一种基于主题建模的代码功能挖掘工具

华哲邦[1,2] 李 萌[1,2] 赵俊峰[1,2] 邹艳珍[1,2] 谢 冰[1,2] 李 扬[3]

(北京大学信息科学技术学院 北京 100871)[1] (高可信软件技术教育部重点实验室 北京 100871)[2]

(神州数码信息系统有限公司 北京 100085)[3]

摘 要 代码复用是重要的软件复用方式之一,复用者需要理解软件代码实现的功能方能有效实施软件复用。基于主题建模技术的程序理解方法逐渐受到研究人员的重视,它能够帮助软件开发者和使用者更好地理解软件的功能。目前,基于主题建模技术的程序理解方法一般欠缺对挖掘出的Topic的语义分析,为此提出的基于代码静态分析和LDA技术的代码功能挖掘(Code Function Mining,CFM)方法可作为对这类方法的补充。CFM是一套以代码为研究对象的挖掘、筛选、组织和描述主题(Topic)的方法,该方法能够生成带描述的功能型Topic的层次结构,以供使用者更清晰和方便地浏览、学习软件的功能。功能型Topic的描述能够帮助复用者理解代码功能,其层次结构能够让复用者从不同抽象层次理解代码功能。CFM方法包括4个部分:挖掘Topic、筛选Topic、组织Topic、描述Topic。以CFM方法为基础,设计并实现了一个CFM工具。CFM工具能够分析用户提交的代码,通过Web页面向用户展示带描述的功能型Topic的层次结构。最后,对CFM方法中的几个关键算法进行实验分析,验证了CFM方法的有效性。

关键词 软件代码,代码静态分析,LDA,代码功能挖掘

中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.09.008

Code Function Mining Tool Based on Topic Modeling Technology

HUA Zhe-bang[1,2] LI Meng[1,2] ZHAO Jun-feng[1,2] ZOU Yan-zhen[1,2] XIE Bing[1,2] LI Yang[3]

(School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)[1]

(Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, China)[2]

(Digital China Information System Co., Ltd, Beijing 100085, China)[3]

Abstract Code reuse is an important way of software reuse. Software engineers need to understand the code functions before they reuse the code. A Code Function Mining (CFM) method based on static code analysis and LDA technologies was proposed. CFM method is a code-oriented method for mining, filtering, organizing and describing topics. The output of CFM method is a hierarchy structure of functional topics with descriptions. Topic descriptions can help better learn code functions and the hierarchy structure can help better understand code functions from different abstraction levels. CFM method can be used as a supplement of traditional methods based on topic modeling technology to make up for the lack of semantic analysis of topics. CFM method includes four parts: Topic Mining, Topic Filtering, Topic Organizing, Topic Describing. A CFM tool based on CFM method can automatically analyze code and show the function topic hierarchy to users through Web page. To verify the validity of CFM method, the experimental analysis was also presented on several key algorithms in it.

Keywords Software code, Static code analysis, LDA, Code function mining

1 引言

软件复用是在软件开发中避免重复劳动的解决方案。通过软件复用,可以提高软件开发的效率和质量[1]。软件复用方式可分为产品复用和过程复用,本文主要研究代码复用方式的软件复用。

复用者在进行代码复用之前,首先需要理解软件代码实现的功能。复用者可通过阅读软件的代码或相关文档的方式来理解功能。虽然文档更容易被复用者理解,但是对于很多软件来说,其相关文档存在缺失、版本过时等问题[2]。文档缺失是较为普遍的问题,特别是近年来,随着敏捷软件开发方法的流行,很多软件不再具备详细的相关文档。文档版本过时也是不可忽视的问题。更改了代码,而没有更新所对应的文档,往往会导致复用者在代码复用的过程中出现偏差。对于软件来说,代码是其功能的直接载体。因此,阅读代码是复用者理解软件代码功能的基本途径之一。然而代码中包含了大

到稿日期:2013-12-25 返修日期:2014-01-16 本文受国家高技术研究发展计划(863计划)(2012AA01A403),国家自然科学基金(61121063),国家重点基础研究发展计划(973计划)(2009CB320703)资助。

华哲邦 男,硕士,主要研究方向为软件工程,E-mail:huazb1989@126.com;**李 萌** 男,博士生,主要研究方向为软件工程、构件技术等;**赵俊峰** 女,博士,副教授,主要研究方向为软件工程、知识工程、服务计算等;**邹艳珍** 女,博士,副教授,主要研究方向为软件工程与软件复用技术等;**谢 冰** 男,博士,教授,主要研究方向为软件工程、形式化方法等;**李 扬** 男,博士,主要研究方向为软件工程、普适计算等。

量的底层实现细节的信息，复用者很难依靠传统的代码静态分析方法（如数据流分析、控制流分析），从抽象层次上理解功能。

近些年，基于主题建模（Topic Modeling）技术的程序理解方法（以下简称 TM 方法）逐渐受到研究人员的重视[3-5]。主题建模技术最早出现于信息检索及自然语言处理领域，其核心是主题模型。主题模型是一个用于揭示文档集中语义结构的概率模型[6]。与传统的静态分析方法不同，TM 方法认为代码中的命名元素包含着软件开发人员的潜在知识，这对分析代码所代表的语义信息很有帮助。软件的代码也可被视为一种文档。TM 方法从代码中挖掘 Topic，将语义关联的代码聚集到一个 Topic 下，可从更高层次上来分析代码。早期研究 TM 方法的 Kuhn 等人列举了如下例子[3]：

```
Public Boolean isMorning(int hours,int minuts,int seconds) {
  If (! isDate(hours,minutes,seconds))
  throw Exception("Invalid input:not a time value. ");
  return hours < 12 && minutes < 60 && seconds < 60;
}
```

如果仅使用静态分析方法来分析该方法，仅仅考虑代码内部结构，而不考虑代码中每个命名元素代表的意思，复用者很难理解该方法的语义。TM 方法则考虑了代码中命名元素的含义。该例中复用者可以通过方法名“isMorning”和方法内部语句来理解该方法是一个“判断时间是否为上午”的方法，而“isMorning”是不被静态分析方法所关注的。与传统基于代码静态分析的程序理解方法相比，TM 方法具有抽象层次高、面向语义的特点，更贴近于复用者理解软件代码功能的方式。

