## （169）ES核心概念和原理

1. 什么是搜索

2、实现搜索功能？

关系型数据库：性能差、不可靠、结果不准确（相关度低）

3、倒排索引（基于document）

（1）、倒排索引：

数据结构：1、包含这个关键词的document list

2、关键词在每个doc中出现的次数 TF - term frequency

3、关键词在整个索引中出现的次数 IDF - inverse doc frequency

4、关键词在当前doc中出现的次数

5、每个doc的长度，越长相关度越低

6、包含这个关键词的所有doc的平均长度

（2）、Lucene：jar包，帮我们创建倒排索引，提供了复杂的API

（3）、如果使用lucune做集群实现搜索，会有哪些问题

①、节点一旦宕，数据丢失，可用性差

②、自己维护（自己创建管理索引），单台节点的承载请求的能力是有限的，需要人工做负载。

1. Elasticsearch：分布式、高性能、高可用、可伸缩、易维护 ES≠搜索引擎
2. 分布式的搜索、存储和数据分析引擎：
3. 优点：

①面向开发者、屏蔽了Lucene的复杂特性，集群自动发现（cluster discovery）

②自动维护数据在多个节点上的建立

③会帮我们做搜索请求的负载均衡

④自动维护冗余副本，保证了部分节点宕机情况下仍然不会有任何数据丢失

⑤es基于Lucene提供了很多高级功能：复合搜索、聚合分析、基于地理位置

⑥对于大公司可以构建几百台服务器的大型分布式集群，处理PB级别数据；对于小公司，开箱即用，门槛低上手简单

⑦相对传统数据库，提供了全文检索，同义词处理，相关度排名，聚合分析以及海量数据的近实时（NTR）处理

1. 应用领域：

①全文检索、高亮、搜索推荐

②用户行为日志（用户点击、浏览、收藏、评论）

③BI（Business Intelligence商业智能），数据分析：数据挖掘统

④Github：代码托管平台，几千亿行代码

⑤ELK：E（数据存储）、L（日志采集）、K（可视化）

1. es核心概念
2. Cluster（集群）：每个集群至少包含两个节点
3. Node：集群中的每个节点，一个节点不代表一台服务器
4. Field：一个数据字段，与index和type一起，可以定位一个doc
5. Document：ES最小的数据单元 Json，可以理解为mysql的row
6. Index：一类相同或者类似的doc，比如一个员工索引、商品索引。可以理解为MySQL的一个db
7. Type：逻辑上的数据分类，可以理解为mysql的一个table
8. Shard分片：

①一个index包含多个Shard，默认5P，默认每个P（主分片，可读可写）分配一个R（副本分片，只读），P的数量在创建所以的呢时候设置，如果想修改，需要重新创建索引

②每个Shard都是一个Lucene实例，有完整的创建索引的处理请求能力

③es会自动在nodes上为我们做shard均衡

④一个doc是不可能同时存在于多个PShard中的，但是可以存在于多个RShard中

⑤P和对应的R不能同时存在于同一个节点，所以最低的可用配置是两台节点，互为主备

1. es的容错机制以及如何实现高可用

（4）、容错机制：

步骤①、Master选举

步骤②、Replica容错

步骤③、重启故障机

步骤④、数据恢复

## （171） ES环境安装、健康值检查以及CRUD

1、图解es高可用

1. es和kibana环境
2. 安装es：

①：JDK->依赖

②：下载

③：启动-> ./elasticsearch -d

④：验证

（2）开发模式和生产模式

①开发模式：默认配置（未配置发现设置），用于学习阶段

②生产模式：会触发es的阴道检查，学习阶段不建议修改集群相关的配置

1. ES-node
2. ：Role

①Master：主节点

②voting：仅投票

1）node.voting.only

③coordinationg：仅协调

1. ：Node-type：

①Master-eligble node（候选节点）

②Date node（数据节点）

③Ingest node：

④Machine learning node（机器学习节点）

1. 集群健康检查 /\_cluster/health /\_cat/health?v
2. Green：所有p shard和r shard均为active，集群很健康
3. Yellow：至少一个replica shard不可用，但数据仍然是完整的
4. Red：至少有一个p shard不可用，数据不完整，集群不可用

