# ElasticSearch-DSL搜索文档

同学们，我们之前学习了ES的基本功能：操作索引库、操作文档，并且利用RestClient向ES中导入了很多酒店数据。不过数据的存储不是我们使用ES目的，索引库的操作、文档的操作也不是最重要的目的，我们使用ES最终也是最重要的目的是为了从这海量的数据中检索/搜索出你所需要的数据，这就需要用到ES的搜索功能：

1：DSL查询文档；

2：DSL对搜索结果处理（如：排序、分页、高亮等）；

3：RestClient查询文档和对搜索结果处理；

4：黑马旅游案例

## 前言

主要从以下方面学习DSL查询文档：

1：DSL查询分类和基本语法

2：全文检索查询

3：精准查询；

4：地理坐标查询

5：组合查询

## DSL Query分类

ElasticSearch提供了基于JSON的DSL（Domain Specific Language）来定义查询，是基于JSON风格的查询语句，即用**JSON来去表述查询条件，然后发送给ES服务，ES服务基于这些查询条件返回相应结果**给你。DSL种包含的查询种类非常多，最常见的查询类型包括：

### 1：查询所有：

查询出所有数据，顾名思义**，就是把所有的数据都查出来，查询所有一般在测试的时候用**。例如：match\_all（顾名思义就是把所有的数据都查出来，当然也不是真的把所有的数据都查出来，意思是查询不加条件限制，理论上能查出所有，但是其实会有分页的限制，一次最多查个20条）

### 2：全文检索（full text）查询：

全文检索查询这个就是跟我们前面讲倒排索引是的查询一样，原理就是**当用户输入查询内容的时候，利用分词器对用户输入的内容分词，得到一个个词条，然后去倒排索引库中去匹配，匹配到词条就找到词条所对应的文档id，然后在拿着文档id去索引库查询文档，返回给用户**。所以全文检索就是我们一开始讲的倒排索引的方式的查询是一样的。全文检索包含的种类也非常多，常见的有两种：match\_query和multi\_match\_query。

### 3：精确查询：

精确查询或者精准查询，顾名思义根据精确的词条值去查找数据，一般查找keword、数值、日期、Boolean等数据类型的字段。

朋友们，这些数据类型的字段的共同特点**是不需要做分词，但是要去做倒排索引**，即字段内容整体作为一个词条存入倒排索引，因此在查找的时候也不需要分词了。这跟上面的全文检索不一样了，**全文检索要根据搜索内容进行分词，精确查询直接把搜索的内容去倒排索引库去匹配就行，这是精确查询和全文检索的区别**，所以一般情况下，全文检索一般是去查能分词的字段，即text类型字段，而精确查询则一般去查不分词的字段。

常见的精确查询有：ids（根据id精确匹配）；range（根据数值范围进行查询，如价格）；term（根据数据的值进行查询，如keyword、和Boolean的字段）。

### 4：地理（geo）查询：

顾名思义，就是根据地理坐标经纬度查询。之前说过ES中地理坐标是有特殊的表示的，所以在查询的时候也有特殊的查询！常见的地理查询有：geo\_distance和geo\_bounding\_box。

### 5：复合（compound）查询：

**复合查询本省是没有查询条件的，可以将上述各种查询条件组合起来，合并查询条件，即这种查询本身是没有查询条件的，是把上述的各种查询条件用某种方式组合在一起，形成一个新的查询，所以叫复合查询**。不同的复合查询组合的方式和目的是不一样的，常见的复合查询有：

bool：顾名思义就是利用布尔或者逻辑的运算把其他查询条件组合起来

function\_score：用来控制相关度算分的

以上就是常见的查询语句。

## DSL Query的基本语法：

就是要指定查询类型、查询条件、搜索条件值这三个东西！

**GET /索引库名/\_search**

**{**

**“query”: {**

**“查询类型”: {**

**“查询条件”: “条件值”**

**}**

**}**

**}**

**对应查询所有的DSL语句就为：**

**GET /hotel/\_search**

**{**

**“query”: {**

**“match\_all”: {**

**}**

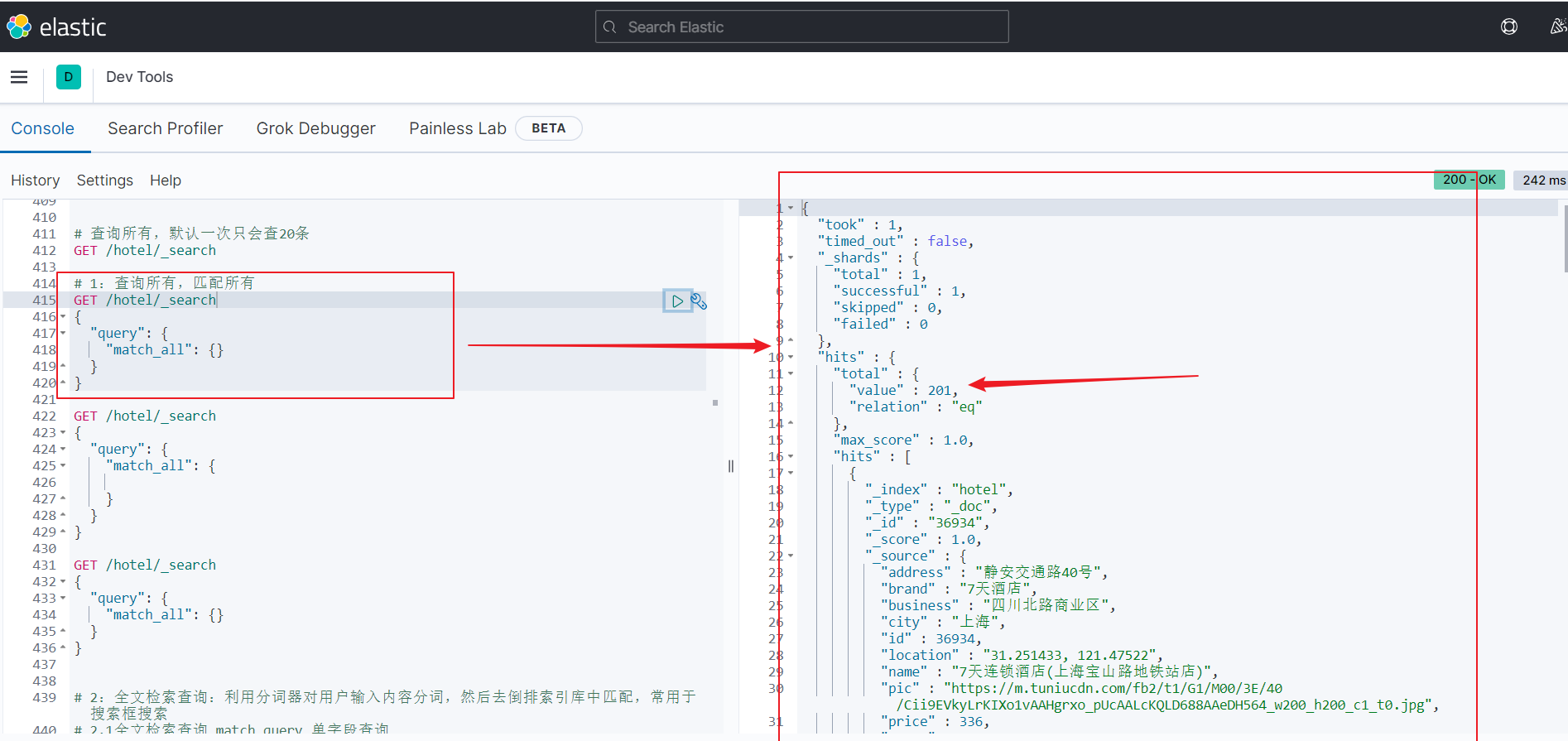
**}**

**}**

如上图，查询的基本语法对应的查询所有的DSL语句如上，查询类型是match\_all，match\_all就是查询所有，因为是查询所有，那么没有查询条件，因此，查询条件和查询值不用写，将来如果查询类型不是match\_all，而是其他，查询条件和查询值就需要写哦！

### DevTools控制台测试查询所有DSL语句：

看到索引库hotel共有文档201条，hits就是本次match\_all查询所有搜索命中的文档，是个数组，数组中就装了一个一个文档对象！查询所有默认会返回10条文档，因为会避免一下子全查出来数据量太多对内存、服务器过大的压力，所以只查了一部分，至于怎么查后面的数据，学习了ES分页查询你就知道了！



