|  |  |
| --- | --- |
|  | 学校代码： 10246 |
|  | 学 号： 16212010030 |
|  |  |



|  |
| --- |
| 硕 士 学 位 论 文 |

（学术学位）

**基于API文档的约束性语句分类及应用**

**Directive Classification and Application based on API Documents**

|  |  |
| --- | --- |
| 院 系： | 软件学院 |
| 专 业： | 计算机软件与理论 |
| 姓 名： | 张凯 |
| 指 导 教 师： | 赵文耘 教授 |
| 完 成 日 期： | 2019年 2月 25日 |

**复旦大学**

**学位论文独创性声明**

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。论文中除特别标注的内容外，不包含任何其他个人或机构已经发表或撰写过的研究成果。对本研究做出重要贡献的个人和集体，均已在论文中作了明确的声明并表示了谢意。本声明的法律结果由本人承担。

作者签名： 日期：

**复旦大学**

**学位论文使用授权声明**

本人完全了解复旦大学有关收藏和利用博士、硕士学位论文的规定，即：学校有权收藏、使用并向国家有关部门或机构送交论文的印刷本和电子版本；允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。涉密学位论文在解密后遵守此规定。

作者签名： 　 导师签名： 日期：

指导小组名单

赵文耘，彭鑫

目录

摘要

API在程序员进行编程时得到了广泛的使用，而程序员对API的查询和理解依赖于API文档。API文档不仅对API进行了功能及使用流程的说明，还提供了一些使用时的规定和限制，我们把这些语句成为约束性语句。约束性语句的是否正确解读决定了程序员能否准确无误的使用这个API，保证程序的正确性。

借助fastText工具，通过对已有API文档的@todo XXX句文本进行解析、训练、识别，我们将约束性语句分为@todo XXXXX几大类，平均准确率为@todo、查全率为@todo。接下来，通过对几大类中的句子进行分析总结，我们针对@todo XX等不同小类定义出了不同的启发式规则。最后，通过通用的匹配模式，我们获取约束性语句中的信息，对代码进行扫描检测，找到其中与约束性语句不符的片段，提示程序员注意与修复。

Abstract

第一章 引言

应用程序编程接口（Application Programming Interface，简称API）是一组定义好的函数，供编程人员调用后实现一系列特定的功能[1]，而实现的细节对编程人员来说是透明的。API通常实现的是一些经常得到复用、通用性比较强的功能，单独把这些功能抽取出来进行包装，能够使得不同程序员在实现该相同功能时能够节省编码的时间，提高构建软件时的效率。API在程序员进行编程时得到了广泛的使用，适用于各种编程场景，实现不同的程序功能[2-4]。

API文档就是对这些API的说明解释，是指导程序员如何使用API的指南。特别是对于第三方组织实现的API接口，由于发布时间较新、接口过于复杂、相关资料过少等原因，程序员必须严重依赖于其API文档，才能获得足够的知识去正确的使用它[5, 6]。

API文档是关于某个API的整体知识库，包含了如何正确使用API的各种信息，包括了API实现的功能、API相关概念解释、示例代码、API使用约束等。API约束性语句是我们关注的重点，它是描述了在使用API编程时应该遵守的限制和规范的语句。它必须被严格遵守，否则编写出来的程序就有可能不能运行或出现错误，降低程序的正确性，甚至可能造成损失。为了对API约束性语句有更直观的印象，我们在API文档中找了几个例子来说明，图1中展示了我们作为示例的3个片段。

第一段的描述片段来源于JDK文档中javax.xml.crypto.dom. DOMCryptoContext类中的getNamespacePrefix方法，该方法将传入的uri进行解析，返回此uri的前缀。其中的约束性语句“Throws:NullPointerException - if namespaceURI is null”表示参数namespaceURI如果为空，则抛出异常，这是对入参非空值的规定，应该被重视和遵守。

第二段的描述片段取自Apache Commons Collections library文档中org.apache.commons.collections4.set.ListOrderedSet的listOrderedSet方法，它的功能是返回一个排好序的集合对象，其中约束性语句“The list and set must both be empty.”规定了方法的入参必须都是空集，不能含有任何元素。

第三段展示了JDK文档中java.sql.Time类中的valueOf方法的描述片段，valueOf方法将一个JDBC格式的字符串转为Time对象，而其中的约束性语句“Parameters:s - time in format "hh:mm:ss"”的意思是s这个参数必须是

图 1 三段示例描述片段，黑色方框中的语句是其中作为代表的约束性语句

**valueOf**

**public static Time valueOf(String s)**

Converts a string in JDBC time escape format to a Time value.

**Parameters:** s - time in format "hh:mm:ss"

**Returns:** a corresponding Time object

**listOrderedSet**

**public static <E> ListOrderedSet<E> listOrderedSet(Set<E> set,**

**List<E> list)**

Factory method to create an ordered set specifying the list and set to use.

The list and set must both be empty.

**Type Parameters:** E - the element type

**Parameters:** set - the set to decorate, must be empty and not null

list - the list to decorate, must be empty and not null

**Returns:** a new ordered set

**Throws:** NullPointerException - if set or list is null

IllegalArgumentException - if either the set or list is not empty

**Since:** 4.0

**getNamespacePrefix**

**public String getNamespacePrefix(String namespaceURI, String defaultPrefix)**

This implementation uses an internal HashMap to get the prefix that the specified URI maps to. It returns the defaultPrefix if it maps to null.

**Specified by:** getNamespacePrefix in interface XMLCryptoContext

**Parameters:** namespaceURI - a namespace URI

defaultPrefix - the prefix to be returned in the event that the the specified namespace URI has not been bound to a prefix.

**Returns:** the prefix that is associated with the specified namespace URI, or defaultPrefix if the URI is not registered. If the namespace URI is registered but has no prefix, an empty string ("") is returned.

