Elasticsearch

Trie树，即单词查找树或键树，是一种哈希树形结构的数据结构。常用于统计和排序大量的字符串内容，因此常被现在的搜索引擎系统用于文本词频率的统计。

例如：求（字典序中的）两个单词的公共前缀长度值，对trie树来说，只需要求两个单词对应结点的最近公共祖先即可。

保证索引库中与数据库中的数据一致(只针对要进行搜索的数据)。采用的方法是，在数据库中做了相应的操作后，在索引库中也做相应的操作。具体的索引库操作，是通过调用相应的IndexDao方法完成的。IndexDao类似于数据库层的Dao(Data Access Object) 数据访问对象。

在网络设计上，1ibevent对于超大规模的并发访问(10K及以上连接)性能有明显下降，因此，找到替换libevent的更优秀策略是下一步需要解决的问题。

libevent是一个典型的reactor模式的实现。这里做一下简单介绍：

我们知道，普通的函数调用机制如下：程序调用某个函数，函数执行，程序等待，函数将结果返回给调用程序（如果含有函数返回值的话），也就是顺序执行的。

而Reactor模式的基本流程如下：应用程序需要提供相应的接口并且注册到reactor反应器上，如果相应的事件发生的话，那么reactor将自动调用相应的注册的接口函数（类似于回调函数）通知你，所以libevent是事件触发的网络库。

1、存储数据时按有序存储；

2、将数据和索引分离；

3、压缩数据；

这就引出了Elasticsearch。

怎么看到的都是索引文档的介绍？关于json数据呢？

性能esrally工具。

Jmeter？

一台物理机上可以部署两个节点吗？

近年ElasticSearch发展迅猛，已经超越了其最初的纯搜索引擎的角色，现在已经增加了数据聚合分析（aggregation）和可视化的特性，如果你有数百万的文档需要通过关键词进行定位时，ElasticSearch肯定是最佳选择。当然，如果你的文档是JSON的，你也可以把ElasticSearch当作一种“NoSQL数据库”， 应用ElasticSearch数据聚合分析（aggregation）的特性，针对数据进行多维度的分析。

可以看一下es如何解析文本文档和json文档的？它们的不同在什么地方？

Lucene怎么索引json格式的文档？

《Elasticsearch in action》Having objects as field values

If you’re familiar with Lucene, you may ask yourself, “How can Elasticsearch documents be hierarchical when Lucene supports only flat structures?” With objects, Elasticsearch flattens hierarchies internally by putting each inner field with its full path as a separate field in Lucene.

**基于 ElasticSearch 的数字图书馆检索系统**

关于elasticsearch.conf和yml文件的配置。否则会严重影响es的性能。Heap size 、cache、limit

用MultiFieldQueryParser 生成 Query 对象

创建 IndexSearcher 实例,进行检索，结果保存到SearchReponse 结果集中。

通过 ObjectMapper 将JSON 数据直接转换成 List ( hashMap ) 样式返回到结果列表中。

要不要把这篇文章也看一看？

基于 Lucene 的数字版权资源库的构建与应用研究

**ElasticSearch分布式搜索引擎在天文大数据检索中的应用研究**

ElasticSearch拥有的River机制使得用户可以并行地将大量数据快速同步到集群.

你可以发现有了这些plugin，如果只是做一个数据的搬运，就可以不用logstash了。当然如果你还需要对数据做一些处理(比如logstash中的filter的那些功能)，那你还是用logstash吧，因为我看了一下，这些plugin没有提供数据处理功能。

因为他们都实现了es中的org.elasticsearch.plugins.AbstractPlugin接口。

**基于ElasticSearch的分布式视频垂直搜索引擎的设计与实现**

使用了缓存策略。

我们使用curl，当然你也可以使用任何可以创建HTTP/REST调用的工具，来使用该功能。

文档由多个字段组成，每个字段可能多次出现在一个文档里，这样的字段叫多值字

段（multivalued）

从客户端的角度看，文档是一个 JSON 对象。

当然，需要记住一个限制，不同的文档类型不能为相同的属性设置不同的类型。例如，在同一索引中的所有文档类型中，一个叫 title 的字段必须具有相同的类型。

每一个文档类型都有自己的映射，即使我们没有明确定义。

其中每个分片都是一个独立的 Apache Lucene 索引。

curl是利用URL语法在命令行方式下工作的开源文件传输工具； curl 是一个强大的命令行HTTP客户端，可以用来向目标服务器发送请求。所以可以使用cURL代替Web浏览器跟Web服务交互。

