目录

[一、 Elasticsearch 2.3.3 JAVA api说明文档 1](#_Toc511760489)

[二、 2  Maven存储库 1](#_Toc511760490)

[三、 3  处理与JAR依赖的冲突问题 2](#_Toc511760491)

[四、 4  使用依赖项嵌入jar 2](#_Toc511760492)

[五、 5  在JBoss EAP6模块中部署 4](#_Toc511760493)

[六、 6  客户端 6](#_Toc511760494)

[（一） 6.1  传输客户端（Transport Client） 6](#_Toc511760495)

[（二） 6.2  将客户端连接到客户端节点 8](#_Toc511760496)

[七、 7  API文档 8](#_Toc511760497)

[（一） 7.1  单个文档操作API 8](#_Toc511760498)

[1、 7.1.1  索引（动词）API 8](#_Toc511760499)

[2、 7.1.2  获取索引API 14](#_Toc511760500)

[3、 7.1.3  删除索引API 15](#_Toc511760501)

[4、 7.1.4  更新索引API 15](#_Toc511760502)

[（二） 7.2  多文档操作API 20](#_Toc511760503)

[1、 7.2.1  多文档获取API 20](#_Toc511760504)

[2、 7.2.2  批量API 21](#_Toc511760505)

[3、 7.2.3  使用批量API线程 23](#_Toc511760506)

[八、 8  查询API 26](#_Toc511760507)

[（一） 8.1  在Java中使用scrolls 27](#_Toc511760508)

[（二） 8.2  多搜索API 28](#_Toc511760509)

[（三） 8.3  使用聚合 30](#_Toc511760510)

[（四） 8.4  终止后 31](#_Toc511760511)

[九、 9  计数API 31](#_Toc511760512)

[（一） 9.1  操作线程 32](#_Toc511760513)

[十、 10  聚合 32](#_Toc511760514)

[（一） 10.1  结构化聚合 33](#_Toc511760515)

[（二） 10.2  聚合指标 34](#_Toc511760516)

[1、 10.2.1  最小聚合(Min Aggregation) 34](#_Toc511760517)

[2、 10.2.2  最大聚合(Max Aggregation) 35](#_Toc511760518)

[3、 10.2.3  聚合求和(Sum Aggregation) 36](#_Toc511760519)

[4、 10.2.4  聚合均值 (Avg Aggregation) 37](#_Toc511760520)

[5、 10.2.5  聚合汇总 (Stats Aggregation) 38](#_Toc511760521)

[6、 10.2.6  扩展统计聚合(Extended Stats Aggregation) 39](#_Toc511760522)

[7、 10.2.7  计数聚合的值(Extended Stats Aggregation) 41](#_Toc511760523)

[8、 10.2.8  百分比聚合(Percentile Aggregation)-小数点后为0 42](#_Toc511760524)

[9、 10.2.9  基数聚合 (Cardinality Aggregation) 44](#_Toc511760525)

[10、 10.2.10  Geo边界聚合 (Geo Bounds Aggregation) 46](#_Toc511760526)

[11、 10.2.11  热门点击汇总 (Top Hits Aggregation) 47](#_Toc511760527)

[12、 10.2.12  脚本度量聚合 (Scripted Metric Aggregation) 50](#_Toc511760528)

[（三） 10.3  聚合容器 55](#_Toc511760529)

[1、 10.3.1  全局聚合(Global Aggregation) 55](#_Toc511760530)

[2、 10.3.2  过滤器聚合(Filter Aggregation) 56](#_Toc511760531)

[3、 10.3.3  多过滤器聚合(Filters Aggregation) 57](#_Toc511760532)

[4、 10.3.4  缺失聚合(Missing Aggregation) 59](#_Toc511760533)

[5、 10.3.5  嵌套聚合(Nested Aggregation) 60](#_Toc511760534)

[6、 10.3.6  反向嵌套聚合(Reverse Nested Aggregation) 61](#_Toc511760535)

[7、 10.3.7  子聚合(Children Aggregation) 63](#_Toc511760536)

[8、 10.3.8  条件聚合(Terms Aggregation) 64](#_Toc511760537)

[9、 10.3.9  排序(Order) 65](#_Toc511760538)

[10、 10.3.10  重要条件聚合(Significant Terms Aggregation) 66](#_Toc511760539)

[11、 10.3.11  范围聚合(Range Aggregation) 68](#_Toc511760540)

[12、 10.3.12  时间范围聚合(Date Range Aggregation) 70](#_Toc511760541)

[13、 10.3.13  P范围聚合(Ip Range Aggregation) 72](#_Toc511760542)

[14、 10.3.14  柱状图聚合(Histogram Aggregation) 75](#_Toc511760543)

[15、 10.3.15  日期柱状图聚合(Date Histogram Aggregation) 77](#_Toc511760544)

[16、 10.3.16  geo距离聚合(Geo Distance Aggregation) 79](#_Toc511760545)

[17、 10.3.17  geo哈希网格聚合(Geo Hash Grid Aggregation) 81](#_Toc511760546)

[十一、 11  渗透过滤API 83](#_Toc511760547)

[十二、 12  查询DSL 85](#_Toc511760548)

[（一） 12.1  匹配所有的查询 86](#_Toc511760549)

[（二） 12.2  全文搜索 86](#_Toc511760550)

[1、 12.2.1  匹配查询 87](#_Toc511760551)

[2、 12.2.2  多匹配查询 87](#_Toc511760552)

[3、 12.2.3  一般条件查询 88](#_Toc511760553)

[4、 12.2.4  查询字符串查询 88](#_Toc511760554)

[5、 12.2.5  简单的查询字符串查询 88](#_Toc511760555)

[（三） 12.3  条件等级查询 89](#_Toc511760556)

[1、 12.3.1  词汇查询 89](#_Toc511760557)

[2、 12.3.2  多词汇查询 90](#_Toc511760558)

[3、 12.3.3  范围查询 90](#_Toc511760559)

[4、 12.3.4  存在查询 91](#_Toc511760560)

[5、 12.3.5  缺失查询 91](#_Toc511760561)

[6、 12.3.6  前缀查询 92](#_Toc511760562)

[7、 12.3.7  通配符查询 93](#_Toc511760563)

[8、 12.3.8  正则表达式查询 93](#_Toc511760564)

[9、 12.3.9  模糊查询 93](#_Toc511760565)

[10、 12.3.10  类型查询 94](#_Toc511760566)

[11、 12.3.11  多ID查询 94](#_Toc511760567)

[（四） 12.4  复合查询 95](#_Toc511760568)

[1、 12.4.1  常数 \_score查询 95](#_Toc511760569)

[2、 12.4.2  布尔查询 96](#_Toc511760570)

[3、 12.4.3  Dis max查询 96](#_Toc511760571)

[4、 12.4.4  函数得分查询 97](#_Toc511760572)

[5、 12.4.5  提升查询 98](#_Toc511760573)

[6、 12.4.6  索引查询 99](#_Toc511760574)

[7、 12.4.7  和查询 100](#_Toc511760575)

[8、 12.4.8  否查询 100](#_Toc511760576)

[9、 12.4.9  或查询 101](#_Toc511760577)

[10、 12.4.10  过滤查询 101](#_Toc511760578)

[11、 12.4.11  限制查询 102](#_Toc511760579)

[（五） 12.5  连接(Join)查询 102](#_Toc511760580)

[1、 12.5.1  嵌套查询 102](#_Toc511760581)

[2、 12.5.2  有子查询 103](#_Toc511760582)

[3、 12.5.3  有父查询 104](#_Toc511760583)

[（六） 12.6  GEO查询 104](#_Toc511760584)

[1、 12.6.1  地理形状查询 105](#_Toc511760585)

[2、 12.6.2  地理边界框查询 107](#_Toc511760586)

[3、 12.6.3  地理距离查询 108](#_Toc511760587)

[4、 12.6.4  地理距离范围查询 109](#_Toc511760588)

[5、 12.6.5  地理多边形查询 110](#_Toc511760589)

[6、 12.6.6  地理散列单元格查询 111](#_Toc511760590)

[（七） 12.7  高级查询 111](#_Toc511760591)

[1、 12.7.1  更多此查询（mlt） 111](#_Toc511760592)

[2、 12.7.2  模板查询 112](#_Toc511760593)

[3、 12.7.3  脚本查询 115](#_Toc511760594)

[（八） 12.8  跨度(Span)查询 116](#_Toc511760595)

[1、 12.8.1  跨度项(span\_term)查询 116](#_Toc511760596)

[2、 12.8.2  跨度多项(span\_multi)查询 117](#_Toc511760597)

[3、 12.8.3  跨度第一(span\_first)查询 117](#_Toc511760598)

[4、 12.8.4  跨度接近查询(span\_near)查询 118](#_Toc511760599)

[5、 12.8.5  跨度或(span\_or)查询 119](#_Toc511760600)

[6、 12.8.6  跨度不(span\_not)查询 119](#_Toc511760601)

[7、 12.8.7  跨度包含(span\_containing)查询 120](#_Toc511760602)

[8、 12.8.8  跨越(span\_within)查询 121](#_Toc511760603)

[十三、 13  索引脚本API 122](#_Toc511760604)

[1、 13.0.1  脚本语言 123](#_Toc511760605)

[十四、 14  Java API管理 123](#_Toc511760606)

[（一） 14.1  指数管理 123](#_Toc511760607)

[1、 14.1.1  创建索引 124](#_Toc511760608)

[2、 14.1.2  设置索引 124](#_Toc511760609)

[3、 14.1.3  进行映射 125](#_Toc511760610)

[4、 14.1.4  刷新 128](#_Toc511760611)

[5、 14.1.5  获取设置 129](#_Toc511760612)

[6、 14.1.6  更新索引的设置信息 130](#_Toc511760613)

[（二） 14.2  集群管理 130](#_Toc511760614)

[1、 14.2.1  集群状况(cluster health) 131](#_Toc511760615)

# Elasticsearch 2.3.3 JAVA api说明文档

原文地址：[https://www.blog-china.cn/template\documentHtml\1484101683485.html](https://www.blog-china.cn/template/documentHtml/1484101683485.html)  
**1  前言**

本节介绍elasticsearch提供的Java API。所有elasticsearch操作**都使用Client对象执行。**所有操作在本质上是**完全异步的（接收到但是未必是马上返回数据）。**另外，**客户端上的操作可以使用批量执行方式。**注意，所有的API通过**Java API公开**（实际上，Java API用于在内部执行它们）。

# 2  Maven存储库

Elasticsearch托管在Maven的仓库中。  
例如，您可以在pom.xml文件中定义最新版本：

<dependency>

<groupId>org.elasticsearch</groupId>

<artifactId>elasticsearch</artifactId>

<version>${es.version}</version>

</dependency>

# 3  处理与JAR依赖的冲突问题

如果要在Java应用程序中使用Elasticsearch，则可能必须处理与第三方依赖项（如Guava和Joda）的版本冲突。例如，Elasticsearch使用Joda 2.8，而你的代码使用Joda 2.1。  
你有两个选择：  
1. 最简单的解决方案是升级。 较新的模块版本可能有固定的老错误。 你落后的越远，就越难升级。 当然，你可能使用第三方依赖，依赖依赖于包的过期版本，这会阻止你升级。  
2. 第二个选项是重定位麻烦的依赖关系，并使用您自己的应用程序或Elasticsearch和Elasticsearch客户端所需的任何插件来隐藏它们。  
“[To shade or not to shade](https://www.elastic.co/blog/to-shade-or-not-to-shad)”博客文章描述了这样做的所有步骤。

# 4  使用依赖项嵌入jar

如果你想创建一个包含你的应用程序和所有依赖项的jar，你不应该使用maven-assembly-plugin，因为它不能处理Lucene jars需要的META-INF / services结构。  
相反，你可以使用maven-shade-plugin并配置如下：

