# ES入门

## 定义

Es是一个开源的高扩展的分布式全文检索引擎，可以近乎实时的存储、检索数据；本身扩展性很好，可以扩展到上百台服务器，处理PB级别的数据。

Es使用java开发并使用Lucene作为其核心来实现所有索引和搜索的功能，但是它的目的是通过简单的Restful Api来隐藏Lucene的复杂性，从而让全文搜索变得简单。

## ES核心概念

|  |  |
| --- | --- |
| 概念名称 | 简单描述 |
| Cluster：集群 | ES可以作为一个独立的单个搜索服务器，但是为了处理大型数据集，实现容错性和高可用性，ES可以运行在许多互相合作的服务器上。这些服务器的集合称为集群。 |
| Node：节点 | 形成集群的每个服务器称为节点 |
| Shard：分片 | 当有大量的文档时，由于内存的限制、磁盘处理能力不足、无法足够快的响应客户端的请求等，一个节点可能不够。这种情况下，数据可以分为较小的分片。每个分片放到不同的服务器上。  当你查询的索引分布在多个分片上时，ES会把查询发送给每个相关的分片，并将结果组合在一起，而应用程序并不知道分片的存在。即：这个过程对用户来说是透明的。 |
| Replia：副本 | 为提高查询吞吐量或实现高可用性，可以使用分片副本。  副本是一个分片的精确复制，每个分片可以有零个或多个副本。ES中可以有许多相同的分片，其中之一被选择更改索引操作，这种特殊的分片称为主分片。  当主分片丢失时，如：该分片所在的数据不可用时，集群将副本提升为新的主分片 |
| 全文检索 | 全文检索就是对一篇文章进行索引，可以根据关键字搜索，类似于mysql里的like语句。  全文索引就是把内容根据词的意义进行分词，然后分别创建索引，例如”你们的激情是因为什么事情来的” 可能会被分词成：“你们“，”激情“，“什么事情“，”来“ 等token，这样当你搜索“你们” 或者 “激情” 都会把这句搜出来 |

## Es数据结构的主要概念（与mysql对比）

|  |  |
| --- | --- |
| Elastic Search | MySQL |
| Index（索引） | Database（数据库） |
| Type（类型） | Table（表） |
| Document（文档） | Row（行） |
| Field（字段） | Column（列） |
| Mapping | Schema |
| Query DSL | SQL |
| GET http://....... | Select \* FROM table |
| PUT http://...... | Update table SET |
| 1. 在一个关系型数据库里面，schema定义了表、每个表的字段，还有表和字段之间的关系。 与之对应的，在ES中：Mapping定义索引下的Type的字段处理规则，即索引如何建立、索引类型、是否保存原始索引JSON文档、是否压缩原始JSON文档、是否需要分词处理、如何进行分词处理等。 2. 在数据库中的增insert、删delete、改update、查search操作等价于ES中的增PUT/POST、删Delete、改\_update、查GET. | |

