## 写在前面

网络货运开发部先后在tcs3.0、调度系统、调度中心（中台服务）、新华冶金项目中引入Elasticsearch来实现大数据量复杂查询以及多维度实时统计的需求场景。实际上这些项目到现在还没有在线上运行，但从公司业务的长远规划和业务系统目前面临的问题来看Elasticsearch都将是今后项目落地、解决系统问题的重要组成部分。

所以将这些项目中应用Elasticsearch的思路和实现方案整理成文档分享给大家，

希望能帮助大家将来在类似项目、业务场景中使用ES落地。

## 业务中的问题

### TCS3.0 运单、单据相关列表查询

问题1：

待支付运单列表：列表可选的筛选条件有9个，条件和展示字段落在多张表中。

sql条件复杂、关联多张表，大数据量情况下性能难以保证【可能100W条数据性能就很慢】



### 2、TCS3.0 工作台实时数据统计

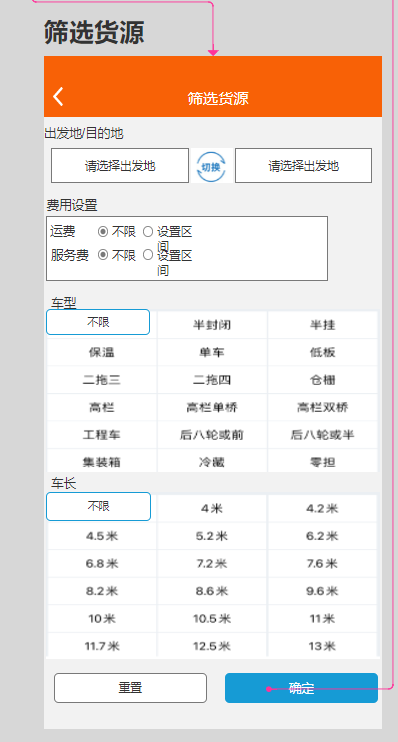
问题2：

实时运输相关数据统计，大数据量情况下性能难以保证



### 3、调度系统，复杂条件匹配货源

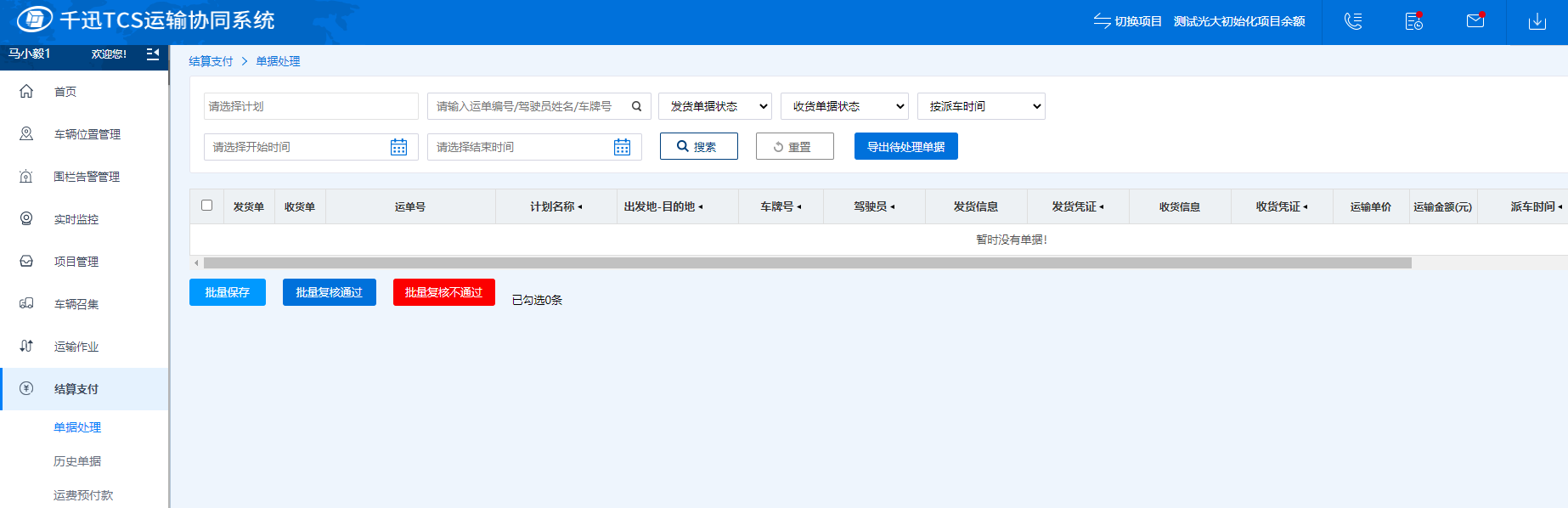
问题3：查询条件复杂且存在多条件模糊匹配，



### 4、TCS2.0运单、单据、发票相关列表响应慢

问题4：

列表响应慢的主要原因是mysql数据量较大，关联查询慢；



## 选择ES

### 选择Elasticsearch的原因

1. 【大数据量多条件复杂查询】 的场景下很快
2. 分布式设计
3. 支持集群多节点、多副本、多分片并且客户端使用无感知
4. 一定的动态扩容设计

### Elasticsearch简介

Elasticsearch 是一个分布式、可扩展、实时的搜索与数据分析引擎，建立在一个全文搜索引擎库 [Apache Lucene™](https://lucene.apache.org/core/" \t "https://www.elastic.co/guide/cn/elasticsearch/guide/current/_top) 基础之上。 Lucene 可以说是当下最先进、高性能、全功能的搜索引擎库【完全用Java编写】—无论是开源还是私有。Lucene 非常 复杂。