<https://nodejs.org/en/download/> node -v

npm install -g grunt-cli grunt -version

<https://github.com/mobz/elasticsearch-head>

改elasticsearch-head-master\Gruntfile.js hostname:’\*’

npm install npm run start

默认端口9100

## ES环境安装、健康值检查以及CRUD

1. 基于XX系统的CRUD
   1. 创建索引：PUT /索引?pretty
   2. 查看索引：GET \_cat/indices?v
   3. 删除索引：DELETE /索引?pretty
   4. 插入数据：

PUT /index/type（写死\_doc）/id

{

Json数据

}

* 1. 查询单个数据：GET /product/\_doc/1
  2. 更新数据：

①全量替换

默认就是（没写进去的field就会被抹掉）

PUT /product/\_doc/1

{

“field1”:value,

“field2”:value,

“field3”:value

}

②指定字段更新

POST /index/type（写死\_doc）/id/\_update

{

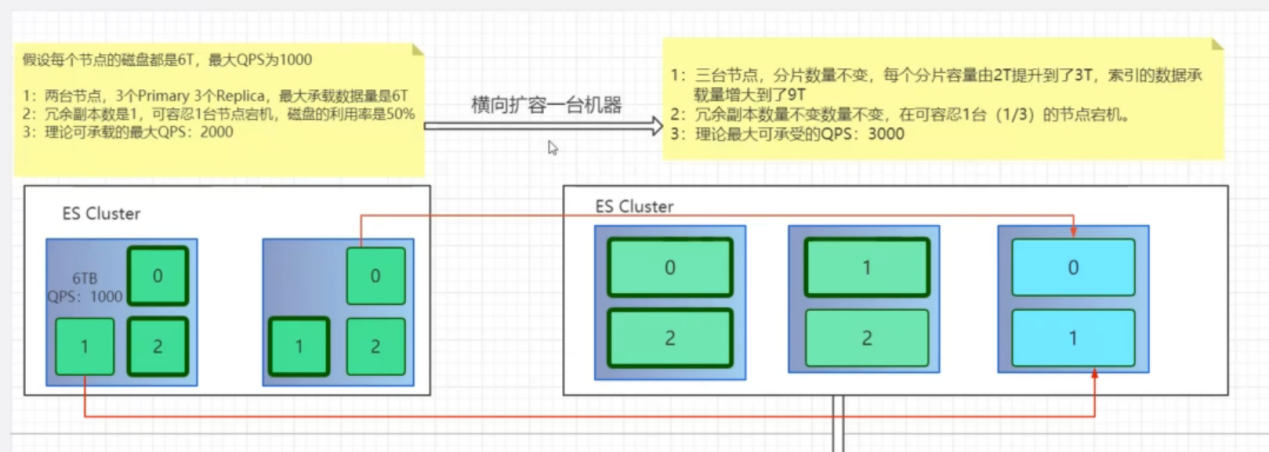
“doc”:{

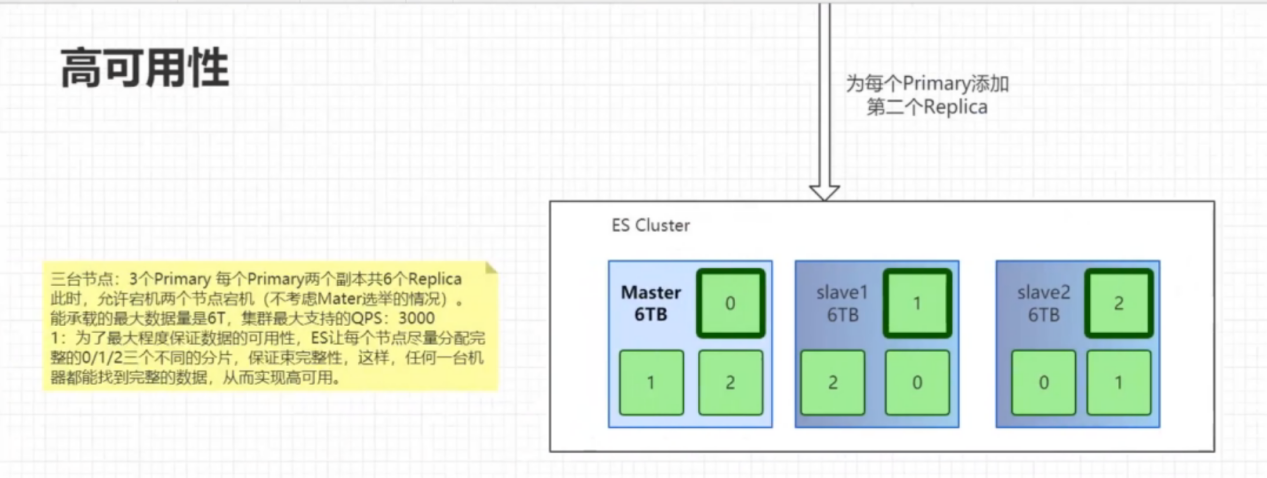
“field”:value