**Throws:** NullPointerException - if namespaceURI is null

**See Also:** XMLCryptoContext.putNamespacePrefix(String, String)

“小时：分钟：秒钟”这样格式的字符串，才能转换出正确的Time对象，否则输出的Time对象可能出现不可预测的错误。

从上面三个示例片段可以看出，首先，API约束性语句是融合在其他类型的描述语句之中的几种类型的语句可能相互连接，互为支撑，形成了一篇完整的API描述文档。其次，API约束性语句在整个文档中占比是比较小的，甚至有些描述文档是不含有API约束性语句的，因此分离工作就如同沙滩上寻找金子一般，虽然明显但也需要付出很大的努力。同时API约束性语句其实还有这各种不同的类别，上面三个片段中每个片段的约束性语句都属于这不同的类别，分别是“不允许空值”“允许空值”和“字符串类型限定”。有些类别自然语言特征结构是比较明显的，但是有些类别的自然语言形态又是多种多样的，甚至有些类别的语言形态是十分相似的，这对自动化分类提出了挑战。总而言之，对API约束性语句的研究存在着以下几个问题：第一个问题是怎么能够将API约束性语句从文档中这么多种类的知识中分离出来；第二个问题是这些多种多样的API约束性语句又是怎么具体分类的。

在现有的研究中，对API约束性语句的识别主要是依靠人工的标注和模式的识别，包括但不限于一些显著的标识，如@param、@exception、@throws、must等等[7]。而对于API约束性语句的具体分类，则只有人工进行阅读、标注来保证准确性。但是对于API文档中成千上万的约束性语句，仅仅依靠人工去标注显然是效率十分低下的一种方法，同时也对大规模识别提出了挑战。在这个背景下，我们对这个问题进行了良好的解答，为大规模自动化识别约束性语句提供了新方法和新思路。

首先，我们利用已有的人工对API约束性语句分类的数据库作为我们的训练集和验证集，利用fastText来进行较为粗糙的大类API约束性语句分类。因为我们的主要研究是针对API约束性语句，因此API文档中除了约束性语句的其他类别全部归为非约束性语句。其他的大类类别还包括@todo 等。然后，针对fastText分好的每一个大类数据，我们人工的阅读其中的每一个句子，观察句子的形态，结合自然语言处理技术中对句子结构的分析，定义解析跟API相关名词的识别方法，得到每个细分小类的一组启发式规则。根据这些启发式规则，我们就能匹配出高准确率的属于某个细分小类的约束性语句。细分小类属于某个大类，如@todo属于@todo，而针对这个细分小类的启发式规则有@todo等等。

整个工作的贡献如下@todo填充一些具体数据进行支撑：

1. 通过fastText的帮助，我们结合机器学习的方法训练出了约束性语句大类分类模型，将原本种类多样繁杂的约束性语句先进行了预分类，将大类间相似启发式规则的相互影响降低，提高了后续分类的准确性。
2. 针对所有细分小类都定义了与其匹配的一组启发式规则，保证了分类的准确性，也为大规模分类提供了方法和思路。
3. 给出了一个应用，将分好类后的约束性语句用于代码正确性的检查，提供了一种保证程序员编程正确性检查的新方法，丰富了代码检查的方式，给代码正确性提供了更多保证。

@todo 文章章节结构简述

第二章 背景知识

2.1 API相关知识

2.1.1 API概念及种类

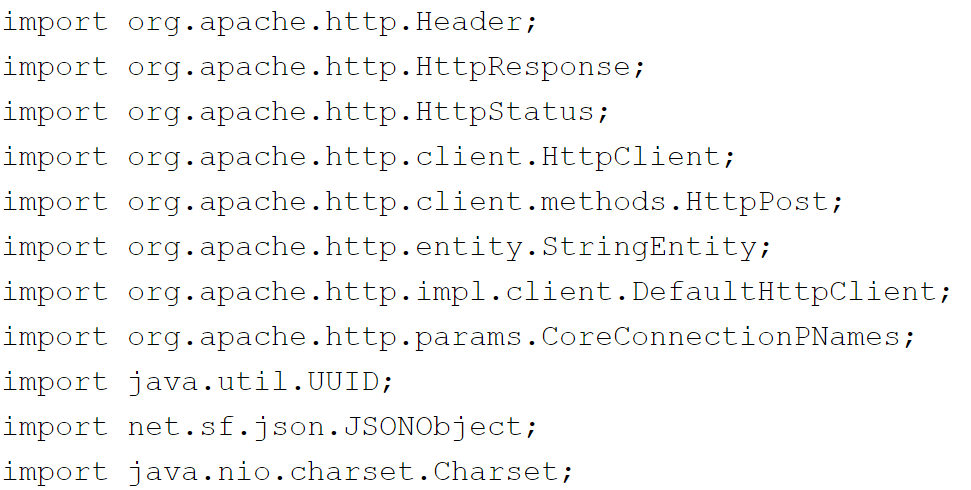
API，又名应用程序接口，是一组程序功能的定义、通信协议或用于软件编程的工具，定义了各个组件之间是如何交互的。应用程序接口通过抽象底层实现，只将需要用到的对象或操作暴露给程序员，底层的源代码或者内部工作机制的细

图 2 JAVA编程中利用HttpClient实现网络信息发送

节对程序员来说是透明的，这使得程序员在编写软件时调用这些功能时更加简单方便。

举例说明，在JAVA编程时，发送HTTP请求是我们在网络中进行交互必不可少的功能。一般来说，程序员通过调用HttpClient这个API进行网络请求的发送。图2展示了API调用的示例代码，首先我们引进（import）相关的API包，这使得程序能够获知需要调用API的接口和对象，保证程序编译的正确性。然后，我们调用API中的方法，填充相关对象作为参数，在这个例子中就是填充HTTP相关的连接信息、组建HTTP包。最后，通过“HttpResponse response = httpClient.execute(post);”这句代码进行网络信息的发送，并检验发送结果。

通过API进行HTTP调用，如果不计算填充调用参数的代码行数，其实只需要短短的三行代码就可以实现。然而，真实的HTTP调用是十分复杂的，API帮调用者屏蔽了底层复杂的实现，无疑大大提高了编程效率。例如，在进行HTTP调用时会传入目标域名，但是域名是便于人方便记忆的一个地址，在整个网络传输过程中是不能通过域名进行定位的。因此，需要对传入的域名（URL）进行域名解析，这涉及到通过缓存、域名服务器DNS解析进行域名到IP地址转换的整个流程。HTTP请求是一个应用层协议，需要依赖于运输层的协议进行数据的传输，因此Http是基于TCP/IP协议的。TCP的三次握手连接、数据传输、四次挥手断开连接、IP层的传输等等底层的协议调用都是API为调用者完成的，不需要调用者去关心具体实现，否则，HTTP调用就不是简单的几行代码能够完成的任务了。

