

# 실무에 활용하는 Elasticsearch 검색엔진 구축 4기

## 2일차 : 색인 및 매핑

2018.06



# 오늘의 아젠다

데이터 색인

색인 운용 기법

매핑 설계



# 데이터 색인

# Elasticsearch 데이터 처리

- 엘라스틱서치는 주로 REST API를 통하여 데이터를 처리
- HTTP METHOD는 GET, POST, PUT, DELETE, HEAD 등이 있음
- 엘라스틱서치의 REST API를 이용하기 위한 형태는 아래와 같음

```
curl -X{METHOD} http://host:port/{INDEX}/{TYPE}/{ID} -d '{DATA}'
```

## HTTP METHOD, CRUD, SQL 비교

HTTP METHOD	CRUD	SQL
GET	READ	SELECT
PUT	UPDATE	UPDATE
POST	CREATE	INSERT
DELETE	DELETE	DELETE



# RESTful이란?

REST는 웹의 창시자(HTTP) 중의 한 사람인 Roy Fielding의 2000년 논문에 의해서 소개  
현재의 아키텍처가 웹의 본래 설계의 우수성을 많이 사용하지 못하고 있다고 판단  
웹의 장점을 최대한 활용할 수 있는 네트워크 기반의 아키텍처를 소개  
그것이 바로 Representational state transfer (REST)

REST는 요소로는 크게 리소스, 메서드, 메시지

```
HTTP POST , http://myweb/users/  
{  Method      Resource (URI)  
  "users":{  
    "name":"terry"  
  }  Message  
}
```

메서드	의미	Idempotent
POST	Create	No
GET	Select	Yes
PUT	Update	Yes
DELETE	Delete	Yes

CRUD를 메서드로 표현

## 데이터 입력

- POST나 PUT 메서드를 이용해 입력
- 처음 입력 시 created는 true
- 도큐먼트 ID를 생략하고 입력 가능
- 임의 ID 입력은 POST 메서드로만 가능

```
$ curl -XPUT http://localhost:9200/books/book/1 -d '{
  "title": "Elasticsearch Guide",
  "author": "Kim",
  "date": "2015-06-25",
  "pages": 250
}'
```

## 데이터 삭제

- DELETE 메서드를 이용해 삭제
- 검색을 이용해 선별적 삭제 가능
- 타입과 인덱스 단위로 삭제 가능
- 인덱스 단위로 삭제할 경우 타입, 도큐먼트가 모두 삭제됨
- 도큐먼트는 삭제 후에도 메타 데이터를 그대로 보존

```
$ curl -XDELETE http://localhost:9200/books/book/AU4eDMKk5_2R8Icukymj
```

# 데이터 갱신

- PUT 메서드를 이용해 갱신
- 갱신 시 created는 false
- \_version 필드가 증가
- 기존 도큐먼트는 보관되지 않으며 삭제되고 새로 쓰임
- 이전 버전으로 변경 불가능

```
$ curl -XPOST http://localhost:9200/books/book/ -d '{
  "title": "Elasticsearch Guide",
  "author": ["Kim", "Lee"],
  "date": "2015-06-25",
  "pages": 300
}'
```





## 벌크 입력 (Bulk Insert)

```
curl -XPOST localhost:9200/_bulk -d '
```

```
{ "index" : { "_index" : "books", "_type" : "book", "_id" : "1" } }  
{ "title" : "Elasticsearch Guide", "author" : "Kim", "pages" : 250 }  
{ "index" : { "_index" : "books", "_type" : "book", "_id" : "2" } }  
{ "title" : "Elasticsearch Easy Guide", "author" : "Lee", "pages" : 300 }  
{ "delete" : { "_index" : "books", "_type" : "book", "_id" : "1" } }  
{ "update" : { "_index" : "books", "_type" : "book", "_id" : "2" } }  
{ "doc" : { "date" : "2014-05-01" } }  
{ "create" : { "_index" : "books", "_type" : "book", "_id" : "3" } }  
{ "title" : "Elasticsearch Guide II", "author" : "Park", "pages" : 400 }  
,
```

