链滴

整合 Elasticsearch

作者：QForever

原文链接：https://ld246.com/article/1574595932646

来源网站：链滴

一、 依赖包

```
<?xml version="1.0" encoding="UTF-8"?>
<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>com.qiang</groupId>
    <artifactId>Elasticsearch</artifactId>
    <version>1.0.0-SNAPSHOT</version>

    <dependencies>
        <!-- 测试依赖 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
        <!-- elasticsearch依赖 -->
        <dependency>
            <groupId>org.elasticsearch</groupId>
            <artifactId>elasticsearch</artifactId>
            <version>6.4.3</version>
        </dependency>
        <dependency>
            <groupId>org.elasticsearch.client</groupId>
            <artifactId>transport</artifactId>
            <version>6.4.3</version>
        </dependency>
        <!-- 日志依赖 -->
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.10.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.10.0</version>
        </dependency>
    </dependencies>

</project>
```

二、 连接到Elasticsearch

```
public class TransportClientUtils {

    public TransportClient transportClient() {
```

```
        TransportClient transportClient = null;
        try {
            Settings settings = Settings.builder()
                    //自动发现新加入集群的机器
                    .put("client.transport.sniff", true)
                    .put("cluster.name", "elasticsearch").build();
            transportClient = new PreBuiltTransportClient(settings)
                    .addTransportAddress(new TransportAddress(new InetSocketAddress("192.168.17
222", 9300)));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return transportClient;
    }
}
```

