
南京大學

# 研 究 生 毕 业 论 文

## （申请工程硕士学位）

| | |
|---|---|
| **论 文 题 目** | 面向 Bugzilla 的知识图谱问答系统 |
| **作 者 姓 名** | 钱雨波 |
| **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** | 刘嘉 副教授 何铁科 助理研究员 |

**2022 年 05 月 20 日**

学　　号：**MF20320111**

论文答辩日期：**2022 年 05 月 20 日**

指 导 教 师：　　**（签字）**


# Question Answering System Based On Knowledge Graph For Bugzilla

By

**Qian Yubo**

Supervised by

Associate Professor **Liu Jia**

Research Assistant **He Tieke**

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

# 学 位 论 文 原 创 性 声 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，本论文不句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。

论文作者签名：_____

日期：　　年　　月　　日

# 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：面向 Bugzilla 的知识图谱问答系统

工程硕士（软件工程领域）专业 2020 级硕士生姓名：钱雨波

指导教师（姓名、职称）：刘嘉 副教授 何铁科 助理研究员

## 摘 要

随着互联网技术的快速普及和蓬勃发展，软件产品的质量问题日益突出。为保证项目开发和迭代的顺利进行，缺陷跟踪系统应运而生。然而，复杂缺陷库不可避免地存在上手门槛高、学习成本大等现实问题，在面向众包测试等大规模人工协作或交接场景尤为严重。

问答技术是信息检索的高级形式，能够快速且准确地对输入问题作出反馈，节约用户排查和筛选的时间。复杂自然语言问题往往具有多跳语义，图结构对于还原三元组路径具有更强的可解释性，逻辑查询更加清晰直观，图算法不会随规模扩大而降低检索速度，形式化查询效率更高。

基于上述背景，本文设计并实现面向 Bugzilla 的知识图谱问答系统，包含原始数据清洗、语义模型训练、知识图谱构建、缺陷报告管理和问答系统开发五个核心模块。本系统支持缺陷报告查重和句式规则匹配两种问答形式。针对缺陷报告查重问答，采用语义相似度匹配技术，在完成原始数据清洗后，根据停用词表和无效词性表对缺陷摘要数据集进行语义压缩处理，再利用 SBERT 和 Faiss 预训练得到句嵌入索引模型。针对句式规则匹配问答，采用模式规则分发技术，预标注常见句型模板和特征词白名单，使用正则表达式将非标问题映射为标准问题，进而构建形式化查询。

本系统是前后端分离的跨平台桌面应用，各服务虚拟化部署于 Docker 容器。前端采用 React+TypeScript 搭建 Hooks 函数编程框架，Antd+Less 达成模块化样式布局，Echarts 绘制可视化图表；后端选用 Django 框架并调用第三方 API 接口实现混合编程；使用 Neo4j 持久化知识图谱、MySQL 保存待审批缺陷库数据；利用 Nginx 处理负载均衡和反向代理，Redis 缓存热点数据，从而提高系统的可用性、可靠性和稳定性。

系统相关软件测试用例均成功通过，测试覆盖率满足 Pareto 法则，核心业务全链路跑通。通过静态资源压缩和热点缓存机制，首屏渲染时长控制在 2.5 秒以内，每秒查询率提升 2.81 倍。

**关键词**：领域场景；知识图谱；问答系统；SBERT；混合问答框架

# 南京大学研究生毕业论文英文摘要首页用纸

THESIS: Question Answering System Based On Knowledge Graph For Bugzilla

SPECIALIZATION: Software Engineering

POSTGRADUATE: Qian Yubo

MENTOR: Associate Professor **Liu Jia** Research Assistant **He Tieke**

## Abstract

With the fast promotion and fiery improvement of Internet innovation, the quality problem of programming items has become progressively unmistakable. To guarantee the smooth advancement of task improvement and emphasis, the defect tracking system appeared. Nonetheless, complex defect libraries unavoidably have functional issues, for example, high threshold and high learning costs, which are especially serious in large-scale manual collaboration or handover scenarios such as crowdsourced testing.

Question answering technology is an advanced form of information retrieval, which can rapidly and precisely respond to user questions, saving users' time for investigation and screening. Complex natural language questions often have multi-hop semantics. The graph structure has more grounded interpretability in reestablishing triplet networks, and the legitimate structure inquiry is all the more clear and natural. The graph algorithm won't diminish the question speed with the extension of the scale, and the formal inquiry effectiveness higher.

In view of the above foundation, the thesis designs and implements a question answering system based on knowledge graph for Bugzilla, which incorporates five core modules: original data cleaning, semantic model training, knowledge graph construction, defect report management and question answering system development. The system supports two question answering forms: defect report duplication checking and sentence pattern rule matching. Aiming at the question answering of duplicate checking of defect reports, the semantic similarity matching technology is used. After cleaning the original data, the defect summary data set is semantically compressed according to the stop words and invalid part of speech, and then the index model of sentence embedding is obtained by pre-training with SBERT and Faiss. For sentence pattern matching

question answering, pattern rule distribution technology is adopted, common sentence pattern templates and feature word whitelist are pre-marked, and regular matching is used to convert non-standard questions into standard questions, and then construct formal queries.

This system is a cross-end desktop application with frontend and backend separated, and is deployed in Docker container in virtualization. The frontend uses React+TypeScript to build the Hooks functional programming framework, Antd+Less achieves style modularization, and Echarts draws visual graphics; the backend uses the Django framework and calls third-party API interfaces to realize hybrid programming; uses Neo4j to persist knowledge graph and MySQL to save pending approval Defect database data; Nginx is used to provide load balancing and reverse proxy services, and Redis caches hot data, thereby improving the availability, reliability and stability of the system.

The system-related software test cases were successfully passed, the test coverage complied with Pareto's principle, and the whole core business link connected. Through static resource compression and hotspot caching mechanism, the rendering time of the first screen is controlled within 2.5 seconds, and the query per second is increased by 2.81 times.

**Keywords:** Domain Scene, Knowledge Graph, Question Answering System, SBERT, Hybrid Question Answering Framework

# 目　录

**表 目 录** ····· **viii**

**图 目 录** ····· **x**

**第一章　引言** ····· **1**

1.1　研究背景与意义 ····· 1

1.2　国内外研究现状 ····· 2

1.3　主要工作 ····· 4

1.4　组织结构 ····· 5

**第二章　技术综述** ····· **7**

2.1　数据处理相关 ····· 7

2.1.1　数据清洗 ····· 7

2.1.2　知识图谱 ····· 7

2.2　文本预处理相关 ····· 9

2.2.1　词性标注 ····· 9

2.2.2　语义相似度模型 ····· 9

2.2.3　近似最近邻检索 ····· 11

2.3　系统研发相关 ····· 11

2.3.1　客户端技术 ····· 11

2.3.2　服务端技术 ····· 13

2.3.3　工程化工具 ····· 13

2.3.4　架构技术 ····· 14

2.4　本章小结 ····· 15

**第三章　需求分析与概要设计** ····· **17**

3.1　系统整体概述 ····· 17

3.2　系统需求分析 ····· 18

3.2.1 涉众分析 …… 18
3.2.2 功能性需求 …… 19
3.2.3 非功能性需求 …… 20
3.2.4 系统用例描述 …… 20
3.3 系统总体设计 …… 25
3.3.1 总体架构 …… 25
3.3.2 4+1 视图 …… 27
3.4 数据处理与持久化设计 …… 32
3.4.1 数据预处理 …… 32
3.4.2 文本语义识别 …… 35
3.4.3 本体层设计 …… 38
3.4.4 持久化存储 …… 41
3.5 本章小结 …… 43

**第四章 详细设计与编码实现 …… 44**
4.1 原始数据清洗模块 …… 44
4.1.1 详细设计 …… 44
4.1.2 核心代码 …… 48
4.1.3 结果说明 …… 51
4.2 语义模型训练模块 …… 52
4.2.1 详细设计 …… 52
4.2.2 核心代码 …… 55
4.2.3 效果说明 …… 57
4.3 知识图谱构建模块 …… 59
4.3.1 详细设计 …… 59
4.3.2 核心代码 …… 62
4.3.3 效果说明 …… 63
4.4 缺陷报告管理模块 …… 64
4.4.1 详细设计 …… 64
4.4.2 核心代码 …… 67
4.4.3 效果说明 …… 68

4.5 问答系统开发模块 …… 68
4.5.1 详细设计 …… 68
4.5.2 核心代码 …… 71
4.5.3 效果说明 …… 72
4.6 本章小结 …… 74

**第五章 系统测试与结果分析 …… 75**
5.1 测试准备 …… 75
5.1.1 测试目标 …… 75
5.1.2 测试环境 …… 76
5.2 单元测试 …… 77
5.3 接口测试 …… 78
5.4 功能测试 …… 79
5.5 性能测试 …… 82
5.6 本章小结 …… 83

**第六章 总结与展望 …… 84**
6.1 总结 …… 84
6.2 展望 …… 85

**参考文献 …… 86**

**简历与科研成果 …… 93**

**致谢 …… 94**

# 表 目 录

3-1 不同类型的问答系统对比 …… 17
3-2 涉众分析表 …… 18
3-3 系统功能需求表 …… 19
3-4 用例需求对应表 …… 22
3-5 源数据清洗用例描述表 …… 22
3-6 句嵌入预训练用例描述表 …… 23
3-7 知识图谱构建用例描述表 …… 23
3-8 缺陷报告管理用例描述表 …… 24
3-9 缺陷库管理用例描述表 …… 24
3-10 缺陷库问答用例描述表 …… 25
3-11 标识型元素说明 …… 33
3-12 基础型元素说明 …… 33
3-13 角色型元素说明 …… 34
3-14 描述型元素说明 …… 34
3-15 状态型元素说明 …… 35
3-16 附加型元素说明 …… 35
3-17 标准问题模式匹配规则 …… 38
3-18 三元组定义 …… 40
3-19 Neo4j 实体文件表 …… 41
3-20 Neo4j 关系文件表 …… 41
3-21 表 user 结构设计 …… 41
3-22 表 report 结构设计 …… 42
3-23 表 report2user 结构设计 …… 42
3-24 表 report2report 结构设计 …… 42

4-1 源文件数据清洗前后对比 …… 51
4-2 模型评价指标结果 …… 58

5-1 系统测试环境 …… 76
5-2 软件版本信息 …… 77
5-3 单元测试用例执行单 …… 78
5-4 接口测试用例执行单 …… 79
5-5 缺陷报告填写测试用例执行单 …… 79
5-6 缺陷报告审批测试用例执行单 …… 80
5-7 实体关系查询测试用例执行单 …… 80
5-8 缺陷报告查重测试用例执行单 …… 81
5-9 句式规则匹配测试用例执行单 …… 81
5-10 缓存压力测试对比表 …… 82
5-11 首屏时长测试对比表 …… 82

# 图 目 录


2.1 知识图谱应用概览 …… 8
2.2 BERT 核心结构 …… 9
2.3 SBERT 核心算法图 …… 10
2.4 WarpSelect 概述 …… 11
2.5 Redis 内置数据类型关系图 …… 15

3.1 总体结构图 …… 17
3.2 系统用例图 …… 21
3.3 系统架构图 …… 26
3.4 4+1 视图模型 …… 27
3.5 逻辑视图 …… 28
3.6 进程视图 …… 29
3.7 开发视图 …… 30
3.8 物理视图 …… 31
3.9 缺陷报告源数据 infos 字段示例 …… 32
3.10 三种模式依赖图 …… 36
3.11 Penn Treebank 停用词性 …… 36
3.12 停用词表示意图 …… 37
3.13 关键词白名单示意图 …… 37
3.14 模式数据概念图 …… 38
3.15 模式数据实例图 …… 39

4.1 原始数据清洗活动图 …… 45
4.2 原始数据清洗时序图 …… 46
4.3 源文件数据清洗文件操作代码 …… 48
4.4 源文件数据清洗核心代码 …… 49
4.5 目标实体文件生成代码 …… 50
4.6 目标关系文件生成代码 …… 51

4.7 语义压缩示意图 …… 52
4.8 语义模型训练活动图 …… 53
4.9 语义模型训练时序图 …… 54
4.10 去停用词代码 …… 56
4.11 句嵌入代码 …… 56
4.12 索引模型构建代码 …… 57
4.13 模型评价标准和指标示意图 …… 57
4.14 知识图谱构建活动图 …… 59
4.15 知识图谱构建时序图 …… 60
4.16 创建实体代码 …… 62
4.17 创建关系代码 …… 63
4.18 知识图谱构建链路图 …… 63
4.19 Neo4j 数据库信息示意图 …… 64
4.20 缺陷报告管理活动图 …… 65
4.21 缺陷报告管理时序图 …… 66
4.22 异步网络通信代码 …… 67
4.23 axios 单例模式代码 …… 67
4.24 缺陷报告填写界面 …… 68
4.25 问答系统开发活动图 …… 69
4.26 问答系统开发时序图 …… 70
4.27 模式匹配分发规则代码 …… 72
4.28 核心链路图 …… 72
4.29 缺陷报告查重界面 …… 73
4.30 自然语言问答界面 …… 73

5.1 软件测试流程图 …… 77

# 第一章　引言

## 1.1　研究背景与意义

由于复杂性、一致性、易变性和隐匿性 [1] 是现代软件系统无法规避的固有特性，故而软件开发“没有银弹”。“焦油坑”使软件系统或多或少都存在缺陷/漏洞（Bug），因此提升软件质量是永无终点的旅程 [2]。互联网发展带来软件发布和管理方式的颠覆性变革，开源社区应运而生。大部分缺陷都是可复现的 [3]，为避免解决或修复相同缺陷造成的重复性工作，具有开源精神的程序员将缺陷报告上报并存储于缺陷跟踪管理系统（Bug-Tracking System）。互联网公司内部也都具有负责缺陷追踪的管理系统，主要用于提升缺陷修复和管理的效率，保障整个敏捷开发生命周期的软件迭代质量，如 Atlassian 的 JIRA、阿里巴巴的 Aone 和 Google 的 Buganizer 等。开源免费的缺陷追踪管理系统以 Mozilla Foundation 的 Bugzilla 知名度最高。

问答系统（Question Answering System，QA）集知识表示、语义识别、信息检索和智能推理等技术于一体，能准确且简练地回答用户以自然语言提出的问题，满足用户对于快速、准确地获取信息的需求。随着人机交互技术的发展，问答系统在软件工程的诸多领域广泛落地。相较于传统的检索系统或搜索引擎仅利用信息检索，反馈结果不够清晰直观，问答系统是自然语言处理和信息检索的超集，按序执行知识获取、问题分析、信息检索和答案抽取等步骤，能识别用户意图并应答复杂提问。一个健壮鲁棒的基于知识库的问答系统，通过解析自然语言识别用户意图，根据问题分析结果缩小答案范围，最后从潜在的信息或数据中提炼最佳答案。

知识图谱是一种基于图的结构化语义知识库，将杂乱无章的信息有效地组织为相互关联的知识，用以形式化地描述现实世界的概念及其相互间的关系。知识图谱具有强大的语义处理能力与开放互联能力且查询速度不会因规模扩大而变慢。相比于搜索式问答系统，知识图谱的时间成本更低；相比于匹配对问答系统，知识图谱的空间成本更低。

以缺陷库为代表的缺陷追踪应用场景存在以下问题，针对这些难点基于知识图谱的问答系统给出相应的解决方案：

1. 新用户使用缺陷库门槛较高，对诸如众包测试等大规模人工协作或交接场景以及非项目强相关人员不友好，而问答系统能简化流程，帮助新用户快速

上手；

2. 非知识图谱构建的问答系统存在不精准或开销太大等问题，而基于图的算法能准确且高效地检索答案，通过多元交叉验证数据的质量，对还原复杂问题的语义有强解释性，促进人机互信；

3. 通用问答系统框架不能直接迁移并适配于专业化程度突出的问答场景，限定领域问答往往需要特征识别预处理和模式规则预标注，而良好的本体归纳和三元组定义能有效识别特征实体、准确匹配模式规则。

综上所述，开放领域利用一般本体和世界知识，限定领域利用特定本体和术语，通用问答框架应用于特殊场景往往表现不佳。如果有一个基于定义良好的特定本体层的知识图谱，就能够快速将缺陷库中聚集的大量专业术语、产品信息、缺陷描述以及用户行为有效地组织起来。相较于信息检索或者推荐系统，问答系统可以更快、更准、更清晰地回答某个具体问题，解决面向 Bugzilla 场景的自动问答痛点，节约用户大量检索和查阅时间，减少缺陷报告重复提交等问题的发生，尤其是在帮助研发人员（如开发、测试等）以及非专业人员（如项目经理、运营等）或非直接项目成员（如管理者、目标客户等）快速了解和融入新项目有巨大价值。

## 1.2 国内外研究现状

知识图谱（Knowledge Graph，KG）是一系列相互关联的实体及其属性构成的集合 [4]。从宏观层面，知识图谱的诞生和成长得益于互联网（Web）的快速发展和数据的爆炸式增长。从微观层面，知识图谱是知识表示（Knowledge Represention，KR）、自然语言处理（Natural Language Processing，NLP）和机器学习（Machine Learning，ML）等多领域交叉学科的融合。

二十世纪六十年代，用图来表示知识的结构化方法 —— 语义网络（Semantic Networks） [5] 被提出，这是自然语言处理范畴内知识表达的一种形式。语义网络的节点表示信息，不同节点通过带标记的有向线段相连，代表节点之间的关系。语义网络的发明，使用户更容易地理解节点的语义和节点间的语义关系。但早期语义网络缺乏统一的标准，在工业界难以落地。

1980s，本体（ontology） [6] 这一哲学概念被引入到人工智能领域。本体作为组成万物的最基本元素，最早的研究可追溯到古希腊时期。在知识图谱领域，一个定义良好的本体层是评估知识图谱优劣的决定性因素。

万维网（World Wide Web）之父 Berners-Lee [7] 在 1989 年提出全新的知识组织形式，即超文本链接。这是一种全局性的信息结构，使信息能通过交互的

方式得到检索。在文本的基础上，随后又发展出包含语音、图像和视频等多模态数据资源，极大程度上促进互联网（Web）的高速发展。

在万维网的基础上，Berners-Lee 逐渐意识到计算机语义理解的重要性，并于 1998 年提出语义网（Semantic Web） [8, 9]，实现从超链接到语义链接的跨越。语义网用更丰富的组织方式来表达数据深层次的含义，赋予机器理解网页元数据的能力，此后互联网成为通用的信息交换媒介。

Linked Open Data Project [10] 在 2007 年号召创建互联网跨源语义互联，以关联数据（Linked Data）作为语义网的表达形式，开创三元组描述信息数据的先河，截止 2018 年已收纳 2973 个数据集，总计包含大约 1494 亿个三元组。

2012 年谷歌发布第一款基于知识图谱的搜索引擎产品，正式诞生 Knowledge Graph 的概念 [11]。这是一种较以往更为复杂的结构化数据，用于处理自然语言查询，解决传统检索技术无法理解语义和无法精准问答的核心诉求。

从此，知识图谱成为人工智能热点研究方向，2007 年至 2017 年间 [12] 大型开源知识图谱的数量从 12 增长为 1139。Gaur [13] 等人提出符号化的知识图谱，弥补深度学习在可解释性方面的缺陷。Li [14] 等人针对社会基金构建领域知识图谱，为公众监督其资金流转提供交互式可视化数据访问服务。Yuan [15] 等人提出一种基于非结构化生物医学领域特定上下文的最小监督知识图构建的通用方法，高精度地提取上万条结构化事实。

搜索引擎和检索系统通过综合考量受欢迎程度、关键词匹配、访问频率等多维度指标呈现相关链接的有序列表，但是它们并没有真正完成用户的信息检索任务。而问答系统能够快速查找或推断自然语言问题的答案 [16]，帮助用户简化检索信息流程和精准定位最终结果，即更快速地识别问题、更准确地定位答案、更清晰地反馈结果。

问答系统最早的设想是 1950 年英国数学家 Turing 提出的图灵测试 [17]。测试群体 A 分别与机器 B 和人类 C 多轮对话，据此验证机器能否思考，若机器 B 平均能让每个参与者做出的误判超过 30%，则认为具有思维能力和应答能力。

1966 年麻省理工学院的 Jaseph Weizenbaum 教授采用关键词匹配的方法实现史上第一个 QA 系统 Eliza [18]。这个时期的问答系统主要面向限定领域或结构化数据，典型代表是棒球领域的 BASEBALL [19] 和地质领域的 LUNAR [20]。

Terry Winograd 和 Bobrow 分别在 1971 年和 1977 年开发的 SHRDLU [21]（积木游戏）和 GUS [22]（旅行咨询）属于多轮对话问答系统。20 世纪 70 年代开始出现以耶鲁大学人工智能实验室 SAM [23] 为代表的具备阅读理解能力的问答系统。

美国麻省理工学院人工智能实验室在 1993 年推出第一个基于 Web 的通用问答系统 START[1]，该系统面向开放数据集，基于知识库和信息检索，能够回答地理、电影、人物等跨领域英文问题。1995 年 Richard 开发设计第一个聊天机器人 ALICE [24]。1999 年 TREC [25] 的 QA track 设立，极大地推动问答系统的发展。

随着自然语言处理和检索技术的发展，一系列智能问答系统如雨后春笋般涌现，国外知名的有 IBM Watson [26]、Apply Siri [27] 和 Microsoft Cortana [28] 等，国内知名的有百度小度 [29]、小爱同学 [30]、天猫精灵 [31] 和复旦大学 FDUQA [32] 等。

目前智能问答系统涉及的关键技术包括原始数据的知识抽取与结构表示，用户问题的意图识别与语义理解，推理结果的范围锁定和内容生成。Huang [33] 等人提出一个基于知识嵌入的问答（KEQA）框架，专注于回答最常见的问题类型。Sawant [34] 等人提出基于知识图谱和语料库的问答系统，将问题转换为结构化查询，通过多种卷积网络的组合信号，对实体识别进行评分。Dubey [35] 等人提出双策略问答框架，执行实体链接和关系链接的联合任务，利用知识图谱节点间的连接密度。Saffari [36] 等人提出一种适用于弱监督数据集的端到端预训练模型，将基于知识图谱问答系统的学习边界扩展到包括实体训练和解析组件。

## 1.3 主要工作

由于软件开发的固有特性，计算机程序系统存在软件缺陷是不可避免的。因此，保障软件质量是敏捷开发的必然要求。以 Bugzilla 为代表的缺陷管理系统在软件缺陷的追踪和修复过程中扮演重要角色。随着人们对于更快速、更准确和更清晰的信息检索需求，问答技术发展迅猛。相较于其他形式的问答技术，基于知识图谱的问答技术在空间复杂度和时间复杂度上均有较好的表现，对于复杂问题的语义识别具有更强的可解释性。

本项目是面向 Bugzilla 场景的基于知识图谱的问答系统，目的是解决特殊专业领域的自动问答痛点，涉及的主要工作包括：

（1）背景现状调研，结合知识图谱和问答系统的国内外研究现状，对项目进行涉众分析和需求分析。按照项目需求，宏观上将系统功能划分为原始数据清洗模块、语义模型训练模块、知识图谱构建模块、缺陷报告管理模块和问答系统开发模块。

[1] http://start.csail.mit.edu/index.php

（2）相关技术选型，结合当前最前沿的技术和最新的设计理念，深入理解其底层原理，充分权衡技术框架的稳定性、可用性和可扩展性。根据技术栈作用域划分为数据处理相关、文本预处理相关和系统研发相关。

（3）语义模型训练，主要涉及缺陷报告查重（缺陷标识获取）。计算缺陷摘要的句向量，在句嵌入结果的基础上构建最邻近索引作为模型的召回，通过比较文本相似性，快速检索向量库中与输入向量最相似或者距离最近的合法缺陷摘要。

（4）知识图谱构建，本体归纳和三元组定义。本体用于数据约束和实体识别，描述知识的结构。三元组是知识图谱的基本组成单位，包括两个节点及其之间的关系。本体层和三元组是知识图谱最底层、原子级别的组成元素，是蕴含知识的数据结构。

（5）问答系统开发，基于模式匹配规则，根据预设的常见句式模板对输入文本进行识别和分发处理。模式匹配是针对字符串数据的一种基本运算，对于给定的非标准问题，根据是否符合正则匹配式，判断其能否转换为标准问题，进而转发至相关处理函数。

（6）关键链路测试，涵盖单元测试、接口测试、功能测试和性能测试等内容。验证系统的可用性、可靠性和稳定性，确保系统核心服务正常运行，满足既定的项目需求和业务功能。

## 1.4 组织结构

本文的组织结构如下：

第一章为引言部分。本章首先阐述面向缺陷库场景的知识图谱问答系统的选题背景及意义；其次，概述国内外知识图谱和问答系统的发展历程及现状；再次，从全局角度归纳本文的主要工作；最后，总结概括全文的组织结构。

第二章为相关技术综述。本章主要介绍本文系统核心链路涉及的数据处理、文本预处理、系统研发相关的技术栈。完成数据清洗、知识图谱、词性标注、语义相似度模型、近似最近邻检索、客户端技术、服务端技术、工程化工具和架构技术等具体任务的技术选型。

第三章为需求分析与概要设计。本章首先从涉众分析、功能性需求、非功能性需求和系统用例视角描述目标系统的需求分析。然后从整体层面出发，依托系统架构图、“4+1”视图对系统进行总体设计。最后设计和说明领域场景下语义识别的思路、问题分发的策略、知识图谱建模的原则和多模态数据持久化的方法。

第四章为详细设计与编码实现。本章将系统划分为五个主要模块，即原始数据清洗模块、语义模型训练模块、知识图谱构建模块、缺陷报告管理模块和问答系统开发模块。分别从详细设计、核心代码和效果说明三个维度来论述。详细设计涉及模块的期望目标，以活动图、时序图和伪代码算法的方式进行详细说明；核心代码是伪代码算法的可编译源代码；模块最终效果通过评估指标、数据可视化和系统界面等形式展示。

第五章为系统测试与结果分析。本章围绕测试目标开展单元测试、接口测试、功能测试和性能测试，对测试结果进行分析。通过测试用例执行单和对比实验的方式说明测试思路、测试方案和测试流程，从可用性、可靠性和稳定性等角度核验系统的软件质量。

第六章为总结与展望。本章总结面向 Bugzilla 的知识图谱问答系统相关工作的完成情况，针对系统存在的问题和不足进行说明，对缺陷库领域场景的问答系统未来优化方向作出展望。

# 第二章　技术综述

## 2.1　数据处理相关

### 2.1.1　数据清洗

真实场景下的数据通常是不一致的、不规范的和不完整的，即有噪声的。因此，不可避免地存在冗余、缺失、矛盾和模糊等一系列问题 [37]。数据预处理技术的目标是对原始数据进行必要的清洗、集成、变换和规约，使之在知识获取阶段达到数据质量标准 [38]。

数据清洗是检测目标数据集是否存在不符合期望的异常数据并加以修复，从而提高数据质量的完整过程 [39]。异常数据覆盖拼写错误、不合法值、空值、上下文矛盾和重复项等多种情况，故亦称为脏数据。数据清洗利用数理统计、数据挖掘或预定义的清理规则将脏数据加工至满足质量要求的数据，广泛应用于数据仓库、机器学习、自然语言处理和知识图谱等领域的预处理阶段 [40]。本系统的数据源自网络爬虫，存在冗余嵌套、空值、不合法值等诸多问题，因此需要提前完成数据清洗。

数据清洗包括且不限于数据去重、数据补全、数据整理等步骤。

1. 数据去重简而言之就是删除重复数据，从某种意义上说也是一种数据压缩技术。重复值清洗一般包括：单一变量去重和多变量去重。

2. 数据补全是数据清洗领域研究的主要问题之一，对极端情况的特判处理。不完整的数据会严重影响后续工作的鲁棒性和完备性。缺失值清洗一般包括：忽略和填充。

3. 数据整理是根据具体的研究任务和要求，对搜集到的经过处理后的数据进行汇总和整合，使之条理化、结构化和系统化。数据整理一般包括：规约化和格式化。

### 2.1.2　知识图谱

知识图谱本质上是一种揭示实体与实体之间关系的语义网络 [41]，即一个基于有向图结构的知识库。知识图谱是对语义网络、本体论、万维网、语义网和关联数据的继承与发展。知识图谱构建流程主要包括四个步骤：数据获取、知识抽取、知识融合和知识加工。

数据获取是构建知识图谱的第一个环节，通过采集射频数据、传感器数据、系统日志数据、数据库数据和互联网数据等多源异构数据获得大量结构化、半结构化及非结构化样本。原始数据获取是知识图谱的前提和基础。

知识抽取的任务是将结构化（链接数据、关系型数据库）、半结构化（表格、列表）、非结构化（纯文本、音视频）等多模态数据转换为结构化知识并以三元组形式存储。知识抽取阶段的具体任务包括命名实体识别、数据抽取、关系抽取、事件抽取和共指消解等。

知识融合的任务是将多源同构的实体或概念的描述信息融合起来，故亦称为本体对齐 [42]。知识融合阶段主要的技术难点是数据的质量（数据错误、数据缺失、命名模糊、缩写）和规模（数据总量、数据多样性）挑战。实现知识融合的方法是通过计算属性相似度和实体相似度，从而根据相关性进行聚类。

知识加工的任务包括知识推理、本体构建、质量评估和知识更新。知识推理的目的是通过各种方法（产生式规则、模式匹配等）获取新的知识或者结论。本体分为领域本体和通用本体，本体层的质量是影响知识图谱优劣的直接因素，因此本体层定义是知识图谱构建最重要的任务。质量评估，又称 Knowledge Graph Refinement [43]，负责知识图谱数据补全和错误检测，通过不断迭代改进，最终获得 SOTA 模型（state-of-the-art model）。知识更新指产生新数据或修改原数据后追加至预训练模型的过程，是终身机器学习 [44] 的重要内容。

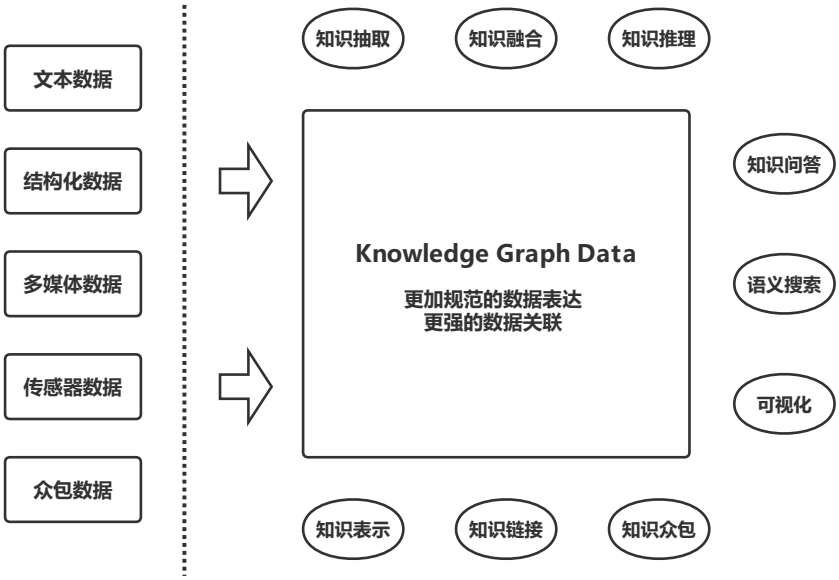



图 2.1: 知识图谱应用概览

如图2.1是知识图谱的技术体系和应用场景。首先，通过知识抽取、知识融合等技术，从海量文本、结构化信息和多媒体文件等原始数据中抽取相关数

据；其次，利用知识表示、知识链接等方法规范有序地组织上述数据，完成知识持久化存储；最终，在知识问答、语义搜索和可视化等软件工程领域落地。

## 2.2 文本预处理相关

### 2.2.1 词性标注

词性标注（Part-of-Speech tagging，POS tagging) 是指为语料库内每个单词结合上下文语境按其含义进行词性标记的文本处理技术[1]，即将分词处理后的每个单词识别为名词、动词、形容词等词性标签的过程。经典算法是基于隐马尔可夫模型（Hidden Markov Model，HMM）的词性标注方法，即在独立同分布情况下使用最大似然估计概率初始化 HMM [45]，并利用 Viterbi 算法 [46] 求出可能性最高的状态序列。斯坦福大学自然语言处理团队 [47] 结合额外四个规则：1）通过依赖网络充分考虑上下文信息；2）广泛地使用多维度词汇特性；3）在条件对数线性模型中充分利用先验项；4）对未知词的特性进行细粒度建模，据此开发的词性标注器 [48] 在 Penn Treebank WSJ [49] 数据集上取得 97.24% 的超高准确率。本系统使用词性标注完成第一轮的语义压缩操作。