### 总结：查询DSL的基本语法是什么？

**GET /索引库名/\_search**

**{**

**“query”: {**

**“查询类型”: {**

**“查询条件”: “条件值”**

**}**

**}**

**}**

## 查询所有：match\_all

### 对应查询所有的DSL语句就为：

**GET /hotel/\_search**

**{**

**“query”: {**

**“match\_all”: {**

**}**

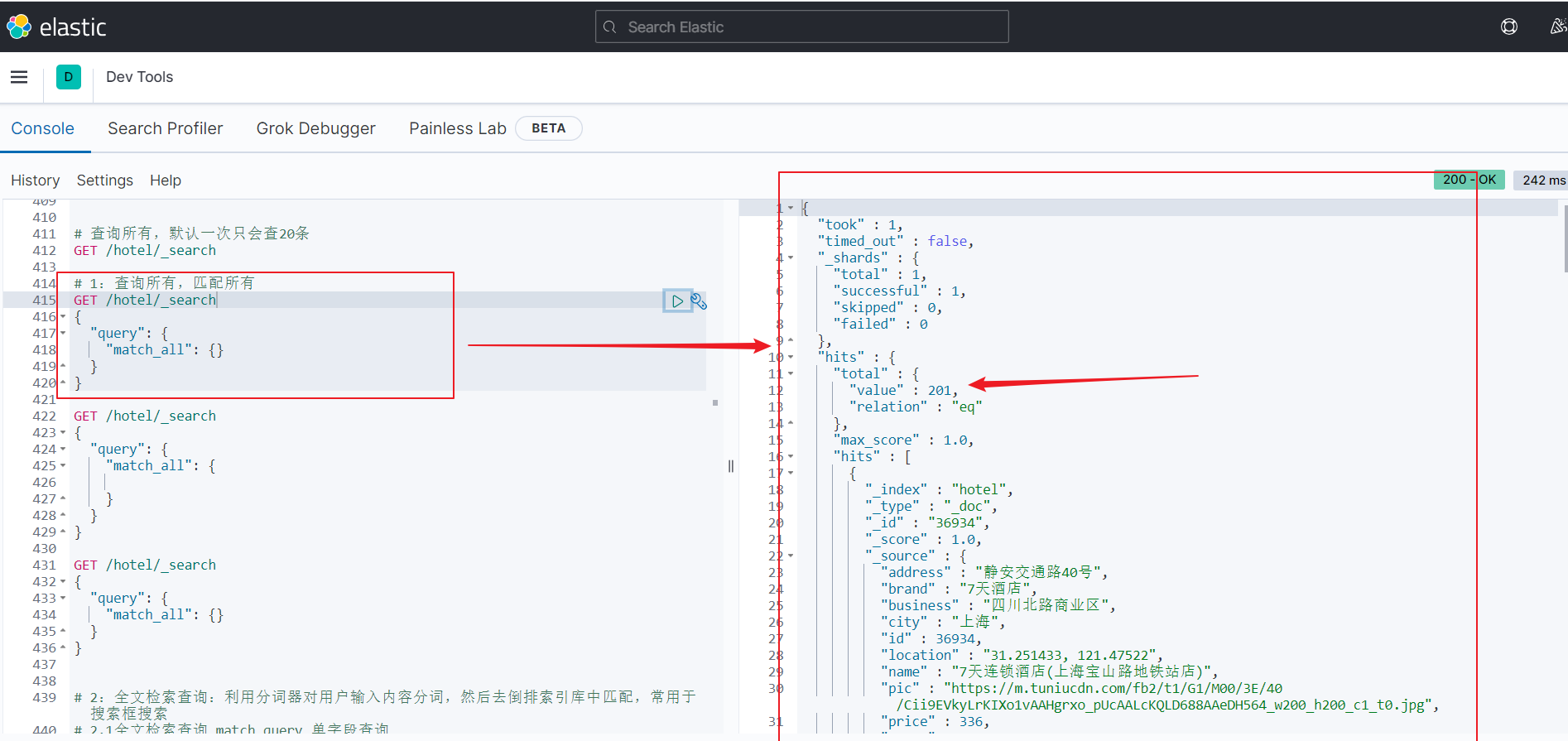
**}**

**}**

如上，查询所有的DSL语句如上，查询类型是match\_all，match\_all就是查询所有，因为是查询所有，那么没有查询条件，因此，查询条件和查询值不用写，将来如果查询类型不是match\_all，而是其他，查询条件和查询值就需要写哦！

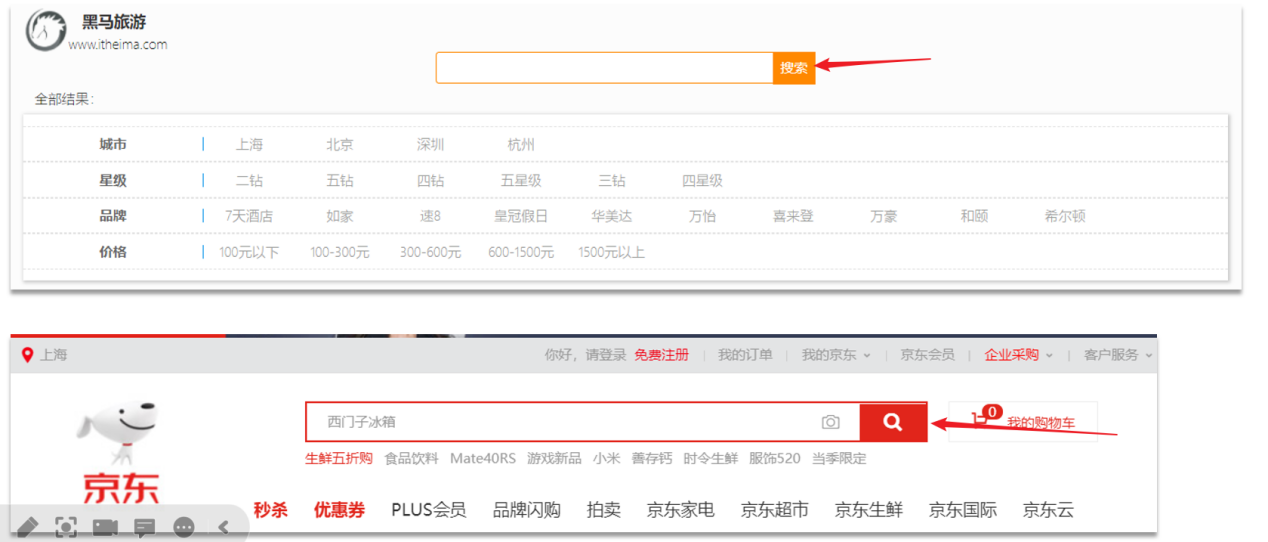
### DevTools控制台测试查询所有DSL语句：

看到索引库hotel共有文档201条，hits就是本次match\_all查询所有搜索命中的文档，是个数组，数组中就装了一个一个文档对象！查询所有默认会返回10条文档，因为会避免一下子全查出来数据量太多对内存、服务器过大的压力，所以只查了一部分，至于怎么查后面的数据，学习了ES分页查询你就知道了！



## 全文检索查询：match & multi\_match

全文检索查询会对用户输入内容利用分词器进行分词，然后去倒排索引库中匹配，常用于搜索框搜索。什么是搜索框搜索呢？比如去京东买东西，京东的首页会有搜索框，在搜索框输入想要购买的商品信息，比如“西门子冰箱”，这个时候，点击搜索按钮，京东后台就可以对输入的搜索内容做分词，然后得到“西门子”和“冰箱”两个词/词条，然后拿着这两个词分别去倒排索引库去检索和匹配，最终得到相关的文档，也就是商品，然后返回给你。这就是全文检索常见的场景：搜索框搜索！



我们知道，全文检索又分为match查询和multi\_match查询

### Match查询：

全文检索查询的一种，会对用户输入的内容分词，然后去倒排索引库中检索、匹配！

#### 通用语法如下：

**GET /索引库名/\_search**

**{**

**“query”: {**

**“match”: {**

**“FIELD”: “TEXT”**

**}**

**}**

**}**

FIELD表示你要根据什么字段去做查询，一个文档里包括的字段是比较多的，而且字段的类型也不尽相同，有keyword、text、数值等，作为全文检索来说，要对搜索内容做分词，然后去倒排索引表做匹配，所以一般来说，全文检索的match查询中所要查询的FIELD字段是文档中TEXT文本类型的字段。

例如我们创建的hotel索引库来讲，在我们的酒店索引当中，我们其实text类型还挺多哦，但是我们将来参与搜索的字段应该是什么字段？我们把品牌、商圈、酒店名的信息都copy\_to到了all这个复合字段中了，还有印象吧，所以我们将来参与搜索的应该是all字段！TEXT则是我们要搜索的内容了。

