执行搜索流程

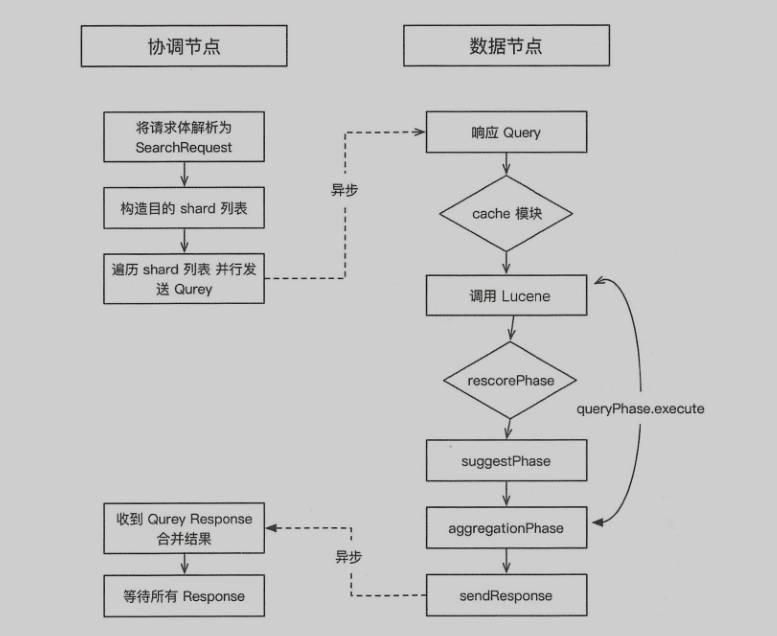
1. 对检索的字段使用和索引时相同的分词器进行分析，产生token列表
2. 根据查询语句构造语法树
3. 查找符合语法树的文档
4. 对查找到的文档进行评分，使用TF/IDF
5. 根据评分结果进行排序

search type分为两种

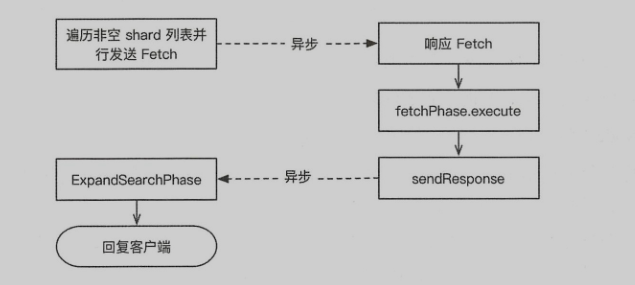
DFS\_QUERY\_THEN\_FETCH

QUERY\_THEN\_FETCH (默认)

Query



Fetch



//HTTP

Netty4HttpRequestHandler.channelRead0()

AbstractHttpServerTransport.dispatchRequest()//372

RestController.tryAllHandlers()

**根据request获取具体的handler**

RestController.getAllHandlers(request.params(), rawPath);//281

//根据requestMethod获取处理handler,保存在SecurityRestFilter中的restHandler

例如RestGetAction,RestIndexAction,RestDeleteAction

handler = handlers.getHandler(requestMethod);

SecurityRestFilter.handleRequest(request, channel, client);

BaseRestHandler.handleRequest()

action.accept(channel);//108

HttpChannelTaskHandler.execute()

NodeClient.executeLocally

TransportAction.execute()

requestFilterChain.proceed(task, actionName, request, listener);//129

i=0,先执行

this.action.filters[i].apply(task, actionName, request, listener, this);

上面方法执行完后会回调equestFilterChain.proceed()方法

然后执行this.action.doExecute(task, request, listener);

TransportSearchAction.doExecute();

TransportSearchAction.executeSearch();

#TransportSearchAction

@Override

protected void doExecute(Task task, SearchRequest searchRequest, ActionListener<SearchResponse> listener) {

if (remoteClusterIndices.isEmpty()) {//是否设置ccs

executeLocalSearch(task, timeProvider, searchRequest, localIndices, clusterState, listener);

} else {

执行ccs

if (shouldMinimizeRoundtrips(searchRequest)) {

ccsRemoteReduce();

}else{

}

}

}

private void executeSearch(){

//本地索引

final Index[] indices = resolveLocalIndices()

Map<String, AliasFilter> aliasFilter = buildPerIndexAliasFilter()

Map<String, Set<String>> routingMap = indexNameExpressionResolver.resolveSearchRouting()

...

executeSearch();

...

