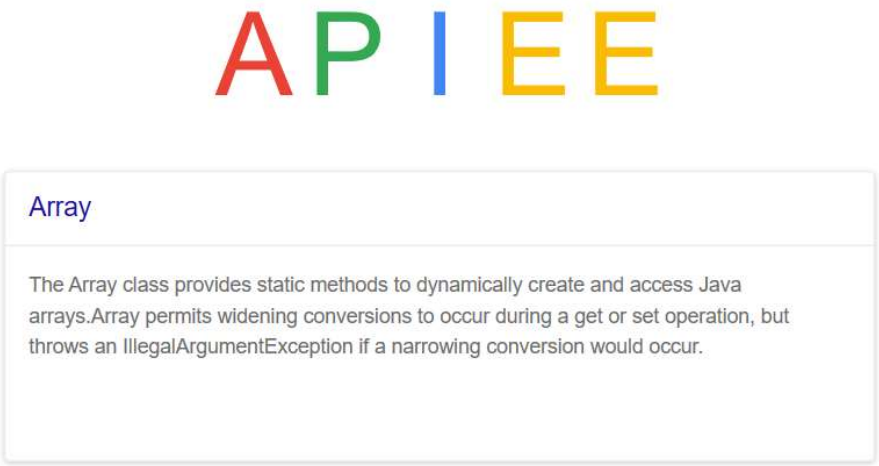

图 3-8 描述页面

3.5.3 后端

后端提供了三个供前端调用的接口，分别用于返回用户搜索结果、热词以及增加词的热度。这些接口都是通过 DB-API 连接数据库后，执行相应的 SQL，最后向前端返回查询结果实现的。其中热词基于热词表实现，该表存储用户搜索词以及相应的热度，表 3-7 为热词表的表结构

表 3-7 热词 hotword 表结构

字段	字段类型	字段描述
id	INT	主键，唯一标识
words	VARCHAR	用户搜索词
degree	INT	搜索词的热度

表中 degree 字段表示搜索词的热度，当该搜索词再一次被搜索时，degree 字段加 1。后端在查询热词时，按照热度降序排列，并选择热度前 K 高的热词及对应的热度。

3.6 本章小结

本章就论文中各个部分的设计与实现细节进行了详细的介绍。本章最开始呈

现了本文的方法概览，让读者能够对系统的实现过程有一个整体的了解。这之后，本章着重介绍了数据准备模块以及分类器模块。由于前端与后端实现较为容易，所以本章在最后对其进行了简单介绍。除此之外，本章就论文中遇到的一些关键问题进行了详细的描述和探究，就各个问题可能存在的解决方案进行了比较，并分析方案的可行性以确定最终方案。这些关键问题包括 URL 去重、文本相似度计算以及分类器所需数据集的构建。其中 URL 去重主要有两类方案，第一类方案是无重复方案，第二类方案是哈希算法。无重复方案适合于 URL 较少或者满足某一规则的场景，应用范围较窄，哈希算法则是更为通用的解决方案。本文选择解释性强且实现简单的余弦相似度方案计算文本相似度，该方案主要包括分词、统计词频、构建词向量以及相似度计算这四个步骤。构建数据集的关键在于数据收集以及数据标注。本文比较了数据收集中常用的两种策略，广度优先搜索和深度优先搜索，并分析了这两种策略各自的优缺点以及应用场景。而数据标注方面，本文采用的是自动标注，因为人工标注成本超出了承受范围。

第四章 实验评估

4.1 分类器评估

本文使用上述 para_small、para_mid 以及 para_large 这三个数据集训练搭建好的模型，结果如表 4-1 所示。

表 4-1 分类器实验结果

数据集	数据量	epochs	训练集正确率	测试集正确率
para_small	5 万	10	99.99%	83.50%
		20	99.92%	80.95%
		40	99.61%	80.16%
para_mid	10 万	10	96.41%	82.34%
		20	97.64%	82.17%
		40	98.51%	81.81%
para_large	50 万	10	90.97%	84.34%
		20	93.82%	84.50%
		40	95.70%	84.78%

