软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(5):1498−1509 [doi: 10.13328/j.cnki.jos.005718]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

企业级海量代码的检索与管理技术*

刘志伟[1], 邢永旭[1], 于 澔[1], 李 涛[2], 张晓东[3]

[1](百度(中国)有限公司,上海 201210)

[2](百度在线网络技术(北京)有限公司,北京 100193)

[3](西安交通大学 计算机科学与技术系,陕西 西安 710049)

通讯作者: 刘志伟, E-mail: liuzhiweihome@gmail.com

摘 要: 在大型IT企业中,尤其像Google或者百度,代码搜索已是软件开发过程中不可或缺且频繁的活动,其通过借鉴或复用已有代码,加速开发过程的速度.多年以来,已有大量的研究人员关注代码搜索,且设计出很多优秀的工具.但是已有的研究和工具主要是在小规模或者编程语言单一的代码数据集上,没有从企业实际搜索需求出发,且对用户的查询输入也有所限制,尚缺少一套针对企业级海量代码的检索与管理技术方案.提出了一套企业级海量数据代码搜索引擎的方案和系统实现,面向开发过程中用户最直接的需求,通过离线分析与在线分析,完成对海量代码库的索引构建与检索.其中,离线分析负责代码相关数据的获取与分析、构建索引集群.在线过程负责变换用户的query、对搜索的结果进行高级排序、生成摘要.本系统部署在百度代码库上,为数十 TB 级的 Git 代码库构建了索引,平均一次检索时间在 1s 之内.在百度推出应用以来,访问量逐步增加,现每周平均用户有数千人,每周查询平均有数万次,广受百度工程师好评.

关键词: 代码搜索;索引;排序;海量代码

中图法分类号: TP311

中文引用格式: 刘志伟,邢永旭,于澔,李涛,张晓东.企业级海量代码的检索与管理技术.软件学报,2019,30(5):1498−1509. http://www.jos.org.cn/1000-9825/5718.htm

英文引用格式: Liu ZW, Xing YX, Yu H, Li T, Zhang XD. Retrieval and management technology for industrial-scale massive code. Ruan Jian Xue Bao/Journal of Software, 2019,30(5):1498−1509 (in Chinese). http://www.jos.org.cn/1000-9825/5718.htm

Retrieval and Management Technology for Industrial-scale Massive Code

LIU Zhi-Wei[1], XING Yong-Xu[1], YU Hao[1], LI Tao[2], ZHANG Xiao-Dong[3]

[1](Baidu (China) Co., Ltd, Shanghai 201210, China)

[2](Baidu Online Network Technology (Beijing) Co., Ltd, Beijing 100193, China)

[3](Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: In large IT companies, especially like Google or Baidu, code search is an indispensable and frequent activity in the software development process, which speeds up the development process by learning or reusing existing code. Over the years, a large number of researchers have focused on code search and designed many excellent tools. However, the existing research and tools are mainly on a small-scale or single programming language code data set, not from the actual requirement of industries, and the user's query input is also limited; there is still a lack of a set of industrial-scale massive code retrieval and management technology solutions. This study proposes a code search engine solution and system implementation based on industrial-scale massive data, oriented to the most direct needs of users in the development process, through offline analysis and online analysis, complete the index construction and retrieval of massive code

* 基金项目: 国家重点研发计划(2018YFB1003900)
Foundation item: National Key Research and Development Program of China (2018YFB1003900)
本文由智能化软件新技术专刊特约编辑申富饶教授和李戈副教授推荐.
收稿时间: 2018-08-31; 修改时间: 2018-10-31; 采用时间: 2018-12-13

base. Among them, offline analysis is responsible for the acquisition and analysis of code-related data and building an index cluster. The online process is responsible for transforming the user's query, sorting the results of the search, and generating a summary. The system is deployed on the Baidu code base, and the index is built for dozens of TB-level Git code bases. The average retrieval time is within 1s. Since the launch of Baidu's application, the number of visits has gradually increased. There are thousands of users per week and tens of thousands of times searching. The system is widely praised by Baidu engineers.

Key words: code search; indexing; rank; massive code

软件开发过程中所涉及到的活动非常广泛,比如需求编写、代码开发、编译、部署等等.研究发现,代码搜索是软件开发活动中最为频繁的活动之一[1].比如,Google 工程师每工作日使用代码搜索多达 12 次[2].为了避免重复造轮子的过程,尤其在企业开发实践中代码搜索更为普遍,因为在企业里构建软件几乎都不是从零开始,而是通过复用大量代码完成构建.因此,寻找程序员感兴趣的代码片段或者 API 使用方式等代码搜索活动变得非常重要[3].

然而,构建企业级的代码搜索系统是非常困难的.

- 首先,代码量巨大,Google 代码库超过 20 亿行[4],百度使用 Git(https://git-scm.com/)进行代码管理,Git 存储代码规模空间就多达 TB 级.更为关键的是,代码量时刻在急速增长中,百度平均每天新增代码 153 万行,删除代码 63 万行.因此,企业级的代码搜索工具必须能处理海量的代码数据,并且能保证代码更新的时效性,这是极为困难的.
- 其次,大规模的企业里所使用的开发语言往往较为繁杂,在百度,包括各种脚本语言、数据库语言等在内,使用了多达 30 种以上的编程语言.由于不同语言的语法分析不同,并且不能限制在单语言内进行搜索,使得在企业里提供针对语法代码检索变得非常困难.
- 再次,不同企业对代码的管理方式不同,比如,百度利用多分支与多 tag 的方式进行代码管理.虽然 Git 会优化多分支与多 tag 的存储,但是如果不加以对于重复的处理,将会造成代码索引的迅速膨胀,使得搜索引擎工具很难容纳如此海量的索引数据.
- 然后,企业内对如何提升海量代码搜索质量的研究仍不足,比如在百度,通过用户输入一个单词或几个单词组成的 query(query 是用户进行搜索的关键词,比如方法名或者类名),在数十亿行的代码文件进行查找,返回给用户想要的结果是非常难的.因此,必需要利用丰富的特征信息,对大量代码搜索结果进行排序.此方面的研究尚比较缺乏.
- 最后,如何保证检索效率,能够在百度 TB 级的索引数据中 ms 级内完成检索,并且呈献给用户,也极具挑战性.

