> 你知道的越多，你不知道的越多

> 点赞再看，养成习惯

> **GitHub** https://github.com/JavaFamily上已经收录有一线大厂面试点脑图、个人联系方式和技术交流群，欢迎Star和指教

前言

这是帅丙真实事件，大家都知道很多公司都是有故障等级这么一说的，这就是敖丙在公司背的P0级故障，敖丙差点因此**被解雇**，事情经过**十分惊心动魄**，我的**心脏病都差点复发**。

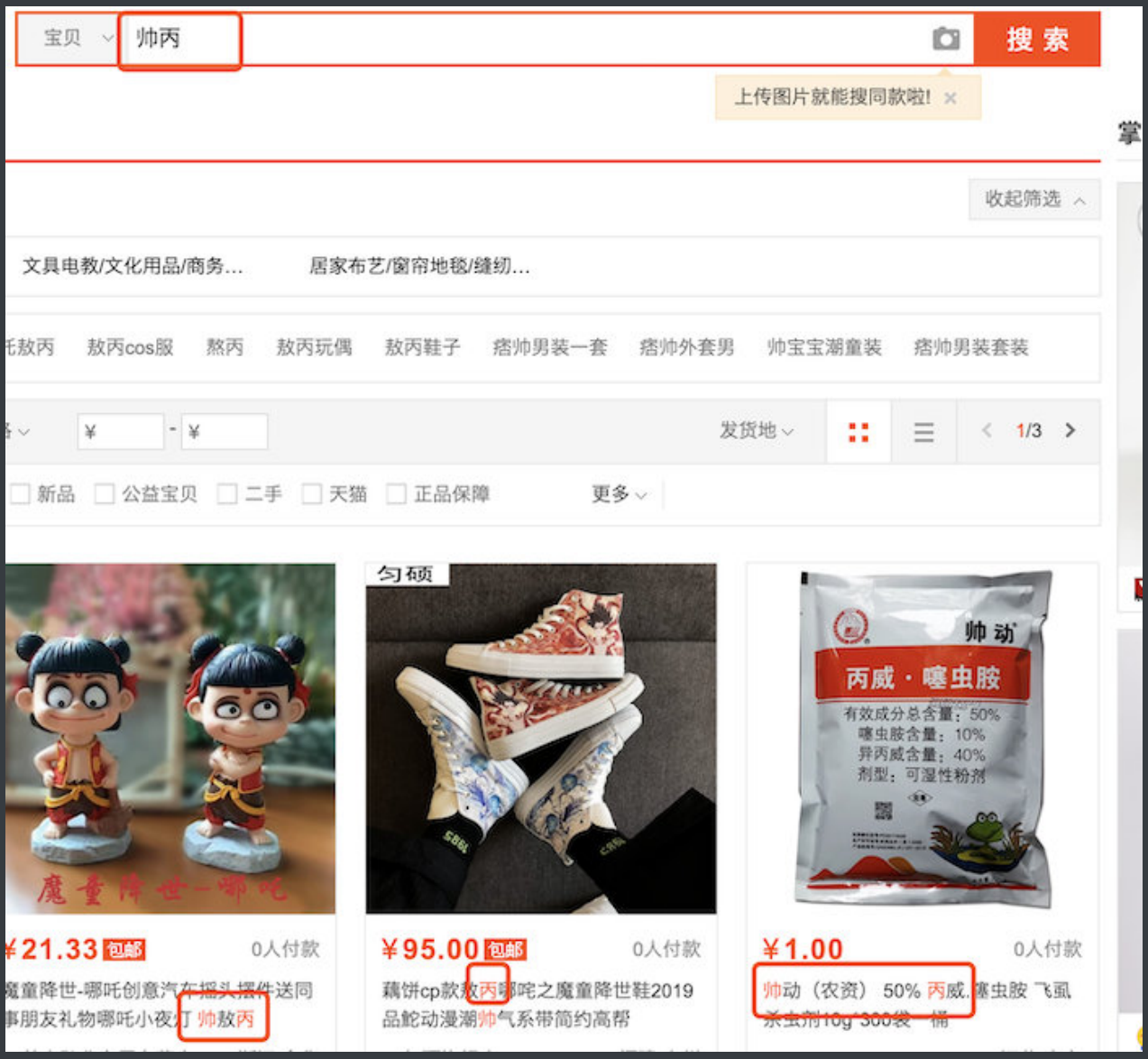
> 事故等级主要针对生产环境，划分依据类似于bug等级。
>
> P0属于最高级别事故，比如崩溃，页面无法访问，主流程不通，主功能未实现，或者在影响面上影响很大（即使bug本身不严重）。
>
> P1事故属于高级别事故，一般属于主功能上的分支，支线流程，核心次功能等，后面还有P2，P3等，主要根据企业实际情况划分。