#### 演示Match查询

##### DSL语句如下：

**GET /hotel/\_search**

**{**

**"query": {**

**"match": {**

**"all": "如家外滩"**

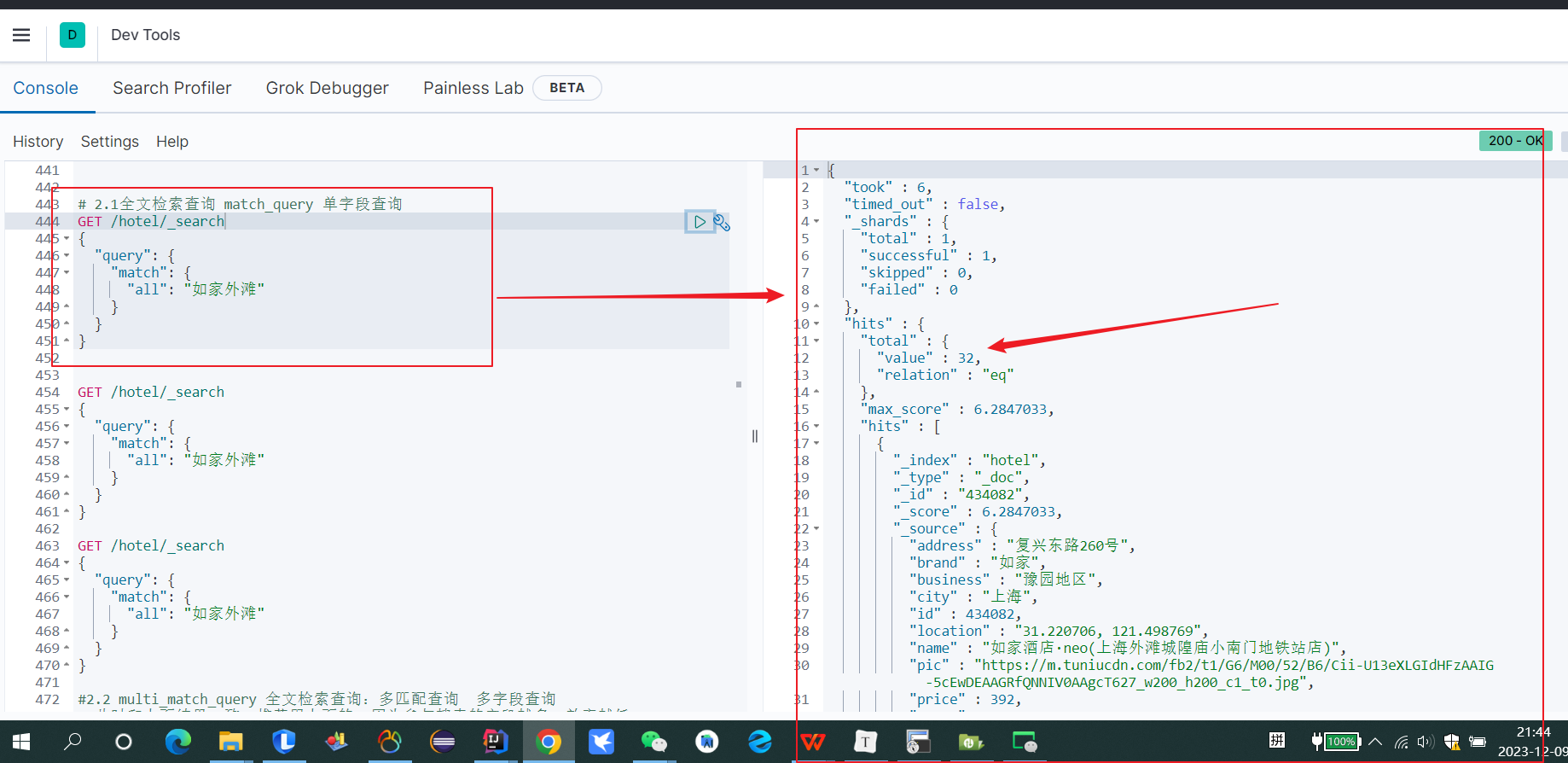
**}**

**}**

**}**

##### 在DevTools控制台测试match查询：

你看你看，查询到32条相关文档，并且查询结果中，相关度高的排在前面呢！很符合我们全文检索的需求！



### multi\_match查询

多匹配查询，多字段查询，与上面的match查询类似，区别在于这个multi\_match可以多字段查询，也就是说match查询只能查一个字段，multi\_match可以查好几个字段。

#### 通用语法如下：

**GET /索引库名/\_search**

**{**

**“query”: {**

**“multi\_match”: {**

**“query”: “TEXT”,**

**“fields”: [“FIELD1”, “FIELD2”]**

**}**

**}**

**}**

查询类型是“multi\_match”,“query”表示搜索的内容，“fields”中指定多个字段！

#### 演示multi\_match查询

##### DSL语句如下：

**GET /hotel/\_search**

**{**

**"query": {**

**"multi\_match": {**

**"query": "如家外滩",**

**"fields": ["brand", "name", "business"]**

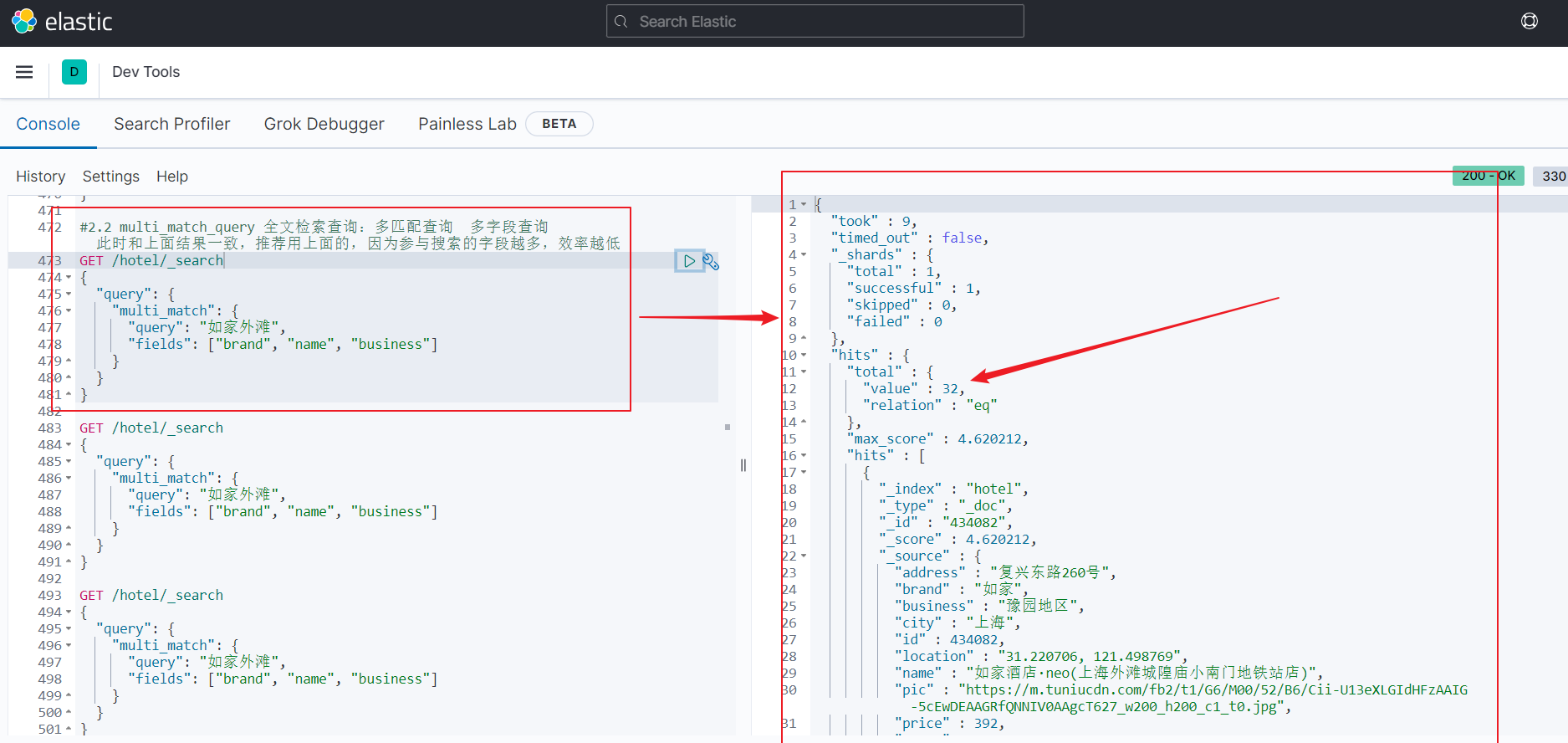
**}**

**}**

**}**

##### 在DevTools控制台测试multi\_match查询：

你看你看，查询到32条相关文档，并且查询结果中，相关度高的排在前面呢！很符合我们全文检索的需求！此时和上面match查询结果一致，因为上面演示的match查询虽然只搜一个“all”字段，但是我们“brand”、“name”、“business”这三个字段copy\_to到“all”这个复合字段中去了，效果是一样的！推荐用上面的match搜索，因为参与搜索的字段越多，效率越低！就是说你不要整多个字段去查，而是想办法把多个字段拷到一个字段里去查！可以理解吧



#### 总结：match查询和multi\_match查询的区别是什么？

Match：根据一个字段查询；

Multi\_match：根据多个字段查询，参与查询的字段越多，查询性能越差！

## 精确查询：term & range

精确查询一般是查找keyword、数值、日期、boolean等类型字段。这些类型的共同特点就是这些值是不可分割的整体。所以我们**在做搜索的时候，对搜索的条件有要求：查询的条件也不需要分词**！而且搜素条件要跟查找的字段值完全匹配，所以叫精确查询！

精确查询不会对搜索条件分词。这些字段的特点是它们的值是不可分割的整体，所以在搜索的时候，对搜索的条件有要求，就是这个查询的条件也不需要分词，而且搜索条件要跟查找的字段值完全匹配，所以叫做精确查询。因为精确查询的字段搜是不分词的字段，因此查询的条件也必须是不分词的词条。查询时，用户输入的内容跟自动值完全匹配时才认为符合条件。如果用户输入的内容过多，反而搜索不到数据。

