Document API

henry hsiao

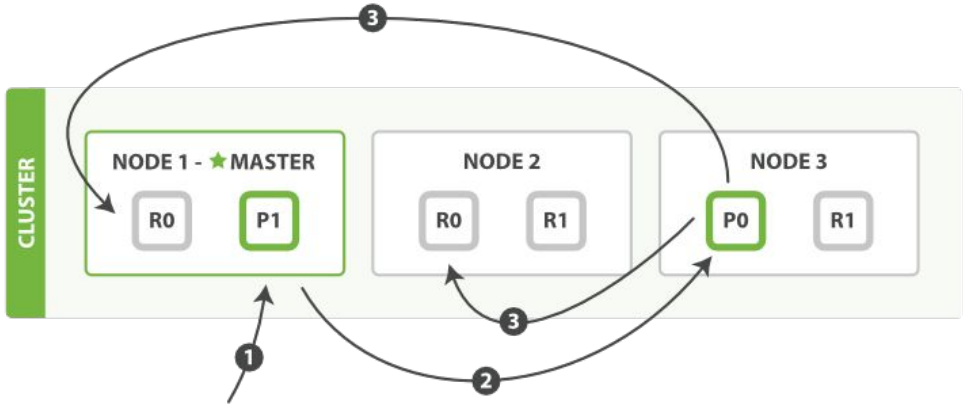
Reading and Writing documents

data replication model 基於 primary-backup model

先使用 routing 解析後再使用 document ID 轉發到 replication group

轉發的過程是

1. master 收到請求 (Node 1 master)
2. primary shard (P0)
3. replication (其他 replica)



Basic write model

先使用 routing 解析後再使用 document ID 轉發到 replication group

首先 shard 收到請求

1. 驗證操作
2. 在本地執行操作，近一步驗證
3. 轉發到 in-sync 的 replica
4. 一旦 replica 完成就會通知 shard
5. shard 再通知 client 端成功消息

Failure handling

6. primary shard 確認同步失敗
7. 送消息給 master
8. master 會移除該分片，並請另一個node建立分片

Basic read model

用 **id** 查找是非常不消耗資源的。aggregations 消耗大量 CPU 資源
因為模型確保資料一致，所以 single in-sync 也可以處理查詢請求

coordinating node 協調節點

1. 發請求到相關 replica 可能多個
2. 從 shard replication group 中尋找一個活躍副本，primary or a replica 都可以
3. 發一個 shard level read requests 到這些副本上
4. 合併結果(如果是by id 就不用合併)，並回傳結果

Failure handling

1. 如果選擇的節點不能完成任務，協調節點會選擇另一個副本。
2. 如果都沒有可用的 副本 就會錯誤
3. 某些情況下，es 為盡快回應資料，儘管只有部分資料也會直接回應 (可從回應的 _shard 確認是否是部分結果)

A few simple implications

Efficient reads

Read unacknowledged

Two copies by default

Failures

A single shard can slow down indexing

Dirty reads

Index API

Index API

total - 表示有多少個 **shard copies** (分片副本)(**primary** (主)分片和 **replica** (副本) 分片)執行本次操作。

successful - 表示索引操作成功的 shard replica 的数量。超過 1 可視為成功。

failed - 操作失敗的 shard replica 數量

有可能 successful 少於 total 而且沒有 fail 的情況。


```
{
   "_shards" : {
      "total" : 2,
      "failed" : 0,
      "successful" : 2
   },
   "_index" : "twitter",
   "_type" : "tweet",
   "_id" : "1",
   "_version" : 1,
   "created" : true,
   "result" : created
}
```

Index API

automatic index creation

- 自動建立 index, 其實就是幫你呼叫 create api
- mapping 會參照已存在的 Index templates

action.auto_create_index

- false 禁止
- 黑白名單 twitter,index10,-index1*,+ind* (可以自動建立 twitter, index10, indxxxxx, 不能 index1xxxx)

index.mapper.dynamic

- false 禁止

op_type

如果已經存在就創建失敗
ex: PUT /twitter/tweet/1?op_type=create

Index API

automatic ID generation 自動生成 ID

POST 時 **op_type 自動設定為 create**

Optimistic concurrency control :

- _seq_no and _primary_term：藉由指定兩者來確保更新，避免受其他的更新影響

Routing

- POST 'localhost:9200/twitter/tweet?routing=kimchy'
- 這時候會直接找到 kimchy 這個 shard 來取資料
- 一但設置 Routing，未來就都必須提供，否則失敗

Distributed

Refresh

wait for active shards

index.write.wait_for_active_shards 一般來說 設置 1，就只需要 shard 完成就可以，如果不滿足條件，就會等待直到超時或滿足(最大為 **number_of_replicas+1)**

Noop Updates

如果 操作後 document 沒有更改，就不需要增加 version

Timeout

default 1 分鐘

ex: PUT /twitter/tweet/1?timeout=5m

Versioning
可以自定義版本號

大於 **0** 且小于大约 **9.2e+18**, **0** 的話不能使用 **update_by_query**

PUT twitter/_doc/1?version=2&version_type=external

Version types

internal (default)

external 或 external_gt (non-negative long number)

external_gte (non-negative long number) 謹慎使用

GET API

Get API

基本用法

GET twitter/_doc/0

只判斷存在與否

HEAD my_index/_doc/1

Realtime

default: true


```
{
  "_index" : "twitter",
  "_type" : "_doc",
  "_id" : "0",
  "_version" : 1,
  "_seq_no" : 10,
  "_primary_term" : 1,
  "found": true,
  "_source" : {
    "user" : "kimchy",
    "date" : "2009-11-15T14:12:12",
    "likes": 0,
    "message" : "trying out Elasticsearch"
  }
}
```

Get API

Source filtering

GET my_index/_doc/1?_source=false

GET my_index/_doc/1?_source=*.id,retweeted

GET my_index/_doc/1?_source_includes=*.id&_source_excludes=entities

Get API

Stored Fields

mapping 時需設定 "store": true

GET twitter/_doc/1?stored_fields=tags,counter

always returned as an array

```
PUT twitter
{
  "mappings": {
    "_doc": {
      "properties": {
        "counter": {
          "type": "integer",
          "store": false
        },
        "tags": {
          "type": "keyword",
          "store": true
        }
      }
    }
  }
}
```

