ES连接池使用文档

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 修改日期 | 修改人 | 修改项 | 备注 |
| V1.0 | 2018-08-01 | 陈洪杰 | 初稿 |  |

## 项目背景

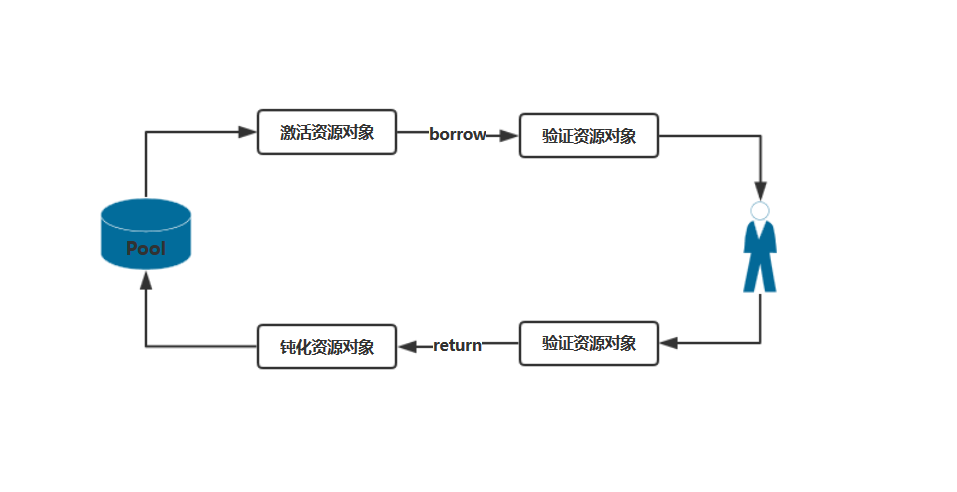
目前公司多个项目在使用elasticsearch，每套项目都写一套连接比较麻烦，而且也不好管理,所以将公共代码抽离出来，还有一些公共的操作抽离出来打成jar包。

## 项目实现

项目依赖于elasticsearch官方的rest-high-level-client客户端和common-pool2的对象池技术.将es客户端封装到对象池中.

### 连接池架构

连接池架构图如下:



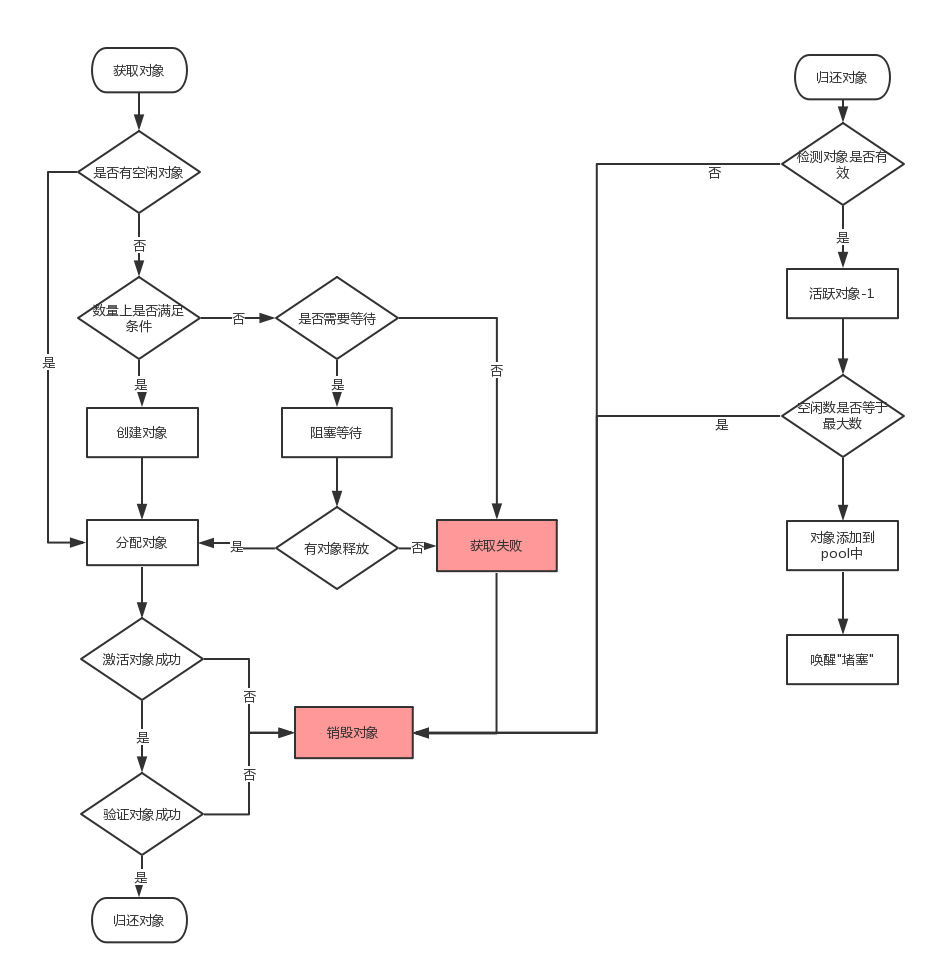
从以上的架构图中可以得出:

1、使用者发出获取对象资源的请求

2、池收到请求，激活对象，并且验证资源的有效性

3、使用者使用完对象资源返还给池中，并且验证该资源对象是否还有效，钝化对象，存入池中等待下次借出.

### 程序流程



## 使用方法

### 初始化连接池

#### 1、连接池设置

|  |
| --- |
| EsConnectionPool pool = EsPool.***ESCLIENT***.getPool();  //如果池已经初始化,先关闭再初始化.  **if**(pool!=**null**){  pool.close();  }  PoolConfig config = **new** PoolConfig();  //池中最大连接数 默认 8  config.setMaxTotal(8);  //最大空闲数,当超过这个数的时候关闭多余的连接 默认8  config.setMaxIdle(8);  //最少的空闲连接数 默认0  config.setMinIdle(0);  //当连接池资源耗尽时,调用者最大阻塞的时间,超时时抛出异常 单位:毫秒数 -1表示无限等待  config.setMaxWaitMillis(10000);  // 连接池存放池化对象方式,true放在空闲队列最前面,false放在空闲队列最后 默认为true  config.setLifo(**true**);  //连接空闲的最小时间,达到此值后空闲连接可能会被移除,默认即为30分钟  config.setMinEvictableIdleTimeMillis(1000L \* 60L \* 30L);  //连接耗尽时是否阻塞,默认为true,为false时则抛出异常  config.setBlockWhenExhausted(**true**);  //向调用者输出“链接”资源时，是否检测是有有效，如果无效则从连接池中移除，并尝试获取继续获取。默认为false。建议保持默认值.  config.setTestOnBorrow(**true**);  //把资源返回连接池时检查是否有效,默认为false  config.setTestOnReturn(**true**); |

#### 2、ES集群配置

|  |
| --- |
| EsConnectConfig esConnectConfig = **new** EsConnectConfig();  //设置集群名称  esConnectConfig.setClusterName("bigData-cluster");  //集群IP数组  String [] nodes={"192.169.2.98:9200","192.169.2.188:9200"};  esConnectConfig.setNodes(nodes);  //设置集群连接协议,默认http  esConnectConfig.setScheme(EsConnect.***HTTP***); |
|  |

#### 3、自定义ES网络配置(可选)

因为有时候业务的需要我们需要对ES去连接集群时进行一些设置,当然这些都是可不配置的，不配置的时候都会选默认值

|  |
| --- |
| //设置连接超时时间 默认值为:1000毫秒  esConnectConfig.setConnectTimeoutMillis(1000);  //设置连接请求超时时间 默认:500毫秒  esConnectConfig.setConnectionRequestTimeoutMillis(500);  //设置网络超时时间 默认:30秒  esConnectConfig.setSocketTimeoutMillis(30000);  //设置ES响应超时时间 默认:30秒  esConnectConfig.setMaxRetryTimeoutMillis(30000); |
|  |

这些配置主要就是对客户端去请求集群时候的网络设置,注意,有时候索引库数据大的时候往往需要响应时间久,这个时候我们就可以设置ES响应超时时间，但需要注意的是如果设置ES响应超时时间也需要设置网络超时时间,并且最好保证MaxRetryTimeoutMillis<=SocketTimeoutMillis

#### 4、初始化连接池

|  |
| --- |
| //把连接池配置和ES集群配置加载进池中  EsConnectionPool pool = **new** EsConnectionPool(config, esConnectConfig); |

