计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201000127

基于情境感知的 API 个性化推荐

陈　晨[1]　周　宇[1,2]　王永超[1]　黄志球[1,2]
1 南京航空航天大学计算机科学与工程学院　南京 211100
2 南京航空航天大学高安全系统的软件开发与验证技术工信部重点实验室　南京 211100
(786324182@qq.com)

摘　要　在软件开发的过程中,开发人员在遇到编程困境时通常会检索合适的 API 来完成编程任务。情境信息和开发者画像在有效的 API 推荐中起着至关重要的作用,却在很大程度上被忽视了。因而文中提出了一种基于情境感知的 API 个性化推荐方法。该方法利用程序静态分析技术,对代码文件做抽象语法树解析,提取信息构建代码库,并对开发者 API 使用偏好建模。然后计算开发者当前查询语句与历史代码库中查询的语义相似度,检索出 top-k 个相似历史查询。最终利用查询语句信息、方法名信息、情境信息以及开发者 API 使用偏好信息对 API 进行重排序并推荐给开发者。通过模拟编程任务开发的不同阶段,使用 MRR,MAP,Hit,NDCG 评估指标来验证所提方法的有效性。实验结果表明,所提方法的 API 推荐效果优于基准方法,能够为开发者推荐更想要的 API。

关键词:API 推荐;程序静态分析;情境感知;开发者画像
中图法分类号　TP391

Context-aware Based API Personalized Recommendation

CHEN Chen[1],ZHOU Yu[1,2],WANG Yong-chao[1] and HUANG Zhi-qiu[1,2]
1 School of Computer Science and Engineering,Nanjing University of Aeronautics and Astronautics,Nanjing 211100,China
2 Key Laboratory of Safety-Critical Software,Ministry of Industry and Information Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211100,China

Abstract　In the process of software development,developers often search for appropriate APIs to complete programming tasks when encountering programming difficulties. Contextual information and developer portraits play a critical role in effective API recommendation,but they are largely overlooked. This paper proposes a novel context-aware based API personalized recommendation approach. This approach leverages program static analysis technology (abstract syntax tree) to parse the code file to extract information to construct the code base and model developer API usage preferences. Then it calculates the semantic similarity between the developer's current query and the queries in the historical code base,and retrieves top-k similar historical queries. Finally,it leverages the information of query,method name,context and developer API usage preference to re-rank the candidate APIs and recommend to developers. MRR,MAP,Hit and NDCG are used to verify the effectiveness of the method in different stages of simulation programming. The experimental results show that the proposed approach outperforms the baseline method and it is more likely to recommend the APIs that developers want.

Keywords　API recommendation,Program static analysis,Context awareness,Developer portraits

1　引言

在当前开源的软件生态环境下,海量软件的代码可以免费获取,有效的软件复用是提高软件开发效率、降低开发成本的重要方式。研究表明,软件开发人员平均消耗 19%的开发时间来检索待完成的编程任务所需要的代码片段[1]。如何让开发人员快速获取到开源代码以完成现有编程任务,成为当今研究的一大热点。

在实际软件开发的过程中,代码推荐是一种常用的实现代码复用的方法。现有研究提出了各种各样的 API 推荐方

到稿日期:2020-10-22　返修日期:2021-03-12　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划项目(2018YFB1003900);国家自然科学基金(61972197);江苏省自然科学基金(BK20201292);江苏高校"青蓝工程";南京航空航天大学研究生创新基地(实验室)开放基金(kfjj20191606)
This work was supported by the National Key R & D Program of China(2018YFB1003900),National Natural Science Foundation of China(61972197),Natural Science Foundation of Jiangsu Province,China(BK20201292),Qing-Lan Project of Jiangsu Province and Foundation of Graduate Innovation Center in NUAA(kfjj20191606).
通信作者:周宇(zhouyu@nuaa.edu.cn)

