2021-01-28 Richard技术分享

人岗匹配技术调研分享

背景

1. 简历的形式多种多样，很难从简历中定位重要信息。
2. 公司每天会收到大量的简历投递，人力筛选效率低。
3. 大量非结构化的简历难以直接比较。
4. 匹配已解析的CV数据和JD数据，给出匹配得分，从求职者中筛选出优质候选人。

在人岗匹配的任务中存在**HR、职位（JD）、简历（CV）**三种实体，人岗推荐系统中由HR发布职位，根据发布职位来推荐简历，提升HR更高的工作效率，提升岗位和简历的匹配度来减少招聘人才的成本。

技术调研

机器学习方向

人工智能、机器学习、深度学习的关系

概括来说，人工智能、机器学习和深度学习覆盖的技术范畴是逐层递减的。人工智能是最宽泛的概念。机器学习是当前比较有效的一种实现人工智能的方式。深度学习是机器学习算法中最热门的一个分支，近些年取得了显著的进展，并替代了大多数传统机器学习算法。三者的关系即：人工智能 > 机器学习 > 深度学习。

人工智能：人工智能是一个非常大的概念，其最初定义是要让机器的行为看起来就像是人所表现出的智能行为一样。我们经常听到的语音识别、图像识别、自然语言理解等领域都是具体的人工智能方向，而机器学习、神经网络等概念都属于实现人工智能所需要的一些技术。

机器学习：机器学习是人工智能的一门分支，指通过学习过往经验来提升机器的智能性的一类方法。根据样本和训练的方式，又可以分为监督学习、无监督学习、半监督学习和强化学习等类型。

神经网络：在人工智能领域一般指人工神经网络，是一种模仿动物神经网络行为特征，进行分布式并行信息处理人工智能模型。我们通常使用的神经网络都需要通过训练数据进行参数的学习，所以神经网络也可以被归为一种机器学习方法。

深度学习：作为人工智能领域的新兴方向，深度学习目前还没有严格的定义，一般我们把一些具备较多中间隐含层的神经网络称为深度学习模型。

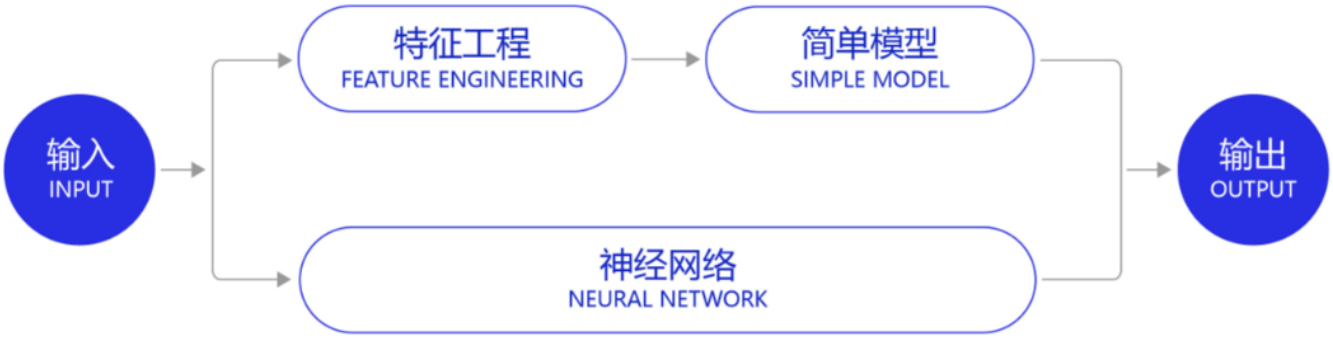
> 机器学习：一种实现人工智能的方法； 深度学习：一种实现机器学习的技术

人工智能做了什么?

- 语音识别：小米家的小爱同学，苹果的Siri；
- 图像识别：交通摄像头的违规拍照识别，刷脸解锁手机；
- 视频识别：抖音视频内容审核，视频社交App的审核机制；
- 文字识别：身份证识别，银行卡识别，扫一扫翻译；
- 语义理解：智能问答机器人，也包括上面的小爱同学、Siri。

深度学习改变了AI 应用的研发模式

深度学习改变了很多领域算法的实现模式。在深度学习兴起之前，很多领域建模的思路是投入大量精力做特征工程，将专家对某个领域的“人工理解”沉淀成特征表达，然后使用简单模型完成任务（如分类或回归）。而在数据充足的情况下，深度学习模型可以实现端到端的学习，即不需要专门做特征工程，将原始的特征输入模型中，模型可同时完成特征提取和分类任务。

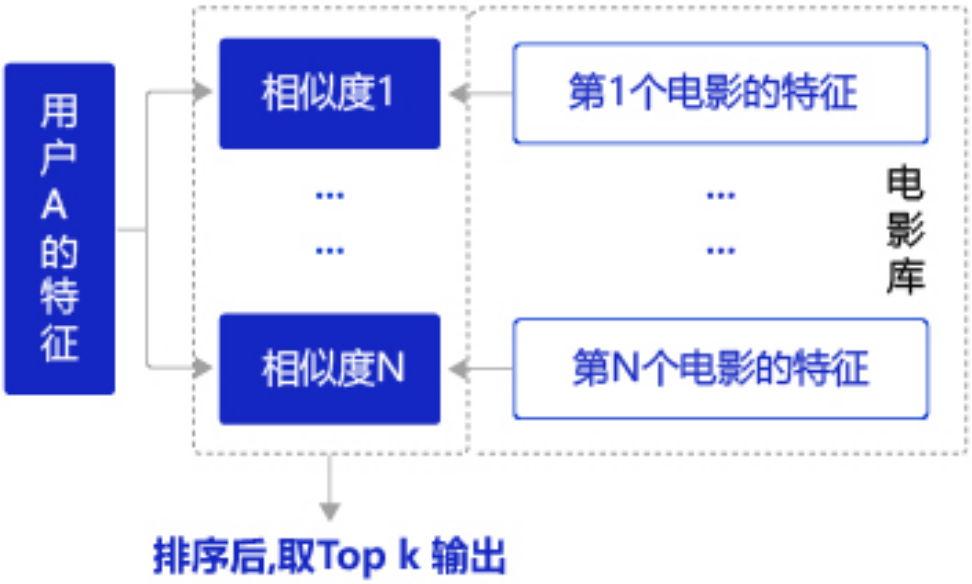


深度学习的应用

- 计算机视觉：卷积神经网络（Convolutional Neural Networks, CNN）是计算机视觉技术最经典的模型结构，图片识别；
- 自然语言处理（Natural Language Processing，简称NLP），文字识别，语义理解；
- 推荐系统，视频平台的电影推荐，电商平台的商品推荐。

深度学习在电影推荐系统中的应用

如果能将用户A的原始特征转变成一种代表用户A喜好的特征向量，将电影1的原始特征转变成一种代表电影1特性的特征向量。那么，我们计算两个向量的相似度，就可以代表**用户A对电影1的喜欢程度**。据此，推荐系统可以如此构建：



假如给用户A推荐，计算电影库中“每一个电影的特征向量”与“用户A的特征向量”的余弦相似度，根据相似度排序电影库，取 Top K 的电影推荐给A。

深度学习框架

- TensorFlow：2015年11月 Google 研制，最流行的深度学习框架，社区强大；
- PyTorch：2016年10月 Facebook 研制，据说用过 TensorFlow 再用 PyTorch 都说香；
- PaddlePaddle(飞桨)：百度研制 2016年 开源，国内第一款深度学习框架，现已平台化；
- MXNet：2017年1月 MXNet 项目进入 Apache 基金会成为 Apache 的孵化器项目。