# 基础入门练习

PUT 类似于SQL中的增

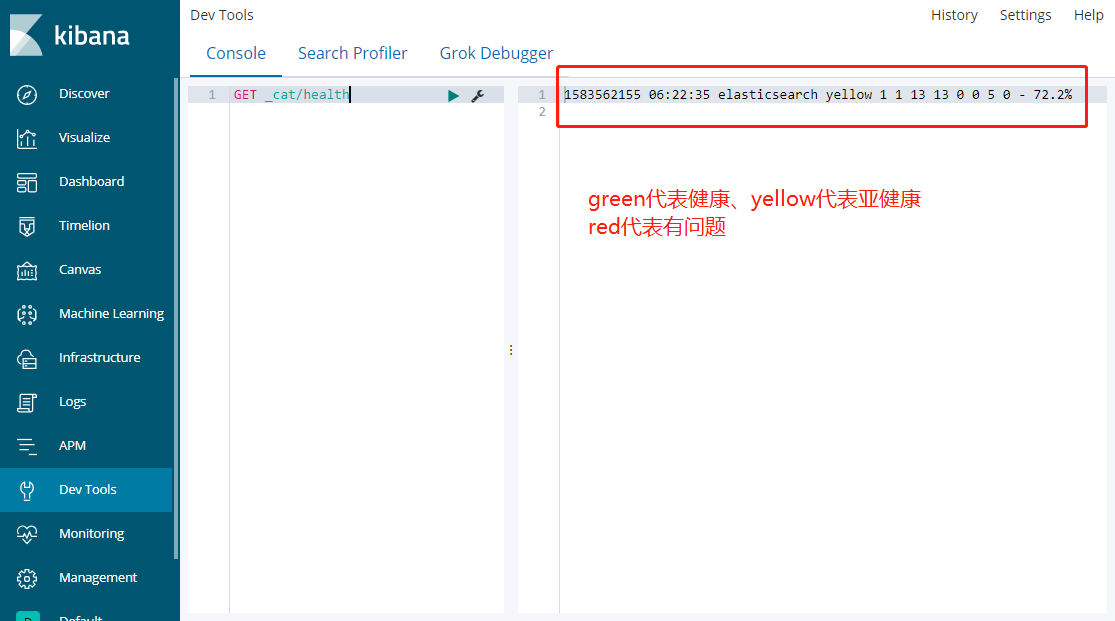
DELETE 类似于SQL中的删

POST 类似于SQL中的改

GET 类似于SQL中的查

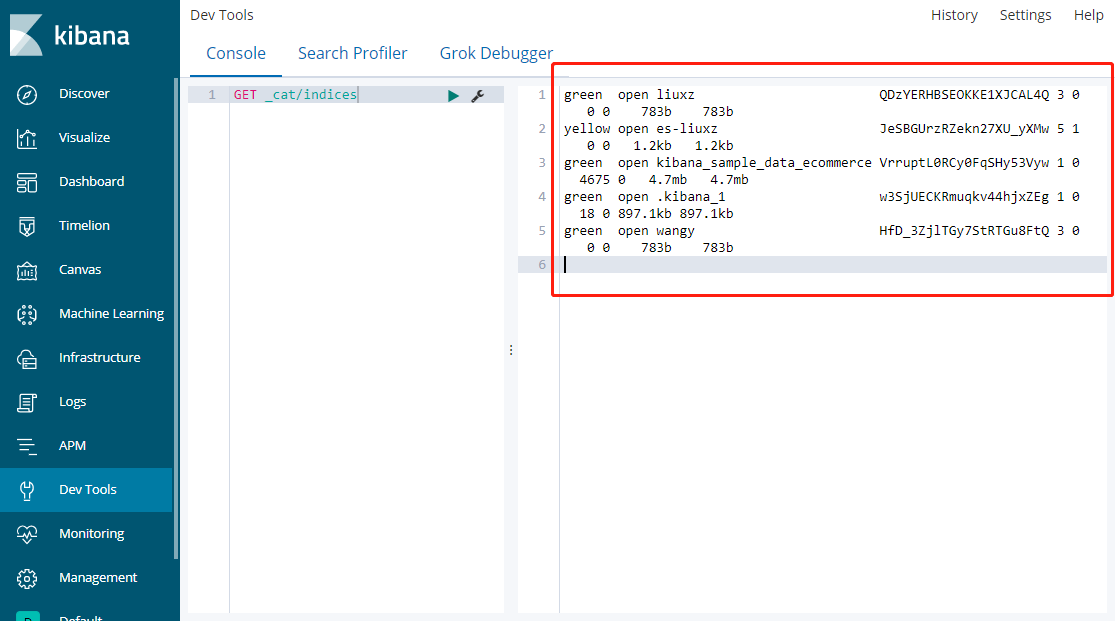
## 查看集群的健康情况

### 命令：GET \_cat/health

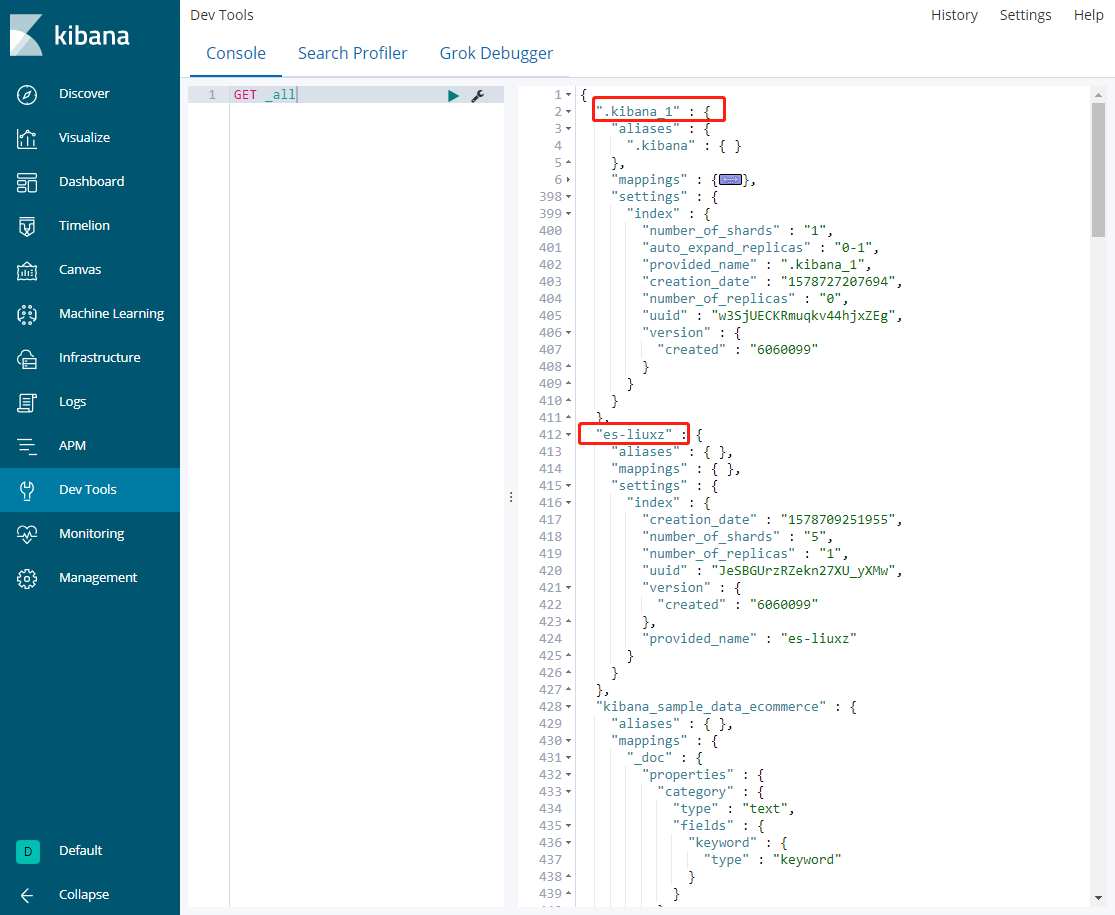


## 查询所有的index

### 命令1：GET \_cat/indices



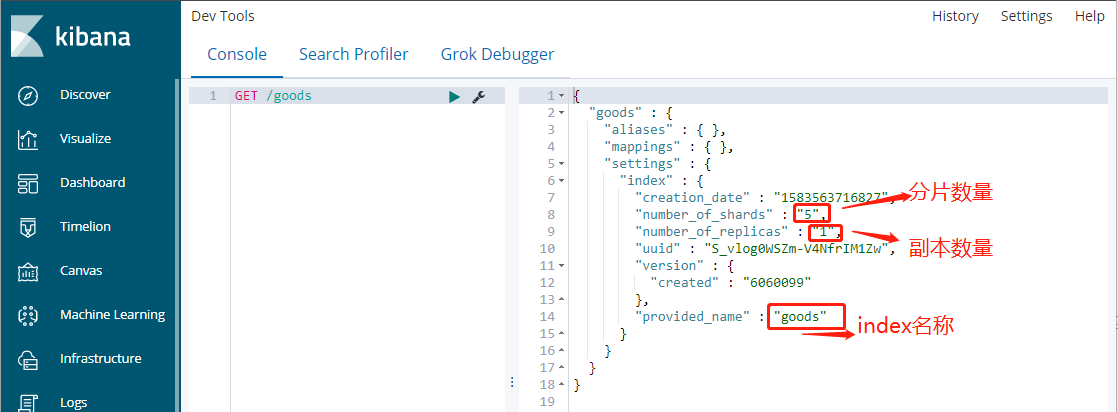
### 命令2：GET \_all



## 新建index

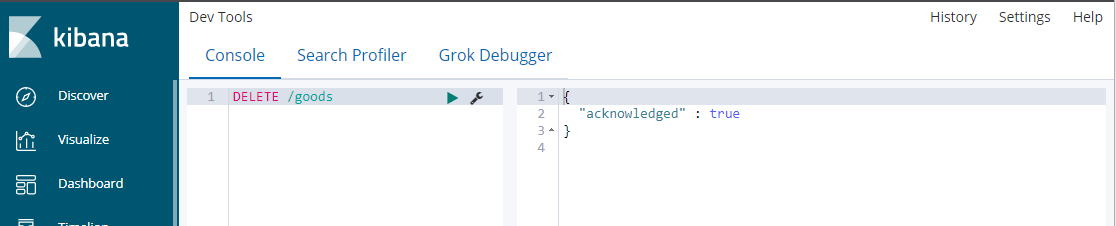
### 命令：PUT /索引名称

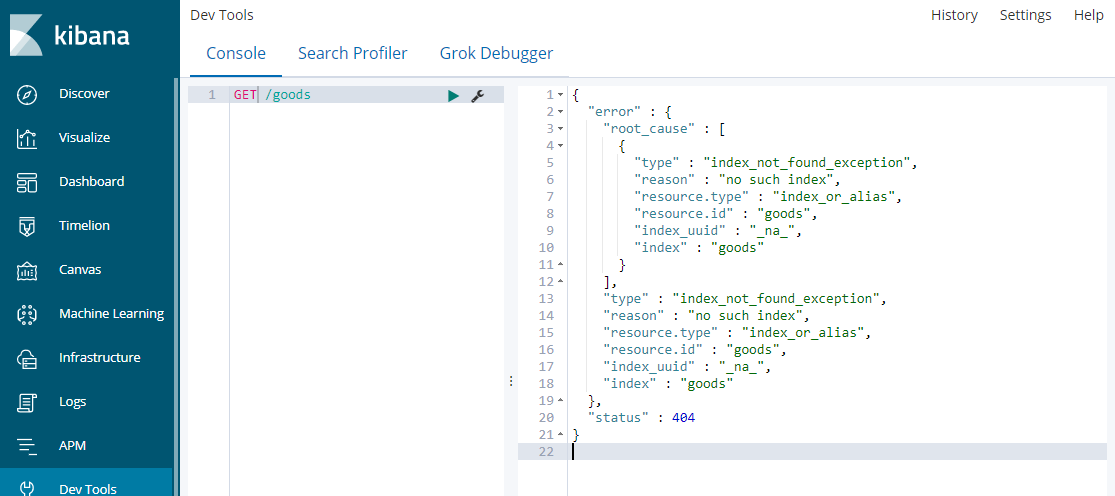




