**Elasticsearch文档**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 发布时间 | 修订章节 | 作者 |
| 1.0 | 2016.11.30 | 第一版 | 刘吴强 |
|  |  |  |  |
|  |  |  |  |

手册目的

ElasticSearch学习资料

声明

无

名词定义和缩略语说明

表 2 名词定义及缩略语说明

| 序号 | 缩写 | 说明 |
| --- | --- | --- |
| 1 | ES | Elasticsearch，一种设计用于云计算的分布式全文索引解决方案。 |
|  |  |  |
|  |  |  |

目录

[ 参考资料 2](#_Toc468269579)

[ 手册目的 2](#_Toc468269580)

[ 声明 2](#_Toc468269581)

[ 名词定义和缩略语说明 2](#_Toc468269582)

[1、总述 4](#_Toc468269583)

[1.1、简介 4](#_Toc468269584)

[1.2、国外使用案例 4](#_Toc468269585)

[1.3、Scaling Lucene 5](#_Toc468269586)

[1.1.1. Building Blocks 6](#_Toc468269587)

[1.1.2. Partitioning 6](#_Toc468269588)

[1.1.3. Replication 8](#_Toc468269589)

[1.1.4. Transaction Log 9](#_Toc468269590)

[2、服务器搭建 10](#_Toc468269591)

[3、Moduls 10](#_Toc468269592)

[3.1、Cluster 10](#_Toc468269593)

[3.2、Shards 10](#_Toc468269594)

[3.3、Replicas 10](#_Toc468269595)

[3.4、Recovery 11](#_Toc468269596)

[3.5、River 11](#_Toc468269597)

[3.6、Gateway 11](#_Toc468269598)

[3.7、discovery.zen 11](#_Toc468269599)

[3.8、Transport 11](#_Toc468269600)

[4、Java API 11](#_Toc468269601)

[4.1与集群交互 11](#_Toc468269602)

[4.1.1、Node方式 12](#_Toc468269603)

[4.1.2、TransportClient方式 12](#_Toc468269604)

# 1、总述

## 1.1、简介

ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是当前流行的企业级搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。

我们建立一个网站或应用程序，并要添加搜索功能，令我们受打击的是：搜索工作是很难的。我们希望我们的搜索解决方案要快，我们希望有一个零配置和一个完全免费的搜索模式，我们希望能够简单地使用JSON通过HTTP的索引数据，我们希望我们的搜索服务器始终可用，我们希望能够一台开始并扩展到数百，我们要实时搜索，我们要简单的多租户，我们希望建立一个云的解决方案。Elasticsearch旨在解决所有这些问题和更多的问题。

## 1.2、国外使用案例

**Github**

“Github使用Elasticsearch搜索20TB的数据，包括13亿的文件和1300亿行的代码”

这个不用介绍了吧，码农们都懂的，Github在2013年1月升级了他们的代码搜索，由solr转为elasticsearch，目前集群规模为26个索引存储节点和8个客户端节点（负责处理搜索请求），详情请看官方博客https://github.com/blog/1381-a-whole-new-code-search

**Foursquare**

”实时搜索5千万地点信息？Foursquare每天都用Elasticsearch做这样的事“

Foursquare是一家基于用户地理位置信息的手机服务网站，并鼓励手机用户同他人分享自己当前所在地理位置等信息。与其他老式网站不同，Foursquare用户界面主要针对手机而设计，以方便手机用户使用。

**SoundCloud**

“SoundCloud使用Elasticsearch来为1.8亿用户提供即时精准的音乐搜索服务”

SoundCloud是一家德国网站，提供音乐分享社区服务，成长很快，Alexa世界排名已达第236位。你可以在线录制或上传任何声音到SoundCloud与大家分享，可在线上传也可以通过软件客户端来上传音乐文件，没有文件大小限制，但免费版限制上传音频总长不可超过2个小时播放时长，每首歌曲限最多100次下载。SoundCloud允许音乐通过Flash播放器方式嵌入到网页中。

**Fog Creek**

“Elasticsearch使Fog Creek可以在400亿行代码中进行一个月3千万次的查询“

**StumbleUpon**

”Elasticsearch是StumbleUpon的关键部件，它每天为社区提供百万次的推荐服务“

StumbleUpon是个能发现你喜欢的网页的网站，进去时先注册，注册完就选择你感兴趣的东西，它会自动帮你推荐一些网页，如果你喜欢这个网页就点喜欢按钮，按 stumble按钮就会推荐下一个网页。

