计算机科学
COMPUTER SCIENCE

基于EBRCG的API结构模式信息增强方法研究

钟林辉，祝艳霞，黄琪轩，屈乔乔，夏子豪，郑燚

引用本文

钟林辉，祝艳霞，黄琪轩，屈乔乔，夏子豪，郑燚. 基于EBRCG的API结构模式信息增强方法研究[J]. 计算机科学，2024，51(11A)：230900121-10.

ZHONG Linhui, ZHU Yanxia, HUANG Qixuan, QU Qiaoqiao, XIA Zihao, ZHENG Yi. Study on Information Enhancement Method of API Structural Pattern Based on EBRCG [J]. Computer Science, 2024, 51(11A): 230900121-10.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230900121

基于 EBRCG 的 API 结构模式信息增强方法研究

钟林辉[1]　祝艳霞[1,2]　黄琪轩[1]　屈乔乔[1]　夏子豪[1]　郑　燚[1]
1 江西师范大学计算机信息工程学院　南昌 330022
2 桐城师范高等专科学校　安徽 桐城 231400
(shiningto@jxnu.edu.cn)

摘　要　针对 API 调用模式缺乏结构信息及结果高冗余等问题,提出了基于扩展的分支保留调用图(the Extended Branch-Reserving Call Graph,EBRCG)的 API 结构模式信息增强方法。以 Java 开源项目源代码为研究对象,使用 EBRCG 来表示 Java 类的方法的结构信息,在 EBRCG 中,同时考虑了 API 调用语句、分支语句(将 if 语句和所有循环语句视为分支语句)、switch-case 多分支语句、异常语句等,并提出了 EBRCG 裁剪算法来获取特定 API 调用模式的代码结构。同时,采用聚类和排序的方法对 API 调用模式的多个代码结构信息进行筛选,最终选择具有代表性的 API 调用模式的代码结构。为验证该方法的效果,将该方法与 TextRank 方法进行了 3 组实验比较。结果显示,该方法能有效地获取 API 调用模式的代码结构,相比 TextRank 方法能更准确地描述 API 的使用,有一定的研究意义,并为软件开发人员提供了参考。

关键词:API 调用模式;扩展的分支保留调用图;代码结构;K-Means 聚类
中图分类号　TP311

Study on Information Enhancement Method of API Structural Pattern Based on EBRCG

ZHONG Linhui[1],ZHU Yanxia[1,2],HUANG Qixuan[1],QU Qiaoqiao[1],XIA Zihao[1] and ZHENG Yi[1]
1 School of Computer Information Engineering,Jiangxi Normal University,Nanchang 330022,China
2 Tongcheng Teachers College,Tongcheng,Anhui 231400,China

Abstract　A method for enhancing API structural pattern information is proposed in response to issues such as lack of structural information and high redundancy in API call modes. The method is based on the extended branch-reserving call graph(EBRCG), which is used to represent method structural information in Java open source project source code. In the EBRCG,API call statements,branch statements(which treat if statements and all loop statements as branch statements),switch-case multi-branch statements,and exception statements are considered. The EBRCG pruning algorithm is proposed to obtain code structures for specific API call modes. Additionally,clustering and sorting methods are used to filter multiple code structure information for API call modes,and representative API call mode code structures are selected. To validate the effectiveness of this method,three sets of experiments are compared with the TextRank method. The results show that the proposed method can effectively obtain code structures for API call modes,more accurately describing API usage than the TextRank method. This method has certain research significance and provides a reference for software developers.

Keywords　API invocation pattern,Extended branch-reserving call graph,Code structure,K-Means clustering

1　引言

应用程序接口(Application Programming Interface,API)是一组可访问的软件库接口,用于支持应用程序之间的通信。通过 API,应用程序可以调用和使用其他程序的功能,而不需要了解其他程序的具体实现细节。这有助于开发人员实现代码复用,避免重复工作,进一步提高软件开发效率[1]。同时,API 还为解决软件危机、提高软件开发质量和效率提供了现实可行的途径[2]。

目前,API 的开发、使用和发展仍然面临着许多挑战[3],诸如文档质量不高,因为 API 文档中存在的一些不准确、难以理解或缺乏详细说明的问题[4-5]使得开发人员无法正确地理解和使用 API,从而导致开发错误或低效率;API 调用顺序不易确定,如在使用多个 API 进行开发时,调用它们的顺序可能并不明确或容易混淆,导致程序出错或产生意外的结果;调用约束不完备,API 的调用约束可能存在缺陷或未完善的情况,导致开发人员在使用 API 时犯错或忽略安全性问题,带来潜在的风险。这些问题导致使用 API 可能会危及应用

基金项目:国家自然科学基金(62062039,61966017);江西省自然科学基金(20212BAB202017,20224BAB202013,20212BAB202018),校教改课题(JXSDJG2044)
This work was supported by the National Natural Science Foundation of China(62062039,61966017),Jiangxi Province Natural Science Foundation(20212BAB202017,20224BAB202013,20212BAB202018) and School Education Reform Project(JXSDJG2044).
通信作者:郑燚(49177152@qq.con)

程序和用户隐私数据的安全，造成潜在风险[6-9]。

为了解决上述问题，研究者提出了各种挖掘 API 调用模式的方法[10]，其中基于聚类[11-13]或预测建模[14]的方法被广泛采用。但是这些技术获得的 API 调用模式仍然存在缺乏结构信息[15]及结果高度冗余[14]等问题。因此，本文在 BRCG[16]的基础上提出了扩展的分支保留调用图 EBRCG，并在 EBRCG 中考虑了分支语句(将 if 语句和所有循环语句视为分支语句)、API 调用语句、case 多分支语句、try-catch-finally 语句以及顺序关系、分支关系、多分支关系、异常结构关系等，从源代码中生成 API 调用模式的结构信息。同时，对于一个 API 调用模式生成的代码结构有多个的情况，为了从中筛选出具有代表性的 API 调用模式的代码结构，本文引入了代码的行数、分支语句的个数、结点层数加权、最大宽度、深度等属性信息，构造了五维属性向量。再采用 K-Means 聚类算法对这些代码结构进行分组聚类，并从每个分组中选取一个代表进行排序，筛选出具有代表性的代码结构。

本文第 2 章介绍了相关工作；第 3 章介绍了基于 EBRCG 来生成 API 调用模式的代码结构；第 4 章介绍了用 K-Means 聚类算法来选出具有代表性的 API 调用模式的代码结构；第 5 章介绍了实验设计与结果分析；最后总结全文。

2 相关工作

API 信息增强指通过对 API 的相关信息进行补充和丰富，来提供更多有用的功能和数据，从而增强 API 的能力和灵活性。它旨在为开发人员提供关于 API 的更多信息，以便更好地理解和使用 API。