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>2.4.1</version>

<executions>

<execution>

<phase>package</phase>

<goals><goal>shade</goal></goals>

<configuration>

<transformers>

<transformer implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

注意，如果你有一个主类，且想在使用java – jar命令运行你的jar时自动调用你它，只需将下面的内容添加到transformers：

<transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

<mainClass>org.elasticsearch.demo.Generate</mainClass>

</transformer>

# 5  在JBoss EAP6模块中部署

Elasticsearch和Lucene类需要在同一个JBoss模块中。  
你应该这样定义一个module.xml文件：

<?xml version="1.0" encoding="UTF-8"?>

<module name="org.elasticsearch">

<resources>

<!-- Elasticsearch -->

<resource-root path="elasticsearch-2.0.0.jar"/>

<!-- Lucene -->

<resource-root path="lucene-core-5.1.0.jar"/>

<resource-root path="lucene-analyzers-common-5.1.0.jar"/>

<resource-root path="lucene-queries-5.1.0.jar"/>

<resource-root path="lucene-memory-5.1.0.jar"/>

<resource-root path="lucene-highlighter-5.1.0.jar"/>

<resource-root path="lucene-queryparser-5.1.0.jar"/>

<resource-root path="lucene-sandbox-5.1.0.jar"/>

<resource-root path="lucene-suggest-5.1.0.jar"/>

<resource-root path="lucene-misc-5.1.0.jar"/>

<resource-root path="lucene-join-5.1.0.jar"/>

<resource-root path="lucene-grouping-5.1.0.jar"/>

<resource-root path="lucene-spatial-5.1.0.jar"/>

<resource-root path="lucene-expressions-5.1.0.jar"/>

<!-- Insert other resources here -->

</resources>

<dependencies>

<module name="sun.jdk" export="true" >

<imports>

<include path="sun/misc/Unsafe" />

</imports>

</module>

<module name="org.apache.log4j"/>

<module name="org.apache.commons.logging"/>

<module name="javax.api"/>

</dependencies>

</module>

# 6  客户端

您可以通过多种方式使用Java客户端：  
1. 在现有集群上执行**标准索引，获取，删除和搜索**操作  
2. 在正在运行的集群上执行管理任务  
获取elasticsearch客户端很简单。 获取客户端的最常见方法是创建一个连接到集群的TransportClient。  
、  
注意：客户端必须具有与集群中的节点相同的主版本号（例如2.x或5.x）。 客户端可以连接到具有不同小版本（例如2.3.x）的群集，但是可能不支持新的功能。 理想情况下，客户端应具有与群集相同的版本。

## 6.1  传输客户端（Transport Client）

传输客户端（Transport Client）使用传输模块远程连接到Elasticsearch集群。 它不加入集群，而是简单地获得一个或多个初始传输地址，并在每个动作上以循环方式与它们通信（尽管大多数动作可能是“两跳”操作）。

// on startup

Client client = TransportClient.builder().build()

.addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("host1"), 9300))

.addTransportAddress(new InetSocketTransportAddress(InetAddress.getByName("host2"), 9300));

// on shutdown

client.close();

请注意，如果使用与“elasticsearch”不同的名称，则必须设置集群名称：

Settings settings = Settings.settingsBuilder()

.put("cluster.name", "myClusterName").build();

Client client = TransportClient.builder().settings(settings).build();

//Add transport addresses and do something with the client...

传输客户端附带一个群集嗅探功能，允许它动态添加新主机和删除旧主机。当启用嗅探时，传输客户端将连接到其内部节点列表中的节点，这是通过调用addTransportAddress构建的。此后，客户端将调用这些节点上的内部群集状态API以发现可用的数据节点。客户端的内部节点列表将仅由这些数据节点替换。默认情况下，此列表每五秒刷新一次。请注意，嗅探器连接的IP地址是那些在那些节点的elasticsearch配置中声明为发布地址的IP地址。  
请记住，如果该节点不是数据节点，则该列表可能不包括它连接的原始节点。例如，如果您最初连接到主节点，则在嗅探后，不再有请求将去往该主节点，而是到任何数据节点。传输客户端排除非数据节点的原因是为了避免将搜索流量发送到仅主节点。  
为了启用嗅探，请将client.transport.sniff设置为true：

Settings settings = Settings.settingsBuilder()

.put("client.transport.sniff", true).build();

TransportClient client = TransportClient.builder().settings(settings).build();

其他传输客户端级别设置包括：

|  |  |
| --- | --- |
| **参数** | **描述** |
| client.transport.ignore\_cluster\_name | 设置为true可忽略连接的节点的集群名称验证。 （从0.19.4开始） |
| client.transport.ping\_timeout | 等待来自节点的ping响应的时间。 默认为5秒。 |
| client.transport.nodes\_sampler\_interval | 定时对列出和连接的节点进行抽样/ ping操作。 默认为5秒。 |

## 6.2  将客户端连接到客户端节点

您可以在本地启动客户机节点，然后在连接到此客户机节点的应用程序中创建一个Transport Client。  
这样，客户端节点将能够加载您需要的任何插件（例如，发现插件）。

# 7  API文档

本节介绍以下增删改查（CRUD）API：

## 7.1  单个文档操作API

### 7.1.1  索引（动词）API

索引API允许将要键入的JSON文档索引（动词）到特定索引（名词）中，并使其可搜索。

#### 7.1.1.1  生成JSON文档

有几种不同的生成JSON文档的方法：  
1. 手动（自己做）使用本地byte []或作为一个字符串  
2. 将其自动转换为与JSON等效值的Map对象  
3. 使用第三方库来序列化你的bean，如[Jackson](http://wiki.fasterxml.com/JacksonHom)  
4. 使用内置帮助器XContentFactory.jsonBuilder（）  
在内部，每个类型转换为byte []（因此String被转换为byte数组）。 因此，如果对象已经是这种形式，则直接使用它。 jsonBuilder是高度优化的JSON生成器，直接构造一个byte []。

#### 7.1.1.2  自己做

这里没有什么很难，但请注意，您将必须根据[日期格式](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/mapping-date-format.htm)编码日期。

String json = "{" +

"\"user\":\"kimchy\"," +

"\"postDate\":\"2013-01-30\"," +

"\"message\":\"trying out Elasticsearch\"" +

"}";

#### 7.1.1.3  使用Map对象

Map是一个键值对：值对集合。 它代表一个JSON结构：

Map<String, Object> json = new HashMap<String, Object>();

json.put("user","kimchy");

json.put("postDate",new Date());

json.put("message","trying out Elasticsearch");

#### 7.1.1.4  序列化你的对象

Elasticsearch已经使用Jackson。 所以你可以使用它来将你的bean序列化为JSON：

import com.fasterxml.jackson.databind.\*;

// instance a json mapper

ObjectMapper mapper = new ObjectMapper(); // create once, reuse

// generate json

byte[] json = mapper.writeValueAsBytes(yourbeaninstance);

#### 7.1.1.5  使用elasticsearch 帮助类

Elasticsearch提供内置帮助器来生成JSON内容。

import static org.elasticsearch.common.xcontent.XContentFactory.\*;

XContentBuilder builder = jsonBuilder()

.startObject()

.field("user", "kimchy")

.field("postDate", new Date())

.field("message", "trying out Elasticsearch")

.endObject()

注意，你也可以使用startArray（String）和endArray（）方法添加数组。 顺便说一句，field方法接受许多对象类型。 您可以直接传递数字，日期甚至其他XContentBuilder对象。  
如果您需要查看生成的JSON内容，可以使用string()方法。

String json = builder.string();

#### 7.1.1.6  索引文档

以下示例将会把JSON文档索引成索引名为twitter，类型名为tweet，id为1的一个索引：

import static org.elasticsearch.common.xcontent.XContentFactory.\*;

IndexResponse response = client.prepareIndex("twitter", "tweet", "1")

.setSource(jsonBuilder()

.startObject()

.field("user", "kimchy")

.field("postDate", new Date())

.field("message", "trying out Elasticsearch")

.endObject()

)

.get();

请注意，您还可以将文档索引为JSON字符串，并且您不必提供ID：

String json = "{" +

"\"user\":\"kimchy\"," +

"\"postDate\":\"2013-01-30\"," +

"\"message\":\"trying out Elasticsearch\"" +

"}";

IndexResponse response = client.prepareIndex("twitter", "tweet")

.setSource(json)

.get();

IndexResponse对象会给你一个返回报告：

// Index name

String \_index = response.getIndex();

// Type name

String \_type = response.getType();

// Document ID (generated or not)

String \_id = response.getId();

// Version (if it's the first time you index this document, you will get: 1)

long \_version = response.getVersion();

// isCreated() is true if the document is a new one, false if it has been updated

boolean created = response.isCreated();

有关索引操作的更多信息，请查看REST[索引](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/docs-index_.html)) docs。

#### 7.1.1.7  操作线程

索引API允许设置线程模型，当在相同节点上执行API的实际执行时执行操作（该API在分配在同一服务器上的分片上执行）的时候，这个操作将会被执行。  
选项是在不同的线程上执行操作，或者在调用线程上执行它（注意，API仍然是异步的）。 默认情况下，operationThreaded设置为true，这意味着操作在不同的线程上执行。

### 7.1.2  获取索引API

获取索引API允许从索引根据其id获取一个类型化的JSON文档。 下面的示例从名为Twitter、类型名为tweet的索引中得到一个ID的值为1的JSON文件：

GetResponse response = client.prepareGet("twitter", "tweet", "1").get();

有关get操作的更多信息，请查看REST[get](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/docs-get.htm)文档。

#### 7.1.2.1  操作线程

获取索引API允许设置一个线程模型，当在相同节点上执行API的实际执行时执行操作（该API在分配在同一服务器上的分片上执行）的时候，这个操作将会被执行。  
选项是在不同的线程上执行操作，或者在调用线程上执行它（注意，API仍然是异步的）。 默认情况下，operationThreaded设置为true，这意味着在不同的线程上执行操作。 下面是一个将其设置为false的示例：

GetResponse response = client.prepareGet("twitter", "tweet", "1")

.setOperationThreaded(false)

.get();

### 7.1.3  删除索引API

删除索引API允许根据其id从特定索引中删除键入的JSON文档。 以下示例从名为twitter、类型为tweet）的索引下删除id为1的JSON文档：

DeleteResponse response = client.prepareDelete("twitter", "tweet", "1").get();

有关删除操作的详细信息，请查看[delete API](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/docs-delete.htm)文档。

#### 7.1.3.1  操作线程

删除索引API允许设置一个线程模型，当在相同节点上执行API的实际执行时执行操作（该API在分配在同一服务器上的分片上执行）的时候，这个操作将会被执行。  
选项是在不同的线程上执行操作，或者在调用线程上执行它（注意，API仍然是异步的）。 默认情况下，operationThreaded设置为true，这意味着操作在不同的线程上执行。 下面是一个将其设置为false的示例：

DeleteResponse response = client.prepareDelete("twitter", "tweet", "1")

.setOperationThreaded(false)

.get();

### 7.1.4  更新索引API

您可以创建UpdateRequest并将其发送到客户端：

UpdateRequest updateRequest = new UpdateRequest();

updateRequest.index("index");

updateRequest.type("type");

updateRequest.id("1");

updateRequest.doc(jsonBuilder()

.startObject()

.field("gender", "male")

.endObject());

client.update(updateRequest).get();

或者你可以使用prepareUpdate()方法：

client.prepareUpdate("ttl", "doc", "1")

.setScript(new Script("ctx.\_source.gender = \"male\"" [注释1] , ScriptService.ScriptType.INLINE, null, null))

.get();

client.prepareUpdate("ttl", "doc", "1")

.setDoc(jsonBuilder()[注释2]

.startObject()

.field("gender", "male")

.endObject())

.get();

[注释1]. 你的脚本。它也可以是本地存储的脚本名称。在这种情况下，您需要使用ScriptService.ScriptType.FILE  
[注释2]. 将合并到现有文档的文档。  
请注意，您不能同时使用脚本和doc更新。

#### 7.1.4.1  根据脚本更新

更新API允许根据提供的脚本更新文档：

UpdateRequest updateRequest = new UpdateRequest("ttl", "doc", "1")

.script(new Script("ctx.\_source.gender = \"male\""));

client.update(updateRequest).get();

#### 7.1.4.2  通过合并文档进行更新

更新API还支持传递部分文档，这将被合并到现有文档中（简单递归合并，内部合并对象，替换核心“键/值”和数组）。 例如：

UpdateRequest updateRequest = new UpdateRequest("index", "type", "1")

.doc(jsonBuilder()

.startObject()

.field("gender", "male")

.endObject());

client.update(updateRequest).get();

#### 7.1.4.3   Upsert

还支持upsert。 如果文档不存在，则upsert元素的内容将用于对新文档进行索引：

IndexRequest indexRequest = new IndexRequest("index", "type", "1")

.source(jsonBuilder()

.startObject()

.field("name", "Joe Smith")

.field("gender", "male")

.endObject());

UpdateRequest updateRequest = new UpdateRequest("index", "type", "1")

.doc(jsonBuilder()

.startObject()

.field("gender", "male")

.endObject())

.upsert(indexRequest); [注释1]

client.update(updateRequest).get();

[注释1]. 如果文档不存在，则会添加indexRequest中的一个   
如果索引为index/type/ 1的文档已经存在，那么在这个操作之后我们将有一个这样的文档：

{

"name" : "Joe Dalton",

"gender": "male" [注释1]

}

[注释1]. 此字段在执行更新请求的时候被添加  
如果它不存在，我们将会添加一个新的文档

{

"name" : "Joe Smith",

"gender": "male"

}

## 7.2  多文档操作API

### 7.2.1  多文档获取API

多文档获取API允许基于它们的索引（index），类型（type）和id来获得多个文档的列表：

MultiGetResponse multiGetItemResponses = client.prepareMultiGet()

.add("twitter", "tweet", "1") [注释1]

.add("twitter", "tweet", "2", "3", "4") [注释2]

.add("another", "type", "foo") [注释3]