好好看看es的配置文件！

curl: (6) Could not resolve host: '10.222.175.32原来windows命令行中不需要打引号！

curl 10.222.175.32:9200/\_cat/indices?v 加上v可以显示表头

pretty=true表示格式化输出

创建索引customer

curl -XPUT 10.222.175.32:9200/customer?pretty

curl -X<REST Verb> <Node>:<Port>/<Index>/<Type>/<ID>

在索引的时候，ID部分是可选的。如果不指定，Elasticsearch将产生一个随机的ID来索引这个文档。这时候用post而不是put

但要注意，Elasticsearch底层并不支持原地更新。在我们想要做一次更新的时候，Elasticsearch先删除旧文档，然后再索引更新的新文档。这只是底层的，命令行不变，依旧是一句post，id后面跟\_update。

根据实验来看，如果有了具体的id，然后用post指令，直接改变字段的值，是可以成功的。而且version加1.

\_bulk API实现。这个功能之所以重要， 是因为它提供了非常高效的机制来尽可能快的完成多个操作，与此同时使用尽可能少的网络往返。

带参数**创建索引**：

curl -XPUT [address]/blog -d’{

//可以自定义索引分析器的！

}’

**创建映射（扁平结构）：默认是自动mapping的**

curl -XPUT [address]/blog/\_mapping/article?pretty -d '{

         "properties":{

                   "id":{"type":"long"},

                   "name":{"type":"string"},

                   "published":{"type":"date"}

         }

}'

**带参数创建映射：**

[**Elasticsearch创建索引和映射结构详解**](http://www.cnblogs.com/sheeva/p/4837881.html)

 "precision\_step":"0"       //指定为该字段生成的词条数，值越低，产生的词条数越多，查询会更快，但索引会更大。默认4

每一只能够类型可以用参数定义自己的映射机制，包括索引的一些特性和每一个property的一些特性。"analyzer":"xxx",     //定义用于索引和搜索的分析器名称，默认为全局定义的分析器名称。可以开箱即用的分析器:

standard,simple,whitespace,stop,keyword,pattern,language,

snowball

mapping的定义中有些是公共属性，像“store”，“index”，“boost”等，有些是数值特有属性，“precision\_step”，有些是字符串特有属性“analyzer”，"index\_analyzer"，"search\_analyzer"等，还有日期特有属性："precision\_step"，"format"。

**创建映射（非扁平结构）**

1.树形结构：没有特别看懂！通过定义一个path\_analyzer

2.对象：字段的mapping定义中“type”:”object”

3.嵌套对象：嵌套对象允许我们连接一个主文档和多个附属文档

 “type”:”nested”, 这其中为什么直接存储还无法进行and的查询？

Filter当中的path是什么？

# [elasticsearch简单查询](http://www.cnblogs.com/sheeva/p/4853524.html)

从jdbc中导入数据，肯定是只能导入扁平结构的数据。

比较存储空间的时候，其实比较的是单副本占用的空间和mysql占用空间.

与mysql数据库查询相对比，可以发现Elasticsearch在聚合查询和非主键排序上效果提升明显

每个bulk索引请求提交的长度 默认1000

一个bulk请求的最大容量 默认10M

XPUT和XPOST、XGET的区别

在river机制中，如果没有指定\_id的话，那么会随机生成。

在curl命令中如果是不指定id去put json呢？注意，在上面的情形中，由于我们没有指定一个ID，我们使用的是POST而不是PUT

curl -XPOST 'localhost:9200/customer/external?pretty' -d '

后者查询时是不指定类型。如果要指定类型呢？

curl -XPOST 'localhost:9200/bank/\_search?pretty' -d '

除了query，还有size、from、sort、\_source、aggs

query中，有match\_all查询、match查询、match\_phrase，但是都是只有一个条件

布尔查询bool。是针对每一个文档。

我们把json的格式是不是搞错了！

Elasticsearch中的所有的查询都会触发相关度得分的计算。对于那些我们不需要相关度得分的场景下，Elasticsearch以过滤器的形式提供了另一种查询功能。

通过一个区间的值来过滤文档。这通常被用在数字和日期的过滤上。用filtered。

被过滤的查询包含一个match\_all查询（查询部分）和一个过滤器（filter部分）。

{

"query": {

"filtered": {

"query": { "match\_all": {} },

"filter": {

"range": {

"balance": {

"gte": 20000,

"lte": 30000

}

}

}

}

}

}

查询部分中可以放入其他查询，在filter部分放入其它过滤器。

通常情况下，要决定是使用过滤器还是使用查询，你就需要问自己是否需要相关度得分。

一般情况下过滤器比查询用得更多。

除了match\_all，multi\_match, match, bool，filtered 和rang查询，还有很多其它类型的查询/过滤器，我们这里不会涉及。

