文章编号:2096-1472(2023)09-0012-06
DOI:10.19644/j.cnki.issn2096-1472.2023.009.003

基于类原型与深度学习的类注释生成方法

李 睿[1],赵逢禹[2],刘 亚[1]

(1.上海理工大学光电信息与计算机工程学院,上海 200093;
2.上海出版印刷高等专科学校信息与智能工程系,上海 200093)
✉ lrui1999@163.com; zhaofengyv@usst.edu.cn; liuya@usst.edu.cn

摘 要:现有的代码注释生成技术大多针对方法粒度,而对于面向对象程序,类才是其核心组成,因此对类生成注释是很有必要的。针对这一问题,提出一种结合类原型与深度学习技术对类生成注释的方法。首先,确定类原型并选择对应类注释模板;其次,提取类中信息填充模板,对类中的方法通过双编码器模型训练得到方法代码注释。实验结果表明,方法粒度上提出的双编码器模型在方法代码注释生成的结果评估中表现较好,类粒度的注释准确性较高。

关键词:代码注释;类注释模板;类原型;双编码器;深度学习
中图分类号:TP311 **文献标志码**:A

Class Annotation Generation Method Based on Class Prototype and Deep Learning

LI Rui[1], ZHAO Fengyu[2], LIU Ya[1]

(1.*School of Optical-Electrical & Computer Engineering, University of Shanghai for Science and Technology, Shanghai* 200093, *China*;
2. *Department of Information and Intelligent Engineering, Shanghai Publishing and Printing College, Shanghai* 200093, *China*)
✉ lrui1999@163.com; zhaofengyv@usst.edu.cn; liuya@usst.edu.cn

Abstract: Most of the existing code annotation generation techniques are targeted at method granularity. For object-oriented programs, classes are their core components, so it is necessary to generate annotations for classes. To solve this problem, this paper proposes a class annotation generation method combining class prototype and deep learning technology. Firstly, the class prototype is determined and the corresponding class annotation template is selected. Secondly, information in the class is extracted to fill the template, and the method code annotation is obtained by training the bi-encoder model for the methods in the class. The experimental results show that the proposed bi-encoder model in terms of method granularity performs better in the result evaluation score of method code annotation generation, and the annotation accuracy of class granularity is higher.

Key words: code annotation; class annotation template; class prototype; bi-encoder; deep learning

0 引言(Introduction)

在许多软件系统中,代码源文件的注释经常出现不完整、过时甚至缺失等情况,对这些系统代码进行维护与完善时,开发人员需要花费大量时间理解代码[1]。当代码内部有较准确的注释文本时,开发人员可以通过浏览注释文本快速理解相关代码片段的含义,从而节省维护时间,提高维护质量。

自动代码注释生成技术是用于提高代码可读性和可维护性的有效方法,目前针对代码注释自动生成的研究大致分为以下两类:基于模板的注释自动生成和基于深度学习的注释自动生成。基于模板自动生成的代码注释可读性较高,但缺乏对具体代码结构信息的获取能力,而基于深度学习的代码注释生成研究大多应用于方法粒度,对于复杂的类结构的注释生成比较

收稿日期:2023-02-20
基金项目:"十三五"国家密码发展基金理论项目(MMJJ20180202)

困难。

针对上述两类自动代码注释生成技术存在的问题,本文结合模板与深度学习技术,提出一种为类自动生成注释的方法,首先确定当前类所属的类原型,并选择对应类注释模板,提取类中的信息填充到模板中,其次针对类中方法的注释生成问题,提出构建一种基于注意力机制的双编码器模型,实现代码到注释的映射,最后通过实验验证了本方法的可行性与实际应用效果。

1　相关研究(Related research)

基于模板的注释生成方法,首先需要预定义一组启发式规则,其次提取代码的关键信息并通过启发式规则生成自然语言描述。HILL 等[2]提出一种基于软件单词使用模型(Software Word Usage Model,SWUM)分析 Java 方法签名的方法,为 Java 方法生成自然语言注释。SRIDHARA 等[3]在 HILL 等研究的基础上进一步考虑了方法体内含有的代码,提出一种基于启发式规则的方法为方法代码生成注释,该方法主要分为内容选择和注释生成两个阶段。MORENO 等[4]将关注的代码粒度从方法级别提升到类级别,首先确定类和方法的原型,其次通过结合原型信息与预先定义的基于方法和数据成员访问级别的启发式规则,并通过现有的文本生成工具对 Java 类生成描述性摘要。基于规则模板生成的代码注释容易理解,但受限于模板只能为特定的代码结构生成注释,对于模板之外的代码无法灵活地生成有效的代码注释。