机器学习方向小结

人岗匹配系统本质就是推荐系统，利用深度学习提取 CV 和 JD 的特征向量，使用相似度算法计算向量距离，排序取 Top K。

优点：训练样本足够多，人岗匹配越精准。

缺点：前期投入成本较高，缺少 AI 大数据人才；前期没有足够训练样本不会提高匹配精准度。

大数据搜索引擎方向

搜索引擎开源项目

- Lucene 大名鼎鼎的搜索引擎类库，如果使用该技术实现，需要对 Lucene 的 API 和底层原理非常了解，而且需要编写大量的 Java 代码；
- Solr 基于 Lucene 使用 Java 实现的一个 Web 应用，可以使用 REST 方式的 HTTP 请求，进行远程 API 的调用；
- Elasticsearch 基于 Lucene ，超越 Lucene，可以使用 REST 方式的 HTTP 请求，进行远程 API 的调用。

Solr vs Elasticsearch

- Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能；
- Solr 支持更多格式的数据，而 Elasticsearch 仅支持json文件格式；
- Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能都由第三方插件提供；
- Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于 Elasticsearch；
- Solr 是传统搜索应用的有力解决方案，但 Elasticsearch 更适用于新兴的实时搜索应用。

Elasticsearch 简介

Elasticsearch 是一个基于 Apache Lucene 的全文检索和分析引擎，可以扩容到上百台服务器，处理PB级结构化 / 非结构化数据。

许多年前，一个叫 Shay Banon 的待业工程师跟随他的新婚妻子来到伦敦，他的妻子想在伦敦学习做一名厨师。而他在伦敦寻找工作的期间，接触到了 Lucene 的早期版本，他想为自己的妻子开发一个方便搜索菜谱的应用。

Elasticsearch发布的第一个版本是在2010年的二月份，从那之后，Elasticsearch 便成了 Github 上最受人瞩目的项目之一，并且很快就有超过 300 名开发者加入进来贡献了自己的代码。后来 Shay 和另一位合伙人成立了公司专注打造 Elasticsearch，他们对 Elasticsearch 进行了一些商业化的包装和支持。但是，Elasticsearch 承诺，永远都将是开源并且免费的。

起源 - Lucene

- 基于 Java 语言开发的搜索引擎库类；
- 创建于1999年，2005年成为 Apache 顶级开源项目；
- Lucene 具有高性能、易扩展的优点；
- Lucene 的局限性：
    - 只能基于 Java 语言开发；
    - 类库的接口学习曲线陡峭；
    - 原生并不支持水平扩展；

从开源到上市

- Elastic Inc - 开源软件 / 上市公司 2018年10月纽交所上市
- 市值超过 100 亿美元，开盘当天涨幅达 94%
- Elasticsearch 软件下载量，超 3.5 亿次
- 10 万＋ 的社区成员
- 7200＋ 订阅用户，分布在 100＋ 国家
- 云服务 - Elastic，Amazon，阿里巴巴，腾讯

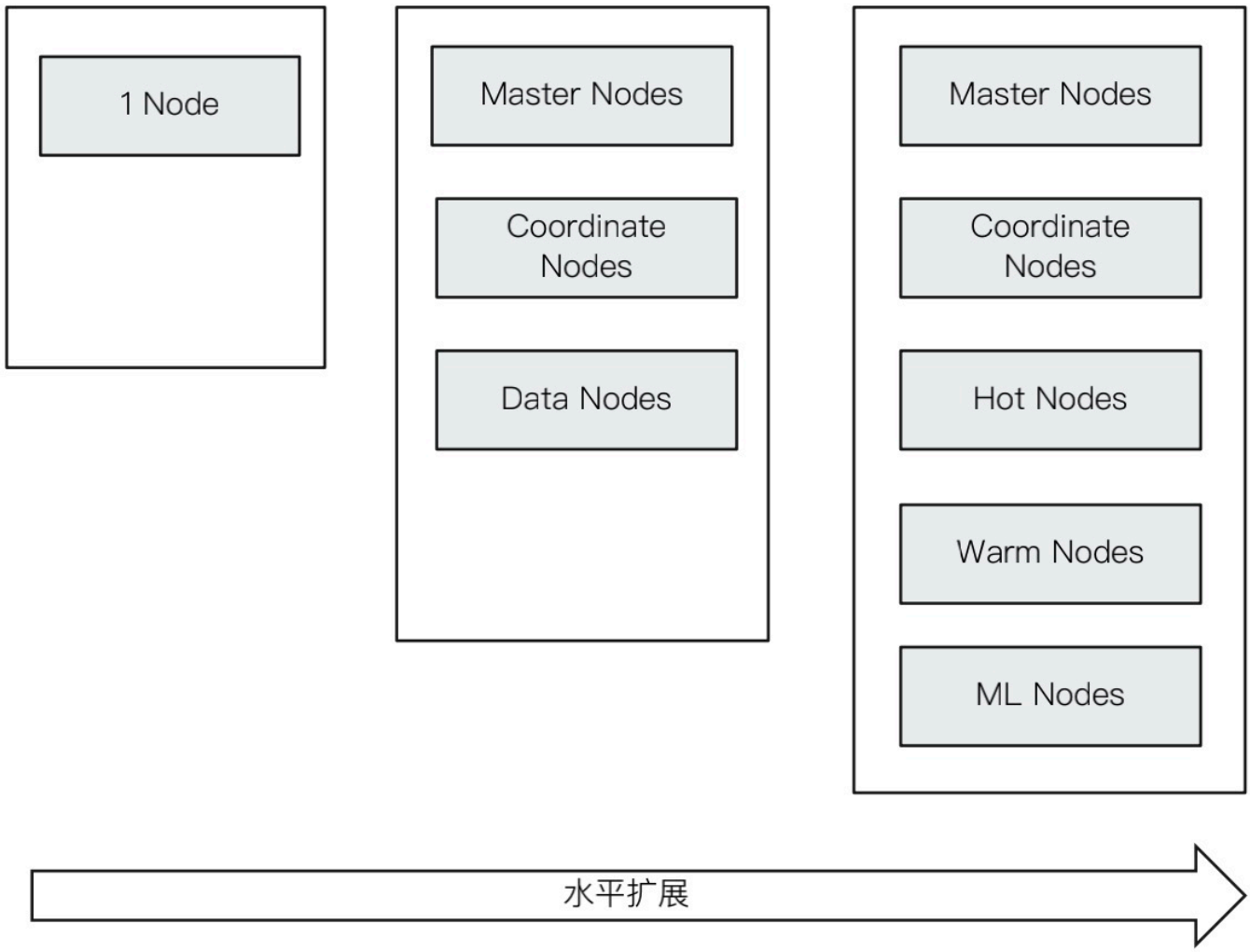
361 systems in ranking, January 2021

Rank Jan 2021	Rank Dec 2020	Rank Jan 2020	DBMS	Database Model	Score Jan 2021	Score Dec 2020	Score Jan 2020
1.	1.	1.	Oracle	Relational, Multi-model	1322.93	-2.66	-23.75
2.	2.	2.	MySQL	Relational, Multi-model	1252.06	-3.40	-22.60
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1031.23	-6.85	-67.31
4.	4.	4.	PostgreSQL	Relational, Multi-model	552.23	+4.65	+45.03
5.	5.	5.	MongoDB	Document, Multi-model	457.22	-0.51	+30.26
6.	6.	6.	IBM Db2	Relational, Multi-model	157.17	-3.26	-11.53
7.	7.	↑ 8.	Redis	Key-value, Multi-model	155.01	+1.38	+6.26
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	151.25	-1.24	-0.19
9.	9.	↑ 10.	SQLite	Relational	121.89	+0.21	-0.25
10.	10.	↑ 11.	Cassandra	Wide column	118.08	-0.76	-2.59
11.	11.	↓ 9.	Microsoft Access	Relational	115.33	-1.41	-13.24
12.	12.	↑ 13.	MariaDB	Relational, Multi-model	93.79	+0.18	+6.34