### 2.2.2 语义相似度模型

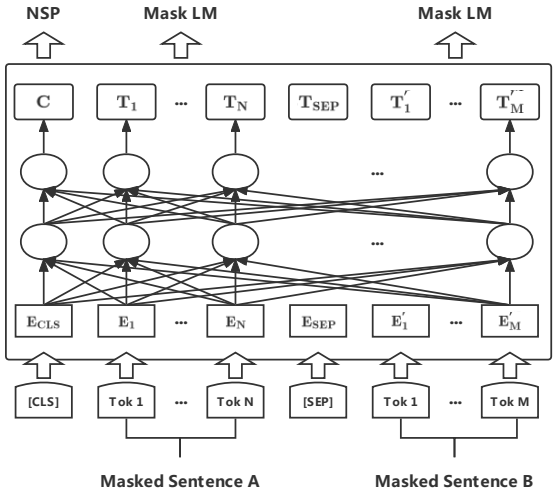



图 2.2: BERT 核心结构

语义建模的动态性和上下文语境的多样性贯穿整个词嵌入（Word Embedding）的研究历程。从最初基于 One-Hot [50]、TF-IDF [51]、TextRank [52] 等词

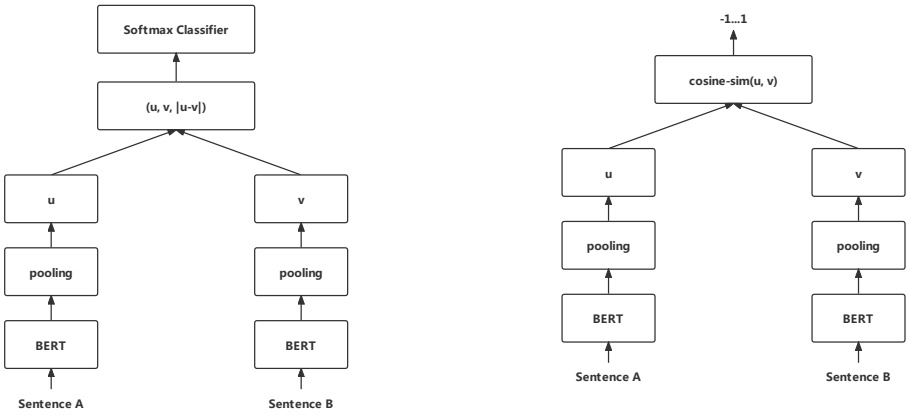
[1]https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/

袋模型（bag-of-words），到结合语义的 LSA/LSI [53]、LDA [54] 等文本主题模型，再到 Word2vec [55]、FastText [56]、Glove [57] 等为代表的固定表征模型，最后发展成以 ELMo [58]、GPT [59]、BERT [60] 等为代表的动态表征模型。BERT 模型在 11 种经典自然语言处理任务上均取得“state-of-the-art”的效果，但其在语义相似度任务中的时间复杂度为 $O(n^2)$（如 2.2所示），并且文本拼接的计算代价极大，因此不适合直接用于海量文本间的相似度计算。SBERT [61] 在 BERT 输出的基础上追加一轮池化操作，将得到句向量 $u$ 和 $v$ 与 $|u-v|$ 拼接并乘以权重 $W_t \in \mathbb{R}^{3n\times k}$（$n$ 表示句向量的维度，$k$ 表示标签的个数），得到的分类目标函数详见图 2.3a及公式 2-1。使用 *cosine* 函数计算上述两个句向量的相似度，得到的回归目标函数详见图 2.3b。

$$o = softmax(W_t(u, v, |u-v|)) \tag{2-1}$$

对于目标句子 $a$，给定正例句 $p$ 和负例句 $n$，利用 triplet loss [62] 计算得到句向量，实现文本的细粒度识别，三元组目标函数详见公式 2-2。$s_x$ 表示 $x$ 的句向量，预训练阶段保证 $s_p$ 比 $s_a$ 到 $s_n$ 至少近 $\epsilon$ 个度量距离单位。

$$max(\left\|s_a - s_p\right\| - \|s_a - s_n\| + \epsilon, 0) \tag{2-2}$$




(a) Classification Objective Function (b) Regression Objective Function

图 2.3: SBERT 核心算法图

语言表征模型的目的是服务下游任务，基于 BERT 的预训练模型能解决一词多义的问题，充分利用上下文信息，体现词的语法、语义等复杂特性。在面向缺陷摘要时，SBERT 比 BERT 的计算量更低，更适合数据规模不断膨胀的缺陷库场景，因此采用 SBERT 作为端到端模型的核心算法。

### 2.2.3 近似最近邻检索

Faiss [63]（Facebook AI Similarity Search）是目前最成熟的 $k$ 近似最近邻检索库（Approximate Nearest Neighbor，ANN），由 Facebook 开发，为高维海量数据场景提供高效且可靠的检索服务。其核心思路是批量搜索潜在近邻数据项而非仅受限于返回全局最优结果，在可接受范围内牺牲一定精度的情况下大大提高检索效率或降低内存占用。给定 $d$ 维数向量集 $x_i$，Faiss 读取并构建数据结构于内存，当输入 d 维度新向量 $x$ 时，它会根据公式 2-3有效执行索引的搜索操作，详见图 2.4。

$$i = argmin_i \|x - x_i\| \tag{2-3}$$

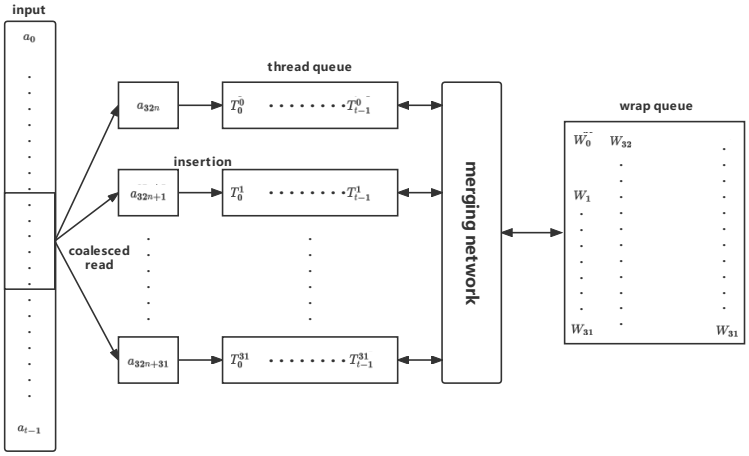



图 2.4: WarpSelect 概述

本系统在实现缺陷摘要向量化后，重点关注如何更快地进行文本相似度匹配，因而创建向量的近似最近邻检索成为召回阶段必不可少的一步。本系统采取的策略是离线构建句嵌入索引模型并做持久化处理，启动服务的同时将索引自动加载至内存，作为句嵌入模型的召回。

## 2.3 系统研发相关

### 2.3.1 客户端技术

React 起源于 Facebook 的内部前端框架，主要用于构建交互式好界面的 JavaScript 框架，是前端三大开源项目（React、Vue、Angular）用户基数最庞

大、社区生态最丰富的主流框架 [64]。React 的最大的特点是声明式和组件化[2]。声明式指在编写用户界面（UI）时，React 为目标应用的每个状态设计视图，当数据改变时能有效地更新并渲染组件，使代码调试更加可靠且方便。组件化是指 React 创建拥有各自状态的组件后，再由组件构成复杂的 UI，在分离组件状态和真实 DOM 的同时，能在目标应用内轻松传递数据。React 利用虚拟 DOM 技术和 Diffing 算法使频繁更新的数据统一且高效地渲染到用户交互界面，尽量减少与真实 DOM 的交互。

TypeScript 是 Microsoft 开源的 JavaScript 的超集，能在任何浏览器、计算机和操作系统上运行。JavaScript 是动态弱类型语言，因此难以在运行之前发现潜在的代码问题。TypeScript 对变量的数据类型加以限制，在变量和调用者之间建立结构化的契约，提供良好的静态类型检查支持，能够编译成标准的 JavaScript 并运行在任何浏览器或操作系统，适合大型项目的开发和维护。

Ant Design（Antd）是蚂蚁金服开发的企业级设计体系，基于四大设计观念：自然、确定性、意义感、生长性[3]，通过模块化解决方案，降低重复冗余的开发成本，使设计人员专注于用户体验。Antd 属于 React 组件库，能为 Web 应用提供丰富的基础 UI 组件，支持按需加载组件，对 TypeScript 的兼容性也非常出色。

Less（Leaner Style Sheets）是层叠样式表（Cascading Style Sheets，CSS）的扩展语言，设计初衷是为提高编码的效率，降低样式文件的修改难度和维护成本。Less 在 CSS 的基础上新增变量（Variables）、函数（Functions）、混合（Mixins）、嵌套（Nesting）、运算（Operations）等语法和功能[4]，使用 Less 替换 CSS 即由静态样式语言升级为动态样式语言。

Electron 是用于构建桌面应用程序的框架，支持 JavaScript、HTML 和 CSS 的开发模式。Electron 内嵌 Chromium 和 NodeJS，因而可以开发跨平台应用[5]（Windows、MacOS、Linux），故不受限于客户端应用的开发经验。包括 Visual Studio Code、Figma、WhatsApp、Twitch、Microsoft Teams 和 Github Desktop 在内的许多知名跨平台桌面应用，都是用 Electron 构建的。

本系统客户端基于 Electron+React 框架，实现网页端和桌面端的跨端运行；Antd+Less 能快速构建管理界面的 UI，实现模块化隔离，避免跨组件样式污染；TypeScript 向后兼容 JavaScript 版本，提供的静态类型检查有助于在编译之前发现客户端的代码问题。

[2]https://react.docschina.org/

[3]https://ant.design/docs/spec/introduce-cn

[4]https://less.bootcss.com/#%E6%A6%82%E8%A7%88

[5]https://www.electronjs.org/docs/latest/

### 2.3.2 服务端技术

Django 是基于 Python 的开源 Web 框架，功能大而全，能快速搭建并启动 Web 应用服务，实现从概念到落地 [65]。Web 开发的路由、数据库等常用配置选项，Django 均预先处理和内置提供。此外，模块化集成的设计思路为后续复用提供便利。因此，开发人员只需要专注于编写业务逻辑，节省重复造轮子的工作量。Django 具有上手迅速（Ridiculously fast）、功能完整（Fully loaded）、安全可靠（Reassuringly secure）、极易扩展（Exceedingly scalable）、全能通用（Incredibly versatile）的特点。Django 还拥有 Python 丰富的第三方库和活跃的用户社区。

Neo4j 是基于 Java 的高性能开源 NoSQL 数据库，底层以原生图结构而非数据表的形式存储节点和关系，即嵌入式的、基于磁盘的、完全事务性的 Java 持久化引擎。Neo4j 提供一套强大的 Cypher 查询语言，内置的可视化 UI。面向稠密连接的数据时，图算法不会因数据规模增大而降低计算性能，比关系型数据库的查询速度提升数千倍 [66]。在生物学、语义网络、推荐系统和智能问答等领域 Neo4j 具有独特优势和广泛应用 [67]。

MySQL 是由瑞典公司 MySQL AB 公司开发的开源关系型数据库，目前隶属于 Oracle 公司。MySQL 既可以作为应用程序的持久化方案，也可以支持内容检索、数据仓库和在线事务处理系统等各种场景 [68]。由于其体积小、速度快、成本低、性能好的优点，包括 Facebook、Twitter、Alibaba 和 Tecent 等在内的几乎所有世界上规模大、发展快的组织或机构都采用 MySQL 搭建高容量网站、存储关键业务数据。

Django 提供封装 Neo4j 和 MySQL 常用指令的 Py2neo 库和 PyMySQL 库，其内置的 ORM（Object Relational Mapping）框架描述 Python 对象和 MySQL 数据库之间的元数据映射关系，执行对象操作时能自动持久化到关系型数据库。因此，本系统使用 Neo4j 存储数据图谱，MySQL 存储原始缺陷报告，服务端基于 Django 框架，既能构建稳定的 Web 服务，又能兼容 Python 计算生态。

### 2.3.3 工程化工具

脚手架是一种快速搭建可运行应用程序软件服务的元编程方法[6]。脚手架的设计遵循复用原则（Reuse Principle）、DRY 原则（Don’t Repeat Yourself）、开闭原则（Open Close Principle）、避免重复原则（Stop Reinventing The Wheel）。

[6]https://stackoverflow.com/questions/tagged/scaffolding

在一些相似业务场景中，利用脚手架可以复用已有的逻辑、减少重复工作，以提高软件开发效率，符合敏捷开发的原则。

Webpack 是一个用于现代 JavaScript 应用程序的静态模块打包工具[7]（module bundler）。Webpack 视 HTML、JavaScript、CSS、图片等文件为一种资源，而每个资源文件映射成一个模块（module）文件，根据模块文件构成的依赖图 (dependency graph) 将所有的模块打包（bundle）成静态资源。Webpack 具备许多优点：采用单一入口打包所有前端资源；使用 loader 转换多种不同的编程语言；运用 plugin 扩展原有功能；提供热替换（hot module replacement）提升开发体验；利用代码压缩、代码分割等技术优化包结构，提高前端资源加载速度。

Yarn 是一个快速、可靠、安全的软件包管理器，还可以作为项目依赖管理工具 [69]。Yarn 由 Facebook、Google、Exponent 和 Tilde 联合推出，弥补 NPM（Node Package Manager）在性能和安全方面的缺陷。Yarn 支持批量、并行地下载依赖包，安装时耗更短，同时缓存已下载的包，避免重复下载，因而比 NPM 更加快速。Yarn 使用详细、简洁的锁文件格式，利用内置的安装算法可以跨系统工作，因而比 NPM 更加可靠。在执行代码前，Yarn 会校验依赖包的完整性，因而比 NPM 更加安全。

工程化工具有助于提升软件开发的效率、管理软件落地的进程，根据渐进式解决方案的思想，工具的存在是辅助应对各种复杂情况。本系统使用脚手架快速创建前后端可运行应用；通过 Webpack 部署本地调试环境，在上线前压缩和打包静态资源；选择 Yarn 管理项目依赖的 NodeJS 包并作为分发工具。

### 2.3.4 架构技术

Redis 是基于键值对的跨平台开源 NoSQL 数据库，遵守 BSD 开源协议 [70]，可用作数据库、缓存和消息中间件，常见应用场景包括：缓存系统、即时通讯和消息队列等。Redis 支持 5 种内置类型和 1 种自定义类型，内置类型包括：字符串（STRING）、列表（LIST）、哈希表（HASH）、集合（SET）和有序集合（ZSET），如图 2.5所示。Redis 具有高性能、原子操作和丰富的特性等优势。单个 Redis 的读写速度分别能达到 110000 次/秒和 80000 次/秒，但高并发场景下单节点模式存在雪崩、穿透和击穿等潜在风险，因此需要部署 Redis Cluster，实现主从副本机制。

Nginx 是轻量级的高性能 HTTP 和反向代理服务器。具有运行稳定、功能丰富、配置简单和低资源消耗等特点。基于 Epoll 开发模型的 Nginx 能够同时

[7]https://webpack.docschina.org/concepts/modules

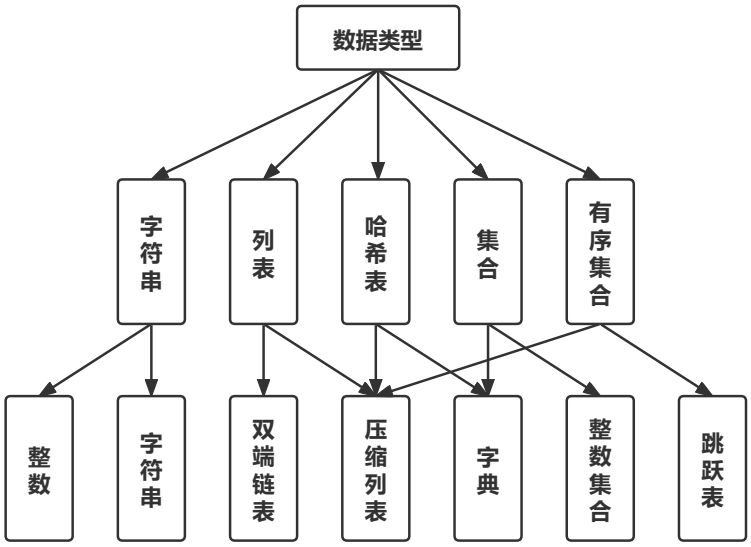



图 2.5: Redis 内置数据类型关系图

响应 50000 个并发连接，处理静态资源时占用更少的内存空间。Nginx 主要用于均衡负载和反向代理，均衡负载指将客户端请求按预设规则分摊到多台服务器；反向代理指客户端请求发送至代理服务器，由代理服务器负责从服务器集群中获取并返回资源，即将代理服务器视为目标服务器，而屏蔽背后复杂的指派行为。

Docker 是基于 Golang 的开源容器虚拟化技术，使用完全沙箱机制，实现应用级隔离，具有可移植、部署快和生态丰富等优势。Docker 服务于 PaaS 层，提供平台和工具来管理容器的生命周期，使应用程序与基础架构分离，主要提供 CI/CD、分布式服务化、应用自动化发布、服务可用性和软件能力复用的解决方案。利用 Docker 能实现快速交付、测试和部署，显著降低编写代码和部署运行之间的延时。

本系统采用完全去中心化的 Redis Cluster 缓存热点数据，不同 Redis 节点间以 PING-PONG 的容错机制维持连接状态，确保集群可用；均衡负载通过将流量分发给不同的服务器，提升系统整体的性能，使服务器集群的吞吐量大于上游的应用服务器，而 Nginx 完全符合上述要求，同时提供静态资源缓存的功能；搭建 Redis 集群和 MySQL 主从分离于 Docker，达成镜像压缩和容器迁移。

## 2.4 本章小结

本章概述项目相关的理论基础、技术框架和开发工具。宏观上分为数据处理、文本预处理和系统研发三大类。数据处理阶段主要介绍数据清洗和知识图

谱的技术成熟度、应用场景和执行步骤。文本预处理阶段主要介绍词性标注、语义相似度计算和近似最近邻检索的算法原理，技术选型方面以 StanfordNLP 用作文本分词和词性标注，SBERT 作为语义相似度计算算法，使用 Faiss 构建句嵌入索引模型。系统研发阶段主要介绍前后端技术框架和工程架构辅助工具，前端基于 React+Antd+TypeScript+Less 模式并利用 Electron 构建跨平台桌面应用程序，后端采用 Django 框架，数据库选用 Neo4j 和 MySQL，热点数据缓存使用 Redis，静态资源打包和软件包管理器分别选择 Webpack 和 Yarn。通过描述上列算法框架的核心原理和适用场景，为项目在软工领域的落地提供理论依据和技术支持。

# 第三章　需求分析与概要设计

本章结合业务场景对系统进行需求分析和概要设计，划分问答系统各个功能及其子功能的边界。参考软件开发生命周期模型，根据面向 Bugzilla 的知识图谱问答系统应用场景，概括性地说明系统需求分析、总体设计、数据处理和持久化设计的总体思路。

## 3.1　系统整体概述

由于软件固有的复杂特性，项目开发必然伴随软件缺陷的出现。为降低缺陷产生的影响和修复的成本，应做好项目相关缺陷库的追踪和管理。作为问题驱动的信息获取过程，问答系统能快速、准确和简洁地在缺陷检索、缺陷查重、缺陷跟踪、缺陷修复和事故追责等业务场景获取用户期望的信息。

表 3-1: 不同类型的问答系统对比

| 维度 | 信息检索 | 模式匹配 | 语义解析 | 深度学习 |
|---|---|---|---|---|
| **前期准备** | 无需处理 | 模板标注 | 词汇表定义 | 模型预训练 |
| **适用场景** | 开放场景 | 限定场景 | 限定 & 开放 | 限定 & 开放 |
| **意图判断** | 不需要 | 直接匹配 | 需要 | 不需要 |
| **泛化能力** | 强 | 弱 | 中 | 强 |

目前主流问答技术如表 3-1，对应的问答系统包括：基于信息检索的问答系统、基于模式匹配的问答系统、基于语义解析的问答系统、基于深度学习的问答系统以及基于混合问答框架的问答系统。


图 3.1: 总体结构图

本系统是基于混合问答框架的问答系统。总体结构见图 3.1，根据提问形式分为缺陷报告查重（缺陷标识获取）问答和句式规则匹配问答。前者原理如

下：首先，建立数据量足够大的语料库；其次，对语料库内文本进行预处理，转成句向量；再次，利用索引（如倒排索引等）在可接受范围内牺牲少量精度，构建句嵌入索引模型，提高检索效率；然后，再使用余弦定理计算输入文本与向量模型的文本相似度；最后，将语料库中相似度较高的 Top K 个答案作为结果返回。后者原理如下：归纳常见自然语言问题为若干句型；对输入文本进行意图识别和模式匹配，并依据规则分发至相应功能函数；返回知识图谱查询内容，格式化处理后输出最终结果。

## 3.2 系统需求分析

### 3.2.1 涉众分析

本系统是面向缺陷库问答领域的跨平台桌面应用，采用基于知识图谱的混合问答框架（深度学习和模式匹配），涉众分析结果如表 3-2所示，主要涉及缺陷报告人员、软件开发人员、问答系统用户和持续集成人员。

表 3-2: 涉众分析表

| 涉众名称 | 涉众特征与期望 |
| --- | --- |
| 持续集成人员 | 句嵌入索引模型和缺陷库的管理员，熟悉模型预训练的流程，负责源数据清洗、句嵌入预训练、知识图谱构建以及缺陷报告审批，主要任务是管理模型的终身学习。期望预处理各步骤具有较高的自动化水平；缺陷报告描述的准确完整、审批的便捷及时。 |
| 缺陷报告人员 | 熟悉项目流程、参与开发或测试，具有一定的专业能力，了解缺陷报告填写规范，能复现并描述缺陷的触发步骤，包括测试人员、开发人员以及能正确辨别缺陷并上报的相关用户。期望提供缺陷快速查重通道、用户友好的填写流程，减少填写工作量的同时，提高缺陷表述的质量。 |
| 软件开发人员 | 负责项目开发和缺陷修复，对项目非常熟悉，认定缺陷表述的明确性、真实性，同步缺陷报告的实时状态。期望缺陷库内容查询、更新的准确性和高效性。 |
| 问答系统用户 | 能用自然语言描述缺陷的主要特征，掌握特殊疑问句、一般疑问句、反意疑问句、选择疑问句、陈述句和多跳疑问句等句式的模糊表述。期望问答系统应答的实时性、知识构成的完整性以及语言模糊表述的容忍性。 |

持续集成人员负责源数据的清洗、知识图谱的构建和句嵌入索引模型的持续集成，确保模型能够不断迭代更新，在维护缺陷库的同时，保障缺陷报告的质量；缺陷报告人员包括能复现缺陷产生流程并且准确填写缺陷报告的所有相关用户；软件开发人员负责系统开发、功能维护和缺陷修复，缺陷表述的质量直接影响软件开发人员的工作效率；问答系统用户通过描述缺陷的主要特征对缺陷报告进行查重或获取缺陷标识，也可以通过常用句式提问获得反馈结果。

### 3.2.2 功能性需求

功能性需求分析是经过深入调研，准确捕获和理解用户的意图，规定软件开发人员必须实现的满足业务需求的系统功能。本系统功能性需求如表 3-3所示，整体上分为十二个功能性需求，分别是缺陷报告清洗、三元组文件生成、语义相似度计算、句嵌入索引建模、知识图谱构建、缺陷报告填写、缺陷报告审批、缺陷库实体查询、缺陷库关系查询、缺陷库内容更新、缺陷报告查重和缺陷知识问答。按照需求重要程度划分，P0 优先级包括三元组文件生成、语义相似度计算、句嵌入索引建模、知识图谱构建、缺陷报告查重和缺陷知识问答；P1 优先级包括缺陷报告清洗；P2 优先级包括缺陷报告填写和缺陷报告审批；P3 优先级包括缺陷库实体查询、缺陷库关系查询和缺陷库内容更新。

表 3-3: 系统功能需求表

| 需求编号 | 需求名称 | 需求描述 | 优先级 |
|---|---|---|---|
| R1 | 缺陷报告清洗 | 对爬虫获取的 Bugzilla 开源 JSON 嵌套缺陷报告数据进行扁平化处理，再根据本体归纳和三元组定义筛选所需字段，输出加工处理后的结构化数据文件。 | P1 |
| R2 | 三元组文件生成 | 完成数据清洗后的结构化数据文件按照本体和三元组输出对应名称的实体文件和关系文件。 | P0 |
| R3 | 语义相似度计算 | 按照预设配置计算语料库内每条缺陷摘要的句向量。 | P0 |
| R4 | 句嵌入索引建模 | 在语义相似度计算得到的句嵌入结果的基础上构建索引，作为模型的召回。 | P0 |
| R5 | 知识图谱构建 | 批量导入指代本体和三元组的实体文件和关系文件，构造知识图谱，并持久化存储于图数据库。 | P0 |
| R6 | 缺陷报告填写 | 缺陷报告人员按照提示和要求填写并创建项目相关的缺陷报告。 | P2 |
| R7 | 缺陷报告审批 | 持续集成人员根据缺陷报告的描述尝试复现并审核结果。审核通过的缺陷报告用于模型再训练。 | P2 |
| R8 | 缺陷库实体查询 | 相关用户输入源实体的标签和属性值，系统返回潜在实体及其近邻查询集。 | P3 |
| R9 | 缺陷库关系查询 | 相关用户输入起始实体和结尾实体，系统返回实体之间的完整路径，即关系链。 | P3 |
| R10 | 缺陷库内容更新 | 缺陷报告人员修改未提交的缺陷报告草稿；持续集成人员更新缺陷报告的状态。 | P3 |
| R11 | 缺陷报告查重 | 相关用户在问答主页描述缺陷摘要或特征，系统计算文本相似度并返回句嵌入索引模型匹配到的缺陷报告。 | P0 |
| R12 | 缺陷知识问答 | 问答系统用户基于缺陷标识或常见句式表述问题，系统依据模板匹配反馈结果。 | P0 |

### 3.2.3 非功能性需求

非功能性需求影响软件系统能否为目标用户提供持续、高效且稳定的服务，即为满足业务需求而具备的除功能需求外的必要特性。

性能是评价软件系统交互体验的主要指标。RAIL 模型[1]衡量系统 Response、Animation、Idle 和 Load 四个维度的性能，本系统对于事件的响应时间应该小于等于 100ms；确保动画在每 16ms 内至少产生一帧，即 FPS 达到 60 及以上；最大化空闲时间，进程任务尽量控制在 50ms 内完成；保证静态资源尤其是关键资源的传输速率，实现客户端在 1000ms 内完成页面加载并可交互。

可用性指在特定场景下，衡量系统所具有的可学习性、可记忆性、效率、出错率和用户满意度 [71]。本系统全年正常运行时长至少达到 99%，完善界面交互响应能力，降低新用户的认知和学习成本，覆盖错误提示，尽可能地提高用户满意度。

可靠性指一定条件下，在规定时间内，软件系统能够满足业务需求的能力，反映软件系统的质量。提高软件系统的可靠性本质上就是减少错误发生率和增强功能鲁棒性。软件系统的可靠性设计需要满足“七项避错设计原理”，即简单、同型、对称、层次、线型、易证和安全。此外，在保证软件测试覆盖率的同时，完善日志和回滚机制。

可扩展性指软件在不远的将来适应潜在业务需求或运行环境的能力。本系统的架构核心思想是模块化和跨平台。前端采用函数式编程，后端按业务划分模块，提高系统复用能力。立足于跨端桌面应用，提高系统环境适应力。

可移植性指当环境发生变化时，软件系统不需要作过多改动就能正常工作。通常从适应性、易安装性、共存性、易替换性和依存性等角度描述软件系统的可移植能力。本系统采用前后端分离的架构体系，前端 Electron 可以打包生成跨平台桌面应用；后端 Python 运行于自带的跨平台解释器上，是迁移性非常强的脚本语言；架构层面的工具均寄宿于 Docker，能快速部署并运行在全新的环境。

### 3.2.4 系统用例描述

用例是从相关用户角度观察到的系统功能单元，本系统用户分为持续集成人员、缺陷报告人员、软件开发人员和问答系统用户。用例图如图 3.2所示，将功能性需求总结归纳为源数据清洗、句嵌入预训练、知识图谱构建、缺陷报告管理、缺陷库问答和缺陷库管理六个核心用例。

[1]https://www.smashingmagazine.com/2015/10/rail-user-centric-model-performance/#rail-perf-model

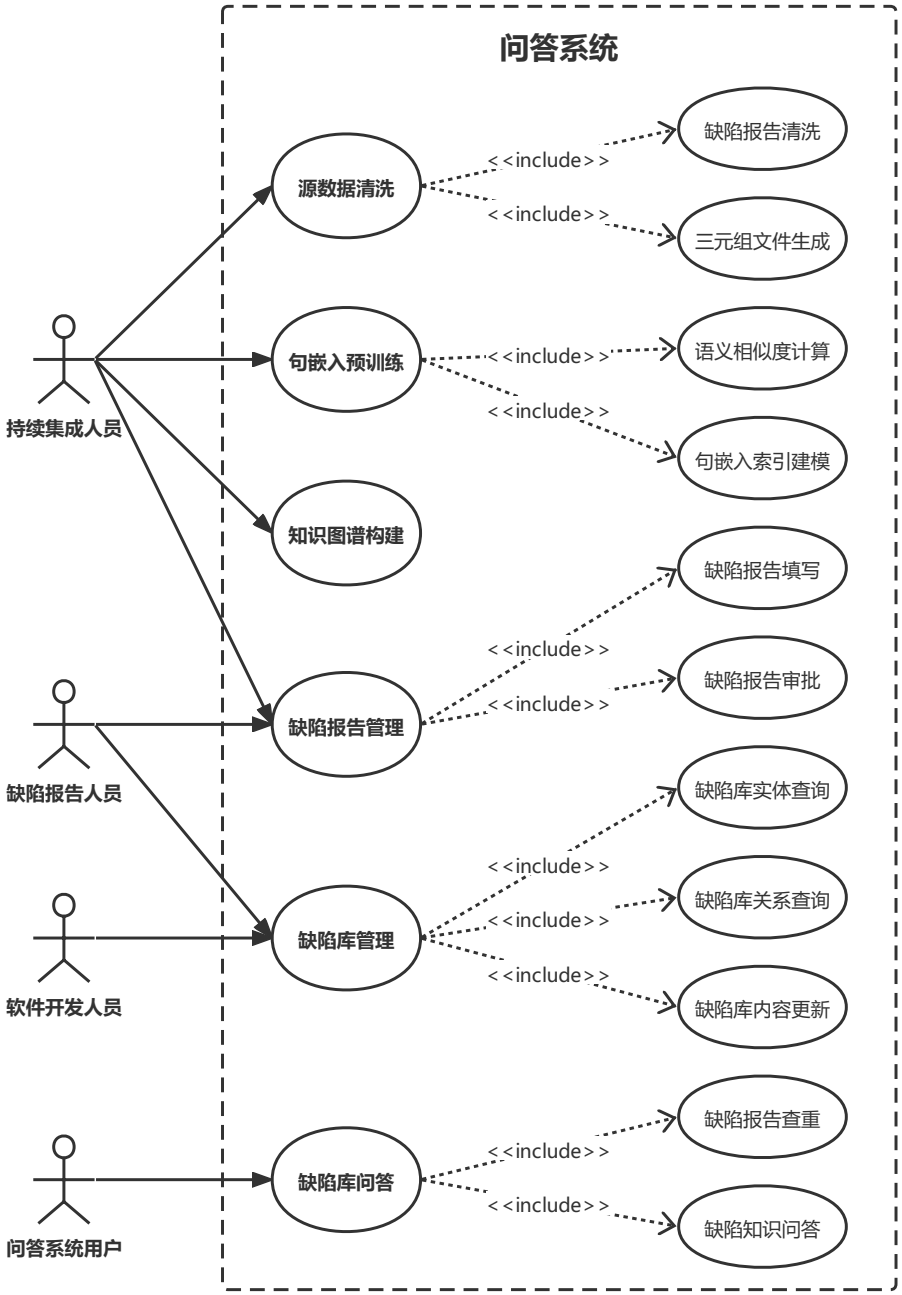



图 3.2: 系统用例图

用例和需求的对应关系如表 3-4所示，需求编号详情参见表 3-3。其中用例 UC1 源数据清洗对应需求 R1 缺陷报告清洗和需求 R2 三元组文件生成；用例 UC2 句嵌入预训练对应需求 R3 语义相似度计算和需求 R4 句嵌入索引建模；用例 UC3 知识图谱构建对应需求 R5 知识图谱构建；用例 UC4 缺陷报告管理对应需求 R6 缺陷报告填写和需求 R7 缺陷报告审批；用例 UC5 缺陷库管理对应需

