计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200200009

代码标识符归一化研究现状及发展趋势

张静宣[1]　　江　贺[2]
1 南京航空航天大学计算机科学与技术学院　南京 211106
2 大连理工大学软件学院　辽宁 大连 116600
(jxzhang@nuaa.edu.cn)

摘　要　作为代码分析和理解的重要内容,代码标识符及其归一化是国际学术界的前沿热点研究领域。标识符归一化旨在将标识符解析成自然语言词汇,以提高代码的可理解性和可维护性。标识符归一化主要包括两个极具挑战性的步骤,分别为组合词拆分和缩写词扩充。文中详细介绍了代码标识符归一化的研究现状,并进行了深入分析,总结出现有工作的困难和不足。同时,为了解决标识符归一化面临的困难和挑战,对该领域可行的解决思路和未来的发展趋势进行了归纳和展望,希望引导更多的研究者投入到这个重要的研究领域。

关键词:源代码分析;标识符归一化;组合词拆分;缩写词扩充;软件演化
中图法分类号　TP311.5

Research Status and Development Trend of Identifier Normalization

ZHANG Jing-xuan[1] and JIANG He[2]
1 College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China
2 School of Software,Dalian University of Technology,Dalian,Liaoning 116600,China

Abstract　As an important research content of source code analysis and comprehension,identifier normalization is the leading field of the current research of software engineering. Identifier normalization aims to parse identifiers into natural language terms so as to improve the understandability and maintainability of source code. There are generally two challenging steps in identifier normalization:identifier splitting and identifier expansion. This paper introduced the research status of identifier normalization in detail,conducted an in-depth analysis of the research status,and summarized the difficulties and deficiencies of the existing work. At the same time,in order to solve the difficulties and challenges in identifier normalization,this paper summarized and prospected the feasible solutions and future development trends in this field,hoping to guide more researchers into this important research field.

Keywords　Source code analysis,Identifier normalization,Identifier splitting,Abbreviation expansion,Software evolution

1　引言

分析和理解程序代码是软件开发的核心任务之一[1-2]。为了提高软件开发的效率,主流的软件系统由一组或者多组软件开发者进行协同开发。在此情形下,代码是软件开发者进行协同开发时交流和沟通的主要媒介。由某个开发者编写的代码经常会被其他开发者阅读、参考和审阅,如进行代码审查。作为代码中的重要组成部分,代码标识符大约占代码词汇总量的70%[3]。因此,代码标识符的可读性和可理解性在很大程度上影响了开发者之间的协作进程和协作效率[4-5]。

软件协同开发团队通常遵循统一的代码规范(Code Convention)来定义代码标识符[6]。一个常见的代码规范要求标识符应该尽量用自然语言表达其所实现的语义功能。语义功能的表示通常很难由单一词汇完全表达出来。因此,在代码标识符的定义过程中产生了大量的由自然语言构成的组合词。同时,代码规范也要求标识符不能过短或者过长,很多项目都对标识符的长度有一定的限制。因此,软件开发者通常会在标识符中使用缩写词,特别是在组合词中也会使用缩写词。在很多情况下,软件开发者会在一定程度上牺牲标识符的可理解性,大量使用组合词和缩写词来加速程序的开发[7-8]。

代码标识符中的组合词和缩写词会对软件开发者理解和分析代码产生重要的影响。一是软件开发者需要人工识别标识符中的组合词和缩写词,并且根据自己的理解将这些组合

到稿日期:2019-12-05　返修日期:2020-02-19　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:国家重点研发项目(2018YFB1003900);国家自然科学基金(61902181)
This work was supported by the National Key Research and Development Plan of China (2018YFB1003900) and National Natural Science Foundation of China (61902181).
通信作者:江贺(jianghe@dlut.edu.cn)

词和缩写词分别进行拆分和扩充。由于软件开发者在很多情况下并不是非常熟悉所要理解和审查的代码，导致他们很难对代码标识符的含义做出正确的理解，进而可能产生潜在的程序缺陷。对标识符的错误理解严重浪费了软件开发者的时间，降低了软件开发的效率。另一个严重的影响是，软件开发者无法将大量使用组合词和缩写词的代码追踪链接到对应的软件开发文档。代码并不是软件开发过程中产生的唯一产品。为了规范软件的开发过程，开发者也会书写和维护大量的由自然语言描述的开发文档，如需求文档、测试报告、缺陷报告等[9]。由于代码中存在标识符的组合词和缩写词，导致代码和开发文档之间的词汇表不一致，严重影响了它们之间的追踪链接关系的构建[10-11]，导致软件难以开发和维护。