> DB Engines 常年排在前十 https://db-engines.com/en/ranking

Elasticsearch 的分布式架构

- 集群规模可以从单个扩展至数百个节点，处理 PB 级数据；
- 高可用 & 水平扩展
  - 服务和数据两个纬度
- 支持不同的节点类型
  - 支持 Hot & Warm 架构

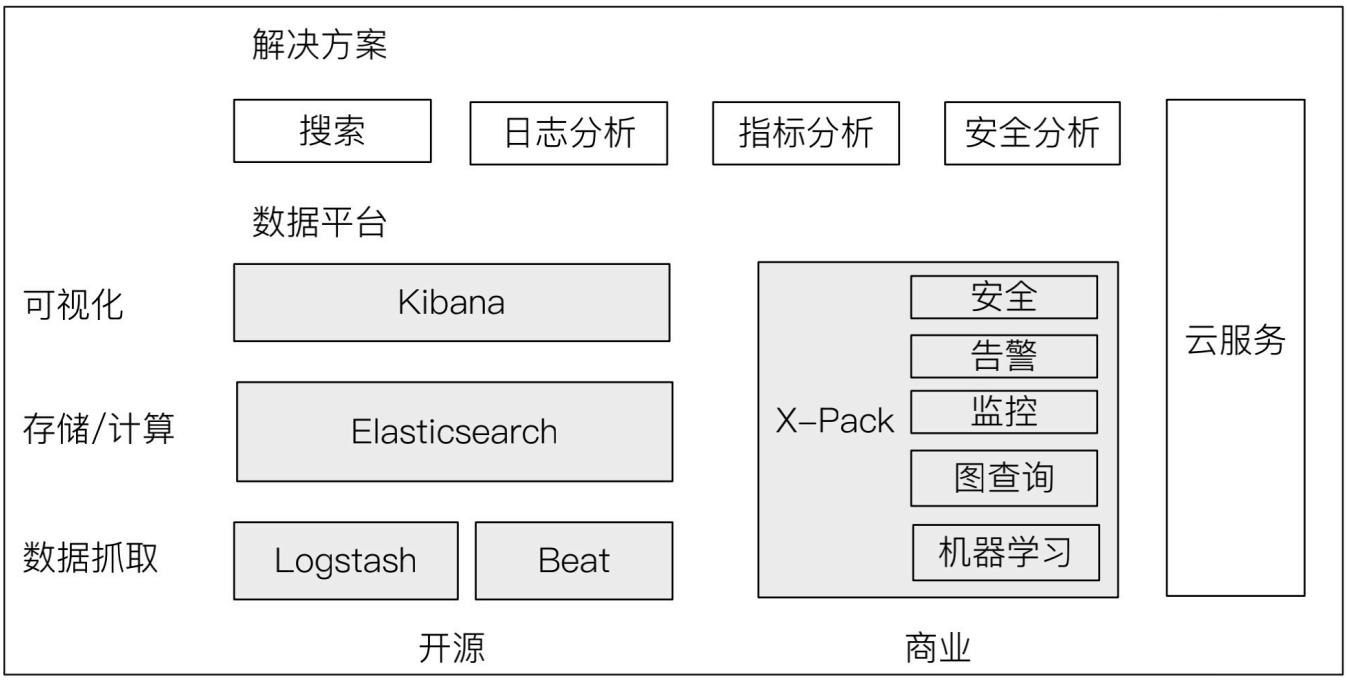


Elasticsearch 的用途

Elasticsearch 在速度和可扩展性方面都表现出色，而且还能够索引多种类型的内容，这意味着其可用于多种用例：

- 应用程序搜索
- 网站搜索
- 企业搜索
- 日志处理和分析
- 基础设施指标和容器监测
- 应用程序性能监测
- 地理空间数据分析和可视化
- 安全分析
- 业务分析

Elastic Stack 生态圈



ELK 应用场景

- 网站搜索/垂直搜索/代码搜索
- 日志管理与分析/安全指标监控/应用性能监控

ES 核心 - 全文检索

在我们生活中的数据总体是分为两种的：结构化数据和非结构化数据。

- 结构化数据：指具有固定格式或有限长度的数据，如数据库，元数据等；
- 非结构化数据：指不定长或无固定格式的数据，如邮件，word文档等磁盘上的文件。

如果是结构化数据，用数据库中的搜索很容易实现，因为数据库中的数据存储是有规律的，有行有列而且数据格式、数据长度都是固定的。

非结构化数据查询有两种办法：

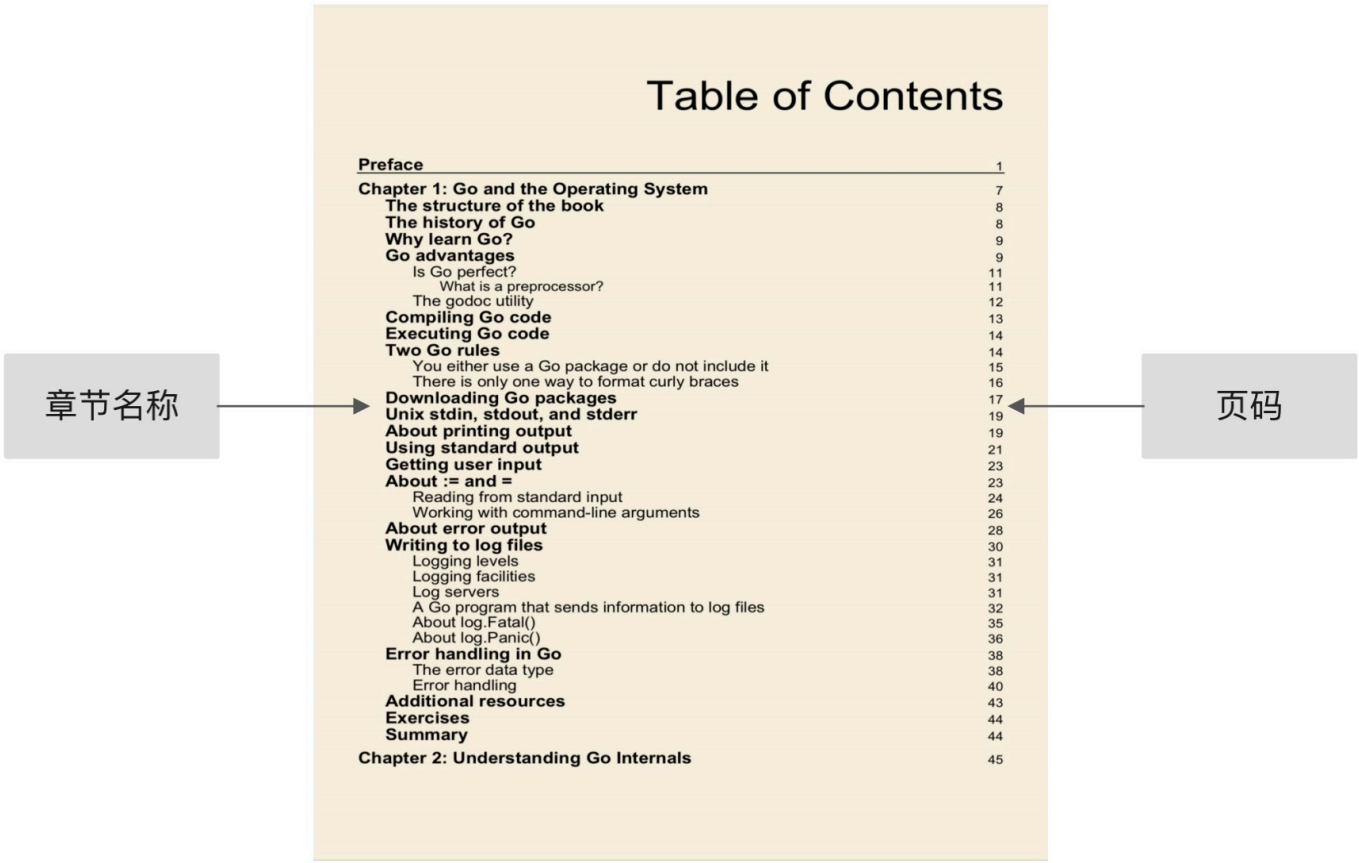
- **顺序扫描法**(Serial Scanning)
- 所谓顺序扫描，比如要找内容包含某一个字符串的文件，就是一个文档一个文档的看，对于每一个文档，从头看到尾，如果此文档包含此字符串，则此文档为我们要找的文件，接着看下一个文件，直到扫描完所有的文件。
- **全文检索**(Full-text Search)
- 将非结构化数据中的一部分信息提取出来，重新组织，使其变得有一定结构，然后对此有一定结构的数据进行搜索，从而达到搜索相对较快的目的。这部分从非结构化数据中提取出的然后重新组织的信息，我们称之**索引**。
    - 例如：字典，字典的拼音表和部首检字表就相当于字典的索引，对每一个字的解释是非结构化的，如果字典没有音节表和部首检字表，在茫茫辞海中找一个字只能顺序扫描。然而字的某些信息可以提取出来进行结构化处理，比如读音，就比较结构化，分声母和韵母，于是将读音拿出来按一定的顺序排列，每一项读音都指向此字的详细解释的页数。我们搜索时按结构化的拼音搜到读音，然后按其指向的页数，