#### 5、设置全局池对象

设置全局对象池,初始化到此结束

|  |
| --- |
| //设为程序全局可用这个连接池  EsPool.***ESCLIENT***.setPool(pool); |

## 四、ES操作索引库

因为在该jar包中还内置了一部分对索引库的操作的包装处理,其中包括，对索引库的创建库设置，对数据的新增(单条新增,批量新增)、修改、删除、聚合查询、滚动查询、分页查询。这些方法都是对ES原官方jar包的封装，使这些操作变得简单易使用。也可不用这些处理器自行使用官方的接口。这里我们都是以index为demo,type为demo进行操作

注意:每个处理函数都必须先实现builder方法才可使用,使用完成必须close

### 1、对index操作

|  |
| --- |
| IndexEsHandler index = **new** IndexEsHandler();  **try** {  //新建一个索引库  EsBasicModelConfig indexConfig = **new** EsBasicModelConfig("demo", "demo");  //设置数据模版(不设置ES会自动识别)  indexConfig.setMappings("{\"demo\": {\"properties\" : {\"id\": {\"type\": \"long\"}}}}");  //设置索引设置(默认 5个分片,一个副本)  //分片,副本,刷新,单次可取最大数据条数和分页深度  SettingEntity setting = **new** SettingEntity(5,1,5,12000); indexConfig.setSettings(setting);  //这里设置为null即可  index.builder(**null**);  //开始创建  *logger*.info("创建索引库:{}",index.createIndex(indexConfig));  //检查索引库是否存在  *logger*.info("检查索引库是否存在:{}",index.existsIndex("demo"));  } **catch** (Exception e) {  // **TODO**: handle exception  }**finally**{  index.close();  } |

如上面一样,其中数据模板就是类似我们数据库对没一个字段名参数的类型,只不过这里以json文件形式传入索引中，建议设置数据模板。而setting则是设置这个索引有多少分片和副本，如果数据不大都取默认值即可。

mapping的数据模板设置参考官方这个连接

https://www.elastic.co/guide/en/elasticsearch/reference/6.3/mapping-types.html

### 2、保存插入

对ES的保存插入一共有两种方式,单条插入和批量插入

|  |
| --- |
| EsRequestEntity esRequestEntity = **new** EsRequestEntity("demo","demo");  InsertEsHandler insert = **new** InsertEsHandler();  **try** {  //存在更新不存在插入 true:开启 false：关闭(默认)  insert.docAsUpsert(**true**);  //要插入的内容  List<Map<String,Object>> list = **new** ArrayList<>();  Map<String,Object> map = **new** HashMap<>();  map.put("id",523456);  list.add(map);  //自定义ID值,从数据map中获取,如果不自定义取消掉即可,不定义将自动生成ID  insert.setIdFieldName("id");  //设置配置  insert.builder(esRequestEntity);  //执行批量插入  insert.insertBulk(list);  //执行单条插入  insert.insertOne(map);  //查看是否有插入失败的数据  *logger*.info("插入失败数据:{}",insert.getListFailuresData());  } **catch** (Exception e) {  // **TODO**: handle exception  }**finally**{  insert.close();  } |

插入支持是否开始存在更新不存在插入、自动生成ID,如果开启自动生成ID,如果批量插入失败将会把失败的数据存到ListFailuresData中，如果这个值为空，那就是没有保存失败的数据.

### 3、查询数据

查询输入条件查询数据,可以指定返回多少条数据和排序这些功能

#### 3.1总数查询

|  |
| --- |
| EsRequestEntity esRequestEntity = **new** EsRequestEntity("demo","demo");  SearchEsHandler search = **new** SearchEsHandler();  **try** {  //设置条件  //search.setQueryBuilder(null);  //设置配置  search.builder(esRequestEntity);  //根据条件查询总数  *logger*.info("查询总数:{}",search.count());  //获取条件DSL,可直接在Kibana中直接运行  *logger*.info("总数查询DSL:{}",search.toDSL());  } **catch** (Exception e) {  // **TODO**: handle exception  }**finally** {  //使用完关闭,避免造成堵塞  search.close();  } |