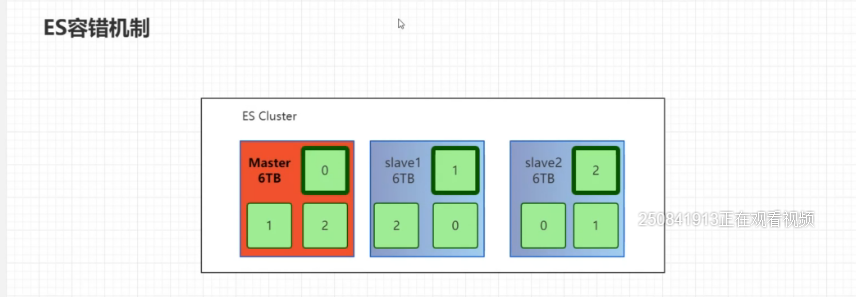
}

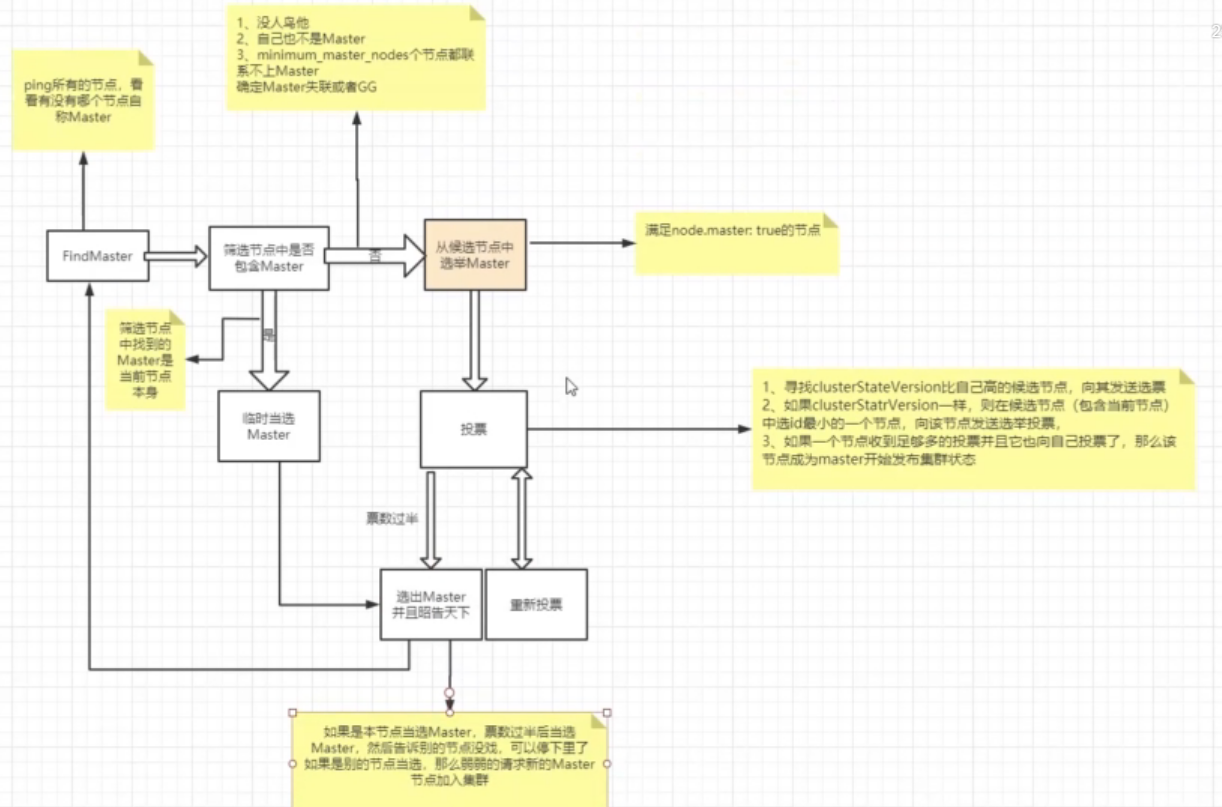
}

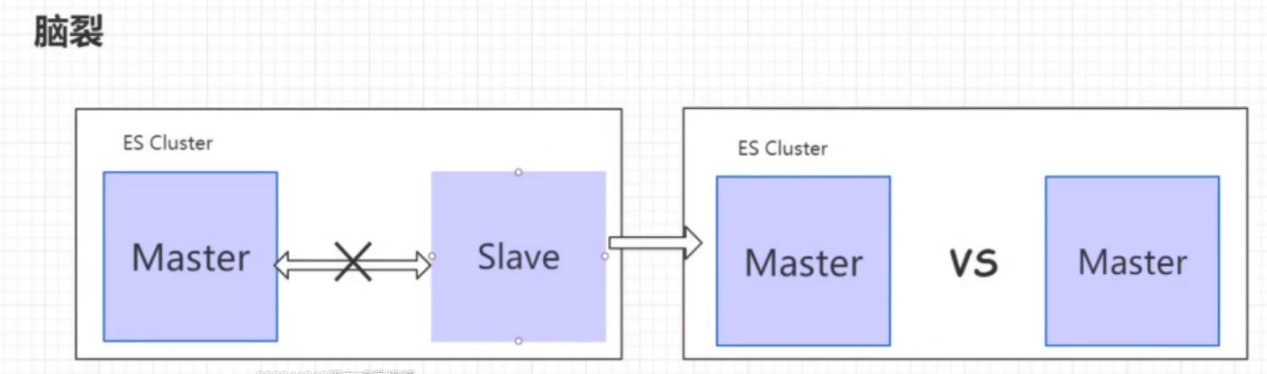
* 1. 删除数据（根据id） DELETE /product/\_doc/1

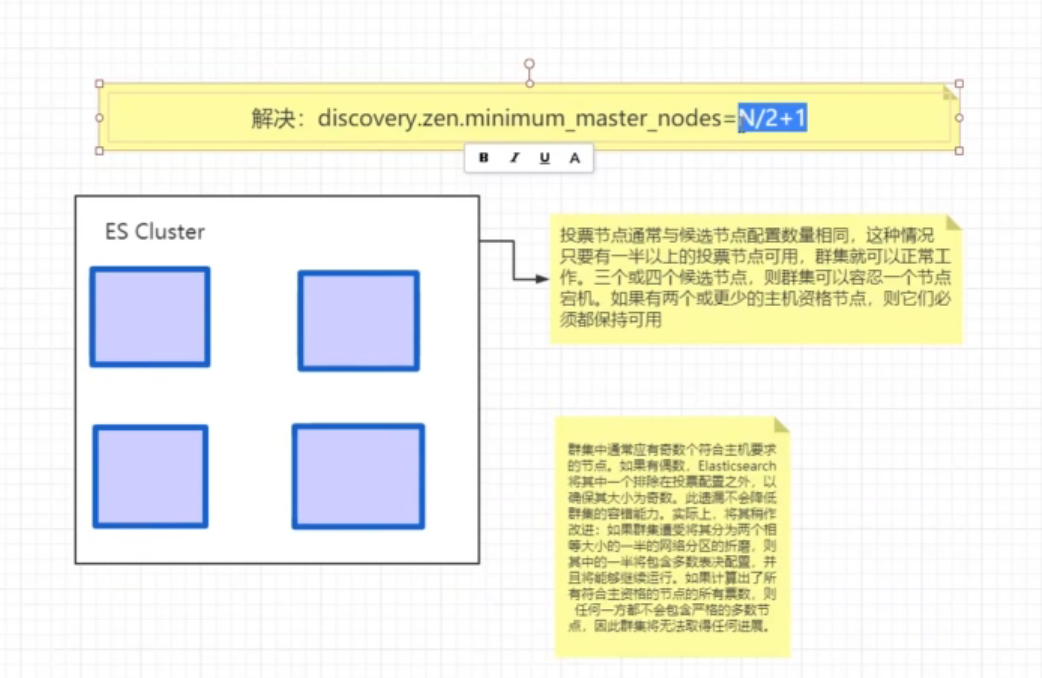


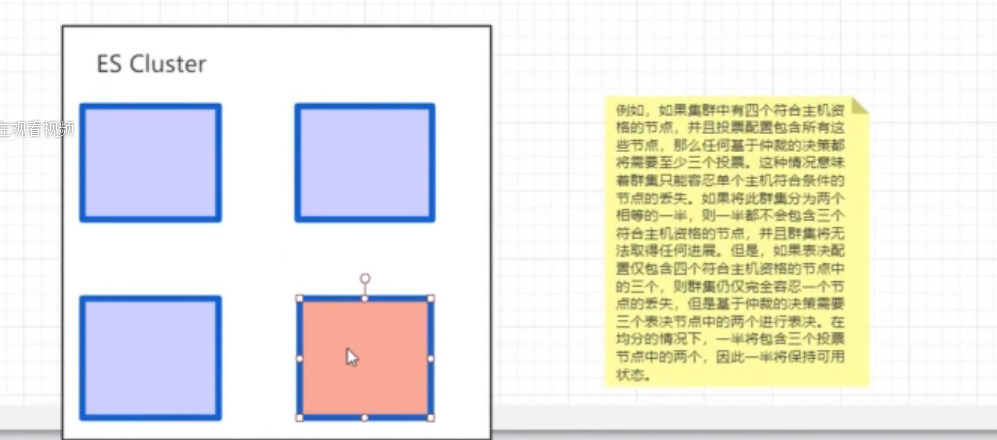












1. ES如何实现高可用（生产环境均为一台机器一个节点）
2. es在分配单个索引的分片时会将每个分片尽快可能分配到更多的节点上。但是，实际情况取决于集群有用的分片和索引的数量以及他们的大小，不一定总能均匀分布。
3. ES不允许primary和他的replica放在同一个节点中，并且同一个节点不接受完全相同的两个replica。
4. 同一个节点允许多个索引的分片同时存在。
5. 容错机制
6. 什么是容错
7. ES-node

<1>、

①master：主节点，每个集群都有且只有一个

1. 尽量避免master节点node.data = true

②voting：投票节点

1. node.voting\_only = true（仅投票节点，即使配置了data.master = true， 也不会参选，但是仍然可以作为数据节点）

③coordinating：协调节点

每一个节点都隐式的是一个协调节点，如果同时设置了node.master = false 和 node.data = false

④master-eligble node（候选节点）：