三、 操作Elasticsearch

数据

_index	_type	_id	_score ▲	名字	颜色	时间	@timestamp
animals	dogs	PVGa5W4BCACHllmZVTes	1	傻狗1	白色1	20191208210207056	2019-12-08T13:02:07.068Z
animals	dogs	QFGa5W4BCACHllmZVjdF	1	傻狗2	白色2	20191208210207430	2019-12-08T13:02:07.430Z
animals	dogs	QVGa5W4BCACHllmZVjfF	1	傻狗3	白色3	20191208210207556	2019-12-08T13:02:07.555Z

```
public class ElasticsearchBuilder {
    TransportClient transportClient = new TransportClientUtils().transportClient();

    @Test
    public void test1() throws Exception {
        //判断索引是否存在
        boolean exists = transportClient.admin().indices().prepareExists("animals").execute().acti
nGet().isExists();
        System.out.println(exists);
    }

    @Test
    public void test2() throws Exception {
        //添加
        for (int i = 1; i <= 3; i++) {
            long s = Long.parseLong(new SimpleDateFormat("yyyyMMddHHmmssSSS").format(n
w Date()));
            IndexResponse indexResponse = transportClient.prepareIndex("animals", "dogs")
                    .setSource(jsonBuilder()
                            .startObject()
                            .field("名字", "傻狗" + i)
                            .field("颜色", "白色" + i)
                            .field("时间", s)
                            .field("@timestamp", new Date())
                            .endObject())
                    .get();
            Thread.sleep(100);
            System.out.println(indexResponse);
        }
```

```
    }

    @Test
    public void test3() throws Exception {
        //删除
        DeleteResponse deleteResponse = transportClient.prepareDelete("animals", "dogs", "xV
65G4BCACHllmZ5ygu").get();
        System.out.println(deleteResponse);
    }

    @Test
    public void test4() throws Exception {
        //修改
        UpdateResponse updateResponse = transportClient.prepareUpdate("animals", "dogs", "
FA65G4BCACHllmZ5ygi").setDoc(jsonBuilder()
                .startObject()
                .field("颜色", "黑色9")
                .endObject())
                .get();
        System.out.println(updateResponse);
    }

    @Test
    public void test5() throws Exception {
        //根据ID查询
        GetResponse getResponse = transportClient.prepareGet("animals", "dogs", "xFA65G4BC
CHllmZ5ygi").get();
        String sourceAsString = getResponse.getSourceAsString();
        System.out.println(sourceAsString);
    }

    @Test
    public void test6() throws Exception {
        //根据索引查询,默认10条，setSize()设置大小
        String index = "animals";
        QueryBuilder query = QueryBuilders.matchAllQuery();
        SearchResponse response = transportClient.prepareSearch(index).setQuery(query).setSiz
(100).execute().actionGet();
        for (SearchHit searchHit : response.getHits()) {
            String sourceAsString = searchHit.getSourceAsString();
            System.out.println(sourceAsString);
        }
    }

    @Test
    public void test7() throws Exception {
        //根据类型查询,默认10条，setSize()设置大小
        SearchResponse response = transportClient.prepareSearch("animals").setTypes("dogs").s
tSize(100).execute().actionGet();
        for (SearchHit searchHit : response.getHits()) {
            String sourceAsString = searchHit.getSourceAsString();
            System.out.println(sourceAsString);
        }
    }
```

```
@Test
public void test8() throws Exception {
    //单条件查询
    QueryBuilder queryBuilder = QueryBuilders.termQuery("名字.keyword", "傻狗3");
    SearchResponse response = transportClient.prepareSearch("animals").setTypes("dogs")
            .setQuery(queryBuilder)
            .setFrom(0).setSize(1000).setExplain(true)
            .execute()
            .actionGet();
    //直接遍历
    for (SearchHit searchHit : response.getHits()) {
        String sourceAsString = searchHit.getSourceAsString();
        System.out.println(sourceAsString);
    }
    //存入集合后再遍历
    SearchHits hits = response.getHits();
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
    for (int i = 0; i < hits.getHits().length; i++) {
        Map<String, Object> maps = hits.getAt(i).getSourceAsMap();
        list.add(maps);
    }
    for (Object o : list) {
        System.out.println(o);
    }
}

@Test
public void test9() throws Exception {
    //多条件查询
    Map<String, String> map = new HashMap<String, String>();
    map.put("名字.keyword", "傻狗3");
    map.put("颜色.keyword", "白色3");
    BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
    for (String key : map.keySet()) {
        //得到每个key的value值
        String value = map.get(key);
        boolQueryBuilder.must(QueryBuilders.termQuery(key, value));
    }
    SearchResponse response = transportClient.prepareSearch("animals").setTypes("dogs")
            .setQuery(boolQueryBuilder)
            .setFrom(0).setSize(1000).setExplain(true)
            .execute()
            .actionGet();
    //直接遍历
    for (SearchHit searchHit : response.getHits()) {
        String sourceAsString = searchHit.getSourceAsString();
        System.out.println(sourceAsString);
    }
    //存入集合后再遍历
    SearchHits hits = response.getHits();
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
    for (int i = 0; i < hits.getHits().length; i++) {
