Searching

Location & Implementation

As a search engine, **Searching** is an essential feature of **Elasticsearch**. The format of the request that can be handled is defined in the server/src/main/java/org/elasticsearch/rest/action/search/RestSearchAction.java file

There are four kinds of requests provided:

```
//server/src/main/java/org/elasticsearch/rest/action/search/RestSearchAction.java
public RestSearchAction(RestController controller) {
    controller.registerHandler(GET, "/_search", this);
    controller.registerHandler(POST, "/_search", this);
    controller.registerHandler(GET, "/{index}/_search", this);
    controller.registerHandler(POST, "/{index}/_search", this);
}
```

The REST API allows us to execute a search query and obtain search hits that match the query. Both a simple query string and a request body are acceptable. The string type parameter <index> is optional, and we can use a comma-separated list to represent multiple indices.

The typical index request of searching a specific index is like this:

`GET /twitter/_search?q=user:kimchy`

The response returned by the API is in the following format:

```
{
  "took" : 5,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.3862944,
    "hits" : [
      {
        "_index" : "twitter",
        "_type" : "_doc",
        "_id" : "0",
        "_score" : 1.3862944,
        "_source" : {
          "date" : "2009-11-15T14:12:12",
          "likes" : 0,
          "message" : "trying out Elasticsearch",
          "user" : "kimchy"
        }
      }
    ]
  }
}
```

To search several indices:

```
GET /kimchy,elasticsearch/_search?q=user:kimchy
```

And we can search all indices in a cluster by omitting the <index> parameter:

```
GET /_search?q=user:kimchy
```

Client Side

1. The client receives and handles the requests in **RestSearchAction** according to the routing.
2. The method **parseSearchRequest()** parses the rest request on top of the SearchRequest, preserving values that are not overridden by the rest request.The searching parameters in the request body is parsed and verified, and package them into **SearchRequest**.
3. Then, **RestSearchAction** specifies the action of the request that the server is going to handle, build a target shared list, traversal the list and send Query to the server side.
4. After receiving the response from the Server side, the client will merge all of the Query Response and wait until all of the Responses are returned.

Server Side

1. Firstly, the master node creates the corresponding **ShardsIterator** according to the index of the **SearchRequest**, which consists of **localShardsIterator** and **remoteShardIterators**. Based on the searching rules as follows, traverse all of the shards.
    - The maximum number of shards to traverse is 2^63-1
    - If there’s only one shard, the searching type can only be ***QUERY_THEN_FETCH***
    - Whether to search the cache or not
    - The maximum number of concurrent traversing is calculated as Math.min(256, Math.max(num_of_nodes, 1)*5)

2. Secondly, the server creates an asynchronous request and transmits it to all of the other nodes, then waits for the callback.

3. Thirdly, traverses all of the nodes and creates *ShardSearchTransportRequest* object, then conduct the query on each node:

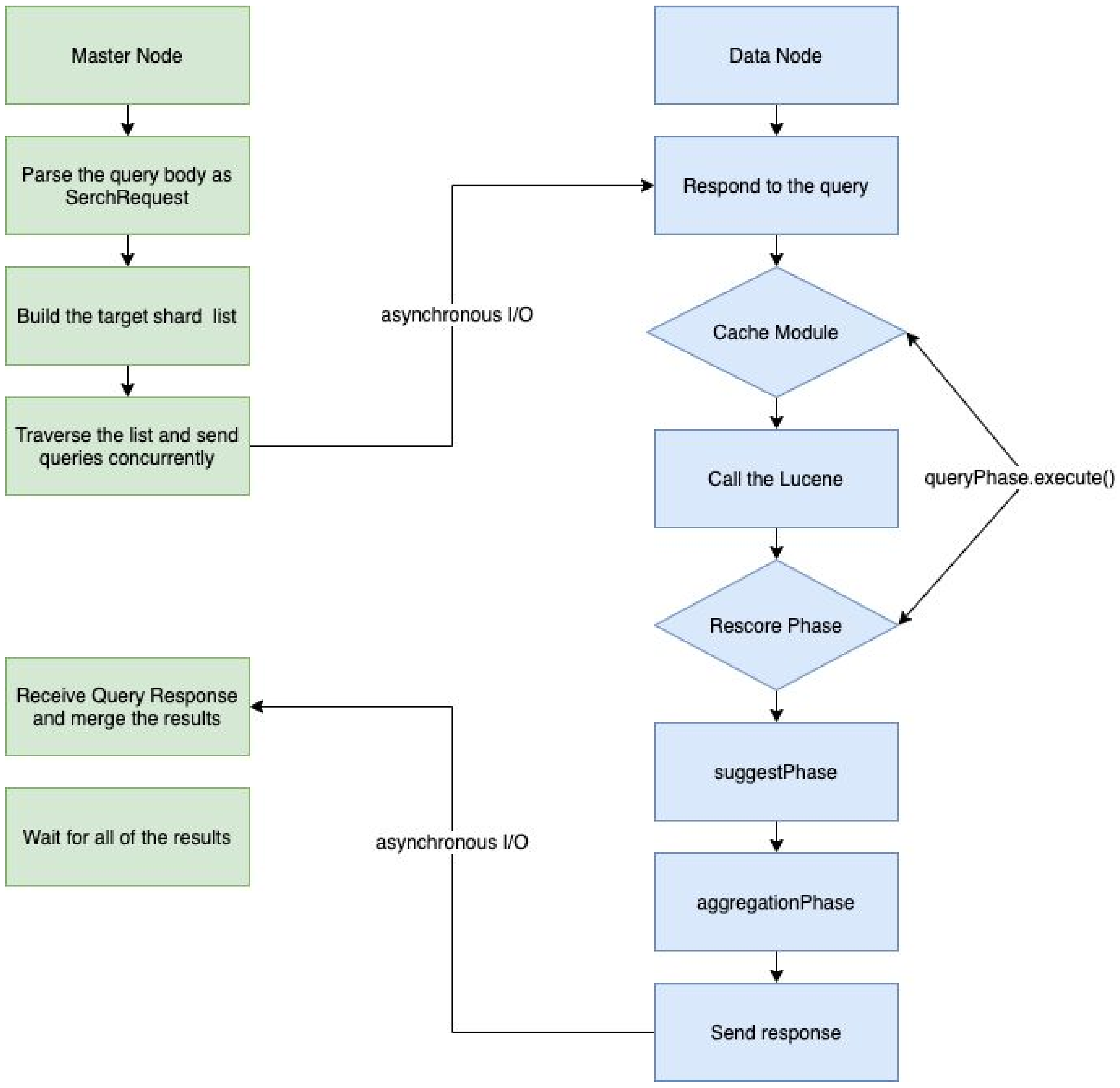
```
//server/src/main/java/org/elasticsearch/action/search/AbstractSearchAsyncActi
```

```
on.java
for (int index = 0; index < shardsIts.size(); index++) {
   final SearchShardIterator shardRoutings = shardsIts.get(index);
   assert shardRoutings.skip() == false;
   performPhaseOnShard(index, shardRoutings, shardRoutings.nextOrNull());
}
```

4. The query stage. Firstly check whether the query has already existed in the cache, if so , do the cache query, otherwise execute the QueryPhase class which calls the searcher.search() method of Lucene. If the query succeeds then callback the onShardResult() method and return the docIds, otherwise callback the onShardFailure() to count the failure and try to search on the duplicated shard.
   In this stage, the program will calculate the score of relevance for ranking:

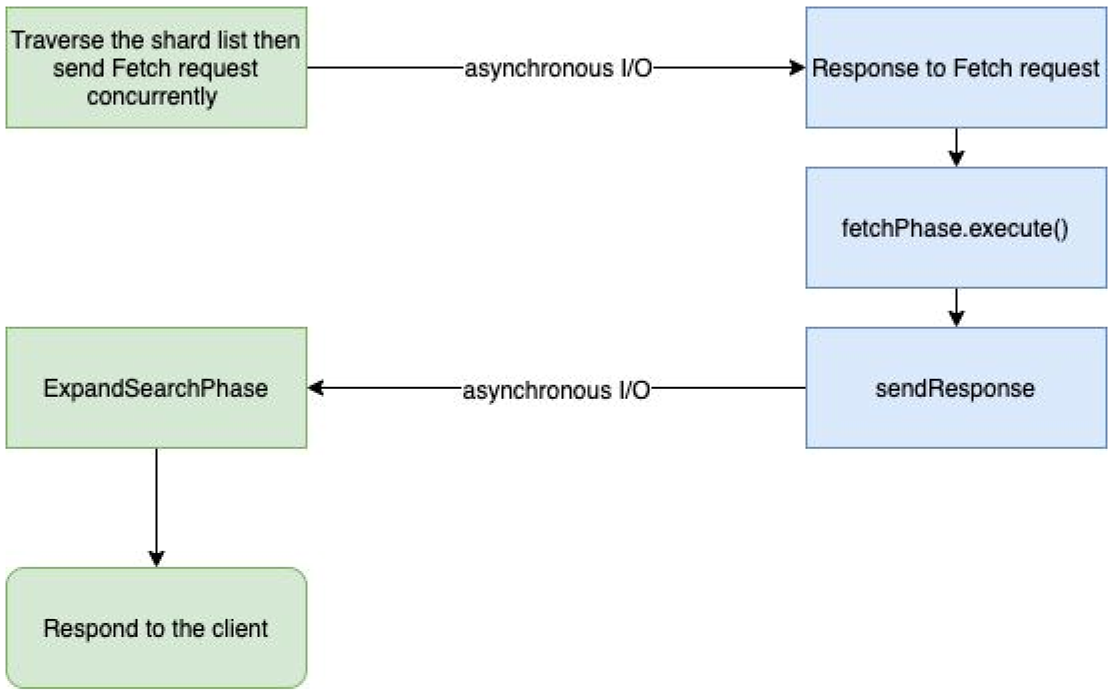
```
//server/src/main/java/org/elasticsearch/search/query/QueryPhase.java
if (rescore) { // only if we do a regular search
   rescorePhase.execute(searchContext);
}
```

