**基本概念：Index、Document 和 REST API**

* Index & Document 是比較偏向開發人員視角，是種邏輯概念
* Node & Shard 是比較偏向維運人員的視角，是種物理概念

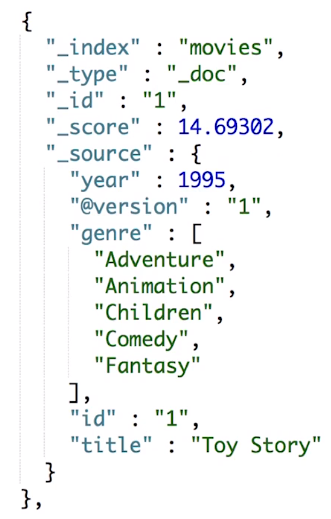
**Document**

* Document 是可以被搜尋數據的最小單位(可能是 log 文件中的一筆紀錄 / 一部電影或唱片的相關訊息 / RDBMS 中的一筆 record)
* Document 會被序列化成 JSON(由一堆 Key/Value 的資料組成，並有其資料格式) 格式，保存在 Elasticsearch 中
* 每個 Document 都有一個 UID(Unique ID)，可自己指定或交由 Elasticsearch 自動產生

**JSON document**

* 包含多個 Key/Value 組合，就像是資料庫中的一筆資料
* 但跟資料庫不一樣的是，JSON 格式靈活不受限，不須預先定義格式
* 每個 Key/Value 的類型(string, number, boolean … etc) 可以自己指定或是由 Elasticsearch 幫忙推算

**Metadata**



document metadata 就是描述 document 本身屬性用的資料，通常會包含以下內容：

* \_index：document 所屬的 index 名稱
* \_type：document 類型 (例如：**\_doc**)
* \_id：document ID
* \_source：document 的原始 JSON 資料樣貌
* \_version：版本訊息 (有這欄位就表示 ES 具有版本控管的能力)
* \_score：查詢時的算分結果 (每次的搜尋都會根據 document 對於搜尋內容的相關度進行算分)

**Index**

* index 在 ES 中是個邏輯空間的概念，用來儲存 document 的容器，而這些 document 內容都是相似的 (跟其他領域的 index 用法不太一樣)
* shard 在 ES 中則是個物理空間的的概念，**index 中的資料會分散放在不同的 shard 中**
* index 由以下幾個部份組成：
  + data：由 document + metadata 所組成
  + mapping：用來定義每個欄位名稱 & 類型
  + setting：定義資料是如何存放(例如：replication 數量, 使用的 shard 數量)
* 下圖是 setting 的設定範例：  
  
* 在 ES 7.0 的版本後，index 在 type 部份只能設定為 \_doc (在以前的版本是可以設定不同的 type)

**Elasticsearch 與 RDBMS 的比較 & 取捨**

以下表格是 Elasticsearch & RDBMS 的對比：(不是完全符合，但概念上是很接近的)

| **RDBMS** | **Elasticsearch** |
| --- | --- |
| Table | Index |
| Row | Document |
| Column | Field |
| Schema | Mapping |
| SQL | DSL |

* ES 是 schemaless 的，資料格式可以隨意定，非常適合用來做全文檢索 or 查詢與資料的相關性
* RDBMS 的強項在於處理對於資料事務性(交易)要求特別高的任務

**常用搜尋**

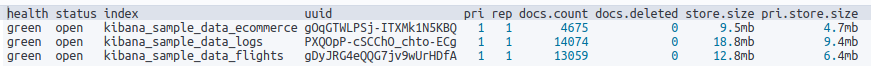
在 Kibana Dev Tools 頁面中，可以直接下查詢語法，以下舉幾個與 index 相關的搜尋：

**查詢 index**

|  |  |
| --- | --- |
| 1 2 3 4 5 | # 取得指定 index 資訊，包含 mapping & setting ... 等資訊 GET kibana\_sample\_data\_ecommerce  # 取得此 index 中的 document 數量 GET kibana\_sample\_data\_ecommerce/\_count |

**搭配 \_cat 做搜尋**

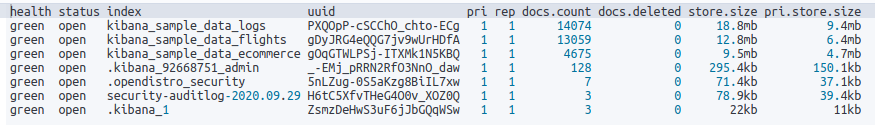
|  |  |
| --- | --- |
| 1 2 | # 透過 \_cat 查詢 index 相關資訊，搭配正規表示式  GET /\_cat/indices/kibana\*?v |



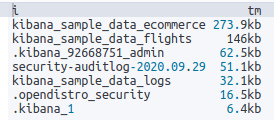
|  |  |
| --- | --- |
| 1 2 | # 加上過濾條件 GET /\_cat/indices/kibana\*?health=green |

Elasticsearch - _cat search 2

|  |  |
| --- | --- |
| 1 2 | # 使用排序 GET /\_cat/indices?v&s=docs.count:desc |



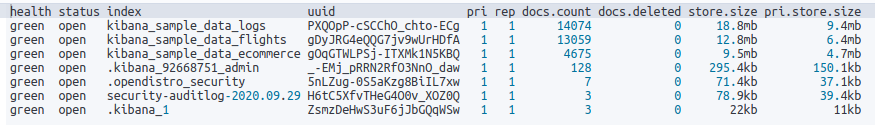
|  |  |
| --- | --- |
| 1 2 | # 查詢每個 index 所消耗的 memory 為多少，搭配排序 GET /\_cat/indices?v&h=i,tm&s=tm:desc |



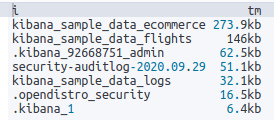
|  |  |
| --- | --- |
| 1 2 | # 加上過濾條件 GET /\_cat/indices/kibana\*?health=green |

Elasticsearch - _cat search 2

|  |  |
| --- | --- |
| 1 2 | # 使用排序 GET /\_cat/indices?v&s=docs.count:desc |



|  |  |
| --- | --- |
| 1 2 | # 查詢每個 index 所消耗的 memory 為多少，搭配排序 GET /\_cat/indices?v&h=i,tm&s=tm:desc |



**Document 的基本 CRUD 與批次操作**

**Document CRUD**

* GET：取得 document
  + 語法為 [GET \_index/\_type/[ID]](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-get.html)，例如 **GET /users/\_doc/1**
  + document 會有 version control 的功能，因此即使被刪除，version 欄位的值也會不斷增加
  + \_source 欄位包含了 document 的原始訊息