Get API

Getting the _source directly

/{index}/{type}/{id}/_source

GET twitter/_doc/1/_source

GET twitter/_doc/1/_source?_source_includes=*.id&_source_excludes=entities

一樣可以用 HEAD 檢查，但如果 mapping 就禁止 _source，就會找不到存在的 document

HEAD twitter/_doc/1/_source

Get API

Routing

GET twitter/_doc/2?routing=user1

當mapping 設定 routing = required 時，之後就必須使用 routing 來取得檔案

Preference

從哪個副本取資料的優先度

- _primary
- _local
- Custom (string) value
  - 確保使用相同副本會有相同定義 值
  - 類似web session id或者user name

Get API

Refresh

確保刷新，但影響效能，也減慢index的建立

Distributed

副本可以承擔需求，越多副本效率越好

Versioning support

可以指定 version，只有在當前 version 跟指定的 version 一樣時，才會返回資料，否則 409 錯誤

Delete API

Delete API

基本語法

DELETE /twitter/_doc/1

Optimistic concurrency control

Versioning

1. 可以指定 version 只有當前版本跟指定版本相同時，才會刪除，否則錯誤
2. 刪除也會增加版本

Delete API

Routing

DELETE /twitter/_doc/1?routing=kimchy

當mapping 設定 routing = required 時，刪除一樣必須指定routing，未指定會噴錯誤。

Automatic index creation

刪除也會自動建立 index

Delete API

Distributed

Wait For Active Shards

Refresh

Timeout

預設 1 分鐘

DELETE /twitter/_doc/1?timeout=5m

Delete By Query API

Delete By Query API

語法跟 search API 一樣

也可以用 q 參數傳遞

```
POST twitter/_delete_by_query
{
  "query": {
    "match": {
      "message": "some message"
    }
  }
}
```

```
{
  "took" : 147,
  "timed_out": false,
  "deleted": 119,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "total": 119,
  "failures" : [ ]
}
```

Delete By Query API

_delete_by_query 在開始時先取得 index 快照並使用 internal version 來刪除.

當取得快照到實際執行間，如果 document 版本改動 (version change)，就會導致 version conflict 錯誤。反之，當版本一致時 document 將會被刪除。

PS: _delete_by_query 不支援刪除版本號為 0 的資料

_delete_by_query 執行期間會執行多個查詢請求，每發現一部分資料便會執行對應的刪除，如果查詢請求被拒絕，默認採用重試策略（最多10次），達到上限時返回錯誤。

當錯誤發生時，只會終止執行，不會 rollback 。

Delete By Query API

遇到版本衝突時可以添加 conflicts=proceed
來避免中止

預設為 abort

```
POST twitter/_doc/_delete_by_query?conflicts=proceed
{
  "query": {
    "match_all": {}
  }
}
```

Delete By Query API

也可以一次刪除多個 index 或多個 type

或者 _all 全部 index

```
POST twitter,blog/_docs,post/_delete_by_query
{
  "query": {
    "match_all": {}
  }
}
```

Delete By Query API

預設使用 scroll 每批次查詢 1000 筆。
可以使用 scroll_size 更改數量

```
POST twitter/_delete_by_query?scroll_size=5000
{
  "query": {
    "term": {
      "user": "kimchy"
    }
  }
}
```

Delete By Query API

URL Parameters

refresh : 只針對刪除牽涉的 shards 刷新，不支援 wait_for 參數。

wait_for_completion : 預設 true，如果為 false 時，只處理預檢查並啟動請求後便返回 task。

wait_for_active_shards：控制 每次 請求之前必須有多少 shard 處於 active 狀態

timeout : 控制 每次 請求等待時間，預設 1m

scroll : 維持快照多久 (5m)

Delete By Query API

URL Parameters

requests_per_second : 控制每秒多少 sub-request，可為 (0, 1.4, 1000)，關閉為 -1，預設為 -1

因為預設 batch size 為 1000，假設 requests_per_second 為 500

target_time = 1000 / 500 per second = 2 seconds
wait_time = target_time - write_time = 2 seconds - .5 seconds = 1.5 seconds

因為批次請求是基於 _bulk 執行，如果設定過大，導致短時間內會建立多個請求。這會 突發 (bursty) 而非平滑 (smooth) 的執行。

如果要讓 ES 可以有更多等待時間可以執行其他任務，加大 requests_per_second，反之縮小 。

如果要讓每次請求的 _bulk 執行更多操作，加大 scroll_size， 反之縮小

Delete By Query API

Response body

`took`
　從開始到結束花費多少 毫秒

`timed_out`
　如果請求過程中遇到 timeout 則回應 true

`total`
　成功處理筆數

`deleted`
　成功刪除筆數

`batches`
　因 delete_by_query， 發出的 scroll 請求回應次數


```
{
  "took" : 147,
  "timed_out": false,
  "total": 119,
  "deleted": 119,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "failures" : [ ]
}
```

Delete By Query API

Response body

`version_conflicts`
遭遇到的版本衝突次數

`noops`
永遠為 0，只為結構一致。

`retries`
bulk 跟 search 重複嘗試次數

`throttled_millis`
因 requests_per_second 限制而休息的毫秒數。

`requests_per_second`
每秒執行多少 requests 。


```
{
  "took" : 147,
  "timed_out": false,
  "total": 119,
  "deleted": 119,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "failures" : [ ]
}
```

Delete By Query API

Response body

`throttled_until_millis`

永遠為 0，只有在 Task API 時為了 requests_per_second 時為了

`failures`

如果在執行的過程中，有任何不可回復的錯誤，會搜集所有錯誤並以陣列返回。如果有錯誤就會中止請求。也可以使用 conflicts=process 來防止因為 version conflict 造成的請求中止。


```
{
  "took" : 147,
  "timed_out": false,
  "total": 119,
  "deleted": 119,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "failures" : [ ]
}
```

Delete By Query API

Works with the Task API

GET _tasks?detailed=true&actions=*/delete/byquery

當 total 欄位 = updated + created + deleted 時結束 task


```
{
 "nodes" : {
  "r1A2WoRbTwKZ516z6NEs5A" : {
   "name" : "r1A2WoR",
   "transport_address" : "127.0.0.1:9300",
   "host" : "127.0.0.1",
   "ip" : "127.0.0.1:9300",
   "attributes" : {
    "testattr" : "test",
    "portsfile" : "true"
   },
   "tasks" : {
    "r1A2WoRbTwKZ516z6NEs5A:36619" : {
     "node" : "r1A2WoRbTwKZ516z6NEs5A",
     "id" : 36619,
     "type" : "transport",
     "action" : "indices:data/write/delete/byquery",
     "status" : {
      "total" : 6154,
      "updated" : 0,
      "created" : 0,
      "deleted" : 3500,
      "batches" : 36,
      "version_conflicts" : 0,
      "noops" : 0,
      "retries": 0,
      "throttled_millis": 0
     },
     "description" : ""
    }
   }
  }
 }
}
```