聚合查询可以一次性返回命中的文档和聚合结果。

注意我们将size设置成0，这样我们就可以只看到聚合结果了，而不会显示命中的结果。

注意，我们把average\_balance聚合嵌套在了group\_by\_state聚合之中。这是所有聚合的一个常用模式。你可以在任意的聚合之中嵌套聚合，这样就可以从你的数据中抽取出想要的结果。

curl -XPOST 10.222.175.32:9200/bank/\_search?pretty --data {"query":{"match\_all":{}}}

成功了！看来是少写了一个小括号导致的！哈哈哈！

但是为什么其它的不可以。

父子文档的mapping

curl-XPOST "http://192.168.0.224:9200/company/branch/\_bulk?pretty"

--data-binary @company.json

binary在这里是什么作用？

可以bulk创建父子关系。子类需要制定父类文档的\_id.

可以通过子文档查询父文档，也可以通过父文档查询子文档。

一个子文档不能够有多个父文档，但是一个父文档可以有多个子文档。

所以子文档对副文档，肯定是多对1的关系。

嵌套对象的mapping

Mapping的写入、查看与修改：

curl -XGET 10.222.175.32:9200/bank/\_mapping?pretty

{

"bank" : {

"mappings" : {

"account" : {

"properties" : {

"account\_number" : {

"type" : "long"

},

"address" : {

"type" : "string"

},

"age" : {

"type" : "long"

},

…

}

}

}

}

}

创建新索引的时候，mappins里面为空。

curl-XPOST "http://127.0.0.1:9200/productindex/product/\_mapping?pretty" -d

这是添加type并设置mapping

es支持的数据类型：string double data boolean integer？如果是数组类型呢？

为什么不能修改一个字段的type？原因是一个字段的类型修改以后，那么该字段的所有数据都需要重新索引。Elasticsearch底层使用的是lucene库，字段类型修改以后索引和搜索要涉及分词方式等操作，不允许修改类型在我看来是符合lucene机制的。

elasticsearch有各种core-type，另外还有各种复杂结构，比如nested，child-parent等。

在es中array是默认支持的，并不存在单独的一个type=array。

这个是不是需要试验一下的？

partital-update的方法，用setDoc+upsert来实现？用update来实现？？？

看原来那篇ifc的论文是怎么将set、list等数值类型进行存储的。

特点优势  
    （1）Open Source（开源）   
    （2）Apache Lucene（基于 Lucene）   
    （3）Schema Free(模式自由)   
    （4）Document Oriented(面向文档型的设计)   
    （5）Real Time Data & Analytics（实时索引数据）   
    （6）Distributed（分布式）   
    （7）High Availability（高可靠性）   
    （8）其他特性：RESTful API；JSON format；multi-tenancy；full text search；conflict management；per-operation persistence

想搜：

{

"\_index" : "xxx",

"\_type" : "xxx",

"\_id" : "123",

"\_version" : 1,

"\_score" : 1,

"\_source" : {

"iD" : 123,

"name" : "xxx",

"tags" : [

"文字",

"恋爱",

"冒险"

],

"keywords" : []

}

}

但出现了：

{

"\_index" : "xxx",

"\_type" : "xxx",

"\_id" : "124",

"\_version" : 1,

"\_score" : 1,

"\_source" : {

"iD" : 123,

"name" : "xxx",

"tags" : [

"A 文字 A",

"恋爱",

"冒险"

],

"keywords" : []

}

}

用 term 搜索。另外 mappings 需要把 index 改成 not\_analyzed 。因为默认是 standard analyzer ，所以 stop words 就被隔开了。

中文分词：对于索引可能最关系的就是分词了 一般对于es 来说默认的smartcn  但效果不是很好。<http://www.cnblogs.com/zhangchenliang/p/4186702.html>

这里有如何进行分词器配置。

先进行分词器插件的安装，然后就可以在mapping中定义分词器的使用。Indexanalyzer、searchanalyzer的配置等等。

测试分词（应该是分词器吧）可用调用下面api，注意indexname为索引名，随便指定一个索引就行了

http://localhost:9200/indexname/\_analyze?analyzer=ik&text=测试elasticsearch分词器

client连接封闭到一个连接池当中。

上面这个连接居然有比较完整地介绍了es底层的lucene是怎么将文本文件进行索引的。

Mapping的作用域也是从cluster、node、index、type。

经过测试，得出以下结论：

1. es可以存储数组、字符数组、整型数组、嵌套对象、对象数组。
2. 为什么值的类型不同但是却可以正确存储而且还能被检索？是不是因为没有固定mapping的缘故？XGET mapping，是第一次创建的时候的类型。
3. 为什么可以在curl客户端查询，但是head无法进行dept索引的查询（bank可以）？