基于深度学习的注释生成方法通过神经网络对代码中包含的信息进行挖掘,得到其特征向量表示,并通过模型进行训练,最终得到自然语言描述。目前,大量的研究主要采用神经机器翻译(Neural Machine Translation,NMT)模型,通过提取不同的代码特征信息,使用不同的神经网络结构进行训练得到代码注释。IYER 等[5]率先将深度神经网络引入自动代码摘要研究,提出代码-描述嵌入神经网络(Code-Description Embedding Neural Network,CODENN)方法,在长短期记忆网络(Long-Short Term Memory,LSTM)的基础上通过引入注意力机制构建一个端到端的学习网络,从而自动生成代码摘要。HUANG 等[6]采用门控循环单元(Gated Recurrent Units,GRU)作为编码器和解码器,并为代码块生成注释,GRU 是对 LSTM 网络的一种改进,能够保留较长的上下文信息[7]。CAI 等[8]提出使用树形长短期记忆网络(Tree-LSTM),通过引入语法类型信息对程序代码进行结构化编码,并在解码时采用多种策略对抽象语法树的节点进行筛选,最终生成代码注释。基于深度学习的自动代码注释方法可以较灵活地为代码片段生成注释,在面对复杂程序代码时,仍然具有良好的结构学习能力,但目前大多只应用于方法粒度,针对类粒度的代码片段无法生成较为全面的注释。

2　类注释生成方法(Class annotation generation method)

本文提出的基于类原型与深度学习为类生成注释的方法活动图如图 1 所示,主要包括以下几步:第一步,提取类中方法,确定类中方法原型,并通过类中方法原型的分布确定类原型;第二步,为不同的类原型选择不同的类注释模板;第三步,针对类注释模板中各部分的信息设计相应的提取规则,并将提取到的信息填充到类注释模板中;第四步,对于类中的主要方法通过深度学习模型得到方法注释,将方法代码注释填充到类注释模板中,得到完整类注释。

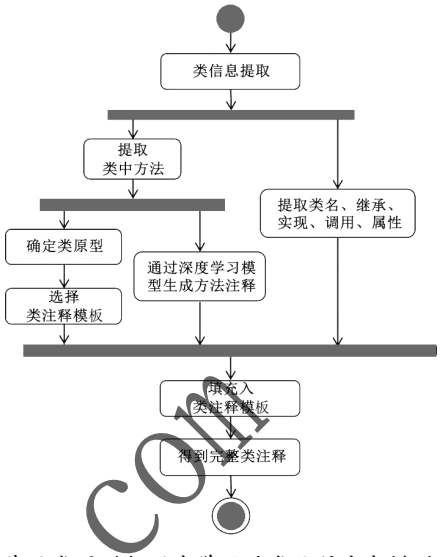

图 1　基于类原型与深度学习的类注释生成模型活动图

Fig. 1 Activity diagram of class annotation generation model based on class prototype and deep learning

2.1　类原型分类

类原型是项目设计中类的角色和责任的简单抽象,开发人员可以通过类原型较快地了解类在系统中的一般职责。在软件的维护和发展过程中,对类原型的准确描述是必要的,STARON 等[9]验证了类原型信息在程序理解任务中的有效性。本文首先对类的原型进行识别,确定类的职责,其次为不同的类原型选择不同的类注释模板。

目前,在类与方法的原型识别领域已有大量研究,文献[10]首先提出了一种方法原型的自动识别技术,并给出了方法原型的分类法;文献[11]在此基础上提出一种根据方法原型分布确定类原型的方法,并给出了类原型的分类法。为了构建符合类职责的注释模板,本文参照文献[11]对方法原型和类原型的分类方法进行重新定义,将方法原型分类为 5 类,类原型分类为 4 类,其中方法原型分布是类原型分类的重要特征。表 1 给出了本文定义的方法原型分类及原型描述,表 2 中给出了本文定义的类原型分类、原型描述及分类规则。