Delete By Query API

Works with the Task API

查找 task

GET /_tasks/r1A2WoRbTwKZ516z6NEs5A:36619

建立這任務的成本是，當設置 wait_for_completion = false 時，會自動建立一個檔案在 .task/task/${taskId}。

取消任務

POST _tasks/r1A2WoRbTwKZ516z6NEs5A:36619/_cancel

取消會盡快完成，但可能會需要幾秒鐘。

Rethrottling

POST _delete_by_query/task_id:1/_rethrottle?requests_per_second=-1

利用 task id 來更改 requests_per_second

Delete By Query API

Slicing - Manual slicing

支援 `slice-scroll` 來並行化處理。

```
POST twitter/_delete_by_query
{
  "slice": {
    "id": 0,
    "max": 2
  },
  "query": {
    "range": {
      "likes": {
        "lt": 10
      }
    }
  }
}
```

```
POST twitter/_delete_by_query
{
  "slice": {
    "id": 1,
    "max": 2
  },
  "query": {
    "range": {
      "likes": {
        "lt": 10
      }
    }
  }
}
```

Delete By Query API

Slicing - Automatic slicing

可以依據 `slices` 自動切分

如果是 `slices=auto` ，則依照 `shard` 數量切分，如果有多個 `index (indices)`，則採用 `indices` 中最少 `shard` 的數量。

```
POST twitter/_delete_by_query?slices=5
{
  "query": {
    "range": {
      "likes": {
        "lt": 10
      }
    }
  }
}
```

```
POST twitter/_delete_by_query?slices=auto
{
  "query": {
    "range": {
      "likes": {
        "lt": 10
      }
    }
  }
}
```

Delete By Query API

使用 slices 會有一些影響

- 你會在 Task API 看到一些來自於 slices 的子任務
- 取得任務狀態時，只會看到已完成的 slices 子任務
- 這些子任務可以單獨 取消 或 設置節流閥 rethrottling
- 對 slices 設置節流閥 rethrottling 將按比例重新限制未完成的子任務
- 取消 slices 將取消所有子任務
- 因為 slices 的性質，每個子任務不會有相同的文檔分佈。當然所有 document 都會被處理，但預期更大的切片會有更均勻的分佈。
  (40,400,60) or (50, 50, 60, 90, 50, 60, 40)
- 因為文檔分佈不均勻，如果同時使用 size 與 slice 可能導致 size 的不精準。(ex: size 100 實際 10)
- 儘管大約同一時間執行快照，每個子任務可能還是會有略微不同大小的快照。

Picking the number of slices

slices 為 auto，系統會自動提供一個合理的切片數字，但如果你選擇手動分片，這邊有一些建議：

在查詢的角度，當 slices 等於 index 的 shrads 數量時，最有效率。

如果設置過大 (像是, 500) 會造成大量的效能影響，一般來說不會加快速度，反而增加開銷。請選擇小於 shard 的數字。

刪除性能在可用的資源間，隨 slices 數量線性成長。

究竟是查詢的效能還是刪除的效能影響執行，取決於許多因素，例如 重建索引 或 reindexing 。

Update API

Update API

允許基於 script 操作來更新document

1. 從 index 取得資料 (shard)
2. 執行 script
3. index 記錄結果 (也可能刪除或忽略)

使用版本控制，確保 get 跟 reindex 期間沒有版本衝突

還是 full document reindex，只是避免了網路往返與減少 get 跟 reindex 的時候版本衝突。

lang : painless

Update API

Scripted updates

```
POST test/_doc/1/_update
{
   "script" : {
      "source": "ctx._source.counter += params.count",
      "lang": "painless",
      "params" : {
         "count" : 4
      }
   }
}
```

```
POST test/_doc/1/_update
{
   "script" : {
      "source": "ctx._source.tags.add(params.tag)",
      "lang": "painless",
      "params" : {
         "tag" : "blue"
      }
   }
}
```

Update API

```
POST test/_doc/1/_update
{
   "script" : {
      "source": "if (ctx._source.tags.contains(params.tag)) {
ctx._source.tags.remove(ctx._source.tags.indexOf(params.tag)) }",
      "lang": "painless",
      "params" : {
         "tag" : "blue"
      }
   }
}
```

ctx 還對應到 document 上

_source
_index
_type
_id
_version
_routing
_now (the current timestamp).

Update API

加新欄位

```
"script" : "ctx._source.new_field = \"value_of_new_field\""
```

刪除欄位

```
"script" : "ctx._source.remove(\"new_field\")"
```

或什麼都不做

```
"script" : {
        "source": "if (ctx._source.tags.contains(params.tag)) {
ctx.op = 'delete' } else { ctx.op = 'none' }",
        "lang": "painless",
        "params" : {
            "tag" : "green"
        }
    }
```

Update API

Updates with a partial document

也可以只做部分文檔更新

```
POST test/_doc/1/_update
{
   "doc" : {
      "name" : "new_name"
   }
}
```

注意 同時存在 script 與 doc 時，doc 將被忽略。

Update API

如果更新完全相同。將回傳 "result": "noop"，並且完全忽略此請求。version 也不會更新。

也可以通過設置 detect_noop 來強制更新。

```
POST test/_doc/1/_update
{
    "doc" : {
        "name" : "new_name"
    },
    "detect_noop": false
}
```

Update API

Upserts

如果 document 不存在，則將 upsert 內容插入成一個新的文檔。

如果存在將轉而執行 script

```
POST test/_doc/1/_update
{
   "script" : {
      "source": "ctx._source.counter += params.count",
      "lang": "painless",
      "params" : {
         "count" : 4
      }
   },
   "upsert" : {
      "counter" : 1
   }
}
```

Update API

scripted_upsert

如果無論 document 存不存在，script 都要執行，則設置 scripted_upsert = true。

如果不存在，由 upsert 建立新文檔後執行 script

```
POST test/_doc/2/_update
{
  "scripted_upsert": true,
  "script": {
    "inline": "if (ctx.op == \"create\") ctx._source.numbers = [1,2,3]; else ctx._source.numbers = [44]"
  },
  "upsert": {}
}
```

Update API

doc_as_upsert

將 doc 內容作為 upsert

原本的 doc 做法是，如果 document 不存在就不更新

```
POST test/_doc/3/_update
{
   "doc" : {
      "name" : "new_namess"
   },
   "doc_as_upsert" : true
}
```

Update API

Parameters

retry_on_conflict

routing: 不能更新既有的 routing 設定

timeout

wait_for_active_shards

refresh

_source

version：不支援 external 與 external_gte 、forced

if_seq_no and if_primary_term

Update By Query API

Update By Query API

POST twitter/_update_by_query?conflicts=proceed


```
{
  "took" : 147,
  "timed_out": false,
  "updated": 120,
  "deleted": 0,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "total": 120,
  "failures" : [ ]
}
```

Update By Query API

_update_by_query 在開始時先取得 index 快照並使用 internal version 來更新.