正文

敖丙之前也负责公司的商品搜索业务，因为业务体量增速太快了，商品表中的商品数据也很快跃入千万级别，查询的RT（response time 响应时间）也越来越高了，而且产品说需要根据**更多维度去查询商品**。

因为之前我们都是根据商品的名称去查询的，但是电商其实都会根据很多个维度去查询商品。

就比如大家去淘宝的查询的时候就会发现，你搜商品名称、颜色、标签等等多个维度都可以找到这个商品，就比如下图的搜索，我只是搜了**【帅丙】**你会发现，名字里面也没有连续的帅丙两个字，有帅和丙的出来了



大家知道的传统的关系型数据库都是用什么 name like %帅丙% 这样的方式查询的，而且查询出来的结果肯定只能是name里面带帅丙的对吧。

那你还想搜别的字段比如什么尺寸、关键词、价格等等，都能搜到帅丙，这相当于是多个维度的了，传统的关系型数据库做不到呀。

做技术选型的时候，帅丙第一时间想到了搜索引擎。

当时市面是比较流行的有：**Apache Lucene**、**Elasticsearch**、**Solr**

搜索引擎我后面会讲**ELK（Elasticsearch、Logstash、Kibana）**和**Canal**，我呀真的是太宠你们了，这样会不会把你们惯坏了。