Elasticsearch 也是使用 Java 编写的，它的内部使用 Lucene 做索引与搜索，但是它的目的是使全文检索变得简单， 通过隐藏 Lucene 的复杂性，取而代之的提供一套简单一致的 RESTful API。

然而，Elasticsearch 不仅仅是 Lucene，并且也不仅仅只是一个全文搜索引擎。 它可以被下面这样准确的形容：

* 一个分布式的实时文档存储，每个字段可以被索引与搜索
* 一个分布式实时分析搜索引擎
* 能胜任上百个服务节点的扩展，并支持 PB 级别的结构化或者非结构化数据

### 为什么快

ES是通过 Lucene 的倒排索引技术实现比关系型数据库更快的过滤，特别是它对多条件的过滤支持非常好。

简单来说选择ES的主要原因是【大数据量多条件复杂查询】 的场景下依然很快。

#### 倒排索引

重点介绍下倒排索引的原理 和 数据结构，后续落地中要考虑的index的字段类型、磁盘容量、内存大小 实质上都是与倒排索引紧密相关。

倒排索引的构成



如下有一个es index，包含有文档id、姓名、年龄

| **docId** | **姓名** | **年龄** |
| --- | --- | --- |
| 1 | 张发 | 20 |
| 2 | 王启 | 20 |
| 3 | 刘玉 | 17 |

对这个index的document建立的倒排索引如下

姓名（term posting list）

| **张发** | **1** |
| --- | --- |
| 王启 | 2 |
| 刘玉 | 3 |

年龄

| **20** | **[1,2]** |
| --- | --- |
| 17 | [3] |

每个字段都创建自己的倒排索引，倒排索引中的【字段】称为term，

docId的数组（ 如[1,3]） 称为posting list它是一个int数组存储了包含当前term的document的id

#### term

每个字段或者字段中的值都创建自己的倒排索引，倒排索引中的【字段的值】称为term。

#### posting List

posting list它是一个int数组存储了包含当前term的document的id 如[1,3]）

实际上lucene 使用Frame of Reference 编码对posting list 进行了压缩极大减少磁盘占用，这个优化对于减少索引尺寸有非常重要的意义。

