elastic 中文社区

Elastic 中文社区

深圳Meetup

2019/11/16

Saturday 13:00

合作伙伴

vivo 神鲸 Coca-Cola 可口可乐 Broadview www.broadview.com.cn

合作社区

segmentfault 创造属于开发者的时代 OSCHINA 掘金 珠三角技术沙龙 http://techparty.org

Gitee的Elasticsearch检索优化

关于Gitee和我

Gitee(码云)：

世界第二，国内第一的代码托管平台，350万开发者，600万仓库。
https://gitee.com/

陈鑫(狮子的魂)：

技术极客，多个大型开源软件的作者和维护者，有丰富的机器学习，自然语言处理，信息检索，语音交互，Web系统架构的实战经验。

Gitee首席架构师，现在负责Gitee的搜索和AI。

目录

一、分词模式的分析和优化

一.分词模式的分析和优化—Gitee仓库的文本特点

1. 仓库的名字和描述包含很多特殊符号，例如“-_#.+/”等。
2. 仓库的名字或者描述使用多态性，例如：“jfinal-weixin，wechat-php-sdk，WxJava等，微信会被随机写成：weixin，wechat，wx”等
3. 很多特殊的中文、英文、标点的组合词条，例如：“c++，c#，c语言，d语言，卡拉ok，w信”等。
4. 不规则、不标准的英文组合，例如：“j2cache，layuimini，anycmd，hutool，JeeSpringCloud，mpsdk4j”等。
5. 仓库数据维度很多，包含：名字，名字命名空间，路径，路径命名空间，作者/企业/组织的名字，仓库描述，仓库标签和众多其他的属性等。不同属性集合需要的文本处理方式不一样。

一.分词模式的分析和优化—分词器的要求

1. 中文分词支持最多分词来提高命中率。例如：“中文分词”切分结果如下：“中/中文/中文分词/文/分/分词/词”。
2. 英文分词支持类中文的切分或者n-gram的切分。例如：”elasticsearch”被切分成：”elastic/search/elasticsearch”，“openarkcompiler”被切分成”open/ark/compiler”。
3. 支持自动拼音的追加，用于中文和拼音的自动对应检索支持。例如：”微信”被切分成：”微信/weixin“, “分词”被切分成”分词/fenci”。
4. 支持拼音的切分用于拼音的反向检索：例如”fenci”切分成”fenci”或者”fen/ci”
5. 支持同义词的追加，用于同义词的相互检索。例如：”中文“被切分成：”中文/国语/普通话/汉语”，”微信“被切分成：”weixin/wechat/v信/w信“。
6. 支持中文、英文、标点等组合词条的识别。例如：可以识别”c++, c#, c语言,卡拉ok,v信“等。
7. 支持多种切分模式，例如：中英文的最多切分模式，大颗粒准确率切分模式，n-gram切分，分隔符切分等。

一.分词模式的分析和优化—Gitee分词器的使用

Gitee的分词系统基于Jcseg改造，详情请访问Gitee仓库（https://gitee.com/lionsoul/jcseg）。具体的使用配置如下：

功能字段	索引分词	分词配置	检索分词	分词配置	设计优点	设计缺点
检索字段：仓库名字，仓库描述，作者/企业/组织，仓库路径等	最多分词模式	√ 同义词 √ 拼音	大颗粒精准分词模式	×同义词 ×拼音	最大化命中率 检索结果相关度 同义词反向检索 拼音反向检索	lucene的索引会线性增大，给计算和存储增加一定的压力
过滤字段：仓库的公私属性、作者/组织/企业、fork/编程语言/分类等	分隔符模式	分隔符=,	分隔符模式	分隔符=,	可以存储不限量的过滤属性，支持n纬度同时过滤，分词速度极快	众多过滤属性聚合在一起，更新相对麻烦
检索提示：作者/组织/企业+仓库名称	n-gram	N=1	n-gram	N=1	输入建议最大范围化，建议精度随时可调	建议的准确度问题会有偏差(通过查询配置和检索模型来补足)