API也有不同的类别，如库和框架的API，操作系统API，远程API，网络API等等[[1]](#endnote-1)：

1. 库API：API通常与软件库有关，API是一个功能的描述，而库则是对这个功能的具体实现。因此，单个API可以拥有相同程序接口不同的库形式的多种实现，这有一个好处就是一种语言可能可以调用另一种语言实现的库，例如SCALA开发时就可以调用JAVA的API。API的使用可能因为语言的不同而不同，比如面向对象语言的API提供了类与方法的规范，我们的研究主要针对的就是面向对象语言的API。
2. 操作系统API：API定义了操作系统和应用程序之间的接口，如windows操作系统提供了Windows API库供开发者使用。
3. 远程API：这种类型的API的作用是让程序员通过某种协议来操作远程资源，而特定的协议标准使得不同语言、不同平台的程序能够协同工作。JAVA远程方法调用就是一个很典型的例子，它通过JAVA远程方法协议来远程操作函数，但使用感受就像调用本地方法一样。因为JAVA远程方法API会在本地执行一个远程方法的代理对象，而通过代理对象经过协议去调用远程的方法，并返回远程方法执行的结果作为本地代理对象方法的返回值。因此，本地代理对象和远程对象之间存在一定映射关系，如果要修改其中一个，两者都要同时被修改。
4. 网络API：在进行网络开发时，API通常被定义为一组规范，比如HTTP的消息结构、HTTP传输协议等等。网络API在历史上几乎已经成为网络服务的同义词。但是随着网络2.0的发展，网络API也逐渐发展成为不同领域、不同类型的消费者的不同应用程序提供某种服务的程序接口。

2.1.2 API文档

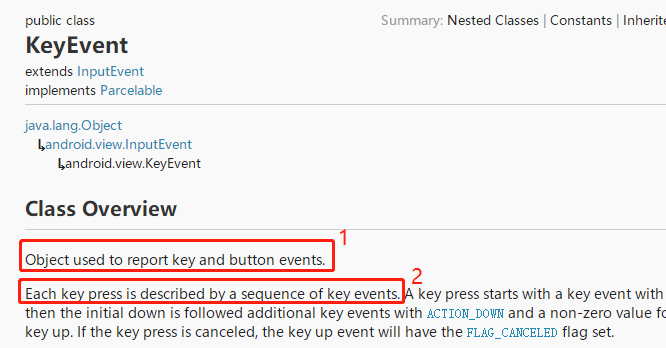
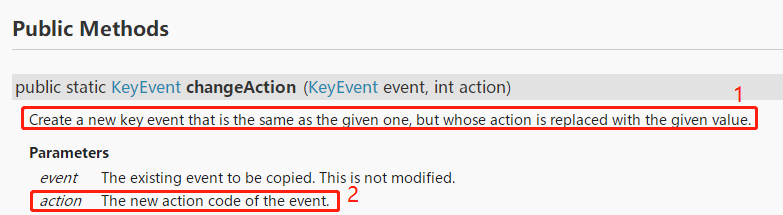
一个程序员在尝试使用API解决问题的时候，会想要知道如何准确有效的使用，这时API文档就起到了很大的作用。API文档是一种技术内容可交付成果，包含了如何有效使用和集成API的说明，也是一个简明的参考手册[[2]](#endnote-2)。

文档一般能在API的配套文档文件中能够找到，比如Javadoc、Pydoc等，他们具有相同的呈现方式和文档结构，但是其中的内容会因API的而不同[8]。API文档主要包括了类和方法的描述、设计原理、典型的使用场景、性能表现、示例代码片段等等，而API本身的内部实现细节一般是隐藏起来的。这类信息主要蕴含在功能性语句、概念性语句和值定义语句三种句子中。功能性语句主要表达了API是在什么场景下为了实现什么功能而编写的，在何种情况下性能如何，使用的流程是什么样的。如图3中类描述和方法描述的第一个方框中的那个语句，前一句表达了KeyEvent这个类的功能就是表示按键的事件，后一句说明了changeAction这个方法是把event 的action改成参数值传入的值。概念性语句描述了跟这个API相关的一些改变，具有一定的解释性、通用性。比如图3中类描述第二个方框中的那个语句，解释了每个按键的按压动作由一系列事件组成这个原理。但是如果开发者对这个原理很了解的话，这句话除去不看也不影响整篇文档的理解，这句话使得文档更加通俗易懂，这就是概念性语句的作用。值定义语句描述了一个方法参数或者返回值的属性，比如方法描述第二个方框中入参action其实就是需要新传入的事件的动作码。

除此之外，文档还包含了API的限制和约束，具体例子如图1所示。比如，此函数可能已经被废弃了，推荐使用别的新版本的方法；此函数的入参不能为空等等。这些语句称为约束性语句，是我们研究的重点。因为API文档对于使用者来说需要仔细阅读和理解，所以也对作者提出了更高的要求，需要随着API的更新而更新，还要保证文档的正确性。然而，即便是大型机构的API官方文档，API文档还是会存在一些错误，这是无可避免的，需要使用者在阅读和使用的时候格外注意[7]。

图 3 安卓API官网文档中示例片段，取自KeyEvent类

2.2 机器学习相关方法及工具

2.2.1 有监督学习

在机器学习领域，主要有两种类型的任务：监督和无监督，两者之间的区别在于是否需要根据基础事实来完成，也就是说我们是否事先知道样本的输出应该是什么样的。无监督学习是没有示例样本，根据给定数据推断这组数据中存在的自然结构；而有监督学习则是根据给定的样本数据和期望的输出来推测最近似于样本数据的输入和输出之间的一种关系。两者之间各有优劣，选择时需要根据具体的场景需求来判断哪种学习方式更合适。