互联网搜索引擎比如百度、Google 等也有检索代码的功能,但是还没有考虑代码本身特征进行索引排序等策略,效果还不理想.Google 内部有强大的代码搜索引擎,兼顾易用性以及准确性,当前对外可以通过 chromium (https://cs.chromium.org/chromium/) 体验.商业的代码搜索软件,比如 Krugle(http://www.krugle.com/) 以及 insight.io 等,通过关键词进行搜索,以及代码托管网站,比如 Github(https://github.com/)、Bitbucket(https://bitbucket.org/)等也提供了通过关键字进行代码搜索功能,这类工具的搜索准确性还有待提升.同时,已有大量代码搜索的研究工作[5−11],有的工作对用户查询的输入形式进行约束,比如 Kashyap 等人[6]开发的 Source Forage 引擎以 C/C++函数的形式作为输入,以便更好地利用语法与语义,更精确地对相似代码进行查找.有的工作是定义规约化的语言,让用户描述自己想要搜索的需求,从而能更容易地找到用户需要的结果.然而,这种约束用户输入复杂的查询条件的方法在企业里往往不现实.与自然语言的搜索引擎一样,复杂的输入将会限制用户的使用.在企业的实际开发过程中,用户只是会输入简单的一个或几个单词组成的查询语句,比如搜索函数 sort,期望找到大家普遍使用、功能为排序的函数;如果用户输入 code search,那么可能搜索 code search 这个系统的代码库,或者出现在文件路径上的这个单词,其次才可能是包括这个词的文件.另外,比如近期的研究中,Gu 等人[5]是以自然语言作为 query,利用神经网络学习的方法搜索到与自然语言语义上所匹配的代码片段,达到利用自然语言检索到实现相关功能代码段的目的.

然而,从企业级软件开发的角度看,绝大部分是逻辑较为复杂的业务代码,而不是可以用自然语言具名描述的算法或者过程;同时,企业开发中,往往由于业务进度压力,代码缺乏丰富的注释,文档也经常落后代码迭代的速度,这也使得无法训练出解释代码语义的模型.根据 Google 的研究[2],在实际研发过程中,工程师搜索代码的需求主要包括如何使用一个 API、API 如何实现、API 定义在哪里等,由此可见,企业级软件开发过程中代码搜索这一环节,工程师搜索出发点是已经有关键词.此外,每个 Google 工程师每次搜索平均使用 1.85 个关键词(中位数是 1 个关键词).因此,针对企业级的海量代码,与自然语言的搜索引擎一样,利用简单关键词进行检索匹配代码的方案更为实用与迫切.

针对百度的实际需求以及企业开发中的现状,本文提出了一套企业级的代码搜索的实现方案,包括离线建库部分和在线端用户搜索部分.离线部分负责将不同的代码相关的源数据,比如 Git 代码库,以批量或实时增量等方式进行获取,然后经过数据分析,比如进行语法识别、语法解析等,持久化到分布式海量数据库(Elasticsearch 集群)中.在线端负责完成用户的实时代码搜索,主要包括:首先是针对用户的请求进行鉴权,保证权限安全;其次,在确定搜索请求安全后,对用户的 query 关键词进行变换处理,以搜索更多的相关的代码;再次,向分布式数据库进行请求,并获得检索到的结果;最后,对检索结果进行排序,利用高级定制化的排序算法返回给用户更合理的结果.另外,本系统使用缓存管理以提高搜索的性能,利用摘要生成模块以为每条结果提供描述.

此代码搜索系统自推出上线以来,索引了百度 TB 级的 Git 库,并能在分钟级准确索引变更代码,支撑平均每天新增代码 153 万行、平均每日删除代码 63 万行的索引,能够针对文件、代码库、代码提交时间等进行搜索.该系统在百度内部使用过程中,每周平均为数千名工程师提供代码搜索服务,每周平均完成数万次的搜索,平均每次检索时间为 1s 以内.

根据调研,本工作是首次站在企业海量的代码数据上,提出以及应用包括针对代码的 query 变换、提出了更多代码相关的特征并应用其进行排序的方法、针对代码搜索结果的排序评估方法、摘要生成等创新性的技术,形成一套实用的代码搜索方案.本文的主要工作贡献如下.

- 在工业级的海量代码上,提出了代码的 query 变换,应用新的代码相关特征进行排序,且提出代码排序的评估方法;同时,结合摘要生成,使得根据用户简单的输入即可给出用户想要的搜索结果.
- 并且针对此方案,实现了一套实用效果极佳的企业级系统,索引了百度 TB 级的代码,且能在分钟级处理每日百万级代码行的变更.

本文第 1 节概述系统框架并简介应用到的技术.第 2 节介绍代码搜索系统的工作流程.第 3 节对系统进行实际应用的评估.第 4 节列举系统当前不足之处.第 5 节是相关的研究工作.第 6 节进行总结与未来工作的展望.

1 方案框架概述以及应用到的技术简介

本文提出的企业级代码搜索系统主要由离线端和在线端两部分组成.离线端采用 Elasticsearch 作为持久化存储,Elasticsearch 是一个基于 Apache Lucene(TM)的开源搜索引擎,它提供了一个分布式多用户能力的全文搜索引擎,是当前流行的企业级搜索引擎.离线端主要目标是将不同的代码源数据以批量或实时增量等方式获取,同时导入代码等相关数据,然后经过数据分析,持久化储存到 Elasticsearch 集群中.为了实现此目标,离线端分为 3 层:数据导入、数据分析、索引管理.