* Create(**PUT**)：
  + 建立新的 document，如果 ID 已經存在會發生錯誤
  + 語法為 PUT \_index/\_create/[ID] or PUT \_index/\_doc/[ID]?op\_type=create，例如：**PUT /users/\_create/1** (也可以不帶 ID，就會自動生成)
  + **較不建議指定 ID 的作法，可能會撞到效能不彰的問題**
* Create(**POST**)
  + 系統會自動產生 document ID (**這是比較建議的方式**)
  + 語法為 POST \_index/\_doc
* Index(**PUT**)：
  + 如果 ID 不存在，則建立新的 document；若 ID 已經存在，則刪除現存的 document 再建立新的，**version** 的部份會增加
  + 語法為 [PUT \_index/\_doc/[ID]](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index_.html)，例如：**PUT /users/\_doc/1**
* Update(**PUT**)：
  + PUT 其實也可以作為更新 document 用，但更新的範圍是整個 document
  + 實際上，Elasticsearch document 是無法修改的；而更新這個操作其實是新增一個新的 document，將原有的 **\_version** 加 1 後，舊的 document 被標示為 **deletion**
* Partially Update(**POST**)：
  + document 必須已經存在，更新時只會對 document 中相對應的欄位作增量更新 or 對應欄位的修改
  + json payload 需要包含在 doc 欄位中 (可參考[官網文件](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-update.html" \t "_blank))
  + 語法為 [POST \_index/\_update/[ID]](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-update.html)，例如：**POST /users/\_update/1**
  + POST 也可以拿來作為新增 document 用
* 呼叫 API 時傳輸的數據不宜過大(預設單一個 document 大小不能超過 100MB)，過大的 document 建議拆成 5~15MB 分次匯入

**批次操作**

* 批次操作基本上是用來提昇 API 呼叫時的效能
* 但每次的 API request 不要發送過多的資料，因為過多的資料可能會造成 ES cluster 過大的壓力導致效能下降(必須記得 ES cluster 每秒都需要服務相當多的 API request)

Elasticsearch 中支援幾種批次操作 API，常用的有以下幾個：

[**/\_bulk**](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html)

|  |  |
| --- | --- |
| 1 2 3 | POST /\_bulk  POST /<target>/\_bulk |

* 讓使用者可以在同一個 API request 中送出多個操作，支援 **Index/Create/Update/Delete**，提昇效率
* request 中的每一筆資料都會有對應的 return code，其中的任何一個操作失敗不會影響其他操作

範例如下：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | POST \_bulk { "index" : { "\_index" : "test", "\_id" : "1" } } { "field1" : "value1" } { "delete" : { "\_index" : "test", "\_id" : "2" } } { "create" : { "\_index" : "test", "\_id" : "3" } } { "field1" : "value3" } { "update" : {"\_id" : "1", "\_index" : "test"} } { "doc" : {"field2" : "value2"} } |

[**/\_mget**](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-multi-get.html)

|  |  |
| --- | --- |
| 1 2 3 | GET /\_mget  GET /<index>/\_mget |

* 一次讀取多個不同 index 中特定 ID 的 document

使用範例：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | GET /\_mget {  "docs": [  {  "\_index": "my-index-000001",  "\_id": "1"  },  {  "\_index": "my-index-000001",  "\_id": "2"  }  ] } |

**[/\_msearch]**

|  |  |
| --- | --- |
| 1 | GET /<target>/\_msearch |

* [官方文件說明](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-multi-search.html)
* 一次作多個大範圍的搜尋

使用範例：

|  |  |
| --- | --- |
| 1 2 3 4 5 | GET my-index-000001/\_msearch { } {"query" : {"match" : { "message": "this is a test"}}} {"index": "my-index-000002"} {"query" : {"match\_all" : {}}} |

**其他注意事項**

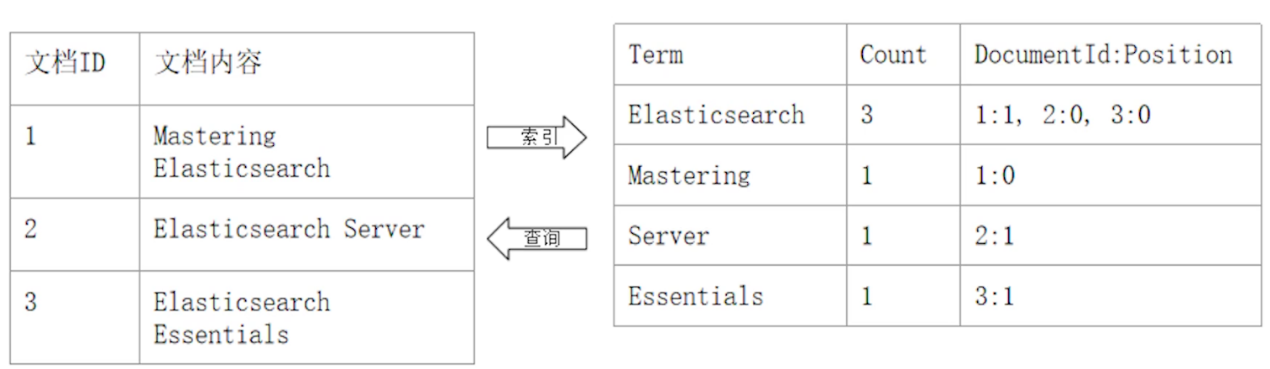
* 大版本的升級，document 必須重建 index
* Elasticsearch 預設會提供 dynamic mapping 的功能，因此不用預先設定好 index 結構；但在生產環境中，建議先做好 mapping 的設定後再寫入資料
* 透過 X-Pack security plugin，可以提供 index-level or field-level 的 role based 資料存取控制

**API 行為區分**

* index： 針對整個文檔，既可以新增又可以更新；
* create：只是新增操作，已有報錯，可以用 PUT 指定 ID，或 POST 不指定 ID；
* update：指的是部分更新，官方只是說用POST，request body 裡用 script 或 doc 裡包含文檔要更新的部分；
* delete 和 read：就是 delete 和 get 請求了，比較簡單

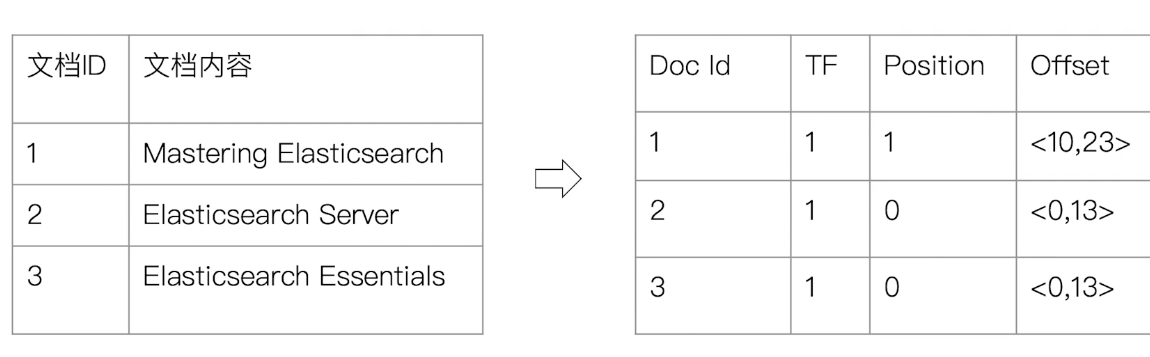
**Inverted Index(倒排索引)介紹**

* Forward Index(正排索引)：Document ID 到 Document 內容到單詞的關聯
* Inverted Index(倒排索引)：單詞到 Document ID 的關係