.get();

for (MultiGetItemResponse itemResponse : multiGetItemResponses) { [注释4]

GetResponse response = itemResponse.getResponse();

if (response.isExists()) { [注释5]

String json = response.getSourceAsString();[注释6]

}

}

[注释1]. 通过单个ID获取  
[注释2]. 或由相同索引/类型(index/type)中根据id获取文档列表  
[注释3]. 你也可以从另一个索引获取文档  
[注释4]. 遍历结果集  
[注释5]. 您可以检查文档是否存在  
[注释6]. 访问\_source字段  
有关多获取操作的更多信息，请查阅REST[多文档获取](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/docs-multi-get.html))文档。

### 7.2.2  批量API

批量API允许在单个请求中索引和删除多个文档。 下面是一个用法示例：

import static org.elasticsearch.common.xcontent.XContentFactory.\*;

BulkRequestBuilder bulkRequest = client.prepareBulk();

// either use client#prepare, or use Requests# to directly build index/delete requests

bulkRequest.add(client.prepareIndex("twitter", "tweet", "1")

.setSource(jsonBuilder()

.startObject()

.field("user", "kimchy")

.field("postDate", new Date())

.field("message", "trying out Elasticsearch")

.endObject()

)

);

bulkRequest.add(client.prepareIndex("twitter", "tweet", "2")

.setSource(jsonBuilder()

.startObject()

.field("user", "kimchy")

.field("postDate", new Date())

.field("message", "another post")

.endObject()

)

);

BulkResponse bulkResponse = bulkRequest.get();

if (bulkResponse.hasFailures()) {

// process failures by iterating through each bulk response item

}

### 7.2.3  使用批量API线程

BulkProcessor类提供了一个简单的接口，可根据请求的数量或大小，或在给定时间段后自动刷新批量操作。  
要使用它，首先创建一个BulkProcessor实例：

import org.elasticsearch.action.bulk.BackoffPolicy;

import org.elasticsearch.action.bulk.BulkProcessor;

import org.elasticsearch.common.unit.ByteSizeUnit;

import org.elasticsearch.common.unit.ByteSizeValue;

import org.elasticsearch.common.unit.TimeValue;

BulkProcessor bulkProcessor = BulkProcessor.builder(

client, [注释1]

new BulkProcessor.Listener() {

@Override

public void beforeBulk(long executionId,

BulkRequest request) { ... } [注释2]

@Override

public void afterBulk(long executionId,

BulkRequest request,

BulkResponse response) { ... } [注释3]

@Override

public void afterBulk(long executionId,

BulkRequest request,

Throwable failure) { ... } [注释4]

})

.setBulkActions(10000) [注释5]

.setBulkSize(new ByteSizeValue(1, ByteSizeUnit.GB)) [注释6]

.setFlushInterval(TimeValue.timeValueSeconds(5)) [注释7]

.setConcurrentRequests(1) [注释8]

.setBackoffPolicy(

BackoffPolicy.exponentialBackoff(TimeValue.timeValueMillis(100), 3)) [注释9]

.build();

[注释1]. 添加您的elasticsearch客户端  
[注释2]. 此方法在批量执行之前调用。例如，您可以使用request.numberOfActions()查看numberOfActions，  
[注释3]. 此方法在批量执行后调用。例如，您可以使用response.hasFailures()检查是否存在一些失败的请求，  
[注释4]. 此方法在批量失败并调用Throwable时调用  
[注释5]. 我们希望每10 000次请求执行批量  
[注释6]. 我们想要每1gb批量刷新一次  
[注释7]. 无论请求数量多少，我们都希望每5秒刷新一次  
[注释8]. 设置并发请求数。值为0表示只允许执行单个请求。值为1表示在累积新的批量请求时允许执行1个并发请求。  
[注释9]. 设置自定义退避策略，该策略最初将等待100毫秒，按指数增加并重试最多三次。每当一个或多个批量项目请求失败并且EsRejectedExecutionException指示可用于处理请求的计算资源太少时，尝试重试。要禁用后退，请传递BackoffPolicy.noBackoff()。   
然后，您可以简单地将您的请求添加到BulkProcessor：

bulkProcessor.add(new IndexRequest("twitter", "tweet", "1").source(/\* your doc here \*/));

bulkProcessor.add(new DeleteRequest("twitter", "tweet", "2"));

默认情况下的BulkProcessor设置为：  
1. 将bulkActions设置为1000  
2. 将bulkSize设置为5mb  
3. 不设置flushInterval  
4. 将concurrentRequests设置为1  
5. 将backoffPolicy设置为具有8次重试和50ms的开始延迟的指数退避。 总等待时间大约为5.1秒。  
当所有文档都加载到BulkProcessor时，可以使用awaitClose或close方法关闭它：

bulkProcessor.awaitClose(10, TimeUnit.MINUTES);

或者使用下面的方法关闭：  
bulkProcessor.close();  
[code end]  
两种方法都能刷新任何剩余的文档，并禁用所有其他刷新计划（如果通过设置flushInterval进行调度）。 如果启用了并发请求，则awaitClose方法会等待所有要完成的批量请求的指定超时，然后返回true，如果在所有批量请求完成之前经过了指定的等待时间，则返回false。 close方法不会等待任何剩余的批量请求完成，并立即退出。

# 8  查询API

搜索API允许执行搜索查询并获得匹配查询的搜索匹配。 它可以跨一个或多个索引(index)并跨越一个或多个类型(type)执行。 查询可以使用[查询Java API](https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.3/java-query-dsl.htm)提供。 搜索请求的正文使用SearchSourceBuilder构建。 下面是一个例子：

import org.elasticsearch.action.search.SearchResponse;

import org.elasticsearch.action.search.SearchType;

import org.elasticsearch.index.query.QueryBuilders.\*;

SearchResponse response = client.prepareSearch("index1", "index2")

.setTypes("type1", "type2")

.setSearchType(SearchType.DFS\_QUERY\_THEN\_FETCH)

.setQuery(QueryBuilders.termQuery("multi", "test")) // Query

.setPostFilter(QueryBuilders.rangeQuery("age").from(12).to(18)) // Filter

.setFrom(0).setSize(60).setExplain(true)

.execute()

.actionGet();

请注意，所有参数都是可选的。 这里是你调用搜索时的最简略写法：

// MatchAll on the whole cluster with all default options

SearchResponse response = client.prepareSearch().execute().actionGet();

注意：虽然Java API定义了其他搜索类型QUERY\_AND\_FETCH和DFS\_QUERY\_AND\_FETCH，但这些模式是内部优化，并且不应由API的用户显式指定。  
有关搜索操作的更多信息，请查看[REST搜索](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search.htm)文档。

## 8.1  在Java中使用scrolls

首先阅读[scrolls](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-request-scroll.htm)文档！

import static org.elasticsearch.index.query.QueryBuilders.\*;

QueryBuilder qb = termQuery("multi", "test");

SearchResponse scrollResp = client.prepareSearch(test)

.addSort(SortParseElement.DOC\_FIELD\_NAME, SortOrder.ASC)

.setScroll(new TimeValue(60000))

.setQuery(qb)

.setSize(100).execute().actionGet(); //100 hits per shard will be returned for each scroll

//Scroll until no hits are returned

while (true) {

for (SearchHit hit : scrollResp.getHits().getHits()) {

//Handle the hit...

}

scrollResp = client.prepareSearchScroll(scrollResp.getScrollId()).setScroll(new TimeValue(60000)).execute().actionGet();

//Break condition: No hits are returned

if (scrollResp.getHits().getHits().length == 0) {

break;

}

}

注意：size参数为集群中的每个分片，所以如果你对多个索引（导致查询中涉及到很多分片）运行查询，结果可能是每次执行的scrolls比你所期望的更多的文档！

## 8.2  多搜索API

查看[多搜索查询API](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-multi-search.html))文档. 

SearchRequestBuilder srb1 = client

.prepareSearch().setQuery(QueryBuilders.queryStringQuery("elasticsearch")).setSize(1);

SearchRequestBuilder srb2 = client

.prepareSearch().setQuery(QueryBuilders.matchQuery("name", "kimchy")).setSize(1);

MultiSearchResponse sr = client.prepareMultiSearch()

.add(srb1)

.add(srb2)

.execute().actionGet();

// You will get all individual responses from MultiSearchResponse#getResponses()

long nbHits = 0;

for (MultiSearchResponse.Item item : sr.getResponses()) {

SearchResponse response = item.getResponse();

nbHits += response.getHits().getTotalHits();

}

## 8.3  使用聚合

下面的代码显示如何在搜索中添加两个聚合：

SearchResponse sr = client.prepareSearch()

.setQuery(QueryBuilders.matchAllQuery())

.addAggregation(

AggregationBuilders.terms("agg1").field("field")

)

.addAggregation(

AggregationBuilders.dateHistogram("agg2")

.field("birth")

.interval(DateHistogramInterval.YEAR)

)

.execute().actionGet();

// Get your facet results

Terms agg1 = sr.getAggregations().get("agg1");

DateHistogram agg2 = sr.getAggregations().get("agg2");

有关详细信息，请参阅[Aggregations Java API](https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.3/java-aggs.htm)文档。

## 8.4  终止后

每个分片收集的文档的到达最大数量时，查询执行将提前终止。 如果设置，您将能够通过在SearchResponse对象中请求isTerminatedEarly()来检查操作是否提前终止：

SearchResponse sr = client.prepareSearch(INDEX)

.setTerminateAfter(1000) [注释1]

.get();

if (sr.isTerminatedEarly()) {

// We finished early

}

[注释1]. 在1000个文档后结束

# 9  计数API

警告：在2.1.0中已弃用。请改用search api，并将大小设置为0。  
计数API允许人们容易地执行查询并获得该查询的匹配数。 它可以跨一个或多个索引(index)并跨越一个或多个类型(type)执行。 可以使用[查询DSL](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl.htm)提供查询。

import static org.elasticsearch.index.query.QueryBuilders.\*;

CountResponse response = client.prepareCount("test")

.setQuery(termQuery("\_type", "type1"))

.execute()

.actionGet();

有关计数操作的更多信息，请查看REST[计数](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-count.html))文档。

## 9.1  操作线程

计数API允许设置线程模型，当在相同节点上执行API的实际执行时执行操作（该API在分配在同一服务器上的分片上执行）的时候，这个操作将会被执行。  
有三种线程模式。NO\_THREADS模式意味着将在调用线程上执行计数操作。 SINGLE\_THREAD模式意味着计数操作将在所有本地分片的单个不同线程上执行。 THREAD\_PER\_SHARD模式意味着计数操作将在每个本地分片的不同线程上执行。  
默认模式为SINGLE\_THREAD。

# 10  聚合

Elasticsearch提供了一个完整的Java API来使用聚合。 请参阅[聚合指南](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations.htm)。  
使用工厂方法聚合构建器（AggregationBuilders），并在查询并将其添加到搜索请求时添加要计算的每个聚合：

SearchResponse sr = node.client().prepareSearch()

.setQuery( /\* your query \*/ )

.addAggregation( /\* add an aggregation \*/ )

.execute().actionGet();

请注意，您可以添加多个聚合。有关详细信息，请参阅[搜索Java API](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-search.html))。  
要构建聚合请求，请使用AggregationBuilders帮助程序。 只需将它们导入您的类：

import org.elasticsearch.search.aggregations.AggregationBuilders;

## 10.1  结构化聚合

如“[聚合Java API](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations.htm)”指南中所述，您可以在聚合中定义子聚合。  
聚合可以是度量聚合或桶聚合。  
例如，下面是一个3级聚合组成的：  
1. 字词汇总（值区）  
2. 日期直方图聚合（值区）  
3. 平均汇总（指标）

SearchResponse sr = node.client().prepareSearch()

.addAggregation(

AggregationBuilders.terms("by\_country").field("country")

.subAggregation(AggregationBuilders.dateHistogram("by\_year")

.field("dateOfBirth")

.interval((DateHistogramInterval.YEAR)

.subAggregation(AggregationBuilders.avg("avg\_children").field("children"))

)

)

.execute().actionGet();

## 10.2  聚合指标

### 10.2.1  最小聚合(Min Aggregation)

下面是如何运用Java API使用[最小聚合](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-min-aggregation.htm) (Min Aggregation)。

#### 10.2.1.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.min("agg")

.field("height");

#### 10.2.1.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.min.Min;

// sr is here your SearchResponse object

Min agg = sr.getAggregations().get("agg");

double value = agg.getValue();

### 10.2.2  最大聚合(Max Aggregation)

下面是如何运用Java API使用[最大聚合(Max Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-max-aggregation.htm)。

#### 10.2.2.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.max("agg")

.field("height");

#### 10.2.2.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.max.Max;

// sr is here your SearchResponse object

Max agg = sr.getAggregations().get("agg");

double value = agg.getValue();

### 10.2.3  聚合求和(Sum Aggregation)

下面是如何运用Java API使用[聚合求和](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-sum-aggregation.htm) (Sum Aggregation)。

