基于上下文的智能化代码复用推荐

彭鑫[1,2]，陈驰[1,2]，林云[3]
1. 复旦大学计算机科学技术学院，上海 200438；2. 上海市数据科学重点实验室，上海 200438；
3. 新加坡国立大学计算机学院，新加坡 117418

摘要
基于代码大数据分析、挖掘和学习的智能化代码复用推荐能够有效地提高软件复用的效率和质量，包括特定领域内的共性代码单元以及与领域无关的通用代码单元。围绕基于上下文的智能化代码复用推荐这一主题，阐述了基于模板挖掘的代码复用推荐和基于深度学习的代码复用推荐两个方面的研究工作。在此基础上，针对基于上下文的智能化代码复用推荐的未来发展方向进行了展望。

关键词
软件复用；代码推荐；代码上下文；API；代码模板；深度学习

中图分类号：TP311 文献标识码：A doi: 10.11959/j.issn.2096-0271.2021003

Context-based intelligent recommendation for code reuse

PENG Xin[1,2], CHEN Chi[1,2], LIN Yun[3]
1. School of Computer Science, Fudan University, Shanghai 200438, China
2. Shanghai Key Laboratory of Data Science, Shanghai 200438, China
3. School of Computing, National University of Singapore, Singapore 117418, Singapore

Abstract
Intelligent code reuse recommendation based on code-related big data analysis, mining, and learning can improve the efficiency and quality of software reuse significantly. The targets of reuse include domain specific common code units and domain independent common code units. Context-based intelligent recommendation for code reuse was focused, template mining based code reuse recommendation and deep learning based code reuse recommendation were described. Based on these two parts of work, the future trend of context based intelligent recommendation for code reuse was discussed further.

Key words
software reuse, code recommendation, code context, API, code template, deep learning

1 引言

各种形式的代码复用一直是软件开发人员广泛使用的一种高效的辅助开发手段，复用的对象包括相似功能模块、代码片段以及应用编程接口（application programming interface，API）等不同粒度的代码单元。在传统的代码复用方式中，开发人员需要利用互联网搜索引擎或企业代码库搜索等手段获取特定领域的或与领域无关的可复用代码单元，同时查找样例代码和文本解释等帮助信息，在此基础上选择代码单元，并完成修改和集成。这种代码复用方式虽然有效，但依赖开发人员的经验，同时需要进行大量的信息查找和确认，因此复用效率不高，而且容易在代码中引入缺陷。

开源和企业代码库以及软件技术文档、软件开发问答等软件开发资源的大量积累形成了代码大数据。例如，代码托管平台GitHub上已经聚集了超过2.7亿个代码仓库，软件开发问答网站Stack Overflow上已经积累了超过1 700万个软件开发问题。软件开发过程中很多时候遇到的是重复性的开发问题，如通用功能实现、通用API及其使用模式，因此通过代码大数据分析、挖掘和学习可以实现智能化的代码复用推荐。对于软件开发人员而言，最有效的方式是基于代码上下文的智能化代码复用推荐，即根据集成化开发环境（integrated development environment，IDE）中当前开发任务已经完成的代码等上下文信息，针对性地推荐可用的代码单元，同时辅助开发人员进行定制化的代码修改和集成。

针对基于上下文的智能化代码复用推荐，研究人员使用数据挖掘、统计模型、深度学习等各种技术手段开展了一系列研究和探索，包括基于上下文匹配的API使用模式推荐[1]、基于上下文概率模型的代码补全[2]、基于上下文图统计语言模型的API推荐[3]等。这些方法推荐的可复用代码单元覆盖了API使用模式、代码片段、单个API和语句行等不同粒度，其有效性已经在一定范围内得到了验证。然而，现有的智能化代码复用推荐还无法提供一种具有广泛适用性以及能够按需调整推荐粒度及内容的系统性的智能辅助开发支持。

一般而言，软件开发人员的代码复用对象包括特定领域的共性代码单元以及与领域无关的通用代码单元。前者的复用范围局限在特定领域内，但与核心业务关系更密切，例如以代码片段或功能模块的形式出现的相似业务功能的代码实现变体。后者的复用范围更广，但与核心业务关系较弱，如通用API及其使用模式、通用算法与功能实现等。不同类型的代码复用对象需要采用不同的智能化方法进行分析和推荐。本文围绕基于上下文的智能化代码复用推荐这一主题，从基于模板挖掘的代码复用推荐和基于深度学习的代码复用推荐两个方面介绍笔者的研究工作。其中，基于模板挖掘的代码复用推荐支持代码片段和功能模块两个层次的复用推荐，同时所需的相似代码副本数量较少，适用于面向特定领域的共性代码复用推荐。基于深度学习的代码复用推荐支持单行及多行API调用代码的推荐，所需的训练数据较多，适用于与领域无关的通用代码复用推荐。在此基础上，本文还将针对基于上下文的智能化代码复用推荐的未来发展方向进行展望。