有监督学习一般遵循以下几个基本步骤：首先，确定训练数据的类型，收集训练集。比如，在进行句子分类的时候，需要确定要分类的句子是哪些，然后将这些句子的示例从网站上、书籍中等地方收集起来。其次，确定学习函数的输入、结构和相应的学习算法。函数的输入一般来说取决于输入对象的表达方式，通常会被转换成特征向量。特征向量包含了许多能够描述对象的特征，这些特征不应该太多，但又需要蕴含足以准确预测输出的信息，所以要控制好特征向量的维度。现有的学习函数也有许多，如支持向量机、线性回归、决策树等等，需要根据使用场景和数据特征来选择。最后一步就是在收集的训练集上运行学习算法，有时可以通过验证集去优化学习算法的参数，使模型更加准确。

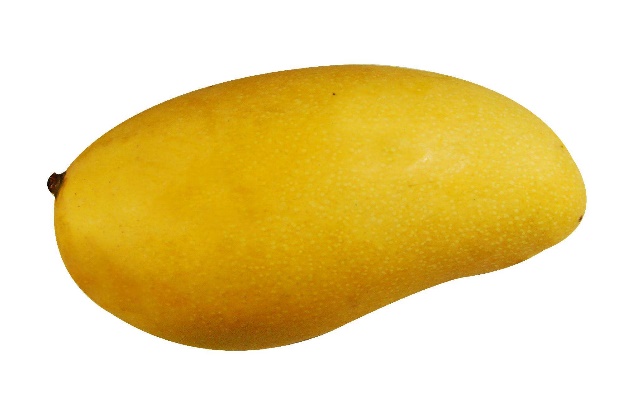
可以用一个例子来说明一下有监督学习的过程，假设现在有一个装满了不同水果盘，如图4上半张图所示，那么这个水果盘里面的水果就是我们的训练集，我们会做如下训练：如果水果的形状是椭圆形的且表面光滑颜色为黄色，那么它会被标记为芒果；如果水果形状是不规则的、表面有一个个小点、表皮为红色且顶端带有绿色叶片的，那么它会被标记为草莓。假设在我们训练了这个数据之后，给这个模型一个单独的水果，如图4下半张所示，要求识别这个水果的类型。由于模型已经从原来的果盘中学到了水果的各种特征，它将识别新给的这个水果是黄色的、表皮光滑且为椭圆形，因此归类为芒果。这就是有监督学习将训练数据（给定的果盘中的水果）学习到的特征结构等知识用于辨别测试数据（用于被识别的芒果）的过程。分类问题是有监督学习典型的应用场景，它的训练数据是一些已经分好类的数据，输出变量是类别，例如“红色”或“蓝色”或“绿色”。

图 4 上半张图是作为训练集装有各种水果的水果盘，下半张图是需要通过模型判别类型的给定水果

2.2.2 n元语法

n元语法（N-Gram）是一种概率语言模型算法，主要用于根据前n-1个词推断文本中下一个词是什么。Gram就是一个文本片段，按照事先设定的阈值进行所有出现的Gram的频度的统计，我们可以得到这个完整文本的特征向量。

n元语法基于一种假设，那就是一段文本中每个词的出现只与它前面的n-1个词有关，因此每个词出现的概率就是该词与前面n-1个词同时出现的概率，而整段文本出现的概率就是每个词出现概率的乘积。

图 5 Google搜索中的N-Gram应用

n元语法中的n对整个模型有着重要的影响。当n越大时，对每个词的约束性就更大每个Gram里面含有的信息就越多，Gram的维度就越大，能够更好的分辨每一个词，但是会使得形成的每个词的特征向量更加稀疏。当n越小时，每个词的特征向量更丰富，但是约束性条件更少。举例说明，假设目前有这么一句话：“你好吗”，那么词表中所有可能的2-Gram的总数就是32=9个，分别是“你你”，“你好”，“你吗”，“好好”，“好你”，“好吗”，“吗你”，“吗好”，“吗吗”。而如果是3-Gram的话，词表中的词向量数量就有可能为33=27.

n元语法有一个非常常见的应用就是搜索引擎或者输入法中的备选词联想，如图5所示，我们在进行Google搜索时，只输入前两个单词“白天”，他就会自动联想出我们可能输入的下一些词是什么，且越靠前的说明其查询频率越高，这是根据词库中储存的N-Gram词向量实现的。假设我们使用的是2-Gram语言模型来预测下一个词，那么在这个例子中，Google的词库储存的词向量存在以下信息：p(“不懂夜的黑”|“白天”)>p(“黑夜交错”|“白天”)>……>p(“天羽”|“白天”)。这些概率值就是根据用户输入的查询数据用上面介绍的N-Gram方法求出来的

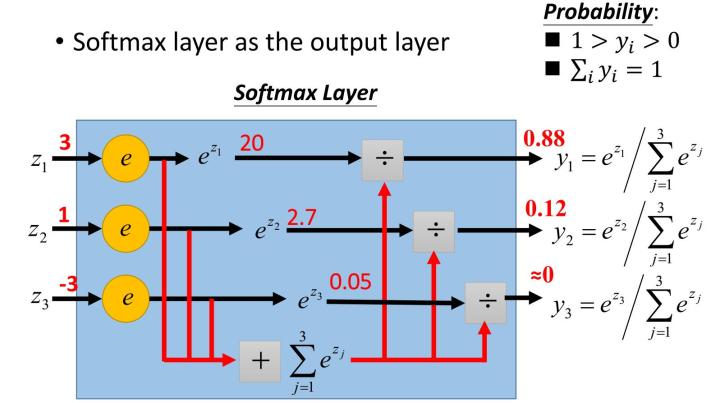
2.2.3 SoftMax

图 6 SoftMax得到最后的分类概率

SoftMax函数[[3]](#endnote-3)，又称归一化指数函数，它将一个非标准化的实数向量归一化到一个概率分布里面。在应用SoftMax之前，实数向量的元素可能出现负值或大于1，应用之后，使得每个元素的范围都在[0,1]之中，并且所以元素的和为1。假如我们有一个数组，数组中的元素个数为n，那么，第i个元素的SoftMax值通过公式(1)计算：