通过term、posting list构成的倒排索引能够通过查询条件检索出数据，但是目前的term是无序的要遍历整个term的倒排索引，先对term排序。

#### term dictionary

**排序后的term集合**

查询某个关键词时，lucene遍历倒排索引中所有的term找到对应的docid，如果term数量非常的庞大，遍历的速度就很慢时间复杂度O(n)。

为了避免这种情况，lucene对倒排索引的term进行了排序 通过二分查找 查询term，时间复杂度降为O(logn)。这个排序后的term表就是term dictionary【类似mysql的索引B+树】。

term dictionary保存在磁盘上，要通过logn次磁盘访问拿到结果，但是磁盘的随机读操作任然是非常昂贵的，（一次磁盘的随机读取random access大概需要10ms），所以要尽量少的读磁盘，把term dictionary放到内存。但是整个term dictionary本身又太大了，无法完整的放到内存里，于是就有了term index。

#### term index

类似一本字典大的章节表，索引字段只包含term部分前缀（value保存磁盘offset），再加上索引压缩技术【FST】term index的大小可以只有原来term 的几十分之一，所以内存中可以缓存整个term dictionary。通过term index可以快速找到某个term dictionary的offset，然后从这个位置再往后顺序查找，大大减少了磁盘读取次数，查询速度就非常快。

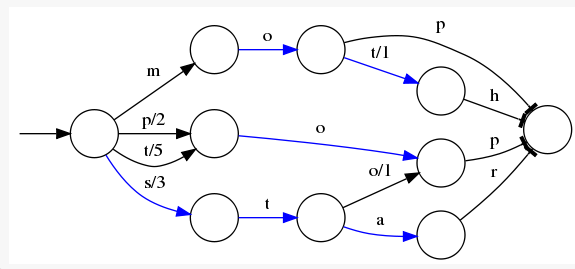
##### FST**（Finite State Transducer）**

使用有限状态机作为数据结构，有两个优点:

1. 空间占用小。通过对词典中单词前缀和后缀的重复利用，压缩了存储空间;
2. 查询速度快。O(len(str))【term index中保存的term前缀长度】的查询时间复杂度。

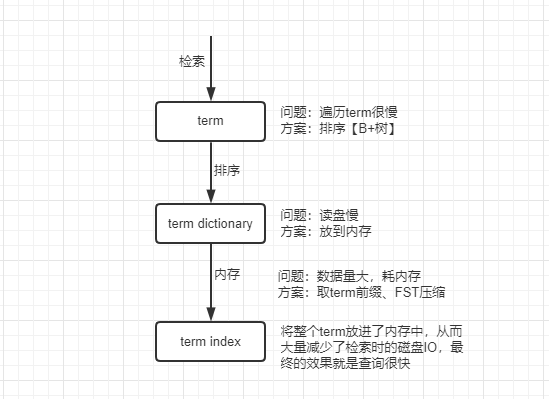
假设我们现在要将mop,moth, pop,star, stop and top(term index里的term

前缀)映射到序号:0，1，2，3，4，5(term dictionary的block位置)。最简单的做法就是定义个Map<string, integer>，大家找到自己的位置对应入座就好了，但Lucene从内存占用少的角度使用FST存储term index：



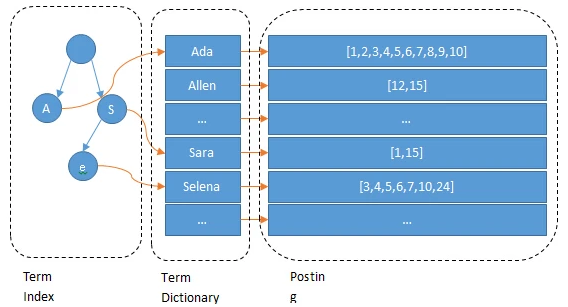
##### 演进过程

遍历term很慢 -> term 排序（查询快）-> term dictionary -> 读盘慢 -> 放到内存 -> 数据量太大 -> 索引压缩 -> term index



#### 索引查询

##### 单个索引查询



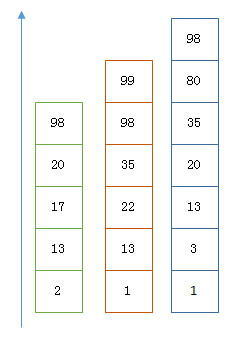
##### 联合索引查询

上面说过选择ES多条件复杂查询的场景下依然很快，原因就是**Lucene支持联合索引查询并且数据合并效率很高**

支持联合索引查询的方式有两种：

* 使用skip list 【跳表】 数据结构。同时遍历多个term【字段的值】的posting list，互相skip；
* 使用bitset【咆哮位图：压缩后的位图】数据结构，对多个filter分别求出bitset，对多个bitset做AN 操作。
* Elasticsearch 支持以上两种的联合索引方式，如果查询的 filter 缓存到了内存中（以 bitset 的形式），那么合并就是两个 bitset 的 AND。如果查询的 filter 没有缓存，那么就用 skip list 的方式去遍历两个 on disk 的 posting list。