}

采用线程池 Executor executor = threadPool.executor(ThreadPool.Names.SEARCH);

#SearchQueryThenFetchAsyncAction->AbstractSearchAsyncAction

public final void start() {

executePhase(this);

}

@Override

public final void run() {

//循环每一个分片进行搜索

for (int index = 0; index < shardsIts.size(); index++) {

final SearchShardIterator shardRoutings = shardsIts.get(index);

assert shardRoutings.skip() == false;

performPhaseOnShard(index, shardRoutings, shardRoutings.nextOrNull());

}

}

private void performPhaseOnShard(...){

...

executePhaseOnShard(shardIt, shard,

new SearchActionListener<Result>(shardIt.newSearchShardTarget(shard.currentNodeId()), shardIndex) {

@Override

public void innerOnResponse(Result result) {

try {

onShardResult(result, shardIt);

} finally {

executeNext(pendingExecutions, thread);

}

}

@Override

public void onFailure(Exception t) {

try {

onShardFailure(shardIndex, shard, shard.currentNodeId(), shardIt, t);

} finally {

executeNext(pendingExecutions, thread);

}

}

});

...

}

在onShardResult()里面会判断如果所有请求分片都成功方法，这回执行PhaseDone

if (xTotalOps == expectedTotalOps) {

onPhaseDone();//最终调用SearchFetchPhase.innerRun()

}

executePhaseOnShard()方法会执行

SearchQueryThenFetchAsyncAction.executePhaseOnShard

-> SearchTransportService->sendExecuteQuery{

//请求到该分片的数据节点  
 transportService.sendChildRequest(connection, QUERY\_ACTION\_NAME, request, task,

new ConnectionCountingHandler<>(handler, reader, clientConnections, connection.getNode().getId()));

}

数据节点接收search请求

在SearchTransportService的registerRequestHandler方法可以看到QUERY\_ACTION\_NAME的注册handler

,数据节点接收的请求就是在下面红色标注的handler里面执行

public static void registerRequestHandler(){

transportService.registerRequestHandler(QUERY\_ACTION\_NAME, ThreadPool.Names.SAME, ShardSearchRequest::new,

(request, channel, task) -> {

searchService.executeQueryPhase(request, (SearchTask) task, new ChannelActionListener<>(

channel, QUERY\_ACTION\_NAME, request));

});

}

private SearchPhaseResult executeQueryPhase(){

loadOrExecuteQueryPhase(request, context);

if (request.numberOfShards() == 1) {

//如果分片只有一个立即执行fetch阶段

return executeFetchPhase(context, afterQueryTime);

}

return context.queryResult();//大于一个分片返回

}

#QueryPhase

public void execute(SearchContext searchContext){

boolean rescore = execute(searchContext, searchContext.searcher(), searcher::setCheckCancelled);

suggestPhase.execute(searchContext);

aggregationPhase.execute(searchContext);

}

Fetch阶段

#FetchPhase

@Override

public void execute(SearchContext context) {

SearchHit[] hits = new SearchHit[context.docIdsToLoadSize()];

for (int index = 0; index < context.docIdsToLoadSize(); index++) {

if (rootDocId != -1) {

searchHit = createNestedSearchHit(context, docId, subDocId, rootDocId,

storedToRequestedFields, subReaderContext);

} else {

//获取文档

searchHit = createSearchHit(context, fieldsVisitor, docId, subDocId,

storedToRequestedFields, subReaderContext);

}

hits[index] = searchHit;

}

}

#SearchFetchPahse

private void innerRun() throws IOException {

for (int i = 0; i < docIdsToLoad.length; i++) {

...

executeFetch(i, searchShardTarget, counter, fetchSearchRequest, queryResult.queryResult(),

connection);

...