便可找到我们的非结构化数据——也即对字的解释。

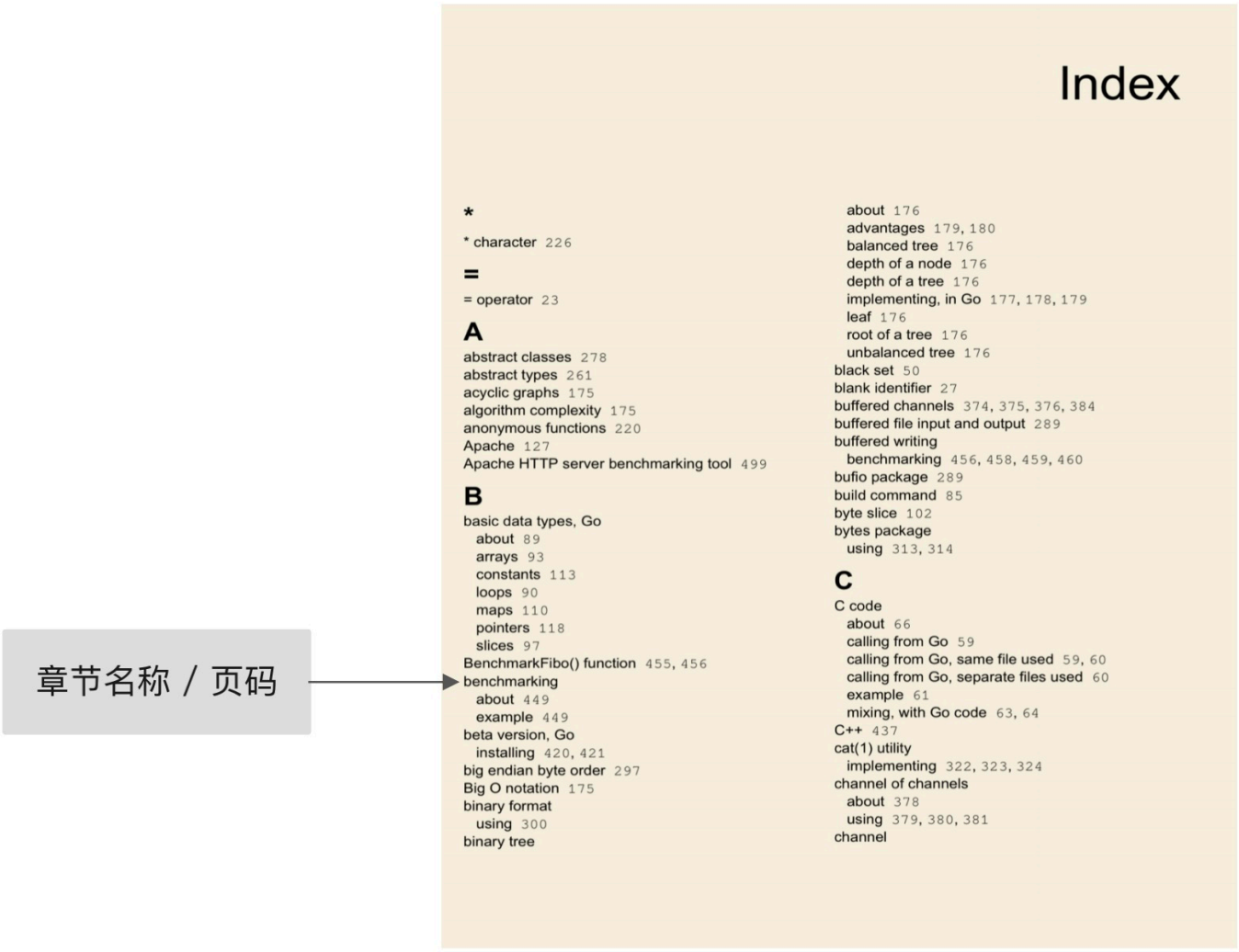
> 这种先建立索引，再对索引进行搜索的过程就叫全文检索(Full-text Search)。

ES 核心 - 倒排索引

目录 - 正排索引



索引页 - 倒排索引



图书和搜索引擎的类比

- 图书
  - 正排索引 - 目录页
  - 倒排索引 - 索引页
- 搜索引擎
  - 正排索引 - 文档ID 到文档内容和单词的关联
  - 倒排索引 - 单词到文档 ID 的关系

ES 倒排索引的核心组成

- 倒排索引包含两个部分
  - 单词词典（Term Dictionary），记录所有文档的单词，记录单词到倒排列表的关联关系
    - 单词词典一般比较大，可以通过B＋树或哈希拉链法实现，以满足高性能的插入与查询
  - 倒排列表（Posting List）- 记录了单词对应的文档结合，由倒排索引项组成
    - 倒排索引项（Posting）
      - 文档 ID
      - 词频 TF - 该单词在文档中出现的次数，用于相关性评分
      - 位置（Position）- 单词在文档中分词的位置。用于语句搜索（phrase query）
      - 偏移（Offset）- 记录单词的开始结束位置，实现高亮显示

文档ID	文档内容
1	Mastering Elasticsearch
2	Elasticsearch Server
3	Elasticsearch Essentials