"name":"yingdi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":"shuxue","mark":92.0}]

}

} ]

}

}

Diak@Dia ~

$ curl -XGET 192.168.31.187:9200/dept/\_search?pretty -d '

{"query":{"match":{"address.city":"lushan"}}}'

{

"took" : 3,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 0,

"max\_score" : null,

"hits" : [ ]

}

}

Diak@Dia ~

$ curl -XGET 192.168.31.187:9200/dept/\_search?pretty -d '

{"query":{"match":{"address.county":"lushan"}}}'

{

"took" : 2,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 3,

"max\_score" : 0.5945348,

"hits" : [ {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "2",

"\_score" : 0.5945348,

"\_source":

{

"name":"yadi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":"shuxue","score":92.0}]

}

}, {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "4",

"\_score" : 0.5945348,

"\_source":

{

"name":"yingdi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":10,"mark":92.0}]

}

}, {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "3",

"\_score" : 0.30685282,

"\_source":

{

"name":"yingdi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":"shuxue","mark":92.0}]

}

} ]

}

}

Diak@Dia ~

$ curl -XGET 192.168.31.187:9200/dept/\_search?pretty -d '

{"query":{"match\_all":{}}}'

{

"took" : 4,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 4,

"max\_score" : 1.0,

"hits" : [ {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "2",

"\_score" : 1.0,

"\_source":

{

"name":"yadi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":"shuxue","score":92.0}]

}

}, {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "4",

"\_score" : 1.0,

"\_source":

{

"name":"yingdi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":10,"mark":92.0}]

}

}, {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "1",

"\_score" : 1.0,

"\_source":

{

"name":"xiaokai",

"age":27,

"hobby":["sleep","read","music"],

luckyNumbers:[3,9,23],

"address":{

"city":"pingdingshan",

"county":"baofeng"}

,

"scores":[

{"subject":"yuwen","score":91},

{"subject":"shuxue","score":92}]

}

}, {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "3",

"\_score" : 1.0,

"\_source":

{

"name":"yingdi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":"shuxue","mark":92.0}]

}

} ]

}

}

Diak@Dia ~

$ curl -XGET 192.168.31.187:9200/dept/\_search?pretty -d '

{"query":{"match\_all":{}}}'

{

"took" : 2,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 4,

"max\_score" : 1.0,

"hits" : [ {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "2",

"\_score" : 1.0,

"\_source":

{

"name":"yadi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":"shuxue","score":92.0}]

}

}, {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "4",

"\_score" : 1.0,

"\_source":

{

"name":"yingdi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":10,"mark":92.0}]

}

}, {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "1",

"\_score" : 1.0,

"\_source":

{

"name":"xiaokai",

"age":27,

"hobby":["sleep","read","music"],

luckyNumbers:[3,9,23],

"address":{

"city":"pingdingshan",

"county":"baofeng"}

,

"scores":[

{"subject":"yuwen","score":91},

{"subject":"shuxue","score":92}]

}

}, {

"\_index" : "dept",

"\_type" : "employee",

"\_id" : "3",

"\_score" : 1.0,

"\_source":

{

"name":"yingdi",

"age":25,

"hobby":["eat","gym","music"],

luckyNumbers:[3,23],

"address":{

"city":"pingdingshan",

"county":"lushan"}

,

"scores":[

{"subject":"yuwen","score":91.0},

{"subject":"shuxue","mark":92.0}]

}

} ]

