# 0.场景说明

目前Mysql 数据库数据量为千万级，直接在Mysql查询出现各种效率问题，因此想着将数据导一份到ES，从而实现大数据快速检索的功能。

通过Logstash插件批量导数据，会出现各种奇怪的问题，例如ES 内存暴满，mysql 所在服务器内存暴，最主要的是在一次导数时不能所导的数据量不能太大。经过一次次试探Logstash与优化Logstash导数的最后，终于还是动手直接运用ES提供的api进行导入了。

目前直接模拟测试批量导数据，无论是通过Logstash还是ES 提供的Api峰值均能达到4000每秒左右。下面上代码，主要是通过官网提供的api （RestHighLevelClient、BulkProcessor）整理而来。目前由于Mysql 查询出来的数据需要进行一些处理，基本可达到2000-3000每秒(主要受限于服务器性能和参数配置以及Mysql数据本身的大小)。这个速度还有不小的优化空间，比如通过稍微修改下述代码，启动几个线程同时执行bulk多张表，从kibana界面看出速度达到了成倍的提升.

# 一、Maven配置

|  |
| --- |
| <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>ElasticSearchDemo</groupId>  <artifactId>ElasticSearchDemo</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>ElasticSearchDemo</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>3.8.1</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.elasticsearch.client</groupId>  <artifactId>elasticsearch-rest-high-level-client</artifactId>  <version>6.6.2</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.11</version>  </dependency>  <dependency>  <groupId>org.apache.logging.log4j</groupId>  <artifactId>log4j-core</artifactId>  <version>2.11.1</version>  </dependency>  </dependencies>  </project> |