帅丙我呀，噼里啪啦一顿操作，最后得出结论：

> 相对来讲，如果考虑静态搜索，Sorl相对更合适。
>
> 如果考虑实时，涉及到分布式，Elasticsearch相对合适。

那我们商品还是要实时的呀，你后台改了价格啥的，是不是都要实时同步出去，不然不是炸了嘛。

看到这，我想**可爱的你**和帅丙心中都有了答案：Elasticsearch这是个神一样的引擎。

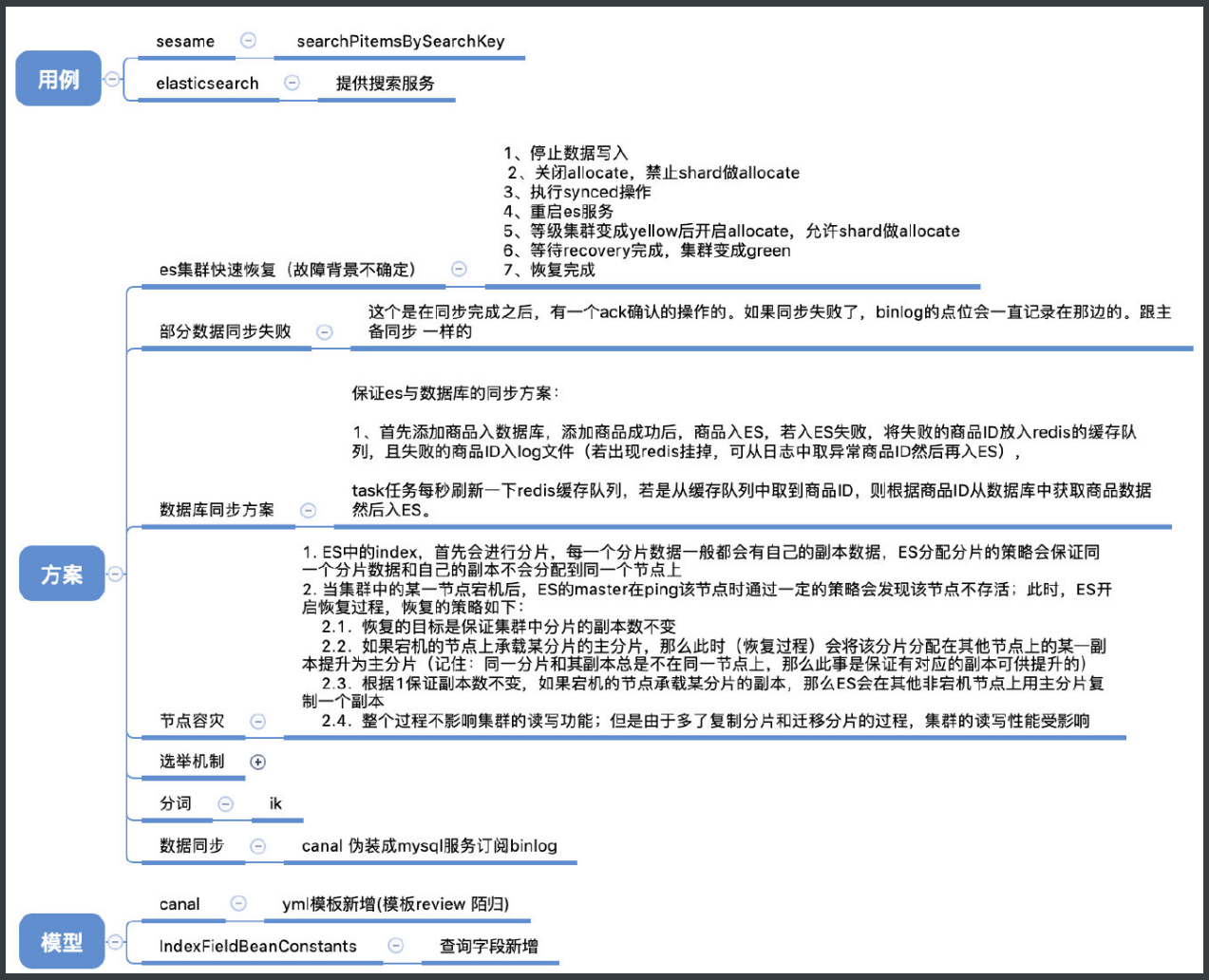
我这里就做一个简单的介绍就好了，细节的点我们后面去他的章节讲，啥都写了，敖丙哪里有这么多素材写文章?

> ElasticSearch是一个基于Lucene的搜索服务器。
>
> 它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。
>
> Elasticsearch是用Java语言开发的，并作为Apache许可条款下的开放源码发布，是一种流行的企业级搜索引擎。
>
> ElasticSearch用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。官方客户端在Java、.NET（C#）、PHP、Python、Apache Groovy、Ruby和许多其他语言中都是可用的。
>
> 根据DB-Engines的排名显示，Elasticsearch是最受欢迎的企业搜索引擎，其次是Apache Solr，也是基于Lucene。

看过敖丙之前文章的朋友都知道，我们做技术选型之前，要做什么呀，**设计**！

我们要去了解这玩意的**好处**、**坏处**、**常见的坑**、**出了问题的应急预案**等等，还有他的**数据同步机制**啊，**持久化机制啥**的，就是高可用嘛。