表 1　方法原型分类

Tab.1 Method prototype taxonomy

方法原型分类	原型描述
accessor	访问器方法:返回本地字段的相关信息
mutator	变换器方法:更改本地字段
constructor	构造方法:在创建对象时调用
collaborator	协作方法:将对象与其他类型对象连接处理
controller	控制器方法:更改外部对象状态的逻辑处理方法

表2 类原型分类描述

Tab.2 Taxonomy and description of class prototype

类原型分类	原型描述	分类规则
Entity	封装数据和行为的实体，由访问器、变换器、构造方法和少数协作方法组成，不包括控制器方法	同时满足以下三条规则： 规则1为 $acc+mut+con>\frac{4}{5}mtd$； 规则2为 $col<\frac{1}{5}mtd$； 规则3为 $con=0$
Boundary	在系统中负责通信的类，具有较多协作方法、较少的控制器方法和构造方法	同时满足以下两条规则： 规则1为 $col>\frac{2}{3}mtd$； 规则2为 $cont+con<\frac{1}{3}mtd$
Controller	负责提供功能处理外部对象数据的类，主要由控制器方法和构造方法组成	满足以下一条规则： 规则1为 $cont+con>\frac{4}{5}mtd$
Large class	具有较多功能的类，由访问器、变换器、构造方法和协作方法组成	满足以下任意一条规则： 规则1为 $acc\neq0$，$mut\neq0$，$con\neq0$，$col\neq0$； 规则2为当前类不属于{Entity，Boundary，Controller}

在表2的类原型分类规则中，acc 表示类中访问器类型方法的数量，mut 表示类中变换器类型方法的数量，con 表示类中构造方法的数量，col 代表类中协作方法的数量，$cont$ 表示类中控制器类型方法的数量，mtd 表示类中方法的总数。

2.2 类注释模板

不同类原型代表不同的职责，代码注释主要关注的信息也不同，例如Entity类型代表实体类，重点关注继承关系、类中属性以及类在项目中的使用情况，由于Entity类中方法大多数为访问器、变换器及创建方法，因此不需要对类中方法进行注释；Boundary类型代表边界类，多用于与其他类进行通信，因此重点关注类在项目中的使用情况和类中方法的功能；Controller类型代表控制类，主要负责处理外部对象数据，重点关注继承关系、实现关系、类在项目中的使用情况和类中方法的功能。受篇幅限制，本文以Controller类原型为例，展示其注释模板（如表3所示）。

表3 Controller类原型注释模板

Tab.3 Controller class prototype annotation template

Controller类原型注释模板
该类为<类名>的Controller类
该类的父类为<父类>
该类具有以下子类：<子类$_1$>，…，<子类$_n$>
该类实现了以下接口：<接口$_1$>，…，<接口$_m$>
该类在以下类中被调用：<类$_1$>，…，<类$_k$>
该类主要功能为<方法$_1$ 注释>，…，<方法$_s$ 注释>

2.3 类中信息提取规则

表4给出了类注释模板中各部分注释内容的提取规则。首先，通过代码抽象语法树（Abstract Syntax Tree，AST）结构提取父类、接口的实现类和类中属性；其次，在项目中进行代码搜索提取其子类及该类在其他类中的使用情况，将上述信息填充到预定义的类注释模板中。类中方法是类功能的具体实现，类中方法的注释生成是类注释的难点。针对这一问题，本文首先识别类中的主要方法，对主要方法使用深度学习的技术建立方法代码到代码注释的映射模型，通过该模型获得方法代码的注释并填充到类注释模板中，最终获得完整的类注释。

表4 类中信息提取规则

Tab.4 Rules for extracting information from classes

类中信息	提取规则
类名	分析AST
继承关系（父类）	分析AST
继承关系（子类）	在项目中使用代码搜索
实现关系	分析AST
使用情况	在项目中使用代码搜索
类中属性	分析AST
类中方法	通过深度学习训练得到方法代码注释

3 基于注意力机制的双编码器方法代码注释生成模型（A model for code annotation generation in bi-encoder method based on attention mechanism）