一.分词模式的分析和优化—分词器使用举例

1，例如对于demo文档：“中文分词”

2，索引分词的结果如下：

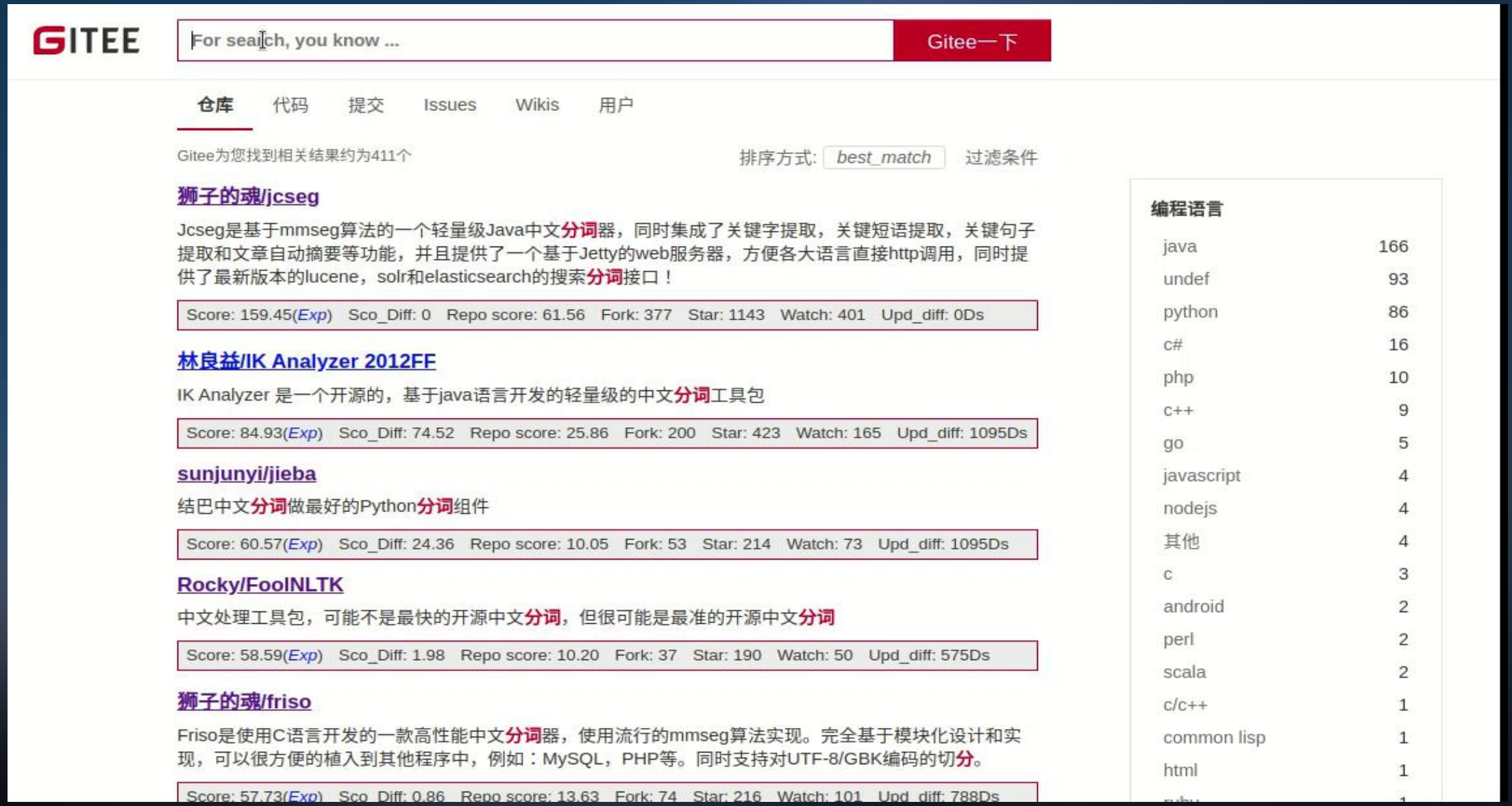
中 zhong 中文 zhongwen 汉语 hanyu 国语 guoyu 普通话 putonghua 中文分词 zhongwenfenci 文 wen 分 fen 分词 fenci 词 ci