法来实现代码复用。通常将这些 API 推荐方法根据输入分为两类:有显式查询和无显式查询。有显式查询的方法通常需要良好设计的查询语句来捕获开发者的编程意图,这种方法被定义为信息检索方法,它将查询语句转化为一个词向量,然后使用文本匹配的方法来寻找最匹配的 API[2]。由于自然语言和代码之间的词法差异,这类研究通常会使用额外的构件包括 API 文档[3]、问答网站中的帖子[1,4]等辅助 API 推荐。无显式查询由于没有显式查询作为输入,因此必须利用情境信息来推断开发者需要使用哪些 API,情境信息一般包括周围的代码片段[5]、API 调用图[6-7]等。现实中一个很常见的现象是开发人员在已经完成了一个编程任务的部分代码后,不知道该如何编写下面的代码,便通过利用诸如 Stack Overflow 问答网站输入查询语句寻找正确的 API 以继续编程。现有的 API 推荐方法要么依赖于查询语句,要么依赖于已经编写的代码片段。然而这些方法有时不能推荐出开发者想要的 API。例如,对于下述查询语句:Copy a file in java,在编程情境已经明确使用高效字符缓冲流(java. io. BufferedReader, java. io. BufferedWriter)来完成时,根据该查询语句推荐出来的 API 可能是 java. io. FileReader. read,这并不能帮助开发者完成编程任务。而单纯利用情境信息,有时无法捕捉开发者的编程意图,当情境信息很少甚至没有时,根本无法推荐出开发者想要的 API。

针对上述问题,本文提出了一种基于情境感知的 API 个性化推荐方法。该方法利用程序静态分析技术,对从 Github 收集的代码文件做抽象语法树解析,提取信息构建代码库,并对开发者 API 使用偏好建模;计算开发者当前查询语句与历史代码库中查询的语义相似度,检索出 top-k 个相似历史查询;根据客户端提取到的情境信息,利用莱文斯坦比计算当前情境信息与检索到的相似历史查询的情境相似度得分;与此同时,分别计算当前方法名与检索到的 top-k 个相似历史查询的方法名的相似度得分以及开发者 API 使用偏好与检索到的 top-k 个相似历史查询的 API 列表的相似度得分;对查询语句相似度得分、方法名相似度得分、情境相似度得分以及开发者 API 使用偏好得分进行加权后再将候选 API 排序推荐给开发者。

本文从 Github 选取 5 名拥有至少五年 Java 开发经验的开发者,提取他们项目中的数据,从每个人的数据集中随机选取 20 条查询,共 100 条查询作为测试集,并通过删除部分代码片段来模拟编程现场以验证方法的有效性。实验结果表明,基于情境感知的 API 个性化推荐方法能有效提高 API 推荐效果,为开发人员输入的自然语言查询提供精准的 API 推荐。

本文工作的主要贡献在于以下 3 个方面:

(1)利用程序静态分析技术,设计启发式规则,提取 Github 项目中的数据集,并进行了清洗和筛选,构建了一个高质量的代码库,用于 API 推荐的研究工作。

(2)首次提出将情境信息融入到基于查询语句的 API 推荐中,通过查询语句检索 top-k 个历史相似查询缩小候选 API 的搜索范围;对开发者当前的编程情境进行编码,与检索到的 top-k 个相似历史查询进行情境相似度计算,利用情境信息对 API 进行排序。

(3)首次将开发者画像融入到基于查询语句的 API 推荐中,通过对提取到的开发者代码信息进行分析,对开发者 API 使用偏好进行建模,利用开发者 API 使用偏好信息对 API 进行排序。

2　相关工作

当前,很多文本检索技术在各种软件工程任务得到了应用[8]。然而,由于自然语言和代码之间的词法差异,这些技术的性能通常是欠佳的[9]。为了弥补这一缺陷,大量的方法被提出。特别是针对 API 推荐,一些方法[9-11]从代码中提取 API 实体,并使用相应的 API 文档来提高排序结果;其他方法[5,12-16]利用 Q&A 帖子来推荐 API 以及代码片段。

Mcmillan 等[9]通过度量开发者查询语句与 API 实体的词汇相似性将拥有得分更高的 API 实体的代码排在前面。Bajracharya 等[10]认为拥有共同标记的代码会使用相同的 API 实体。Ye 等[11]将代码中使用的所有 API 实体的描述连接起来,并直接衡量查询语句与连接后文档的词汇相似度。Rahman 等提出了 Rack[1],构建了一个关键字与 API 映射数据库。该数据库从 Stack Overflow 问题中提取关键字,并从相应接受答案中收集映射的 API。基于此数据库,Rack 推荐给定自然查询语句的 API 类的排名列表。BIKER[5]利用 Stack Overflow 帖子弥合编程任务与 API 之间的知识鸿沟,并结合了 Stack Overflow 问题和 API 文档中的信息来衡量 API 与编程任务描述的相关性。Zhou 等[16]将开发者的反馈整合到推荐循环中,并利用排序学习和主动学习来提高推荐效果。