## 실습해보기

JSON으로 직접 데이터를 입력,수정,삭제 해보기  
cerebro 혹은 크롬 Postman 이용

실습파일 참조

Elasticsearch client

Low level Client

High Level Client

Transport Client

JEST : Apache HttpComponents 프로젝트를 이용해서 만든 경량 클라이언트

Spring Data Elasticsearch : 스프링 데이터 레파지토리를 추상화하여 영속화 저장소를 위한 데이터 액세스  
레이어 구성



# Java High Level REST Client

## Document APIs

- 단일 문서의 색인
- 단일 문서 조회
- 문서 존재여부
- 문서의 삭제
- 문서의 갱신
- 벌크 색인

## Search APIs

- 문서 검색
- 문서 검색 스크롤링
- 멀티 검색

## Miscellaneous APIs

- 기본 정보 조회
- 핑 조회

## Indices APIs

- 색인의 생성
- 색인의 삭제
- 색인 존재 여부
- 색인의 열고 닫음
- 색인의 분할
- 색인의 축소
- 색인의 분할
- 색인의 갱신 (Refresh)
- 색인의 쓰기 (Flush)
- 강제 병합
- 매핑 입력 등등

## Cluster APIs

- 클러스터의 세팅을 업데이트

## Building Queries

- 검색 쿼리를 생성

## Building Aggregations

- 어그리게이션을 생성

# 실습해보기

github에서 소스체크 아웃

# 매핑 설계

# 매핑이란?

ElasticSearch 별도스키마 설정이 없이도 자동으로 자료구조를 이해하고 저장 (동적 매핑)

특정 인덱스의 명시적 구조를 정의

한 인덱스에 매핑을 여러개 만들수 있으나 필드명은 중복 불가

매핑은 필드의 추가는 가능, 수정 삭제는 불가능

예외)

신규 속성은 객체 데이터타입 필드에 추가 가능

신규 다중필드는 기존 필드에 추가 가능

doc\_values 는 비활성화 될 수 있지만, 다시 활성화 불가

ignore\_above는 수정 가능

기존 매핑과 충돌의 경우 무시하는 옵션이 있었으나 Deprecated (~~ignore\_conflicts~~)

매핑 삭제 옵션은 제공X (~~2.0부터 DELETE mapping API 없음~~)



# 매핑 입력 및 조회

인덱스 생성시 입력

```
curl -XPOST ...:9200/my_index -d '{
  "settings" : {
    # .. index settings
  },
  "mappings" : {
    "my_type" : {
      # mapping for my_type
    }
  }
}'
```