3，对于检索以下类似的输入都可以命中，同时同义词/拼音得到的检索和高亮效果一致：

中文分词
汉语分词
国语分词
普通话分词
分词
中文 分词
中 文 分 词
...

zhongwen分词
zhong wen 分 ci
hanyu分词
guoyu分词
guo yu 分词
han yu fen ci
han yu fen词
...

一.分词模式的分析和优化—分词器效果



二、检索模型的分析和优化

二.检索模型的分析和优化—BM25模型的认识

Lucene最新的几个版本默认使用的全文检索模型是LegacyBM25Similarity的实现，也是Lucene提供的全部的全文检索模型中相对最优秀的一个，基于TF-IDF，可控性也比较强，其模型计算公式如下：

$$\mathbf{score(Q,d)} = \sum_{i}^{n} \boldsymbol{IDF(qi)} \times \frac{\boldsymbol{TF(qi)} + (\boldsymbol{k1}+\mathbf{1})}{\boldsymbol{TF(qi)} + \boldsymbol{k1} \times (\mathbf{1} - \boldsymbol{b} + \boldsymbol{b} \times \frac{\boldsymbol{dl}}{\boldsymbol{avgdl}})}$$

$$\mathbf{TF(qi)} = \mathbf{fi}$$

$$\mathbf{IDF(qi)} = \frac{\boldsymbol{N} - \boldsymbol{n(qi)} + \mathbf{0.5}}{\boldsymbol{n(qi)} + \mathbf{0.5}}$$

二.检索模型的分析和优化—BM25模型的分析

BM25模型的各个符号的含义和变化关系如下：

因子	含义	变化关系
N	文档总数	对于某个文档/字段计算这个值是固定的
n(qi)	包含qi词条的文档总数	n越小IDF值越大
fi	词条qi的词频，在d中出现的次数	fi越大TF值越大
dl	文档d的长度（词条数目）	dl越小IDF值越大，也就是突出短文档
avgdl	文档d集合的平均长度（词条数目）	对于某个词条的这个值是固定
k1	非线性词频归一化控制因子	k1越小，线性趋向关系越强
b	控制文档长度规范化的程度	b越小文档长度影响越小

二.检索模型的分析和优化—Gitee的BM25的调优

依据上面描述的仓库数据特点分析出的检索需求，我们将默认的BM25模型更改如下：

检索需求	需求理由	调优方案
弱化文档长度的影响	我们期望用户提供信息量大的名字或者说为仓库编写详细的介绍描述，而不是突出短文档	将BM25模型中的b值从默认的0.75调整为0.18，取值的过程需要取到全部仓库描述的平均长度。
弱化词频的影响	仓库描述中，经常出现大量重复词条，但并不重要，我们的检索关系用的是逻辑与，更强调整体的信息量	备选方案：常量词频，二次模型，对数模型，经过测试对数模型效果最理想，即：$$TF(qi) = 1 + \log_e fi$$
强化部分词条和文档的相关度	仓库描述中经常出现很多词条，这些词条本身的IDF值偏大，但确和仓库属性本身关系不大，例如：仓库中经常提及的各种技术和软件名词	新版本的Similarity接口已经不支持增加针对文档的词条加权。我们通过设计针对加权字段和查询的加权来实现的类似功能。

二.检索模型的分析和优化—Similarity接口的变更

public abstract long computeNorm(FieldInvertState state)

变更：match对长字段用的是估算值，所以对这个方法的实现官方建议state.getLength()越大返回越小的值，越小返回则设置越大的值。

public abstract float score(float freq, long norm)