3　API 推荐方法

本节将对提出的代码推荐方法进行详细介绍,该方法的整体架构如图 1 所示。本方法致力于根据开发人员输入的自然语言查询,推荐出与开发人员当前编程上下文相匹配的 API。本文新提出的方法由 3 个核心模块组成:数据收集与处理、检索相似的历史查询和 API 相关性重排序。

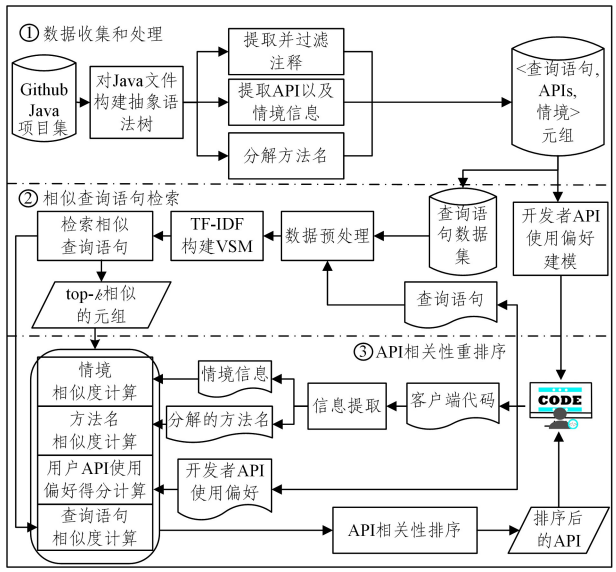


图 1　整体架构

Fig. 1　Overall framework

3.1 数据收集和处理

3.1.1 查询语句以及方法名的提取

Java 方法注释通常包含在"/**"和"*/"之间，并且可以从抽象语法树 AST 的 Javadoc 节点获得。注释的第一句话通常高度概括了 Java 方法所需完成的任务。但是这些句子可能表述欠佳，在这种情况下，本文预定义了一组模板和规则来过滤这些句子。这些模板和规则旨在过滤表明当前 Java 方法是测试、调试、未实现或者无用等，或者由某些 IDE 工具自动生成的(以"TODO"开头的注释)，以及不建议被使用的查询语句。本文还过滤了仅包含数字或标点符号的查询语句。大部分研究通常仅使用注释信息，而忽略了 Java 方法名的语义信息。由于 Java 方法名通常能表达开发者的编程意图，因此本文同时考虑了 Java 方法名包含的丰富的语义信息。将 Java 方法名按照"驼峰"命名法进行分解，并利用分解的 Java 方法名对 API 进行排序。

表 1 预定义的用于过滤带有 Automation 和 Indications 的两类标记的模板

Table 1 Defined templates/rules for filtering comments with two categories of Automation and Indications

Category	Temlates/Rules
Automation	TODO
Indications	Test for*
	For test*
	bug/Bug/BUG*
	For debug*
	WARNING:
	note/Note/NOTE:
	Not implement*
	Not been implemented
	NOT IMPLEMENTED*
	TEST METHOD*
	NEVER EVER SAVE THIS REFERENCE*

注：以 * 结尾的模板表示其小写字母是另一种形式的模板

3.1.2 情境信息以及 API 序列提取

情境信息指编程人员在编写一个编程任务(即一个 Java 方法定义)的过程中方法体内的上下文信息。为了具体化情境信息，本文使用静态程序分析技术集中关注两种类型的声明：类的实例化声明以及方法调用声明。直觉上，这两种声明通常能够确定或者至少强烈影响随后被调用的 API。此外，本文还考虑到与这两种类型语句相关的控制流信息(例如，保留了 if，for，while，break 等)。因为它们表示 API 序列的执行路径，可以用来丰富情境信息。通过对每个 Java 文件构造抽象语法树，并在遍历抽象语法树的过程中提取这些信息。

通过遍历方法体的抽象语法树，用 Java 包名.类名的形式表示提取 Java 类的实例化，用 Java 包名.类名.方法名的形式表示提取到的 Java 方法调用，以及用自定义的映射控制节点与字符的编码表来表示提取到的控制 API 调用的节点信息，从而对情境信息进行编码。情境信息用一个字符串表示。