2 基于模板挖掘的代码复用推荐

软件开发人员经常会实现相似或相同

的功能，相应的实现代码也是相似的。这种代码片段级别的复用行为在开源和企业软件开发中十分普遍。在这种代码复用中，开发人员经常需要对所复用的代码进行定制化修改。此时，开发人员可能因为对代码本身以及不同部分之间的逻辑关联不够了解而造成遗漏修改或错误修改，进而导致缺陷的产生。此外，在功能模块级别上，开发人员可能会通过代码复制粘贴实现更大粒度的复用，其中隐含着对设计结构的复用。通过代码片段和功能模块两个层次的模板抽取，可以实现相应的代码复用推荐。

2.1 代码片段模板抽取与复用推荐

针对代码片段级别的复用，可以通过代码克隆检测发现当前所复用代码片段在项目库中的所有相似副本。这些相似副本被称为克隆实例，它们构成的集合被称为克隆类。该方法的基本思想在于，项目库中的每一个相似副本（克隆实例）都被视为一次基于代码复制的复用的结果，而这些克隆实例之间的差异则被视为代码复用过程中所做的修改。因此，开发人员通过代码复制复用一个代码片段时，可以找出所有与之相似的克隆实例，分析它们之间的差异，并从中抽取代码模板以及其中蕴含的关联关系用于复用推荐。

该方法的基本过程如图1所示[4]。在针对当前所复制代码片段的克隆检测结果的基础上，利用多段代码之间的差异比较技术来检测这些克隆实例之间的差异[5]，并将每一个差异转化为一个代码可变点。在此基础上，该方法通过基于历史代码信息（曾在可变点处出现过的代码）和上下文代码的分析，挖掘每个可变点上的内容选项以及相互之间的关联关系，并在开发人员的代码编辑过程中以交互式推荐的方式辅助实现代码复用。当开发人员在可变点中选择了某些代码内容选项或输入新的代码内容时，该方法可以基于挖掘的关联规则动态调整其他相关可变点上的代码内容选项及其推荐排序。

基于该方法实现的Eclipse插件CCDeamon（code cloning deamon）如图2所示。其中，代码内容上的方框表示所复制的代码上经过模板抽取后识别出的可变点，点击后将显示经过排序的推荐选项列表。

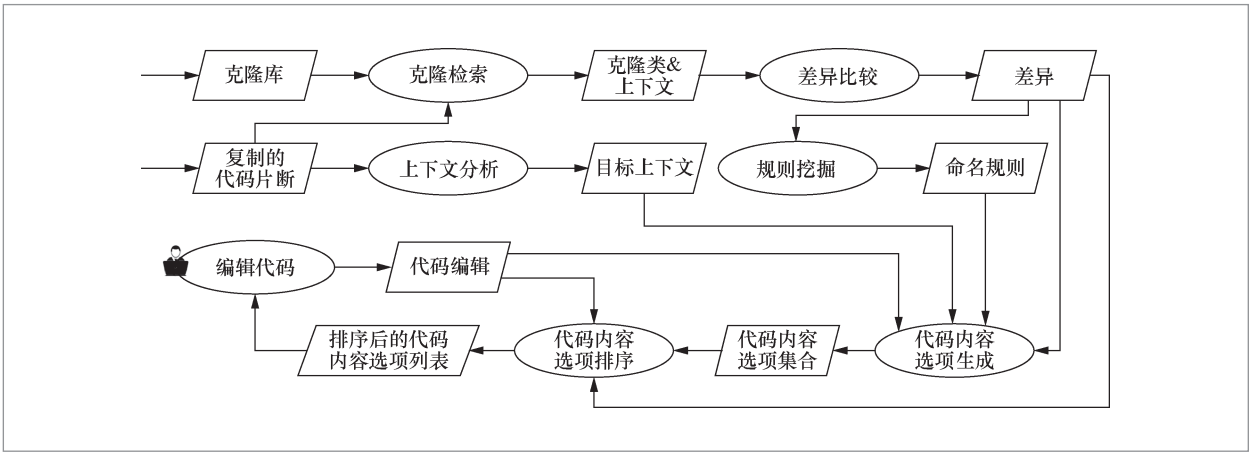

图1 代码片段模板抽取与复用推荐方法[4]

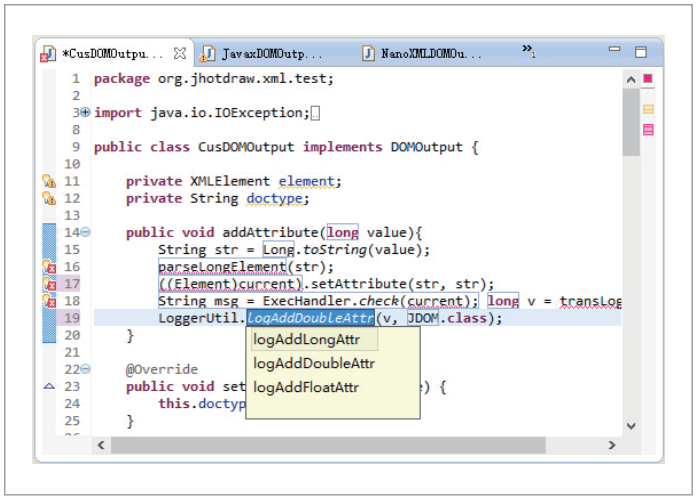

图 2 CCDeamon 工具插件 [4]