变更：

1. 词频增大，Score不能减少。

2. 对于相同的term-document频率，norms增大，Score不能增大。

$$\mathbf{score}(\mathbf{Q},\mathbf{d}) = \sum_{i}^{n} \frac{\boldsymbol{N} - \boldsymbol{n}(\boldsymbol{qi}) + \mathbf{0.5}}{\boldsymbol{n}(\boldsymbol{qi}) + \mathbf{0.5}} \times \frac{\boldsymbol{fi} + (\boldsymbol{k1} + \mathbf{1})}{\boldsymbol{fi} + \boldsymbol{k1} \times (\mathbf{1} - \boldsymbol{b} + \boldsymbol{b} \times \frac{\boldsymbol{abs}(\boldsymbol{dl} - \boldsymbol{avgdl})}{\boldsymbol{avgdl}})}$$

吐槽： 最新的Similarity接口的设计被固化了，例如要实现上述模型，越趋向平均长度的文档，打分越高，就非常不好实现，官方给出的解释大意是，便于对分数贡献不大的词条进行忽略或者近似计算从而加速相似度计算的过程，例如BM25对文档长度的编解码和计算缓存。

二.检索模型的分析和优化—检索提示的EFR模型

Gitee检索时基于仓库名字检索提示，需要一个轻量级、快速并且更符合这需求的检索模型，EFR(Equality From Randomness)模型定义如下：

$$\mathbf{score}(\mathbf{Q},\mathbf{d}) = \sum_{i}^{n} 1 + \frac{1}{dl}$$

dl：为文档d的长度，token数量。

作用：得到的score为一个浮点数，在命中相同词条数的情况下，整数部分为命中的词条数，小数部分为匹配的百分比，命中相同词条的情况，dl越短打分越高，得到的效果如下：

二.检索模型的分析和优化—检索提示的效果

三、排序模型的分析和优化

三.排序模型的分析和优化—检索模型的不足和优化措施

问题：BM25Similarity检索模型计算的Score对Gitee的代码仓库的检索排序出来的结果堪称**不忍直视**。

原因：检索模型对仓库数据的分析仅限词条的TF和IDF两个维度，仅仅是站在信息论的角度对文本数据进行的一个维度的区分计算，她非常优秀，但满足不了我们的需求。

解决方案

我们需要多个量化模型对数据进行更多维度的量化然后参与到BM25模型得出的_score的加权中，以提供更好的排序结果，Google的排序模型使用超过了200个维度的量化因子对网页进行了求权。

三.排序模型的分析和优化—BM25检索效果

对于检索“中文分词”，使用默认的BM25的排序结果如下：

三.排序模型的分析和优化—Gitee的排序模型

在确保检索的相关度的情况下，Gitee对仓库通过如下的维度进行了量化计算并且参与到排序的加权中，本质上是Gitee对一个仓库的评测定义量化的过程。

排序因子（十几个）：star数、watch数、fork数、是否推荐、是否为GVP、是否为fork、仓库代码提交者人数、代码仓库文件数、代码仓库字节数、仓库提交次数、分支数、PR数、Issue数，Gitee的用户评论数，仓库最后更新时间等。

排序模型如下：

$$\mathbf{score}(\mathbf{Q},\mathbf{d}) = \_score_i + \sum_j^n \_score_i \times weight_j \times norm_j$$

$\_score_i$: BM25检索模型对查询Q和第i个文档计算返回的浮点相似度分数。

$weight_j$：第i个因子的权重调节系数，取值0~1

$norm_j$：第i个量化模型，取值因因子不同而异，具体请看下一页

三.排序模型的分析和优化—Gitee的排序因子

为了加速计算，Gitee的仓库排序因子聚合成了如下几个大类，每个大类的量化数值都会在索引的时候提前计算好，其计算公式和排序模型具体如下：

排序因子	因子集合	量化模型	排序模型
公开分数	star数/fork数/watch数/是否推荐/是否为GVP	f(v) = (w1 * norms(stars) + w2 * norms(watches) + w3 * norms(forks)) * (vip->w4) * (gvp->w5) 每个因子归一化为0到100	$y = -0.0001666x^2 + 0.1033253x$ norm=y (取值为0~10) weight=0.54
活跃分数	代码提交者人数/仓库文件数/仓库字节数/仓库提交次数	f(v) = (w1 * norms(contributors) + w2 * norms(files) + w3 * norms(bytes) + w4 * norms(commits) 每个因子都归一化为0到100	$y = -0.0001666x^2 + 0.1033253x + 1$ norm=y（取值为0~10) weight=0.24