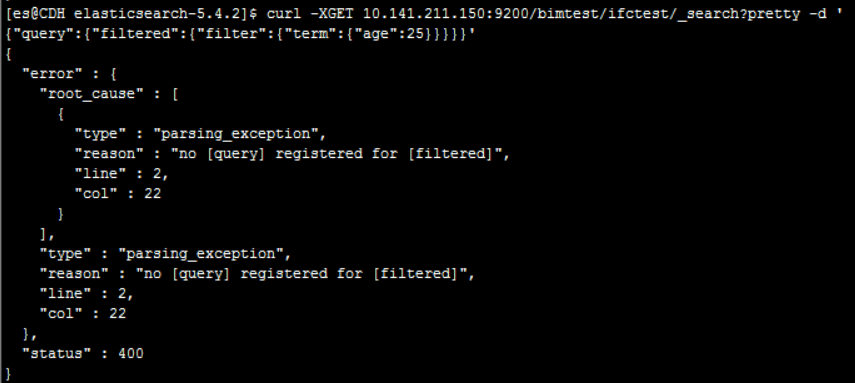
}

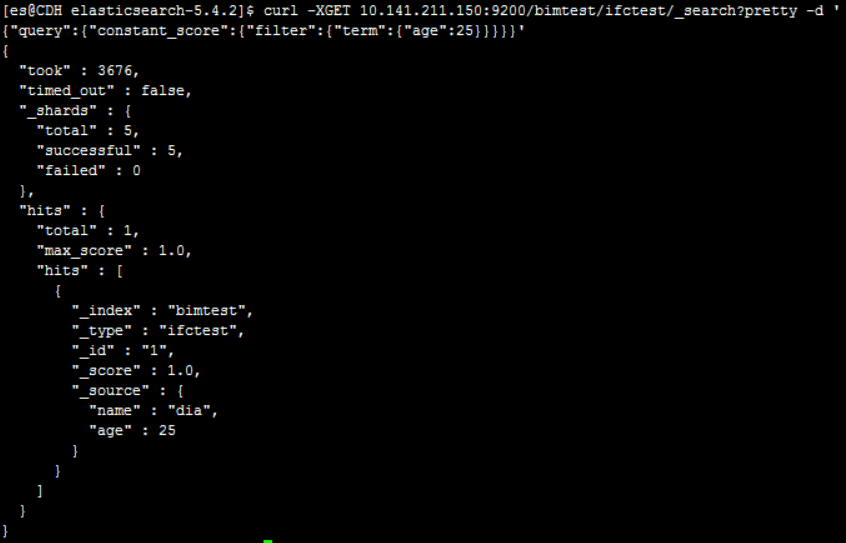
}

单台Web服务器的处理能力有限，当单台服务器承受的压力达到极限时，需要有更多的服务器分担工作，我们需要想办法将流量合理分配到更多的服务器上。

实现负载均衡有很多种方法，如HTTP重定向、基于DNS的轮询解析、反向代理服务器。无论用哪种方法，最终目的都是把流量分配到更多的服务器上，从而降低单台服务器的压力。







作用Linux系统中grep命令是一种强大的文本搜索工具,

PS是LINUX下最常用的也是非常强大的进程查看命令

ps -ef | grep elastic

kill -9 26500

或者是在head插件里面动作-关停（但是好像没有用）

es.yml配置文件中增加http最大传输量：http.max\_content\_length:1024m,重启时，设置内存为2G。原来5.4版本中config/jvm.options的默认配置了2g的堆内存。应该设置成其他的也可以！

重启时遇到最大文件描述符的问题，ulimit -n 65536

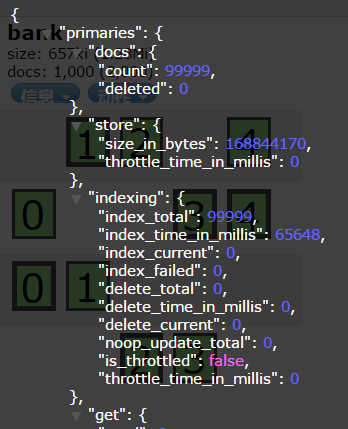
然后重新进行数据生成，只生成500m的。参数设置为一个区间，json数据值改为aaaaaaa。

root 1234567890

es 111111

传入10万行ifcwall，大小与json文件大小差不多。时间应该是：65,648ms。也就是一分钟10万行。





传入20万行ifcwall：

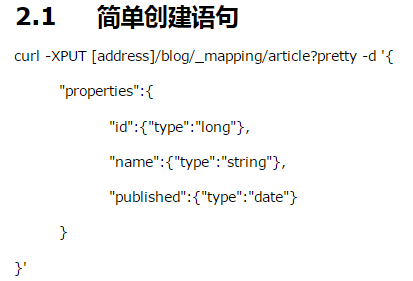




传入子类IfcPropertySet时出错，这时候应该先测试一下父子文档是否需要先进行mapping定义。



看来的确是需要进行mapping定义的。试一下。





这段话是什么意思？

关于新版本的字符串的映射： keyword，关键词搜索，not analyzed； text 全文搜索，analyzed。

es默认自动建立映射如下：



如果不需要同时这两种定义，那么按照下面这种显式mapping来做。



试验中我们测试不定义child\_name，只定义child\_age.这是定义了type的mapping。

建立文档的父子关系最重要的一步是在创建索引的时候在mapping中声明哪个是父文档哪个是子文档。

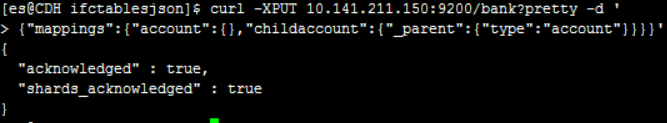
给出的例子：company是索引，branch是type，employee是type。然后employee的\_parent是branch。