###### 利用 Skip List 合并



以上是三个 posting list。我们现在需要把它们用 AND 的关系合并，得出 posting list 的交集。首先选择最短的 posting list，然后从小到大遍历。遍历的过程可以跳过一些元素，比如我们遍历到绿色的 13 的时候，就可以跳过蓝色的 3 了，因为 3 比 13 要小。

最后得出的交集是 [13,98]，所需的时间比完整遍历三个 posting list 要快得多。但是前提是每个 list 需要指出 Advance 这个操作，快速移动指向的位置。

从概念上来说，对于一个很长的 posting list，比如

[1,3,13,101,105,108,255,256,257]

我们可以把这个 list 分成三个 block：

[1,3,13] [101,105,108] [255,256,257]

然后可以构建出 skip list 的第二层：

[1,101,255]

1,101,255 分别指向自己对应的 block。这样就可以很快地跨 block 的移动指向位置了。

###### 利用 bitset 合并

Bitset 是一种很直观的数据结构，对应 posting list 如：

[1,3,4,7,10]

对应的 bitset 就是：

[1,0,1,1,0,0,1,0,0,1]

每个文档按照文档 id 排序对应其中的一个 bit。Bitset 自身就有压缩的特点，其用一个 byte 就可以代表 8 个文档。所以 100 万个文档只需要 12.5 万个 byte。但是考虑到文档可能有数十亿之多，在内存里保存 bitset 仍然是很奢侈的事情。

所以秘诀就在于需要有一个数据结构：

* 可以很压缩地保存上亿个 bit 代表对应的文档是否匹配 filter；
* 这个压缩的 bitset 仍然可以很快地进行 AND 和 OR 的逻辑操作。

于是就有了咆哮位图，其压缩的思路其实很简单。与其保存 100 个 0，占用 100 个 bit。还不如保存 0 一次，然后声明这个 0 重复了 100 遍。

## 落地

如何落地：

1. 解决方案
2. ES API调用
3. ES与DB数据同步
4. 服务器资源规划【磁盘容量、内存大小】

### 问题的解决方案

#### 问题1

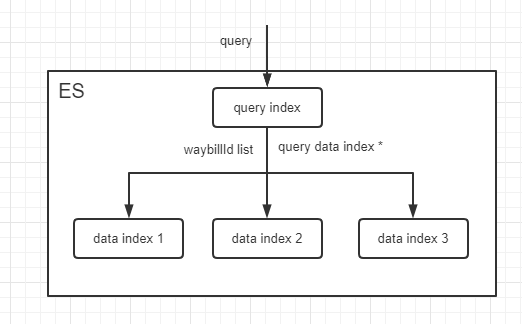
**解决方案**

针对运单相关列表所有的查询条件所对应的字段存放到ES的同一个index中，列表查询数据分两步实现：

1、检索ES返回满足条件数据的waybillId（运单id）结果集

2、拿到waybillId结果集后，查询mysql数据库 获取列表所需要的数据

如果这里的mysql表做了分库或者分表，运单id分散在多个表或者库中那通过waybillId获取mysql数据成本就比较大，这种情况可以将全量的运单相关数据同步到ES，就有了两类index一类是只存了检索条件的**query index**一类是存有全量数据的**data index**。检索query index拿到waybillId再通过waybillId查询data index组装数据。