三.排序模型的分析和优化—Gitee的排序因子(续)

剩下的排序因子如下：

排序因子	因子集合	量化模型	排序模型
仓库更新	最后更新时间	f(v) = norms(timestamp) 结果为最后更新时间距离现在的天数，取值为0~1095	$y = -0.0008333x^2 - 0.0083343x + 1$ norm=y(取值为0~1) weight=0.20
惩罚分数	描述、README内容等基本信息的缺失	量化为0~100	调整因子的指数函数

三.排序模型的分析和优化—排序因子使用过程

例如对于查询“中文分词”其中某个文档的BM25得分为：30.279736，计算过程如下：

狮子的魂/jcseg Java

Jcseg是基于mmseg算法的一个轻量级Java**中文分词**器，同时集成了关键字提取，关键短语提取，关键句子提取和**文**章自动摘要等功能，并且提供了一个基于Jetty的web服务器，方便各大语言直接http调用，同时提供了最新版本的lucene，solr和elasticsearch的搜索**分词**接口！

Score: 244.65(*Exp*) Sco_Diff: 0 Repo score: 61.66 Fork: 377 Star: 1148 Watch: 403 Upd_diff: 0Ds


```
{
  value: 30.279736,
  description: "weight(description:中文分词 in 433478) [PerFieldSimilarity], result of:",
  details: [
    {
      value: 30.279736,
      description: "score(freq=1.0), product of:",
      details: [
        { 3 items },
        {
          value: 8.386532,
          description: "idf, computed as log(1 + (N - n + 0.5) / (n + 0.5)) from:",
          details: [ 2 items ]
        },
        {
          value: 1.1250528,
          description: "tf=1.0, modifer=logp1, computed as freq * / (freq + k1 * (1 - b + b * dl / avgdl)) from:",
          details: [ 5 items ]
        }
      ]
```

三.排序模型的分析和优化—排序因子使用过程(续)

其repo_score，也就是公开分数为61.66，经过NLP提取的关键字和repo_score加权后得分为：

狮子的魂/jcseg Java

Jcseg是基于mmseg算法的一个轻量级Java**中文分词**器，同时集成了关键字提取，关键短语提取，关键句子提取和**文**章自动摘要等功能，并且提供了一个基于Jetty的web服务器，方便各大语言直接http调用，同时提供了最新版本的lucene，solr和elasticsearch的搜索**分词**接口！

Score: 244.65(*Exp*) Sco_Diff: 0 Repo score: 61.66 Fork: 377 Star: 1148 Watch: 403 Upd_diff: 0Ds