매핑 API를 이용한 입력

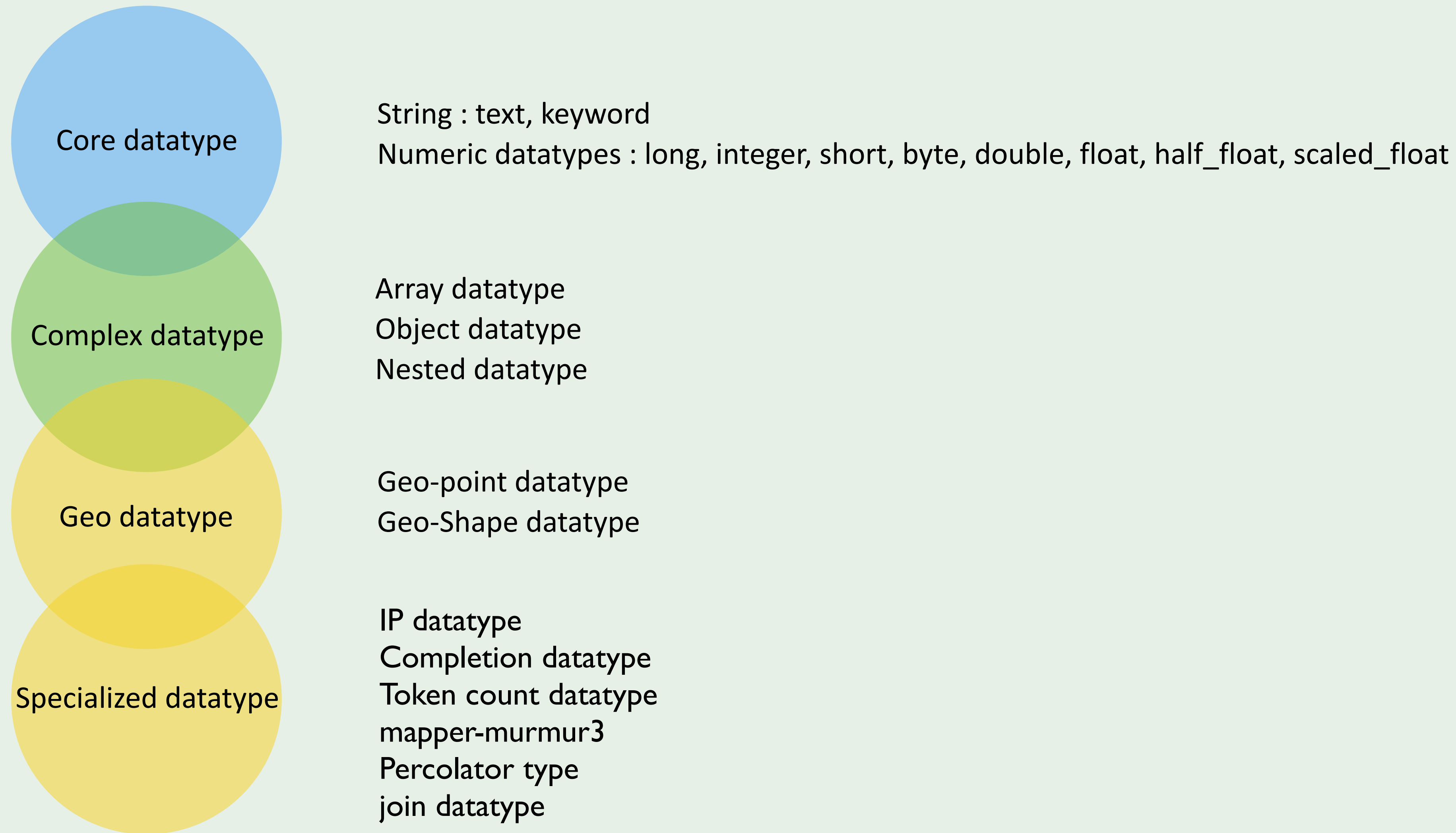
```
$ curl -XPUT 'http://localhost:9200/my_index/my_type/_mapping' -d '
{
  "my_type" : {
    # mapping for my_type
  }
}'
```

조회

```
curl -XGET 'localhost:9200/news/_mapping/article?pretty'
```



# 필드 데이터타입 (Mapping datatype)



# 메타 필드

`_all` 필드 : 모든 필드를 하나로 합쳐서 색인함 (필드명 없는 검색에 대응)

`_field_names` 필드 : 필드 이름 자체를 검색하고자 할때

`_id` field : 키값 저장

`_index` field : 다중 인덱스 질의시 어떤 인덱스의 값인지 알기 위함

`_meta` field : 추가 메타 정보 입력

`_routing` field : 라우팅 정보 저장

`_source` field : 원본 정보 저장

`_type` field : 어떤 타입 정보인지 알기 위함 (Deprecated)