虽然已经有不少工作使用了 TM 方法来帮助挖掘代码功能，但现有的研究工作通常都是依靠人工来推测 Topic 的类型和语义，这种方式费时费力，并且带有主观因素。为了挖掘出软件代码的功能，需要研究功能型 Topic 关联的代码之间的内在特性，区分出功能型 Topic 与其它类型的 Topic。另外，软件代码的功能存在不同的抽象层次，一个抽象层次高的功能包含若干抽象层次低的功能。研究功能型 Topic 之间的层次关系，对其进行合理的层次组织，将更能贴近复用者理解软件代码功能的方式。除此之外，TM 方法挖掘出的功能型 Topic 还存在欠缺语义描述的问题，这不利于复用者理解功能型 Topic 体现的代码功能。结合静态分析的技术则能更好地组织 TM 方法挖掘出来的功能，并为这些功能找出核心的类、方法以及描述文本。

本文主要工作是设计并实现一种自动化的代码功能挖掘（Code Function Mining，CFM）工具。CFM 工具能够为代码生成带描述的功能型 Topic 的层次结构，帮助复用者更好地理解代码的功能。

本文第 2 节介绍了已有的相关工作；第 3 节给出了 CFM 的解决方案与技术细节；第 4 节对该方法中相关算法进行了实验与分析；最后进行了总结和展望。

2 相关工作

2.1 LDA 模型及其算法的相关研究

LDA 是一种基于主题模型的方法。在主题模型中，文档集合蕴含着一组主题（Topic）。每个 Topic 由一组语义关联的词组成，而文档集合中的每一个文档则由这组 Topic 按照一定的概率分布生成。LDA 技术包括 LDA 模型及其算法。LDA 的模型如图 1 所示，文档集中的每个矩阵代表一个文档，矩阵中每一格代表一个单词出现的次数。每个文档被看作是由下面的 Topic 混合生成。LDA 算法的作用是从文档集中逆向挖掘出潜在的 Topic。

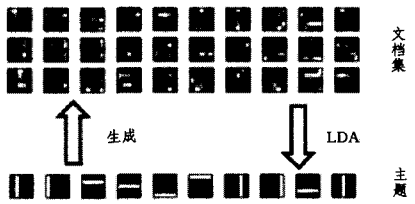

图 1 LDA 模型

在 LDA 模型中，输入是文档集，文档集中的每个文档是单词的集合。每个文档被看作是由一些 Topic 按照一定的概率分布生成的，而每个 Topic 被看作是由单词按照一定的概率分布构成的。设 D 为文档的总数量，W 为词汇表中单词的总数量，T 为 Topic 的总数量。有如下两个关联概率矩阵：

$T\times D:\Theta=(\theta_{td})$

$W\times T:\Phi=(\phi_{wt})$

LDA 模型假设上述两个概率分布均为 Dirichlet 分布，θ_{*d} 和 ϕ_{*t} 为带先验参数 α 和 β 的 Dirichlet 分布：$\theta_{*d}\sim$ Dirichlet(α)，$\phi_{*t}\sim$Dirichlet(β)。Dirichlet 分布是一个连续分布，它描述了事件发生概率的概率。数学模型如下：

给定 N 种事件 $e_1,e_2,\cdots,e_n$，猜测其发生的概率为 $x_1,x_2,\cdots,x_n$，且 $\sum_{i=1}^{n}x_i=1$。现在观测到一共有 m 个事件发生，事件 e_i 发生的次数为 α_i，且 $\sum_{i=1}^{n}\alpha_i=m$。那么，Dirichlet 分布的概率密度函数表示的则是每种事件 e_i 发生概率为 x_i 这种情况下的概率：

$$f(x_1,x_2,\cdots,x_n,\alpha_1,\alpha_2,\cdots,\alpha_n)=\frac{\prod_{i=1}^{n}\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{n}\alpha_i)}\prod_{i=1}^{n}x_i^{\alpha_i-1} \quad (1)$$

LDA 算法的目标是生成最佳的关联概率矩阵 Θ 和 Φ，所谓最佳的 Θ 和 Φ 是指在这组 Θ 和 Φ 下，文档集的似然值最大。似然值的计算公式如式(2)所示：

$$\text{Likelihood}=\prod_{i=1}^{D}p(d_i) \quad (2)$$

其中，$p(d_i)=\sum(\prod_{j=1}^{len_i}p(\theta_{*d_1})p(\varphi_{(w_{i,j)}},))$。上述两个公式中，$len_i$ 表示文档 d_i 包含的单词个数，$w_{i,j}$ 表示文档 i 中的第 j 个单词。公式的直观解释是，在指派了单词属于哪个 Topic 的情况下，文档中每个位置的单词的生成概率为 Topic 与文档的关联概率和 Topic 与该单词的关联概率的乘积。文档在一种指派情况下的生成概率是每个位置单词的生成概率的乘积。对每种指派情况下的生成概率进行求和，便得到文档的生成概率。生成的文档集的最大似然性为每个文档生成概率的乘积的最大值。关于上述公式更详细的研究，可参考 Bela 等人的技术报告“Introduction to the Dirichlet Distribution and Related Processes”[7] 以及 Gregor 的技术报告“Parameter estimation for text analysis”[8]。

在 LDA 算法中，可采用不同的迭代算法（如变分-EM 算法、Gibbs 抽样法）对 Θ 和 Φ 进行估计。以下粗略介绍一下采用 Gibbs 抽样法的 LDA 算法思路：

1）初始设定主题数 K，先验值 α、β，迭代总次数 I；

2）生成初始矩阵 Θ 和 Φ；

3）迭代选取一个文档中的一个单词，重新计算它所属的

主题以提高模型的准确性，并更新 Θ 和 Φ；直至一定的迭代次数或者模型收敛后结束迭代。

采用 Gibbs 抽样法实现的迭代过程如上所述。通过反复迭代，找到使得文档集似然值最大的最优 Θ 和 Φ。

2.2 LDA 技术挖掘代码 Topic 的相关研究

近年来，已有不少关于使用 LDA 技术挖掘代码 Topic 的研究。Pierre F. Baldi 等人以 4632 个不同领域的软件代码为实验数据，使用 LDA 技术挖掘其中的 Topic，然后人工分析这些 Topic 的含义，找出不同软件代码中的共性 Topic[9]。他们的实验发现：不同领域的共性 Topic 主要为软件开发实现中常用的 Java 语言相关的基本概念和方法，这与软件开发人员的经验一致。他们的实验结果说明，使用 LDA 技术挖掘出的代码 Topic 能够体现出一种代码的语义聚类。