(1)

通过一个例子来说明SoftMax的工作原理，如图6所示[[4]](#endnote-4)，当我们获得了前面三个动作{a,b,c}经过一系列处理后的输出{3,1,-3}之后，通过上面的公式算出每个值对应的SoftMax值是{0.88,0.12,0},因此我们会选取概率最大的也就是0.88对应的那个动作a来作为最后的结果。那么如果对应的多分类问题，想选取两个动作，就找到概率最大的两个{0.88,0.12}对应的动作{a,b}。

2.3 自然语言处理

自然语言处理[[5]](#endnote-5)是计算机科学、信息工程和人工智能交互的一个子领域，主要涉及计算机和人类（自然）语言之间的交互，特别是如何编写处理和分析大规模自然语言数据的计算机程序。自然语言处理又可以分成很多个子任务，如词性标注、句法分析、文本分类、命名实体识别、信息检索等等。我们主要涉及的是词性标注、句法分析和命名实体识别。

词性标注是为了给句子中的每个单词确定正确的词性，比如动词、名词、形容词、代词等等。而句法分析则假设每句话里面只有一个单词是核心词而存在，其他词都直接或间接依赖于核心词，即修饰词。每个词能且只能依赖于另一个词而存在，不能同时依赖于多个次。句法分析就是将句子中的依赖关系（如表1所示）标记出来。命名实体识别则是将句子中具有特定意义或指代性的实体，比如公司名、地名、人名、时间、专有名词等等，按照业务要求从非结构化的句子中识别出来。

|  |  |  |  |
| --- | --- | --- | --- |
| 关系 | 标签 | 描述 | 示例 |
| 主谓关系 | SBV | 主语-谓语 | 你给我一张纸（你🡨给） |
| 动宾关系 | VOB | 直接宾语， verb-object | 你给我一张纸（给🡪纸） |
| 间宾关系 | IOB | 间接宾语，indirect-object | 你给我一张纸（给🡪纸） |
| 定中关系 | ATT | attribute | 蓝精灵（蓝🡨精灵） |
| 介宾关系 | POB | preposition-object | 在大海中（在🡪中） |

表 1 句法分析中的依赖关系，只列出部分作为参考

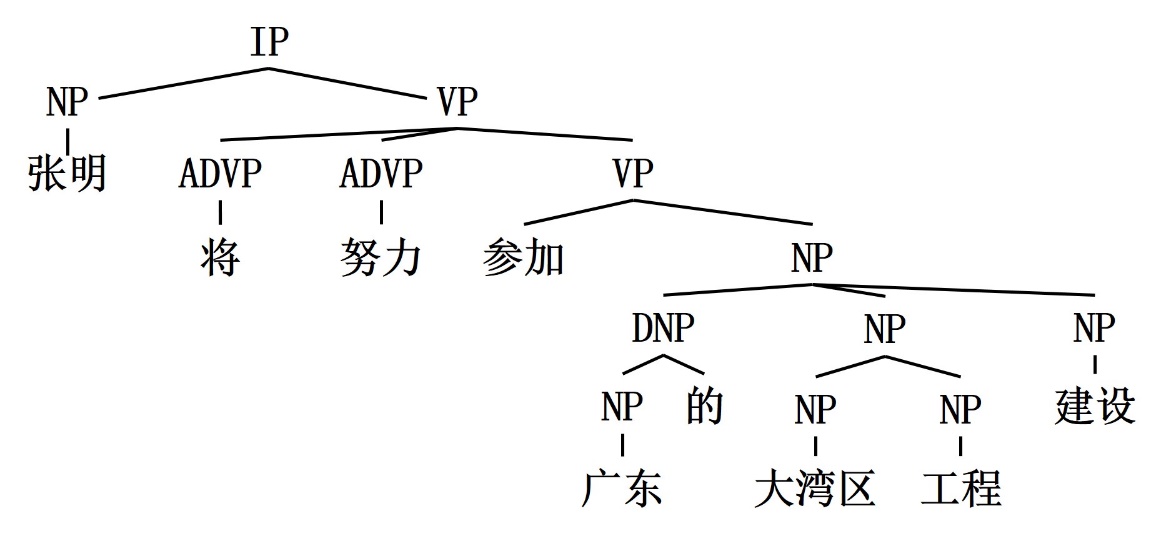


图 7 词性标注示例

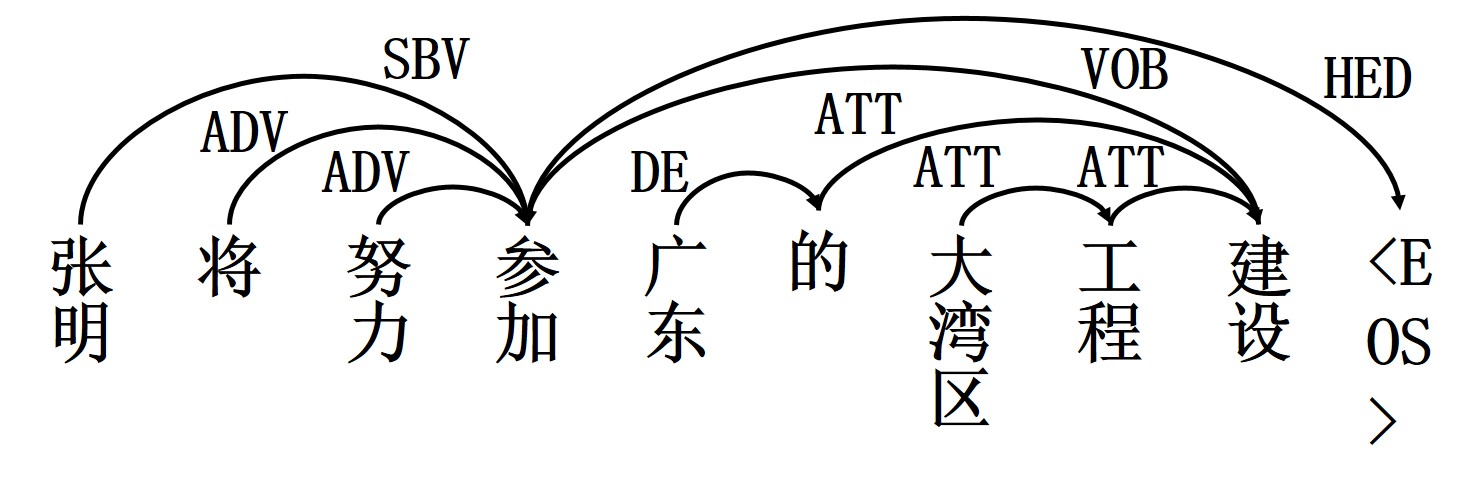
假设现在我们要分析这样一句话：“张明将努力参加广东的大湾区工程建设”，那么我们可以先用命名实体识别找出其中的潜在实体：“张明”、“广东”、“大湾区”，这些实体可以为我们提供重要的信息。其次，我们可以进行词性标注，得到每个单词的词性，如图7所示。词性标注可以展示为树的形式，由终结点，非终结点和短语标记组成，非终结点需要进行进一部分解，一系列终结点构成了一个短语。句法分析结果则如图8所示，依赖关系用有向弧来表示，从修饰词指向核心词。