有很多研究任务都集中在 API 文档上，Treude 等[17]尝试在 Stack Overflow 中使用富有洞察力的句子来增强 API 文档。Wu 等[18]提出 CoDocent 使用完全合格的 API 来跟踪相关的 API 文档，以帮助开发人员理解代码。Hoffman 等[19]通过具有预期输出的可执行测试用例来增强 API 文档。Stylos 等[20]开发一个工具 Jadeite 来帮助开发人员发现和实例化正确的 API。Subramanian 等[21]识别代码片段中的 API 元素并将其链接到 API 文档。Chen 等[22]提出了一个将众包常见问题解答集成到 API 文档中的原型。除了以上工作，有些研究者还研究了 API 相关的文档增强以及使用场景。如 Kim 等[23]提出了一种名为 eXoaDocs 的开创性 API 文档增强方法，用代码样本自动增强 API 文档。Mover 等[24]提出了一种从应用程序语料库中挖掘框架使用图的方法，开发了一种新的 groum 挖掘算法，用于描述多个交互对象类型的方法之间的控制流和数据依赖关系，从程序语料库中学习面向对象框架的使用模式。Zhang 等[25]则为了应对 eXoaDocs 方法的质量挑战和映射挑战，提出了 ADECK，与 eXoaDocs 不同，ADECK 提取使用场景，从问答网站中的最佳答案中提取相应的代码示例，形成〈使用场景，代码示例〉元组；接下来，对与每个 API 链接的元组进行聚类，并根据其大小对结果聚类进行排序；最后在排名靠前的集群中获得最高用户分数的元组被嵌入到基于预定义模板的 API 文档中。另外，也有其他研究针对信息增强进行了研究，例如文献[26]利用知识图谱进行信息增强，从而改进协同过滤算法的性能。

3 基于 EBRCG 的 API 调用模式的代码结构生成

3.1 相关定义

定义 1(EBRCG) 在 EBRCG 中，结点是一条 API 调用语句、一条分支语句或分支语句中的一个分支、一条多分支语句或多分支语句中的一个分支、异常语句中的 try 语句、catch 语句、finally 语句。结点之间有顺序关系(Squential)、分支关系(Branching)、多分支关系(Multi_Branching)和异常结构关系(Try_Catch)。

本文使用五元组 $EBRCG=\langle N,S,B,M,T\rangle$ 来对 EBRCG 进行定义，其中：

1) N 是 API 调用语句、分支语句和分支语句中的分支、多分支语句和多分支语句中的分支、异常语句中的 try 语句、catch 语句、finally 语句的集合。

2) S 是顺序关系的集合，其中 $\forall\langle n_1,n_2\rangle\in S,n_1\in N$ 且 $n_2\in N$。

3) B 是分支关系的集合，其中 $\forall\langle n_1,n_2\rangle\in B,n_1\in N$ 且 $n_2\in N$。

4) M 是多分支关系的集合，其中 $\forall\langle n_1,n_2,\cdots n_n\rangle\in M$，$n_1\in N,n_2\in N$ 且 $n_n\in N$。

5) T 是异常结构关系的集合，其中 $\forall\langle n_1,n_2,n_3\rangle\in T$，$n_1\in N,n_2\in N$ 且 $n_3\in N$。

6) $\forall\langle n_1,n_2\rangle\in S$ 且 $\forall n_3\in N,\langle n_1,n_3\rangle\notin B$，对于 $\forall\langle n_1,n_2\rangle\in B$ 且 $\forall n_3\in N,\langle n_1,n_3\rangle\notin S$。

7) $\forall\langle n_1,n_2,\cdots n_n\rangle\in S$ 且 $\forall n_{n+1}\in N,\langle n_1,n_2,\cdots n_{n+1}\rangle\notin M$，对于 $\forall\langle n_1,n_2,\cdots n_n\rangle\in M$ 且 $\forall n_{n+1}\in N,\langle n_1,n_2,\cdots n_{n+1}\rangle\notin S$。

8) $\forall\langle n_1,n_2,n_3\rangle\in S$ 且 $\forall n_4\in N,\langle n_1,n_2,n_4\rangle\notin T$，对于 $\forall\langle n_1,n_2,n_3\rangle\in T$ 且 $\forall n_4\in N,\langle n_1,n_2,n_4\rangle\notin S$。

9) $\forall\langle n_1,n_2\rangle\in B$ 且 $\forall n_3\in N,\langle n_1,n_3\rangle\notin M$，对于 $\forall\langle n_1,n_2,n_3\rangle\in M$ 且 $\forall n_4\in N,\langle n_1,n_2,n_4\rangle\notin B$。

10) $\forall\langle n_1,n_2,n_3\rangle\in M$ 且 $\forall n_4\in N,\langle n_1,n_2,n_4\rangle\notin T$，对于 $\forall\langle n_1,n_2,n_3\rangle\in T$ 且 $\forall n_4\in N,\langle n_1,n_2,n_4\rangle\notin M$。

条件 1)－5)说明 EBRCG 具有顺序关系、分支关系、多分支关系和异常结构关系等 4 种关系，条件 6)－10)说明 EBRCG 中的结点仅通过一种关系连接到它的子结点。

定义 2(API 调用模式的代码结构) 实现特定功能的某个 API 调用模式及相关结构信息(顺序关系、分支关系、多分支关系或异常结构关系等)组成的代码结构上下文。

3.2 EBRCG 生成

本小节给定了 EBRCG 的生成算法，对于给定 Java 源代码类的方法，本文根据 3.1 节中对 EBRCG 的定义来遍历 Java 类的方法中的程序语句，生成相应的 EBRCG，生成算法如算法 1 所示。

算法 1 扩展的分支保留调用图 EBRCG 生成算法

输入：Java 类的方法

输出：类的方法的 EBRCG

1. step1：创建一个以输入的 Java 类的方法名为根结点的图 G
2. step2：for t do //t 表示方法体中的任意语句
3. if(t 是一条 API 调用语句)then
4. 创建标记为该 API 调用签名的结点 f_n，并插入图 G 中

