基于约束求解的代码查询技术在 StackOverflow 上的实证研究

陈正钊 姜人和 潘敏学 张天 李宣东

(南京大学计算机软件新技术国家重点实验室 南京 210023)

摘要 代码查询在代码复用的过程中起着十分重要的作用，而面向程序员的专业问答网站 StackOverflow 上围绕代码的问答则是代码复用的一个典型场景。在这个现实场景中，采取的是人工回答的方式，而人工回答往往存在着实时性较差、提问描述不准确、回答可用性不高等缺点，但如果采取代码查询的方式搜寻可用代码，实现自动化并替代人工回答，便可以省去大量的人力和时间成本。如今已经出现了许多代码查询技术，但大都缺少在真实案例上的应用经验，我们以 Satsy 的思路为参考，实现了针对 Java 语言的基于约束求解的代码查询技术，并设计了实证研究，以 StackOverflow 为研究对象，主要研究如何将基于约束求解的代码查询技术应用在该网站上围绕代码的问答中。我们首先对网站上的问题进行了分析，针对 Java 语言提取了浏览量高的 35 个问题，作为查询问题；然后在 GitHub 上抓取了共约 3 万行代码，将它们转换成约束的形式并构建了一个较大规模的代码库以支持代码查询；最后通过对这 35 个问题的查询结果的分析，评估了该技术在 StackOverflow 上实际应用效果。结果表明，该技术在所研究的具体问题和代码规模上具有较好的实际应用效果，在相当高的程度上能替代人工回答。

关键词 代码查询;约束求解;开源代码库;实证研究

Empirical Study of Code Query Technique Based On Constraint Solving On StackOverflow

CHEN Zhengzhao JIANG Renhe PAN Minxue ZHANG Tian LI Xuandong

(State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210093, PR China)

Abstract Code query plays an important role in code reuse, and the Q&A around code on StackOverflow, a professional question-and-answer site for programmers, is a typical scenario for code reuse. In this real scene, it adopts the method of the artificial answer, which usually has the disadvantages of poor real-time, some questions to describe is not accurate, and answer availability not high enough. However, if the way of code query and search for available code is taken, the process can be automated and artificial answer can be replaced, that will save a lot of manpower and time cost. Now there are already many code query technologies, but mostly lack in experience of application in the real case. Based on the ideas of Satsy, we implement the code query technology based on constraint solving for Java language, and design the empirical study -- use StackOverflow as the research object, mainly study how to apply the code query technology based on constraint solving of Q&A about code on the website. First of all, we analyzed the problems on the website, and extracted 35 problems with high traffic in Java language as query problems. Then, about 30,000 lines of code were captured from GitHub, and they were converted into the form of constraints as well as built as a large code base to support code query. Finally, through the analysis of the query results of these 35 questions, the practical application effect of the technology on StackOverflow was evaluated. The results show that the technology has good practical application effect on the specific questions and code scale studied, and can replace the artificial answer on a considerable scale.

Keywords Code Query;Constraint Solving;Open Source Code Database;Empirical Study

1 引言

代码查询在代码的分析、测试和复用等工作中都具有十分重要的地位，在软件架构分析、逆向工程、一致性验证、代码审查、程序插桩等方面都有着广泛的应用[1]。本文所提到的代码查询，主要是指在代码复用过程中所要进行的代码查询工作。

在大多数现有的代码查询工作中，搜索方式都是基于关键词的搜索，如 Google Code Search[2]，Krugle[3]，SourceForge[4]等，这种方式效率可能高一些，但是准确度却常常会很低，并且需要在项目提供者对项目打上正确的标签的基础上进行。准确度更高的查询方式，应该是基于代码功能分析的语义化的代码查询，在这方面已经有许多相关的尝试，如利用了 API 间数据流信息的 Exemplar[5][6]，基于测试驱动的查询方式的 CodeGenie[7]，记录了数据结构、类型等信息的 Sourcerer[8][9]，利用 SMT 求解器处理的

Satsy[10]等。

随着计算机的普及，各行各业的从业者都开始逐渐需要掌握编程这一技能，专业化的程序问答社区越来越多，开发者之间的交流也由最初的发帖回帖变得更有社交性，其中最典型的当数StackOverflow。它面向的是编程人员群体，自2008 年创建的几年间的月独立访客量、月 Page View 量都飞速提升，成为了最大的程序问答网站且一直保持至今。

基于如此大的用户群体，StackOverflow 拥有着海量的涉及计算机方方面面的问题，而其中点赞量和浏览率最高的一些问题往往都是计算机学科中比较常见的问题，会被编程者们在现实中经常碰到。对于一类关于特定功能代码实现的提问，回答者通常只需要提供一段代码即可，这往往是通过提问者与回答者进行自然语言沟通来进行的。这种提问方式在以下方面存在着不足：