对于data index，应尽量只对必要字段建立倒排索引，减少其对内存的消耗，内存留给query index。

PS：新华冶金项目需求类似，解决方案

### ES运维部署

官方文档

<https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html>

#### **1.安装**

##### **es版本**

elasticsearch-7.8.0

国内镜像下载地址：<https://mirrors.huaweicloud.com/elasticsearch/7.8.0/>

##### **可视化工具**

###### **Cerebro**

测试环境当前使用的该工具

测试环境地址：http://192.168.2.79:9000/#/connect

下载地址：[https://github.com/lmenezes/cerebro/releases](https://github.com/lmenezes/cerebro/releases" \t "https://www.jianshu.com/p/_blank)

###### **kibana**

国内镜像下载地址：<https://mirrors.huaweicloud.com/kibana/7.8.0/>

#### **2.配置**

##### **elasticsearch.yml**

指定名字、路径

#集群名  
cluster.name: elasticsearch\_production  
#节点名  
node.name: node-1  
#对所有ip访问做出响应  
network.host: 0.0.0.0  
#端口默认9200  
http.port: 9200  
#集群master节点  
cluster.initial\_master\_nodes: ["node-1"]  
#数据  
path.data: /path/to/data1,/path/to/data2   
#日志路径  
path.logs: /path/to/logs  
# 插件  
path.plugins: /path/to/plugins

##### **JVM**

jvm.options

-Xms1g -Xmx1g

修改初始化、最大堆内存大小(默认1G)，堆内存设置的大小取决于数据容量、检索频率、耗内存的检索操作【大数据量检索、聚合、计算】等多个方面

#### **容量规划**

##### 容量评估

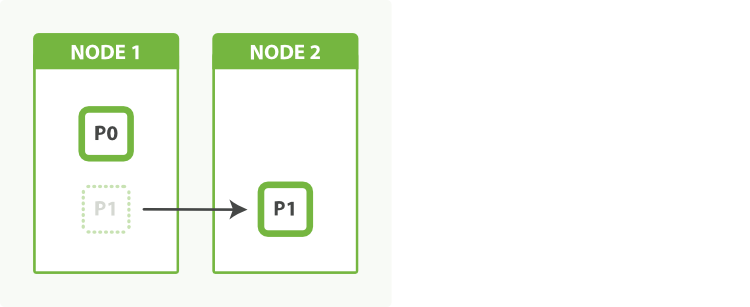
只针对waybill一个index进行容量评估【PS：数据引用自tcs3.0运单】

| **doc数量** | **3分片** |
| --- | --- |
| 5W | 7.5mb |
| 10w | 14.2mb |
| 15W | 22.2mb |
| 50w | 72.5mb |
| 100W | 196.4mb |
| 200w | 311mb |

每1W个 doc大约1.5m, waybill 单个doc大约0.15k，按照公司理想的业务增长情况每天100W个运单，一年产成3.6亿个waybill doc约53G数据5年达到300G。每个分片容量30-50G，30G最佳不超过50G，单个节点5个分片磁盘空间250G。

按照上述情况规划每天100W个运单 5个分片5年内不需要扩容，若每天10W个运单左右3个分片足够。

实际上分片数量还应从集群节点的性能均衡方面考虑【查询性能由最慢节点的响应时间决定】，分片应均匀分布在各节点上平衡各节点的压力。总的来说原则就是为集群分配比node数更多的shard数也就是一个节点多个shard并且分布均匀【一个节点多个shard原因在于后续增加节点ES会自动迁移shard到新的节点上】



第二，预留存储以备错误。(Elastic 官方推荐经验值）

预留 15%警戒磁盘水位空间。

为错误余量和后台活动预留+ 5％。

保留等效的数据节点以处理故障。

第三，容量预估计算方法如下：

总数据量（GB） = 原始数据量（GB） /每天 X 保留天数 X 净膨胀系数 X （副本 + 1）

磁盘存储（GB） = 总数据量（GB）\* ( 1 + 0.15 + 0.05)

数据节点 = 向上取证（磁盘存储（GB）/ 每个数据节点的内存量 / 内存：数据比率）+ 1

Tips：腾讯云 在 2019 4 月的 meetup 分享中建议：磁盘容量大小 = 原始数据大小 \* 3.38。

#### **4.内存规划**

##### **硬件配置**

选择合理的硬件，数据节点尽量使用 SSD；

搜索等性能要求高的场景，建议使用 SSD ，按照 1:10 的比例，配置内存和磁盘；

日志类和查询并发低的场景，可以考虑使用机械硬盘存储，按照 1:50 的比例分配内存和磁盘；

单节点数据建议控制在 2TB 以内，最大不建议超过 5TB ;

