

Query String +request body

----request body

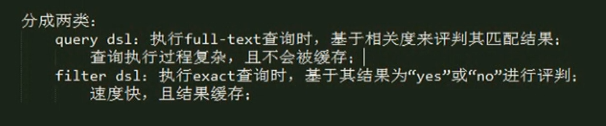


查询和过滤使用非常相近

执行full-text的时候 评判文档和查询条件的相关度

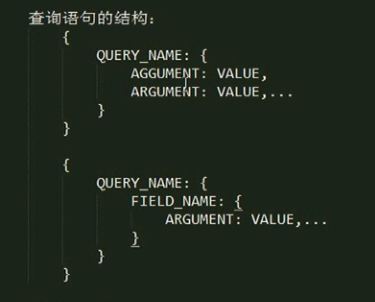
\*\*根据查询条件的相关度匹配 就用query dsl

\*\*做精确值查询 用filter dsl



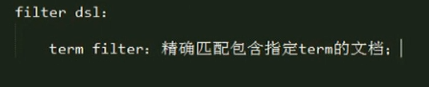
得益于倒排索引 query dsl也是接近实时

查询语句的内部

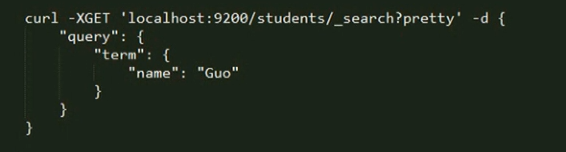


-------------------------- filter dsl ------------------

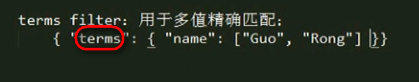
Term filter ----过滤包含指定term中的文档



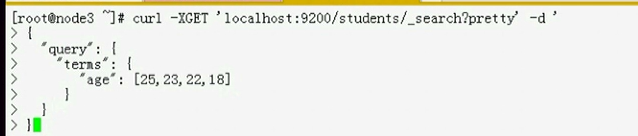
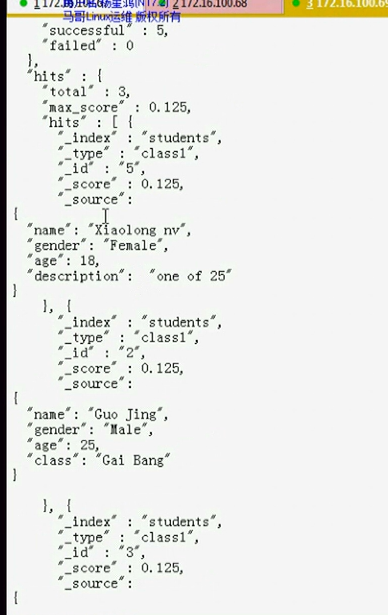
{“term”:{“name”:”Guo”}} 用term filter



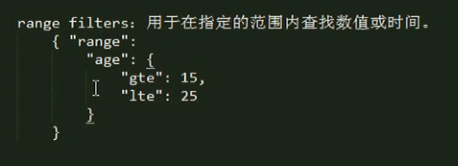
还有个是terms filter ---用于实现多值精确匹配



这样就是匹配 name字段有Guo或者Rong的 doc 这就是多值匹配 term是单值匹配

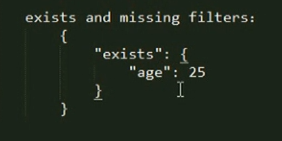
----- range filter 范围匹配 用于在指定的范围内 查找数值 和 时间 只能查找数值 和时间



