文章编号：2095-2163(2020)05-0009-05 中图分类号：TP311 文献标志码：A

基于多源数据融合的 Java 代码知识图谱构建方法研究

苏 佳，苏小红，王甜甜
（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘 要： 开发人员在软件开发过程中有学习和检索来自不同来源不同种类的代码相关知识的需要，代码知识图谱不仅能够对知识资源进行描述，还可以分析和描绘知识间的联系，越来越受到重视。本文提出了一种 Java 代码知识图谱的构建方法，将提取的 Java 代码相关的多源知识融合，进行知识图谱构建，并对现有 API 实体识别方法进行了实验和比较。

关键词： 代码知识图谱；知识融合；实体识别；知识图谱构建

A Construction Method of Java Code Knowledge Graph based on multi-source data fusion

SU Jia, SU Xiaohong, WANG Tiantian
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

【Abstract】 Developers need to learn and retrieve different kinds of code-related knowledge from different sources in the software development process. Code knowledge graph can not only describe knowledge resources, but also analyze and display the relationship between knowledge, which has been paid more and more attention. In this paper, a method of constructing knowledge graph of Java code is proposed. Multi-source knowledge related to Java code is extracted and fused to construct knowledge graph, and existing API entity recognition methods are tested and compared.

【Key words】 code knowledge graph; knowledge fusion; entity identification; knowledge graph construction

0 引 言

知识图谱(Knowledge Graph)是 Google 在 2012 年正式提出的概念，它以图的形式来表达客观世界的实体(概念，人，事物)以及实体之间关系的知识库，后来得到广泛关注和应用研究。以其大规模、可解释、可推理等特点，现已经应用于智能问答、语义搜索、可解释推荐、情报分析、决策支持、知识导航和医疗等领域。

知识图谱是一个由知识和知识间的关系组成的结构化的语义知识库，典型的知识图谱采用三元组(头实体，关系，尾实体)描述事实。知识图谱每个节点表示客观世界中存在的概念或者实体，边则描述概念或者实体之间的语义关系。知识图谱提供了一个通用的结构化框架来存储和表示知识，从而实现基于实体和关系的挖掘、推理和分析。在软件工程领域，代码知识图谱目前研究较少，相关表示方法主要有以下几种：

Zeqi Lin 等人[1]分析了代码中的结构化信息，提取代码元素之间的结构依赖关系(方法调用、继承关系)来构造代码图谱(Code Graph)，再利用 TransR 方法学习共享表示空间的嵌入表示，再计算 tf-idf 为代码元素加权，利用土堆移动距离(EMD)，通过移动“分布质量”的方式，把一个分布转换为另一个分布所需要的最小工作量，来计算文档与代码之间的距离；同一个团队[2]又利用软件源代码中特定于软件的概念知识来改进 API 学习资源的检索，利用 Recodoc 和基于关键字的启发式算法提取文本中的 API，生成 API Graph，每个查询或文档都表示为一个加权的 API 实体集合。利用多关系数据嵌入算法 TransR 计算 API 实体相似性，在传统方法获得的检索排名基础上，用成对的 API 实体相似性来计算文档和查询之间的概念相关性得分，对 API 学习资源检索结果进行重排序(文档获得更高分数排到顶部，反之底部)，提高检索准确性。

Wang L 等人[3]从软件历史仓库中收集 bug 数据，如 bug 报告、commit 提交信息、代码文件等。用自然语言处理技术对数据进行预处理，提取 bug 描述信息和相关开发者的信息。使用 LDA 主题模型

基金项目： 国家自然科学基金项目(61672191)。

作者简介： 苏 佳(1995-)，男，硕士研究生，主要研究方向：程序分析、智能软件、深度学习；苏小红(1966-)，女，博士后，教授，博士生导师，主要研究方向：程序分析、软件错误定位、软件漏洞检测；王甜甜(1980-)，女，博士，副教授，主要研究方向：程序分析、软件错误定位与修复。

收稿日期： 2020-03-04

来处理 bug 描述信息,建立不同数据之间的联系。根据 bug 报告的属性(depend on 和 duplicate)建立 bug 之间的关系。根据 bug 描述信息的文本相似度,建立不同 bug 之间的相似关系。用同样的方法建立 commit 之间的相似关系。最后利用 bug id,来构建 bug、commit 之间的关系,最终得到 bug 知识图谱(Bug Knowledge Graph)。