# 二、代码示例

|  |
| --- |
| package service;  import java.io.IOException;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.ResultSetMetaData;  import java.util.ArrayList;  import java.util.HashMap;  import java.util.concurrent.TimeUnit;  import java.util.function.BiConsumer;  import org.apache.http.HttpHost;  import org.apache.logging.log4j.LogManager;  import org.apache.logging.log4j.Logger;  import org.elasticsearch.action.ActionListener;  import org.elasticsearch.action.admin.indices.create.CreateIndexRequest;  import org.elasticsearch.action.admin.indices.create.CreateIndexResponse;  import org.elasticsearch.action.bulk.BackoffPolicy;  import org.elasticsearch.action.bulk.BulkProcessor;  import org.elasticsearch.action.bulk.BulkRequest;  import org.elasticsearch.action.bulk.BulkResponse;  import org.elasticsearch.action.index.IndexRequest;  import org.elasticsearch.client.RequestOptions;  import org.elasticsearch.client.RestClient;  import org.elasticsearch.client.RestHighLevelClient;  import org.elasticsearch.common.settings.Settings;  import org.elasticsearch.common.unit.ByteSizeUnit;  import org.elasticsearch.common.unit.ByteSizeValue;  import org.elasticsearch.common.unit.TimeValue;  import utils.DBHelper;  /\*\*  \* @author Ye  \* @time 2019年3月29日  \*  \* 类说明：通过BulkProcess批量将Mysql数据导入ElasticSearch中  \*/  public class BulkProcessDemo {    private static final Logger logger = LogManager.getLogger(BulkProcessDemo.class);  public static void main(String[] args) {  try {  long startTime = System.currentTimeMillis();  String tableName = "testTable";  createIndex(tableName);  writeMysqlDataToES(tableName);  logger.info(" use time: " + (System.currentTimeMillis() - startTime) / 1000 + "s");  } catch (Exception e) {  logger.error(e.getMessage());  e.printStackTrace();  }  }  /\*\*  \* 创建索引  \* @param indexName  \* @throws IOException  \*/  public static void createIndex(String indexName) throws IOException {  RestHighLevelClient client = new RestHighLevelClient(RestClient.builder(new HttpHost("es01", 9200, "http")));  // ES 索引默认需要小写，故笔者将其转为小写  CreateIndexRequest requestIndex = new CreateIndexRequest(indexName.toLowerCase());  // 注： 设置副本数为0，索引刷新时间为-1对大批量索引数据效率的提升有不小的帮助  requestIndex.settings(Settings.builder().put("index.number\_of\_shards", 5)  .put("index.number\_of\_replicas", 0)  .put("index.refresh\_interval", "-1"));  // CreateIndexResponse createIndexResponse = client.indices().create(requestIndex, RequestOptions.DEFAULT);  client.close();  }  /\*\*  \* 将mysql 数据查出组装成es需要的map格式，通过批量写入es中  \*  \* @param tableName  \*/  private static void writeMysqlDataToES(String tableName) {  RestHighLevelClient client = new RestHighLevelClient(RestClient.builder(new HttpHost("eshost", 9200, "http")));// 初始化  BulkProcessor bulkProcessor = getBulkProcessor(client);  Connection conn = null;  PreparedStatement ps = null;  ResultSet rs = null;  try {  conn = DBHelper.getConn();  logger.info("Start handle data :" + tableName);  String sql = "SELECT \* from " + tableName;  ps = conn.prepareStatement(sql, ResultSet.TYPE\_FORWARD\_ONLY, ResultSet.CONCUR\_READ\_ONLY);  ps.setFetchSize(Integer.MIN\_VALUE);  rs = ps.executeQuery();  ResultSetMetaData colData = rs.getMetaData();  ArrayList<HashMap<String, String>> dataList = new ArrayList<HashMap<String, String>>();  // bulkProcessor 添加的数据支持的方式并不多，查看其api发现其支持map键值对的方式，故笔者在此将查出来的数据转换成hashMap方式  HashMap<String, String> map = null;  int count = 0;  String c = null;  String v = null;  while (rs.next()) {  count++;  map = new HashMap<String, String>(128);  for (int i = 1; i <= colData.getColumnCount(); i++) {  c = colData.getColumnName(i);  v = rs.getString(c);  map.put(c, v);  }  dataList.add(map);  // 每10万条写一次，不足的批次的最后再一并提交  if (count % 100000 == 0) {  logger.info("Mysql handle data number : " + count);  // 将数据添加到 bulkProcessor 中  for (HashMap<String, String> hashMap2 : dataList) {  bulkProcessor.add(new IndexRequest(tableName.toLowerCase(), "gzdc", hashMap2.get("S\_GUID"))  .source(hashMap2));  }  // 每提交一次便将map与list清空  map.clear();  dataList.clear();  }  }  // count % 100000 处理未提交的数据  for (HashMap<String, String> hashMap2 : dataList) {  bulkProcessor.add(  new IndexRequest(tableName.toLowerCase(), "gzdc", hashMap2.get("S\_GUID")).source(hashMap2));  }  logger.info("-------------------------- Finally insert number total : " + count);  // 将数据刷新到es, 注意这一步执行后并不会立即生效，取决于bulkProcessor设置的刷新时间  bulkProcessor.flush();    } catch (Exception e) {  logger.error(e.getMessage());  } finally {  try {  rs.close();  ps.close();  conn.close();  boolean terminatedFlag = bulkProcessor.awaitClose(150L, TimeUnit.SECONDS);  client.close();  logger.info(terminatedFlag);  } catch (Exception e) {  logger.error(e.getMessage());  }  }  }  /\*\*  \* 创建bulkProcessor并初始化  \* @param client  \* @return  \*/  private static BulkProcessor getBulkProcessor(RestHighLevelClient client) {  BulkProcessor bulkProcessor = null;  try {  BulkProcessor.Listener listener = new BulkProcessor.Listener() {  @Override  public void beforeBulk(long executionId, BulkRequest request) {  logger.info("Try to insert data number : " + request.numberOfActions());  }  @Override  public void afterBulk(long executionId, BulkRequest request, BulkResponse response) {  logger.info("\*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : " + request.numberOfActions() + " , id: "  + executionId);  }  @Override  public void afterBulk(long executionId, BulkRequest request, Throwable failure) {  logger.error("Bulk is unsuccess : " + failure + ", executionId: " + executionId);  }  };  BiConsumer<BulkRequest, ActionListener<BulkResponse>> bulkConsumer = (request, bulkListener) -> client  .bulkAsync(request, RequestOptions.DEFAULT, bulkListener);  // bulkProcessor = BulkProcessor.builder(bulkConsumer, listener).build();  BulkProcessor.Builder builder = BulkProcessor.builder(bulkConsumer, listener);  builder.setBulkActions(5000);  builder.setBulkSize(new ByteSizeValue(100L, ByteSizeUnit.MB));  builder.setConcurrentRequests(10);  builder.setFlushInterval(TimeValue.timeValueSeconds(100L));  builder.setBackoffPolicy(BackoffPolicy.constantBackoff(TimeValue.timeValueSeconds(1L), 3));    bulkProcessor = builder.build();  } catch (Exception e) {  e.printStackTrace();  try {  bulkProcessor.awaitClose(100L, TimeUnit.SECONDS);  client.close();  } catch (Exception e1) {  logger.error(e1.getMessage());  }  }  return bulkProcessor;  }  ｝ |