图 8 句法分析示例

2.4 静态程序分析

在计算机科学中，程序分析[[6]](#endnote-6)是为了自动化的分析计算机程序正确性、稳健性、安全性等指标的过程，主要分为两种方式：静态程序分析和动态程序分析。静态程序分析指的是在程序不运行的状态下分析程序的源码或机器码来完成对代码的检查，而动态分析则恰恰相反，需要在程序执行的时候进行代码的分析，可能会对程序的运行效率造成影响。两种分析方法并不是完全互不相容的，他们都有各自的优缺点，在一定程度上能起到相辅相成的作用。静态程序分析会考虑到程序所有要执行的路径，而动态分析只检查得到其中某一条被执行的路径；但动态分析能够得到运行的真实值，相对来说更加精确。

我们主要涉及到的是静态程序分析技术，它包含了很多可以单独使用的子方法，如数据流分析、控制流分析、语法树分析等等。语法树分析[[7]](#endnote-7)是源代码语法结构的一种抽象表示，它的每个节点表示的是源代码的其中一个语义结构，但它是抽象的，不是所有源代码的细节都会在树状结构中作为一个节点被体现出来，比如if-else 判断条件，在抽象语法树中是作为一个分叉去表示的。数据流分析则是通过收集程序运行到不同地方时各个值的变化信息，构成一个数据的流向图。

2.5 本章小结

我们介绍了一系列在研究中运用到的软件技术，这些技术来自各个领域，包括代码分析、自然语言处理、机器学习、深度学习等，我们对这些技术进行了应用，巧妙地穿插在研究的各个阶段。各个领域的知识并无边界，都可以相辅相成，完成跨领域的联合研究，这不仅体现了我们对新技术新方法的学习，还能够使我们的研究跟上最新的研究水平，不会老旧落后，也能得到更好更精确的结果。

第三章 整体框架

3.1 框架概览

整体框架如图所示，分为两个阶段：图9的大类分类阶段和图10的细分小类分类阶段，两个阶段相互连接、共同工作。

在大类分类训练阶段，我们首先得到训练数据，训练数据已经是每句话对应其大类类别的形式，然后需要将训练数据进行平衡，这里主要平衡的是非约束性语句的数据，使其能够和其他类别差别不是这么大。接着就进入了FastText[9]的整个训练流程，过程是先进行N-Gram的提取，将语料的词序列和提取出来的N-Gram词向量一起作为隐层的输入，输出经过SoftMax得到最后训练模型结果。

在大类分类预测阶段，我们将还未分类的原生句子作为训练好的FastText模型输入，模型会自动为我们预测出句子所属的类别。而我们可以通过FastText得到SoftMax分成各类别的概率，当预测类别的概率高于某个阈值时，我们将此结果重新作为训练数据进行模型的训练。如此迭代的将新数据作为训练数据更新模型，可以使模型接收到更多种多样的语句样式而提高其准确性和完备性。

在细分小类分类阶段，我们先将前面大类分类过的结果进行自然语言的处理，得到其中蕴含的命名实体、词性、词与词之间的依赖关系，这些信息是为之后的模式识别作辅助。然后，我们人工的观察所有带有细分小类标注的约束性语句，总结所有细分小类中可能存在的匹配模式。通过这些匹配模式，我们就能自动化的得到属于这个细分小类的约束性语句。

3.2 约束性语句分类

在参考了Martin Monperrus等[10]人对于约束性语句的分类之后，我们结合自己对数据的观察和工具的特性，得到了一个有少许不同的约束性语句分类模型。

3.3 数据平衡

在API文档的预料中，普遍存在着数据量不平衡的现象，比如在Java、JFace和Commons.collections的API官网文档里面，约有5%、7%、12%的约束性语句，这么大的比例差距会使得在训练非约束性语句和约束性语句的不同特征时造成比较大的影响。在机器学习分类模型的训练时，尽可能使得所有分类数据比例平衡是构建更加准确的分类模型的好办法，因此我们就要进行数据平衡[11, 12]工作。