目前其数据量达到 25亿，基本数据存储在HBase中，并用 elasticsearch建立索引，elasticsearch 在其中除了用在搜索功能还有在推荐和统计功能。之前他们是使用solr作为搜索，由于solr满足不了他们的业务增长需要而替换为 elasticsearch。

**Mozilla**

Mozilla公司以火狐著名，它目前使用 WarOnOrange 这个项目来进行单元或功能测试，测试的结果以 json的方式索引到elasticsearch中，开发人员可以非常方便的查找 bug。

Socorro是Mozilla 公司的程序崩溃报告系统，一有错误信息就插入到 Hbase和Postgres 中，然后从 Hbase中读取数据索引到elasticsearch中，方便查找。

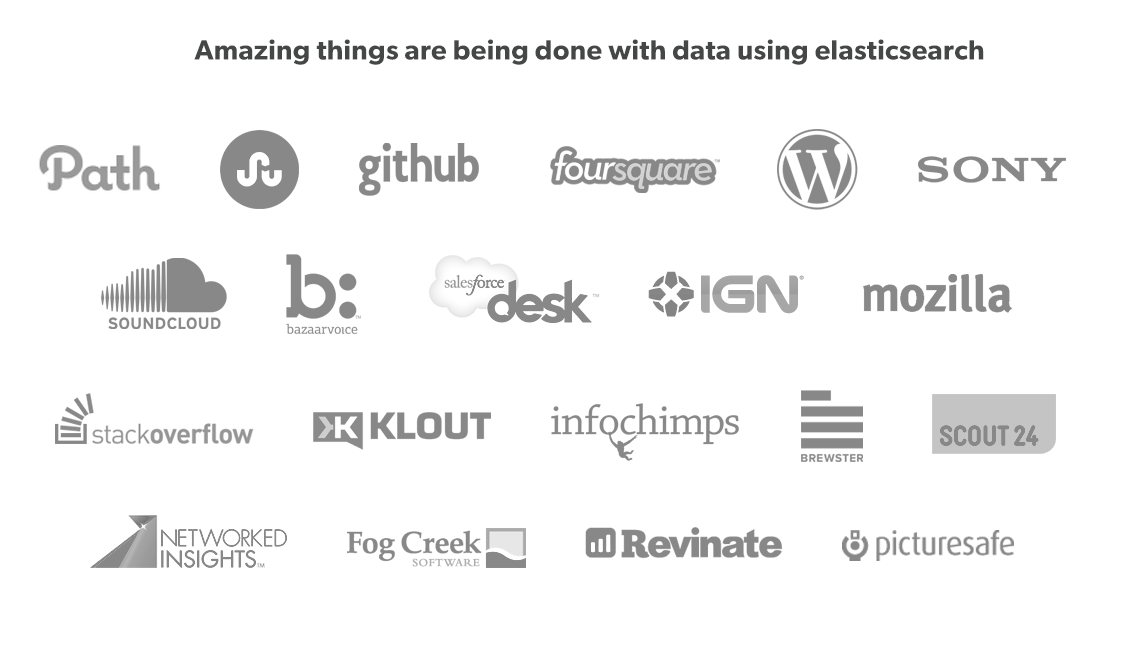
**Sony**

Sony公司使用elasticsearch 作为信息搜索引擎

**Infochimps**

“在 Infochimps，我们已经索引了25亿文档，总共占用 4TB的空间”。

Infochimps是一家位于德克萨斯州奥斯丁的创业公司，为大数据平台提供商。它主要提供基于hadoop的大数据处理方案。



## 1.3、Scaling Lucene

怎样在Lucene之上构建一个分布式、高度伸缩、接近实时的搜索引擎呢?

让我们回顾一下在搜索引擎（基于lucene）伸缩性这条路上都做了那些尝试，并且elasticsearch是如何尝试并去解决这些挑战的。

首先我们了解下最基础的理论知识 **building blocks** （这些理论基础是构建分布式近实时搜索引擎的基础）。 接着我们研究一下到底哪种才是最佳的分区策略 **partitioning** （将lucene索引文档分割到多个分布式的分片中去）。 然后我们同样需要决定使用哪种分区复制方式 **replication** （复制能够保证系统的高可用以及提高搜索的吞吐）。 最后，我们再看一下事务日志 **transaction log** （事务日志在elasticsearch里面是一个保证数据一致性的非常酷的功能）。

### Building Blocks

当我们要构建一个分布式接近实时的搜索引擎，并且要让lucene可伸缩可扩展，必须首先知道lucene的关键概念以及它们与我们要达成目标的一些局限性.