當取得快照到實際執行間，如果 document 版本改動 (version change)，就會導致 version conflict 錯誤。反之，當版本一致時 document 就會更新。

PS: _update_by_query 因為內部使用 internal，所以不支援更新版本號為 0 的資料

_update_by_query 執行期間會執行多個查詢請求，每發現一部分資料便會執行對應的更新，如果查詢請求被拒絕，默認採用重試策略（最多10次），達到上限時返回錯誤。

當錯誤發生時，只會終止執行，不會 rollback 。

Delete By Query API

遇到版本衝突時可以添加 conflicts=proceed
來避免中止

預設為 abort

可以使用 Query DSL 查詢 Script 更新

ctx.op

noop : 當完全一致時不更新 version，並計數在 noops 。

delete : 將刪除符合文檔，並計數在 deleted

```
POST twitter/_update_by_query?conflicts=proceed
{
  "script": {
    "source": "ctx._source.likes++",
    "lang": "painless"
  },
  "query": {
    "term": {
      "user": "kimchy"
    }
  }
}
```

Update By Query API

```
POST twitter,blog/_doc,post/_update_by_query

POST twitter/_update_by_query?routing=1

POST twitter/_update_by_query?scroll_size=100


PUT _ingest/pipeline/set-foo
{
  "description" : "sets foo",
  "processors" : [ {
      "set" : {
        "field": "foo",
        "value": "bar"
      }
  } ]
}
POST twitter/_update_by_query?pipeline=set-foo
```

Update By Query API

URL Parameters

pretty

refresh

wait_for_completion

wait_for_active_shards

timeout

scroll

requests_per_second

Update By Query API

返回格式與 delete by query 一樣

noops
　由於用於查詢更新的腳本而忽略的文檔數返回了 ctx.op 的 noop 數量。


```
{
  "took" : 147,
  "timed_out": false,
  "total": 5,
  "updated": 5,
  "deleted": 0,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "failures" : [ ]
}
```

Update By Query API

章節內容跟 delete by query 一樣

- Works with the Task API
- Works with the Cancel Task API
- **Rethrottling**
- **Slicing**
  - Manual slicing
  - Automatic slicing
  - Picking the number of slices

Update By Query API

Pick up a new property

"dynamic": false,

// 不會被 index 只會存放在 _source

這時加入的資料都不會被 search

就算更改 mapping 還是不能查到

只能全部 reindex

```
PUT test
{
  "mappings": {
    "_doc": {
      "dynamic": false,
      "properties": {
        "text": {"type": "text"}
      }
    }
  }
}
```

Update By Query API

Pick up a new property

這時可以使用 update_by_query 來更新 mapping

POST test/_update_by_query?refresh&conflicts=proceed

Multi Get API

Multi Get API

```
GET /_mget
{
   "docs" : [
      {
         "_index" : "test",
         "_type" : "_doc",
         "_id" : "1"
      },
      {
         "_index" : "test",
         "_type" : "_doc",
         "_id" : "2"
      }
   ]
}
```

```
GET /test/_mget
{
   "docs" : [
      {
         "_type" : "_doc",
         "_id" : "1"
      },
      {
         "_type" : "_doc",
         "_id" : "2"
      }
   ]
}
```

```
GET /test/_doc/_mget
{
   "docs" : [
      {
         "_id" : "1"
      },
      {
         "_id" : "2"
      }
   ]
}
```

```
GET /test/_doc/_mget
{
   "ids" : ["1", "2"]
}
```

Multi Get API

```
GET test/_doc/_mget
{
    "docs" : [
        {
            "_id" : "1",
            "_source" : false
        },
        {
            "_id" : "2",
            "_source" : ["field3", "field4"]
        },
        {
            "_id" : "3",
            "_source" : {
                "include": ["user"],
                "exclude": ["user.location"]
            }
        }
    ]
}
```

```
{
  "docs" : [
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "1",
      "_version" : 1,
      "_seq_no" : 0,
      "_primary_term" : 1,
      "found" : true
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "2",
      "_version" : 1,
      "_seq_no" : 0,
      "_primary_term" : 1,
      "found" : true,
      "_source" : {
        "field3" : "field3",
        "field4" : "field4"
      }
    },
    {
      "_index" : "test",
      "_type" : "_doc",
      "_id" : "3",
      "_version" : 1,
      "_seq_no" : 0,
      "_primary_term" : 1,
      "found" : true,
      "_source" : {
        "user" : {
          "name" : "nick"
        }
      }
    }
  ]
}
```

Multi Get API

```
GET /test/_doc/_mget?stored_fields=field1,field2
{
   "docs" : [
      {
         "_id" : "1"
      },
      {
         "_id" : "2",
         "stored_fields" : ["field3", "field4"]
      }
   ]
}
```

_id 1 : 返回 field1 和 field2
_id 2 : 返回 field3 和 field4

Multi Get API

Routing

```
GET /_mget?routing=key1
{
    "docs" : [
        {
            "_index" : "test",
            "_type" : "_doc",
            "_id" : "1",
            "routing" : "key2"
        },
        {
            "_index" : "test",
            "_type" : "_doc",
            "_id" : "2"
        }
    ]
}
```

Security

elasticsearch.yml

rest.action.multi.allow_explicit_index: true

預設是 true，body 內設定的 index 可以覆蓋 url 上的 index。
反之，false

Partial responses

為確保快速響應，如果一個或多個分片失敗，多獲取 API 將以部分結果作為響應。有關詳細信息，請參閱分片故障。

Bulk API

Bulk API

https://elasticsearch-py.readthedocs.io/en/6.8.2/api.html#elasticsearch.Elasticsearch.bulk

REST API endpoint is /_bulk

JSON (NDJSON)

action_and_meta_data\n
optional_source\n
action_and_meta_data\n
optional_source\n
....
action_and_meta_data\n
optional_source\n