The query and response progress can be illustrated as follows:

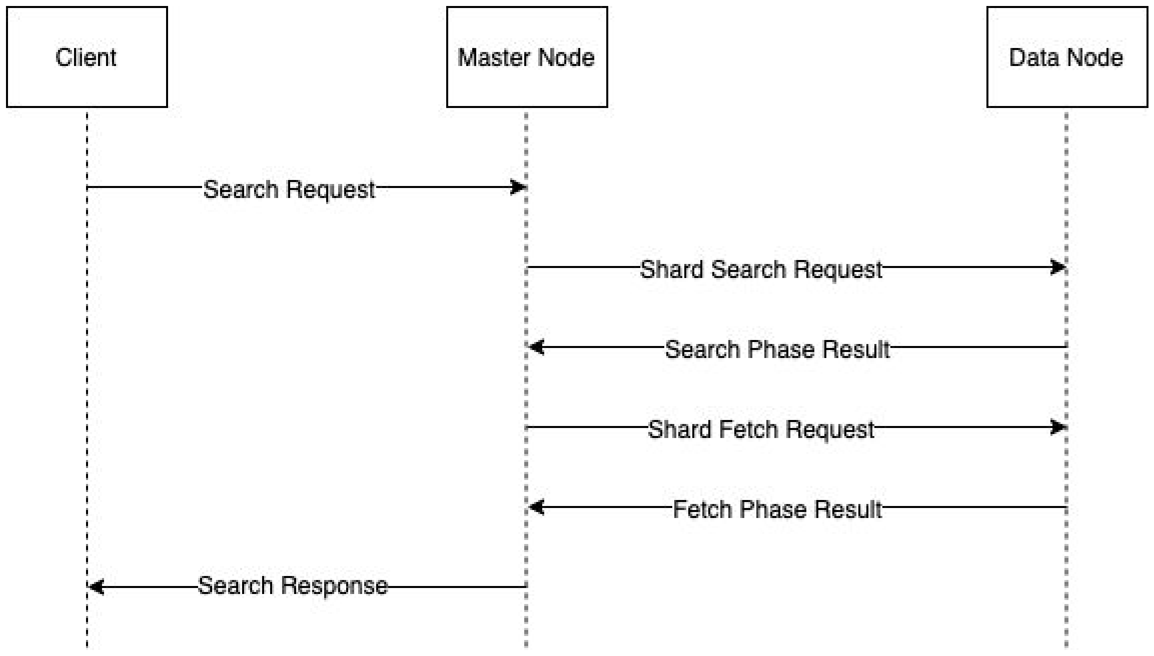

Master Node
Parse the query body as SerchRequest
Build the target shard list
Traverse the list and send queries concurrently
asynchronous I/O
Receive Query Response and merge the results
Wait for all of the results
asynchronous I/O
Data Node
Respond to the query
Cache Module
Call the Lucene
queryPhase.execute()
Rescore Phase
suggestPhase
aggregationPhase
Send response

5. The fetch stage. After the master receives docIds returned by every node, it sends the data fetch request to gain the data according to the docIds and shard id. Then combine the data and send it to the client.