#### 3.2分页查询

|  |
| --- |
| EsRequestEntity esRequestEntity = **new** EsRequestEntity("demo","demo");  //实例一个查询对象  SearchEsHandler search = **new** SearchEsHandler();  **try** {  //注意:浅分页建议用,深分页不建议使用  //设置需要分页  esRequestEntity.setNeedPaging(**true**);  //页数,默认1  esRequestEntity.setPageNo(1);  //每页的数据量,默认10条  esRequestEntity.setPageSize(1);  //设置配置(可不设置)  search.addSort("id", SortOrder.***DESC***);  //设置配置  search.builder(esRequestEntity);  //执行查询，dataList为查询出来的数据  List<Map<String,Object>> dataList=search.sraechSourceAsList();  *logger*.info("分页查询数据:{}",dataList);  //该分页数据总量,只有执行分页了才能得到数据总量  **long** total = search.getTotalCount();  *logger*.info("分页查询总数:{}",total);  *logger*.info("分页查询DSL:{}",search.toDSL());  } **catch** (Exception e) {  }**finally**{  search.close();  } |

分页查询建议在浅分页使用,深分页最好使用滚动查询

#### 3.3 scroll滚动查询

|  |
| --- |
| EsRequestEntity esRequestEntity = **new** EsRequestEntity("demo","demo");  ScrollEntity<Map<String,Object>> scrollEntity = **new** ScrollEntity<>();  //实例一个查询对象  SearchEsHandler search = **new** SearchEsHandler();  **try** {  //设置滚动间隔  scrollEntity.setKeepAlive(10);  //设置配置  search.builder(esRequestEntity);  List<Map<String,Object>> dataList=**null**;  **do**{  //执行遍历  scrollEntity=search.searchScroll(scrollEntity);  //获取数据  dataList=scrollEntity.getDataList();  *logger*.info("滚动查询数据:{}",dataList);  //对数据进行处理的逻辑....  //如果List为空则是取不到数据了,意味着已经取出完毕  }**while**(Objects.*nonNull*(dataList)&&!dataList.isEmpty());  *logger*.info("滚动查询DSL:{}",search.toDSL());  //清除遍历ID(可选)  search.clearScroll(scrollEntity.getScrollId());  } **catch** (Exception e) {  // **TODO**: handle exception  }**finally**{  search.close();  } |

滚动查询类似数据库的游标查询,滚动查询只能向后查询,可以设置

esRequestEntity.setLimit(10000);

来设置每次查询出来的条数

#### 3.4聚合查询

聚合查询可以参考官方文档,以下为一个简单的聚合查询：

|  |
| --- |
| EsRequestEntity esRequestEntity = **new** EsRequestEntity("demo","demo");  //以下聚合类似select recvtime as datatm,mftid,count() as msgNum from im\_msg group by recvtime,mftid  //自定义时间  String DEFAULT\_FORMAT="yyyyMMdd";  AggregationEsHandler aggsearch = **new** AggregationEsHandler();  **try** {  //设置聚合字段  AggregationBuilder datatm=AggregationBuilders.*dateHistogram*("datatm").field("recvtime").format(DEFAULT\_FORMAT).interval(86400000);  AggregationBuilder mftId=AggregationBuilders.*terms*("mftid").field("mftid");  //合并聚合字段  datatm.subAggregation(mftId);  //设置聚合对象  aggsearch.setAggregationBuilder(datatm);  //设置配置  aggsearch.builder(esRequestEntity);  //执行聚合  List<Map<String, Object>> resultList =aggsearch.sraechAgg("msgNum");  } **catch** (Exception e) {  // **TODO**: handle exception  }**finally**{  aggsearch.close();  } |

### 4、更新数据

数据的更新较为简单

|  |
| --- |
| EsRequestEntity esRequestEntity = **new** EsRequestEntity("demo","demo");  /\*\*\*\*\*\*\*\*\*\*\*\*数据更新\*\*\*\*\*\*\*\*\*\*\*\*\*/  UpdateEsHandler update = **new** UpdateEsHandler();  **try** {  //要更新的内容  List<Map<String,Object>> list = **new** ArrayList<>();  //更新的数据  Map<String,Object> map = **new** HashMap<>();  map.put("id",523456);  list.add(map);  //必须需要,否则抛出异常  update.setIdFieldName("id");  //设置配置  update.builder(esRequestEntity);  //执行批量更新  update.updateBulk(list);  //执行单条更新  update.updateOne(map);  } **catch** (Exception e) {  }**finally**{  update.close();} |