## 删除index

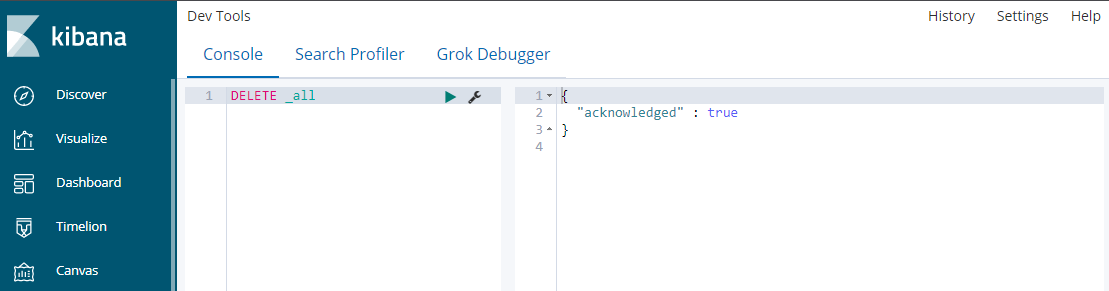
### 命令：DELETE /索引名称





## 删除所有index

### 命令：DELETE \_all



# ES的CRUD操作

案例描述：

通过演示一个购物品类的例子，来感受以下es的语法特点

Index：手机

Type：苹果手机

Id：具体的苹果手机型号，如5s、6p、6plus ……

## 新建一个商品

### 语法：PUT /index/type/id

PUT /mobile/apple/1

{

"name":"iPhone 11pro",

"desc":"苹果11 pro",

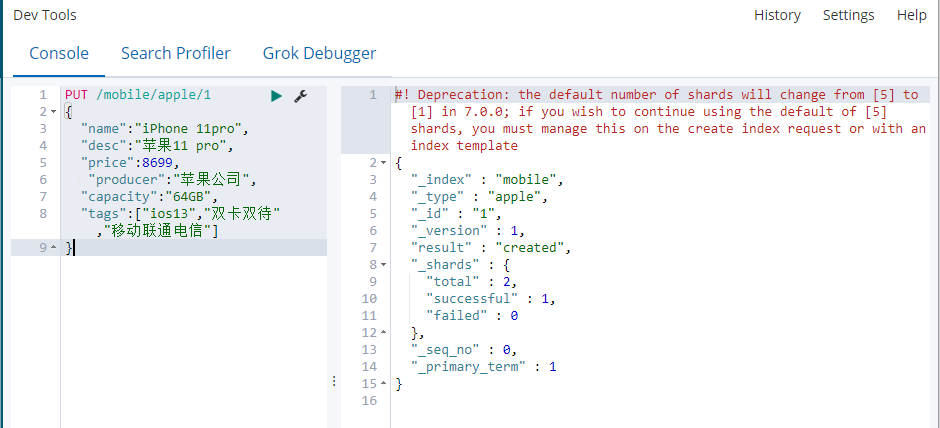
"price":8699,

"producer":"苹果公司",

"capacity":"64GB",

"tags":["ios13","双卡双待","移动联通电信"]