#### 10.2.3.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.sum("agg")

.field("height");

#### 10.2.3.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.sum.Sum;

// sr is here your SearchResponse object

Sum agg = sr.getAggregations().get("agg");

double value = agg.getValue();

### 10.2.4  聚合均值 (Avg Aggregation)

下面是如何运用Java API使用[聚合均值](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-avg-aggregation.html)) (Avg Aggregation)。

#### 10.2.4.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.avg("agg")

.field("height");

#### 10.2.4.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.avg.Avg;

// sr is here your SearchResponse object

Avg agg = sr.getAggregations().get("agg");

double value = agg.getValue();

### 10.2.5  聚合汇总 (Stats Aggregation)

下面是如何运用Java API使用[聚合汇总](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-stats-aggregation.html)) (Stats Aggregation)。

#### 10.2.5.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.stats("agg")

.field("height");

#### 10.2.5.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.stats.Stats;

// sr is here your SearchResponse object

Stats agg = sr.getAggregations().get("agg");

double min = agg.getMin();

double max = agg.getMax();

double avg = agg.getAvg();

double sum = agg.getSum();

long count = agg.getCount();

### 10.2.6  扩展统计聚合(Extended Stats Aggregation)

下面是如何运用Java API使用[扩展统计聚合(Extended Stats Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-extendedstats-aggregation.htm)。

#### 10.2.6.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.extendedStats("agg")

.field("height");

#### 10.2.6.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.stats.extended.ExtendedStats;

// sr is here your SearchResponse object

ExtendedStats agg = sr.getAggregations().get("agg");

double min = agg.getMin();

double max = agg.getMax();

double avg = agg.getAvg();

double sum = agg.getSum();

long count = agg.getCount();

double stdDeviation = agg.getStdDeviation();

double sumOfSquares = agg.getSumOfSquares();

double variance = agg.getVariance();

### 10.2.7  计数聚合的值(Extended Stats Aggregation)

下面是如何运用Java API使用[计数聚合的值(Value Count Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-valuecount-aggregation.htm)。

#### 10.2.7.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.count("agg")

.field("height");

#### 10.2.7.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.valuecount.ValueCount;

// sr is here your SearchResponse object

ValueCount agg = sr.getAggregations().get("agg");

long value = agg.getValue();

### 10.2.8  百分比聚合(Percentile Aggregation)-小数点后为0

下面是如何运用Java API使用[百分比聚合(Percentile Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-percentile-aggregation.htm)。

#### 10.2.8.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.percentiles("agg")

.field("height");

您可以提供自己的百分比，而不是使用默认值：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.percentiles("agg")

.field("height")

.percentiles(1.0, 5.0, 10.0, 20.0, 30.0, 75.0, 95.0, 99.0);

#### 10.2.8.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.percentiles.Percentile;

import org.elasticsearch.search.aggregations.metrics.percentiles.Percentiles;

// sr is here your SearchResponse object

Percentiles agg = sr.getAggregations().get("agg");

// For each entry

for (Percentile entry : agg) {

double percent = entry.getPercent(); // Percent

double value = entry.getValue(); // Value

logger.info("percent [{}], value [{}]", percent, value);

}

基本上会产生下述结果：

percent [1.0], value [0.814338896154595]

percent [5.0], value [0.8761912455821302]

percent [25.0], value [1.173346540141847]

percent [50.0], value [1.5432023318692198]

percent [75.0], value [1.923915462033674]

percent [95.0], value [2.2273644908535335]

percent [99.0], value [2.284989339108279]

### 10.2.9  基数聚合 (Cardinality Aggregation)

下面是如何运用Java API使用[基数聚合(Cardinality Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-cardinality-aggregation.htm)。

#### 10.2.9.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.cardinality("agg")

.field("tags");

#### 10.2.9.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.cardinality.Cardinality;

// sr is here your SearchResponse object

Cardinality agg = sr.getAggregations().get("agg");

long value = agg.getValue();

### 10.2.10  Geo边界聚合 (Geo Bounds Aggregation)

下面是如何运用Java API使用[Geo边界聚合(Geo Bounds Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-geobounds-aggregation.htm)。

#### 10.2.10.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

GeoBoundsBuilder aggregation =

AggregationBuilders

.geoBounds("agg")

.field("address.location")

.wrapLongitude(true);

#### 10.2.10.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.metrics.geobounds.GeoBounds;

// sr is here your SearchResponse object

GeoBounds agg = sr.getAggregations().get("agg");

GeoPoint bottomRight = agg.bottomRight();

GeoPoint topLeft = agg.topLeft();

logger.info("bottomRight {}, topLeft {}", bottomRight, topLeft);

基本上会产生下述结果：  
bottomRight [40.70500764381921, 13.952946866893775], topLeft [53.49603022435221, -4.190029308156676]  
[code end]

### 10.2.11  热门点击汇总 (Top Hits Aggregation)

下面是如何运用Java API使用[热门点击汇总(Top Hits Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-top-hits-aggregation.htm)。

#### 10.2.11.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.terms("agg").field("gender")

.subAggregation(

AggregationBuilders.topHits("top")

);

您可以使用大多数可用于标准搜索的选项，如from，size，sort，highlight，explain 等等

AggregationBuilder aggregation =

AggregationBuilders

.terms("agg").field("gender")

.subAggregation(

AggregationBuilders.topHits("top")

.setExplain(true)

.setSize(1)

.setFrom(10)

);

#### 10.2.11.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.terms.Terms;

import org.elasticsearch.search.aggregations.metrics.tophits.TopHits;

// sr is here your SearchResponse object

Terms agg = sr.getAggregations().get("agg");

// For each entry

for (Terms.Bucket entry : agg.getBuckets()) {

String key = entry.getKey(); // bucket key

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], doc\_count [{}]", key, docCount);

// We ask for top\_hits for each bucket

TopHits topHits = entry.getAggregations().get("top");

for (SearchHit hit : topHits.getHits().getHits()) {

logger.info(" -> id [{}], \_source [{}]", hit.getId(), hit.getSourceAsString());

}

}

基本上会产生下述结果：

key [male], doc\_count [5107]

-> id [AUnzSZze9k7PKXtq04x2], \_source [{"gender":"male",...}]

-> id [AUnzSZzj9k7PKXtq04x4], \_source [{"gender":"male",...}]

-> id [AUnzSZzl9k7PKXtq04x5], \_source [{"gender":"male",...}]

key [female], doc\_count [4893]

-> id [AUnzSZzM9k7PKXtq04xy], \_source [{"gender":"female",...}]

-> id [AUnzSZzp9k7PKXtq04x8], \_source [{"gender":"female",...}]

-> id [AUnzSZ0W9k7PKXtq04yS], \_source [{"gender":"female",...}]

### 10.2.12  脚本度量聚合 (Scripted Metric Aggregation)

下面是如何运用Java API使用[脚本度量聚合(Scripted Metric Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-metrics-scripted-metric-aggregation.htm)。  
如果要在嵌入式数据节点中运行Groovy脚本（例如，对于单元测试），请不要忘记在类路径中添加Groovy。 例如，使用Maven，将此依赖项添加到pom.xml文件中：

<dependency>

<groupId>org.codehaus.groovy</groupId>

<artifactId>groovy-all</artifactId>

<version>2.3.2</version>

<classifier>indy</classifier>

</dependency>

#### 10.2.12.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.scriptedMetric("agg")

.initScript("\_agg['heights'] = []")

.mapScript(new Script("if (doc['gender'].value == \"male\") " +

"{ \_agg.heights.add(doc['height'].value) } " +

"else " +

"{ \_agg.heights.add(-1 \* doc['height'].value) }"));

您还可以指定将在每个分片上执行的组合脚本：

MetricsAggregationBuilder aggregation =

AggregationBuilders

.scriptedMetric("agg")

.initScript(new Script("\_agg['heights'] = []"))

.mapScript(new Script("if (doc['gender'].value == \"male\") " +

"{ \_agg.heights.add(doc['height'].value) } " +

"else " +

"{ \_agg.heights.add(-1 \* doc['height'].value) }"))

.combineScript(new Script("heights\_sum = 0; for (t in \_agg.heights) { heights\_sum += t }; return heights\_sum"));

您还可以指定将在获取请求的节点上执行的reduce脚本： 

MetricsAggregationBuilder aggregation =

AggregationBuilders

.scriptedMetric("agg")

.initScript(new Script("\_agg['heights'] = []"))

.mapScript(new Script("if (doc['gender'].value == \"male\") " +

"{ \_agg.heights.add(doc['height'].value) } " +

"else " +

"{ \_agg.heights.add(-1 \* doc['height'].value) }"))

.combineScript(new Script("heights\_sum = 0; for (t in \_agg.heights) { heights\_sum += t }; return heights\_sum"))

.reduceScript(new Script("heights\_sum = 0; for (a in \_aggs) { heights\_sum += a }; return heights\_sum"));

#### 10.2.12.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.terms.Terms;

import org.elasticsearch.search.aggregations.metrics.tophits.TopHits;

// sr is here your SearchResponse object

ScriptedMetric agg = sr.getAggregations().get("agg");

Object scriptedResult = agg.aggregation();

logger.info("scriptedResult [{}]", scriptedResult);

请注意，结果取决于您构建的脚本。 对于第一个例子，这将基本上产生：

scriptedResult object [ArrayList]

scriptedResult [ {

"heights" : [ 1.122218480146643, -1.8148918111233887, -1.7626731575142909, ... ]

}, {

"heights" : [ -0.8046067304119863, -2.0785486707864553, -1.9183567430207953, ... ]

}, {

"heights" : [ 2.092635728868694, 1.5697545960886536, 1.8826954461968808, ... ]

}, {

"heights" : [ -2.1863201099468403, 1.6328549117346856, -1.7078288405893842, ... ]

}, {

"heights" : [ 1.6043904836424177, -2.0736538674414025, 0.9898266674373053, ... ]

} ]

第二个例子将产生下述结果：

k scriptedResult object [ArrayList]

scriptedResult [-41.279615707402876,

-60.88007362339038,

38.823270659734256,

14.840192739445632,

11.300902755741326]

最后一个例子将产生下述结果：

scriptedResult object [Double]

scriptedResult [2.171917696507009]

## 10.3  聚合容器

### 10.3.1  全局聚合(Global Aggregation)

下面是如何运用Java API使用[全局聚合(Global Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-global-aggregation.htm)。

#### 10.3.1.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilders

.global("agg")

.subAggregation(AggregationBuilders.terms("genders").field("gender"));

#### 10.3.1.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.global.Global;

// sr is here your SearchResponse object

Global agg = sr.getAggregations().get("agg");

agg.getDocCount(); // Doc count

### 10.3.2  过滤器聚合(Filter Aggregation)

下面是如何运用Java API使用[过滤器聚合(Filter Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-filter-aggregation.htm)。

#### 10.3.2.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilders

.filter("agg")

.filter(QueryBuilders.termQuery("gender", "male"));

#### 10.3.2.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.filter.Filter;

// sr is here your SearchResponse object

Filter agg = sr.getAggregations().get("agg");

agg.getDocCount(); // Doc count

### 10.3.3  多过滤器聚合(Filters Aggregation)

下面是如何运用Java API使用[多过滤器聚合(Filters Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-filters-aggregation.htm)。

#### 10.3.3.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.filters("agg")

.filter("men", QueryBuilders.termQuery("gender", "male"))

.filter("women", QueryBuilders.termQuery("gender", "female"));

#### 10.3.3.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.filters.Filters;

// sr is here your SearchResponse object

Filters agg = sr.getAggregations().get("agg");

// For each entry

for (Filters.Bucket entry : agg.getBuckets()) {

String key = entry.getKeyAsString(); // bucket key

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], doc\_count [{}]", key, docCount);

}

这将会产生如下结果：

key [men], doc\_count [4982]

key [women], doc\_count [5018]

### 10.3.4  缺失聚合(Missing Aggregation)

下面是如何运用Java API使用[缺失聚合(Missing Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-missing-aggregation.htm)。

#### 10.3.4.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilders.missing("agg").field("gender");

#### 10.3.4.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.missing.Missing;

// sr is here your SearchResponse object

Missing agg = sr.getAggregations().get("agg");

agg.getDocCount(); // Doc count

### 10.3.5  嵌套聚合(Nested Aggregation)

下面是如何运用Java API使用[嵌套聚合(Nested Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-nested-aggregation.htm)。

#### 10.3.5.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilders

.nested("agg")

.path("resellers");

#### 10.3.5.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.nested.Nested;

// sr is here your SearchResponse object

Nested agg = sr.getAggregations().get("agg");

agg.getDocCount(); // Doc count

### 10.3.6  反向嵌套聚合(Reverse Nested Aggregation)

下面是如何运用Java API使用[反向嵌套聚合(Reverse Nested Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-reverse-nested-aggregation.htm)。