例如我们hotel索引库中的city、brand、score、starName、price这些字段都是精确值，那么就都可以进行精确查询。

### 精确查询种类

精确查询常见的有两种：

#### term查询：根据词条精确值查询。就是精确匹配，搜索内容和文档字段要一模一样！

#### range查询：根据值的范围查询

这两种精确查询选那种要看具体的业务！

### term词条查询

#### 通用语法：

**GET /索引库名/\_search**

**{**

**“query”: {**

**“term”: {**

**“FIELD”: {**

**“value”: “VALUE”**

**}**

**}**

**}**

**}**

#### DSL语句如下

**GET /hotel/\_search**

**{**

**"query": {**

**"term": {**

**"city": {**

**"value": "北京"**

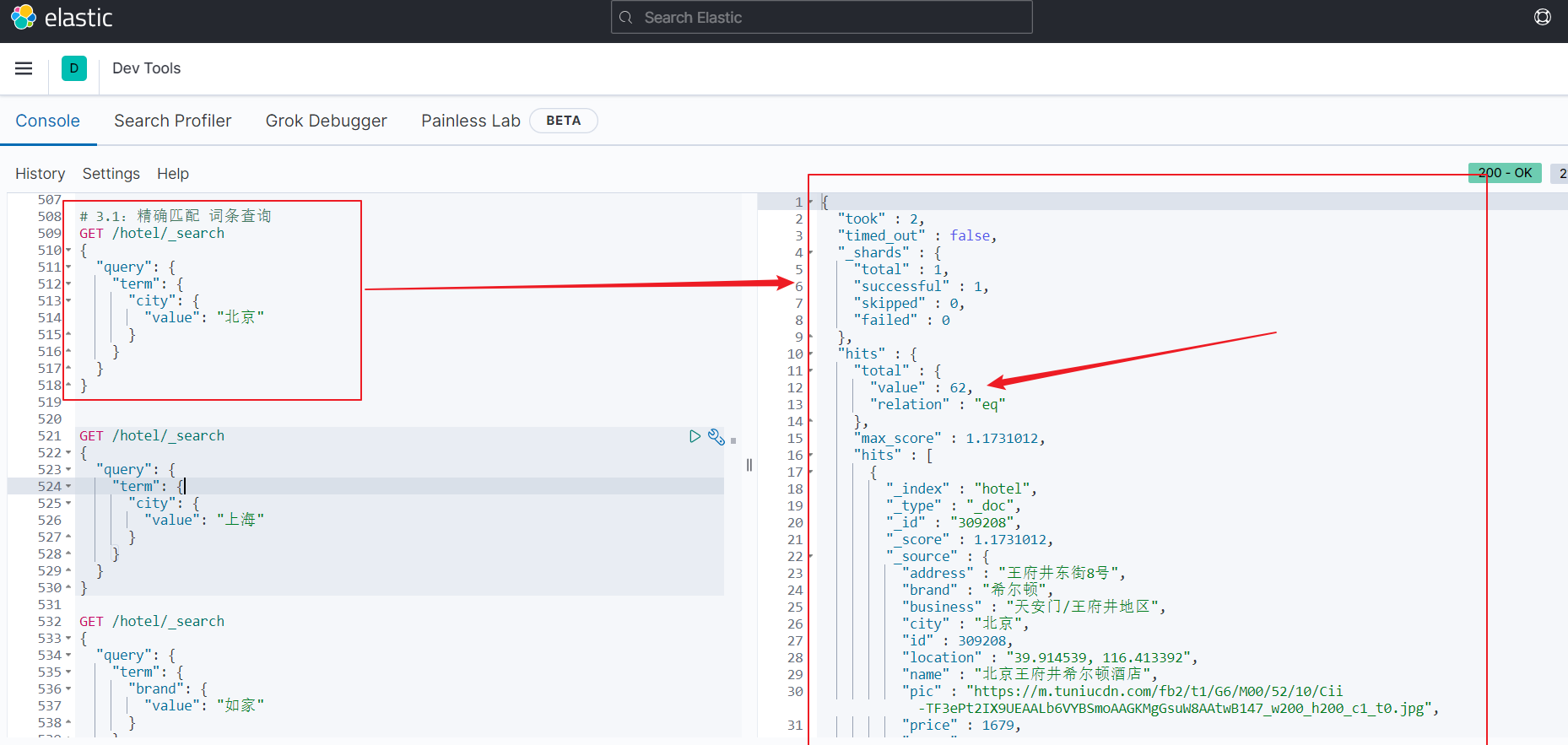
**}**

**}**

**}**

**}**

#### DevTools控制台查看搜索结果



### range范围查询

#### 通用语法

**GET /索引库名/\_search**

**{**

**“query”: {**

**“range”: {**

**“FIELD”: {**

**“gte”: “value”,**

**“lte”: “value”**

**}**