为了降低代码标识符中的组合词和缩写词对软件开发和维护的消极影响，研究者相继提出一些代码标识符归一化(Identifier Normalization)方法。标识符归一化是指将标识符中的组合词和缩写词分别进行拆分和扩充，将代码标识符重新构造成软件开发者能够理解的自然语言的单词的过程[12]。代码标识符归一化处理的一般流程如图1所示。针对代码标识符，首先根据标识符中的特殊符号及大小写等识别该标识符是否是组合词。如果是组合词，需要根据一些启发式规则，采用正则表达式匹配的方式进行拆分。接着，将拆分后的词与自然语言的单词进行匹配，判断其是否为缩写词。针对缩写词，查找通用的包含常见的缩写词及其扩充词的缩写词词典进行扩充。如果某个缩写词有多种扩充方式，则需要设计相应的扩充算法，从多种扩充方式中选择最合适的扩充词。最后，对原始的代码标识符进行组合词拆分和缩写词扩充，从而得到归一化后的标识符。

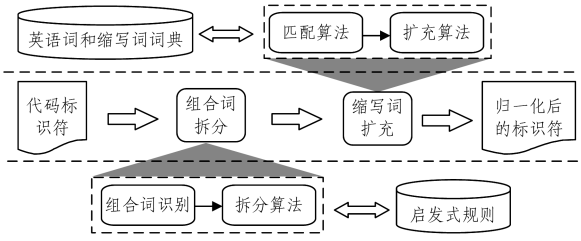

图1 标识符归一化的一般流程

Fig. 1 Workflow of identifier normalization

2 研究现状

代码标识符分析是代码分析的重要研究内容之一。标识符归一化主要包括两个步骤：将标识符组合词进行拆分和将标识符缩写词进行扩充[12]。这两个步骤都可以独立地作为一个研究任务被调研，同时这两个步骤也相互影响。

2.1 标识符组合词拆分

如果标识符为组合词，那么需要将其进一步拆分成单个的单词(称其为基本词)。根据基本词的合并方式可以将组合词划分成两类：硬词(hard words)和软词(soft words)[12-13]。硬词是指具有明显标记的比较容易拆分的组合词。常见的硬词是包含连字符或者使用驼峰构词法构成的组合词。相反，软词是指没有明显标记构成的组合词。软词的拆分相对于硬词来说更加困难。

针对包含连字符的硬词，将预定义的连字符替换成空格就可以将这些硬词进行拆分。如将“bug_number”中的连字符“_”替换成空格就得到“bug”和“number”两个单词[14]。针对使用驼峰构词法构成的硬词，可以使用一些启发式规则对其进行拆分。驼峰构词法可以根据字母的大小写情况分成3种形式[14]。1)第一个单词以小写字母开始，从第二个单词开始以后的每个单词的首字母都采用大写字母。针对这种形式的驼峰词，需要识别出其中的大写字母，并在这些大写字母之前的位置进行拆分。2)在第一种形式的基础上，第一个单词的首字母也大写。与第一种拆分方式相同，对此类驼峰词在大写字母的前面进行拆分。3)两个及以上大写字母的后面连接若干小写字母。针对这种形式的驼峰词，需要在倒数第一个大写字母的前面进行拆分。

软词的拆分相对于硬词更加困难，需要设计专门的算法[15-16]。Enslen等提出了Samurai方法来自动地拆分标识符组合词[17]。Samurai引入了两个词频表，即项目特定频率表和全局频率表，分别记录每个单词在该项目和该数据集出现的频率；其基于这两个频率表，计算不同的拆分位置的得分值，最终决定拆分的最佳位置。Guerrouj等提出了TIDIER来拆分标识符组合词[18-19]。该方法依赖于一个包含英语单词和技术专用单词的词典，使用一种在语音识别方面非常有效的算法，即动态时间规整(Dynamic Time Warping)，来计算词典中的单词和标识符的子串之间的距离；以此距离为基础计算最终的适应度函数，进而进行拆分。Butler等提出了标识符组合词拆分方法INTT[20]。该方法在已有方法的基础上引入一些特殊的启发式规则来处理标识符中的数字和小写字母。Sureka充分利用Yahoo网页搜索和图片搜索来提高标识符组合词的拆分效果[21]。其基本思路是标识符组合词中的基本词应该表达相应的概念，使用搜索引擎的返回结果可以在一定程度上反映基本词的流行程度和相关性。