#### 10.3.6.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.nested("agg").path("resellers")

.subAggregation(

AggregationBuilders

.terms("name").field("resellers.name")

.subAggregation(

AggregationBuilders

.reverseNested("reseller\_to\_product")

)

);

#### 10.3.6.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.nested.Nested;

import org.elasticsearch.search.aggregations.bucket.nested.ReverseNested;

import org.elasticsearch.search.aggregations.bucket.terms.Terms;

// sr is here your SearchResponse object

Nested agg = sr.getAggregations().get("agg");

Terms name = agg.getAggregations().get("name");

for (Terms.Bucket bucket : name.getBuckets()) {

ReverseNested resellerToProduct = bucket.getAggregations().get("reseller\_to\_product");

resellerToProduct.getDocCount(); // Doc count

}

### 10.3.7  子聚合(Children Aggregation)

下面是如何运用Java API使用[子聚合(Children Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-children-aggregation.htm)。

#### 10.3.7.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.children("agg")

.childType("reseller");

#### 10.3.7.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.children.Children;

// sr is here your SearchResponse object

Children agg = sr.getAggregations().get("agg");

agg.getDocCount(); // Doc count

### 10.3.8  条件聚合(Terms Aggregation)

下面是如何运用Java API使用[条件聚合(Terms Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-terms-aggregation.htm)。

#### 10.3.8.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilders

.terms("genders")

.field("gender");

#### 10.3.8.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.terms.Terms;

// sr is here your SearchResponse object

Terms genders = sr.getAggregations().get("genders");

// For each entry

for (Terms.Bucket entry : genders.getBuckets()) {

entry.getKey(); // Term

entry.getDocCount(); // Doc count

}

### 10.3.9  排序(Order)

按照其doc\_count对buckets进行升序排序：

AggregationBuilders

.terms("genders")

.field("gender")

.order(Terms.Order.count(true))

)

);

按照其条件对buckets进行升序排序：

AggregationBuilders

.terms("genders")

.field("gender")

.order(Terms.Order.term(true))

通过单值度量值和子聚合（由聚合名称标识）对buckets进行排序：

AggregationBuilders

.terms("genders")

.field("gender")

.order(Terms.Order.aggregation("avg\_height", false))

.subAggregation(

AggregationBuilders.avg("avg\_height").field("height")

)

### 10.3.10  重要条件聚合(Significant Terms Aggregation)

下面是如何运用Java API使用[重要条件聚合(Significant Terms Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-significantterms-aggregation.htm)。

#### 10.3.10.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.significantTerms("significant\_countries")

.field("address.country");

// Let say you search for men only

SearchResponse sr = client.prepareSearch()

.setQuery(QueryBuilders.termQuery("gender", "male"))

.addAggregation(aggregation)

.get();

#### 10.3.10.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.significant.SignificantTerms;

// sr is here your SearchResponse object

SignificantTerms agg = sr.getAggregations().get("significant\_countries");

// For each entry

for (SignificantTerms.Bucket entry : agg.getBuckets()) {

entry.getKey(); // Term

entry.getDocCount(); // Doc count

}

### 10.3.11  范围聚合(Range Aggregation)

下面是如何运用Java API使用[范围聚合(Range Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-range-aggregation.htm)。

#### 10.3.11.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.range("agg")

.field("height")

.addUnboundedTo(1.0f) // from -infinity to 1.0 (excluded)

.addRange(1.0f, 1.5f) // from 1.0 to 1.5 (excluded)

.addUnboundedFrom(1.5f); // from 1.5 to +infinity

#### 10.3.11.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.range.Range;

// sr is here your SearchResponse object

Range agg = sr.getAggregations().get("agg");

// For each entry

for (Range.Bucket entry : agg.getBuckets()) {

String key = entry.getKeyAsString(); // Range as key

Number from = (Number) entry.getFrom(); // Bucket from

Number to = (Number) entry.getTo(); // Bucket to

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], from [{}], to [{}], doc\_count [{}]", key, from, to, docCount);

}

这将会产生如下的结果：

key [\*-1.0], from [-Infinity], to [1.0], doc\_count [9]

key [1.0-1.5], from [1.0], to [1.5], doc\_count [21]

key [1.5-\*], from [1.5], to [Infinity], doc\_count [20]

### 10.3.12  时间范围聚合(Date Range Aggregation)

下面是如何运用Java API使用[时间范围聚合(Date Range Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-daterange-aggregation.htm)。

#### 10.3.12.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.dateRange("agg")

.field("dateOfBirth")

.format("yyyy")

.addUnboundedTo("1950") // from -infinity to 1950 (excluded)

.addRange("1950", "1960") // from 1950 to 1960 (excluded)

.addUnboundedFrom("1960"); // from 1960 to +infinity

#### 10.3.12.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.range.Range;

// sr is here your SearchResponse object

Range agg = sr.getAggregations().get("agg");

// For each entry

for (Range.Bucket entry : agg.getBuckets()) {

String key = entry.getKeyAsString(); // Date range as key

DateTime fromAsDate = (DateTime) entry.getFrom(); // Date bucket from as a Date

DateTime toAsDate = (DateTime) entry.getTo(); // Date bucket to as a Date

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], from [{}], to [{}], doc\_count [{}]", key, fromAsDate, toAsDate, docCount);

}

这将会产生如下效果：

key [\*-1950], from [null], to [1950-01-01T00:00:00.000Z], doc\_count [8]

key [1950-1960], from [1950-01-01T00:00:00.000Z], to [1960-01-01T00:00:00.000Z], doc\_count [5]

key [1960-\*], from [1960-01-01T00:00:00.000Z], to [null], doc\_count [37]

I

### 10.3.13  P范围聚合(Ip Range Aggregation)

下面是如何运用Java API使用[IP范围聚合(Ip Range Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-iprange-aggregation.htm)。

#### 10.3.13.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.ipRange("agg")

.field("ip")

.addUnboundedTo("192.168.1.0") // from -infinity to 192.168.1.0 (excluded)

.addRange("192.168.1.0", "192.168.2.0") // from 192.168.1.0 to 192.168.2.0 (excluded)

.addUnboundedFrom("192.168.2.0"); // from 192.168.2.0 to +infinity

请注意，您也可以使用ip掩码作为范围：

AggregationBuilder aggregation =

AggregationBuilders

.ipRange("agg")

.field("ip")

.addMaskRange("192.168.0.0/32")

.addMaskRange("192.168.0.0/24")

.addMaskRange("192.168.0.0/16");

#### 10.3.13.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.range.Range;

// sr is here your SearchResponse object

Range agg = sr.getAggregations().get("agg");

// For each entry

for (Range.Bucket entry : agg.getBuckets()) {

String key = entry.getKeyAsString(); // Ip range as key

String fromAsString = entry.getFromAsString(); // Ip bucket from as a String

String toAsString = entry.getToAsString(); // Ip bucket to as a String

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], from [{}], to [{}], doc\_count [{}]", key, fromAsString, toAsString, docCount);

}

第一个列子将会产生如下结果：

key [\*-192.168.1.0], from [null], to [192.168.1.0], doc\_count [13]

key [192.168.1.0-192.168.2.0], from [192.168.1.0], to [192.168.2.0], doc\_count [14]

key [192.168.2.0-\*], from [192.168.2.0], to [null], doc\_count [23]

第二个例子将会产生如下结果：

key [192.168.0.0/32], from [192.168.0.0], to [192.168.0.1], doc\_count [0]

key [192.168.0.0/24], from [192.168.0.0], to [192.168.1.0], doc\_count [13]

key [192.168.0.0/16], from [192.168.0.0], to [192.169.0.0], doc\_count [50]

### 10.3.14  柱状图聚合(Histogram Aggregation)

下面是如何运用Java API使用[柱状图聚合(Histogram Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-histogram-aggregation.htm)。

#### 10.3.14.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.histogram("agg")

.field("height")

.interval(1);

请注意，您也可以使用ip掩码作为范围：

import org.elasticsearch.search.aggregations.bucket.histogram.Histogram;

// sr is here your SearchResponse object

Histogram agg = sr.getAggregations().get("agg");

// For each entry

for (Histogram.Bucket entry : agg.getBuckets()) {

Long key = (Long) entry.getKey(); // Key

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], doc\_count [{}]", key, docCount);

}

### 10.3.15  日期柱状图聚合(Date Histogram Aggregation)

下面是如何运用Java API使用[日期柱状图聚合(Date Histogram Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-datehistogram-aggregation.htm)。

#### 10.3.15.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.dateHistogram("agg")

.field("dateOfBirth")

.interval(DateHistogramInterval.YEAR);

或者如果要设置10天的间隔：

AggregationBuilder aggregation =

AggregationBuilders

.dateHistogram("agg")

.field("dateOfBirth")

.interval(DateHistogramInterval.days(10));

#### 10.3.15.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.histogram.Histogram;

// sr is here your SearchResponse object

Histogram agg = sr.getAggregations().get("agg");

// For each entry

for (Histogram.Bucket entry : agg.getBuckets()) {

DateTime key = (DateTime) entry.getKey(); // Key

String keyAsString = entry.getKeyAsString(); // Key as String

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], date [{}], doc\_count [{}]", keyAsString, key.getYear(), docCount);

}

这回产生如下的结果：

key [1942-01-01T00:00:00.000Z], date [1942], doc\_count [1]

key [1945-01-01T00:00:00.000Z], date [1945], doc\_count [1]

key [1946-01-01T00:00:00.000Z], date [1946], doc\_count [1]

...

key [2005-01-01T00:00:00.000Z], date [2005], doc\_count [1]

key [2007-01-01T00:00:00.000Z], date [2007], doc\_count [2]

key [2008-01-01T00:00:00.000Z], date [2008], doc\_count [3]

### 10.3.16  geo距离聚合(Geo Distance Aggregation)

下面是如何运用Java API使用[geo距离聚合(Geo Distance Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-geodistance-aggregation.htm)。

#### 10.3.16.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.geoDistance("agg")

.field("address.location")

.point(new GeoPoint(48.84237171118314,2.33320027692004))

.unit(DistanceUnit.KILOMETERS)

.addUnboundedTo(3.0)

.addRange(3.0, 10.0)

.addRange(10.0, 500.0);

#### 10.3.16.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.range.Range;

// sr is here your SearchResponse object

Range agg = sr.getAggregations().get("agg");

// For each entry

for (Range.Bucket entry : agg.getBuckets()) {

String key = entry.getKeyAsString(); // key as String

Number from = (Number) entry.getFrom(); // bucket from value

Number to = (Number) entry.getTo(); // bucket to value

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], from [{}], to [{}], doc\_count [{}]", key, from, to, docCount);

}

这将会产生如下结果：

key [\*-3.0], from [0.0], to [3.0], doc\_count [161]

key [3.0-10.0], from [3.0], to [10.0], doc\_count [460]

key [10.0-500.0], from [10.0], to [500.0], doc\_count [4925]

### 10.3.17  geo哈希网格聚合(Geo Hash Grid Aggregation)

下面是如何运用Java API使用[geo哈希网格聚合(Geo Hash Grid Aggregation)](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-aggregations-bucket-geohashgrid-aggregation.htm)。

#### 10.3.17.1  准备聚合请求

下面是一个如何创建聚合请求的例子：

AggregationBuilder aggregation =

AggregationBuilders

.geohashGrid("agg")

.field("address.location")

.precision(4);

#### 10.3.17.2  使用聚合响应

导入聚合定义类

import org.elasticsearch.search.aggregations.bucket.geogrid.GeoHashGrid;

// sr is here your SearchResponse object

GeoHashGrid agg = sr.getAggregations().get("agg");

// For each entry

for (GeoHashGrid.Bucket entry : agg.getBuckets()) {

String keyAsString = entry.getKeyAsString(); // key as String

GeoPoint key = (GeoPoint) entry.getKey(); // key as geo point

long docCount = entry.getDocCount(); // Doc count

logger.info("key [{}], point {}, doc\_count [{}]", keyAsString, key, docCount);

}

这将会产生如下结果：

key [gbqu], point [47.197265625, -1.58203125], doc\_count [1282]

key [gbvn], point [50.361328125, -4.04296875], doc\_count [1248]

key [u1j0], point [50.712890625, 7.20703125], doc\_count [1156]

key [u0j2], point [45.087890625, 7.55859375], doc\_count [1138]

...