LiuMingwei 等人[4]爬取了 API 官方文档,提取了 API 相关的结构性知识和描述性知识,来构建 API 知识图谱。他们首次提出了 API 概念,利用词的词汇相似性和上下文相似性,将词进行层次聚类,连接 API 的不同的描述性句子,建立 refer to 关系。在通用知识图谱的基础上,用分类器获取软件概念,将软件概念和 API 概念、API 实体用句向量计算相似度并建立 related to 关系,生成了更好的面向 task 的 API 摘要。

不同的应用场景下,需要定义不同的知识图谱的实体和关系来满足不同的需要。现有的代码知识图谱构建的数据来源都较为单一,缺乏对代码解释性知识的获取和融合。

本文在分析代码相关知识图谱国内外研究现状的基础上,提出一种基于多源异构数据的 Java 代码知识图谱构建方法,该代码知识图谱可用于代码的知识检索,代码摘要等场景。

1 Java 代码知识图谱构建方法

1.1 Java 代码知识图谱设计

为了挖掘 API 知识之间的显、隐式关系和丰富代码知识,引入了 API 相关概念[4],记为 api_concept 实体。本文设计的代码知识图谱包含 package、class、method、functional _ description、question_description、api_concept 等 6 种实体,以及 haveClass、extend、haveMethod、hasFunctional Description、hasQuestionDescription、referTo 等 6 种关系。构建流程如图 1 所示。

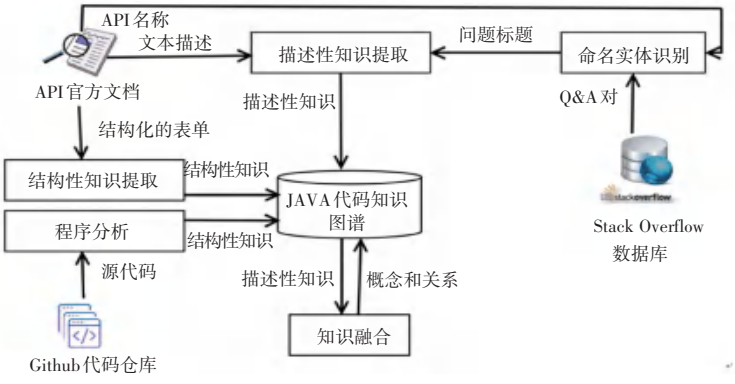

图 1 Java 代码知识图谱构建流程

Fig. 1 Java Code Knowledge Graph Construction Process

本文将从 github、API 官方文档以及 Stack Overflow 问答社区(以下简称 SO)的 Q&A 对等三种数据源进行数据挖掘,提取 Java 语言相关的代码知识。在获取到来自不同数据来源的知识后,通过知识融合将它们合理、统一地组织到同一个图网络中,构建 Java 代码知识图谱。

1.2 Java 代码知识提取

本文将从三种数据来源中提取 java 代码的结构性知识、描述性知识以及概念和关系,并对这些不同来源的知识进行融合。结构性知识主要包括 API 的相关实体,package,class,method,还有 parameter 和 return values 等,在其他应用场景中还可以添加相关属性,例如完全限定名称,加入的版本号以及 API Document URL。代码中的相关实体,除 package,class,method 等外,还包括方法体内的 API 调用序列。描述性知识主要包括来自 API 文档的功能性描述以及 SO 的问题性描述,前者主要描述了 API 的功能,后者主要描述了使用者使用时遇到的相关问题。

1.2.1 基于 AST 的 JAVA 源代码的代码知识提取

从 github 获取的源码中提取的知识有:(1)类相关的实体和关系。(2)方法相关的实体和关系。(3)方法体中的 API 调用序列。具体来说,本文通过源码获取的实体有 package,class,method;关系有 haveClass,extend, havaMethod。

抽象语法树(Abstract Syntax Tree, AST),是通过使用树状结构来表示源代码的抽象语法结构,它作为程序分析中一种重要的中间表示形式,在代码分析、代码重构、语言翻译等领域得到广泛的应用。现有的一些相关工具中都含有将源代码直接转换为抽象语法树的模块。用程序分析技术,通过解析源代码的 AST,遍历包定义、类定义、成员变量表以及