1）实时性，在问题被提出以后通常需要等待相当长的时间才会有人来对这个问题进行回答；

2）问题的质量，如果问题没表述清楚或质量较低会让人难以进行回答，甚至会因为差评太多而封禁；

3）回答的准确性和可用性，在一些情况下回答者只会给出问题可参考的资料或者其他线索性的答案。

而实际上，这些问题可以利用语义化的代码查询工具来搜寻能解决的代码，实现自动化并替代人工回答，省去其中大量的人力和时间成本。

本文所提供的基于约束求解的代码查询技术，是针对 Java 代码的处理技术。在查询时与实际的提问行为类似，提问者只需要举出若干个所需代码的输入输出实例即可，而不需要事先知道代码的内部实现；而对于代码的处理，在我们的技术中是通过将代码的不同执行路径编码为约束形式，用这些约束来概括代码的功能。从路径分析的角度来看，一段程序的所有行为可以理解为其所有路径的集合，因为程序的每一次执行实际上就是其某一条路径的执行，那么我们就可以枚举其可能的执行路径并分别转换成约束，这些约束的集合就可以表示这段代码的功能。

因此我们首先要知道在 StackOverflow 上有多少常见问题能用输入输出实例来进行描述，并且构建这些问题的若干输入输出实例作为测试脚本。对于这些问题，我们还需要在开源代码库中找到相关的代码作为问题的回答，故本文选取了刚刚被微软以 75 亿美元收购的最大的开源代码托管网站 GitHub，作为本文实证研究的代码来源，将其中与问题相关的代码提取并编码为约束形式。最后进行查询实验，统计并分析实验结果，确定基于约束求解的代码查询技术能否为不同的问题查询到可用的正确的代码，实现对人工回答的替代。于是，我们有了以下四个研究问题（Research Question, RQ）来对本文实证研究进行评估：

RQ1：StackOverflow 上的常见问题中有多少可以使用 I/O 实例来描述？

RQ2：GitHub 上有多少与这些问题相关的代码可以被提取和编码？

RQ3：基于约束求解的代码查询的有效性如何？

RQ4：基于约束求解的代码查询的效率如何？

最终，我们通过本文的实验设计对以上四个研究问题进行了回答，可以得出结论认为对于 StackOverflow 上关于特定功能代码提问的回答，基本上都可以利用我们的代码查询技术进行自动化的替代，且有着较好的查询有效性和效率。

本文的组织结构如下：第 1 节介绍了研究的背景与动机，第 2 节对本文所用的基于约束求解的代码查询技术进行简单的介绍，第 3 节按照四个研究问题对本文实证研究的实验设计进行介绍，第 4 节对四个研究问题分别介绍实验结果并进行回答，第 5 节对本文工作受局限的部分与实验细节进行讨论，第 6 节对现阶段已有的相关的代码查询实证研究进行介绍和对比，第 7 节是本文实证研究的结论。

2 基于约束求解的代码查询

本文实现了一种基于约束求解的代码查询技术（Solver-based COde Search Technique，SCOST），该技术由 Kathryn 的工作所启发，在对其技术进行参考的基础上做了改进，通过分析遍历程序控制流图（Control Flow Graph, CFG）来得到程序的执行路径，然后将不同的路径编码为约束形式用于代码查询，在代码查询时使用 I/O 实例表示的查询规约，将代码所生成的约束与用户提供的每组 I/O 实例进行绑定并求解，将能够完成 I/O 实例中的数据转换的代码返回给用户。

根据 SCOST 的实现过程可以将其分为 3 个模块：转换模块、数据模块和查询模块。这三个模块之间相互独立，它们之间的关系由图 1 进行了总体的描绘。在确定了模块间数据传递的格式规范后可以分别演化而不互相干扰。框架的核心是数据模块，转换模块不断将新的 Java 源代码进行约束生成从而将代码与约束一一对应添加到数据模块的数据库中，查询模块进行查询时则是按照特定的搜索规则在数据模块中查找能够符合查询语句的约束与对应的代码。

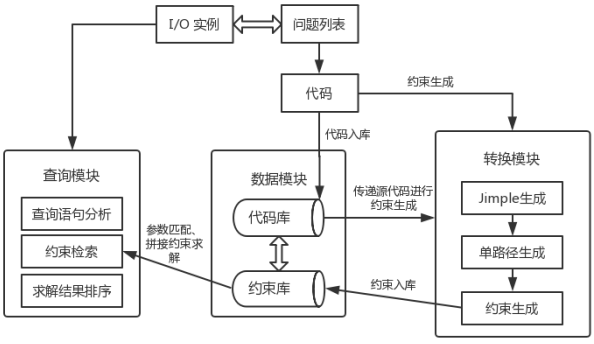

图 1 总体框架

Fig. 1 Main framework

本节下面将对这三个模块进行简单的介绍，然后列举一个从代码查询的例子以方便理解技术的细节，直观展现从代码到约束再到查询的过程。

2.1 转换模块

转换模块主要实现从 Java 源代码到约束的转换过程，包括 Java 源代码的 Jimple 生成、Jimple 的单路径格式转换、单路径的约束生成三个步骤。其中，单路径是指分析程序执行路径过程中，研究某一条单独路径时对该路径的简称；Soot[14] 是由麦吉尔大学（McGill University）的 Sable 研究小组为了供更快的 Java 程序执行工具而开发的 Java 优化框架，利用它可以对 Java 字节码的程序流和控制流进行分析；Jimple 是 Soot 的一种中间表示(Immediate Representation, IR)，是基于语句的类型化的三地址码表示，这意味着每个语句都只有一个操作符，且每个操作数都绑定了相应的类型，这与 SMT 约束规范十分契合，可以利用 Jimple 非常方便地将 Java 源代码转换为相对应的约束表示形式。