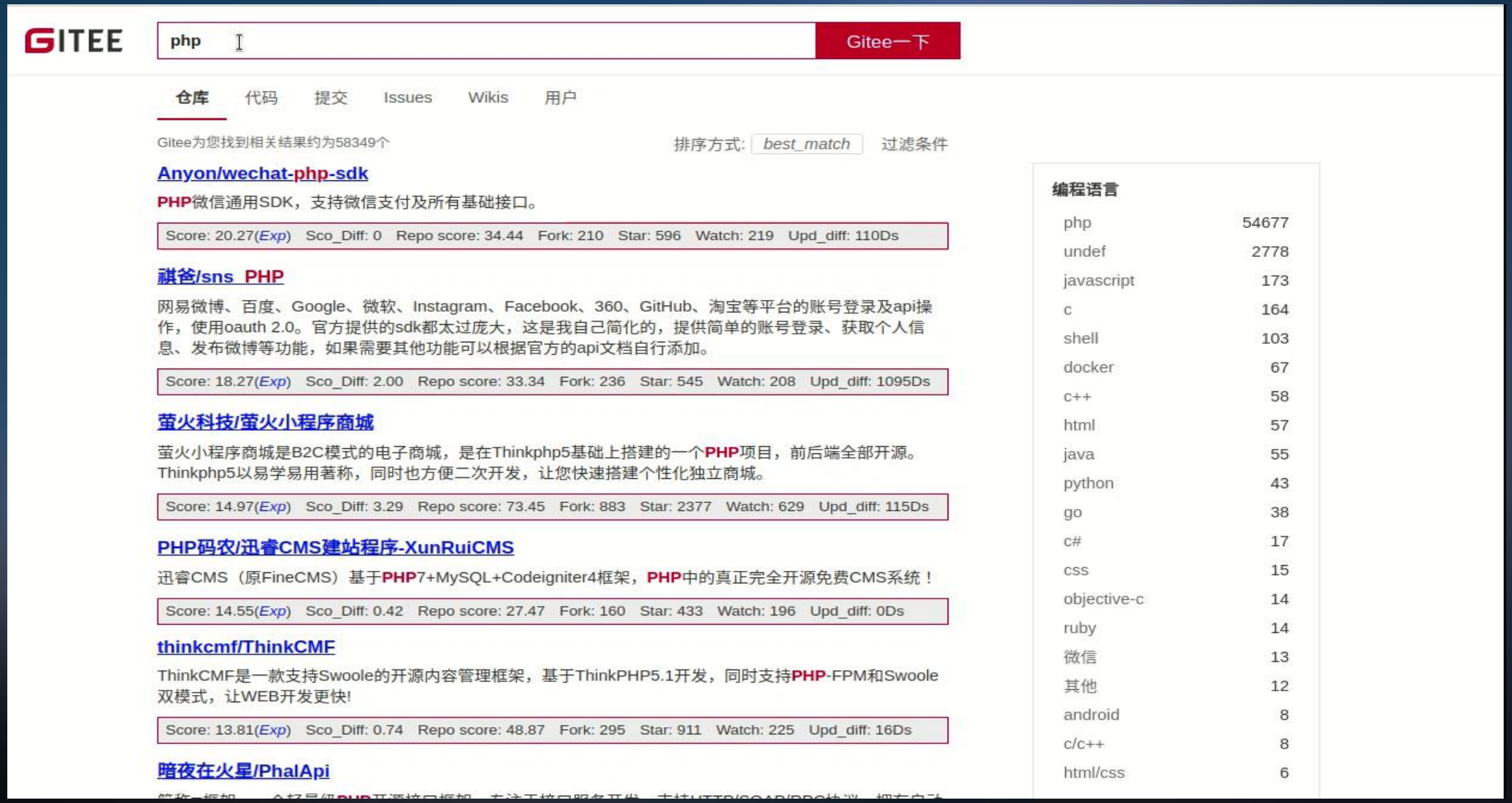
},
_explain: {
value: **244.65196**,
description: "min of:",
details: [
{
value: **244.65196**,
description: "script score function, computed with script:"Script{type=inline, lang='painless', idOrCode='_score + _score * params.repo_weight_w * (-0.0001666 * doc['repo_score'].value * doc['repo_score'].value + 0.1033253 * doc['repo_score'].value) + _score * (doc['updated_td'].value <= 30 ? params.tdif_weight_w * (-0.0008333 * doc['updated_td'].value * doc['updated_td'].value - 0.0083343 * doc['updated_td'].value + 1) : (doc['updated_td'].value <= params.tdif_decent_d ? 0 : params.tdif_decent_w))', options={}, params={tdif_weight_w=0.2, tdif_decent_w=-0.164, tdif_decent_d=300, repo_weight_w=0.48}}"",
details: [
{
value: **61.875885**,
description: "_score: ",
details: [1 item]
}

