南京大学

研 究 生 毕 业 论 文

（申请工程硕士学位）

论 文 题 目	基于知识图谱的开源社区关联性自动化构建系统
作 者 姓 名	李成浩
学科、专业名称	工程硕士（软件工程领域）
研 究 方 向	软件工程
指 导 教 师	陈振宇 教授

2022 年 05 月 20 日

学　　号：**MF20320077**

论文答辩日期：**2022 年 05 月 20 日**

指 导 教 师：　　**（签字）**

Automated system for building open source community relevance based on knowledge graphs

By

Chenghao Li

Supervised by

Professor Zhenyu Chen

A Thesis
Submitted to the Software Institute
and the Graduate School
of Nanjing University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Engineering

Software Institute

May 2022

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目： 基于知识图谱的开源社区关联性自动化构建系统

工程硕士（软件工程领域） 专业 2020 级硕士生姓名： 李成浩

指导教师（姓名、职称）： 陈振宇 教授

摘 要

随着互联网的不断发展，海量的信息充斥着我们的生活，同时随着各类开源社区的用户量不断增多，由不同用户进行讨论、资料分享和项目共享，开源社区俨然已经成为了学习的重要场所。但也正因为开源社区开源的特性，所有用户都可以参与问题的讨论回答、项目的修改更迭，这就导致了开源社区数据冗余且分散、知识密度低的问题。用户在利用开源社区带来的资料、项目的便利的同时，又因为开源社区知识密度低使得用户不得不花费太多时间在信息筛选上，最终导致用户对开源社区的利用率不高。

为解决上述开源社区中存在的问题，本文设计并实现了基于知识图谱的开源社区关联性自动化构建系统，通过利用知识图谱的特性，挖掘开源社区中用户文本描述间存在的关联性。本文使用 Scrapy 组件来高效爬取开源社区的文本内容，对爬取到的原始文本，使用深度学习模型进行实体和关系的识别与抽取并使用 KNN 算法来对实体进行分类。使用 CNN-BLSTM-CRF 联合模型进行实体识别，使用 PCNN 模型挖掘实体间的关联性，将抽取到的实体和关系存储到数据库中作为知识库用以知识图谱的构建。

本系统选择使用 Vue 框架设计系统的前端页面，使用 Spring Boot 框架来设计系统的后端，使用 D3.js 来实现知识图谱的可视化，在数据库方面，使用 nosql 型数据库中的 MongoDB 来对数据进行持久化操作，使用 python 脚本构建知识图模块，并使用 Neo4j 图数据库存储知识图谱相关的内容。同时，通过使用 Redis 作为系统的缓存机制，并引入 Nginx 实现负载均衡以提高系统的稳定性和可用性，使用 Docker 容器进行系统的部署以保证系统的可扩展性和可移植性，以遵守系统设计高内聚、低耦合的设计理念。

系统进行了对应的功能测试和非功能测试并顺利通过，可以证明系统满足预期，系统实现了知识图谱构建功能和关联性挖掘功能，可以通过在实体之间构建最短的联通路径从而帮助用户发现各种实体之间潜在的关联性，帮助用户在海量数据中查找直接需要的信息，从而提升用户对开源社区的使用效率。

关键词：开源社区，知识图谱，实体识别，关联性挖掘

南京大学研究生毕业论文英文摘要首页用纸

THESIS: Automated system for building open source community relevance based on knowledge graphs

SPECIALIZATION: Software Engineering

POSTGRADUATE: Chenghao Li

MENTOR: Professor **Zhenyu Chen**

Abstract

With the development of the Internet, a huge amount of information is flooding our lives, At the same time, as the number of users in various open source communities continues to increase, and different users conduct discussions, data sharing and project sharing, open source communities have become an important place for learning. However, it is precisely because of the open source characteristics of the open source community that all users can participate in the discussion and answering of questions, and the modification and replacement of the project, which leads to the problem of redundant and scattered data and low knowledge density in the open source community.Users have to spend too much time sifting through the information because of the low knowledge density of the open source community, which ultimately leads to low utilisation of the community.

To solve the above-mentioned problems in open source communities, this thesis designs and implements an automated system for building relevance in open source communities based on knowledge graphs, which exploits the characteristics of knowledge graphs to explore the relevance of user text descriptions in open source communities. This thesis uses Scrapy components to efficiently crawl the text content of open source communities. For the original text crawled, a deep learning model is used to identify and extract entities and relationships and a KNN algorithm is used to classify the entities. A joint CNN-BLSTM-CRF model is used for entity recognition, a PCNN model is used to mine the correlation between entities, and the extracted entities and relationships are stored in a database as a knowledge base for the construction of the knowledge graph.

The system uses the Vue framework to design the front-end pages, the Spring Boot framework to design the back-end, D3.js to visualise the knowledge graph, MongoDB, a nosql database, to persist the data, python scripts to build the knowledge graph module, and Neo4j graphs to store the knowledge graph related content. At the same time, Redis was used as the system's caching mechanism and Nginx was introduced to achieve load balancing to improve the stability and availability of the system. Docker containers were used to deploy the system to ensure scalability and portability, in order to comply with the design philosophy of high cohesion and low coupling in the system design.

The system has carried out corresponding functional tests and non-functional tests and passed it successfully, which can prove that the system meets the expectations. The system realizes the function of knowledge graph construction and correlation mining, and it can help users discover potential associations between various entities by constructing the shortest connection path between entities. It can also help users find the information they need directly in the massive data, thereby improving the efficiency of users' use of the open source community.

Keywords: Open Source Community, Knowledge Graph, Entity Recognition, Relevance Mining

目录

表 目 录

图 目 录

第一章　引言

1.1　项目背景及意义

开源社区一直以来都是极佳的学习场所，以 Stack Overflow 和 GitHub 为例，作为一个技术答疑网站，在 Stack Overflow 中包含大量专业水平极高、深度极深的各领域问题与解答；GitHub 则是一个源代码托管，大量技术人员在该平台学习源代码和练习项目，并提供团队合作方式，让不同的人得以共同开发项目，因而需要重视针对开源社区的研究与利用 [1]。开源社区的最核心的特点就是它的开源性，这也就意味着所有的用户都可以参与其中，开源社区的数据量也越来越大，必然会导致开源社区知识密度低、数据冗余且分散的状况。如果要在海量的数据中找到需要的数据或资料，需要用户自己进行仔细的甄别和筛选，显然这将是一个复杂冗长的过程，通常来说，对于 GitHub，用户可以根据每个项目的 stars 来判断这个项目的价值，对于 Stack Overflow，用户则可以根据每个回答的支持率 (votes) 和浏览量 (views) 来判断回答的价值。但这并不能很好地解决开源社区数据冗余且分散的问题，为此亟需一种工具来帮助用户在海量数据中简洁、直观地罗列出用户需要的信息。

伴随着信息技术的不断发展，互联网技术也得到了极大发展，作为互联网技术中关键的 Web 技术也相应得到了突破。现在的 Web 技术已经不再是从前简单的网页，而是变成了用户与复杂系统交互的语义网络，Web 技术正在逐步向语义网络 [2] 的方向演变。

知识图谱本身是一个语义网络，语义网络即是由数据所构成的网络（Web of data），利用语义网络技术，用户可以得到期望的查询环境，语义网络本身可以将知识进行加工、推理并以数据结构中图的结构返回给用户。知识图谱作为一个全新的概念首先由谷歌提出，在我国，中文知识图谱已经由各大厂商、机构、学校开始着手研究和使用，并且已经取得了一定的研究成果。同时，随着谷歌公司对知识图谱技术地不断完善、各机构对知识图谱技术的不断钻研，国内外基于知识图谱而生的产品正改变着我们的生活，例如 WordNet、DBpedia 等，因此知识图谱技术也得到了学术界的重视。

知识图谱相较于其他语义网络工具所不同的是，知识图谱注重于以更加智能化的方式实现语义检索，例如，传统的搜索引擎技术为了提高其检索效率往往会计算用户查询的比重进行网页的快速排序。然而，由于信息技术的发展导致的网络的文本规模在不断增长，这种传统的网页检索方法已经不再能够保证

用户可以快速地得到想要的信息，甚至无法保证用户可以正确地得到想要的信息，为了保证检索信息的正确性，通常需要引入人工审查地环节，而一旦引入人工审查，其效率就不可避免的大打折扣，为了解决这样的问题而在搜索引擎及信息处理环节中引入知识图谱作为辅助 [3]。因此本文选择利用知识图谱的特性，利用深度学习算法对开源社区的文本内容进行数据清洗后，构建用户需要的知识图谱帮助用户从开源社区筛选出相关联的信息，并以知识图谱实体关系连线的形式帮助用户选择自己需要的相关实体信息。

知识图谱的构建需要结构化的数据 [4]，但两个社区的数据类型是复杂的，它们都有如每个项目或回答都有对应的标签等半结构化数据，也有用户的自然语言回答、公式块、代码块等非结构化数据，本文所关注的是开源社区中纯文本的内容，因而在获取文本的时候需要剔除页面中的公式块、代码块 [5]，然而问题之一在于不同的开源社区有着不同的规则，针对开源社区进行文本获取需要针对不同社区设计独特的爬取规则。此外，得益于这些年来自然语言处理技术 (NLP，Natural Language Processing)[6] 的不断发展，许多专家学者开始将深度学习的算法结合神经网络处理人类的各种语言文本信息。为了顺利完成开源社区海量文本中实体和关系的识别与抽取的数据结构化工作，本文选用自然语言处理技术中的深度学习模型 [7] 对纯文本中的自然语言内容进行分析，抽取其中的实体与关系 [8]。为了顺利构建用户需要的知识库，则需要对不同的模型进行比较，选出最适合的本系统的模型并进行训练。整体而言，本文通过利用知识图谱的特性和深度学习模型分析来分析开源社区的文本内容，通过构建用户期望的知识图谱来帮助用户筛选开源社区中有用的信息，从而提高用户对开源社区利用效率。

1.2 国内外研究现状

1.2.1 知识图谱研究现状

知识图谱 [16] 是一种基于图的网络数据结构，是在大数据背景下的应运而生的一种技术，它由实体 (即节点) 和关系 (即边) 构成，将知识以 <head，relation，tail> 的三元组形式存储，其中 head 表示头部实体，tail 表示尾部实体，relation 表示两者之间的关系，并通过图的形式表现出来 [17]，使得用户可以直观的观察诸多实体之间的关系。知识图谱是大数据时代催生的产物，利用知识图谱的特点可以简洁明快地表现物品与物品、人物与人物、物品与人物之间的关系，目前已经广泛的应用于如语义搜索 [18]、智能推荐 [19][20] 等领域。

知识图谱可以分为通用知识图谱和领域知识图谱两大类，国内外的相关知识图谱调研如表1.1所示。

表 1.1: 知识图谱调研

知识图谱	规模	构建特点
WordNet[9]	包含 15 万个词和 20 万个语义	主要用于消除词义之间的歧义，构建各种类型单词之间的关系。
DBpedia[10]	2800 万个实体，30 亿个 RDF 三元组	在多种语言的维基百科中提取信息，并使其以关联数据的形态发表后，它就构成了一种包括对人物、地点、歌曲、电影等定义的本体。
YAGO[11]	1000 万个实体，1.2 亿条三元组知识	建立了一个比较丰富的实体分类系统，给很多条目添加时间和空间层次上的属性说明。
Knowledge vault[12]	4500 万个实体，4469 种关系，2.7 亿条三元组	自动到网络上收集信息，通过深度学习和机器学习算法进行数据清洗，从而转变为可用知识。
知心	现有教育、医疗、游戏等多个知识集群	百度公司研发和打造的中文信息图谱，主要运用于帮助百度等搜索引擎实现深度检索和实体推荐。
Zhishi.me[13]	1400 万个实体来自于百度百科，550 万个实体来自于互动百科，90 万个实体来自于中文维基百科	抽取各种开放百科中的数据并进行结构化处理，对外通过一个 SPARQL 终端提供用户查询功能。
OpenKN[14]	3000 万个实体，10 亿条边	通过不断获取新知识，不仅更新自身包含的知识，而且存储其他开放知识库中有用的知识。
CN-DBpedia[15]	1600 多万个实体，2 亿多个关系数，产生 10 亿次 API 调用	在 DBpedia 的基础上进行改进，可以处理纯文本信息，将纯文本信息经过文本过滤、知识融合、知识推断等数据清洗操作，最终形成可用的结构化数据。

领域知识图谱，顾名思义就是面向那些专业领域而进行专门服务地知识图谱，以行业内部的统计信息为基础建立；通用的知识图谱则面向于普通领域或大众领域。

知识图谱大致构建过程如图1.1所示，通过获取需要的结构化、非结构化数据并通过知识抽取 [21][22]、知识融合 [23] 等过程构建知识图谱。知识抽取会提取出数据中地实体和实体间地相互关系等信息。在知识构建完成后，可以利用已经构建完成地知识库挖掘实体间隐含的联系，同时可以利用外部知识库作为扩充来完善知识图谱。

鉴于知识图谱是一种有向信息异构网络，其通过对数据的规范与整合向人们提供有价值的结构化信息，且因其数据都是结构化的特点，在利用知识库中实体与关系间联系的基础上通过不断挖掘，从而在实体之间建立连通路径以进一步发掘出实体之间的潜在关联性 [24]。

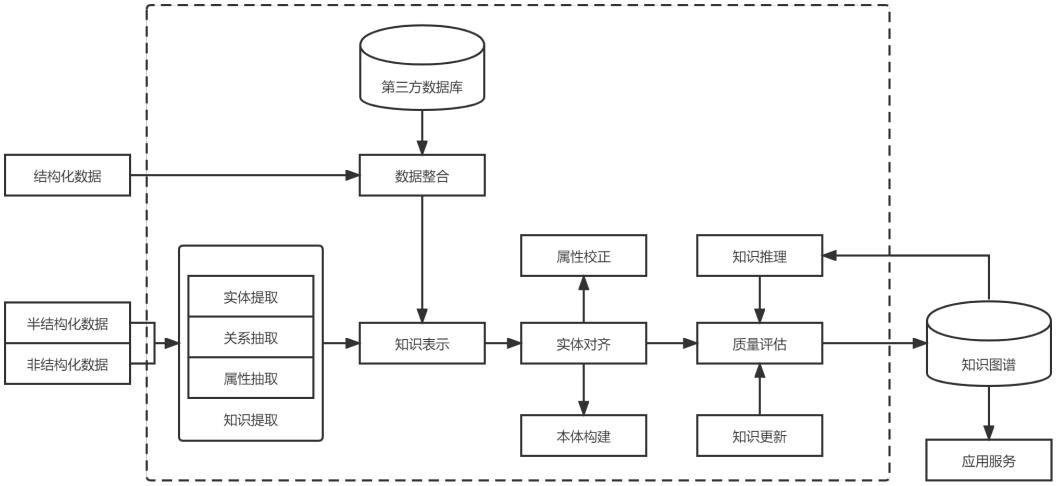

图 1.1: 知识图谱构建

1.2.2 实体与关系抽取模型研究现状

知识图谱的构建需要依赖实体和关系，而很多的实体与关系都隐藏在海量的文本之中需要去进行甄别与挖掘 [25]，其中又以关系抽取更为困难，这项工作如果人为来进行不可避免会消耗很多时间且仍然存在着诸多难以发现的关系，因而需要使用自然语言处理所带来的便利。

对于实体抽取 [26]，在很大程度上依赖分词的结果，如果分词效果不佳，往往导致模型错误识别实体的边界，因此机器学习的实体抽取方法被逐渐应用并不断改进，包括 Maximum Entropy(MEM)[27][28] 模型，Hidden Markov(HMM)[29] 模型以及 Conditional Random Field(CRF)[30] 模型等。随着神经网络 [31] 的出现，深度学习方法不再像先前那样需要大量的数据训练模型需要的特征。

现在已经由许多机器学习和神经网络相结合的模型被应用于实体识别与抽取工作中，例如最为基础的循环神经网络 (Recurrent Neural Network, RNN)[32] 和卷积神经网络 (Convolutional Neural Networks，CNN)[33]，以及克服了传统 RNN 模型可能会导致梯度弥散的问题的长短时记忆网络 (Long-Short Term Memory，LSTM)[34]，还有在原本 LSTM 模型的基础上引入信息累计的速度控制技术来解决 RNN 模型中存在的长期依赖问题的双向长短时记忆网络 (Bi-directional Long-Short Term Memory，BLSTM)[35]，BLSTM 模型可以更好地捕捉语义依赖。由于 CRF 模型则克服了 MEM 可能会产生标签偏差的问题而成为最为广泛使用的实体识别抽取模型，因而 MA X 等人 [36] 和 CHIU J P 等人 [37] 分别在 ACL2016 上先后通过文章提出了将 CRF 和 BLSTM 相结合的神经网络的方法，首先通过

CNN 模型将自然语言转为基于字符的词向量传入到 BLSTM 模型中，得到的结果利用线性 CRF 进行解码，这样的联合方式在不同领域都取得了很好的效果。

对于关系的抽取 [38]，通常选择利用 CNN 模型，或者在此基础上发展出来的诸多 CNN 模型的衍生模型。

Simple CNN[39] 模型首次引入 CNN 结构进行关系抽取，CNN 模型的结构简单到没有池化层，且使用 Synonym Embedding 作为词的特征而非词向量嵌入，正是因为使用的 CNN 模型结构简单，在没有池化 (Pooling) 层的情况下受噪音影响就比较明显，也没能真正做到端到端 (end-to-end) 的关系抽取。

CNN with Max Pooling and Word Embedding[40] 模型则在 CNN 模型的基础上引入位置特征 (Position Feature) 作为位置信息，因为 CNN 更多是在局部使用 N-Gram 模型计算特征，而位置特征更加注重句子中每个实体 (entity) 带来的影响，从而不再将实体和其他词笼统考虑。尽管模型不再像 simple CNN 模型一样使用单个卷积核，但因为引入的多个卷积核都是同一个缓冲包尺寸 (windos-size)，这样的结果就是只能提取比较单一地特征，这就导致引入最大池化 (Max Pooling) 和词嵌入 (Word Embedding) 的 CNN 模型反而更像是基础 CNN 模型文本分类的结构。为此，Nguyen 等人提出了 CNN with multi-sized window kernels[41] 模型，其在引入最大池化 (Max Pooling) 和词嵌入 (Word Embedding) 的 CNN 模型基础上更进一步，通过引入多尺寸 (multi-sized) 卷积核，改变了以往多个卷积核使用同一个缓冲包尺寸的问题，从而得以在局部使用 N-Gram 模型提取更多的特征，但问题在于该模型使用的仍旧是传统的 CNN 模型结构，性能提升有限。

Multi-Level Attention CNN[42] 模型同样以 CNN 模型为基础，通过引入两层 Attention 机制来标记句子中对关系标签有更大的影响的部分，同时使用 word similarity 来标记单词与目标实体的相似度，两个 Attention 机制均基于 embedding 内积进行运算，但也存在着结构复杂度高，收敛困难的问题。与之类似的 Attention CNNs[43] 模型则是通过设置每个单词在不同的关系中设置不同的权重 Attention 矩阵。

最后是 Piecewise Convolutional Neural Networks(PCNN)[44] 模型，输入的是 word embedding 和 position embedding，之后会交付到卷积层进行卷积计算，并且选择的是局部最大池化而非全局最大池化，通过将句子以两个关键实体划分为三段，从而更加有效的得到句子信息 [45]。

综上所述，关系抽取方向已经有着相对成熟的技术，模型总结如表1.2所示，现有模型的改进也依旧在不断进行，因而对于知识图谱的构建工作将带来诸多便利，同时由于 PCNN 模型在进行关系抽取时与上述各类模型比较有最高的精度、召回率和 F-Score，因而本文选择 PCNN 作为关系抽取模型。

表 1.2: 关系抽取模型总结

模型名称	模型特点
Simple CNN Model	1. 首次尝试利用 CNN 处理关系抽取，但 CNN 模型较简单。 2. 由于没有池化层，训练时易受噪声影响。 3. 使用同意词嵌入而非词嵌入。 4. 依然需要传统机器学习中的特征训练，并非真正端到端的关系抽取。
CNN with Max Pooling and Word Embedding	1. 在原有 CNN 模型基础上引入池化层。 2. 引入位置特征作为位置信息，使得关键实体可以与其他词区分开来。 3. 使用多个卷积核，并对每个卷积核计算最有特征。 4. 使用词嵌入代替原来的同意词嵌入。 5. 尽管引入多个卷积核，但计算卷积核特征所用的缓冲包尺寸却是同一个，因而提取到的最优特征区分度有限。
CNN with multi-sized window kernels	1. 放弃了 CNN with Max Pooling and Word Embedding 中人为引入的词汇特征，减轻了人工训练的压力。 2. 卷积核之间不再使用统一标准的缓冲包尺寸，抽取到的最优特征更有区分度。 3. 模型任然采用的是传统 CNN 结构，尽管引入池化层和复数的卷积核，效果提升有限。
Multi-Level Attention CNN	1. 引入两层 Attention 处理句子中的关系标签。 2. 使用单词相似度处理句子中的各个单词。 3. 由于引入了两层 Attention，不仅导致模型复杂化，还可能导致模型收敛困难的问题。
Attention CNNs	1. 该模型在 Multi-Level Attention CNN 的基础上引入权重矩阵，使得模型在进行关系抽取时直接关注到词和关系。 2. 使用最大池化代替原来的权重池化。
Piecewise Convolutional Neural Networks	1. 输入的是词嵌入和位置嵌入用以计算。 2. 放弃原有的全局最大池化，选用局部最大池化计算各个卷积核的最优特征。 3. 用两个关键实体将句子划分为三个部分，可以更加有效地抽取句子特征，提高关系抽取的准确率。

1.3 本文主要工作

本文所实现的系统是一个针对于开源社区的知识图谱构建系统，旨在通过爬取开源社区的讨论回答、项目文件，通过数据结构化清洗来构建实体之间的关联性知识图谱构建自身的知识库，在此基础上通过不断建立实体之间的联系，发掘实体之间通常难以发现的关联，以此协助数据整理人员从海量数据中找到需要的信息。

针对上文中提到的开源社区数据量大且复杂的问题，利用知识图谱可以将

各项关系以图的形式直观的展现给用户，同时在知识库的不断扩充后，以实体与实体间最短连通路径的形式挖掘实体间的潜在关联可以帮助用户更好地利用开源社区中零星散落的知识，这些便是知识图谱的优越性，总的来说，知识图谱以其图谱的形式直观的展现各个实体之间的关联，特别适用于在数据量巨大的内容中找寻需要的信息，因而本文从选择知识图谱的特性出发，提出了基于知识图谱的开源社区关联性自动化构建技术，希望通过知识图谱的特性来展现实体与实体之间的关系。

生成知识图谱所需要的数据是结构化数据，而在实际生活中开源社区的数据往往是用户直接编辑的自然语言，也就是非结构化数据为主，辅之以作为标签的半结构化数据。对于 GitHub 和 Stack Overflow 来说，每个项目或回答中中存在的还有代码块、公式块等结构，而 GitHub 中还有项目文件等部分存在，这些需要在生成的时候剔除，然后再针对文本进行识别，找到文本中于目标实体的关联性，进而生成知识图谱，并在之后不断丰富知识库。

然而，在知识图谱生成并通过不断知识融合壮大已有的知识图谱后，其规模可能过于庞大，需要用户来控制规模的大小。此外，本文旨在发掘实体间通常难以发现的联系，但其联系并不一定就是正确的，因此在图谱生成后，用户需要以图谱为辅助工具来辨识自己需要的模块。

本文主体部分将采用 Spring Boot 框架进行搭建，考虑到知识图谱的构建需要不断到各个页面上进行内容爬取因而使用 Python 语言来进编写，并通过 Thrift 连通两个语言模块。为保证系统的可用性，本系统将采用 Nginx 来进行负载均衡，并采用 Redis 作为缓存。

1.4 本文组织结构

本论文大致分为六个章节，其组织结构如下：

第一章，引言。介绍了知识图谱的发展现状及国内外的相关研究，列举、比较实体识别和关系抽取常用的模型，并提出了生成知识图谱所面临的一些问题，以及本项目的主要任务和为解决问题选用的深度学习算法。