这个转换过程如图 2 所示，具体细节包括：首先利用 Soot 将 Java 源代码转换成易于分析的 Jimple 中间格式及程序的控制流图，然后对控制流图进行广度遍历分析得到程序的可能的执行路径，最后对所有的路径进行从 Jimple 到约束的转换，利用 Z3 求解器判断其合法性，所有可满足的路径约束的集合便是最终的转换结果。

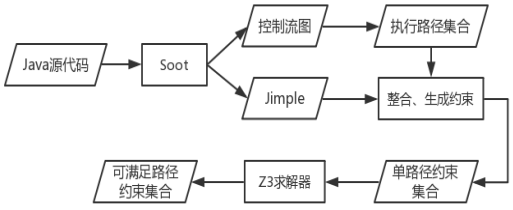

图 2 约束生成方法流程

Fig.2 Procedure of constraint generation

2.2 数据模块

数据模块主要包括代码库与约束库，代码与约束存储在文件系统中，它们的文件路径被存储到数据库中，并保持了代码与约束的对应关系：一段功能性代码对应一个约束文件夹，该文件夹下包含了这段代码的所有单路径约束文件。由于代码库和约束库有严格的一一对应关系，因此可以将两者看做是一个以约束形式表示的代码库。

代码库以 Java 源码文件为单位，每一段 Java 源码作为一个表项存储到代码库中，同时代码库也将记录这段源代码的详细信息，包括来源地址、

是否已经进入约束库等。用户可以向数据模块添加新的功能性代码片段，也能调用转换模块来对代码库中未转换的Java源码进行转换和约束入库。

约束库依赖于代码库而存在，每当代码库新增表项并且成功转换为约束后，约束库也相应地增加一个表项，记录这段代码的参数和返回值的个数与类型信息、约束文件路径、对应的代码位置、变量名称等。约束库主要是在查询模块进行查询时要进行检索操作，根据规定的匹配规则寻找到符合查询要求的约束，并将其对应的代码信息等一起返回给查询模块。

2.3 查询模块

在代码查询阶段，查询模块将起到主要作用。首先查询者构建 I/O 实例来描述目的代码的功能信息，SCOST 会对 I/O 实例表示的查询语句进行解析，处理成 JSON[11]格式的中间形式，然后跟以约束形式表示的代码库中可以匹配的约束分别进行 input 和 output 的变量绑定，将绑定好的约束交给求解器求解。如果求解成功，说明当前代码是能完成目标功能的代码，将代码加入到返回给查询者的结果列表中，若求解失败则继续查询。最后将查询结果按照正确度排序后返回给用户。

查询模块将提供一个前端页面，对查询语句的语法进行简单说明，接收用户输入的查询语句并进行解析；后端则根据用户语句的 I/O 实例的类型等信息与约束库中的约束信息进行比对和拼接，逐个测试符合 I/O 实例条件的约束，将满足计算结果的约束与代码按正确度排序。

2.4 实例

我们以图 3 中的代码为例，介绍其转换为约束形式的过程。这段代码的功能是判断数组中是否有重复元素其逻辑为创建一个该类型的集合，逐个将数组中的元素加到集合中，若集合中已包含了当前处理的元素，则认为是检测到了重复元素。其对应的问题为：

- Java Array, Finding Duplicates?

利用 soot 分析其控制流图得到结果如图 3，可以看到图中的语句已经是 Jimple 格式的语句，通过广度遍历的方式遍历控制流图可以得到程序的不同的执行路径。对于这一类有循环结构的代码，其执行路径理论上是有无数条的，于是我们在遍历时对一条路径中基本块的数量设定了限制。我们从控制流图的初始节点开始进行广度优先遍历，并设定遍历的最大基本块数量作为遍历的终点，当遍历的基本块没有后继节点时便将该便利的路径记录下来作为程序的其中一条可能的执行路径。

最后，在得到了以 Jimple 格式语句序列表示的单路径后，我们只需要对每一条路径中的每一个语句进行转换以编码为约束形式即可。为了直观，我们将其中一条路径对应的约束展示出来，如表 1 所示，所选择的路径为图 3 中“初始基本块 => label0 => label3”的所示的路径。

作为示例，我们为该问题设计了两个 I/O 实例对的测试脚本，表示为：

([1,2,3] -> false),([1,1,3]-> true)

进行 JSON 格式解析的结果为：

[{"input":{"list":["[1,2,3]"]},"output":{"bool":["false"]}},{"input":{"list":["[1,1,3]"]},"output":{"bool":["true"]}}]

在进行变量绑定时，将 JSON 中 input 和 output 部分中的值分别与约束中记录的 input 和 output 变量绑定，在这个例子中对于第一个 I/O 实例，就是将 zipcodelist_0 与[1,2,3]绑定，再将 temp$9_0 与 false 绑定，然后将绑定后的约束交给求解器求解，若求解成功，说明这段约束能实现 I/O 实例中输入输出的转换。我们将所有能够求解成功的约束对应的代码按照能实现转换的比例进行排行，并全部返回。