注意:目前更新只能根据ID来更新,还不能自定义条件更新,后续将想办法升级出来的，但是如果需要批量根据条件更新数据,可以使用\_update\_by\_query,如下面这个例子

POST mt\_msg/mt\_msg/\_update\_by\_query?conflicts=proceed

{ "script": {

"lang": "painless",

"source":"ctx.\_source['message']='test'"

},"query":{"bool":{"must":[{"term":{"msgcode":{"value":0,"boost":1.0}}}] ,"adjust\_pure\_negative":true,"boost":1.0}}}

### 5、删除数据

数据的删除操作代码如下

|  |
| --- |
| EsRequestEntity esRequestEntity = **new** EsRequestEntity("demo","demo");  /\*\*\*\*\*\*\*\*\*\*\*\*数据删除\*\*\*\*\*\*\*\*\*\*\*\*\*/  DeleteEsHandler del = **new** DeleteEsHandler();  **try** {  //设置条件  del.setQueryBuilder(**null**);  //创建接口  del.builder(esRequestEntity);  //根据ID删除数据  del.delById("id值");  //注意:true为同步删除 ,false为异步删除 需要删除量大时建议使用异步  del.delDocByQuery(**true**);  *logger*.info("分页查询DSL:{}",del.toDSL());  } **catch** (Exception e) {  // **TODO**: handle exception  }**finally**{  del.close();  } |

在删除数据量大时尽量使用异步删除,因为ES集群收到删除命令后就算超时没有响应出来，该命令还是会在后台执行的,我们可以使用以下命令在kibana平台中看到正在执行的删除线程，并且可以看到执行到哪里。

##查看正在进行的删除任务

GET \_tasks?detailed=true&actions=\*/delete/byquery

如果删除过程中需要停止删除线程,可以使用以下命令进行停止.但已删除的数据不会回滚.

##取消任务

POST \_tasks/yXFoabQLSJidu-MrLX4lLQ:2229/\_cancel

## 五、条件设置

因为ES官方给的条件设置比较多，比较繁杂，但往往我们只需要其中一两个,这里我抽离大于***gt***,大于等于***gte***,小于***lt***,小于等于***lte***,等于***equal***,不等于***unequal***,字段存在***exist***,字段不存在***unexist***。八个维度的与或查询

|  |
| --- |
| ConditionLogic con = **new** ConditionLogic()  //\*\*\*\*\*\*\*以下开始设置条件\*\*\*\*\*\*\*\*\*\*/  //name字段必须存在  .and(ConditionType.***exist***,"name")  //age字段必须不存在  .and(ConditionType.***unexist***,"age")  //class 必须等于 12  .and(ConditionType.***equal***, "class",12)  //name 必须不等于lilin  .and(ConditionType.***unequal***, "bb","lilin")  //creattm 必须大于2013-08-02  .and(ConditionType.***gt***, "creattm","2013-08-02")  //creattm 必须大于等于2013-08-02  .and(ConditionType.***gte***, "creattm","2013-08-02")  //creattm 必须小于2013-08-02  .and(ConditionType.***lt***, "creattm","2013")  //creattm 必须小于等于2013-08-02  .and(ConditionType.***lte***, "creattm","2013")  //name字段或存在  .or(ConditionType.***exist***,"name")  //age字段或不存在  .or(ConditionType.***unexist***,"age")  //class 或等于 12  .or(ConditionType.***equal***, "class",12)  //name 或不等于lilin  .or(ConditionType.***unequal***, "bb","lilin")  //creattm 或大于2013-08-02  .or(ConditionType.***gt***, "creattm","2013-08-02")  //creattm 或大于等于2013-08-02  .or(ConditionType.***gte***, "creattm","2013-08-02")  //creattm 或小于2013-08-02  .or(ConditionType.***lt***, "creattm","2013")  //creattm 或小于等于2013-08-02  .or(ConditionType.***lte***, "creattm","2013");  org.elasticsearch.index.query.QueryBuilder queryBuilder =con.toResult();  //可直接在kibana中使用  System.***out***.println("打印生成的查询JSON："+con.toDSL()); |

得到queryBuilder对象可以在处理器中setQueryBuilder(queryBuilder)进行设置