2.2 功能模块模板抽取与复用推荐

针对功能模块级别的复用，可以从相似副本中抽取出相似功能的设计和实现模板，并支持开发人员基于模板进行定制化的功能实现。该方法将抽取的抽象设计以类图形式保存为模板，在此基础上，支持开发人员基于模板自动生成骨架代码以及骨架内的部分实现代码[6]。

该方法的基本过程如**图3**所示，采用自底向上的聚合和抽象技术来完成代码模板的抽取[6]。该方法首先从项目源代码中逆向恢复出整个项目的静态设计模型，其中包含整个项目中的类、接口、方法、属性以及它们之间的关系。然后，该方法定义了模型内程序元素（即类、接口、方法和属性）的对应匹配方式，被匹配的程序元素被聚合成一个多重集。接着，该方法根据多重集间元素的关系（如继承和调用关系等）将多个多重集进行拼接，并进一步对每个多重集内部的程序元素进行抽象，形成模板程序元素；最后，该方法根据生成的模板支持开发人员通过定制生成新的应用程序骨架代码以及骨架内部代码。

该技术已经被实现，称为Eclipse插件MICoDe（mining implicit code design），如**图4**所示。**图4**右侧的模板视图列出了模板库中所有抽取出的可复用模板，中间的模板编辑器支持开发人员对模板进行二次编辑。UML类图可视化了一个模板中的类、接口、方法、属性以及它们之间的关系（如继承和调用关系等）。同时，工具底部的代码差异视图支持开发人员查看模板中某个元素（如方法）在项目中的多

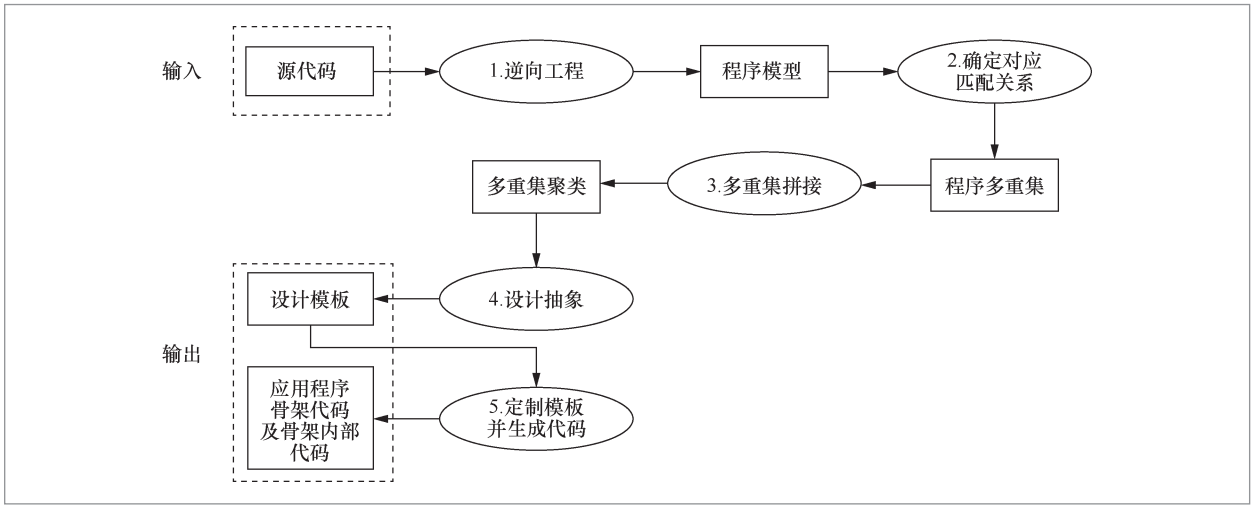

图 3 功能模块实现模板抽取与复用推荐方法 [6]