}



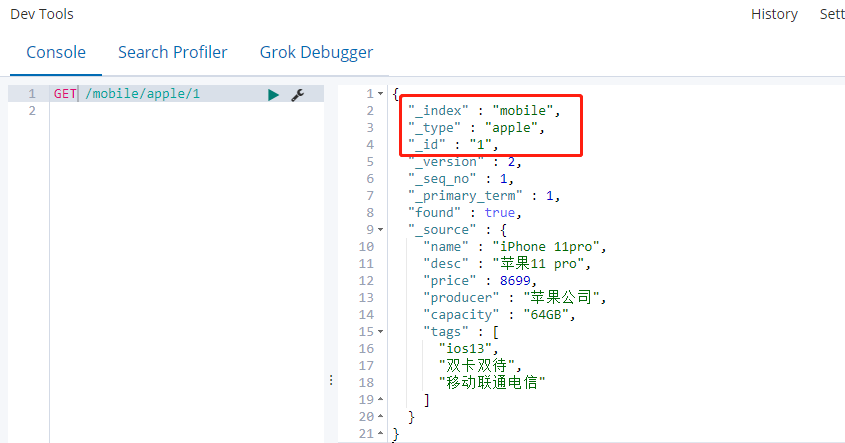
注意：我们插入数据的时候，如果我们的语句中指明了index和type，如果ES里面不存在，默认帮我们自动创建。

相同index/type/id的输入内容再次添加，版本号会叠加，结果会提示updated更新



## 2、查询一个商品

### 语法： GET /index/type/id



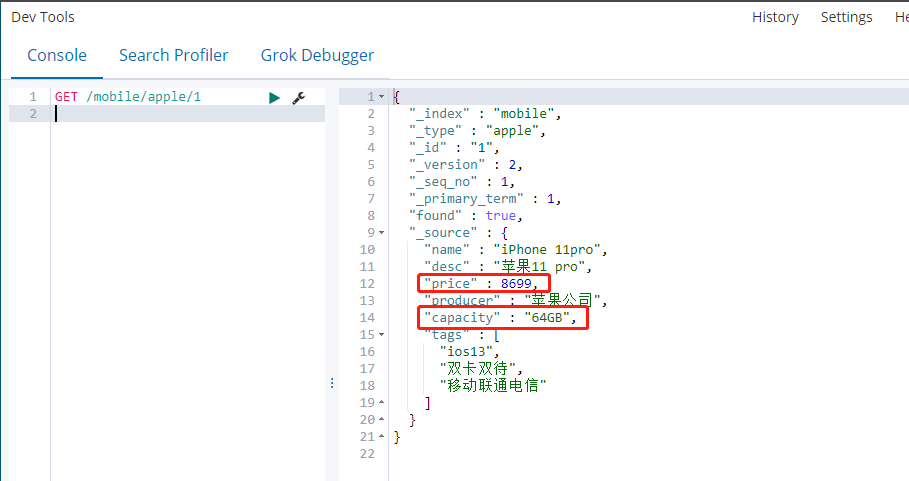
## 3、修改商品数据

可以使用POST来修改数据，也可以使用PUT修改数据，两者的区别在于：

* POST是局部更新数据，别的数据不动。
* PUT是全局更新

### 3.1、PUT方式修改——演示不丢数据

#### 修改前：



#### 修改后：

修改参数：将price修改为9999 ，capacity修改为256G

PUT /mobile/apple/1

{

"name":"iPhone 11pro",

"desc":"苹果11 pro",

"price":9999,

"producer":"苹果公司",

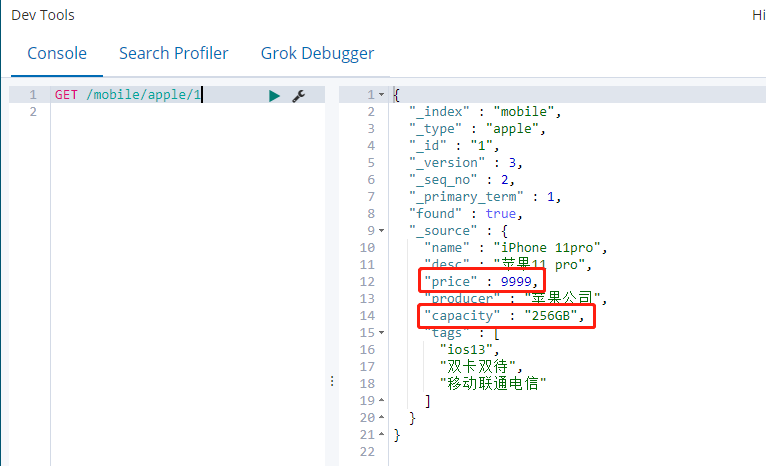
"capacity":"256GB",

"tags":["ios13","双卡双待","移动联通电信"]

}

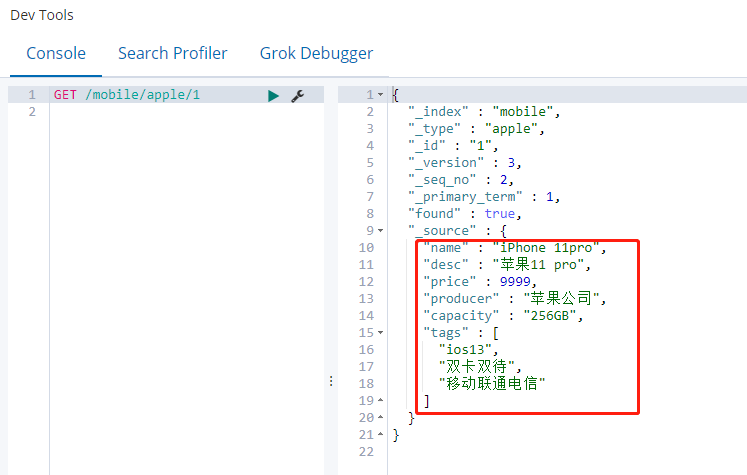


修改成功：



### 3.2、PUT方式修改——演示丢数据

#### 修改前：



#### 修改后：

修改内容：将price修改为12999 ，capacity修改为512G

注意：只传递要修改的字段，不传递不用修改的字段

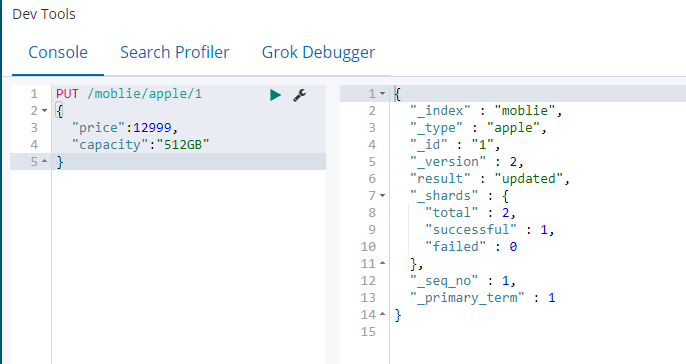
PUT /moblie/apple/1

{

"price":12999,

"capacity":"512GB"