* **Directory**

Lucene Directory 是一个抽象的文件系统的接口，用来允许你读写文件，不管lucene的索引是存放在内存中还是在物理磁盘上，它都是通过lucene的Directory抽象层来访问和维护的。

* **IndexWriter**

IndexWriter 用来添加、删除和更新lucene里面的索引文档。这些操作是在内存中完成以保证更好的性能，但是如果要保证这些操作的持久化，这些操作是需要flush到磁盘的。并且，flush操作或者是显式的commit提交开销都是比较大的，因为这些操作通常需要处理很多的文件，而处理这些文件又涉及到大量的磁盘io。  
此外, 每次只能有一个IndexWriter对象来对一个索引目录进行索引操作，并且创建这个对象的开销很大，所以必须尽可能去重用这个对象.

* **Index Segments**

Lucene 索引被分解为很多段（segments）。每个索引段实际上是一个功能完整的lucene索引，一旦一个索引段创建完成，它将是不可变的，并且不能删除段里面的索引文档。commit提交操作用来往索引里面添加一个新段。lucene内部会来对这些段进行合并，所以我们必须要有策略来控制这些合并(MergePolisy, MergeScheuler, … etc)。Because segments need to be kept at bay they are being merged continuously by internal Lucene processes (MergePolisy, MergeScheuler, … etc)。

因为段是不可变的，所以用来做缓存（caching）是一个很好的选择，你可以加载所有的term词条并且创建一个跳跃列表（ skip lists ) ，或者用来构造FieldCache，如果段没有变化，你就不需要重新加载。

* **IndexReader**

IndexReader 用来执行搜索索引。这个对象通过IndexWriter来提供，并且创建代价也是比较高。一旦IndexReader打开之后，它就不能够发现打开之后的索引变化，如果要知道这些由IndexWriter产生的索引变化，除非刷新IndexReader对象（当然前提需要flush操作）。搜索操作在内部其实是按段来进行的（每次一个段）.

* **Near Real-Time Search**

获取一个新的IndexReader开销很大，所以也是我们不能每次一有索引操作就真的去获取一个新的IndexReader，你可以隔一段时间去刷新一下，比如每隔一秒钟等等，这也是我们在这里称之为接近实时( near real-time )的原因.

### Partitioning

可能用来伸缩Lucene的途径（Possible approach to Scale Lucene）：

* **Distributed Directory**

其中一个途径用来伸缩Lucene就是使用分布式文件系统，大文件会被拆分成chunks块并且会保存到分布式存储系统（比如 Coherence, Terracota, GigaSpaces or Infinispan等等)。这样IndexWriter和IndexReader都是工作在一个自定义的Directory分布式实现上，每个操作后面其实是分布了很多个节点，每个节点上面存储了索引文件的一部分.

**但是这种方案有一些问题**：

首先，这种方案会产生密集的网络流量。尽管可以用一些高级的方法如本地缓存等，但仍然会产生大量的网络请求，因为最主要的原因是因为这种将文件拆分为块的想法与lucene索引文件构建方式和使用方式实在相隔太远，结论就是使用这种方式来做大规模索引和搜索是不切实际的。（ps：所以solandra这种玩意还是不要去考虑了）。

其次，大的索引必然会使IndexReader变的无法分布式。IndexReader是一个很重的对象，并且term词条越多，其消耗的内存也会越多。

最后，索引操作也会变的非常困难，因为只有一个单一的IndexWriter能够写索引。所以，我们把目光投向另一种方式。

* **Partitioning**

有2种通过将数据分区方式来scale搜索引擎: **基于文档（Document based partitioning）** and **基于词条（Term based partitioning）**. Elasticsearch 使用的基于文档的分区方式。

**基于文档的分区（Document Based Partitioning）**

每一个文档只存一个分区，每个分区持有整个文档集的一个子集，分区是一个功能完整的索引。

**优点：**

每个分区都可以独立的处理查询。

可以非常简单的添加以文档为单位的索引信息。

网络开销很小，每个节点可以分别执行搜索，执行完了之后只需用返回文档的ID和评分信息就可以了，然后在其中一个我们执行分布式搜索的节点上执行合并就可以了。

**缺点：**

查询如果需要在所有的分区上执行，那么它将执行 O(K\*N) 次磁盘操作（K是词条（Term，或者理解为Field）的数量，N是分区的数量）。

在实用性的角度来看基于文档的分区方式已经被证明是一个构建大型的分布式信息检索系统的一种行之有效的方法, 关于这方面的详细内容，可以看 这里 talk by Jeffrey Dean (Google)。