- 数据导入模块主要负责适配各种代码数据源.在百度的实际应用中,主要以 Git 仓库为主,除了批量获取数据,还对数据源进行监听以实时导入变更代码库.
- 数据分析模块主要由多个流水线解析器组成,解析器主要完成编码检测、代码提交信息萃取等工作.通过对代码以及其衍生数据(比如编码信息、作者提交信息、文本属性等)的解析,获得抽象语法树(abstract syntax tree,简称 AST)和代码相关数据,其中,AST 解析由 Eclipse CDT/JDT 完成,Eclipse CDT/JDT 是基于 Eclipse 平台并提供 C/C++(CDT),Java(JDT)集成开发环境的插件.
- 索引管理模块主要负责海量数据在分布式情况下的持久化存储与管理,主要采用 Elasticsearch 作为数据索引存储.索引化数据除了支持代码检索,也应用在了代码跳转上.由于代码跳转是另外的话题,本文

不详细展开.

在线端主要负责实时响应用户关键词搜索,实现包括接口管理、搜索管理、队列管理.

- 接口管理模块主要包括请求鉴权和数据收集,其中,请求包括两类:一是单个用户的关键词搜索,二是来自各个产品线的 API 调用(比如代码重复度查询).百度产品线众多,不同的产品代码有着严格的保密等级,因此,标识用户身份、防止机密代码泄露,做到代码安全是搜索系统所必须的.因安全相关与本文关联不大,这里暂略过.
- 搜索管理主要包括查询变换、Ranker、缓存管理、摘要生成模块.此外,搜索管理模块还通过分析用户搜索行为数据,反馈给 Ranker,提高了搜索质量和相关性.
- 队列管理主要负责对 Elasticsearch 分布式数据库的并行访问与参数优化.

如图 1 所示,当用户在前端发生一次代码提交后,系统监听到代码变更消息,通过增量的方式完成整个流水线,以达到实时性的目标.当用户在检索端尝试搜索时,系统适当调整 query 关键词后,根据不同用户权限,通过在缓存、全文索引、语法索引等不同 index 中搜索,在原始召回的结果中应用 ranker 中的排序策略,并在最终的结果中提炼最相关的摘要片段返回给用户,以完成整个检索过程.

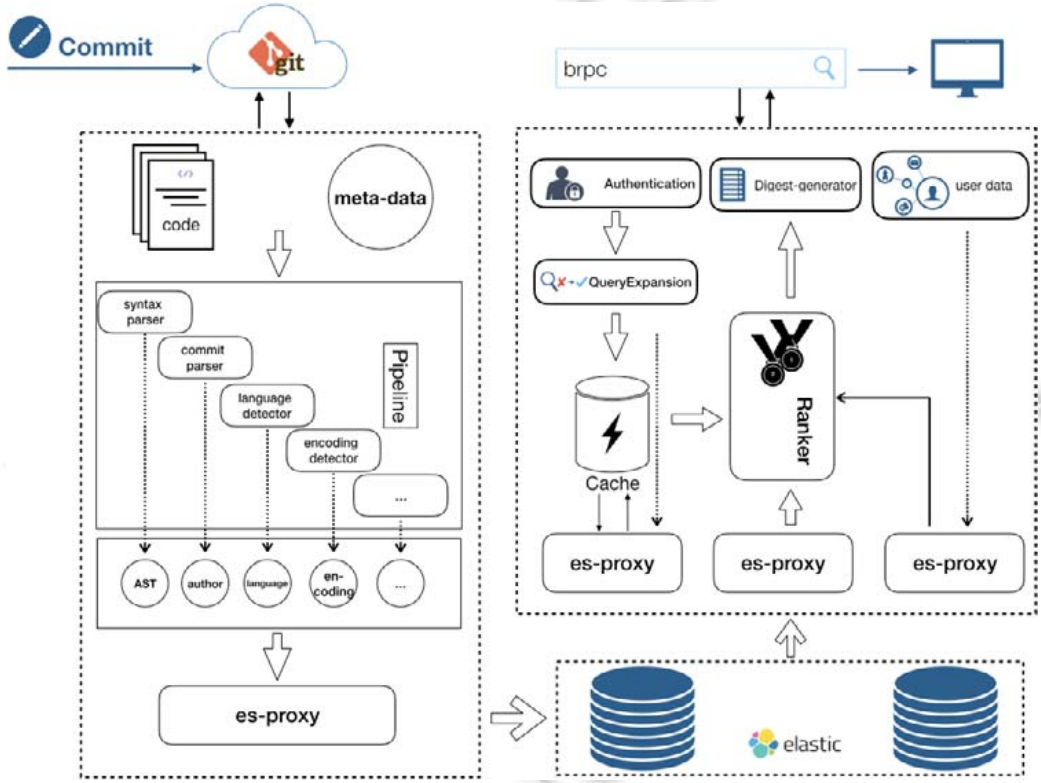

Fig.1 Interactive process chart of Baidu code search

图 1 百度代码搜索用户交互流程图

2 方案实现

2.1 代码数据爬虫与代码索引

爬虫是搜索引擎系统中不可缺少的组成部分,代码检索系统的爬虫同互联网爬虫一样[12−14],从远端系统获取数据.代码数据爬虫负责是从各种代码管理系统获取海量的代码数据,比如 SVN(https://subversion.apache.org/)、Git,然后进行代码索引.除此之外,在企业中数据安全也是数据爬虫端遇到的重要问题,尤其是代码数据,

是企业的核心资产,代码数据数据的爬虫与索引同样需要极高的安全保障,不能使得爬下来的代码发生泄漏的可能.由于安全是另外非常重要的话题,且并不是本文的重点,因此这里并不会展开阐述.

代码数据爬虫需要解决的主要问题是数据的时效性和数据的准确性:时效性指在海量的代码数据情况下,比如百度数 TB 以上的 Git 代码库基础上,还能实时处理平均每日新增与删除的数百万代码数据,能够确保新变更的代码数据能够被快速索引到,以便用户把代码合入代码库后,即可进行搜索;数据的准确性是指在如此大规模的数据变更索取下,索引的数据不能和真实的数据有所偏差,否则会造成用户的使用障碍.因此,本文系统所设计的代码数据爬虫分为 3 个服务:批量爬虫服务、定时爬虫服务、实时爬虫服务.