}

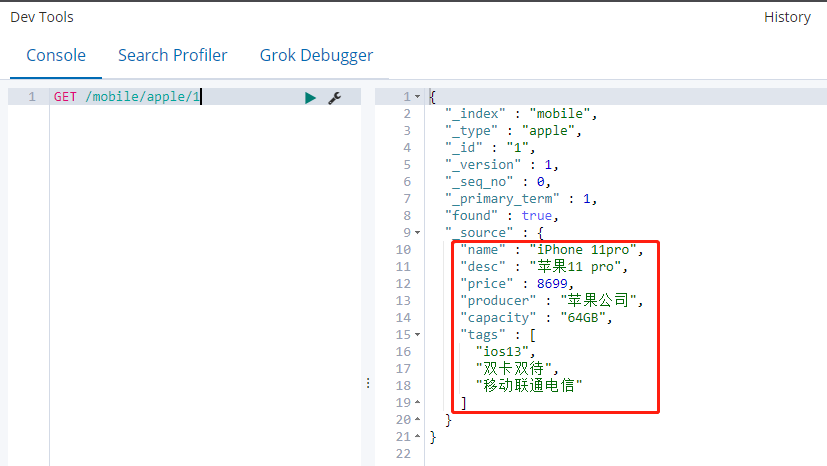




### 3.3 POST方式修改——不加\_update

不加\_update**，**此种方式效果等同3.2

#### 修改前：



#### 修改后：

修改内容：将price修改为12999 ，capacity修改为512G

注意：只传递要修改的字段，不传递不用修改的字段

POST /mobile/apple/1

{

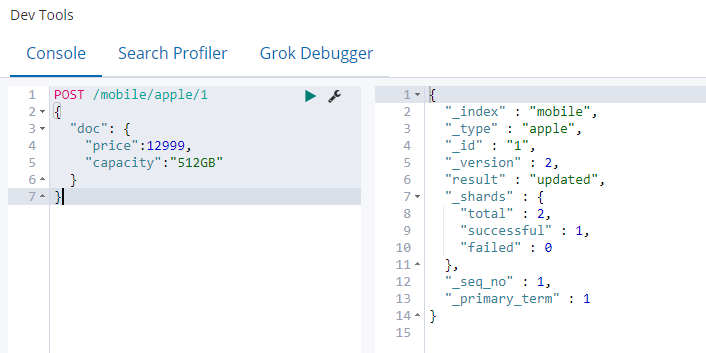
"doc": {

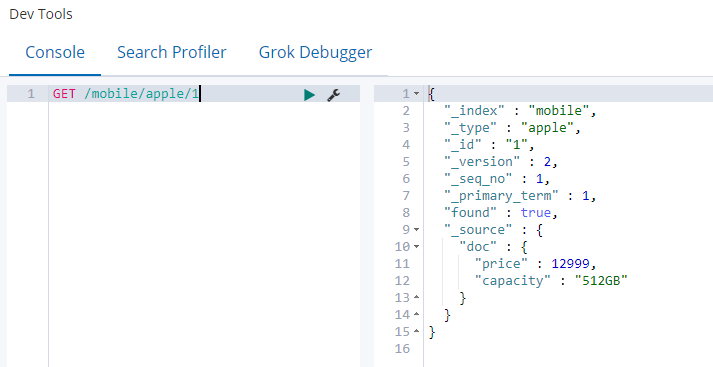
"price":12999,

"capacity":"512GB"

}

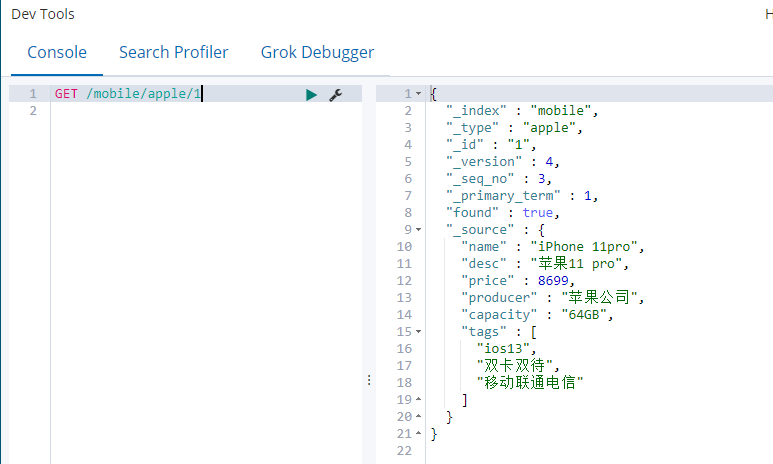
}





### 3.4、POST方式修改——加\_update

#### 修改前：



#### 修改后：

修改内容：将price修改为12999 ，capacity修改为512G

注意：只传递要修改的字段，不传递不用修改的字段

**POST** /mobile/apple/1/**\_update**

{

**"doc": {**

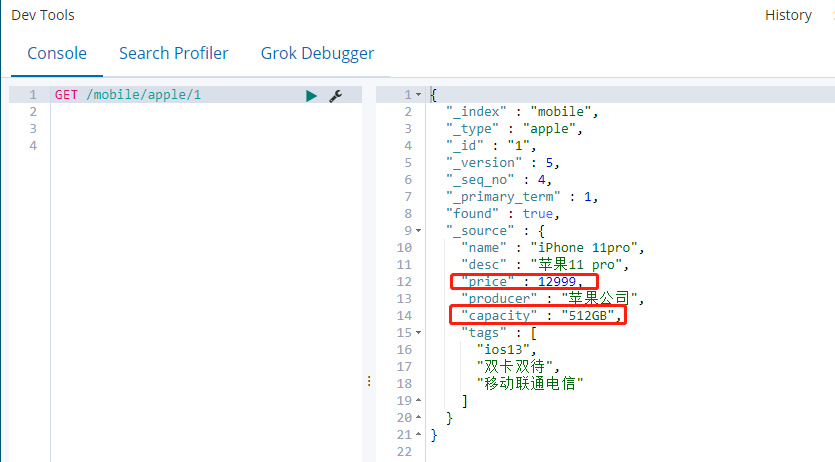
"price":12999,

"capacity":"512GB"