在训练完模型之后，对其进行了测试。这里给出四个对 Java 官方文档上数据的分类结果及四个对 Stack Overflow 上数据的分类结果。图 4-1 是 Java 官方文档上的 Java 类解释，图 4-2 是 Stack Overflow 上问题的回答。

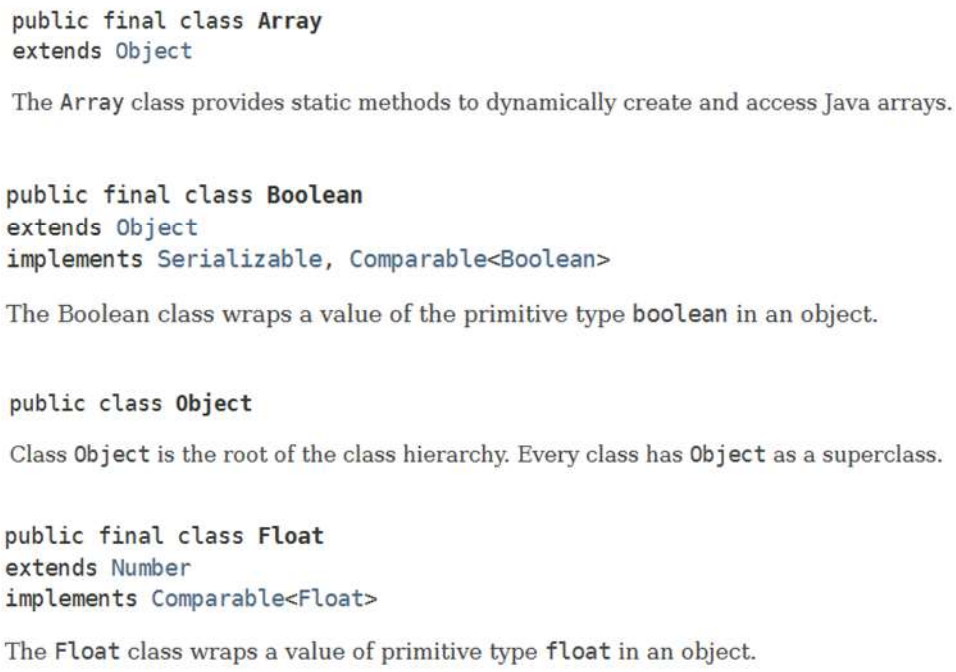

图 4-1 Java 官方文档上的 Java 类解释

模型对这四个 Java 类解释的预测结果都是非解释性文本，这与期望的结果有较大的出入。

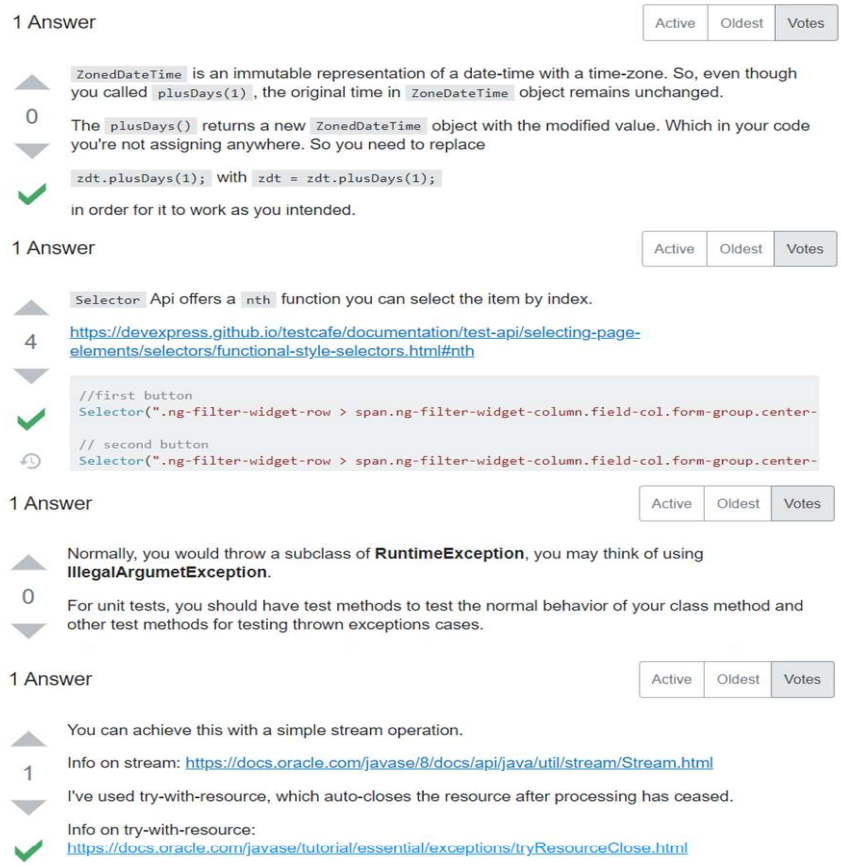

图 4-2 Stack Overflow 上问题的回答

模型将前两个问题的回答预测为解释性的文本，将后两个问题的回答预测为非解释性的文本，与预期的结果相符。

由实验结果可知，使用 para_small 数据集和 para_mid 数据集时，存在过拟合的情况，并且在测试集上的正确率均较低。使用 para_large 数据集时，无明显的过拟合现象，在测试集上的正确率也较高。因此最终选择 para_large 数据集，设置 40 个 epochs 对模型进行训练。

