

Elasticsearch

8장 엘라스틱서치의 내부 동작 상세

8-1 엘라스틱 서치의 데이터 분산 처리 과정

새인 API 등의 쓰기 요청이 들어올 때 조정단계, 주 샤드 단계, 복제 단계의 3단계로 수행된다. 종료는 역순으로 이루어 진다.

조정단계

1. 클라이언트로부터 쓰기 작업 요청을 받는다.
2. 라우팅으로 적절한 주 샤드를 찾아 요청을 넘겨준다.
3. 주 샤드 단계 작업을 마치면 최초 요청을 수신했던 조정 노드에게 주 샤드 작업 결과 응답을 넘겨준다.
4. 클라이언트에게 응답을 돌려준다.

주 샤드

1. 작업 요청을 검증한다.
2. 주 샤드는 로컬에서 쓰기 작업을 수행하고 완료되면 **in-sync** 복제본에 병렬적으로 요청을 넘긴다.
3. 모든 복제본이 작업을 성공적으로 수행하고 주 샤드에 응답을 돌려준다.
4. 주 샤드가 작업 완료 응답을 보낸다.

복제단계

In-sync 복제본 샤드는 주 샤드에게 받은 요청을 로컬에서 수행하고 주 샤드에게 작업이 완료됐음을 보고한다.

in-sync 복제본: 마스터 노드가 관리하는 복제받을 샤드 목록

8-1 엘라스틱 서치의 데이터 분산 처리 과정

낙관성 동시성 제어

분산시스템 특성상 두 요청 중 어떤 요청이 먼저 복제본 샤드로 들어올지는 보장할 수 없다.

메시지 순서의 역전이 일어날 수 있다

views를 2로 색인하는 요청이 먼저 들어온 뒤에 **views**를 1로 색인하는 요청이 나중에 들어온다면 복제본에서는 최종값이 1로 역전될 수도 있다.

이러한 현상을 막기 위해 **_seq_no**가 존재한다.

_seq_no

_seq_no 는 각 주 샤드마다 들고 있는 시퀀스 숫자값이며 매 작업마다 1씩 증가한다.

- 엘라스틱서치는 문서를 색인할 때 이 값을 함께 저장한다.
- 문서가 색인된 상태에서 **_id** 값은 같은데 **_seq_no** 값은 더 작은 색인 요청을 받는다면 이전 버전의 요청을 늦게 들어온 것으로 판단하고 작업을 수행하지 않는다.

_primary_term

주 샤드를 들고 있는 노드에 문제가 발생하여 해당 노드가 클러스터에서 빠진다면 엘라스틱서치는 복제본 샤드 중 하나를 주 샤드로 지정한다.

- **_primary_term** 이라는 값으로 새로 임명된 주 샤드를 구분한다.
- 주 샤드가 새로 지정될 때 1씩 증가한다.

8-1 엘라스틱 서치의 데이터 분산 처리 과정

복제 요청이 역전되어 돌아왔다면 복제본 샤드는 `_seq_no` 값이 색인된 상태에서 `_id` 값은 같은데 `_seq_no` 값이 더 작은 `n`인 색인 요청을 받는다. 엘라스틱 서치는 이 경우 이전 버전의 요청을 늦게 들어온 것으로 판단하고 작업을 수행하지 않는다

- 문서에 색인할 때 `if_seq_no` 와 `if_primary_term` 에 각각 `_seq_no` 와 `_primary_term` 을 넣을 수 있다.
- 이 경우 문서의 `_seq_no` 와 `_primary_term` 이 지정한 값과 같을 때만 색인 작업이 수행된다.
- 클라이언트가 자신이 데이터를 확인했을 때와 완전히 동일한 상태인 경우에만 작업을 수행하고 싶을 때 이용한다.

버전

- `_version` 은 `_seq_no` 와 `_primary_term` 처럼 동시성을 제어하기 위한 메타데이터로 모든 문서마다 붙는다.
- `_version` 은 0이상 long 범위 이내의 정수여야 한다.
 - 1부터 시작해서 업데이트나 삭제 작업을 수행할 때마다 1씩 증가한다.
- `_seq_no`, `_primary_term` 과 다른 점은 클라이언트가 문서의 `_version` 값을 직접 지정할 수 있다.
 - 현재의 버전값보다 낮은 값으로 색인할 수 없다.
 - `version_type` 을 `external(external_gt)` 이나 `external_gte` 로 설정하면 클라이언트가 직접 `_version` 을 지정할 수 있다.
 - default `version_type` 은 `internal`

다른 스토리지에서 저장된 데이터의 버전을 따로 관리하고 있고 그 데이터를 엘라스틱서치로 받아 와서 2차 스토리지로 동기화하여 사용하는 경우 활용하기 좋다.