**Inverted Index 組成**

* Term Dictionary (單詞辭典)
  + 為了滿足快速的插入 & 查詢，且因為 term 的數量龐大，因此通過 **B+ tree** or **Open Hashing** 的方式實現
  + 記錄 Document 中所有的單詞，記錄單詞到 posting list(倒排列表) 的關聯關係
* Posting List (倒排列表)：由 posting(倒排索引項組合) 組成，包含以下內容：
  + Document ID
  + 詞頻 (Term Frequency)：term 在 document 中出現的次數，用在相關性評分
  + 位置 (Position)：term 在 document 的位置，用在搜尋
  + 偏移 (Offset)：記錄 trem 開始 & 結束位置，用於高亮顯示



* Elasticsearch 的 JSON document 中的 term 都會有自己的倒排索引
* 可以對某些欄位不作索引：
  + 優點：節省儲存空間
  + 缺點：該欄位無法被搜尋

**References**

* [倒排索引 | Elasticsearch: 權威指南 | Elastic](https://www.elastic.co/guide/cn/elasticsearch/guide/current/inverted-index.html)

**通過 Analyzer 進行分詞**

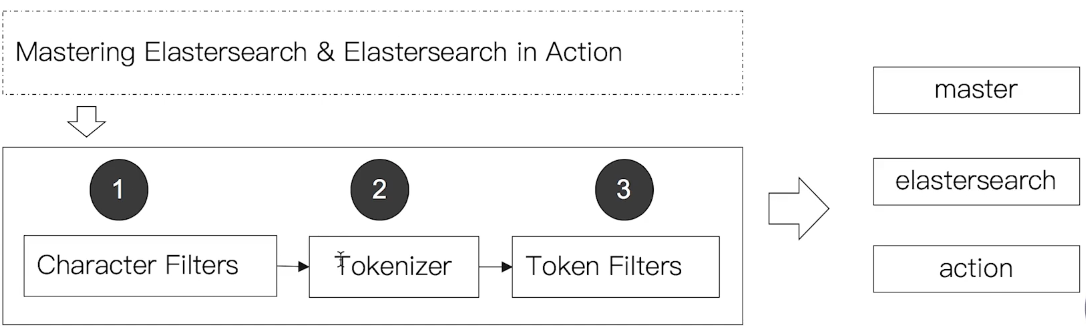
* Analysis 是將 document 的內容轉換為一系列單詞(term/token) 的過程，也叫**分詞**
* Analysis 是由 Analyzer 來實現，Analyzer 由 Character Filter -> Tokenizer -> Token Filter 三個部份所組成，每個部份都可以自訂

**Analyzer 的組成**

Analyzer 是專門處理分詞的組件，由三個部份組成：

* Character Filter：針對原始文件進行處理，例如：去除 HTML tag
* Tokenizer：根據規則切分 term
* Token Filter：將分割後的 term 進行加工，例如：轉小寫、刪除 [stopwords](https://zh.wikipedia.org/wiki/%E5%81%9C%E7%94%A8%E8%AF%8D" \t "_blank)、增加同義詞、stemming(例如將 box, boxed, boxing … 等字轉換成 box)

有時後會希望不要過濾 stopwords，而是直接把內容當成完整的 phrase 看待 (例如：**to be or not to be**)



* Elasticsearch 內建很多 Analyzer，每個 Analyzer 會由不同的 character filter, tokenizer, token filter 組合而成，使用者也可以自訂 Analyzer

**Elasticsearch 內建的 Analyzer**

以下是幾個 Elasticsearch 內建的 Analyzer，有興趣的人可以試試看每個 Analyzer 處理完後資料的效果：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 | # character filter: standard (按詞切分) # tokenizer: 轉小寫 # token filter: N/A (stopword 會保留) GET \_analyze {  "analyzer": "standard",  "text": "2 running Quick brown-foxes leap over lazy dogs in the summer evening." }  # character filter: simple (按詞切分，去除非字母字元) # tokenizer: 轉小寫 # token filter: N/A (stopword 會保留) GET \_analyze {  "analyzer": "simple",  "text": "2 running Quick brown-foxes leap over lazy dogs in the summer evening." }  # character filter: simple (按詞切分，去除非字母字元) # tokenizer: 轉小寫 # token filter: 去除 stopwords GET \_analyze {  "analyzer": "stop",  "text": "2 running Quick brown-foxes leap over lazy dogs in the summer evening." }   # character filter: whitespace (單純以空白切分) # tokenizer: 轉小寫 # token filter: N/A (stopword 會保留) GET \_analyze {  "analyzer": "whitespace",  "text": "2 running Quick brown-foxes leap over lazy dogs in the summer evening." }  # character filter: N/A (不進行分詞) # tokenizer: keyword (將所有的輸入直接轉成 token) # token filter: N/A GET \_analyze {  "analyzer": "keyword",  "text": "2 running Quick brown-foxes leap over lazy dogs in the summer evening." }  # character filter: pattern (使用正規表示式分詞) # tokenizer: 轉小寫 # token filter: N/A (stopword 會保留) GET \_analyze {  "analyzer": "pattern",  "text": "2 running Quick brown-foxes leap over lazy dogs in the summer evening." }   # character filter: english (使用英文辭典分詞) # tokenizer: 轉小寫 # token filter: # token filter: 去除 stopwords GET \_analyze {  "analyzer": "english",  "text": "2 running Quick brown-foxes leap over lazy dogs in the summer evening." } |

**Search API 概覽**

* search API 分成 URI search(使用 GET) & request body search(同時支援 GET & POST，使用 ES 提供的 DSL)

查詢範圍：

| **語法** | **範圍** |
| --- | --- |
| /\_search | cluster 上所有的 index |
| /index1/\_search | index1 |
| /index1,index2/\_search | index1 + index2 |
| /index\*/\_search | 以 **index** 開頭的 index |

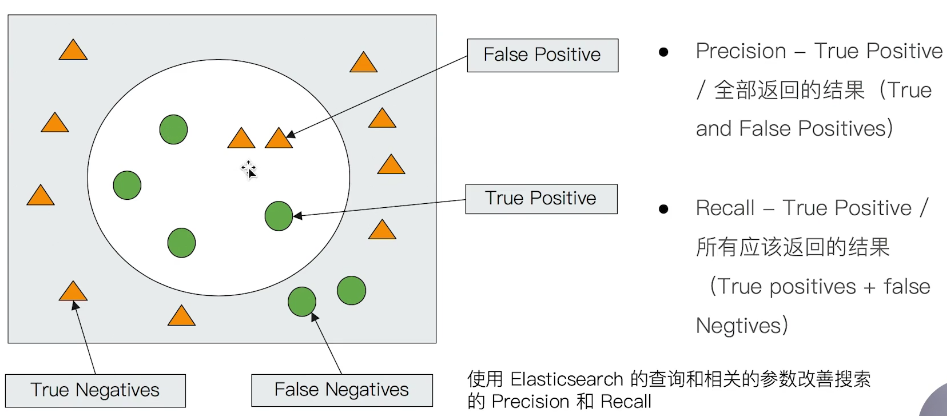
**搜尋與相關性**

* 搜尋行為是客戶對於搜尋引擎的操作 & 互動
* 客戶關心的是**搜尋結果的相關性**：
  + 是否可以找到相關的內容?
  + 搜尋結果中包含了多少不相關的內容?
  + 搜尋結果的算分是否合力
  + 結合實際業務需求，平衡搜尋結果的排名

**衡量相關性**

Information Retrieval

* Precision (查準率)：盡可能返回較少的無關 document
* Recall (查全率)：盡量返回較多的相關 document
* Ranking：是否能夠依照相關度進行排序?