Java 源码

```
import java.util.HashSet;
import java.util.Set;

public class S1 {
  public boolean duplicates(final int[] zipcodelist)
  {
     Set<Integer> lump = new HashSet<>();
     for (int i : zipcodelist)
     {
        if (lump.contains(i)) return true;
        lump.add(i);
     }
     return false;
  }
}
```

控制流图

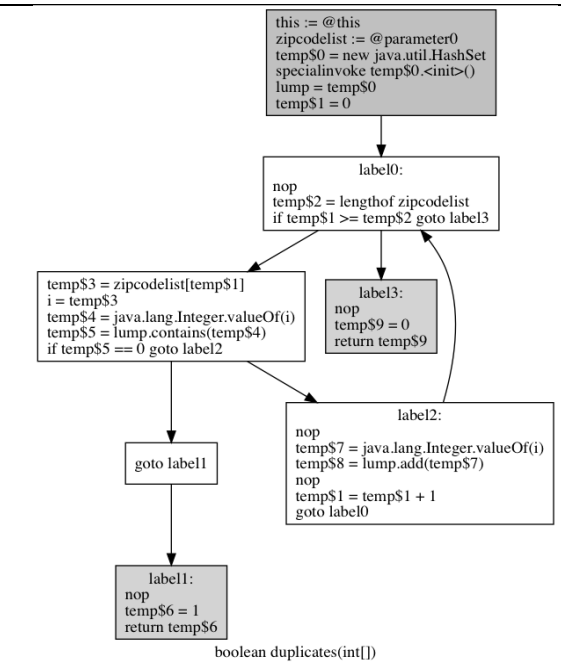

图 3 Java 代码与对应的控制流图

Fig.3 Java source code and corresponding control flow graph

表 1 实例约束

Table 1 Example contraint

1	;Inputs: zipcodelist_0#(as seq.empty (Seq Int))
2	;Outputs: temp$9_0#0
3	(declare-const zipcodelist_0 (Seq Int))
4	(declare-const temp$0_0 (Array Int Int))
5	(assert (= temp$0_0 ((as const (Array Int Int)) 0)))
6	(declare-const lump_0 (Array Int Int))
7	(assert (= lump_0 temp$0_0))
8	(declare-const temp$1_0 Int)
9	(assert (= temp$1_0 0))
10	(declare-const temp$2_0 Int)
11	(assert (= temp$2_0 (seq.len zipcodelist_0)))
12	(assert (>= temp$1_0 temp$2_0))
13	(declare-const temp$9_0 Int)
14	(assert (= temp$9_0 0))

3 实验设计

针对本文所提出的四个研究问题，我们做了相应的实验设计来进行研究。

我们从 StackOverflow 上的问题开始，先对能够用 I/O 实例来描述的问题进行筛选并构建其 I/O 实例（RQ1）；根据问题关键词在 GitHub 上抓取相关的 Java 代码，提取其中封装为 Java 方法（method）的可直接复用的代码进行约束转换，构建以约束形式表示的代码库，以使得 SCOST 能够应用于实际的问题与代码查询中（RQ2）；最后，我们需要从查询有效性和查询效率两方面对 SCOST 的查询效果进行评估（RQ3、RQ4），从而为我们的实证研究给出结论。

3.1 RQ1：StackOverflow 上的常见问题中有多少可以使用 I/O 实例来描述？

StackOverflow 作为最大的程序问答网站，有着巨大的用户群体，且由于对问题和回答都有着严格的管理，网站上的问题都是有质量的。而对于浏览量和点赞率最高的一些问题来说，它们都可以称为程序员最有可能提出的问题。对于这些问题，若我们的查询工具能进行回答，就能证明工具在实际应用中有相对良好的效果。

首先需要明确的是，能用 I/O 实例描述的问题必须是与数据处理有关的问题，所对应的也就是能够用代码处理的输入输出实例来体现功能的代码。如问题

- Replace a character at a specific index in a string? [1]

即给定一个字符串，如何将其特定下标的字符修改成另一个字符，这个问题可以用多个类似于(“aaa”, 1, b -> “aba”)的 I/O 实例对来体现其功能；解决这个问题的代码也是对一个数据区域（string）进行处理的功能性代码，这类代码是可以转换成约束的。而其他的问题有的并不是代码相关的问题，如问题

- What is a NullPointerException, and how do I fix it? [2]

还有的问题所对应的代码并不能用 I/O 实例对来体现其功能，这些都是我们需要筛选掉的问题。为了便于理解，部分问题与对应的规约化的 I/O 实例如表 2 所示。我们所设计的 I/O 实例规约也如表所示，一对圆括号里的内容表示一个 I/O 实例，括号里面用箭头的符号来分隔输入与输出的实例数据，不同实例间用逗号分隔。

表 2 部分问题与对应的规约化 I/O 实例

Table 2 Questions and specified I/O examples

问题	I/O 实例
How to get the separate digits of an int number? [3]	(0->[0]),(12->[1,2]), (345->[3,4,5]),(1100->[1,1,0,0])
How do I reverse an int array in Java? [4]	([1]->[1]),([1,2]->[2,1]), ([1,2,3]->[3,2,1])
Properly removing an Integer from a List<Integer> [5]	([1],1->[]),([2],1->[2]), ([1,2],1->[2]),([1,2],3->[1,2])