8-1 엘라스틱 서치의 데이터 분산 처리 과정

읽기 작업 시 엘라스틱서치 동작


1. 조정 노드(`Coordination Node`)는 클라이언트로부터 읽기 작업을 요청받는다.
2. 조정 노드는 라우팅을 통해 적절한 샤드를 찾아 요청을 넘긴다.
 - 쓰기 작업과 다르게 주 샤드가 아니라 복제본 샤드로 요청이 넘어갈 수 있다.
3. 요청을 넘겨받은 각 샤드는 로컬에서 읽기 작업을 수행한 뒤 그 결과를 조정 노드로 돌려준다.
4. 조정 노드는 이 결과를 모아서 클라이언트에게 응답한다.

체크포인트와 샤드 복구 과정

만약 문제가 생겨서 특정 노드가 재기동되었다면 그 노드가 들고 있던 샤드에 복구 작업이 진행된다.

- 복구 중인 샤드가 현재 주 샤드의 내용과 일치하는지를 파악할 필요가 있다.
- 주 샤드가 색인 작업이 있었다면 그 내용을 복제본 샤드에도 반영해야 한다.

`_primary_term` 과 `_seq_no` 를 조합하면 샤드와 샤드 사이에 어떤 반영 차이점이 있는지를 알 수 있다.

각 샤드는 로컬에 작업을 수행하고나면 몇 번 작업까지 순차적으로 수행을 완료했는지를  로컬 체크포인트 로 기록한다.

샤드 이력 보존(Shard History Retention Leases)