批量数据爬虫负责遍历全代码库进行索引,主要的用途是第 1 次构建代码搜索,或者由于代码索引技术的升级,需要再次进行全库重新索引,比如在第 1 次索引代码的时候没有进行代码变更时间的索引,所以需要再次全库重新索引,这时都需要利用批量数据爬虫服务.实时索引是代码库合入事件触发的,每次只会索引变更的代码文件,因此能保证在分钟级甚至更少的时间索引完成.定时索引的任务是为了确保数据的准确性,因为在整个系统中并不能保证服务是 100%可用的,我们必须进行容错处理,比如事件触发可能没有通知到爬虫系统,或者爬虫系统所在的机器异常等情况.基于系统是可能出错的前提下,需要保证代码索引数据的准确性,就需要一个额外的定时索引任务,它会对比索引的数据与真实代码数据之间的差异,处理可能由于错误引发的数据异常.为了提高代码数据爬虫索引代码的效率,数据爬虫系统被设计为是多个进程、多个线程在多物理机上执行,能够很好地使得机器的负载均衡.

代码数据爬虫系统除了关键的 3 个服务以外,还有一个特殊的代码去重的服务.因为正如在第 1 节简介中所描述的,在百度是通过分支、tag 对代码进行管理,如果针对分支、tag 进行代码索引,就需要跟默认开发分支或者称为主干代码进行去重,因为分支、tag 的代码跟主干代码会非常大量的重复代码.对大量的重复代码进行索引,不仅造成存储空间的浪费,同时降低了搜索性能,并且如果用户搜索出大量的重复代码,并没有什么意义,这降低了用户的体验,所以必须要对索引代码进行去重.对索引代码去重有多种方式,首先推荐的是在代码索引之前,利用 Git 命令查找出,不同分支、tag 与主干代码的区别,然后只对这些有区别的代码进行索引.这种方法往往意味着再次遍历全代码库,并且效率也很低.另外一个方法是,利用 Elasticsearch 的索引库进行去重,因为在 Elasticsearch 进行信息检索非常高效.判断文件内容的重复,可以利用一个文件内容的 md5 或者直接使用一个文件的 Git blob(https://git-scm.com/book/en/v2/Git-Internals-Git-Objects)信息,md5 或者 Git blob 可以唯一标识一个文件,如果在索引库里一个 md5 或者 Git blob 值出现了多个文件,那么就需要对这些文件进行去重.

利用上述方案,在代码数据爬虫解决了数据的时效性和准确性后,就可以进行真正的代码索引了.因为在企业实际应用中,用户不仅可以对代码进行搜索,也希望对代码库进行搜索,所以也需要索引代码库本身的属性信息.但是代码库本身的属性信息和代码文件是完全不同的索引,需要对他们进行分别索引.

对于代码库的搜索而言,图 2 中代码库的属性信息是非常重要的,比如代码库的类型,它是一个公开的还是一个私有的代码库,这个属性在企业里控制代码库是否开放给其他人进行访问查看.再比如代码库编程语言,如果一个进行搜索的开发者,他本身是长期开发 PHP 的,那么这个开发者在搜索字符串的时候,比如 sort,他有很大的概率是在寻找 PHP 下的 sort 函数,而不是 C++或其他编程语言的,因此,索引了代码库的下编程语言有哪些这样的数据,代码搜索系统就可以进行更为精确的查找.

同样,对于代码搜索而言,图 2 中的代码文件的属性也是对于搜索来说非常重要的,比如编程语言的识别,同上面讲的代码库的编程语言识别作用一样.再比如最近提交作者与时间,则对丰富代码搜索系统的功能很重要,使得用户可以检索一段时间内的代码,或者针对开发者进行搜索.其中,对代码搜索而言,非常重要的一个属性就是语法.代码不同于自然语言,更加结构化.语法的信息能够帮助系统更容易给出用户想要的结果,比如用户在搜索代码时,经常会使用函数名、类名、变量名等作为 query,因此需要对所有语言进行语法解析,获取到代码的语言元素,从而提高搜索质量和相关性.虽然语法解析的工具已经很成熟,但是实际上在企业里解析海量代码的时候,依然困难重重,比如,C++是百度的主流编程语言之一,虽然使用了 CDT 这样非常成熟的工具,但它的解析还是很困难,因为除了语法本身比较复杂(https://en.wikipedia.org/wiki/Most_vexing_parse),每个编译单元(单

个 cpp 文件)还需要准确的给定头文件输入和预处理宏定义才可能进行正确的语法处理,本系统通过解析百度内部编译工具 Bcloud(开源版本 broc(https://github.com/baidu/broc))的中间产出,收集了所有头文件和宏定义,再通过项目依赖关系进行二次推测来降低 CDT 解析符号的错误率完成索引.

Fig.2 Index data of code

图 2 代码索引数据

2.2 query变换

代码搜索是基于文本搜索匹配的技术,而用户给出的 query 和真正的代码片段可能不尽相同[15],为了提高检索效率和相关性,适当地对用户的输入词进行纠错或调整,是每个搜索引擎所必须的.

近年来,不少用于代码搜索的query变换方法被相继提出,比如:Wang等人[16]将用户的意见纳入代码搜索引擎返回的反馈代码片段,以优化结果列表;Roldan-Vega 等人[17]通过计算同一个词与查询中的词的频率来建议替代查询词;Lu 等人[18]通过利用查询和 WordNet[19]中每个单词的词性(POS)来扩展查询,并提出了一种表示为PWordNet 的查询扩展方法;Lemos 等人[20]基于 WordNet 和与代码相关的同义词库自动扩展测试用例.

本系统通过分析大量用户行为,发现大部分错误均为诸如下划线(_)和短横线(-)混淆等误写 1~2 个字符的情况,借鉴文献[17],本系统保存最常引用、最常搜索点击的词为候选集合,并基于编辑距离的贝叶斯模型获得最可能的输入调整.算法详细如下.