**}**

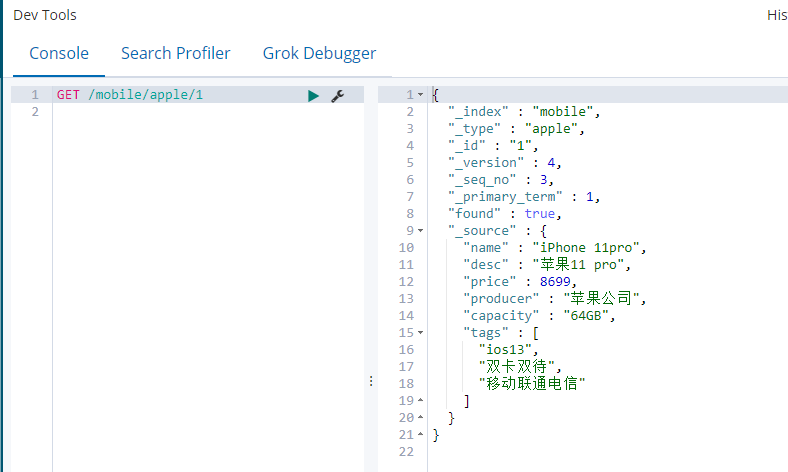
}





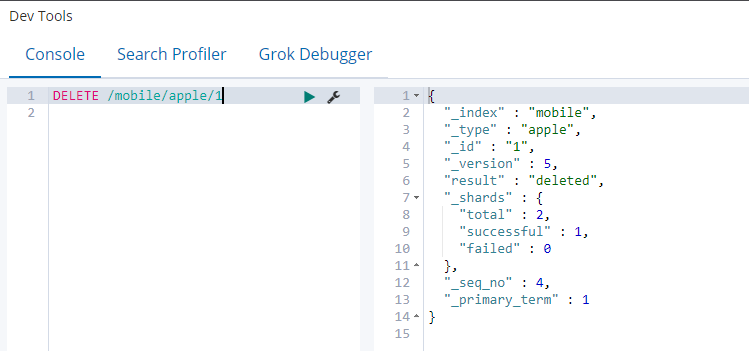
## 4、删除商品数据

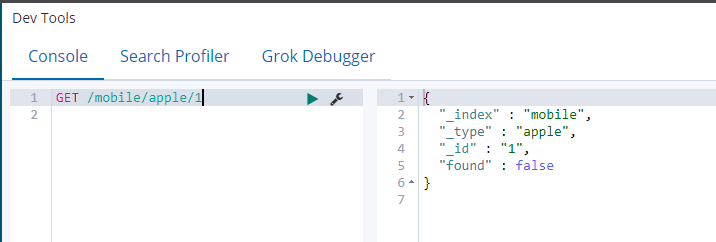
#### 删除前：



#### 删除后：

DELETE /mobile/apple/1





# DSL语言

ES最主要是用来做搜索和分析的。所以DSL还是对于ES很重要的

DSL：domain Specialed Lanaguage 在特定领域的语言

|  |
| --- |
| PUT /mobile/apple/1  {  "name":"iPhone 11pro",  "desc":"苹果11 pro",  "price":8699,  "producer":"富士康深圳公司",  "capacity":"64GB",  "tags":["ios13","双卡双待","移动联通电信"]  }  PUT /mobile/apple/2  {  "name":"iPhone X",  "desc":"苹果 X",  "price":7299,  "producer":"富士康郑州公司",  "capacity":"128GB",  "tags":["ios13.3","双卡双待","OLED 屏幕"]  }  PUT /mobile/apple/3  {  "name":"iPhone 8plus",  "desc":"苹果 8p",  "price":5399,  "producer":"富士康东莞公司",  "capacity":"64GB",  "tags":["ios12.1","双卡双待","普通屏幕"]  }  PUT /mobile/apple/4  {  "name":"iPhone 7plus",  "desc":"苹果 7p",  "price":4399,  "producer":"富士康南京公司",  "capacity":"516GB",  "tags":["ios11.1","销量最多","普通屏幕"]  }  PUT /mobile/apple/4  {  "name":"iPhone 5s",  "desc":"苹果 5s",  "price":3999,  "producer":"富士康惠州公司",  "capacity":"32GB",  "tags":["ios9.1","小巧玲珑","普通屏幕"]  } |

新建以上四种机型，作为练习数据。

## 查询Type下的所有Document

### 对比：select \* from table

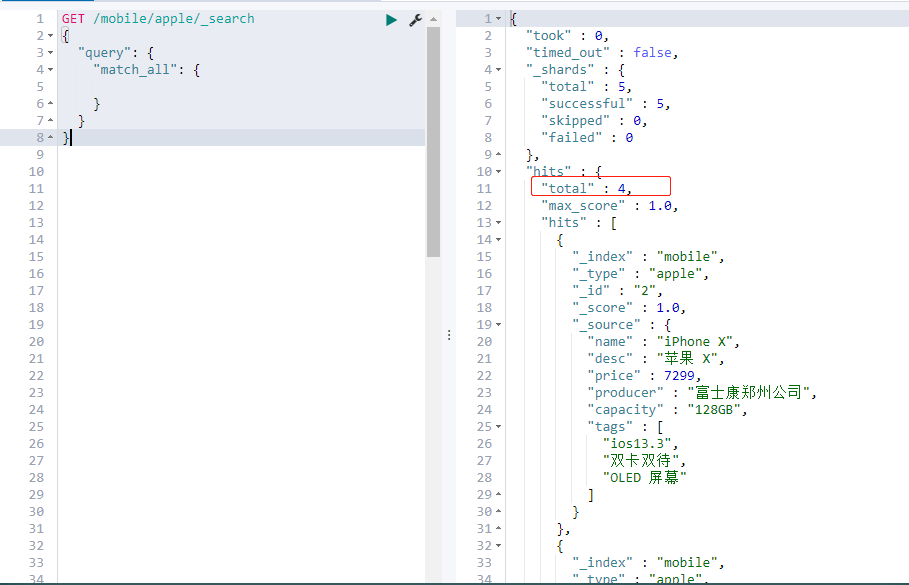
类似mysql里面的查询指定表的所有数据：select \* from table