第二章，相关技术概述。首先介绍了构建系统所涉及到的相关工具及框架做了简单的介绍，并对选用的关系抽取模型 PCNN 模型进行简要概述，随后介绍系统构建时需要用到的一些组件和实现模型的深度学习框架。

第三章，系统的需求分析与设计。由于本文通过知识图谱来对开源社区的文本内容进行分析，核心就在于对开源社区的自然语言处理，在功能性需求方面考虑用户在构建知识图谱时所需要的相关功能，如知识库构建、知识图谱构

建等，非功能性需求方面则在用户视角考虑包括系统的易用性、系统的可移植性方面进行设计，辅之以重要模块的详细介绍共同阐述系统的设计思路。

第四章，系统的具体实现。在完成需求分析后，根据系统的每个模块设计为切入点，详细介绍系统的实现过程，并通过统一建模语言进一步介绍系统间模块的交互情况，对于系统关键的功能部分将展示关键代码，提供相关的界面展示。

第五章，系统测试。在完成系统构建后，将以本文提出的系统级需求作为本文的验收标准，设计对应的测试用例进行验收测试，在完成测试后分析系统的测试结果，对系统功能进行验收。

第六章，总结与展望。通过本章回顾本文对基于知识图谱的开源社区关联性自动化构建系统的构建，总结本文所用的相关技术以及模块实现，分析不足，提出改进方向。

第二章　相关技术概述

2.1　系统架构相关技术

为了保证代码的可读性、高可用以及可扩展性，本系统在构建时选择利用成熟的框架作为辅助，在前端，主要使用了 Vue 框架，在后端使用 Spring Boot 框架，使用 Docker 容器 [46] 完成项目的环境配置与系统部署。数据存储方面选择使用 MongoDB 数据库进行数据存储，而知识图谱相关数据则采用 Neo4j 图数据库进行存储，为实现系统的高可用性而选择 Redis 作为系统的缓存中间件避免太多请求造成的数据库压力。

2.1.1　Vue 框架

框架本身有着可以规范并简化软件开发过程、集成相关工具等特性得到广泛使用。因此在开发过程中选择使用框架进行辅助开发可以方便项目的进展，提高开发效率。

Vue 作为一个广受欢迎的前端架构，由于其渐进式的特性不同于其他的前端架构，使 Vue 在进行了前端建设之后能够自底向上逐步搭建，并且由于 Vue 的核心库只关注于视图层，它能够很容易地和所有项目实现集成，并且很容易导入第三方库，也就表明了 Vue 本身就是容易掌握的。同时，Vue 不但能够对更复杂的单页应用进行驱动，并且通过 Vue 所设计出的页面也是响应式的，这使得这类网页都能得到很好的用户反馈和效果，同时比之于传统页面利用超链接的方式进行页面切换和跳转，Vue 使用路由不会导致页面的频繁刷新，通过利用 Vue 的组件开发可以大大减少代码的编写量。

基于上述的优点，本文在前端开发过程中使用 Vue.js 框架进行开发，以保证完成前端设计的同时保持前端界面的简洁美观。

2.1.2　Spring Boot 框架

Spring 框架是 Java 程序开发的开源框架，Spring 框架的出现为 Java 语言带来了新的活力，也正是其控制反转、依赖注入的特性，为大规模 Web 程序开发时开发人员带来了诸多便利。Spring 框架所提供的另一特点是面向切面编程，开发人员可以通过声明事务管理简化开发流程，且 Spring 框架本身是开源的，由众多开发人员不断注入心血，使得 Spring 框架得以兼容其他多种 Web 框架来简化开发，也正是由于与 Web 框架的兼容使得 Java 语言在 Web 领域有着举足轻

重的地位。

尽管 Spring 是轻量级框架，但其需要由开发人员自己去编写大量 XML 文件进行相关配置仍然给开发带来了不少麻烦。Spring Boot 框架不仅保留了原来 Spring 框架带来的便利，还可以自动进行配置工作，简化操作流程，可以说 Spring Boot 框架的出现就是改变了原有 Spring 的诸多不足，同时又有诸如：嵌入 Servlet 容器免去打包运行，通过注解进行文件配置，配置自动化等优势而再次给 Java 语言注入活力，因而 Spring Boot 框架成为最热门的 Java 开发框架。

正是由于 Spring Boot 框架成熟且易于使用的特性，本文选择在后端开发的过程中使用 Spring Boot 框架。

2.1.3 Docker 容器

软件开发需要一定的环境依赖，不同的语言、不同的操作系统、不同的技术都需要特定的运行环境，因而环境配置是重要的环节。为了能够保证在程序开发过程中、程序开发完成后系统的可移植性，在不同环境下部署系统能够运行的运行环境是极为重要的一个问题。

为了解决运行环境的问题，开发人员往往会选择在不同的系统上安装同一个虚拟机，并在虚拟机上尝试还原出本来的环境配置，但虚拟机作为一种特殊的应用程序会大量占用系统资源，影响其他程序的正常运行，且很多时候虚拟机的安装配置是一个繁琐的过程，因此该方法正逐渐被淘汰。

为了能够解决虚拟机带来的诸多不便，Linux 提出了虚拟化容器技术，虚拟容器不是一个完整的操作系统，仅仅是将程序打包隔离，用虚拟环境下的资源供其运行。Docker 容器就是虚拟化容器技术的最为出色的实现，它是由 dotCloud 在 2013 年发布的开源应用容器引擎，提供接口供开发者方便的创建、启动和管理容器。在使用 Docker 对程序进行打包后，Docker 会将应用程序本身和需要的环境依赖写入一个称作“镜像”的文件中去 [47]，当用户需要在其他操作系统或其他环境中运行该程序时，只需要运行这个镜像就能在镜像配置的虚拟环境中运行该程序，避免了开发人员再度进行环境配置。

考虑到系统可能需要在不同的操作系统上运行，为实现系统的可移植性，简化环境配置问题而选择使用 Docker 容器进行系统部署。

2.1.4 MongoDB

信息技术的不断发展带来的是数据量的爆炸性增长，传统关系型数据库在应对如此之多的数据并不保证高效地存储与访问，对于数据的可扩展也存在一定困难。为了解决关系型数据库遵守的数据一致性约束所带来的影响，人们开

始考虑非关系型数据库。非关系型数据库不再像传统关系型数据库那样被表的结构所限制，同时也因其没有 ACID 的约束而有着良好的可扩展性，相较于传统的关系型数据库有着更好的使用前景。

MongoDB[48] 是 C++ 语言开发的开源的高性能文档型数据库。MongoDB 的数据结构类似 Redis 中的键值对的形式，因而可以存储较为复杂的数据类型，包括那些易变化的数据。由于是键值对的存储形式，根据键来搜索是一个非常迅速地过程，同时 MongoDB 也可也可以像传统数据库那样通过构建索引对查询进行优化，即使是在数据量特别大的情况下也能够完成实时修改和更新的需求。

为了知识图谱能够顺利构建，需要旁的知识库作为基础，爬取的数据需要进行存储，这些信息非常复杂且随着系统功能需求变化可能存在变更，需要经常访问，因此在 Spring Boot 框架中引入 MongoDB 依赖，使用 MongoDB 数据库进行存储保证访问的效率。

2.1.5 Redis 缓存

由于知识图谱的构建不可避免会频繁访问数据库，这将给数据库带来不少的压力，例如频繁访问导致的表锁定，数据库是存放在硬盘中的，而硬盘的访问速度有限，为了解决硬盘访问速度的问题，可以考虑将这部分数据存放到内存中进行内存读写以提升效率。

为了实现将硬盘读写变成内存获取，Redis 可以满足需求。Redis 作为出众的内存数据库，是常见的用以提升系统性能的中间件，CPU 对内存的读写是一个非常迅速地过程，因此尽管 Redis 是单线程的，但 Redis 仍然能够应对高并发的情况，同时 Redis 的所有操作都是原子的操作，同时会定期将内存中的所有数据持久化到硬盘中，原子性保证了数据的一致性，数据持久化到硬盘中又避免了内存在断电等异常情况导致的数据丢失保证了数据的持久性，因此 Redis 被广泛用作缓存中间件。

2.1.6 Thrift 框架

Thrift 框架由 FaceBook 公司研发，是一款轻量级且支持多种编程语言的远程过程调用 (Remote Procedure Call，RPC) 框架。远程过程调用 (RPC) 协议的主要功能是让客户端能够在不考虑底层具体构造方式的情况下，实现远程调用服务端中的功能，这个调用的过程可以达到客户端调用程序本地对象的效果，不同在于 RPC 利用的是网络进行远程调用的服务的协议。

如图2.1所示的是 RPC 框架的流程示意，主要分为 RPC 客户端和 RPC 服务端两部分，客户端调用服务而服务端提供服务，服务端本身无法感知到客户端发

起的服务调用。RPC 框架通过网络传输请求，Thrift 框架则通过传输层中的 TCP 协议进行网络传输，调用发在发起请求时会将请求内容的参数对象进行序列化，使之不断序列化为二进制数据并通过协议编码后经由传输层通过 TCP 协议传输到服务端，服务端得到数据后会按照 RPC 协议进行协议解码和反序列化操作，根据还原后的数据查找到客户端真正请求的对象找到对应的服务完成方法调用，服务端完成发放调用后会同样将执行结果进行序列化、协议编码后交由 TCP 协议返回给 RPC 客户端，客户端获取到返回的数据后进行解码和反序列化得到执行结果，最终完成一次远程调用流程。

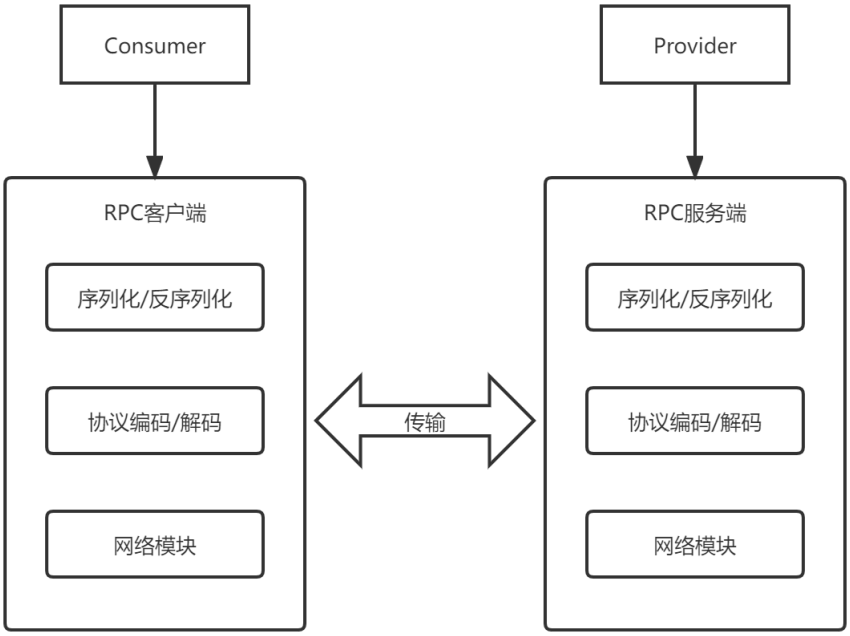

图 2.1: RPC 调用流程示意

Thrift 作为 PRC 协议的实现框架之一有其自身存在的优势，一方面，Thrift 框架可以为 RPC 通讯自动生成需要的代码，简化了一定的设计流程，另一方面，RPC 协议需要进行序列化和编码，Thrift 框架可以则对数据传输和序列化提供服务上的支持，对操作进行了简化处理，同时，RPC 协议并没有使用传统的 HTTP 传输协议，而是为增强通讯的效率选择 TCP 协议并简化了传输内容。Thrift 框架还提供了适应多语言版本的接口文件，用户可以以此完成不同语言开发的系统之间的服务调用。Thrift 框架还支持非阻塞模式和线程池模式等多种工作模式，也正是因为 Thrift 框架所具有的良好的适用性、性能以及跨语言通讯能力，可以提供高效的对外服务。

2.1.7 Scrapy 框架

Scrapy 框架是采用 Python 编程语言所实现的一种高效、快捷的屏幕爬取和 Web 爬取架构，主要用来从各个 Web 网页中发现和获取其中的结构化数据，也就是由于其效率快的优点而被广泛应用在数据挖掘和大数据监测等方面。

Scrapy 作为好用的爬虫框架，在 twisted 框架的基础上衍生而来，以非阻塞的方式完成并发操作，框架本身提供了诸如 BaseSpider、Sitemap 等多种基础爬虫类，且因其框架的特性，用户可以根据自身需要往框架中添加需要的内容，从而完成开发的需要，实现想要的功能。

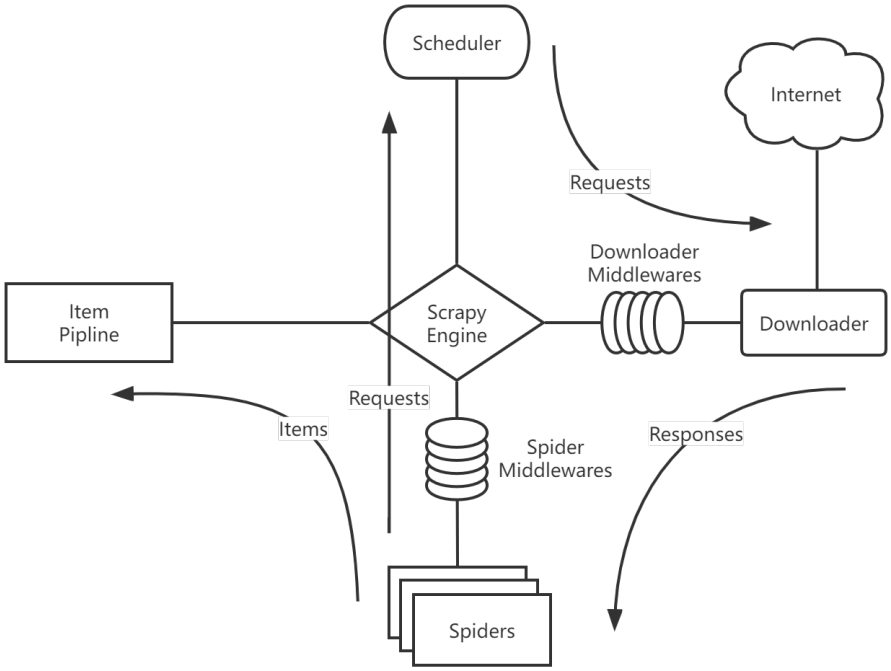

图 2.2: Scrapy 框架结构图

图2.2所示是 Scrapy 框架的结构图，其中，SrapyEngine 是引擎模块，用于控制框架中组件之间的数据流向，同时监听、处理各个组件的触发事件；Scheduler 是组件的调度器，可以理解为一个存储了 URL 的任务队列，调度器主要用来处理由 Engine 模块提交的请求，通过将 URL 存储到任务队列中决定抓取的网站；Downloader 是下载器，用于具体抓取页面上的内容，抓取到的内容会返回给 Engine 存储；ItemPiplines 是项目管道，主要是根据 Item 类将抓取到的页面内容进行数据清洗、验证等结构化操作并存储到数据库中；Spiders 即为爬虫中间件，用来处理爬虫爬取页面时的输入和输出。

当下网络信息呈现爆炸式增长，普通的爬虫无法应对如此大量的页面内容，同时也无法应对如网络中断、断电等突发情况，而 Scrapy 框架可以支持多线程操作，能够充分利用 CPU 资源，且 Scrapy 框架支持断点续传，能够在断电、网络中断等情况恢复后继续完成工作因而拥有稳定的性能。Scrapy 框架的工作流程如下：Engine 向 Scheduler 发起请求，Scheduler 将任务队列中的地址 (URL) 返回给 Engine 从而开始抓取任务，Engine 将得到的 URL 封装成 Request 提交给 Downloader 进行下载，下载后的页面资源封装成 Response 包交由 Spiders 进行解析，在 Spiders 完成 Item 的解析后由 Pipline 完成结构化处理。

本文为完成大规模的知识库构建，因而选择使用 Scrapy 框架来高效地进行分布式爬取。

2.2 知识图谱相关技术

2.2.1 余弦相似度

在进行实体分类的时候，不可避免需要用到词向量建得相似度，余弦相似度则是用两个向量间夹角的余弦值表示两个向量的相似度的一种方法。通过计算两个向量间夹角的余弦值，不仅可以知道两个向量的相似程度，还能了解两个向量在其对应维度空间中方向是否一致。定义两个向量夹角为 0° 时的余弦值 $\cos < \alpha,\beta >= 1$，其他任何角度的余弦值 $\cos < \alpha,\beta >= 1$ 都属于 $[-1,1]$ 这个区间，余弦值结果中的符号代表两个向量的方向关系，当两个向量拥有完全相同的指向时，其夹角为 0°，其余弦相似度的值 $\cos < \alpha,\beta >= 1$，而两个向量正交时，其余弦相似度的值 $\cos < \alpha,\beta >= 0$，当两个向量处在完全相反的指向时，也就意味着两个向量的夹角为 180°，其余弦相似度的值 $\cos < \alpha,\beta >= -1$。综上所述，余弦相似度的计算仅仅与两个向量的指向相关，并不考虑向量自身的长度，且余弦值的符号代表两个向量的方向关系。

余弦相似度的计算利用的是欧氏空间的计算公式，n 维欧氏空间中余弦相似度的计算公式如2.1所示，在欧氏空间的实数范围内余弦值 $\cos\theta \in [-1,1]$，这个余弦值的区间在任意维的欧氏空间都是适用的，且余弦相似性最常用于高维的欧式空间。例如，在信息检索过程中，每个词项都有可能包含不同的维度，每个维度都可以用一个向量进行表示，那么每个向量的值就可以代表该词项出现的频率。

$$cos\theta = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{2.1}$$

2.2.2 Neo4j 图数据库

Neo4j 作为流行的图数据库 [49] 之一，图数据库可以存储图形结构并提供对外的展现及查询，是一种不同于其他非关系型数据库的新型的图数据库，其以图论为基础来设计数据库的存储结构和数据的查找方式，连接节点的边则在图数据库中以节点间的关系表现出来，因而根据 Neo4j 的特点，可以用来存储知识图谱的所需要用到的图形结构、节点关系。

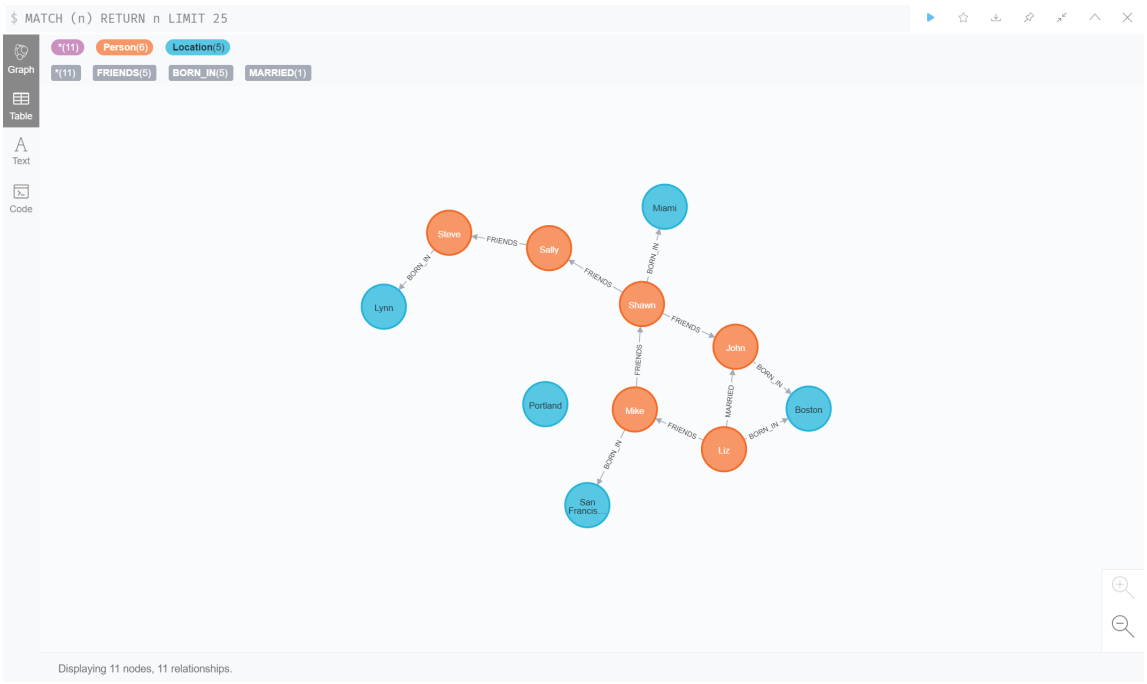

图 2.3: Neo4j 示例

Neo4j 可以称得上是目前图数据库中使用最为广泛的，其原因就在于 Neo4j 专门优化了图结构的存储与管理，例如当两个实体结点在知识图谱中相互关联，那么两个实体结点的物理地址也是相互关联的，这样的关联就为查找提供了便利。知识图谱本身就是图结构中的一种，因此非常适合使用 Neo4j 来存储并构建知识图谱。

2.2.3 KNN 算法

在实体识别的过程中，需要对文本的特征进行提取，考虑到 KNN 算法仅需要比较相似度而不必通过构建词向量，因而在这里选择 KNN 算法作为分类算法的分类器。KNN 算法的流程如1所示：

Algorithm 1: KNN 算法流程

输入: 已训练样本集及样本标签，未知分类标签的新增数据

1 训练集 $\{a_1, a_2, ..., a_n\}$，样本的训练标签，未知标签的新增数据.;

2 新增数据的分类标签.;

3 计算新增数据与现有数据之间的特征的相似度;

4 选择新增数据中与已有训练集相似度最高的 K 个样本，成为 K 邻近;

5 K 邻近中出现次数最多的分类标签即为新增数据的分类标签;

输出: 新增数据的分类标签

KNN 算法的原理：在已有训练集数据并且已经能够确定每个样本的分类信息的前提条件下，如果需要知晓一个新增数据的标签分类情况，则可以计算并比较新增数据和样本集中各个样本特征的相似读，按照相似度进行排序，选择出相似度最高的 K 个样本，这 K 个样本的分类标签就可以作为新增数据的分类标签。例如，现有 $a_1, a_2, ..., a_n$ 共 n 个向量，在计算相似度时往往采用计算欧式距离的方法来计算，假设其中 $a_r(x)$ 表示示例 x 的第 r 个属性值，那么两个实例 x_i 和 x_j 之间的距离就可以通过公式2.2计算。

$$d(x_i, x_j) = \sqrt{\sum_{i=1}^{n} (a_r(x_i) - a_r(x_j))^2} \tag{2.2}$$

在本文中，考虑到开源社区以文本描述为主，在爬取时筛去代码块、公式块等部分，因而对于文本中实体分类的需求，在这里选择中 K 为 10，即选择出已有训练集中 10 个相似度最高的样本标签作为新增数据的分类标签。对于后续实体抽取、关系抽取过程中已经构建词向量的训练集中，可以通过公式2.1计算向量间的余弦相似度，然后再选取 10 邻近构建新增数据的分类标签。

2.3 深度学习模型和框架

2.3.1 分段卷积神经网络模型

在使用机器学习方法进行关系抽取时不可避免的一个问题是训练集的构建，通常应对这个问题的方法是利用远程监督来训练，但是问题也随之而来，一则，远程监督的假设性太强，容易造成错误标注的情况，例如两个句子在共同表达同一个主题的情况下，提到这个实体的两个句子在知识库中并不能表述预期的关系，使用远程监督方法任然有可能将之选为一个训练实例，这必然会对基于

此类噪声数据进行模型训练的性能；二则，在使用传统的远程监督方法进行数据标注的同时，会利用自然语言处理工具添加一些训练特征，但由于自然语言处理工具中不可避免地存在一些问题或错误，因此使用这些传统特征将会导致错误的不断积累，同时，远程监督方法实现的关系抽取往往会直接从网络上抽取文本，这其中就会存在很多非正式文本，因此当句子长度不断增加时，对句子的词法分析、句法分析的准确度显著降低。