对于给定的 query 关键词 w,期望从候选集合中找到用户实际最可能搜索的词 c,即求解条件概率最大的 c:

$$\mathrm{argmax}_{c\in candidates}P(c|w).$$

根据贝叶斯公式(https://en.wikipedia.org/wiki/Bayes'_theorem),上式等价于

$$\mathrm{argmax}_{c\in candidates}P(c)P(w|c)/P(w).$$

不妨假设 $P(w)$为常数,即用户搜索任意关键词概率相等,得出:

$$\mathrm{argmax}_{c\in candidates}P(c)P(w|c).$$

问题转变为求出 c 的概率和 w 与 c 的某种概率关系.

- 首先构造候选词 c 集合,通过分析已索引中的数据和用户行为统计,将最常引用、最常搜索点击的词存为候选集,即$\{word:\langle ref_count,click_count\rangle\}$,那么 $P(c)=\langle ref\_count,click\_count\rangle$,因 $total\_count$ 对于所有 c 相等,遵循避免除法原则,$P(c)=\langle ref\_count,click\_count\rangle$.
- 其次,w 与 c 的概率关系,如上所述,用户多数情况下为手写错误或混淆某个字符,因此,系统通过编辑距离来模拟,即,编辑距离越近,则概率越大.此外,除了用户最多误写 1 个~2 个字符,加之对系统实时性能的约束,系统最终采用了编辑距离小于 2 的概率计算.

2.3 搜索策略

2.3.1 Elasticsearch 原生排序的问题

Elasticsearch 的原始结果是根据文本匹配程度排序的(https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html),每个结果会计算 query 出现的次数(TF)和 query 在倒排索引中出现的次数(IDF),TF/IDF 作为此结果的得分,并用来排序.但对于代码搜索,Elasticsearch 的打分能够提供一定的参考,但文本匹配程度并不是最理想的排序方式,因为它忽略了代码的结构、语义等[8].比如,当搜索函数名的时候,用户更希望获得它的函数声明、定义和针对该函数的单元测试等.

2.3.2 ranker 模块整体方案

如图 3 所示,从 Elasticsearch 获取的结果,首先根据库归类为多个群组;接下来,根据库本身的属性调整分数;

最后,在每个库群组的结果中,根据语法策略判断结果的语法属性,最后对所有结果重新排序分页.

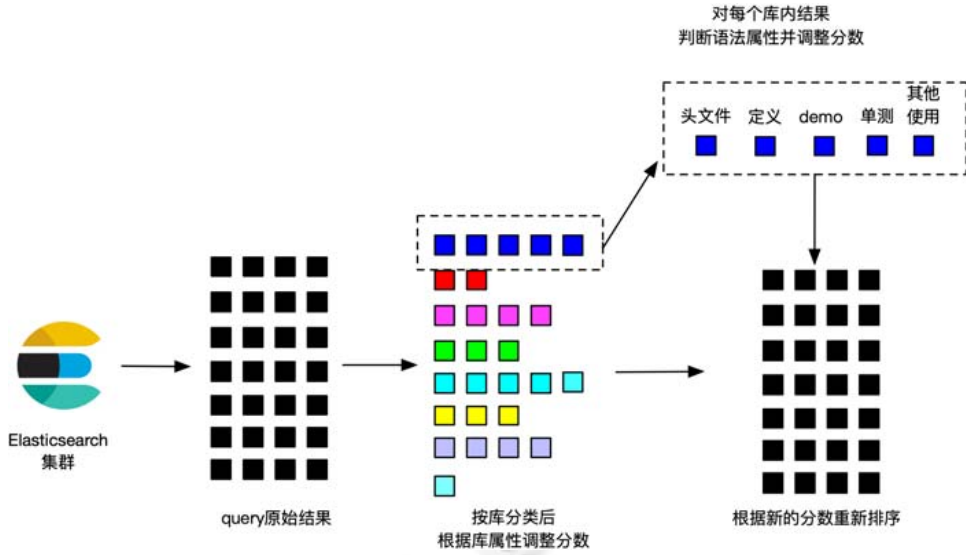

Fig.3 Rough sketch of search strategy

图 3 搜索策略示意图

2.3.3 根据代码库权重调整

结果会按照下面公式根据代码库权重调整:

$$Score=Score_{ES}\times b_{certi}\times b_{depend}\times b_{click}\times b_{vote},$$

其中,

- $Score_{ES}$ 为 Elasticsearch 的原始得分,代表结果的文本匹配程度;
- b_{certi} 为认证的公共基础模块(包括开源三方库)的权重调整系数,该认证列表由公司的各编程语言技术委员会提供;
- b_{depend} 为离线计算的库之间依赖关系的权重调整系数;
- b_{click} 为用户在搜索中点击数据生成的权重调整系数;
- b_{vote} 为用户对搜索结果投票数据生成的权重调整系数.

其中,b_{click} 和 b_{vote} 会在线定时根据新的数据更新权重.

2.3.4 语法策略

经过代码库权重的调整,每组结果会根据离线生成的语法数据库判断属性,如果能够判断为函数或类的声明、定义、单元测试等,则标记为对应的语法元素.以 C++为例,同一个函数的声明所在头文件会优先于对应的源代码文件.

2.3.5 摘要生成

为了提高用户体验,绝大多数搜索引擎都会为用户提供摘要预览,代码搜索也不例外,通过展现与搜索词最相关的代码片段(摘要),以减少用户寻找相关结果的时间,提高点击的精准率.

3 实际应用评估

3.1 实施场景

本系统索引了百度所有的 Git 代码库,总数据量为 TB 级,平均每天新增代码 153 万行,删除代码 63 万行.

系统功能强大,同时支持代码搜索、代码库搜索和文件搜索.搜索结果可以根据编程语言归类,支持全部主流编程语言.提供包括库名、编程语言等的筛选.