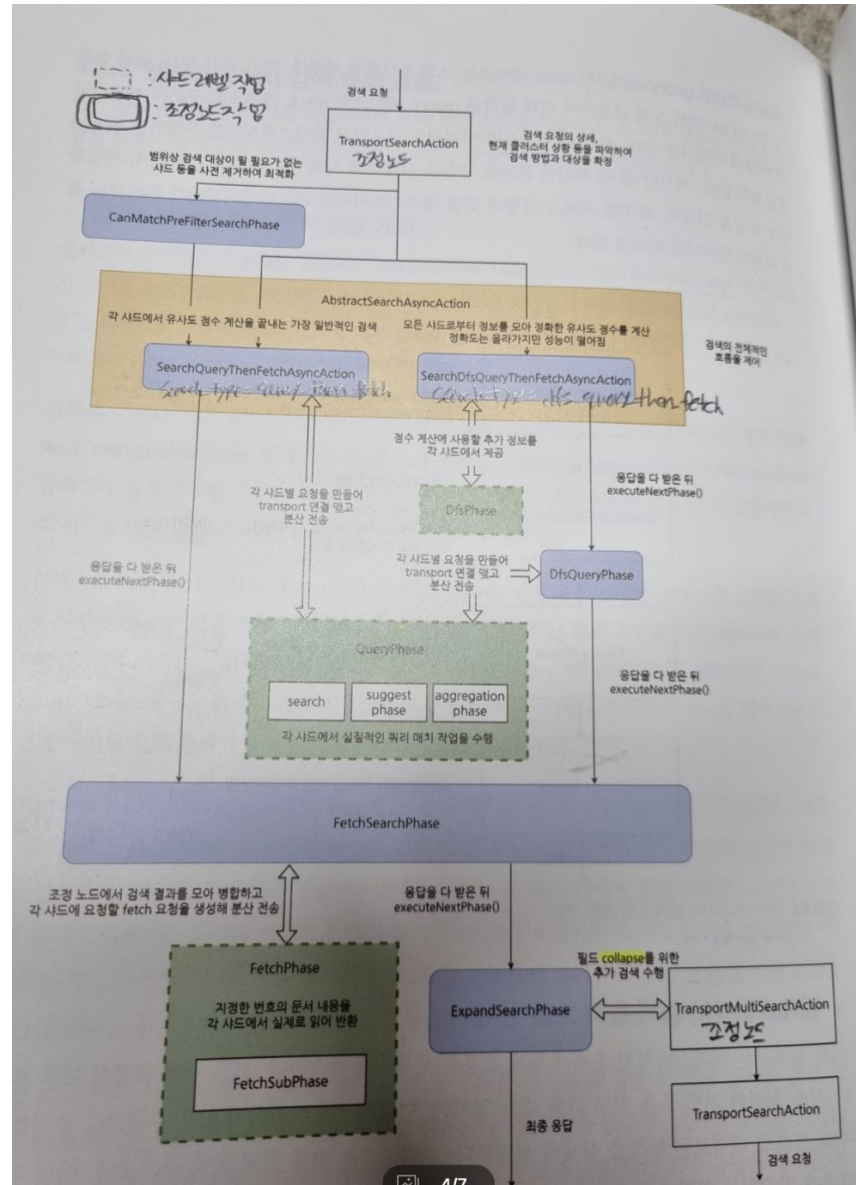
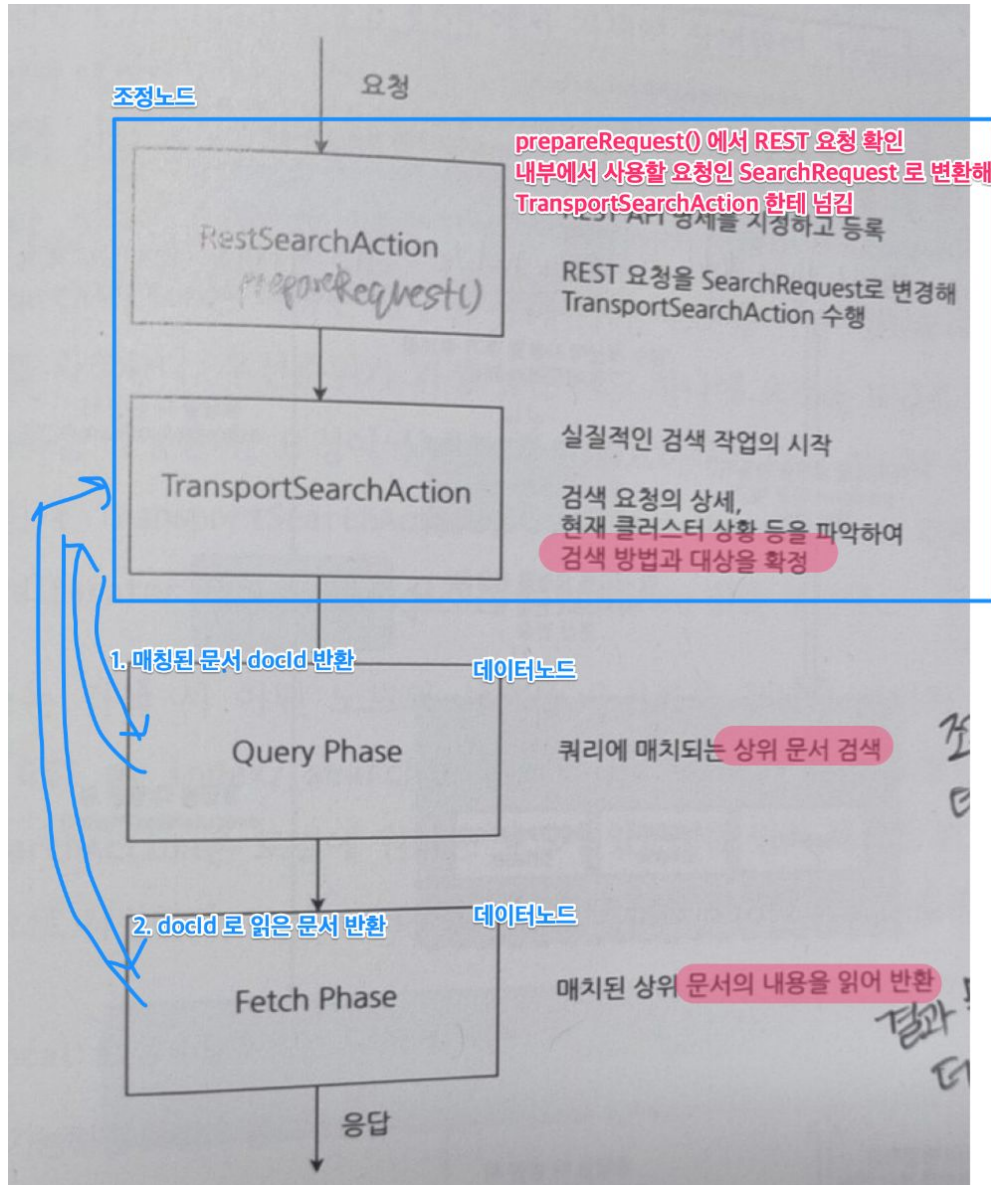
엘라스틱서치는 논리적 삭제를 도입했다.
최근 삭제한 문서를 일정 기간 보존해 두고 작업 재처리에 활용한다.