同样的我不大篇幅介绍了，以后都会写的嘛，我就给大家看看我当时做的设计吧。


用例
sesame
searchPitemsBySearchKey
elasticsearch
提供搜索服务
方案
es集群快速恢复（故障背景不确定）
1、停止数据写入
2、关闭allocate，禁止shard做allocate
3、执行synced操作
4、重启es服务
5、等级集群变成yellow后开启allocate，允许shard做allocate
6、等待recovery完成，集群变成green
7、恢复完成
部分数据同步失败
这个是在同步完成之后，有一个ack确认的操作的。如果同步失败了，binlog的点位会一直记录在那边的。跟主备同步 一样的
数据库同步方案
保证es与数据库的同步方案：
1、首先添加商品入数据库，添加商品成功后，商品入ES，若入ES失败，将失败的商品ID放入redis的缓存队列，且失败的商品ID入log文件（若出现redis挂掉，可从日志中取异常商品ID然后再入ES），
task任务每秒刷新一下redis缓存队列，若是从缓存队列中取到商品ID，则根据商品ID从数据库中获取商品数据然后入ES。
节点容灾
1. ES中的index，首先会进行分片，每一个分片数据一般都会有自己的副本数据，ES分配分片的策略会保证同一个分片数据和自己的副本不会分配到同一个节点上
2. 当集群中的某一节点宕机后，ES的master在ping该节点时通过一定的策略会发现该节点不存活；此时，ES开启恢复过程，恢复的策略如下：
2.1. 恢复的目标是保证集群中分片的副本数不变
2.2. 如果宕机的节点上承载某分片的主分片，那么此时（恢复过程）会将该分片分配在其他节点上的某一副本提升为主分片（记住：同一分片和其副本总是不在同一节点上，那么此事是保证有对应的副本可供提升的）
2.3. 根据1保证副本数不变，如果宕机的节点承载某分片的副本，那么ES会在其他非宕机节点上用主分片复制一个副本
2.4. 整个过程不影响集群的读写功能；但是由于多了复制分片和迁移分片的过程，集群的读写性能受影响
选举机制
分词
ik
数据同步
canal 伪装成mysql服务订阅binlog
模型
canal
yml模板新增(模板review 陌归)
IndexFieldBeanConstants
查询字段新增

这个只是最初的demo，详细的终稿我就不给大家看了，因为有很多公司内部的逻辑。

不过大家还是可以看到敖丙真的考虑了很多，还是那句话，不打没把握的仗！

设计做好敖丙就~~卡卡卡~~的用起来了。

说实话，**真香**，这玩意真的好用，学习成本也很低，查询语句分分钟掌握了，官网文档把功能介绍得清晰无比。

> https://www.elastic.co/cn/

用着用着重头戏来了，你们都知道敖丙我是做电商活动的，都是什么很高的流量打进来这样，还是如往常一样上线了一个活动。

这是一个月飞风高的夜晚，丝丝凉风迎面吹来，敖丙悠闲的坐在椅子上，手里拿着破旧的茶杯，喝着外婆炒的苦荆茶，享受着这惬意的时光。

突然，说时迟那时快，运维打来了紧急电话ES集群CPU打到了99%要挂了，**我的心蓦然一痛**，心里还在庆幸还是集群没崩。

然后他接着说了一句，不好集群挂了！

敖丙卒，本篇完....

开玩笑的哈，不过当时敖丙真的**要死的心真的都要有了**，就在崩掉的1分钟内，就有用户反馈搜索未响应，我第一时间想到的就是重启，于是我一个健步冲出去，开启电脑，进机器，输入了重启命令。

好了，是的好了，还好有惊无险，不过只过了10秒，集群又99%了，呐呢?

我又只能重启了，这次没挂，过了很久很久，直到活动结束，还是没挂。

查找问题

但是这次影响到线上，3分钟的搜索未响应，我想我估计明天是要去财务领工资，提前回家过年了。

还好Leader说没事，先找到问题，把他修复掉。

你们都知道敖丙天才来的，我第一时间想到的就是看日志，我登上去看es没报错，再看本身的服务，除了超时的错误啥都没有，卧槽，是的当时我脑袋嗡嗡响。

不过我继续想为啥是我的搜索挂了，**会不会是有人搜了什么奇怪的东西?**

我打开了我的搜索日志！！！

卧槽这不是吧，哪个坑爹玩意搜这么长的一串中文，差不多250个字吧。

但是我一想，搜这么长也不应该打挂服务啊，**会不会是我写了bug**!

我脸颊流下一滴汗水💦，我看了看周围，发现没人注意到我的紧张，我**故作镇定**的把它擦掉。

我仔细一想，别人查询虽然长，就算查数据库也没事啊，为啥es就报错了？会不会?