方法定义表，可以获取所需要的结构性知识。在方法体内提取 API 调用序列，通过访问 methodInvoke 节点，根据节点绑定的数据信息，进行正则匹配和过滤即可获得 API 调用序列，它除了作为代码方法级别的一种知识，还在源代码和代码知识图谱之间通过 API 实体建立了联系。

1.2.2　基于网络爬虫的 API 文档的代码知识提取

基于网络爬虫从 API 官方文档爬取数据并提取代码结构性知识以及描述性知识的流程如图 2 所示。

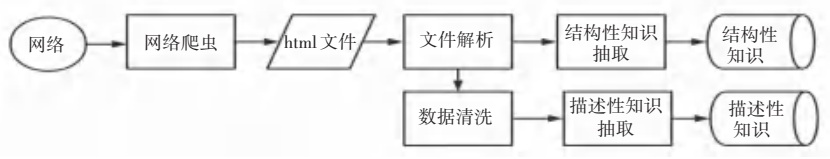

图 2　基于网络爬虫的 API 文档的代码知识构建流程

Fig. 2　Code knowledge construction process of API document based on web crawler

结构性知识提取：利用正则表达式解析超链接，可以获得 package，class，method 实体，以及 hasClass，hasMethod 关系。识别表格中的<td></td>标签，可以获得 returnType，parameter 属性。

描述性知识提取：为了获取完整的描述性知识，需要利用 bs4 解析 html 文档，按以下规则进行网页内容清洗：(1)恢复被“< p >”和“< li >”等标记打断的句子；(2)<blockquote></blockquote> 用“_CODE__”替换代码片段；(3)<code></code>标签直接过滤，留下内容。其中，从 API 官方文档中提取的 method description 涵盖了 API 的功能描述以及使用方法，可以提供 Java 方法的相关信息，作为 java 代码的一种知识。

1.2.3　基于启发式规则的 Stack Overflow 代码知识提取

从 SO 中获取知识，主要是获取和 API 相关的问题描述作为描述性知识，具体流程如图 3 所示。

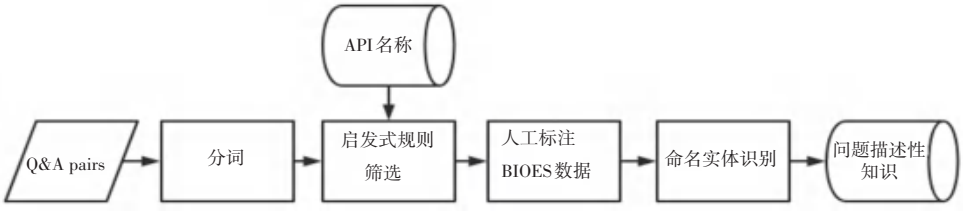

图 3　Stack Overflow 的代码知识构建流程

Fig. 3　Code knowledge construction process from Stack Overflow

获取 SO 数据后，按照标签<java>获取 Q&A 对，并进行分词。由于问答语句是自然语言，是非结构化数据，为了将 Q&A 对和 API 联系起来，需要识别和 API 有关的 Q&A 对，并识别自然语言描述中的 API 实体。

根据之前提取 API 文档中的 API 实体，分别记录全限定名称和非限定名称，按照以下启发式规则，获取 Q&A 数据和 API 之间的对应关系：

(1)如果 API 的全限定名称出现在标题或者问题描述中，那么该问题和这个 API 相对应。(2)如果 question body 中含有指向 API 文档的超链接，那么该问题和链接的 API 相对应。(3)如果 question body 中含有<code>标签，那么该问题和<code></code>中间的非限定名称对应；为 API 实体和 Q&A 数据建立 hasQuestionDescription 关系，如果一个 API 实体对应多个 Q&A 数据，那么只连接 score 分数最高的 Q&A。

为了增强可扩展性，本文在基于启发式规则获取到的 Q&A 对和 API 的对应关系基础上，采用 NLP 领域中的命名实体识别模型和方法进行 Java API 实体识别。

1.3　JAVA API 实体识别

命名实体识别(Named Entity Recognition，NER)是 NLP 的基础任务，指从文本中识别出命名性指称项。在本文中，将用来识别 SO 中的 Q&A 语句中的 API 实体。