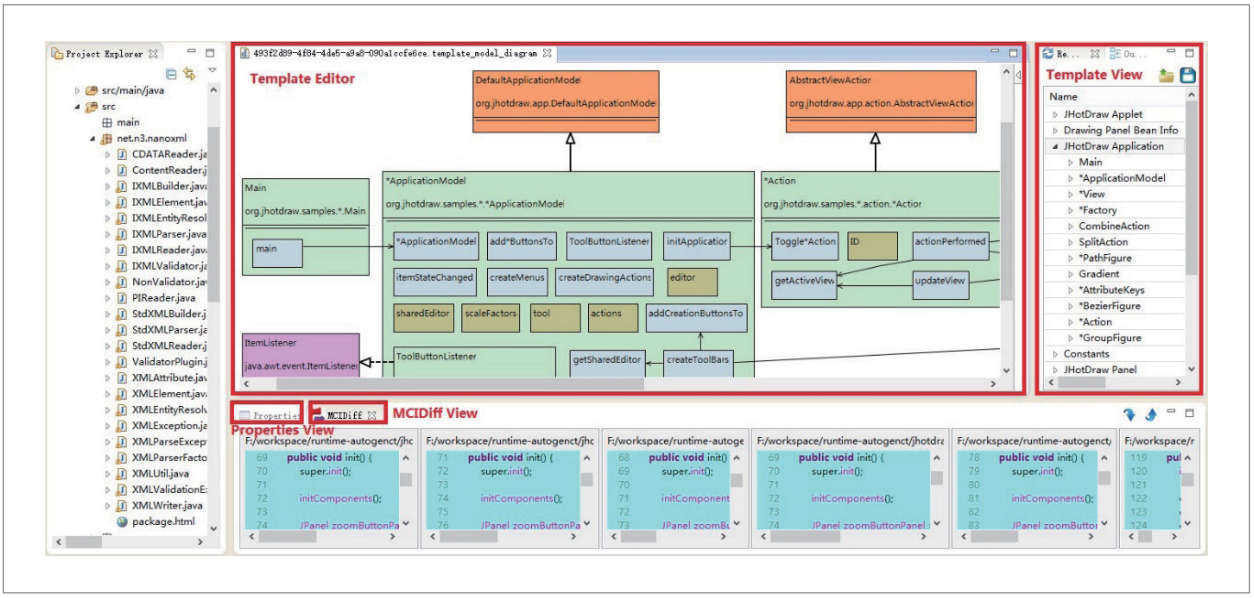

图 4 MICoDe 工具示意图 [6]

个实现示例，并高亮显示出这些示例之间的差异。

3 基于深度学习的代码复用推荐

由于深度学习在自然语言处理领域表现出的优越性，其被迁移到代码复用推荐领域。相对于模板挖掘这种显式的方式，深度学习通过隐式的方式学习代码的使用模式。基于深度学习的代码复用推荐方法分为训练阶段和预测阶段。在训练阶段，基于深度学习的代码复用推荐方法通常将代码解析为某种代码表示形式（如代码序列表示、代码树结构表示以及代码图结构表示等），设计或应用符合该代码表示形式的深度学习网络（如长短期记忆（long short-term memory，LSTM）网络、Tree-LSTM网络以及门控图神经网络（gated graph sequence neural network，GG-NN）等）进行学习和训练。在预测阶段，基于深度学习的代码复用推荐方法应用训练好的模型对给定的带有窟窿（即待完成的部分）的不完整代码进行预测。

3.1 基于Tree-LSTM网络的生成式API推荐

基于传统统计学习的API推荐方法[2,7-11]以及基于深度学习的API推荐方法[11-15]将代码解析为代码序列，并利用传统统计模型（如N-Gram模型等）或深度学习模型（如LSTM网络等）进行学习和训练。然而，这些研究方法没有考虑代码的结构信息（如控制流和数据流），不能有效地捕获相隔较远的代码之间的关系，并且没有实例化API中的参数。

基于Tree-LSTM网络的生成式API推荐方法DeepAPIRec如**图5**所示[16]。在训练阶段，DeepAPIRec训练了一个语句模型用于预测抽象API语句，并构造了一个参数模型用于实例化抽象API语句中的参数，从而形成实例化API语句。在预测阶段，对于给定的带有窟窿的程序，DeepAPIRec预测出实例化API语句供开发人员选择。