# 11  渗透过滤API

过滤器允许用户针对索引注册查询，然后发送包括doc的渗透请求，从该组注册查询中获取与该文档匹配的查询。  
阅读本指南之前，请阅读主[渗滤文档](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-percolate.htm)。

//This is the query we're registering in the percolator

QueryBuilder qb = termQuery("content", "amazing");

//Index the query = register it in the percolator

client.prepareIndex("myIndexName", ".percolator", "myDesignatedQueryName")

.setSource(jsonBuilder()

.startObject()

.field("query", qb) // Register the query

.endObject())

.setRefresh(true) // Needed when the query shall be available immediately

.execute().actionGet();

这里将以名称myDesignatedQueryName对上述条件查询进行索引。  
为了根据注册的查询检查文档，请使用此代码：

//Build a document to check against the percolator

XContentBuilder docBuilder = XContentFactory.jsonBuilder().startObject();

docBuilder.field("doc").startObject(); //This is needed to designate the document

docBuilder.field("content", "This is amazing!");

docBuilder.endObject(); //End of the doc field

docBuilder.endObject(); //End of the JSON root object

//Percolate

PercolateResponse response = client.preparePercolate()

.setIndices("myIndexName")

.setDocumentType("myDocumentType")

.setSource(docBuilder).execute().actionGet();

//Iterate over the results

for(PercolateResponse.Match match : response) {

//Handle the result which is the name of

//the query in the percolator

}

# 12  查询DSL

Elasticsearch以与[REST查询DSL](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl.htm)类似的方式提供完整的Java查询dsl。 查询构建器的工厂是QueryBuilders。 查询准备就绪后，您可以使用[Search API](https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.3/java-search.htm)。  
要使用QueryBuilders只需在您的类中导入它们：

import static org.elasticsearch.index.query.QueryBuilders.\*;

注意，您可以使用toString()方法在QueryBuilder对象上轻松打印（也称为调试）JSON生成的查询。  
然后，QueryBuilder可以与任何接受查询的API（例如count和search）一起使用。

## 12.1  匹配所有的查询

查看 [匹配所有的查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-match-all-query.htm)。

QueryBuilder qb = matchAllQuery();

## 12.2  全文搜索

高级全文搜索通常用于在全文字段（如电子邮件正文）上运行全文搜索。 他们了解如何分析查询的字段，并在执行之前将每个字段的分析器（或search\_analyzer）应用于查询字符串。  
此组中的查询为：  
1. \l "\_匹配查询" 匹配查询：用于执行全文查询的标准查询，包括模糊匹配和词组或邻近查询。  
2. \l "\_多匹配查询" 多匹配查询:多字段版本的匹配查询。  
3. \l "\_一般条件查询" 一般条件查询：一个更专门的查询，更多地偏爱不常见的单词。  
4. \l "\_查询字符串查询" 查询字符串查询：支持紧凑的Lucene查询字符串语法，允许您在单个查询字符串中指定AND | OR | NOT条件和多字段搜索。 仅限专家用户。  
5. \l "\_简单的查询字符串查询" 简单的查询字符串查询：一个更简单，更健壮的query\_string语法版本，适合直接暴露给用户。

### 12.2.1  匹配查询

查看[匹配查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-match-query.htm) 

QueryBuilder qb = matchQuery(

"name", [注释1]

"kimchy elasticsearch" [注释2]

);

[注释1]：属性  
[注释2]：文本

### 12.2.2  多匹配查询

查看 [多匹配查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-multi-match-query.htm) 

QueryBuilder qb = multiMatchQuery(

"kimchy elasticsearch", [注释1]

"user", "message" [注释2]

);

[注释1]：文本  
[注释2]：属性列表

### 12.2.3  一般条件查询

查看[一般条件查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-common-terms-query.htm) 

QueryBuilder qb = commonTermsQuery("name", [注释1]

"kimchy"); [注释2]

[注释1]：属性  
[注释2]：值

### 12.2.4  查询字符串查询

查看[查询字符串查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-query-string-query.htm) 

QueryBuilder qb = queryStringQuery("+kimchy -elasticsearch"); [注释1]

[注释1]：文本

### 12.2.5  简单的查询字符串查询

查看[简单的查询字符串查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-simple-query-string-query.htm) 

QueryBuilder qb = simpleQueryStringQuery("+kimchy -elasticsearch"); [注释1]

[注释1]：文本

## 12.3  条件等级查询

虽然全文搜索将在执行之前分析查询字符串，但是条件等级查询对存储在反向索引中的确切条件进行操作。  
这些查询通常用于结构化数据，如数字，日期和枚举，而不是全文本字段。 或者，它们允许您绘制低级查询，以前的分析过程。  
此组中的查询为：

### 12.3.1  词汇查询

查找包含在指定字段中指定的确切词汇的文档。  
查看[词汇查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-term-query.htm) 

QueryBuilder qb = termQuery(

"name", [注释1]

"kimchy" [注释1]

);

[注释1]：属性  
[注释2]：文本

### 12.3.2  多词汇查询

查找包含指定字段中指定的所有确切词汇的文档。  
查看[多词汇查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-terms-query.htm) 

QueryBuilder qb = termsQuery("tags", [注释1]

"blue", "pill"); [注释2]

[注释1]：属性  
[注释2]：多个要匹配的文本词汇

### 12.3.3  范围查询

查找指定字段包含指定范围内的值（日期，数字或字符串）的文档。  
查看[范围查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-range-query.htm) 

QueryBuilder qb = rangeQuery("price") [注释1]

.from(5) [注释2]

.to(10) [注释3]

.includeLower(true) [注释4]

.includeUpper(false); [注释5]

[注释1]：属性  
[注释2]：起始值  
[注释3]：结束值  
[注释4]：是否包含起始值，false：不包含，true：包含  
[注释5]：是否包含结束值，false：不包含，true：包含

// A simplified form using gte, gt, lt or lte

QueryBuilder qb = rangeQuery("age") [注释1]

.gte("10") [注释2]

.lt("20"); [注释3]

[注释1]：属性  
[注释2]：设置为10，includeLower设置为true  
[注释3]：设置为20，includeUpper 设置为false

### 12.3.4  存在查询

查找指定的字段包含任何非空值的文档  
查看[存在查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-exists-query.htm) 

QueryBuilder qb = existsQuery("name"); [注释1]

[注释1]：属性

### 12.3.5  缺失查询

查找指定的字段缺少或仅包含空值的文档。  
警告：在2.2.0中已弃用。请把must\_not()替换为query()。  
查看[缺失查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-missing-query.htm) 

QueryBuilder qb = missingQuery("user"); [注释1]

.existence(true) [注释2]

.nullValue(true); [注释3]

[注释1]：属性  
[注释2]：查找不存在的丢失的属性  
[注释3]：查找为Null值的丢失的属性

### 12.3.6  前缀查询

查找指定字段包含具有指定的精确前缀的术语的文档。  
查看[前缀查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-prefix-query.htm) 

QueryBuilder qb = prefixQuery(

"brand", [注释1]

"heine" [注释2]

);

[注释1]：属性  
[注释2]：需要查询的前缀词汇

### 12.3.7  通配符查询

查找指定字段包含与指定模式匹配的术语的文档，其中该模式支持单字符通配符（？）和多字符通配符（\*）  
查看[通配符查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-wildcard-query.htm) 

QueryBuilder qb = wildcardQuery("user", "k?mc\*");

### 12.3.8  正则表达式查询

查找指定字段包含与指定的正则表达式匹配的术语的文档。  
查看[正则表达式查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-regexp-query.htm) 

QueryBuilder qb = regexpQuery(

"name.first", [注释1]

"s.\*y"); [注释2]

[注释1]：属性  
[注释2]：正则表达式

### 12.3.9  模糊查询

查找指定字段包含与指定术语模糊相似的术语的文档。模糊性测量为1或2的Levenshtein编辑距离。  
查看[模糊查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-fuzzy-query.htm) 

QueryBuilder qb = fuzzyQuery(

"name", [注释1]

"kimzhy" [注释2]

);

[注释1]：属性  
[注释2]：文本

### 12.3.10  类型查询

查找指定类型的文档。  
查看[类型查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-type-query.htm) 

QueryBuilder qb = typeQuery("my\_type");[注释1]

[注释1]：索引类型(type)

### 12.3.11  多ID查询

查找具有指定类型和ID的文档。  
查看[多ID查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-ids-query.htm) 

QueryBuilder qb = idsQuery("my\_type", "type2")

.addIds("1", "4", "100");

QueryBuilder qb = idsQuery()[注释1]

.addIds("1", "4", "100");

[注释1]：id列表

## 12.4  复合查询

复合查询包装其他复合或叶查询，要么结合其结果和分数，更改其行为，或从查询切换到过滤器上下文。  
此组中的查询为：

### 12.4.1  常数 \_score查询

一个查询，它包装另一个查询，但在过滤器上下文中执行。所有匹配的文档都赋予相同的“常量”\_score。  
查看[常数 \_score查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-constant-score-query.htm) 

QueryBuilder qb = constantScoreQuery(

termQuery("name","kimchy") [注释1]

)

.boost(2.0f); [注释2]

[注释1]：你的查询  
[注释2]：查询分数

### 12.4.2  布尔查询

用于组合多个叶子或复合查询子句的默认查询，必须为must，must\_not或过滤子句。 must和should子句具有它们的分数组合 - 更多的匹配子句，更好 - 而must\_not和过滤器子句在过滤器上下文中执行。  
查看[布尔查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-bool-query.htm) 

QueryBuilder qb = boolQuery()

.must(termQuery("content", "test1")) [注释1]

.must(termQuery("content", "test4")) [注释2]

.mustNot(termQuery("content", "test2")) [注释3]

.should(termQuery("content", "test3")); [注释4]

[注释1]：必须查询  
[注释2]：必须查询  
[注释3]：必须不查询  
[注释4]：应该查询

### 12.4.3  Dis max查询

接受多个查询的查询，并返回与任何查询子句匹配的任何文档。当bool查询组合来自所有匹配查询的分数时，dis\_max查询使用单个最佳匹配查询子句的分数。  
查看[Dis max查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-dis-max-query.htm) 

QueryBuilder qb = disMaxQuery()

.add(termQuery("name", "kimchy")) [注释1]

.add(termQuery("name", "elasticsearch")) [注释2]

.boost(1.2f) [注释3]

.tieBreaker(0.7f); [注释4]

[注释1]：添加你的查询  
[注释2]：添加你的查询  
[注释3]：条件标高  
[注释4]：中断标高

### 12.4.4  函数得分查询

使用函数修改主查询返回的分数，以考虑诸如流行度，新近度，距离或使用脚本实现的自定义算法等因素。  
查看[函数得分查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-function-score-query.htm)   
使用ScoreFunctionBuilders 仅需要在你的项目中导入下面的类：

import static org.elasticsearch.index.query.functionscore.ScoreFunctionBuilders.\*;

QueryBuilder qb = functionScoreQuery()

.add(

matchQuery("name", "kimchy"), [注释1]

randomFunction("ABCDEF") [注释2]

)

.add(

exponentialDecayFunction("age", 0L, 1L) [注释3]

);

[注释1]：添加第一个基于一个查询的函数(方法)  
[注释2]：基于给定的分数排序方式随机化分数  
[注释3]：给定一个基于年龄属性的其他函数

### 12.4.5  提升查询

返回与肯定查询匹配的文档，但减少与否定查询匹配的文档的分数。  
查看[提升查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-boosting-query.htm) 

QueryBuilder qb = boostingQuery()

.positive(termQuery("name","kimchy")) [注释1]

.negative(termQuery("name","dadoonet")) [注释2]

.negativeBoost(0.2f); [注释3]

[注释1]：查询将提升的文档  
[注释2]：查询将降级的文档  
[注释3]：降低的值

### 12.4.6  索引查询

对指定的索引执行一个查询，对其他索引执行另一个查询。  
查看[索引查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-indices-query.htm) 

// Using another query when no match for the main one

QueryBuilder qb = indicesQuery(

termQuery("tag", "wow"), [注释1]

"index1", "index2" [注释2]

).noMatchQuery(termQuery("tag", "kow"));[注释3]

[注释1]：查询将提升的文档  
[注释2]：查询将降级的文档  
[注释3]：降低的值

// Using all (match all) or none (match no documents)

QueryBuilder qb = indicesQuery(

termQuery("tag", "wow"), [注释1]

"index1", "index2" [注释2]

).noMatchQuery("all"); [注释3]

[注释1]: 要执行的选定的索引查询。  
[注释2]：选择指数。  
[注释3]：无（无匹配文档）和全部（匹配所有文档）。 默认为all。

### 12.4.7  和查询

bool查询的同义词。  
注意：在2.0.0中已弃用。请使用布尔查询替代。  
查看[和查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-and-query.html))

QueryBuilder query = andQuery(

rangeQuery("postDate").from("2010-03-01").to("2010-04-01"), [注释1]

prefixQuery("name.second", "ba")); [注释2]

[注释1]: 查询条件。  
[注释2]：

### 12.4.8  否查询

bool查询的同义词。  
警告：在2.1.0中已弃用，使用布尔查询添加并且添加mustNot()方法替代这个方法。  
查看[否查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-not-query.html))。

QueryBuilder qb = notQuery(

rangeQuery("price").from("1").to("2") [注释1]

);

[注释1]: 查询条件。

### 12.4.9  或查询

bool查询的同义词。  
警告：在2.0.0中已弃用。使用bool查询替代。  
查看[或查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-or-query.html))

QueryBuilder qb = orQuery(

rangeQuery("price").from(1).to(2), [注释1]

matchQuery("name", "joe") [注释2]