搜索响应时间在1s以内.搜索结果会对代码更改保持接近实时的更新,代码从提交到可被搜索到,间隔时间为分钟级.

本系统的用户包括了大部分的百度开发人员,每周的使用用户数千人,每周总搜索次数达数万次,工作日提交代码的开发者平均进行7次搜索.

3.2 搜索结果精度

3.2.1 评估指标

信息检索的常用评估指标为精确率和召回率(https://en.wikipedia.org/wiki/Precision_and_recall).

- 这两个评估指标在某种程度上是互相矛盾的.对于搜索引擎,由于召回率相对容易满足,一般更常用精确率作为主要评估指标.
- 评估单个搜索关键词的精确率,本系统采用*Precision@N*(以下简称*P@N*).即,对每个关键词预设*N*个最佳结果,和实际搜索结果的前*N*项进行比较,匹配数量/*N*作为此关键词的精确率.根据定义,*P@N*结果在0~1之间,实际选取的*N*值为5.
- 评估代码搜索的整体精确率,本系统采用Mean Average Precision(https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)#Mean_average_precision)(以下简称MAP),即,对所有预设的关键词得到的*P@N*结果,再取平均值.*MAP*的得分在0~1之间.

3.2.2 关键词选取和设置预设最佳结果

关键词从用户实际搜索的热门关键词中选取,经过人工选择.

预设最佳结果同样要避免过多的人为因素,采用用户数据分析加人工优化选择方式.用户数据包括用户实际搜索的点击数据、用户都搜索结果的投票统计等.人工选择部分,针对不同的用户搜索模式设置不同的评估原则,见表1.

Table 1 Measurement principle of different types of query words

表1 不同关键词类别的评估原则

关键词类别	搜索结果顺序
类名	类的声明>类的定义>相关的子类>demo>单元测试
函数名	函数声明>函数定义>demo>单元测试>库内部的使用
库名/子模块名	核心类/函数的声明>对应的定义>Readm文件>demo>单元测试
常量名/变量名	定义所在头文件>对应源代码>库内的使用

以上评估原则综合参考了多种方法[2,21,22]并进行了用户调研,调研覆盖了百度内部多个部门的各种主流编程语言开发人员,超过80%的受访者认可上述原则.系统上线后的统计信息也表明,表1中的关键词类别覆盖了超过90%的搜索.

3.2.3 一个例子

正如前文概述中所述,工程师常用的搜索大多是一个API或者类名,从线上环境取一个用户实际query baidu::rpc::Controller作为本次评估用例,这是baidu-rpc库(一个C++的网络服务基础库,即百度对外开源的brpc(https://github.com/brpc/brpc)的内部版本)的一个类名,根据前述原则和代码实际结构,设置5个期待结果及说明见表2.

Table 2 Expected results of baidu::rpc::Controller

表2 baidu::rpc::Controller的期待结果

结果文件名	说明
src/baidu/rpc/controller.h	声明所在头文件
src/baidu/rpc/controller.cpp	类的定义所在的源代码文件
test/test_controller.cpp	专门测试此类的测试文件
example/http_c++/http_server.cpp	使用此类的一个demo
test/test_builtin_service.cpp	使用此类的一个测试文件

实际搜索结果共计 2 700 余个,其中前 5 个均为本库内结果,可见表 3.

Table 3 Actual results of baidu::rpc::Controller

表 3 baidu::rpc::Controller 的实际结果

结果文件名
src/baidu/rpc/controller.h
example/http_c++/http_server.cpp
test/test_streaming_rpc.cpp
tools/rpc_view/rpc_view.cpp
test/test_controller.cpp

和前述期待结果对照,共计 3 个匹配,此搜索词 *P@N* 得分为 0.6.

作为对比,在 GitHub 上进行对应的搜索,注意:开源版本的 brpc 和公司内 baidu-rpc 在 namespace 和代码结构有所区别,但基本可以对应.比如,搜索关键词相应的变更为 brpc::Controller,它的声明头文件在 brpc 库的 src/brpc/controller.h

对全 GitHub 代码的搜索中(https://github.com/search?q=brpc%3A%3AController&type=Code),搜索结果的前 2 页没有任何来自 brpc 库的结果.第 1 个来自 brpc 的结果出现在第 27 条.

对 brpc 库内搜索(https://github.com/brpc/brpc/search?q=brpc%3A%3AController),其前 5 条结果见表 4.

Table 4 Actual results of brpc::Controller on GitHub code search

表 4 brpc::Controller 在 GitHub 代码搜索的实际结果

结果文件名
src/brpc/builtin/pprof_service.h
src/brpc/builtin/hotspots_service.h
test/brpc_controller_unittest.cpp
src/brpc/builtin/rpcz_service.h
src/brpc/builtin/vlog_service.h

和前述期待结果对照,只有第 3 条匹配,GitHub 在此搜索词的 *P@N* 得分为 0.2.

3.2.4 搜索精度得分

如表 5 所示,作为对比,同时列出 Elasticsearch 原始结果的 MAP 得分.

Table 5 MAP score

表 5 MAP 得分

方案	MAP 得分
本系统方案	0.5689
Elasticsearch 原始结果	0.2223

4 不足之处

当前实现的系统还有几个不足之处.

- 首先是还没有更好地利用代码相关的各种属性使得搜索结果更好.代码相对于自然语言有着明确的语法约束以及语义,现在仅能对语法元素在搜索结果的排序上进行处理,但是尚未通过分析或者挖掘代码语义进行优化搜索结果的排序.
- 其次,尚未充分利用代码之间的调用关系.在传统的信息检索中,PageRank 算法使得相关度较高的网页获得更靠前的排序[23].与之类似,代码中的函数与函数之间、类与类之间、文件与文件之间、代码库与代码库之间,皆存在着清晰明确的调用关系.目前还没有充分挖掘调用关系以及和代码相关联的数据特征,在未来可以利用知识图谱[24]、PageRank 等技术优化代码搜索的排序结果.
- 再次,尚未利用开发者的自身信息.代码不是孤立存在的,编写代码的开发者也具有独特的特征,比如在企业里,每个开发者的代码权限是不一样的,每个开发者的技能水平也是不一样的.