Es有Bug！没错肯定是Es的锅。

那为啥会这样呢，我直接跟老大这样解释也好像不行啊，**还是要被开除的吧！**

于是我去看看看代码，我在关键词使用了通配符，我当时是为了匹配更多内容才这么做的，类似数据库的like，Es的通配符就是： * 帅丙 * 这样**在关键词前后加“*”号去查询**。

后面我发现就是通配符的锅，那**柯南丙**就说一下为啥会这样的问题出现。

许多有RDBMS/SQL背景的开发者，在初次踏入ElasticSearch世界的时候，很容易就想到使用通配符(Wildcard Query)来实现模糊查询（比如用户输入补全），因为这是和SQL里like操作最相似的查询方式，用起来感觉非常舒适。

然而帅丙的故障就揭示了，**滥用Wildcard query可能带来灾难性的后果**。

我当时首先复现了问题

复现方法

1. 创建一个只有一条文档的索引

```
POST test_index/type1/?refresh=true

{

"foo": "bar"

}
```

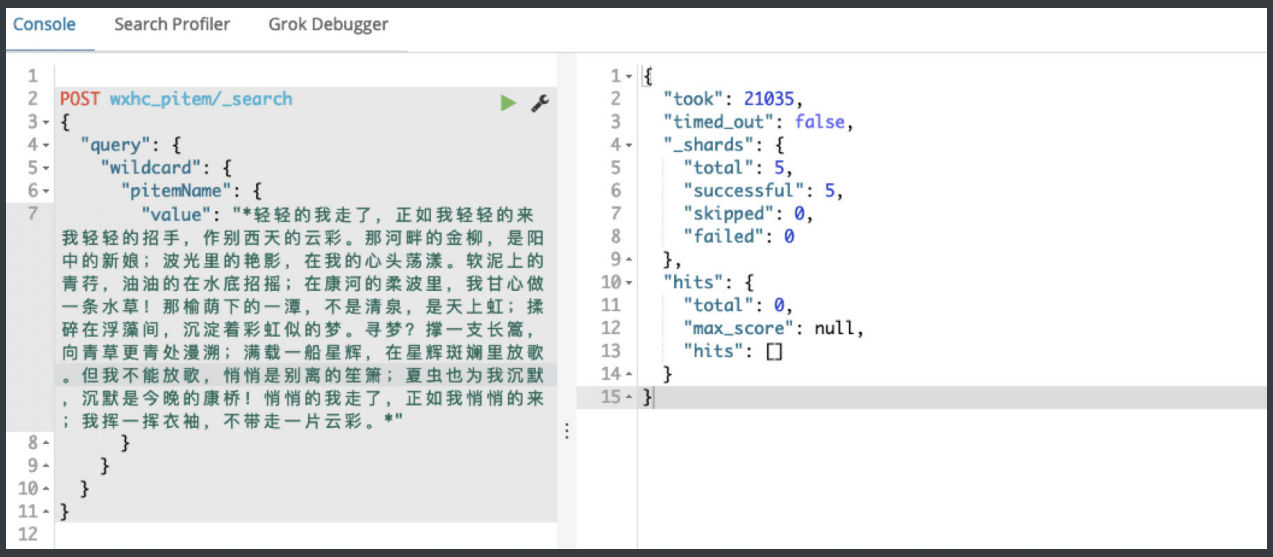
2.使用wildcard query执行一个首尾带有通配符*的长字符串查询

```
POST /test_index/_search

{

"query": {

"wildcard": {

"foo": {

"value": "轻轻的我走了，正如我轻轻的来；我轻轻的招手，作别西天的云彩。那河畔的金柳，是夕阳中的新娘；波光里的艳影，在我的心头荡漾。软泥上的青荇，油油的在水底招摇；在康河的柔波里，我甘心做一条水草！那榆荫下的一潭，不是清泉，是天上虹；揉碎在浮藻间，沉淀着彩虹似的梦。寻梦？撑一支长篙，向青草更青处漫溯；满载一船星辉，在星辉斑斓里放歌。但我不能放歌，悄悄是别离的笙箫；夏虫也为我沉默，沉默是今晚的康桥！悄悄的我走了，正如我悄悄的来；我挥一挥衣袖，不带走一片云彩。"

}

}

}

}
```