求 R8 缺陷库实体查询、需求 R9 缺陷库关系查询和需求 R10 缺陷库内容更新；用例 UC6 缺陷库问答对应需求 R11 缺陷报告查重和需求 R12 缺陷知识问答。

表 3-4: 用例需求对应表

| 用例编号 | 名称 | 需求编号 |
|---|---|---|
| UC1 | 源数据清洗 | R1、R2 |
| UC2 | 句嵌入预训练 | R3、R4 |
| UC3 | 知识图谱构建 | R5 |
| UC4 | 缺陷报告管理 | R6、R7 |
| UC5 | 缺陷库管理 | R8、R9、R10 |
| UC6 | 缺陷库问答 | R11、R12 |

表 3-5为源数据清洗用例的详细描述。本系统数据来自开源缺陷追踪系统 Bugzilla，关联项目是 Eclipse[2]。持续集成人员将爬虫获取到的 CSV 源文件解析为结构化数据，将嵌套属性扁平化，根据本体层定义将需求相关字段转化为三元组文件。源数据清洗是后续流程的基础和前提，而数据清洗和知识挖掘的程度视实际需求变更情况而定。

表 3-5: 源数据清洗用例描述表

| **ID** | UC1 | **用例名称** | 源数据清洗 | **优先级** | P0 |
|---|---|---|---|---|---|
| **用例描述** | 清洗项目需求相关数据字段 | | | | |
| **参与者** | 持续集成人员 | | | | |
| **前置条件** | 通过爬虫获取 Bugzilla 开源数据集 | | | | |
| **后置条件** | 将清洗后的数据存储于 CSV 文件 | | | | |
| **正常流程** | 1. 使用 Pandas 库读取爬虫得到的 CSV 源文件；<br>2. 将网络请求字段扁平化，获取 data 属性；<br>3. 根据本体层获取对应键值数据并输出初加工 CSV 文件；<br>4. 使用 Pandas 库读取初加工 CSV 文件，输出缺陷报告摘要文件、实体文件和关系文件；<br>5. 控制台打印数据清洗执行时长。 | | | | |
| **扩展流程** | 3a. 爬虫数据格式非法检测<br>1. 字段 description 中图片乱码内容匹配并剔除；<br>2. 非空判定，空字段补充缺省值。 | | | | |

表 3-6为句嵌入预训练用例的详细描述。用例 UC1 生成的缺陷报告摘要文件是本用例的前置依赖。本阶段的任务是模型的预训练，主要包括两步骤：1. 将缺陷报告摘要转化为可计算、向量化的句嵌入；2. 将句嵌入数据集用近似最

[2]https://bugs.eclipse.org/bugs/

近邻算法构建向量索引。持续集成人员负责数据更新后的模型再训练，实现句嵌入索引模型的终身学习。

表 3-6: 句嵌入预训练用例描述表

| **ID** | UC2 | **用例名称** | 句嵌入预训练 | **优先级** | P0 |
|---|---|---|---|---|---|
| **用例描述** | 对缺陷报告摘要进行预训练，生成句嵌入索引模型 | | | | |
| **参与者** | 持续集成人员 | | | | |
| **前置条件** | 已获取缺陷报告的摘要数据集 | | | | |
| **后置条件** | 持久化句嵌入索引模型 | | | | |
| **正常流程** | 1. 使用 Pandas 库读取数据清洗后的缺陷报告摘要文件；<br>2. 选择并安装 SBERT 预训练模型；<br>3. 配置 SBERT 模型参数；<br>4. 缺陷报告摘要去停用词和无效词性后，利用 SBERT 计算句向量；<br>5. 根据真实场景选择 Faiss 索引类型算法；<br>6. 输出训练生成的句嵌入索引模型。 | | | | |
| **扩展流程** | 4a. 二进制形式序列化句向量数据集 | | | | |

表 3-7为知识图谱构建用例的详细描述。用例 UC1 生成实体文件和关系文件是本用例的前置依赖。持续集成人员需要完成知识图谱构建的任务，包括根据本体归纳和三元组定义将实体和关系录入到图数据库。

表 3-7: 知识图谱构建用例描述表

| **ID** | UC3 | **用例名称** | 知识图谱构建 | **优先级** | P0 |
|---|---|---|---|---|---|
| **用例描述** | 根据本体层定义的规则构建知识图谱并存储于图数据库 | | | | |
| **参与者** | 持续集成人员 | | | | |
| **前置条件** | 已获取完成数据清洗的实体文件和关系文件 | | | | |
| **后置条件** | 构建后的知识图谱持久化于图数据库 | | | | |
| **正常流程** | 1. 逐一读取“节点”文件夹的实体文件；<br>2. 以追加覆盖的模式在图数据库中创建实体；<br>3. 逐一读取“关系”文件夹的三元组文件；<br>4. 以批量导入的模式在图数据库中构建关系。 | | | | |
| **扩展流程** | 2a. 同源异构数据聚类<br>1. 不同性质的缺陷报告均归为一类；<br>2. 不同职务人员均归为同一类。 | | | | |

表 3-8为缺陷报告管理用例的详细描述。持续集成人员和缺陷报告人员分别负责缺陷报告审批和缺陷报告填写，前置条件是按照缺陷报告的步骤描述能够复现目标软件缺陷。本用例的目标是扩充缺陷库的数据规模，鉴于 Bugzilla 数据充足且 Eclipse 项目稳定，因此用例 UC4 的优先级定为 P2。

表 3-8: 缺陷报告管理用例描述表

| **ID** | UC4 | **用例名称** | 缺陷报告管理 | **优先级** | P2 |
|---|---|---|---|---|---|
| **用例描述** | 新发现缺陷报告的填写和新提交缺陷报告的审核 | | | | |
| **参与者** | 持续集成人员、缺陷报告人员 | | | | |
| **前置条件** | 缺陷可复现 | | | | |
| **后置条件** | 关系型数据库中新增一条记录或修改缺陷报告审核状态 | | | | |
| **正常流程** | 1. 缺陷报告人员点击进入缺陷报告填写界面；<br>2. 缺陷报告人员按提示填写缺陷报告相关信息；<br>3. 点击提交按钮，向后端发起异步网络请求并写入数据库；<br>4. 持续集成人员复现并审批新提交的缺陷报告；<br>5. 点击切换报告状态，向后端发起异步网络请求并将更新数据库。 | | | | |
| **扩展流程** | 3a5a. 判空操作和错误边界<br>1. 输入为空值则前端直接提示，不往后台发送数据；<br>2. 查询内容不存在或网络请求错误，前端处理错误边界。<br>2a. 系统新增并保存缺陷报告，状态置为“待审核” | | | | |

表 3-9为缺陷库管理用例的详细描述。缺陷报告人员可查询缺陷库的实体和关系，为缺陷报告填写提供参考；软件开发人员在修复缺陷或其他影响缺陷报告状态的行为后及时修改报告状态，进而更新缺陷库内容。由于本用例没有直接阻塞问答系统的核心链路，因此优先级定为 P3。

表 3-9: 缺陷库管理用例描述表

| **ID** | UC5 | **用例名称** | 缺陷库管理 | **优先级** | P3 |
|---|---|---|---|---|---|
| **用例描述** | 已有缺陷报告的 CURD 等基本原子操作 | | | | |
| **参与者** | 缺陷报告人员、软件开发人员 | | | | |
| **前置条件** | 非新增操作需要确保缺陷库中存在对应缺陷报告 | | | | |
| **后置条件** | 根据原子操作更新缺陷库 | | | | |
| **正常流程** | 1. 选择要查询的实体；<br>2. 输入目标实体的标签及属性值，系统返回实体及其邻节点；<br>3. 选择要查询的起始实体和结尾实体；<br>4. 输入头尾实体的标签及属性值，系统返回实体间完整路径；<br>5. 缺陷报告人员修改或更新尚未审核的缺陷报告内容；<br>6. 软件开发人员根据能否复现和产品需求更新缺陷报告状态。 | | | | |
| **扩展流程** | 1a3a. 判空操作和错误边界<br>1. 输入为空值则前端直接提示，不往后台发送数据；<br>2. 查询内容不存在或网络请求错误，前端处理错误边界。 | | | | |

表 3-10为缺陷库问答用例的详细描述。本用例是系统的核心需求，包括缺陷报告查重和缺陷知识问答。获取缺陷唯一标识或填写软件缺陷报告前查重验

证时，问答系统用户通过自然语言的形式描述缺陷的主要特征（等价于缺陷报告摘要），系统据此识别语义、作出应答并返回最终答案。当问答系统用户按照常见问题句型或缺陷唯一标识提问时，系统根据预设模板进行模式匹配并返回最终结果。

表 3-10: 缺陷库问答用例描述表

| **ID** | UC6 | **用例名称** | 缺陷库问答 | **优先级** | P0 |
|---|---|---|---|---|---|
| **用例描述** | 按照缺陷摘要相似度查重、模式规则匹配问答 | | | | |
| **参与者** | 问答系统用户 | | | | |
| **前置条件** | 句嵌入索引模型已训练，知识图谱已构建 | | | | |
| **后置条件** | 返回应答结果 | | | | |
| **正常流程** | 1. 用自然语言描述缺陷主要特征；<br>2. 点击搜索按钮，句嵌入索引模型将阈值内的最邻近结果返回；<br>3. 用常见自然语言句式提问；<br>4. 根据句式匹配规则分发至功能函数，反馈应答结果。 | | | | |
| **扩展流程** | 2a4a. 判空操作和错误边界<br>1. 输入为空则前端直接校验并提示错误信息；<br>2. 查询内容不存在或网络请求出错，前端处理错误边界。 | | | | |

## 3.3 系统总体设计

### 3.3.1 总体架构

本系统采用基于 4R 理论 [3]的前后端分离、架构与运维相结合的设计模式。系统整体架构如图 3.3所示。从顶层（Rank）结构视角定义系统的角色（Role）组成、角色关系（Relation）和运作规则（Rule）。

展示层使用 React 描述页面行为；Typscript 校验代码的静态类型；Antd 确定页面样式的整体基调；使用 LESS 提高层叠样式表的开发效率，利用模块化技术避免样式的全局污染；系统涉及的可视化图表源自 Echarts；跨域解决方案通过在 Webpack 中配置 proxy 进行处理；配置 Axios 请求拦截器，实现前后端数据的安全传输，保证同一时间段内有且只有一个实例，避免浏览器内存泄漏；Electron 内置 Chromium 和 Node.js 引擎，具备跨端迁移的优势。本系统是一款基于 Electron 和 React 的跨平台桌面应用。

转发层使用 Django，该框架拥有活跃的社区用户和丰富的第三方库，功能强大而齐全，通过脚手架自动生成项目实例所需的设置项集合，包括数据库配

[3]https://www.jianshu.com/p/cdce0b40ec5d

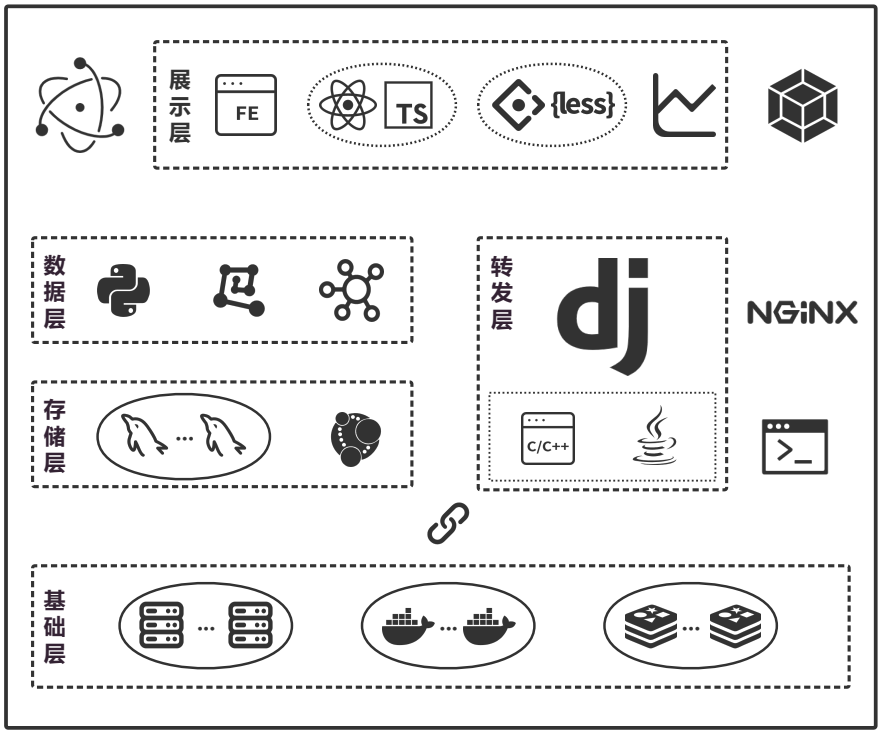



图 3.3: 系统架构图

置、Django 配置和应用程序配置等。Python Web 应用框架能直接调用诸如 SentenceTransformer、Pandas 等第三方库，以及提供稳定可靠的基于 Java 的 StanfordCoreNLP、基于 C++ 的 Faiss 等跨编程语言的 API 接口。

数据层通过 Python 编写数据清洗自动化脚本，抽取结构化数据、摒弃冗余属性、整合样本集合。首先，使用 StanfordNLP 对每个缺陷摘要进行文本分词、词性标注和去停用词。其次，利用 SBERT 计算缺陷摘要的句向量并调用 Pickle 将结果序列化存储到硬盘，与其他词向量模型相比，SBERT 在语义识别和处理的表现更好。最后，针对训练后的高维空间数据，采用 Faiss 构建高效而可靠的索引聚簇并持久化模型，至此预训练工作结束。

存储层描述数据存储解决方案的具体落地情况。数据完成预处理后通过自动化脚本按块读取并按照预先定义好的本体层结构以三元组的形式存储于 Neo4j。MySQL 集群则保存缺陷摘要集，提问框通过节流轮询的方式，依据最左前缀匹配的原则，返回缺陷库的相关摘要信息。

基础层是支撑整个系统可用性、可靠性、可移植性和高性能的基础设施。中间件通过虚拟化技术部署于 Docker 容器。Nginx 代理服务器集群，在高并发场景下能够及时削峰，避免服务器宕机。搭建 Redis 集群以防止缓存穿透、击穿和雪崩等问题。

### 3.3.2 4+1 视图

“4+1”视图 [72] 是 Kruchten 提出的分别从逻辑视图、进程视图、开发视图、物理视图和场景视图这 5 个不同的视角来描述软件体系结构的模型，如图 3.4所示。场景视图等价于系统用例图，详情参见上文，此处不再赘述，以下将围绕其他四个视图对系统结构和功能展开说明。

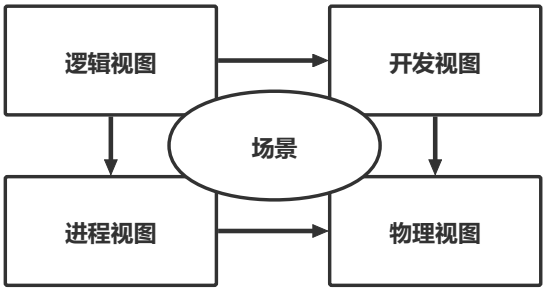



图 3.4: 4+1 视图模型

**逻辑视图**基于用户视角，专注于系统提供给最终用户的服务，即功能性需求。软件设计人员结合问题域按照面向对象的原则将系统分为多个关键抽象的集合。采用 Rational/Booch 方法通过类图和类模板来表示逻辑视图。类图描述一组类及其逻辑关系：关联、用途、组合、继承等；类模板描述每个单独的类，强调主要的类操作和关键对象特征。本系统的逻辑视图如图 3.5所示。

HomePage 组件描述问答界面的前端展示，负责缺陷库问答的页面交互；ManagePage 组件描述管理界面的前端展示，负责缺陷报告管理和缺陷库管理的页面交互。HomePage 根据问题形式，应答结果也分为缺陷报告查重结果组件 SimilarityAnswer、模式匹配结果组件 StructureAnswer 和置空结果组件 EmptyAnswer。前端的状态管理工具采用 Recoil，用于组件间的状态共享和数据传递。

Urls 是 Django 的路由分发控制器，负责接收网络请求并按路由规则分发到相应类的功能函数中。HintService 提供用户输入缺陷特征时的文本补全或提示服务，以节流的形式每隔固定时长，前端会向后端发起请求，后端根据当前输入内容按最左前缀匹配原则查询 MySQL 数据库并返回结果。ReportService 负责缺陷报告管理和缺陷库管理，包括缺陷报告的填写和审批、更新已有缺陷报告的状态，将修改后的内容同步到数据库。KGService 负责数据图谱相关操作，包括实体和关系的 CRUD 操作，下辖 Neo4j 封装类，提供多种常见图数据库操作指令以及直接使用 Cyper 查询 Neo4j。QAService 负责问答服务，内置 XModel 核心模型类，集成管理模型预训练 ModelTraining，文本分词和词性标注 StanfordCoreNLP，模式匹配分发规则 RuleMatch。

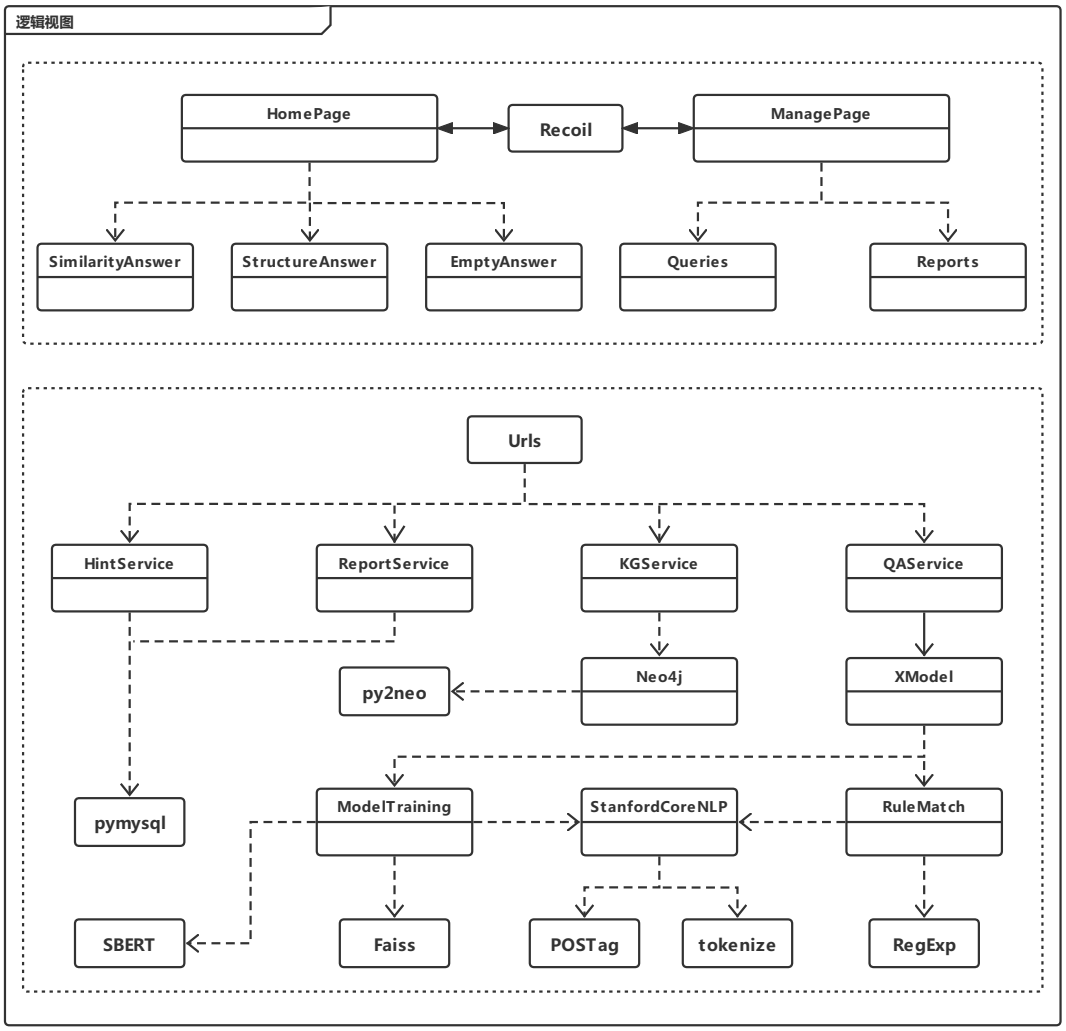



图 3.5: 逻辑视图

**进程视图**基于运行特性视角，专注于软件层面的非功能性需求，如系统运行时的性能、可用性等，涉及分布式并发控制、系统完整性和容错能力。进程视图定义逻辑视图主要标识类的操作由哪个控制线程负责执行。设计人员将系统划分为多个抽象级别的进程视图，每个级别解决不同层级的问题。本系统的进程视图如图 3.6所示。

本系统的前端进程运行在 Electron 客户端，用户在问答界面输入框内进行提问，前端进程会根据输入框状态以节流轮询的方式向 Nginx 发起 HTTP 请求，Nginx 根据负载均衡算法反向代理到对应服务器。后端会首先查询 Redis 内是否缓存相关数据，若已存在则直接返回，否则，先去查询 MySQL 并将数据缓存至 Redis。缺陷报告的提交和更新则直接与 MySQL 同步数据。缺陷报告查重或缺陷标识查询由问答服务判定并分发到对应子服务，先对文本进行词性标

注并去停用词，再计算句向量，最后利用 Faiss 查询 Top K 缺陷报告摘要。模式匹配问答由分发规则转至对应的功能函数处理，查询白名单获取关键实体和关系，通过补全字符串模板为 Cyper 查询语言将 Neo4j 查询结果返回。

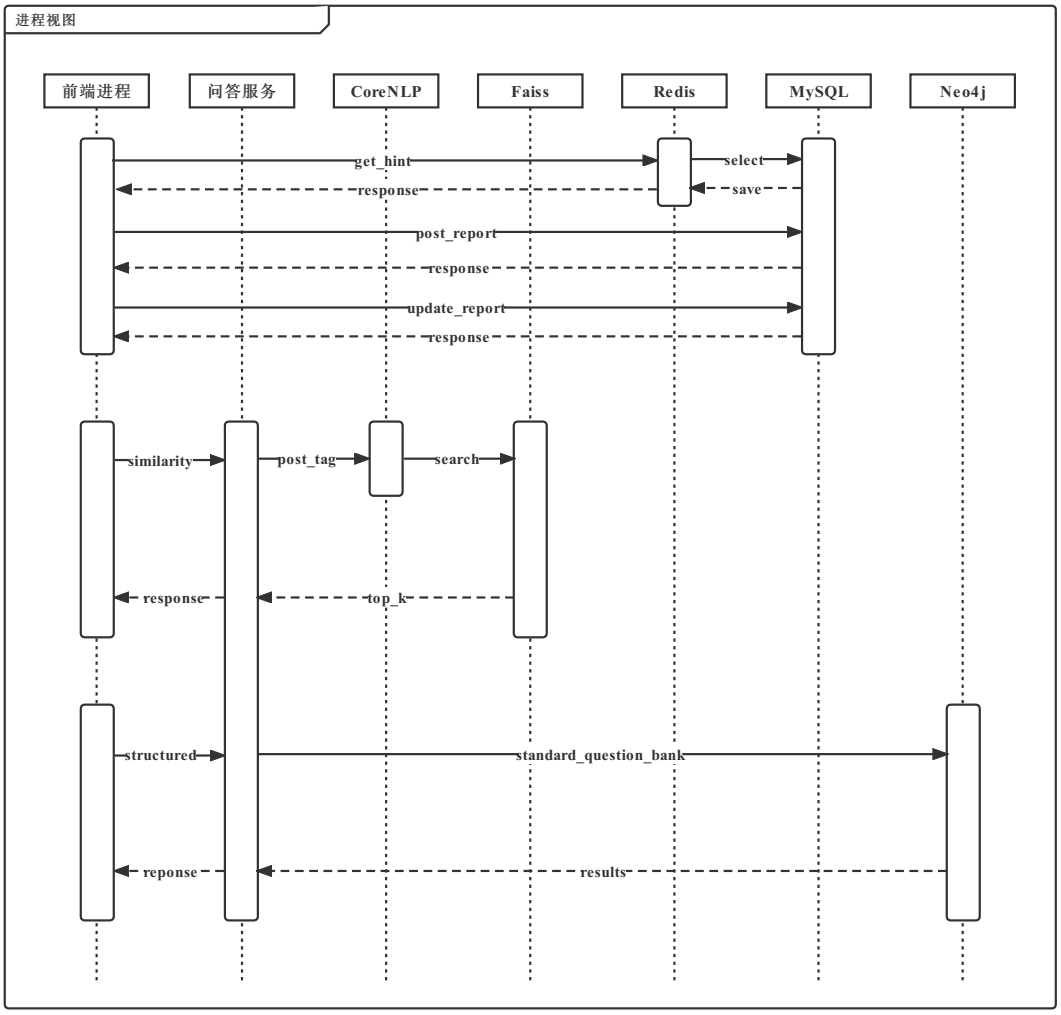



图 3.6: 进程视图

**开发视图**基于开发人员视角，专注于软件的静态组织结构。将软件系统自上而下逐层划分为若干可以被独立开发和管理的模块或子系统，每层为上一层提供接口，层级越低通用性越好。开发视图是需求分配的基础，有助于成本评估、提前计划、项目管理、进度监控和软件重用，易于获得开发人员的任务分工情况。本系统的开发视图如图 3.7所示。

宏观上系统总共划分为五个业务层级。前端（展示层）采用 Electron 内嵌 React+Antd+LESS+TypeScript 的技术栈。按文件夹划分，api 存放网络请求等接口文件；component 存放可复用函数组件；page 存放问答主页和管理主页；state 存放 Recoil 相关文件；util 存放各类工具类文件。后端（转发层）采用

Django ORM 框架便于直接调用 Python 三方库和操作数据库。KGController 控制知识图谱相关的操作；QAController 控制问答交互相关的操作；HintController 控制问题补全和提示相关的操作；ReportController 控制缺陷报告和缺陷库相关的操作。数据层存放模型相关的文件，DictInit 是持久化字典数据的预加载文件，CONSTANT 保存各类常量，XModel 是核心模型类，export2neo 封装 Neo4j 指令。模型层是数据层的子层，StanfordNLP 负责文本分词和词性标注；FaissIndex 是预训练索引模型；Sentence Transformer 是 SBERT 的 Python 包；WhiteList 定义数据白名单。存储层包括 MySQL 主从复制集群和 Neo4j 图数据库。

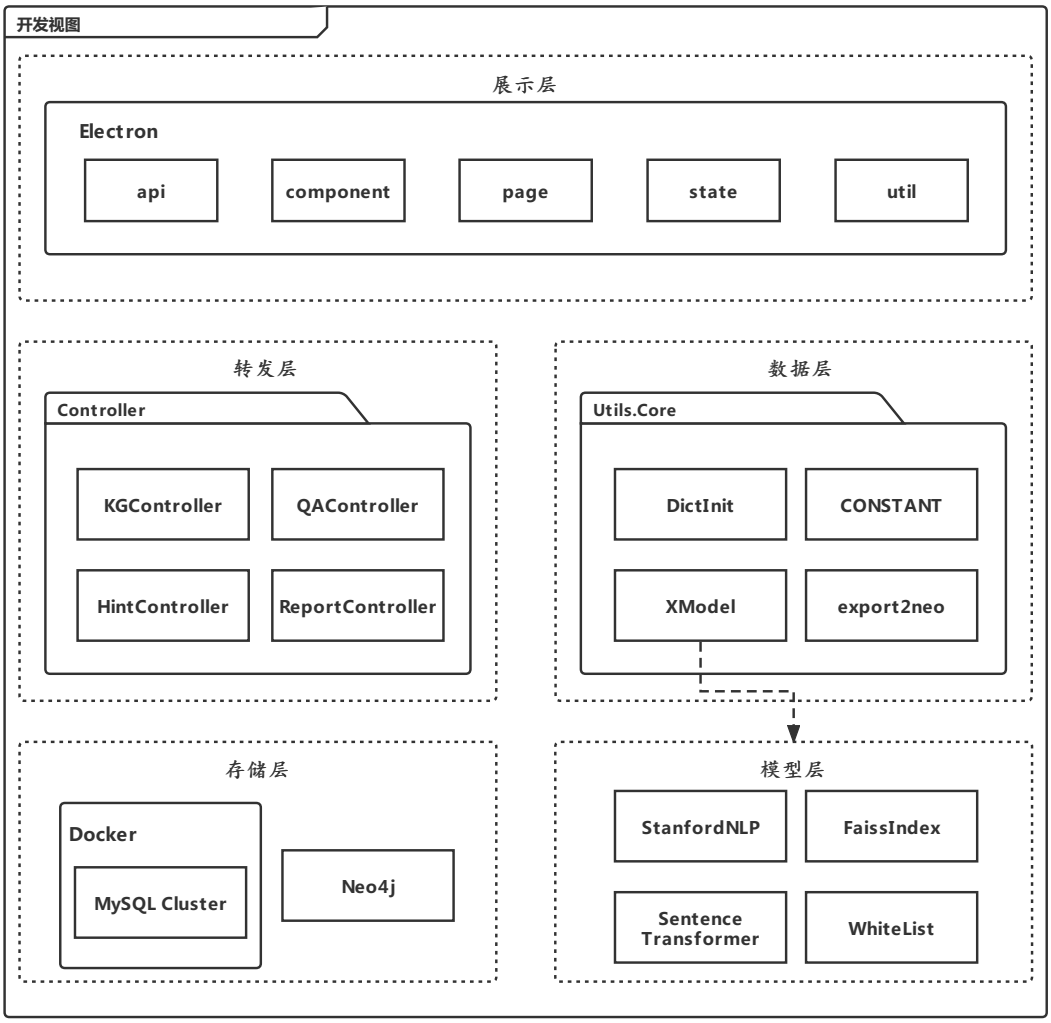



图 3.7: 开发视图

**物理视图**基于运维人员视角，专注于软件系统底层的非功能性需求，如系统物理层面的资源使用、可靠性、可扩展性以及软硬件分布等信息。物理视图将逻辑视图、进程视图和开发视图的网络、进程、任务和对象的标识元素映射

到节点描述，用于解决软件系统的拓扑结构、配置安装和通信方式等问题。本系统的物理视图如图 3.8所示。

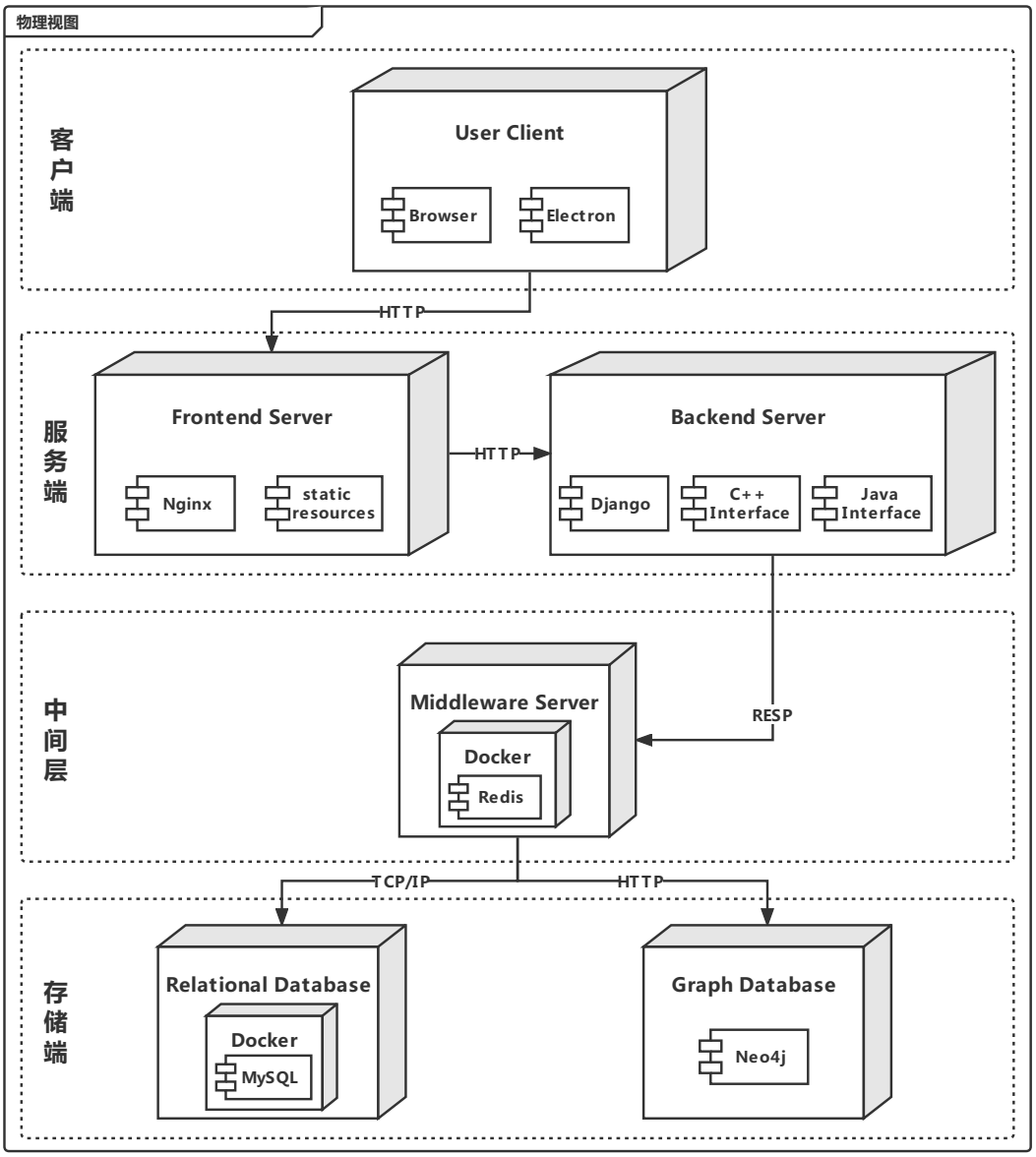



图 3.8: 物理视图

用户通过个人计算机的桌面应用或浏览器发起 HTTP 请求访问 Nginx 服务器；Nginx 开启静态资源压缩，均衡负载策略采用 “ip_hash” 确保 session 的一致性，反向代理到指定后端服务器；前后端通信选用 Axios 处理异步网络请求；后端使用 Django 框架调用 Faiss 的 Java 接口和 SBERT 的 C++ 接口；在 Docker 上部署 Redis 多端口集群缓存区；存储层 MySQL 配置实现主从复制读写分离，通过 py2neo 执行 Neo4j 相关指令。