通过上述步骤，获得一个包含 211 万条〈查询语句，APIs，情境〉元组的数据集。自定义的映射控制节点与字符的编码表如表 2 所列。其中一个大写字母代表一个重要 AST 节点的声明，其小写字母代表该 AST 节点声明的结束。

表 2 自定义的控制节点与字符的编码表

Table 2 Defined encoding table of control nodes and characters

AST Node Type	Symbol	AST Node Type	Symbol
EnhancedForStatement	A…a	ThrowStatemen	H
IfStatement	B…b	SwitchStatement	I…i
ForStatement	C…c	SynchronizedStatement	J…j
ReturnStatement	D	AssertStatement	K
BreakStatement	E	CatchClause	L…l
WhileStatement	F…f	ContinueStatement	M
TryStatement	G…g	DoStatement	N…n

3.2 检索相似的历史查询

为了衡量两个查询语句的相似性，我们需要将文本构建成向量空间模型(Vector Space Model，VSM)[17]。为此，本文先对从方法注释信息中提取到的查询语句进行小写化、分词、删除停用词、提取词干的操作，然后使用 TF-IDF 算法[18]来计算每个词语的权重。

文档频率(Term Frequency，TF)指的是某一个给定的词语在该文件中出现的频率，通过对词数归一化来防止它偏向内容长的文档。针对某一特定文档d_j中的词语t_i，其 TF 的计算公式如下：

$$TF_{i,j}=\frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

其中，$n_{i,j}$是词语t_i在文件d_j中出现的次数，而分母则是在文件d_j中所有词语出现的次数之和。

逆文档频率(Inverse Document Frequency，IDF)是一个词语普遍重要性的度量。某一特定词语t_i的 IDF 的计算公式如下：

$$IDF_i=\log\frac{|D|}{1+|j:t_i\in d_j|} \tag{2}$$

其中，$|D|$是语料库中的文件总数，$|\{j:t_i\in d_j\}|$表示语料库中包含词语t_i的文件数目。

文档d_j中的单词t_i的权重计算公式如下：

$$TF\text{-}IDF_{i,j}=TF_{i,j}\times IDF_i \tag{3}$$

余弦相似度常用来度量两个向量之间的相似程度。本文使用余弦相似度来计算向量空间中两个查询语句的相似度，计算公式如下：

$$QuerySim(\boldsymbol{T},\boldsymbol{Q})=\frac{\sum_{i=1}^{n}(T_i\times Q_i)}{\sqrt{\sum_{i=1}^{n}T_i^2}\sqrt{\sum_{i=1}^{n}Q_i^2}} \tag{4}$$

其中，$\boldsymbol{T}$表示历史查询中的一条查询语句的向量，$\boldsymbol{Q}$代表当前查询语句的向量。

本文利用余弦相似度来计算两条查询语句使用 TF-IDF 转换后向量的相似性。为了避免在情境相似计算时，当前的编程情境需要与代码库中所有查询计算情境相似度得分，本文从代码库中选取相似的 50 条查询，并记录它们与开发者查询语句的相似度得分 $QuerySim$，从而大大降低时间消耗。

3.3 API 相关性重排序

3.3.1 计算情境以及方法名的相似度得分

对开发者当前客户端的情境信息根据 3.1.2 节进行编码，将其与检索到的 top-k 个相似历史查询的情境信息进行相似度计算。同时计算当前方法名与检索到的 top-k 个相似历史查询的方法名的相似度。本文使用莱文斯坦(Levensh-

tein)类编辑距离来计算两个数组或字符串 $a=a_1a_2a_3\cdots a_m$ 和 $b=b_1b_2b_3\cdots b_n$ 的距离，计算公式如下：

$$Lev_{a,b}(i,j)=\begin{cases}\max(i,j), & \min(i,j)=0\\ \min\begin{cases}Lev_{a,b}(i-1,j)+1\\ Lev_{a,b}(i,j-1)+1\\ Lev_{a,b}(i-1,j-1)+2_{(a_i\neq b_j)}\end{cases}, & \min(i,j)\neq 0\end{cases} \tag{5}$$

其中，m,n 分别为字符串或者数组 a,b 的长度，$Lev_{a,b}(i,j)$ 对应字符串或者数组 a,b 位置为 i,j 的类编辑距离。

计算莱文斯坦比来度量两个数组或字符串的相似度，计算公式如下：

$$LevRatio=\frac{m+n-Lev_{a,b}(m,n)}{m+n} \tag{6}$$

本文利用莱文斯坦比分别计算当前编程情境与检索到的 top-k 个相似查询的情境的相似度以及当前方法名与检索到的 top-k 个相似查询的方法名的相似度，并记录他们的相似度得分 *ContextSim* 和 *MethodnameSim*。