3.2 RQ2：GitHub 上有多少与这些问题相关的代码可以被提取和编码？

首先，对于所筛选出来的问题，既然在 StackOverflow 上的浏览与点赞量排行靠前，那这些问题必然是程序开发者所经常遇到的问题，根据经验可以认为，在开源代码仓库基本上都能找到相应能解决问题的代码，而要对此进行验证，也需要下一步查询实验的佐证。

因此，本文选取了最大的开源代码托管网站 GitHub 作为实验的代码来源，将所需代码编码为约束形式以构建代码库。但 GitHub 上有数以百万计的代码，我们的实验也无法将所有代码囊括其中，因此本文选择了一个较为实用的方法：根据所筛选的问题获得问题的关键词，然后在 GitHub 上用这些关键词抓取开源项目与代码。

对于所抓取的代码，我们还需要进一步的筛选。首先，我们所需要的是已经封装好的有参数和返回值的 Java 方法，其次这些方法还需要满足以下三个条件：

（1）不能是抽象方法或本地方法（native method）；

（2）不能以 main/set/get/test 开头；

（3）方法所涉及的变量类型只能为基本数据类型、String 类或容器类。

3.3 RQ3：基于约束求解的代码查询的有效性如何？

在拥有了查询问题与对应的 I/O 实例以及一个相当规模的代码库之后，我们需要进行查询实验，对基于约束求解的代码查询的效果进行评估。在本文的查询实验中，我们为每个问题都设计了 5 组 I/O 实例作为查询脚本输入，这 5 组实例应当都是较为简单但具有代表性的，能够尽可能地覆盖所需代码处理不同数据时的处理情况。本文将从有效性和效率两个方面分别做分析统计以解答研究问题，下面先对有效性进行阐述。

由于 I/O 实例对代码功能描述具有局限性，用 5 组 I/O 实例有时并不足以完全描述所需代码；并且本文的代码查询技术对代码的处理是通过分析路径来实现的，这就意味着有时并不能穷尽所有路径，因此，查询实验所搜索到的代码有时候并不一定是所需代码。我们引入了“正确度”的概念来对查询结果进行度量，在本文中，查询结果

1 https://stackoverflow.com/questions/6952363
2 https://stackoverflow.com/questions/218384
3 https://stackoverflow.com/questions/3389264
4 https://stackoverflow.com/questions/2137755
5 https://stackoverflow.com/questions/4534146

的正确度为代码能够完成输入输出转换的实例在5组实例中的比例。综上，我们对所查询到的代码按照5组I/O实例的处理结果分成了3类：疑似正确（likely）、实例正确（valid）和证实正确（correct）。疑似正确是指代码能够完成5个I/O实例中一部分的输入输出转换，正确度小于1；实例正确是指代码能够完成全部5个I/O实例的输入输出转换，证实正确是指代码确实是问题所需的代码，这两个的正确度都为1，且最后一个需要进行人工检验。举个例子，如问题“判断一个字符串是否是合法数字”，它的I/O实例应该包含如下两个：(“1”->true),(“1.5”->true)；若此时有一段“判断一个字符串是否为整数”的代码I，它能处理完成前一个实例的转换（判断“1”为字符串表示的整数），而对后一个实例的处理为(“1.5”->false)，故代码I属于疑似正确，正确度为0.5；而“判断一个字符串是否是合法数字”的代码II则能够完成所有实例的转换，故属于实例正确，正确度为1，在人工检验后也会被证实为证实正确，即问题所需代码。

要对有效性进行评估，我们需要对查询结果中的疑似正确数和实例正确数进行统计，并人工检验和记录第一个证实正确代码的排行（Rank of First Correct，记为RFC）。而证实正确的代码便是在现实情况中，提问者希望得到的代码，在实例正确数足够多的情况下，RFC越高，说明查询结果越好。

3.4 RQ4：基于约束求解的代码查询的效率如何？

根据本文查询技术的特点，我们判断其查询效率与查询过程中所遍历的代码路径数以及约束中符号的总个数有关。因此，我们为每个问题进行查询的查询时间、代码路径数、符号总数进行了统计，查看它们之间的关系。

4 结果

4.1 RQ1：StackOverflow上的常见问题中有多少可以使用I/O实例来描述？

最终，我们对带有Java标签的浏览量和点赞率排行最高的前1000个问题进行筛选，得到了88个问题（约占9%），可见这是一个不小的比例，然后在GitHub中寻找能这些问题的相关代码。

对于这88个问题，我们需要分析其中所可能涉及的API调用，将有关处理基本数据类型、String类和Container类数据的问题标记出来，因此对这些问题进行了类型统计，得到的统计结果如表3所示，有些问题有可能不止属于一种问题类型。

表3 StackOverflow问题筛选统计结果

Table 3 Statistical result of questions

问题类型	个数
与处理基本数据类型或其数组相关	18
与处理String类型相关	15
与处理Container类型相关	15
与处理Date类型相关	7
与处理Random类型相关	4
与处理format操作相关	9
其他，如自定义类相关等	34