第一遍输入错误，第二遍输入正确，但是提示index exists 错误。





所以要删除bank 这个index再来一遍。



正确执行。查看head端：一切正确。

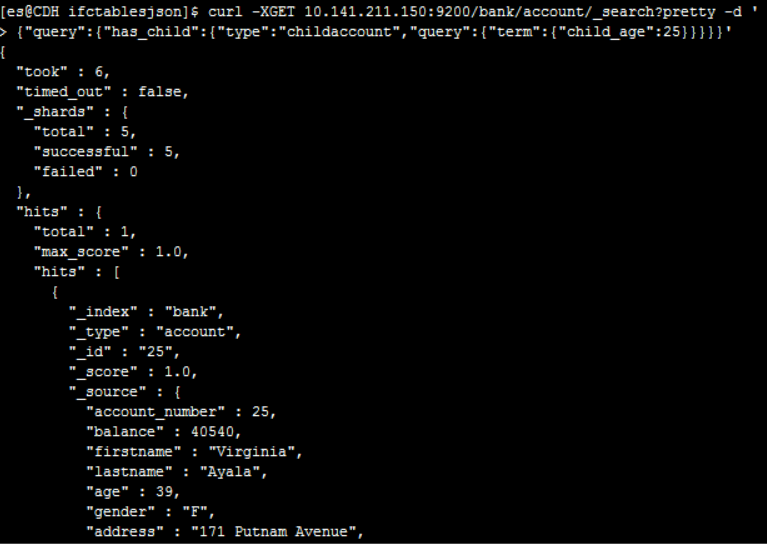
bulk传入account数据。bulk传入childaccount数据（文件形式）。