備註：最後一行必須以換行符 **\n** 結尾。每個換行符前面都可以有 **\r\n**。向此端點發送請求時，Content-Type標頭應設置為 application / x-ndjson。

Bulk API

如果要為 curl 提供文本輸入，則必須使用 --data-binary 而不是 plain -d。後者不保留換行符。

例：

```
cat requests
{ "index" : { "_index" : "test", "_type" : "_doc", "_id" : "1" } }
{ "field1" : "value1" }

curl -s -H "Content-Type: application/x-ndjson" -XPOST localhost:9200/_bulk --data-binary "@requests";

curl -s -H "Content-Type: application/x-ndjson" -XGET localhost:9200/test/_doc/1
```

Bulk API

index : 如果 document 存在則更新

create : 如果 document 存在則失敗

delete : 不需要下一行 optional_source

update : 需要在 下一行 optional_source，指定 doc, upsert 或 script 行為。

```
POST _bulk
{ "index" : { "_index" : "test", "_type" : "_doc", "_id" : "1" } }
{ "field1" : "value1" }
{ "delete" : { "_index" : "test", "_type" : "_doc", "_id" : "2" } }
{ "create" : { "_index" : "test", "_type" : "_doc", "_id" : "3" } }
{ "field1" : "value3" }
{ "update" : {"_id" : "1", "_type" : "_doc", "_index" : "test"} }
{ "doc" : {"field2" : "value2"} }
```

Bulk API


```
{
  "took": 30,
  "errors": false,
  "items": [
    {
      "index": {
        "_index": "test",
        "_type": "_doc",
        "_id": "1",
        "_version": 1,
        "result": "created",
        "_shards": {
          "total": 2,
          "successful": 1,
          "failed": 0
        },
        "status": 201,
        "_seq_no" : 0,
        "_primary_term": 1
      }
    }
```

```
{
  "delete": {
    "_index": "test",
    "_type": "_doc",
    "_id": "2",
    "_version": 1,
    "result": "not_found",
    "_shards": {
      "total": 2,
      "successful": 1,
      "failed": 0
    },
    "status": 404,
    "_seq_no" : 1,
    "_primary_term" : 2
  }
},
```

```
{
  "create": {
    "_index": "test",
    "_type": "_doc",
    "_id": "3",
    "_version": 1,
    "result": "created",
    "_shards": {
      "total": 2,
      "successful": 1,
      "failed": 0
    },
    "status": 201,
    "_seq_no" : 2,
    "_primary_term" : 3
  }
}
```

```
{
  "update": {
    "_index": "test",
    "_type": "_doc",
    "_id": "1",
    "_version": 2,
    "result": "updated",
    "_shards": {
      "total": 2,
      "successful": 1,
      "failed": 0
    },
    "status": 200,
    "_seq_no" : 3,
    "_primary_term" : 4
  }
}
]
}
```

Bulk API

/_bulk
/{index} /_ bulk
/{index} /{type} /_bulk

1. 協同節點只負責解析 action_meta_data 部分
2. 單個操作的失敗不會影響剩餘的操作
3. 單次 bulk 操作數量沒有定論，須自行找到最佳大小
4. 如果使用HTTP API，請確保客戶端不發送HTTP塊（http chunks），因為這會降低速度。(header `Transfer-Encoding：chunked`)

PS: 一次太多操作會增加 CUP 使用率

Bulk API

Optimistic Concurrency Control

Versioning : 支援 version_type

Routing : 可單獨設定

Wait For Active Shards : 支援 wait_for_active_shards 參數

Refresh：只有收到 bulk request 的 shard 才會受到影響

Bulk API

Update

retry_on_conflict 可各自設定

```
POST _bulk
{ "update" : {"_id" : "1", "_type" : "_doc", "_index" : "index1", "retry_on_conflict" : 3} }
{ "doc" : {"field" : "value"} }
{ "update" : { "_id" : "0", "_type" : "_doc", "_index" : "index1", "retry_on_conflict" : 3} }
{ "script" : { "source": "ctx._source.counter += params.param1", "lang" : "painless", "params" : {"param1" : 1}},
"upsert" : {"counter" : 1}}
{ "update" : {"_id" : "2", "_type" : "_doc", "_index" : "index1", "retry_on_conflict" : 3} }
{ "doc" : {"field" : "value"}, "doc_as_upsert" : true }
{ "update" : {"_id" : "3", "_type" : "_doc", "_index" : "index1", "_source" : true} }
{ "doc" : {"field" : "value"} }
{ "update" : {"_id" : "4", "_type" : "_doc", "_index" : "index1"} }
{ "doc" : {"field" : "value"}, "_source": true}
```

Security

elasticsearch.yml

rest.action.multi.allow_explicit_index: true

預設是 true，body 內設定的 index 可以覆蓋 url 上的 index。
反之，false

Partial responses

為確保快速響應，如果一個或多個分片失敗，多獲取 API 將以部分結果作為響應。有關詳細信息，請參閱分片故障。

Reindex API

Reindex API

從 來源index 複製資料到另一個 目標 index

使用 Reindex API 需要所有 document 啟用 _source

目標 index 不存在會自動建立

但 Reindex 不會嘗試設定 index, 也不會複製 來源index 的設定。

在運行 Reindex 之前要先設置 目標 index, 包括設置 mapping, shards, replicas 等。

Reindex API

```
POST _reindex
{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter"
  }
}
```


```
{
  "took" : 147,
  "timed_out": false,
  "created": 120,
  "updated": 0,
  "deleted": 0,
  "batches": 1,
  "version_conflicts": 0,
  "noops": 0,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": -1.0,
  "throttled_until_millis": 0,
  "total": 120,
  "failures" : [ ]
}
```

Reindex API

就像 _update_by_query，reindex 操作也是先取得 來源index 快照

目標 index 與 來源index 必須不同

設置 version_type:”internal” (預設值)，會直接覆蓋相同的 type, id 資料

設置 version_type:”external_gt”，遇到 id 已存在時，可以保留 目標 index 新版本的資料 (比較兩者，取版本最大的)

```
POST _reindex
{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter",
    "version_type": "internal"
  }
}
```

Reindex API

設置 op_type = create，可以只建立不存在的 document，但會導致所有存在的 document 都會發生版本衝突

```
POST _reindex
{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter",
    "op_type": "create"
  }
}
```

設置 "conflicts": "proceed" 確保執行完畢

```
POST _reindex
{
  "conflicts": "proceed",
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter",
    "op_type": "create"
  }
}
```