**URI Search詳解**

URI search 範例：

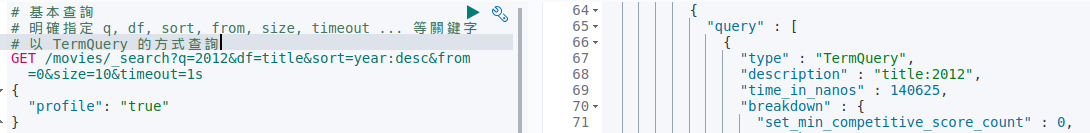
|  |  |
| --- | --- |
| 1 2 3 4 | GET /movies/\_search?q=2012&df=title&sort=year:desc&from=0&size=10&timeout=1s {  "profile": true } |

* q：指定查詢語句，使用 **Query String Syntax**
* df：預設查詢的 field，若不指定則會對所有的 field 進行查詢
* sort：排序
* from & size 用於分頁
* profile：可以檢查查詢是如何被執行的

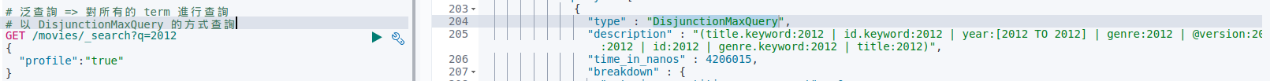
**搜尋範例**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | # 基本查詢 # 明確指定 q, df, sort, from, size, timeout ... 等關鍵字 # 以 TermQuery 的方式查詢 GET /movies/\_search?q=2012&df=title&sort=year:desc&from=0&size=10&timeout=1s {  "profile": "true" } |

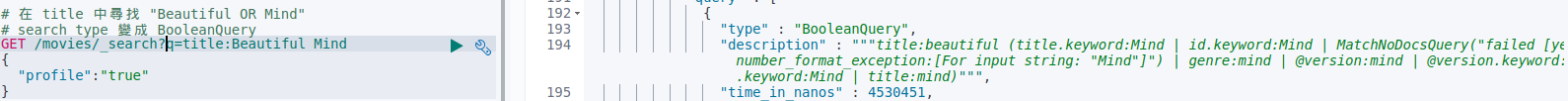
GET /movies/\_search?q=2012&df=title == GET /movies/\_search?q=title:2012



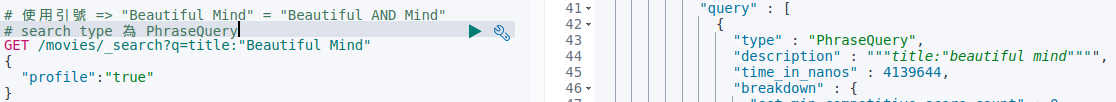
|  |  |
| --- | --- |
| 1 2 3 4 5 6 | # 泛查詢 => 對所有的 term 進行查詢 # 以 DisjunctionMaxQuery 的方式查詢 GET /movies/\_search?q=2012 {  "profile":"true" } |

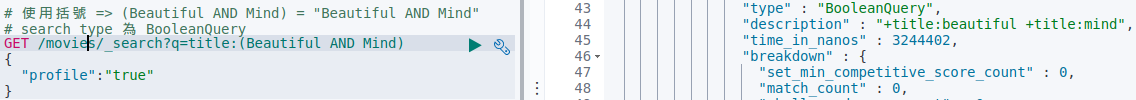


|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | # 在 title 中尋找 "Beautiful OR Mind" # 搜尋效果 => TermQuery(title:beautiful) + DisjunctionMaxQuery(在所有 term 中搜尋 mind) # search type = BooleanQuery GET /movies/\_search?q=title:Beautiful Mind {  "profile":"true" } |

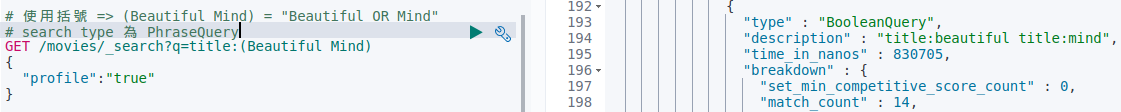


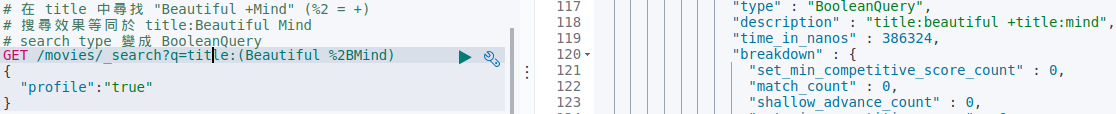
|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | # 使用引號 => "Beautiful Mind" = "Beautiful AND Mind" # search type 為 PhraseQuery (同時出現 & 按照順序) GET /movies/\_search?q=title:"Beautiful Mind" {  "profile":"true" }  # 使用括號 => (Beautiful AND Mind) = "Beautiful AND Mind" # search type 為 BooleanQuery GET /movies/\_search?q=title:(Beautiful AND Mind) {  "profile":"true" } |



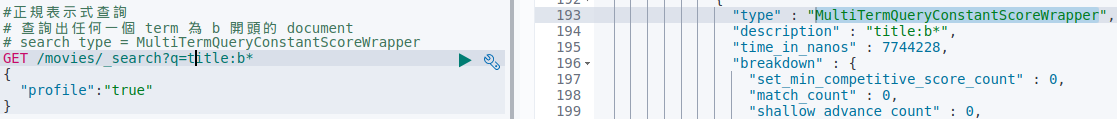


|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | # 使用括號 => (Beautiful Mind) = "Beautiful OR Mind" # 搜尋效果 => TermQuery(title:beautiful) + TermQuery(title:mind) # search type 為 BooleanQuery GET /movies/\_search?q=title:(Beautiful Mind) {  "profile":"true" }  # 在 title 中尋找 "Beautiful +Mind" (%2 = +) # 搜尋效果 => TermQuery(title:beautiful) + TermQuery(+title:mind) # mind 一定要出現，但 beautiful 不一定要出現 # search type = BooleanQuery GET /movies/\_search?q=title:(Beautiful %2BMind) {  "profile":"true" } |