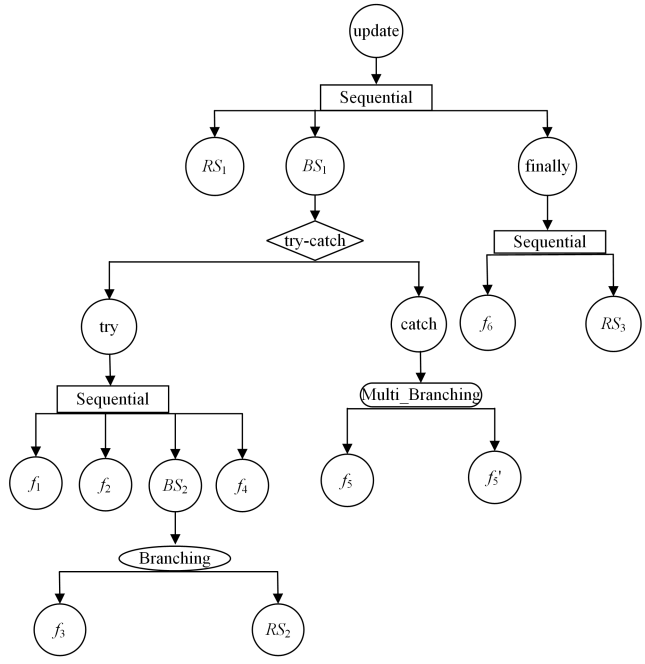
5. 在根结点和 f_n 之间建立顺序关系
6. if(t 是一条分支语句)then
7. 创建标记为“Branching”的结点 BS_n，并插入图 G 中
8. 在根结点和 BS_n 结点之间建立顺序关系
9. for t 中的每个分支 do
10. 创建标记为该分支语句签名的结点，并将其插入图 G 中
11. 将 BS_n 和分支结点之间的分支关系插入图 G
12. if(t 是 Switch-case 多分支语句)then
13. 创建标记为"Multi_Branching"的结点 BS_n
14. 在根结点和 BS_n 结点之间建立顺序关系
15. for 每个 case 分支 do
16. 创建标记为该 case 分支语句签名的结点 B_n，并插入图 G
17. 将 BS_n 结点和 B_n 结点之间的多分支关系插入图 G
18. if(t 是 try-catch-finally 异常语句)then
19. 创建标记为"try_catch"的结点 BS_n 以及结点 finally
20. 在根结点和 BS_n 结点之间建立顺序关系
21. 在根结点和 finally 结点之间建立顺序关系
22. for 异常语句中的 try 和 catch 语句 do
23. 创建标记为 try 的结点，并将其插入图 G
24. 将 BS_n 和 try 结点之间的异常关系插入图 G
25. 创建标记为 catch 的结点，并将其插入图 G
26. 将 BS_n 和 catch 结点之间的异常关系插入图 G
27. if(t 是其他语句)then
28. 创建标记为该语句的结点 RS_n
29. 在根结点和 RS_n 结点之间建立顺序关系
30. for try、catch 和 finally 中的所有 API 调用语句、分支语句、case 语句、异常语句
31. do step2 //按照 step2 处理
32. for case 语句中的所有 API 调用语句、分支语句、case 语句、异常语句
33. do step2 //按照 step2 处理
34. for 分支语句中的所有 API 调用语句、分支语句、多分支语句、异常语句
35. do step2 //按照 step2 处理
36. return G
37. end

图 1(a)给出了一段 Java 源代码类的方法，图 1(b)给出了该方法对应的 EBRCG。

```
//通用的增删改操作
public void update(String sql,Object ...args){
Connection conn = null;
PreparedStatement ps = null;
try {
                //1.获取数据库的连接
                conn = JDBCUtils.getConnection(); //f1
                //2.预编译sql语句， 返回PreparedStatement的实例
                ps = conn.prepareStatement(sql); //f2
                //3.填充占位符
                for(int i = 0;i < args.length;i++) {
                ps.setObject(i+1, args[i]); //f3
                                }
                                //4.执行
                                ps.execute(); //f4
                                }catch (SQLException e) {
                //处理 SQL 异常
                 e.printStackTrace();//f5
                   } catch (Exception e) {
                         //处理其他异常
                     e.printStackTrace();//f5'
              }finally {
                                //5.资源的关闭
                                JDBCUtils.closeResource(conn, ps); //f6


                                }
    }
}
```

(a)Java 类中的方法



(b)代码所对应的 EBRCG

图 1　Java 源代码类的方法的 EBRCG

Fig. 1　EBRCG of a method in a Java source code class

在图 1(a)中，有 3 个连续的块，即 update 方法中的 try-catch 语句块 finally 语句块以及无关项 RS1，在 try 语句块中有两个 API 调用语句 $f1$ 和 $f2$，分支语句 BS2 和 BS2 的两个分支分别为 API 调用语句 $f3$ 和 RS2，catch 语句块中有两个 API 调用语句 $f5$ 和 $f5'$，finally 语句块中有语句 $f6$ 和 RS3。上述所有信息使用 EBRCG 表示，如图 1(b)所示。

3.3　API 调用模式的代码结构生成

由于 Java 源代码类的方法的 EBRCG(G)中可能包含太多来自源代码的信息，如果需要知道其中关于某个 API 调用模式(L)的 EBRCG，则需要对类的方法的 EBRCG 进行裁剪。裁剪步骤如下：

1)首先对 G 进行遍历。

2)将 G 中与 L 中的元素签名相同的结点进行了标记，并记录下这些结点的位置信息，然后创建结点 f_n，将被标记结点的信息赋给 f_n，将 f_n 插入空图 g_n。

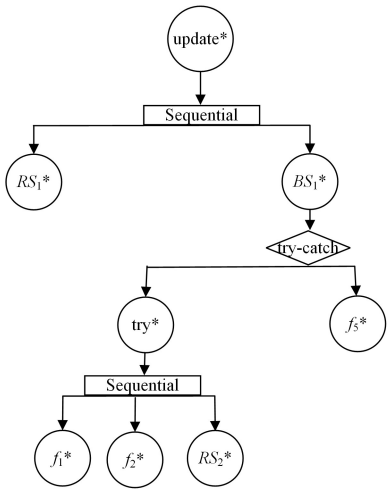
3)判断标记结点是否存在父结点、兄弟结点，若存在则标记父结点、兄弟结点并将其与父结点、兄弟结点的信息插入 g_n，然后继续按照步骤 3)对父结点进行遍历，直到父结点的父结点为空，返回生成的多个子图 g_{ns}。

4)从生成的子图 g_{ns} 中任选 i 个($i=len(L)$) g_n 进行多次组合合并，并将 g_n 之间结点签名相同的结点合并为一个结点，返回合并后的 G_n。

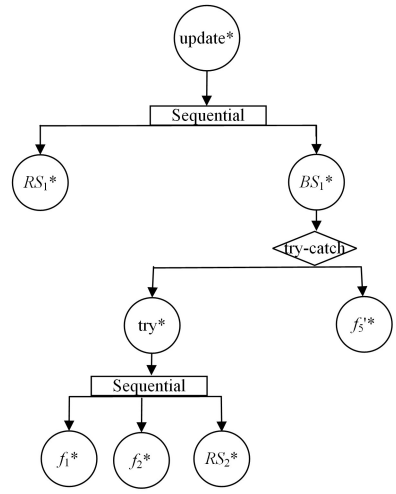
5)G_n 中 f_n 结点签名的先后次序如果与 L 中一致，且 f_n 的位置大小关系与在类的方法的 EBRCG 中一致，则 G_n 为 API 调用模式的 EBRCG，G_{ns} 为多个 API 调用模式的