3.4 FastText

3.5 迭代式模型训练

3.6 自然语言处理

3.7 启发式模式识别

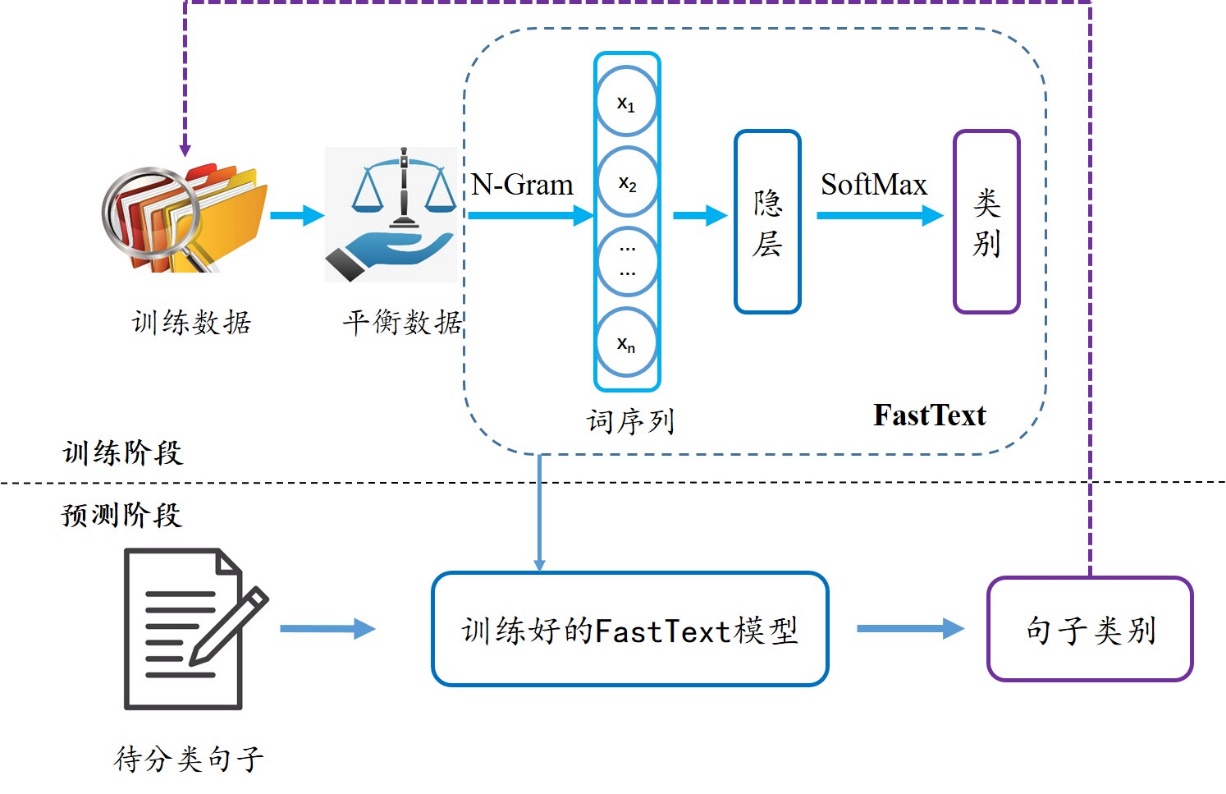


图 9 大类分类框架图

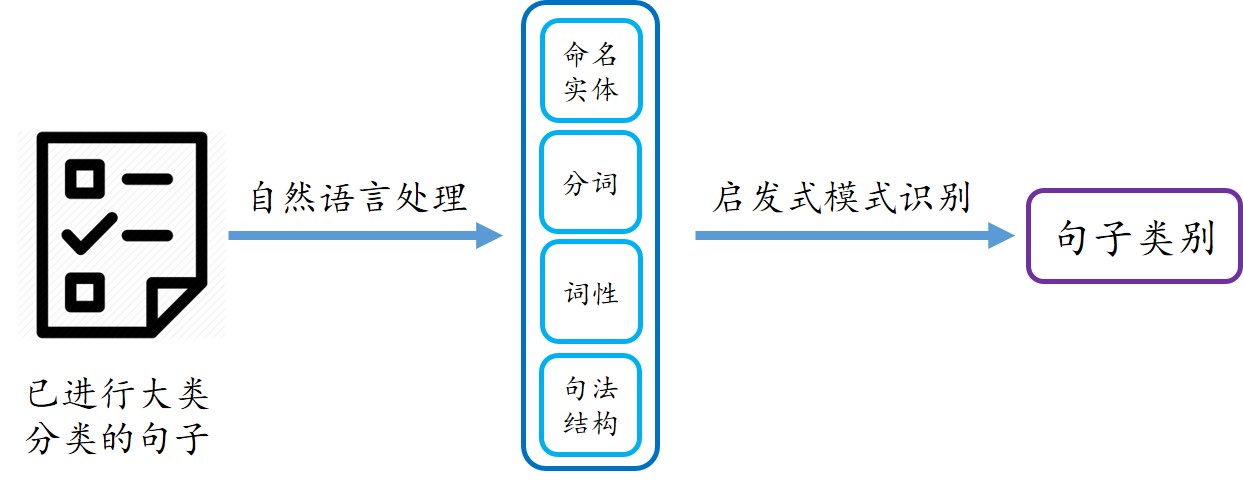


图 10 细分小类分类框架图

|  |  |  |
| --- | --- | --- |
| 大类 | 细分小类 | 匹配模式 |
| 方法调用相关约束性语句 | 非空 | 1. @exception|@throws (be|is|are) (equal|equivalent to) null 2. non-null 3. (never|not) (be) null 4. (neither|none of) (be|is|are) (equal|equivalent to)\* null |
| 返回值 | 1. return|result（从开头匹配） 2. Return|Result|@return 3. (return|returns|returned|returning) (a|the|this) 实体 4. (value|parameter|object|实体)s? (is|be|are)? (return|returns|returned|returning) 5. (方法名) (must|should|can|could|may)? (return|returns|returned|returning) (a|the|this)? (实体|true|false|element|object|null) 6. (result|results|returned 实体|实体returned) (must|should|can|could|may)? (is|be|are) 7. (must|should|can|could|may)? return |
| 方法可见性 | 1. be necessary to use method 2. (this|it) (be|is|are) called (from|outside) 3. call (method|方法名) directly 4. ((should not)|(discouraged from)) call (method|方法名) 5. Method (should)\* not be (relied|used|referenced) 6. method is for … only 7. 方法名 should (no|not)\* be (relied|used|referenced|called|instantiated) 8. may (call|instantiate) (method|class) 9. (method|class) (may|not) be instantiated |
| 抛出异常 | 1. Exception (be|is|are) thrown 2. @exception|@throws 3. throw Exception 4. raise Exception |
| 允许空值 | 1. (@param)\* (can|could|may) ((be|is|are) (equivalent|equal) to) null 2. null (be|is|are)\* ignored 3. null (results|use|be passed in) 4. or null|null means|this parameter is null|null returns|null otherwise|null indicates|If null|Specify null to |
| 字符串格式化 | 1. (must|should)\* (be|is|are|start with|starting|ends with) (".\*"|'.\*'|string) 2. string must (contain|be) 3. the format (follow|(be taken from)) 字符串编码类型 4. must conform to字符串编码类型 5. string must be digit 6. sequence of characters must represent |
| 数值范围 | 1. (equals|>|<|=|<=|>=|<>) 数字或某特定值 2. (be|is|are) (less|greater|larger|equal|equivalent|below) (than|to) 数字或某特定值 3. (be|is|are) (less|greater|larger|below) 4. (be|is|are) (at least|at most) 数字或某特定值 5. (be|is|are) (range|between) 数字 (and|to|-|~|,) 数字 6. 实体and实体(be|is|are) the same 7. (be|is|are) (in|out of|outside) range 8. (be|is|are) (in|out) of bounds 9. (be|is|are) 数字或某特定值 10. one of … (constant|value) … (such as|:) 数字或某特定值 11. (either of|one of|(be|is|are)) ({(数字,XXX)+}| 数字或特定值各种集合形式) 12. (be|is|are) (negative|positive|non-negative|non-positive) |
| 方法参数类型 | 1. (be|is|are) of type 类型实体 2. subclass of 类型实体 3. (@throws|@exception|parameter|property|value|@param|key|element) (must|should)\* (be|is|are|implement) 类型实体 4. (be|is|are) (equivalent|come from|equal to|a class derived from|an instance of) 类型实体 5. (key|value|element|property|argument)s? (must|should) implement 类型实体 |
| 方法调用顺序 | 1. (after|before) .\*((being used)|calling|using|invoking|called) 2. (should|must) be called (whenever|when|associated with) 3. (invoked|called|call|calls) .\*(before|after) |
| 继承相关约束性语句 | 方法重写 | 1. (subclass)(es)? (reimplement|override|extend|override|extend) 2. (extend|override) method 3. method (override|overridden|re-implemented|extended|implemented) |