**基于词条的分区（Term Based Partitioning）**

每个分区拥有一部分词条，词条里面包含了整个index的文档数据。

一些基于词条分区的系统，如Riak Search (built on top of Riak key-value store engine) 或是 Lucandra/Solandra (on top of Cassandra). 尽管这些系统不是完全一样，但是它们都面临一个相似的挑战，当然也得益于相同的设计理念。

**优点：**

一般来说，你只需要在很少的部分分区上执行查询就行了，比如，我们有5个term词条的查询，我们将至多命中5个分区，如果这5个term词条都保存同一个分区中，那么我们只需用访问一个分区即可，而不管我们是不是实际上有50个分区。

另外一个优势就是对应K个Term词条的查询，你只需用执行 O(K) 次磁盘查找（假设我们使用的优化过的实现）。

**缺点：**

最主要的问题是Lucene Segment概念里面固有的很多结构都将失去。

The main problem is that whole notion of Lucene Segment which is inherent to a lot of constructs in Lucene is lost.

对于那些复杂的查询，网络开销将会变得非常高，并且可能使得系统可用性大大降低，尤其是那些会expand出大量的term词条的查询，如fuzzy或者prefix查询。

另外一个问题就是获取每个文档的信息将会变得非常困难，举例来说，如果你想获取文档的一部分数据来做进一步的控制，比如（google的PageRank算法），获取每个文档的这些数据都会变得非常困难，因为这种分区的方式使得文档的数据被分散到了不同的地方，所以实现faceting、评分、自定义评分等等都将变得难以实现。

### Replication

分布式系统的另外一方面就是复制(replication)了。通过复制我们可以得到2个主要的好处：

High Availability (HA高可用性)。如果一个节点挂了，另外一个节点能从它趴下的地方应头顶上，如果一个节点上面持有索引分片，而另一个节点持有该分片的副本，那么我们的数据就有了一个备份。

拥有数据多个副本的另一个好处就是 scalability (可伸缩性)。我们没有理由不通过增加副本来提高搜索能力，而我们只需要简单的增加几个副本或从节点（slave nodes）就能提升我们搜索的吞吐，何乐而不为呢。

一般有两种方式来实现复制: **Push Replication（推模式）** 和 **Pull Replication（拉模式）**。 Elasticsearch 使用的是Push Replication(推模式)。

* **Push Replication**

工作起来非常简单， 当你往 [master] 主分片上面索引一个文档，该分片会复制该文档(document)到剩下的所有 [replica] 副本分片中，这些分片也会索引这个文档。

**缺点：**

同一个文档重复索引多次，相比拉模式而言，要传输相对较少的数据(众所周知，Lucene索引速度非常快)。

You index the same document several times， but we transfer much less data compared to Pull replication (and Lucene is known to index very fast)。

这就需要在并发索引的时候进行一些微妙的控制，比如对同一个文档进行非常频繁的索引，在主分片和副本分片上面执行索引操作的时候，你必须保证每次更新是按照正确的顺序，或者通过版本（versioning）来拒绝旧版本的操作，而拉模式就没有这个问题。

**优点：**

一旦文档索引完毕，那么该文档在所有的分片及副本上都是立即可用的。 索引操作会等待直到确认所有的副本也执行了同样的索引操作(注意: 如果需要，你也可以使用异步复制)。 这意味着索引的实时性。 然后你只需要 refresh 一下 IndexReader 就能搜索到新的数据了。

这样的架构能让你非常方便的在节点之间进行切换，假如包含主分片（primary shard）的节点挂了，我们能够很快的进行切换，因为其它的分片和主分片都是一模一样的。

* **Pull Replication**

拉模式是一种主从方式（master – slave）(Solr 用的就是这种)。 当一个文档在master上面进行索引，并且数据通过commit操作产生了新的段文件（segment），这个时候，从节点（slave）把这些段文件（segments）拉到自己的机器然后再执行相应的刷新操作，并保证lucene能够使用这些新的数据。