考虑到本文所用技术仍具有一定的局限性：暂时只支持基本数据类型及其一维数组、String类、容器类，且一维数组可以看做是容器类List的一种，于是代码查询规约只根据表3中前三类问题设计，本文分别根据这三个类型选取了排行前10、10、15的共35个问题，并为这些问题人工构建I/O实例作为实验查询的测试脚本，作为本文查询实验的研究测试问题。这些问题如表4所示。

4.2 RQ2：GitHub上有多少与这些问题相关的代码可以被提取和编码？

我们希望构建一个有一定规模的代码库来进行查询实验，并且希望所构建的代码库中存在能解决这些问题的代码，但我们需要确定这些与问题相关的代码能被提取和编码。于是我们利用GitHub提供的API根据问题的关键词来抓取代码，但即便仅根据关键词搜到的开源项目仍然很多，代码量更是一个十分大的数量级，因此我们规定了如下的筛选规则：项目描述与关键词匹配度最高的前5个项目，以及项目代码文本与关键词匹配度最高的前15个项目作为要抓取的项目。

最终，本文从总共 333 个项目中的 32261 Java 方法里面，将其中 1226 个符合要求的方法筛选出来并转换成约束形式，并构建了一个较大规模的代码库，根据经验可以认为其中包含了能解决问题的代码，并且也包含了更多的与问题无关的实现其他功能的代码，这个代码库可以看做为现实中代码查询环境的模拟。代码库的这些性质也可以在我们的查询实验中得到验证。

4.3 RQ3：基于约束求解的代码查询的有效性如何？

实验结果各个评估指标的统计如表 5 所示，可以看到，35 个问题中有 30 个在查询实验中都能查到问题所需的代码，并且大部分（25个）问题的 RFC 都为 1，其余 5 个的 RFC 也不超出前 10；RFC 越小，说明证实正确的代码在查询排行中越高，查询正确性也越高。这个统计结果主要说明了两点：第一，在开源代码中，存在着现成的代码能作为 StackOverflow 上典型代码提问的回答，并能通过一些代码查询技术进行自动化的查找；第二，本文所用的基于约束求解的代码查询技术在这个方面实际应用的有效性比较高，查询到的代码量多且能将正确的代码排在较前的位置。

对于少部分未能回答的问题，一方面是由于本文工具对数据类型支持的局限性，一方面也是由于基于约束求解方法的局限性，在代码查询无果的情况下仍可以交给人工回答。

4.4 RQ4：基于约束求解的代码查询的效率如何？

根据表 5 可以十分直观地看出，代码路径数和符号总数越多的查询情况，所用的查询时间也就越长。

通过对 35 个问题的查询时间进行统计，可以得到平均的查询时间约为 206 秒，即 3 分钟左右，这是一个相对人工回答较快的查询效率。当然，随着代码库规模的增大和代码库中约束复杂度的增长，查询时间也会随之而变长。

5 讨论

查询问题的筛选。在 88 个能用 I/O 实例进行描述的问题里，可以发现基本数据类型及其数组、String 类和容器类相关的问题占了比较高的比例，而我们的工具暂时也是仅能对这些类型进行支持，故在选择查询问题时候时以这些类型作为规则筛选。

代码库的构建。根据经验，我们认为在真实的查询环境中，实验中的查询问题所需的代码是真实存在的，同时存在着许多与问题无关的代码干扰查询的进度和结果。我们希望构建一个有相当规模的、能模拟真实代码查询环境的代码库，而在最终得到的代码库中，对于每一个问题都有能解决的代码，且还有许多与这个问题无关的代码作为干扰，可以认为达成了我们的目标。虽然在构建代码库前有了关键词的一层过滤，但根据某一个问题的关键词过滤的代码对另一个问题的查询仍然是一种干扰。而鉴于工具的局限性，我们暂时也无法进行全网的代码查询。

查询有效性的标准。我们的目标是对人工的代码查询进行替代，因此在对查询有效性的标准选择上进行了一些斟酌。精确度(precision)是指在查询到的结果中正确结果的比例，而我们的实验目标是找到正确代码并将其排在靠前位置，因为在现实查询中查询者往往更希望能尽早得到正确结果，而不那么关心一个查询结果页面中有多少正确结果，所以精确度并没有作为我们查询有效性的评判标准。而另一个度量召回率(recall)也在我们的考虑之中，指的是对于代码库中所有正确的结果我们的查询是否能全部找到而不遗漏，但由于代码库规模对人工统计来说相当庞大，难以确定库中有多少是问题所需的代码，因此我们最终也没有选择召回率作为度量标准。