EBRCG。

对于图 1(b)所示源代码的 EBRCG，若输入的 API 调用模式是 $f1, f2, f5$，按照裁剪算法步骤，裁剪后生成了两个关于 API 调用模式 $f1, f2, f5$ 的 EBRCG，如图 2 所示。



(a) f_1, f_2, f_5 对应的第一个裁剪后的 EBRCG



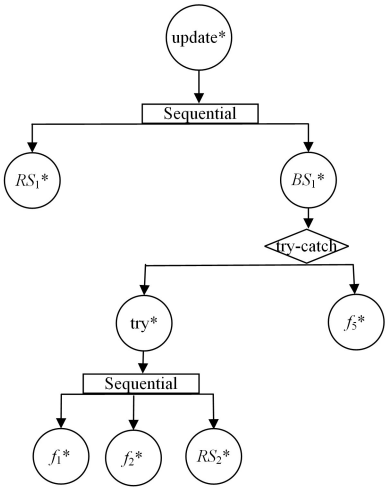
(b) f_1, f_2, f_5 对应的第二个裁剪后的 EBRCG

图 2 裁剪生成的 EBRCG

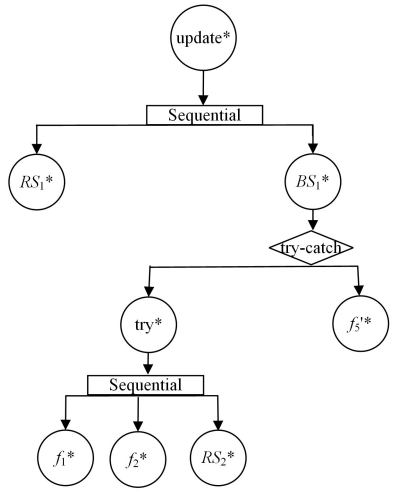
Fig. 2 Trimming the generated EBRCG

裁剪生成的 API 调用模式的 EBRCG，本文采用先序遍历的方法来遍历图中的每个结点并输出，生成 API 调用模式的代码结构。

先序遍历图 2 裁剪生成的 EBRCG，生成图 3(c)、图 3(d)所示的代码结构，即 API 调用模式 f1, f2, f5 的代码结构。在这个结构中，除了保留了 API 调用模式信息，还保留了与该调用模式相关的结构上下文信息，对于一些与结构信息无关的语句，本文定义为 RSn，在遍历生成的代码结构中使用“?”表示。



(a) f_1, f_2, f_5 对应的第一个裁剪后的 EBRCG



(b) f_1, f_2, f_5 对应的第二个裁剪后的 EBRCG

```
update( ){
?
try {
                conn = JDBCUtils.getConnection(); //f1
                ps = conn.prepareStatement(sql); //f2
                ?
                                }catch (?) {
                  e.printStackTrace();//f5
                    }
}
```

(c)图 3(a)对应的代码结构

```
update( ){
?
try {
                conn = JDBCUtils.getConnection(); //f1
                ps = conn.prepareStatement(sql); //f2
                ?
                                }catch (?) {
                  e.printStackTrace();//f5'
                    }
}
```

(d)图 3(b)对应的代码结构

图 3 API 调用模式的代码结构生成

Fig. 3 Code structure generation for API invocation mode

4 API 调用模式的代码结构筛选

为了解决生成的 API 调用模式的代码结构结果冗余的问题，本文使用 K-Means 聚类算法[27]对结果进行聚类。首先提取相关属性，生成代码结构的五维属性向量表；然后对这些代码结构分组聚类，并选出每组距离中心点最近的点作为代表点[28]；最后对这些代表点进行排序。

4.1 属性提取

本文引入了 5 个属性 $A1, A2, A3, A4, A5$ 来描述代码结构特征。其中，$A1$ 表示 API 调用模式的代码结构中有效

代码的行数；$A2$ 表示 API 调用模式的代码结构中分支语句的个数；$A3$ 表示 API 调用模式的 EBRCG 的结点层数加权；$A4$ 表示 API 调用模式的 EBRCG 中的最大宽度；$A5$ 表示 API 调用模式的 EBRCG 的深度。这些属性的计算式如下：

$A1$＝初始行数＋if-else 块中的行数＋for 循环块中的行数＋while 循环块中的行数＋do-while 循环块中的行数＋Switch-case 块中的行数＋try-catch-finally 语句块中的行数＋其他语句的行数　　(1)

其中，初始行数的值为 1，其他语句的行数指 API 方法调用语句的个数与无关项语句 RS_n 的个数之和。

$$A2=BS_1+BS_2+BS_3+\cdots+BS_n \tag{2}$$

其中，BS_n 表示第 n 个分支语句。

$$A3=1*nodes_1+2*nodes_2+3*nodes_3+\cdots+n*nodes_n \tag{3}$$

其中，$nodes_n$ 表示 API 调用模式的 EBRCG 中第 n 层结点的个数。

$$A4=MAX\_N_i(i=1,2,3,\cdots,I) \tag{4}$$

$A4$ 表示 API 调用模式的 EBRCG 的最大宽度，N_i 为第 i 层的结点数量，MAX_N_i 为所有层级中结点数量的最大值。

$$A5=\begin{cases}0, & \text{EBRCG}^*=0\\ 1+\text{maxDepth}(v)v\in EBRCG^*, & \text{其他}\end{cases} \tag{5}$$

其中，v 表示 EBRCG 中的任意结点，$Depth(v)$ 表示根结点到结点 v 的路径长度。

最后，本文使用〈$A1$，$A2$，$A3$，$A4$，$A5$〉来表示 API 调用模式的代码结构。

4.2　API 调用模式的代码结构聚类

提取属性后，使用 *distance* 作为如下定义的相似性度量来应用 K-Means 算法。

$$distance(V_1,V_2)=\sum_{i=1}^{n}|x_i-y_i| \tag{6}$$

其中，V_1 和 V_2 是 n 维特征向量，x_i 和 y_i 分别是 V_1 和 V_2 的第 i 个元素。

在本文的问题中，K 是给定 API 调用模式的代码结构的簇数。为了使结果具有代表性，需要选择合适的 K 值来进行聚类。Silhouettte Score[29] 是一种用于评估聚类效果的指标，通常用于 K-Means 算法中，某一 K 值下的 Silhouettte Score 值越高，则聚类效果越好。对于 API 调用模式（io. netty. buffer. ByteBuf. writeByte，io. netty. buffer. ByteBuf. writeBytess，io. netty. buffer. ByteBuf. writeBytes），生成的代码结构有 24 个，提取其属性，结果如表 1 所列。