⑤data node（数据节点）：

⑥ingest node：

⑦machine learning node（机器学习节点）：

<2>两个配置 node.master 和 node.data

1. node.master = true node.data = true

默认配置。即作为候选节点又作为数据节点，这样的节点一旦被选举为master，压力比较大，通常来说master只承担较为轻量级的任务，比如创建删除索引、分片均衡等。

1. node.master = true node.data = false

只作为候选节点，不作为数据节点，可参选master节点，当选后成为真正的master节点。

1. node.master = false node.data = false

既不当候选节点，也不作为数据节点，那就是仅协调节点，负责负载均衡。

④ node.master = false node.data = true

不作为候选节点，但是作为数据节点。这样的节点主要负责数据存储和查询服务。

1. 图解容错机制

①第一步：master选举（加入宕机节点时master）

1. 脑裂：可能会产生多个master节点
2. 解决：discovery.zen.minimum)master\_nodes=N/2+1

②第二步：

replica容错，新的（或者原有的）master节点会将丢失的primary对应的某个副本提升为primary

③第三步：master节点会尝试重启故障机

④第四步：

数据同步，master会将宕机期间丢失的数据同步到重启机器对应的分片上去

1. 总结（如何提高es分布式系统的可用性以及性能最大化）
2. 每台节点的shard数量越少，每个shard分配的cpu、内存和io资源越多，单个shard的性能越好，当一台机器一个shard时，单个shard性能最好。
3. 稳定的master节点对于集群健康非常重要！理论上讲，应该尽可能的减轻master节点的压力，分片数量越多，master节点维护管理shard的任务越重，并且节点可能就要承担更多的数据转发任务，可增加“仅协调”节点来缓解master节点和data节点的压力，但是在集群中添加过多的协调节点会增加整个集群的负担，因为选择的主节点必须等待每个节点的集群状态更新确认。
4. 反过来说，如果相同资源分配相同的前提下，shard数量越少，单个shard的体积越大，查询行能越低，速度越慢，这个取舍前应该根据实际集群状况和结合应用场景等综合因素考虑。
5. 数据节点和master节点一定要分开，集群规模越大，这样做的意义也就越大。
6. 数据节点处理与数据相关的操作，例如crud，搜索和聚合。这些操作是i/o，内存和cpu密集型的，所以他们需要更高的配置的服务器以及更高的带宽，并且集群的性能冗余非常重要。
7. 由于仅投票节点不参与master竞选，所以和真的master节点相比，他需要的内存和cpu较少。但是，所有候选节点以及仅投票节点都可能是数据节点，所以它们都需要快速稳定低延迟的网络。
8. 高可用性（HA）集群至少需要三个主节点，其中至少两个不是仅投票节点。即使其中一个节点发生故障，这样的集群也将能够选举一个主节点。生产环境最好设置3台仅master候选节点（node.master=true node.data=true）