## 3.4 数据处理与持久化设计

### 3.4.1 数据预处理

数据预处理目的是将原始数据转换为易于理解或者符合需求的结构化数据。本系统面向 Bugzilla 缺陷库，案例数据集是开源项目 Eclipse。源数据由爬虫脚本根据递增的缺陷报告唯一标识符 BugID 补全 URL 路径并发起网络请求，通过解析服务器返回的 HTML 源码标签和过滤无关信息，分析具有潜在价值的标签及其内容并存储为 JSON 格式字段的 CSV 文件。

源数据文件共 3 个字段，分别是 id、infos 和 comment，其中 infos 字段含 38 个属性，如图 3.9所示，是最具价值的字段；id 字段是缺陷报告唯一标识符；comment 字段数组的第一个元素是缺陷描述，往后的元素是若干条用户评论。由于部分缺陷报告涉密，因此源数据开放的 id 并非严格递增且连续。


```
{
   'id': 203357,
   'summary': 'XML and XMI files always transfer binary',
   'classification': 'Tools', 'product': 'Target Management', 'version': '2.0',
   'component': 'RSE', 'platform': 'All', 'op_sys': 'All',
   'severity': 'normal', 'priority': 'P3', 'resolution': 'DUPLICATE', 'status': 'RESOLVED',
   'creator': 'mober.at+eclipse@gmail.com', 'assigned_to': 'dmcknigh@ca.ibm.com',
   'cc': ['dmcknigh@ca.ibm.com', 'jeff.maxwell@gmail.com'], 'qa_contact': 'mober.at+eclipse@gmail.com',
   'creator_detail': {
      'id': 13001, 'name': 'mober.at+eclipse@gmail.com',
      'email': 'mober.at+eclipse@gmail.com', 'real_name': 'Martin Oberhuber'
   },
   'assigned_to_detail': {
      'id': 455, 'name': 'dmcknigh@ca.ibm.com',
      'email': 'dmcknigh@ca.ibm.com', 'real_name': 'David McKnight'
   },
   'cc_detail': [{
      'id': 455, 'name': 'dmcknigh@ca.ibm.com',
      'email': 'dmcknigh@ca.ibm.com', 'real_name': 'David McKnight'
   }, {
      'id': 41756, 'name': 'jeff.maxwell@gmail.com',
      'email': 'jeff.maxwell@gmail.com', 'real_name': 'Jeff Maxwell'
   }],
   'qa_contact_detail': {
      'id': 13001, 'name': 'mober.at+eclipse@gmail.com',
      'email': 'mober.at+eclipse@gmail.com', 'real_name': 'Martin Oberhuber'
   },
   'depends_on': [199773], 'blocks': [203114], 'keywords': ['investigate'], 'flags': [], 'see_also': [],
   'groups': [], 'alias': [], 'url': '', 'whiteboard': '', 'target_milestone': '3.0 M4',
   'creation_time': '2007-09-13T20:20:57Z', 'last_change_time': '2007-11-14T19:12:22Z',
   'deadline': None, 'dupe_of': 203114, 'is_open': False, 'is_confirmed': True,
   'is_creator_accessible': True, 'is_cc_accessible': True
}
```


图 3.9: 缺陷报告源数据 infos 字段示例

源数据文件的 id 字段为数值型数据，infos 字段和 comment 字段均为序列化后的对象数据，主要元素大致可分为六类，以下将以 BugID-203357 作为案例进行说明，参见图 3.9，即：

1. 标识型元素见表 3-11，主要包括：缺陷报告唯一标识项 id、依赖列表项 depends_on、阻塞列表项 blocks、重复列表项 dupe_of 和其他标识项 flags 等。示例报告的唯一标识为 203357；报告的前置依赖列表为 [199773]；报告的前置阻塞列表为 [203114]；缺陷的重复项为 203114；报告的其他标识列表为空。

表 3-11: 标识型元素说明

| 属性名称 | 对应元素 | 元素说明 |
|---|---|---|
| id | id | 当前缺陷报告的唯一标识符 |
| depends | depends_on | 当前缺陷报告的前置依赖项 |
| blocks | blocks | 当前缺陷报告的前置阻塞项 |
| duplicates | dupe_of | 当前缺陷报告的重复项 |
| flags | flags | 缺陷的其他标识 |

2. 基础型元素见表 3-12，主要包括：项目类别项 classification、产品名称项 product、软件版本项 verison、组件名称项 component、平台名称项 platform、操作系统项 op_sys、严重程度项 severity、优先级项 priority、解决情况项 resolution 和缺陷状态项 status 等。示例报告的项目类别为 Tools；所属项目的名称为 Target Management；问题软件版本为 2.0；问题组件的名称为 RSE；能够复现缺陷的平台是 ALL；能够复现缺陷的操作系统是 ALL；缺陷的严重程度是 P3（P1~ P5，数字越小越严重）；缺陷的优先级为 normal；缺陷解决情况是 DUPLICATE；缺陷状态为 RESOLVED。

表 3-12: 基础型元素说明

| 属性名称 | 对应元素 | 元素说明 |
|---|---|---|
| classification | classification | 复现缺陷的项目类别 |
| product | product | 复现缺陷的产品名称 |
| version | version | 复现缺陷的项目版本 |
| component | component | 复现缺陷的组件名称 |
| platform | platform | 复现缺陷的运行平台名称 |
| op_sys | op_sys | 复现缺陷的操作系统 |
| severity | severity | 缺陷的严重程度 |
| priority | priority | 解决缺陷的优先级 |
| resolution | resolution | 缺陷的解决情况 |
| status | status | 缺陷报告的状态 |

3. 角色型元素见表 3-13，主要包括：缺陷创建者项 creator、负责人项 assigned_to、抄送人项 cc 和质检员项 qa_contact 等，detail 元素是对应角色的详

细身份信息。示例报告的缺陷创建者为 mober.at+eclipse@gmail.com；负责人为 dmcknigh@ca.ibm.com；质量检测员为 mober.at+eclipse@gmail.com；抄送人列表为 [dmcknigh@ca.ibm.com, jeff.maxwell@gmail.com]。人员详细信息包括用户唯一标识项 id、用户名项 name、邮箱项 email 和真实姓名项 real_name。

表 3-13: 角色型元素说明

| 属性名称 | 对应元素 | 元素说明 |
|---|---|---|
| creator | creator | 缺陷报告的创建者 |
| assignee | assigned_to | 缺陷的负责人 |
| cc | cc | 缺陷报告的抄送人 |
| QA | qa_contact | 缺陷报告的质量检测人员 |
| creator_detail | creator_detail | 缺陷报告创建者的详细信息 |
| assignee_detail | assigned_to_detail | 缺陷负责人的详细信息 |
| cc_detail | cc_detail | 缺陷报告抄送人的详细信息 |
| QA_detail | qa_contact_detail | 缺陷报告质量检测人员的详细信息 |

4. 描述型元素见表 3-14，主要包括：缺陷报告摘要项 summary、缺陷描述项 description、缺陷评论项 comment、缺陷别名项 alias 和关键词项 keywords 等。示例报告的摘要为 XML and XMI files always transfer binary；缺陷别名列表为空；缺陷关键词为 investigate。缺陷描述和缺陷评论过长而不做展开介绍。

表 3-14: 描述型元素说明

| 属性名称 | 对应元素 | 元素说明 |
|---|---|---|
| summary | summary | 缺陷报告的摘要 |
| description | comment[0] | 缺陷报告的详细描述 |
| comment | comment[1:] | 缺陷的评论 |
| alias | alias | 缺陷的别名 |
| keywords | keywords | 缺陷的关键词 |

5. 状态型元素见表 3-15，主要包括：创建时间项 creation_time、最后修改时间项 last_change_time、开放状态项 is_open、缺陷验证项 is_confirmed、创建者访问权限项 is_creator_accessible 和抄送人访问权限项 is_cc_accessible 等。示例报告的创建时间为 2007-09-13T20:20:57Z；该缺陷最后一次修改时间为 2007-11-14T19:12:22Z；缺陷状态已被关闭；缺陷被确认为真实存在（可复现）；创建者有权访问该报告；抄送人有权访问该报告。

6. 附加型元素见表 3-16，主要包括：相关链接列表项 see_also、分组列表项 groups、网络地址项 url、白板信息项 whiteboard、里程碑项 target_milestone

表 3-15: 状态型元素说明

| 属性名称 | 对应元素 | 元素说明 |
|---|---|---|
| creation | creation_time | 缺陷报告的创建时间 |
| last_change | last_change_time | 缺陷报告最近一次修改时间 |
| open | is_open | 缺陷报告是否开放 |
| confirmed | is_confirmed | 缺陷是否被验证 |
| creator_accessible | is_creator_accessible | 创建者是否有权限访问 |
| cc_accessible | is_cc_accessible | 抄送人是否有权限访问 |

和截止期限项 deadline 等。示例报告的相关链接列表为空；分组列表为空；统一资源定位器为空；白板信息为空；缺陷修复的里程碑为 3.0 M4；缺陷暂时没有截止期限。

表 3-16: 附加型元素说明

| 属性名称 | 对应元素 | 元素说明 |
|---|---|---|
| see_also | see_also | 缺陷相关链接 |
| groups | groups | 缺陷组别 |
| url | url | 缺陷的网络地址 |
| whiteboard | whiteboard | 缺陷报告的白板信息 |
| milestone | target_milestone | 缺陷修复的里程碑 |
| deadline | deadline | 缺陷修复的最后期限 |

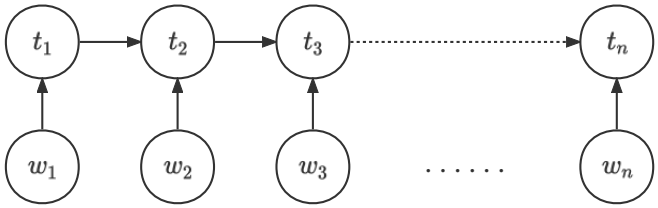
### 3.4.2 文本语义识别

本系统提供两种形式的问答：缺陷报告查重和句式规则匹配。前者解析自然语言描述，根据文本相似度转换为向量化查询；后者映射为标准问题，根据模式匹配规则转换为形式化查询。据此，文本语义识别阶段涉及到的数据处理内容包括词性标注、停用词整理、句向量转换、白名单生成和标准问题构建。

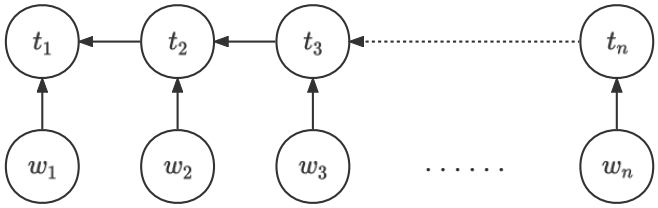
$$score(x) = \prod_i P(x_i|Pa(x_i)) \tag{3-1}$$

词性标注采用 Stanford Log-linear Part-Of-Speech Tagger，这是一个利用泛词汇、双向推理和有效正则等技术手段的富特征模型。当存在平滑效应或其他条件特征（如单词）的相互作用时，如图 3.10a和图 3.10b所示，单向依赖网络提供的语义因子是不够的，即对序列问题的处理不充分。如图 3.10c所示，双向依赖网络不是标准的贝叶斯网络，故使用最大熵模型构建联合概率，通过维特

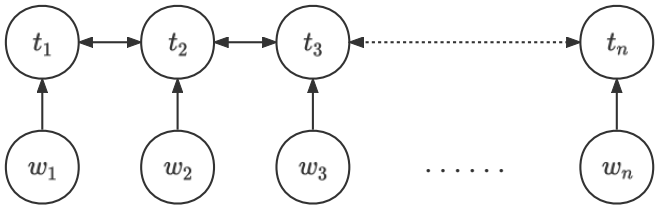
比（Viterbi）算法获得精确最大化的序列，如公式 3-1，其中 $Pa(x)$ 是与节点 $x$ 之间有弧的节点。




(a) 前向条件马尔可夫模型




(b) 后向条件马尔可夫模型




(c) 双向依赖网络

图 3.10: 三种模式依赖图

对示例缺陷报告的摘要进行词性标注，得到结果 [(XML, NN), (and, CC), (XMI, NN), (files, NNS), (always, RB), (transfer, VBP), (binary, JJ)]。为节省存储空间并提高搜索效率，参照 Penn Treebank WSJ 去除部分对语义无影响或影响甚微的次要词性，如图 3.11所示。词性压缩后的剩余文本为 XML XMI files always transfer binary。

```
{
    'CC', 'CDZ', 'DT', 'EX', 'IN', 'LS', 'MD', 'PP', 'PPZ',
    'TO', 'UH', 'WDT', 'WP', 'WPZ', 'WRB', '-LRB-', '-RRB-',
    '-LSB-', '-RSB-', '.'
}
```

图 3.11: Penn Treebank 停用词性

剔除部分语义成分极低的次要词性后还需要整理停用词，以完成第二轮语义压缩——去停用词。停用词是人工输入、非自动化生成的，根据项目真实场

景整理停用词，最终形成停用词表，如图 3.12所示。经过去停用词后的文本为 XML XMI files always transfer binary。

```
{
    '!!', '?!', '??', '!?', '`', '``', "''", '-lrb-', '-rrb-', '-lsb-', '-rsb-', ',', '.', ':',
    ';', '"', "'", '?', '<', '>', '{', '}', '[', ']', '+', '-', '(', ')', '&', '%', '$', '@',
    '!', '^', '#', '*', '..', '...', "'ll", "'s", "'m", 'a', 'about', 'above', 'after', 'again',
    'against', 'all', 'am', 'an', 'and', 'any', 'are', "aren't", 'as', 'at', 'be', 'because',
    'been', 'before', 'being', 'below', 'between', 'both', 'but', 'by', 'can', "can't", 'cannot',
    'could', "couldn't", 'did', "didn't", 'do', 'does', "doesn't", 'doing', "don't", 'down', 'during',
    'each', 'few', 'for', 'from', 'further', 'had', "hadn't", 'has', "hasn't", 'have', "haven't",
    'having', 'he', "he'd", "he'll", "he's", 'her', 'here', "here's", 'hers', 'herself', 'him',
    'himself', 'his', 'how', "how's", 'i', "i'd", "i'll", "i'm", "i've", 'if', 'in', 'into', 'is',
    "isn't", 'it', "it's", 'its', 'itself', "let's", 'me', 'more', 'most', "mustn't", 'my', 'myself',
    'no', 'nor', 'not', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'ought', 'our', 'ours ',
    'ourselves', 'out', 'over', 'own', 'same', "shan't", 'she', "she'd", "she'll", "she's", 'should',
    "shouldn't", 'so', 'some', 'such', 'than', 'that', "that's", 'the', 'their', 'theirs', 'them',
    'themselves', 'then', 'there', "there's", 'these', 'they', "they'd", "they'll", "they're", "they've",
    'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 'very', 'was', "wasn't", 'we',
    "we'd", "we'll", "we're", "we've", 'were', "weren't", 'what', "what's", 'when', "when's", 'where',
    "where's", 'which', 'while', 'who', "who's", 'whom', 'why', "why's", 'with', "won't", 'would',
    "wouldn't", 'you', "you'd", "you'll", "you're", "you've", 'your', 'yours', 'yourself', 'yourselves',
    '###', 'return', 'arent', 'cant', 'couldnt', 'didnt', 'doesnt', 'dont', 'hadnt', 'hasnt', 'havent',
    'hes', 'heres', 'hows', 'im', 'isnt', 'its', 'lets', 'mustnt', 'shant', 'shes', 'shouldnt', 'thats',
    'theres', 'theyll', 'theyre', 'theyve', 'wasnt', 'were', 'werent', 'whats', 'whens', 'wheres', 'whos',
    'whys', 'wont', 'wouldnt', 'youd', 'youll', 'youre', 'youve'
}
```

图 3.12: 停用词表示意图

SBERT 是基于 BERT 的预训练语言表征模型，利用 MLM 生成能融合左右上下文信息的深度双向语言表征。将完成去停用词处理后的文本数据集批量输入至 SBERT 的 Python 库 SentenceTransformer 转换为句向量形式存储，后续使用 Faiss 构建索引生成句嵌入索引模型。

```
{
    'id': 'id', 'bugId': 'id', 'summary': 'summary', 'description': 'description',
    'create_time': 'creation_time', 'creation_time': 'creation_time',
    'create': 'creation_time', 'created': 'creation_time',
    'block': 'block', 'depend': 'depend', 'blocks': 'block', 'depends': 'depend',
    'duplicate': 'duplicate', 'cc': 'cc', 'duplicates': 'duplicate', 'ccs': 'cc',
    'creator': 'creator', 'assignee': 'assignee', 'creators': 'creator', 'assignees': 'assignee',
    'qa': 'qa', 'classification': 'classification', 'qas': 'qa', 'classifications': 'classification',
    'product': 'product', 'component': 'component', 'products': 'product', 'components': 'component',
    'op_sys': 'op_sys', 'os': 'op_sys', 'platform': 'platform', 'platforms': 'platform',
    'status': 'status', 'resolution': 'resolution', 'statuses': 'status', 'resolutions': 'resolution',
    'priority': 'priority', 'severity': 'severity', 'priorities': 'priority', 'severities': 'severity'
}
```

图 3.13: 关键词白名单示意图

关键词在面向领域知识问答场景具有很强的可解释性，通过数据去重和人工标注，生成的关键词白名单如图 3.13。利用关键词白名单能快速定位权重语义词，在句式规则匹配问答中尤为重要而有效。

自然语言句式提问是基于模式匹配的问答，核心任务是标准问题模板的构建。本系统支持特殊疑问句、一般疑问句、反意疑问句、选择疑问句、陈述句和多跳疑问句六种句型的提问形式，句式结构及示例问句见表 3-17。

表 3-17: 标准问题模式匹配规则

| 句式名称 | 句式结构 | 示例问句 |
|---|---|---|
| 特殊疑问句 | ^wh(ose\|at\|ich\|ere\|en\|y\|o)\|^how.m | How many status in system? |
| 一般疑问句 | is\|are\|was\|were | #203357's platform is PC? |
| 反意疑问句 | ,.*(is\|are)n.?t | #203357 is RESOLVED, isnt it? |
| 选择疑问句 | #\d+.*\bor\b | #203357's platform is PC or Other? |
| 陈述句 | ^#\d+(?!.*(is\|are\|was\|were)) | #203357's cc. |
| 多跳疑问句 | ^how.?m.+#\d+\'s.+ | How many bugs belong to #203357's status? |

### 3.4.3 本体层设计

本体建模的目的是抽象表达特定领域场景下的实体或概念及其属性相互间的关系。使用“模式 & 数据图”描述本体建模如图 3.14所示，上半部分表示模式图 $G_{OWL}$，下半部分表示数据图 $G_{RDF}$，中间虚线部分为关系 *Relation*，概念的内部特征用属性来表示，实体的外部联系用关系来表示。

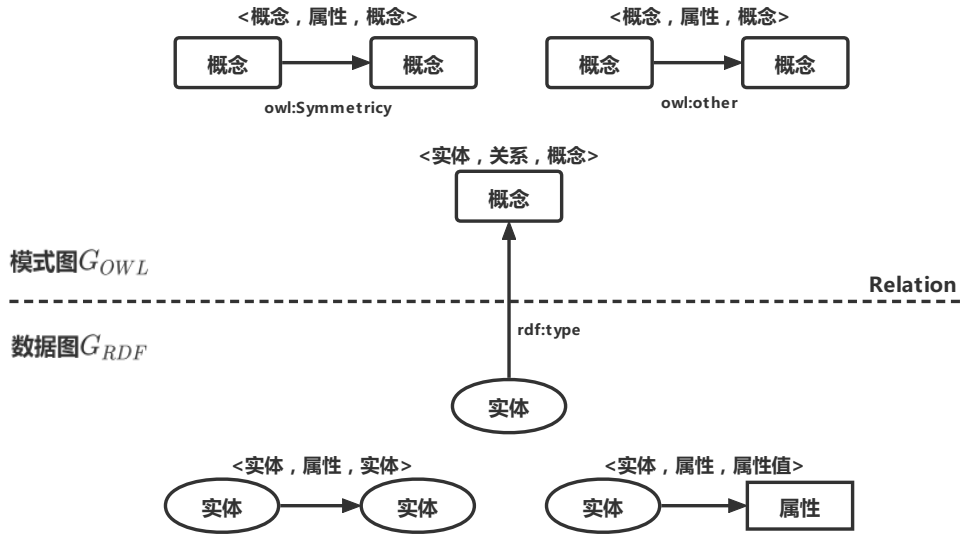



图 3.14: 模式数据概念图

模式图 $G_{OWL}$ 的节点表示本体，本体概念可以是领域内的类别或个体；边表示关联概念间的语义关系，属性边可以是语义网络或本体语言的属性，也可

以是自定义属性。数据图 $G_{RDF}$ 的节点分为实体节点和属性节点，实体是真实世界的具体事物；属性值是事物内在特征的描述，具有原子性；边分为外在属性（关系）和内在属性，是三元组蕴含语义的成分。

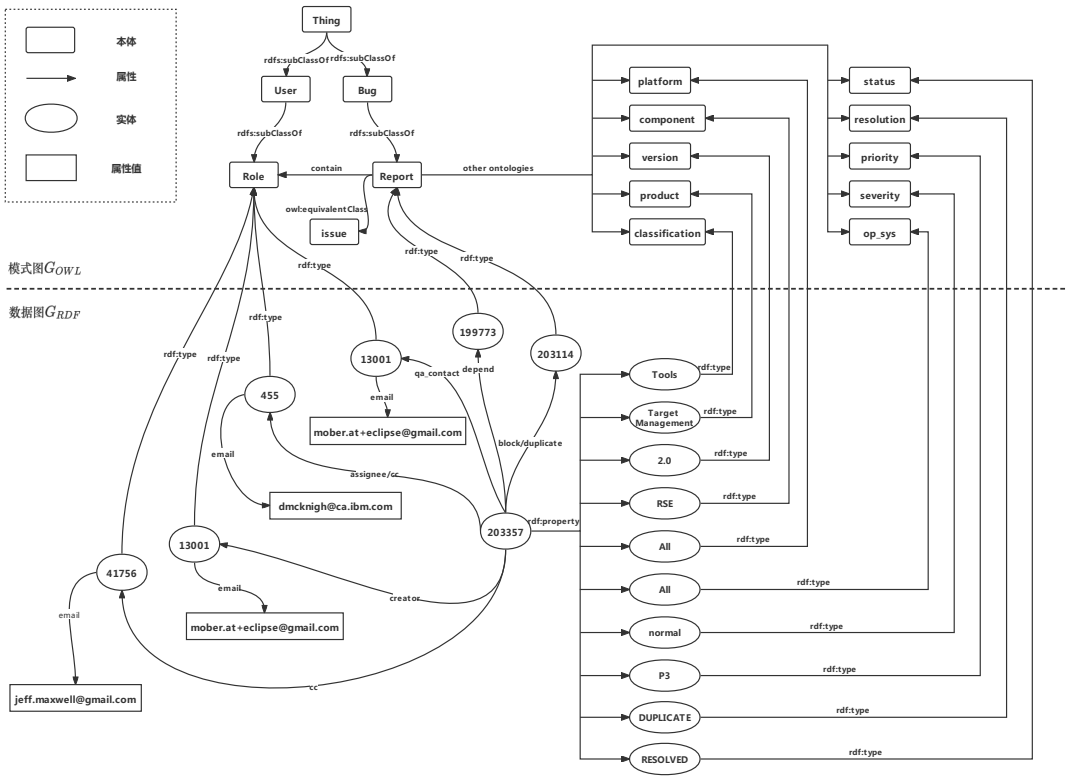



图 3.15: 模式数据实例图

本体的定义过程如图 3.15所示。上半部分是 Bugzilla 缺陷管理系统 Eclipse 项目场景的“本体-属性”模式图，其中本体 Role 表示“角色”的概念；本体 Report 表示“缺陷报告”的概念；本体 issue 是 Report 非结构化源数据；本体 User 表示“用户”的概念；本体 Bug 表示“缺陷”的概念；Thing 是 OWL 层次树的根。此外，本体 platform 表示“平台”的概念；本体 component 表示“组件”的概念；本体 version 表示“产品版本”的概念；本体 product 表示“产品名称”的概念；本体 classification 表示“类别”的概念；本体 status 表示“缺陷状态”的概念；本体 resolution 表示“解决情况”的概念；本体 priority 表示“优先级”的概念；本体 severity 表示“严重程度”的概念；本体 op_sys 表示“操作系统”的概念。

下半部分是“实体-属性”数据图，以下仍以 BugId-203357 作为案例进行说明。

内在属性的三元组形式为 <Node, rdf:property, owl:Class>，如 userId 为 13001

的用户及其email构成的三元组为<13001, email, mober.at+eclipse@gmail.com>。内在属性涉及缺陷唯一标识、角色信息和报告内容信息，抽象概念关联抽象概念，实例化事物关联实例化事物。

外在属性的三元组形式为 <Node, rdf:type, owl:Class>，如 BugId 为 203357 的缺陷报告及其创建者的 userId 构成的三元组为 <203357, creator, 13001>。外在属性是实体和概念的相互映射，例如案例的“status”概念被实例化为“RESOLVED”实体。

表 3-18: 三元组定义

| 形式 | 名称 | 说明 |
|---|---|---|
| <bug, bug2bug, bug> | bug2block | tail bug 是 head bug 的阻塞项 |
| | bug2depend | tail bug 是 head bug 的依赖项 |
| | bug2duplicate | head bug 和 tail bug 互为重复项 |
| <bug, bug2user, user> | bug2cc | user 是 bug 的抄送人 |
| | bug2creator | user 是 bug 的创建者 |
| | bug2assignee | user 是 bug 的负责人 |
| | bug2qa | user 是 bug 的质量检测人 |
| <bug, bug2node, node> | bug2classification | node 是 bug 所属的项目类别 |
| | bug2product | node 是 bug 所属的产品名称 |
| | bug2component | node 是 bug 所属的组件名称 |
| | bug2op_sys | node 是可复现 bug 的操作系统 |
| | bug2platform | node 是可复现 bug 的平台名称 |
| | bug2status | node 是 bug 的状态 |
| | bug2resolution | node 是 bug 的解决情况 |
| | bug2priority | node 是 bug 的优先级 |
| | bug2severity | node 是 bug 的严重程度 |

知识图谱三元组 <head, relation, tail> 抽取，如表 3-18所示，主要包括三种形式 <bug, bug2bug, bug>、<bug, bug2user, user>、<bug, bug2node, node>。

1.<bug, bug2bug, bug>，关系 bug2bug 指 head bug 与 tail bug 相同类型实体之间的关系，具体包括：阻塞 bug2block、依赖 bug2depend 和重复 bug2duplicate。

2.<bug, bug2user, user>，关系 bug2user 指 bug 与 user 不同类型实体之间的关系，具体包括：抄送人 bug2cc、创建者 bug2creator、负责人 bug2assignee 和 bug2qa。

3.<bug, bug2node, node>，关系 bug2node 指 bug 与 node 之间的“种属-实体”对应关系，具体包括：项目分类 bug2classification、产品名称 bug2product、组件名称 bug2component、操作系统 bug2op_sys、平台名称 bug2platform、缺

陷状态 bug2status、解决情况 bug2resolution、优先级 bug2priority 和严重程度 bug2severity。

表 3-19: Neo4j 实体文件表

| | | | |
|---|---|---|---|
| block.csv | bug.csv | classification.csv | component.csv |
| depend.csv | duplicate.csv | op_sys.csv | platform.csv |
| priority.csv | product.csv | resolution.csv | severity.csv |
| status.csv | user.csv | | |

根据本体和三元组定义的结果，创建实体文件夹和关系文件夹，对源文件进行数据清洗，筛选目标字段并生成结构化数据文件，输出内容如表 3-19和表 3-20，最终将知识图谱导入图数据库 Neo4j。

表 3-20: Neo4j 关系文件表

| | | | |
|---|---|---|---|
| bug2assignee.csv | bug2block.csv | bug2cc.csv | bug2classification.csv |
| bug2component.csv | bug2creator.csv | bug2depend.csv | bug2duplicate.csv |
| bug2op_sys.csv | bug2platform.csv | bug2priority.csv | bug2product.csv |
| bug2qa.csv | bug2resolution.csv | bug2severity.csv | bug2status.csv |

### 3.4.4 持久化存储

持久化的目标是将内存中的数据对象存储到数据库，即数据模型转换为存储模型的过程。缺陷报告业务数据存储于关系型数据库，通过字段名、数据类型、含义和备注来解释说明。

表 user 用于保存用户的基本信息，包括用户唯一标识、用户名、用户邮箱和真实姓名。具体内容如表 3-21所示。

表 3-21: 表 user 结构设计

| 字段名 | 数据类型 | 含义 | 备注 |
|---|---|---|---|
| id | INT(11) | 用户唯一标识 | 自增主键 |
| name | VARCHAR | 用户名 | |
| email | VARCHAR | 用户邮箱 | |
| realName | VARCHAR | 真实姓名 | |