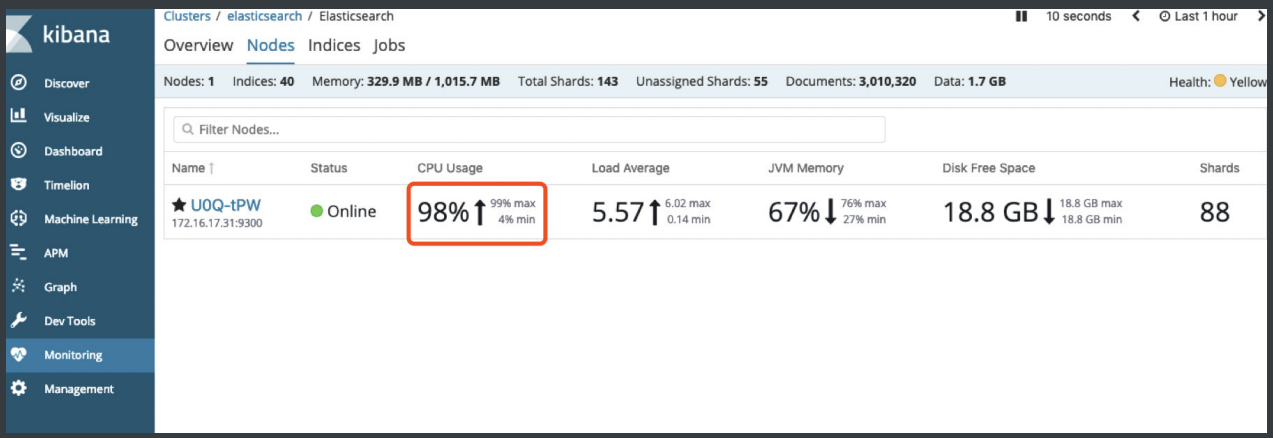
3. 查看结果

```
{

"took": 3445,

"timed_out": false,

"_shards": {

"total": 5,

"successful": 5,

"failed": 0

},

"hits": {
```

```
    "total": 0,
    "max_score": null,
    "hits":
  }
  }
```



即使no hits，耗时却是惊人的3.4秒 (测试机是macbook pro, i7 CPU)，并且执行过程中，CPU有一个很高的尖峰。



线上的查询比我这个范例要复杂得多，会同时查几个字段，实际测试下来，一个查询可能会执行十几秒钟。

再有比较多长字符串查询的时候，集群可能就DOS了。

探查深层次根源

为什么对只有一条数据的索引做这个查询开销这么高？ 直觉上应该是瞬间返回结果才对!

回答这个问题前，可以再做个测试，如果继续加大查询字符串的长度，到了一定长度后，ES直接抛异常了，服务器ES里异常给出的cause如下:

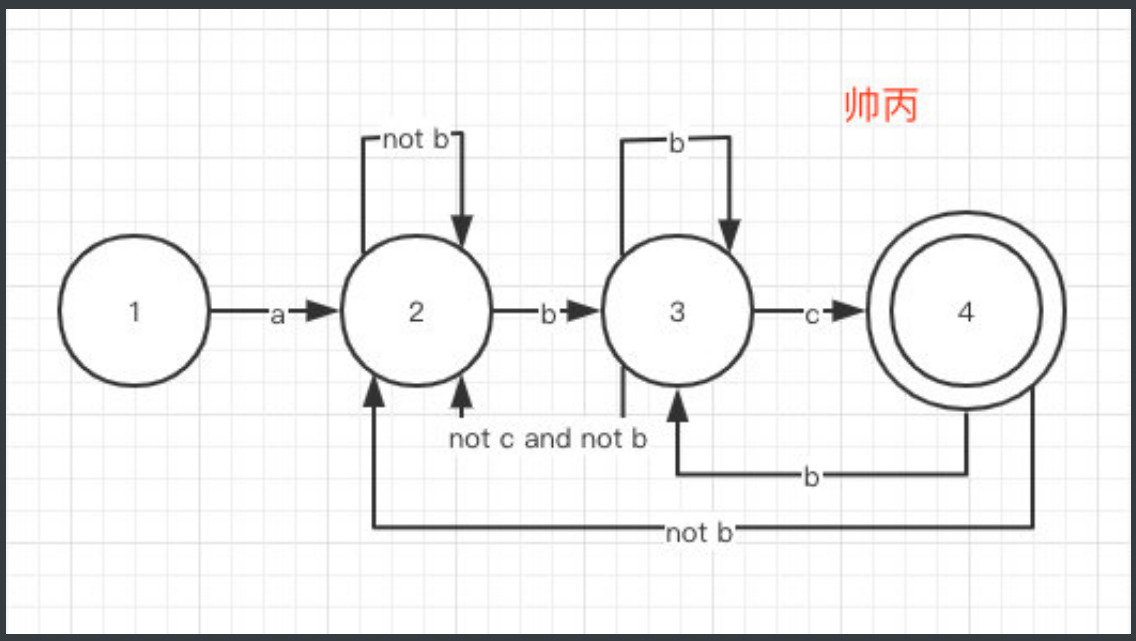
> Caused by: org.apache.lucene.util.automaton.TooComplexToDeterminizeException: Determinizing automaton with 22082 states and 34182 transitions would result in more than 10000 states. at org.apache.lucene.util.automaton.Operations.determinize(Operations.java:741) ~[lucene-core-6.4.1.jar:6.4.1

