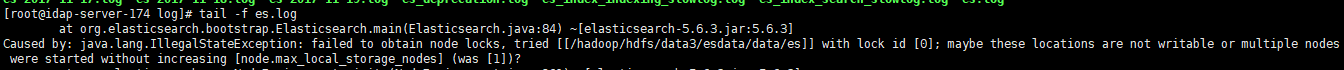
# 启动报错

## Node锁

Elasticsearch 重启的时候

failed to obtain node locks, tried [[/hadoop/hdfs/data3/esdata/data/es]] with lock id [0]; maybe these locations are not writable or multiple node



思路：node加锁导致的，读取权限问题：

Cd /hadoop/hdfs/data3/esdata/data/nodes

rm -rf node.lock

nohup /hadoop/hdfs/data3/esdata/elasticsearch/bin/elasticsearch &

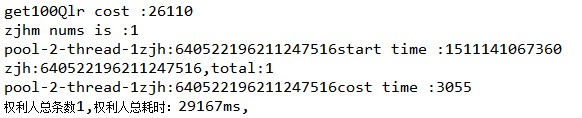
解决问题，恭喜自己。

# 调优

## 统计耗时太大

### 查询场景：查询权利人，条件：权利人+工作单位，查询权利人信息

总共50亿 客体10亿 权利人20亿 权利20亿



猜想：

jave gc调优问题

Elasticsearch jvm以外参数优化问题

#### 从gc上考虑

##### 查看es 配置信息

Jps -l

jinfo 36512

VM Flags:

Non-default VM flags: -XX:+AlwaysPreTouch -XX:CICompilerCount=15 -XX:CMSInitiatingOccupancyFraction=75 -XX:+HeapDumpOnOutOfMemoryError -XX:InitialHeapSize=21474836480 -XX:MaxHeapSize=21474836480 -XX:MaxNewSize=2442723328 -XX:MaxTenuringThreshold=6 -XX:MinHeapDeltaBytes=196608 -XX:NewSize=2442723328 -XX:OldPLABSize=16 -XX:OldSize=19032113152 -XX:ThreadStackSize=1024 -XX:+UseCMSInitiatingOccupancyOnly -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -XX:+UseFastUnorderedTimeStamps -XX:+UseParNewGC

Command line: -Xms20g -Xmx20g -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=75 -XX:+UseCMSInitiatingOccupancyOnly -XX:+AlwaysPreTouch -Xss1m -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djna.nosys=true -Djdk.io.permissionsUseCanonicalPath=true -Dio.netty.noUnsafe=true -Dio.netty.noKeySetOptimization=true -Dio.netty.recycler.maxCapacityPerThread=0 -Dlog4j.shutdownHookEnabled=false -Dlog4j2.disable.jmx=true -Dlog4j.skipJansi=true -XX:+HeapDumpOnOutOfMemoryError -Des.path.home=/hadoop/hdfs/data3/esdata/elasticsearch

假想：分配20G内存过大，gc卡顿时间过长导致

##### 查看GC情况 是否堵塞

jstat -gc 36512



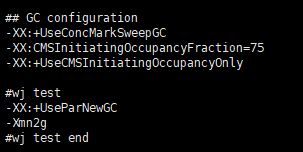
jstat -gcutil 36512





Es在进行查询的时候，耗时8秒左右，垃圾回收用了1.8秒，剩下的6.2秒应该全部是其他瓶颈导致。

后来修改jvm参数



## GC configuration

-XX:+UseConcMarkSweepGC

-XX:CMSInitiatingOccupancyFraction=75

-XX:+UseCMSInitiatingOccupancyOnly

#wj test

-XX:+UseParNewGC

-Xmn2g

#wj test end

性能提升妥妥的





#### 从Elasticsearch本身其他参数考虑

##### 获取es客户端需要负载均衡

/\*\*\*\*

\* 获取es客户端

\*

\* **@return** TransportClient

\* **@throws** UnknownHostException

\*/

**public** **static** TransportClient getClient1withNOxpack() **throws** UnknownHostException {

Settings settings = Settings.*builder*().put("cluster.name", *clustername*).build();