Reindex API

可以下查詢，這樣將只複製符合查詢的資料

```
POST _reindex
{
  "source": {
    "index": "twitter",
    "type": "_doc",
    "query": {
      "term": {
        "user": "kimchy"
      }
    }
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

Reindex API

可以複製多個 index 與 type

```
POST _reindex
{
  "source": {
    "index": ["twitter", "blog"],
    "type": ["_doc", "post"]
  },
  "dest": {
    "index": "all_together",
    "type": "_doc"
  }
}
```

注意：Reindex API不會處理ID衝突，因此最後編寫的文檔將“獲勝”，但順序通常不可預測，因此依賴此行為並不是一個好主意。相反，請確保使用script使ID唯一。

Reindex API

也可以設定 size 要複製的數量

```
POST _reindex
{
  "size": 1,
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

Reindex API

可以設定 sort 排序，但這將導致 sroll 效率降低。

```
POST _reindex
{
  "size": 10000,
  "source": {
    "index": "twitter",
    "sort": { "date": "desc" }
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

Reindex API

也支援 _source

```
POST _reindex
{
  "source": {
    "index": "twitter",
    "_source": ["user", "title"]
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

Reindex API

支援 script，甚至支援對 document 的 metadata 操作 (ex: _version)，update_by_query 就不支援

```
POST _reindex
{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter",
    "version_type": "external"
  },
  "script": {
    "source": "if (ctx._source.foo == 'bar') {ctx._version++; ctx._source.remove('foo')}",
    "lang": "painless"
  }
}
```

Reindex API

ctx.op = "noop"

ctx.op = "delete"

請小心注意，你可以更改

- _id
- _type
- _index
- _version
- _routing

Reindex API

Routing

keep
　　將針對每個匹配發送的批量請求的路由設置為匹配上的路由。(default)

discard
　　將針對每個匹配發送的批量請求的路由設置為null。

=<some text>
　　將針對每個匹配發送的批量請求的路由設置為=之後的所有文本。

```
POST _reindex
{
  "source": {
    "index": "source",
    "query": {
      "match": {
        "company": "cat"
      }
    }
  },
  "dest": {
    "index": "dest",
    "routing": "=cat"
  }
}
```

Reindex API

也可以設置 Ingest Node

```
POST _reindex
{
  "source": {
    "index": "source"
  },
  "dest": {
    "index": "dest",
    "pipeline": "some_ingest_pipeline"
  }
}
```

Reindex API

Reindex from Remote

host 參數必須包含 host 和 port
uri 可選填。ex: http://hhost:9200/proxy

對 remote elasticsearch 的基本身份驗證

username
password

使用基本身份驗證時 務必使用 https，否則密碼將以純文本格式發送。

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "http://otherhost:9200",
      "username": "user",
      "password": "pass"
    },
    "index": "source",
    "query": {
      "match": {
        "test": "data"
      }
    }
  },
  "dest": {
    "index": "dest"
  }
}
```

Reindex API

必須將 reindex.remote.whitelist 在 elasticsearch.yaml 中將 remote host 明確列入白名單。

它可以設置為允許的遠程主機和端口組合的逗號分隔列表（例如 otherhost:9200，another:9200，127.0.10.*:9200，localhost:*）。

白名單忽略 uri 僅使用 host 和 port，例如：

```
reindex.remote.whitelist: "otherhost:9200, another:9200, 127.0.10.*:9200, localhost:*"
```

reindex 不能向前兼容版本，例如 從 7.x cluster reindx 到 6.x cluster 是不行的。

遠端 reindex 也不支援 manual 與 automatic slicing.

Reindex API

Reindx 使用 on-heap buffer， 預設最大上限是 100mb。

如果單一文檔資料過大，請縮小 batch size

```
POST _reindex
{
  "size":100,
  "source": {
    "remote": {
      "host": "http://otherhost:9200"
    },
    "index": "source",
    "size": 10,
    "query": {
      "match": {
        "test": "data"
      }
    }
  },
  "dest": {
    "index": "dest"
  }
}
```

Reindex API

也可以設置 socket_timeout 來限制讀取超時

並使用 connect_timeout 來限制連線超時

兩者預設都是 30 秒

```
POST _reindex
{
  "source": {
    "remote": {
      "host": "http://otherhost:9200",
      "socket_timeout": "1m",
      "connect_timeout": "10s"
    },
    "index": "source",
    "query": {
      "match": {
        "test": "data"
      }
    }
  },
  "dest": {
    "index": "dest"
  }
}
```

Configuring SSL parameters

elasticsearch.yml 文件中指定

Reindex API

URL Parameters

pretty

refresh : 會刷新該 index 所有 shards, 支援 wait_for

wait_for_completion

wait_for_active_shards

timeout

scroll : 默認 5 分鐘

requests_per_second

Reindex API

Response body


```
{
  "took": 639,
  "timed_out": false,
  "total": 5,
  "updated": 0,
  "created": 5,
  "deleted": 0,
  "batches": 1,
  "noops": 0,
  "version_conflicts": 2,
  "retries": {
    "bulk": 0,
    "search": 0
  },
  "throttled_millis": 0,
  "requests_per_second": 1,
  "throttled_until_millis": 0,
  "failures": [ ]
}
```

Reindex API

Works with the Task API

GET _tasks?detailed=true&actions=*reindex


```
{
 "nodes" : {
  "r1A2WoRbTwKZ516z6NEs5A" : {
   "name" : "r1A2WoR",
   "transport_address" : "127.0.0.1:9300",
   "host" : "127.0.0.1",
   "ip" : "127.0.0.1:9300",
   "attributes" : {
    "testattr" : "test",
    "portsfile" : "true"
   },
```



```
"tasks" : {
 "r1A2WoRbTwKZ516z6NEs5A:36619" : {
  "node" : "r1A2WoRbTwKZ516z6NEs5A",
  "id" : 36619,
  "type" : "transport",
  "action" : "indices:data/write/reindex",
  "status" : {
   "total" : 6154,
   "updated" : 3500,
   "created" : 0,
   "deleted" : 0,
   "batches" : 4,
   "version_conflicts" : 0,
   "noops" : 0,
   "retries": {
    "bulk": 0,
    "search": 0
   },
   "throttled_millis": 0,
   "requests_per_second": -1,
   "throttled_until_millis": 0
  },
  "description" : "",
  "start_time_in_millis": 1535149899665,
  "running_time_in_nanos": 5926916792,
  "cancellable": true,
  "headers": {}
```


Works with the Cancel Task API

Rethrottling

Reindex API

Slicing

Automatic slicing