解释：该异常来自org.apache.lucene.util.automaton这个包，异常原因的字面含义是说"**自动机过于复杂而无法确定状态： 由于状态和转换太多，确定一个自动机需要生成的状态超过10000个上限**"

柯南丙网上查找了大量资料后，终于搞清楚了问题的来龙去脉。

为了加速通配符和正则表达式的匹配速度，Lucene4.0开始会将输入的字符串模式构建成一个DFA (Deterministic Finite Automaton)，带有通配符的pattern构造出来的DFA可能会很复杂，**开销很大**。

比如a*bc构造出来的DFA就像下面这个图一样:



Lucene构造DFA的实现

看了一下Lucene的里相关的代码，构建过程大致如下:

1. org.apache.lucene.search.WildcardQuery里的toAutomaton方法，遍历输入的通配符pattern，将每个字符变成一个自动机(automaton)，然后将每个字符的自动机链接起来生成一个新的自动机。

```
    public static Automaton toAutomaton(Term wildcardquery) {
            List<Automaton> automata = new ArrayList<>();
            String wildcardText = wildcardquery.text();
```

```
            for (int i = 0; i < wildcardText.length();) {
                final int c = wildcardText.codePointAt(i);
                int length = Character.charCount(c);
                switch(c) {
                    case WILDCARD_STRING:
                        automata.add(Automata.makeAnyString());
                        break;
                    case WILDCARD_CHAR:
                        automata.add(Automata.makeAnyChar());
                        break;
                    case WILDCARD_ESCAPE:
                        // add the next codepoint instead, if it exists
                        if (i + length < wildcardText.length()) {
                            final int nextChar = wildcardText.codePointAt(i +
    length);
                            length += Character.charCount(nextChar);
                            automata.add(Automata.makeChar(nextChar));
                            break;
                        } // else fallthru, lenient parsing with a trailing \
                    default:
                        automata.add(Automata.makeChar(c));
                }
                i += length;
            }
            return Operations.concatenate(automata);
        }
```

2. 此时生成的状态机是不确定状态机，也就是Non-deterministic Finite Automaton（NFA)。
3. org.apache.lucene.util.automaton.Operations类里的determinize方法则会将NFA转换为DFA

```
    /**
      \* Determinizes the given automaton.
      \* <p>
      \* Worst case complexity: exponential in number of states.
      \* @param maxDeterminizedStates Maximum number of states created when
      \*  determinizing. Higher numbers allow this operation to consume more
      \*  memory but allow more complex automatons. Use
      \*  DEFAULT_MAX_DETERMINIZED_STATES as a decent default if you don't know
      \*  how many to allow.
      \* @throws TooComplexToDeterminizeException if determinizing a creates an
      \*  automaton with more than maxDeterminizedStates
      */
```

代码注释里说这个过程的时间复杂度最差情况下是状态数量的指数级别！

为防止产生的状态过多，消耗过多的内存和CPU，类里面对最大状态数量做了限制

```
    /**
     * Default maximum number of states that {@link Operations#determinize}
    should create.
     */
    public static final int DEFAULT_MAX_DETERMINIZED_STATES = 10000;
```

在有首尾通配符，并且字符串很长的情况下，这个determinize过程会产生大量的state，甚至会超过上限。

至于NFA和DFA的区别是什么？ 如何相互转换?