由于方法代码结构复杂，包含较多的序列信息和结构信息，因此仅使用预定义规则提取，无法为方法代码生成较为准确的注释，而深度学习可以捕捉代码的结构信息，并可以通过大量训练得到从代码到注释的映射，本文使用深度学习技术为方法代码生成代码注释。本文在经典编码器-解码器模型的基础上，提出了一种双编码器-解码器的方法代码注释生成模型，其中双编码器包括一个序列编码器和一个树结构编码器，图2展示了该模型网络结构。该模型首先对数据进行预处理，在源码文件中提取以方法为粒度的代码，构建算法提取代码序列和代码抽象语法树结构信息；其次分别将代码序列和抽象语法树序列的向量表示输入各自对应的编码器中进行训练；最后经过注意力机制关注解码重点，将序列向量和抽象语法树向量进行拼接后，由解码器解码得到方法代码注释序列。

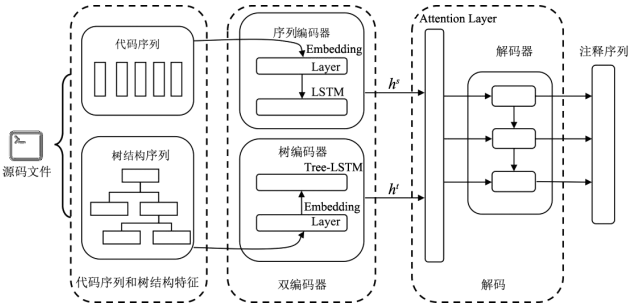

图2 双编码器-解码器模型

Fig. 2 Dual encoder and decoder model

3.1 序列编码器

在经典编码器-解码器模型中,编码器通常使用循环神经网络(RNN)按照时序提取代码序列特征,据目前已有的研究可知,当序列较长时,基于递归机制的RNN可能存在梯度消失或者梯度爆炸问题。GERS等[12]提出了一种长短期记忆网络(LSTM),相对于RNN而言,LSTM在处理长序列数据时表现更为出色。因此,本文选择LSTM作为双编码器中的序列编码器处理代码序列信息。

为了构造序列编码器中的输入方法代码序列,首先利用Antlr4语法分析器生成工具将方法代码转化为抽象语法树(AST),其次对方法AST进行前序遍历,组成代码序列。算法1给出了提取方法代码序列的算法。

算法1:方法代码序列提取算法。

输入:方法代码 Method。

输出:方法代码对应的 token 序列。

处理:

1. 初始化 List={};初始化 Stack;
2. Root=AST(Method);//将方法代码转为抽象语法树
3. Stack. push(Root);//根节点入栈
4. while Stack 不为空 do
    node=Stack. pop();
    List. add(node);
    child=Collections. reverse(node. children);//逆序该节点的子节点
    Stack. addAll(child);//添加所有的逆序子节点
   end while
5. 输出代码序列 List 的字符串,结束算法。

通过上述算法将方法代码转为token序列后,使用词嵌入将token序列转换为向量表示 $\boldsymbol{X}^s=(\boldsymbol{x}_1^s,\cdots,\boldsymbol{x}_n^s)$,其中 s 代表序列信息,$\boldsymbol{x}_n^s$ 代表方法片段中第 n 个 token 的向量表示。将向量 $\boldsymbol{X}^s$ 作为序列编码器的输入,经LSTM训练后输出表示向量,在图2中表示为序列向量 $\boldsymbol{h}^s$。

3.2 树结构编码器

本文使用树形长短期记忆网络(Tree-LSTM)作为树结构编码器处理代码结构信息,Tree-LSTM能够自下而上地学习代码的抽象语法树结构,有效地避免了代码结构信息丢失的问题[13]。相对于LSTM,Tree-LSTM门控向量和记忆细胞的更新取决于子节点的状态,对于给定AST的任意节点 j,$C(j)$ 表示其子节点集合,树编码器通过输入节点向量 $\boldsymbol{x}_j^t$ 和其子节点的隐藏状态 $\boldsymbol{h}_{C(j)}^t$ 计算并输出对应的隐藏状态 $\boldsymbol{h}_j^t$,其计算公式如公式(1)所示:

$$\boldsymbol{h}_j^t=f(\boldsymbol{x}_j^t,\boldsymbol{h}_{C(j)}^t) \tag{1}$$

其中,t 代表树结构信息。

为了提取方法代码的树结构信息,首先将方法代码表示为抽象语法树(AST),其次对方法AST进行后序遍历的同时记录其代码层级关系,得到描述代码树形结构中子节点和父节点关系的序列。算法2展示了提取方法代码树结构序列的算法。