为解决以上问题而提出了分段卷积神经网络 (Piecewise Convolutional Neural Networks，PCNN) 模型。通过 PCNN 模型中引入的多实例学习功能，在多实例学习流程中，训练集由多个包 (bag) 构成，每个包中包含多个实例。包的标签是已知，但包中的每个实例的标签未知，通过设置包级别的目标函数，在以包为单位的多实例学习流程中，包中实例标签就将被纳入考虑，避免了因为假设性太强而造成的标签错误的问题；另一方面，通过采用卷积结果来自动学习模型需要的相关特征，将每个卷积核进行局部最大池化以获取到其最优特征，以此来避免用自然语言处理方法进行预处理，也就得以避开自然语言处理工具中的错误。

总的来说，PCNN 模型可以在无需人为构建训练特征的情况下进行远程监督的关系抽取，避免了使用自然语言处理工具进行预处理的过程，通过引入多实例学习流程解决标签错误的问题，同时改变传统模型中全局最大池化选用局部最大池化抽取各个卷积核的最优特征，通过这些改变，PCNN 模型与其他应用于关系抽取的深度学习模型相比表现出了更好的性能，因此本文选择使用 PCNN 模型完成关系抽取任务。

2.3.2 Tensorflow 深度学习框架

Tensorflow[50] 是由 Google 提供的深度学习框架，其前身为 Google 的 DistBelif V2，Tensorflow 的基本数据结构是 Tensor，Flow 代表 dataflow，即数据流，这也就表明 Tensorflow 基于数据流编程，作为一个端到端的开源平台，Tensorflow 提供了的各种工具、库和资源，利用这些工具开发者就可以有能力推动相关技术的发展、模型的改进，同时更为便捷的实现机器学习的相关模型、部署相关应用。且由于 Tensorflow 还支持使用 GPU 和 TPU 进行高性能的数值计算，因而常常被用来实现深度学习模型和模型训练。

Tensorflow 深度学习框架相比于 Pytorch 更适合大规模的部署，特别是当涉及到跨平台或者嵌入式部署时更能显示其性能，借助此框架的工具支持，本文选择使用 Tensorflow 实现 PCNN 模型。

2.4 本章小结

本章节主要介绍了系统应用到的相关技术，首先对系统实现所用到的相关工程框架和工具进行了介绍，包括前后端框架，数据库，以及用以在不同操作系统上部署的 Docker 容器和用作缓存机制以提升系统性能的 Redis 组件，其次对知识图谱应用到的相关技术进行了介绍，包括图数据库 Neo4j 和计算向量相似度的余弦相似度算法，然后介绍了实现本文所用到的深度学习算法以及对应的实现工具，通过介绍这些成熟技术的优势和选用理由来为后续系统实现做铺垫。

第三章　基于知识图谱的开源社区关联性自动化构建系统的需求分析与概要设计

本章节主要针对知识图谱的开源社区关联性自动化构建系统进行需求分析，设计出真实可行的功能性和非功能性需求，随后针对系统的功能性需求列出主要的用例描述情况表，在非功能性需求部分则主要列出系统需要达到了对应指标。采用 4+1 视图的形式介绍系统的顶层设计，之后将系统进行划分，通过主要模块的形式逐个讲每个解模的设计情况、核心类图等，以此来构成系统的总体设计阐述。

3.1　系统整体概述

根据前文的调研，在目前众多的知识图谱中，绝大多数现存的知识图谱依赖于其本身的知识库，且大都无法应对大规模非结构化数据的问题。为解决知识图谱构建过程依赖结构化数据的问题，本文将开源社区中的文本提取并进行数据清洗以构建自身所需要的知识库，同时引入外部知识图谱来补充所需要的知识。

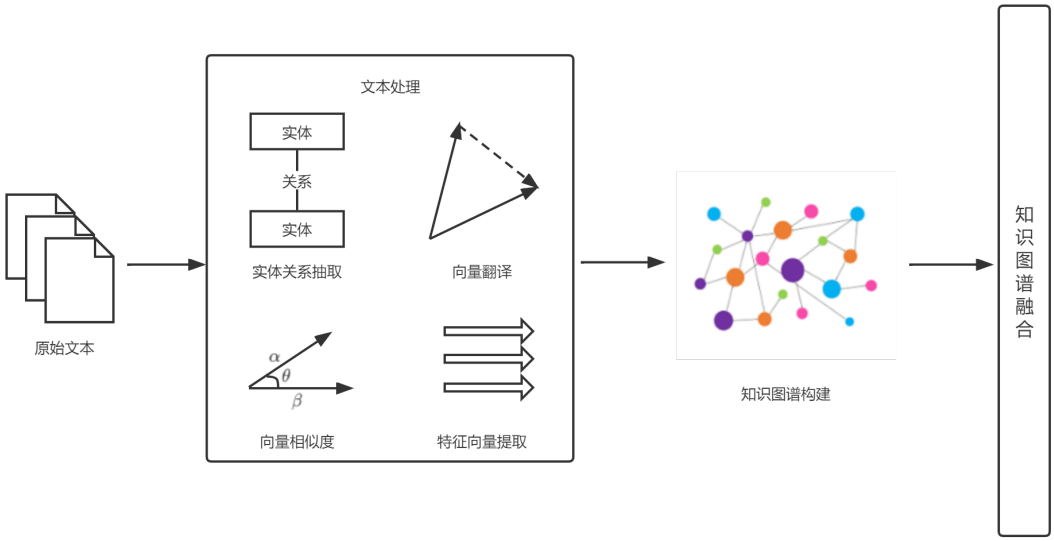

图 3.1: 系统整体流程图

如图3.1所示，系统会根据地址爬取对应的文本作为原始文本，交由 NLP 模块进行文本处理，主要是通过相应的模型抽取隐藏在文本中的实体和关系。NLP 技术将使用分词、词性分析和依存关系分析等来进行。在内容爬取时，选择根据对应帖子的标签和点赞数等结构化数据排序，以此来获取当下最为热门的数据进行知识库的构建、扩充，并尝试通过引入第三方分类知识图谱对现有的知识库进行补充，所得的知识存入 Neo4j 中。数据集选择使用远程监督方法进行构建，采用谷歌的深度学习框架 Tensorflow 实现 PCNN 模型并进行模型训练。

3.2 系统需求分析

3.2.1 功能性需求

本系统的功能性需求分析如表3.1所示。由于本系统是基于知识图谱的关联性自动化构建系统，当用户通过认证进入到系统中，可以总览系统的各个功能，包括用于知识库构建的文本识别功能，用于知识图谱生成的知识图谱构建和关联性挖掘功能，以及用于管理的知识图谱管理页面。

表 3.1: 系统功能性需求

需求 ID	需求名称	需求描述
R1	文本识别	用户可以借助系统发现文本中的实体。
R2	知识图谱构建	用户可以通过构建知识图谱发现直接关联的实体。
R3	关联性挖掘	用户可以以知识图谱的形式验证实体间的关联是否存在，或尝试挖掘实体间潜在的关联。
R4	知识图谱存储	用户可以将生成的知识图谱进行存储，以便在日后继续使用。
R5	知识图谱管理	用户对已经保存的知识图谱进行管理操作，管理主要包括对知识图谱的删除、导出。用户还可以从此页面查看每个保存的图谱的详情。
R6	知识图谱融合	用户可以选择保存的知识图谱进行融合操作，通过知识图谱融合扩充原有的知识图谱。
R7	知识图谱导出	用户可以将保存的知识图谱根据自己的需求导出成图片格式保存在本地。
R8	查看图谱详情	用户可以通过知识图谱管理页面进入保存的知识图谱详情页，在详情页可以查看构建后的知识图谱，对知识图谱的相关信息进行修改、发起融合请求以及导出到本地等操作。
R9	知识库构建	用户可以根据在开源社区搜索相应内容，并将链接交由系统自行对开源社区内容进行爬取、文本处理，处理后得到的相关实体和关系将存储到数据库中作为知识图谱构建的知识库。

在用户需要对知识图谱进行构建时，用户可以根据自身的需要选择知识图谱进行知识图谱构建功能、关联性挖掘功能进行构建。在知识图谱构建模块，通过搜索出与目标实体相关的实体，以及它们之间的关系，并将之展现出来。在关联性挖掘模块，其核心目的是将输入的两个实体之间所存在的各种关系、实体连接并展示出来，此外，还提供展示两个实体之间最短连通路径的功能，其目的在于通过显示出两个实体之间的最短路径来发掘连个实体间存在的联系，进而找到实体与实体之间存在的不易发觉的关联性。

3.2.2 非功能性需求

表 3.2: 系统非功能性需求

需求名称	需求描述
可用性	系统应该具有良好的可用性，需要及时备份关键数据以免丢失，当系统遇到进程崩溃、网络波动等意外情况时，系统能够及时重启并恢复服务提供。
可扩展性	系统应对系统关键功能功能提供高层次抽象，做好功能模块之间的解耦，以方便后续进行功能上的扩展，算法上的的改进，框架上的升级等。
易用性	系统的设计应当直观简洁、易于使用，采纳合理的人机交互设计理念，使得用户容易上手。
可移植性	本系统应当能够在当下主流的操作系统及国产主流操作系统上完成安装部署并正常运行。
性能需求	保证系统在生成图谱时给予动态反馈，在构建、融合等操作的响应上不超过 1s，图谱融合的可视化展示应在 5s 内。

表3.2所示的是系统需要实现的非功能性需求，考虑到系统需要保持较高的可用性，数据库的主从备份方法可以一定程度上避免数据的丢失从而保证数据库的稳定性，通过使用 Nginx 负载均衡保证了系统内资源的优化分配，提高了系统应对风险的程度，通过上述方法实现可用性。同时，对于系统中关键的功能模块做好了抽象，对系统模块之间进行了解耦，方便系统在之后的升级改造中进行功能上的升级和模块上的替换，保证了系统的可拓展性。因系统面向真正的企业用户，因此在系统的界面的设计上追求简单高效，人机交互尽可能设计合理，无论是新手还是熟练使用系统的人员都能够高效地使用该系统，此外为了适应当下不同操作系统的环境，系统应当具备良好的可移植性，即系统可以在不同的操作系统上如 Mac OS 和 Ubuntu 上安装使用。

根据上文中对系统功能性需求的分析，可以得到如图3.2所示的系统用例图，主要包含：文本识别、知识图谱构建、关联性挖掘、知识图谱管理、知识图谱融合、知识图谱导出、查看图谱详情和知识库构建共 9 个系统用例。

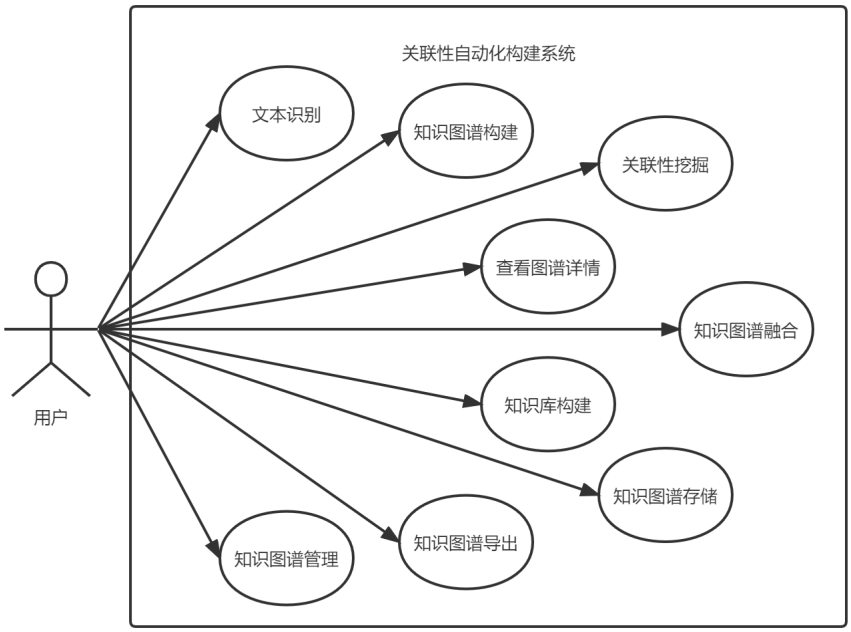

图 3.2: 系统用例图

3.2.3 用例描述

实体是知识图谱构建的一个重要部分，知识图谱作为语义网络，其构成需要依赖于每一个实体节点，为了能够向用户提供从文本中发现有用的实体的功能，在这里提供了文本识别这一功能。其用例描述如表3.3所示。

表 3.3: 文本识别用例描述

ID	UC1
名称	文本识别
参与者	用户，目的是发现原始文本中存在的实体
触发条件	用户想要进行文本识别
前置条件	用户进入文本识别页面
后置条件	无
优先级	中
正常流程	1. 用户进入到文本识别页面，在文本框中输入文本内容 2. 用户点击识别按钮 3. 系统进行实体识别并展示识别后的实体
特殊需求	识别后的实体将自动添加到知识库中

通过知识图谱构建功能，用户可以利用某一实体构建与之直接关联的实体的知识图谱，从而发现于目标实体有直接关联的实体从而帮助用户了解其关联性。其用例描述如表3.4所示。

表 3.4: 知识图谱构建用例描述

ID	UC2
名称	知识图谱构建
参与者	用户，目的是发掘与之相关的实体
触发条件	用户想要了解某一实体直接相关联的实体
前置条件	知识库中已经存在对应实体
后置条件	无
优先级	高
正常流程	1. 用户进入到知识图谱构建页面 2. 用户选择一个实体并点击构建“按钮” 3. 系统根据知识库中的实体进行选择 4. 系统生成以目标实体为中心的知识图谱 5. 系统展示知识图谱
异常流程	4.a 系统提示知识库中暂无该实体

关联性挖掘功能是本系统的针对实体间关联的另一个重要功能，其本质是通过在知识库中递归查找与目标实体有关联的实体及关系。关联性挖掘的用例描述如表3.5所示。

表 3.5: 关联性挖掘用例描述

ID	UC3
名称	关联性挖掘
参与者	用户，目的是发掘实体间的关联性
触发条件	用户想要了解实体间的关联情况
前置条件	知识库中已经存在对应实体
后置条件	无
优先级	高
正常流程	1. 用户进入到关联性挖掘界面 2. 用户输入需要进行关联性挖掘的内容 2.1 用户指定实体 1 或实体 2 2.2 用户指定实体 1 和关系或关系和实体 2 2.3 用户指定实体 1 和实体 2 2.4 用户指定实体 1 和实体 2 和关系 3. 用户点击“挖掘”按钮 4. 系统根据用户指定内容，查找知识库，并将之展现出来 4.1 系统展示与目标实体直接相关的实体及关系 4.2 系统展示与目标实体和关系相符合的实体 4.3 系统展示两个实体之间是否存在联通的最短路径 4.4 系统判断并展示是否存在对应关系 5. 系统展示知识图谱
异常流程	4.1a 系统提示知识库中暂无该实体

关联性挖掘功能与知识图谱构建不完全相同，知识图谱构建的主要目的在于展现目标实体有哪些直接相关联的实体，该功能更加注重于将以用户的需求验证实体间的关联性。该功能触发后，根据用户输入的不同，系统会生成用户需要的知识图谱并展示。在仅输入两个实体的情况下，系统会尝试在实体间构建最短路径，用户可以借此发现实体间的潜在关联。

表 3.6: 知识图谱存储用例描述

ID	UC4
名称	知识图谱存储
参与者	用户，目的是保存已经生成的知识图谱
触发条件	用户需要储存有关图谱
前置条件	系统已经生成相关知识图谱
后置条件	无
优先级	中
正常流程	1. 用户进入到系统并发起生成图谱相关命令 2. 系统完成对应知识图谱的构建 3. 用户选择知识图谱存储 4. 系统完成对应存储

知识图谱的存储是一个相对常见的功能，当用户利用知识图谱构建或关联性挖掘功能生成知识图谱之后，对于已经生成的知识图谱，用户可以选择是否将之存储到数据库中，并对之进行相应的备注操作，以便后续进行如导出、融合等操作。其用例描述如表3.6所示。

表 3.7: 知识图谱管理用例描述

ID	UC5
名称	知识图谱管理
参与者	用户，目的是对已经存储的知识图谱进行管理
触发条件	用户处在知识图谱管理界面
前置条件	用户已经被授权
后置条件	无
优先级	高
正常流程	1. 用户进入知识图谱管理页面 2. 用户查看已存储的知识图谱 3. 系统展示知识图谱 4. 用户选择多个知识图谱并点击知识图谱融合 5. 系统对选择的知识图谱进行知识融合并生成融合后的图谱 6. 用户选择导出知识图谱 7. 系统将对应图谱生成文件导出 8. 用户选择删除知识图谱 9. 系统删除指定知识图谱并提示删除成功

由于本系统主要功能是对开源社区的一个知识图谱生成，知识图谱的存储与管理则是一个不可或缺的功能，对于已经生成并存储在系统中的知识图谱，用户应当可以在系统中查看并进行相关的管理操作。系统将保存的图谱以详细列表的形式展示出来，每个存储的知识图谱包含对应的生成日期，备注等信息，且在管理页面可以进行查看、知识图谱融合、导出、删除及修改信息等操作。其用例描述如表3.7所示。

表 3.8: 知识图谱融合用例描述

ID	UC6
名称	知识图谱融合
参与者	用户，目的是扩展已有的知识图谱
触发条件	用户需要进行知识图谱融合操作
前置条件	用户处在知识图谱管理界面
后置条件	无
优先级	高
正常流程	1. 用户选择多个知识图谱并发起融合请求 2. 系统根据实体之间的关联性进行知识融合操作 3. 系统将融合后的知识图谱展示出来 4. 用户查看新的知识图谱

尽管已经存在关联性挖掘功能，但多次的重复操作会让操作变得繁琐、复杂，在这里，用户可以在知识图谱管理界面可以选择将多个知识图谱进行实体之间的融合，通过将多个图谱内的实体和关系合并到一个图谱中去，尽可能让每个节点都进行一次关联性挖掘操作，通过这样多次的查找实体之间是否存在连通路径来将复数个图谱中的所有节点连接起来，扩展已有的知识图谱且更为直观的展现实体间潜在关联性。其用例描述如表3.8所示。

表 3.9: 知识图谱融合用例描述

ID	UC7
名称	知识图谱导出
参与者	用户，目的是将已经生成的图谱导出到系统
触发条件	用户需要进行知识图谱导出操作
前置条件	用户处在知识图谱管理界面
后置条件	无
优先级	中
正常流程	1. 用户选择知识图谱并发起导出请求 2. 用户选择导出的图片格式并命名 3. 系统将知识图谱以用户期望的方式导出 4. 用户选择保存地址，借助浏览器保存图谱到本地

对于已经生成的知识图谱，用户还可以选择以图片的形式将之保存到系统中去，系统应当可以向用户提供知识图谱导出的功能，且可以满足用户对于图片格式的选择，最后通过浏览器下载将文件保存到用户期望的位置。其用例描述如表3.9所示。

表 3.10: 查看图谱详情用例描述

ID	UC8
名称	查看图谱详情
参与者	用户，目的查看并修改已经保存的知识图谱
触发条件	用户需要查看已保存的知识图谱
前置条件	用户处在知识图谱管理界面
后置条件	无
优先级	中
正常流程	1. 用户选择已保存的知识图谱并点击“查看”按钮 2. 系统跳转到对应知识图谱详情页面 3. 用户在知识图谱详情页面对图谱相关信息进行修改 4. 系统提示修改成功

查看图谱详情的用例描述如表3.10所示，用户可以通过知识图谱管理界面查看已保存的知识图谱的详细信息，在知识图谱的详情界面还可以对图谱的可编辑内容进行修改，如给图谱添加备注。

表 3.11: 知识库构建用例描述

ID	UC9
名称	知识库构建
参与者	用户，目的是扩充知识库
触发条件	用户需要为知识库添加新的内容
前置条件	用户在开源社区找到相应内容
后置条件	无
优先级	高
正常流程	1. 用户进入到知识库构建页面 2. 用户在开源社区查找对应资料，复制 URL 到选项框中 3. 系统根据 URL 对用户指定内容进行爬取 4. 系统对爬取到的内容进行实体和关系抽取将之保存到数据库中 5. 用户发起停止构建请求 6. 系统停止对页面的爬取但会持续处理已爬取到的文本内容

知识库构建的主要目的是帮助用户高效利用开源社区的各种信息资料，例如用户可以在 Stack Overflow 搜索 Tensorflow，并将对应 URL 复制到页面对应的选项框中，系统会自动爬取每个讨论帖下的内容并对其中的文本信息进行实

体和关系抽取，用户可以通过发起终止爬取请求终止对文本内容的爬取，系统在完成实体和关系抽取后会直接将内容存储到数据库中用以扩充已有的知识库。其用例描述如表3.11所示。

3.3 总体设计

3.3.1 总体架构设计

基于知识图谱的开源社区关联性自动化构建系统是的 Web 服务平台，其中，本系统主要基于 Spring Boot 和 Vue 进行前后端编写，此外，知识图谱部分单独使用 Python 用语言设计，Neo4j 用来存储知识图谱的三元组结构，使用 Thrift 进行服务的远程调用。系统架构如图3.3所示。

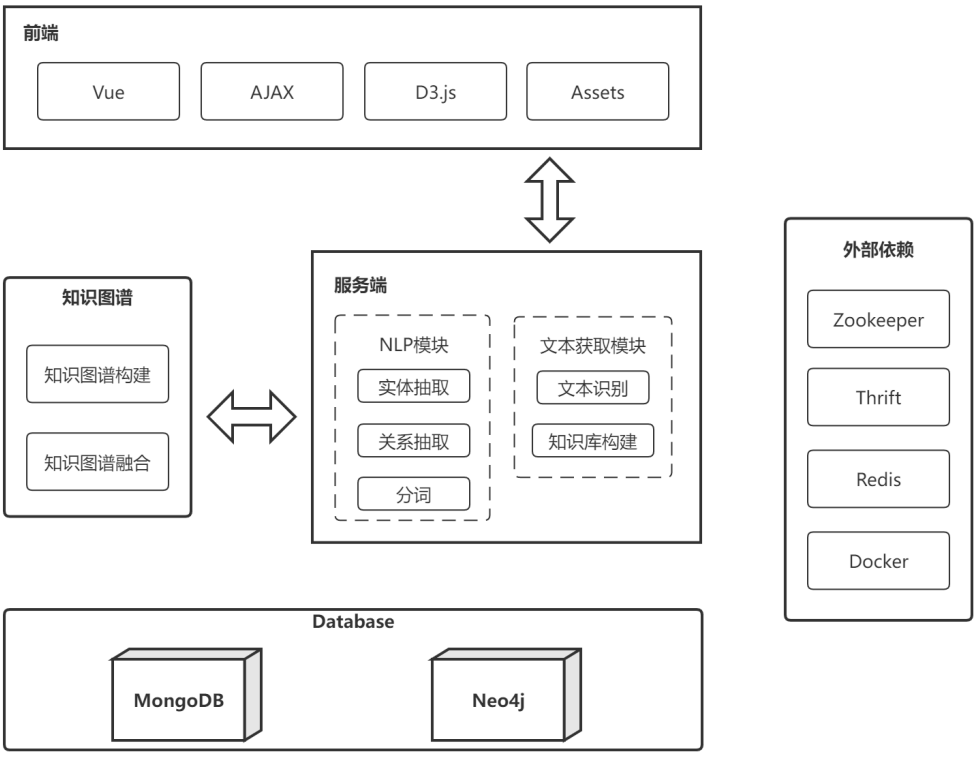

图 3.3: 系统架构图

在本系统的前端部分，因 Vue 具有具有组件化开发以及高效渲染的特点，且简洁易于使用，因而这里选择使用 Vue 框架，采用 jQuery 库进行事件处理，使用 jNotify 插件实现无阻塞的消息通知。此外，系统中的知识图谱部分的图将使