模型对 Java 官方文档上 Java 类解释的预测效果不理想，这是因为整个数据集是基于维基百科上面的词条构建的，对 Java 类解释不具备分类能力。不过 Java 类解释本身就是解释性文本，并不需要使用分类器进一步提纯，同理从维基百科上提取的解释也不需要进一步提纯。模型对 Stack Overflow 上问题回答的预测效果尚可，能够将大部分的非解释性文本及解释性文本预测正确，因而可以应用。

在本文的实现中，仍然存在许多可以改进的地方。例如可以扩大数据集的构建范围，不仅仅局限于维基百科上的数据以增强模型的分类能力；可以对模型进行扩展，变成一个多分类器而不仅仅是一个二分类器，这样可以对提取的数据进行更细化的处理，提高分类器模块整体的分类效果。

4.2 系统评估

本文首先选取了 20 个与 Java 相关度较大的概念以及 20 个与 Java 相关度较小的概念对 API 相关概念解释提取系统的效果进行测试，结果如图 4-3 所示。

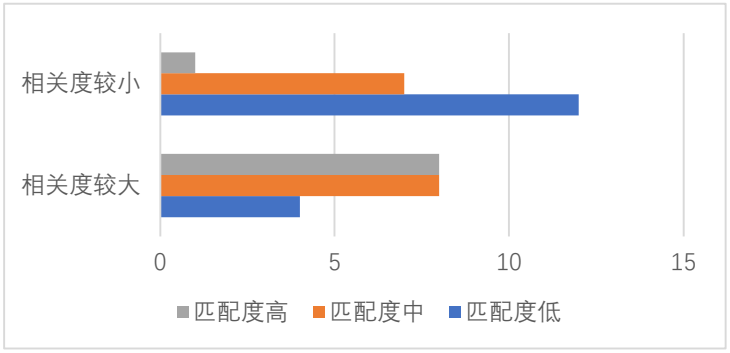

图 4-3 系统测试结果

由图 4-3 可知，对于与 Java 相关度较小的概念而言，概念解释中匹配度高的不超过 10%；对于与 Java 相关度较大的概念而言，概念解释中匹配度高的达到了 40%。可知此系统的应用场景为搜索 Java 领域内的概念解释，为了进一步说明系统的应用场景，本文给出四个具体的搜索示例。图 4-4 为 ArrayList 及 Stack 的搜索结果，图 4-5 为 CNN 及 Input 的搜索结果。