# 매핑 파라미터(Mapping parameters)

analyzer : 분석기 지정 (Default standard)

boost : 필드별 가중치

fielddata : 집계 정렬을 위해 필드데이터를 메모리에 올릴것인지 여부

fields : 집계나 정렬을 위해 같은 값을 다른 분석기로 분석할지 여부

index : 색인 할것인지 여부(default false)

index\_options : 색인시 텀,빈도,포지션,오프셋 값을 저장할것인지 여부

- docs
- freqs
- positions (근접 연산자 및 phrase 검색시, 스트링 필드의 기본 값)
- offsets (하일라이팅)

norms : 스코어링에서 문서의 길이를 적용시킬것인지

store : 데이터를 저장 할것인지 여부 (\_source 와 별도)

search\_analyzer : 검색시 분석기를 별도로 지정할 것인지 여부

term\_vector : 텀벡터를 저장할 것인지 여부

copy\_to : 필드의 원본을 복사하여 다른 필드로 전송

similarity : 스코어 유사도 계산 방식을 정함 (classic/bm25/bool)



# Array datatype VS Object datatype VS Nested datatype

```
{
  "title": "Nest eggs",
  "body": "Making your money work...",
  "tags": [ "cash", "shares" ],
  "comments": [
    {
      "name": "John Smith",
      "comment": "Great article",
      "age": 28,
      "stars": 4,
      "date": "2014-09-01"
    },
    {
      "name": "Alice White",
      "comment": "More like this please",
      "age": 31,
      "stars": 5,
      "date": "2014-10-22"
    }
  ]
}
```

GET /\_search

```
{
  "query": {
    "bool": {
      "must": [
        { "match": { "name": "Alice" } },
        { "match": { "age": 28 } }
      ]
    }
  }
}
```

← 검색

```
{
  "title": [ eggs, nest ],
  "body": [ making, money, work, your ],
  "tags": [ cash, shares ],
  "comments.name": [ alice, john, smith, white ],
  "comments.comment": [ article, great, like, more, please, this ],
  "comments.age": [ 28, 31 ],
  "comments.stars": [ 4, 5 ],
  "comments.date": [ 2014-09-01, 2014-10-22 ]
}
```

cross-object 매칭 발생

서로의 데이터가 독립적이지 못하고  
서로 합해져서 크로스해서 조건이 걸리게 됨

# Array datatype VS Object datatype VS Nested datatype

이러한 문제를 풀기 위해 Object 가 디자인

```
{
  "mappings": {
    "my_type": {
      "properties": {
        "region": {
          "type": "keyword"
        },
        "manager": {
          "properties": {
            "age": { "type": "integer" },
            "name": {
              "properties": {
                "first": { "type": "text" },
                "last": { "type": "text" }
              }
            }
          }
        }
      }
    }
  }
}
```

```
PUT my_index/my_type/1
{
  "region": "US",
  "manager": {
    "age": 30,
    "name": {
      "first": "John",
      "last": "Smith"
    }
  }
}
```

문제는 이것으로는 1개 값 밖에 데이터를 입력 못함

# Array datatype VS Object datatype VS Nested datatype

이러한 문제를 풀기 위해 nested 데이터 타입을 사용

## 1) 매핑

```
PUT my_index
{
  "mappings": {
    "_doc": {
      "properties": {
        "user": {
          "type": "nested"
        }
      }
    }
  }
}
```

## 2) 입력

```
PUT my_index/_doc/1
{
  "group" : "fans",
  "user" : [
    {
      "first" : "John",
      "last" : "Smith"
    },
    {
      "first" : "Alice",
      "last" : "White"
    }
  ]
}
```

## 3) 검색

```
{
  "query": {
    "nested": {
      "path": "user",
      "query": {
        "bool": {
          "must": [
            { "match": { "user.first": "Alice" } }
            { "match": { "user.last": "White" } }
          ]
        }
      }
    }
  }
}
```

# Array datatype VS Object datatype VS Nested datatype

```
{ (1)
  "comments.name": [ john, smith ],
  "comments.comment": [ article, great ],
  "comments.age": [ 28 ],
  "comments.stars": [ 4 ],
  "comments.date": [ 2014-09-01 ]
}
{ (2)
  "comments.name": [ alice, white ],
  "comments.comment": [ like, more, please, this ],
  "comments.age": [ 31 ],
  "comments.stars": [ 5 ],
  "comments.date": [ 2014-10-22 ]
}
{ (3)
  "title": [ eggs, nest ],
  "body": [ making, money, work, your ],
  "tags": [ cash, shares ]
}
```

```
PUT /my_index
{
  "mappings": {
    "blogpost": {
      "properties": {
        "comments": {
          "type": "nested",
          "properties": {
            "name": { "type": "string" },
            "comment": { "type": "string" },
            "age": { "type": "short" },
            "stars": { "type": "short" },
            "date": { "type": "date" }
          }
        }
      }
    }
  }
}
```