用 D3.js 来绘制，同时，在前后端的交互方面选择使用 Ajax 来进行异步通信，目的是提升系统的性能。

在系统的后端部分主要选择使用 Spring Boot 框架，该框架省去了原本 Spring 框架复杂的 XML 文件编写流程，同时由于国内外开发者对该框架的不断改进，使得该框架有着良好的性能，因而在使用 Spring Boot 框架进行开发时，开发者无需关注太多的依赖关系而可以更多关注到项目的逻辑本身。后端采用经典的 MVC 模式来组织项目代码，其中，Controller 层接收前端的请求并转发到 Service 层；Service 层处理业务逻辑，将相关操作转发到 DAO 层处理，并由 DAO 层来直接与数据库交互。由于系统需要与文本获取模块及知识图谱模块进行交互，这里选择使用 Thrift 来进行服务的远程调用，为使能够及时修改系统配置，这里使用 Zookeeper 来管理系统的配置。在数据的持久化方面，本系统中主要使用非关系型数据库 MongoDB 来存储数据，并用图数据库 Neo4j 来对知识图谱部分产生的的数据进行存储。知识图谱部分主要使用 Python 语言进行编写，并将之作为单独的服务通过 Thrift 来与主系统进行交互。

最后，考虑到系统的高可用性方面，本系统选择使用 Redis 作为系统的缓存，Redis 作为由 C 语言编写的一个高效的、基于内存的缓存组件，具有更好的性能和更强的拓展性，可以用于保证系统后续的可拓展性并提高性能。

3.3.2 模块划分

根据软件设计中的高内聚、低耦合的软件设计原则，本系统大致可以分成为知识图谱模块、文本获取模块、NLP 模块 3 大模块，如图3.3所示。在知识图谱模块可以进一步细分为知识图谱构建和知识图谱融合两部分，在完成对原始文本进行实体和关系抽取的知识库构建后，知识图谱构建模块则是将知识库中存储的各项数据以图的形式构建成知识图谱并表现出来，生成的图数据存储在 Neo4j 图数据库中；知识图谱融合部分则是将已经生成的知识图谱进行融合，将现有的实体之间通过发掘潜在关联进行连接从而将知识图谱扩充。知识图谱模块选择使用 Python 语言进行开发，并使用 Thrift 框架来跟服务端进行通信。文本获取模块主要用于到开源社区根据目标实体爬取相关数据。在 NLP 模块中，使用 HanLP 和 Keras 进行分词、词性标注、命名实体识别，并通过 KNN 进行实体分类，使用远程监督方法构建数据集，利用 tensorflow 训练 PCNN 模型，最终使得到的数据得以用于知识图谱模块。

3.3.3 总体设计

通常，人们会用软件架构详细阐述软件结构的设计，但仅仅使用一种图往往难以完整进行阐述，“4+1” 视图因其多角度阐述的特点而为广大程序开发人

员所采用。所以本文同样采用“4+1”试图，根据给出的系统总体架构设计，辅之以逻辑视图、进程视图、开发视图和物理视图五个方面进行详细描述。

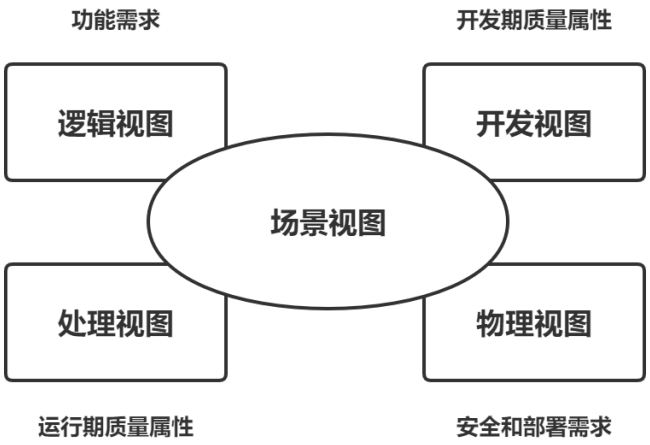

图 3.4: 4+1 视图结构图

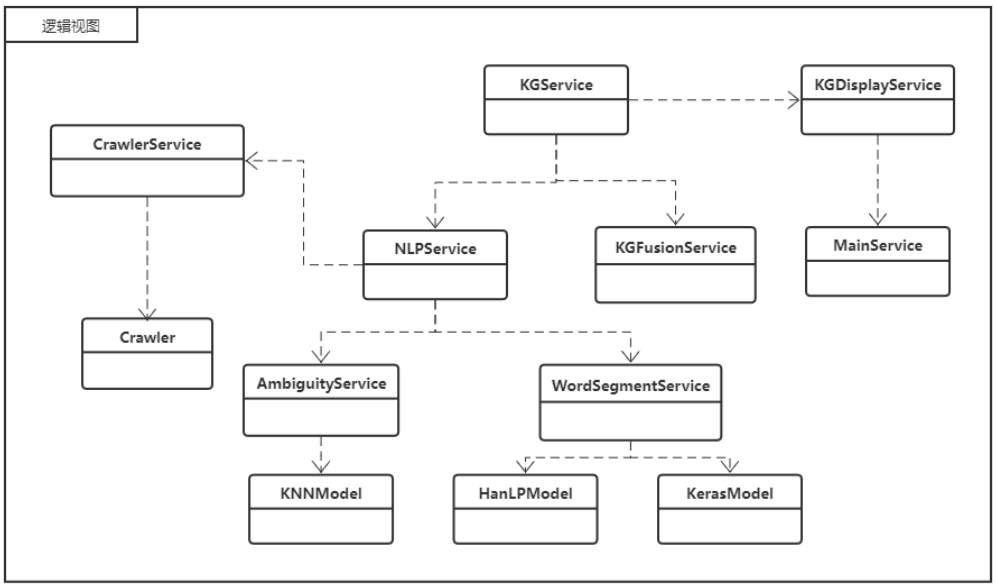

图 3.5: 逻辑视图

逻辑视图 图3.5所展示的是系统的逻辑视图，逻辑视图面向的是系统的用户，通过使用 UML 建模中的类图对系统的依赖关系进行分析。

系统的核心类是 KGService 即知识图谱模块，其与多个模块进行交互来完成系统中的功能。KGService 本身负责知识图谱的构建，包括实体识别，关系抽取，关键词发掘，关联性定义等，并将相关数据存储至 Neo4j 数据库。Crawler 用于爬取开源社区中的相关内容，NLPService 部分将会对爬取的内容进行数据清洗，其中 Ambiguity 模块用于对文本进行同义词替换增强文本效果，KNNModel 使用 KNN 算法来进行实体分类，WordSegmentService 主要用于对文本进行分词和词性标注工作，其中 HanLP 用于中文文本的分词工作，Keras 用于英文文本的分词工作，并将相关数据存储到 MongoDB 和 MySQL 数据库中。KGFusionService 模块用以将知识图谱进行融合，从而扩展已经得到的知识图谱，利用已经得到的相关知识，并通过外部知识图谱辅助发掘实体间的潜在联系。KGDisplayService 用以将知识图谱展现出来，这里会通过 Thrift 接口调用知识图谱模块获得相关数据，用 MainService 来管理知识图谱的相关服务，并提供知识图谱下载的相关功能。

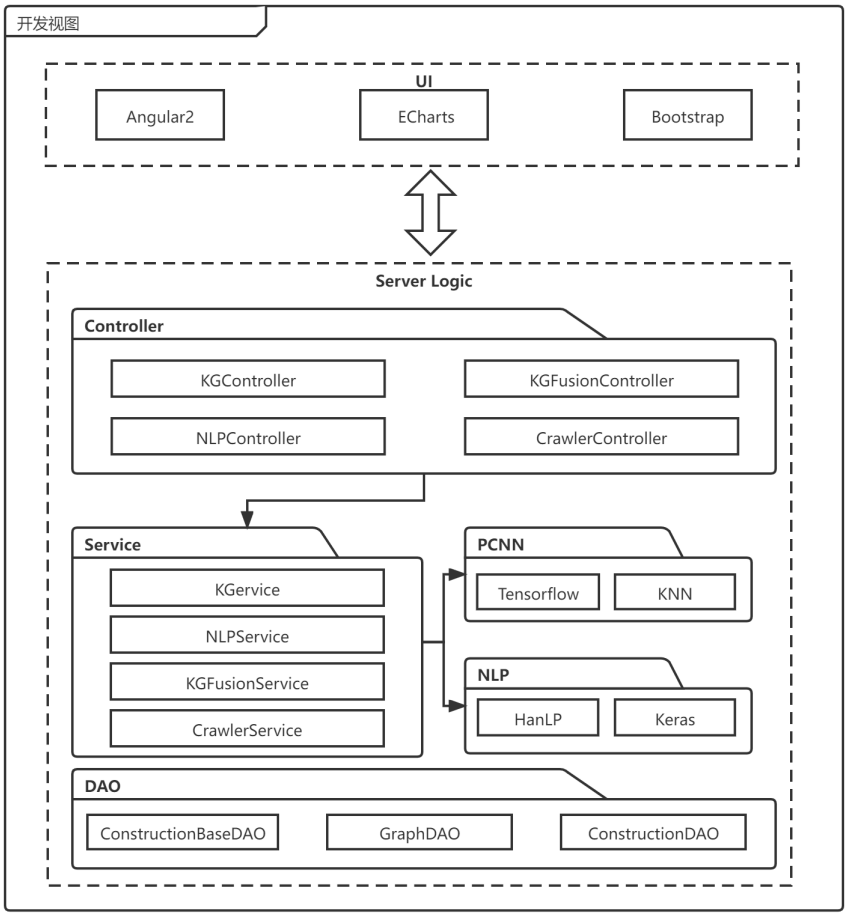

图 3.6: 开发视图

开发视图 开发视图面向的是系统的开发人员，以软件开发人员的角度描述系统的模块。系统开发主要由 UI 和 Service Logic 两部分构成，如图3.6所示。

在 UI 部分选择使用 Vue 框架进行前端的单页应用开发，使用 Echarts 进行构建前端图表、展示，并通过 Bootstrap 提供的开源工具包进行辅助开发。在 Service Logic 部分则可以继续分为 Controller、Service 和 DAO 三部分，其中，Controller 部分提供对外的的 HTTP 调用，主要包括 KGController、KGFusionController、NLP-Controller 和 CrawlerController 这四个部分；另一边，Service 部分则将实现系统逻辑的代码进行了封装，其中 KGService 主要提供知识图谱的相关服务，例如查询实体间的连通性，其利用 NLPService 和 CrawlerService 从开源社区中获取可以用于知识图谱构建的结构化数据，KGFusitonService 则主要用于知识图谱的融合；PCNN 模块利用 KNN 算法计算实体间的相似性并以此为依据进行实体类划分，同时使用使用远程监督方法构建数据集，用 Tensorflow 训练 PCNN 模型；NLP 模块主要使用 HanLP 和 Keras 进行分词和词性标注等工作；DAO 部分主要用于实体类和数据库的相关操作，与数据库及云服务进行交互。

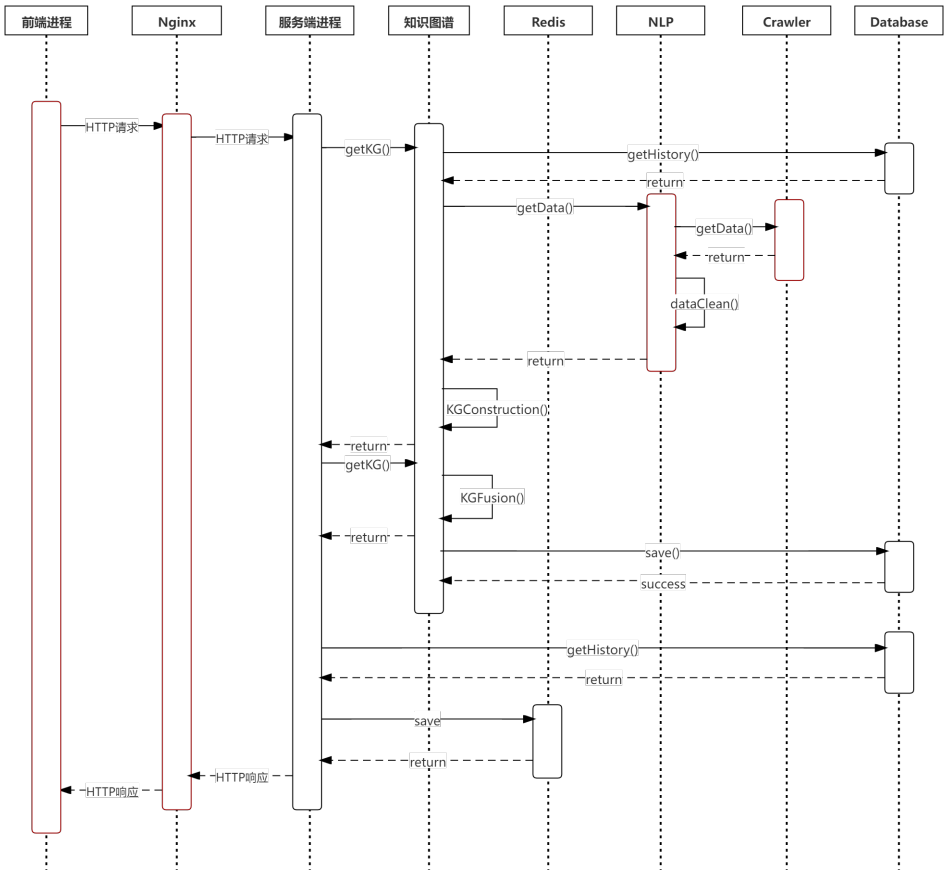

图 3.7: 处理视图

处理视图 处理视图即进程视图，其从系统运行时的主要进程和进程之间的交互来描述系统，本系统的处理试图如图3.7所示。

当系统运行时，用户在前端进行操作并触发前端方法，前端发送 HTTP 请求到 Nginx 服务器上，Nginx 进行请求解析，通过负载均衡算法将请求分发至对应的进程，当服务端发起获取知识图谱时，服务端采用 RESTful 请求调用知识图谱服务，知识图谱模块会查找数据库，如果不存在需要的实体及关系则由知识图谱进程调用 NLP 模块进程获取数据，NLP 模块进程在调用 Crawler 进程爬取数据后进行奇异消除、同义词替换、数据结构化等数据清洗操作后将数据返回给知识图谱进程，再由知识图谱进程进行知识图谱构，构建完成后会存储到数据库中，并将图谱返回给服务端进程。服务端发起知识图谱融合指令后，由知识图谱进程进行图谱融合，在图谱构建完成后会将图谱存储到到数据库中同时也会返回给服务端进程。对于热点数据，如已经生成的知识图谱或实体关联路径，可以先从 Redis 中查找，如果 Redis 中没有再去访问数据库。服务端完成业务请求的处理之后，会将得到的数据提交给给 Nginx 服务器，并由 Nginx 将数据提交给前端，前端得到数据后数据通过解析进行展示。

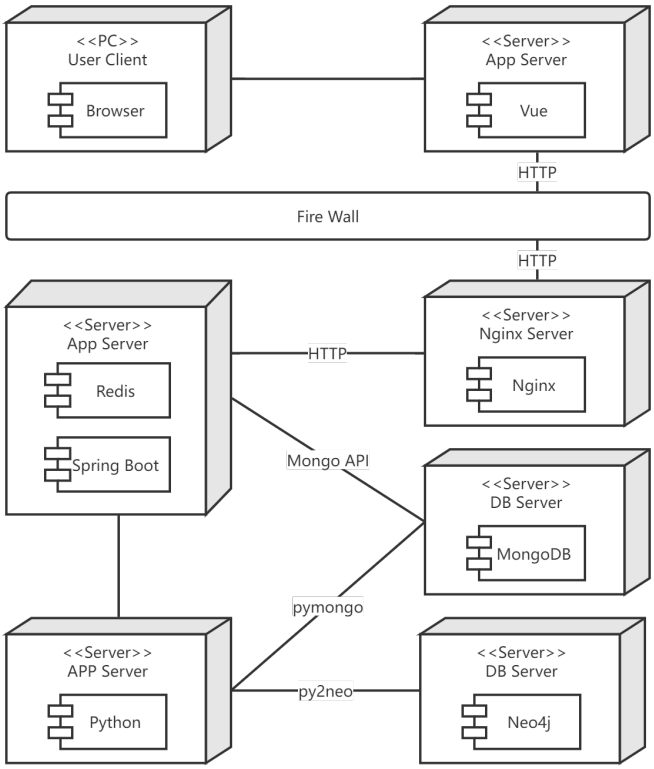

图 3.8: 物理视图

物理视图 物理视图将会帮助运维人员解决系统的安装与部署问题，并解释各个节点间的通讯关系。本系统的物理视图如图3.8所示。

首先，用户的请求会以 HTTP 请求的形式通过浏览器访问 WEB 服务器，经过防火墙 (Fire Wall) 的过滤后通过 Nginx 服务器进行任务分发，Nginx 主要用于负载均衡功能的实现。在服务端，App Sever 接收到来自浏览器的 HTTP 请求后进行逻辑处理，并将系统产生的缓存数据存储在 Redis 中。数据库服务作为单独的服务建立在两个 DB Server 上，分别对应 MongoDB 和 Neo4j 数据库，使用基于 TCP 的数据库操作查询语言进行数据相互通讯。另一方面 Python 作为爬虫模块用以获取开源社区的文本部署在单独的 APP Server 上。

3.4 文本获取模块详细设计

知识图谱的生成，实体关系的查询依赖于大规模的语料库，自然而然地，对于原始数据的获取自然也是重要的一环，通常所见的爬虫 (如 urllib2 等) 对于一个 Web 网页的爬取效率是有限的，若想要要构建一个规模较大的语料库用以知识图谱的生成，其完成所需要的时间可能相当长，因此本文考虑使用 Scrapy 框架来进行原始数据的获取。

3.4.1 流程设计

文本获取的流程大致如图3.9所示，其主要由 Scrapy 框架所带来的支持多线程爬虫、性能稳定，且支持爬虫的断点保存的特点来提高文本爬取的效率，从而更好地构建知识库。

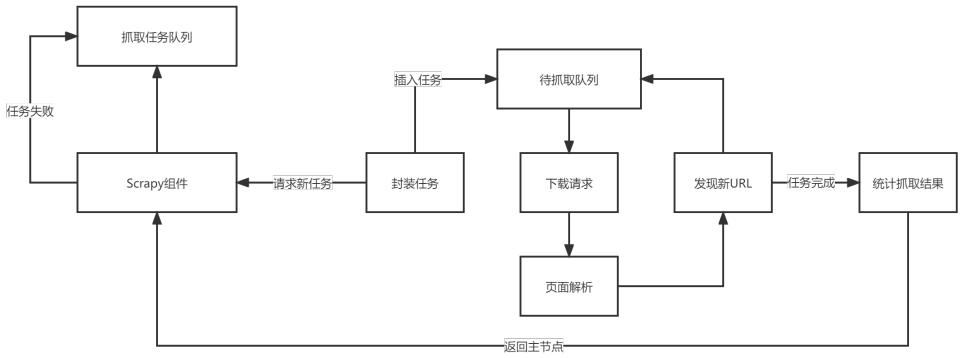

图 3.9: 文本获取流程图

文本获取模块主要由 Scrapy 组件来进行任务分配，其中包括抓取任务队列和待抓取队列两个队列，当抓取任务队列中存在抓取任务并且抓取已经启动时，

则由 Scrapy 组件从抓取任务队列中取得任务并将之封装，如若任务失败，则同样由 Scrapy 组件将任务返回给抓取任务队列并向系统报告任务失败；如若任务成功则将任务提交到待抓取队列，并发起下载请求，通过页面解析来确定是否有新的抓取任务出现，在完成任务后将抓取结果返回给 Scrapy 组件，并由 Scrapy 组件将爬取结果提交给系统。

3.4.2 核心类设计

文本获取模块的核心类图如图3.10所示，主要利用 Scrapy 组件针对不同的开源社区进行内容获取，在这里主要针对 GitHub 社区和 Stack Overflow 社区。

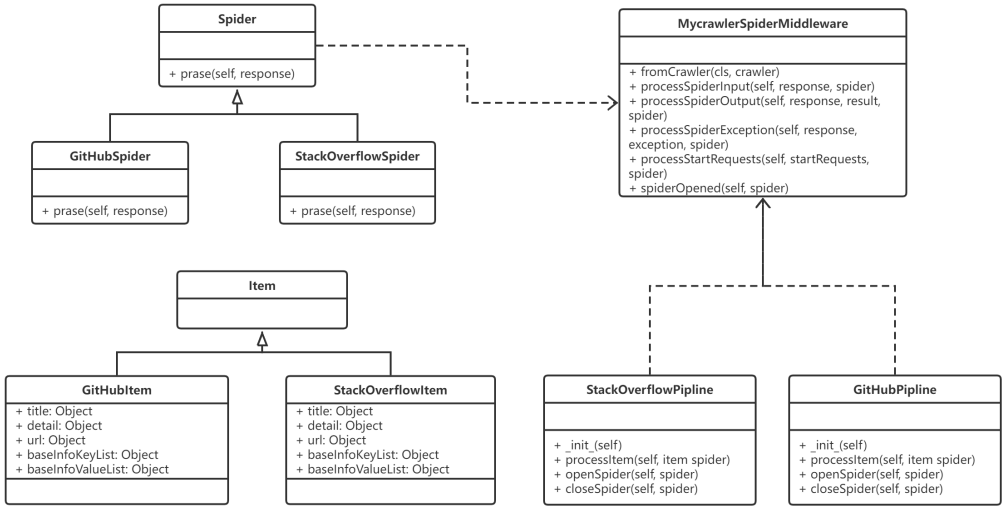

图 3.10: 文本获取核心类图

其中，Spider 类提供爬虫的基本功能，并由 GitHubSpider 和 StackOverflowSpider 两个类继承后具体实现相关的功能，Item 类将挖各个项目，即从对应社区的主页获取到各个项目的超链接，逐层进入，最后到达到项目的主页爬取项目内容，这里与 Spider 类相似，由 GitHubItem 和 StackOverflowItem 两个类继承后具体实现，两个 Pipline 则是主要通过启动进程去获取项目中的文本内容；这些类共同依赖 Scrapy 组件提供的 Spider 中间件即 MycrawlerSpiderMiddleware 来进行任务分配与执行。

3.5 NLP 模块详细设计

知识图谱的构建过程在前文已经提及，这里可以简单的将知识图谱的构建划分为两个流程：1. 实体抽取，2. 关系抽取。需要注意的是，知识图谱的构建有

必要要在一开始就明确好构建的目的，也就是需要以业务为导向，根据业务需求进行实体抽取和关系抽取，最后将抽取得到的知识构建成一张网络。

3.5.1 基于 BLSTM-CNNs-CRF 的实体抽取技术

在实体抽取部分，当下最为常用的算法是用 BLSTM+CNNs+CRF 联合使用进行实体识别，其网络结构如图3.11所示。

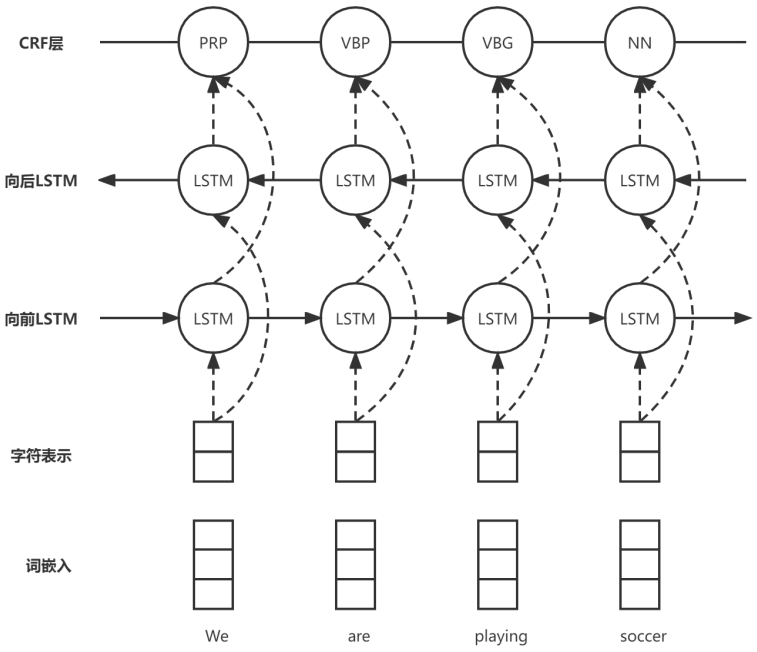

图 3.11: BLSTM-CNNs-CRF 网络结构示意图

其中，CNN 可以有效抽取每个单词中如单词的前缀、后缀等字符特征，从而形成字符级别的表示特征；一方面，通过使用 LSTM 可以一定程度上缓解梯度消失、梯度放大的问题，从而捕获远程依赖关系，另一方面在 LSTM 的基础上进一步采用双向 LSTM 则可以进一步充分利用单词的上下文信息，并捕获句子的演化信息；CRF 的引入则可以计算出序列隐变量间的约束关系。