表 report 用于保存缺陷报告的基本信息，包括用户唯一标识、缺陷报告摘要、项目类别、产品名称、组件名称、操作系统、运行平台、缺陷状态、解决情况、优先级、严重程度和创建日期。具体内容如表 3-22所示。

表 3-22: 表 report 结构设计

| 字段名 | 数据类型 | 含义 | 备注 |
|---|---|---|---|
| id | INT(11) | 缺陷唯一标识 | 自增主键 |
| summary | INT(11) | 缺陷报告摘要 | 外键，关联 summary 表 |
| classification | INT(11) | 项目类别 | 外键，关联 classification 表 |
| product | INT(11) | 产品名称 | 外键，关联 product 表 |
| component | INT(11) | 组件名称 | 外键，关联 component 表 |
| op_sys | INT(11) | 操作系统 | 外键，关联 opsys 表 |
| platform | INT(11) | 运行平台 | 外键，关联 platform 表 |
| status | INT(11) | 缺陷状态 | 外键，关联 status 表 |
| resolution | INT(11) | 解决情况 | 外键，关联 resolution 表 |
| priority | INT(11) | 优先级 | 外键，关联 priority 表 |
| severity | INT(11) | 严重程度 | 外键，关联 severity 表 |
| create_time | DATE | 创建日期 | 自动生成 |

表 report2user 用于保存“缺陷报告-用户角色”关系的基本信息，包括主键 id、缺陷唯一标识 bugId、用户唯一标识 userId 和用户在缺陷报告中扮演的角色名称。具体如表 3-23所示。

表 3-23: 表 report2user 结构设计

| 字段名 | 数据类型 | 含义 | 备注 |
|---|---|---|---|
| id | INT(11) | 主键 | 自增主键 |
| bugId | INT(11) | 缺陷唯一标识 | 外键，关联 bug 表 |
| userId | INT(11) | 用户唯一标识 | 外键，关联 user 表 |
| roleName | VARCHAR | 用户缺陷的关系 | 角色名称 |

表 report2report 用于保存“缺陷报告-缺陷报告”关系的基本信息，包括主键、首端缺陷唯一标识 headReport、尾端缺陷唯一标识 tailReport 和缺陷间的关系。具体如表 3-24所示。

表 3-24: 表 report2report 结构设计

| 字段名 | 数据类型 | 含义 | 备注 |
|---|---|---|---|
| id | INT(11) | 主键 | 自增主键 |
| headReport | INT(11) | 缺陷唯一标识 | 外键，关联 report 表 |
| tailReport | INT(11) | 缺陷唯一标识 | 外键，关联 report 表 |
| relation | VARCHAR | 缺陷间的关系 | 关系名称 |

其他属性关联表结构简单且受限于篇幅，故不作展开说明。

## 3.5 本章小结

本章主要分析系统的业务功能需求，以期完成核心模型和系统架构的设计。首先，对面向 Bugzilla 知识图谱的问答系统进行整体概述；其次，对系统进行涉众分析，分别从功能性需求和非功能性需求概述系统的需求和目标；然后，结合系统用例图和用例表阐述系统功能；在明确系统的整体架构后，基于架构图和“4 + 1”视图对系统的技术选型和体系结构进行详细说明；最后，围绕业务需求论述模型构建和持久化存储阶段涉及到的数据结构，给出数据预处理、文本语义识别、本体层定义和持久化设计的总体思路。

# 第四章　详细设计与编码实现

本章主要介绍面向 Bugzilla 的知识图谱问答系统的详细设计与编码实现。项目采取模块化的研发思维，把系统划分成若干个模块，每个模块实现一个功能，单独对这些功能模块进行持续集成、持续部署，以满足和丰富用户的需求。根据第三章的需求分析、总体设计、数据处理和持久化设计方案，本系统的业务功能划分为原始数据清洗、语义模型训练、知识图谱构建、缺陷报告管理和问答系统开发。下面，将从以上五个模块逐个进行详细说明。

## 4.1　原始数据清洗模块

### 4.1.1　详细设计

Bugzilla 数据集是关于开源项目 Eclipse 在开发、测试或使用过程中发现的可复现缺陷报告的汇总。源数据是利用爬虫对公开缺陷报告的超文本标记语言进行解析、转换和存储等一系列操作后的持久化文件。为保证数据价值最大化，数据清洗的目标如下：

1. 数据扁平化，原始数据对象存在嵌套深、无效嵌套等问题。因此需要去除冗余、厚重和繁杂的过度装配，减少层次、精简数据。

2. 主成分提炼，原始数据存在大量无关信息的问题。通过数据清洗保留项目相关的字段和属性，从而实现降维并提高数据处理的速度。

3. 磁盘空间压缩，原始数据中不必要的缺陷评论占据大量磁盘空间的问题。通过数据清洗释放磁盘空间，降低服务器性能门槛。

4. 数据规范化，原始数据存在数据类型不对齐、图片乱码等问题。通过数据清洗转换数据类型、置空图片乱码等处理，预防后续数据库写入操作时类型报错、非法输入等潜在问题。

原始数据清洗主要分为两个阶段：第一阶段，源文件数据清洗，根据项目需求对原始数据进行字段清洗和属性筛选，将完成清洗的数据持久化为逗号分隔值文件（CSV 文件）。第二阶段，目标文件分装，按照本体和三元组定义从上一阶段的输出文件派生出缺陷摘要文件、实体文件和关系文件。

数据清洗的两个阶段均属于预处理阶段，由持续集成人员负责执行。原始数据清洗模块的活动图如图 4.1所示，描述原始数据清洗的交互活动。用户

（持续集成人员）配置源文件路径、输出文件路径以及各项参数，通过合法性校验后分块读取、逐行处理源文件数据；对按照预设规则处理后的数据作追加存储；以分块逐行的方式读取清洗后的源文件数据，结合本体归纳和三元组定义选取指定字段和属性；按实体名称和关系名称分装成结构化数据文件，同时单独生成缺陷摘要文件。

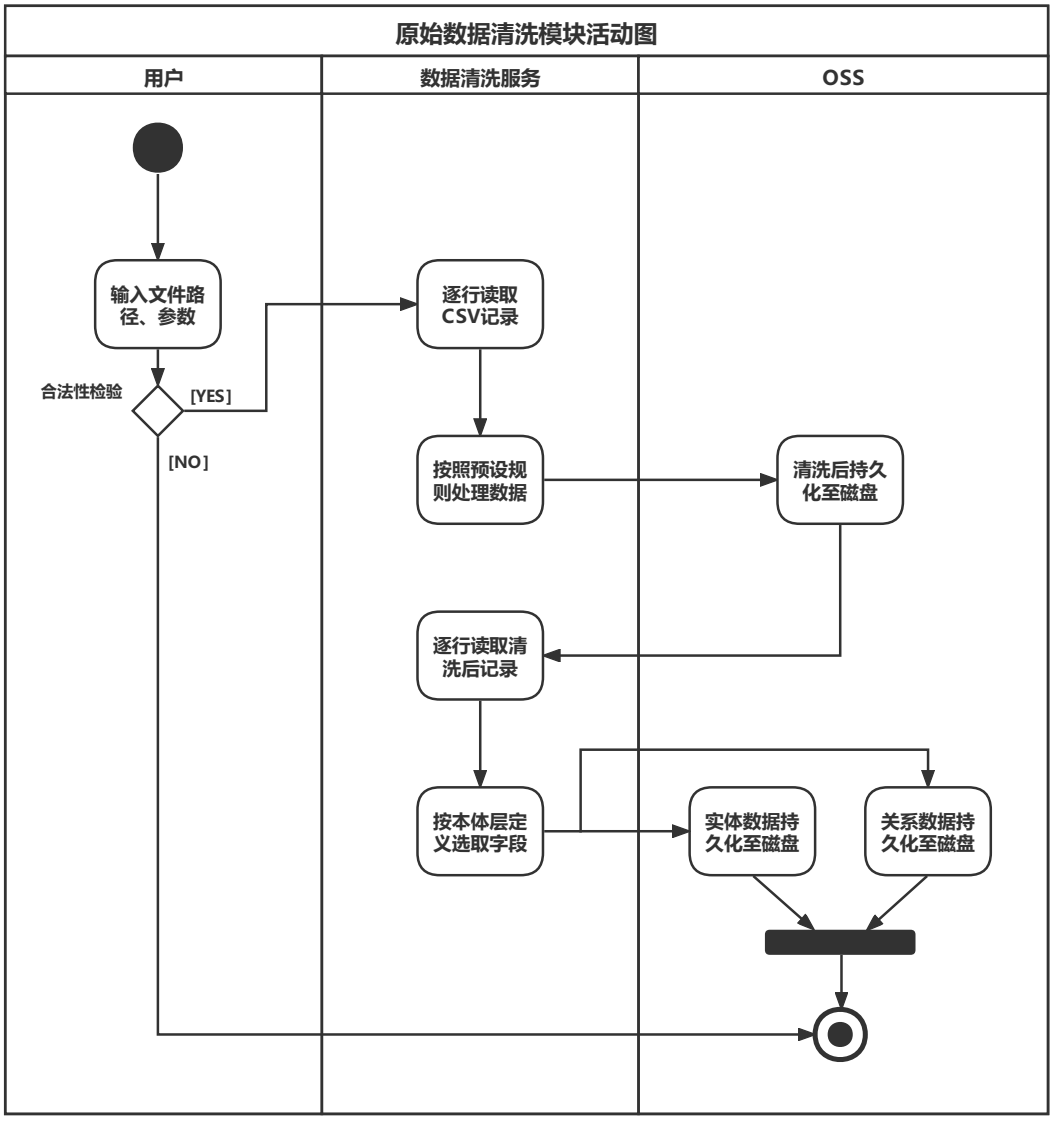



图 4.1: 原始数据清洗活动图

原始数据清洗模块的时序图如图4.2所示，描述原始数据清洗的行为顺序。持续集成人员首先利用 pandas 库的 read_csv() 函数分块读取源数据文件并创建 DataFrame 数据帧，通过迭代器遍历数据帧并生成行索引及行对象；接着执行源数据清洗算法，将清洗后的数据持久化为 CSV 文件；然后在上一步骤的基础

上，结合本体层和三元组定义，利用实体生成服务的 export_nodes() 函数和关系生成服务的 export_relations() 函数合成数据块，通过 write2csv() 函数分别追加写入缺陷报告摘要文件、实体文件夹和关系文件夹。

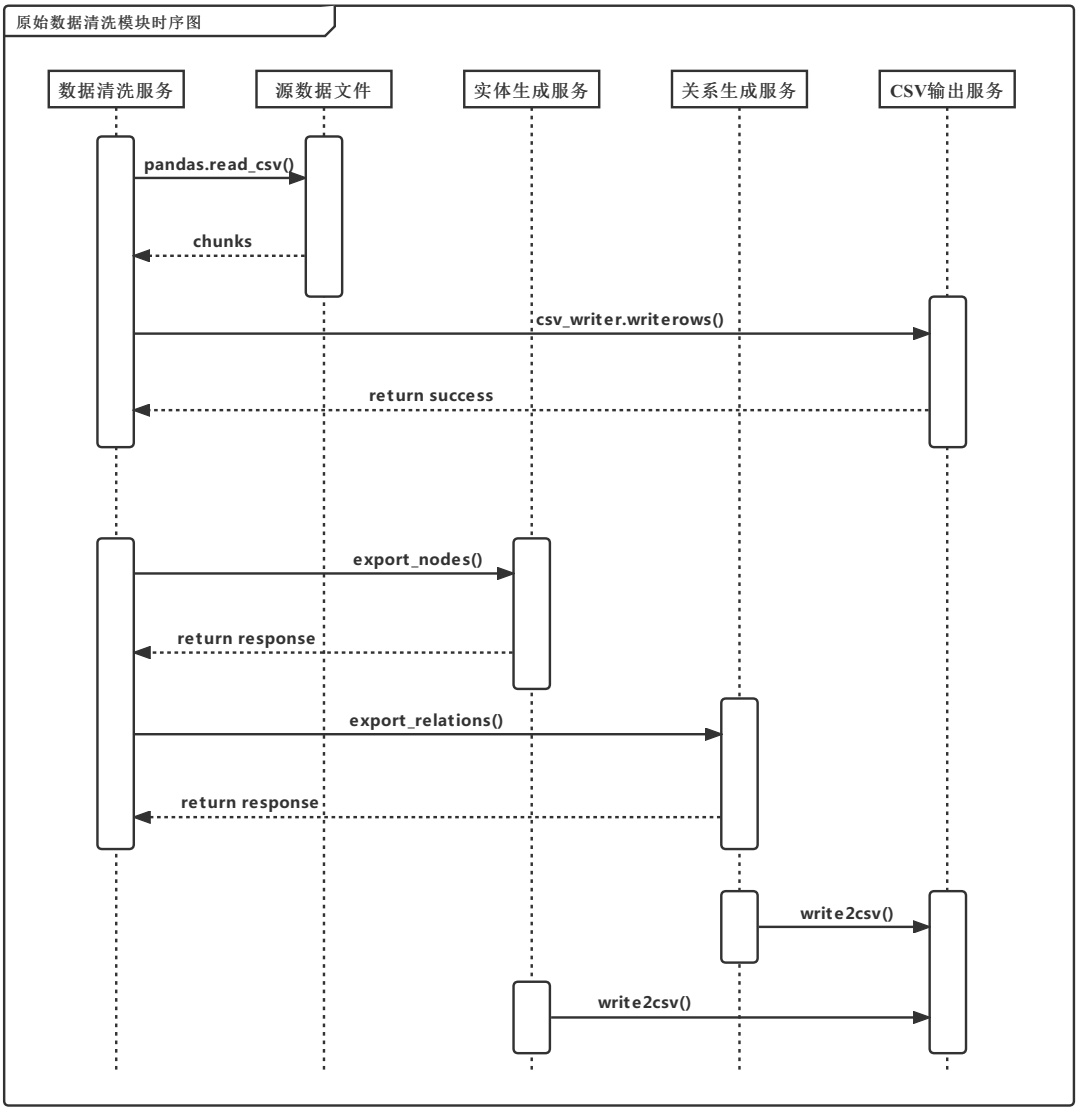



图 4.2: 原始数据清洗时序图

源文件数据清洗的实现方案如算法1所示。算法的输入是 Bugzilla 缺陷管理系统 Eclipse 开源项目数据集，由若干记录组成的源文件，字段格式为 JSON；算法的输出是完成数据清洗的结构化 CSV 文件。分块读取源文件，获取其中的 bugId、summary、description 以及按需筛选的 properties 属性并暂存至列表 chunkArr。每处理完一个单位的 chunkSize，将之以追加的方式写入结构化 CSV 文件中。

**Algorithm 1:** 源文件数据清洗算法

```
input : Source Data CSV
output: Target Data CSV
Function Main:
    for chunks ← 1 to n do
        chunkArr = [ ];
        for index, chunk in Iterator(chunks) do
            bugId, summary ← chunk.id, chunk.summary;
            description ← chunk.comment [0];
            properties ← SelectProperty(chunk.info);
            data ← [bugId, summary, description, properties];
            ListAppend(chunkArr, data);
        end
        csv_writerows(chunkArr);
    end
```

生成实体文件的方案如算法2所示。算法的输入是完成源文件数据清洗后的结构化 CSV 文件；算法的输出是符合本体层定义的目标实体文件。通过识别实体的类型，分发至 ExportBugs()、ExportUsers()、ExportProperties() 和 single_column() 函数加工处理，最终生成对应实体名称的结构化文件。

**Algorithm 2:** 实体文件生成算法

```
input : Source Data CSV
output: Target Node CSV
Function Main:
    if node is bug then
        res ←ExportBugs(data);
    else if node is user then
        res ←ExportUsers(data);
    else if node is property then
        res ←ExportProperties(data);
    else
        res ←single_column(data);
    csv_writerows(res);
```

生成关系文件的方案如算法3所示。算法的输入是完成源文件数据清洗后的结构化 CSV 文件；算法的输出是符合三元组定义的目标关系文件。通过识别 tailNode 的类型，分别由 Bug2BugTemplate()、Bug2CC() 和 single_column() 函数负责处理，最终生成对应关系名称的结构化文件。

**Algorithm 3:** 关系文件生成算法

**input :** Source Data CSV
**output:** Target Relation CSV
**Function** Main**:**
  **if** tailNode is bug **then**
    res ←Bug2BugTemplate(data) ;
  **else if** tailNode is cc **then**
    res ←Bug2CC(data) ;
  **else**
    res ←single_column(data) ;
  csv_writerows(res) ;

### 4.1.2 核心代码

源文件数据清洗文件操作代码如图 4.3所示。文件操作代码主要负责配置输出文件路径、文件打开模式和 CSV 文件的字段名。源代码实现的功能为打开文件并返回文件流，采用 UTF-8 编码以追加的形式写入记录，初始化文件字段名为 id、summary、description 和 properties，并统计总写入行数。

```
csv_file = open(output_file, 'a', newline='', encoding='utf-8')
csv_writer = csv.writer(csv_file, dialect='excel')
csv_headers = ['id', 'summary', 'description', 'properties']
csv_writer.writerow(csv_headers)
rows = 0

... data cleaning core code ...

csv_file.close()
```

图 4.3: 源文件数据清洗文件操作代码

源文件数据清洗核心代码如图 4.4所示。数据清洗核心代码主要负责检查数据合法性、处理缺失值、压缩无效值和应对特殊值。源代码实现的功能为利

用 pandas 库分块读取源文件数据，使用生成器对块内记录进行迭代遍历。首先对 comments 做扁平化处理；其次使用正则表达式置空二进制图片数据；最后序列化指定字段属性，持久化至 CSV 结构化文件。

```
for chunks in pandas.read_csv(input_file, chunksize=chunkSize, names=['id', 'infos', 'comment']):
    chunkArr = []
    for index, chunk in chunks.iterrows():
        bugId = chunk['id']
        try:
            comments = next(iter(json.loads(chunk['comment'])['bugs'].values()))['comments']
            description = 'No Description.'
            if len(comments) > 0:
                description = re.sub(r'begin(.*)\.gif(.|\n)*\nend', '', comments[0]['raw_text'])
            infos = eval(chunk['infos'])
            summary = re.sub(r'%', ' percent', infos['summary'])
            properties = {
                "classification": infos['classification'],
                "product": infos['product'] + ' ' + infos['version'],
                "component": infos['component'],
                "op_sys": infos['op_sys'],
                "platform": infos['platform'],
                "status": infos['status'],
                "resolution": infos['resolution'],
                "priority": infos['priority'],
                "severity": infos['severity'],
                "creator": infos['creator'],
                "assignee": infos['assigned_to'],
                "QA": infos['qa_contact'],
                "creation_time": infos['creation_time'],
                "blocks": infos['blocks'],
                "depends": infos['depends_on'],
                "duplicates": infos['dupe_of'],
                "cc": infos['cc'],
            }
            if not summary:
                continue
            chunkArr.append([bugId, summary, description, properties])
            rows += 1
        except:
            print(bugId)
            print(chunk['comment'])
    csv_writer.writerows(chunkArr)
```

图 4.4: 源文件数据清洗核心代码

目标实体文件生成代码如图 4.5所示。目标实体文件生成代码主要根据本体层定义生成实体文件，为后续知识图谱构建提供实体数据支持。根据实体类型，由 export_bugs()、export_user()、bugs_template() 和 single_column() 四个函数分别处理。export_bugs() 处理缺陷实体，包括 bug 及其属性；export_user() 处理用户角色实体，包括 creator、cc、qa 和 assignee；bugs_template() 处理缺陷类实体，包括 block、depend 和 duplicate；single_column() 处理属性类实体，包括 classification、product、version、component、operation system、status、resolution、priority 和 severity。上述功能函数均是以分块读取文件的形式，遍历块内记录，根据数据特征提取蕴含价值的字段属性并生成实体文件，对特殊情况进行额外处理。

目标关系文件生成代码如图 4.6所示。目标关系文件生成代码主要根据三

```
def export_bugs(self) -> str:
    res, ids = [], set()
    for chunks in pandas.read_csv(self.filepath, names=self.header, **self.csv_config):
        for _, chunk in chunks.iterrows():
            bug_id = chunk['id']
            summary = chunk['summary']
            description = chunk['description']
            creation_time = eval(chunk['properties'])['creation_time']
            ids.add(bug_id)
            res.append([bug_id, summary, description, creation_time])
    write2csv('./nodes/bug.csv', ['id', 'summary', 'description', 'creation_time'], res)
    return 'bug done'

def export_user(self) -> str:
    res = set()
    for chunks in pandas.read_csv(self.filepath, names=self.header, **self.csv_config):
        for _, chunk in chunks.iterrows():
            properties = eval(chunk['properties'])
            creator = properties['creator']
            cc = properties['cc']
            qa = properties['QA']
            assignee = properties['assignee']
            res = res | {creator, qa, assignee} | set(cc)
    res.discard('')
    data = [[r] for r in res]
    write2csv('./nodes/user.csv', ['user'], data)
    return 'user done'


def bugs_template(self, prop) -> list:
    res, ids = [], set()
    for chunks in pandas.read_csv(self.filepath, names=self.header, **self.csv_config):
        for _, chunk in chunks.iterrows():
            try:
                target_column = eval(chunk['properties'])[prop]
                if target_column:
                    if isinstance(target_column, int):
                        if target_column not in ids:
                            res.append([target_column, ' ', ' ', ' '])
                            ids.add(target_column)
                    else:
                        res.extend([[bugId, ' ', ' ', ' '] for bugId in target_column if bugId not in ids])
                        ids |= set(target_column)
            except:
                print(chunk['properties'])
    return res

def single_column(self, prop) -> list:
    res = set()
    for chunks in pandas.read_csv(self.filepath, names=self.header, **self.csv_config):
        for _, chunk in chunks.iterrows():
            target_column = eval(chunk['properties'])[prop]
            if target_column:
                res.add(target_column)
    res.discard('')
    return [[r] for r in res]

... ...
```

图 4.5: 目标实体文件生成代码

元组定义生成关系文件，为后续知识图谱构建提供关系数据支持。根据关系类型，由 bug2bug_template()、single_column() 和 bug2cc() 三个函数分别进行处理。bug2bug_template() 处理缺陷之间的关系，包括 bug2block、bug2depend 和 bug2duplicate；bug2cc() 处理缺陷与抄送人 · 之间的关系，包括 bug2cc，由于“抄送人”为列表类型，因此需要作特殊处理；single_column() 处理剩余关系，包括 bug2classification、bug2component、bug2creator、bug2assignee、bug2status、

bug2op_sys、bug2product、bug2platform、bug2qa、bug2severity、bug2priority 以及 bug2resolution。上述功能函数均是以分块读取文件的形式，遍历块内记录，根据数据特征生成三元组关系文件。

```
def bug2bug_template(self, prop) -> list:
    res = []
    for chunks in pandas.read_csv(self.filepath, names=self.header, **self.csv_config):
        for _, chunk in chunks.iterrows():
            bug_id = chunk['id']
            target_column = eval(chunk['properties'])[prop]
            if target_column:
                if isinstance(target_column, int):
                    res.append([bug_id, target_column])
                else:
                    res.extend([[bug_id, bid] for bid in target_column])
    return res

def single_column(self, prop) -> list:
    res = []
    for chunks in pandas.read_csv(self.filepath, names=self.header, **self.csv_config):
        for _, chunk in chunks.iterrows():
            bug_id = chunk['id']
            target_column = eval(chunk['properties'])[prop]
            if target_column:
                res.append([bug_id, target_column])
    return res

def bug2cc(self) -> str:
    res = []
    for chunks in pandas.read_csv(self.filepath, names=self.header, **self.csv_config):
        for _, chunk in chunks.iterrows():
            bug_id = chunk['id']
            target_column = eval(chunk['properties'])['cc']
            if target_column:
                for cc in target_column:
                    res.append([bug_id, cc])
    write2csv('./relations/bug2cc.csv', ['id', 'cc'], res)
    return 'bug2cc done'
... ...
```

图 4.6: 目标关系文件生成代码

### 4.1.3 结果说明

表 4-1: 源文件数据清洗前后对比

| 对比项 | 数据清洗前 | 数据清洗后 |
|---|---|---|
| 扁平化程度 | 有冗余嵌套层 | 无冗余嵌套层 |
| 属性总数 | 40 | 21 |
| 磁盘空间 | 10GB | 700MB |
| Cyper 规范度 | 出现类型报错 | 无任何报错 |

源文件数据清洗的前后对比如表 4-1所示。通过数据清洗，剔除冗余嵌套层，实现对象扁平化；结合项目需求，选取主要字段和属性，属性数量从 40 个减少至 21 个；充分释放磁盘空间，删除以用户评论为主的非强相关信息，数据

量从 10GB 压缩至 700MB；规范化数据条目，使之转换为 Neo4j 和 MySQL 支持的数据类型。源文件数据清洗的输出结果为目标文件分装的输入数据，有助于节省时间并提高效率，有更强的可理解性。

## 4.2 语义模型训练模块

### 4.2.1 详细设计

预训练的句嵌入索引模型主要用于缺陷报告的查重以及缺陷唯一标识的获取。由于 Bugzilla 数据集属于限定领域场景的技术性信息描述，因此缺陷报告摘要在语义压缩后的每个词单元均具有明显的专业领域特征。语义压缩示意图如图 4.7所示。针对每个输入的缺陷报告摘要，语义压缩处理步骤如下:

1. 文本分词，将输入文本按规则进行分词处理，便于提取文本的特征值，为文本提供特征值对比的词组。

2. 词性标注，在分词的基础上对每个单词根据上下文信息进行标注，对照自定义的无关词性表第一轮过滤介词、拟声词等虚词。

3. 去停用词，根据自定义的停用词表第二轮过滤没有意义的单词，如符号、助词、语气词等。

4. 输出词组，词组是列表形式的若干单词，是执行文本分词、词性标注和去停用词等操作后具有语义成分的单词集合。

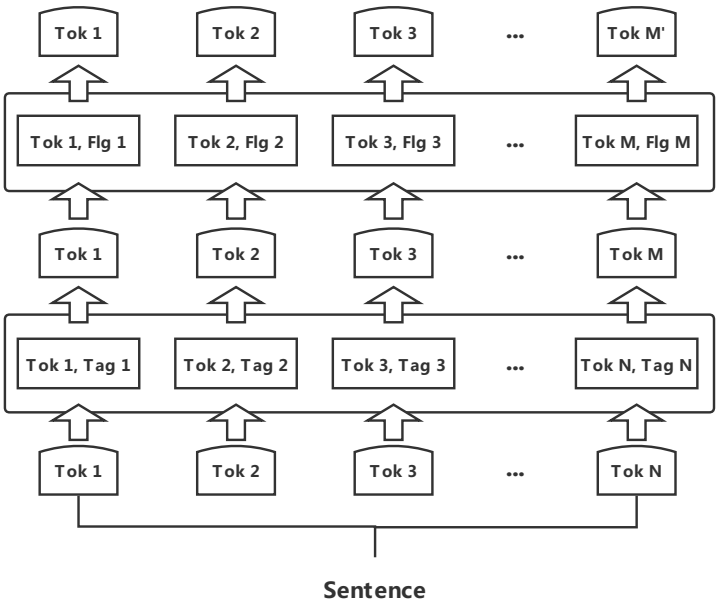



图 4.7: 语义压缩示意图

语义模型训练模块的活动图如图4.8所示，描述语义模型训练的交互活动。用户（持续集成人员）配置源文件路径、模型输出路径以及各项参数；通过合法性校验后，下载选中的 BERT STS 预训练模型，同时加载白名单和停用词；分块读取、逐行处理源文件数据；缺陷摘要经过语义压缩后执行句嵌入算法，计算并序列化当前词组的句向量；在句嵌入模型的基础上构建索引，将训练得到的句嵌入索引模型持久化至硬盘。

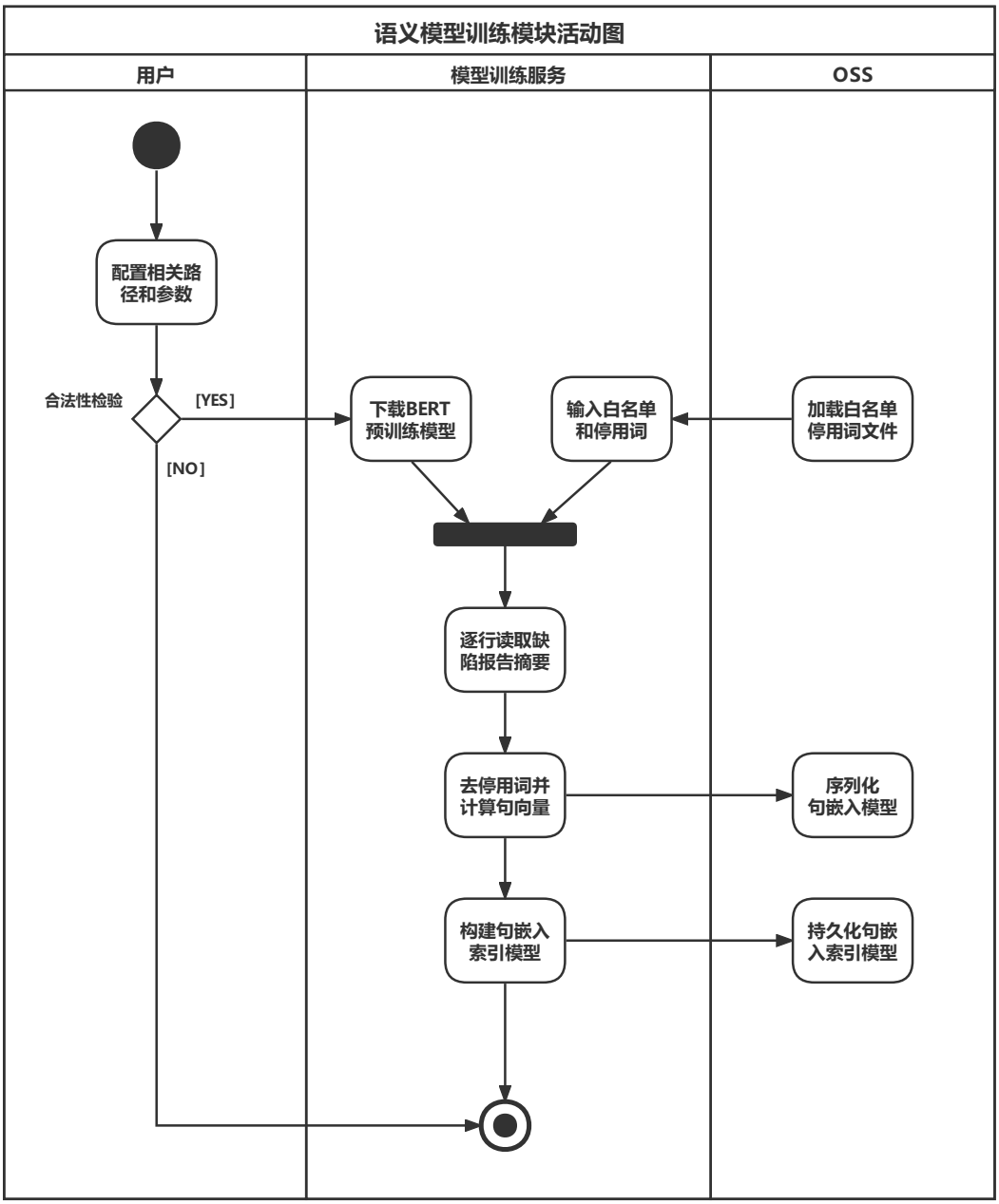



图 4.8: 语义模型训练活动图

语义模型训练模块的时序图如图4.9所示，描述语义模型训练的行为顺序。持续集成人员按步骤先向 StanfordNLP 服务传输完成文本分词后的缺陷摘要词组；其次，StanfordNLP 服务调用 pos_tag() 函数对词组进行词性标注，并使用 init_stopwords() 函数对词组去停用词；然后，利用 sentence_bert.encode() 函数计算剩余词组的句向量，并用 pickle.dump() 将句嵌入执行结果存储至磁盘；最后，通过 faiss.index_factory() 函数在句向量的基础上构建索引，同时做持久化处理并返回文件名。