3.3.2　开发者 API 使用偏好

Yang 等[19]提出使用文本分析、Web 数据分析以及代码分析技术来分析开发者在 Github 上的数据以构建开发者画像，并利用开发者画像对基准工具推荐列表重排序以提高代码段推荐的准确性。然而，针对 Yang 等提取的开发者年龄、兴趣、变量命名风格等信息，本文在提取情境信息时已经忽略了开发者使用的变量名，因为这类信息对于 API 推荐的影响很小。因此，本文仅针对开发者所开发的项目中的 API 所在包的使用次数进行统计，并按照使用频率由低到高进行排序。特别地，开发者通常会频繁地调用 JDK 中的 API，JDK 中包的使用频率通常都比较高，排名比较靠后。但 JDK 里的 API 属于 Java 基础编程，当开发者编程到达一定阶段时，往往会使用其他 Jar 包中的 API 代替使用 JDK 中的 API。例如，一个成熟的开发者为了高效开发，通常会使用 apache 提供的 org.apache.commons.io.IOUtil.copy 来实现文本复制操作，而不是利用 JDK 中的 API。因此，本文将 JDK 中的包统一以 JDK 命名，并把它放在开发者 API 使用偏好的第一个位置，同时本文也删除了统计出的使用次数为 1 的包。

本文使用 $P=P_1,P_2,P_3,\cdots,P_q$ 表示开发者 API 使用偏好，P_i 代表开发者使用频率最低的第 i 个包名。当开发者使用过 JDK 中的 API，则 P_1 代表 JDK，否则 P_1 代表开发者使用频率最低的包名。当开发者输入查询语句检索 top-k 相似的历史查询后，开发者会更偏向于选择自己频繁使用的 API 来完成编程任务。开发者对检索 top-k 相似的历史查询的 API 偏好得分的计算公式如下：

$$PortraitSim(P,H)=\frac{\sum_{j=1}^{len(H)}\{i\mid H_j\in P_i\}}{q\times len(H)} \tag{7}$$

其中，H 表示检索 top-k 相似的历史查询的某一查询的 API 调用序列；H_j 表示 API 调用序列中的第 j 个 API；$len(H)$ 表示 API 调用序列 H 的长度；$\{i\mid H_j\in P_i\}$ 表示 API 调用序列中的第 j 个 API，如果属于第 i 个开发者偏好使用的包，其得分值为 i。

计算开发者 API 使用偏好对检索 top-k 个相似历史查询的 API 列表的得分，记为 *PortraitSim*。

3.3.3　相似度加权并排序 API

对 3.2 节、3.3.1 节、3.3.2 节计算得到的查询语句相似度得分、方法名相似度得分、情境相似度得分以及开发者 API 使用偏好得分进行加权，然后将检索到的 top-k 个相似历史查询进行排序，推荐出 API 列表给开发人员。加权相似度的计算公式如下：

$$sim_{\text{total}}=\alpha*QuerySim+\beta*MethodnameSim+\gamma*ContextSim+\delta*PortraitSim \tag{8}$$

$$\gamma+\delta=s \tag{9}$$

$$\gamma=0.5*c_{\text{ratio}} \tag{10}$$

其中，$\alpha+\beta+s=1$，c_{ratio} 表示情境比例。直觉上，当一个开发者在编写一个编程任务时，写的代码越多，情境信息对 API 推荐任务的约束就会增强，而开发者的 API 个性化因素对 API 推荐任务的影响就会减小。因此本文引入情境因子 c_{ratio} 来动态调节情境相似度得分与开发者 API 使用偏好得分的权重。本文中 α 取 0.3，β 取 0.1，s 取 0.6。

4　实验评估

4.1　实验设置

为了研究所提方法的有效性，模拟开发人员编写编程任务的不同阶段来测试本文提出的方法的 API 推荐效果。本文的实验平台配备了 32 核 2.6 GHz 处理器以及 128.0 GB RAM 的工作站，搭载 Windows 10 操作系统。