2.2 标识符缩写词扩充

给定一个标识符缩写词，Lawrie等尝试从该缩写词所在的上下文中找到对应的扩充词[22]。该方法维护了多个候选列表，包括该缩写词所在位置的代码中抽取的自然语言单词列表、词组列表、停用词列表、通用英语单词列表，其中代码中抽取的列表优先级较高。将缩写词和具有不同优先级的列表依次匹配，直到找到正确的扩充词为止。但是，该方法不适用于一个缩写词匹配多个扩充词的情况。针对该问题，Hill等设计了一些规则来扩充缩写词，在缩写词所在的函数、类、项目级别分别使用正则表达式查找缩写词对应的扩充词[8]。特别是在一个缩写词有多个扩充词的情况下，该方法选择使用频率较高的扩充词进行扩充。Jiang等针对函数参数中的标识符这一特殊领域进行了缩写词的扩充[3]。针对函数参数中的缩写词，该算法分别从参数对应的形参(实参)、参数的类型、通用缩写词词典3个方面来构建启发式规则并依次对其进行匹配。在充分利用了特定领域信息后，该方法对于函数参数中缩写词的扩充具有很好的效果。

2.3 标识符归一化整体方法

除了对标识符组合词拆分和缩写词扩充分别进行研究，研究者也提出了一些包含这两个步骤的标识符归一化整体方法。Lawrie等提出了一种代码词汇归一化方法Normalize[23]，该工作是标识符归一化较早期的工作之一。Norma-

lize 对标识符组合词的每一个可能的拆分位置进行分数评估,对获得最高分数的位置进行拆分。标识符缩写词的扩充依赖于通配符,在缩写词中插入通配符,然后在代码中进行匹配和查找。Guerrouj 等提出了另一种代码标识符归一化方法 TRIS[14]。TRIS 将标识符的拆分和扩充问题转化成为图优化问题,在标识符构成的加权无环图中寻找最优化路径,进而对标识符进行归一化。Corazza 等提出了 LINSEN[24],该方法从多个方面抽取信息来构建词典,包括代码的注释、程序和技术关键词、常见的英语词词典、缩写词词典。在构建大量词典的基础上,LINSEN 使用一种近似字符串匹配的技术来完成标识符的归一化。

3 研究现状分析

通过对现有研究工作的分析可以发现,针对代码标识符归一化的工作各有特色,也取得了一定的成果,但是代码标识符归一化工作也存在一些不足,下面从 3 个层面进行归纳。

(1)在算法设计方面,现有的代码标识符归一化算法依赖于组合词的拆分启发式规则和缩写词词典,它们的构建过程需要大量的人工操作,可扩展性不强。启发式规则的构建和缩写词词典的构建需要在编程上下文数据的基础上,充分利用代码标识符的特定领域知识[25-26]。代码标识符是软件开发者在编程规范的基础上,根据编程上下文的数据开展的个性化创造活动[27]。因此,代码标识符兼具编程规范的通用性和开发者代码风格的独特性,现有工作并没有充分利用代码标识符的这些性质。

(2)在演化分析方面,虽然现有的研究工作总结了标识符的一些性质及其对代码的分析和理解的影响,但是并没有有效分析和归纳标识符归一化的演化规律。研究者应该站在发展的角度对代码标识符进行分析。已有的工作已经表明[28-29],随着软件版本的演化迭代,标识符及其归一化的结果也在不断演化,需要进行一定程度的调整和适应。对标识符及其归一化的演化情况进行分析有望进一步增强开发者对标识符演化规律的了解,进而理解软件的演化趋势。另外,由于代码的组织结构具有高内聚低耦合的性质,代码标识符及其归一化在不同的代码组织结构的不同粒度上可能存在不同的形态。归纳代码标识符及其归一化在不同软件代码粒度上的变化有望进一步提高代码与其他软件文档的链接效果。

(3)在应用方面,代码标识符及其归一化的应用范围需要得到进一步拓展。研究者已经充分认识到了代码标识符在代码相关研究任务的重要性。在针对典型的基于信息检索的软件开发任务中(如代码推荐),代码标识符及其归一化的影响力研究尚未得到有效开展。尤其是在开源网站上存储了大量开源代码的基础上,我们可以获取和利用这些软件代码库进行大规模实证研究,进一步探索和拓展标识符及其归一化的应用范围[30]。