        Map<String, Object> maps = hits.getAt(i).getSourceAsMap();
```

```
            list.add(maps);
        }
        for (Object o : list) {
            System.out.println(o);
        }
    }

    @Test
    public void test10() throws Exception {
        //单条件模糊查询
        WildcardQueryBuilder wBuilder = QueryBuilders.wildcardQuery("名字.keyword", "*" + "
狗" + "*");
        SearchResponse response = transportClient.prepareSearch("animals").setTypes("dogs")
                .setQuery(wBuilder)
                .setFrom(0).setSize(10000).setExplain(true)
                .execute()
                .actionGet();
        //直接遍历
        for (SearchHit searchHit : response.getHits()) {
            String sourceAsString = searchHit.getSourceAsString();
            System.out.println(sourceAsString);
        }
        //存入集合后再遍历
        SearchHits hits = response.getHits();
        List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
        for (int i = 0; i < hits.getHits().length; i++) {
            Map<String, Object> maps = hits.getAt(i).getSourceAsMap();
            list.add(maps);
        }
        for (Object o : list) {
            System.out.println(o);
        }
    }

    @Test
    public void test11() throws Exception {
        //多条件模糊查询
        Map<String, String> map = new HashMap<String, String>();
        map.put("名字.keyword", "傻狗3");
        map.put("颜色.keyword", "白色3");
        BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
        for (String key : map.keySet()) {
            //得到每个key的value值
            String value = map.get(key);
            boolQueryBuilder.must(QueryBuilders.wildcardQuery(key, "*" + value + "*"));
        }
        SearchResponse response = transportClient.prepareSearch("animals").setTypes("dogs")
                .setQuery(boolQueryBuilder)
                .setFrom(0).setSize(10000).setExplain(true)
                .execute()
                .actionGet();
        //直接遍历
        for (SearchHit searchHit : response.getHits()) {
            String sourceAsString = searchHit.getSourceAsString();
```

```
            System.out.println(sourceAsString);
        }
        //存入集合后再遍历
        SearchHits hits = response.getHits();
        List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
        for (int i = 0; i < hits.getHits().length; i++) {
            Map<String, Object> maps = hits.getAt(i).getSourceAsMap();
            list.add(maps);
        }
        for (Object o : list) {
            System.out.println(o);
        }
    }

    @Test
    public void test12() throws Exception {
        //模糊查询
        //设置索引和类型
        SearchRequestBuilder requestBuilder = transportClient.prepareSearch("animals").setType
("dogs");
        //设置模糊查询条件
        WildcardQueryBuilder queryBuilder = QueryBuilders.wildcardQuery("名字.keyword", "*狗*
");
        //开始查询
        SearchRequestBuilder searchRequestBuilder = requestBuilder.setQuery(queryBuilder);
        //设置多少条数据
        SearchResponse responses = searchRequestBuilder.setFrom(0).setSize(10).execute().acti
nGet();
        for (SearchHit searchHit : responses.getHits()) {
            String sourceAsString = searchHit.getSourceAsString();
            System.out.println(sourceAsString);
        }
    }

    @Test
    public void test13() throws Exception {
        //求最大值
        AggregationBuilder max = AggregationBuilders.max("max").field("时间");
        SearchResponse sr = transportClient.prepareSearch("animals").addAggregation(max).get(
;
        Max maxValue = sr.getAggregations().get("max");
        long value = (long) maxValue.getValue();
        System.out.println(value);
    }

    @Test
    public void test14() throws Exception {
        //求最小值
        AggregationBuilder min = AggregationBuilders.min("min").field("时间");
        SearchResponse sr = transportClient.prepareSearch("animals").addAggregation(min).get(
;
        Min minValue = sr.getAggregations().get("min");
        long value = (long) minValue.getValue();
        System.out.println(value);
```

```
    }

    @Test
    public void test15() throws Exception {
        //求平均值
        AggregationBuilder avg = AggregationBuilders.avg("avg").field("时间");
        SearchResponse sr = transportClient.prepareSearch("animals").addAggregation(avg).get()

        Avg avgValue = sr.getAggregations().get("avg");
        long value = (long) avgValue.getValue();
        System.out.println(value);
    }

    @Test
    public void test16() throws Exception {
        //求和的值
        AggregationBuilder sum = AggregationBuilders.sum("sum").field("时间");
        SearchResponse sr = transportClient.prepareSearch("animals").addAggregation(sum).get
);
        Sum sumValue = sr.getAggregations().get("sum");
        long value = (long) sumValue.getValue();
        System.out.println(value);
    }

    @Test
    public void test17() throws Exception {
        //时间范围查询
        SearchResponse response = transportClient
                .prepareSearch("animals")
                .setTypes("dogs")
                .setQuery(QueryBuilders
                        .rangeQuery("@timestamp")