表 1　属性向量表

Table 1　Attribute vectors

属性坐标
(27,11,60,6,14),(27,11,60,6,14),(7,1,21,3,4)
(5,0,3,4,2),(27,11,57,6,14),(5,0,3,3,2),(7,1,5,2,4)
(25,10,53,3,14),(27,11,59,6,14),(27,11,59,6,14)
(27,11,59,6,14),(25,10,51,3,14),(21,8,41,3,12),(9,2,7,4,4)
(25,10,52,3,14),(27,11,61,6,14),(27,11,61,6,14)
(7,1,5,2,4),(21,8,40,3,12),(17,6,29,4,10),(21,8,39,3,12)
(27,11,58,6,14),(25,10,53,3,14),(27,11,59,6,14)

使用 K-means 算法对这些数据点进行聚类，并取 k 值范围为[2，24]，不同 K 值下的 Silhouettte Score 值如图 4 所示。

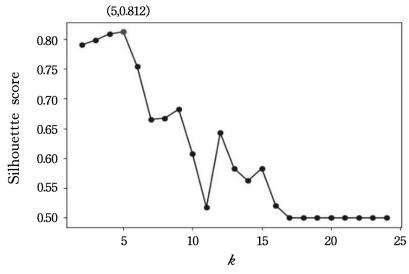


图 4　不同 K 值下的 Silhouettte Score

Fig. 4　Silhouettte Score at different K values

当 K 取 5 时，Silhouettte Score 为 0.812，此时为 Silhouettte Score 的最大值，故 K 取 5 时，聚类效果最佳。

在这里取 K＝5，再使用 K-Means 聚类算法将这些代码结构聚类到具有相似特征的组中，结果如表 2 所列。

表 2　聚类结果

Table 2　Clustering results

类别	中心点坐标	每组代表点 R_n
1	(27,11,59.3,6,14)	R_1＝(27,11,59,6,14)
2	(6.67,0.83,4.83,2.83,3.33)	R_2＝(7,1,5,2,4)
3	(21,8,40,3,12)	R_3＝(21,8,40,3,12)
4	(25,10,52.25,3,14)	R_4＝(25,10,52,3,14)
5	(17,6,29,4,10)	R_5＝(17,6,29,4,10)

由表 2 可知，K-Means 聚类后，生成了 5 类结果，同时求解出聚类后每类的中心点坐标，结合表 1 可知，类别 1，2，4 的中心点坐标不在这些数据点中，为了便于对数据进行进一步分析，本文选取距离中心点最近的点作为代表点 R_n。

4.3　API 调用模式的代码结构排序

为了从生成的 API 调用模式的代码结构中选取出最具有代表性的代码结构，本文采用了 eXoaRank 排序方法[30] 对聚类后得到的 R_n 结点进行排序。

首先计算各点之间的归一化相似性，归一化相似矩阵如下：

$$\begin{bmatrix} & R_1 & R_2 & R_3 & R_4 & R_5\\ R_1 & & 0.01 & 0.03 & 0.07 & 0.02\\ R_2 & 0.01 & & 0.02 & 0.01 & 0.02\\ R_3 & 0.03 & 0.02 & & 0.05 & 0.05\\ R_4 & 0.07 & 0.01 & 0.05 & & 0.02\\ R_5 & 0.02 & 0.02 & 0.05 & 0.02 & \end{bmatrix}$$

得到各点之间的归一化相似性后，需要设置合适的阈值来构建图模型，本文根据归一化相似性值的分布情况来确定合适的阈值。归一化相似性矩阵主对角线以下 R_1，R_2，R_3，R_4，R_5 的归一化相似性值的分布情况如图 5 所示。

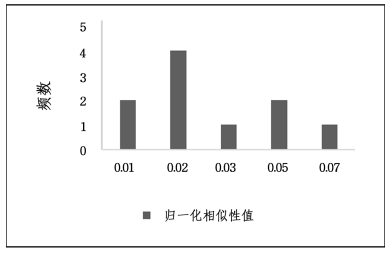


图 5　R_n 归一化相似性值分布

Fig. 5　Distribution of normalized similarity values R_n

图 5 中，横坐标表示归一化相似性值，纵坐标表示值的出现频数，通过观察可以发现，集中分布区域的值接近 0.02，因此最适合的阈值为 0.02。

确定合适的阈值后，保留矩阵中R_n结点之间归一化相似性值大于 0.02 的边，同时去掉归一化相似性值小于 0.02 的边，如下为各点间的关系矩阵。

	R_1	R_2	R_3	R_4	R_5
R_1	0	0	1	1	1
R_2	0	0	1	0	1
R_3	1	1	0	1	1
R_4	1	0	1	0	1
R_5	1	1	1	1	0

在关系矩阵中，1 表示两点之间存在边，0 表示两点之间不存在边，根据关系矩阵构建图模型，如图 6 所示。

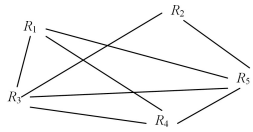


图 6 R_n关系图模型

Fig. 6 R_n relationship graph model

构建了R_n关系图模型后，开始计算各结点的中心性分值。第 0 次迭代时，各结点的初始中心性分值如下：

$centrallity_0(R_1)$＝连接R_1的边的数量之和＝3

$centrallity_0(R_2)$＝连接R_2的边的数量之和＝2

$centrallity_0(R_3)$＝连接R_3的边的数量之和＝4

$centrallity_0(R_4)$＝连接R_4的边的数量之和＝3

$centrallity_0(R_5)$＝连接R_5的边的数量之和＝4

从第 $N(N\geqslant 1)$次迭代开始，R_1，R_2，R_3，R_4，R_5之间的中心性值满足以下关系：

$$centrallity_N(R_1)=\frac{centrallity_{N-1}(R_3)}{4}+\frac{centrallity_{N-1}(R_4)+}{3}+\frac{centrallity_{N-1}(R_5)}{4} \tag{7}$$

$$centrallity_N(R_2)=\frac{centrallity_{N-1}(R_3)}{4}+\frac{centrallity_{N-1}(R_5)}{4} \tag{8}$$

$$centrallity_N(R_3)=\frac{centrallity_{N-1}(R_1)}{3}+\frac{centrallity_{N-1}(R_2)+}{2}+\frac{centrallity_{N-1}(R_4)}{3}+\frac{centrallity_{N-1}(R_5)}{4} \tag{9}$$

$$centrallity_N(R_4)=\frac{centrallity_{N-1}(R_1)}{3}+\frac{centrallity_{N-1}(R_3)+}{4}+\frac{centrallity_{N-1}(R_5)}{4} \tag{10}$$

$$centrallity_N(R_5)=\frac{centrallity_{N-1}(R_4)}{3}+\frac{centrallity_{N-1}(R_2)}{2}+\frac{centrallity_{N-1}(R_3)+}{4}+\frac{centrallity_{N-1}(R_4)}{3} \tag{11}$$

按照上述关系式进行迭代计算，直至$centrallity_N(R_1)$，$centrallity_N(R_2)$，$centrallity_N(R_3)$，$centrallity_N(R_4)$，$centrallity_N(R_5)$的大小关系收敛，最终R_1，R_2，R_3，R_4，R_5之间的排序结果为$(R_3=R_5)>(R_1=R_4)>R_2$。