**缺点：**

需要在master上面执行commit操作来产生这些段文件（segment），这样slave才能够执行pull操作。 不知道你还记不记得前面说过，lucene的commit的开销是非常大的，如果可能，commit次数越少越好。

数据的传输会有不必要的冗余。 在分布式系统里面，网络通常来说是非常宝贵的资源(如果你跑在EC2上面，那将更加宝贵，$$$) 并且最终要移动的数据会越来越多，举例来说，如果你有2个段文件，里面包含了文档，文档里面的字段都是存储的（stored fields），并且Lucene决定要合并这2个段文件，那么你也必须要传输这部分数据（合并之后的段文件），因为这是一个新的段文件，但是实际上你传输的是一份相同的数据。

这将造成一个这样的局面，所有的slaves确实是在master后面。 也可能是确实没有理由每次都进行commit或者花大量时间来传输一个大的段文件。但是至少意味着你的slave会丢失 high availability，并且不可能当成是一个实时的slave（a real time high available slave）。 实时搜索不可能存在，并且（使用拉模式）也不可能有这种1秒的刷新率，然后lucene就能实时搜索。

### Transaction Log

正如前面提到过的，索引提交（commit）的开销实在太大，但是我们又必须通过提交操作来保证数据被可靠的持久化，如果拥有数据的节点突然崩溃的话，那么最后一次提交操作之后产生的数据操作将会丢失。

* **数据可靠性（Data Persistency）**

ElasticSearch通过使用 transaction log (或预写日志(write ahead log)) 来解决这个问题，通过日志记录发生在索引上的各种操作，来保证就算没有调用commit操作也能保证数据的持久化。并且能够很自然的支持推送复制（push replication），因为我们能够让每个不同的shard都拥有 transaction log ，就算某些节点崩溃了，如果有必要，可以很轻松对日志操作进行重放（replay）。

Transaction log 周期性的将数据刷新(flushed)到磁盘，你可以通过 参数 来进行控制。 简单来说就是保存两次提交之间的连续数据操作的记录。

尽管你只运行了一个elasticsearch的服务节点（可能暂时不需要分布式），trasncation log也能够使你的es即使被强制结束进程（ “kill -9” ）也不会丢失任何数据。

当然，还不止这些！Transaction log还有一个重要的功能就是可以保证当你生成快照（ shared gateway snapshot ）、分片恢复（ peer shard recovery ）或是分片热迁移（shard “Hot” relocation）的时候，索引数据不会丢失。

* **Shared Gateway Snapshot**

使用共享gateway时，会周期性的生成数据改变(changes)的快照 ( snapshots ) ，并存储到共享存储中（shared storage)，并且transaction log也是持久化数据的一部分。

* **Peer Shard Reovery**

当分片从一个节点迁移到另一个节点或者需要分配更多的分片(比如你 增加 了副本数) 的时候，数据会从某一个节点上取来进行恢复，而不是从gateway。

迁移数据时，首先我们保证不会删除Lucene的段文件（segment files)，然后禁用flushing操作，这个时候保证不调用commit操作，然后开始迁移这些段文件，这个时候产生的索引改变，我们存放到transaction log中，一旦这个步骤结束（ie：索引索引文件拷贝完毕），我们开始对transaction log里面的日志在replica分片上进行重放操作（replay），完毕之后，我们就可以进行切换了，数据迁移成功！

迁移操作进行时，你仍然可以进行索引，仍然可以进行搜索，只有索引切换的时候会有一段很短的时间阻塞（blocking），但是直到切换前，迁移对你来说是完全透明的。

# 2、服务器搭建

<Elasticsearch集群搭建.docx>

# 3、Moduls

## 3.1、集群（Cluster）

代表一个集群，集群中有多个节点，其中有一个为主节点，这个主节点是可以通过选举产生的，主从节点是对于集群内部来说的。es的一个概念就是去中心化，字面上理解就是无中心节点，这是对于集群外部来说的，因为从外部来看es集群，在逻辑上是个整体，你与任何一个节点的通信和与整个es集群通信是等价的。

## 3.2、分片（Shards）

代表索引分片，es可以把一个完整的索引分成多个分片，这样的好处是可以把一个大的索引拆分成多个，分布到不同的节点上。构成分布式搜索。分片的数量只能在索引创建前指定，并且索引创建后不能更改。