Gt> Lt < gte >= lte <=

---- exists 和missing

Exists判断指定的字段是否有这值 missing相反 exists的取反



---- boolean filter

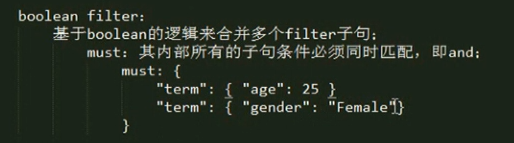
**基于bool的逻辑 来合并多个filter子句的**

**年龄在15-25中 并且 name中含有Guo的**

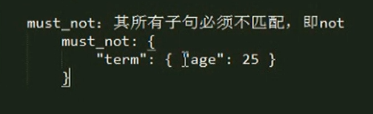
----- 写法诡异 ----

Must 表示所有的**子句必须同时匹配 就是and意义**

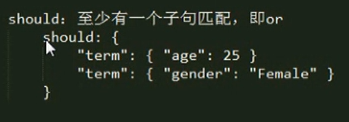
Must就是外面的Boolean filter 里面还是其他的filter

 表示 年龄是25并且性别是女的

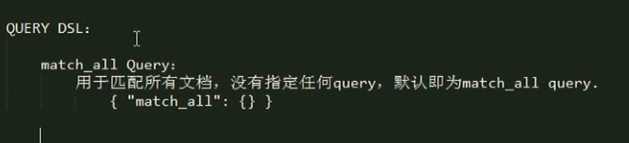
Must not ---- 所有Filter子句必须不匹配 ---- 即 not

 年龄必须不能是25

Should ---- 至少一个子句匹配 即or



下面是Query DSL 上面是filter DSL



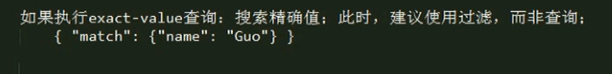
Match query 在几乎任何域上执行Full-text或者exact-value查询



如果执行full-text查询 首先对查询的语句进行分析 然后根据匹配读进行查询 如果执行exact-valu查询 不会做分析 搜索精确值 ----- 此时建议使用过滤 而非查询 因为过滤可以缓存 但是Query不可以