通过 CNN 的模型训练可以得到字符级别的表示，得到表示后将这些字符级别的 embedding 和词汇级别的 embedding 结合并作为参数输入到 BLSTM 中，经过 BLSTM 的计算，将得到的向量输入到 CRF 中，通过联合解码得到最优序列标注。

3.5.2 关系抽取

关系抽取是知识图谱生成的关键捕捉，其主要针对以关键词为核心的文本。关系提取主要使用了 NLP 的相关技术，对于开源社区的文本内容首先需要经过同义词替换，由于自然语言中同义词的存在对文本抽取的影响，通过替换同义词能够显著减少差异化描带来的影响。例如句子“知识图谱关系提取”和“知识图谱关系抽取”仅仅只是用词的不同，其文本描述的意思是几乎完全相同的，同义词替换就是将那些语义相近的词语进行替换，从而保持文本描的述风格一致性，达到文本增强的效果，例如在上文中“提取”即可替换为“抽取”，从而使文本都变成“知识图谱关系抽取”，达到增强文本的目的。

在将文本通过同义词替换操作之后，接下来需要做的是对句子的描述内容进行分词，本文选择使用 HanLP 分词模型。分词，顾名思义即是把句子拆分成多个词语，分词是自然语言处理分析的重要初始步骤，句子在被拆分成单个词语后继而会进行词性标注，也就是判断句子中该词语的词性，确定词语的词性后将之标注，例如“系统提示栈内存溢出”这句话经过分词模型的分词后，会划分为“系统”、“提示”、“栈内存”、“溢出”4 个词，词性标注中会将这 4 个词分别标记为名词、动词、名词、动词。经过分词和词性分析后进行依存句法分析，仍然以“系统提示栈内存溢出”这句话为例，对此句生成对应的依存句法分析树，可以提取出该句的主语为“系统”，宾语为“内存溢出”，谓语是“提示”，“栈”作为状语修饰“内存溢出”。

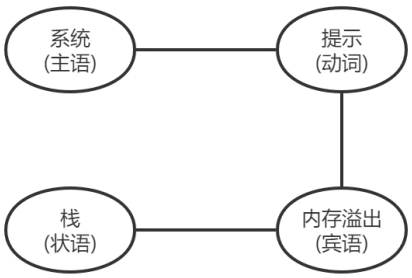

图 3.12: 依存句法树

3.5.3 基于 Piece-Wise-CNN 的关系抽取技术

在关系抽取部分，比较常用的算法是：Piece-Wise-CNN 算法和 Attention 和 LSTM 联合使用的模型。

如 1.2 节中所介绍，卷积神经网络 (Convolutional Neural Networks，CNN) 模型被广泛应用于自然语言处理，而在 CNN 模型的基础上改进的 PCNN 模型通过

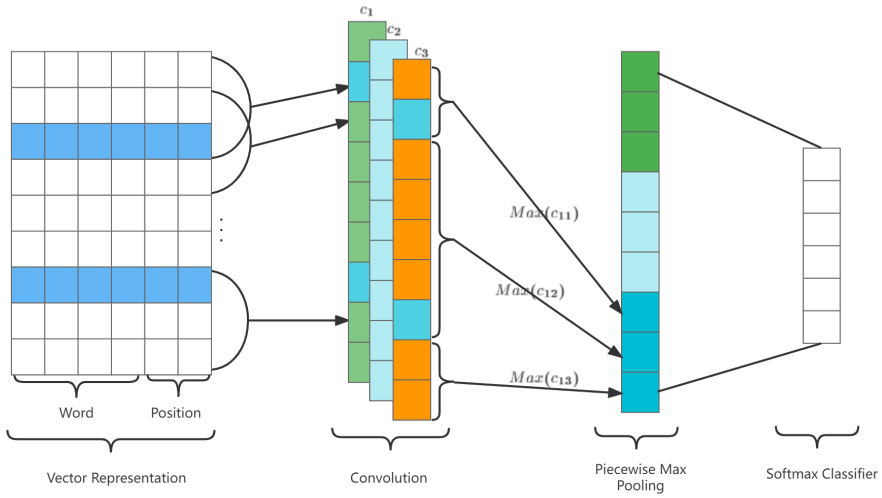

图 3.13: Piece-Wise-CNN 网络结构示意图

修改池化方式得到业界的广泛认可，被视为关系抽取效果和反馈较好的抽取模型之一。PCNN 的运行流程大致如下：在经历过实体识别确认实体位置后，模型将会以实体的位置为中心，按句中各词离实体的距离重新编码，并将句子以实体边界进行切分，通过将位置特征以及文本特征进行拼接，并将先前所切分好的句子分别利用 CNN 模型计算其特征，把提取出来的特征交由 max pooling 层进行拼接，然后交由 softmax 层，最终就可以得到关系的分类。PCNN 模型的主要构成如图3.13所示。

第一部分为向量表达层 (Vector representation)，用词向量和向量的位置嵌入信息组成句子向量，用句子中每个单词到目标词间的绝对距离来表示位置嵌入的相对距离，并用 Skip-gram 算法训练句子向量。在这里可以将句子向量表示为 $S = \mathbb{R}^{s \times d}$，其中，s 表示句子中单词的个数，$d = d_w + d_p \times 2$。

第二部分为卷积层 (Convolution)，在这里将长度为 s 的句子进行首尾 w-1 的长度填充，在设置 n 个卷积核时，卷积的输出为:

$$C = \{c_1, .., c_n\}, c_{ij} = w_i q_{j-w+1}, 1 \leq i \leq n \tag{3.1}$$

第三部分为局部最大池化层 (Piecewise max pooling)，在经历过卷积层的输出后，为了方便接下来的任务，需要将卷积层的输出通过池化操作来使之独立于序列的长度而非依赖句子的长度。局部最大池化是 PCNN 模型的一大亮点，经

由两个实体将局部最大池化输出的向量长度分为 3 段，即 $\{v_{i1}, v_{i2}, v_{i3}\}$，然后依次对 $\{v_{i1}, v_{i2}, v_{i3}\}$ 分别进行最大池化，计算每个卷积核的最优特征:

$$V_{ij} = max(c_{ij}), 1 \leq i \leq n, j \leq j \leq n \tag{3.2}$$

经过池化后，C 的每行向量会变成长度为 3 的向量 V，接下来将这些向量进行拼接，使用非线性的函数 *tanh* 进行非线性变换，则可以得到 $g = tanh(v_i)$，其中 $g \in \mathbb{R}^{3n}$。

第四部分是 Softmax 层，在这一层，需要将得到的 g 输入到 Softmax 分类器中进行转换:

$$o = W_1 g + b, 1 \leq i \leq n, j \leq j \leq n \tag{3.3}$$

其中，$W_1 \in \mathbb{R}^{n1 \times 3n}$ 为权重矩阵，$o \in \mathbb{R}^{n1}$ 作为最终输出。

PCNN 模型使用半监督的多实例学习，若设置 T 个包 $\{M_1, ..., M_T\}$ 作为训练集，则其中每个 $M_i = \{m_i^1, ..., m_i^{q_i}\}$，且包中的 q_i 是相互独立的不同实体，那么对于神经网络 o 中第 r 个关系对应的概率为:

$$p(r|m_i^j; \theta) = \frac{e^{O_r}}{\sum_{k=1}^{n1} e^{o^k}} \tag{3.4}$$

另一方面，为了降低包中数据标注错误带来的影响，在每个包的标签已知，实力标签未知的情况下，训练过程将采取包标签的最大概率作为输出，其目标函数为:

$$J(\theta) = \sum_{i=1}^{T} log\ p(y_i|m_i^j; \theta) \tag{3.5}$$

$$J^* = arg\ \max p(y_i|m_i^j; \theta) 1 \leq j \leq q_i \tag{3.6}$$

综上所述，PCNN 多实例学习流程如2所示。

Algorithm 2: PCNN 多实例学习流程

1 初始化. 设定标准包 b_s，并以此大小划进行包的切分
2 随机选取切分后的小批量包，并逐个送入网络;
3 根据公式3.6，在每个包中寻找第 j 个实例 $m_i^j(1 \le i \le b_s)$;
4 根据算法 Adadelta，基于 $m_i^j(1 \le i \le b_s)$ 的梯度更新参数θ.
5 重复步骤 2-4，直至收敛或训练到达最大次数;

3.6 知识图谱模块详细设计

图3.14所展示的是本系统中知识图谱模块的架构设计，知识图谱模块的输入内容是爬虫到对应开源社区根据关键词爬取的相关文本，该模块的主要目的是通过处理原始文本，获取实体间关系，从而利用知识图谱清晰明确的特点来发掘实体间存在的联系进而找到潜在的关联。本系统将以 GitHub 和 Stack Overflow 为例爬取对应内容，并将之添加到知识库中，在知识图谱生成时直接调取数据库中存在的实体。

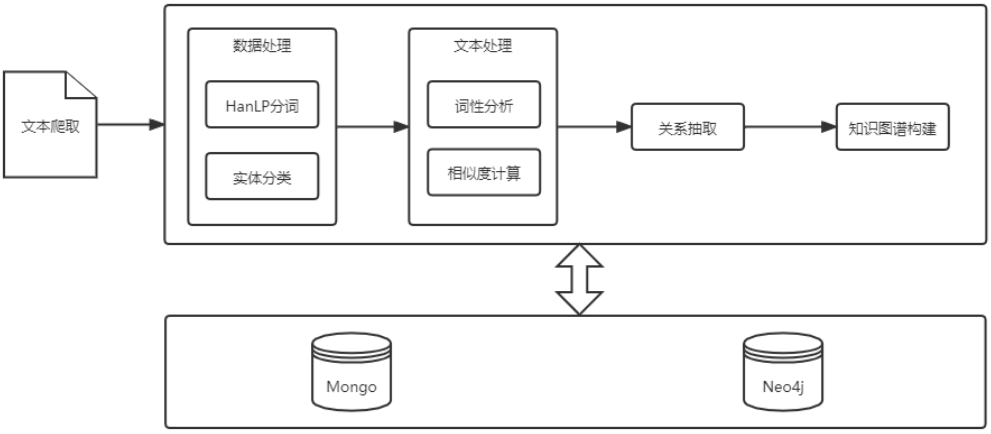

图 3.14: 知识图谱架构

3.6.1 详细设计

知识图谱模块选择使用 Python 语言编写，并使用 Thrift 进行来与主服务进行通信，本模块的核心类图如图3.15所示。

在本模块中，KGService 作为核心，负责交互各个模块，主要提供知识图谱构建和知识图谱融合两个功能，并由其两个子类 KGConstruct 类和 KGFusion 类具体实现，KGConstruct 类负责知识图谱的构建，其可以调用 NLPService 进行数据清洗获得实体和关系，也可也直接从数据库中获取，KGFusion 则直接从生

成的知识图谱中获取实体和关系，并根据数据库中现有的对两个或多个知识图谱进行扩充融合工作；NLPService 主要功能是对获取到的文本进行自然语言处理的相关操作，该模块提供了中英文分词、同义词替换、依存关系分析等功能，主要目的是将得到的数据进行清洗，使之变成结构化数据，在此基础上可以通过模型训练进行关系和实体的抽取，抽取到知识存将直接储到对应的数据库中，以便后期直接使用，由对应的 EntityDao 和 RelationDao 负责连接数据库并处理对应的数据；CrawlerService 则是在构建知识图谱时，知识库中没有对应的实体和关系，需要进行文本获取时由 NLPService 发起调用，并从对应的开源社区获取原始文本后交由 NLPService 进行数据清洗，主要起到构建知识库的作用。

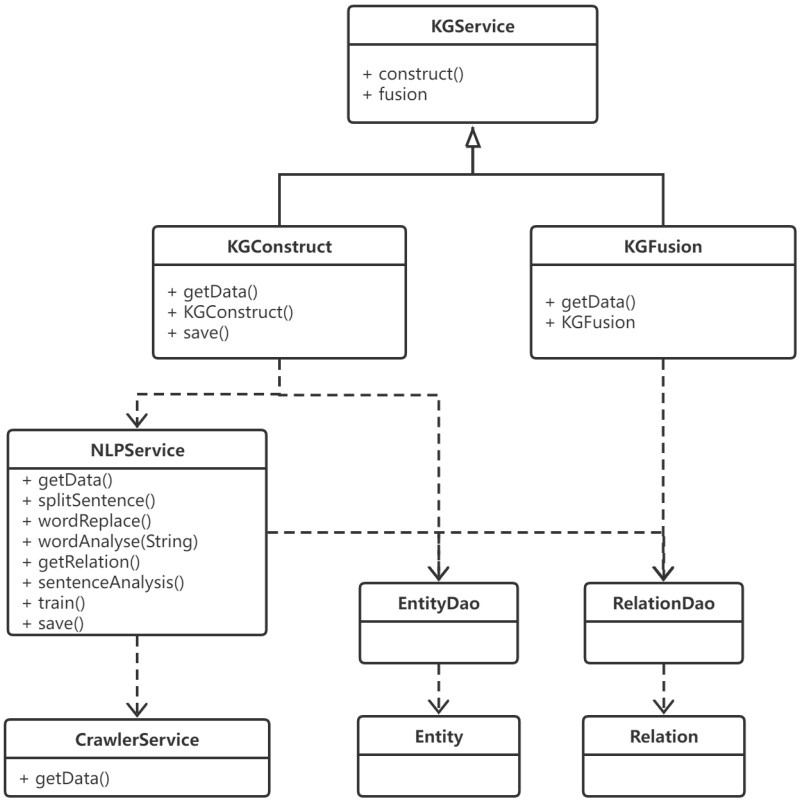

图 3.15: 知识图谱模块核心类图

3.6.2 数据库设计

由于本系统需要借助知识图谱来发掘实体之间的关联性，因此在数据库存储方面只需要设置实体 (Entity) 和关系 (Relation) 两张表即可，将抽取好的数据直接存储到 Neo4j 图数据库中，在需要构建的时候可以直接从数据库中获取对应的实体和对应的关系即可。

表 3.12: Entity 表

字段	含义	类型	说明
id	实体唯一标识	BIGINT	id 为主键
description	对于实体的描述	VARCHAR	不可为空
entity_type	实体的类型，用以区分实体	VARCHAR	不可为空
entity_name	实体的名称，用以标注实体	VARCHAR	不可为空

如表3.12所展示的是 Entity 表的字段的详细说明，该表主要用于 NLP 阶段记录下开源社区中挖掘到的相关实体类，主要记录的本次针对开源社区爬取到的内容进行数据清洗后保留的相关实体。表中包含了 id、description、entityType、entityName 这 4 个字段。

表 3.13: Relation 表

字段	含义	类型	说明
id	关系的唯一标识	INT	id 为主键
relation_name	关系的名称	VARCHAR	不可为空
relation_description	关系的描述，用以描述关系	VARCHAR	不可为空
entity_start	关系的起始实体，此处记录的应为实体 id	BIGINT	不可为空
entity_end	关系的目标实体，此处记录的应为实体 id	BIGINT	不可为空

如表3.13所展示的是 Relation 表的字段的详细说明，与 Entity 表的内容相似，这里主要记录的是经过数据清洗后保留的实体之间的关系，用以标注两个实体的联系。表中包含了 id、relationDescription、relationName、entityStart、entityEnd 这 5 个字段。

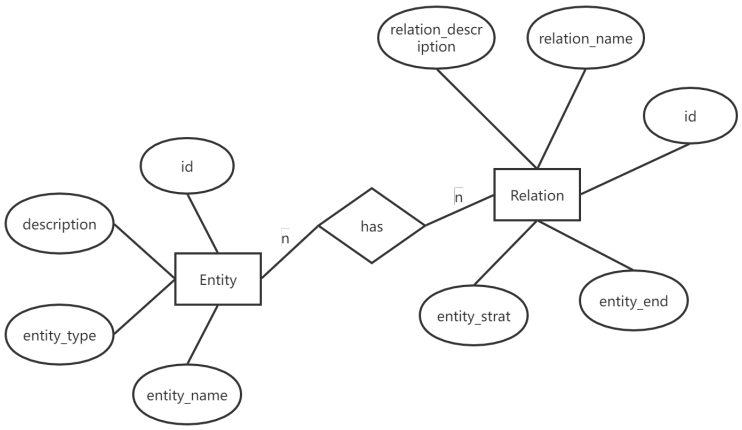

图 3.16: 知识图谱模块数据库 ER 图

图3.16所示是本系统知识图谱模块的 ER 图，其主要描述了知识图谱模块的数据库实体及实体间关系，知识图谱的构建主要是两个元素，即实体 (Entity) 和关系 (Relation)，其中表 Entity 则是代表用于知识图谱构建的实体表，另一方面，Relation 表则是代表用于知识图谱构建的关系表，实体与关系相互包含，一个实体可以对应多个关系，同时多个实体之间可能是同一种关系，因而二者之间是多对多的关系。

3.7 本章小结

本章节主要针对基于知识图谱的开源社区关联性自动化构建系统进行了需求分析和系统设计。紧接着通过用例图和用例描述表具体描述了系统的功能性需求和非功能性需求，并通过 “4+1” 视图从不同视角对系统的架构和实现进行阐述。然后使用类图、架构图和 ER 图等对系统的各个模块进行详细设计的说明，为第四章提供了坚实的技术基础。

第四章　基于知识图谱的开源社区关联性自动化构建系统的实现

在经历过第三章对系统的整体设计及模块的架构设计进行了主要的描述后，本章节的主要工作是详细介绍本系统的具体实现，主要使用时序图来展示进程间的相关交互，并通过关键代码解析的方式来详细阐述相关模块，通过展示系统的运行截图来展示有页面相关模块，用以展示系统的最终效果。

4.1　文本获取模块的实现

文本获取模块主要是需要进行大规模的知识储备用以构建知识库，通过体量足够大的知识库，构建实体间关联性的效果也就更加卓越，自然而然也就能够进一步发掘实体间的潜在联系，方便用户能够在海量的信息中找到有用的资源。为了能够更高效的从各种社区获取到原始数据并用于模型训练，本文选择使用 Scrapy 组件，利用其分布式爬取且可以支持断点续传的特点进行高效的文本获取，作为后续知识库以及模型训练的基础。

4.1.1　文本获取模块流程

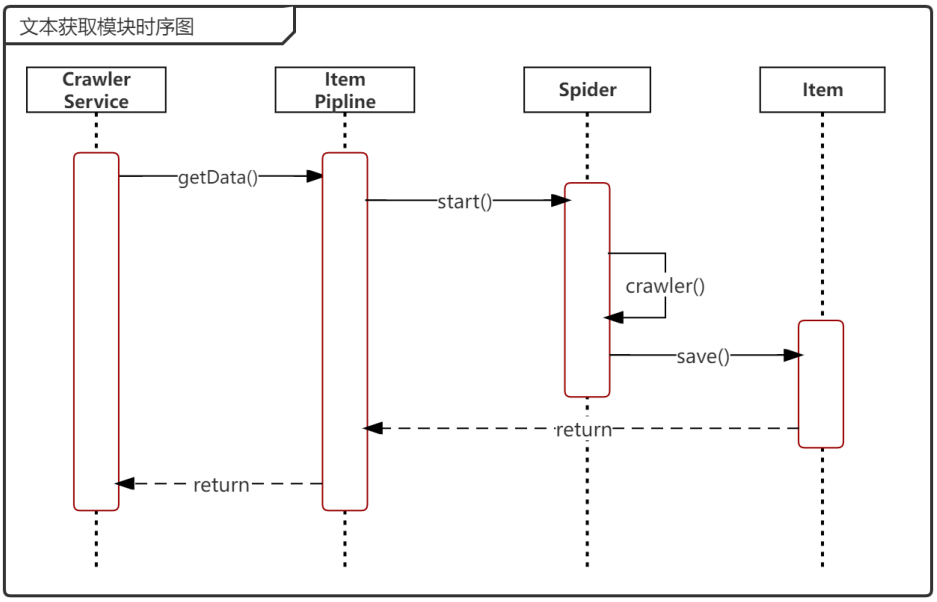

图 4.1: 文本获取模块时序图

图4.1所示的是利用 Scrapy 组件进行文本爬取的时序图，以 CrawlerService 类为程序的入口，负责提供对外的服务以及模块内部各类之间的协调工作，并由 CrawlerService 向 ItemPipline 类发起文本获取请求。ItemPipline 作为一个管道，可以将爬取时需要进行的数据分析、调参等分散的操作聚集为一个整体，并由该类来协调 Item 类和调用 Spider 类对网页进行分析工作，Item 类则作为容器，用以保存 Spider 爬取到的数据，Spider 类则是直接进行爬虫的类，在爬取完成后交回给 ItemPipline 类，并由它提交给 CrawlerService 类用以给付给 NLP 模块进行后续的数据清洗和实体、关系抽取等流程。

4.1.2 关键代码

```
class GitHubSpider(scrapy.Spider):
    name = "gitHub"
    allowed_domains = ["https://github.com/"]
    ...#省略相关设定

    def parse(self, response):
        # 获取title
        title = ""
        for p in title_div.xpath('.//h2/text()'):
            title = p.extract().strip()

        # 爬取detail内容
        detail = ""
        for p in detail_div.xpath('.//p/text()'):
            detail = detail + p.extract().strip() + "\n"

        # 爬取imageList内容
        imageList = ""
        for p in imageList_div.xpath(".//p/img/@src"):
            imageList += p.extract().strip() + " "
        ...#省略Item设定
        yield item
```

图 4.2: GitHubSpider 类关键代码

图4.2所示的是针对 GitHub 设计的 Spider 类，用以从 GitHub 网站中爬取所需要的内容，对应的是所需要的标题、图片和链接等信息，同时利用 Item 类来定义并存储需要保存的元素，在这里表现为 title、image、url 和 detail 这四部分。在进行爬取时，最主要的目的就是从各种非结构化的数据源中抽取出结构化数据，在 Scrapy 组件里则是通过提供各种对应的 Item 类来满足结构化数据抽取的需求。考虑到不同开源社区的文本类型不同，共同的地方在于标题、图片、链接等信息，不同在于 Stack Overflow 中每个问题下的回答由多个用户共同构成，在这里可以考虑将之抽取成一个详细信息的列表，而在 GitHub 中每个项目通常只

有一个说明文档，可以直接保留下对应的详细信息即可，因而对于两个 Item 类这里都将之贴出。

```
class GitHubItem(scrapy.Item):
    # Item对应GitHub中的一个项目
    title = scrapy.Field() #标题
    image = scrapy.Field()  #图片
    detail = scrapy.Field()  #详细信息
    url = scrapy.Field() #链接
    tag = scrapy.Field() #标签
```

图 4.3: GitHubSpider 类关键代码

在 Scrapy 组件中，每当使用 Spider 类将 Item 中的数据进行收集完成后，这些数据将会被传送到对应的 ItemPipeline 中，并利用组件按照一定的预设的顺序执行对 Item 的处理操作，图4.3所示为针对 GitHub 网站的 Item 类设计。

```
class GitHubPipeline(object):   ##用于将GitHubItem转化为json，并存到文件中
    def __init__(self):
        self.count = 0
        self.file = open('MyCrawler/data/GitHub_pedia.json', 'w')
        self.start = time.time()

    def process_item(self, item, spider):
        if item['title'] != 'error':
            line = ""
            if(self.count > 0):
                line += ","
            line += json.dumps(dict(item),ensure_ascii=False) + '\n'
            self.file.write(line)
            self.count += 1
            cur = time.time()
            print("count: "+str(self.count))
            return item
        else:
            raise DropItem("找不到对应页面！")
```