**}**

**}**

**}**

#### DSL语句如下：

**GET /hotel/\_search**

**{**

**"query": {**

**"range": {**

**"price": {**

**"gte": 1000,**

**"lte": 3000**

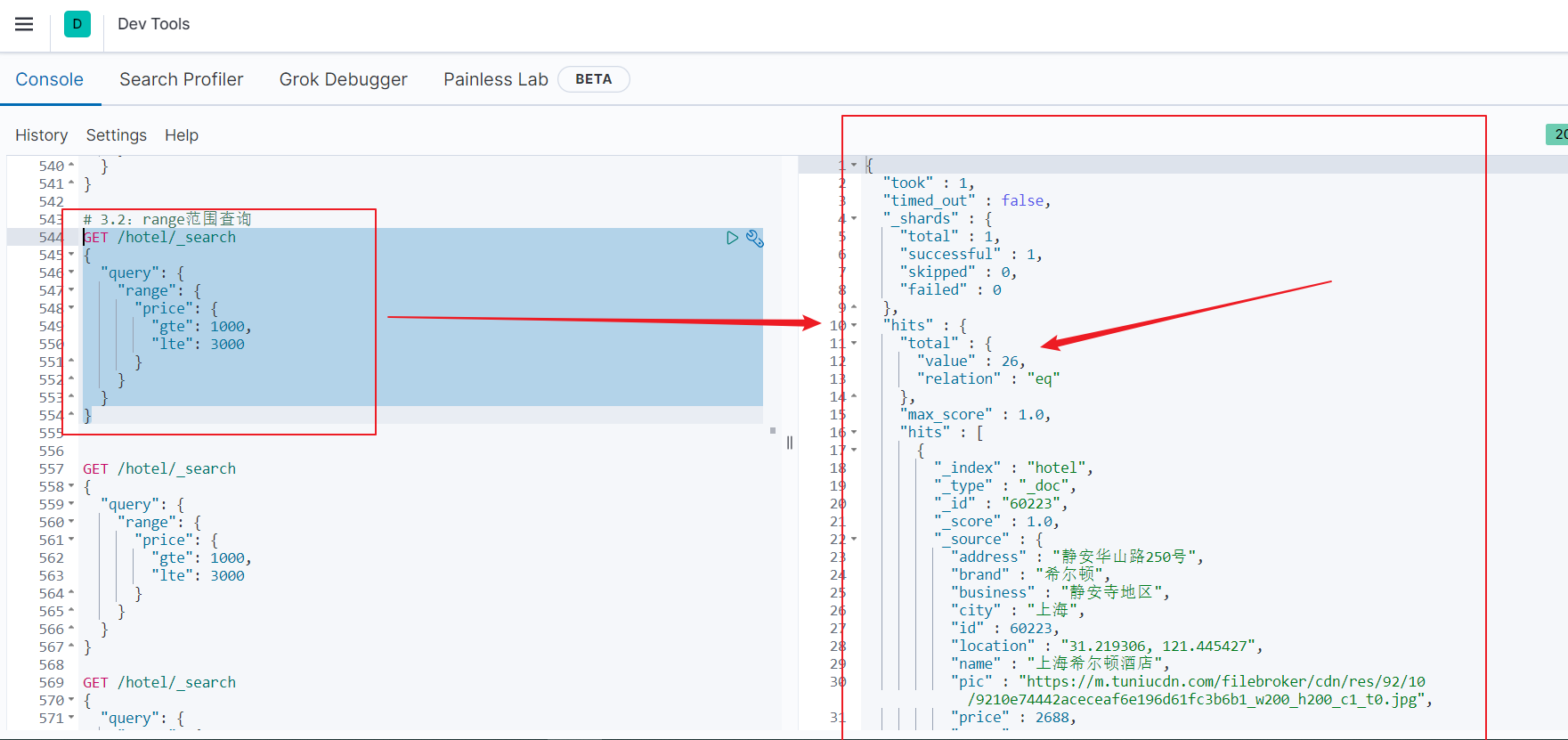
**}**

**}**

**}**

**}**

#### DevTools查询搜索结果



### 总结：精确查询常见的有哪些？

Term查询：根据词条精确值匹配，一般搜索keyword类型、数值类型、布尔类型、日期类型字段；

Range查询：根据数值范围查询，可以是数值、日期的范围

## 地理查询：geo\_point & geo\_shape

地理查询根据经纬度查询文档，常见的使用场景包括：

携程：搜索我附近的酒店

滴滴：搜索我附近的出租车

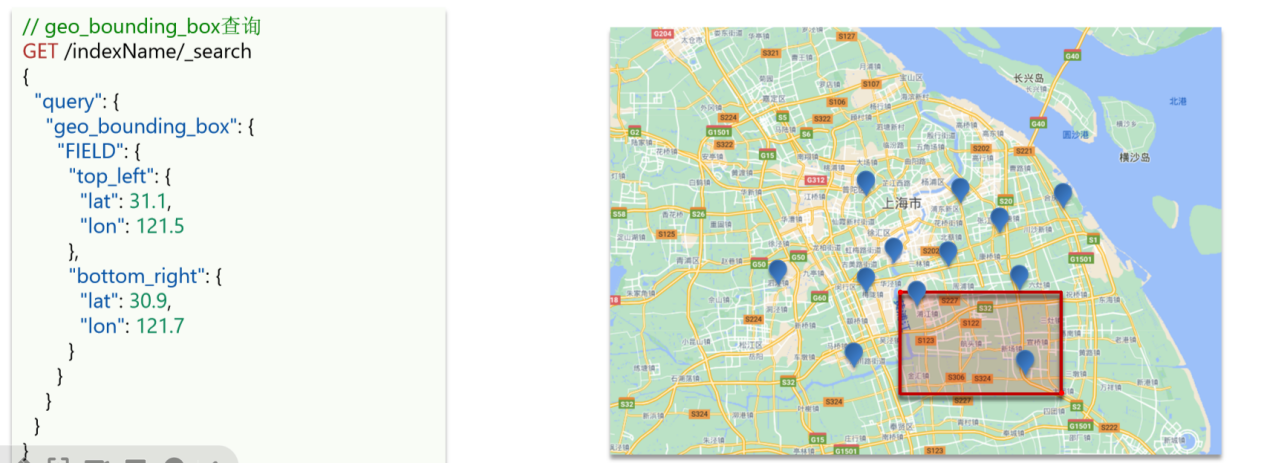
微信：搜索我附近的人

美团单车：搜索我附近的单车

以上都可以使用ES中的地理查询，地理查询常见的：

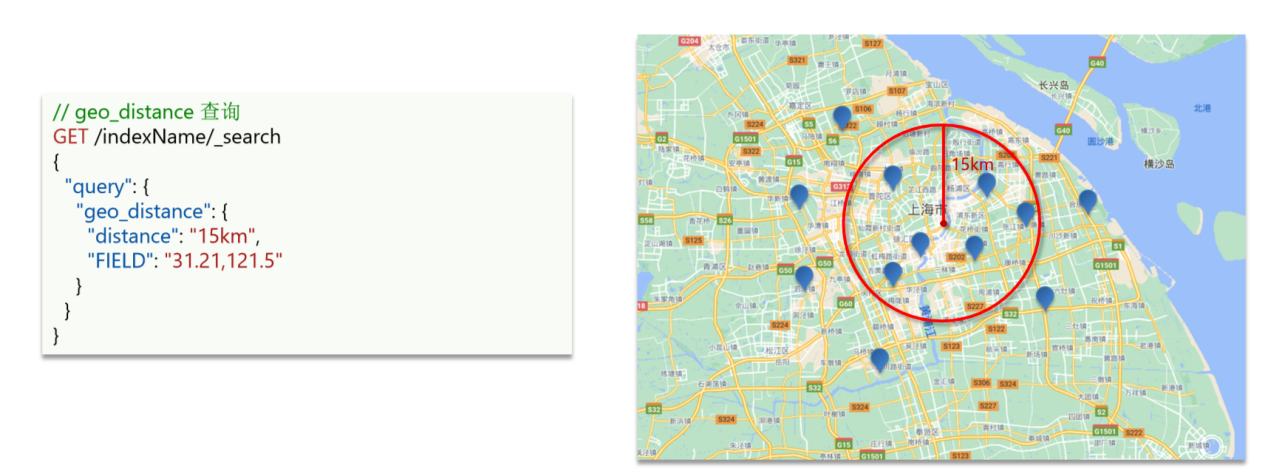
### geo\_bounding\_dox：

查询geo\_point地理坐标点值落在某矩形范围内的所有文档。如下图：图中的FIELD就是你文档中类型为geo\_point类型的字段！我们的酒店hotel索引库中的location字段的类型就是geo\_point！