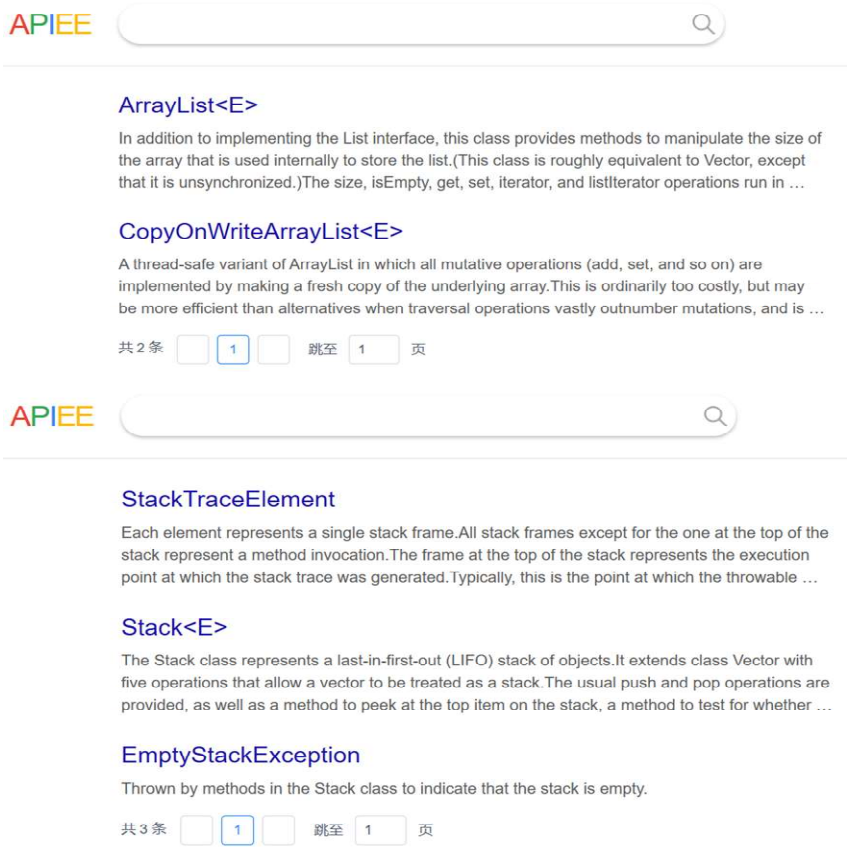

图 4-4 ArrayList 及 Stack 搜索结果

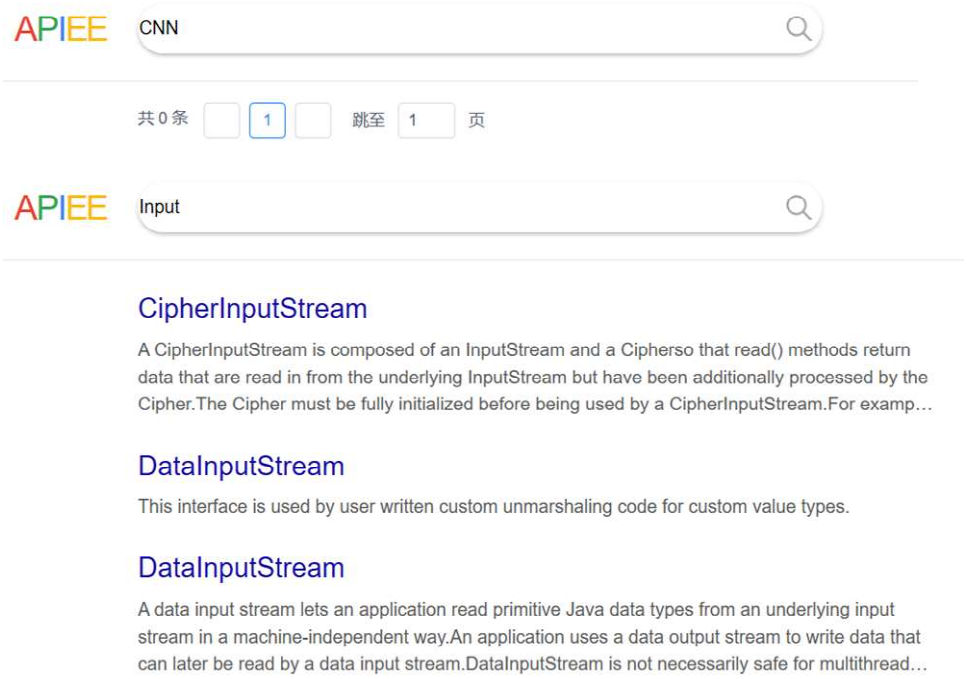

图 4-5 CNN 及 Input 搜索结果

对于 ArrayList 和 Stack 这两个与 Java 相关度较大的概念而言，此系统能够给出匹配度高的概念解释。对于 Input 这个与 Java 相关度较低的概念而言，此系统给出了匹配度低的概念解释。而对于 CNN 这个与 Java 相关度很低的概念而言，由于系统没有收集相应的数据，所以没有给出概念解释。

4.3 本章小结

本章就分类器的效果以及整个系统的效果进行了评估，分析了评估结果产生的原因，并说明了分类器及整个系统的应用场景。其中分类器适合于对 Stack Overflow 上的数据进行提纯，不适于对 Java 官方文档和维基百科上的数据进行提纯。整个 API 相关概念解释提取系统适合于查询 Java 领域内的概念解释，不适于查询其他领域内的概念解释。

第五章 总结与展望

5.1 本文总结

本文旨在从各个平台上提取 Java 领域内 API 相关概念的解释，对提取到的概念解释进行处理与提纯，并为开发者提供一个 API 相关概念的解释查询工具。