## nested query 사용

```
GET /my_index/blogpost/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "title": "eggs" } }, (1)
        {
          "nested": {
            "path": "comments", (2)
            "query": {
              "bool": {
                "must": [
                  { "match": { "comments.name": "john" } }, (3)
                  { "match": { "comments.age": 28 } }
                ]
              }
            }
          }
        }
      ]
    }
  }
}
```



# 매핑 문법

```
{
  "mappings": {
    "_doc": {
      "properties": {
        "user": {
          "type": "text",
          "boost": 2,
          "doc_values": false,
          ...
          ...
          ...
        }
      }
    }
  }
}
```



```
"type": "text",
"boost": 2,
"doc_values": false,
"ignore_above": 20,
"fielddata": true,
"index": false,
"norms": false,
"analyzer": "autocomplete",
"search_analyzer": "standard",
"similarity": "classic",
"store": true,
"fields": {
  "english": {
    "type": "text",
    "analyzer": "english"
  }
}
,
"properties": {
  "age": { "type": "integer" },
  "name": {
    "properties": {
      "first": { "type": "text" },
      "last": { "type": "text" }
    }
  }
}
```

Object형





# 매핑 설계시 주의점

1) 동적 매핑은 지양

"index.mapper.dynamic": false

2) \_all 기능 disable

3) text 필드 사용 자제 (자연어 검색이 필요한 필드 이외에.)

4) \_source 기능 disable (성능 개선 효과가 크진 않고 디스크 사용량에 관련)

5) 검색대상이 되는 필드와 그렇지 않은 필드를 명확히 구별

indexed: false (검색이 필요 없는 필드 대상)

6) 집계 정렬의 대상이 되는 필드와 그렇지 않은 필드를 명확히 구별

doc\_values : false (집계 및 정렬이 필요 없는 필드 대상)

7) refresh 타임 조정

## 실습해보기

매핑을 입력 삭제 수정을 실시해 본다.

실습파일 참조

# 클러스터 및 색인 운용 기법

# 병합정책(Merge Policy)

새로 인덱싱된 문서는 우선 Lucene의 IndexWriter에 의해 RAM에 보관

주기적으로 RAM 버퍼가 가득 차거나 Elasticsearch의 flush 또는 refresh를 트리거하면 이 문서들은 디스크의 새 세그먼트에 쓰여짐

세그먼트가 너무 많아지게 되면 병합 정책 및 스케줄러에 따라 병합이 수행

## MergeFactor에 의한 세그먼트의 병합 전략

세그먼트를 병합하는 빈도와 크기를 제어

<http://blog.mikemccandless.com/2011/02/visualizing-lucenes-segment-merges.html>



## 강제 병합

```
curl -XPOST 'localhost:9200/news/_forcemerge?pretty'
```

max\_num\_segments : 병합될 세그먼트 수

only\_expunge\_deletes : 삭제 필드가 있는 세그먼트만 실행

flush : 색인후 flush할지 여부 (기본값 true)

# 리밸런싱(rebalancing)

Elasticsearch에서 적절히 각 노드의 균형을 맞추기 위해 샤드를 재배포하는 것

리밸런싱 정지

```
"transient" : {  
  "cluster.routing.allocation.enable" : "none"  
}
```

리밸런싱 재개

```
"transient" : {  
  "cluster.routing.allocation.enable" : "all"  
}
```



# 별칭(Alias)

alias를 사용하는 경우

- 1) 색인을 중단 없이 교체
- 2) 많은 인덱스의 대표 이름을 붙일때

별칭 입력

POST /\_aliases

```
{
  "actions": [
    { "add": { "index": "test1", "alias": "alias1" } }
  ]
}
```