Doc Id	TF	Position	Offset
1	1	1	<10,23>
2	1	0	<0,13>
3	1	0	<0,13>

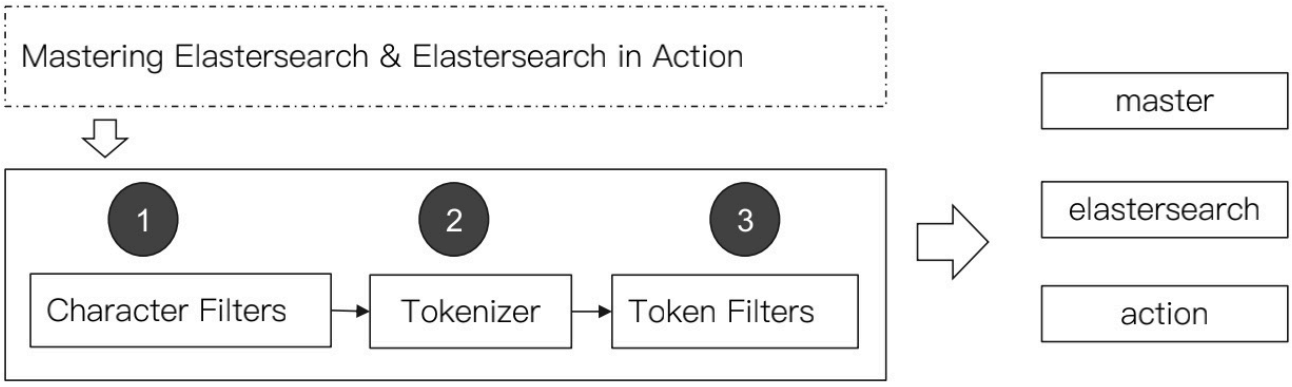
Posting List

ES 核心 - Analyzer

- Analysis - 文本分析是把全文本转换一系列单词（term / token）的过程，也叫分词
- Analysis 是通过 Analyze 实现
  - 可以使用 ES 内置的分析器或者按需定制化分析器
- 除了在数据写入时转换词条，匹配 Query 语句时候也需要用相同的分析器对查询语句进行分析

Analyze 的组成

- 除了在数据写入时转换词条，匹配 Query 语句时候也需要用相同的分析器对查询语句进行分析
  - Character Filters（针对原始文本处理，例如去除html）
  - Tokenizer（按照规则切分为单词）
  - Token Fiter（将切分的的单词进行加工，小写，删除 stopwords，增加同义词）



Elasticsearch 的内置分词器

- Standard Analyzer - 默认分词器，按词切分，小写处理；
- Simple Analyzer - 按照非字母切分（符号被过滤），小写处理；
- Stop Analyzer - 小写处理，停用词过滤（the，a，is）；
- Whitespace Analyzer - 按照空格切分，不转小写；
- Keyword Analyzer - 不分词，直接将输入当作输出；
- Patter Analyzer - 正则表达式，默认\W+（非字符分隔）；
- Language - 提供了30多种常见语言的分词器；
- Customer Analyzer 自定义分词器。

中文分词的难点

- 中文句子，切分成一个一个词（不是一个个字）

- 英文中，单词有自然的空格作为分隔
- 一句中文，在不同的上下文，有不通的理解
    - 这个苹果，不大好吃 / 这个苹果，不大，好吃！

生产级中文分词器

- HanLP 分词器：开源，社区活跃，原始模型用的训练语料是人民日报的语料，当然如果你有足够的语料也可以自己训练。
- IK 分词器 支持自定义字库，支持热更新分词字典。

ES 核心 - 相关性算分

ES 对查询关键字和索引文档的相关度进行打分，得分高的就排在前边。

- 搜索的相关性算分，描述了一个文档和查询语句匹配的程度。ES 会对每个匹配查询条件的结果进行算分 _score
- 打分的本质是排序，需要把最符合用户需求的文档排在前面。ES5之前，默认的相关性算分采用 TF-IDF，现在采用 BM 25

词（Term）	文档（Doc ID）
区块链	1，2，3
的	2，3，4，5，6，7，8
应用	2，3，8，9，10

词频 TF

- Term Frequency: 检索词在一篇文档中出现的频率
    - 检索词出现的次数除以文档的总字数
- 度量一条查询和结果文档相关性的简单方法：简单将搜索中每一个 词的 TF 进行相加
    - $TF(\text{区块链}) + TF(\text{的}) + TF(\text{应用})$
- Stop Word
    - "的"在文档中出现了很多次，但是对贡献相关度几乎没有用处，不应该考虑他们的 TF

逆文档频率 IDF

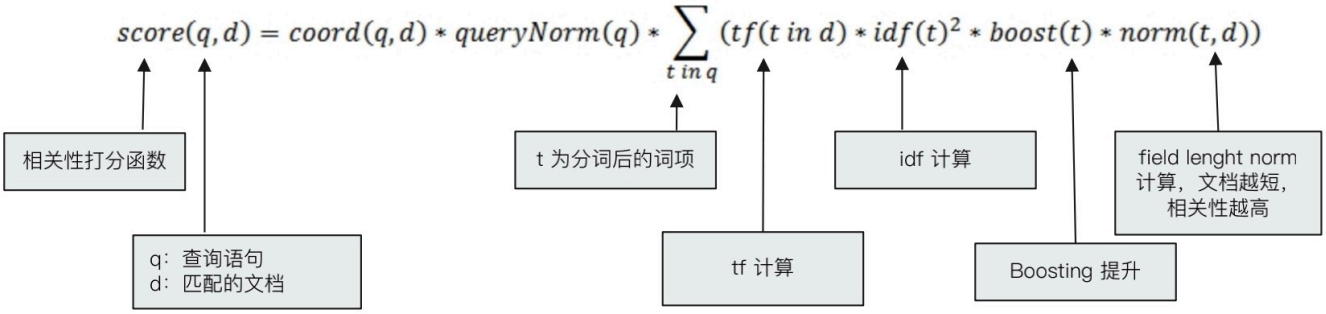
- DF：检索词在所有文档中出现的频率
    - "区块链"在相对比较少的文档中出现
    - "应用"在相对比较多的文档中出现
    - "Stop Word"在大量的文档中出现
- Inverse Document Frequency：简单说 $= log(\text{全部文档数}/\text{检索词出现过的文档总数})$
- TF-IDF 本质上就是将 TF 求和变成了加权求和
- $TF(\text{区块链}) * IDF(\text{区块链}) + TF(\text{的}) * IDF(\text{的}) + TF(\text{应用}) * IDF(\text{应用})$

	出现的文档数	总文档数	IDF
区块链	200万	10亿	log(500) = 8.96
的	10亿	10亿	log(1) = 0
应用	5亿	10亿	log(2) = 1

TF-IDF 的概念

- TF-IDF 被公认为是信息检索领域最重要的发明
- 除了在信息检索，在文献分类和其他相关领域有着非常广泛的应用
- IDF 的概念，最早是剑桥大学的"斯巴克.琼斯"提出
  - 1972年 一 "关键词特殊性的统计解释和它在文献检索中的应用"
  - 但是没有从理论上解释 IDF应该是用 log（全部文档数 / 检索词出现过的文档总数），而不是其他函数。他也没有做进一步的研究
- 1970，1980年代萨尔顿和罗宾逊，进行了进一步的证明和研究，并用香农信息论做了证明
  - http://www.staff.city.ac.uk/~sbrp622/papers/foundations_bm25_review.pdf
- 现代搜索引擎，对 TF-IDF 进行了大量细微的优化