```
POST _reindex?slices=5&refresh
{
  "source": {
    "index": "twitter"
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

Manual slicing

```
POST _reindex
{
  "source": {
    "index": "twitter",
    "slice": {
      "id": 0,
      "max": 2
    }
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

```
POST _reindex
{
  "source": {
    "index": "twitter",
    "slice": {
      "id": 1,
      "max": 2
    }
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

Reindex API

Reindex to change the name of a field

移除 flag 欄位，改成 tag

```
POST _reindex
{
  "source": {
    "index": "test"
  },
  "dest": {
    "index": "test2"
  },
  "script": {
    "source": "ctx._source.tag = ctx._source.remove(\"flag\")"
  }
}
```

Reindex API

Reindexing many indices

使用 bash 腳本

```
for index in i1 i2 i3 i4 i5; do
  curl -HContent-Type:application/json -XPOST localhost:9200/_reindex?pretty -d'{
    "source": {
      "index": "'$index'"
    },
    "dest": {
      "index": "'$index'-reindexed"
    }
  }'
done
```

Reindex API

Reindex daily indices

可以結合 Painless 來做每日 reindex

假設有包含以下文檔的 index :

```
PUT metricbeat-2016.05.30/_doc/1?refresh
{"system.cpu.idle.pct": 0.908}
PUT metricbeat-2016.05.31/_doc/1?refresh
{"system.cpu.idle.pct": 0.105}
```

Reindex API

```
POST _reindex
{
  "source": {
    "index": "metricbeat-*"
  },
  "dest": {
    "index": "metricbeat"
  },
  "script": {
    "lang": "painless",
    "source": "ctx._index = 'metricbeat-' + (ctx._index.substring('metricbeat-'.length(), ctx._index.length())) + '-1'"
  }
}
```

下面的腳本從索引名稱中提取日期，並創建一個附加了-1的新索引。來自metricbeat-2016.05.31的所有數據將重新編入metricbeat-2016.05.31-1。

Reindex API

Extracting a random subset of an index

也可以隨機取值來寫入 index 跟 _search 一樣

```
POST _reindex
{
  "size": 10,
  "source": {
    "index": "twitter",
    "query": {
      "function_score" : {
        "query" : { "match_all": {} },
        "random_score" : {}
      }
    },
    "sort": "_score"
  },
  "dest": {
    "index": "random_twitter"
  }
}
```

Term Vectors

Term Vectors

及時回傳 terms 的計算結果，可以以關閉即時計算。

GET /twitter/_doc/1/_termvectors

GET
/page_term_relation/page_term_relation/QcwuZIMBZBMD5pJGTuS1/_termvectors?fields=product_description

Term Vectors


```
{
  "_index" : "page_term_relation",
  "_type" : "page_term_relation",
  "_id" : "QcwuZIMBZBMD5pJGTuS1",
  "_version" : 1,
  "found" : true,
  "took" : 180,
  "term_vectors" : {
    "product_description" : {
      "field_statistics" : {
        "sum_doc_freq" : 25854,
        "doc_count" : 112,
        "sum_ttf" : 45941
      }
```

```
"terms" : {
  "24" : {
    "term_freq" : 1,
    "tokens" : [
      {
        "position" : 82,
        "start_offset" : 90,
        "end_offset" : 92
      }
    ]
  },
  "60" : {
    "term_freq" : 1,
    "tokens" : [
      {
        "position" : 86,
        "start_offset" : 96,
        "end_offset" : 98
      }
    ]
  },
```

Term Vectors

Return values

可以請求三種類型的值：詞項信息（term information），詞項統計（term statistics）和字段統計（field statistics）。

預設回傳 **term information** & **field statistics**

Term information

- term_freq 字段中詞項頻率（一直返回）
- position 詞項位置 (positions: true)
- start_offset 開始和 end_offset 結束偏移(offsets: true)
- payload 詞項負載（payloads：true），作為base64編碼的字節

如果請求的信息不存在 index 中，則將盡可能即時計算。此外，可以為甚至不存在於Index中但由用戶提供的文檔計算術語向量。

使用 UTF-16 encoding 計算偏移量，如果要使用偏移量來取值，須確保文檔也是採用 UTF-16 encoding

Term Vectors

Term statistics

預設為 false，改為 true 將出現

ttf 總詞項頻率(詞項在所有文檔中出現的頻率)
doc_freq 文件頻率(包含當前詞項的文件數)

Field statistics

預設為 true，改為 false 將隱藏

sum_ttf 文檔計數(包含此字段的文檔數)
doc_count 文件頻率之和(該字段中所有詞項的 **doc_freq** 之和)
sum_doc_freq 總詞項頻率之和(該字段中每個詞項的 **ttf** 之和)

Term Vectors

Terms Filtering

max_num_terms : 每個字段必須返回最大詞數，默認為 25

min_term_freq : 忽略源文檔中頻率低於此頻率的詞，默認為 1

max_term_freq : 忽略源文檔中頻率高於此頻率的詞，默認為 無限制

min_doc_freq : 忽略至少在這麼多文檔中沒有出現的詞，默認為 1

max_doc_freq : 忽略超過這麼多文檔中出現的單詞。默認為無限制。

min_word_length : 最小字串長度，低於該字串長度將被忽略。默認為0。

max_word_length : 最大字串長度，高於該字串長度將被忽略。默認為無限制(0)。

Term Vectors

Behaviour

詞和字段統計數據不精準。

刪除的文檔不會被考慮在內。

僅從請求的文檔所在的 shard 檢索信息。因此，術語和字段統計僅用作相對度量，而絕對數量在此上下文中沒有意義。

默認情況下，在請求人工文檔的術語向量時，隨機選擇用於獲取統計數據的shard。僅使用 routing 來命中特定的分片。

Term Vectors

Example: Returning stored term vectors

```
GET /twitter/_doc/1/_termvectors
{
  "fields" : ["text"],
  "offsets" : true,
  "payloads" : true,
  "positions" : true,
  "term_statistics" : false,
  "field_statistics" : true
}
```

Term Vectors

Example: Generating term vectors on the fly

some_field_without_term_vectors 將自動生成 term vectors

```
GET /twitter/_doc/1/_termvectors
{
  "fields" : ["text", "some_field_without_term_vectors"],
  "offsets" : true,
  "positions" : true,
  "term_statistics" : true,
  "field_statistics" : true
}
```

Term Vectors

Example: Artificial documents

也可以人工文檔。_id 跟 doc 只能擇一。

```
GET /twitter/_doc/_termvectors
{
  "doc" : {
    "fullname" : "John Doe",
    "text" : "twitter test test test"
  }
}
```