   The progress of the fetch stage can be illustrated as follows:



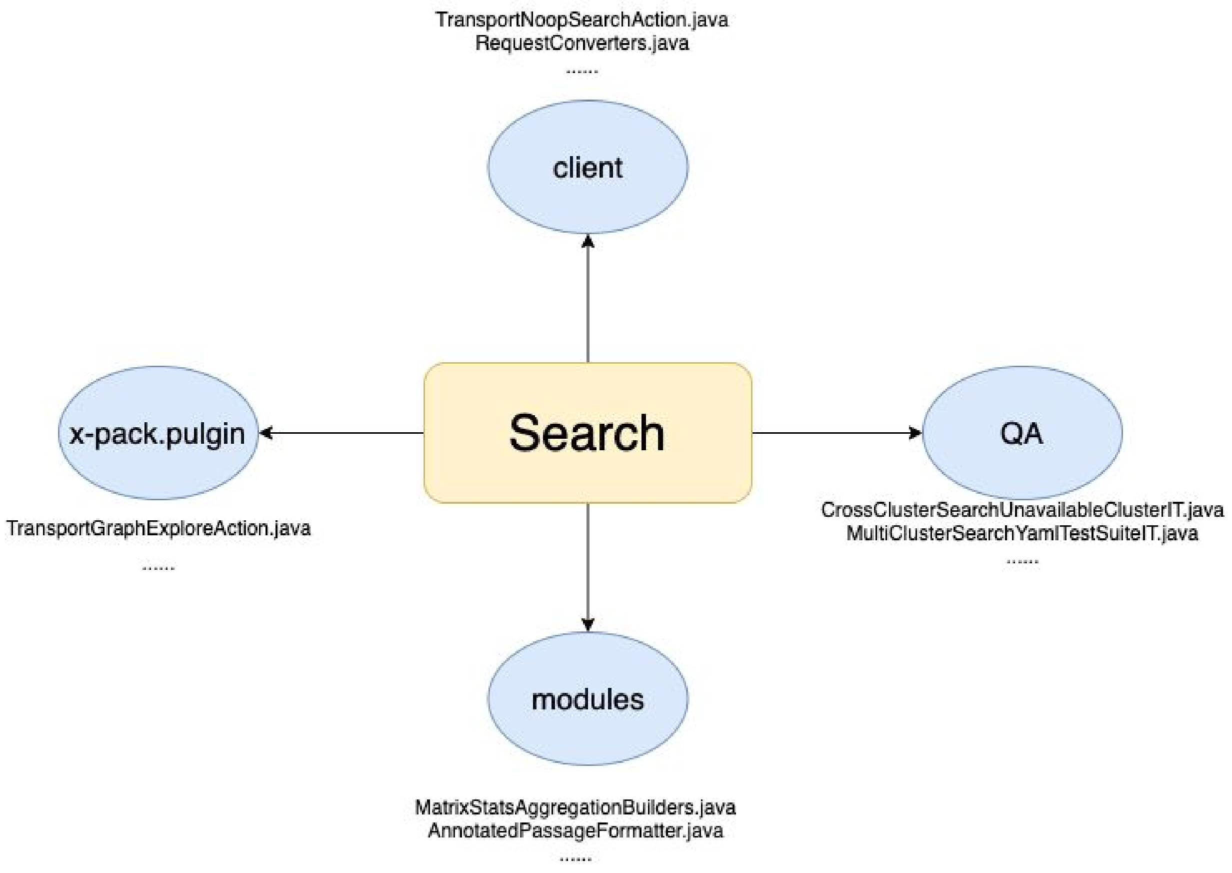
The overall progress can be simply illustrated as follows:



Relation with other parts

RestSearchAction class is a specific REST API Handler used to handler Searching Requests, which is the subclass of the class **BaseRestHandler** in the server/src/main/java/org/elasticsearch/rest folder. And the class **BaseRestHandler** inherited from the class **RestHandler**, also in the same package, the **org.elasticsearch.rest** package.

Other parts in the system which use the searching features can be illustrated as follows:


TransportNoopSearchAction.java
RequestConverters.java
......
client
x-pack.pulgin
Search
QA
TransportGraphExploreAction.java
......
CrossClusterSearchUnavailableClusterIT.java
MultiClusterSearchYamlTestSuiteIT.java
......
modules
MatrixStatsAggregationBuilders.java
AnnotatedPassageFormatter.java
......

- The searching feature needs to receive query requests from the **clients** module and send the searching result to the **clients**.
- Most of the actions of searching feature are from **modules**.
- Most modules of **X-pack.plugin** module import the searching module and use it many times.
- The QA module test and evaluate the performance of the searching module.