写入分析

//TransportBulkAction

protected void doExecute(Task task, BulkRequest bulkRequest, ActionListener<BulkResponse> listener) {

//提取pipeline

for (DocWriteRequest<?> actionRequest : bulkRequest.requests) {

if (indexRequest != null) {

// Each index request needs to be evaluated, because this method also modifies the IndexRequest

boolean indexRequestHasPipeline = resolvePipelines(actionRequest, indexRequest, metaData);

hasIndexRequestsWithPipelines |= indexRequestHasPipeline;

}

}

//执行pipeline

if (hasIndexRequestsWithPipelines) {

...

}

if (needToCheck()) {

//获取请求的索引

final Set<String> indices = bulkRequest.requests.stream().

.map(DocWriteRequest::index)

.collect(Collectors.toSet());

for (String index : indices) {

boolean shouldAutoCreate;

try {

shouldAutoCreate = shouldAutoCreate(index, state);//判断是否需要创建索引

} catch (IndexNotFoundException e) {

shouldAutoCreate = false;

indicesThatCannotBeCreated.put(index, e);

}

if (shouldAutoCreate) {

autoCreateIndices.add(index);

}

}

if (autoCreateIndices.isEmpty()) {//没有需要创建的索引，直接执行executeBulk

executeBulk(task, bulkRequest, startTime, listener, responses, indicesThatCannotBeCreated);

} else {

for (String index : autoCreateIndices) {

createIndex(index, bulkRequest.timeout(), new ActionListener<CreateIndexResponse>() {

@Override

public void onResponse(CreateIndexResponse result) {

//所有索引创建完毕，执行executeBulk()

if (counter.decrementAndGet() == 0) {

threadPool.executor(ThreadPool.Names.WRITE).execute(

() -> executeBulk(task, bulkRequest, startTime, listener, responses,... ));

}

}

}

}

}

}else{

executeBulk(task, bulkRequest, startTime, listener, responses, emptyMap());

}

}

执行executeBulk(),调到

#BulkOperation

@Override

protected void doRun() {

for (int i = 0; i < bulkRequest.requests.size(); i++) {

switch (docWriteRequest.opType()) {

case *INDEX*:

//生成uuid

indexRequest.process(indexCreated, mappingMd, concreteIndex.getName());

break;

}

}

...

// first, go over all the requests and create a ShardId -> Operations mapping

Map<ShardId, List<BulkItemRequest>> requestsByShard = new HashMap<>();

for (int i = 0; i < bulkRequest.requests.size(); i++) {

//按shardId分组存储到requestsByShard中

ShardId shardId = clusterService.operationRouting().indexShards(clusterState, concreteIndex, request.id(),request.routing()).shardId();

List<BulkItemRequest> shardRequests = requestsByShard.computeIfAbsent(shardId, shard -> new ArrayList<>());

shardRequests.add(new BulkItemRequest(i, request));

}

for (Map.Entry<ShardId, List<BulkItemRequest>> entry : requestsByShard.entrySet()) {

//生成bulkShardRequest

BulkShardRequest bulkShardRequest = new BulkShardRequest(...);

shardBulkAction.execute(bulkShardRequest, new ActionListener<BulkShardResponse>() {

@Override

public void onResponse(BulkShardResponse bulkShardResponse) {

for (BulkItemResponse bulkItemResponse : bulkShardResponse.getResponses()) {

// we may have no response if item failed

if (bulkItemResponse.getResponse() != null) {

bulkItemResponse.getResponse().setShardInfo(bulkShardResponse.getShardInfo());

}

responses.set(bulkItemResponse.getItemId(), bulkItemResponse);}

if (counter.decrementAndGet() == 0) {

finishHim();//所有主分片结果返回后执行

}

}

@Override

public void onFailure(Exception e) {

}

}

}

}

执行

shardBulkAction.execute()->ReroutePhase.doRun()

@Override

protected void doRun() {

final DiscoveryNode node = state.nodes().get(primary.currentNodeId());

if (primary.currentNodeId().equals(state.nodes().getLocalNodeId())) {

//主分片是否在本地节点

performLocalAction(state, primary, node, indexMetaData);

} else {

performRemoteAction(state, primary, node);

}

}

//调用performRemoteAction后执行

#TransportReplicationAction

private void performAction(...){

transportService.sendRequest(node,action,requestToPerform,transportOptions,new TransportResponseHandler<Response>() {}

}

//performLocalAction和performRemoteAction实现如下，最终都会调用

//performAction方法，不过传入的action分别为transportPrimaryAction

//（BulkAction.NAME + "[s][p]"）和actionName(BulkAction.NAME + "[s]")，

//具体节点会根据此action找到注册到transportService中的请求处理Handler，

//除了上面两种action，还有一种是副分片使用的action，即transportReplicaAction（BulkAction.NAME + "[s][r]"）

#TransportReplicationAction构造函数可看出，执行上诉的request会在handlePrimaryRequest和handleReplicaRequest

transportService.registerRequestHandler(transportPrimaryAction, executor, forceExecutionOnPrimary, true,in -> new ConcreteShardRequest<>(requestReader, in), this::handlePrimaryRequest);

// we must never reject on because of thread pool capacity on replicas

transportService.registerRequestHandler(transportReplicaAction, executor, true, true,

in -> new ConcreteReplicaRequest<>(replicaRequestReader, in), this::handleReplicaRequest);

protected void handlePrimaryRequest(...) {

new AsyncPrimaryAction().run();

}

#AsyncPrimaryAction

#ReplicationOperation

public void execute() throws Exception {

...

//执行成功后调用handlePrimaryResult()方法

primary.perform(request, ActionListener.wrap(this::handlePrimaryResult, resultListener::onFailure));

}

private void handlePrimaryResult(final PrimaryResultT primaryResult) {

if (replicaRequest != null) {

performOnReplicas(replicaRequest, globalCheckpoint, maxSeqNoOfUpdatesOrDeletes, replicationGroup);//执行副本

}

}

#TransportShardBulkAction

protected void shardOperationOnPrimary(BulkShardRequest request, IndexShard primary,

ActionListener<PrimaryResult<BulkShardRequest, BulkShardResponse>> listener) {

...

//shardOperationOnPrimary方法为TransportShardBulkAction中定义的写操作在主分片

performOnPrimary();

...

}

#IndexShard调用

private Engine.IndexResult index(Engine engine, Engine.Index index) throws IOException {

...

//调用internalEngine.index方法写入lucene

result = engine.index(index);

if (index.origin().isFromTranslog() == false) {

//写translog

location = translog.add(new Translog.Index(index, indexResult));

}

...

}