GirishMaskeri 等人详细叙述了他们使用 LDA 技术挖掘代码 Topic 工作的细节[4]。他们利用代码静态分析技术分析代码中的各个元素，选取具有代表性的部分元素作为 LDA 算法的输入。在他们的方法中，α 和 β 参数由人工指定。他们采用一种最大似然估计方法决定最佳的 Topic 数目：对 Topic 数量从 1 到 N 进行遍历实验，选择使得所有文档集似然值最大的结果作为最佳结果输出。他们以 Linux 项目为例，指出 Topic 数量和文档集的最大似然值大致呈图 2 的关系（最终选取 270 为主题数）。该图中纵轴的 $\log p(w|T)$ 中的 $p(w|T)$ 为上一节中的 Likelihood。

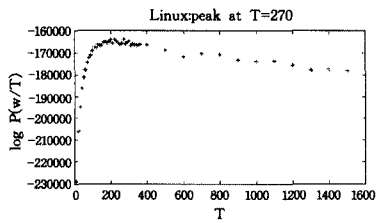

图 2 Topic 数量和文档集的最大似然性的关系图

在实验部分，他们指出挖掘出的 Topic 不仅包括领域 Topic，还包括底层实现的 Topic 和侧面 Topic。其中，领域 Topic 与本文所定义的功能型 Topic 类似。

已有的研究工作，无论是基于 PLSI 技术的方法，还是基于 LDA 技术的方法，主要依靠人工对挖掘出的 Topic 的类型和语义进行分析，该方法费时费力，并且难免有主观判断的成分在内。有必要研究如何利用代码包含的信息自动地确定 Topic 的类型和语义。

2.3 组织 Topic 的相关研究

采用 LDA 技术挖掘出 Topic，这些 Topic 之间也存在紧密或松散的语义关联。有不少研究人员在 LDA 技术的基础上，研究如何组织挖掘出的 Topic。

David M. Blei 等人指出在 LDA 模型中，各个 Topic 假设是相互独立的，而没有发现 Topic 的相关性。因此，他们提出了一种相关主题模型（Correlated Topic Model，CTM）[10]。基于该模型的技术不仅能够挖掘出 Topic，还能构造出 Topic 之间的关系网。然而，CTM 不适合用作代码功能组织结构的模型。代码功能组织结构一般为层次结构，CTM 无法体现高抽象层次 Topic 和低抽象层次 Topic 之间的层次包含关系。

Blei 等人还提出了一种层次化 Topic 模型，他们称之为 hLDA 模型[11,12]。基于 hLDA 模型的技术扩展了 LDA 技术，通过使用一种“中国餐馆过程”的算法，它能够构造出层次 Topic。但是，在 hLDA 模型中，与上层 Topic 关联概率高的单词一般与该 Topic 包含的下层 Topic 的关联概率低，这导致生成的很多 Topic 无法提供直观的语义解释。hLDA 模型也不适用于需要帮助理解代码功能的应用场景。

Segal 等人研究如何对抽象分类进行层次化组织。他们提出了概率抽象层次（Probabilistic Abstraction Hierarchies，PAH）的框架[13]。在 PAH 中，每个结点为一个抽象分类，由类特定概率模型（class-specificprobabilistic model，CPM）表示，CPM 为实例的概率分布。在 PAH 框架中，上层概念和它的下层概念之间在实例概率分布上存在相似性。他们以不同概念之间的实例概率分布的相似度作为度量依据，使得上层概念包含与它相似的下层概念。这种思想可被借鉴用于组织 Topic。

2.4 LDA 技术挖掘软件项目 Topic 的相关研究

现有一些工作研究如何能更有效地使用 LDA 技术对软件项目进行主题建模。Panichella 等人提出了一种基于遗传算法的方式，来确定使用 LDA 技术为软件项目建模时需要设定的各个参数，以便更有效地提取软件项目的主题[15]。

使用 LDA 技术来从软件项目源码中挖掘主题，作为软件的功能或业务主题等也是一个研究热点。Savage 等人开发实现了一个 Eclipse 的插件 Topic_{XP}[16]，使用 LDA 技术从软件项目的源码中提取主题，并通过可交互的可视化方式来帮助使用者了解和学习软件，提高软件维护的效率。Girish-Maskeri 等人提出了一种使用 LDA 从软件项目源码中抽取领域业务主题的方法，同时建立了这些主题与代码之间的关联，用来在软件业务文档缺失的情况下进行软件维护[4]。

现有的工作多是单独使用了 LDA 技术来提取软件项目的主题，并结合一些可视化、人机交互等方式来帮助使用者进行学习，但其并不能很好地保证提取出的主题的质量，缺乏对主题的组织与更丰富的描述。本文结合了主题建模技术与静态分析技术，以更好地组织与描述提取出的主题，帮助使用者理解软件功能。

3 基于主题建模的代码功能挖掘方案

3.1 问题分析与解决方案

为了生成带描述的功能型 Topic 层次结构，本文需要从以下 4 个方面进行研究：

1）挖掘 Topic：目前已有的实现了 LDA 算法的工具是针对自然语言文档进行处理的。本文需要研究如何预处理代码，生成 LDA 算法的输入。此外，LDA 算法涉及到许多参数，需要研究如何合理地对参数进行设置。

2）筛选 Topic：不是所有的 Topic 都能体现代码功能。在使用 LDA 技术挖掘出的各种类型的 Topic 中，只有功能型 Topic 能够体现代码的功能。功能型 Topic 关联的代码之间应该存在更紧密的相互依赖关系。因此，需要分析同一 Topic 下的代码之间的关系紧密程度，作为筛选 Topic 的依据。