- 最后,尚未挖掘用户行为信息.代码搜索系统推出以后,大量用户开始使用,已经累积大量的用户行为数据,比如在什么时间进行了哪些搜索、在哪页第几个结果上进行了点击、用户浏览搜索结果的停留时间、用户查看结果并且更新了搜索词进行了二次搜索、用户对结果条目进行了满意度评价等等.这些行为数据可以用来进行点击率预估[25],使得系统给出的搜索结果更加符合用户预期.

5 相关研究工作

近年来,代码搜索领域的研究者提出了许多不同的搜索方法,绝大多数通过自然语言或者自定义的规格对用户输入约束,返回抽象调用序列、函数片段等不同粒度的代码.简要介绍如下.

- Yahav[26]以部分代码作为输入,基于对象类型查找语义相关的代码片段.
- DEEPAPI[5]使用深度学习循环神经网络的方法,对于给定的查询语句,生成 API 用法序列.
- Sourcerer[27]在代码实体完全限定名上使用 TF-IDF 技术,结合 right-most handside boosting 技术和图排序算法来查找功能的实现代码、现有代码片段的使用及包含特定属性及模式的代码.
- Xsnippet[10]给对象实例化任务提供示例代码,灵活设定查询范围,并根据代码片段长度、出现频率等进行启发式排序.
- Portfolio[28]利用函数调用图建立模型,模拟编程者浏览及关联其他函数行为,检索到能够解决查询语句中描述的多项任务的具体函数.
- Mica[29]为用户提供更多 API 使用的相关信息,返回相关的 API 方法类和域,并根据频率相关性排序.
- PARSEWeb[30]与 Google Code Search Engine 交互,通过对代码分析,将代码片段抽象表示成方法调用序列的形式,然后对方法调用序列进行聚类和排序.
- Exemplar[31]结合信息检索和程序分析技术,使用方法关键字匹配查询应用的描述、源代码以及帮助文档.
- RACS[32]聚焦 JavaScript 框架代码特征,支持自由形式的查询语句,使用方法调用关系图来描述 API 调用之间的复杂关系;同时,从自然语言查询语句中也捕捉相对应的关系,提高了 JavaScript 框架代码示例搜索的准确率.
- SP[8]基于用户提供的规格描述(包括关键字、类或方法签名、测试用例、约束、安全限制),检索出满足要求的函数及类.
- Prospector[33]工具为程序员编写 API 客户端代码提供帮助.用户描述输入/输出类型,自动使用 API 方法签名及从大量示例客户端代码中挖掘到的 jungloids 来合成便于复用的查询结果.
- DERECS[34]方法基于方法调用及代码片段的结构特征对代码进行描述增强的方法,减小了被搜索的代码与自然语言查询语句之间的差异,提高了搜索的准确性.

本文针对企业级的海量代码,提出了和自然语言搜索引擎一样,利用简单关键词进行检索匹配代码的方法.

6 总结与未来工作

本文提出了企业级海量代码下的代码检索与管理方案以及系统实现,整个系统分为离线建库端和在线搜索端:离线建库使得代码数据通过批量和实时的方式进行获取、语法解析、索引建库;在线搜索端接收用户的输入请求,对用户的请求进行变换,然后在 Elasticsearch 上进行数据检索,拿到 Elasticsearch 基于词频等特征给出搜索结果,利用多个排序策略再次进行排序,最后呈现搜索结果.系统在百度上线以后,广受百度工程师用户的好评.

然而正如在第 4 节所描述的,未来还有大量的优化工作需要进行:首先,需要进一步挖掘代码本身语法与语义信息,更好地匹配用户的搜索意图;其次,需要挖掘代码、文件、代码库、开发者等之间的关系,利用知识图谱、PageRank 等算法优化搜索结果;最后,用户的行为数据是真实地指明系统现状以及用户想要的结果是什么,所以接下来会挖掘用户的行为数据,进行点击率预估,进一步优化系统给出的结果.

同时,根据百度内部的调查,用户在使用后,反馈的需求和改进包括:根据提交信息搜索、支持搜索注释、能够识别和过滤敏感信息(如误递交的帐号密码等)等.接下来也会在这些功能上进行完善.经过这些措施完善后的代码搜索工具,借助于百度研发效率云对外开放.

References:

[1] Singer J, Lethbridge T, Vinson N, Anquetil N. An examination of software engineering work practices. In: Proc. of the '97 Conf. of the Centre for Advanced Studies on Collaborative Research (CASCON'97). IBM Press, 1997. 174−188.

[2] Sadowski C, Stolee KT, Elbaum S. How developers search for code: A case study. In: Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering. Bergamo, 2015. 191−201.

[3] Xia X, Bao LF, Lo D, Kochhar PS, Hassan AE, Xing ZC. What do developers search for on the Web? Empirical Software Engineering, 2017,22(6):3149−3185.

[4] Potvin R, Levenberg J. Why Google stores billions of lines of code in a single repository. Communications of the ACM, 2016,59(7): 78−87. [doi: 10.1145/2854146]

[5] Gu X, Zhang H, Kim S. Deep code search. In: Proc. of the 40th Int'l Conf. New York: ACM Press, 2018. 933−944. http://doi.org/10.1145/3180155.3180167

[6] Kashyap V, Brown DB, Liblit B, Melski D, Reps TW. Source forager—A search engine for similar source code. arXiv preprint arXiv:1706.02769, 2017.

[7] Linstead E, Bajracharya SK, Ngo TC, Rigor P, Lopes CV, Baldi P. Sourcerer: Mining and searching Internet-scale software repositories. Data Mining and Knowledge Discovery, 2009,18(2):300−336.

[8] Reiss SP. Semantics-based code search. In: Proc. of the 31st Int'l Conf. on Software Engineering. 2009. 243−253.