재처리할 내용을 추적하는 메커니즘

루씬의 세그먼트가 병합되는 도중에도 샤드 이력은 지정한 기간 동안은 보존된다.
(default: 12시간)

`index.soft_deletes.retention_lease.period`

8-2 엘라스틱 서치의 검색 동작 흐름



8-2 엘라스틱 서치의 검색 동작 흐름

TransportSearchAction

적절한 검색 방법과 대상(인덱스, 샤드)을 확정한다. 클러스터 간 검색이 필요한지도 체크한다.

<검색 대상 정하기>

- 인덱스명이 alias 나 와일드카드로 들어오면 정확한 인덱스 목록을 확정하고
- 어떤 샤드로 요청을 보낼 지를 확정한다.
- 정해진 샤드로 보냈을 때 실패할 수도 있으니 다음으로 보낼 샤드 순서 목록도 작성한다. (by `ShardIterator` 인터페이스)
 - `GET index/_search?preference=_local` 로 요청하면 맨 먼저 `preference` 값을 읽어 로컬 샤드를 최우선으로 작업 수행한다.
 - `_only_local`, `_local`, `_only_nodes`, `_prefer_nodes`, `_shards`
 - `preference` 가 없으면 adaptive replica selection 에 의해 응답속도, 소요 시간, 스레드 풀 상황등을 고려해 응답 빨리 줄 것같은 샤드를 고른다.
 - `cluster.routing.use_adaptive_replica_selection` : false 로 지정해 끌 수도 있음
- `pit(point in time)` 이 지정됐으면 처음부터 `pit` 값으로 검색 문맥 가져와 검색 대상 인덱스와 샤드를 가져온다.

8-2 엘라스틱 서치의 검색 동작 흐름

CanMatchPreFilterSearchPhase

특정 조건들을 만족하면 본격적인 검색 들어가기 전에 `CanMatchPreFilterSearchPhase` 를 거쳐 단 하나라도 매치될 가능성이 있는지 점검 및 최적화 작업을 수행한다.

- 검색 속도를 빠르게 하기 위한 최적화 작업이다.
- 조정노드에서 하는 점검이다
- 검색될 문서가 없다면 그 샤드는 검색 대상에서 제거한다.
- 점검 마치면 각 샤드별 점검 요청을 노드 단위로 묶어 transport 채널로 분산 전송하고, 요청 수신한 노드는 `SearchService.canMatch()` 실행해 또 점검한다.

*특정 조건

*

- `search_type` 이 `query_then_fetch` 여야 함 (각 샤드레벨에서 유사도 점수 계산 끝내기)
- 검색 대상 샤드 수가 128개를 초과하지 않는다.
- 읽기 전용 인덱스를 포함한다.
- 첫 번째 경렬이 색인된 필드이다

단 하나라도 매치될 가능성이 없는지 점검 및 최적화하는 법

- 인덱스 메타데이터로 필드 범위상 매칭되는 문서가 확실히 없는지 체크
- 첫 번째 경렬 기준으로 각 샤드가 갖고 있는 최솟값 최댓값 가지고 상위 문서 보유한 샤드가 먼저 수행되도록 함

8-2 엘라스틱 서치의 검색 동작 흐름

AbstractSearchAsyncAction

-검색 대상 샤드 별로 검색 요청 만들어 분산 전송한뒤 응답 수집하고 다음 페이지로 이동

- `search_type` 이 `query_then_fetch` (샤드 레벨 유사도 점수계산) → `SearchQueryThenFetchAsyncAction` 클래스에서 제어

- `search_type` 이 `dfs_query_then_fetch` (샤드 레벨 점수 취합회 정확한 유사도 점수 계산) → `SearchDfsQueryThenFetchAsyncAction` 클래스에서 제어

- DFS: Distributed Frequency Search

- `SearchQueryThenFetchAsyncAction`, `SearchDfsQueryThenFetchAsyncAction` 는 `AbstractSearchAsyncAction` 추상클래스 확장하고 있다.

SearchPhase - CanMatchPreFilterSearchPhase,
SearchQueryThenFetchAsyncAction, SearchDfsQueryThenFetchAsyncAction,
DfsQueryPhase, FetchSearchPhase, ExpandSearchPhase

검색 동작을 페이지 단위로 구분하기 위한 `SearchPhase` 추상클래스를 사용한다. `SearchPhase` 작업이 끝나면 다음 SearchPhase 로 넘어가는 흐름이다.