3）组织 Topic：层次组织结构较为符合复用者理解软件代码功能的方式。通过对 Topic 数量的设置，可以使用 LDA 技术挖掘出不同抽象层次的 Topic。本文需要研究如何确定抽象层次高的 Topic 与抽象层次低的 Topic 之间的包含关系，进而按层次组织 Topic。

4)描述 Topic:目前 TM 方法主要依靠人工对 Topic 进行语义描述,该方法费时费力,而如果仅通过标示关键词作为 Topic 的解释,不足以辅助理解 Topic 的语义。本文研究了一种功能型 Topic 的语义描述模型。基于该描述模型,研究如何从代码中提取信息丰富的 Topic 的描述。

本文以 Java 语言的软件代码(包括代码中的注释)为研究对象,提出了一种 CFM 方法。图 3 为 CFM 方法的框架,该框架也适用于分析其它面向对象语言的软件代码。

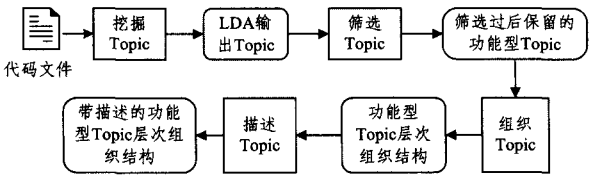

图 3 CFM 方法的框架

3.2 挖掘 Topic

代码与编程语言密切相关,为了挖掘出更多能体现代码功能的 Topic,需要对代码进行预处理,选取具有语义代表性而非编程语言相关的元素作为 LDA 算法的输入。LDA 算法的输出不仅包括挖掘出的 Topic,还包括这些 Topic 与所有代码之间的关联概率矩阵。CFM 方法通过设定关联概率阈值的方式界定 Topic 包含代码的范围。挖掘 Topic 由 3 步骤组成:代码预处理、LDA 技术挖掘 Topic、界定 Topic 包含代码的范围。

3.2.1 代码预处理

图 4 为代码预处理的流程,该流程分为选词、切词、过滤、词根化 4 个阶段。代码预处理步骤将具有语义代表性的单词加入到 LDA 输入文档。

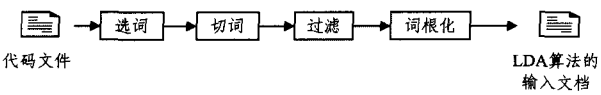

图 4 代码预处理流程

1)选词

CFM 方法从代码元素中选取的单词包括:类名、接口名、类或接口中属性的类型和名字、类或接口中方法的名字、方法的返回类型、方法参数的类型和名字。该选词方法符合 Pierre F. Baldi 等人通过实验验证的结论[9]。

2)切词

代码中的一些元素命名通常会是好几个单词的组合形式,例如"IndexReader"是由"Index"和"Reader"组成的,"IndexWriter"是由"Index"和"Writer"组成的。如果不对这些元素单词切词,"IndexReader"和"IndexWriter"在 LDA 算法中将会被视为两个不相关的单词。根据软件开发人员的常见命名习惯,CFM 方法按照元素命名中出现大写字母的位置进行切词,例如"IndexReader"被切为"Index"和"Reader"。另外,CFM 方法将几个连续大写字母切成一个单词,例如"UserDAO"被切为"User"和"DAO"。

此外,有些单词组合具有特殊的语义。有些单词可能总是共同出现在相同元素当中,但是这些单词并不经常单独出现。CFM 方法将高频出现的单词组合也视作一个单词,加入到候选单词集中。单词组合的概率计算公式如式(3)所示:

$$P(W_1,\cdots,W_n)=\frac{Count^n(W_1\cdots W_n)}{\prod_{i=1}^{n}Count(W_i)} \tag{3}$$

其中,$Count(W_1\cdots W_n)$表示单词组合"$W_1\cdots W_n$"在代码中出现的次数,$Count(W_i)$表示单词"W_i"在代码中出现的次数。当 $P(W_1\cdots W_n)$大于某一阈值时(CFM 方法将该阈值设为 0.5),"$W_1\cdots W_n$"单词组合将被视作一个单词。

3)过滤

在获取到候选单词集之后,CFM 方法还对候选单词集进行了过滤:(a)过滤 Java 语言的关键字,例如"import"、"class"、"public"等关键字。(b)过滤英语及编程语言中常见的停用词,如"the"、"a"、"set"、"get"等单词。(c)过滤 Java 语言常用的类名和接口名,Java 语言常用的类和接口大部分在 JDK 或 JRE 中 rt.jar 的 java.lang 目录下,如"String"、"Math"、"System"等。

4)词根化

过滤阶段完成后,需要对单词进行词根化,以免单词的形态对 LDA 输出的结果产生影响。在 CFM 方法中,只有经过词根化阶段后的单词才能被加入到 LDA 算法的输入文档。加入词根化阶段的 CFM 方法挖掘出的 Topic 的效果好于省略词根化阶段的方法挖掘出的 Topic 的效果。

3.2.2 LDA 挖掘 Topic

JGibbLDA[1)]是一个实现了 LDA 方法的开源项目,该开源项目使用 Gibbs 抽样方法生成最优的 Θ 和 Φ。CFM 方法关于 LDA 技术部分的工作复用了该项目的代码。在进行 LDA 迭代算法之前,需要设置一些参数,如 α 初始值、β 初始值、迭代次数和构造的 Topic 数量。有相关研究根据经验表明,当 α 初始值设为 $50/K$(K 为要构造的 Topic 数量)、β 初始值设为 0.1 时,LDA 算法挖掘出的 Topic 的效果较好[14]。和自然语言文档相比,从代码中抽取的单词数量较少,迭代次数不必设置较大。本文对 4 个不同规模、不同领域的软件项目的代码进行了实验:API_Example、TSR、CoWS 和 Lucene。其中前 3 个项目为北京大学软件工程研究所的软件资源库小组开发的软件项目,LuceneCore 是 Lucene 项目的核心部分。Lucene[2)]是一个属于 apache[3)]软件基金会的开源项目,它是目前最为流行的开源全文搜索引擎工具包。实验表明,将迭代次数设为 1000 和 2000,LDA 算法计算出的文档集似然值差别不大。故本文提出的 CFM 方法中,LDA 算法对 Θ 和 Φ 计算的迭代次数设为 1000。