);

[注释1]: 多个查询条件。  
[注释2]：

### 12.4.10  过滤查询

将查询上下文中的查询子句与过滤器上下文中的另一个组合。 [2.0.0]在2.0.0中已弃用。使用bool查询。  
警告：在2.0.0中已弃用。使用bool查询替换使用must子句查询并且使用filter子句过滤。  
查看[过滤查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-filtered-query.html))

QueryBuilder qb = filteredQuery(

matchQuery("name", "kimchy"), // [注释1]

rangeQuery("dateOfBirth").from("1900").to("2100") // [注释2]

);

[注释1]: 将要被用在卷轴里面的查询。  
[注释2]：仅仅用在过滤结果集合的查询

### 12.4.11  限制查询

限制每个分片检查的文档数。  
查看[限制查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-limit-query.html))

QueryBuilder qb = limitQuery(100); [注释1]

[注释1]：每个分片的文档数

## 12.5  连接(Join)查询

在像Elasticsearch这样的分布式系统中执行一个SQL样式连接代价是非常昂贵的。 相反，Elasticsearch提供了两种形式的连接，它们被设计为水平缩放。

### 12.5.1  嵌套查询

文档可能包含嵌套类型的字段。 这些字段用于索引对象数组，其中每个对象可以作为独立文档查询（使用嵌套查询）。  
查看[嵌套查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-nested-query.html))

QueryBuilder qb = nestedQuery(

"obj1", [注释1]

boolQuery() [注释2]

.must(matchQuery("obj1.name", "blue"))

.must(rangeQuery("obj1.count").gt(5))

)

.scoreMode("avg"); [注释3]

[注释1]: 嵌套文档的路径。  
[注释2]：你的查询。 查询中引用的任何字段必须使用完整路径（完全限定）。  
[注释3]: 分数模式可以是max，total，avg（默认）或none。

### 12.5.2  有子查询

单个索引中的两个文档类型之间可能存在父子关系。 has\_child查询返回其子文档与指定查询匹配的父文档，而has\_parent查询返回其父文档与指定查询匹配的子文档。  
查看[有子查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-has-child-query.html))

QueryBuilder qb = hasChildQuery(

"blog\_tag", [注释1]

termQuery("tag","something") [注释2]

);

[注释1]：要查询的子类型  
[注释2]：查询。

### 12.5.3  有父查询

单个索引中的两个文档类型之间可能存在父子关系。 has\_child查询返回其子文档与指定查询匹配的父文档，而has\_parent查询返回其父文档与指定查询匹配的子文档。  
查看[有父查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-has-parent-query.html))

QueryBuilder qb = hasParentQuery(

"blog", [注释1]

termQuery("tag","something") [注释2]

);

[注释1]：要查询的父类型  
[注释2]：查询。

## 12.6  GEO查询

Elasticsearch支持两种类型的地理数据：支持纬度/经度对的geo\_point字段和支持点，以及支持线，圆，多边形，多多边形等的geo\_shape字段。  
此组中的查询为：

### 12.6.1  地理形状查询

查找具有指定地理形状的相交，包含或不相交的地理形状的文档。  
查看[地理形状查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-geo-shape-query.html))  
注意：geo\_shape类型使用Spatial4J和JTS，它们都是可选的依赖关系。 因此，为了使用此类型，必须将Spatial4J和JTS添加到类路径中：

<dependency>

<groupId>com.spatial4j</groupId>

<artifactId>spatial4j</artifactId>

<version>0.4.1</version> [注释1]

</dependency>

<dependency>

<groupId>com.vividsolutions</groupId>

<artifactId>jts</artifactId>

<version>1.13</version> [注释2]

<exclusions>

<exclusion>

<groupId>xerces</groupId>

<artifactId>xercesImpl</artifactId>

</exclusion>

</exclusions>

</dependency>

[注释1]：检查[Maven Central](http://search.maven.org/#search%7Cga%7C1%7Cg%3A%22com.spatial4j%22%20AND%20a%3A%22spatial4j%22))中的更新。  
[注释2]：检查[Maven Central](http://search.maven.org/#search%7Cga%7C1%7Cg%3A%22com.spatial4j%22%20AND%20a%3A%22spatial4j%22))中的更新。

// Import ShapeRelation and ShapeBuilder

import org.elasticsearch.common.geo.ShapeRelation;

import org.elasticsearch.common.geo.builders.ShapeBuilder;

QueryBuilder qb = geoShapeQuery(

"pin.location", [注释1]

ShapeBuilder.newMultiPoint() [注释2]

.point(0, 0)

.point(0, 10)

.point(10, 10)

.point(10, 0)

.point(0, 0),

ShapeRelation.WITHIN); [注释3]

[注释1]：属性  
[注释2]：地理模型。  
[注释3]：关系可以是ShapeRelation.WITHIN，ShapeRelation.INTERSECTS或ShapeRelation.DISJOINT。

// Using pre-indexed shapes

QueryBuilder qb = geoShapeQuery(

"pin.location", [注释1]

"DEU", [注释2]

"countries", [注释3]

ShapeRelation.WITHIN) [注释4]

.indexedShapeIndex("shapes") [注释5]

.indexedShapePath("location"); [注释6]

[注释1]：属性  
[注释2]：包含预先建立索引的形状的文档的ID。  
[注释3]：预索引形状所在的索引类型。  
[注释4]：关系  
[注释5]：预索引形状所在的索引的名称。 默认为形状。  
[注释6]：指定为包含预先建立索引的形状的路径的字段。 默认为shape。

### 12.6.2  地理边界框查询

查找具有落入指定矩形中的地理点的文档。  
查看[地理边界查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-geo-bounding-box-query.html))

QueryBuilder qb = geoBoundingBoxQuery("pin.location") [注释 1]

.topLeft(40.73, -74.1) [注释 2]

.bottomRight(40.717, -73.99); [注释 3]

[注释1]：属性  
[注释2]：左边边界框顶部。  
[注释3]：右边边界框底部。

### 12.6.3  地理距离查询

使用在中心点指定距离内的地理点查找文档。  
查看[地理距离查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-geo-distance-query.html))

QueryBuilder qb = geoDistanceQuery("pin.location") [注释 1]

.point(40, -70) [注释 2

.distance(200, DistanceUnit.KILOMETERS) [注释 3]

.optimizeBbox("memory") [注释 4]

.geoDistance(GeoDistance.ARC); [注释 5]

[注释1]：属性  
[注释2]：中心点  
[注释3]：距中心点的距离。  
[注释4]：优化边界框：内存，索引或无  
[注释5]：距离计算模式：GeoDistance.SLOPPY\_ARC（默认），GeoDistance.ARC（略微更精确，但效率更慢）或GeoDistance.PLANE（更快，但在长距离和接近极点不准确）。

### 12.6.4  地理距离范围查询

与geo\_point查询类似，但范围从距中心点的指定距离开始。  
查看[地理距离范围查询](https://www.blog-china.cn/template/documentHtml/%3C/span%3E%3Cspan%3E%20%20%3C/span%3E%3Cspan%3Ehttp:/www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-geo-distance-range-query.htm)

QueryBuilder qb = geoDistanceRangeQuery("pin.location") [注释 1]

.point(40, -70) [注释 2]

.from("200km") [注释 3]

.to("400km") [注释 4]

.includeLower(true) [注释 5]

.includeUpper(false) [注释 6]

.optimizeBbox("memory") [注释 7]

.geoDistance(GeoDistance.ARC); [注释 8]

[注释1]：属性  
[注释2]：中心点  
[注释3]：从中心点的起始距离。  
[注释4]：距中心点的结束距离.  
[注释5]：包括最小值，表明值false时‘from’是gt或值为true时‘from’是gte.  
[注释6]：包括最大值，表明当值为false时‘to’是lt或值为true时‘to’是lte。  
[注释7]：优化边界框：内存，索引或无  
[注释8]：距离计算模式：GeoDistance.SLOPPY\_ARC（默认），GeoDistance.ARC（略微更精确，但显着更慢）或GeoDistance.PLANE（更快，但不准确在长距离和接近极点）   
译者注：[注释5]、 [注释6]中的意思是是否包含最大值或者最小值。

### 12.6.5  地理多边形查询

查找具有指定多边形内的地理点的文档。  
查看[地理多边形查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-geo-polygon-query.html))

QueryBuilder qb = geoPolygonQuery("pin.location") [注释 1]

.addPoint(40, -70) [注释 2]

.addPoint(30, -80) [注释 3]

.addPoint(20, -90); [注释 4]

[注释1]：属性   
[注释2]：添加属于你的文档的特点的多边形   
[注释3]：添加属于你的文档的特点的多边形  
[注释4]：添加属于你的文档的特点的多边形

### 12.6.6  地理散列单元格查询

查找geohash与指定点的geohash相交的地理位置。  
查看[地理散列单元格查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-geohash-cell-query.htm)

QueryBuilder qb = geoHashCellQuery("pin.location", [注释 1]

new GeoPoint(13.4080, 52.5186)) [注释 2]

.neighbors(true) [注释 3]

.precision(3); [注释 4]

[注释1]：属性   
[注释2]：点， 也可以像u30一样的哈希。  
[注释3]：过滤器的邻居选项提供了过滤给定单元格旁边的单元格的可能性。  
[注释4]：精度水平。

## 12.7  高级查询

这组包含不适合其他组的查询：

### 12.7.1  更多此查询（mlt）

此查询查找与指定的文本，文档或文档集合相似的文档。  
查看\*

QueryBuilder qb = moreLikeThisQuery("name.first", "name.last") [注释 1]

.like("text like this one") [注释 2]

.minTermFreq(1) [注释 3]

.maxQueryTerms(12); [注释 4]

[注释1]：属性   
[注释2]：文本。  
[注释3]：忽略阈值。  
[注释4]：生成的查询中的最大字数。

### 12.7.2  模板查询

模板查询接受一个胡子模板（内联，索引或来自文件）和参数映射，并将两者组合以生成要执行的最终查询。  
查询[搜索模板](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/search-template.html))文档  
在Map中定义你的模板参数:

Map<String, Object> template\_params = new HashMap<>();

template\_params.put("param\_gender", "male");

您可以在config / scripts中使用您存储的搜索模板。 例如，如果您有一个名为config /scripts/template\_gender.mustache的文件包含：

{

"template" : {

"query" : {

"match" : {

"gender" : "{{param\_gender}}"

}

}

}

}

定义你的查询模板:

QueryBuilder qb = templateQuery(

"gender\_template", [注释 1]

ScriptService.ScriptType.FILE, [注释 2]

template\_params); [注释 3]

[注释1]：模板名称   
[注释2]：模板存储在磁盘中的gender\_template.mustache上。  
[注释3]：所有的查询参数。  
您还可以将模板存储在名为.scripts的特殊索引中：

client.preparePutIndexedScript("mustache", "template\_gender",

"{\n" +

" \"template\" : {\n" +

" \"query\" : {\n" +

" \"match\" : {\n" +

" \"gender\" : \"{{param\_gender}}\"\n" +

" }\n" +

" }\n" +

" }\n" +

"}").get();

请使用ScriptService.ScriptType.INDEXED来执行索引中的模板，：

QueryBuilder qb = templateQuery(

"gender\_template", [注释 1]

ScriptService.ScriptType.INDEXED, [注释 2]

template\_params); [注释 3]

[注释1]：模板名称   
[注释2]：模板存储在索引中。  
[注释3]：所有的查询参数。

### 12.7.3  脚本查询

此查询允许脚本充当过滤器。 另请参见[function\_score](https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.3/java-compound-queries.html#java-query-dsl-function-score-query))查询。  
查询[脚本查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-script-query.html))

QueryBuilder qb = scriptQuery(

new Script("doc['num1'].value > 1") [注释 1]

);

[注释1]：内联的脚本  
如果你在每个数据节点上存储了一个名为mygroovyscript.groovy的脚本：

doc['num1'].value > param1

你可以这样用它：

QueryBuilder qb = scriptQuery(

new Script(

"mygroovyscript", [注释 1]

ScriptService.ScriptType.FILE, [注释 2]

"groovy", [注释 3]

ImmutableMap.of("param1", 5)) [注释 4]

);

[注释1]：脚本名称  
[注释2]：脚本类型：ScriptType.FILE，ScriptType.INLINE或ScriptType.INDEXED。  
[注释3]：脚本引擎。  
[注释4]：所有的以Map封装的查询参数。

## 12.8  跨度(Span)查询

跨度查询是低级别位置查询，其提供对指定项的顺序和接近度的专家控制。 这些通常用于实施对法律文件或专利的非常具体的查询。  
跨度查询不能与非跨度查询混合（除了span\_multi查询）。  
此组中的查询为：

### 12.8.1  跨度项(span\_term)查询

相当于术语query，但用于其他span查询。  
查看[span\_term 查询]( http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-span-term-query.html)[link end]

QueryBuilder qb = spanTermQuery(

"user", [注释 1]

"kimchy" [注释 2]

);

[注释1]：属性  
[注释2]：值

### 12.8.2  跨度多项(span\_multi)查询

包含字词，范围，前缀，通配符，regexp或模糊查询。  
查看[span\_multi查询](https://www.blog-china.cn/template/documentHtml/%3C/span%3E%3Cspan%3E%20%20%3C/span%3E%3Cspan%3Ehttp:/www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-span-multi-term-query.html))

QueryBuilder qb = spanMultiTermQueryBuilder(

prefixQuery("user", "ki") [注释 1]

);

[注释1]：可以是任何MultiTermQueryBuilder类的构建器，例如：  
FuzzyQueryBuilder, PrefixQueryBuilder, RangeQueryBuilder, RegexpQueryBuilder or WildcardQueryBuilder.