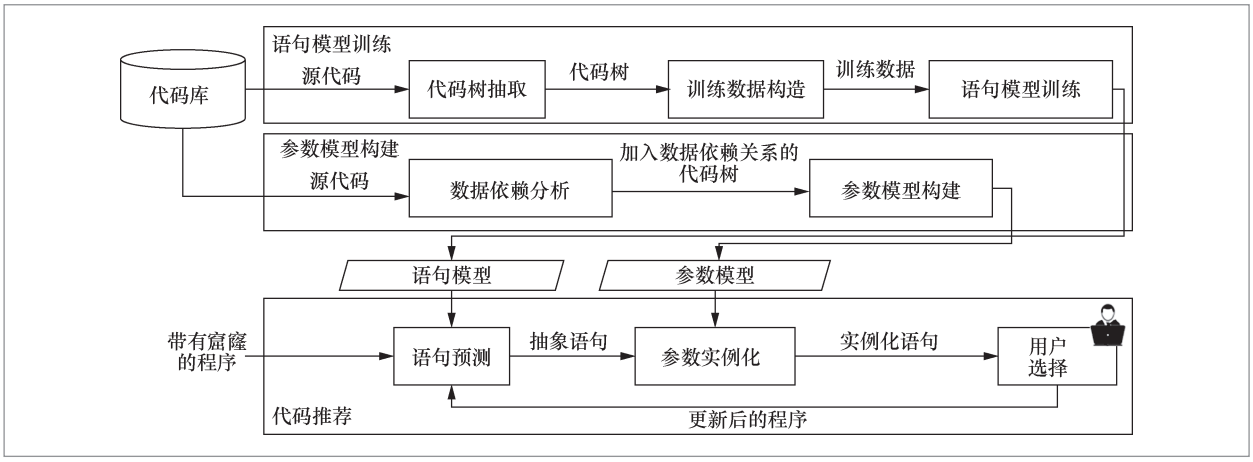

图 5 基于 Tree-LSTM 网络的生成式 API 推荐方法 DeepAPIRec[16]

在语句模型训练阶段，DeepAPIRec将代码解析为包含代码控制流信息的代码树。代码树中的每一个结点表示抽象的API语句、变量声明/赋值语句、控制结构或代码窟窿，边表示它们之间的控制流关系。DeepAPIRec结合Tai K S等人[17]提出的Child-Sum Tree-LSTM网络和N-ary Tree-LSTM网络，作为深度学习网络对代码树进行学习和训练，从而有效地捕获相隔较远的代码之间的关系以及代码结构信息的语义。

在参数模型构建阶段，DeepAPIRec在代码树上加入了数据依赖分析，从而引入数据流边，构成加入数据依赖关系的代码树。DeepAPIRec通过统计存在数据依赖关系的代码树中两个结点之间在相应路径上的数据依赖次数来构造参数模型。给定一个待预测的API结点及对应的加入数据依赖关系的代码树，参数模型计算每个结点在所有可以到达待预测API结点且长度大于2的路径上与待预测API结点产生数据依赖的次数，次数越高表示数据依赖概率越大，即该结点表示的变量越有可能成为预测的API中的参数。

在代码推荐阶段，给定一段带有窟窿的程序（如图6所示），DeepAPIRec首先将其解析为代码树（如图7所示），并输入语句模型进行预测，得到抽象API语句；其次，DeepAPIRec将其解析为加入数据流的代码树（如图8所示）；最后，用抽象API语句结点替换hole结点，并通过参数模型进行参数实例化，以形成实例化的API语句供开发人员选择。对于图6的代码示例，DeepAPIRec可以成功地推荐正确的API语句Signature signature = Signature.getInstance(signMode)。

3.2 基于深度学习及代码上下文结构和文本信息的API推荐

代码包含代码结构信息和代码文本信息两种核心信息。代码结构信息（如控制流和数据流）反映代码的程序逻辑特性，代码文本信息（如方法名、变量名等）反映代码在自然语言中的语义。现有的API推荐方法[2,7-15,18]要么将代码按照文本处理的方式处理为代码序列，要么只考虑代码的结构信息，并将代码解析为树或者图表示。这些API推荐方法仅独立建模代码结构信息或代码文本信息，没有将代码结

构信息和文本信息进行结合。此外，在考虑代码结构信息时应考虑代码的全局语义（即将代码看作一个完整的图表示），而非像GraLan[3]那样只考虑代码的局部语义（即将代码处理为子图进行推荐）。如**图9**所示，GraLan只考虑局部语义，因此GraLan无法成功地预测出正确的API。如果考虑全局语义，那么可以得知hole处的语义为“对一个String类型的变量进行某种处理，以获取一个int类型的值”。然而，只考虑代码结构信息无法得到hole处的准确语义，这是因为无法确定需要对String类型的变量进行何种处理。如**图10**所示，其代码结构信息与**图9**中的代码非常相似，但是hole处的语义与**图9**中的代码不一样。如果结合代码文本信息进行考虑，就可以得知**图9**中hole处的语义为“计算字符串的Hash值”，而**图10**中hole处的语义为“将字符串直接转为整数”。因此笔者团队提出一种名为APIRec-CST的基于深度学习及代码上下文结构和文本信息的API推荐方法[19]。

APIRec-CST将代码解析为API上下文图，以反映代码中的结构信息。API上下文图中的每个结点表示抽象API方法调用、成员变量访问、变量声明/赋值、控制结构或代码窟窿，边表示它们之间的控制流或数据流关系。APIRec-CST将代码文本信息（如方法名、参数名和变量名）提取为代码Token词袋，并通过分词、词形还原、去重等方式处理得到的代码Token词袋。对于得到的API上下文图及代码Token词袋，APIRec-CST通过**图11**所示的网络进行联合学习和训练。其中，APIRec-CST通过API上下文图网络学习API上下文图的全局语义，通过代码Token网络学习代码文本信息语义，并通过联合层结合代码结构信息和文本信息。