关于如何确定 Topic 数量的问题的解决方案,本文参考了 Girish 等人的工作[4]。在他们确定最佳 Topic 数量的工作中,对 Topic 数量从 1 到 N 进行遍历实验,然后选择使得所有文档集似然值最大的结果作为最佳结果输出。为了提高效率,本文按照 $1+(i-1)*Step(i=1,2,\cdots,Step=15)$的方式进行。当发现第 i 次的实验效果比第$(i-1)$和$(i+1)$的效果都好时,说明最佳的 Topic 数量在$[1+(i-2)*Step]$到$[(1+i*Step]$之间,然后再在该区间查找最佳结果,如果第 1 次

1) http://sourceforge.net/projects/jgibblda/?source=directory

2) http://lucene.apache.org/

3) http://www.apache.org/

的实验结果最好，那么最佳 Topic 数量就为 1。本文方法使得遍历效率显著提高。该方法成立的前提假设是遍历结果大致呈如图 2 的抛物线，本文对 API_Example、TSR、CoWS、LuceneCore 的实验验证了该假设。图 5 为对上述 4 个软件代码实验的结果，其中横轴是选取的主题个数，纵轴是 $\log p(w|T)$。

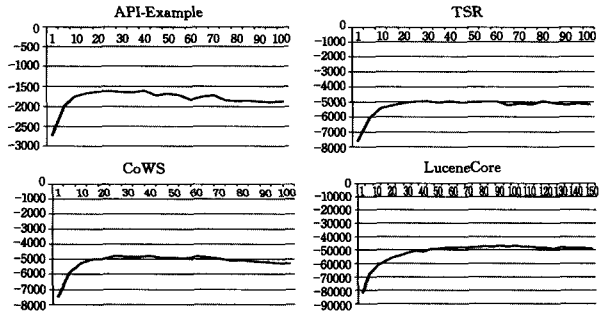

图 5　文档集似然值与 Topic 数量关系图

3.2.3　界定 Topic 包含代码的范围

LDA 算法的输出不仅包括挖掘出的 Topic，还包括这些 Topic 和代码之间的关联概率矩阵。在该矩阵中，每个 Topic 和每个代码文件都有一个关联概率。对于每个代码文件而言，它与所有 Topic 的关联概率之和为 1。假设某个代码文件与所有 Topic 关联程度一致，Topic 的数量为 TopicNum，那么，该代码属于其中任何一个 Topic 的概率应该为 1/TopicNum。故本文将 1/TopicNum 设为 Topic 包含代码文件的阈值。当代码文件与 Topic 之间的关联概率大于该阈值时，说明代码与 Topic 的关联紧密程度强于一般情况。

3.3　筛选 Topic

使用 LDA 技术挖掘出的各种类型的 Topic 中，只有功能型 Topic 能够体现代码的功能。功能型 Topic 包含的代码之间存在更紧密的相互依赖关系。本小节首先分析 Topic 与代码功能的联系，然后分析功能型 Topic 和其它类型 Topic 包含的代码之间依赖关系紧密程度的区别，并给出筛选Topic的解决方案。

3.3.1　Topic 与代码功能的联系

面向代码的 LDA 技术挖掘出的 Topic 是对一组语义关联的代码的聚类，两段代码包含的相同的单词越多，语义越接近；而共同实现一个功能的代码之间存在着语义关联。同一功能下的代码具有如下特点：

1)元素命名上的相似性：完成同一功能的代码一般在元素命名上有相似性。

2)单词的重合性：实现同一功能的各个代码之间必然有关系。例如，在类 ClassA 有一个类 ClassB 对象的域，那么类 ClassA 中会存在一个域声明“ClassB b”，从代码的文本角度看，两个类都包含“ClassB”。

同一功能下的代码一般具有上述两个特点，所以一个功能对应的代码有可能被 LDA 技术聚到同一个 Topic 下。使用 LDA 技术挖掘出 Topic 之后，需要删除不能和功能包含代码范围有较好匹配的 Topic，合并在包含代码范围上被功能覆盖但规模较小的 Topic，保留与功能包含代码范围较为接近的 Topic。

3.3.2　基于代码依赖关系分析的度量方法

同一个 Topic 包含的代码所对应的各个类或接口之间可能关系紧密，也可能关系松散，而同一功能下的各个类或接口之间关系是紧密的。故代码中的各个类或接口之间的关系紧密程度，在一定程度上可以反映出 Topic 是否为功能型Topic。本文将类或接口之间的依赖关系也称为代码关系或代码之间的依赖关系。

在利用 LDA 技术挖掘出 Topic 之后，CFM 方法利用代码静态分析技术，分析同一 Topic 下代码之间的依赖关系，排除代码关系不紧密的 Topic。CFM 方法以一种基于代码关系紧密程度分析的度量方法来度量 Topic 包含的类或接口之间关系的疏密。

关系类型有一般-特殊、整体-部分、关联、消息 4 种。类与类之间的关系按如下规则进行识别：

1)一般-特殊关系：Java 语言中的关键字“extends”用于类或接口声明表示一般-特殊关系。

2)整体-部分关系和关联关系：一个类与其属性所属的类之间可以建立关系。

3)消息关系：如果类 A 中的一个方法中调用了类 B 的方法，则类 A 和类 B 具有消息关系。

此外，CFM 方法对于“this”和“super”等关键字也进行了相应的处理。本文定义的代码关系紧密程度的计算公式如式(4)所示：

$$Density=E/N \tag{4}$$

其中，E 代表 Topic 包含的类或接口参与的关系数量，N 代表 Topic 包含类或接口的数量。CFM 方法设置的关系紧密程度阈值为：$T=0.8 * [全局的代码关系紧密程度]$。代码关系紧密程度大于该阈值的 Topic 被视为一个功能型 Topic。

3.4　组织 Topic

层次组织结构较为符合复用者理解软件代码功能的方式。LDA 算法可根据指定的 TopicNum 来挖掘不同抽象层次的 Topic，TopicNum 越小，挖掘出的 Topic 越抽象。CFM 方法采取算法 1 来层次组织 Topic。