算法2:方法代码提取树结构序列算法。

输入:方法代码 Method。

输出:方法代码对应的树结构序列。

处理:

1. 初始化 List={<node, nodeDepth>};初始化 Stack;
2. Root=AST(Method);//将方法代码转为抽象语法树
3. Stack. push(Root);//根节点入栈
4. while Stack 不为空 do
    node=Stack. pop();
    currentDepth=Depth(node);//获取当前节点的最大深度
    //将当前节点所有子节点入栈
    node. children. forEach(child->{
        Stack. push(child);
    });
    List. addFirst(<node, currentDepth>);
   end while
5. 输出描述代码树结构的 List,结束算法。

通过自底向上的遍历,最终获取根节点的输出作为树编码器的输出,在图2中通过结构向量 $\boldsymbol{h}^t$ 表示该向量。

3.3 解码器

本文采用融合注意力机制的LSTM作为解码器,首先将序列编码器输出的序列表示向量 $\boldsymbol{h}^s$ 和树编码器输出的结构表示向量 $\boldsymbol{h}^t$ 分别通过注意力机制进行加权求和重新计算权重。其次将注意力机制筛选后的向量进行拼接,得到综合编码向量作为解码器的输入向量。至此,解码器就同时融合了代码的序列信息和结构信息。

4 实验结果与分析(Experimental results and analysis)

为了全面评估本文提出的基于类原型与深度学习为类自动生成注释的方法,将从以下两个方面进行实验验证:一是评估本文提出的基于注意力机制的双编码器方法代码注释生成模型的性能;二是评估本文提出的结合类原型与深度学习的类粒度注释生成的准确性。

4.1 数据集

本文采用文献[14]提供的Java-med数据集进行实验,该数据集收集于软件项目托管平台GitHub,包括1 000个Java顶级项目。该数据集包含约400万个方法,涵盖模型需要的源代码信息和注释信息。本文将数据集按照8∶2的比例划分训练集和测试集,其中训练集用于训练双编码器-解码器模型,测试集用于评估双编码器-解码器模型性能及Java类注释生成效果。

4.2 评估指标

本文从两个方面评估本文提出方法的应用效果。对于方法代码注释自动生成的双编码器-解码器模型的评估问题,采用BLEU[15]、ROUGE[16]和METEOR[17]评估生成方法代码注释的质量,这些指标广泛应用于机器翻译领域。其中,BLEU表示生成句子和真实句子的相似程度,更侧重于准确率,本文选择其中

的 BLEU-4 作为评估指标；ROUGE 与 BLEU 类似，但更加关注召回率，ROUGE-L 基于生成句子和真实句子的最长公共子序列共现统计召回率和准确率；METEOR 引入了同义词匹配，在计算得分时考虑了词性变换和同义词的情况。三种机器翻译性能评估指标与生成的自然语言注释质量呈正相关。

针对类粒度注释的准确性与完整性评估问题，由于类粒度注释生成的相关研究与通用数据集较少，因此本实验采用具有 Java 软件领域开发经验的工程师对类注释进行人工方式的准确性评估。

4.3 实验结果分析

4.3.1 方法粒度的注释生成模型性能对比

本文提出的双编码器-解码器模型基于 Python 编程语言和 PyTorch 深度学习框架实现，在训练过程中，使用 Adam 作为优化器，初始学习率设置为 0.001。其中，双编码器和解码器均具有两个隐藏层，隐藏状态设为 256 维，嵌入单词的维度也设为 256 维。