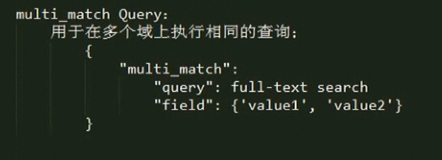


表示在students这个索引库中的任何位置搜索Guo这个字符串



这个就是指定域的精确值查询 必须是Guo 不能是别的 ---不做切词 也不做倒排索引查询

还有一个multi\_match Query 用在多个域上执行相同的查询

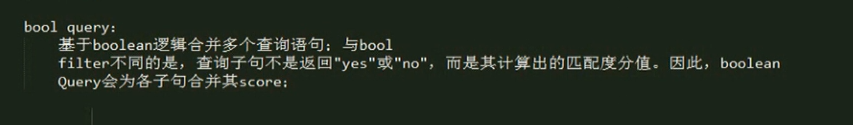


如果在name和description上搜索Guo的字符串 该怎么写？

 这个Query表示要在哪个索引库查询什么内容 然后 field指定这个索引库的哪些字段上查询目标关键字

--- bool query 基于bool逻辑 合并多个查询语句 而不是在是合并多个过滤语句

查询子句不是返回yes或者no 而是计算出来的匹配度分值 --- bool query会为各个子句合并其Score

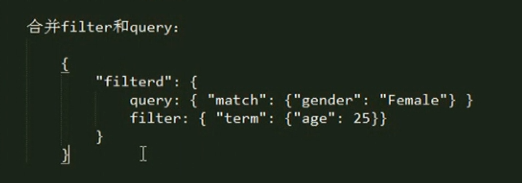


也是must must\_not和should

===== filter和query进行合并 =====

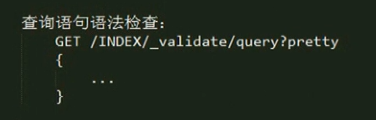
Query是查询 filter是进行过滤

这样把filter中嵌入到Query 但是反过来很少这样做

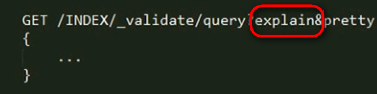
 这样query和filter合并起来

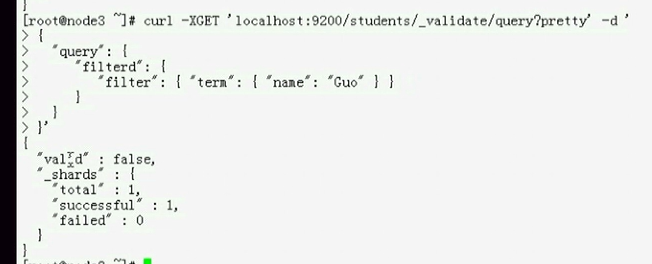
---- 查询语句语法进行检查

GET /INDEX/\_validate/query?pretty –d {语句} ---- 这样可以在对应的index上面检查Query的语法是否正确

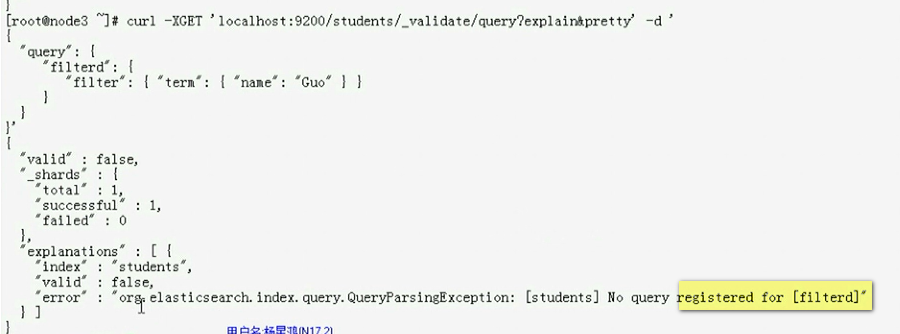


如果想知道查询错误错在哪

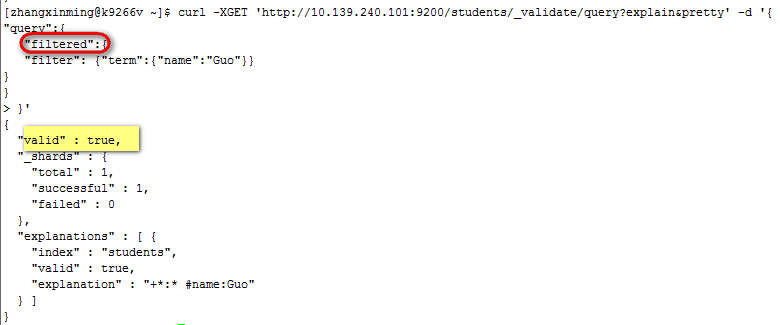


 valid:false 说明语句写错了

哪里写错了呢

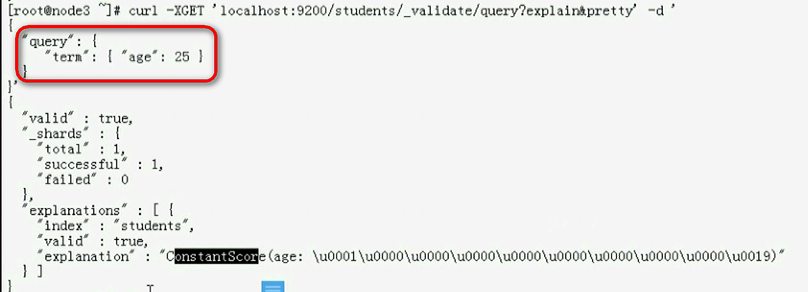


【老师没有说对 其实就是filterd这个单次写错了 应该是filtered



】

【filtered用于query和filter组合使用】



--------- ES Stack中的另外两个组件

L: logstash K Kinbana

Logstash --- 能够向产生日志的服务器部署一个agent 搜集日志 并通过一个统一的管道 传输给ElasticSearch的组件 并存储起来

索引组件和数据有两种联系方式

---- 爬虫 出去pull数据 --- 主动填充数据

也可以等待数据push进来 ---- logstash ---- 被动获取数据

有一堆web服务器 在每一台的web服务器上面部署一个logstash agent 监控某一个日志文件 或者某一堆日志文件 然后搜集对应的日志数据 搜集过来以后 由logstash 统一写到elasticsearch集群中 ----- 并把数据存储到elasticsearch