本文从 Github 选取拥有至少 5 年 Java 开发经验的开发者，并分别从每个人提取的数据集中随机选取 20 条查询，共 100 条查询作为测试数据。为了体现个性化推荐，针对每个开发者的数据集中出现的具有明确指向使用某个 API，某个 Jar 包或者某种技术的查询语句，采用人工的方式在不改变编程目的的同时进行删除，然后从中随机选取。例如，当开发者使用的查询语句为“parser a json file by jackson”时，在实验阶段将其更改为“parser a json file”。本文模拟开发过程的不同编程阶段以验证本文提出的方法是否适用于现实场景[20]。为此，删除程序的某些部分代码来模拟真实场景。特别地，本文分别以程序长度的 0%，20%，40%，60% 和 80% 作为上下文(以代码行作为单位)。程序其余部分中使用的 API 被用来作为有效的答案 $GT(q,c)$，其中 q 是查询语句，c 是客户端代码中的上下文信息。

4.2　评估指标

本文使用 $REC(q,c)$ 表示所提方法针对查询语句 q 和客户端代码的情境信息 c 推荐出的 API 列表，$GT(q,c)$ 作为有效的 API。本文采用 Hit，NDCG，MAP，MRR 4 个指标来衡量该方法推荐 API 的性能。特别地，MRR 和 MAP 是信息检索[21]中标准的评估指标，而 Hit 和 NDCG 经常用于评估推荐[22]。针对给定的推荐列表，开发人员通常仅对 top-N 推荐实体感兴趣。因此，计算 Hit，NDCG，MAP，MRR 时使用 N 作为截断值。通常，将 N 设置为 5 或 10。据此，本文使用 $REC_N(q,c)$ 表示 top-N 推荐实体集合，使用 $match_N(q,c)=GT(q,c)\cap REC_N(q,c)$ 表示 top-N 推荐列

表与有效 API 匹配的实体集合。

本文用 R 代表包含查询语句以及情境信息(q,c)的集合，Hit 衡量 top-N 推荐列表中至少有一个 API 是有效的，即出现在 $GT(q,c)$中的比率。

$$Hit@N=\frac{\sharp(|match_N(q,c)|>0)}{|R|}\times 100\% \tag{11}$$

其中，$\sharp(\varphi)$表示 φ 为真的次数，$|R|$表示集合 R 的大小，$(q,c)\in R$。

归一化折损累计增益(NDCG)根据结果所在位置计算每个结果的增益来衡量排名的质量[22]。NDCG 计算公式如下：

$$NCG@N=\sum_{i=1}^{N}\frac{2^{rel(i)}-1}{\log_2(i+1)} \tag{12}$$

$$NDCG@N=\frac{DCG@N}{idealDCG@N} \tag{13}$$

其中，i 代表出现的位置；$rel(i)$表示位置 i 的 API 是否正确，若位置 i 的 API 是正确的则取值为 1，否则为 0。

平均准确率(MAP)衡量查询可能有多个正确答案时排名的质量[23-24]。MAP 定义为所有查询的平均准确率的平均值，通过下式计算：

$$MAP@N=\frac{1}{|R|}\sum_{(q,c)\in R}\frac{\sum_{i=1}^{N}(P(i)\times rel(i))}{|match_N(q,c)|} \tag{14}$$

其中，$P(i)=\frac{match_i(q,c)}{i}$，平均准确率通过给定的位置 i 截断。

平均倒数排名(MRR)是另一种广泛使用的衡量排序质量的指标[23-24]。MRR 是所有查询的倒数排名的平均值。单个查询的倒数排名是推荐列表中第一个正确答案位置的倒数。因此，MRR 的计算公式如下：

$$MRR@N=\frac{1}{|R|}\sum_{(q,c)\in R}\frac{1}{N_{Rank}(q,c)} \tag{15}$$

其中，$N_{Rank}(q,c)$表示针对(q,c)推荐列表的第一个正确答案的位置。

4.3 实验结果

为了验证本文提出的方法在 API 推荐上的有效性，本文对比了该方法和基准研究在 API 推荐上的效果。本文选取的基准方法是直接利用 TF-IDF 检索相似的历史查询，并根据相似度得分对 API 进行排序。通过加入情境因素、开发者 API 使用偏好对 API 重排序可以得到更好的效果，能验证情境信息以及开发者 API 使用偏好对 API 推荐的重要作用。