### 语法：match\_all

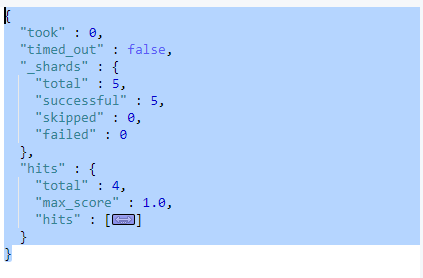
使用match\_all 可以查询到所有文档，是没有查询条件下的默认语句。

|  |
| --- |
| GET /mobile/apple/\_search  {  "query": {  "match\_all": {    }  }  } |

### 查询结果：



### 查询结果各字段含义解释



{

"took" : 0, // 运行命令到获取结果所耗费的时长（毫秒）

"timed\_out" : false, // 是否超时：true是、false否

"\_shards" : { // 分片的情况

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : { // 获取到的数据的情况

"total" : 4, // 总的数据条数

"max\_score" : 1.0, // 所有数据里面打分最高的分数

"hits" : []

}

}

## 查询Type下满足查询条件、指定排序、并分页的Document

截止当前，在索引mobile类型apple下面共有四个文档，皆为不同型号的苹果手机。为了方便演示，现在加入两个文档：一个Mac Pro文档、一个iPad文档

PUT /mobile/apple/5

{

"name":"macbook pro",

"desc":"苹果电脑 mac pro",

"price":16299,

"producer":"富士康郑州公司",

"capacity":"512GB",

"tags":["高分辨率","指纹解锁"]

}

PUT /mobile/apple/6

{

"name":"ipad pro",

"desc":"苹果笔记本 ipad",

"price":9989,

"producer":"富士康郑州公司",

"capacity":"256GB",

"tags":["教育优惠","指纹解锁"]

}

### 对比：

类似mysql里面的查询指定表的满足条件的数据：

select \* from table where name like xxx order by price desc limit 0,2

### 语法：match

**注意：默认是条件分词，如果要精确匹配，请使用match\_phrase**

GET /mobile/apple/\_search

{

"**query**": {

"match": {

"name": "iphone"

}

},

"**sort**": [

{

"price": {

"order": "desc"

}

}

],

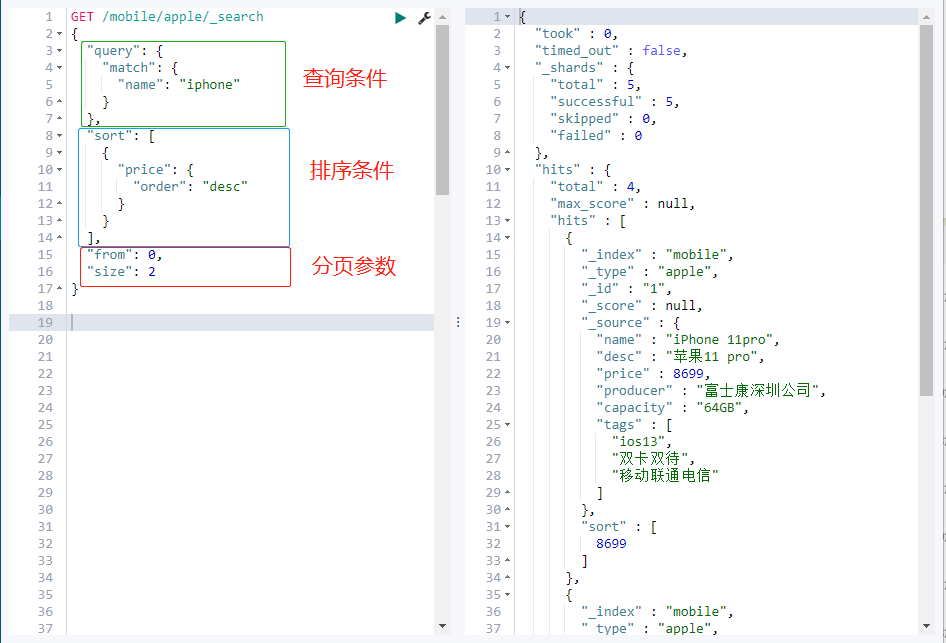
"**from**": 0,

"**size**": 2

}

### 查询结果：

查询name里面包含iPhone字眼的文档，并按照价格进行排序，分页返回每页展示2个商品



### 返回指定字段，不返回全部

#### 类比：

类似select name，price，capacity from apple where name like ‘%iphone%’

Order by price desc limit 0,2

查询参数

#### 参数：

GET /mobile/apple/\_search

{

"**\_source**": ["capacity","price","name"],

"query": {

"match": {

"name": "iphone"

}

},

"sort": [

{

"price": {

"order": "desc"

}

}

],

"from": 0,

"size": 2

}

#### 查询结果：



## 多条件查询(条件分词)

如果需要多个查询条件拼接在一起查询，使用**bool**关键字，bool 过滤可以用来合并多个过滤条件查询结果的布尔逻辑，它包含以下操作符：

* **must ::** 多个查询条件的完全匹配,相当于 and。
* **must\_not ::** 多个查询条件的相反匹配，相当于 not。
* **should ::** 至少有一个查询条件匹配, 相当于 or。

 这些参数可以分别继承一个过滤条件或者一个过滤条件的数组

### 类比：

select name，price，desc, capacity from apple where name like ‘%iphone%’ and

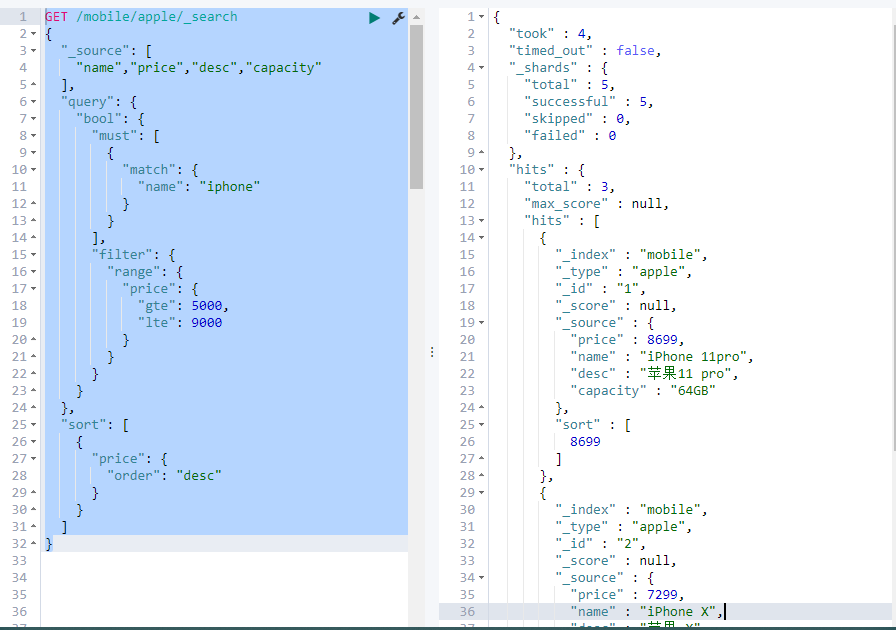
price between 5000 and 8000Order by price desc