所以 logstash可以把日志数据存储到elasticsearch

是server-agent结构 ---- 每一个日志文件中的产生的信息 搜集到logstash server ---- 按照时间序列 排序 写入到elasticsearch

====== es是**通用搜索引擎** 更多和logstash组合 作为日志搜索引擎 也可以和爬虫一起

有了elasticsearch存储了 logstash的数据以后 就要进行检索了 但是 elasticsearch的检索语句太复杂 所以 可以借助kibana这个进行检索

Kibana可以对数据进行趋势展示 ---- Kibana是基于nodejs做的

Es是独立完整的 只不过Kibana为es提供了更加友好的搜索和展示界面 ---- 易用的接口

Logstash非常重量级 logstash-agent 非常重量级 基于jruby的 性能相当不高

所以 每个节点部署一个logstash-agent部署之后 不发给logstash server 而是把数据发送给消息队列 然后消息队列发送到elasticsearch中

这样 自己写agent 自己写server程序 ------ 用到的MessageQueue 就是Kafka 性能表现非常高级 非常强大

数据收集器 有很多种 可以自己写 就是监控一个文件的新增数据

================= logstash和Kibana的应用 ====================

这俩都属于elasticsearch

早起logstash是一个独立的开源组织维护

\*\*如何使用logstash

日志搜集的困难性：搜索分析

Grep awk对海量数据分析 是一个小蜗牛拉大车

但是 syslog 面对海量数据也无法完成

Logstash可以 但是 重量级

能够实现日志分析搜集 存储 ---- 自己的索引构建组件和es不是一个数量级的 所以 最常用logstash的就是日志搜集 logstash的索引不用

Logstash支持多种数据获取机制 --- 通过TCP/UDP协议 文件 syslog甚至windows自己的Event logs 以及标准输入输出 STDIN等 这么多种数据获取机制

也支持很丰富的数据输出机制

获取之后 可以对数据执行过滤 修改等操作

因此 数据过滤完之后 发送给elasticsearch

Logstash是基于jruby上研发的 需要jvm 工作于agent-server模型

Logstash工作机制

在每一个产生日志的节点 部署一个agent 被称作shipper 负责搜集日志 并发送给server ----- 发送给服务器的时候 有一个问题 就是logstash的server未必能够处理过来这么多的web服务器的agent发送过来的日志写入请求 ----- 可能压垮