表 3 列出了本文方法和基准方法在编程的不同阶段对于 API 推荐的 MRR，MAP，Hit，NDCG 结果。实验结果表明，本文方法在编程的不同阶段的推荐效果均大幅超过基准方法。值得注意的是，当情境信息为 0 时，通过利用开发者 API 使用偏好信息和方法名信息，能推荐出更受开发者青睐的 API。本文方法的平均 $Hit@5$ 为 0.674，比基准方法的 0.380 提高了 77.37%，同时本文方法的平均 $MRR@10$ 为 0.592，比基准方法的 0.295 提高了 100.81%。由此可见，本文提出的方法能把更受开发者青睐的 API 推荐在列表靠前的位置。

表 3 本文方法和基准方法对于 API 推荐的 MRR，MAP，Hit，NDCG 结果

Table 3 MRR, MAP, Hit, NDCG for API recommendation of the proposed approach and baseline method

情境信息比例	评估指标	Top-5			Top-10		
		本文方法	基准方法	相对提升值/%	本文方法	基准方法	相对提升值/%
0%	MRR	0.545	0.282	93.26	0.551	0.295	86.78
	MAP	0.457	0.235	94.47	0.518	0.269	92.57
	Hit	0.670	0.380	76.32	0.720	0.480	50.00
	NDCG	0.582	0.312	86.54	0.603	0.351	71.79
20%	MRR	0.570	0.282	102.13	0.574	0.295	94.58
	MAP	0.484	0.235	105.96	0.541	0.269	101.12
	Hit	0.660	0.380	73.68	0.690	0.480	43.75
	NDCG	0.597	0.312	91.35	0.610	0.351	73.79
40%	MRR	0.592	0.282	109.93	0.597	0.295	102.37
	MAP	0.495	0.235	110.64	0.555	0.269	106.32
	Hit	0.670	0.380	76.32	0.710	0.480	47.92
	NDCG	0.615	0.312	97.12	0.630	0.351	79.49
60%	MRR	0.614	0.282	117.73	0.617	0.295	109.15
	MAP	0.518	0.235	120.43	0.573	0.269	113.01
	Hit	0.680	0.380	78.95	0.700	0.480	45.83
	NDCG	0.635	0.312	103.53	0.644	0.351	83.48
80%	MRR	0.621	0.282	120.21	0.623	0.295	111.19
	MAP	0.531	0.235	125.96	0.585	0.269	117.47
	Hit	0.690	0.380	81.58	0.710	0.480	47.92
	NDCG	0.643	0.312	106.09	0.653	0.351	86.04
Average	MRR	0.588	0.282	108.65	0.592	0.295	100.81
	MAP	0.497	0.235	111.49	0.554	0.269	106.10
	Hit	0.674	0.380	77.37	0.706	0.480	47.08
	NDCG	0.614	0.312	96.92	0.628	0.351	78.92

4.4 有效性威胁

本方法的威胁在于实验中数据的有效性。本文采用类似其他工作的方法，提取 Java 方法注释的第一句话作为查询语句[25]。然而，注释中的其他语句也可能包含丰富的信息。此

外，尽管本文使用预定义的规则和模板过滤了低质量的查询语句，但是查询语句仍然包含噪声。将来会对数据集做进一步处理，使查询语句包含更丰富的语义信息，同时减少噪声。

本文针对的是 Java 相关的 API 推荐工作，没有拓展到其他编程语言。然而这并不是方法本身的问题，仅仅是方法实现上的限制。由于其他编程语言也包含注释以及代码信息，通过采用与本文提取数据类似的手段，本文方法也能够应用到其他编程语言中。

结束语　本文提出了基于情境感知的 API 个性化推荐方法。该方法首次利用情境信息以及用户画像为开发者提供 API 精确推荐。在编程不同阶段，本文提出的方法的 MRR，MAP，Hit，NDCG 结果明显优于基准方法。未来将会研究影响代码推荐任务的其他情境因素以及用户画像属性，并把本文方法应用到其他编程语言代码推荐任务中，同时开发一个集成环境插件协助开发人员编程。

参 考 文 献

[1] RAHMAN M M，ROY C K，LO D. Rack：Automatic api recommendation using crowdsourced knowledge[C]// Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis，Evolution，and Reengineering (SANER). IEEE，2016：349-359.

[2] LV F，ZHANG H，LOU J G，et al. Codehow：Effective code search based on api understanding and extended boolean model (e)[C]// Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE，2015：260-270.