##### **内存**

1:30 的比例分配内存和磁盘

###### **预留内存的（少于）一半给lucene**

Lucene 的性能取决于和操作系统的相互作用。如果把所有的内存都分配给 Elasticsearch 的堆内存，将不会有剩余的内存交给 Lucene。 这将严重地影响全文检索的性能

###### **少于32G内存分配给Elasticsearch**

JVM 在内存小于 32 GB 的时候会采用一个内存对象指针压缩技术。分配超过32G内存，每个对象的指针都变长了，就会使用更多的 CPU 内存带宽，实际上失去了更多的内存。当内存到达 40–50 GB 的时候，有效内存才相当于使用内存对象指针压缩技术时候的 32 GB 内存。

###### **禁用 swap**

内存交换 到磁盘对服务器性能来说是致命的。一个内存操作必须能够被快速执行。

如果内存交换到磁盘上，一个 100 微秒的操作可能变成 10 毫秒。 再想想那么多 10 微秒的操作时延累加起来。 不难看出 swapping 对于性能是多么可怕。

禁用swap，需要打开配置文件中的 mlockall 开关。 它的作用就是允许 JVM 锁住内存，禁止操作系统交换出去。在 elasticsearch.yml 文件中，设置如下：

bootstrap.mlockall: true

##### **创建index**

###### **waybill**

针对每个index做具体的分片规划

number\_of\_shards 分片数

number\_of\_replicas 副本数

创建index的put请求

​  
PUT waybill  
{  
    "settings" : {  
        "index" : {  
            "number\_of\_shards" : 5,   
            "number\_of\_replicas" : 0  
       }  
   }  
}

### ES API调用

#### **原生API**

官方文档地址：

https://www.elastic.co/guide/cn/elasticsearch/guide/current/structured-search.html

#### **es bboss组件**

bboss elasticsearch是一套基于query dsl语法操作和访问分布式搜索引擎elasticsearch的o/r mapping高性能开发库，底层基于es restful api。

ORM和DSL二者兼顾，类mybatis。

组件：fkh-es-spring-boot-starter，架构组提供的基于bboss二次封装的组件

bboss文档地址：

[https://esdoc.bbossgroups.com/](https://esdoc.bbossgroups.com/" \l "/?id=%e6%90%9c%e7%b4%a2%e5%bc%95%e6%93%8e%e7%9a%84-orm-%e5%ba%93-elasticsearch-bboss)