对于一段给定的带有窟窿的程序（如**图9**、**图10**所示），APIRec-CST将其解析为API上下文图（如**图12**所示）及代码Token词袋（compute、Hash、code、path、result、rd、br以及str），并分别输入**图11**所示的网络中。最后，APIRec-

```
1:public byte[] sign(String message, String digestAlgorithm, PrivateKey pk)
        throws GeneralSecurityException {
2:   byte[] messageByte=message.getBytes();
3:   String signMode=null;
4:   if(pk==null){
5:      pk=getPrivateKey("RSA");
6:      String encryptionAlgorithm=pk.getAlgorithm();
7:      signMode=combine(digestAlgorithm, encryptionAlgorithm);
8:   }else{
9:      String encryptionAlgorithm=pk.getAlgorithm();
10:     signMode=combine(digestAlgorithm,encryptionAlgorithm);
11:  }
12:  $hole$
13:}
```

图 6 利用 Signature 签名加密的代码示例

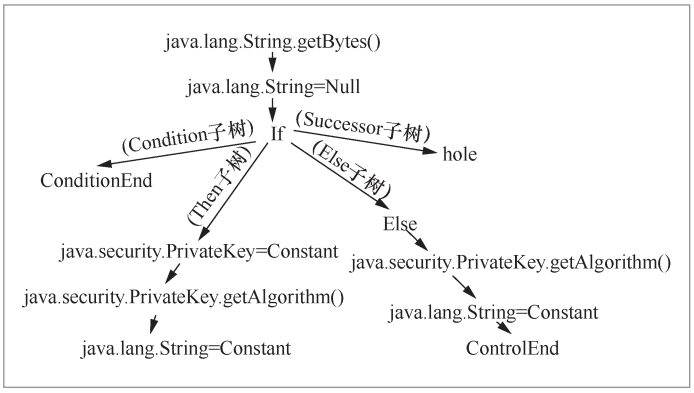

图 7 代码树示例

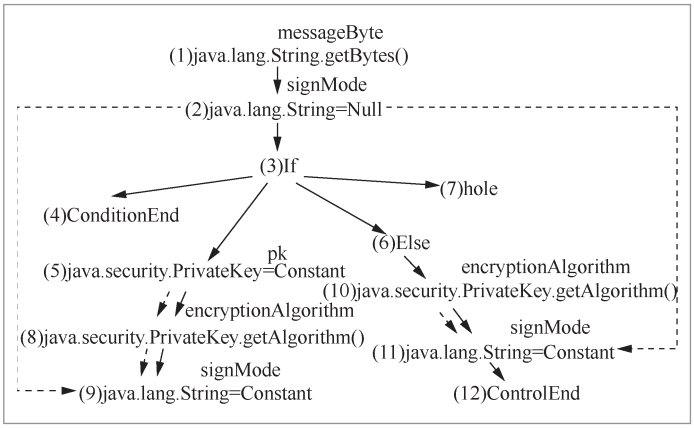

图 8 加入数据流的代码树示例

```
1:  public List<Integer> computeHashCode(String path) throws Exception{
2:          List<Integer> result=new ArrayList<>();
3:          FileReader rd=new FileReader(path);
4:          BufferedReader br=new BufferedReader(rd);
5:          String str=null;
6:          while((str=br.readLine())!=null){
7:              int HashCode;
8:              $hole$;
9:              result.add(HashCode);
10:         }
11:         br.close();
12:         rd.close();
13:         return result;
14: }
```

图 9 计算文件内容哈希值的代码示例

```
1:  public List<Integer> getIntegerScore(String path) throws Exception{
2:          List<Integer> result=new ArrayList<>();
3:          FileReader rd=new FileReader(path);
4:          BufferedReader br=new BufferedReader(rd);
5:          String str=null;
6:          while((str=br.readLine())!=null){
7:              int score;
8:              $hole$;
9:              result.add(score);
10:         }
11:         br.close();
12:         rd.close();
13:         return result;
14: }
```

图 10 从文件获取得分的代码示例

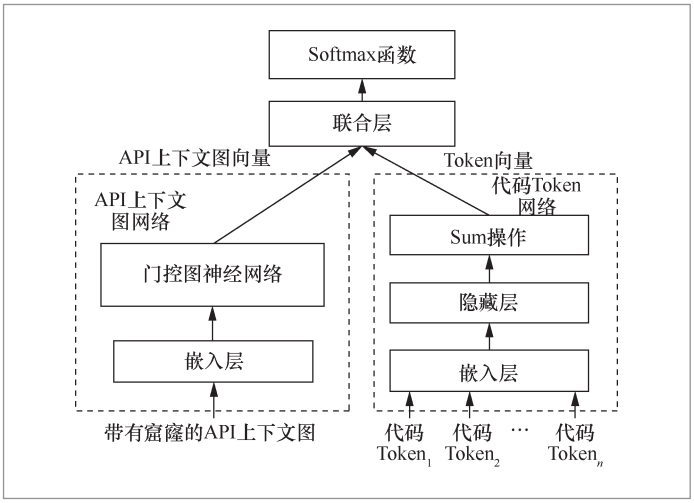

图 11 APIRec-CST 的网络模型结构 [19]