### 查询参数

|  |
| --- |
| GET /mobile/apple/\_search  {  "\_source": [  "name","price","desc","capacity"  ],  "query": {  "bool": {  "must": [  {  "match": {  "name": "iphone"  }  }  ],  "filter": {  "range": {  "price": {  "gte": 5000,  "lte": 9000  }  }  }  }  },  "sort": [  {  "price": {  "order": "desc"  }  }  ]  } |



### 查询结果



## 查询条件不分词

### 反例：查询条件分词

Type类型下的文档中，有name为iPhone 5s、iPhone X、iPhone 8plus、iPhone 11pro等五种名称的apple手机：

条件分词情况下查询：查询条件：iphone 11pro，会将上述五种机型都查询出来

GET /mobile/apple/\_search

{

"\_source": "name",

"query": {

"**match**": {

"name": "iphone 11pro"

}

}

}



### 正例：查询条件不分词（精确匹配）

GET /mobile/apple/\_search

{

"query": {

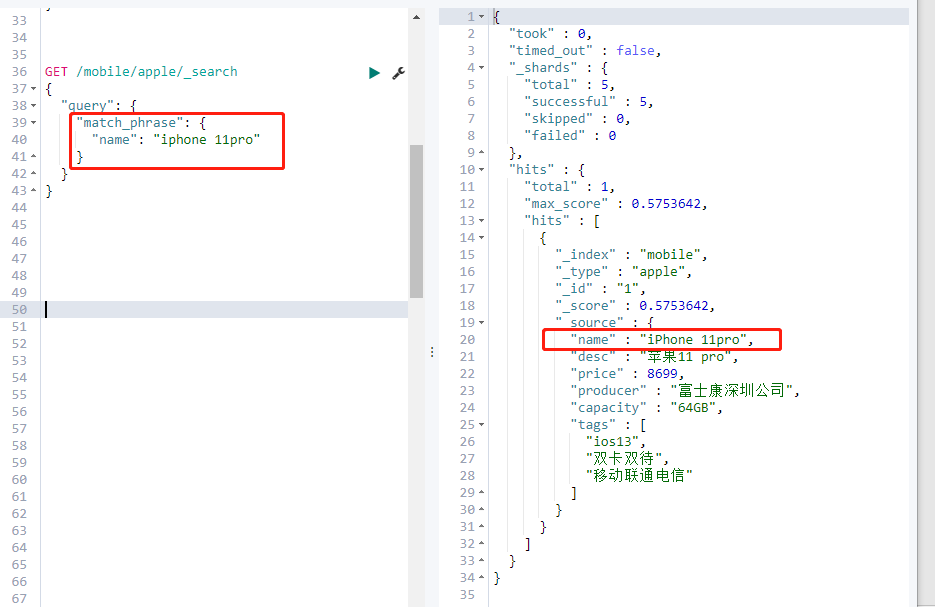
"match\_phrase": {

"name": "iphone 11pro"

}

}

}



# elasticsearch环境搭建

## 1、springboot整合es

略

## 2、kibana安装

略

## 3、elasticsearch-header安装

略

## 4、IK Analyzer安装

略

## 5、spring data查询实例

见代码实例：<https://github.com/wellzhi/es-journey>

# 聚合（aggregation）

solr无此功能，这是es特有的特性，可以让我们极其方便的实现对数据的统计、分析。

|  |
| --- |
| 1. 什么品牌的手机最受欢迎？ 2. 这些手机的平均价格、最高价格、最低价格？ 3. 这些手机每月的销售情况如何？   实现这些统计功能的比数据库的sql要方便的多，而且查询速度非常快，可以实现近实时搜索效果。 |

Elasticsearch中的聚合，包含多种类型，最常用的两种，一个叫**桶**，一个叫**度量**：

## 1、桶（bucket）

1. 桶的**作用**：按照某种方式对数据进行**分组**，每一组数据在ES中称为一个桶，例如我们根据国籍对人划分，可以得到中国桶、英国桶，日本桶……，或者我们按照年龄段对人进行划分：0~10,10~20,20~30,30~40等。
2. 划分桶的方式：

|  |  |
| --- | --- |
| 划分方式 | 说明 |
| Date Histogram Aggregation | 根据日期阶梯分组，例如给定阶梯为周，会自动每周分为一组 |
| Histogram Aggregation | 根据数值阶梯分组，与日期类似 |
| Terms Aggregation | 根据词条内容分组，词条内容完全匹配的为一组 |
| Range Aggregation | 数值和日期的范围分组，指定开始和结束，然后按段分组 |
| ……其他方式 |  |

综上所述，我们发现bucket aggregations 只负责对数据进行分组，并不进行计算，因此往往bucket中往往会嵌套另一种聚合：metrics aggregations即度量

## 2、度量（metrics）

分组完成以后，我们一般会对组中的数据进行聚合运算，例如求平均值、最大、最小、求和等，这些在ES中称为度量。比较常用的一些度量聚合方式：

* Avg Aggregation：求平均值
* Max Aggregation：求最大值
* Min Aggregation：求最小值
* Percentiles Aggregation：求百分比
* Stats Aggregation：同时返回avg、max、min、sum、count等
* Sum Aggregation：求和
* Top hits Aggregation：求前几
* Value Count Aggregation：求总数
* ……

注意：在ES中，需要进行聚合、排序、过滤的字段其处理方式比较特殊，因此不能被分词。这里我们将color和make这两个文字类型的字段设置为keyword类型，这个类型不会被分词，将来就可以参与聚合