### geo\_distance:

查询的是到指定中心点小于某个距离值得所有文档，就是我们理解的附近的感觉



### 演示geo\_bounding\_bos

#### 通用语法：

**GET /索引库名/\_search**

**{**

**“query”: {**

**“geo\_bounding\_box”: {**

**“top\_left”: {**

**“lat”: “”,**

**“lon”: “”**

**},**

**“bottom\_right”： {**

**“lat”: “”,**

**“lon”: “”**

**}**

**}**

**}**

**}**

#### DSL语句如下：

**GET /hotel/\_search**

**{**

**"query": {**

**"geo\_bounding\_box": {**

**"location": {**

**"top\_left": {**

**"lat": 31.1,**

**"lon": 121.5**

**},**

**"bottom\_right": "30.9,121.7"**

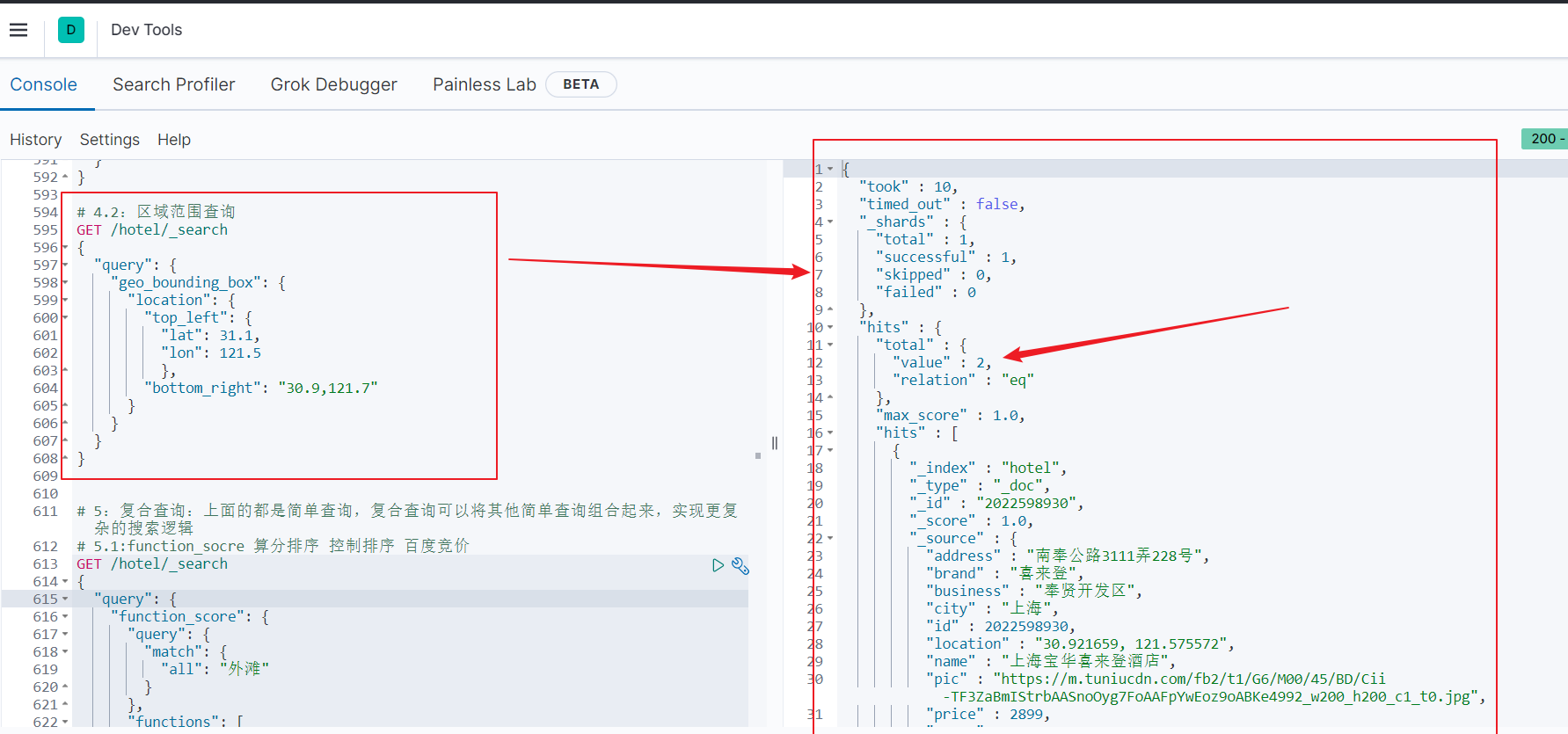
**}**

**}**

**}**

**}**

#### DevTools控制台查看搜索结果：



### 演示geo\_distance

#### 通用语法

**GET /索引库名/\_search**

**{**

**“query”: {**

**“geo\_distance”: {**

**“distance”: “半径距离值”,**

**“FIELD”: “地理坐标geo\_point值”**

**}**

**}**

**}**

#### DSL语句如下：

**GET /hotel/\_search**

**{**

**"query": {**

**"geo\_distance": {**

**"distance": "10km",**

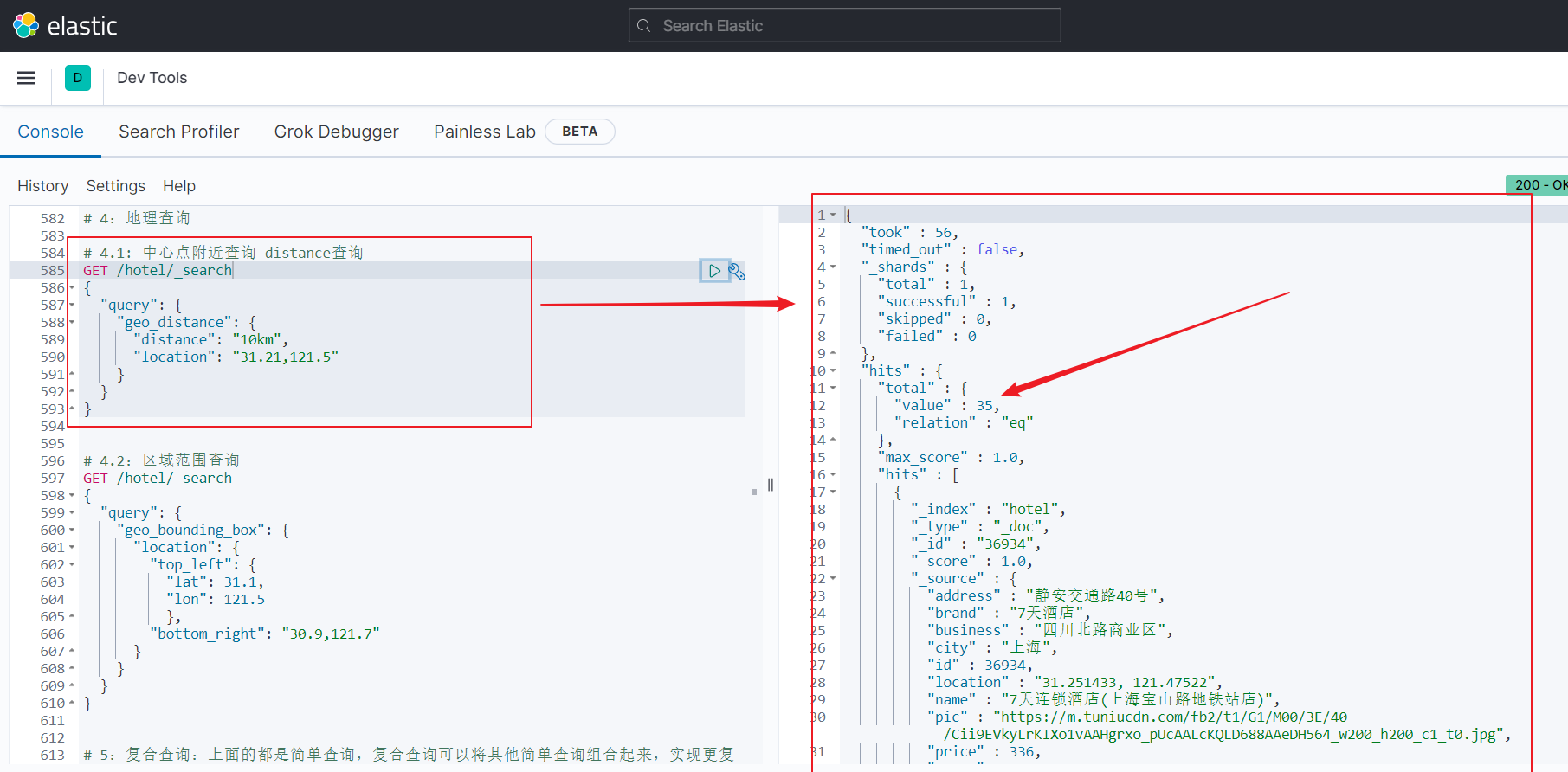
**"location": "31.21,121.5"**

**}**

**}**

**}**

#### DevTools控制台查看搜索结果



## 复合查询

朋友们，前面的查询所有、全文检索查询、精确查询、地理查询，这些查询在ES中都称为简单查询，与简单查询相对的是复合（compound）查询！

复合查询可以将其他简单查询组合起来，实现更复杂的搜索逻辑，即在简单查询的基础上去做一些功能的增加，去实现更复杂的搜索，或者是其他的功能。复合查询常见的有：

### function\_score：算分函数查询

#### 什么是相关性？

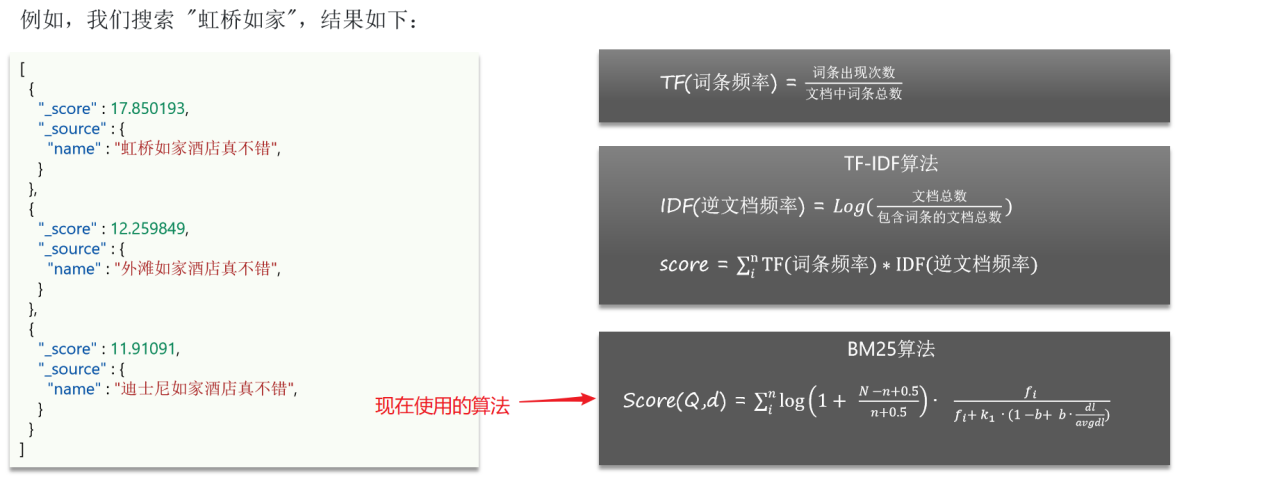
这种查询可以在原有的查询的基础上，去控制文档的相关性算分，控制文档排名，什么是相关性算分呢？**比如我们之前去搜索”外滩如家”这样的酒店信息，我们发现同时包含”外滩”和”如家”这样的酒店排名比较靠前，而只包含”外滩”或者“如家”的酒店排名就会在后面，这就是相关性，ES会对相关性进行打分，相关度越高，打分越高，排名也就越靠前**，但是，在实际中，需求往往不是如此，例如百度竞价就允许搜索结果排名第一的是广告，就是人工对搜索排名进行了干预，要想实现这种干预，就得使用function\_score。

怎么用function\_score去影响相关性打分呢？首先要了解一下ES中打分的算法：例如，当我们利用match\_all查询时，文档结果会根据与搜索词条的关联度打分（\_score），返回结果时按照分值降序排列。关联度高的文档排在前面！

#### Elasticsearch中的相关性打分算法是什么？

TF-IDF算法：在ES5.0之前，会随着词频增加而越来越大。

BM25算法：在ES5.0之后，会随着词频增加而增大，但增长曲线会趋于水平。



#### 如何用function\_score来控制打分？

使用function\_socre查询，可以修改文档的相关性算分（query\_score），根据新得到的算分排序。

经过上面的学习我们知道，ES在搜索文档的时候，会对文档做相关性打分，文档与搜索关键字的相关度越高，那么打分就越高，排名就会越靠前！不过有的时候，我们希望人为地去控制文档的排名，如竞价排名，他给了钱，我就要让他想关性算分高一点，排名靠前一点！这个时候，就要用到function\_score查询了，这种查询可以在原始的相关性算分的基础上加以修改，得到你想要的算分，从而去影响文档的排名！