### 12.8.3  跨度第一(span\_first)查询

接受另一个span查询，其匹配必须出现在字段的前N个位置。  
查看[span\_first查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-span-first-query.html))

QueryBuilder qb = spanFirstQuery(

spanTermQuery("user", "kimchy"), [注释 1]

3 [注释 2]

);

[注释1]：查询  
[注释2]：最大的结束位置

### 12.8.4  跨度接近查询(span\_near)查询

接受多个span查询，其匹配必须在彼此的指定距离内，并且可能以相同的顺序。  
查看[跨度接近查询](https://www.blog-china.cn/template/documentHtml/%3C/span%3E%3Cspan%3E%20%20%3C/span%3E%3Cspan%3Ehttp:/www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-span-near-query.html))

QueryBuilder qb = spanNearQuery()

.clause(spanTermQuery("field","value1")) [注释 1]

.clause(spanTermQuery("field","value2")) [注释 2]

.clause(spanTermQuery("field","value3")) [注释 3]

.slop(12) [注释 4]

.inOrder(false) [注释 5]

.collectPayloads(false); [注释 6]

[注释1]：跨度项查询  
[注释2]：跨度项查询  
[注释3]：跨度项查询  
[注释4]：坡度系数：介入的不匹配位置的最大数量  
[注释5]：是否需要匹配被排序的请求  
[注释6]：收集或不收藏有效载荷

### 12.8.5  跨度或(span\_or)查询

组合多个span查询 - 返回与任何指定查询匹配的文档。  
查看[跨度或span\_or查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-span-or-query.html))

QueryBuilder qb = spanOrQuery()

.clause(spanTermQuery("field","value1")) [注释 1]

.clause(spanTermQuery("field","value2")) [注释 2]

.clause(spanTermQuery("field","value3")); [注释 3]

[注释1]：跨度项查询列表  
[注释2]：跨度项查询列表  
[注释3]：跨度项查询列表

### 12.8.6  跨度不(span\_not)查询

包装另一个span查询，并排除与该查询匹配的任何文档。  
查看[跨度不span\_not查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-span-or-query.html))

QueryBuilder qb = spanOrQuery()

.clause(spanTermQuery("field","value1")) [注释 1]

.clause(spanTermQuery("field","value2")) [注释 2]

.clause(spanTermQuery("field","value3")); [注释 3]

[注释1]：跨度项查询列表  
[注释2]：跨度项查询列表

### 12.8.7  跨度包含(span\_containing)查询

接受span查询的列表，但只返回与第二个span查询匹配的span。  
查看[跨度b包含span\_containing查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-span-containing-query.html))

QueryBuilder qb = spanContainingQuery()

.little(spanTermQuery("field1","foo")) [注释 1]

.big(spanNearQuery() [注释 2]

.clause(spanTermQuery("field1","bar"))

.clause(spanTermQuery("field1","baz"))

.slop(5)

.inOrder(true)

);

[注释1]：小的部分  
[注释2]：大的部分 

### 12.8.8  跨越(span\_within)查询

返回单个span查询的结果，只要其span位于由其他span查询列表返回的span内即可。  
查看[跨越span\_within查询](http://www.elastic.co/guide/en/elasticsearch/reference/2.3/query-dsl-span-within-query.html))

QueryBuilder qb = spanWithinQuery()

.little(spanTermQuery("field1", "foo")) [注释 1]

.big(spanNearQuery() [注释 2]

.clause(spanTermQuery("field1", "bar"))

.clause(spanTermQuery("field1", "baz"))

.slop(5)

.inOrder(true)

);

[注释1]：小的部分  
[注释2]：大的部分

# 13  索引脚本API

索引脚本API允许人们与存储在elasticsearch索引中的脚本和模板进行交互。 它可用于创建，更新，获取和删除索引的脚本和模板。

PutIndexedScriptResponse = client.preparePutIndexedScript()

.setScriptLang("groovy")

.setId("script1")

.setSource("script","\_score \* doc['my\_numeric\_field'].value")

.execute()

.actionGet();

GetIndexedScriptResponse = client.prepareGetIndexedScript()

.setScriptLang("groovy")

.setId("script1")

.execute()

.actionGet();

DeleteIndexedScriptResponse = client.prepareDeleteIndexedScript()

.setScriptLang("groovy")

.setId("script1")

.execute()

.actionGet();

要存储模板，只需对scriptLang使用“mustache”。

### 13.0.1  脚本语言

API允许设置与之交互的索引脚本的语言。 如果没有提供，将使用默认的脚本语言。

# 14  Javav API管理

Elasticsearch提供了一个完整的Java API来处理管理任务。  
要访问它们，您需要从客户端调用admin()方法以获取AdminClient：

AdminClient adminClient = client.admin();

注意：在本指南的其余部分，我们将使用client.admin();

## 14.1  指数管理

要访问索引Java API，需要从AdminClient调用indices()方法：

IndicesAdminClient indicesAdminClient = client.admin().indices();

注意：在本指南的其余部分，我们将使用client.admin().indices();

### 14.1.1  创建索引

使用[IndicesAdminClient](https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.3/java-admin-indices.html))，您可以使用所有默认设置来创建索引，并且不需要进行映射：

client.admin().indices().prepareCreate("twitter").get();

### 14.1.2  设置索引

创建的每个索引可以具有与其相关联的特定设置。

client.admin().indices().prepareCreate("twitter")

.setSettings(Settings.builder() [注释 1]

.put("index.number\_of\_shards", 3)

.put("index.number\_of\_replicas", 2)

)

.get(); [注释 2]

[注释1]：索引设置  
[注释2]：执行索引并等待结果

### 14.1.3  进行映射

映射API允许您在创建索引时添加新类型：

client.admin().indices().prepareCreate("twitter") [注释 1]

.addMapping("tweet", "{\n" + [注释 2]

" \"tweet\": {\n" +

" \"properties\": {\n" +

" \"message\": {\n" +

" \"type\": \"string\"\n" +

" }\n" +

" }\n" +

" }\n" +

" }")

.get();

[注释1]：[创建一个叫做’tweet’的索引](https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.3/java-admin-indices.html#java-admin-indices-create-index))  
[注释2]：同时也添加了一个名为’tweet’的映射类型  
映射API也允许对已经存在的索引添加新的映射类型：

client.admin().indices().preparePutMapping("twitter") [注释 1]

.setType("user") [注释 2]

.setSource("{\n" + [注释 3]

" \"properties\": {\n" +

" \"name\": {\n" +

" \"type\": \"string\"\n" +

" }\n" +

" }\n" +

"}")

.get();

// You can also provide the type in the source document

client.admin().indices().preparePutMapping("twitter")

.setType("user")

.setSource("{\n" +

" \"user\":{\n" + [注释 4]

" \"properties\": {\n" +

" \"name\": {\n" +

" \"type\": \"string\"\n" +

" }\n" +

" }\n" +

" }\n" +

"}")

.get();

[注释1]：向已给已经存在的名为’ twitter’的索引添加一个映射  
[注释2]：添加一个名为’user’的映射类型  
[注释3]：‘user’有一个预定义的类型  
[注释4]：映射类型也可以在setSource()中提供  
你可以使用同样的API来更新一个映射：

client.admin().indices().preparePutMapping("twitter") [注释 1]

.setType("user") [注释 2]

.setSource("{\n" + [注释 3]

" \"properties\": {\n" +

" \"user\_name\": {\n" +

" \"type\": \"string\"\n" +

" }\n" +

" }\n" +

"}")

.get();

[注释1]：向已给已经存在的名为’ twitter’的索引添加一个映射  
[注释2]：要更新的名为’user’的映射类型  
[注释3]：现在‘user’映射类型有一个名为’user\_name’的新的属性

### 14.1.4  刷新

刷新API允许明确的刷新一个或者多个索引：

client.admin().indices().prepareRefresh().get(); [注释 1]

client.admin().indices()

.prepareRefresh("twitter") [注释 2]

.get();

client.admin().indices()

.prepareRefresh("twitter", "company") [注释 3]

.get();

[注释1]：刷新所有的索引  
[注释2]：刷新名为’ twitter’ 的映射类型  
[注释3]：刷新多个索引

### 14.1.5  获取设置

获取设置API允许得到一个或多个索引的设置:

GetSettingsResponse response = client.admin().indices()

.prepareGetSettings("company", "employee").get(); [注释 1]

for (ObjectObjectCursor<String, Settings> cursor : response.getIndexToSettings()) { [注释 2]

String index = cursor.key; [注释 3]

Settings settings = cursor.value; [注释 4]

Integer shards = settings.getAsInt("index.number\_of\_shards", null); [注释 5]

Integer replicas = settings.getAsInt("index.number\_of\_replicas", null); [注释 6]

}

[注释1]：获取多个索引的设置信息  
[注释2]：遍历所有的结果  
[注释3]：索引名称  
[注释4]：给定索引的设置信息  
[注释5]：该索引的分片数  
[注释6]：该索引的副本数

### 14.1.6  更新索引的设置信息

你可以通过下面的方法改变索引的设置i：

client.admin().indices().prepareUpdateSettings("twitter") [注释 1]

.setSettings(Settings.builder() [注释 2]

.put("index.number\_of\_replicas", 0)

)

.get();

[注释1]：要更新名为’ twitter’的索引设置信息  
[注释2]：设置

## 14.2  集群管理

想要允许集群Java API，你需要从AdminClient中调用cluster()方法：

ClusterAdminClient clusterAdminClient = client.admin().cluster();

注意：在本指南的其余部分，我们将使用client.admin().cluster();

### 14.2.1  集群状况(cluster health)

#### 14.2.1.1  状况(health)

集群状况API允许获取关于集群运行状况的非常简单的状态，并且还可以为您提供有关每个索引的集群状态的一些技术信息：

ClusterHealthResponse healths = client.admin().cluster().prepareHealth().get(); [注释1]

String clusterName = healths.getClusterName(); [注释2]

int numberOfDataNodes = healths.getNumberOfDataNodes(); [注释3]

int numberOfNodes = healths.getNumberOfNodes(); [注释4]

for (ClusterIndexHealth health : healths) { [注释5]

String index = health.getIndex(); [注释6]

int numberOfShards = health.getNumberOfShards(); [注释7]

int numberOfReplicas = health.getNumberOfReplicas(); [注释8]

ClusterHealthStatus status = health.getStatus(); [注释9]

}

[注释1]：获取所有索引的运行状况信息  
[注释2]：获取集群名称  
[注释3]：获取所有数据节点数量  
[注释4]：获取所有节点数量  
[注释5]：遍历所有的状况  
[注释6]：索引名称  
[注释7]：该索引的分片数  
[注释8]：该索引的副本数  
[注释8]：索引状态

#### 14.2.1.2  等待状态

您可以使用集群状态API等待整个集群或给定索引的特定状态：

client.admin().cluster().prepareHealth() [注释1]

.setWaitForYellowStatus() [注释2]

.get();

client.admin().cluster().prepareHealth("company") [注释3]

.setWaitForGreenStatus() [注释4]

.get();

client.admin().cluster().prepareHealth("employee") [注释5]

.setWaitForGreenStatus() [注释6]

.setTimeout(TimeValue.timeValueSeconds(2)) [注释7]

.get();

[注释1]：准备状况请求  
[注释2]：设定要等待的集群状况为黄色  
[注释3]：向一个名为’ company’准备一个状况请求  
[注释4]：等待集群状况为黄色  
[注释5]：向一个名为’ employee’准备一个状况请求  
[注释6]：等待集群状况为黄色  
[注释7]：最多等待2分钟  
如果索引没有达到返回预期的状态结果，并且你希望在此情况下结束，则需要明确解释结果：

ClusterHealthResponse response = client.admin().cluster().prepareHealth("company")

.setWaitForGreenStatus() [注释1]

.get();

ClusterHealthStatus status = response.getIndices().get("company").getStatus();

if (!status.equals(ClusterHealthStatus.GREEN)) {

throw new RuntimeException("Index is in " + status + " state"); [注释2]

}

[注释1]：设定要等待的集群状况为绿色  
[注释2]：如果没有变绿则抛出一个异常