三.排序模型的分析和优化—Gitee排序模型效果

对于检索“中文分词”，使用默认的BM25+Gitee排序模型的结果如下：

三.排序模型的分析和优化—排序模型效果对比

BM25模型和BM25+Gitee排序模型的对比结果如下：



四、索引的结构和查询语法

四.索引的结构和查询语法—Gitee的仓库搜索Mapping

Gitee的仓库搜索使用的Mapping局部字段结构如下：

```
 "description" : {                    // 仓库描述
  "type" : "text",
  "similarity" : "gitee_bm25",
  "analyzer" : "for_index",
  "search_analyzer" : "for_search"
},
"makeup_for_weight" : {          //加权关键字
  "type" : "text",
  "similarity" : "gitee_bm25",
  "analyzer" : "for_index",
  "search_analyzer" : "for_search"
},
"path_for_suggest" : {             //搜索建议字段
  "type" : "text",
  "store" : true,
  "similarity" : "gitee_efr",
  "analyzer" : "unigram"
}
```

```
"attr_set" : {      # 属性集合
  "type" : "text",
  "analyzer" : "comma_delimiter"
},
"name" : {          # 仓库名字
  "type" : "text",
  "similarity" : "gitee_bm25",
  "analyzer" : "for_index",
  "search_analyzer" : "for_search"
},
"name_for_hl" : {          # 名字高亮字段
  "type" : "text",
  "similarity" : "gitee_bm25",
  "analyzer" : "for_index",
  "search_analyzer" : "for_search"
}
```

四.索引的结构和查询语法—Gitee的仓库搜索Mapping

Gitee的仓库搜索使用的Mapping的similarity定义如下：

```
"similarity" : {
  "gitee_efr" : {
    "discount_overlaps" : "false",
    "type" : "EFR"
  },
  "gitee_bm25" : {
    "discount_overlaps" : "true",
    "const_freq" : "1",
    "b" : "0.18",
    "freq_modifier" : "logp1",
    "type" : "GiteeBM25",
    "k1" : "1.26"
  }
}
```

四.索引的结构和查询语法—Gitee的仓库搜索Mapping

Gitee的仓库搜索使用的Mapping的analysis定义如下：

```
"for_index" : {              # 索引分词
  "jcseg_maxlen" : "5",
  "jcseg_appendpinyin" : "1",
  "jcseg_appendsyn" : "1",
  "jcseg_enwordseg" : "1",
  "jcseg_ensecondseg" : "1",
  "type" : "jcseg_most",
  "jcseg_pptmaxlen" : "0"
},
"for_search" : {             # 检索分词
  "jcseg_maxlen" : "5",
  "jcseg_appendpinyin" : "0",
  "jcseg_appendsyn" : "0",
  "jcseg_enwordseg" : "1",
  "jcseg_ensecondseg" : "1",
  "type" : "jcseg_complex",
  "jcseg_pptmaxlen" : "0"
}
```

```
"unigram" : {                # 一元分词
  "jcseg_gram" : "1",
  "jcseg_appendpinyin" : "0",
  "jcseg_appendsyn" : "0",
  "type" : "jcseg_ngram"
},
"comma_delimiter" : {        # 分隔符分词
  "jcseg_appendpinyin" : "0",
  "jcseg_appendsyn" : "0",
  "jcseg_delimiter" : ",",
  "type" : "jcseg_delimiter"
}
```

Gitee检索优化实战

Thank you

微信：lionsoul2014

关于

Elastic 中文社区 http://elasticsearch.cn

Elastic Meetup 是由 Elastic 中文社区定期举办的线下交流活动，主要围绕 Elastic 的开源产品（Elasticsearch、Logstash、Kibana 和 Beats ）及 Elastic Stack 周边技术，探讨在搜索、数据实时分析、日志分析、安全等领域的实践与应用。

欢迎加入 Elastic 中文社区，**参与分享交流** 或 **赞助社区活动！**

深圳联络人：杨振涛
微信：nodexy
邮箱：nodexy@qq.com

本次活动回顾及现场照片在“vivo互联网技术”公众号发布，欢迎关注浏览。

微信扫码关注