TransportClient client = **new** PreBuiltTransportClient(settings);

client.addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("10.110.13.174"), 9300));

client.addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("10.110.13.175"), 9300));

client.addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("10.110.13.176"), 9300));

client.addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("10.110.13.177"), 9300));

client.addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("10.110.13.178"), 9300));

client.addTransportAddress(**new** InetSocketTransportAddress(InetAddress.*getByName*("10.110.13.179"), 9300));

**return** client;

}

##### 对某字段多个值匹配进行优化

/\*\*\*\*\*\*

\* 查询出客体前100条记录，条件：坐落+行政区划，根据每条客体记录的不动产单元号，获取权利人 毫秒

\*/

**public** **static** **void** getQlrByZlXzqh(TransportClient client, String zl, String xzqh) **throws** UnknownHostException {

**long** start = System.*currentTimeMillis*();

// 首先获取课题的100条信息

SearchRequestBuilder ketiSearchRB = client.prepareSearch("keti\_1y").setTypes("keti\_1y").setSize(100);

BoolQueryBuilder ketiBoolQueryQueryBuilder1 = QueryBuilders.*boolQuery*()

.must(QueryBuilders.*termQuery*("records", "0")).filter(QueryBuilders.*termsQuery*("zl", zl))

.filter(QueryBuilders.*termsQuery*("qx", xzqh));

SearchResponse ketiResponse = ketiSearchRB.setQuery(ketiBoolQueryQueryBuilder1).execute().actionGet();

SearchHits ketiHits = ketiResponse.getHits();

**long** ketiEnd = System.*currentTimeMillis*();

**long** cost = ketiEnd - start;

System.***out***.println("keti costtime:" + cost + "ms");

List<String> bdcdyhList = **new** ArrayList<String>();

**for** (**int** i = 0; i < ketiHits.getHits().length; i++) {

bdcdyhList.add(String.*valueOf*(ketiHits.getHits()[i].getSource().get("bdcdyh")));

}

//对布尔过滤优化，可以传入collection对象

BoolQueryBuilder qlrBoolQueryQueryBuilder1 = QueryBuilders.*boolQuery*()

.filter(QueryBuilders.*termQuery*("records", "0")).filter(QueryBuilders.*termsQuery*("bdcdyh", bdcdyhList));

SearchRequestBuilder qlrRb = client.prepareSearch("qlr\_1y").setTypes("qlr\_1y").setSize(100);

SearchResponse response1 = qlrRb.setQuery(qlrBoolQueryQueryBuilder1).execute().actionGet();

SearchHits hits1 = response1.getHits();

**long** qlrCount = hits1.getTotalHits();

**for** (**int** j = 0; j < hits1.getHits().length; j++) {

// System.out.print("\txm:" +

// hits1.getHits()[j].getSource().get("xm") + "\tbdc:"

// + hits1.getHits()[j].getSource().get("bdcdyh") + "\tzjh:"

// + hits1.getHits()[j].getSource().get("zjh") + "\trecords:"

// + hits1.getHits()[j].getSource().get("records") + "\n");

}

**long** end = System.*currentTimeMillis*();

System.***out***.println("客体总条数" + ketiHits.getTotalHits() + ",总耗时：" + (end - start) + "ms," + qlrCount

+ "条 qlr cost:" + (end - ketiEnd));

}

##### 对布尔过滤的研究

//must 是与的关系

QueryBuilders.*boolQuery*().must(QueryBuilders.*termQuery*("records", "0"))

//shold是或的关系

QueryBuilders.*boolQuery*().shold(QueryBuilders.*termQuery*("records", "0"))

//只是过滤 不计分

QueryBuilders.*boolQuery*().filter(QueryBuilders.*termsQuery*(key, paramV));