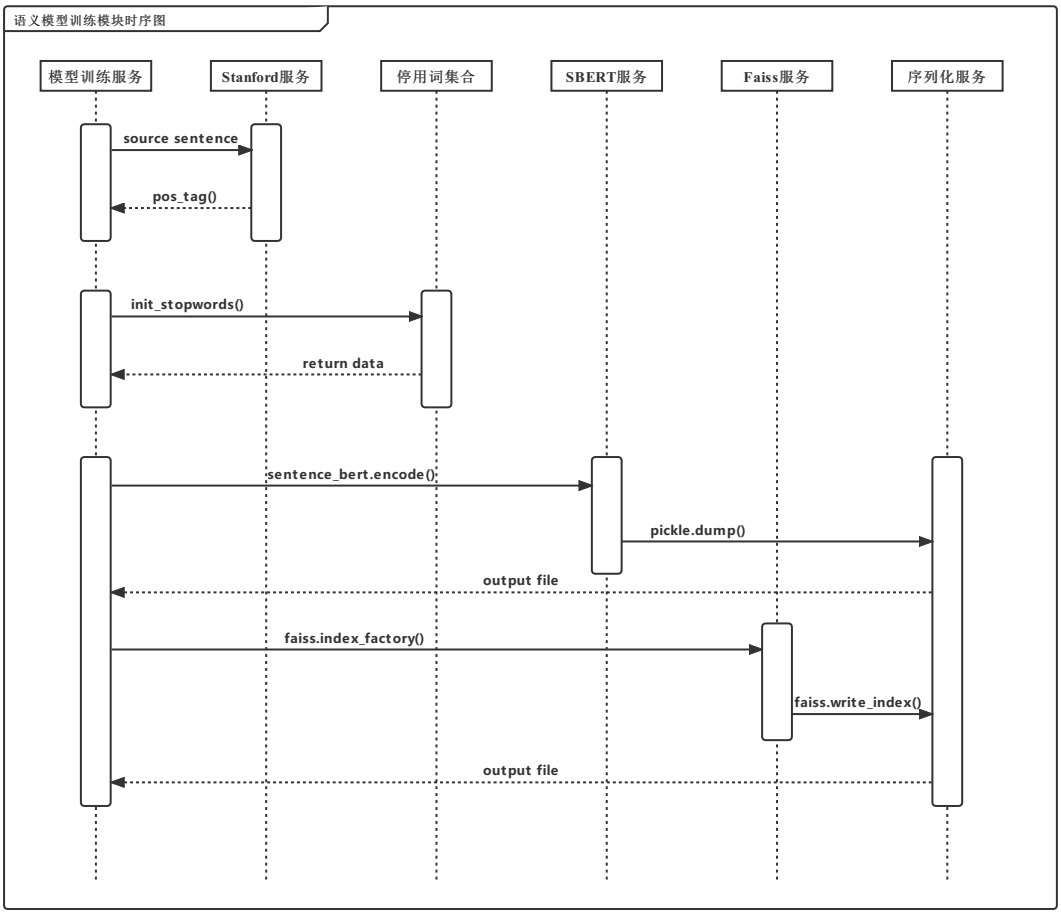



图 4.9: 语义模型训练时序图

句嵌入的实现方案如算法4所示。算法的输入是缺陷摘要文件路径和句嵌入模型导出路径；算法的输出是句向量列表和缺陷唯一标识列表。分块读取源文件，获取缺陷唯一标识添加到 ids 列表；将语义压缩后的缺陷摘要词组添加到 texts 列表；通过 sbert_encode() 函数计算 texts 列表中各文本的句向量；使用 pickle.dump() 函数对句嵌入结果做序列化和持久化处理。句嵌入模型能够极大地降低输入例子在整个语料库中寻找 Top K 文本语义相似度（Semantic Textual Similarity）的计算量。

**Algorithm 4:** 句嵌入算法

```
input : source_file, target_file
output: encoded_data, ids
function train_vector(source_file, target_file):
    texts, ids = [ ], [ ];
    for chunks ← 1 to n do
        for index, chunk in Iterator(chunks) do
            ListAppend(ids, chunk.id);
            ListAppend(texts, sentence_parse(chunk.summary));
        end
    end
    encoded_data ← sbert_encode(texts);
    pickle.dump(encoded_data, target_file);
    return encoded_data, ids;
```

索引模型构建的实现方案如算法5所示。算法的输入是句嵌入索引模型导出路径、句向量列表和缺陷唯一标识列表；算法的输出是句嵌入索引模型文件导出路径。使用 faiss.index_factory() 初始化 Faiss 配置参数；在句向量列表上构建索引，缺陷唯一标识列表作为索引标识符；持久化句嵌入索引模型并返回模型文件的导出路径。

**Algorithm 5:** 索引模型构建算法

```
input : output_file, vector, ids
output: output_file
function train_index(output_file, vector, ids):
    index ← faiss.index_factory(..., );
    index.add_with_ids(vector, numpy.array(ids));
    faiss.write_index(index, output_file);
    return output_file;
```

### 4.2.2 核心代码

去停用词代码如图 4.10所示。去停用词主要负责参照无关词性表和停用词表对输入文本完成去无关词性和去停用词处理。源代码实现的功能为对输入

sentence 做分词操作和词性标注；参照无关词性表，过滤与结果非相关词性的单词；参照停用词表，过滤有关停用词；返回去停用词后的文本词组。

```
def sentence_parse(self, sentence) -> str:
    pos_tag = self.nlp.pos_tag(sentence)
    words = [re.sub(r'^\W|\W$', '', tag[0].strip()).lower()
             for tag in pos_tag if tag[1] not in self.__POSTrashTags]
    return ' '.join([word for word in words if word not in self.__POSStopwords])
```

图 4.10: 去停用词代码

句嵌入代码如图 4.11所示。句嵌入主要负责计算缺陷报告摘要的句向量。源代码实现功能为 load_vector() 函数负责判断句向量模型是否存在，若存在则直接加载，否则进行模型训练。模型训练函数 train_vector() 通过分块读取缺陷报告摘要文件，将缺陷唯一标识和语义压缩后的词组添加到 ids 列表和 texts 列表的尾部，使用 sentence_bert.encode() 函数计算词组的句向量，以二进制文件的形式持久化至磁盘，返回句向量列表 encode_data 和缺陷标识列表 ids。

```
def load_vector(file):
    path = pathlib.Path(file)
    if path.is_file():
        print('load')
        return pickle.load(open(file, 'rb'))
    else:
        print('no such file')
        return np.array([])

def train_vector(self, source_file, target_file) -> Tuple[any, list]:
    texts, ids = [], []
    names = get_csv_header(source_file)
    for chunks in pd.read_csv(source_file, chunksize=chunksize, nrows=nrows, names=names):
        for _, chunk in chunks.iterrows():
            ids.append(chunk['id'])
            texts.append(self.sentence_parse(chunk['summary']))
    encoded_data = self.sentence_bert.encode(texts, convert_to_numpy=True)
    pickle.dump(encoded_data, open(target_file, 'wb'))
    print('dump')
    return encoded_data, ids
```

图 4.11: 句嵌入代码

索引模型构建代码如图 4.12所示。索引模型构建主要负责在句嵌入模型的基础上构建索引。源代码实现功能为 load_index() 函数负责判断句嵌入索引模型是否存在，若存在则直接加载，否则构建索引；鉴于数据集大小及特点，索引训练函数初始化为向量维数 768，构建 IDMap、Flat 类型的索引，度量方法采用 METRIC_L2；使用缺陷标识作为索引标识，在句向量上构建索引；使用 write_index() 函数将训练得到的模型持久化至磁盘。

```
def load_index(self, file) -> None:
    path = pathlib.Path(file)
    if path.is_file():
        self.faiss_index = faiss.read_index(file)
        print('read index')
    else:
        print('no such file')

def train_index(output_file, vector, ids) -> str:
    index = faiss.index_factory(768, 'IDMap, Flat', faiss.METRIC_L2)
    index.add_with_ids(vector, np.array(ids or np.array(range(len(vector)))))
    faiss.write_index(index, output_file)
    print('write index')
    return output_file
```

图 4.12: 索引模型构建代码

### 4.2.3 效果说明

预训练模型需要结合真实场景选择合适的评价体系和标准。本节基于二元混淆矩阵性能指标，设真阳性分类为 True Positive（TP），即被模型识别为正的正样本；假阳性分类为 False Positive（FP），即被模型识别为正的负样本；假阴性分类为 Fasle Negative（FN），即被模型识别为负的正样本；真阴性分类为 True Negative（TN），即被模型识别为负的负样本。

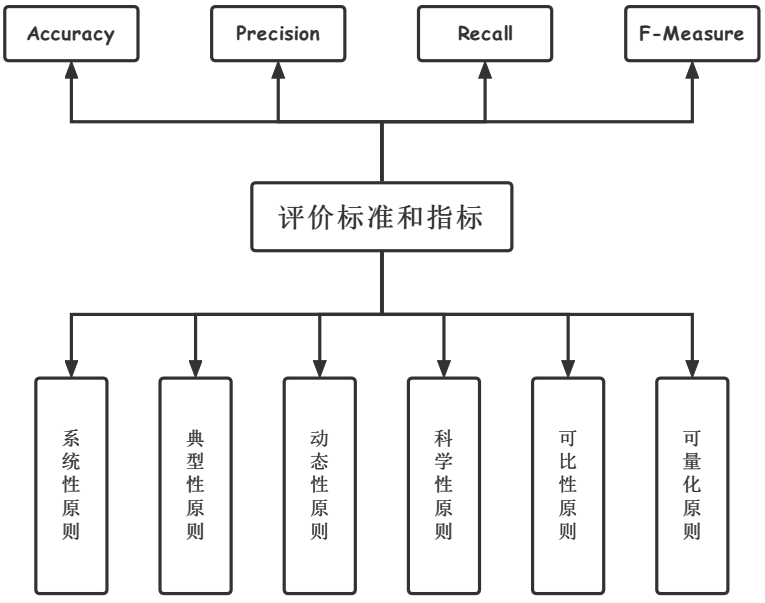



图 4.13: 模型评价标准和指标示意图

标准和指标用于考核、评估和比较模型质量及其效果。模型评价标准和指标如图 4.13所示。评价标准满足系统性、典型性、动态性、科学性、可比性和

可量化原则。综上所述，本研究的评价指标采用准确率（*Accuracy*）、精确率（*Precision*）、召回率（*Recall*）和 F1 值（$F1-Score$）。

准确率是最常见的评价指标，指识别正确的样本数占所有样本数的比例，通常准确率越高，模型效果越好，见公式4-1。

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4-1}$$

精准率是真正为正样本的数量占所有识别为正样本数量的比值，衡量问答模型的查准率，见公式 4-2。

$$Precision = \frac{TP}{TP + FP} \tag{4-2}$$

召回率是被分到正样本中正确识别的样本数量，是真实的正样本数占实际识别为正样本总数的比值，衡量的是问答模型的查全率，见公式 4-3。

$$Recall = \frac{TP}{TP + FN} \tag{4-3}$$

F 值是综合考虑精准率和召回率的指标，是两者的加权调和平均值，F 值越高结果越理想。设参数 $\beta = 1$，此时即为 F1 值，见公式 4-4。

$$F_\beta = \frac{(\beta^2 + 1) \times Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{4-4}$$

句嵌入索引模型选择 20 万条原始缺陷摘要作为预训练数据集；选择 40 万条原始缺陷摘要作为测试集，其中正样本 20 万条，负样本 20 万条。利用文本数据增强技术 EDA [73] 扩增测试集使用场景，对测试集的缺陷摘要分别做随机交换（Randomly Swap）和随机删除（Randomly Delete）处理。

表 4-2: 模型评价指标结果

| 测试数据集 | Accuracy/% | Precision/% | Recall/% | F1 |
|---|---|---|---|---|
| 原始缺陷摘要 | 99.56 | 100 | 99.12 | 0.9956 |
| 随机序列交换 | 97.33 | 100 | 94.66 | 0.9726 |
| 随机删除一词 | 90.11 | 100 | 80.22 | 0.8902 |
| 随机删除两词 | 79.44 | 100 | 58.88 | 0.7412 |

模型的评价指标结果如表4-2所示。模型在原始缺陷摘要、随机序列交换、随机删除一词和随机删除两词的测试集场景 *Accuracy* 分别为 99.56%、97.33%、

90.11%和79.44%，*Precision*均为100%，*Recall*分别为99.12%、94.66%、80.22%和58.88%，*F*1分别为0.9956、0.9726、0.8902和0.7412。

## 4.3 知识图谱构建模块

### 4.3.1 详细设计

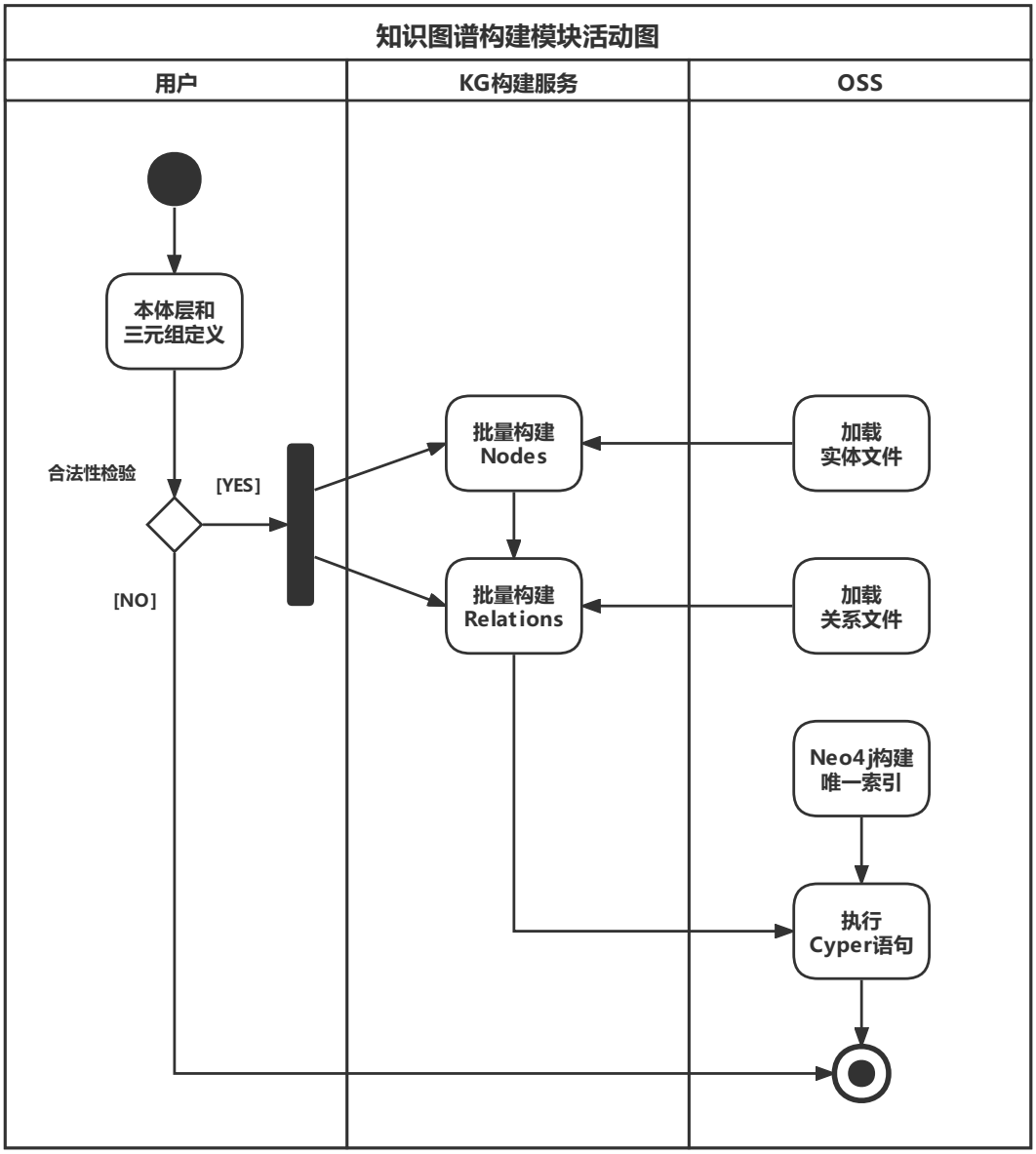



图4.14: 知识图谱构建活动图

信息是描述事物的客观事实，而知识是对事物客观规律的归纳和总结。知识图谱本质上是描述实体及其相互间关系的语义网络，即在信息存储的基础

上，构建实体之间的联系，形成一个有向图结构的知识库。知识图谱构建的主要步骤包括：

1. 本体层定义，本体是特定领域的抽象，是共享概念模型的形式化规范说明，为特殊目的而构建。本体归纳是知识图谱构建的基础。

2. 三元组定义，描述两个节点及其之间关系，是知识图谱的最小单位。通过限定领域知识驱动抽取三元组集合，形成具有语义成分的图数据结构。

3. 知识存储，以“node-edge”的组织形式存储于图数据库。图遍历算法的性能并不会随着数据的增大而受到影响，因此基于原生图计算引擎的 Neo4j 具有非常好的遍历查询性能。

知识图谱构建模块的活动图如图 4.14所示，描述知识图谱构建的交互活动。用户（持续集成人员）检验本体层和三元组定义的合法性；加载实体文件和关系文件，新建节点对象和补全 Cypher 语句；在 Neo4j 图数据库中创建唯一索引，批量导入实体节点，再执行 Cypher 语句构建三元组关系。

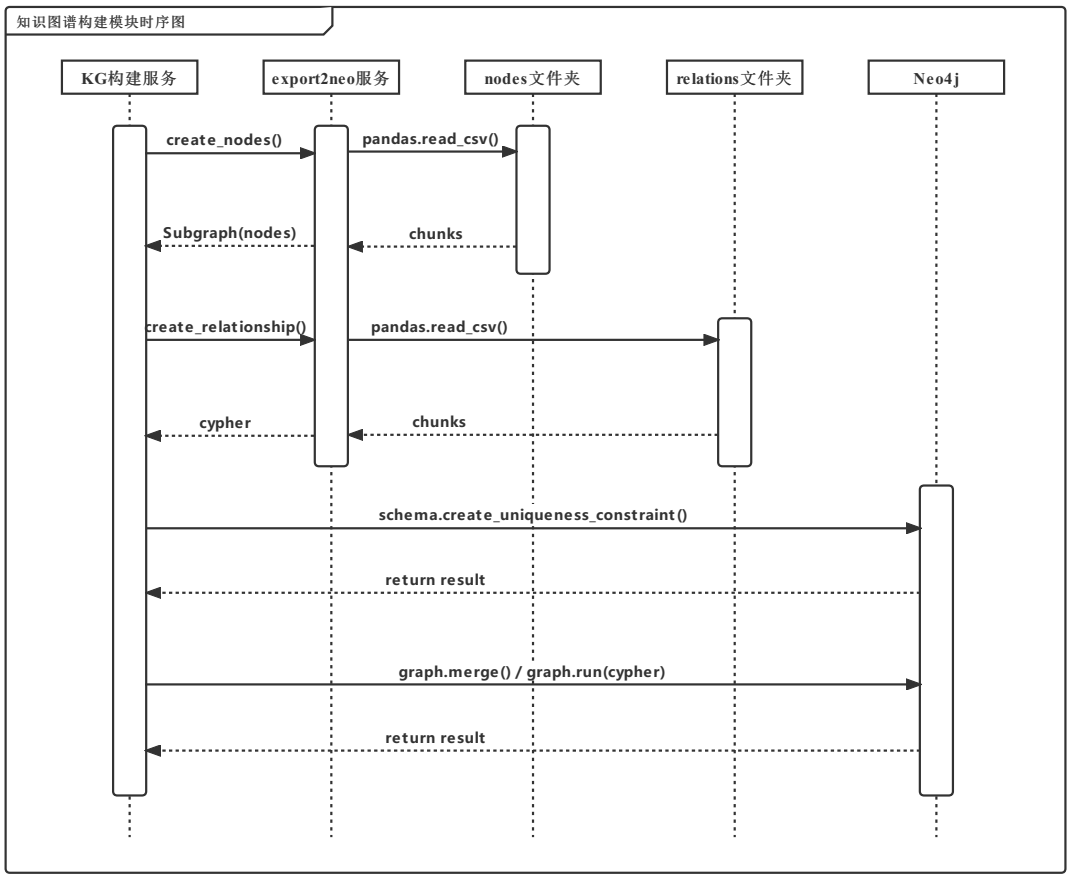



图 4.15: 知识图谱构建时序图

知识图谱构建模块的时序图如图 4.15所示，描述知识图谱构建的行为顺

序。持续集成人员首先向 export2neo 服务发起 create_nodes() 请求，分块读取对应实体文件，export2neo 服务构建子图对象 Subgraph；其次向 export2neo 服务发起 create_relationship() 请求，分块读取对应关系文件，补全模板 Cypher 语法；然后，在 Neo4j 中创建唯一索引，用于加快知识图谱的构建和查询速度；最后，执行 Cypher，在 Neo4j 中批量创建实体和关系。

实体批量导入的实现方案如算法6所示。算法的输入是目标实体文件；算法的输出是执行状态结果。分块读取实体文件，根据字段名称和字段值创建 node 节点并添加至 nodes 列表；若列表非空，则批量导入 Neo4j，同时清空 nodes 列表，进入下一轮循环的数据处理。

**Algorithm 6:** 实体批量导入算法

```
input  : node_data_csv
output: result status
function create_nodes:
    try:
        for chunks ← 1 to n do
            nodes = [ ] ;
            for index, chunk in Iterator(chunks) do
                node = node(label, **chunk) ;
                ListAppend(nodes, node) ;
            end
            if nodes.length then
                graph.merge(Subgraph(nodes), label, *properties) ;
                nodes.clear() ;
            end
        end
        return success ;
    except errors.ClientError:
        return error ;
```

关系批量导入的实现方案如算法7所示。算法的输入是目标关系文件；算法的输出是执行状态结果。获取关系文件的字段名，补全模板 Cypher 语句，通过 graph.run() 函数执行语句，向 Neo4j 数据库批量导入三元组关系。Cypher 是 Neo4j 内置的声明式图查询语言，充分支持高效的图查询或更新。

**Algorithm 7:** 关系批量导入算法

**input :** headNnode, tailNode, relation

**output:** result status

```
function create_relationships:
    try:
        cypher = < template string > % headNode, tailNode, relation ;
        graph.run(cypher) ;
        return success ;
    except errors.ClientError:
        return error ;
```

### 4.3.2 核心代码

创建实体代码如图 4.16所示。创建实体代码主要负责向 Neo4j 图数据库批量导入实体。源代码实现的功能为格式化 label，获取实体属性 property_keys 及实体名 header。分块读取实体文件，创建实体对象 Node 并暂存于 nodes 列表；一个 chunksize 的数据遍历结束后，将 nodes 列表转换为子图对象 Subgraph，调用 py2neo 库的 api 接口批量导入非空子图。

```
def create_nodes(self, filepath, label, property_keys) -> str:
    try:
        label = auto_label(label)
        property_keys = list(map(auto_key, property_keys))
        header = [auto_key(key) for key in get_csv_header(filepath)]
        for chunks in pd.read_csv(filepath, names=header, **self.neo4j_config):
            nodes = []
            for _, chunk in chunks.iterrows():
                node = Node(label, **dict(chunk))
                nodes.append(node)
            if len(nodes) > 0:
                self.graph.merge(Subgraph(nodes), label, *property_keys)
                nodes.clear()
        return 'finish create node' + label
    except errors.ClientError:
        traceback.print_exc()
        return 'create nodes error'
```

图 4.16: 创建实体代码

创建关系代码如图 4.17所示。创建关系代码主要负责向 Neo4j 图数据库批量导入三元组关系。源代码实现的功能为格式化 head 实体的 label 和 tail 实体

的 label；获取非空 relationship 的名称，补全 Cypher 模板；调用 py2neo 库的 graph.run() 函数，直接执行 Cypher 语句，批量构建知识图谱的关系边。

```
def create_relationship(self, filepath, label1, label2, relationship=None) -> str:
    try:
        label1 = auto_label(label1)
        label2 = auto_label(label2)
        if not relationship:
            relationship = (re.search(r'[^/|\\]+\.csv$', filepath, re.I).group() or 'undefined').replace('.csv', '')
        [key1, key2] = get_csv_header(filepath)
        cypher = "LOAD CSV WITH HEADERS FROM \'file:///%s\' AS line " \
                 "MATCH (from:%s), (to:%s) " \
                 "MERGE (from)-[:%s]->(to);" \
                 % (filepath, format_node(label1, key1), format_node(label2, key2), relationship)
        self.graph.run(cypher)
        return 'finish create relationship {}.'.format(relationship)
    except errors.ClientError:
        traceback.print_exc()
        return 'create relationships error'
```

图 4.17: 创建关系代码

### 4.3.3 效果说明

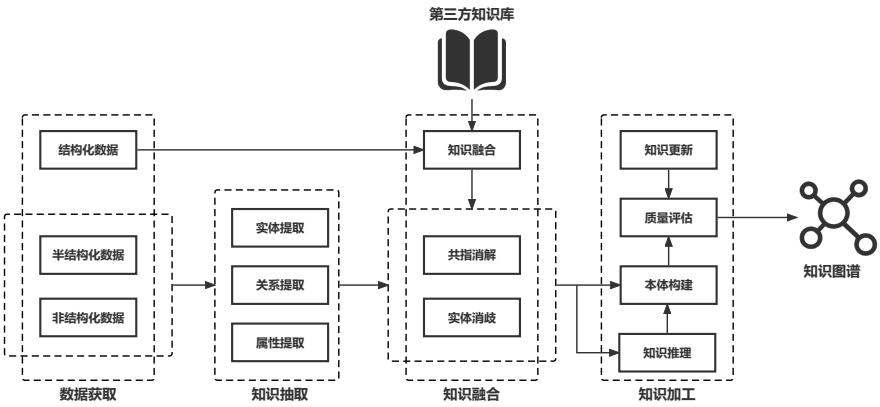



图 4.18: 知识图谱构建链路图

知识图谱构建的核心链路如图 4.18所示。整体上分为四个阶段，即数据获取、知识抽取、知识融合和知识加工。数据获取阶段，采集结构化、半结构化和非结构化等多模态数据；知识抽取阶段，将半结构化和非结构化数据转换为结构化数据；知识融合阶段，对同构数据做共指消解和实体消歧处理；知识加工阶段，通过知识推理定义本体层，依托质量评估构建知识图谱，之后不断进行知识更新，实现模型的终身机器学习。

Neo4j 图数据库存储的知识图谱信息如图 4.19所示。三元组实体（Node Labels）见左图，三元组关系（Relationship Types）见右图。实体总数为 204798 项，实体类型共 11 种；关系总数为 2127717 项，关系类型共 16 种。

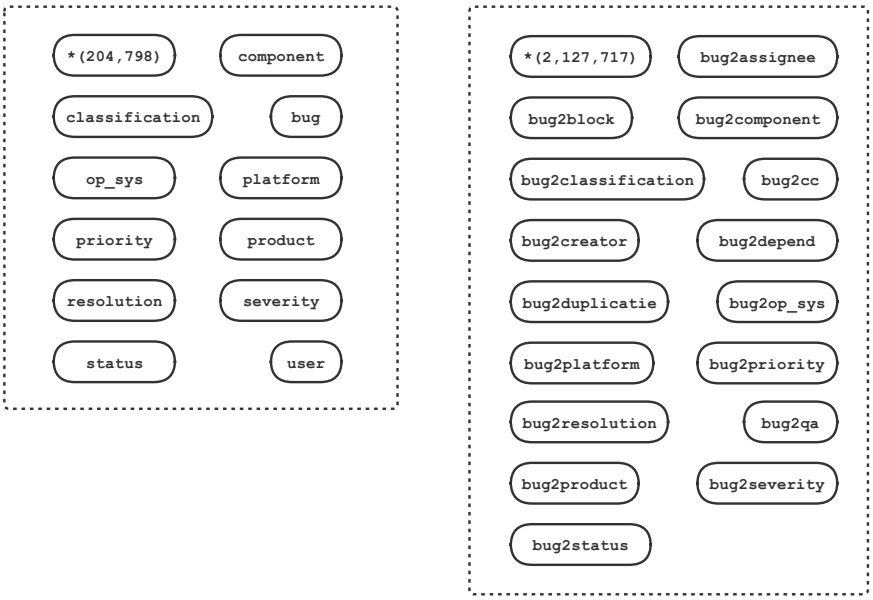



图 4.19: Neo4j 数据库信息示意图

## 4.4 缺陷报告管理模块

### 4.4.1 详细设计

缺陷报告管理主要用于缺陷库的数据扩增，同时增加项目的通用性，使模型不局限于 Bugzilla 缺陷跟踪管理系统或 Eclipse 项目，为不同的缺陷跟踪管理系统或软件项目定制统一数据入口。缺陷报告管理的目标包括：

1. 缺陷报告的填写，在开发、测试或使用过程中发现可复现的缺陷，通过描述缺陷特征和填写相关信息形成缺陷报告。缺陷报告是测试人员和开发人员之间沟通的重要途径。

2. 缺陷报告的修改，管理或开发人员收到 new、open 或 reopen 等状态的缺陷 issue，确认缺陷是否可复现或符合预期需求。若为真实存在的缺陷，则需要及时修复并修改缺陷状态。

3. 实体或关系的查询，可视化图的结构和路径、简化图数据库的维护和管理，实体查询需要给定实体名称及其标识，返回查询实体为核心的子图结构；关系查询需要给定两个实体名称及其标识，返回查询实体之间的完整路径。

缺陷报告管理模块的活动图如图 4.20所示，描述缺陷报告管理的交互活动。用户（缺陷报告人员）填写可复现的缺陷报告，通过合法性校验后，向 MySQL 发送新建指令，新增一条数据库记录。用户（软件开发人员）确认缺陷或修复缺陷后，修改缺陷报告状态，向 MySQL 发送更新指令，修改数据库相