条件随机场(Conditional Random Field，CRF)是一种基于机器学习的方法，在马尔科夫随机场的基础上增加了观测变量，将所有特征进行全局归一化，可以获得全局最优解。本文用启发式方法选取如下特征来进行 CRF 模型训练：(1)当前词是否首字母大写，其他字母小写。(2)当前词的词性。(3)前一个词的词性。(4)当前词是否含有“.”。(5)当前词是否全部为大写。(6)当前词的后缀。(7)当前词是否含有数字。

近年来，越来越多的研究已经说明了深度学习

在 NLP 任务上的有效性。BiLSTM－CNNs－CRF 在双向 LSTM－CNNs 的基础上，加入了 CRF，从而对于输出序列进行优化。BERT 是一个用 Transformer 作为特征提取器的深度双向预训练语言理解模型，由多层的双向 Transformer 连接而成，利用 Position Embedding 来学习位置信息，通过训练 MLM（Masked Language Model）和 NSP（Next Sentence Prediction）任务获取到丰富的语言知识，在多种 NLP 任务中获得突破性进展。将 BERT 应用于 NER 任务，只需用 BERT 替换 word embedding 来进行语义编码。

本文使用 CRF，BiLSTM－CNNs－CRF 以及用 BERT 的三种模型对手工标注数据进行 API 命名实体识别。

1.4 JAVA 代码知识融合

为了将从不同数据源获得的知识统一在同一个知识图谱当中，需要对这些知识进行融合。融合主要包括两方面：一方面是 API 实体融合；另一方面是 API 概念的融合。

API 实体融合指的是根据 API 实体名称进行统一和融合，建立其他知识和 API 实体之间的关系。利用 API 官方文档中获取的 API 知识，构建基本的 API 知识图谱作为基础，再往 API 实体上补充来自源码的结构性知识，以及 SO 的问题描述，最后将 API 功能性描述和 SO 上的问题性描述和 API 概念相连接。

API 概念融合指的是从和 API 相关的描述性知识中提取 API 概念，作为描述之间的桥梁，从而建立描述性知识之间的关系。本文中的 API concept 是由基本名词短语构成，所以需要进行基本名词短语提取。

依存句法分析（Dependency Parsing，DP）通过分析语言单位内成分之间的依存关系揭示其句法结构。使用语义依存刻画句子语义，通过词汇所承受的语义框架来描述该词汇，而其数目相对词汇来说数量小。这个框架表示大部分的句子，同时也能据此迅速提取句子的核心内容。本文使用斯坦福句法分析器（stanford corenlp）对前文提取的描述性知识句法分析并提取依存关系。在句法分析后，获取所有 NP 节点作为候选节点。在此基础上，按如下规则进行剪枝过滤：（1）去除句法树上子节点含有 NP 的节点。（2）根据依存关系｛compound 复合，nmod 复合名词修饰（只保留连词 or，and），amod 形容词修饰（过滤常见词）｝，保留 NP 内的节点。（3）最后过滤停用词和无关符号，即可得到该句所需要的 API 概念集合。api_conception 和对应的描述性语句之间建立（description，referTo，concept）关系，从而连接了不同的描述性语句。

1.5 JAVA 代码知识图谱构建和可视化

Neo4j 是一种图数据库管理系统，属于原生图数据库，其使用的存储后端专门为图结构数据的存储和管理进行定制和优化的，在图上互相关联的节点在物理地址也指向彼此，因此更能发挥出图数据的优势。知识图谱非常适合用 Neo4j 进行存储，基于 Neo4j 生成部分 java 代码知识图谱的可视化结果如图 4 所示。

图 4 java 代码知识图谱的部分可视化结果

Fig. 4 Partial visualization results of Java code knowledge graph

2 实验和分析

2.1 实验设置

Q&A 数据作为有标签的样本数据，标签为 Q&A 含有的 API。为保证数据的质量，从中按照 score 得分，选取 Top－2000 数据，进行人工标注，API 命名实体识别。标注采用 BIOES 标注方法，用 B 表示这个

词处于一个实体的开始(Begin)，I 表示内部(Inside)，O 表示外部(Outside)，E 表示这个词处于一个实体的结束，S 表示这个词自己就可以组成一个实体(Single)。标注示例如图 5 所示。