5 实验与分析

实验部分主要验证以下问题：

RQ1：对于本文的研究方法与 TextRank 研究方法得到的 API 调用模式的代码结构，哪种方法下代码结构中的 API 调用模式更完整？

RQ2：对于本文的研究方法与 TextRank 研究方法得到的 API 调用模式的代码结构，哪种方法下的代码结构与源代码中对应方法体的语义更接近？

RQ3：本文方法得到的 API 调用模式的代码结构与 TextRank 方法下的代码结构与源代码中的对应方法体在结构上有何差异？

5.1 实验评估标准

为了验证本文方法的有效性，引入了精确率 P、欧几里得距离 D 以及余弦相似度 Similarity 等指标来进行分析。

精确率 P 的计算式如下：

$$P=\frac{T}{T+F}\times 100\% \tag{12}$$

其中，T 表示预测值结果与实际值结果相同的数量，F 表示预测值结果与实际值结果不相同的数量。

欧几里得距离 D 的表达式如下：

$$D=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2+\cdots+(y_{n1}-y_{n2})^2} \tag{13}$$

将两个文本的语义向量的每一维相减并求平方值，D 值越小说明两个文本的相似性越高。

余弦相似度(Similarity)的表达式如下：

$$Similarity=\cos\theta=\frac{A\cdot B}{|A||B|}=\frac{\sum_{i=1}^{n}A_i\times B_i}{\sqrt{\sum_{i=1}^{n}A_i{}^2}\times\sqrt{\sum_{i=1}^{n}B_i{}^2}} \tag{14}$$

Similarity 表示两个文本在空间中的夹角的余弦值，取值范围在－1 到 1 之间，值越接近 1 表示两个文本的相似度越高，值越接近－1，表示它们的差异越大。

5.2 实验问题验证

5.2.1 RQ1 的验证和分析

首先选取了 netty 项目的 67 个 Java 文件、spring-data-neo4j 的 105 个 Java 文件、weld 的 199 个 Java 文件，共 371 个 Java 文件作为实验测试对象。然后使用 PAM 工具[14]从这些项目中挖掘出了 API 调用模式序列，再结合本文提出的方法以及 Textrank 方法得到了这些序列的代码结构信息。

随机从项目中选取部分 API 调用模式，如 M1－M15 进行 API 调用模式完整性度量。

表 3 不同 API 调用模式

Table 3 Different API call patterns

名称	API 调用模式
M1	io. netty. buffer. ByteBuf. writeByte, io. netty. buffer. ByteBuf. writeByte, io. netty. buffer. ByteBuf. writeByte
M2	io. netty. channel. Channel. write
M3	org. springframework. data. neo4j. support. node. Neo4jHelper. cleanDb
M4	org. springframework. data. neo4j. support. Neo4jTemplate. createNode, org. springframework. data. neo4j. support. Neo4jTemplate. createNode

（续表）

名称	API 调用模式
M5	org. jboss. weld. environment. se. WeldContainer. instance
M6	org. jboss. weld. environment. se. Weld. shutdown
M7	io. netty. channel. MessageEvent. getChannel
M8	io. netty. channel. socket. nio. NioServerSocketChannelFactory. 〈init〉，io. netty. bootstrap. ServerBootstrap. 〈init〉，io. netty. bootstrap. ServerBootstrap. setPipelineFactory，io. netty. bootstrap. ServerBootstrap. bind
M9	io. netty. channel. ChannelFuture. awaitUninterruptibly
M10	org. springframework. data. neo4j. support. Neo4jTemplate. createNode
M11	org. jboss. weld. context. bound. BoundConversationContext. deactivate
M12	org. jboss. weld. context. bound. BoundRequestContext. invalidate，org. jboss. weld. context. bound. BoundRequestContext. deactivate，org. jboss. weld. context. bound. BoundRequestContext. dissociate
M13	org. jboss. weld. context. http. HttpConversationContext. activate
M14	io. netty. channel. ChannelHandlerContext. getChannel
M15	io. netty. buffer. ChannelBuffer. writeInt

两种方法下 API 调用模式的完整性比较结果如表 4 所列。

表 4　本文方法与 TextRank 方法下代码结构中 API 调用模式的完整性比较

Table 4　Comparison of completeness of code structure in API invocation mode between our method and TextRank method

API 调用模式	本文方法的 P 值	TextRank 方法的 P 值
M1	1.00	0.00
M2	1.00	0.00
M3	1.00	0.17
M4	1.00	1.00
M5	1.00	0.00
M6	1.00	0.00
M7	1.00	0.29
M8	1.00	0.17
M9	1.00	0.09
M10	1.00	0.40
M11	1.00	0.00
M12	1.00	0.00
M13	1.00	0.00
M14	1.00	0.10
M15	1.00	0.00

完整性对比结果分析如图 7 所示。

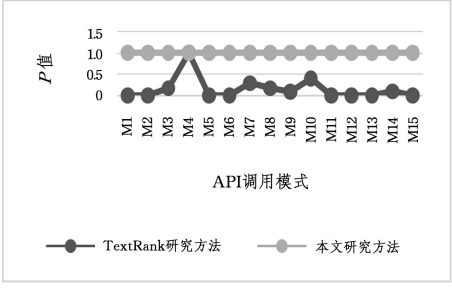


图 7　完整性对比结果分析

Fig. 7　Analysis of comparison results of completeness

由图 7 的分析结果可知，本文研究方法得到的 API 调用模式的代码结构中，API 调用模式序列的完整性为 1，而 TextRank 方法得到的代码结构所包含的 API 调用模式序列的完整性较低。

5.2.2　RQ2 的验证和分析

为了将本文方法生成的 API 调用模式的代码结构、TextRank 方法生成的代码结构分别与源代码中对应的方法体进行语义相似性比较，选取 M1－M15 等 15 种调用模式在不同方法下生成的代码结构作为测试对象，使用 TF-IDF 模型构造不同方法下代码结构的语义向量进行语义相似性比较，结果如表 5、表 6 所列。

表 5　本文方法与 Java 源代码中对应方法体的语义相似度比较

Table 5　Comparison of semantic similarity between our method and the corresponding method body in Java source code

API 调用模式	欧几里得距离	余弦相似度
M1	1.21	0.27
M2	1.30	0.15
M3	1.29	0.16
M4	1.38	0.04
M5	1.32	0.13
M6	1.29	0.16
M7	1.28	0.17
M8	1.35	0.09
M9	1.30	0.15
M10	1.39	0.04
M11	1.36	0.07
M12	1.32	0.13
M13	1.37	0.06
M14	1.33	0.11
M15	1.23	0.25