[3] THUNG F，WANG S，LO D，et al. Automatic recommendation of API methods from feature requests[C]// Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE，2013：290-300.

[4] CAI L，WANG H，HUANG Q，et al. BIKER：a tool for Bi-information source based API method recommendation[C]// Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019：1075-1079.

[5] AI L，HUANG Z，LI W，et al. Sensory：Leveraging code statement sequence information for code snippets recommendation [C]// Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC). IEEE，2019：27-36.

[6] CHAN W K，CHENG H，LO D. Searching connected API subgraph via text phrases[C]// Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. 2012：1-11.

[7] NGUYEN A T，NGUYEN T N. Graph-based statistical language model for code[C]// Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE，2015：858-868.

[8] HAIDUC S，BAVOTA G，MARCUS A，et al. Automatic query reformulations for text retrieval in software engineering[C]// Proceedings of the 2013 35th International Conference on Software Engineering (ICSE). IEEE，2013：842-851.

[9] MCMILLAN C，GRECHANIK M，POSHYVANYK D，et al. Exemplar：A source code search engine for finding highly relevant applications[J]. Proceedings of the IEEE Transactions on Software Engineering，2012，38(5)：1069-1087.

[10] BAJRACHARYA S K，OSSHER J，LOPES C V. Leveraging usage similarity for effective retrieval of examples in code repositories[C]// Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2010：157-166.

[11] YE X，BUNESCU R，LIU C. Learning to rank relevant files for bug reports using domain knowledge[C]// Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014：689-699.

[12] PONZANELLI L，BAVOTA G，DI PENTA M，et al. Mining Stack Overflow to turn the IDE into a self-confident programming prompter[C]// Proceedings of the 11th Working Conference on Mining Software Repositories. 2014：102-111.

[13] PONZANELLI L，SCALABRINO S，BAVOTA G，et al. Supporting software developers with a holistic recommender system [C]// Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE，2017：94-105.

[14] TREUDE C，ROBILLARD M P. Augmenting api documentation with insights from stack overflow[C]// Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE，2016：392-403.

[15] ZHANG J，JIANG H，REN Z，et al. Recommending apis for api related questions in stack overflow[C]// Proceedings of the 2018 IEEE Access. 2018，6：6205-6219.

[16] ZHOU Y，YANG X，CHEN T，et al. Boosting API Recommendation with Implicit Feedback[C]// Proceedings of the IEEE Transactions on Software Engineering. 2021.

[17] BAEZA-YATES R，RIBEIRO-NETO B. Modern information retrieval[C]// Proceedings of the ACM Press / Addison-Wesley. 1999：463.

[18] VILLARROEL L，BAVOTA G，RUSSO B，et al. Release planning of mobile apps based on user reviews[C]// Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE，2016：14-24.

[19] YANG W，PAN M，ZHOU Y，et al. Developer Portraying：A Quick Approach to Understanding Developers on OSS Platforms [J]. Proceedings of the Inf. Softw. Technol.，2020，125：106336.

[20] NGUYEN P T，DI ROCCO J，DI RUSCIO D，et al. Focus：A recommender system for mining api function calls and usage patterns[C]// Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE，2019：1050-1060.

[21] LAM A N，NGUYEN A T，NGUYEN H A，et al. Combining deep learning with information retrieval to localize buggy files

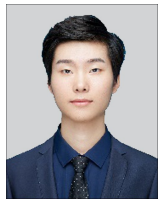
for bug reports (n)[C]// Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2015: 476-481.

[22] AVAZPOUR I, PITAKRAT T, GRUNSKE L, et al. Dimensions and metrics for evaluating recommendation systems[C]// Proceedings of the Recommendation Systems in Software Engineering. Springer, 2014: 245-273.

[23] YE X, SHEN H, MA X, et al. From word embeddings to document similarities for improved information retrieval in software engineering[C]// Proceedings of the 38th International Conference on Software Engineering. 2016: 404-415.

[24] ZHOU J, ZHANG H, LO D. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports[C]// Proceedings of the 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 14-24.

[25] GU X, ZHANG H, ZHANG D, et al. Deep API learning [C]// Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016: 631-642.



CHEN Chen, born in 1995, postgraduate. His main research interests include software evolution analysis, artificial intelligence, and mining software repositories.

ZHOU Yu, born in 1981, Ph.D, professor. His main research interests mainly include software evolution analysis, mining software repositories, software architecture, and reliability analysis.