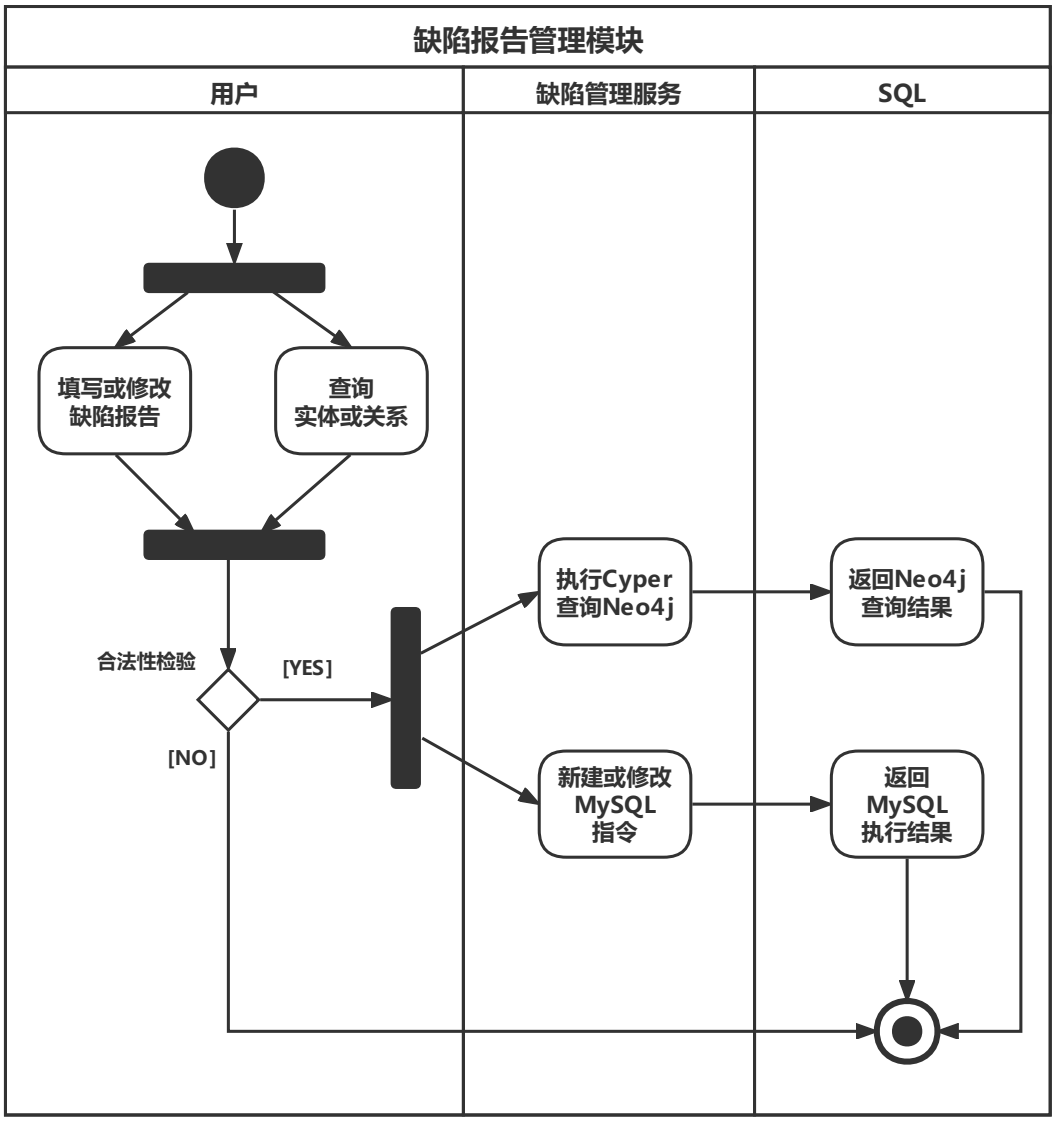



图 4.20: 缺陷报告管理活动图

关记录。用户查询实体或关系，通过合法性校验后，执行相关 Cyper 向 Neo4j 发起查询请求，图数据库返回响应结果。

缺陷报告管理模块的时序图如图 4.21所示，描述缺陷报告管理的行为顺序。缺陷报告人员首先按提示填写缺陷报告并运行 postReport() 函数向后端提交数据；其次，缺陷报告服务作合法性校验，调用 insertReport() 函数往 MySQL 中插入一条记录；然后，软件开发人员确认或修复缺陷，使用 postReport() 函数更新数据；最后，缺陷报告服务调用 updateReport() 函数更新 MySQL 相关记录。对于实体关系查询，用户首先调用 queryNode() 函数或 queryRelation() 函数发起实体或关系查询；然后，实体查询服务或关系查询服务分别向 Neo4j 发起

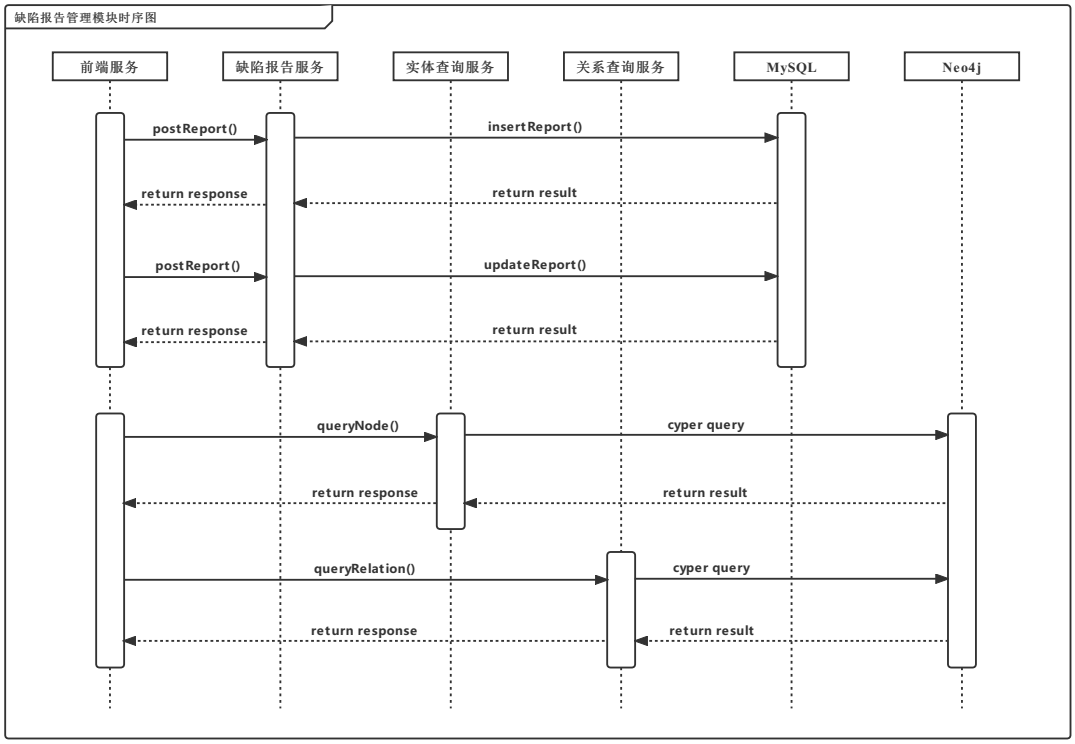



图 4.21: 缺陷报告管理时序图

Cyper 查询请求；最后，Neo4j 返回经格式化处理后的结果。

异步网络通信的实现方案如算法8所示。算法的输入是请求 url 网址、请求数据 data 和请求类型 type；算法的输出是请求响应 response。根据参数传递类型调用 axios 发起 GET 或 POST 请求，以及复杂网络请求如 PUT、DELETE 等，还需要确保同一时间客户端有且仅有一个 axiosInstance 对象被创建。

**Algorithm 8:** 异步网络通信算法

```
input  : url, data, type
output: reponse message
function ajax:
    switch type:
        case GET:
            return axiosInstance.get(url, data) ;
        case POST:
            return axiosInstance.get(url, data) ;
        default:
            return not support ;
```

### 4.4.2 核心代码

```
const ajax = (url: string, data={}, type="GET"): any => {
  switch(type) {
    case "GET":
      return axiosInstance.get(url, { params: data });
    case "POST":
      return axiosInstance.post(url, JSON.stringify(data));
    default:
      return '暂不支持当前网络请求方式';
  }
}
```

图 4.22: 异步网络通信代码

异步网络通信代码如图 4.22所示。异步网络通信代码负责配置统一的网络请求。源代码实现的功能为根据请求类型分发调用 axiosInstance 实例的 get() 请求或 post() 请求，分别向后端接口传递 params 对象和 stringify() 函数处理后的 JSON 对象。

```
const axiosInstance: AxiosInstance = axios.create({
  baseURL: process.env.REACT_APP_BASE_URL,
});
axiosInstance.interceptors.request.use(
  (request: AxiosRequestConfig) => {
    return request;
  },
  (error: any) => {
    return Promise.reject(error);
  }
);
axiosInstance.interceptors.response.use(
  (response: AxiosResponse) => {
    if (response.status !== 200 && response.status !== 304) {
      showMessage(response.status);
      return response;
    }
    return response;
  },
  (error: any) => {
    const { response } = error;
    if (response) {
      message.error(showMessage(response.status));
      return Promise.reject(response.data);
    } else {
      message.error('网络连接异常,请稍后再试!');
    }
  }
);
```

图 4.23: axios 单例模式代码

axios 单例模式代码如图 4.23所示。axios 单例模式代码确保客户端运行时只存在一个 axios 对象，并配置拦截请求和错误边界的处理逻辑。源代码实现的功能为创建 axios 实例，使用 interceptor 拦截请求发送和请求响应，并做统一

的容错处理。拦截响应状态码不是 200 或 304 的网络请求，通过 showMessage() 函数作异常和错误的细节处理。

### 4.4.3 效果说明


图 4.24: 缺陷报告填写界面

缺陷报告管理-缺陷报告填写界面如图 4.24所示。页面排版基于栅格的响应式布局，采用表单控件填写报告信息，满足人机交互需求，充分考虑异常和错误的边界处理。用户在该页面填写缺陷报告相关信息并通过合法性校验后，调用 axios 发起异步网络请求，最后反馈服务端响应网络请求的结果。

## 4.5 问答系统开发模块

### 4.5.1 详细设计

问答技术满足用户期望快速且直接地获取信息的需求，以准确、简洁的自然语言或结构化反馈对提问作出应答。本系统基于 Electron 跨平台桌面应用技术，采用即问即答的问答形式，即输入问题直接返回答案，没有多余的交互约束。问答系统开发的任务包括：

1. 缺陷描述补全，用于缺陷查重场景。用户描述缺陷特征时，前端采用节流轮询的方式向后台发起补全请求，后端按照最左前缀匹配原则查询数据库并返回补全提示信息。

2. 句式分发规则，用于模式匹配场景。总结自然语言常见句型，预定义若干模板，通过匹配规则分发至对应功能函数，进而得到形式化查询。

3. 页面交互设计，定义两个或若干个组件或子模块之间的通信方式。从“可用性”和“用户体验”两个层面进行设计，注重以人为本，以用户为中心设计交互需求。

4. 用户体验优化，指用户使用问答服务的主观体验感受。保证人机界面同核心功能之间的一致性，并不断更新、迭代和优化。

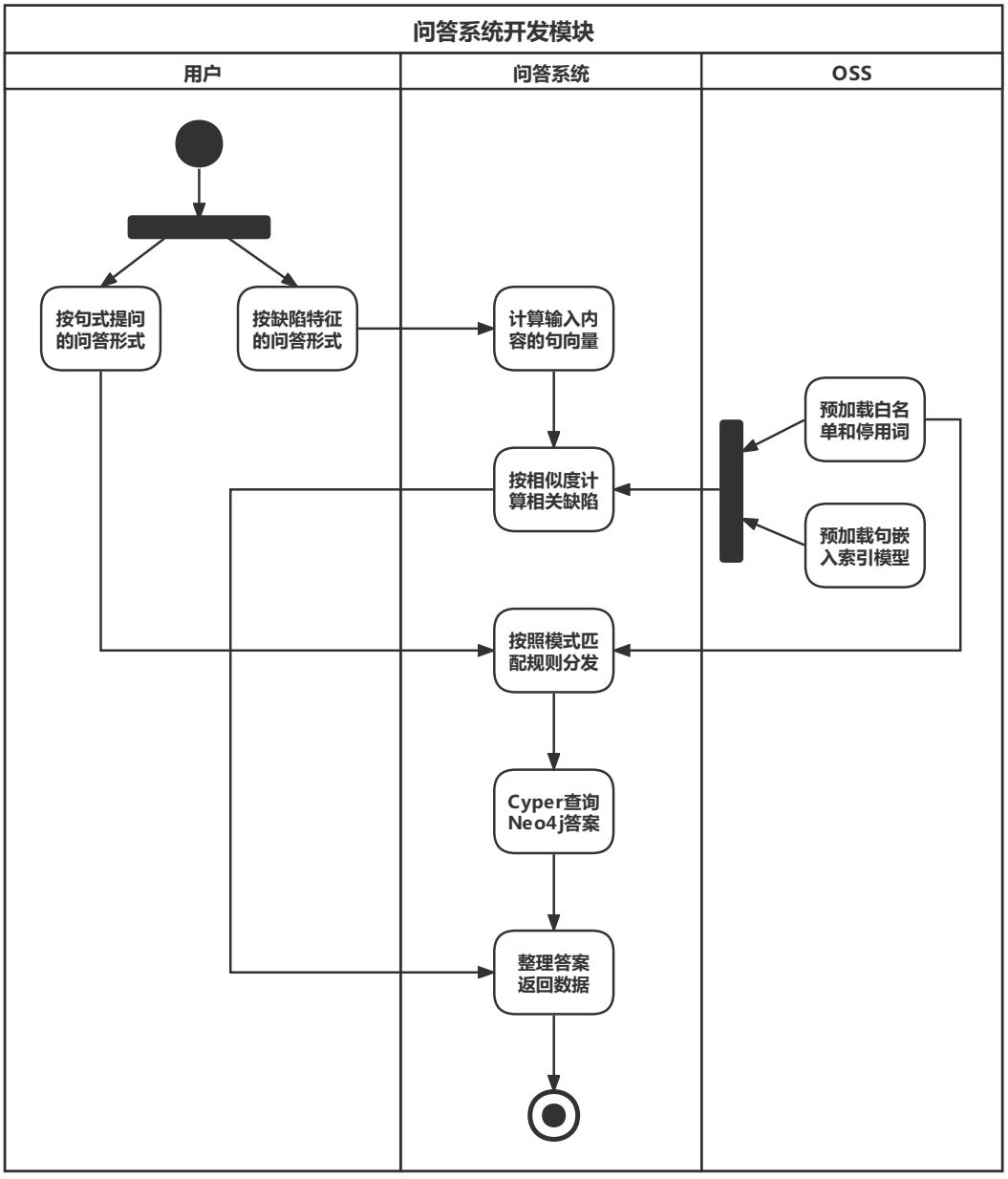



图 4.25: 问答系统开发活动图

问答系统开发模块的活动图如图 4.25所示，描述问答系统开发的交互活动。问答用户按缺陷特征提问；问答系统预加载停用词，同时预加载句嵌入索引模型；对输入问题进行语义压缩、计算句向量并转化为逻辑形式的查询；检索句嵌入索引模型以获取阈值内语义最相近的缺陷标识；根据预设规则格式化结果并返回。问答用户按常见自然语言句式提问；问答系统预加载白名单，同时按照模式匹配规则分发至功能函数；功能函数识别意图后补全 Cypher 模板，向 Neo4j 发起形式化查询请求；格式化查询结果并返回答案。

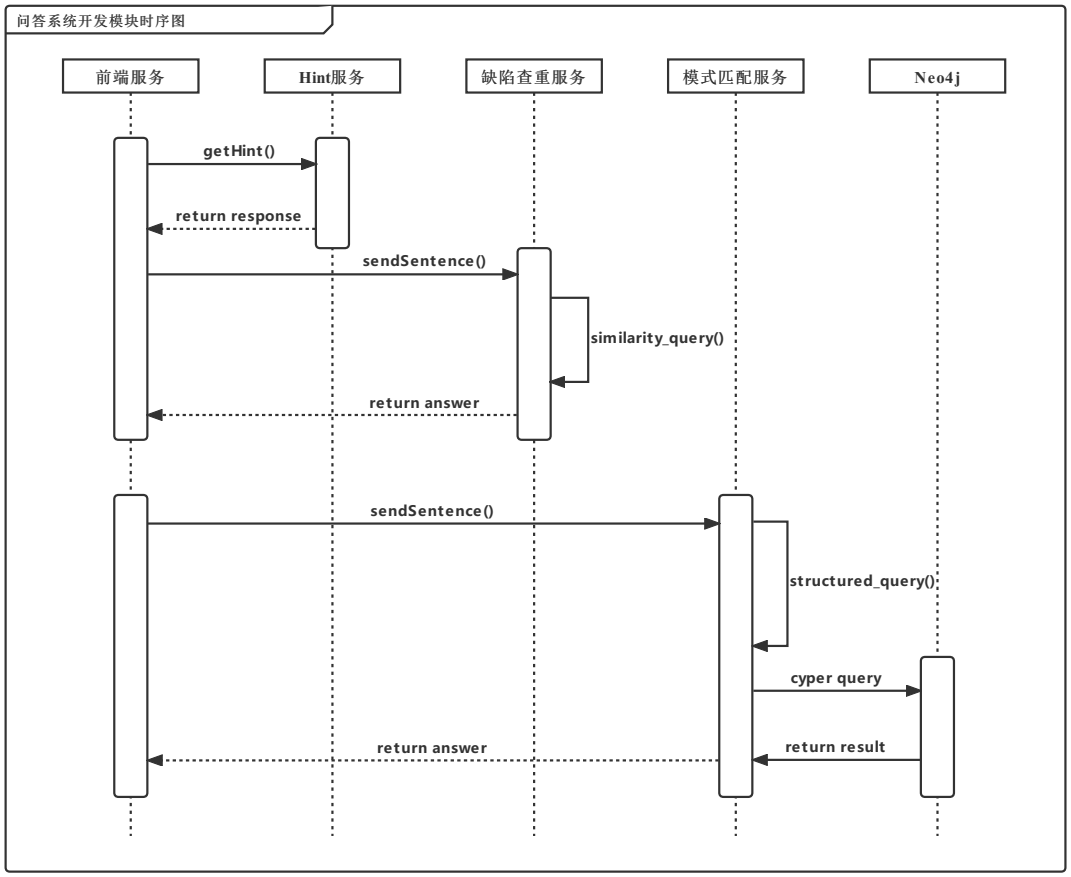



图 4.26: 问答系统开发时序图

问答系统开发模块的时序图如图 4.26所示，描述问答系统开发的行为顺序。问答用户首先按缺陷特征提问，前端采用节流的方式轮询调用 getHint() 函数向 Hint 服务发起数据请求；再次，Hint 服务根据最左匹配原则查询数据库，返回缺陷摘要补全提示；然后，前端服务使用 sentSentence() 函数将缺陷特征描述文本发送至缺陷查重服务；最后，利用 similarity_query() 函数查询句嵌入索引模型中语义最相近的缺陷摘要，返回匹配结果。对于常见句式问答，首先使用 sentSentence() 函数发送至模式匹配服务；其次，调用 structured_query() 函数

按照预定义模板执行规则分发；然后，补全 Cypher 语句并转化为形式化查询语言，向 Neo4j 发起数据请求；最后，格式化处理结果，返回最终答案。

**Algorithm 9:** 模式匹配规则算法

**input :** sentence
**output:** result
**function** rule_distribution**:**
  **if** re.search(^*wh*(*ose*|*at*|*ich*|*ere*|*en*|*y*|*o*)|^*how.m*, sentence) **then**
    result = select_question(sentence) ;
  **else if** re.search(*is*|*are*|*was*|*were*, sentence) **then**
    result = declarative_sentence(sentence) ;
  **else if** re.search(, . ∗ (*is*|*are*)*n*.?*t*, sentence) **then**
    result = multi_skip_question(sentence) ;
  **else if** re.search(#*d* + . ∗ *bor**b*, sentence) **then**
    result = special_question(sentence) ;
  **else if** re.search(^#*d* + (?!. ∗ (*is*|*are*|*was*|*were*)), sentence) **then**
    result = anti_question(sentence) ;
  **else if** re.search(^*how*.?*m*. + #*d* + \′*s*.+, sentence) **then**
    result = general_question(sentence) ;
  **return** result;

模式匹配规则的分发方案如算法9所示。算法的输入是常见自然语言句式；算法的输出是应答结果。针对特殊疑问句、一般疑问句、反意疑问句、选择疑问句、陈述句和多跳疑问句六种句型预设模板问题；根据匹配规则分发至不同的功能函数；功能函数识别语义并补全模板字符串，向 Neo4j 发起数据查询，整理结果并返回答案。

### 4.5.2 核心代码

模式匹配分发规则代码如图 4.27所示。模式匹配分发规则代码负责将输入问题按照预设模板匹配规则分发至各功能函数。源代码实现的功能为判断常见句型：若为特殊疑问句则转至 special_question() 函数；若为一般疑问句则转至 general_question() 函数；若为反意疑问句则转至 anti_question() 函数；若为选择疑问句则转至 select_question() 函数；若为陈述句则转至 declarative_sentence() 函数；若为多跳疑问句则转至 multi_skip_question() 函数。

```
def select_question(self, sentence, bug_id, properties, relations) -> list or None: ...

def declarative_sentence(self, bug_id, properties=None, relations=None) -> dict: ...

def multi_skip_question(self, sentence, bug_id, relations) -> dict: ...

def special_question(self, sentence, bug_id, properties=None, relations=None) -> dict: ...

def anti_question(self, sentence, bug_id, properties, relations) -> str: ...

def general_question(self, sentence, bug_id, properties, relations) -> str: ...

def standard_question_bank(self, sentence='') -> str:
    sentence = sentence.strip().lower()
    bug_id, properties, relations = self.handler_factory(sentence)
    result = 'nothing match'
    if re.search(r'#\d+.*\bor\b', sentence):
        result = self.select_question(sentence, bug_id, properties, relations)
    elif re.search(r'^#\d+(?!.*(is|are|was|were))', sentence):
        result = self.declarative_sentence(bug_id, properties, relations)
    elif re.search(r'^how.?m.+#\d+\'s.+', sentence):
        result = self.multi_skip_question(sentence, bug_id, relations)
    elif re.search(r'^wh(ose|at|ich|ere|en|y|o)|^how.m', sentence):
        result = self.special_question(sentence, bug_id, properties, relations)
    elif re.search(r',.*(is|are)n.?t', sentence):
        result = self.anti_question(sentence, bug_id, properties, relations)
    elif re.search(r'is|are|was|were', sentence):
        result = self.general_question(sentence, bug_id, properties, relations)
    return result
```

图 4.27: 模式匹配分发规则代码

### 4.5.3 效果说明

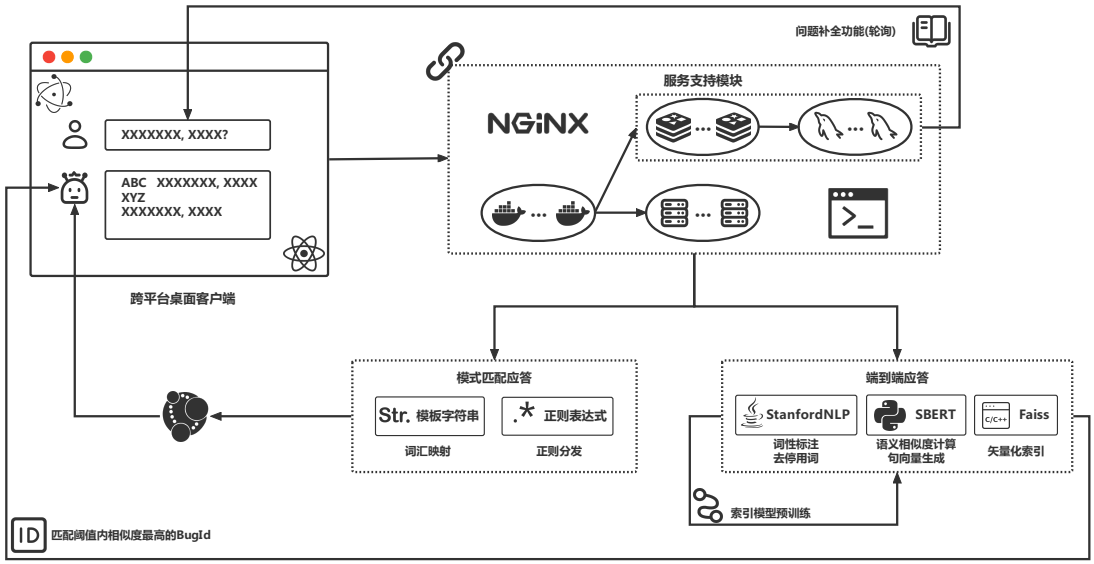



图 4.28: 核心链路图

问答系统核心链路如图 4.28所示。问答用户在跨平台桌面客户端以自然语言描述缺陷特征或按常见句式提问；服务提供模块按照均衡负载规则将网络请

求反向代理至后端服务器；根据提问方式分别调用模板匹配应答服务或端到端应答服务；格式化结果数据，返回最终答案。


图 4.29: 缺陷报告查重界面

缺陷报告查重（缺陷唯一标识查询）界面如图 4.29所示。若缺陷库中能够匹配误差阈值范围内的已有缺陷报告，则返回缺陷唯一标识符、缺陷摘要和缺陷描述；否则，作异常或错误处理。

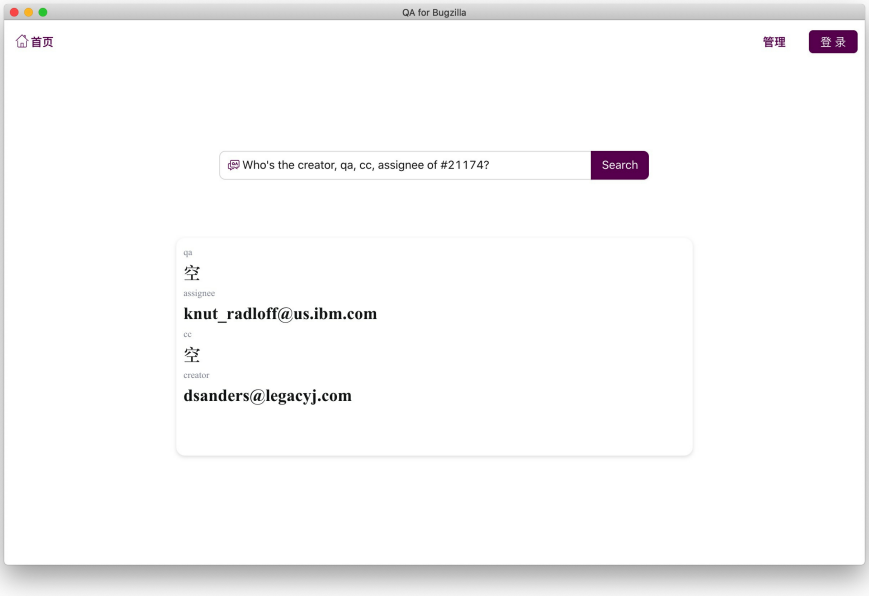



图 4.30: 自然语言问答界面

句式规则匹配问答界面如图 4.30所示。若输入文本为特殊疑问句、一般疑问句、反意疑问句、选择疑问句、陈述句和多跳疑问句六种模板句型之一，同时成功解析语义、识别意图，且知识图谱存在相关路径，则格式化结果并返回答案；否则，作空态处理并显示无法识别或找不到结果的提示信息。

## 4.6 本章小结

本章介绍五个主要功能模块的详细设计和具体实现，包括原始数据清洗模块、语义模型训练模块、知识图谱构建模块、缺陷报告管理模块和问答系统开发模块。通过活动图、时序图、伪代码等多种方式对各个模块的设计思路和实现逻辑进行阐述，给出每个模块的目标或步骤以及核心功能的源代码，以评价指标或界面截图的方式说明和展示最终的实现效果。

# 第五章　系统测试与结果分析

## 5.1　测试准备

### 5.1.1　测试目标

软件测试是项目开发的最后一道防线，是产品正式上线前最重要的质量保障环节。测试的目标是确保软件完成既定业务的功能、满足系统性能的要求、提供质量评估的依据、改进和管理产品的质量。为保证系统能够提供稳定可用的服务，本项目采用测试驱动开发 [74]（Test-Driven Development，TDD），确保现有功能准确无误，预留可扩展空间，快速响应环境变化。检验系统核心模块的需求是否得到满足，需要执行以下测试流程：

1. 单元测试，软件项目的最小可测试单元，通常把函数作为系统的最小功能单位，因此单元测试主要面向各功能函数，关注函数内部逻辑。研究表明超过 85% 的缺陷都在代码早期阶段产生，发现缺陷的阶段越靠后，修复成本就越高。单元测试能较早地检测出缺陷，从而降低缺陷影响。测试工具和方法包括 Code Review、静态代码扫描、Mock 等。

2. 集成测试，建立在单元测试的基础上，用于检测服务之间以及内部子服务之间的交互点。测试的重点是检查网络数据的交换、传递和服务间的依赖关系等。主要步骤是：获取接口的 URL 地址；查看接口的发送方式；配置请求头和请求体；发起网络请求并核验响应结果是否符合预期。测试工具和方法包括 Postman、编程语言的 Request 库等。

3. 系统测试，完成功能模块后按照需求设计集成软硬件环境模拟真实业务场景，用于验证软件系统的功能和性能等是否满足规约指定的要求。主要步骤是：根据测试场景，设定功能测试计划；编写功能测试用例；上报测试发现的可复现缺陷；跟踪并重新测试缺陷；重复上述步骤直到功能测试完成。测试工具和方法包括黑盒测试、回归测试等。

单元测试保证系统原子功能的稳定性，集成测试保证程序部件的可用性，系统测试保证功能或模块的完整性。单元测试阶段、集成测试阶段和系统测试阶段均需要编写测试用例文档（执行单）梳理各个测试环节。用例文档是测试任务的描述，覆盖完整的测试流程，形成测试闭环，体现测试方案和策略，包括测试用例名称、测试用例标识、测试目标、测试说明、前提与约束、终止条件、测试步骤和预期结果等组成元素。

### 5.1.2 测试环境

测试环境提供运行测试用例所需的设置，准确反馈被测应用在运行时的行为和质量。根据系统物理视图定义，系统测试环境如表 5-1所示，涉及服务器软硬件资源信息，具体包括由 4 台阿里云 ECS 组成的服务器集群。测试硬件环境资源信息为：双核 CPU；操作系统为 CentOS；4GB 内存容量；1Mbps/3Mbps 宽带各两台；40GB 硬盘空间。测试软件环境资源信息为：一台服务器负责前端静态资源，部署 Nginx 服务并开启静态资源压缩和缓存功能；两台服务器负责搭建集群系统，根据端口号（开启 8080 端口和 8082 端口）和 ip 地址划分，逻辑上组成 4 台服务器的集群；一台服务器负责提供数据服务，部署 MySQL 关系型数据库和 Neo4j 图数据库，以及运行 Redis 缓存。

表 5-1: 系统测试环境

| 服务器名称 | 硬件环境 | 软件环境 |
| --- | --- | --- |
| 阿里云 ECS_A | 双核 4G 内存<br>CentOS64 位<br>宽带 3Mbps<br>硬盘 40G | Nginx<br>Docker |
| 阿里云 ECS_B | 双核 4G 内存<br>CentOS64 位<br>宽带 1Mbps<br>硬盘 40G | Sever8080<br>Sever8082<br>Docker |
| 阿里云 ECS_C | 双核 4G 内存<br>CentOS64 位<br>宽带 1Mbps<br>硬盘 40G | Sever8080<br>Sever8082<br>Docker |
| 阿里云 ECS_D | 双核 4G 内存<br>CentOS64 位<br>宽带 3Mbps<br>硬盘 40G | Redis Cluster<br>MySQL Cluster<br>Neo4j<br>Docker |

软件版本信息如表 5-2所示，表中展示系统运行所需的软件环境依赖和版本。操作系统均采用 CentOS64 位 7.7；桌面端框架为 Electron 15.3.0；前端框架选择 React 17.0.2；前端开发语言选择 Typescript 4.1.2；后端开发语言选择 Python 3.7.12；关系型数据库使用 MySQL 8.0.16；图数据库使用 Neo4j 4.2.1；利用 Nginx 1.20.2 实现均衡负载和反向代理；接口测试选用 Postman 8.12.0；压力测试选用 ApacheBench 2.3；虚拟化工具选用 Docker 20.10.12；通过 Secure-CRT 远程控制和管理云服务器终端。

表 5-2: 软件版本信息

| 环境项 | 版本 |
| --- | --- |
| OS | CentOS64 位 7.7 |
| Electron | Electron 15.3.0 |
| React | React 17.0.2 |
| Typescript | Typescript 4.1.2 |
| Python | Python 3.7.12 |
| Redis | Redis 4.0.14 |
| MySQL | MySQL 8.0.16 |
| Neo4j | Neo4j 4.2.1 |
| Nginx | Nginx 1.20.2 |
| Docker | Docker 20.10.12 |
| Postman | Postman 8.12.0 |
| ApacheBench | ApacheBench 2.3 |
| SecureCRT | SecureCRT 8.5 |

## 5.2 单元测试

单元测试是一种面向最小可测试单元的软件测试，目的是验证软件代码的每个单元能否按预期执行。单元测试是软件生命周期不可或缺的部分，所处位置如图 5.1所示，不恰当的单元测试方案会导致在集成测试、系统测试甚至构建应用程序后的 Beta 测试期间修复缺陷的成本大大增加。本系统前端选用 JavaScript 测试框架 jest，后端选用 Python 内置单元测试框架 unittest。