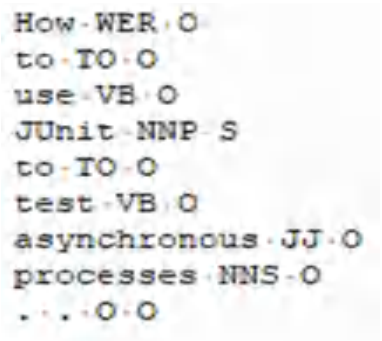

图 5　API 命名实体识别的 BIOES 标注示例

Fig. 5　Examples of BIOES Labeling for API Named Entity Recognition

其中第一列是自然语言句子中的单词，第二列是单词的词性，第三列为人工按 BIOES 的标注的标签。

2.2　评估指标

本文在 API 实体识别时，采用如表 1 所示的准确率、精确率、召回率以及 *F*1 值作为评价指标。

表 1　API 命名实体识别评价指标

Tab. 1　API Named Entity Identification Evaluation Criterion

指标	公式
准确率(*Acc*)	$\frac{TP + TN}{TP + TN + FP + FN}$
精确率(*P*)	$\frac{TP}{TP + FP}$
召回率(*R*)	$\frac{TP}{TP + FN}$
*F*1	$\frac{2TP}{2TP + FP + FN}$

其中：*TP* 将 API 实体预测为 API 实体数，*FN* 将 API 实体预测为其他类型数，*FP* 将其他类型预测为 API 实体数，*TN* 将其他类型预测为其他类型数。

2.3　实验结果和分析

2000 个样本随机选用 1600 个进行训练，200 个作为验证集，200 个作为测试集。测试句子中 200 个句子一共有 1394 个 token，203 个 API 实体，实验结果如表 2 所示。

表 2　API 命名实体识别测试集实验结果

Tab. 2　Experimental Results of API Named Entity Recognition　%

Model	*Acc*	*P*	*R*	*F*1
CRF	78.48	70.52	60.10	64.89
BiLSTM-CNN-CRF	86.53	78.73	79.44	79.01
BERT-base	**88.02**	**82.06**	**87.86**	**84.86**

可以看出深度学习模型相较于传统的机器学习模型在 API 识别方面具有优势，而 BERT 模型在四个指标上均可以达到最好效果。

3　结束语

本文提出了一种 Java 代码知识图谱的构建方法，将 github 源码、API 文档和 Stack Overflow 问答社区的多源知识进行提取和融合，构建成图谱，并通过实验验证了 BERT 模型相比于其他现有模型可以获得更好的 API 实体识别效果。

参考文献

[1] LIN Z, ZHAO J, ZOU Y, et al. Document Distance Estimation via Code Graph Embedding [C]// The 9th Asia - Pacific Symposium of Internetware. 2017.

[2] LIN Z, ZOU Y, ZHAO J, et al. Improving software text retrieval using conceptual knowledge in source code [C]// 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). ACM, 2017.

[3] WANG L, SUN X, WANG J, et al. Construct Bug Knowledge Graph for Bug Resolution [C]// 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). ACM, 2017.

[4] LIU M, PENG X, MARCUS A, et al. Generating query-specific class API summaries [C]//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019: 120-130.

(上接第 8 页)

[11] MILLER T. Explanation in artificial intelligence: Insightsfrom the social sciences[J]. Artificial Intelligence, 2019, 267: 1-38.

[12] DOSHI - VELEZ F, KIM B. Towards a rigorous science of interpretable machine learning [J]. arXiv preprint arXiv: 1702. 08608, 2017.

[13] MOLNAR, C. Interpretable Machine Learning. 2019. Available online: https://christophm. github. io/ interpretable - ml - book/ (accessed on 22 January 2019).

[14] TEMIZER S, KOCHENDERFER M, KAELBLING L, et al. Collision avoidance for unmanned aircraft using Markov decision processes [C]//AIAA guidance, navigation, and control conference. 2010: 8040.

[15] WEXLER R. When a computer program keeps you in jail: How computers are harming criminal justice [J]. New York Times, 2017, 13.

[16] DONNELLY C, EMBRECHTS P. The devil is in the tails: actuarial mathematics and the subprime mortgage crisis[J]. Astin Bulletin, 2010, 40(1): 1-33.