CST能够预测出正确的API推荐java.lang.String.HashCode()。

4 未来展望

各种智能化技术特别是深度学习技术的应用使得基于上下文的智能化代码复用推荐成为可能。然而，由于软件开发固有的复杂性和不确定性，在企业软件开发中全面实现智能化代码复用推荐还存在一系列技术挑战，主要包括：软件需求和设计中蕴含的创造性和不确定性、各种软件项目在业务和技术领域的多样性和差异性、代码及其他软件开发资源的数据质量问题[20]。

基于上下文的智能化代码复用推荐是一种辅助开发手段，需要与开发人员的主观经验、思考和判断能力相结合。另外，理想的智能化代码复用推荐应当提供一种集成化、系统化的智能开发辅助，实现通过单一的入口和渠道满足开发人员对不同粒度（如功能模块、代码片段、单个API或单行代码等）以及不同形态（参考代码、原理解释、解决方案描述等）的可复用资源和信息的需要。因此，未来有效的智能化代码复用推荐应当以一种人机协作的智能化助手的形式来实现。这种智能化助手隐藏在IDE之中，持续观察开发人员的行为以及代码上下文的变化，及时发现问题，并在需要的时候提供所需的各种帮助，例如推荐API或代码片段、提供代码说明并指导代码修改、给出解决方案建议、回答开发人员问题等。智能化助手的工作方式应当类似于结对编程，能够与软件开发人员持续交互和协作。为此，这种智能化助手应当具有以下多个方面的能力[20]。

- 交互式澄清与解释。以交互式的方式澄清开发人员的意图，同时对给出的推荐提供原理解释和说明。

● 逐步细化解决方案建议。按照开发人员的思维方式，由粗到细逐步推荐和细化代码复用及问题解决方案，而不是一开始就陷入细节。

● 获取和利用技术和业务背景知识。具备编程语言、API、领域模型等方面的技术和业务背景知识，以此来支持与开发人员的高效交互和复用推荐。

● 按需提供不同粒度和不同形式的解决方案。根据开发人员的意图和代码上下文，按需提供文本方案描述、问题解答、功能模块、代码片段、API模式、单个API、代码行及参数等不同粒度和形式的解决方案。

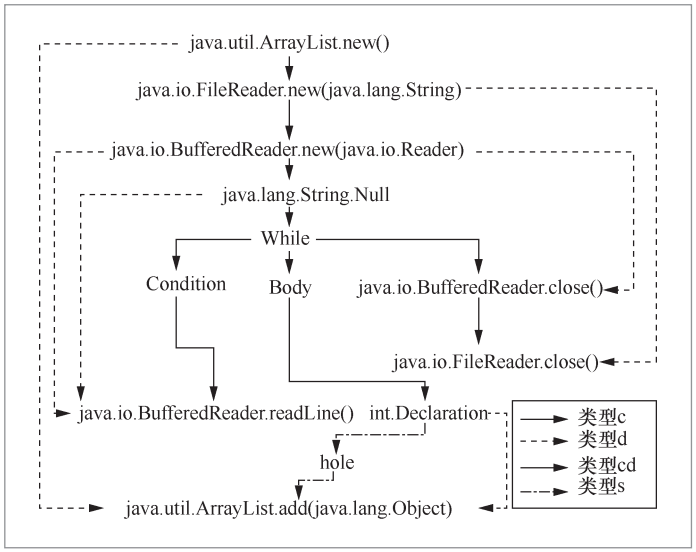

图 12 API 上下文图示例

5 结束语

以代码为核心的软件开发大数据的积累以及各种智能化分析技术的发展，使得智能化的代码复用推荐成为可能。通过代码模板挖掘以及代码深度学习等不同的智能化分析技术，可以实现面向特定领域以及与领域无关的智能化代码复用推荐。然而，现有的方法和工具大部分只对与领域无关的通用代码单元较为有效，如通用API使用模式、常用算法实现等。支持特定领域代码复用的代码模板挖掘等方法的适应性和可组合性都十分有限。上述问题导致现有的方法和工具仅能在简单的编码任务上提供一些有限的智能化支持，无法提供全面、系统化的智能化软件开发支持。

未来有效的智能化代码复用推荐应当以一种人机协作的智能化助手的形式来实现，通过持续观察开发人员的行为以及代码上下文的变化，及时发现问题，并在需要的时候提供各种帮助。这种智能化开发助手需要与软件开发人员持续交互和协作，并具备交互式澄清与解释、逐步细化解决方案建议、获取和利用技术和业务背景知识、按需提供不同粒度和不同形式的解决方案等方面的能力。

参考文献:

[1] NGUYEN A T, NGUYEN T T, NGUYEN H A, et al. Graph-based pattern-oriented, context-sensitive source code completion[C]// The 2012 34th International Conference on Software Engineering. Piscataway: IEEE Press, 2012: 69-79.