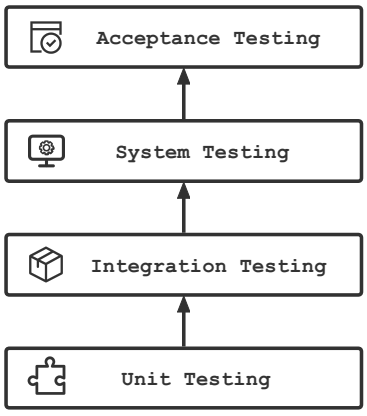



图 5.1: 软件测试流程图

按照系统的架构划分进行单元测试，分别验证展示层、转发层、数据层和存储层的功能可用性。根据各层的主要职责，测试的侧重点也并不相同。展示

层主要验证前端交互、网络请求和错误边界处理；转发层主要验证转发控制和请求参数；数据层主要验证数据的可靠性和加载的及时性；存储层主要验证数据库操作的原子性。

表 5-3: 单元测试用例执行单

| ID | 测试项 | 类型 | 说明 | 用例数 | 结果 |
|---|---|---|---|---|---|
| UT1 | HomePageTest | 展示层 | 问答主页 | 5 | 通过 |
| UT2 | ManagePageTest | 展示层 | 管理主页 | 11 | 通过 |
| UT3 | HintControllerTest | 转发层 | 问题补全相关 | 1 | 通过 |
| UT4 | KGControllerTest | 转发层 | 知识图谱相关 | 2 | 通过 |
| UT5 | QAControllerTest | 转发层 | 问答请求分发相关 | 3 | 通过 |
| UT6 | ReportControllerTest | 转发层 | 缺陷报告数据管理 | 5 | 通过 |
| UT7 | NodeTest | 数据层 | 本体层管理 | 1 | 通过 |
| UT8 | TripleTest | 数据层 | 三元组管理 | 1 | 通过 |
| UT9 | DictInitTest | 数据层 | 初始化数据加载 | 6 | 通过 |
| UT10 | XModelTest | 数据层 | 模型加载和配置相关 | 27 | 通过 |
| UT11 | Export2neoTest | 存储层 | Neo4j 封装存储操作 | 14 | 通过 |
| UT12 | PyMySQLTest | 存储层 | mysql 封装存储操作 | 17 | 通过 |
| 总计 | | | | 93 | 通过 |

参照单元测试方案，本系统的单元测试用例执行单如表 5-3所示。测试项共 12 个，包括展示层 2 项，转发层 4 项，数据层 4 项，存储层 2 项。用例数共 93 个，其中测试通过 93 个，测试覆盖率符合 Pareto 法则。后续的迭代需要继续完善和扩展单元测试用例，运用持续集成和持续测试的方式，在构建发布前做充分的单元测试，切实保障测试覆盖率，进而保证功能单元的可用性。

## 5.3 接口测试

接口测试属于集成测试阶段，用于测试接口的一致性、可用性和可靠性，验证子系统或系统之间的通信能否正确传达。通过模拟客户端向服务器发送请求报文，服务器接收请求报文后处理业务并向客户端反馈结果，客户端接收并分析应答结果，执行剩余逻辑或处理错误边界。接口测试重点关注服务器逻辑验证和检查数据的交换、传递。

本系统是前后端分离的架构模式，因此主要测试 HTTP 网络接口。接口组成包括请求（request）、响应（response）、服务器（host）、路径（path）、端口（port）、参数（query）、状态码（code）和请求类型（method）等网络

请求内容。接口测试能够缩短系统上线时间、提高代码的鲁棒性。

表 5-4: 接口测试用例执行单

| ID | 说明 | 输入项 | 预期输出 | 结果 |
|---|---|---|---|---|
| IT1 | 新增一份缺陷报告 | 表单数据的网络请求 | 新增成功提示 | 通过 |
| IT2 | 修改缺陷报告状态 | 状态修改的网络请求 | 修改成功提示 | 通过 |
| IT3 | 缺陷摘要补全 | 缺陷特征描述 | 最左匹配查询结果 | 通过 |
| IT4 | 下拉列表数据获取 | 待获取数据的名称 | 查询数据库结果 | 通过 |
| IT5 | 用户数据获取 | 用户名或用户邮箱 | 最左匹配查询结果 | 通过 |
| IT6 | 实体查询链路闭合 | 实体标签及其值 | 返回实体关联信息 | 通过 |
| IT7 | 关系查询链路闭合 | 头尾实体标签及其值 | 返回实体之间路径 | 通过 |
| IT8 | 特征描述应答 | 描述缺陷的主要特征 | 返回缺陷主要信息 | 通过 |
| IT9 | 常见句式应答 | 常见自然语言问句 | 返回问题答案 | 通过 |

本系统是选用 Postman 作为批量接口测试工具，接口测试用例见表 5-4所示，主要包含系统业务功能向外暴露的网络接口、StanfordNLP 和 Faiss 等第三方库提供的 api 接口。

表中所示信息，测试用例数为 9 个，轮询测试 10 轮，共计 90 次网络请求，测试覆盖率符合 Pareto 法则。通过核验这些请求返回的状态码、响应头部和响应体，测试结果全部通过。

## 5.4 功能测试

表 5-5: 缺陷报告填写测试用例执行单

| 测试用例标识 | TC1 | | |
|---|---|---|---|
| 测试目标 | 前端有错误约束；网络请求连通；数据库写入正常 | | |
| 测试说明 | 验证新增缺陷报告服务可用 | | |
| 前提与约束 | 进入缺陷报告填写界面 | | |
| 终止条件 | 功能链路闭环；任意步骤报错 | | |
| 测试过程 | | | |
| 序号 | 输入及操作说明 | 期望测试结果 | 结果 |
| 1 | 点击“缺陷报告填写”菜单 | 进入“缺陷报告填写”界面 | 通过 |
| 2 | 按要求填写缺陷报告单 | 校验规则认证 | 通过 |
| 3 | 点击下拉框 | 弹出选项列表 | 通过 |
| 4 | 点击“提交”按钮 | 按钮进入“loading”状态 | 通过 |
| 5 | 等待“loading”结束 | 全局提示“提交”结果 | 通过 |

功能测试属于系统测试阶段，又称为黑盒测试。在完全忽略系统内部结构和内部特性的前提下，根据需求规格说明书测试系统程序的软件界面和交互行为，核验单个模块或功能是否正常运行以满足设计需求。

缺陷报告填写功能的测试用例执行单如表5-5所示。测试重点为人机交互、规则校验和数据正确性。额外流程还包括手动暂存报告和自动保存草稿。

表 5-6: 缺陷报告审批测试用例执行单

| **测试用例标识** | TC2 | | |
|---|---|---|---|
| **测试目标** | 网络请求连通；数据库写入成功 | | |
| **测试说明** | 验证修改缺陷状态服务可用 | | |
| **前提与约束** | 进入缺陷报告审批界面 | | |
| **终止条件** | 功能链路闭环；任意步骤报错 | | |
| **测试过程** | | | |
| **序号** | **输入及操作说明** | **期望测试结果** | **结果** |
| 1 | 点击“缺陷报告审批”菜单 | 进入“缺陷报告审批”界面 | 通过 |
| 2 | 点击某一下标页 | 列表更新为该页数据 | 通过 |
| 3 | 点击某行的“审批”按钮 | 弹出审批下拉框 | 通过 |
| 4 | 选择某个审批状态 | 全局提示“审批”处理结果 | 通过 |
| 5 | 点击某行的“删除”按钮 | 全局提示“删除”处理结果 | 通过 |

缺陷报告审批功能的测试用例执行单如表 5-6所示。测试重点为人机交互和数据正确性。额外流程还包括列表排序和列表搜索。

表 5-7: 实体关系查询测试用例执行单

| **测试用例标识** | TC3 | | |
|---|---|---|---|
| **测试目标** | 网络请求连通；数据库查询正常 | | |
| **测试说明** | 验证图数据库查询服务可用 | | |
| **前提与约束** | 进入实体/关系查询界面 | | |
| **终止条件** | 功能链路闭环；任意步骤报错 | | |
| **测试过程** | | | |
| **序号** | **输入及操作说明** | **期望测试结果** | **结果** |
| 1 | 点击“实体查询”菜单 | 进入“实体查询”界面 | 通过 |
| 2 | 输入缺陷库中存在的实体 | 提示查询成功返回实体信息 | 通过 |
| 3 | 输入缺陷库中不存在的实体 | 提示缺陷库中不存在该实体 | 通过 |
| 4 | 点击“关系查询”菜单 | 进入“关系查询”界面 | 通过 |
| 5 | 输入缺陷库中存在的关系 | 提示查询成功并可视化结果 | 通过 |
| 6 | 输入缺陷库中不存在的关系 | 提示缺陷库中不存在该关系 | 通过 |

实体关系查询功能的测试用例执行单如表 5-7所示。测试重点为人机交互和数据正确性。额外流程还包括实体或三元组结构的可视化显示和交互。

表 5-8: 缺陷报告查重测试用例执行单

| **测试用例标识** | TC4 | | |
|---|---|---|---|
| **测试目标** | 网络请求连通；模型正常运行 | | |
| **测试说明** | 验证缺陷查重服务可用 | | |
| **前提与约束** | 进入问答主页；模型完成预训练 | | |
| **终止条件** | 功能链路闭环；任意步骤报错 | | |
| **测试过程** | | | |
| **序号** | **输入及操作说明** | **期望测试结果** | **结果** |
| 1 | 点击“首页”按钮 | 进入问答系统主页 | 通过 |
| 2 | 描述存在的缺陷特征 | 下拉框补全提示 | 通过 |
| 3 | 点击“查询”按钮 | 按钮进入“loading”状态 | 通过 |
| 4 | 等待“loading”结束 | 显示唯一标识、摘要和描述 | 通过 |
| 5 | 描述不存在的缺陷特征 | 提示缺陷库不存在该缺陷 | 通过 |

缺陷报告查重功能的测试用例执行单如表 5-8所示。测试重点为数据正确性。额外流程还包括输入框清空和结果复制。

表 5-9: 句式规则匹配测试用例执行单

| **测试用例标识** | TC5 | | |
|---|---|---|---|
| **测试目标** | 网络请求连通；句式匹配规则逻辑合理 | | |
| **测试说明** | 验证自然语言问答服务可用 | | |
| **前提与约束** | 进入问答主页 | | |
| **终止条件** | 功能链路闭环；任意步骤报错 | | |
| **测试过程** | | | |
| **序号** | **输入及操作说明** | **期望测试结果** | **结果** |
| 1 | 点击“首页”按钮 | 进入问答系统主页 | 通过 |
| 2 | 使用自然语言句式提问 | 判空处理 | 通过 |
| 3 | 点击“查询”按钮 | 按钮进入“loading”状态 | 通过 |
| 4 | 等待“loading”结束 | 展示准确的应答结果 | 通过 |
| 5 | 使用非自然语言句式提问 | 提示“超纲问题” | 通过 |

句式规则匹配功能的测试用例执行单如表 5-9所示。测试重点为数据正确性。额外流程还包括输入框清空。

上述功能测试用例共计 5 项，测试覆盖率符合 Pareto 法则。基于功能测试结果，本系统提出的功能性需求均得到实现，能够稳定地提供相关服务，满足

用户的业务需求。在后续系统版本迭代更新后，仍可依据本节定义的测试用例和测试步骤进行回归测试。

## 5.5 性能测试

性能测试属于系统测试阶段的非功能性测试，通过自动化的测试工具模拟正常、峰值以及异常负载等情况来验证软件系统在给定工作负荷下的稳定性、可扩展性、并发能力和响应能力，包括负载测试、压力测试等技术手段。

表 5-10: 缓存压力测试对比表

| 测试编号 | 无缓存 | 有缓存 |
|---|---|---|
| PT1 | 216.14 | 605.23 |
| PT2 | 204.75 | 616.32 |
| PT3 | 225.23 | 598.95 |
| PT4 | 217.30 | 632.74 |
| PT5 | 217.61 | 588.53 |
| **Average** | **216.21** | **608.35** |

缓存压力测试对比如表5-10所示。每秒查询率（Queries Per Second，QPS）指查询一个特定的服务器在规定时间内所处理流量多少，即每秒能处理的响应请求数，用于量化服务器的最大吞吐能力。使用压力测试工具 Apache Bench 模拟 200 并发量场景下对服务器发起网络请求。在只读的理想情况下，Redis 缓存使 QPS 性能提升约 2.81 倍。

表 5-11: 首屏时长测试对比表

| 测试编号 | 无压缩 (s) | | | 静态压缩 (s) | | |
|---|---|---|---|---|---|---|
| | Chrome | Firefox | Electron | Chrome | Firefox | Electron |
| FCP1 | 10.13 | 10.15 | 10.21 | 2.41 | 2.32 | 2.38 |
| FCP2 | 10.43 | 10.19 | 10.63 | 2.02 | 2.73 | 2.13 |
| FCP3 | 11.27 | 10.32 | 11.10 | 2.47 | 2.66 | 2.83 |
| FCP4 | 10.18 | 10.29 | 10.16 | 2.39 | 2.30 | 2.51 |
| FCP5 | 10.33 | 10.38 | 10.55 | 2.18 | 2.37 | 2.29 |
| **Average** | **10.47** | **10.27** | **10.53** | **2.29** | **2.48** | **2.43** |

首屏时间（Speed Index），即首次内容绘制（First Content Paint，LCP），表示填满首屏所消耗的时间，即主要内容渲染完成的时长。一次完整的 HTTP 请求包括：1. 域名解析；2. 发起 TCP 的 3 次握手；3. 建立 TCP 连接后发起 http

请求；4. 服务器响应 http 请求，浏览器得到 html 代码；5. 浏览器解析 html 代码，并请求 html 代码中的资源。通过分析一次完整的 HTTP 请求流程，为降低白屏时长，需要尽可能压缩静态资源以提升页面加载速度。首屏时长测试对比如表 5-11所示，通过开启 GZIP 压缩机制使静态资源包缩小为原先的四分之一，进而将首屏时长限制在 2~3 秒。

## 5.6 本章小结

本章主要涉及软件生命周期的测试阶段。为保证开发、运行和维护中系统的可用性、可靠性和稳定性，分别从代码层面、接口层面、功能层面以及性能层面进行单元测试、接口测试、功能测试和性能测试。通过详细说明测试设计和结果，证明在一定并发量和输入异常数据的情况下各模块及其子模块均可正常运行，满足用户对于业务功能和服务可用的要求。

# 第六章　总结与展望

## 6.1　总结

伴随着互联网的快速普及和发展，软件产品层出不穷。由于“焦油坑”的存在，如何保障软件的质量贯穿整个软件过程。缺陷报告是缺陷报告人员（以测试人员为主）和软件开发人员之间按照预先约定格式描述缺陷生命周期的文档。缺陷跟踪管理系统能够记录、分析和更新软件开发或软件测试中可复现的缺陷及其状态，有效记录和管理已知缺陷，有助于把控需求的完成度和软件本身的质量。

在众包测试等大规模人工协作场景，尤其是用户交接时，新用户学习和熟悉复杂缺陷库的门槛很高。问答技术作为信息检索的高级形式，能够快速且准确地对输入问题作出反馈，节约用户排查和筛选的时间。相较于关系型数据库，图结构在存储三元组知识方面的可解释性更强，基于图的算法不会因规模扩大而降低查询速度，因此逻辑查询更加清晰直观，形式化查询效率更高，知识图谱也更适合缺陷库问答。

结合上述分析，本文面向缺陷跟踪系统 Bugzilla 的开源数据集 Eclipse 项目 issue，对知识图谱问答系统在软工领域的落地开展相关研究。

本文首先介绍项目的背景和意义，包括知识图谱和问答系统的国内外研究现状，基于成熟的理论支持，受现有工作启发，提出面向 Bugzilla 的知识图谱问答系统的研究计划。其次，介绍和说明项目涉及到的数据处理、文本预处理和系统研发相关的科学理论、技术框架和开源工具以及技术选型理由。按照研发流程的先后顺序，阐述相应技术或工具的基本原理和应用场景。然后，对系统进行涉众分析和需求分析，在此基础上描述系统用例。根据分析结果落实系统总体架构、数据预处理、语义识别、本体层定义和持久化存储的总体设计，并使用“4+1”视图展示系统的架构体系和模块划分。接着，利用活动图、时序图和关键代码等技术阐述系统的详细设计和编码实现，对五个主要模块以界面截图和评价指标等方法辅助展示业务实现效果。最后为保证系统的可用性、可靠性和稳定性，遵循 Pareto 法则，按序完成单元测试、接口测试、功能测试和性能测试，从而保证系统的软件质量。

最终研发出一款基于混合问答框架的问答系统。软件系统由展示层、转发层、数据层、存储层和基础层组成。展示层基于 Electron+React 框架搭建跨端

桌面应用，Antd+Less 实现模块化样式隔离，Echarts 可视化知识图谱；转发层选用 Django 框架，更好地兼容 Python 三方库 API 接口，并根据模式匹配规则分发自然语言问题；数据层在完成数据清洗后预训练句嵌入索引模型，用于缺陷报告查重和缺陷标识查询；存储层通过构建知识图谱持久化缺陷库数据于 Neo4j，MySQL 保存待审批缺陷报告；基础层部署于 Docker 容器，利用 Nginx 进行负载均衡和反向代理，Redis 集群对热点数据进行缓存，目的是提高系统的性能和吞吐量。

## 6.2 展望

基于限定领域场景的知识图谱问答系统的软工落地还处在起步摸索阶段，尚且没有明确而统一的解决方案。本系统仍存在一些不足和改进的空间，具体包括如下几点：

1）完善多跳句式规则。目前系统只针对部分多跳句式进行规则标注。限定领域场景的多跳语义需要进行更多的特判处理，这是一件工作量较大的任务。现有非标问题映射标准问题的预设模板数量还不足以回答绝大多数多跳问句。因此，系统在应对复合句提问时转化为形式化查询的能力存在不足。

2）识别复杂问题语义。鉴于问答用户的受教水平、认知习惯和文化风格不尽相同，描述缺陷特征的方式也大不一样，此时自然语言问题将会变得非常复杂。例如用户以掺杂无关信息的一长串事实描述作为输入问题，这对于语义解析和句向量计算将是巨大挑战，意图识别会变得异常困难。因此，系统在应对多模态复杂深层语义提问时存在不足。

3）简化终身学习流程。当前句嵌入索引模型为静态模型，即预训练后模型语义识别能力固定不变。为实现终身机器学习，需要持续集成人员在缺陷库更新后手动再训练。在现在的大数据背景下，迁移并整合知识是有必要的。因此，系统在应对可变缺陷库，尤其是大规模可变缺陷库时，终身机器学习的更新能力不足。

综上所述，本文提出的面向 Bugzilla 的知识图谱问答系统仍有许多可以提升或优化的地方。针对多跳句式规则，归纳常见句型，尽可能地处理各种问法；针对复杂问题语义，进一步完善停用词表、无关词性表和特征词白名单；针对终身学习流程，在解决自动化模型融合或模型追加的难点后，提供一键式对外接口。

# 参考文献


[1] FP Brooks Jr. The mythical man-month: essays on software engineering[M]. London, UK : Pearson Education, 1995.

[2] WS Humphrey, WR Thomas. Reflections on management: How to manage your software projects, your teams, your boss, and yourself[M]. London, UK : Pearson Education, 2010.

[3] P Graham. Hackers & painters: big ideas from the computer age[M]. Sebastopol, USA : O'Reilly Media, 2004.

[4] JZ Pan, G Vetere, JM Gomez-perez, et al. Exploiting linked data and knowledge graphs in large organisations[M]. New York, USA : Springer, 2017.

[5] AM Collins, MR Quillian. Retrieval time from semantic memory[J]. Journal of verbal learning and verbal behavior, 1969, 8(2) : 240 – 247.

[6] T Hofweber. Logic and ontology[J]. Stanford Encyclopedia of Philosophy, 2004.

[7] TJ Berners-Lee. Information management: A proposal[R]. 1989.

[8] TJ Berners-Lee. What the Semantic Web can represent[J], 1998.

[9] TJ Berners-Lee, others. Semantic web road map[J], 1998.

[10] F Bauer, M Kaltenböck. Linked open data: The essentials[J]. Edition mono/-monochrom, Vienna, 2011, 710.

[11] A Singhal. Introducing the knowledge graph: things, not strings[J]. Official google blog, 2012, 5 : 16.

[12] A Abele, JP Mccrae, P Buitelaar, et al. Linking open data cloud diagram 2017[J]. URL: http://lod-cloud. net (Accessed: 31.12. 2018). Insight-Centre, 2017.

[13] M Gaur, A Desai, K Faldu, et al. Explainable AI Using Knowledge Graphs[C] // ACM CoDS-COMAD Conference. 2020.

[14] Y Li, V Zakhozhyi, D Zhu, et al. Domain Specific Knowledge Graphs as a Service to the Public: Powering Social-Impact Funding in the US[C] // Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020 : 2793 – 2801.

[15] J Yuan, Z Jin, H Guo, et al. Constructing biomedical domain-specific knowledge graph with minimum supervision[J]. Knowledge and Information Systems, 2020, 62(1) : 317 – 336.

[16] P Cimiano, C Unger, J Mccrae. Ontology-Based interpretation of natural language[J]. Synthesis Lectures on Human Language Technologies, 2014, 7(2) : 1 – 178.

[17] AM Turing, J Haugeland. Computing machinery and intelligence[M]. Cambridge, MA : MIT Press, 1950.

[18] J Weizenbaum. ELIZA—a computer program for the study of natural language communication between man and machine[J]. Communications of the ACM, 1966, 9(1) : 36 – 45.

[19] BF Green, AK Wolf, C Chomsky, et al. Baseball: an automatic question-answerer[C] // Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference. 1961 : 219 – 224.

[20] WA Woods. Progress in natural language understanding: an application to lunar geology[C] // Proceedings of the June 4-8, 1973, national computer conference and exposition. 1973 : 441 – 450.

[21] T Winograd. Shrdlu: A system for dialog[J], 1972.

[22] DG Bobrow, RM Kaplan, M Kay, et al. GUS, a frame-driven dialog system[J]. Artificial intelligence, 1977, 8(2) : 155 – 173.

[23] RC Schank, RP Abelson. Scripts, plans, and knowledge[C] // IJCAI : Vol 75. 1975 : 151 – 157.

[24] RS Wallace. The anatomy of ALICE[G] // Parsing the turing test. New York, USA : Springer, 2009 : 181 – 210.

[25] EM Voorhees. The TREC question answering track[J]. Natural Language Engineering, 2001, 7(4) : 361 – 378.

[26] R High. The era of cognitive systems: An inside look at IBM Watson and how it works[J]. IBM Corporation, Redbooks, 2012, 1 : 16.

[27] JR Bellegarda. Spoken language understanding for natural interaction: The siri experience[J]. Natural interaction with robots, knowbots and smartphones, 2014 : 3 – 14.

[28] MB Hoy. Alexa, Siri, Cortana, and more: an introduction to voice assistants[J]. Medical reference services quarterly, 2018, 37(1) : 81 – 88.

[29] DJ Teece. China and the reshaping of the auto industry: A dynamic capabilities perspective[J]. Management and Organization Review, 2019, 15(1) : 177 – 199.

[30] Q Hu, Y Lu, Z Pan, et al. Can AI artifacts influence human cognition? The effects of artificial autonomy in intelligent personal assistants[J]. International Journal of Information Management, 2021, 56 : 102 – 250.

[31] XT Xiao, SI Kim. A study on the user experience of smart speaker in China-focused on Tmall Genie and Mi AI speaker[J]. Journal of Digital Convergence, 2018, 16(10) : 409 – 414.

[32] L Wu, X Huang, L You, et al. Fduqa on trec2004 qa track[J]. system, 2004, 1(90) : 39 – 1.

[33] X Huang, J Zhang, D Li, et al. Knowledge graph embedding based question answering[C] // Proceedings of the twelfth ACM international conference on web search and data mining. 2019 : 105 – 113.

[34] U Sawant, S Garg, S Chakrabarti, et al. Neural architecture for question answering using a knowledge graph and web corpus[J]. Information Retrieval Journal, 2019, 22(3) : 324 – 349.

[35] M Dubey, D Banerjee, D Chaudhuri, et al. EARL: joint entity and relation linking for question answering over knowledge graphs[C] // International Semantic Web Conference. 2018 : 108 – 126.

[36] A Saffari, A Oliya, P Sen, et al. End-to-End Entity Resolution and Question Answering Using Differentiable Knowledge Graphs[C] // Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 2021 : 4193 – 4200.

[37] I Düntsch, G Gediga. Rough set data analysis[J]. Encyclopedia of computer science and technology, 2000, 43(28) : 281 – 301.

[38] S Sumathi, S Sivanandam. Introduction to data mining and its applications : Vol 29[M]. New York, USA : Springer, 2006.

[39] E Rahm, HH Do. Data cleaning: Problems and current approaches[J]. IEEE Data Eng. Bull., 2000, 23(4) : 3 – 13.

[40] 徐俊刚, 裴莹. 数据 ETL 研究综述 [J]. 计算机科学, 2011, 38(4) : 15 – 20.

[41] D Fensel, U Şimşek, K Angele, et al. Introduction: what is a knowledge graph[G] // Knowledge Graphs. New York, USA : Springer, 2020 : 1 – 10.

[42] M Ehrig. Ontology alignment: bridging the semantic gap : Vol 4[M]. Berlin, Germany : Springer Science & Business Media, 2006.

[43] H Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods[J]. Semantic web, 2017, 8(3) : 489 – 508.

[44] Z Chen, B Liu. Lifelong machine learning[J]. Synthesis Lectures on Artificial Intelligence and Machine Learning, 2018, 12(3) : 1 – 207.

[45] C Manning, H Schutze. Foundations of statistical natural language processing[M]. Cambridge, USA : MIT press, 1999.

[46] GD Forney. The viterbi algorithm[J]. Proceedings of the IEEE, 1973, 61(3) : 268 – 278.

[47] CD Manning, M Surdeanu, J Bauer, et al. The Stanford CoreNLP natural language processing toolkit[C] // Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations. 2014 : 55 – 60.

[48] K Toutanova, D Klein, CD Manning, et al. Feature-rich part-of-speech tagging with a cyclic dependency network[C] // Proceedings of the 2003 Human Language

Technology Conference of the North American Chapter of the Association for Computational Linguistics. 2003 : 252 – 259.

[49] M Marcus, B Santorini, MA Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank[J]. Computational Linguistics, 1993.

[50] J Buckman, A Roy, C Raffel, et al. Thermometer encoding: One hot way to resist adversarial examples[C] // International Conference on Learning Representations. 2018.

[51] J Ramos, others. Using tf-idf to determine word relevance in document queries[C] // Proceedings of the first instructional conference on machine learning : Vol 242. 2003 : 29 – 48.

[52] R Mihalcea, P Tarau. Textrank: Bringing order into text[C] // Proceedings of the 2004 conference on empirical methods in natural language processing. 2004 : 404 – 411.

[53] T Hofmann. Probabilistic latent semantic indexing[C] // Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval. 1999 : 50 – 57.

[54] DM Blei, AY Ng, MI Jordan. Latent dirichlet allocation[J]. the Journal of machine Learning research, 2003, 3 : 993 – 1022.

[55] T Mikolov, I Sutskever, K Chen, et al. Distributed representations of words and phrases and their compositionality[C] // Advances in neural information processing systems. 2013 : 3111 – 3119.

[56] A Joulin, E Grave, P Bojanowski, et al. Fasttext.zip: Compressing text classification models[C] // 5th International Conference on Learning Representations. 2017.

[57] J Pennington, R Socher, CD Manning. Glove: Global vectors for word representation[C] // Proceedings of the 2014 conference on empirical methods in natural language processing. 2014 : 1532 – 1543.

[58] ME Peters, M Neumann, M Iyyer, et al. Deep Contextualized Word Representations[C] // Proceedings of the 2018 Conference of the North American Chapter of

the Association for Computational Linguistics: Human Language Technologies. 2018 : 2227 – 2237.

[59] A Radford, K Narasimhan, T Salimans, et al. Improving language understanding by generative pre-training[J], 2018.

[60] J Devlin, MW Chang, K Lee, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C] // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2019 : 4171 – 4186.

[61] N Reimers, I Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks[C] // Proceedings of the 2019 conference on empirical methods in natural language processing. 2019.

[62] F Schroff, D Kalenichenko, J Philbin. Facenet: A unified embedding for face recognition and clustering[C] // Proceedings of the IEEE conference on computer vision and pattern recognition. 2015 : 815 – 823.

[63] J Johnson, M Douze, H Jégou. Billion-scale similarity search with gpus[J]. IEEE Transactions on Big Data, 2019.

[64] A Fedosejev. React.js essentials[M]. Birmingham, UK : Packt Publishing Ltd, 2015.

[65] A Holovaty, J Kaplan-moss. The definitive guide to Django: Web development done right[M]. New York, USA : Apress, 2009.

[66] D Fernandes, J Bernardino. Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB.[C] // Data. 2018 : 373 – 380.

[67] J Webber. A programmatic introduction to neo4j[C] // Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity. 2012 : 217 – 218.

[68] B Schwartz, P Zaitsev, V Tkachenko. High performance MySQL: optimization, backups, and replication[M]. Sebastopol, USA : O'Reilly Media, 2012.

[69] DB Duldulao, RJL Cabagnot. Getting Started with the Node Package Manager[G] // Practical Enterprise React. New York, USA : Springer, 2021 : 11 – 19.

[70] J Carlson. Redis in action[M]. New York, USA : Simon and Schuster, 2013.

[71] J Nielsen. Usability inspection methods[C] // Conference companion on Human factors in computing systems. 1994 : 413 – 414.

[72] PB Kruchten. The 4+1 view model of architecture[J]. IEEE software, 1995, 12(6) : 42 – 50.

[73] J Wei, K Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks[J]. arXiv preprint arXiv:1901.11196, 2019.

[74] K Beck. Test-driven development: by example[M]. Boston, USA : Addison-Wesley Professional, 2003.

# 简历与科研成果

**基本情况** 钱雨波，男，汉族，1997 年 10 月出生，浙江嘉兴人。

**教育背景**

**2020 年 9 月 ~ 2022 年 6 月** 南京大学软件工程 硕士

**2016 年 9 月 ~ 2020 年 6 月** 延边大学计算机科学与技术 本科

**读研期间的成果**

1. Desheng Wang, **Yubo Qian**, Yu Xing, Yansong Li, Yifei Yang, Qing Wu, Tieke He: Problem Kit System for Expression Ability Evaluation. DSA 2020
2. Yansong Li, Yu Xing, **Yubo Qian**, Desheng Wang, Yifei Yang, Qing Wu, Tieke He: Design and Implementation of Test System for Expression Ability Evaluation. DSA 2020

# 致　　谢

至此，廿年的求学生涯即将画上句点。天意怜幽草，人间重晚情。顿忆往事，六年小学时光，天真无邪、奋发踔厉；六年中学岁月，升腾跌宕、茫然若失；六年大学生涯，幡然醒悟、拨云见日。心怀感恩，所遇皆美；心怀善念，所遇皆暖。今昔之感，思绪万千。谢谢，这一路上，遇到的人，馈赠的美好。

首先，栽培之恩。感谢导师何铁科博士。何老师平易近人，良师益友。解探学术科研，参预项目课题，规划人生的发展方向，分享生活的趣闻轶事，令研究生涯完整而充实。在毕设选题、模型构建和论文撰写阶段给予细致的指导、提出宝贵的建议。厚谊常存魂梦里，深恩永志我心中。

其次，知遇之恩。感谢 iSE 陈振宇教授。陈老师公正开明，师道尊严，是标杆，亦是榜样。在开题答辩、实验进展和写作督促方面牢牢掌控流程节奏，提供许多帮助和指示。因风道感谢，情至笔载援。

再次，煦伏之恩。感谢默默守护着的父母。从呱呱坠地到咿呀学语，从入学升学到择业就业。你们的鼓励，是我前行最大的动力；你们的关怀，是我漂泊最暖的港湾。应似园中桃李树，花落随风子在枝。

另外，相遇之恩。感谢所见所遇的同窗、同僚。离别请不要悲伤，只是暂时去逐梦而已。人逢知己千杯少，终将牢记彼此过往的深谊，永生难忘经历的点点滴滴。高风相宾友，古义仍兄弟。

最后，相知之恩。感谢一直支持和陪伴着的女友。与你相遇，是时；与你相识，是缘；与你相知，是幸；与你相恋，是情；与你相爱，是真；与你相守，是余生。忆君心似西江水，日夜东流无歇时。

恰逢百年未有之大变局，肩负中华民族伟大复兴。愿此生，仰不愧于炎黄先烈，俯不怍于人民子孙。宠辱不惊，看庭前花开花落；去留无意，望天上云卷云舒。行而不辍，未来可期。