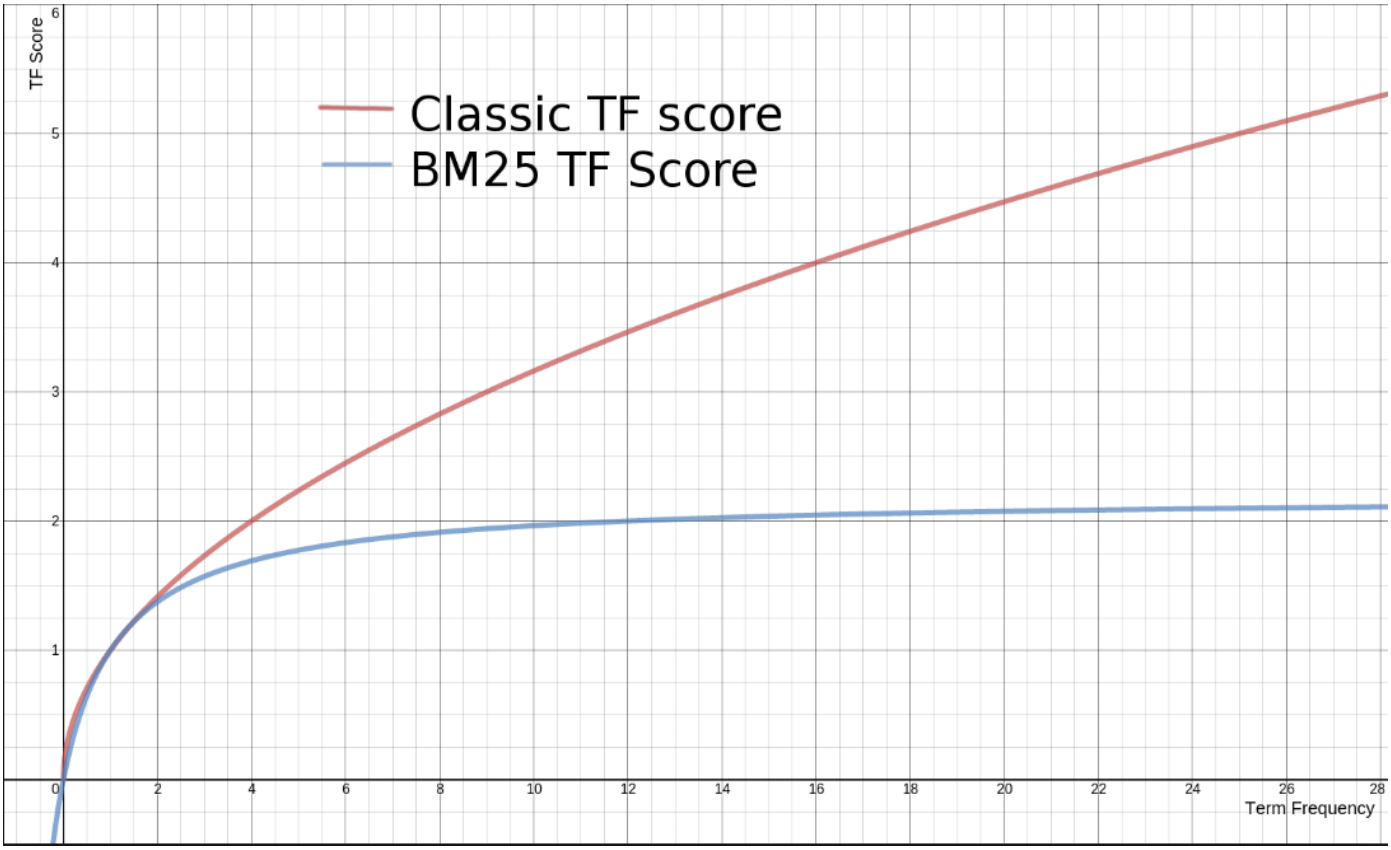
Lucene 中的 TF-IDF 评分公式



- **score(q,d)** 是指查询输入Q和当前文档D的相关性得分；
- **coord(q,d)** 是协调因子，表示输入的Token被文档匹配到的比例；
- **queryNorm(q)** 是查询输入归一化因子，其作用是使最终的得分不至于太大，从而具有一定的可比性；
- **tf(t,d)** 表示输入的一个Token在文档中出现的频率，频率越高，得分越高；
- **idf(t)** 表示输入的一个Token的频率级别，它具体的计算与当前文档无关，而是与索引中出现的频率相关，出现频率越低，说明这个词是个稀缺词，得分会越高；
- **boost(t)** 是查询时指定的权重；
- **norm(t,d)** 是指当前文档的Term数量的一个权重，文档越短，相关性越高。

BM 25

- 从 ES 5开始，默认算法改为 BM25；
- 和经典的 TF-ID F相比，当 TF无限增加时，BM 25 算分会趋于一个数值。



Boosting Relevance

- Boosting 是控制相关度的一种手段
  - 索引，字段 或查询子条件
- 参数 boost的含义
  - 当 boost > 1时，打分的相关度相对性提升
  - 当0 < boost < 1时，打分的权重相对性降低
  - 当 boost < 0 时，贡献负分

ES 核心 - Suggester API

帮助用户在输入搜索的过程中，进行自动补全或者纠错。通过协助用户输入更加精准的关键词，提高后续搜索阶段文档匹配的程度

- 原理：将输入的文本分解为 Token，然后在索引的字典里查找相似的 Term 并返回
- 根据不同的使用场景，Elasticsearch 设计了 4 种类别的 Suggesters
  - Term & Phrase Suggester
  - Complete & Context Suggester