假设判定某软件代码的最佳 Topic 数量为 30，指定的 Topic 层次缩小倍数为 3，那么，最底层先用 LDA 技术生成 30 个 Topic，然后根据缩小倍数，用 LDA 技术再生成 10 个Topic。新生成的 10 个 Topic 为更抽象的 Topic。然后分析 10 个抽象层次高的 Topic 与 30 个抽象层次低的 Topic 的包含关系。抽象层次低的 Topic 作为抽象层次高的 Topic 的子Topic。每个子 Topic 被与它相似度最高的抽象 Topic 包含。以此反复迭代，直到最后的根 Topic 数量不大于指定的 Topic 层次缩小倍数时，算法停止。

算法 1　组织 Topic 的算法，形成层次组织结构

```
reduceRate = 指定的每层 Topic 数量缩小的倍数;
topicNum = 判定的最佳 Topic 数量;
rootTopicList = LDA(topicNum);
while(topicNum>reduceRate) {
    topicNum = topicNum / reduceRate;
    newTopicList = LDA(topicNum);
    for(each topic in rootTopicList) {
        for(each newTopic in newTopicList) {
            计算 topic 和 newTopic 的相似度;
        }
        fatherTopic = newTopicList 中与 topic 相似度最高的 topic;
```

```
        fatherTopic.childTopicList.add(topic);
    }
    rootTopicList = newTopicList;
}
```

每个 Topic 和每个代码文件都有关联概率，因此，每个 Topic 有一个关于代码文件的关联概率向量。本文以向量空间模型中向量夹角的余弦值作为两个 Topic 相似度，计算公式见式(5)：

$$Similarity_{topica,topicb} = \frac{\sum_d P_{topica,d} * P_{topicb,d}}{\sqrt{\sum_d P_{topica,d} * P_{topica,d}} * \sqrt{\sum_d P_{topicb,d} * P_{topicb,d}}} \quad (5)$$

其中，topica 和 topicb 分别代表主题 a 与主题 b；$P_{topica,d}$ 代表 Topica 与代码文件 d 的关联概率。该相似度的度量值可以有效地使低抽象层次 Topic 作为和它最相似的高抽象层次 Topic 中的子 Topic。

3.5 描述 Topic

功能型 Topic 欠缺语义描述，不易被理解。本文首先提出了一种功能型 Topic 的语义描述模型(Code Topic Description Model，CTDM)。CFM 方法基于该描述模型自动地从代码中提取信息来描述 Topic。

3.5.1 一种软件代码功能型 Topic 的描述模型

在研究描述 Topic 的方法之前，需要首先确定软件代码功能型 Topic 的描述模型。本文提出的 CTDM 如图 6 所示。CTDM 具有如下特点：1)以层次结构组织功能：大功能包含小功能，小功能包含更小的功能。2)对功能主要从 3 个方面进行描述：文字性描述、核心类或接口、方法示例。

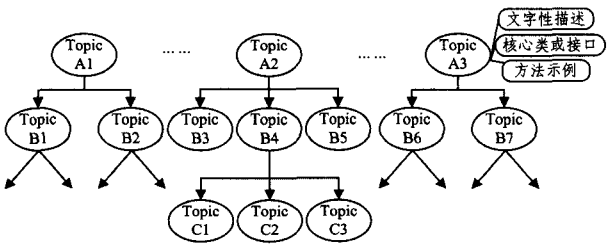

图 6 CTDM 示例

3.5.2 丰富 Topic 的描述

根据上节提出的 CTDM，从文字性描述、核心类或接口、方法示例 3 个方面丰富 Topic 的描述。

在软件代码中，最贴近于文字性描述的元素是注释。CFM 方法自动生成文字性描述的方式是首先将注释按句子切分，然后对这些句子进行评分排序，将排名最高的句子作为 Topic 的介绍性描述。本文计算句子与 Topic 的关键词列表的相似度，并将其作为这些句子的得分。相似度的度量值也是向量空间模型中向量夹角的余弦值。

一个 Topic 中的核心类或接口，一般会被该 Topic 中较多的其它类或接口依赖。通过代码静态分析技术，可以得到所有类或接口之间的依赖关系。本文采取如下策略对 Topic 中的类或接口的核心程度评分：

1)如果类或接口被该 Topic 下的某个类或接口依赖，那么该类或接口的被依赖数加 1。

2)如果类或接口被其它类或接口继承/实现，那么该类或接口相应的被依赖数加上：继承它的类或接口的依赖数 * 0.5。

3)类或接口被依赖数即为其核心程度评分。

本文将根据上述策略计算出的得分前 10 的类或接口，作为该 Topic 的核心类或接口。

一般来说，核心类或接口参与方法的程度越高，该方法越重要。本文将其作为使用示例提取出来，采取如下策略对 Topic 下的类的方法进行评分：

1)如果该方法有参数，且参数的类型为核心类或接口，那么该方法的得分增加的值为：参数对应类或接口的得分。

2)如果该方法有返回类型，且返回类型为核心类或接口，那么该方法的得分增加的值为：该核心类或接口的得分。

3)如果该方法有局部变量声明，且局部变量的类型为核心类或接口，那么该方法的得分增加的值为：局部变量类型的对应类或接口的得分。

4)如果该方法内部执行中有方法调用，且方法调用的发起者的类型为核心类或接口，那么该方法的得分增加的值为：方法调用的对应类或接口的得分。

本文将根据上述策略计算出的得分前 10 的方法，作为该 Topic 使用示例。

由于代码信息的局限性，自动化的效果有时不一定理想。因此，本文实现的 CFM 工具还提供了一种人机交互的方式，让用户可以丰富 Topic 的描述。

4 实验结果分析

本小节对本文方法的几个关键步骤进行实验分析与讨论，从而说明本文方法的有效性及适用范围。

4.1 分析 LDA 技术对代码文件的聚类效果

LDA 技术能够将语义相近的共同实现同一功能的代码聚集在同一个 Topic 下，这是本文工作的理论基础。本文设计了如下实验进行验证：LuceneCore 是 Lucene 项目的核心部分的代码包，本文将该代码包下的 3 个子目录“analysis”、“index”、“search”下的代码提取出来，使用 LDA 技术从这些代码中挖掘 Topic。LuceneCore 是一个给复用者使用的搜索引擎工具包，它是以功能对代码进行包组织的。如果设置生成的 Topic 数目为 3 时，对该 3 个目录的代码进行 LDA 实验，生成的 Topic 与目录成为一一对应关系，则说明 LDA 挖掘出的 Topic 的确能够体现代码的功能。