图 4.4: GitHubPipline 类关键代码

图4.4给出的是 GitHubPipline 类的代码，其主要功能是将 GitHubItem 中得到的数据转化为 json 文件并保存，需要将对应的 item 和 spider 类作为参数传入，保存位置为项目文件夹下 MyCrawler/data/GitHubPedia 的文件中。

这里只给出了针对 GitHub 网站的文本获取代码，因其与 Stack Overflow 的实现过程几乎相似，因此在此不再赘述。

4.2 NLP 模块的实现

在 NLP 模块，该模块的主要工作是将文本获取模块得到的数据进行数据清洗，通过 PCNN 模型进行实体和关系抽取，并将实体和关系存储到对应的数据库中，通过不断地迭代进行文本获取、数据清洗和属性抽取从而不断扩充知识库，在后续需要进行知识图谱构建的时候用以提供数据，并可以用于实体之间的潜在关系挖掘。

4.2.1 NLP 模块流程

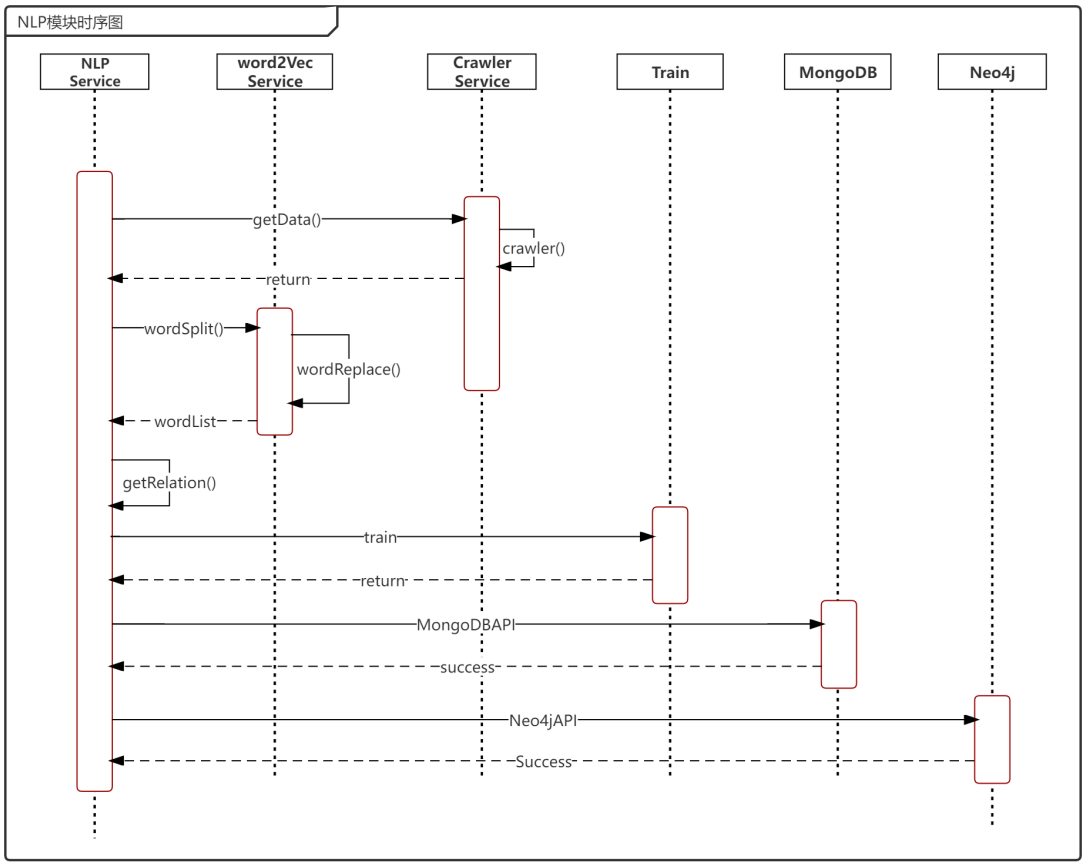

图 4.5: NLP 模块时序图

图4.5展示的是 NLP 模块的时序图，作为模块的关键类，NLPService 是该模块的入口，由此类完成与其他类的交互并对外提供关系抽取、实体抽取的服务。NLPService 类每次会向 CrawlerService，即文本获取模块发起文本获取的请求并由其到对应的开源社区爬取对应的数据返回给 NLPService，获得到初始文本之

后，将文本交由 word2VecService 进行单词切分和同义词替换等工作并生成对应的词向量，完成词向量生成后向 NLPService 类返回单词列表，由其进行关系抽取，并将之交由 Train 类进行模型训练和词向量的数据分析。得到的实体和关系通过调用对应的 API 存储到 MongoDB 和 Neo4j 中。

4.2.2 Word2VecService 模型

在获取到原始的文本数据之后，会经历分词阶段的操作，由于实体 Entity 的特征，需要以其为中心对文本的上下文进行推测，从而在这里考虑使用 Skip-Gram 模型进行关系抽取中的文本预测。

```
# word2vec 模型训练
def word2vec_trans_model(bug_data):
    # 设置词向量维度，采用 skip-gram 模型训练并设置训练线程
    model = Word2Vec(size=200, workers=5, sg=1)
    model.build_vocab(cut_descriptions)

    # 训练模型
    model.train(cut_descriptions,total_examples=model.corpus_count,epochs=model.iter)

    model.save('model/bug_data_model')
```

图 4.6: Word2Vec 模型

在设置模型参数的时候，考虑到开源社区的开放性，讨论贴、用户发言及用户文档中的文本数据量相当庞大，为了得到更好的训练效果，在这里将词向量的维度尽量设置大一点，在这里将词向量的维度设置成 200，并使用 5 个训练线程进行模型构建。在经过 buildVocab 方法建立词汇表后进行模型训练，即调用 train 方法进行模型的训练，在模型训练完成之后调用 save 方法将模型保存下来，以便后续使用。

4.2.3 使用 Tensorflow 实现 PCNN 模型

在关系抽取模块，选择使用 Tensorflow 来实现 PCNN 模型，PCNN 模型的参数设置如图4.7所示，根据2所示的 PCNN 模型多实例学习流程，在这里将 PCNN 的算法设置最小的 instance 单位设置为 0 和最小的包 b_s 设置为 1，以此进行切分将句子切分为多个小批量，将训练用的最小包单位设置为为 2 以进行模型训练。同时对 PCNN 模型在执行过程中完成训练时需所要的，如最大句子长度、训练次数、最大训练次数等参数的设置。

```
# 以instance作为最小单位
MODE_INSTANCE = 0
# 以bag为最小单位
MODE_ENTPAIR_BAG = 1
# 以bag为最小单位用以模型训练
MODE_RELFACT_BAG = 2
root_path = os.getcwd()
...#省略相关设定
class model:
    #句子最长长度
    max_length = 60
    batch_size  = 16
    train_level  = MODE_RELFACT_BAG
    test_level =  MODE_ENTPAIR_BAG
    encoder = "pcnn"
    max_epoch = 60
    learning_rate= 0.05
    ...
    pcnn_kernel_size = 3
    pcnn_hidden_size = 230
    pcnn_stride_size = 1
    pcnn_activation = tf.nn.relu
......
```

图 4.7: PCNN 模型参数

PCNN 的函数实现如图4.8所示，PCNN 模型会对句子首先进行清洗，根据实体将句子切割成左、中、右三个部分，每个部分会对应一个句子向量，如图中所示的 left、mid 和 right 三个向量，以及最终需要输出的特征向量 finalFeature，对这些向量单独设置 filters 和 kernelSize，用以在之后的模型中将同样通过调用对应的 Train 方法进行模型训练，使之达到预期的目标。

```
def Piece_Wise_CNN(left_emb,mid_emb,right_emb,feature_map,n_class):
    # PCNN模型中文本会被两个entity实体将句子分为左中右三个部分
    # 每个部分单独定义向量
    # 设置左侧句子向量
    left = tf.keras.layers.Conv1D(filters=feature_map,kernel_size=3)(left_emb)
    left = tf.keras.layers.GlobalMaxPool1D()(left)
    # 设置中间句子向量
    mid = tf.keras.layers.Conv1D(filters=feature_map,kernel_size=3)(mid_emb)
    mid = tf.keras.layers.GlobalMaxPool1D()(mid)
    # 设置右侧句子向量
    right = tf.keras.layers.Conv1D(filters=feature_map,kernel_size=3)(right_emb)
    right = tf.keras.layers.GlobalMaxPool1D()(right)
    final_feature = tf.concat([left,mid,right],1)
    ...#省略输出
```

图 4.8: PCNN 函数实现

4.2.4 文本识别功能实现

如图4.9所示是文本识别的效果展示，文本识别功能基于 NLP 模块提供的实体抽取和关系抽取功能，在本系统中，用户可以通过在给定的文本框中输入原始文本，通过“识别”按钮向系统发起识别请求，系统会在下方显示经过实体识别和分词的结果。

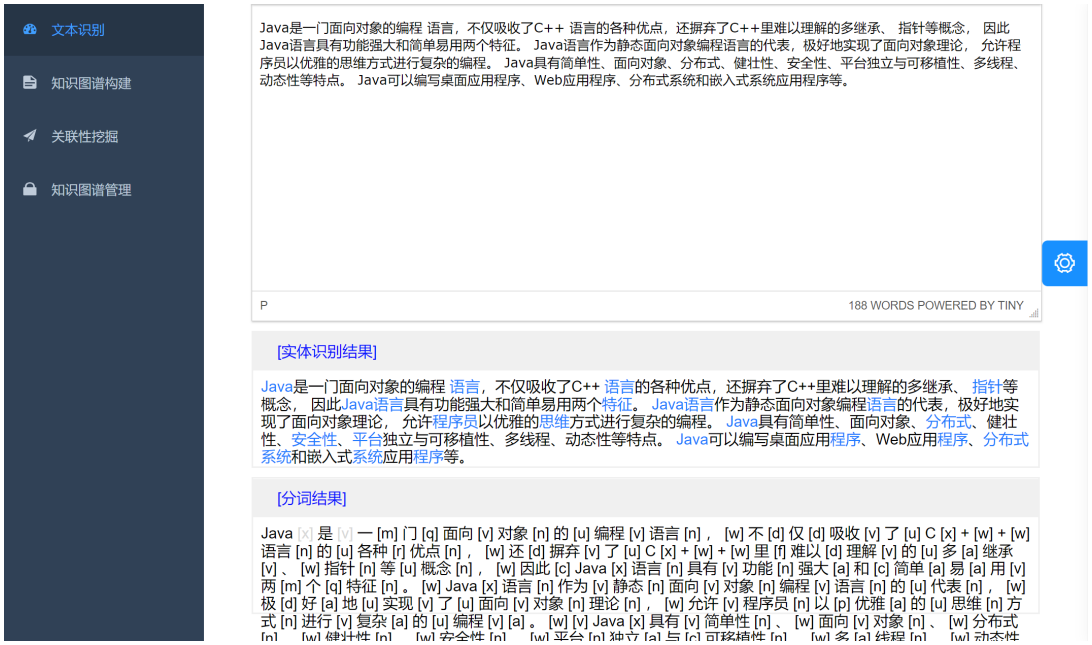

图 4.9: 文本识别效果图

本功能的主要意义在于让用户能够及时发现文本中存在的实体，当实体抽取完成后，用户则可以根据文本中的实体进行相关的知识图谱构建以及关联性挖掘，方便用户更加快捷且及时发现文本中有用的信息。且当数据库中没有该实体时会自动将抽取出来的实体加入到数据库中。

4.3 知识图谱模块的实现

本文需要以知识图谱为基础，则离不开知识图谱的构建，那么知识图谱模块的主要任务则是在文本获取模块和 NLP 模块的基础上利用已经得到的知识库构建需要的知识图谱，并通过查找构建完的知识库在查找实体之间的最短路径的过程中发现实体间的潜在联系，并以此为基础进行知识图谱的融合工作，也就是本文所说的关联性自动化构建。

4.3.1 知识图谱模块流程

图4.10所展示的是知识图谱模块的时序图，用以展示知识图谱模块运行时各个类之间的相互交互关系。

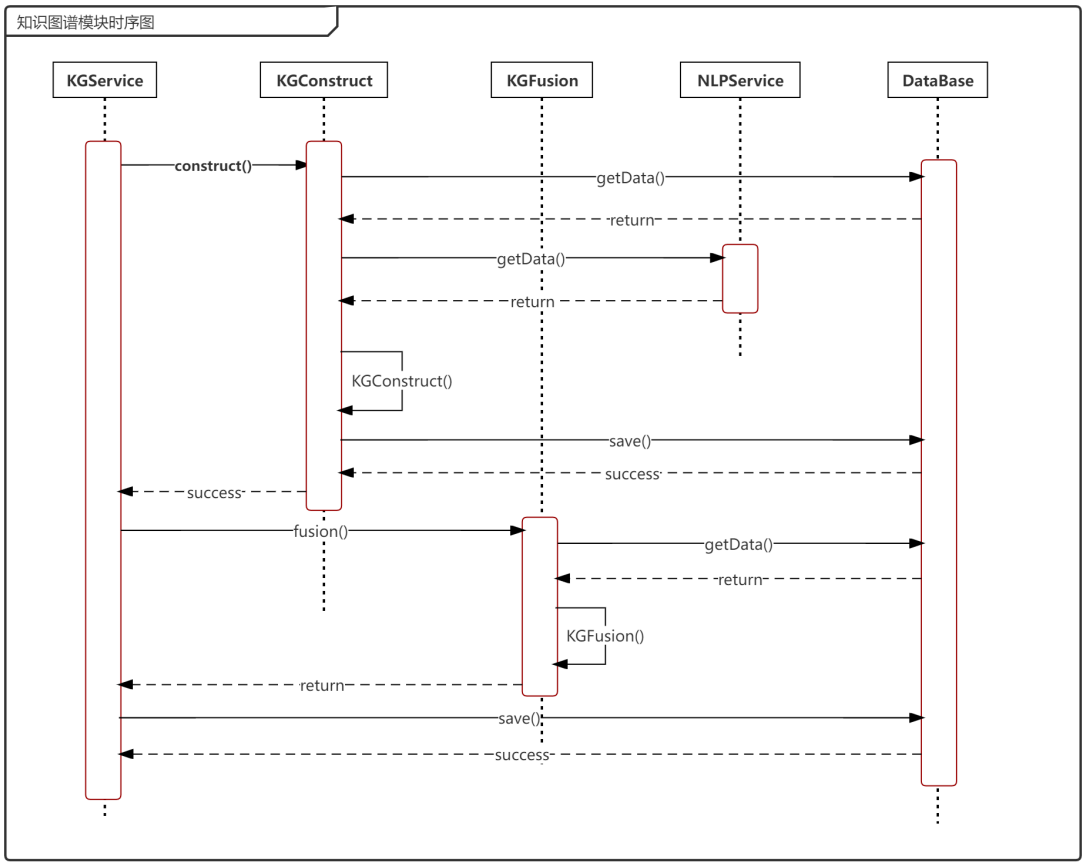

图 4.10: 知识图谱模块时序图

如图所示，KGService 是知识图谱模块的入口，由此类负责协调各类，并通过该类调用其他接口完成整个知识图谱的构建工作。在需要构建知识图谱的时候，由 KGService 从对应数据库 (Neo4j 和 MongoDB，即知识库) 中获取构建图谱所需要的实体和关系，如果无法从数据库中获取，则需要向 NLP 模块发起数据获取请求，并由 NLP 模块调动文本获取模块从对应站点获取数据，得到的数据将由 word2Vec 类构建成词向量，并进行句子切分、同义词替换等工作，完成后将词列表返回给 NLPService。NLPService 以此进行实体和关系抽取并调用对应的 API 将数据存储到对应的数据库中。Train 类则主要负责模型的训练。

4.3.2 知识图谱构建功能详细设计

在得到关键的实体和关系之后，接下来的任务就是利用这些实体和关系来构建用户所需要的知识图谱。

知识图谱构建的功能即是将用户输入的目标实体为核心，在知识库中搜索相关的实体，并以此来生成知识图谱，生成的知识图谱由前端展示给用户并由用户决定是否保存。

```
def KGConstruct(request):
    ctx = {}
    #根据传入的实体名称搜索出关系
    if(request.GET):
        entity = request.GET['user_text']
        #连接数据库
        db = neo_con
        entityRelation = db.getEntityRelationbyEntity(entity)
        if len(entityRelation) == 0:
            #若数据库中无法找到该实体，则返回数据库中无该实体
            return render(request,'entity.html',{'ctx':json.dumps(ctx,ensure_ascii=False)})
        else:
        #返回查询结果
        #统计结查询结果并进行排序
            entityRelation = sortDict(entityRelation)
            return render(request,'entity.html',
                        {'entityRelation':json.dumps(entityRelation,ensure_ascii=False)})

    return render(request,"entity.html",{'ctx':ctx})
```

图 4.11: 知识图谱构建类代码

图4.11所示为知识图谱构建 (KGConstruct) 类的代码实现，由前端输入目标实体并发起 Http 请求，由于实体和关系都存在于 Neo4j 图数据库中，需要连接数据库，在数据库中进行查找，如果数据库中有对应的实体，即 $entityRelation! = 0$，则直接查找到对应的实体并进行排序，将之返回到前端进行展示，展示效果为以目标实体为中心，以实体和关系连线的形式连接与目标实体相关的实体，连线代表实体之间的关系，并统计实体之间关系出现的次数，通过 sortDict 方法排序后将之展现出来，用以展示。如果在数据库中找不到相关联的实体，即 $entityRelation == 0$，这代表数据库中不存在对应的实体，则直接向前端返回数据库中无该实体。其最终实现效果如图4.12所示，从图中可以看出，用户在“查看条件”的选项框中输入需要的关键词并点击查询按钮向系统发起知识图谱构建请求，系统则会以关键词为中心向用户展示对应的知识图谱。

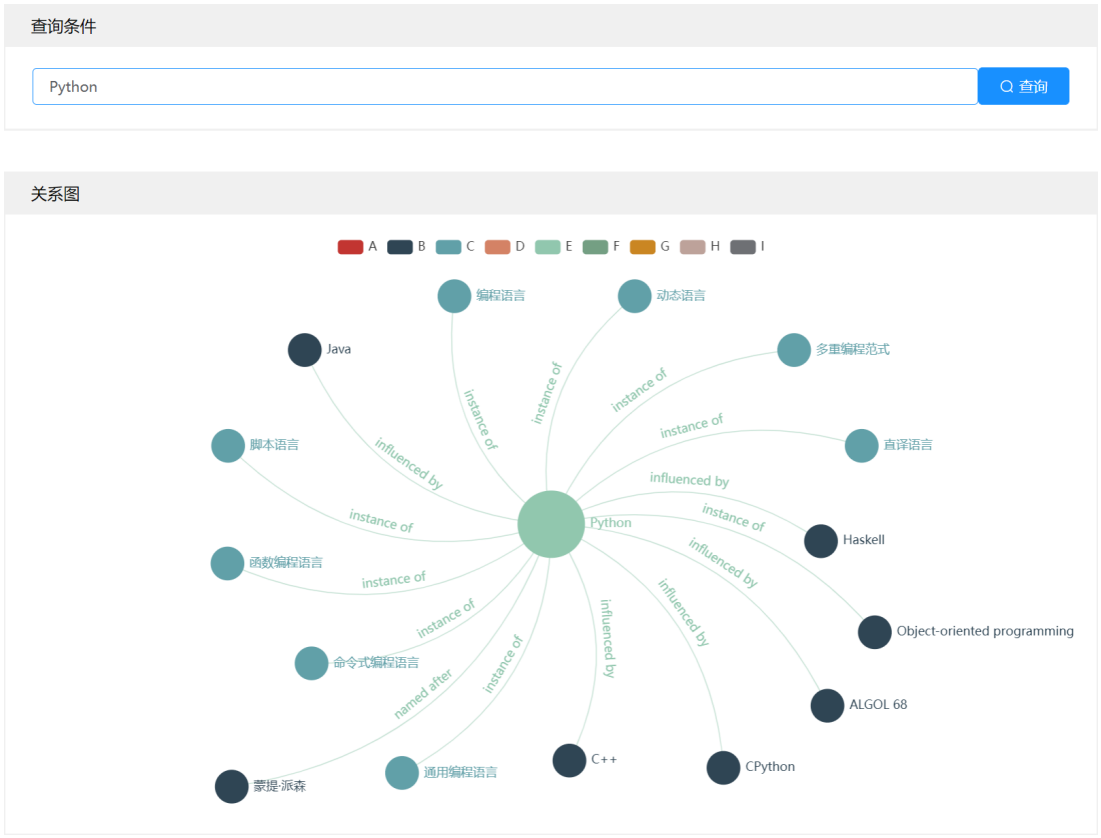

图 4.12: 知识图谱构建效果图

4.3.3 关联性挖掘功能详细设计

图4.13所展示的是关联性挖掘 (search_relation) 类的部分实现，其同样需要由用户完成在前端的指定，根据 3.3 节中关联性挖掘的用例描述所述，当用户需要进行关联性挖掘时，用户需要选择对实体 1、实体 2 和关系进行设定，根据不同的用户设定，由后端会接收后完成对应的响应，例如当用户只设定第一个实体时，后端会根据接收到的实体连接到数据库中，在数据库中进行查找，并向前端展示与指定实体有之间关联的实体和关系；而当用户设定了两个实体，后端接收到之后则会尝试通过数据库中的数据在两个实体之间建立起一条最短的联通路径，也即本文的核心功能之一，发掘实体之间的潜在关联性。在这部分中，例如用户设定两个实体，后端接收到 entity1 和 entity2 则根据这两个实体在数据库中遍历查找对应的关系及实体，找到实体之后通过 sortDict 方法对查找到的关系进行排序，最后向前端返回得到的实体并以实体和练线的形式用过最短路径表现出来，从而展现实体之间的联系。

```
def SearchRelation(request):
    ...#省略相关设定
    #只输入一个实体，则仅输出与实体有直接关系的实体和关系
    if(len(entity1) != 0 and len(relation) == 0 and len(entity2) == 0):
        searchResult = db.findRelationByEntity(entity1)
        searchResult = sortDict(searchResult)
        if(len(searchResult)>0):
            return render(request,'relation.html',
                    {'searchResult':json.dumps(searchResult,ensure_ascii=False)})
    if(len(entity2)!=0 and len(relation)!=0 and len(entity1) == 0):
        searchResult = db.findOtherEntities2(entity2,relation)
        searchResult = sortDict(searchResult)
        if(len(searchResult)>0):
            return render(request,'relation.html',
                    {'searchResult':json.dumps(searchResult,ensure_ascii=False)})
    #只输入单个实体和关系，则输出与实体且具有对应关系的其他实体
    if(len(entity1)!=0 and len(relation)!=0 and len(entity2) == 0):
        searchResult = db.findOtherEntities(entity1,relation)
        searchResult = sortDict(searchResult)
        if(len(searchResult)>0):
            return render(request,'relation.html',
                    {'searchResult':json.dumps(searchResult,ensure_ascii=False)})
    ……
    #用户如若输入两个实体，则输出两个实体之间的最短路径
    if(len(entity1) !=0 and len(relation) == 0 and len(entity2)!=0):
        searchResult = db.findRelationByEntities(entity1,entity2)
        if(len(searchResult)>0):
            print(searchResult)
            searchResult = sortDict(searchResult)
            return render(request,'relation.html',
                    {'searchResult':json.dumps(searchResult,ensure_ascii=False)})
    #用户如果完整指定两个实体和关系，则向用户展示两个实体之间是否存在对应关系
    if(len(entity1)!=0 and len(entity2)!=0 and len(relation)!=0):
        searchResult = db.findEntityRelation(entity1,relation,entity2)
        if(len(searchResult)>0):
            return render(request,'relation.html',
                    {'searchResult':json.dumps(searchResult,ensure_ascii=False)})
```