为了从各个平台上提取数据，本文设计并实现了收集器，编写了用于提取不同平台数据的爬虫。其中，docs 爬虫用于提取 Java 官方文档中各个 Java 类的解释，stackoverflow 爬虫用于提取 Stack Overflow 上的问题回答，url 爬虫用于提取维基百科页面中包含的 URL ，wikipedia 爬虫用于提取维基百科上 API 相关概念的解释，wikipara 用于提取维基百科上解释性文本和非解释性文本。

为了对概念解释进行处理与提纯，本文设计并实现了 Scrapy 管道以及分类器。其中 Scrapy 管道能够对数据进行修改、丢弃以及存储。分类器则用于对数据进行提纯，那些被分类器判定为非解释性文本的数据将不会被 API 相关概念解释提取系统的用户搜索到。

在完成上述工作的过程中，本文遇到了 URL 去重、文本相似度计算及数据集构建这三个关键问题。URL 去重时，不同场景下有不同的解决方案。本文提出了无重复方案及基于 Hash 的解决方案，并在实现网络爬虫时选择最适合的方案。计算文本相似度的方案也有多种，本文比较了各种解决方案的优缺点及适用场景，最终选择余弦相似度方案计算文本相似度。数据集构建主要分为数据收集和数据标注这两个步骤。本文编写网络爬虫，应用深度优先搜索策略进行数据收集，并且在数据收集时自动标注。

本文在最后对分类器及整个系统进行了实验评估。其中分类器对 Java 官方文档上 Java 类解释的预测效果不理想，这是因为整个数据集是基于维基百科上面的词条构建的，对 Java 类解释不具备分类能力。整个系统对于与 Java 相关度较大的概念，能够给出匹配度高的解释；对于与 Java 相关度较低的概念，只能给出匹配度低的解释，甚至不给出解释。