#### function\_socre查询语法案例：

**GET /hotel/\_search**

**{**

**“query”: {**

**“function\_score”: {**

**“query”: {“match”: {“all”: “外滩”}},**

**“functions”: [**

**{**

**“filter”: {“term”: {“id”: “1”}},**

**“weight”: 10**

**}**

**],**

**“boost\_mode”: “multiply”**

**}**

**}**

**}**

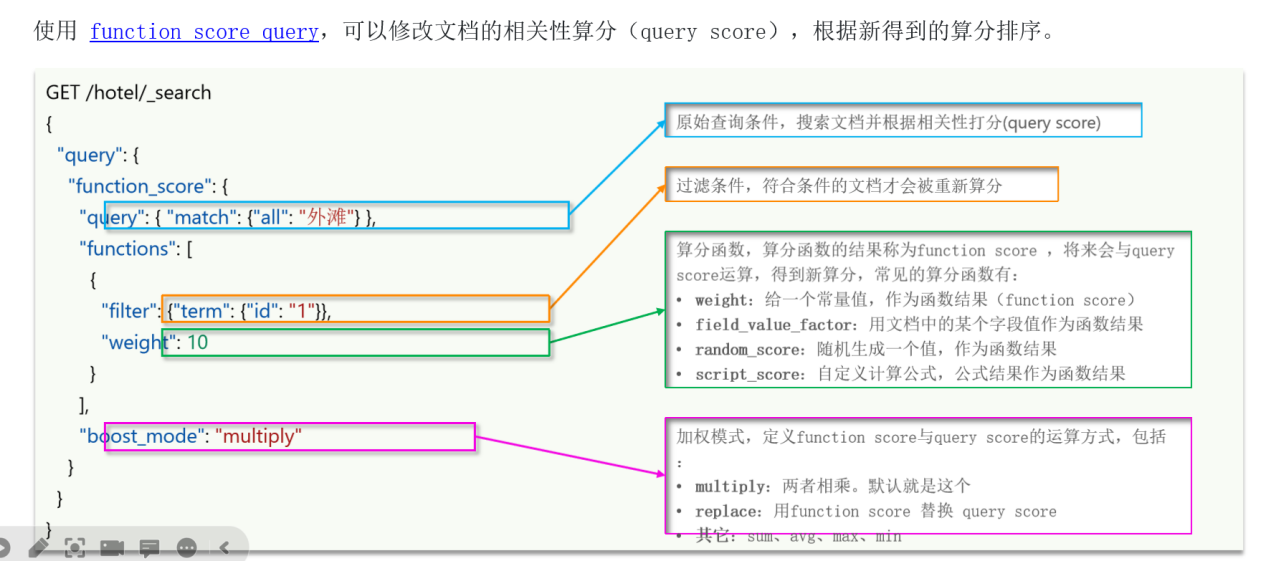
**function\_score**：是查询类型；

**query**：是原始查询条件，根据用户所想搜索的条件去进行查询，根据原始搜索条件搜索出来的文档默认会根据相关性打分（query score）；

**filter**：是过滤条件，符合条件的文档才会被重新算分，也就是说不会所有搜索出来的文档都会做重新算分处理，部分文档；

**weight**：是算分函数，算分函数的结果称为function\_score，将来会与query score运算，得到新算分；

**boost\_mode**：是加权模式，即function\_score与query score怎么去运算。



#### **使用function\_score查询：让“如家”酒店的搜索排名靠前一些：** **GET /hotel/\_search**

**{**

**“query”: {**

**“function\_score”: {**

**“query”: {**

**“match”: {**

**“all”: “外滩”**

**}**

**},**

**“functions”: [**

**{**

**“filter”: {**

**“term”: {**

**“brand”: “如家”**

**}**

**},**

**“weight”: 2**

**}**

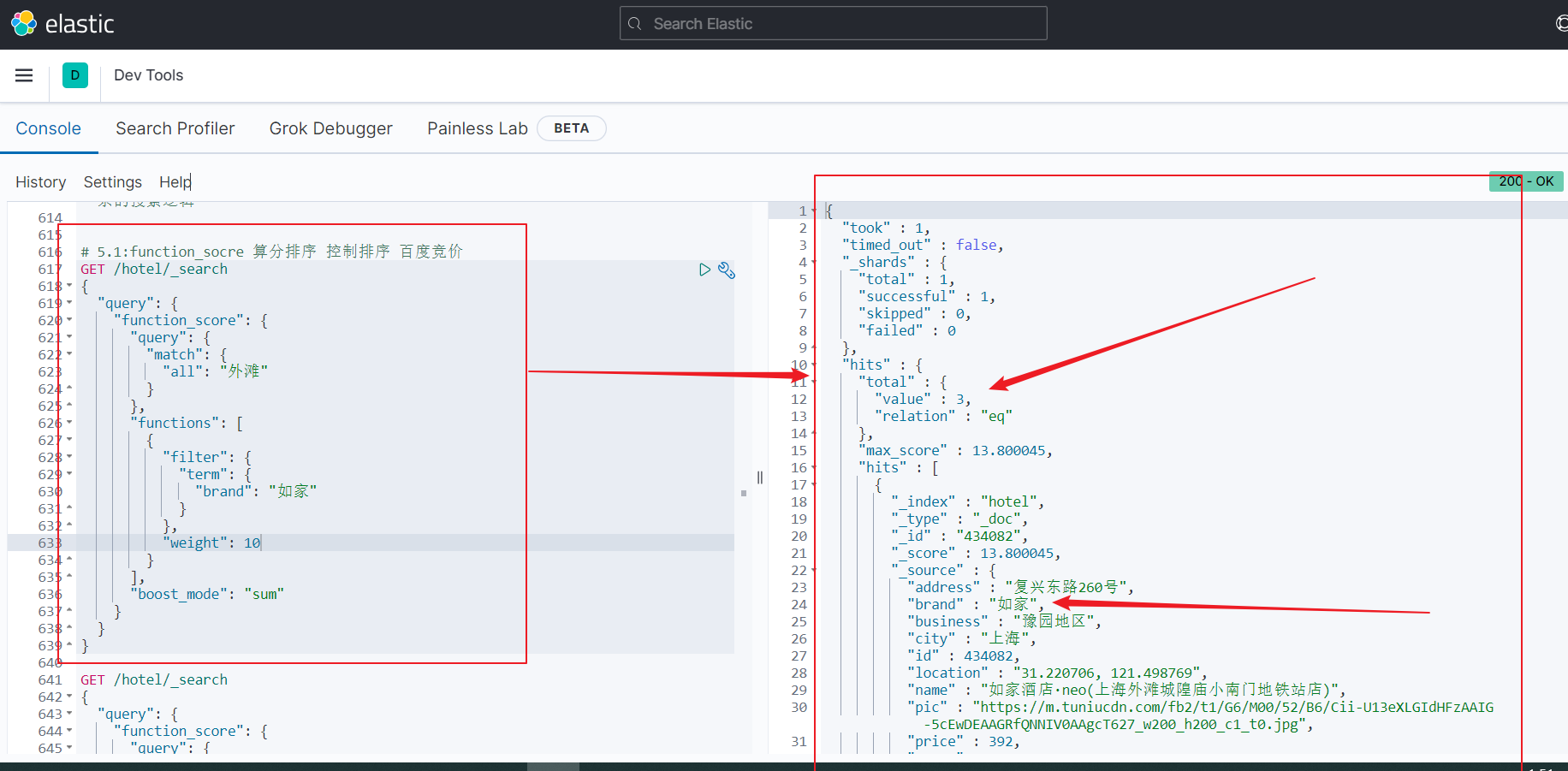
**],**

**“boost\_mode”: “sum”**

**}**

**}**

**}**



#### 总结：function score query定义的三要素是什么？

1：过滤条件filter：哪些文档重新计算相关性得分；

2：算分函数weight：定义如何计算function score

3：加权方式boost\_mode：function score与query core如何运算

### Boolean Query：布尔查询

这种复合查询与上面的算分函数查询不同之处是，算分函数查询主要是在原始查询的基础上去修改一下文档的算分，而**boolean查询则是把多个查询语句组合在一起形成新查询**，这些被组合的查询语句叫做子查询，子查询的组合方式有四种：

#### 子查询组合方式

**1：must：必须匹配每个子查询，类型“与”；**

**2：should：选择性匹配子查询，类似“或”；**

**3：must\_not：必须不匹配，不参与算分，类型“非”；**

**4：filter：必须匹配，不参与算分**

注意，filter从字面来看是必须匹配，和must很像！但是filter底层和must是不一样的，must\_not也一样，filter、must\_not的特点是在搜索过滤的时候是不参与算分的，意思就是大家都知道ES默认在做搜索的时候，不仅仅去判断文档是否匹配，还要去看文档与关键字之间的相关度，要打分，分值越高的文档越靠前，这个打分我们有非常复杂的算分函数（BM25算法），每做一次算分都会消耗资源，如果子查询比较多，每一个都差与算分，那么查询性能就会受到影响。

**而filter、must\_not是不参与算分的，只会返回满足或不满足，是或者否，因此这种查询就不需要做算分，性能就会很好，又因为filter、must\_not查询结果是简单的是或否**，ES把这种过滤性的插叙放到缓存中去，将来再次查询的时候会进一步去查询性能。再说一遍，filter和must\_not返回的是是或者否！

因此，虽然我们说Boolean组合查询中有四种方式，但是除了跟算分相关的，一般就是关键字嘛，用户输入的关键字，即**除了关键字应该放到must和should里面，剩下的过滤条件都应该放到filter或者是must\_not里面，尽可能地减少算分，提高查询效率**。

#### 需要用到Boolean查询的场景：

比如你去搜索酒店信息，首先在搜索框里输入填写一些酒店信息，比如酒店名称之类的，而后你还可以去点击下面的过滤条件吧，比如价格范围要求在100-300之间，同时要求地址在上海，这个时候，你的搜索要求就是满足搜索框中的名字，并且酒店在上海，并且酒店价格在100到300之间，你看你看你看，是不是多个搜索组合了。这个时候啊，搜索框关键字的搜索，你可以放到must，因为你要参与算分啊，下面的过滤条件：品牌也好，城市也好、价格也好，。。。最好放到filter和nust\_not当中！因为不参与算分性能更好！



#### Boolean查询DSL语句示例

可以看到must、should、must\_not、filter都是数组，也就说在Boolean查询的每一个子查询里里面都可应定义多个子查询！以下的DSL语句的意思就是：