数据库连接类

|  |
| --- |
| package utils;  import java.sql.Connection;  import java.sql.DriverManager;    public class DBHelper {    public static final String url = "jdbc:mysql://xx.xx.xx.xx:3306/xxdemo?useSSL=true";  public static final String name = "com.mysql.cj.jdbc.Driver";  public static final String user = "xxx";  public static final String password = "xxxx";    public static Connection conn = null;    public static Connection getConn() {  try {  Class.forName(name);  conn = DriverManager.getConnection(url, user, password);//获取连接  } catch (Exception e) {  e.printStackTrace();  }  return conn;  }  } |

日志文件配置文件：log4j2.properties

|  |
| --- |
| property.filePath=logs  property.filePattern=logs/%d{yyyy}/%d{MM}  #\u8F93\u51FA\u683C\u5F0F  property.layoutPattern=%-d{yyyy-MM-dd HH:mm:ss SSS} [ %p ] [ %c ] %m%n  rootLogger.level = info      appender.console.type = Console  appender.console.name = STDOUT  appender.console.target = SYSTEM\_OUT  appender.console.layout.type = PatternLayout  appender.console.layout.pattern = ${layoutPattern}  rootLogger.appenderRef.stdout.ref = STDOUT    appender.I.type = RollingFile  appender.I.name = InfoRollingFile  appender.I.fileName = ${filePath}/es-info.log  appender.I.filePattern = ${filePattern}/es\_info.log  appender.I.layout.type = PatternLayout  appender.I.layout.pattern = ${layoutPattern}  appender.I.policies.type = Policies  appender.I.policies.time.type = TimeBasedTriggeringPolicy  appender.I.policies.time.interval = 1  appender.I.policies.time.modulate = true  appender.I.policies.size.type = SizeBasedTriggeringPolicy  appender.I.policies.size.size=20M  appender.I.strategy.type = DefaultRolloverStrategy  appender.I.strategy.max = 100  #\u8FC7\u6EE4INFO\u4EE5\u4E0A\u4FE1\u606F  appender.I.filter.threshold.type = ThresholdFilter  appender.I.filter.threshold.level = WARN  appender.I.filter.threshold.onMatch = DENY  appender.I.filter.threshold.onMisMatch=NEUTRAL    rootLogger.appenderRef.I.ref = InfoRollingFile  rootLogger.appenderRef.I.level=INFO      appender.E.type = RollingFile  appender.E.name = ErrorRollingFile  appender.E.fileName = ${filePath}/es-error.log  appender.E.filePattern = ${filePattern}/es\_error.log  appender.E.layout.type = PatternLayout  appender.E.layout.pattern = ${layoutPattern}  appender.E.policies.type = Policies  appender.E.policies.time.type = TimeBasedTriggeringPolicy  appender.E.policies.time.interval = 1  appender.E.policies.time.modulate = true  appender.E.policies.size.type = SizeBasedTriggeringPolicy  appender.E.policies.size.size=20M  appender.E.strategy.type = DefaultRolloverStrategy  appender.E.strategy.max = 100  #\u8FC7\u6EE4ERROR\u4EE5\u4E0A\u4FE1\u606F  appender.E.filter.threshold.type = ThresholdFilter  appender.E.filter.threshold.level = FATAL  appender.E.filter.threshold.onMatch = DENY  appender.E.filter.threshold.onMisMatch=NEUTRAL    rootLogger.appenderRef.E.ref = ErrorRollingFile  rootLogger.appenderRef.E.level=ERROR |

# 三、验证上述代码结果

将代码打成jar包后，通过java -jar xxx.jar 方式在服务器中运行, 依据服务器性能的不同，导入速度也有快慢。

运行的部分日志如下：

|  |
| --- |
| 2019-03-29 17:31:34 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2679  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Mysql handle data number : 13500000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2681  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2683  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2682  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:36 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2685  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2686  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2687  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2684  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2688  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2689  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2691  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2693  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2692  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2690  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] Try to insert data number : 5000  2019-03-29 17:31:37 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2696  2019-03-29 17:31:38 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2694  2019-03-29 17:31:38 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2695  2019-03-29 17:31:38 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2697  2019-03-29 17:31:38 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2698  2019-03-29 17:31:38 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2699  2019-03-29 17:31:38 [ INFO ] [ service.BulkProcessDemo ] \*\*\*\*\*\*\*\*\*\*\*\*\*\* Success insert data number : 5000 , id: 2700 |

原文地址: <https://blog.csdn.net/u013850277/article/details/88904303>