表 4 问题列表
Table 4 Question list

编号	描述	关键词
1	How to get the separate digits of an int number?	separate digits number
2	Java reverse an int value without using array	reverse int value
3	How to concatenate int values in java?	concatenate int value
4	sorting integers in order lowest to highest java	int array sort
5	Java - how to convert letters in a string to a number?	convert string to int
6	Rounding a double to turn it into an int (java)	round double int
7	Dividing two integers to a double in java	divide integer
8	Round a double to 2 decimal places [duplicate]	round double places
9	Checking if an int is prime more efficiently	check int prime
10	How can I pad an integers with zeros on the left?	pad integer
11	How do I convert a String to an int in Java?	convert string to int
12	How to check if a String is numeric in Java	string numeric
13	Java: How do I count the number of occurrences of a char in a String?	count char in string
14	Reverse a string in Java	string reverse
15	Check whether a string is not null and not empty	string null empty
16	Check string for palindrome	string palindrome
17	How do I concatenate two strings in Java?	string concatenate
18	Evaluating a math expression given in string form [closed]	string evaluate
19	Best way to convert an ArrayList to a string	convert list to string
20	How to remove the last character from a string?	string remove last
21	How can I test if an array contains a certain value?	array contain
22	How can I concatenate two arrays in Java?	array concatenate
23	How do I reverse an int array in Java?	array reverse
24	How do I remove objects from an array in Java?	array remove
25	How to add new elements to an array?	array add
26	Finding the max value in an array of primitives using Java	array max
27	Finding the min value in an array of primitives using Java	array min
28	Java: finding index of maximum from slice of an array	array max index
29	Sort an array in Java	array sort
30	Java Array: Finding Duplicates	array find duplicate
31	Java Array Sort descending?	array sort descending
32	How to find the index of an element in an array in Java?	array find index
33	How do I remove repeated elements from ArrayList?	array remove repeated
34	Intersection of ArrayLists in Java	arraylist intersection
35	Union of ArrayLists in Java	arraylist union

表 5 查询结果统计

Table 5 Statistics of query result

问题编号	疑似正确数	实例正确数	RFC	查询时间	代码路径数	符号总数
1	14	3	6	220.42	1929	78824
2	210	94	1	2118.42	14059	445656
3	78	13	-	166.67	11570	481866
4	183	11	1	486.57	24221	1031520
5	14	4	1	76.94	2376	75702
6	9	4	1	1.19	64	1272
7	5	4	1	1.32	70	530
8	19	18	1	6.68	348	2918
9	125	59	1	73.02	4155	46816
10	0	0	-	287.03	1025	37607
11	28	3	1	101.55	2744	112728
12	34	12	1	44.5	2282	51342
13	3	2	1	13.87	996	31835
14	30	3	1	54.2	2467	104670
15	62	6	1	37.85	2230	55033
16	40	12	1	44.46	2242	53061
17	5	3	1	21.43	1213	49263
18	15	1	-	226.19	3668	155794
19	2	1	2	2.02	447	34890
20	30	2	4	58.02	2488	106642
21	20	5	1	13.51	1883	62553
22	14	0	1	66.65	6933	351513
23	196	9	1	370.93	20627	862181
24	44	0	1	345.94	18813	870706
25	4	2	1	365.66	20885	965350
26	63	34	1	183.91	12913	540022
27	101	30	1	105.78	9537	358077
28	98	44	1	119.47	8832	323026
29	201	5	7	481.48	21746	924149
30	45	29	1	22.19	2187	44929
31	198	5	5	462.02	21780	927721
32	43	16	1	116.29	10988	498282
33	181	3	-	395.57	21813	915638
34	21	0	-	59.65	6121	305515
35	22	0	1	56.33	5885	290951

6 相关工作

当前的代码查询技术基本都沿着一个“提取——转换——展示”的框架来实现，“提取”即将代码中能够被有效利用或者应用需要的部分筛选出来，“转换”即把代码变换成一种中间形式以便于进行查询，“展示”即对以中间形式表示的代码按照查询要求所查到的结果进行排序并展示出来。

为了对开发者进行代码查询的规模和目的进行了调研，Sim 等人对 Google, Koders[12], Krugle,

Google Code Search, SourceForge[13]这 5 个不同的代码查询工具在进行代码查找方面的表现进行了实证研究，得出了基于特定代码检索的查询工具在子系统（subsystems，如一个完整的实现某个功能的算法）上的效果要更好，而 Google（基于关键词）则在代码块（blocks，如几行短代码）上的检索效果更好些的结论。这对于进行代码查询工具的开发有一定的指导意义，也说明本文在针对 StackOverflow 特定功能代码提问的解决上，使用语义化的代码查询工具的效果应该更好一些。

另外，我们还调研了几种当前比较具有代表性的语义化的代码查询技术，作为本文工作的参考。

在 ICSE2018 一篇文章[15]中 Kim 等人提出了另一种代码查询技术 FaCoY，该技术是从代码到代码的查询，查询输入是一段代码片段，输出是一段功能相似的代码片段，其实现思路采取了对代码、问题进行索引的方式，通过从代码到问题、再到代码的过程来找到相似代码。该文章的实验对 FaCoY 和 Krugle、searchcode 的查询效果进行对比，证实了其方法比当前已有的工具查询效果要好。

Sourcerer 是一种记录了数据结构及变量引用关系的代码查询技术，由 Sushil 等人实现，在他们的文章 Sourcerer: A Search Engine for Open Source Code Supporting Structure-Based Search[8]中，介绍了 Sourcerer 的特性和适用范围，以及一个实验来评估其可用性。通过文章可以发现，Sourcerer 的查询规约仍是基于关键词的搜索，所做出的改进是在分析代码进行中间结构转换的步骤中，记录了代码中特殊数据结构的信息以及某些变量之间的引用关系，通过这些信息与关键词进行比对进一步筛选以提高查询的准确度；并且 Sourcerer 比较侧重于数据结构的检测，能够比较准确地定位到某个数据结构的实现位置，但很难进行特定功能代码的查询，对于本文所研究的查询问题处理效果较差。