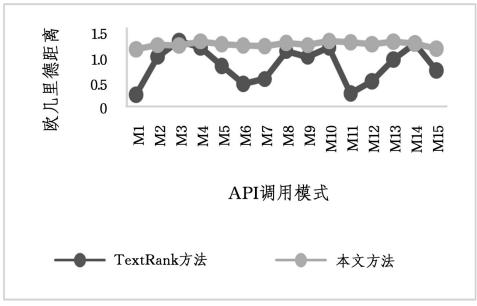
表 6　TextRank 方法与 Java 源代码中对应方法体的语义相似度比较

Table 6　Comparison of semantic similarity between TextRank method and the corresponding method body in Java source code

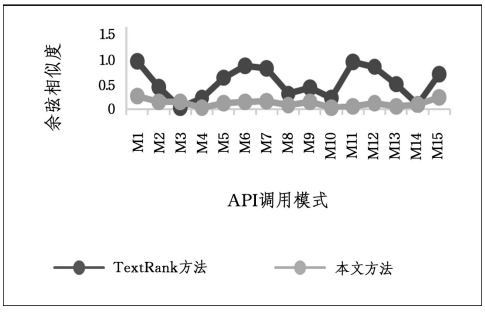
API 调用模式	欧几里得距离	余弦相似度
M1	0.25	0.97
M2	1.05	0.45
M3	1.38	0.04
M4	1.24	0.23
M5	0.85	0.64
M6	0.48	0.88
M7	0.58	0.83
M8	1.17	0.32
M9	1.05	0.44
M10	1.24	0.23
M11	0.28	0.96
M12	0.53	0.86
M13	0.99	0.51
M14	1.33	0.11
M15	0.76	0.71

对表 5、表 6 两种方法下的代码结构与源代码对应方法体的语义相似性实验结果进行分析，如图 8 所示。

由图 8 可知，在 API 调用模式相同的情况下，进行语义相似性比较的结果如下：1）在实验中，93% 的 API 调用模式在 TextRank 方法下生成的代码结构与源代码对应方法体的欧几里得距离比在本文方法下生成的代码结构与源代码对应方法体的欧几里得距离小；2）87% 的 API 调用模式在 TextRank方法下生成的代码结构与源代码对应方法体的余弦相似度比本文方法下生成的代码结构与源代码对应方法体的余弦相似度大。而欧几里得距离的值越小，文本的语义相似度越高，余弦相似度 Similarity 的值越大，文本的语义相似度越高。



（a）两种方法的欧氏距离



（b）两种方法的语义相似性

图 8 语义相似性对比结果分析

Fig. 8 Analysis of comparison results of semantic similarity

5.2.3 RQ3 的验证和分析

在这个步骤中，为了比较本文研究方法和 TextRank 方法下生成的关于 API 调用模式的代码结构与对应源代码方法体在结构上的差异，构造了一个结构字典{*F*，{*C*1，*C*2，…，*Cn*}，*E*}，F 表示方法名，默认值取 1，*Cn* 表示分支语句、多分支语句以及 API 调用语句等的数量，E 表示异常语句的数量，本文在实验中将结构字典的内容设置如下：{‘method_name’：默认值为 1，{‘condition_count’：条件语句的个数，‘switch_count’：switch-case 语句的个数，‘for_count’：for 语句的个数，‘while_count’：while 语句的个数，‘do_while_count’：do_while 语句的个数，‘api_call_count’：API 调用语句的条数}，‘exception_count’：异常语句的个数}。

选取 M1－M15 等 15 种调用模式在不同方法下生成的代码结构作为测试对象，生成不同方法下代码结构的结构字典进行比较。并使用余弦相似度值以及欧几里得距离来评价 3 者在结构上的差异，结果如表 7 所列。

表 7 本文方法与 Java 源代码中对应方法体的结构相似度比较

Table 7 Comparison of structural similarity between our method and the corresponding method body in Java source code

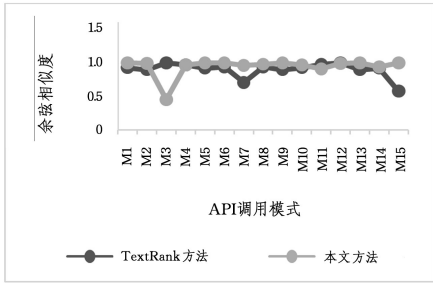
API 调用模式	欧几里得距离	余弦相似度
M1	3.16	0.99
M2	16.06	0.98
M3	2.00	0.45
M4	15.00	0.96
M5	1.00	0.99
M6	18.05	0.99
M7	64.03	0.95
M8	12.04	0.97
M9	17.14	0.99
M10	15.00	0.96
M11	20.02	0.90
M12	4.00	0.98
M13	6.32	0.99
M14	22.02	0.93
M15	1.00	0.99

表 8 TextRank 方法与 Java 源代码中对应方法体的结构相似度比较

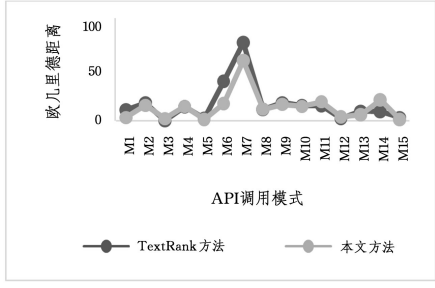
Table 8 Comparison of structural similarity between TextRank method and the corresponding method body in Java source code

API 调用模式	欧几里得距离	余弦相似度
M1	11.4	0.92
M2	19.02	0.89
M3	0.00	0.99
M4	15.00	0.96
M5	2.45	0.91
M6	42.00	0.93
M7	83.47	0.70
M8	12.00	0.93
M9	19.03	0.89
M10	16.00	0.92
M11	16.00	0.97
M12	2.00	0.99
M13	10.00	0.89
M14	9.85	0.91
M15	3.16	0.57

对表 7、表 8 的实验结果分析如图 9 所示。



（a）两种方法的语义相似性



（b）两种方法的欧氏距离

图 9 结构相似性对比结果分析

Fig. 9 Analysis of comparison results of structural similarity

由图 9 可知，在 API 调用模式相同的情况下，进行结构相似性比较的结果如下：1）在实验中，60％的 API 调用模式在本文方法下生成的代码结构与源代码对应方法体的欧几里得距离比在 TextRank 方法下生成的代码结构与源代码对应方法体的欧几里得距离小；2）60％的 API 调用模式在本文方法下生成的代码结构与源代码对应方法体的余弦相似度比在 TextRank 方法下生成的代码结构与源代码对应方法体的余弦相似度大。

通过对以上 3 组实验的结果分析可知，本文方法生成的 API 调用模式的代码结构在语义相似性上低于 TextRank 方法下生成的代码结构，但在完整性和结构相似性上优于 TextRank 方法下生成的代码结构。

结束语 本文在文献[31]的基础上，研究了 API 调用模式的结构信息，提出了一种 API 调用模式的代码结构信息增强方法，在多个代码结构的情况下采用 K-Means 聚类算法对 API 调用模式的代码结构进行筛选，提高软件开发效率。在实验中比较了本文方法与 TextRank 方法生成的代码结构的完整性、语义相似性度量和结构相似性，结果表明本文方法有效。但是后续还需要在多个方面进行改进，包括增加其他属性进行实验、增加实验对象、扩展到更多的编程语言和进行更先进的方法对比。

参 考 文 献

[1] DE ROOVER C，LÄMMEL R，PEK E. Multi-dimensional exploration of api usage[C]// 2013 21st International Conference on Program Comprehension(ICPC). IEEE，2013：152-161.