为了评估本文在方法粒度的注释自动生成中提出的双编码器-解码器模型的有效性，选用方法注释生成领域中常见的注释生成模型进行实验对比。

(1)Seq2Seq 模型＋注意力机制[5]。Seq2Seq 模型将源代码序列信息输入 LSTM 中进行编码，引入注意力机制，最终由解码器进行解码获得注释。

(2)Code2Seq 模型＋注意力机制[14]。Code2Seq 模型通过使用抽象语法树中组合路径的集合表示代码片段，使用 BiLSTM 作为编码器，并在解码时使用注意力机制选择相关的路径最终解码获得注释序列。

(3)Tree2Seq 模型＋注意力机制[18]。Tree2Seq 模型使用 Tree-LSTM 作为编码器，将源代码转化为抽象语法树结构，使用编码器自下而上递归对 AST 节点进行编码，然后由基于注意力机制的解码器进行解码。

表 5 展示了 4 种模型在选定的 Java 数据集上的实验评估结果对比。从表 5 中可以看到，本文提出的双编码器-解码器模型的 *BLEU*-4 分数达到 35.2，并且在各项指标上均优于其他三种对照方法。相对于其他三种模型，本文模型既融合了代码的序列信息，又融合了代码的结构信息，同时引入了注意力机制(Attn)，能充分地获取代码信息，进而提升了注释生成效果。

表 5 不同模型的评估指标得分对比

Tab.5 Comparison of evaluation index scores of different models

模型	*BLEU*-4	*ROUGE-L*	*METEOR*
Seq2Seq＋Attn	31.8	48.1	21.6
Code2Seq＋Attn	33.6	49.4	21.8
Tree2Seq＋Attn	34.8	51.6	22.5
本文模型	**35.2**	**51.8**	**23.8**

4.3.2 类粒度的注释生成性能评估

为了评估本文结合类原型与深度学习生成类注释的性能，从类原型识别的准确性及类注释的准确性两个方面进行评估。首先在测试数据集中选择部分类，通过人工识别方式进行类原型分类；其次使用本文提出的类原型识别方法对这些类进行原型识别。从表 6 可以看出，本文提出的类原型识别规则能较准确地识别类的原型。

表6 类注释生成性能评估结果

Tab.6 Evaluation results of class annotation generation performance

类型	实体类	边界类	控制类	大类
人工识别数量/个	49	56	46	60
本文方法识别数量/个	42	51	39	58
类注释准确性/%	90	86	80	82

为了评估生成类注释的准确性，评估者阅读选定类的主要功能和结构并给出对相应类的注释描述，然后与本文生成的类注释进行对比。表 6 展示了对不同类原型的类注释准确性的评估，评估者认定当类原型为实体类和边界类时，本文方法能较准确地生成符合该类职责的注释，对于类中方法数量较少的控制类能够较为准确地描述类的使用情况，但对于类的功能描述信息较少，对于大类(Large class)能较为准确地描述类中具体方法的功能，从而反映类的功能。

4.4 示例展示

本文首先通过确定类的原型选择类注释模板，其次提取类中信息填充到模板中，最终生成完整的类注释，表 7 展示了实验中类原型为 Controller 类最终生成的类注释结果。

表7 实验结果

Tab.7 Experimental results

名称	结果

源代码	public class GwtController implements ControllerManager {
	private int index;
	private String name;
	public boolean instance(ControllerListener cl, Collection
	< ControllerListener > arrays){
	for(ControllerListener c: arrays){
	if (c.isInstance(cl))
	return true;
	}
	return false;
	}}

类注释	该类为 GwtController 的 controller 类； 该类实现了以下接口：ControllerManager； 该类在以下类中被调用：GwtHandler； 该类主要功能为 if the ControllerListener is registered return true

5 结论(Conclusion)

本文通过结合类原型与深度学习技术，提出一种为类生成注释的方法，首先通过确定类原型选择类注释模板，其次提取类中的信息填充模板，对于类中的方法使用带有注意力机制的双编码器模型训练后得到方法注释，最终得到完整的类代码注释。实验表明，本文在方法粒度中提出的双编码器模型在性能

和效果上优于对比模型,完整的类粒度的注释经人工评估后确定具有较高的准确性。未来,可以结合信息搜索以及其他深度学习方法,融合更多的代码信息与注释文本,进一步提高自动生成类注释的质量。

参考文献(References)

[1] SONG Q W,KONG X L,WANG L L,et al. An empirical investigation into the effects of code comments on issue resolution[C]//HOSSAIN S. 2020 IEEE 44th Annual Computers, Software, and Applications Conference. New York: IEEE,2020:921-930.