表 1 Topic“term field doc”关联概率排序

代码文件	代码与 Topic 的关联概率
Lucene2\index\SegmentReader.java	0.9942
Lucene2\index\IndexWriter.java	0.9931
Lucene2\index\MultiReader.java	0.9919
Lucene2\index\IndexReader.java	0.9911
Lucene2\index\DocumentsWriter.java	0.9908
Lucene2\index\SegmentCoreReaders.java	0.9903
Lucene2\index\ConcurrentMergeScheduler.java	0.9887
Lucene2\index\IndexFileDeleter.java	0.9887
Lucene2\index\IndexWriterConfig.java	0.9881
Lucene2\index\TermVectorsTermsWriter.java	0.9873
……	……

表 1 为使用 LDA 技术挖掘出的其中 1 个 Topic 的结果。实验结果由两列组成，第一列为代码文件，第二列为该代码文件与 Topic 的关联概率，按照关联概率对结果降序排列。通过实验结果可以发现，与 Topic“term field doc”关联紧密的代

码文件都在 index 目录下；另外两个结果表明，与 Topic“doc reader field”关联紧密的代码文件都在 search 目录下，与 Topic“term offset token”关联紧密的代码文件都在 analysis 目录下。因此，该实验说明，LDA 的聚类效果的确不错，较为精准地反映了 lucene 的开发人员对代码的功能模块划分。

从本实验中还可以发现，Topic 的名字并不容易让人理解它包含代码实现的功能。例如从“doc reader field”3 个词中是无法让人理解代码实现的功能是建 index。

4.2 分析 Topic 的代码包含范围界定的效果

如何界定 Topic 对代码文件的包含范围不是很明晰。选取合适的阈值成为解决该问题的关键，CFM 方法设定的阈值为 1/TopicNum，如果代码与 Topic 的关联概率大于该阈值，那么将该代码看作被 Topic 包含。仍以 LuceneCore 的 3 个子目录为实验数据，本文对该界定代码范围方法的效果进行实验分析，其统计结果如表 2 所列。

表 2 Topic 代码包含范围的实验结果

Topic 对应功能	代码中目录对应文件数	属于对应目录的文件数	不属于对应目录的文件数	准确率	召回率
Index	145	134	11	92.4%	92.4%
Search	140	121	11	91.7%	86.4%
Analysis	74	74	32	69.8%	100%

根据《Lucene in Action》一书的介绍，index 和 search 都是较为独立的功能，故这两个模块的准确率和召回率都比较高；而 analysis 其实是为 index 和 search 两个模块提供服务的第三方模块，该模块更多地与外界交互，因此，有些在 index 和 search 目录下的代码文件被聚集到对应于 analysis 的 Topic 下。通过 LDA 方法挖掘出的 analysis 相关的 topic 准确率没有前两者那么理想，但是召回率达到了 100%。结果与预期相符。

4.3 分析代码关系紧密程度度量方法的效果

功能型 Topic 和其它类型的 Topic 相比，其包含的代码之间具有更强的相互依赖关系。本小节对本文提出的代码关系紧密程度度量方法进行有效性验证。

本文为 Lucene 3 个子目录指定 30 个 Topic，对这 30 个 Topic 的代码关系紧密程度进行实验分析，得出表 3 结果。

表 3 Lucene30 个 Topic 的代码关系紧密程度统计

全局代码关系紧密程度	最大 Topic 代码关系紧密程度	最小 Topic 代码关系紧密程度	大于阈值的 Topic 数	小于阈值的 Topic 数
5.01	7.47	2.91	6	24

实验可以得到主题“queri weight reader”和主题“version array match”分别包含的代码文件列表，从中可以发现，Topic“queri weight reader”包含的同一目录下的代码文件明显多于 Topic“version array match”，即说明 Topic“queri weight reader”更体现出功能性。事实上，这个更具有功能性的主题显然是代表了带权重的查询功能。本文对所有 Topic 进行分析后发现，本文提出的 Topic 代码关系紧密程度度量方法可以在很大程度上反映出该 Topic 是否为功能型 Topic。

4.4 分析基于相似度计算的 Topic 层次组织效果

本实验的实验数据为使用 LDA 技术挖掘出的不同抽象层次的 Topic：分别为 3 个抽象层次高的 Topic 和 30 个抽象层次低的 Topic。本实验通过观察高抽象层次 Topic 与低抽象层次 Topic 之间的包含关系，验证 CFM 方法中的 Topic 相似度计算方法对 Topic 层次结构组织的有效性。

表 4 为 Lucene 的 index 功能对应的高抽象层次 Topic“term field doc”和 30 个低抽象层次 Topic 相似度的计算值。从图中可以观察到，index 功能对应的高抽象层次的 Topic 与低抽象层次的 Topic 之间的相似度计算结果很好地符合了实际的划分结果，该实验说明本文基于 Topic 相似度计算的层次组织结构是可行的。

表 4 不同抽象层次 Topic 之间相似度的计算结果

低抽象层次 Topic	与高层次 Topic “term field doc”相似度	实际代码的职责
field doc thread	0.594015318	index
field info index	0.501280038	index
segment info directori	0.493021104	index
term doc reader	0.452532788	index
skip freq pointer	0.439773049	index
index commit directori	0.438439119	index
term prefix differ	0.436289372	index or search
document doc buffer	0.417881054	index
merg segment info	0.406227614	index
term vector offset	0.36511486	index
reader index processor	0.33527115	index or search
index ref searcher	0.293949897	search or index
posit post array	0.280231937	search or index
iterqueri doc	0.250928697	index or search
extens read reset	0.246849769	analysis or index
……	……	……

4.5 分析提取 Topic 文字性描述的效果

按照 CFM 方法，实验提取的关于 Lucene 的 index 功能和 search 功能的文字性描述如下：

Index—“IndexReader is an abstract class, providing an interface for accessing an index.”

search—“Expert: a FieldComparator compares hits so as to determine their sort order when collecting the top results with -LCB- @link TopFieldCollector -RCB-.”