| 可被继承的类 | 1. (instantiate|extend) (class|interface) 2. (implemented|implement) (interface) 3. (may|should|can) subclass 4. (it|class|interface) (be|is|are) (subclassed|extended|implemented) |
| 方法继承 | 1. (call|invoke) (super.|(.\*super)) 2. (super.) (should|must) be (called|invoked) |
| 继承并调用方法契约 | 1. (subclasses|subclass) .\*(call) .\*(method|constructor) 2. If you override this method, then you should … |
| 其它约束性语句 | 可替换版本 | 1. @deprecated 2. be (preferable|more efficient) to (use)? 新版本 3. 新版本Instead 4. 旧版本be replace 5. 新版本be a better choice |
| 同步异步问题 | 1. thread-safe 2. synchronized |

第四章 实验设计

第五章 实验结果

1．大类分类结果

2．细分分类结果

3．为什么要分成两部去进行

第六章 应用

第七章 相关工作

第八章 总结与未来工作

参考文献

**参考文献**

[1] PICCIONI M, FURIA C A, MEYER B. An Empirical Study of API Usability: Acm, 2013[C].

[2] MAALEJ W, ROBILLARD M P. Patterns of Knowledge in API Reference Documentation[J]. IEEE Transactions on Software Engineering, 2013,39(9):1264-1282.

[3] HE J, ZHANG J, REN Z, et al. An Unsupervised Approach for Discovering Relevant Tutorial Fragments for APIs: IEEE/ACM International Conference on Software Engineering, 2017[C].

[4] PETROSYAN G, ROBILLARD M P, MORI R D. Discovering Information Explaining API Types Using Text Classification: IEEE/ACM IEEE International Conference on Software Engineering, 2015[C].

[5] EARLE R H, ROSSO M A, ALEXANDER K E. User preferences of software documentation genres: International Conference on the Design of Communication, 2015[C].

[6] LI H, XING Z, XIN P, et al. What help do developers seek, when and how? Working Conference on Reverse Engineering, 2013[C].

[7] YU Z, GU R, CHEN T, et al. Analyzing APIs Documentation and Code to Detect Directive Defects: IEEE/ACM International Conference on Software Engineering, 2017[C].

[8] MAALEJ W, ROBILLARD M P. Patterns of Knowledge in API Reference Documentation[J]. IEEE Transactions on Software Engineering, 2013,39(9):1264-1282.

[9] JOULIN A, GRAVE E, BOJANOWSKI P, et al. Bag of Tricks for Efficient Text Classification[J]. 2016.

[10] MONPERRUS M, TEKES E, MEZINI M. What should developers be aware of? An empirical study on the directives of API documentation[J]. Empirical Software Engineering, 2012,17(6):703-737.

[11] KAMEI Y, MONDEN A, MATSUMOTO S, et al. The Effects of Over and Under Sampling on Fault-prone Module Detection[J]. First International Symposium on Empirical Software Engineering & Measurement, 2007,11(1):196-204.

[12] HAN H, WANG W Y, MAO B H. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning: International Conference on Advances in Intelligent Computing, 2005[C].

**校对报告**

当前使用的样式是 [中华人民共和国国家标准\_GBT\_7714-2015]

当前文档包含的题录共13条

有0条题录存在必填字段内容缺失的问题

所有题录的数据正常

致谢

1. https://en.wikipedia.org/wiki/Application\_programming\_interface [↑](#endnote-ref-1)
2. https://swagger.io/blog/api-documentation/what-is-api-documentation-and-why-it-matters/ [↑](#endnote-ref-2)
3. https://zh.wikipedia.org/wiki/Softmax%E5%87%BD%E6%95%B0 [↑](#endnote-ref-3)
4. https://www.slideshare.net/tw\_dsconf/ss-62245351?qid=108adce3-2c3d-4758-a830-95d0a57e46bc&v=&b=&from\_search=3 [↑](#endnote-ref-4)
5. https://en.wikipedia.org/wiki/Natural\_language\_processing [↑](#endnote-ref-5)
6. https://en.wikipedia.org/wiki/Program\_analysis [↑](#endnote-ref-6)
7. https://en.wikipedia.org/wiki/Abstract\_syntax\_tree [↑](#endnote-ref-7)