4 发展趋势

在算法设计方面,代码标识符的通用性和独特性为代码标识符归一化算法设计提供了新的思路。由于编程规范具有通用性,很多软件代码都遵循公认的编程规范,如代码标识符的命名需要在有限长度内表达对应的概念和行为。随着开源软件的流行,GitHub 等代码托管平台上存储了大量的相似的软件项目代码,为我们解决代码标识符归一化问题带来了新的思路。给定一个需要进行标识符归一化的代码库,我们可以挖掘相似代码库的标识符归一化结果,为给定的代码库中的标识符生成归一化方案。同时,标识符的创作过程又被开发者的个性化代码风格所影响。不同的开发者有不同的代码风格,对软件也有不同的理解,很容易将个人对代码的理解反映到代码标识符的命名过程中。因此,有必要挖掘不同开发者书写标识符的不同风格,针对不同开发者的行为习惯和代码风格制定不同的代码标识符归一化方案。开源代码托管平台上的不同开发者提交代码的信息可以容易地被获取和分析,为我们充分挖掘开发者的代码风格奠定了基础。因此,以开源代码平台为数据驱动,充分挖掘代码标识符的通用性和独特性有望极大地提高代码标识符归一化的效果。

在演化分析方面,代码标识符及其归一化的演化性为开发者动态和全面地了解软件的演化规律和概念分布情况提供了新的契机。随着软件版本的演化,代码标识符所反映的软件概念和行为也会得到适当的调整。这种代码标识符及其归一化的演化情况可以分成两种:1)同一个标识符的归一化结果随着软件的演化出现了不同的结果,即同一个标识符随着软件的演化出现了概念和行为的迁移;2)一部分标识符及其归一化的结果逐渐消失,另外一部分标识符及其归一化的结果逐渐形成,即不同的标识符随着软件的演化出现了消亡和新生。同时,软件代码通常以高内聚低耦合为原则按照不同的粒度进行组织。分析代码标识符在代码不同粒度的变化情况,可以使开发者认识到代码的某些粒度和某些功能具有较强的关联,进而在涉及到该功能的修改时快速定位到对应的代码。

在应用方面,代码标识符及其归一化的应用性为众多软件开发任务的有效解决提供了新的方向。随着代码标识符的重要性逐渐凸显,标识符及其归一化亟待被应用在更多软件开发任务上,尤其是基于信息检索的软件开发任务上(如代码推荐),使得原来在词法上无法匹配的标识符通过归一化可以进行词法匹配。现有的研究工作尚未对代码标识符及其归一化对基于信息检索的软件开发任务的影响进行系统性的实证研究。因此,标识符归一化的成果为提升基于信息检索的软件开发任务的效果提供了新的方向,代码标识符归一化的应用范围将会得到进一步拓展。

结束语 针对现有标识符归一化研究工作的困难和不足,本文提出了解决思路和未来的发展趋势,即充分利用标识符的通用性和独特性有望提高标识符归一化的算法效果;针对标识符归一化进行演化分析可以帮助开发者动态了解软件的演化规律。另外,标识符归一化的成果可以为一些软件开发任务的解决提供新思路。

参 考 文 献

[1] ALLAMANIS M,BARR E T,DEVANBU P,et al. A Survey of Machine Learning for Big Code and Naturalness [J]. ACM Computing Surveys,2018,51(4):1-37.

[2] AVIDAN E,FEITELSON D G. Effects of Variable Names on Comprehension:An Empirical Study[C]// International Conference on Program Comprehension (ICPC 17). 2017:55-65.