网上有很多数学层面的资料和论文，限于帅丙算法方面有限的知识，无精力去深入探究。

但是一个粗浅的理解是: NFA在输入一个条件的情况下，可以从一个状态转移到多种状态，而DFA只会有一个确定的状态可以转移，因此DFA在字符串匹配时速度更快。

DFA虽然搜索的时候快，但是构造方面的时间复杂度可能比较高，特别是带有首部通配符+长字符串的时候。

回想Elasticsearch官方文档里对于Wildcard query有特别说明，**要避免使用通配符开头的term**。

> " Note that this query can be slow, as it needs to iterate over many terms. In order to prevent extremely slow wildcard queries, a wildcard term should not start with one of the wildcards * or ?."

结合对上面Wildcard query底层实现的探究，也就不难理解这句话的含义了！

小结： Wildcard query应杜绝使用通配符打头，实在不得已要这么做，就一定需要限制用户输入的字符串长度。

最好换一种实现方式，通过在index time做文章，选用合适的分词器，比如nGram tokenizer预处理数据，然后使用更廉价的term query来实现同等的模糊搜索功能。

对于部分输入即提示的应用场景，可以考虑优先使用completion suggester, phrase/term suggeter一类性能更好,模糊程度略差的方式查询，待suggester没有匹配结果的时候，再fall back到更模糊但性能较差的wildcard, regex, fuzzy一类的查询。

补充： 有同学问regex, fuzzy query是否有同样的问题，答案是有，原因在于他们底层和wildcard一样，都是通过将pattern构造成DFA来加速字符串匹配速度的。

回忆：为啥之前挂了一次重启恢复了，马上又挂了？用户搜了两次。。。

解决方案

其实解决这种问题很简单，既然知道关键词长了会有问题，我就**做限制**嘛，大家可以去看看搜索引擎某度、某宝啥的，是不是都做了长度限制？

我复制了很长的一段汉字进去百度就是这个结果咯，某宝过长都返回**默认页面**了。

如果你的产品一定要给用户一点东西，简单，找出一些热词分析出来就好了，或者给点热搜商品**兜底**。

我怎么做的呢？判断字符串长度大于50我就直接返回空数组了，这样对用户体验好点，你返回个参数错误或者默认错误别人还**以为你有Bug**呢对吧。

总结

其实敖丙我啥事故等级都没背哈哈，这个算是事故，但是敖丙我这么可爱，领导也心疼我啊，肯定不会怪我的拉，主要是我设计都考虑了很多方案和场景了，没想到有这个坑。（yy：敖丙你个渣男，又是标题党，人家还以为你没工作了要养你呢！）

大家也可以通过这次事故体会到，技术选型的时候，**方案的重要性**了吧，就算你考虑不全，但是不至于真正的问题来了手足无措啊，并不是所有的事故都可以像这次这样重启就搞定了，**不要存有侥幸心理，心存敬畏**。

絮叨

敖丙啊，又有牌面了，得到**阿里云消息中间件团队**小伙伴的认可，并且发现居然是我学姐-**风云**（花名）！！！

她是个好学的小姐姐，大家多多像优秀的仔学习，学姐不是做技术的，但是都在不断学习，说实话我的眼角又湿了。

别跑，投票！！！

我准备把我的公众号**JavaFamily** 这个名字改了，这个名字还是差点意思，但是又不能叫敖丙了，被注册商标了，我就问了下群里的人才，目前有两个我比较喜欢的

- **帅丙**
- **三太子敖丙**
- 其他给我留言

因为这个可能会陪伴我很久，甚至直到死去，希望大家都给点建议哈哈。

别问我为啥要跟敖丙这个名字相关，**再问自杀**！

我花名就叫这个，所以😂