[9] Stollee KT, Elbaum SG, Dobos D. Solving the search for source code. ACM Trans. on Software Engineering and Methodology, 2014,23(3):1−45.

[10] Sahavechaphan N, Claypool KT. XSnippet: Mining for sample code. In: Proc. of the 21st Annual ACM SIGPLAN Conf. on Object-oriented Programming Systems, Languages, and Applications. 2006. 413−430.

[11] Begel A. Codifier: A programmer-centric search user interface. In: Proc. of the Workshop on Human-computer Interaction and Information Retrieval. 2007. 23−24.

[12] Heydon A, Najork M. Mercator: A scalable, extensible Web crawler. In: Proc. of the World Wide Web 2.4. 1999. 219−229.

[13] Ahmadi-Abkenari F, Selamat A. An architecture for a focused trend parallel Web crawler with the application of click stream analysis. Information Sciences, 2012,184(1):266−281.

[14] Hsieh JM, Gribble SD, Levy HM. The architecture and implementation of an extensible Web crawler. In: Proc. of the InNSDI 2010. 2010. 329−344.

[15] Furnas GW, Landauer TK, Gomez LM, Dumais ST. The vocabulary problem in human-system communication. Communications of the ACM, 1987,30(11):964−971.

[16] Wang S, Lo D, Jiang L. Active code search: Incorporating user feedback to improve code search relevance. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras, 2014. 677−682.

[17] Roldan-Vega M, Mallet G, Hill E, Fails JA. CONQUER: A tool for NL-based query refinement and contextualizing code search results. In: Proc. of the 2013 29th IEEE Int'l Conf. on Software Maintenance (ICSM). 2013. 512−515.

[18] Lu M, Sun X, Wang S, Lo D, Duan Y. Query expansion via WordNet for effective code search. In: Proc. of the 2015 IEEE 22nd Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). 2015. 545−549.

[19] Miller GA. WordNet: A lexical database for English. Communications of the ACM, 1995,38(11):39−41.

[20] Lemos OA, de Paula AC, Zanichelli FC, Lopes CV. Thesaurus-based automatic query expansion for interface-driven code search. In: Proc. of the 11th Working Conf. on Mining Software Repositories. Hyderabad, 2014. 212−221.

[21] Ko AJ, Myers BA, Coblenz MJ, Aung HH. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Trans. on Software Engineering, 2006,32(12):971−987.

[22] Li H, Xing Z, Peng X, Zhao W. What help do developers seek, when and how? In: Proc. of the 20th Working Conf. on Reverse Engineering (WCRE). IEEE, 2013. 142−151

[23] Page L, Brin S, Motwani R, Winograd T. The pagerank citation ranking: Bringing order to the Web. Technical Report, 1999-66, Stanford InfoLab, 1999.

[24] Lin Y, Liu Z, Sun M, Liu Y, Zhu X. Learning entity and relation embeddings for knowledge graph completion. In: Proc. of the 29th AAAI Conf. on Artificial Intelligence. 2015.

[25] Graepel T, Candela JQ, Borchert T, Her-brich R. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft's Bing search engine. In: Proc. of the 27th Int'l Conf. on Machine Learning (ICML 2010). 2010.

[26] Mishne A, Shoham S, Yahav E. Typestate-based semantic code search over partial programs. ACM SIGPLAN Notices, 2012, 47(10):997−1016. [doi: 10.1145/2398857.2384689]

[27] Bajracharya S, Ngo T, Linstead E, Dou YM, Rigor P, Baldi P, Lopes C. Sourcerer: A search engine for open source code supporting structure-based search. In: Proc. of the Companion to the 21st ACM SIGPLAN Symp. on Object-oriented Programming Systems, Languages, and Applications. ACM Press, 2006. 681−682. [doi: 10.1145/1176617.1176671]

[28] McMillan C, Grechanik M, Poshyvanyk D, Xie Q, Fu C. Portfolio: Finding relevant functions and their usage. In: Proc. of the 33rd Int'l Conf. on Software Engineering. ACM Press, 2011. 111−120. [doi: 10.1145/1985793.1985809]

[29] Stylos J, Myers BA. Mica: A Web-search tool for finding API components and examples. In: Proc. of the 23rd Int'l Conf. on Program. IEEE, 2015. [doi: 10.1109/VLHCC.2006.32]

[30] Treude C, Robillard MP. Augmenting API documentation with insights from stack overflow. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016. 392−403. [doi: 10.1145/2884781.2884800]

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

[32] Li X, Wang ZR, Wang QX, Yan SM, Xie T, Mei H. Relationship-aware code search for JavaScript frameworks. In: Proc. of the 24th ACM SIGSOFT Symp. on the Foundations of Software Engineering. ACM Press, 2016. 690−701. [doi: 10.1145/2950290.2950341]

[33] Mandelin D, Xu L, Bodík R, Kimelman D. Jungloid mining: Helping to navigate the API jungle. ACM SIGPLAN Notices, 2005, 40(6):48−61. [doi: 10.1145/1065010.1065018]

[34] Li X, Wang QX, Jin Z. Description reinforcement based code search. Ruan Jian Xue Bao/Journal of Software, 2017,28(6): 1405−1417 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5226.htm [doi: 10.13328/j.cnki.jos.005226]

附中文参考文献:

[34] 黎宣,王千祥,金芝.基于增强描述的代码搜索方法.软件学报,2017,28(6):1405−1417. http://www.jos.org.cn/1000-9825/5226.htm [doi: 10.13328/j.cnki.jos.005226]

刘志伟(1984－),男,辽宁朝阳人,学士,主要研究领域为智能软件开发.

李涛(1982－),男,硕士,CCF 专业会员,主要研究领域为智能软件开发.

邢永旭(1985－),男,硕士,主要研究领域为智能软件开发.

张晓东(1990－),男,博士,CCF 学生会员,主要研究领域为代码搜索,软件测试与验证.

于澔(1985－),男,硕士,主要研究领域为智能软件开发.