조정노드에서 수행하는 페이지이다.

8-2 엘라스틱 서치의 검색 동작 흐름

SearchDfsQueryThenFetchAsyncAction

- `search_type` 이 `dfs_query_then_fetch` 일때 수행

- 점수 계산에 필요한 정보를 각 샤드에서 가져오기 위해 점수 계산 요청을 만들어 각 샤드들에게 분산 전송한다.

- 요청을 수신한 노드는 `SearchService.executeDfsPhase()` 를 호출해 `ReaderContext` 를 가져오거나 생성한다.
 - pit 요청이라면 기존에 만들어진 `ReaderContext` 를 사용하고, 아니면 새로 생성해 노드에 일정 시간 저장한다.
 - `ReaderContext` 에는 그 시점의 샤드 대상으로 검색하는 루틴 `IndexSearcher` 가 담겨있어 이를 재활용하면 pit 기능인 항상 같은 상태의 샤드를 대상으로 검색 가능하다.

- `ReaderContext` 생성 이후 샤드 단에서 `DfsPhase` 를 수행해 쿼리에 매치되는 텀, DF(document frequency), 여러 통계 데이터를 구해 반환한다. → 점수 계산에 필요한 정보

- `DfsPhase` 응답 받으면 `executeNextPhase()` 를 호출해 다음 페이지인 `DfsQueryPhase` 로 넘어간다.

DfsQueryPhase

- 각 샤드가 보내준 `DfsPhase` 작업결과로 샤드 별 본 검색 요청을 만들어 각 노드로 분산 전송한다.

- 이 요청을 받은 노드는 `SearchService.executeQueryPhase()` 를 호출해 `QueryPhase` 에서 본격적인 쿼리 매치 작업을 수행한다.

- 각 샤드에서 `QueryPhase` 결과 보내주면 다음 페이지인 `FetchSearchPhase` 로 넘어간다.

8-2 엘라스틱 서치의 검색 동작 흐름

SearchQueryThenFetchAsyncAction

- `search_type` 이 `query_then_fetch` 일때 수행
- 사전 작업(점수 계산에 필요한 정보 얻는 것 같은) 없이 바로 샤드 별 검색 요청 만들어 전송한다.
- 이 요청을 받은 노드는 `SearchService.executeQueryPhase()` 를 호출해 `QueryPhase` 에서 본격적인 쿼리 매치 작업을 수행한다.
- 각 샤드에서 `QueryPhase` 결과 보내주면 다음 페이지인 `FetchSearchPhase` 로 넘어간다.

QueryPhase

- `DfsQueryPhase` 나 `SearchQueryThenFetchAsyncAction` 에서 `SearchService.executeQueryPhase()` 를 호출하면 넘어오는 페이지다
- `ReaderContext` 를 가져오거나 (pit 지정됐을 때, DfsPhase 거쳐왔을 때) 새로 생성한다.
- 샤드 요청이 캐시(샤드 레벨에 저장되는 캐시) 가능한 요청인지 확인하고 있으면 응답 바로 반환한다.
- 없으면 `QueryPhase` 작업 수행 후 캐시에 결과를 저장한다. 캐시 불가능한 요청이면 쿼리 매치 작업만 한다.
- `execute()` 에서 주 작업 (검색, 제안, 집계) 를 수행한다.
 - 검색은 루씬의 `IndexSearcher`, `Query`, `Collector` 이용한다.
 - 제안은 오타교정이나 자동완성에 사용한다. `SuggestPhase` 에서 수행한다.
 - 집계는 `AggregationPhase` 에서 수행한다.
- 각 샤드가 수행한 `QueryPhase` 결과가 조정 노드에 모이면 `FetchSearchPhase` 로 넘어간다.

8-2 엘라스틱 서치의 검색 동작 흐름

FetchSearchPhase 와 FetchPhase

- `QueryPhase` 결과를 병합해 샤드 별 fetch 요청을 생성하고 분산 전송한다.
 - 이 요청을 받은 노드는 `SearchService.executeFetchPhase()` 를 호출한다.
 - `ReaderContext` 를 확보한 후 `FetchPhase.execute()` 를 호출해 지정한 번호의 문서 내용을 읽는다.
 - 이후 `ReaderContext` 가 pit 이나 scroll 것이 아니라 단발성 쿼리 위한 문맥이면 `ReaderContext` 를 해제 한다.
 - 문서 내용 읽은 fetch 결과를 조정 노드로 반환한다.
- 각 샤드별 `FetchPhase` 결과가 조정 노드에 모이면 `ExpandSearchPhase` 로 넘어간다.