常用的Query DSL

1：Query string search：类似于Url该参数搜索

//从product中查询，价格为2999，按照price升序

GET /product/\_search?q=price:2999&sort=price:asc

2：Query DSL：

1. Query and filter：
2. Phrase search（短语搜索）：必须全部包含搜多的短语
3. Full text queries（全文检索）：会将搜索词进行分词，包含任何一个分词，都将作为匹配结果，按照匹配分倒序输出
4. Highlight search（高亮搜索）：

## （176）上机实战演练：ES查询语法

Es语法查询

1、Searchtimeout:

（1）设置：默认没有timeout，如果设置了timeout，那么会执行timeout机制。

（2）Timeout机制：假设用户查询结果有1w条数据，但是需要10s才能查询完毕，但是用户设置了1s的timeout，那么不管当前一共查询到了多少数据，都会在1s后es将停止查询，并返回当前数据。

2、es常用查询：

（1）Query\_string:

①查询所有：GET /product/\_search

②带参数： GET /product/\_doc/\_search?q=name:xiaomi

③分页：GET /product/\_search?from=0&size=2&sort=price:asc

（2）**Query DSL:**

①match\_all：

GET /product/\_search

{

“query”:{

“match\_all”:{}

}

}

②match：

GET /product/\_search

{

“query”:{

“match”:{

“name”:”nfc”

}

}

}

③sort：

GET /product/\_search

{

“query”:{

“match”:{

“name”:”nfc”

}

},

“sort”:[

{

“price”:”desc”

}

]

}

④multi match：

GET /product/\_search

{

“query”:{

“multi\_match”:{

“query”:”nfc”,

“fields”:[“name”,”desc”]

}

},

“sort”:[

{

“price”:”desc”

}

]

}

⑤\_source：

GET /product/\_search

{

“query”:{

“match\_all”:{}

},

“\_source”:[“name”,”price”]

}

⑥分页（deep-paging）:

GET /product/\_search

{

“query”:{

“match\_all”:{}

},

“sort”:[

{

“price”:asc

}

],

“from”:0,

“size”:2

}

（3）full-text queries：全文检索

①query-term：不会被分词

GET /product/\_search

{

“query”:{

“term”:{

“name”:”nfc phone”

}

}

}

②match和term：

GET /product/\_search

{

“query”:{

“bool”:{

“must”:[

{“term”:{“name”:”nfc”}},

{“term”:{“name”:”phone”}}

]

}

}

}

或者用以下写法：

GET /product/\_search

{

“query”:{

“terms”:{

“name”:[”nfc”,”phone”]

}

}

}

GET /product/\_search

{

“query”:{

“match”:{

“name”:”nfc phone”

}

}

}

③※**全文检索**：

1. phrase search：短语搜索（包含整个短语）

GET /product/\_search

{

“query”:{

“match\_phrase”:{

“name”:”nfc phone”

}

}

}

1. Query and filter：查询和过滤

①bool：可以组合多个查询条件，bool查询也是采用more\_matches\_is\_better的机制，因此满足must和should子句的文档将会合并起来计算分值。

1. must：必须满足
2. filter：过滤器 不计算相关度分数 cache ※
3. should：可能满足
4. must not：必须不满足 不计算相关度分数

②案例：

1. demo案例

GET /product/\_search

{

“query”:{

“bool“:{

“must”:[

{“match”:{“name”:”xiaomi”}},

{“match”:{“desc”:”shouji”}}

]

“filter”:[

{“match\_phrase”:{“name”:”xiaomi phone”}},

{“range”:{

“price”:{

“gt”:1999

}

}}

]

}

}

}

1. bool多条件name包含xiaomi不包含erji描述里不包含nfc都可以，价钱要大于等于4999

③嵌套查询：