Term Vectors

Per-field 分析器

```
GET /twitter/tweet/_termvectors
{
  "doc" : {
    "fullname" : "John Doe",
    "text" : "twitter test test test"
  },
  "fields": ["fullname"],
  "per_field_analyzer" : {
    "fullname": "keyword"
  }
}
```

fullname 會更改為 keyword 不作字串切分。

Term Vectors

Example: Terms filtering

```
GET /imdb/_doc/_termvectors
{
   "doc": {
     "plot": "When wealthy industrialist Tony Stark is forced to build an armored suit after a life-threatening incident, he ultimately decides to use its technology to fight against evil."
   },
   "term_statistics" : true,
   "field_statistics" : true,
   "positions": false,
   "offsets": false,
   "filter" : {
     "max_num_terms" : 3,
     "min_term_freq" : 1,
     "min_doc_freq" : 1
   }
}
```

Multi termvectors API

Multi termvectors API

```
POST /_mtermvectors
{
  "docs": [
    {
      "_index": "twitter",
      "_type": "_doc",
      "_id": "2",
      "term_statistics": true
    },
    {
      "_index": "twitter",
      "_type": "_doc",
      "_id": "1",
      "fields": [
        "fullname"
      ]
    }
  ]
}
```

```
POST /twitter/_mtermvectors
{
  "docs": [
    {
      "_type": "_doc",
      "_id": "2",
      "fields": [
        "fullname"
      ],
      "term_statistics": true
    },
    {
      "_type": "_doc",
      "_id": "1"
    }
  ]
}
```

```
POST /twitter/_doc/_mtermvectors
{
  "docs": [
    {
      "_id": "2",
      "fields": [
        "fullname"
      ],
      "term_statistics": true
    },
    {
      "_id": "1"
    }
  ]
}
```

Multi termvectors API

如果所有請求的文檔都在相同的索引並且具有相同的類型，並且參數是相同的，則可以簡化請求：

```
POST /twitter/_doc/_mtermvectors
{
    "ids" : ["1", "2"],
    "parameters": {
        "fields": [
            "fullname"
         ],
         "term_statistics": true
    }
}
```

Multi termvectors API

就像 termvectors API 一樣，也可以為用戶提供的文檔生成詞條向量。

所使用的 mapping 由 _index 和 _type 決定，優先於 uri 的設定。

```
POST /twitter/_doc/_mtermvectors
{
  "docs": [
    {
      "_index": "twitter2",
      "_type": "_doc",
      "doc" : {
        "fullname" : "twitter test test test"
      }
    },
    {
      "doc" : {
       "fullname" : "Another twitter test ..."
      }
    }
  ]
}
```

?refresh

?refresh

Empty string or `true`

操作發生後立即刷新相關的主副本分片 (不是整個index)，**只有** 在思考搜尋與 index 的關聯並實際驗證後，才有可能不降低效能。

`wait_for`

等待 index.refresh_interval 自動刷新才返回結果。如果期間有其他立即刷新 (refresh = true) 將導致提早返回。

`false` (the default)

不理會刷新與否直接返回。

?refresh

Choosing which setting to use

最好的做好是保持 false

如果不得已需要確保刷新，那麼你需要在

1. 向 Elasticsearch 施加更多負載 (true)
2. 等待更長時間響應 (wait_for)

之間做出選擇

Choosing which setting to use

1. 如果有大量更改操作，那 wait_for 比起 true 會節省更多，因為只會執行一次
2. true 創建效率較低的索引構造 (小分段)，成本較小，但之後必須將其合併到更高效的索引構造（較大的分段）中。
3. 永遠不要分開多個 refresh=wait_for 請求。請將它們整理成單個批量請求，Elasticsearch 將並行啟動它們並僅在它們全部完成時返回。
4. 如果刷新間隔設置為 -1，禁用自動刷新，則帶有 refresh=wait_for 的請求將無限期等待，除非有其他刷新。
5. 相反，將 index.refresh_interval 設置為比默認值更短的值（例如 200 毫秒）將使 refresh=wait_for 恢復得更快，但它仍然會生成低效的段。
6. refresh=wait_for 只影響它所在的請求，refresh=true 將影響其他正在進行的請求

?refresh

refresh=wait_for Can Force a Refresh

當已經有超過 index.max_refresh_listeners (defaults to 1000) 的請求數量，將視為直接刷新 (true)。

遇到這樣情況時 返回內容會出現 "forced_refresh": true.

對於 bulk 請求只佔一個請求量

?refresh

 # true

PUT /test/_doc/1?refresh

{"test": "test"}

PUT /test/_doc/2?refresh=true

{"test": "test"}

 # false

PUT /test/_doc/3

{"test": "test"}

PUT /test/_doc/4?refresh=false

{"test": "test"}

 # wait_for

PUT /test/_doc/4?refresh=wait_for

{"test": "test"}

Optimistic concurrency control

Optimistic concurrency control

Elasticsearch 是分佈式的。當創建、更新或刪除文檔時，必須將文檔的新版本複製到集群中的其他節點。Elasticsearch 也是異步和並發的，這意味著這些複製請求是並行發送的，並且可能會亂序到達目的地。

序列號(_seq_no)

序列號隨著每個操作而增加，因此新操作保證比舊操作具有更高的序列號。

Elasticsearch 可以使用操作的序列號來確保較新的文檔版本永遠不會被分配給它的序列號較小的更改覆蓋。

Optimistic concurrency control

設置 seq_no_primary_term 後，Search API 可以為每個結果返回 _seq_no 與 _primary_term 。

在 Index API 跟 Delete API 可以設置 if_seq_no 和 if_primary_term，來確保更新前不受其他更新影響，如果現有 _seq_no 或 _primary_term 跟參數不同，返回失敗。

```
PUT products/_doc/1567?if_seq_no=362&if_primary_term=2
{
   "product" : "r2d2",
   "details" : "A resourceful astromech droid",
   "tags": ["droid"]
}
```

Other references

Other references

文檔翻譯 注意版本

https://docs.kilvn.com/elasticsearch/

https://xiaoxiami.gitbook.io/elasticsearch/

https://github.com/zhangchichi/elasticsearch-doc-6.4

API 參數

https://elasticsearch-py.readthedocs.io/en/master/api.html

https://elasticsearch-py.readthedocs.io/en/6.8.2/api.html

https://www.elastic.co/guide/en/elasticsearch/client/javascript-api/6.x/_deletebyquery.html

awoo.ai

Thank You