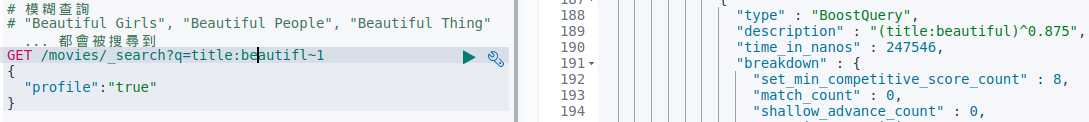


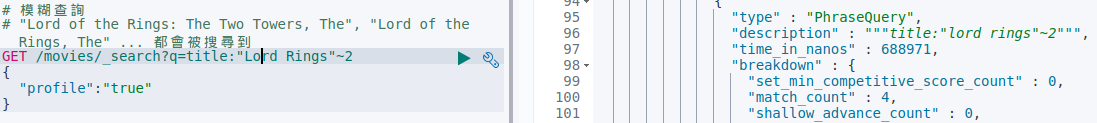
|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | #正規表示式查詢 # 查詢出任何一個 term 為 b 開頭的 document # search type = MultiTermQueryConstantScoreWrapper GET /movies/\_search?q=title:b\* {  "profile":"true" } |



查詢效率低，記憶體消耗大，不建議使用

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | # 模糊查詢 (即使 search term 有輸入錯誤，還是可以查詢) # "Beautiful Girls", "Beautiful People", "Beautiful Thing" ... 都會被搜尋到 GET /movies/\_search?q=title:beautifl~1 {  "profile":"true" }  # 模糊查詢 # "Lord of the Rings: The Two Towers, The", "Lord of the Rings, The" ... 都會被搜尋到 GET /movies/\_search?q=title:"Lord Rings"~2 {  "profile":"true" } |





**Request Body 與 Query DSL 簡介**

* 進階的查詢通常只能用 request body 的方式完成

**一般查詢範例**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 | # ignore\_unavailable=true，可以忽略嘗試訪問不存在的 index “404\_idx” 導致的錯誤 # 查詢 movies index # 開啟 profile POST /movies,404\_idx/\_search?ignore\_unavailable=true {  "profile": true,  "query": {  "match\_all": {}  } }  # 透過 from & size 達到分頁效果 POST /kibana\_sample\_data\_ecommerce/\_search {  "from":10,  "size":20,  "query":{  "match\_all": {}  } }  # 對資料排序，使用 sort 參數 POST kibana\_sample\_data\_ecommerce/\_search {  "sort":[{"order\_date":"desc"}],  "query":{  "match\_all": {}  }  }  # source filtering # 當某些 document 很大時，僅針對特定的幾個 term 做查詢 POST kibana\_sample\_data\_ecommerce/\_search {  "\_source":["order\_date", "category.keyword"],  "from": 10,  "size": 5,   "query":{  "match\_all": {}  } } |

**Scripted Field(腳本欄位) Query**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | # 透過 ES 中 painless script 算出新的 field value # 搜尋結果中都會加上 hello 作為結尾 GET kibana\_sample\_data\_ecommerce/\_search {  "script\_fields": {  "new\_field": {  "script": {  "lang": "painless",  "source": "doc['order\_date'].value+'hello'"  }  }  },  "from": 10,   "size": 5,  "query": {  "match\_all": {}  } } |

**使用查詢表達式 - Match**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | # 預設為 "last OR christmas" POST movies/\_search {  "query": {  "match": {  "title": "last christmas"  }  } }  # 可透過 operator 改為 "last AND christmas" POST movies/\_search {  "query": {  "match": {  "title": {  "query": "last christmas",  "operator": "AND"  }  }  } } |

**使用查詢表達式 - Match Phrase**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 | # 必須按照順序出現 POST movies/\_search {  "query": {  "match\_phrase": {  "title":{  "query": "one love"  }  }  } }  # 加上 slop，中間可以有一個其他的 term 插入 POST movies/\_search {  "query": {  "match\_phrase": {  "title":{  "query": "one love",  "slop": 1   }  }  } } |

**Query String & Simple Query String 查詢**

**Query String Query**

* 類似 URI Query

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | # 可指定 default field(DF) # 可指定 operrator POST /users/\_search {  "query": {  "query\_string": {  "default\_field": "tool",  "query": "Docker AND Kubernetes"  }  } }  # 可指定多個 field POST /users/\_search {  "query": {  "query\_string": {  "fields":["tool","about"],  "query": "(Docker AND Kubernetes) OR (Java AND Elasticsearch)"  }  } } |

**Simple Query String Query**

* 類似 Query String, 但會忽略錯誤的語法，並且僅支援部份查詢語法
* 不支援 **AND**/**OR**/**NOT**，會當作 term 來處理
* term 之間預設的關係是 OR，可指定 operator
* 支援使用 + 取代 AND，| 取代 OR，- 取代 NOT

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | POST /users/\_search {  "query": {  "simple\_query\_string": {  "query": "Docker Kubernetes",  "fields": ["tool"],  "default\_operator": "AND"  }  } } |

**Dynamic Mapping 和常見欄位類型**

**What is Mapping ?**

* Mapping 類似資料庫中的 schema 定義，用途如下：
  + 定義 index 中每個 term 的名稱
  + 定義每個 term 的資料型態，例如：string, Interger, boolean
  + term & inverted index 的相關配置 (要使用哪個 Analyzer，或是不被索引)
* Mapping 會將 JSON document 映射成 Lucene 所需要的扁平格式
* 一個 Mapping 屬於一個 index type
  + 每個 document 都屬於一個 type
  + 一個 type 都會有一個 mapping 定義
  + 7.0 開始，不需要在 mapping 定義中指定 type 資訊

**Term Data Type**

**簡單類型**

* Text / Keyword
* Date
* Integer / Floating
* Boolean
* IPv4 & IPv6

**複雜類型**

* Object
* List

**特殊類型**

* geo\_point & geo\_shape (地理訊息)
* percolator

**What is Dynamic Mapping ?**

* 在寫入 Document 時，如果 index 不存在，則會自動建立 index
* Dynamic Mapping 可以根據 Document 內容，推算出 term data type 並自動建立 mapping，因此不需要手動制定
* 但推算的結果不一定會完全正確(例如：地理位置相關訊息可能會推斷錯誤)
* term data type 推算錯誤可能會導致某些查詢無法正常使用，例如：range 查詢

若是濫用 Dynamic Mapping，導致不斷有新欄位出現卻不管制，cluster state 會一直變更調整，而這樣的調整會需要同步到 cluster 中的所有 node 上，若是因為大量的 field mapping 而導致更新 cluster state 的操作若是太頻繁，很有可能會導致記憶體不足的問題發生，甚至會導致 cluster 掛掉，這現象稱為 Mapping Explosion

**範例**

簡單測試 Dynamic Mapping：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | # 寫入 document，mapping 會動態產生 PUT mapping\_test/\_doc/1 {  "firstName":"Chan",  "lastName": "Jackie",  "loginDate":"2018-07-24T10:29:48.103Z" }  # 透過 "\_mapping" 可以查看 mapping 資訊 # firstName => text + keyword # lastName => text + keyword # loginDate => date GET mapping\_test/\_mapping  # 刪除 index DELETE mapping\_test |