[2] LIZ，WU J Z，LI M S. Study on Key Issues in API Usage[J]. Journal of Software，2018，29(6)：1716-1738.

[3] KO A J，MYERS B A，AUNG H H. Six learning barriers in end-user programming systems[C]// 2004 IEEE Symposium on Visual Languages-Human Centric Computing. IEEE，2004：199-206.

[4] PARNAS D L，MADEY J，IGLEWSKIM. Precise documentation of well-structured programs[J]. IEEE Transactions on Software Engineering，1994，20(12)：948-976.

[5] ROBILLARD M P，DELINE R. A field study of API learning obstacles[J]. Empirical Software Engineering，2011，16：703-732.

[6] LAZAR D，CHEN H，WANG X，et al. Why does cryptographic software fail? A case study and open problems[C]// Proceedings of 5th Asia-Pacific Workshop on Systems. 2014：1-7.

[7] EGELE M，BRUMLEY D，FRATANTONIO Y，et al. An empirical study of cryptographic misuse in android applications [C]// Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. 2013：73-84.

[8] GEORGIEV M，IYENGAR S，JANA S，et al. The most dangerous code in the world：validating SSL certificates in non-browser software[C]// Proceedings of the 2012 ACM Conference on Computer and Communications Security. 2012：38-49.

[9] FAHL S，HARBACH M，PERLH，et al. Rethinking SSL development in an appified world[C]// Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. 2013：49-60.

[10] ROBILLARD M P，BODDEN E，KAWRYKOWD，et al. Automated API property inference techniques[J]. IEEE Transactions on Software Engineering，2012，39(5)：613-637.

[11] NIU H，KEIVANLOO I，ZOU Y. API usage pattern recommendation for software development[J]. Journal of Systems and Software，2017，129：127-139.

[12] WANG J，DANG Y，ZHANG H，et al. Mining succinct and high-coverage API usage patterns from source code[C]// 2013 10th Working Conference on Mining Software Repositories (MSR). IEEE，2013：319-328.

[13] ZHONG H，XIE T，ZHANG L，et al. MAPO：Mining and recommending API usage patterns[C]// 23rd European Conference (ECOOP 2009). Genoa，Italy，Springer Berlin Heidelberg，2009：318-343.

[14] FOWKES J，SUTTON C. Parameter-free probabilistic API mining across GitHub[C]// Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016：254-265.

[15] ACHARYA M，XIE T，PEI J，et al. Mining API patterns as partial orders from source code：from usage scenarios to specifications[C]// Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering. 2007：25-34.

[16] ZHAO W，ZHANG L，LIU Y，et al. SNIAFL：towards a static non-interactive approach to feature location[C]// International Conference of Software Engineering. ICSE，2004.

[17] TREUDE C，ROBILLARD M P. Augmenting API documentation with insights from stack overflow[C]// Proceedings of the 38th International Conference on Software Engineering. 2016：392-403.

[18] WU Y C，MAR L W，JIAUH C. Codocent：Support API usage with code example and API documentation[C]// 2010 Fifth International Conference on Software Engineering Advances. IEEE，2010：135-140.

[19] HOFFMAN D，STROOPER P. API documentation with executable examples[J]. Journal of Systems and Software，2003，66(2)：143-156.

[20] STYLOS J，FAULRING A，YANG Z，et al. Improving API documentation using API usage information[C]// 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE，2009：119-126.

[21] SUBRAMANIAN S，INOZEMTSEVA L，HOLMES R. Live API documentation[C]// Proceedings of the 36th International Conference on Software Engineering. 2014：643-652.

[22] CHEN C，ZHANG K. Who asked what：Integrating crowdsourced faqs into api documentation[C]// Companion Proceedings of the 36th International Conference on Software Engineering. 2014：456-459.

[23] KIM J，LEE S，HWANGS W，et al. Enriching documents with examples：A corpus mining approach[J]. ACM Transactions on Information Systems(TOIS)，2013，31(1)：1-27.

[24] MOVER S，SANKARANARAYANAN S，OLSENR B P，et al. Mining framework usage graphs from app corpora[C]// 2018

IEEE 25th International Conference on Software Analysis, Evolution and Reengineering(SANER). IEEE, 2018: 277-289.

[25] ZHANG J, JIANG H, REN Z, et al. Enriching API documentation with code samples and usage scenarios from crowd knowledge[J]. IEEE Transactions on Software Engineering, 2019, 47(6): 1299-1314.

[26] FENG X, YANG Q H. The Collaborative Filtering Algorithm for Information Enhancement Combined with Knowledge Graph [J]. Journal of Jiangxi Normal University: Natural Science Edition, 2022, 46(4): 386-393. (in Chinese)
冯祥, 杨庆红. 结合知识图谱进行信息强化的协同过滤算法[J]. 江西师范大学学报(自然科学版), 2022, 46(4): 386-393.

[27] HARTIGAN J A, WONGM A. Algorithm AS 136: A k-means clustering algorithm[J]. Journal of the Royal Statistical Society. Series C(Applied Statistics), 1979, 28(1): 100-108.

[28] SALVADOR S, CHAN P. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms [C]// 16th IEEE International Conference on Tools with Artificial Intelligence. IEEE, 2004: 576-584.

[29] ROUSSEEUW P J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis[J]. Journal of Computational and Applied Mathematics, 1987, 20: 53-65.

[30] KIM J, LEE S, HWANGS W, et al. Enriching documents with examples: A corpus mining approach[J]. ACM Transactions on Information Systems(TOIS), 2013, 31(1): 1-27.

[31] ZHONG L H, QI J, YE H T, et al. The Study on the Method for Matching the Software Evolutionary Style Based on Multi-Dimensional Evolutionary Tree[J]. Journal of Jiangxi Normal University: Natural Science Edition, 2021, 45(1): 55-59. (in Chinese)
钟林辉, 齐杰, 叶海涛, 等. 基于多维属性演化树的软件演化风格匹配方法研究[J]. 江西师范大学学报(自然科学版), 2021, 45(1): 55-59.

ZHONG Linhui, born in 1974, Ph.D, associate professor, is a member of CCF (No. 09772M). His main research interests include software architecture, software evolution, and maintenance.

ZHENG Yi, born in 1974, master. His main research interests include computer system architecture and the development of software and hardware platforms for embedded systems.