搜索酒店地址在上海的，酒店品牌是“皇冠假日”或者“华美达”的，同时酒店价格大于500，酒店评分大于45分的酒店。



#### 案例：利用bool查询实现功能

要求：搜索名字包含“如家”，价格不高于400，在坐标31.21,121.5周围10KM范围内的酒店！

DSL语句如下：

**GET /hotel/\_search**

**{**

**"query": {**

**"bool": {**

**"must": [**

**{**

**"term": {**

**"brand": {**

**"value": "如家"**

**}**

**}**

**}**

**],**

**"must\_not": [**

**{**

**"range": {**

**"price": {**

**"gt": 400**

**}**

**}**

**}**

**],**

**"filter": [**

**{**

**"geo\_distance": {**

**"distance": "10km",**

**"location": {**

**"lat": 31.21,**

**"lon": 121.5**

**}**

**}**

**}**

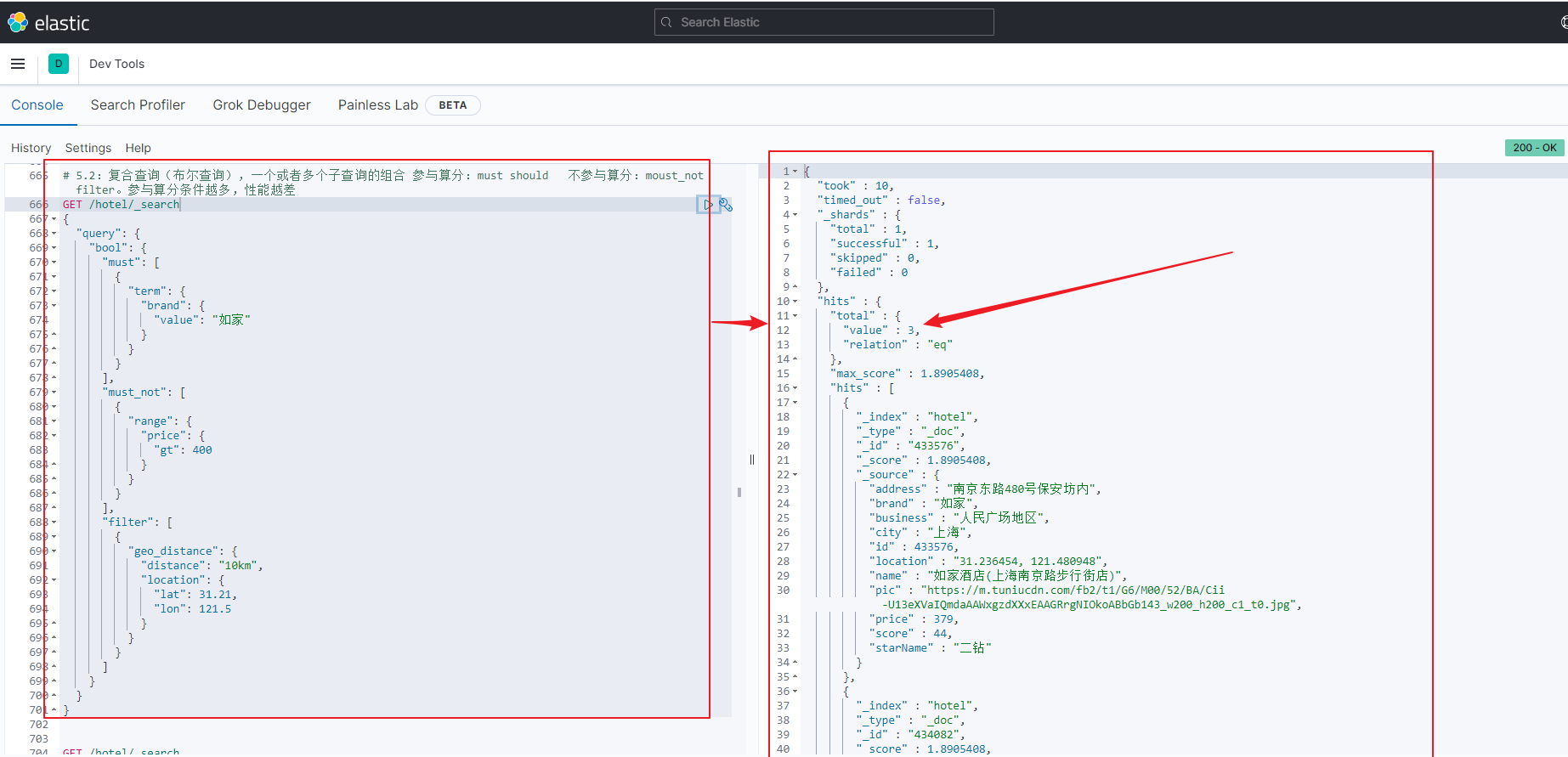
**]**

**}**

**}**

**}**

DevTools控制台查看搜索结果：



我们之前说了must\_not和filter不参与算分，我们可以验证一下：我们把filter中的geo\_distance移到must中去：

**GET /hotel/\_search**

**{**

**"query": {**

**"bool": {**

**"must": [**

**{**

**"term": {**

**"brand": {**

**"value": "如家"**

**}**

**}**

**},**

**{**

**"geo\_distance": {**

**"distance": "10km",**

**"location": {**

**"lat": 31.21,**

**"lon": 121.5**

**}**

**}**

**}**

**],**

**"must\_not": [**

**{**

**"range": {**

**"price": {**

**"gt": 400**

**}**

**}**

**}**

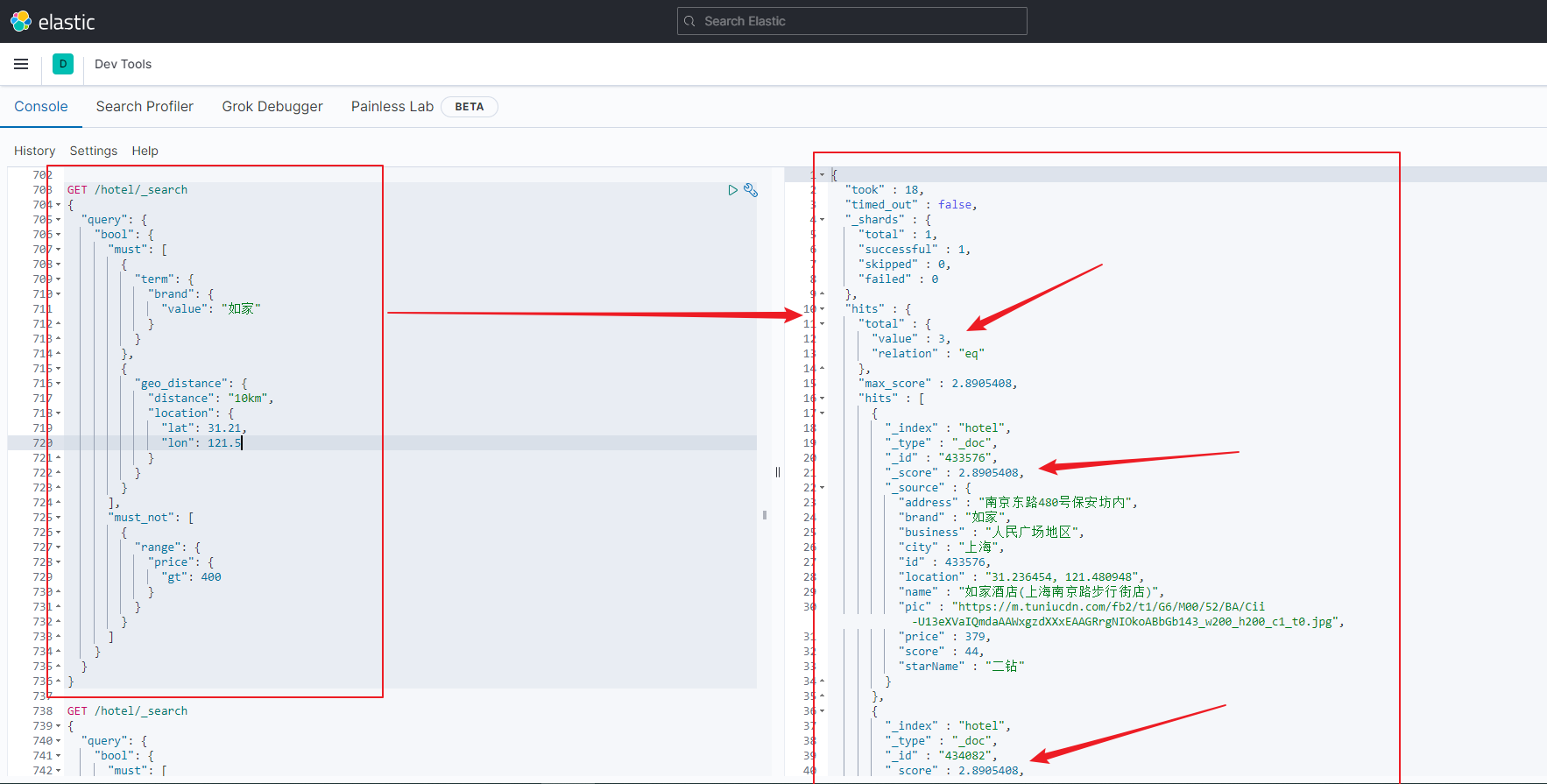
**]**

**}**

**}**

**}**

然后在DevTools控制台搜索数据，会发现搜索出来的文段的算分\_socre和之前不一样了，\_score值都变大了，证明filter和must\_notd都会影响算分！



#### 总结：boolean查询有几种逻辑关系？

must：必须匹配的条件，可以理解为“与”

should：选择性匹配的条件，可以理解为“或”

must\_not：必须不匹配的条件，不参与打分

filter：必须匹配的条件，不参与打分