[2] ALLAMANIS M, SUTTON C. Mining source code repositories at massive scale using language modeling[C]// The 2013 10th Working Conference on Mining Software Repositories. Piscataway: IEEE Press, 2013.

[3] NGUYEN A T, NGUYEN T N. Graph-based statistical language model for code[C]// The 2015 IEEE/ACM 37th International Conference on Software Engineering. Piscataway: IEEE Press,

2015: 858–868.

[4] LIN Y, PENG X, CAI Y F, et al. Interactive and guided architectural refactoring with search-based recommendation[C]// The 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2016: 535–546.

[5] LIN Y, XING Z C, XUE Y X, et al. Detecting differences across multiple instances of code clones[C]// The 36th International Conference on Software Engineering. New York: ACM Press, 2014: 164–174.

[6] LIN Y, MENG G Z, XUE Y X, et al. Mining implicit design templates for actionable code reuse[C]// The 2017 32nd IEEE/ACM International Conference on Automated Software Engineering. Piscataway: IEEE Press, 2017: 394–404.

[7] HINDLE A, BARR E T, SU Z D, et al. On the naturalness of software[C]// The 34th International Conference on Software Engineering. New York: ACM Press, 2012: 837–847.

[8] NGUYEN A T, HILTON M, CODOBAN M, et al. API code recommendation using statistical learning from fine-grained changes[C]// The 2016 ACM 24th SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2016: 511–522.

[9] TU Z P, SU Z D, DEVANBU P T. On the localness of software[C]// The 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2014: 269–280.

[10] NGUYEN T T, NGUYEN A T, NGUYEN H A, et al. A statistical semantic language model for source code[C]// The 2013 9th Joint Meeting of the European Software Engineering Conference. New York: ACM Press, 2013: 532–542.

[11] RAYCHEV V, VECHEV M T, YAHAV E. Code completion with statistical language models[C]// The 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 2014: 419–428.

[12] DAM H K, TRAN T, PHAM T. A deep language model for software code[J]. arXiv preprint, 2016, arXiv:1608.02715.

[13] WHITE M, VENDOME C, VÁSQUEZ M L, et al. Toward deep learning software repositories[C]// The 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. Piscataway: IEEE Press, 2015: 334–345.

[14] NGUYEN A T, NGUYEN T D, PHAN H D, et al. A deep neural network language model with contexts for source code[C]// The 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering. Piscataway: IEEE Press, 2018: 323–334.

[15] YAN J P, QI Y, RAO Q F, et al. Learning API suggestion via single LSTM network with deterministic negative sampling[C]// The 2018 International Conference on Software Engineering and Knowledge Engineering. [S.l.:s.n.], 2018.

[16] CHEN C, PENG X, SUN J, et al. Generative API usage code recommendation with parameter concretization[J]. Science China Information Sciences, 2019, 62(9): 192103.

[17] TAI K S, SOCHER R, MANNING C D. Improved semantic representations from tree-structured long short-term memory networks[C]// The 2015 Annual Meeting of the Association for Computational Linguistics, the International Joint Conference on Natural Language, the Asian Federation of Natural Language Processing. Stroudsburg: ACL Press, 2015: 1556–1566.

[18] LIU X Y, HUANG L G, NG V. Effective API recommendation without historical software repositories[C]// The 2018 ACM/IEEE International Conference

on Automated Software Engineering. Piscataway: IEEE Press, 2018: 282–292.
[19] CHEN C, PENG X, XING Z C, et al. Holistic combination of structural and textual code information for context based API recommendation[J]. arXiv preprint, 2020, arXiv:2010.07514.
[20] PENG X, XING Z C, SUN J. AI-boosted software automation: learning from human pair programmers[J]. Science China Information Sciences, 2019, 62(10): 200104.

作者简介

彭鑫（1979– ），男，博士，复旦大学教授、计算机科学技术学院副院长、软件学院副院长。中国计算机学会软件工程专业委员会副主任，*Journal of Software*: *Evolution and Process*联合主编，*ACM Transactions on Software Engineering and Methodology*编委，《软件学报》编委，*Empirical Software Engineering*编委，IEEE软件维护与进化国际会议（ICSME）执行委员（2017—2020年）。2016年获得NASAC青年软件创新奖。主要研究方向为软件开发大数据分析、智能化软件开发、云原生与智能化运维、泛在计算软件系统等。

陈驰（1993– ），男，复旦大学计算机科学技术学院博士生，主要研究方向为智能化软件开发。

林云（1988– ），男，博士，新加坡国立大学计算机学院高级博士后研究员，主要研究方向为软件调试技术、软件测试、代码推荐与代码重构。

收稿日期： 2020-09-22
通信作者： 彭鑫，pengxin@fudan.edu.cn
基金项目： 国家重点研发计划基金资助项目（No. 2016YFB1000800）
Foundation Item: The National Key Research and Development Program of China (No.2016YFB1000800)