为了避免这种情况 会把agent和server之前 放一个消息队列 进行缓冲

【我们的项目 就是日志一部分落地 另一部分直接通过openresty打到kafka 节省了agent】

---- 这个消息队列 更常见 更易于部署的是redis ---redis也支持发布订阅功能

所以 logstash服务器端从消息队列redis上一次拿一个 一次拿一个 --- 然后logstash对获取到的日志进行过滤修改 然后 再发送给es集群

这个redis叫broker

各个agent发送给broker server从Broker中把数据读出来 进行清洗 然后发给elasticsearch进行存储

这个是logstash的基本工作模式 ---- logstash的server也是插件的工作模式

Logstash必须使用插件 但是es的插件可有可无

Logstash的插件的种类

\*\*输入插件 从何处获取数据 ----- input plugin

--- 把数据搜集进来

\*\*\*\*输入输出的时候 都可以做编码 codec插件 --- codec plugin

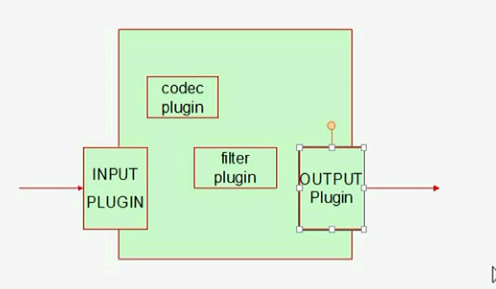
一进来 就进行编码 codec

然后进行过滤 filter plugin ---- 比如多日期 时间过滤

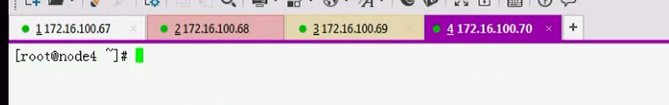
过滤完之后 可以发给其他的地方存储 ---用到输出插件

\*\*输出插件

可以放到es 文件 或者 其他

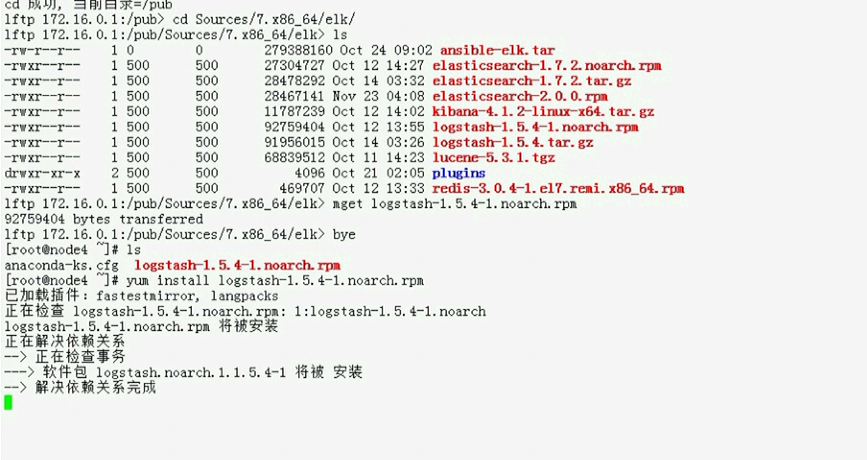


Logstash支持数十种输入 输出 和 过滤



Logstash运行 需要jvm

下载logstash



从ftp下载之后 然后yum安装

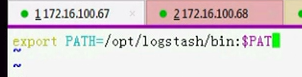
---------------- 先单机模式运行

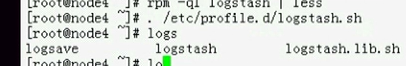
要想使用logstash



安装到opt下面了







这样就可以直接运行Logstash了



配置文件在/etc下面

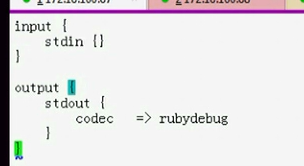


Conf.d下面的.conf都会当作配置文件

创建一个sample.conf

对logstash进行配置 就是定义从哪得到数据 然后输出到哪里去

现在Input通过标准输入来接收数据   
输出到标准输出 使用codec



这个是标准写法 codec=>rubydebug

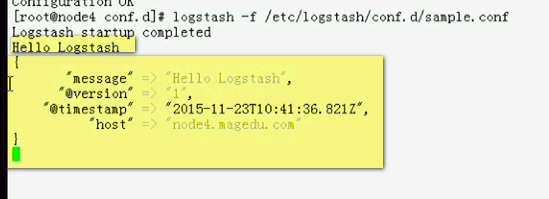
启动Logstash



这个样 –configtest测试配置的原发错误



 有输出是Logstash startup completed就是表示logstash已经启动完成

 上面的是输入 下面的是输出

Host表示事件发生的主机

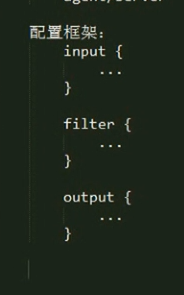
@timestamp表示事件发生时候的时间戳

@version是版本号

Message是信息的完整内容

Logstash就是在stdin 和stdout之间流动的

Logstash的配置框架 就是

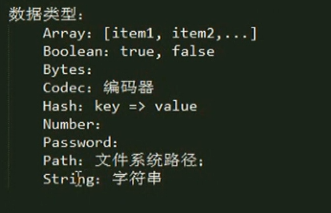
 logstash支持四种类型的插件 input filter codec 和output

不是每一种类型 会定义在配置文件中

  
支持的数据类型

Codec是一种数据类型

PATH表示文件系统路径

 这个是logstash支持的数据类型

字段引用



条件判断也支持

=~是正则匹配 xor是异或