故意將某些資料類型加上雙引號測試：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | # UID 應該為 long，加上雙引號 # isAdmin 應該為 boolean，加上雙引號 PUT mapping\_test/\_doc/1 {  "uid" : "123",  "isVip" : false,  "isAdmin": "true",  "age":19,  "heigh":180 }  # 查看 mapping 設定 # uid => text + keyword # isVip => boolean # isAdmin => text + keyword # age => long # height => long GET mapping\_test/\_mapping |

**修改 Mapping 中的欄位類型**

在新增加欄位的情況下：

* 若 dynamic = true，一旦有新增欄位的 document 寫入時，mapping 資訊也會同時被更新
* 若 dynamic = false，mapping 不會被更新，新增欄位的資料無法被索引，但是資料會出現在 \_source 中
* 若 dynamic = strict，寫入 document 的操作會發生錯誤

若是針對已經存在的欄位，使用其他欄位類型的資料進行寫入操作，是無法變更 mapping 設定的，因為一旦當 reverted index 已經生成後，就無法修改；而若是真的要改變欄位類型，則是需要透過 Reindex API 來重建索引。

若是原有欄位的數據類型可以隨意修改，這樣會讓原本已經被索引的資料無法被搜尋

| **Dynamic Mapping 設定** | **“true”** | **“false”** | **“strict”** |
| --- | --- | --- | --- |
| document 可索引 | Yes | Yes | No (資料寫入會錯誤)) |
| 欄位可索引 | Yes | No | No |
| Mapping 被更新 | Yes | No (新增欄位被丟棄) | No |

**修改 Mapping 範例**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 | # 預設 dynamic mapping 開啟，寫入新的 document 進 index 中 PUT dynamic\_mapping\_test/\_doc/1 {  "newField":"someValue" }  # 資料可以被搜尋到，資料也同時出現在 \_source 中 POST dynamic\_mapping\_test/\_search {  "query":{  "match":{  "newField":"someValue"  }  } }  # 將 dynamic mapping 設定為 false PUT dynamic\_mapping\_test/\_mapping {  "dynamic": false }  # 新增 anotherField POST dynamic\_mapping\_test/\_doc/10 {  "anotherField":"someValue" }   # anotherField 這一筆資料無法被搜索，因為 dynamic mapping 已經被設定為 false POST dynamic\_mapping\_test/\_search {  "query":{  "match":{  "anotherField":"someValue"  }  } }  # 檢視 mapping 設定，還是只有原本就存在的 newField 設定 get dynamic\_mapping\_test/\_mapping  # 將 dynamic mapping 設定修改為 strict PUT dynamic\_mapping\_test/\_mapping {  "dynamic": "strict" }  # 此時寫入操作就會發生錯誤，HTTP Code 400 PUT dynamic\_mapping\_test/\_doc/12 {  "lastField":"value" }  # 移除 index DELETE dynamic\_mapping\_test |

**顯式 Mapping 設置與常見參數介紹**

**自定義 mapping**

往 index 送 PUT request 並帶上 mappings 設定即可：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | PUT movies {  "mappings": {  //define your mappings here  }  } |

撰寫 mapping 其實不是這麼容易，除了可以參考 API 來撰寫之外，也可以透過**寫入一些 sample data 到一個臨時的 index，讓 Elasticsearch 自動產生 mapping 定義後，再根據實際需求進行修改**。

**Index Options**

Inverted Index 根據要記錄的內容，有四種 index options 可以設定：

| **Index Option** | **Inverted Index 中的記錄內容** |
| --- | --- |
| docs | doc id |
| freqs | doc id + term frequency |
| positions | doc id + term frequency + term position |
| offsets | doc id + term frequency + term position + character offsets |

* type text 預設使用 positions，其他則為 docs
* 記錄的資料越多，需要儲存空間越多

**選擇性的讓特定 Field 不被索引**

預設情況下每個 field 都會被索引，但若是確定特定的 field 資料不需要被查詢，也可以不被索引：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 | PUT /users {  "mappings" : {  "properties" : {  "firstName" : {  "type" : "text"  },  "lastName" : {  "type" : "text"  },  //將 index 設定為 false，ES 就不會索引該 field 的資料  "mobile" : {  "type" : "text",  "index": false  }  }  } }  // 新增資料 PUT /users/\_doc/1 {  "firstName":"Leon",  "lastName": "Tseng",  "mobile": "12345678" }  // 搜尋會發生錯誤 // Cannot search on field [mobile] since it is not indexed. POST /users/\_search {  "query": {  "match": {  "mobile":"12345678"  }  } } |

**NULL value 的處理**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 | PUT /users {  "mappings" : {  "properties" : {  "firstName" : {  "type" : "text"  },  "lastName" : {  "type" : "text"  },  "mobile" : {  "type" : "keyword",  //可以透過給一個 "NULL" 字串的方式  //讓 ES 協助生成一個 keyword field 來搜尋  "null\_value": "NULL"  }   }  } }  //新增資料1 PUT /users/\_doc/1 {  "firstName":"Leon",  "lastName": "Tseng",  "mobile": null }  //新增資料2 PUT /users/\_doc/2 {  "firstName":"Leon2",  "lastName": "Tseng2" }  //只會找到資料1 //而且實際存在於 ES 中的是 null 值而非字串  GET /users/\_search {  "query": {  "match": {  "mobile":"NULL"  }  } } |

**copy\_to 設定**

透過 copy\_to 的設定可以新增一個 field，實現類似 \_all 的效果：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 | PUT /users {  "mappings": {  "properties": {  "firstName":{  "type": "text",  //firstName 資料會被複製到 fullName 中  "copy\_to": "fullName"  },  "lastName":{  "type": "text",  //lastName 資料會被複製到 fullName 中  "copy\_to": "fullName"  }  }  } }  //新增資料 PUT users/\_doc/1 {  "firstName":"Leon1",  "lastName": "Tseng1" }  //對新欄位查詢資料 GET /users/\_search?q=fullName=(Leon1) |

* \_all 在 ES 7 後被 copy\_to 取代
* 上述範例中，使用者可以針對 fullName 進行搜尋
* fullName 的資料不會出現在 \_source 中

**Aray Type**

目前 Elasticsearch 不支援 array type，但每個 field 都可以包含多個相同類型的數值：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | //新增一個帶有 string array 的資料 PUT /users/\_doc/1 {  "name":"twobirds",  "interests":["reading","music"] }  //可以正確被搜尋 POST /users/\_search {  "query": {  "match\_all": {}  } }  //而 name & interests 的 data type 都同樣被設定為 text GET /users/\_mapping |

表示 text type 其實是可以記錄 array type 的資料 (其實其他 type 也是可以記錄 array type 資料，例如：long))

**Multiple Field 特性及 Mapping 中配置自定義 Analyzer**

* multiple field 資料中，可透過 keyword field 作精確搜尋
* 若是要做全文搜尋，則是使用 text field 並搭配不同的 **analyzer** & **search analyzer**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 | PUT products {  "mappings": {  "properties": {  "company": {  "type": "text",   "fields": {  //利用 keyword field 來實現精確匹配  "keyword": {  "type": "keyword",   "ignore\_above": 256   }  }  },   "comment": {  "type": "text",   "fields": {  //可以在 field 中增加 sub-field，並指定不同的 analyer & search analyzer  //針對需要進行全文檢索的欄位，設定不同的 analyzer & search analyzer  "english\_comment": {  "type": "text",  "analyzer": "english", //索引用的 analyzer  "search\_analyzer": "english" //搜尋用的 analyzer  }  }  }  }  } } |