一切显示正常：

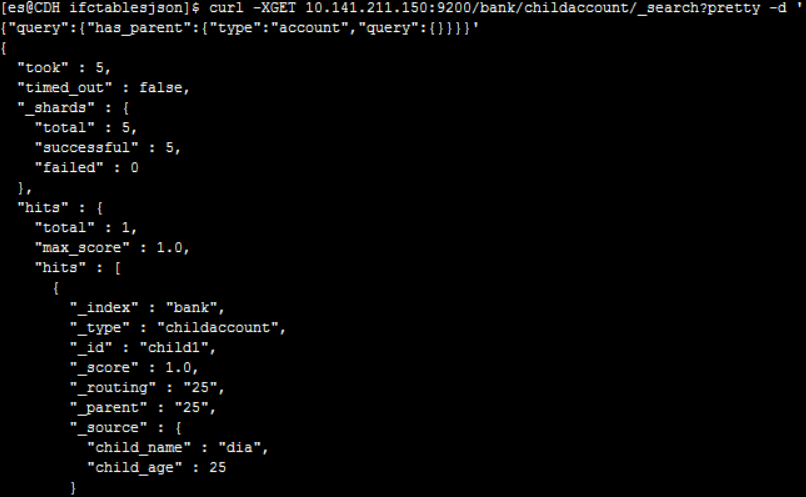


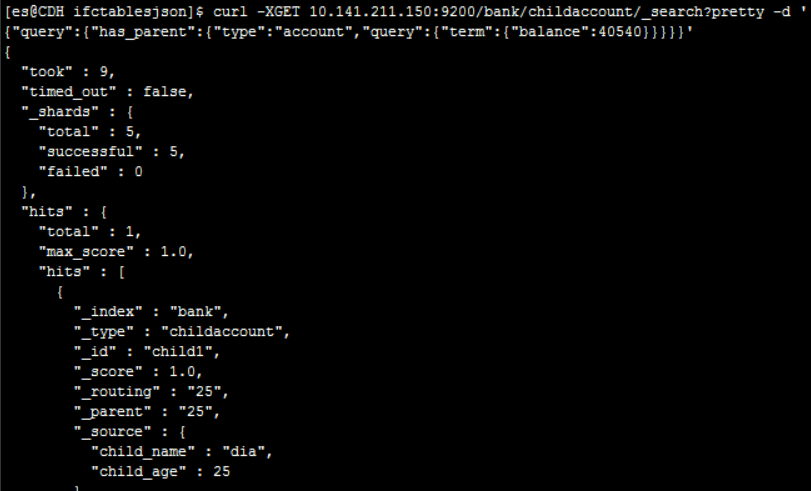
至此证明我之前传入的bimtest的数据没法儿用。

根据子文档条件，查询父文档：



根据父文档，查询子文档：





vi命令 x删除当前字符，dd 删除当前行

关于数据的生成还需要进一步更正：

一个ifcwall要有5个ifcpropertyset，一个ifcpropertyset要有10个if从propertysinglevalue

在mapping时就应该把所有的父子文档都定义下来。

先看看官网文档吧。

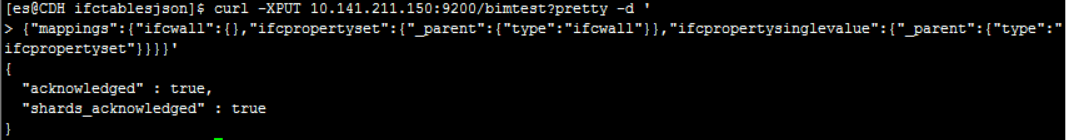
当然，数据非规范化也有弊端。 第一个缺点是索引会更大因为每个博客文章文档的 \_source 将会更大，并且这里有很多的索引字段。

grep（全面搜索正则表达式）

ps（process status）与top比较像

先生成所有的数据，全部上传到服务器上之后，再将数据上传到es上去。考虑要不要进行三层的父子文档。

重新建立index：



上传20万ifcwall之后：





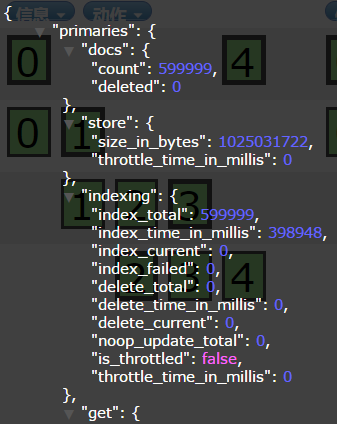
上传20万行ifcpropertyset之后：





上传20万上ifcpropertysinglevalue之后：





然后测试一下一级父子文档、二级父子文档的速度。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 一级父子文档 | | 二级父子文档 | |
|  | 查父 | 查子 | 查父 | 查子 |
| 20万行 | 37ms | 11ms |  |  |
| 40万行 | 13ms | 15ms |  |  |
| 60万行 | 15ms | 18ms |  |  |
| 80万行 |  |  |  |  |
| 100万行 |  |  |  |  |

一级查父语句：

curl -XGET 10.141.211.150:9200/bimtest/ifcwall/\_search?pretty -d ‘

{“query”:{“has\_child”:{“type”:”ifcpropertyset”,”query”:{“term”:{“attribute1”:”bbbbbbbb99”}}}}}’

一级查子语句：

curl -XGET 10.141.211.150:9200/bimtest/ifcpropertyset/\_search?pretty -d ‘

{"query":{"has\_parent":{"type":"ifcwall","query":{"term":{"attribute1":"aaaaaaaa101"}}}}}'

一级查子语句：

二级查子语句：

与 nested objects 相比，父-子关系的主要优势有：

1.更新父文档时，不会重新索引子文档？。那我想问，如果我只修改了一个字段的值时，其它字段需要重新索引吗？

2.创建，修改或删除子文档时，不会影响父文档或其他子文档。这一点在这种场景下尤其有用：子文档数量较多，并且子文档创建和修改的频率高时。

3.子文档可以作为搜索结果独立返回。

父-子文档ID映射存储在 [Doc Values](https://www.elastic.co/guide/cn/elasticsearch/guide/current/docvalues.html) 中。当映射完全在内存中时， [Doc Values](https://www.elastic.co/guide/cn/elasticsearch/guide/current/docvalues.html) 提供对映射的快速处理能力，另一方面当映射非常大时，可以通过溢出到磁盘提供足够的扩展能力。

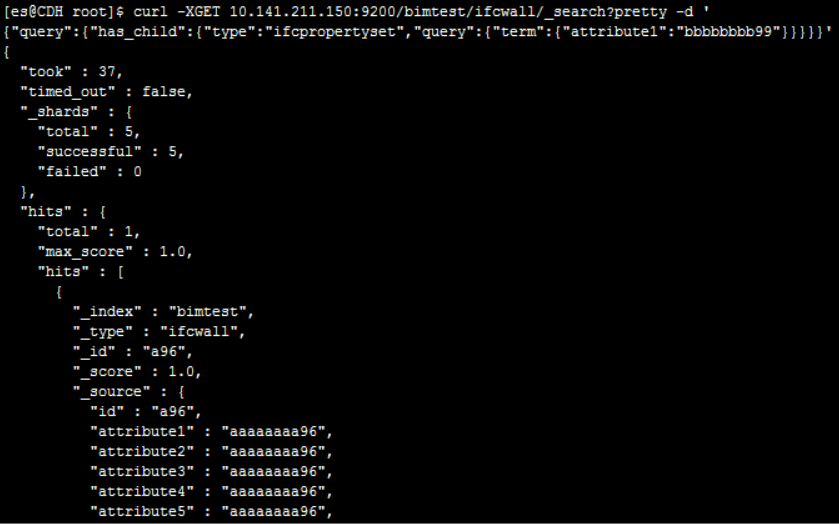
父文档 ID \_parent有两个作用：创建了父文档和子文档之间的关系，并且保证了父文档和子文档都在同一个分片上。