}

}

#SearchTransportService

public void sendExecuteFetch(Transport.Connection connection, final ShardFetchSearchRequest request, SearchTask task,final SearchActionListener<FetchSearchResult> listener) {

sendExecuteFetch(connection, FETCH\_ID\_ACTION\_NAME, request, task, listener);

}

最终请求到达数据节点，流程为

SearchService.executeFetchPhase() -> FetchPahse.execute()

SearchService.executeFetchPhase() -> FetchPahse.innerRun()

SearchPhaseController.reducedQueryPhase()//对每一个分片的结果进行合并

InternalAggregations.reduce()//真正执行reduce逻辑的地方

for (Map.Entry<String, List<InternalAggregation>> entry : aggByName.entrySet()) {

List<InternalAggregation> aggregations = entry.getValue();

// Sort aggregations so that unmapped aggs come last in the list

// If all aggs are unmapped, the agg that leads the reduction will just return itself

aggregations.sort(INTERNAL\_AGG\_COMPARATOR);

InternalAggregation first = aggregations.get(0);

reducedAggregations.add(first.reduce(aggregations, context));

}

进入

InternalTerms.reduce()

b.bind(SearchService.class).toInstance(searchService);

b.bind(SearchTransportService.class).toInstance(searchTransportService);

b.bind(SearchPhaseController.class)

.toInstance(new SearchPhaseController(searchService::createReduceContext));

b.bind(Transport.class).toInstance(transport);

b.bind(TransportService.class).toInstance(transportService);

b.bind(HttpServerTransport.class).toInstance(httpServerTransport);

RestSearchAction

//TCP

transportService.start();

transport.start();

Netty4Transport.doStart();

createServerBootstrap(profileSettings, eventLoopGroup);//137

serverBootstrap.childHandler(getServerChannelInitializer(name));//218

ch.pipeline().addLast("dispatcher", new Netty4MessageChannelHandler(Netty4Transport.this));//353

//当请求过来时

Netty4MessageChannelHandler.channelRead

InboundHandler.inboundMessage()

InboundHandler.messageReceived()

InboundHandler.handleRequest()

IndexSearcher.search()

ContextIndexSearcher.search()

ContextIndexSearcher.searchInternal()

GlobalOrdinalsStringTermsAggregator.getLeafCollector

CancellableBulkScorer.score()

MatchAllDocsQuery.bulkScorer().score()

for (int doc = min; doc < max; ++doc) {

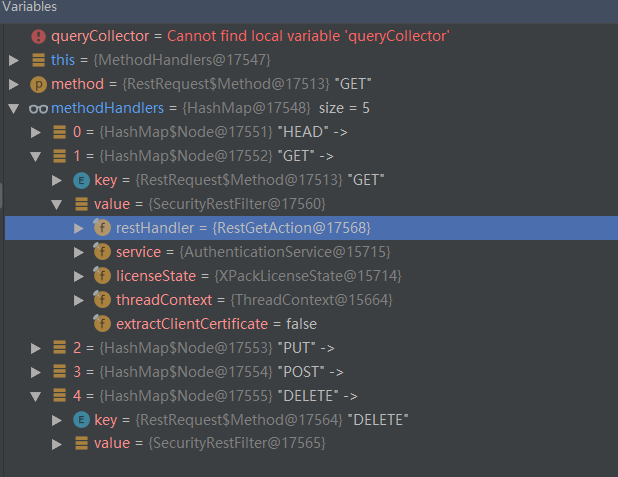
scorer.doc = doc;

if (acceptDocs == null || acceptDocs.get(doc)) {

collector.collect(doc);

}

}



public RestIndicesAction(RestController controller) {

controller.registerHandler(GET, "/\_cat/indices", this);

controller.registerHandler(GET, "/\_cat/indices/{index}", this);

}