**Exact Values v.s. Full Text**

* Exact Value 包含數字/日期/或是特定的字串(例如：”Apple Store”)，在 Elasticsearch 中的 keyword field 存放的就是這一類的資料
* Full Text 則是非結構化的資料，在 Elasticsearch 中的 text field 存放的就是這一類的資料
* Exact Values 不需要做分詞處理，而 Full Text 則會需要分詞處理才能更容易被後續利用

**自訂分詞器(Analyzer)**

Analyzer 是專門處理分詞的組件，由三個部份組成：

* Character Filter (針對原始文件進行處理，例如：去除 HTML tag)
* Tokenizer (根據規則將資料切分 term or token)
* Token Filter (將分割後的 term 進行加工，例如：轉小寫、刪除 [stopwords](https://zh.wikipedia.org/wiki/%E5%81%9C%E7%94%A8%E8%AF%8D" \t "_blank)、增加同義詞)

**Character Filter**

* **可同時設定多個 Character Filter**
* 會影響 Tokenizer 的 position & offset 的資訊
* 目前 Elasticsearch 內建提供的 Character Filter 有 HTML strip、Mapping、Patter replace …等等

**Tokenizer**

* **只能設定一個 tokenizer**
* 將 Character Filter 處理過後的資料，按照一定的規則，切分為詞(term or token)
* 目前 Elasticsearch 內建提供的 tokenizer 有 whitespace/standard/uax\_url\_email/pattern/keyword(不做任何處理)/path hierarchy
* 也可以用 Java 實作自己的 tokenizer

**Token Filter**

* **可同時設定多個 Token Filter**
* 將 Tokenizer 輸出的 term(or token) 進行增加、修改、刪除
* 目前 Elasticsearch 內建提供的 Token Filter 有 lowercase/stop/synonym … 等等

**範例**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | POST \_analyze {    //將 HTML tag 去除  "char\_filter":["html\_strip"],   //keyword 不會進行分詞，會保留全部資料  "tokenizer":"keyword",   "text": "<b>hello world</b>" } |
| 1 2 3 4 5 6 7 | POST \_analyze {  //會將所有可能的目錄結構的分出來  //例如："/user", "/user/ymruan", "/user/ymruan/a" .... etc  "tokenizer":"path\_hierarchy",  "text":"/user/ymruan/a/b/c/d/e" } |

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | POST \_analyze {  //會根據 mapping 設定進行字串的取代  "char\_filter": [  {  "type" : "mapping",  "mappings" : [ "- => \_"]  }  ],   //標準分詞  "tokenizer": "standard",    "text": "123-456, I-test! test-990 650-555-1234" } |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | //char filter 替換表情符號 POST \_analyze {  //透過 mapping 將表情符號替換成一般字串  "char\_filter": [  {  "type" : "mapping",  "mappings" : [ ":) => happy", ":( => sad"]  }  ],   "tokenizer": "standard",    "text": ["I am felling :)", "Feeling :( today"] } |

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 | GET \_analyze {  //使用空白進行分詞  "tokenizer": "whitespace",   //stop words 會被忽略  //若改成 ["lowercase", "stop"]，下面的 "The" 就會被過濾掉了  "filter": ["stop"],    //"The" 會保留著，因為首字母並非小寫  "text": ["The rain in Spain falls mainly on the plain."] } |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | GET \_analyze {  //使用 regular express 進行資料處理  "char\_filter": [  {  "type" : "pattern\_replace",  "pattern" : "http://(.\*)",  "replacement" : "$1"  }  ],   "tokenizer": "standard",    "text" : "[http://www.elastic.co](http://www.elastic.co/)" } |

也可以完全自己定義一個 Analyzer：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 | PUT my\_index {  "settings": {  "analysis": {  "analyzer": {  //裡面使用的 character filter, tokenizer, token filter 幾乎都是下面自行定義的  "my\_custom\_analyzer": {  "type": "custom",  "char\_filter": [ "emoticons" ],   "tokenizer": "punctuation",  "filter": [ "lowercase", "english\_stop" ]  }  },   //自訂 character filter => emoticons  "char\_filter": {  "emoticons": {  "type": "mapping",   "mappings": [  ":) => \_happy\_",   ":( => \_sad\_"  ]  }  },   //自訂 tokenizer => punctuation  "tokenizer": {  "punctuation": {  "type": "pattern",   "pattern": "[ .,!?]"  }  },   //自訂 token filter => english\_stop  "filter": {  "english\_stop": {  "type": "stop",   "stopwords": "\_english\_"  }  }  }  } }  //驗證上面自訂的 analyzer POST my\_index/\_analyze {  "analyzer": "my\_custom\_analyzer",  "text": "I'm a :) person, and you" |

**Index Template 和 Dynamic Template**

* 若 cluster 是用來做 log 管理，每天都產生一個專屬的 index 存放 log，數據管理上較為合理，效能表現也會較好

**Index Template**

Index Template 使用來協助使用者設定 mappings & settings 的相關規則，並可透過套用 template 建立新的 index 來取得在 template 中已經存在的設定

* index template 只有在建立 index 有用，後續修改 template 不會影響已經存在的 index
* 可以設定多個 index template，這些設定會被合併在一起
* 也可以透過指定 order，來調整 index template 合併的過程

那 index template 是如何在 index 被新增時運作的呢?

1. 先選用 Elasticsearch 中預設的 settings & mappings
2. 先套用 order 值低的 index template 中的設定
3. 再套用 order 值高的 index template 中的設定，並覆蓋之前的設定
4. 如果建立 index 時使用者有指定 settings & mappings，就會覆蓋 index template 的設定

以下透過簡單範例，說明 index template 的建立跟使用時實際的流程：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 | //建立 default index template //並設定 order = 0 PUT \_template/template\_default {  "index\_patterns": ["\*"],  "order" : 0,  "version": 1,  "settings": {  "number\_of\_shards": 1,  "number\_of\_replicas":1  } }  //設定 index 為 test 開頭的 index template //並設定 order = 1 //因此 test 開頭的 index 會優先套用此設定 PUT \_template/template\_test {  "index\_patterns" : ["test\*"],  "order" : 1,  "settings" : {  "number\_of\_shards": 1,  //調整 replica 的數量  "number\_of\_replicas" : 2  },  "mappings" : {  //取消日期的偵測  "date\_detection": false,  //開啟數字的偵測  "numeric\_detection": true  } }  //查詢單一 index template 內容 GET /\_template/template\_default //查詢所有 "temp" 開頭的 index template 資訊 GET /\_template/temp\*  //新增一個名稱為 testtemplate 的 index，並新增一筆資料 //由於名稱為 test 開頭，因此會套用 template\_test 的設定 PUT testtemplate/\_doc/1 {  "someNumber":"1",  "someDate":"2019/01/01" }  //日期不會被偵測，而被當作一般的 text，而數字則是會偵測為 long GET testtemplate/\_mapping //shard = 1, replica = 2 GET testtemplate/\_setting  //新增一個 index，並自行指定 settings //自行指定 settings 就會覆蓋 index template 中的所有設定 PUT testmy {  "settings":{  "number\_of\_replicas":5  } }  //新增資料到 testmy index 中 put testmy/\_doc/1 {  "key":"value" }  //index template 中的 settings 被覆蓋成上面指定的 "number\_of\_replicas":5 get testmy/\_settings  //移除 index DELETE testmy DELETE /\_template/template\_default DELETE /\_template/template\_test |