大数据搜索引擎方向小结

搜索推荐本不分家，利用ES全文检索能力实现CV和JD匹配。还可实现日志统计，聚合分析等多个功能。

优点：投入成本相对较低，ES 已成为企业搜索必上的组件，可应用多种场景。

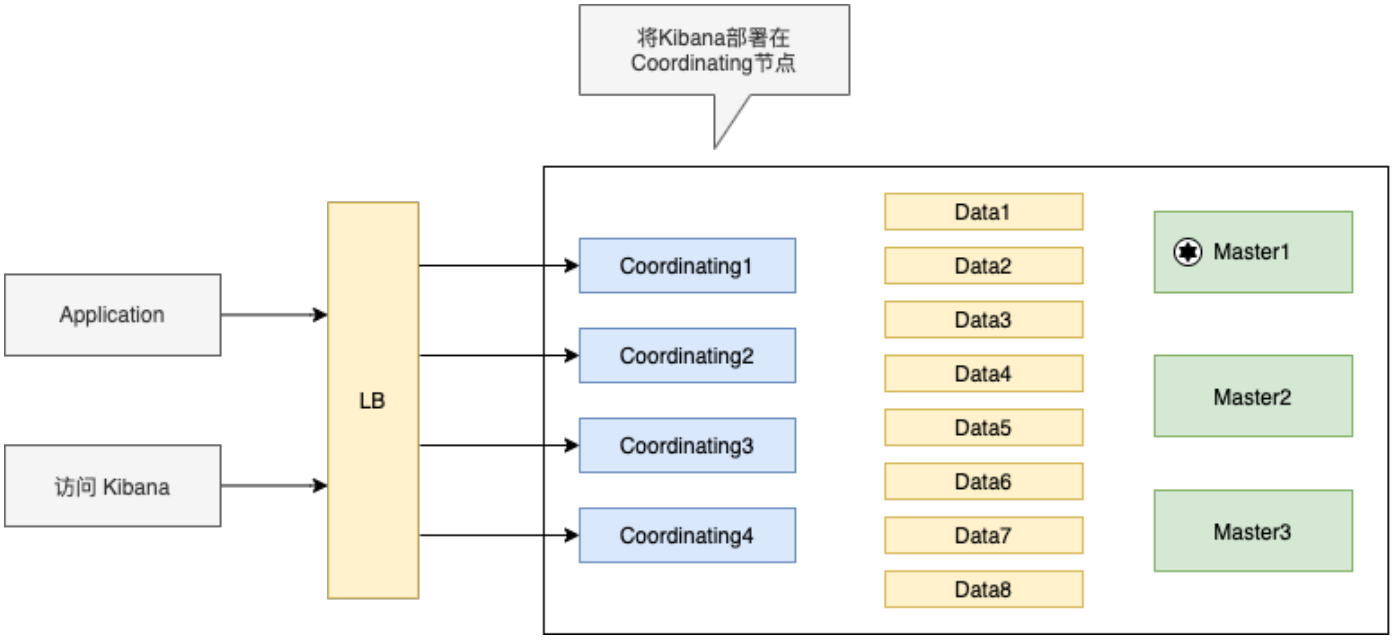
缺点：语义相似性匹配精度受限于分词器。

提高匹配精度方向

1. 扩展分词器字典，提高分词精准度。
2. 探究 ES 向量检索。
3. 内容相似性推荐。

ES 落地方案

ES 生产部署架构



- Master Node
  - 从高可用和避免脑裂的角度出发，生产中 Master Node 建议三节点
  - 负责分片管理，索引创建，集群状态管理等操作
  - 如果和数据节点或者 Coordinating 节点混合部署，数据节点相对有比较大的内存占用，Coordinating节点有时候可能会有开销很高的查询，导致 OOM。这些都有可能影响 Master 节点，导致集群的不稳定
  - 硬件配置：Low CPU；Low RAM；Low Disk
- Data Node
  - 当磁盘 IO 大时，可以增加数据节点
  - 当磁盘容量无法满足需求时，增加数据节点
  - 硬件配置：High CPU；High RAM；SSD
- Coordinating Node
  - 扮演 Load Balancers，负责处理路由请求，数据收集
  - 当系统中有大量的复杂查询及聚合运算时，增加 Coordinating 节点，增加查询性能
  - 硬件配置：Medium/High CPU；Medium/High RAM；Low Disk

JVM 设定

- 从 ES 6 开始， 只支持 64 位 的JVM
  - 配置 `config/vm.options`
- 避免修改默认配置
  - 将内存 Xms 和 Xmx 设置成一样，避免 heap resize 时引发停顿

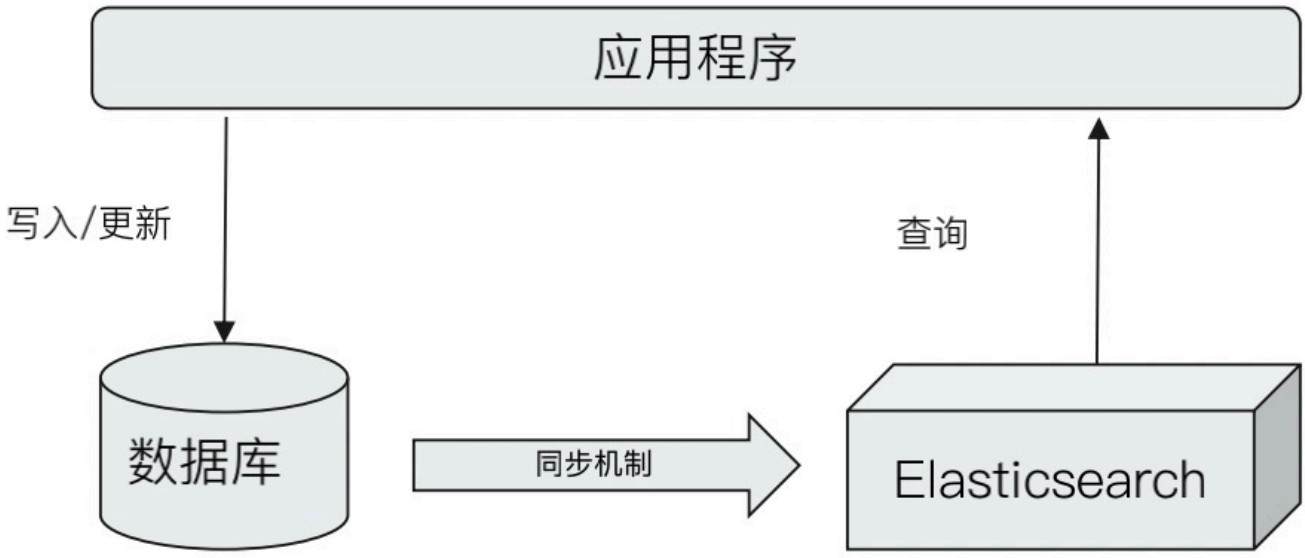
- Xmx 设置不要超过物理内存的50%；单个节点上，最大内存建议不要超过 32G内存
  - https://www.elastic.co/cn/blog/a-heap-of-trouble
- 生产环境，JVM 必须使用 Server 模式
- 关闭 JVM Swapping

Data Node内存设定计算实例

- 内存大小要根据 Node 需要存储的数据来进行估算
  - 搜索类的比例建议: 1:16
  - 日志类: 1:48 - 1:96 之间
- 总数据量 1 T， 设置一个副本 = 2T 总数据量
  - 如果搜索类的项目，每个节点 31*16 = 496G，加上预留空间。所以每个节点最多 400 G数据，至少需要 5 个数据节点
  - 如果是日志类项目，每个节点 31*50 = 1550 GB，2 个数据节点 即可

ES 与数据库集成

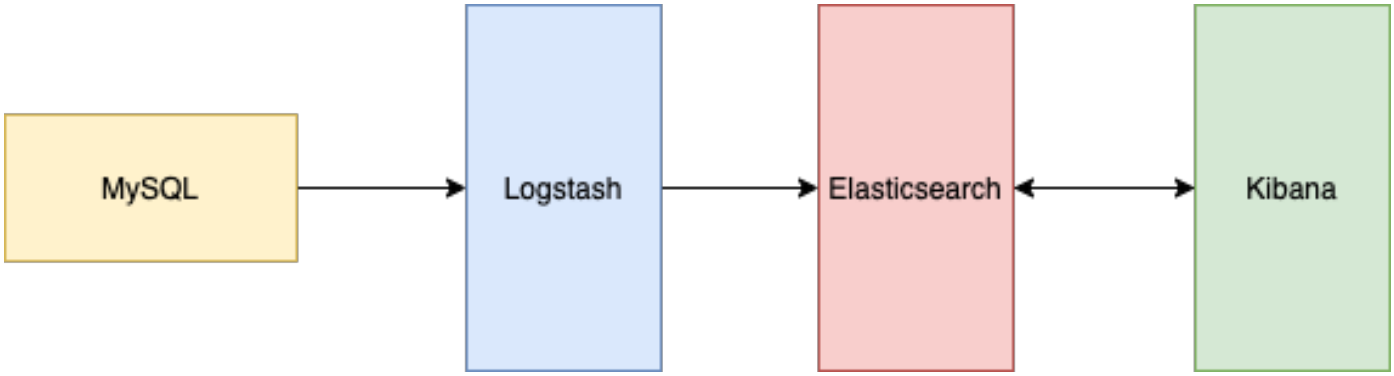
- 单独使用 ES 存储
- 以下情况可考虑与数据库集成
  - 与现有系统的集成
  - 需考虑事务性
  - 数据更新频繁



ES 数据同步方案

冷启动同步

通过 logstash-input-jdbc 插件实现通过 Logstash 批量查询 RDS 中的数据，并将数据迁移到 Elasticsearch。实现的本质是该插件会定期对 RDS 中的数据进行循环轮询，从而在当前循环中找到上次插入或更改的记录，然后批量查询这些记录并迁移至 Elasticsearch。



- 同步全量数据，接受秒级延迟的场景；
- 批量查询数据然后进行同步的场景；
- 使用前，需要先在 Logstash 中上传与 RDS 版本兼容的 SQL JDBC 驱动；
- 需要在RDS的白名单中加入 Logstash 节点的IP地址；
- 需要确保 Logstash 和 RDS 在同一时区（避免同步过程中出现时间标记不符的情况）；
- 需要确保 Elasticsearch 中的`_id`字段与 RDS中 的`id`字段相同；
- 在RDS中插入或更新数据时，需要确保对应记录有一个包含更新或插入时间的字段。