별칭 제거

POST /\_aliases

```
{
  "actions": [
    { "remove": { "index": "test1", "alias": "alias1" } }
  ]
}
```

조건을 설정 별칭 생성

POST /\_aliases

```
{
  "actions": [
    {
      "add": {
        "index": "test1",
        "alias": "alias2",
        "filter": { "term": { "user": "kimchy" } }
      }
    }
  ]
}
```

POST /\_aliases

```
{
  "actions": [
    {
      "add": {
        "index": "test",
        "alias": "alias1",
        "routing": "1"
      }
    }
  ]
}
```

# 템플릿(Template)

색인을 생성시 미리 정의된 색인 구조를 저장해두는 기능 (색인 세팅값, 매핑값)

템플릿 생성

```
PUT _template/template_1
{
  "index_patterns": ["te*", "bar*"],
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "type1": {
      "_source": {
        "enabled": false
      },
      "properties": {
        "host_name": {
          "type": "keyword"
        },
        "created_at": {
          "type": "date",
          "format": "EEE MMM dd HH:mm:ss Z YYYY"
        }
      }
    }
  }
}
```

템플릿 삭제

```
DELETE /_template/template_1
```

템플릿 조회

```
GET /_template/template_1
```

PUT /\_template/template\_1

```
{
  "index_patterns": ["*"],
  "order": 0,
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "type1": {
      "_source": { "enabled": false }
    }
  }
}
```

두 템플릿에 중복 적용될 경우  
우선순위 부여

PUT /\_template/template\_2

```
{
  "index_patterns": ["te*"],
  "order": 1,
  "settings": {
    "number_of_shards": 1
  },
  "mappings": {
    "type1": {
      "_source": { "enabled": true }
    }
  }
}
```





# 다이나믹 템플릿(Dynamic Template)

동적으로 추가 된 필드에 적용 할 수있는 사용자 지정 매핑을 정의

```
{
  "mappings": {
    "_doc": {
      "dynamic_templates": [
        {
          "integers": {
            "match_mapping_type": "long",
            "mapping": {
              "type": "integer"
            }
          }
        },
        {
          "strings": {
            "match_mapping_type": "string",
            "mapping": {
              "type": "text",
              "fields": {
                "raw": {
                  "type": "keyword",
                  "ignore_above": 256
                }
              }
            }
          }
        }
      ]
    }
  }
}
```

```
{
  "mappings": {
    "_doc": {
      "dynamic_templates": [
        {
          "longs_as_strings": {
            "match_mapping_type": "string",
            "match": "long_*",
            "unmatch": "*_text",
            "mapping": {
              "type": "long"
            }
          }
        }
      ]
    }
  }
}
```

PUT my\_index/\_doc/1

```
{
  "long_num": "5",
  "long_text": "foo"
}
```

# 재색인( re-index)

하나의 색인에서 다른 색인으로 복사  
매핑이 변경될 경우 주로 사용

```
POST _reindex
{
  "source": {
    "index": "news"
  },
  "dest": {
    "index": "new_news"
  }
}
```

```
POST _reindex
{
  "source": {
    "index": "news",
    "type": "article",
    "query": {
      "term": {
        "user": "대한민국"
      }
    }
  },
  "dest": {
    "index": "new_twitter"
  }
}
```

특정 쿼리 조건을 부여 가능

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

## 실습해보기

JSON으로 직접 데이터를 입력,수정,삭제 해보기  
cerebro 혹은 크롬 Postman 이용

실습파일 참조

감사합니다.

다음회차 내용

Elasticsearch 분석기

1. Elasticsearch에서의 텍스트 데이터 처리 방법 및 분석기 설정
2. 한글 형태소 분석기 설치 및 운영
3. 형태소 분석기 플러그인 개발
  - Lucene Analyzer 구조
  - 자신만의 customized 형태소 분석기 개발을 위한 Elasticsearch의 플러그인 개발 과정