FetchSubPhase

- `FetchPhase` 작업 중 문서 내용을 읽어 `SearchHit` 을 만드는 과정 중의 여러 하위 작업을 수행하는 인터페이스이다.
- `FetchSubPhase` 구현체는 `SearchModule` 에 등록되며 커스텀 플러그인으로 등록할 수도 있다.
- 기본 구현체: `FetchSourcePhase` (_source 읽기), `FetchScorePhase` (_score 다시 계산하기), `ExplainPhase` (_explanation 위한 검색 중간 과정과 유사도 점수 상세 설명 만들기)

8-2 엘라스틱 서치의 검색 동작 흐름

ExpandSearchPhase

-collapse 를 위한 페이지이다.

- 지정한 필드 값 기준으로 검색 결과를 그룹으로 묶고 그 안에서 다른 기준으로 상위 문서를 지정한 개수만큼 뽑는다.

-본 검색의 hit 수 만큼 새 검색 요청을 만들어 `MultiSearchRequest` 에 담는다.

-`MultiSearchRequest` 는 로컬, 즉 조정노드 자신이 다시 받아 `TransportMultiSearchAction` 에서 처리한다.

- 이 다중 검색의 세부 요청들은 다시 `TransportSearchAction` 이 받아서 독립적인 새 검색 수행하는 것처럼 수행해 조정 노드에 결과를 돌려준다.

-조정 노드가 응답 받으면 결과 모아서 최종 검색 결과 만들어 반환하고 검색 작업을 종료한다.

- `ExpandSearchPhase` 의 응답인 `InternalSearchResponse` 를 최종 응답인 `SearchResponse` 로 변환하는 작업은 다시 `AbstractSearchAsyncAction` 으로 돌아와 수행한다.

8-2 엘라스틱 서치의 검색 동작 흐름

[1] IndexSearcher

-루씬 인덱스를 읽기 전용으로 열어 문서를 검색하는 클래스

-최상위 `IndexReader` 가 하나 있고 각 독립적인 여러 `LeafIndexReader` 로 구성된다.

- 각 `LeafIndexReader` 는 세그먼트 하나의 역색인, 필드, `doc_values` 등 읽는 추상 클래스다.

[2] QueryBuilder

-ES 레벨의 쿼리 정의하는 인터페이스

- 쿼리명, 쿼리 DSL 을 파싱하고 직렬화하는 방법 정의

- `toQuery()` : 루씬 쿼리를 생성함

- `SearchService` 에서 검색 문맥 만들 때 호출된다. (`ReaderContext` 생성 직후, `executeSearchPhase()`, `executeFetchPhase()`, `executeDfsPhase()` 에서)

- `AbstractQueryBuilder` 추상클래스를 확장해 커스텀 할 수 있다.

8-2 엘라스틱 서치의 검색 동작 흐름

[2-1] Query

-루씬 쿼리 정의하는 추상 클래스

- `createWeight()`: `Weight` 을 생성한다.

- `rewrite()`: `Query` 를 다른 기본(primitive) Query 의 조합으로 재구성해 최적화한다.

- primitive query: `createWeight()` 를 오버라이드해 직접 구현 하는 쿼리 (ex. `TermQuery`, `BooleanQuery`, `MatchNoDocsQuery`, `PhraseQuery` 등)
- `IndexSearch` 에서 검색 수행하기 전 호출된다.
- `QueryPhase`, `DfsPhase` 에서 명시적으로 호출되기도 한다.

[2-2] Weight

- `Query` 내부 동작을 구현하는 추상클래스

- `score()`: `Scorer` 를 생성

- `bulkScorer()`: `BulkScorer` 생성

- `explain()`: 쿼리 수행 중간 과정과 유사도 점수 계산 과정을 설명한다. `explain:true` 로 검색 시 `FetchSubPhase` 중 `ExplainPhase` 에서 `explain` 메소드를 호출한다.

8-2 엘라스틱 서치의 검색 동작 흐름

[3-1] Scorer

- 유사도 점수 계산 담당하는 추상 클래스

- `iterater()` : 매치된 문서 순회하는 `DocIdSetIterator` 반환. 즉 `DocIdSetIterator` 에서 순회하는 모든 문서는 쿼리에 매치됐다는 뜻이다. `WeightScorerDocIdSetIterator` 의 순차 과정이 쿼리 매칭되는 문서 고르는 작업이다.