图 4.13: 关联性挖掘类关键代码

关联性挖掘功能与知识图谱构建功能有所异同，不同的是，关联性挖掘功能需要根据用户设置的参数来进行检索展示，在用户仅指定一个实体的情况下其功能与知识图谱构建功能是相似的，在用户设定两个实体时，则将由系统连接数据库并在其中查找对应的关系进而在两个实体之间尝试构建最短路径并以知识图谱的形式，即实体和连线的方式表现出来，从而更为直观的展现出实体之间潜在的关联性。图4.14所展示的是关联性挖掘的效果，其中作为 Python 作者的蒙提 · 派森与 Java 作者的詹姆斯 · 高斯林中间存在着由编程语言这一节点为通路的潜在联系。

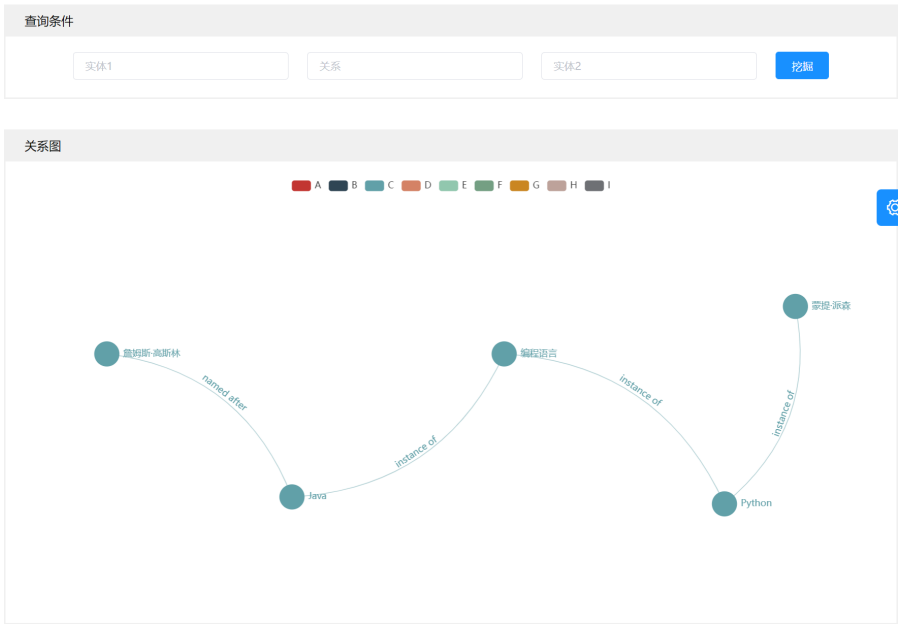

图 4.14: 关联性挖掘效果图

4.4 系统关键效果展示

图4.15展示的是系统的整体效果，在这里系统提供文本识别、知识图谱构建、关联性挖掘和知识图谱管理四个主要的功能。

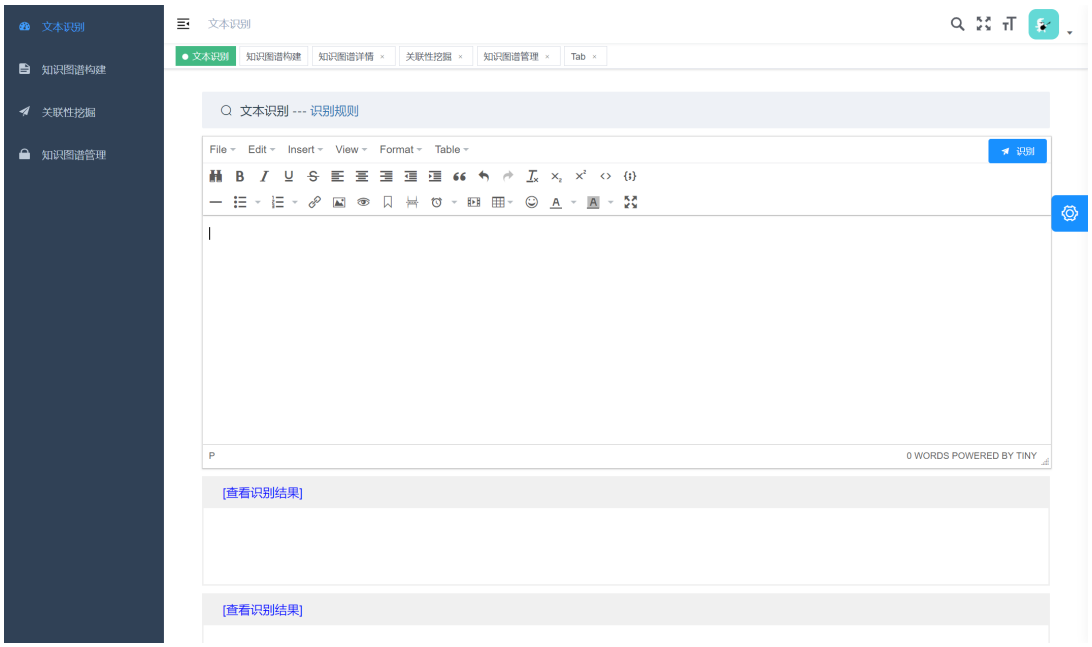

图 4.15: 系统整体效果图

其中，本系统用例设计中的提到的包含的知识图谱的导出、融合、查看等功能一起放在了知识图谱管理部分，接下来将会对知识图谱管理和知识图谱融合的功能进行详细描述与展示。

4.4.1 知识图谱管理功能实现

图4.16所示的是知识图谱管理界面的效果，当用户进入到知识图谱管理界面时，系统会展示保存到数据库中的图谱，对于已经保存的图谱，用户可以进行导出、查看、以及融合操作，用户还可以勾选多个知识图谱再点击“融合”按钮向系统发起知识图谱融合的请求，系统在完成对知识图谱的融合工作后会向前端展示融合后的图谱并自动保存到数据库中，如果用户不需要该图谱，则需要手动进行删除。

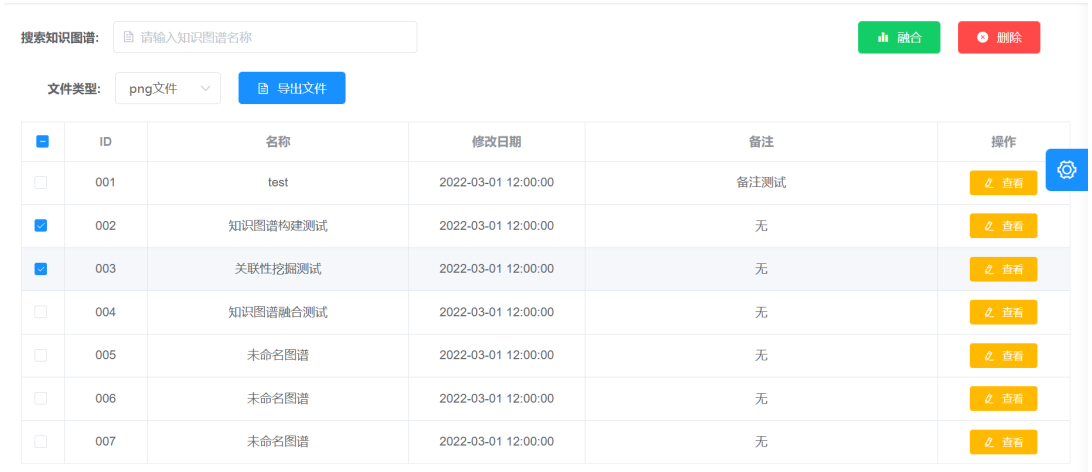

图 4.16: 知识图谱管理效果图

图4.17所展示的是知识图谱的详情页面，在这里还可以根据用户意愿选择文件类型进行导出。

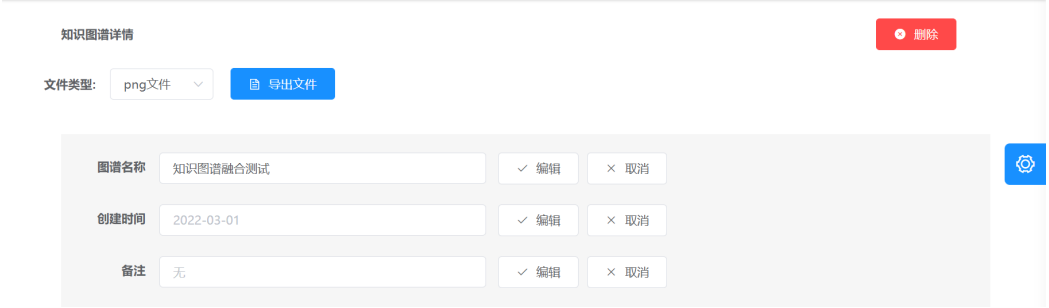

图 4.17: 知识图谱详情页面

在知识图谱管理页面可以通过点击“查看”按钮知识图谱详情页面查看保存的知识图谱详情，并且可以在知识图谱的详情页面对知识图谱的相关信息进行一些修改。

4.4.2 知识图谱融合效功能实现

知识图谱融合的功能建立在知识图谱构建和关联性挖掘功能的基础上，通过遍历两个图谱中共同的节点将之合并，并以此为中心重新构建一个包含原图谱中所有节点的新的知识图谱，无论是否存在共同的节点，都需要在所有节点之间尝试建立实体间的最短路径并展示出来，用以进行知识图谱的扩充融合操作。

图4.18所展示的是知识图谱融合的效果图，从上图可以发现，这里将知识图谱构建功能得到的图谱和关联性挖掘得到的图谱进行了融合，尽管通过关联性挖掘功能就可以得到实体之间的潜在关联性，但是通过知识图谱的融合功能，不仅可以扩充原有的知识图谱、让潜在关联性得以直接地展示出来，还可以省去用户逐个进行关联性挖掘的繁琐过程，使得知识图谱构建的构建效率得以提升，因而知识图谱融合这一功能可以更好的辅助系统进行关联性构建。

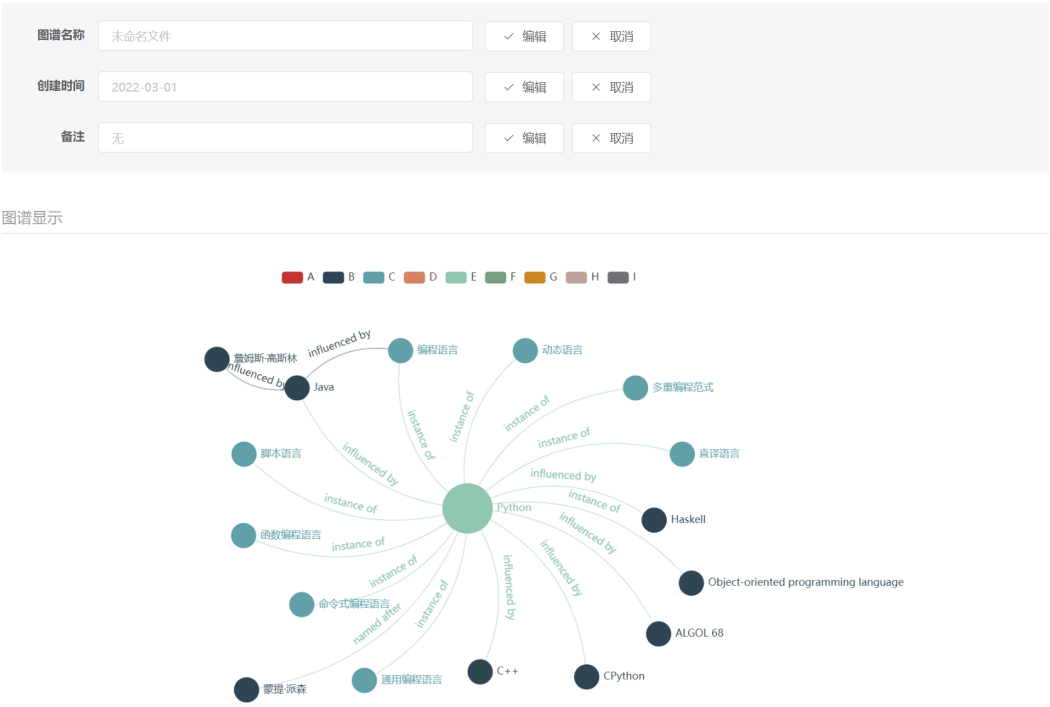

图 4.18: 知识图谱融合效果图

4.5 本章小结

本章节主要对基于基于知识图谱的开源社区关联性自动化构建系统进行了细节方面的阐述，对每个模块都选择使用系统时序图来阐述该模块的实现流程，并附上关键的代码解释核心类实现的原理，然后将系统关键部分的运行截图进行了展示。

第五章　基于知识图谱的开源社区关联性自动化构建系统的测试

在经过论文对系统的需求分析和设计与实现之后，紧接着要做的是根据本文设计的针对系统的功能性要求与非功能性需求进行验收测试，以在保证系统设计完全可行的前提下，保证系统的非功能性要求满足预期目标。因此，本章节的主要目标即是先对系统功能的完整性，以及实用性要求进行检测，并在进行了功能性检测之后再针对诸如可移植性、易用性、性能等方面展开检测与验证效果，测试之前会对测试设计进行描述并展示测试的结果。

5.1　测试准备

5.1.1　测试目标

本文对基于知识图谱的开源社区关联性自动化构建系统的测试主要包含三个方面，首先是对系统的功能点进行测试，其主要目的是验证第三章中所述的功能点能否通过测试用例的测试并达到预期的需求，在完成功能测试后则是对系统的非功能需求进行测试，在本文中则是在第三章中提到的可扩展性、可移植性和性能需求方面的测试。

5.1.2　测试环境

根据图3.8所描述，系统的测试环境包括用于与系统进行交互的用户机，用于运行 Vue 程序的前端服务器，用于运行包含 Nginx 和 Spring Boot 以及 Redis 缓存的后端服务器，用于运行 Python 程序的 Python 后端服务器，以及用于部署 Neo4j、MongoDB 数据库的数据库服务器。在这里将所有的服务部署在阿里云的服务器上，并根据运行的要求分别进行部署，其具体信息如表5.1中所示。

表 5.1: 系统测试环境

设备	运行程序	详细信息
用户机	Edge 浏览器	Windows11 22000.556
前端服务器	Vue	ECS 服务器 Ubuntu 18.04
后端服务器	Spring Boot,Python	ECS 服务器 4G 内存，50M 带宽
数据服务器	Neo4j,MongoDB,Redis	ECS 服务器 4G 内存，50M 带宽

5.2 功能测试

功能测试即黑盒测试，指不考虑程序的内部实现，仅根据程序的输入输出来验证程序是否能够达到需求说明文档的要求的测试过程。在这里将对系统的各个功能模块设计相应的测试用例，以验证系统的各项功能是否完整有效以及是否达到预期的效果。

文本识别的测试用例如表5.2所示，测试的是在用户将待识别的文本正常输入到文本框中并发起实体识别请求时系统对用户请求作出的反应。

表 5.2: 文本识别测试用例

用例编号	TC1
测试名称	文本识别测试
待测试功能	用户进入到文本识别页面，能够通过在文本框内填写需要进行文本识别的原始文本由系统展示分词和文本识别的结果并展示
前置条件	用户得到相应授权
正常流程	1. 进入文本识别页面 2. 在文本框中输入待识别的文本 3. 点击“识别”按钮
预期结果	1. 系统展示文本的分词结果 2. 系统展示实体识别的结果
测试结果	符合预期

表 5.3: 知识图谱构建测试用例

用例编号	TC2
测试名称	知识图谱构建测试
待测试功能	用户进入到实体识别页面，能够通过在文本框内填写需要进行实体识别的原始文本并由系统展示分词和实体识别的结果
前置条件	用户得到相应授权，且知识库中存在内容
正常流程	1. 进入知识图谱构建页面 2. 在文本框中输入需要建立的初始实体 3. 点击“构建”按钮 4. 系统查询数据库判断是否存在对应实体并进行知识图谱构建
预期结果	1. 数据库中存在对应实体，则生成对应的知识图谱 2. 数据库中不存在对应实体，则向前端返回“数据库中暂无该实体”
测试结果	符合预期

知识图谱构建的测试用例如表5.3所示，测试的是系统能否正确利用知识库中已存在的相关实体以用户输入的目标实体为中心构建一个网状的知识图谱的功能。

关联性挖掘的测试用例如表5.4所示，测试的是在用户指定实体和关系的情况下系统能否正确利用数据库中的内容构建实体之间的联系以挖掘实体之间的关联性。

表 5.4: 关联性挖掘测试用例

用例编号	TC3
测试名称	关联性挖掘测试
待测试功能	用户指定实体或关系的情况下，系统能否按需求展示对应的知识图谱
前置条件	用户得到相应授权，且知识库中存在内容
正常流程	1. 进入关联性挖掘页面 2. 在文本框中仅设置单个实体，并点击“挖掘”按钮 3. 在文本框中设置两个实体，并点击“挖掘”按钮 4. 在文本框中设置两个实体并指定关系，并点击“挖掘”按钮 5. 在文本框最终设置单个实体并指定关系，并点击“挖掘”按钮
预期结果	
特殊流程	1. 若知识库中不存在对应实体则显示“数据库中暂无该实体”
测试结果	符合预期

知识图谱存储的测试用例如表5.5所示，测试的是在系统正确生成知识图谱之后，用户点击“存储”按钮发起知识图谱存储请求后，系统能否正确按照用户意愿将之存储的功能。

表 5.5: 知识图谱存储测试用例

用例编号	TC4
测试名称	知识图谱存储测试
待测试功能	用户在通过知识图谱构建或关联性挖掘等知识图谱构建操作之后需要将之保存，系统能否正确将图谱保存下来
前置条件	知识图谱已经生成
正常流程	1. 通过相关操作已经生成知识图谱 2. 点击“保存”按钮 3. 填写相关信息后点击“确定”按钮
预期结果	1. 系统弹出文本框提示设置需要保存的图谱名称 2. 系统根据用户的设置将图谱保存 3. 用户未命名则默认为“未命名图谱”
拓展流程	1. 用户未添加备注则默认备注为“无”
测试结果	符合预期

知识图谱管理的测试用例如表5.6所示，系统需要能够正确的展示保存的知识图谱的名称和日期信息，在这里测试的是当用户进入到知识图谱管理界面时可以管理已经生成的知识图谱，即可以对已保存的内容进行重命名、导出、融合、删除等修改操作。

表 5.6: 知识图谱管理测试用例

用例编号	TC5
测试名称	知识图谱管理测试
待测试功能	用户进入到知识图谱管理页面，页面能否正确展示已经保存的知识图谱信息，且能否进行查看、导出、融合、删除操作
前置条件	用户得到相应授权
正常流程	1. 进入知识图谱管理页面 2. 勾选已保存的知识图谱点击“查看”按钮 3. 勾选已保存的知识图谱点击“删除”按钮 4. 勾择已保存的知识图谱点击“导出”按钮 5. 勾选两个已保存的知识图谱点击“融合”按钮
预期结果	1. 系统跳转到对应图谱详情页面 2. 系统删除该条图谱的相关记录 3. 系统根据选择的图片格式将图谱导出到本地 4. 系统调用融合功能完成知识图谱的融合操作
测试结果	符合预期

知识图谱融合的测试用例如表5.7所示，对于知识图谱融合功能，测试的是系统能否正确通过遍历两个知识图谱中所有的节点并依次进行关联性挖掘以构成一张更大的知识图谱。

表 5.7: 知识图谱融合测试用例

用例编号	TC6
测试名称	知识图谱融合测试
待测试功能	用户进入到知识图谱管理页面，勾选任意两张已保存的知识图谱并选择融合按钮进行知识图谱融合
前置条件	用户得到相应授权，且已保存超过两张知识图谱
正常流程	1. 进入知识图谱管理页面 2. 任意勾选两张知识图谱 3. 点击“知识图谱融合”按钮 4. 对新生成的知识图谱进行保存
预期结果	1. 系统展示知识图谱列表 2. 系统根据用户选择的图谱到数据库中进行关联性挖掘 3. 系统展示融合后更大的知识图谱 4. 系统将知识图谱按用户意愿进行保存并添加到图谱列表
测试结果	符合预期

知识图谱导出的测试用例如表5.8所示，需要测试的是系统能否正确的将已经生成的知识图谱以图片的形式按用户意愿保存到本地中。

表 5.8: 知识图谱导出测试用例

用例编号	TC7
测试名称	知识图谱导出测试
待测试功能	用户进入到图谱管理界面，选择已经存储的知识图谱并选择需要的图片格式将知识图谱保存到本地
前置条件	用户得到相应授权，且知识图谱已经生成
正常流程	1. 进入知识图谱管理页面 2. 选择知识图谱，并点击“导出”按钮 3. 选择“png”格式 4. 点击“确定”按钮
预期结果	1. 系统展示存储的图谱列表 2. 系统弹出需要保存的图片格式选择框 3. 系统通过浏览器下载将图片以 png 格式保存到本地
测试结果	符合预期

查看图谱详情的测试用例如表5.9所示，需要测试的是系统能否正确跳转到对应的知识图谱详情页面并展示，以及系统能否正确响应用户发起的对知识图谱相关信息的修改请求。

表 5.9: 查看图谱详情测试用例

用例编号	TC8
测试名称	查看图谱详情测试
待测试功能	用户进入到图谱管理界面，选择已经存储的知识图谱点击“查看”按钮，验证能否正确查看已保存的知识图谱
前置条件	用户得到相应授权，且知识图谱已经生成
正常流程	1. 进入到知识图谱管理界面 2. 选择已保存的知识图谱并点击“查看”按钮 3. 在知识图谱详情页面对图谱名称进行修改 4. 在知识图谱详情页面给图谱添加备注 5. 在知识图谱详情界面对已有的备注进行修改
预期结果	1. 系统跳转到对应的知识图谱详情页面 2. 系统根据输入的信息完成对图谱名称的修改 3. 系统根据输入的信息给图谱添加备注 4. 系统根据输入的信息完成对原有图谱备注的修改
测试结果	符合预期

知识库构建的测试用例如表5.10所示，需要测试的是系统能否正确完成对页面内容的爬取以及能否对其中的文本内容进行实体和关系抽取操作。

表 5.10: 知识库构建测试用例

用例编号	TC9
测试名称	知识库构建测试
待测试功能	系统能否根据用户输入的 URL 爬取网页内容并进行实体和关系抽取
前置条件	用户得到相应授权
正常流程	1. 用户进入到知识库构建页面 2. 用户在开源社区查找对应资料，复制 URL 到选项框中 3. 用户发起停止构建请求
预期结果	1. 系统根据 URL 对用户指定内容进行爬取 2. 系统对爬取到的内容进行实体和关系抽取将之保存到数据库中 3. 系统停止对页面的爬取 4. 系统完成对爬取到的文本进行实体和关系抽取任务
测试结果	符合预期

综上所述，本系统的功能均能达到预期，说明本系统达到了需求分析部分所提出的功能需求，可以正确提供相应的服务，达到用户的期望，因而具有实用价值，可以投入使用。

5.3 可用性测试

本小节的目的是测试系统的可用性，考虑到本系统对实体间关联性构建时需要满足向用户及时反馈结果的要求，需要确保在系统遇到服务器宕机的问题时能在 2 分钟内恢复服务，因而对系统进行可用性测试。

5.3.1 测试设计

为了验证系统在遇到宕机等情况时的恢复能力，因而在这里考虑在已经成功部署的服务器上人为造成宕机情况，并在测试端不断发起接口调用，记录从开始宕机到恢复的时间进行观察，并以此与目标时间进行比较，判断系统能否达到可用性需求。

5.3.2 测试执行

表 5.11: 可用性测试结果表

实验编号	系统关闭时间	服务恢复时间	服务停滞时间
Test1	10:01:23	10:02:16	53
Test2	10:07:02	10:07:54	52
Test3	10:12:41	10:13:32	51
Test4	10:20:04	10:20:56	52