提取的文字性描述在一定程度上反映了功能语义，但是也存在较为明显的语义表述上的问题，从注释中抽出的句子描述的对象一般为特定几个类而非功能。这些注释句子在一定程度上帮助用户理解 Topic 对应的功能，但仍不足以将功能解释清楚，这是由于注释内容本身的局限性所致。文字性描述提取的效果很依赖于注释的质量。

4.6 分析提取 Topic 核心类或接口的效果

《Lucene in Action》一书介绍的 indexing 功能的核心类包括：IndexWriter、Directory、Analyzer、Document、Field；介绍的 searching 功能的核心类包括：IndexSearcher、Term、Query、TermQuery、Hits。CFM 方法以依赖关系中的入度作为类或接口重要程度的度量指标。本实验对该方法的效果进行分析。

本文排序算法的实验结果如下：

对应于 Index 功能的 Topic 得分排名前 10 的类或接口：Field、Directory、FieldInfo Document、IndexInput、Lock、SegmentInfo、IndexReader、Token、FieldInfos；对应于 Search 功能的 Topic 得分排名前 10 的类或接口：Term、Query、Scorer

Collector、Weight、Filter、Cache、Searcher、Similarity、Entry。

《Lucene in Action》介绍的10个类中，本文方法可以直接找到的类有5个。此外IndexSearcher继承于Searcher，TermQuery继承于Query，Analyzer属于对应于Analysis功能的Topic。

上述实验结果说明本文方法可以帮助找出大部分核心类或接口，或者是与其紧密相关的类或接口。

4.7 分析提取Topic方法示例的效果

《Lucene in Action》介绍的一个index功能的方法示例如下所示：

```
protected void addDocuments(Directory dir)throws IOException {
IndexWriter writer = new IndexWriter(dir,getAnalyzer(),true);
writer. setUseCompoundFile(isCompound());
for (inti = 0;i<keywords. length;i++) {
    Document doc = new Document();
    doc. add(Field. Keyword("id",keywords[i]));
    doc. add(Field. Unindexed("country",unindexed[i]));
    doc. add(Field. Unstored("contents",unstored[i]));
    doc. add(Field. Text("city",text[i]));
    writer. addDocument(doc);
  }
writer. optimize();
writer. close();
}
```

而使用CFM方法提取的一个方法示例如下所示：

```
static void addDocWithIndex(IndexWriter writer,int index)
throwsIOException
{
  Document doc = new Document();
doc. add(newField("content","aaa"+ index,Field. Store. Yes,
      Field. Index. ANALYZED));
doc. add(newField("id",""+ index,Field. Store. Yes,
      Field. Index. ANALYZED));
writer. addDocument(doc);
}
```

这两个方法示例都展示了如何为Document对象doc添加内容。上述实验结果说明CFM方法的策略可以提供有用的方法示例。

结束语 如何能更好地帮助开发人员更好地学习和复用软件资源库中的软件项目是软件复用过程中值得关注的一个问题。与传统代码静态分析方法相比，TM方法具有抽象层次高、面向语义的特点。然而，当前研究欠缺对Topic进一步的语义分析，存在Topic语义不明晰的问题。本文结合了代码静态分析技术与主题建模技术，提出了CFM方法来挖掘软件功能。在已有研究工作的基础上，提出了一整套关于Topic的挖掘、筛选、组织、描述方法，为Topic添加了语义分析，并实现了CFM工具。未来还可以在如何依靠互联网中的信息为Topic添加文字性描述，核心类或接口、方法示例的排序策略等问题上再作深入研究。

参 考 文 献

[1] 杨芙清，梅宏，李克勤. 软件复用与软件构件技术[J]. 电子学报，1999，27(2)：68-75

[2] Cleland-Huang J，Gotel O，Zisman A. Software andSystems Traceability[M]. Springer，2012

[3] Kuhn A，Ducasse S，Girba T. Semanticclustering：Identifying topics in sourcecode[J]. Information and Software Technology，2007，49(3)：230-243

[4] Maskeri G，Sarkar S，Heafield K. Mining business topics in source code using latent dirichletallocation[C]// Proceedings of the 1st India software engineering conference. ACM，2008：113-120

[5] Gethers M，Savage T，Di Penta M，et al. Codetopics：whichtopic am I coding now[C]// 33rd International Conference on Software Engineering (ICSE). IEEE，2011：1034-1036

[6] Blei D M，Lafferty J D. Topic models[J]. Text mining：classification，clustering，and applications，2009(10)：71

[7] Frigyik B A，Kapila A，Gupta M R. Introduction to the Dirichlet Distribution and Related Processes[R]. UWEE Technical Report Number UWEETR-2010-0006. 2010

[8] Heinrich G. Parameter estimation for text analysis[R]. Technical Report. Fraunhofer IGD，Darmstadt，Germany，2009

[9] Baldi P F，Lopes C V，Linstead E J，et al. A Theory of Aspects as Latent Topics[C]//Proceedings of the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications，OOPSLA'08. 2008：543-562

[10] Blei D，Lafferty J. Correlated topic models[J]. Advances in neural information processing systems，2006，18：147

[11] Blei D M，Griffiths T L，Jordan M I，et al. Hierarchical topic models and the nested chinese restaurant process[C]// Advances in Neural Information Processing Systems 16：Proceedings of the 2003Conference. MIT Press，2004，16：17

[12] Blei D M，Griffiths T L，Jordan M I. The Nested Chinese Restaurant Process and BayesianNonparametric Inference of Topic Hierarchies[J]. Journal of the ACM (JACM)，2010，57(2)：1-30

[13] Segal E，Koller D，Ormoneit D. Probabilistic abstraction hierarchies[J]. Advances in Neural Information Processing Systems，2002(2)：913-920

[14] Griffiths T L，Steyvers M. Finding scientific topics[J]. PNAS，2004，101：5228-5235

[15] Panichella A，Dit B，Oliveto R，et al. How to effectively use topic models for software engineering tasks on approach based on genetic algorithm[C]//Proceedings of the 2013 International Conference on Software Engineering. IEEE Press，2013：522-531

[16] Savage T，Dit B，Gethers M，et al. TopicXP：Exploring Topics in Source Code using Latent Dirichlet Allocation[C] // ICSM. 2010：1-6