综上所述，本文设计并实现的 API 相关概念解释提取系统能够快速获取 Java 领域内某个概念的各种解释，为开发者省去了在各平台查找和检索概念解释的时间，提高了开发者的开发效率。

5.2 未来展望

本文实现了 API 相关概念解释提取系统，但仍然存在分类器效果不佳、系统返回的概念解释匹配度不高、不适于查询 Java 领域外的概念解释等问题。未来可以扩大数据集的构建范围，使其包含来自各个平台的数据，以提升分类器的分类效果；可以实现更复杂的关键字匹配算法，以提高关键字和概念解释的匹配度；可以投入更多精力，以收集 Java 领域外的概念解释。

参考文献

[1] Thomas H.Cormen，Charles E.Leiserson，Ronald L.Rivest，Clifford Stein. 算法导论[M].殷建平，徐云，王刚，刘晓光，苏明，邹恒明，王宏志译.3 版.北京：机械工业出版社，2013.

[2] 黄文彬,车尚锟.计算文本相似度的方法体系与应用分析[J].情报理论与实践,2019,42(11):128-134.

[3] T Mikolov, K Chen, G Corrado, et al.Efficient Estimation of Word Representations in Vector Space[J].Computer Science,2013.

[4] LeCun Y, Bottou L, Bengio Y, et al.Gradient-based learning applied to document recognition[C].Proceedings of the IEEE,1998,86(11):2278-2324.

[5] 周飞燕，金林鹏，董军.卷积神经网络研究综述[J].计算机学报,2017,40(06):1229-1251.

[6] Scrapy[EB/OL]. https://docs.scrapy.org/en/latest/.

[7] spaCy[EB/OL]. https://en.wikipedia.org/wiki/SpaCy.

[8] Keras[EB/OL]. https://keras.io/.

[9] Vue.js[EB/OL]. https://cn.vuejs.org/v2/guide/.

[10] HTML[EB/OL]. https://developer.mozilla.org/en-US/docs/Web/HTML.

[11] JavaScript[EB/OL]. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[12] CSS[EB/OL]. https://developer.mozilla.org/en-US/docs/Web/CSS.

[13] 曹姗.基于 TF-IDF 特征提取的短文本分类方法[J].工业控制计算机,2018,31(04):109-110.

[14] 吴波，朱昌杰，任逸卿.文本分类技术探究[J].宿州学院学报,2012,27(05):19-23.

[15] 周震卿,韩立新.基于 TextCNN 情感预测器的情感监督聊天机器人[J].微型电脑应用,2019,35(05):104-106+110.

[16] Kim Y.Convolutional Neural Networks for Sentence Classification[C].Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing.EMNLP 2014, Doha, Qatar, October 24-29, 2014:1746-1751.

[17] Glove[EB/OL]. https://nlp.stanford.edu/projects/glove/.

[18] HTTP[EB/OL]. https://developer.mozilla.org/en-US/docs/Web/HTTP.

[19] View UI[EB/OL]. https://www.iviewui.com/.

致谢

从最初踏进复旦到现在，时间一晃便过了四年。在本科阶段的学习和生活中，我体验过快乐，也经历过痛苦。但在这一切都经历过后，我的专业技能不仅得到了很大的提升，心性也更加成熟。

十分感谢彭鑫老师给予的指导和帮助。彭老师教学严谨且方法得当。在最开始选题时，彭老师给出了众多选题供学生挑选，这让我可以选择自己感兴趣的课题。在确定选题后，彭老师要求我们定时汇报论文进展并安排相应的学长进行一对一的辅导，这让我可以及时解决遇到的问题。

感谢唐嶔崟、魏入磊、朱小宁、王萌同学以及王翀学长在本文写作期间给予的意见，大家互相监督，相互学习。