- `twoPhaseIterator()` : 무거운 매치 작업이면 두 페이즈로 나눠 진행하게 `TwoPhaseIterator` 를 반환한다. 만약 나눌 필요 없으면 `null` 을 반환한다.

- `score()` : 현재 문서(`DocIdSetIterator` 가 가리키는 문서)의 유사도 점수 계산해 `float` 타입으로 반환한다.

- `docID()` : 현재 문서의 doc id 반환한다. 고유한 32 비트 숫자

[3-2] BulkScorer

- 여러 문서 대상으로 한번에 유사도 점수 계산하는 추상 클래스

- 디폴트 구현체는 `DefaultBulkScorer` 이다

- `Collector` 가 `DocIdSetIterator`, `TwoPhaseIterator` 이용해 문서 하나씩 순회하며 `Scorer` 로 점수 계산해서 수집한다.

8-2 엘라스틱 서치의 검색 동작 흐름

[4-1] DocIdSetIterator

- '매치된' 문서의 순회 담당하는 추상 클래스

- `cost()` : 순회하는데 드는 비용 추정값 (보통 매치된 문서 수)
- `advance(target)` : target 값 이상으로 첫 번째 매치되는 문서. `DocIdSetIterator` 순회를 전진시킨다.
- `nextDoc()` : 매치된 다음 문서로 `DocIdSetIterator` 순회를 전진시킨다. 보통 `advance(docID() + 1)`

[4-2] TwoPhaseIterator

- 무거운 매치 여부 판단하는 작업을 두 페이지로 나누어 진행하는 추상 클래스

- 비용 저렴한 매치 먼저 수행해서 후보를 좁히고 그 후에 문서 수집 과정에서 최종 매치를 한다.

- `approximation()` : 간략하게 문서 매치 만족하는 문서 순회하는 `DocIdSetIterator` 를 반환한다.
- `matches()` : 무거운 작업이다. `DocIdSetIterator` 의 문서 돌면서 `match()` 가 수행된다.
- `matchCost()` : `matches()` 에 드는 비용 추정해서 반환한다.

8-2 엘라스틱 서치의 검색 동작 흐름

[5] Collector, LeafCollector

-유사도 점수나 정렬 기준 등 계산하거나 확인하며 검색 결과 수집하는 인터페이스

- `getLeafCollector()` : `LeafCollector` 인스턴스 만들어 반환한다.

bool 쿼리 동작 순서와 DocIdSetIterator 순회

bool 쿼리 안의 `must`, `filter`, `must_not`, `should` 하위의 다양한 쿼리들이 어떤 순서로 수행되는지 정해진 규칙이 없다. 이에 대해 좀 더 알아보겠다.

rewrite, cost

-ES 쿼리를 루인의 여러 쿼리로 쪼개고 조합해 재작성할 때 `Query.rewrite()` 메서드를 사용한다.

-쿼리를 쪼개 후 각 쿼리 수행 시 드는 비용을 내부적으로 추정해 유리할 것으로 판단되는 쿼리를 먼저 수행한다. 비용은 `DocIdSetIterator.cost()` 로 확인한다.

-여러 하부 쿼리를 병렬로 수행하기도 한다.

-Bool 쿼리의 `Weight` 구현체: `BooleanWeight`

- 쿼리 세부 내용에 따라 다양한 종류의 최적화된 `Scorer` 를 만들어 반환한다.
- conjunction 검색과 disjunction 검색을 위한 `Scorer` 가 만드는 `DocIdSetIterator` 를 살펴보자
 - conjunction 검색: AND 검색. 쿼리 매치 조건 모두 만족해야 한다.
 - disjunction 검색: OR 검색. 하나만 만족해도 된다.
 - 둘 다 하위 `Query` 마다 `Weight`, `Scorer` 를 미리 만들어둔 다음 하위 `Scorer` 로 최종 `Scorer` 를 만든다.

8-2 엘라스틱 서치의 검색 동작 흐름

conjunction 검색

-AND 검색

hello / rocket / punch 모두 만족하는 문서 찾아야 함

- 먼저 앞에 두개 동시에 만족하는 문서로 이동하고
- 다음 세번째 문서도 체크해서 세 개 동시에 만족하는 문서를 찾는다.

`TwoPhaseIterator` 방식이라면 최상위 `TwoPhaseIterator` 가 하위 `TwoPhaseIterator` 를 `matchCost` 순으로 정렬해둔다음 `matches()` 가 호출되면 `matchCost` 가 낮은 하위 `TwoPhaseIterator.matches()` 부터 호출한다.

disconjunction 검색

-OR 검색

-하위 `Scorer` 와 `DocIdSetIterator` 를 이용해 최상위 `Scorer` 와 `DocIdSetIterator` 를 생성한다.