[2] HILL E,POLLOCK L,VIJAY-SHANKER K. Automatically capturing source code context of NL-queries for software maintenance and reuse[C]//NARAYANR. 2009 IEEE 31st International Conference on Software Engineering. New York:IEEE,2009:232-242.

[3] SRIDHARA G,HILL E,MUPPANENI D,et al. Towards automatically generating summary comments for Java methods[C]//MAREK G, ADAM C. Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering. New York:IEEE,2010:43-52.

[4] MORENO L,APONTE J,SRIDHARA G,et al. Automatic generation of natural language summaries for Java classes[C]//DIANA D. 2013 21st International Conference on Program Comprehension. stroudsbug:IEEE,2013:23-32.

[5] IYER S,KONSTAS I,CHEUNG A,et al. Summarizing source code using a neural attention model[C]//MARIA B. JOACHIM B. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. Strousbug: Association for Computational Linguistics,2016:2073-2083.

[6] HUANG Y,HUANG S H,CHEN H C,et al. Towards automatically generating block comments for code snippets[J]. Information and Software Technology,2020,127:106373.

[7] LIANG Y D,ZHU K. Automatic generation of text descriptive comments for code blocks[J]. Proceedings of the AAAI Conference on Artificial Intelligence,2018,32(1):5229-5236.

[8] CAI R C,LIANG Z H,XU B Y,et al. TAG:type auxiliary guiding for code comment generation[DB/OL]. (2020-05-06)[2023-02-22]. https://arxiv.org/abs/2005.02835.

[9] STARON M,KUZNIARZ L,WOHLIN C. Empirical assessment of using stereotypes to improve comprehension of UML models:a set of experiments[J]. Journal of Systems and Software,2006,79(5):727-742.

[10] DRAGAN N,COLLARD M L,MALETIC J I. Reverse engineering method stereotypes[C]//TIBOR B. 2006 22nd IEEE International Conference on Software Maintenance. New York:IEEE,2006:24-34.

[11] DRAGAN N,COLLARD M L,MALETIC J I. Automatic identification of class stereotypes[C]//MALCOM G. Proceedings of the 2010 IEEE International Conference on Software Maintenance. New York:IEEE,2010:1-10.

[12] GERS F A,SCHMIDHUBER J,CUMMINS F. Learning to forget: continual prediction with LSTM[J]. Neural Computation,2000,12(10):2451-2471.

[13] 徐少峰,潘文韬,熊赟,等. 基于结构感知双编码器的代码注释自动生成[J]. 计算机工程,2020,46(2):304-308,314.

[14] ALON U,BRODY S,LEVY O,et al. code2seq:generating sequences from structured representations of code[DB/OL]. (2019-02-21)[2023-02-22]. https://arxiv.org/abs/1808.01400.

[15] PAPINENI K,ROUKOS S,WARD T,et al. BLEU:a method for automatic evaluation of machine translation[C]//CLARK A. Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. New York: ACM,2002:311-318.

[16] LIN C Y. Rouge:A package for automatic evaluation of summaries[C]//BEN H,CLAIRE G. Text Summarization Branches Out. New York:Association for Computational Linguistics,2004:74-81.

[17] BANERJEE S,LAVIE A. METEOR:An automatic metric for MT evaluation with improved correlation with human judgments[C]//GABRIEL M,JOHANNA D. Proceedings of the Acl Workshop on Intrinsic and Extrinsic evaluation Measures for Machine Translation and/or Summarization. New York: Association for Computational Linguistics,2005:65-72.

[18] ERIGUCHI A,HASHIMOTO K,TSURUOKA Y. Tree-to-sequence attentional neural machine translation[DB/OL]. (2016-06-08)[2023-02-22]. https://arxiv.org/abs/1603.06075.

作者简介:

李　睿(1999-),女,硕士生。研究领域:自然语言处理,深度学习。

赵逢禹(1963-),男,博士,教授。研究领域:计算机软件与软件系统安全,软件工程与软件质量控制,软件可靠性。

刘　亚(1983-),女,博士,副教授。研究领域:信息安全,密码学。