[3] JIANG Y J, LIU H, ZHU J Q, et al. Automatic and Accurate Expansion of Abbreviations in Parameters [J]. IEEE Transactions on Software Engineering, 2018, PP(99): 1-1.
[4] KIM S, KIM D. Automatic Identifier Inconsistency Detection using Code Dictionary[J]. Empirical Software Engineering, 2016, 12(2): 565-604.
[5] JIN Z, LIU F, LI G. Program Comprehension: Present and Future [J]. Journal of Software, 2019, 30(1): 110-126.
[6] ZHANG J, ZHANG C, XUAN J F, et al. Recent Progress in Program Analysis [J]. Journal of Software, 2019, 30(1): 80-109.
[7] JIANG H, CHEN X, ZHANG J X, et al. Mining Software Repositories: Contributors and Hot Topics [J]. Journal of Computer Research and Development, 2016, 53(12): 2768-2782.
[8] HILL E, FRY Z P, BOYD H, et al. AMAP: Automatically Mining Abbreviation Expansions in Programs to Enhance Software Maintenance Tools[C]// International Working Conference on Mining Software Repositories (MSR 08). 2008: 79-88.
[9] ZHANG J X, JIANG H, REN Z L, et al. Enriching API Documentation with Code Samples and Usage Scenarios from Crowd Knowledge [J]. IEEE Transactions on Software Engineering, 2018, PP(99): 1-1.
[10] JIANG H, ZHANG J X, REN Z L, et al. An Unsupervised Approach for Discovering Relevant Tutorial Fragments for APIs [C]// International Conference on Software Engineering (ICSE 17). 2017: 38-48.
[11] JIANG H, ZHANG J X, LI X C, et al. A More Accurate Model for Finding Tutorial Segments Explaining APIs[C]// International Conference on Software Analysis, Evolution, and Reengineering (SANER 16). 2016: 157-167.
[12] CARVALHO N R, ALMEIDA J J, HENRIQUES P R, et al. From Source Code Identifiers to Natural Language Terms [J]. Journal of Systems and Software, 2015, 100: 117-128.
[13] NEWMAN C D, ALSUHAIBANI R S, COLLARD M L, et al. Lexical Categories for Source Code Identifiers[C]// International Conference on Software Analysis, Evolution and Reengineering (SANER 17). 2017: 228-239.
[14] GUERROUJ L, GALINIER P, GUEHENEUC Y, et al. TRIS: A Fast and Accurate Identifiers Splitting and Expansion Algorithm [C]// Working Conference on Reverse Engineering (WCRE 12). 2012: 103-112.
[15] HILL E, BINKLEY D, LAWRIE D, et al. An Empirical Study of Identifier Splitting Techniques [J]. Empirical Software Engineering, 2014, 19: 1754-1780.
[16] ZHANG B, HILL E, CLAUSE J. Towards Automatically Generating Descriptive Names for Unit Tests [C]// International Conference on Automated Software Engineering (ASE 16). 2016: 625-636.
[17] ENSLEN E, HILL E, POLLOCK L L, et al. Mining Source Code to Automatically Split Identifiers for Software Analysis[C]// International Working Conference on Mining Software Repositories (MSR 09). 2009: 71-80.
[18] GUERROUJ L, PENTA M D, ANTONIOL G, et al. TIDIER: An Identifier Splitting Approach Using Speech Recognition Techniques [J]. Journal of Software: Evolution and Process, 2013, 25(6): 575-599.
[19] MADANI N, GUERROUJ L, PENTA M D, et al. Recognizing Words from Source Code Identifiers using Speech Recognition Techniques [C]// European Conference on Software Maintenance and Reengineering (CSMR 10). 2010: 68-77.
[20] BUTLER S, WERMELINGER M, YU Y, et al. Improving the Tokenisation of Identifier Names[C]// European Conference on Object-oriented Programming (ECOOP 11). 2011: 130-154.
[21] SUREKA A. Source Code Identifier Splitting Using Yahoo Image and Web Search Engine[C]// International Workshop on Software Mining. 2012: 1-8.
[22] LAWRIE D, BINKLEY D. Expanding Identifiers to Normalize Source Code Vocabulary[C]// International Conference on Software Maintenance (ICSM 11). 2011: 113-122.
[23] LAWRIE D, BINKLEY D, MORRELL C. Normalizing Source Code Vocabulary [C]// Working Conference on Reverse Engineering (WCRE 10). 2010: 3-12.
[24] CORAZZA A, MARTINO S D, MAGGIO V. LINSEN: An Efficient Approach to Split Identifiers and Expand Abbreviations [C]// International Conference on Software Maintenance (ICSM 12). 2012: 233-242.
[25] ARNAOUDOVA V, ESHKEVARI L M, PENTA M D, et al. REPENT: Analyzing the Nature of Identifier Renamings [J]. IEEE Transactions on Software Engineering, 2014, 40(5): 502-532.
[26] TU Z, SU Z, DEVANBU P. On the Localness of Software[C]// International Symposium on Foundations of Software Engineering (FSE 14). 2014: 269-280.
[27] HINDLE A, BARR E T, SU Z, et al. On the Naturalness of Software[C]// International Conference on Software Engineering (ICSE 12). 2012: 837-847.
[28] LIN B, SCALABRINO S, MOCCI A, et al. Investigating the Use of Code Analysis and NLP to Promote a Consistent Usage of Identifiers [C]// International Working Conference on Source Code Analysis and Manipulation (SCAM 17). 2017: 81-90.
[29] SCALABRINO S, BAVOTA G, VENDOME C, et al. Automatically Assessing Code Understandability: How Far Are We? [C]// International Conference on Automated Software Engineering (ASE 17). 2017: 417-427.
[30] LUCIA D A, PENTA M D, OLIVETO R. Improving Source Code Lexicon via Traceability and Information Retrieval [J]. IEEE Transactions on Software Engineering, 2011, 37(2): 205-227.

ZHANG Jing-xuan, born in 1988, Ph.D, assistant professor, is member of China Computer Federation. His main research interests include mining software repositories and so on.

JIANG He, born in 1980, Ph.D, professor, Ph.D. supervisor, is Distinguished member of China Computer Federation. His main research interests include mining software repositories and intelligent software engineering.