-or 이므로 `advance(target)` 결과 중 최소값을 반환한다.

쿼리 문맥과 필터 문맥

- 두 문맥 간 수행 순서는 완전이 없다.

-(1단계) `DocIdSetIterator` 인스턴스 생성 끝나면 `LeafCollector` 가 `DocIdSetIterator` 를 순회하며 `collect()` 호출하고

-(2단계) 그 안의 `Scorer.score()` 호출한다.

→ 일단 매치되는 후보군을 먼저 다 뽑고 유사도 계산은 나중에 한다. 필터 문맥 쿼리는 점수 계산 비용을 아끼는 것이지 먼저 수행된다는 의미는 아니다.

-매치되는 단일 문서 마다 유사도 점수 계산하고 수집하는건 하위 쿼리 일부가 병렬적으로 수행된다는 뜻

8-2 엘라스틱 서치의 검색 동작 흐름

캐시 동작

샤드 레벨 요청 캐시 (request cache)

- query 페이즈에서 수행된 작업을 샤드 레벨에 저장한다.

캐시 수행 위치

`SearchQueryThenFetchAsyncAction` 작업 이후 `QueryPhase.execute()` 수행 전에 `SearchService.executeSearch()` 에서 캐시 사용 가능한지 파악한다.

캐시 조건

- `search_type` 이 `query_then_fetch` 여야함

-scroll 검색이 아니어야 함

-profile 요청이 아니어야 함

-now, `Math.random()`, `new Date()` 같은 확정되지 않은 조건이 들어가지 않아야함

-샤드 레벨 요청 캐시 설정했는지 확인

- 명시적 `index.requests.cache.enable` 값이 없으면 `size:0` 이어야 캐시 먹힘
- `index.requests.cache.enable:true` 면 `size:0` 이어도 캐시 먹힘

8-2 엘라스틱 서치의 검색 동작 흐름

노드 레벨 쿼리 캐쉬

-필터 문맥으로 검색 수행시 어떤 문서가 매치됐는지 노드 레벨에 저장

캐시 수행 위치

-QueryPhase 에서 `IndexSearcher.search()` 할 때 `Query.createWeight()` 호출해 `Weight` 를 생성한다.

-유사도 계산 안 하는 Query 면 `ChachingWeightWrapper` 로 감싸 최종 Weight 를 반환한다.

-`Weight` 가지고 검색 수행할 때 `BulkScorer` 나 `Scorer` 가 캐시된 DocIdSet 을 순회하는 `DocIdSetIterator` 를 반환한다.

캐시 조건

-유사도 점수 계산하지 않는 쿼리여야 함

-문서 수 1만개 이상이어야 함

-샤드 내 문서의 3%이상 보유한 세그먼트가 대상이다

-캐시 없어도 충분히 빠를 쿼리는 캐시 하지 않음 (TermQuery, MatchAllDocQuery, MatchNoDocsQuery, DocValuesFieldExistQuery(doc_values 필드 대상 exists 쿼리) ..)

-락 획득해야함

- 캐기 읽기, 쓰기 작업 모두 락 획득 필요하다. 락 획득 실패하면 일반 검색으로 진행한다.

8-2 엘라스틱 서치의 검색 동작 흐름

표 8.2 샤드 레벨 요청 캐시 vs. 노드 레벨 쿼리 캐시 간단 비교

	요청 캐시	쿼리 캐시
수행 위치	QueryPhase 작업 들어가기 직전	래핑한 Weight에서 BulkScorer, Scorer 등 생성 시
키	ShardSearchRequest	Query
값	매치된 상위 문서와 점수의 목록, 제안, 집계 등을 포함한 QueryPhase의 수행 결과	매치된 문서 목록을 나타내는 비트 배열
주요 캐시 조건	① 요청에 requestCache를 명시적으로 지정했거나 ② 요청에 requestCache를 지정하지 않고, index.requests.cache.enable이 true이며, size가 0 (① OR ②) AND (확정적인 검색 등 기본 조건)	필터 문맥에서 쿼리를 많이 자주 수행해야 하고 빠른 쿼리는 제외되며 세그먼트에 일정 이상의 문서가 있어야 함
적재 위치	IndicesService의 IndicesRequestCache	IndicesService의 IndicesQueryCache
캐시 접근 범위	샤드 레벨	노드 레벨