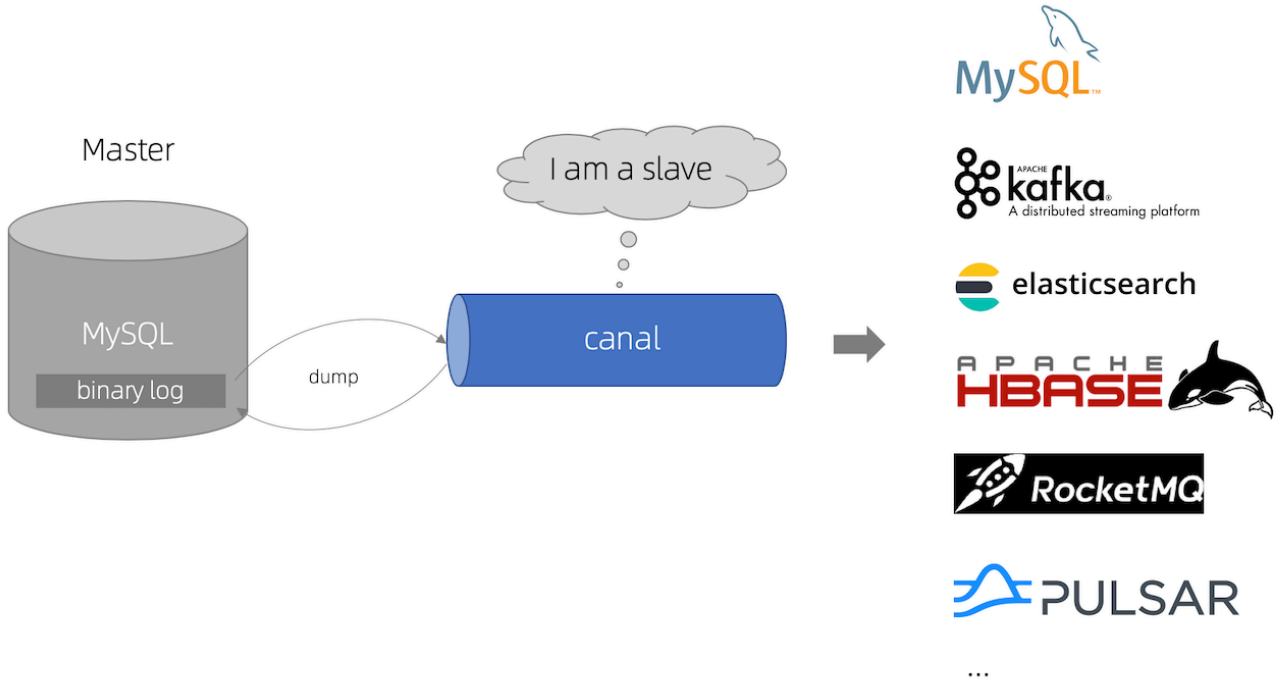
增量同步

Canal 实现 MySQL 数据同步

Canal 是 Alibaba 提供的一个开源产品，能够通过解析数据库的增量日志，提供增量数据的订阅和消费功能。可以用 Canal 将 MySQL 中的增量数据同步至 Elasticsearch。

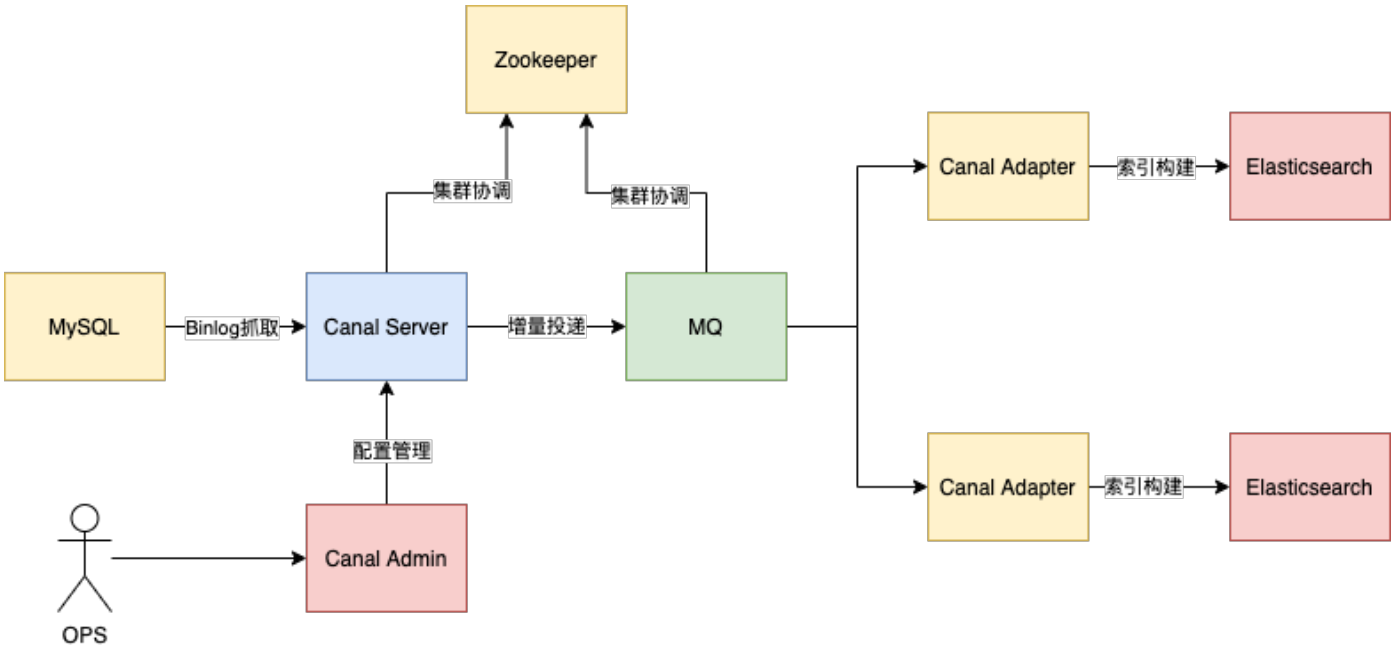
工作原理

canal 会模拟 MySQL 主库和从库的交互协议，从而伪装成 MySQL 的从库，然后向 MySQL 主库发送 dump 协议，MySQL 主库收到dump 请求会向 canal 推送 Binlog，canal 通过解析 binlog 将数据同步到其他存储中去。



- 通过 Binlog 方式实现数据实时同步及订阅；
- 对数据同步的实时性要求较高的场景；
- 支持自定义索引 Mapping 结构，但需保证 Mapping 中定义的字段与 MySQL 中一致。

同步架构



- zookeeper：HA 协调服务；
- canal-server（canal-deploy）：直接监听 MySQL 的 Binlog，把自己伪装成 MySQL 的从库，负责接收数据，投递到下游；
- MQ：可选 Kafka 或 Rocketmq，减少上游数据库订阅压力。将数据库订阅 Binlog 投递到 Rocketmq，下游可以利用 Rocketmq 的Consumer Group，多次、重复消费对应数据，实现业务接耦、缓存一致性等场景；
- canal-adapter：相当于 canal 的客户端，会从MQ中获取数据，然后对数据进行消费，可以同步到 MySQL、Elasticsearch和 HBase 等存储中去；
- canal-admin：为 canal 提供整体配置管理、节点运维等面向运维的功能，提供相对友好的 WebUI 操作界

面，方便更多用户快速和安全的操作。

其他同步方案

- Flink CDC