## 3.3、副本（Replicas）

代表索引副本，es可以设置多个索引的副本，副本的作用一是提高系统的容错性，当个某个节点某个分片损坏或丢失时可以从副本中恢复。二是提高es的查询效率，es会自动对搜索请求进行负载均衡。

## 3.4、Recovery

代表数据恢复或叫数据重新分布，es在有节点加入或退出时会根据机器的负载对索引分片进行重新分配，挂掉的节点重新启动时也会进行数据恢复。

## 3.5、River

代表es的一个数据源，也是其它存储方式（如：数据库）同步数据到es的一个方法。它是以插件方式存在的一个es服务，通过读取river中的数据并把它索引到es中，官方的river有couchDB的，RabbitMQ的，Twitter的，Wikipedia的，river这个功能将会在后面的文件中重点说到。

## 3.6、Gateway

代表es索引的持久化存储方式，es默认是先把索引存放到内存中，当内存满了时再持久化到硬盘。当这个es集群关闭再重新启动时就会从gateway中读取索引数据。es支持多种类型的gateway，有本地文件系统（默认），分布式文件系统，Hadoop的HDFS和amazon的s3云存储服务。

## 3.7、discovery.zen

代表es的自动发现节点机制，es是一个基于p2p的系统，它先通过广播寻找存在的节点，再通过多播协议来进行节点之间的通信，同时也支持点对点的交互。

## 3.8、Transport

代表es内部节点或集群与客户端的交互方式，默认内部是使用tcp协议进行交互，同时它支持http协议（json格式）、thrift、servlet、memcached、zeroMQ等的传输协议（通过插件方式集成）。

# 4、Java API

## 4.1与集群交互

可以通过两种方式来连接到elasticsearch（简称es）集群，第一种是通过在你的程序中创建一个嵌入es节点（Node），使之成为es集群的一部分，然后通过这个节点来与es集群通信。第二种方式是用TransportClient这个接口和es集群通信。

两种方式的比较：

<https://www.elastic.co/guide/en/elasticsearch/guide/current/_transport_client_versus_node_client.html>

### 4.1.1、Node方式

创建嵌入节点的方式如下：

|  |
| --- |
| import static org.elasticsearch.node.NodeBuilder.\*;  //启动节点  Node node = nodeBuilder().node();  Client client = node.client();  //关闭节点  node.close(); |

当你启动一个节点，它会自动加入同网段的es集群，一个前提就是es的集群名（cluster.name）这个参数要设置一致。

默认的话启动一个节点，es集群会自动给它分配一些索引的分片，如果你想这个节点仅仅作为一个客户端而不去保存数据，你就可以设置把node.data设置成false或 node.client设置成true。下面是例子：

|  |
| --- |
| Node node = nodeBuilder().clusterName(clusterName).client(true).node(); |

还有一种情况是你并不想把节点加入集群，只想用它进行单元测试时，就要启动一个“本地”的es，这里“本地”指的是在jvm的级别下运行，即两个不同的es节点运行在同一个JVM中时会组成一个集群。它需要把节点的local参数设置成true，下面是例子：

|  |
| --- |
| Node node = nodeBuilder().local(true).node(); |

### 4.1.2、TransportClient方式

通过TransportClient这个接口，我们可以不启动节点就可以和es集群进行通信，它需要指定es集群中其中一台或多台机的ip地址和端口，例子如下：

|  |
| --- |
| Client client = new TransportClient()  .addTransportAddress(new InetSocketTransportAddress("host1", 9300))  .addTransportAddress(new InetSocketTransportAddress("host2", 9300));  client.close(); |

如果你需要更改集群名（默认是elasticsearch），需要如下设置：

|  |
| --- |
| Settings settings = ImmutableSettings.settingsBuilder()  .put("cluster.name", "myClusterName").build();  Client client = new TransportClient(settings); |

你可以设置client.transport.sniff为true来使客户端去嗅探整个集群的状态，把集群中其它机器的ip地址加到客户端中，这样做的好处是一般你不用手动设置集群里所有集群的ip到连接客户端，它会自动帮你添加，并且自动发现新加入集群的机器。代码实例如下：

|  |
| --- |
| Settings settings = ImmutableSettings.settingsBuilder()  .put("client.transport.sniff", true).build();  TransportClient client = new TransportClient(settings); |

参考资料

* 1. Elasticsearch官网：<http://www.elasticsearch.cn/guide/>
  2. <http://blog.csdn.net/u010994304/article/details/50423883>
  3. <http://blog.csdn.net/geloin/article/details/8444658>