#### springdata-elasticsearch框架

类似mybatis-plus 的orm框架，通过对象组装查询条件

参考文档：

基本使用文档【中文】：

<https://www.cnblogs.com/ifme/p/12005026.html>

官方文档：https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html

### 同步方式

#### 同步

**方案**

业务代码有关数据生成 和 修改的地方**同步写入**

**优点**

数据同步更新，无延迟

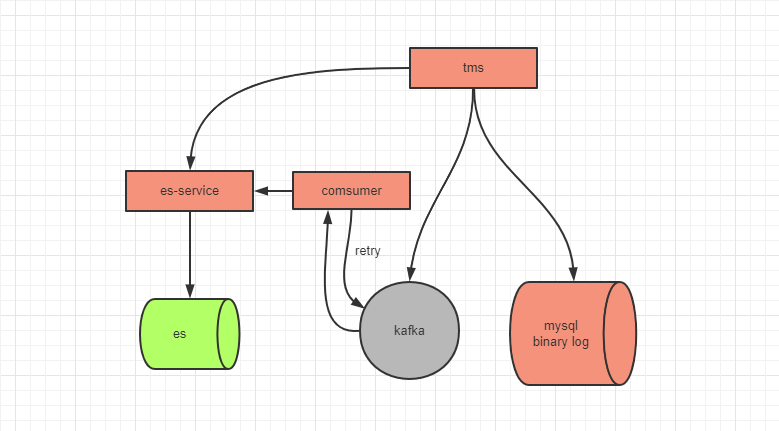
**缺点**

耦合度高、代码逻辑改动大、业务扩展都需要维护，需要处理异常重试、事务控制

#### 异步消息

**方案**

业务代码有关数据生成 和 修改的地方同步写入



**优点**

相对同步写入，对业务代码侵入性更低

**缺点**

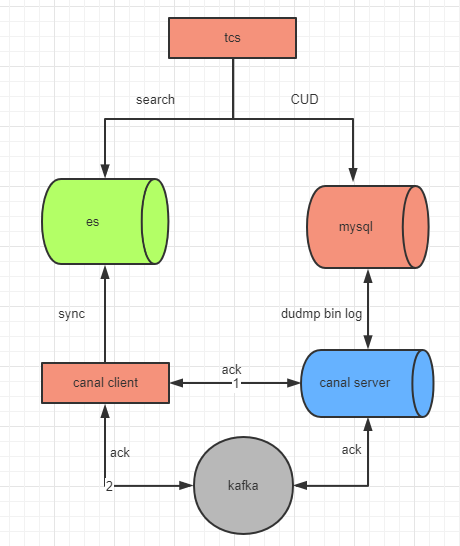
也需要对业务代码进行修改，需要考虑事务问题

相对同步写入，存在一定延时

#### 解析Mysql Binlog

**方案**

**Canal server dump和解析mysql bin log（byte流）为bin log对象，Canal client获到bin log对象映射成java 对象保存到Elasticsearch**



**优点**

与业务完全解耦，后续业务变动、新增数据表 或者未来引入其它的数仓 都不需要对业务代码进行改动

**缺点**

1.存在一定同步延时

2.增加了系统架构复杂度，数据一致性、服务可用性、数据同步吞吐量 等等。。。

最佳实践：

1. canal server将解析的bin log对象投递到消息中间件【kafka】，canal client消费【kafka】消息来同步数据。好处是可以部署多个canal client服务同步数据既保证可用性、数据同步的性能。
2. canal client声明ES index对象，并指定字段的数据类型，需要全文检索的字段设置为text、精确匹配的字段类型为keyword。

* keyword 不支持全文搜索。所以，只能是使用精确匹配、模糊查询，比如 term 查询。
* text 默认支持全文搜索。

### **案例**

##### 新华冶金智慧物流平台

代码地址：

<http://192.168.2.18/v5-product/tcs-mservice-es/tree/canal_1.6.0_20201207/fkh-mservice-es-sync>

###### 服务资源

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **环境** | **ES服务器** | **mysql** | **es监控工具** | **canal server** | **canal client** |
| dev | 2.79 | 2.5 fkh\_logistic\_dev | <http://192.168.2.79:9000/>  账号[host]: skywalking | 2.79 | 2.79 |
| test | 2.5 | 2.5 fkh\_logistic\_test | <http://192.168.2.79:9000/>  账号[host]: mandaloryan | 2.79 | 2.5 |

###### 说明

canal server 只部署在了79上，配置了开发【fkh\_logistic\_dev】、测试【fkh\_logistic\_test】mysql数据库的实例【instance】实例以数据库名来命名。canal client服务分表部署在了2.79开发、2.5测试环境，分别连接canal server的开发、测试实例。开发、测试过程中连接es监控工具查看数据。

## 写在最后

本次分享源于公司的项目中的业务场景，且受限于个人能力本次分享内容比较有限，重在希望帮助大家初步了解Elasticsearch和适用的业务场景以及所要解决的业务中问题，

Elasticsearch其它更加丰富强大的特性、功能、使用场景和细节需要在业务中去实践和学习。

1.ES官方文档地址：

<https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html>

2.bboss文档地址：

<https://esdoc.bbossgroups.com/#/README>

3.springdata-elasticsearch 文档地址：

https://docs.spring.io/spring-data/elasticsearch/docs/current/reference/html

4.公司相关代码地址：

<http://192.168.2.18/v5-product/tcs-mservice-es>

branch : canal\_1.6.0\_20201207



5.Canal 文档地址：

<https://github.com/alibaba/canal>

6.公司已部署可视化工具地址：http://192.168.2.79:9000/#/connect