**Dynamic Template**

dynamic template 相較於前面提到的 index template，擁有比較彈性的方式來制定 template，例如：

* 將所有 field 都設定成 keyword，或是關閉 keyword field
* 將 is 開頭的 field 都設定為 boolean
* 將 long 開頭的 field 都設定為 long 類型

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 | //使用內建的自動辨識 PUT my\_index/\_doc/1 {  "firstName":"Ruan",  "isVIP":"true" }  //可以發現 firstName & isVIP 都被辨識成 text GET my\_index/\_mapping |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 | //移除原本的 index DELETE my\_index  //為 index 設定 dynamic template PUT my\_index {  "mappings": {  "dynamic\_templates": [  {  "strings\_as\_boolean": {  "match\_mapping\_type": "string",  //"is" 開頭的 field 設定為 boolean  "match":"is\*",  "mapping": {  "type": "boolean"  }  }  },  {  "strings\_as\_keywords": {  "match\_mapping\_type": "string",  //其他的部份則設定為 keyword  "mapping": {  "type": "keyword"  }  }  }  ]  } }  //firstName 會變成 keyword //isVIP 會變成 boolean PUT my\_index/\_doc/1 {  "firstName":"Ruan",  "isVIP":"true" }  GET my\_index/\_mapping |

一個以 path 為基礎，並搭配較為複雜處理的設定：

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 | //移除原本的 index DELETE my\_index  // 設定 dynamic template，以 path match 為基礎 PUT my\_index {  "mappings": {  "dynamic\_templates": [  {  "full\_name": {  //若符合以下 path 的 match & unmatch 條件  "path\_match": "name.\*",  "path\_unmatch": "\*.middle",  //將資料複製一份到 full\_name 欄位  "mapping": {  "type": "text",  "copy\_to": "full\_name"  }  }  }  ]  } }  PUT my\_index/\_doc/1 {  "name": {  "first": "John",  "middle": "Winston",  "last": "Lennon"  } }  //後面就可以使用額外加入的 full\_name 進行搜尋 //full\_name 不會出現在 \_source 中 GET my\_index/\_search?q=full\_name:John |

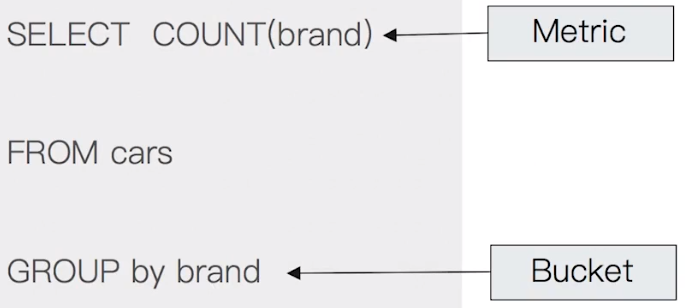
**Elasticsearch 聚合分析簡介**

**Aggregation**

* Elasticsearc 除了一般的搜尋之外，也可透過 aggergation 的方式，提供數據統計分析的功能，可作到很即時的查詢
* 透過 aggregation 可以得到分析 & 總結數據後的總覽，而非單一 document，例如：
  + 某特定區域剩餘的客房數量
  + 進行不同價格區間的搜尋，可預定的平價旅館 & 五星級酒店的數量
* 效能很好，在 ES 端就可以得到分析結果，不用在 client 端開發分析邏輯
* 許多 Kibana 報表中的元素都可以透過 aggregation 的數據來完成。例如：
  + 公司員工的職位分佈、薪水分佈
  + 客戶所在的地理位置分佈
  + 訂單的增減狀況

**Aggregation Family (聚合總類)**

* [Bucket Aggregation](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket.html)：滿足特定條件的 document 集合 (類似傳統 SQL 語法中的 Group By)
* [Metric Aggregation](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics.html)：可透過數學運算，對 document filed 進行統計分析
  + 基於數據集合計算結果；除了支援在 field 上進行計算，也支援在 (painless) script 產生的結果之上進行計算
  + 大多數的 metric 是數學計算，只會輸出一個值，例如：**min** / **max** / **sum** / **avg** / **cardinality**
  + 部份的 metric 支援輸出多個值，例如：**stats** / **percentiles** / **percentile\_ranks**



* [Matrix Aggregation](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-matrix.html)：支援對多個 field 同時進行操作並提供一個結果矩陣
* [Pipiline Aggregation](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-pipeline.html): 對其他 aggregation 的結果再一次的進行 aggregation

**範例：Bucketing (分桶)**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | //根據目的地進行 bucketing 操作 GET kibana\_sample\_data\_flights/\_search {  "size": 0,  "aggs":{  "flight\_dest":{  //使用 terms 關鍵字，指定要進行 bucketing，使用的是 field DestCountry  "terms":{  "field":"DestCountry"  }  }  } } |

**範例：Bucketing + Metric**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 | //查看航班目的地的統計資訊，並以目的地為單位，額外增加平均，最高最低價格 GET kibana\_sample\_data\_flights/\_search {  "size": 0,  "aggs":{  //先進行 bucket aggregation 操作  "flight\_dest":{  "terms":{  "field":"DestCountry"  },  //根據上面結果再進行 metric aggregation 操作  "aggs":{  "avg\_price":{  "avg":{  "field":"AvgTicketPrice"  }  },  "max\_price":{  "max":{  "field":"AvgTicketPrice"  }  },  "min\_price":{  "min":{  "field":"AvgTicketPrice"  }  }  }  }  } } |

**範例：Bucketing + Metric + Matrix**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 | //查看航班目的地的統計資訊，並以目的地為單位，額外增加平均票價 & 目的地天氣的統計資訊 GET kibana\_sample\_data\_flights/\_search {  "size": 0,  "aggs":{  "flight\_dest":{  "terms":{  "field":"DestCountry"  },  "aggs":{  "stats\_price":{  //透過 stats 關鍵字可直接帶出 count/min/max/avg/sum ... 等資訊  "stats":{  "field":"AvgTicketPrice"  }  },  "weather":{  "terms": {  "field": "DestWeather",  //可限定輸出的資料數量，預設為 10  "size": 5  }  }  }  }  } } |

**References**

* [Elasticsearch核心技術與實戰 | 極客時間](https://time.geekbang.org/course/intro/100030501)
* [Elasticsearch Reference [7.9] | Elastic](https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html)

[# Elasticsearch](https://godleon.github.io/blog/tags/Elasticsearch/)