要对特定功能代码进行查询，一般来说需要在查询时对代码的功能进行分析以筛选，当前已出现的代码查询技术中主流的分析方法分为动态分析和静态分析两种，下面分别介绍这两方面比较具有代表性的技术 CodeGenie 和 Sasty。

CodeGenie[7]是测试驱动的代码查询工具，由 Otavio 和 Sushil 等人实现，并集成为一个 Elipse 可拓展平台使用，其主要思路是参照测试驱动开发（Test-Driven Development, TDD），先借助 Junit 构建测试用例（test case）用以描述所需的代码功能，再从代码库中查找和提取能实现目标功能的代码，最后将可用代码“编织”到应用场景中。这个方法相比以前的代码查询技术的好处在于：基于测试驱动所找到的代码理论上就是正确的代码，无需进行人工审查，且代码能够完美集成到应用场景中，无需进行人工调整。但该技术也存在缺点，如动态分析的效率低、开销大的缺点，另外，在文章中也没有说明查询问题与代码的来源，在实际应用中的效果难以评估。

而 Satsy 则是使用对代码进行静态分析的方法，利用路径约束来描述代码功能。Kathryn 等人的文章 Solving the Search for Source Code[10]，在提出并介绍了语义化的代码查询工具 Satsy 后，对该工具的实际应用效果进行了实证研究。文章通过对比 Satsy 和基于关键词查找的 Google 在查询请求相同、所用代码库相同的情况下的查询效果，得出了 Satsy 在查询结果的有效性方面远比 Google 好的结论，也说明了在功能性代码查找方面语义化的代码查询工具确实比基于关键词的查询工具效果要好。然而这个实证研究也具有较多的局限性，文章实验所建立的代码库规模较小，仅仅支持整型和 String 类型变量，且不能处理包含分支、循环等复杂结构的代码；从文章的实验结果展示来看，其仅能实现对 Java String 类的部分较为简单的 API 进行查询，因此文章更多的是提供了利用路径约束来描述代码功能的思路，这也是本文所用技术的思路。

7 结论

本文为了验证 StackOverflow 上关于特定功能代码的提问能够用基于约束求解的代码查询技术进行一定程度的替代，实现自动化回答，设计了相关的实验来进行实证研究。本文按分类收集了 StackOverflow 上的这类问题中最受关注的其中 35 个作为研究对象，然后在 GitHub 上收集了大量的代码并将能够进行处理的一千多段代码转换成约束形式，构建了一个相当规模的代码库。在已有查询问题和代码库的基础上，本文进行了代码查询实验，结果表明，有 30 个问题都能找到能解决的代码，且平均时间仅为 3 分钟。因此，就本文实证研究结果而言，可以得出结论，使用基于约束求解的代码查询技术基本上能对 StackOverflow 关于特定功能代码的提问进行自动化回答，并对人工回答进行替代，节省人工和时间成本。

致谢

This work is supported by National Natural Science Foundation (Grant Nos. 61472180 and 61502228) of China.

参考文献

[1] 曾锃，赵建华. 基于程序分析的代码查询技术[J]. 计算机科学，2012，39(2):143-147.

[2] Google Code Archive. https://code.google.com/archive/search.

[3] Krugle. http://opensearch.krugle.org/projects/, March 2014.

[4] Source Forge. http://sourceforge.net/, March 2014.

[5] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. Exemplar: Executable examples archive. In International Conference on Software Engineering, pages 259–262, 2010.

[6] C. McMillan, M. Grechanik, D. Poshyvanyk, C. Fu, and Q. Xie. Exemplar: A source code search engine for finding highly relevant applications. Software Engineering, IEEE Transactions on, 38(5):1069–1087, Sept 2012.

[7] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, R. S. Morla, P. C. Masiero, P. Baldi, and C. V. Lopes. Codegenie: Using test-cases to search and reuse source code. In Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, ASE ’07, pages 525–526, New York, NY, USA, 2007. ACM.

[8] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. Sourcerer: a search engine for open source code supporting structure-based search. In Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, OOPSLA ’06, pages 681–682, New York, NY, USA, 2006. ACM.

[9] Linstead E, Bajracharya S, Ngo T, et al. Sourcerer: mining and searching internet-scale software repositories[J]. Data Mining & Knowledge Discovery, 2009, 18(2):300-336.

[10] K. T. Stolee, S. Elbaum, and D. Dobos. Solving the search for source code. ACM Trans. Softw. Eng. Methodol., 23(3):26:1–26:45, June 2014.

[11] Wikipedia. JSON. https://en.wikipedia.org/wiki/JSON.

[12] Koders, August 2012.

[13] Source Forge. http://sourceforge.net/, March 2014.

[14] Vallée-Rai R, Co P, Gagnon E, et al. Soot - a Java bytecode optimization framework[C]// Conference of the Centre for Advanced Studies on Collaborative Research. IBM Press, 1999:214-224.

[15] Kim K, Kim D, Bissyande T F, et al. FaCoY – A Code-to-Code Search Engine[C]// Ieee/acm, International Conference on Software Engineering. IEEE Computer Society, 2018:946-957.