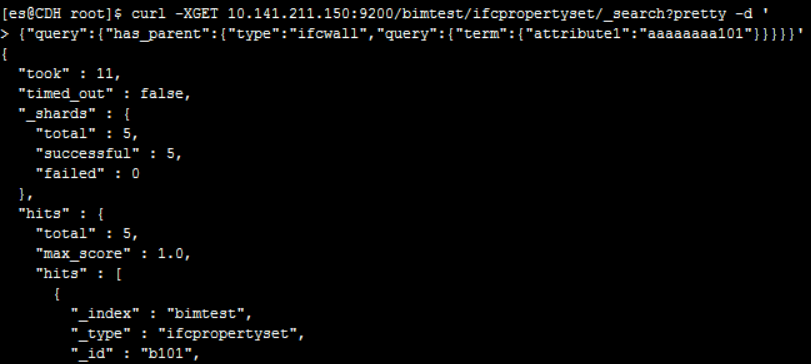
而执行搜索请求时是不需要指定父文档的ID，这是因为搜索请求是向一个索引中的所有分片发起请求，而单文档的操作是只会向存储该文档的分片发送请求。

如果你想要改变一个子文档的 parent 值，仅通过更新这个子文档是不够的，因为新的父文档有可能在另外一个分片上。因此，你必须要先把子文档删除，然后再重新索引这个子文档。

has\_child 过滤的结果没有被缓存. 真的吗？？？

在父-子文档中支持 [子文档聚合](http://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-children-aggregation.html)，这一点和 [嵌套聚合](https://www.elastic.co/guide/cn/elasticsearch/guide/current/nested-aggregation.html) 类似。但是，对于父文档的聚合查询是不支持的（和 reverse\_nested 类似）。





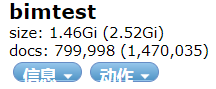
当文档索引性能远比查询性能重要 的时候，父子关系是非常有用的，但是它也是有巨大代价的。其查询速度会比同等的嵌套查询慢5到10倍!

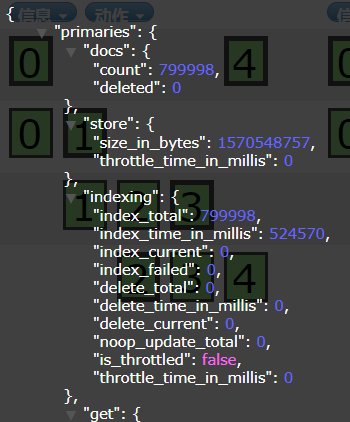
一个分片中父文档越多，那么全局序数的重建就需要更多的时间。父子关系更适合于父文档少、子文档多的情况。

* 尽量少地使用父子关系，仅在子文档远多于父文档时使用。

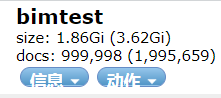
访问es的head插件客户端，10.141.211.152:9100

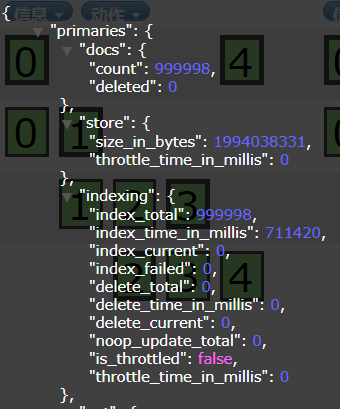
第二次上传ifcwall(id从40万+1，到60万-1)，结果发现，parse错误。原来是忘记了写命令：\_bulk。上传成功后：





第二次上传20万行ifcpropertyset数据(id从40万+1，到60万-1)之后：





第三次上传ifcpropertyset（从20万+1，到40万-1）之后：





第三次上传ifcwall（id从20万+1，到40万-1）之后：





第四次上传ifcwall（id从60万+1，到80万-1）之后：