经过 4 次手动对服务器进行人为关闭重启，其结果如表5.11所示，可见系统启动时间平均耗时为 52s，因而可以得出服务器宕机时间平均在 52s，满足系统宕机能在 2 分钟内恢复地要求。

5.4 可移植性测试

由于操作系统的不断发展，本系统也需要可以在不同的操作系统上运行，因而在此需要将系统在不同的操作系统上进行正常运行的测试，来验证系统的可移植性能否达到目标。

5.4.1 测试设计

对于可移植性测试，这里设计为在统在不同的操作系统上安装部署该系统，并尝试运行、记录系统的运行状况，这里选择在不同的系统上都进行一次 5.2 小节的功能测试以验证该系统能否在不同的操作系统上正确运行。

表 5.12: 系统测试环境

操作系统名称	系统指令集
阿里云 CentOS	x86_64
统信 UOS 系统	ARM
优麒麟操作系统	ARM
Windows v10	x86_64
Mac OS X	ARM
Ubuntu v20.04	ARM

测试的操作系统除本机的 Windows11 外，选择常见的 Mac OS X 和 Ubuntu 两个 ARM 架构的操作系统，同时选择了最为广泛的 Windows10 和阿里云提供的 CentOS 两个 x86_64 架构的操作系统，此外为支持国家信创产业、支持自主研发操作系统，这里还选用了统信 UOS 和优麒麟两款 ARM 架构的国产操作系统，具体如表5.12所示。测试内容为在不同的系统上验证 TC1 到 TC9 共计 9 个测试用例的运行状况。

5.4.2 测试执行

根据表5.13可以发现，在不同操作系统的安装部署后，经过测试运行发现系统可以在不同的操作系统上正常运行，因而可以验证基于知识图谱的开源社区关联性自动化构建系统具有良好的可移植性。其中，测试的系统主要是常见操作系统和两款国产操作系统，均为 ARM 和 x86_64 架构，所有操作系统均使用 Docker 进行部署。

表 5.13: 可移植性测试结果表

操作系统名称	系统指令集	系统运行状况	用例通过情况
阿里云 CentOS	x86_64	正常运行	9/9
统信 UOS 系统	ARM	正常运行	9/9
优麒麟操作系统	ARM	正常运行	9/9
Windows v10	x86_64	正常运行	9/9
Mac OS X	ARM	正常运行	9/9
Ubuntu v20.04	ARM	正常运行	9/9

5.5 性能测试

性能测试主要目的测试系统在高并发的情况下系统能否仍然保持原来的响应速度且是否正常运行。

5.5.1 测试设计

在该小节，性能测试工具选择使用 JMeter 工具来对系统的关键接口进行性能测试，通过设置线程数量来模拟多个用户同时访问的高并发情况。

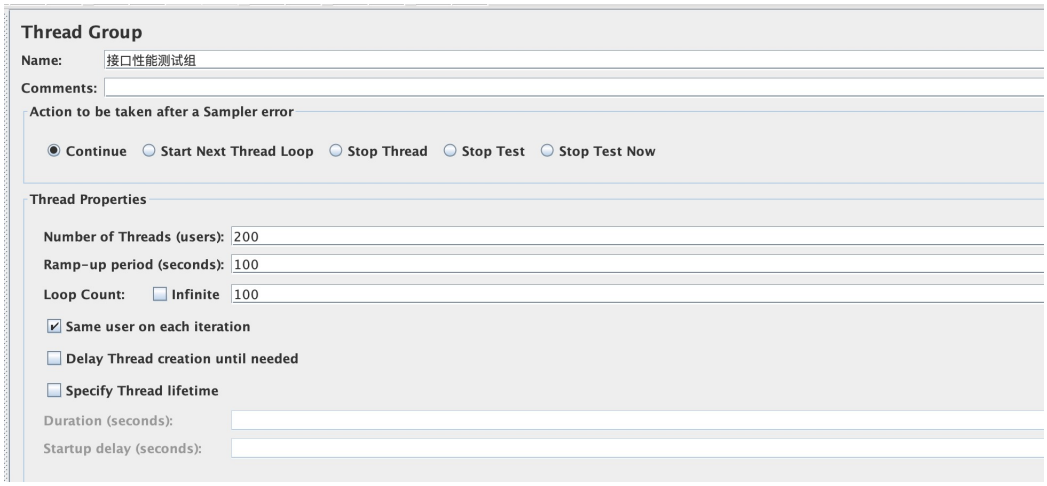

图 5.1: 线程组配置截图

以文本识别接口为例进行性能测试，对应线程组设置如图5.1所示，其中，Number of Threads 代表线程数量，Ramp-Up Period=100 代表 100 秒内全部启动，Loop Count=100 代表循环发出 100 个请求。表5.14列出了需要进行性能测试的接口，接下来将对这些接口进行高并发的性能测试。

表 5.14: 待测试接口表

接口编号	接口描述	接口路径
I1	文本识别	/nlp/textSearch
I2	知识图谱构建	/kg/KGConstruct
I3	关联性挖掘	/kg/searchRelation
I4	查看图谱详情	/control/check

5.5.2 测试执行

经过 JMeter 的模拟并发测试，其结果如表5.15所示，可以看出，在预设的高并发的情况下，系统的各个主要接口能够保持正常无误地运行，且能将响应时间控制在一个能够接受的范围内，因而满足系统对性能的需求，也就代表本系统在高并发环境下能够正常提供服务，具有一定的使用价值。其中，textsearch 接口的响应时间如5.2所示。

表 5.15: 接口性能测试结果表

接口编号	有无错误请求	99% 响应时间
I1	无	137ms
I2	无	133ms
I3	无	140ms
I4	无	121ms

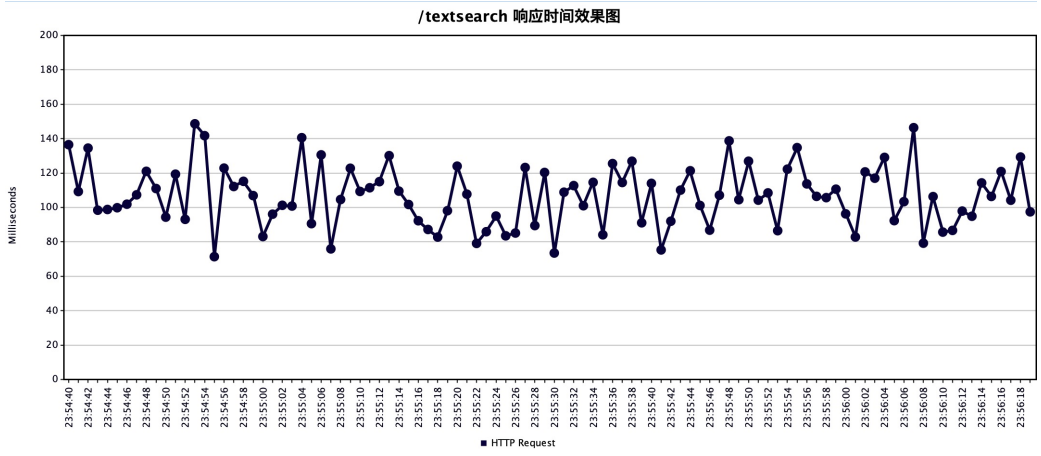

图 5.2: textsearch 响应时间效果图

5.6 模型效果测试

为验证系统选择的模型效果，本文通过设计对比试验，根据实验结果比较在选用模型的情况下实体识别和关系抽取的效果，主要对比指标包括：识别的

精确率、召回率，以及综合召回率和精确率的 F-Socre，其中 F-Score 的计算如公式5.1所示。

$$F-Score = \frac{2 \times 精确率 \times 召回率}{精确率 + 召回率} \tag{5.1}$$

5.6.1 测试设计

对于“实体识别”和“关系抽取”功能，数据采用从知乎、Stack Overflow 和 GitHub 等社区爬取到的包括语言描述、项目介绍、技术讨论等文本数据作为训练集，训练集共包含 14513 个句子，采用复旦知识图谱 CN-DBpedia 提供的数据集经过人工标注后作为本文实验和模型训练的数据集。

在“实体识别”实验，所有训练集分别经过 CRF 模型、LSTM 模型、BLSTM 模型和 CNN-BLSTM-CRF 模型，将得到的结果进行比较。在“关系抽取”实验，经过 CNN 模型、RNN 模型和 PCNN 模型，比较抽取后的结果。

5.6.2 测试执行

表 5.16: 实体识别测试执行效果

模型名称	精确率	召回率	F-Score
CRF	81.33%	72.91%	76.89%
LSTM	83.71%	76.35%	79.86%
BLSTM	85.37%	77.64%	81.32%
CNN-BLSTM-CRF	88.49%	80.84%	84.49%

表 5.17: 关系抽取测试执行效果

模型名称	精确率	召回率	F-Score
CNN	74.63%	59.70%	66.34%
RNN	74.24%	60.19%	66.48%
PCNN	80.92%	62.08%	70.26%

表5.16和表5.17所示是训练集在经过不同模型后得到的指标数值，可以很明显从表中看出本文所选用的 CNN-BLSTM-CRF 模型和 PCNN 模型无论是在精确率和召回率上都有效果提升。

5.7 本章小结

本章节主要对基于知识图谱的开源社区关联性自动化构建系统进行了相关的测试，测试内容为第三章中提到的功能性需求和非功能性需求。对于功能性需求，通过依次执行测试用例确保每一项功能都能正确执行；对于包含可移植性、可用性和性能方面的非功能性需求，其中，可移植性方面验证了系统在主流操作系统以及统信 UOS 和优麒麟两款国产操作系统的运行情况，非功能性需求均达到了预期的目标，可以证明系统具有一定的实用价值。

第六章　总结与展望

6.1　总结

互联网的不断发展使得当下的信息呈爆炸式的增长，其中不免包含真假难辨、浩如烟海且难以统计的信息，这些信息为用户进行筛选带来了不小的麻烦，而在海量的信息中发掘出其间潜在的联系就更为棘手。知识图谱作为语义网络，以其鲜明直观的特点被众多厂商所重视，并不断将知识图谱技术运用到各个领域中去，故本文设计了基于知识图谱的开源社区关联性自动化构建系统，利用知识图谱的特性，依靠实体关系连线的形式直观的展示不同信息 (实体) 之间的联系，并尝试引入外部知识图谱来丰富构建的知识库，从而为关联性构建的准确性更添助力。知识图谱构建需要知识库中大量知识得支持，本文为了有效的构建需要的知识库，选择使用了 Scrapy 爬虫组件，利用其分布式且支持断点续传的特性进行更为高效的原始文本获取。对获取到的数据进行结构化清洗时，在实体抽取部分采用基于 BLSTM-CNNs-CRF 的实体抽取技术来对隐藏在文本中的实体进行发掘与抽取，在关系抽取部分则选择使用 Tensorflow 实现的 PCNN 模型进行发掘与抽取，将抽取到的实体与关系存储到图数据库中，并对外提供文本识别功能，即由用户来输入原始文本进行实体识别，由系统显示识别及分词后的结果。在关联性构建方面，利用知识图谱的特性，用户可以选择根据实体查询直接相关联的实体，或是在两个实体之间通过构建联通路径来发觉潜在关联性，亦或是系统来验证用户指定的关联性是否存在，通过如上的功能来帮助用户更为便捷的找到需要查找的资料的方向，便于用户利用互联网上的庞杂信息。

本文的主要工作如下：首先介绍了知识图谱、实体和关系抽取算法的研究现状进行了调研与阐述。本文对当下现有的知识图谱进行了相关的调研，并就关系抽取算法进行了罗列与比较，最终选择了 PCNN 模型进行关系抽取。使用 KNN 算法来对实体进行分类，虽然 KNN 算法可以不依赖向量，但在不可避免需要词向量的时候选择计算向量间的余弦相似度进行相似度计算。其次，本文为了完成基于知识图谱的开源社区关联性自动化构建系统，将系统分解为文本获取模块、NLP 模块和知识图谱构建模块共计 3 大模块，文本获取模块主要用于分布式从开源社区进行页面爬取，NLP 模块则对爬取到的页面内容进行结构化清洗、构建知识库，知识图谱模块则利用知识库中的知识构建用户需要的知识图谱。在系统实现上，前端使用 Vue 框架，后端部分使用 Spring Boot 框架，有

关知识图谱的部分则使用 Python 语言进行实现，在数据库部分，选择使用 Neo4j 来保存知识图谱所需要的三元组，使用 MongoDB 数据库来存储文本获取模块获取到的原始数据。同时为了保证系统的性能而引入 Redis 缓存，为了保证系统的可用性而使用 Nginx 进行负载均衡，为了保证系统的可移植性而使用 Docker 容器进行部署。紧接着通过详细阐述本系统的具体实现并就系统的功能点、非功能性需求进行详细的测试，最终可以证明本系统是功能完备的且具有良好性能的系统。

6.2 展望

尽管系统已经完成开发并且通过了预期的测试，但系统仍然存在着诸多不足之处需要在日后进行改进，具体问题如下所示：

（1）用户自主性有限。本文所涉及的系统会将抽取到的实体和关系全都存储到数据库中，在需要进行图谱构建时从数据库中查找对应的实体，这也就意味着当数据库中不存在对应实体时，就需要对数据库内容进行扩充。构建知识库后再进行知识图谱构建并不能立刻返回图谱，因而在后续可以尝试扩展用户自主性，例如可以在对开源社区进行文本获取时直接生成知识图谱。

（2）内容获取方式不够多样。开源社区中的内容是复杂的，不仅仅只有文本，还包括代码、公式、图片等内容，因而对于许多内容其实是丢弃的，在未来可以通过引入代码比较、图片识别等功能来改进关联性构建功能。

（3）系统的高可用性。系统在设计之初并未考虑到系统遇到极大并发量的问题，因而在后续当系统需要应对极大并发的问题时，为保证系统的稳定性，可以选择建立分布式数据库，采用主从部署的方式备份数据。

参考文献

[1] HAN J, SHIHAB E, WAN Z, et al. What do programmers discuss about deep learning frameworks[J]. Empirical Software Engineering, 2020, 25(4): 2694–2747.

[2] KENETT Y N, FAUST M. A semantic network cartography of the creative mind[J]. Trends in cognitive sciences, 2019, 23(4): 271–274.

[3] STUDER R, BENJAMINS V R, FENSEL D. Knowledge engineering: principles and methods[J]. Data & knowledge engineering, 1998, 25(1-2): 161–197.

[4] WANG Q, MAO Z, WANG B, et al. Knowledge graph embedding: A survey of approaches and applications[J]. IEEE Transactions on Knowledge and Data Engineering, 2017, 29(12): 2724–2743.

[5] ETZIONI O, BANKO M, SODERLAND S, et al. Open information extraction from the web[J]. Communications of the ACM, 2008, 51(12): 68–74.

[6] 冯志伟. 自然语言的计算机处理 [J]. 中文信息, 1997(4): 26–27.

[7] 赵京胜, 宋梦雪, 高祥. 自然语言处理发展及应用综述 [J]. 信息技术与信息化, 2019(7): 142–145.

[8] 孙紫阳, 顾君忠, 杨静. 基于深度学习的中文实体关系抽取方法 [J]. 计算机工程, 2018, 44(9): 164–170.

[9] MILLER G A. WordNet: a lexical database for English[J]. Communications of the ACM, 1995, 38(11): 39–41.

[10] BIZER C, LEHMANN J, KOBILAROV G, et al. Dbpedia-a crystallization point for the web of data[J]. Journal of web semantics, 2009, 7(3): 154–165.

[11] SUCHANEK F M, KASNECI G, WEIKUM G. Yago: A large ontology from wikipedia and wordnet[J]. Journal of Web Semantics, 2008, 6(3): 203–217.

[12] DONG X, GABRILOVICH E, HEITZ G, et al. Knowledge vault: A web-scale approach to probabilistic knowledge fusion[C] // Proceedings of the 20th ACM

SIGKDD international conference on Knowledge discovery and data mining. 2014 : 601 – 610.

[13] NIU X, SUN X, WANG H, et al. Zhishi. me-weaving chinese linking open data[C] // International Semantic Web Conference. 2011 : 205 – 220.

[14] JIA Y, WANG Y, CHENG X, et al. OpenKN: An open knowledge computational engine for network big data[C] // 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014). 2014 : 657 – 664.

[15] XU B, XU Y, LIANG J, et al. CN-DBpedia: A never-ending Chinese knowledge extraction system[C] // International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. 2017 : 428 – 438.

[16] 杭婷婷, 冯钧, 陆佳民. 知识图谱构建技术: 分类, 调查和未来方向 [J]. 计算机科学, 2021, 48(2) : 175 – 189.

[17] GUARINO N. Review of Knowledge Representation: Logical, Philosophical, and Computational Foundations[J]. AI Magazine, 2001, 22(3) : 123 – 123.

[18] SHI B, WENINGER T. ProjE: Embedding projection for knowledge graph completion[C] // Proceedings of the AAAI Conference on Artificial Intelligence : Vol 31. 2017.

[19] 秦川, 祝恒书, 庄福振, et al. 基于知识图谱的推荐系统研究综述 [J]. 中国科学: 信息科学, 2020, 50(7) : 937 – 956.

[20] WANG X, HE X, CAO Y, et al. Kgat: Knowledge graph attention network for recommendation[C] // Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019 : 950 – 958.

[21] SONG Q, WU Y, LIN P, et al. Mining summaries for knowledge graph search[J]. IEEE Transactions on Knowledge and Data Engineering, 2018, 30(10) : 1887 – 1900.

[22] ABADI M, BARHAM P, CHEN J, et al. Tensorflow: A system for large-scale machine learning[C] // 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16). 2016 : 265 – 283.

[23] ZHANG J, LIU F, XU W, et al. Feature fusion text classification model combining CNN and BiGRU with multi-attention mechanism[J]. Future Internet, 2019, 11(11) : 237.

[24] ZHANG Z, CAI J, ZHANG Y, et al. Learning hierarchy-aware knowledge graph embeddings for link prediction[C] // Proceedings of the AAAI Conference on Artificial Intelligence : Vol 34. 2020 : 3065 – 3072.

[25] RAU L F. Extracting company names from text[C] // Proceedings the Seventh IEEE Conference on Artificial Intelligence Application. 1991 : 29 – 30.

[26] 鄂海红, 张文静, 肖思琪, et al. 深度学习实体关系抽取研究综述 [J]. 软件学报, 2019, 30(6) : 1793 – 1818.

[27] RATNAPARKHI A, OTHERS. A maximum entropy model for part-of-speech tagging.[C] // EMNLP : Vol 1. 1996 : 133 – 142.

[28] BERGER A, DELLA PIETRA S A, DELLA PIETRA V J. A maximum entropy approach to natural language processing[J]. Computational linguistics, 1996, 22(1) : 39 – 71.

[29] SEYMORE K, MCCALLUM A, ROSENFELD R, et al. Learning hidden Markov model structure for information extraction[C] // AAAI-99 workshop on machine learning for information extraction. 1999 : 37 – 42.

[30] YAO K, PENG B, ZWEIG G, et al. Recurrent conditional random field for language understanding[C] // 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2014 : 4077 – 4081.

[31] 周飞燕, 金林鹏, 董军, et al. 卷积神经网络研究综述 [J]. 计算机学报, 2017, 40(6) : 1229 – 1251.

[32] MIKOLOV T, KARAFIÁT M, BURGET L, et al. Recurrent neural network based language model.[C] // Interspeech : Vol 2. 2010 : 1045 – 1048.

[33] KIM P. Convolutional neural network[G] // MATLAB deep learning. [S.l.] : Springer, 2017 : 121 – 147.

[34] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural computation, 1997, 9(8) : 1735 – 1780.

[35] RAY A, RAJESWAR S, CHAUDHURY S. Text recognition using deep BLSTM networks[C] // 2015 eighth international conference on advances in pattern recognition (ICAPR). 2015 : 1 – 6.

[36] MA X, HOVY E. End-to-end sequence labeling via bi-directional lstm-cnns-crf[J]. arXiv preprint arXiv:1603.01354, 2016.

[37] CHIU J P, NICHOLS E. Named entity recognition with bidirectional LSTM-CNNs[J]. Transactions of the association for computational linguistics, 2016, 4 : 357 – 370.

[38] 黄勋, 游宏梁, 于洋. 关系抽取技术研究综述 [J]. 现代图书情报技术, 2013(11) : 30 – 39.

[39] LIU C, SUN W, CHAO W, et al. Convolution neural network for relation extraction[C] // International conference on advanced data mining and applications. 2013 : 231 – 242.

[40] ZENG D, LIU K, LAI S, et al. Relation classification via convolutional deep neural network[C] // Proceedings of COLING 2014, the 25th international conference on computational linguistics: technical papers. 2014 : 2335 – 2344.

[41] NGUYEN T H, GRISHMAN R. Relation extraction: Perspective from convolutional neural networks[C] // Proceedings of the 1st workshop on vector space modeling for natural language processing. 2015 : 39 – 48.

[42] WANG L, CAO Z, DE MELO G, et al. Relation classification via multi-level attention cnns[C] // Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2016 : 1298 – 1307.

[43] ZHU J, QIAO J, DAI X, et al. Relation classification via target-concentrated attention cnns[C] // International Conference on Neural Information Processing. 2017 : 137 – 146.

[44] ZENG D, LIU K, CHEN Y, et al. Distant supervision for relation extraction via piecewise convolutional neural networks[C] // Proceedings of the 2015 conference on empirical methods in natural language processing. 2015 : 1753 – 1762.

[45] 刘伟, 陈鸿昶, 黄瑞阳. 基于 Tree-based CNN 的关系抽取 [J]. 中文信息学报, 2018, 32(11) : 34 – 40.

[46] 刘熙, 胡志勇. 基于 Docker 容器的 Web 集群设计与实现 [J]. 电子设计工程, 2016, 24(8) : 117 – 119.

[47] BOETTIGER C. An introduction to Docker for reproducible research[J]. ACM SIGOPS Operating Systems Review, 2015, 49(1) : 71 – 79.

[48] RAHARTOMO A, AJI R F, RULDEVIYANI Y. The application of big data using mongodb: Case study with scele fasilkom UI forum data[C] // 2016 International Workshop on Big Data and Information Security (IWBIS). 2016 : 51 – 56.

[49] HOLZSCHUHER F, PEINL R. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j[C] // Proceedings of the Joint EDBT/ICDT 2013 Workshops. 2013 : 195 – 204.

[50] SHUKLA N, FRICKLAS K. Machine learning with TensorFlow[M]. [S.l.] : Manning Greenwich, 2018 : 189 – 199.

简历与科研成果

基本情况 李成浩，男，汉族，1998 年 3 月出生，江苏省泰州市人

教育背景

2020.9-2022.6	南京大学软件学院	硕士
2016.9-2020.6	湖南工业大学理学院	本科

致　　谢

两载春秋，匆匆而逝，在这论文即将完成之际，借此向所有帮助过我的亲友们献上我最真挚的祝福。

首先，最需要感谢的是我的导师陈振宇老师给了我求学的机会，让我能够有幸成为 ISE 实验室的一员，在这个大家庭里我获得了许多学习与磨炼的机会，也正是得益于在 ISE 实验室的两年，让我对软件工程专业有了更深的认识。

同时，还要感谢刘嘉老师和沈厚才老师在我迷茫无措时的指引，感谢刘佳玮学姐在我论文撰写过程中给予的诸多帮助。

此外，我要感谢南京大学软件学院对我的历练，让我得以完成从数学系到软件工程专业的转变与适应，让我从一窍不通的门外汉成长为一个可以独立参与开发的软件工程师。

感谢我的父母在我读研期间再次承担了我的学费和生活费，以及他们对我这二十多年的照顾与支持，感谢我的室友对我的论文提出了许多重要的意见，感谢管耀鹏老师给予的极大帮助。

我还要感谢咸诗宇同学在我颓废时给予的支持，两年的求学时光总不可避免会害怕、畏惧，是咸诗宇同学一直在认可我，鼓励我，相信我可以做好这一切。

最后，我向所有百忙之中抽空进行审阅的各位专家献上由衷的感谢。

版权与原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期：2022 年 05 月 20 日