                        .gte("2019-12-08T13:02:07.069Z").lte("2019-12-08T13:02:07.554Z"))
                .setFrom(0).setSize(10000).setExplain(true)
                .execute()
                .actionGet();
        //直接遍历
        for (SearchHit searchHit : response.getHits()) {
            String sourceAsString = searchHit.getSourceAsString();
            System.out.println(sourceAsString);
        }
    }

    @Test
    public void test18() throws Exception {
        //long类型的数据范围查询
        SearchResponse response = transportClient
                .prepareSearch("animals")
                .setTypes("dogs")
                .setQuery(QueryBuilders
                        .rangeQuery("时间")
                        .gte("20191208210207059").lte("20191208210207554"))
                .setFrom(0).setSize(10000).setExplain(true)
                .execute()
```

```
                .actionGet();
        //直接遍历
        for (SearchHit searchHit : response.getHits()) {
            String sourceAsString = searchHit.getSourceAsString();
            System.out.println(sourceAsString);
        }
    }

    @Test
    public void test19() {
        //分组聚合,查询字段是以名字为分组的文档有多少个
        AggregationBuilder aggregationBuilder = AggregationBuilders
                .terms("name")//定义分组聚合的名字
                .field("名字.keyword");//查询以port字段为分组出现的次数
        SearchResponse response = transportClient
                .prepareSearch("animals")//索引
                .addAggregation(aggregationBuilder)
                .execute()
                .actionGet();
        Terms terms = response.getAggregations().get("name");//获取定义分组聚合的名字
        for (Terms.Bucket tb : terms.getBuckets()) {
            System.out.println(tb.getKey() + ":" + tb.getDocCount());
        }
    }

    @Test
    public void test20() {
        //聚合分类统计，强制返回空bucket
        SearchResponse response = transportClient.prepareSearch("animals")
                .setTypes("dogs")
                .addAggregation(
                        AggregationBuilders.dateHistogram("histogram")//定义名字
                                .field("@timestamp")
                                .dateHistogramInterval(DateHistogramInterval.MONTH)//以一个月为间隔
位
                                .format("yyyy-MM-dd")
                                .minDocCount(0)//可以返回没有数据的月份
                                .extendedBounds(
                                        new ExtendedBounds("2019-01-01", "2019-12-31")//强制返回数据的
围
                                )
                )
                .setSize(0)
                .get();
        Histogram histogram = response.getAggregations().get("histogram");//获取定义的名字
        List<? extends Histogram.Bucket> buckets = histogram.getBuckets();
        for (Histogram.Bucket bucket : buckets) {
            System.out.println(bucket.getKeyAsString());
            System.out.println(bucket.getKey());
            System.out.println(bucket.getDocCount());
        }
    }

    @Test
```

```
    public void test21() {
        //过滤聚合,单个条件
        //设置过滤条件，时间字段为20191208210207430的文档有多少个
        QueryBuilder builder = QueryBuilders.termsQuery("时间", "20191208210207430");
        AggregationBuilder aggregationBuilder = AggregationBuilders
                .filter("time", builder);//定义过滤聚合的名字
        SearchResponse response = transportClient
                .prepareSearch("animals")//索引
                .addAggregation(aggregationBuilder)
                .execute()
                .actionGet();
        Filter filter = response.getAggregations().get("time");//获取定义过滤聚合的名字
        System.out.println(filter.getDocCount());
    }

    @Test
    public void test22() throws IOException {
        //过滤聚合,多个条件,名字是傻狗1的文档有多少个，颜色是白色2的文档有多少个
        FiltersAggregationBuilder aggregationBuilder = AggregationBuilders.filters("filters",//定
名字
                new FiltersAggregator.KeyedFilter("名字", QueryBuilders.termQuery("名字.keyword",
傻狗1")),
                new FiltersAggregator.KeyedFilter("颜色", QueryBuilders.termQuery("颜色.keyword",
白色2"))
        );
        SearchResponse response = transportClient
                .prepareSearch("animals")
                .addAggregation(aggregationBuilder)
                .execute()
                .actionGet();
        Filters filters = response.getAggregations().get("filters");//获取定义的名字
        for (Filters.Bucket fb : filters.getBuckets()) {
            System.out.println(fb.getKey() + ":" + fb.getDocCount());
        }
    }

    @Test
    public void test23() {
        //获取 elasticsearch 服务器时间戳
        long timestamp = transportClient.admin().cluster().prepareClusterStats().get().getTimest
mp();
        System.out.println(timestamp);
    }
}
```

四、 常见错误

错误

ERROR StatusLogger Log4j2 could not find a logging implementation. Please add log4j-core to the classpath. Using SimpleLogger to log to the console...

解决

```
<!-- 日志依赖 -->
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.10.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.10.0</version>
        </dependency>
```

log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console"/>
        </Root>
    </Loggers>
</Configuration>
```