ElasticSearch

> 参考文档：
>
> 官网：https://www.elastic.co/cn/elasticsearch/
>
> Github：https://github.com/elastic/elasticsearch
>
> 官方文档：https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html
>
> 官网入门教程（需填写邮箱）：https://www.elastic.co/cn/webinars/getting-started-elasticsearch
>
> API 文档 https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html
>
> python库： https://elasticsearch-py.readthedocs.io/en/v8.4.3/
>
> python 调用: https://www.cnblogs.com/remainsu/p/python-cha-xun-elasticsearch-chang-yong-fang-fa-qu.html
>
> 图解：https://zhuanlan.zhihu.com/p/62892586
>
> 原理图解：https://zhuanlan.zhihu.com/p/450562371
>
> 参考教程：
>
> ```
> *  https://www.cnblogs.com/coderxz/p/13268417.html
> *  https://learnku.com/docs/elasticsearch73/7.3
> *  https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html
> *  https://www.knowledgedict.com/tutorial/elasticsearch-index-mapping.html
> ```

简介

Elasticsearch 是一个 面相文档的，分布式, RESTful风格的 搜索、分析 引擎。

可用于存储，搜索，管理数据。例如：Logs 日志， 搜索后台，实时监控，终端安全数据等等。

> 百度目前广泛使用ElasticSearch作为文本数据分析，采集百度所有服务器上的各类指标数据及用户自定义数据，通过对各种数据进行多维分析展示，辅助定位分析实例异常或业务层面异常。目前覆盖百度内部20多个业务线（包括casio、云分析、网盟、预测、文库、直达号、钱包、风控等），单集群最大100台机器，200个ES节点，每天导入30TB+数据

ElasticSearch与solr的对比

- Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能;
- Solr 支持更多格式的数据，而 Elasticsearch 仅支持json文件格式；
- Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供；
- Solr 在传统的搜索应用中表现好于 Elasticsearch，但在**处理实时搜索应用时效率明显低于 Elasticsearch**

安装/部署

下载

下载地址：https://www.elastic.co/cn/downloads/elasticsearch

docker 部署：https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html

启动

解压下载包 并切换到对应目录

```
cd path/to/elasticsearch
./bin/elasticsearch
```

确认 浏览器访问

http://localhost:9200/

> **9300是tcp通信端口，es集群之间使用tcp进行通信，9200是http协议端口**

启动时遇到问题

http://www.javacui.com/tool/669.html

1. 启动报错

```
[ERROR][o.e.i.g.GeoIpDownloader ] [06A8DD4C-A847-4] exception during geoip databases
update
org.elasticsearch.ElasticsearchException: not all primary shards of [.geoip_databases]
index are active
```

conf/elasticsearch.yml，mac 为config/elasticsearch.yml增加配置

```
ingest.geoip.downloader.enabled: false
```

DOCKER

https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html

1. 为 Elasticsearch and Kibana 创建网络

   ```
   docker network create elastic
   ```

2. 启动容器与端口，留意终端输出的端口与ca证书

   ```
   docker run --name es01 --net elastic -p 9200:9200 -p 9300:9300 -it
   docker.elastic.co/elasticsearch/elasticsearch:8.5.0
   ```

3. 保存密码，如果需要重制密码，需使用`elasticsearch-reset-password`工具，在容器中的`/usr/share/elasticsearch/bin/elasticsearch-reset-password`文件夹下

```
docker exec -it es01 /usr/share/elasticsearch/bin/elasticsearch-reset-password
```

4. 复制`http_ca.crt`证书到本地

```
docker cp es01:/usr/share/elasticsearch/config/certs/http_ca.crt .
```

5. 通过证书访问服务并输入密码

```
curl --cacert http_ca.crt -u elastic https://localhost:9200
```

可视化工具

```
docker pull docker.elastic.co/kibana/kibana:8.5.0
docker run --name kib-01 --net elastic -p 5601:5601
docker.elastic.co/kibana/kibana:8.5.0
```

运行后 会出现对应的网址 http://0.0.0.0:5601/?code=535844 ，进行配置将启动es 时出现的Kibana粘贴到浏览器

如需中文 则需要进入容器后 在 config/kibana.yml 中添加

```
i18n.locale: "zh-CN"
```

配置

- 虚拟机内存修改，修改conf\jvm.option文件，避免虚拟内存过大导致程序无法启动

```
将#-Xms2g
  #-Xmx2g修改成为：
-Xms340m
-Xmx340m
否则因为虚拟机内存不够无法启动
```

- 跨域，elasticsearch-5.6.8\config\elasticsearch.yml中末尾加入：

```
http.cors.enabled: true
http.cors.allow-origin: "*"
network.host: 127.0.0.1
```

- ssl 相关，若在无ssl 证书的机器部署 需要禁用ssl 配置

```
# Enable security features
xpack.security.enabled: false

xpack.security.enrollment.enabled: false
```

```
# Enable encryption for HTTP API client connections, such as Kibana, Logstash, and
Agents
xpack.security.http.ssl:
 ``enabled: false
 ``keystore.path: certs/http.p12

# Enable encryption and mutual authentication between cluster nodes
xpack.security.transport.ssl:
 ``enabled: false
 ``verification_mode: certificate
 ``keystore.path: certs/transport.p12
 ``truststore.path: certs/transport.p12
# Create a new cluster with the current node only
# Additional nodes can still join the cluster later
cluster.initial_master_nodes: ["06A8DD4C-A847-4"]

# Allow HTTP API connections from anywhere
# Connections are encrypted and require user authentication
http.host: 0.0.0.0

# Allow other nodes to join the cluster from anywhere
# Connections are encrypted and mutually authenticated
#transport.host: 0.0.0.0
```

可视化插件

https://github.com/mobz/elasticsearch-head

核心概念

索引: 存放数据的位置

类型：定义数据结构

文档：一个文档就是一条记录

ElasticSearch	MySQL
索引(Indices)	库(Databases)
类型(Types)	表(Tables)
文档(Documents)	行(Rows)
字段(Fields)	列(Colums)
映射(Mapping)	键(Key)

index索引

一个索引就是一个拥有几分相似特征的文档的集合。一个索引由一个名字来标识（必须全部是**小写字母**的）

比如说，你可以有一个客户数据的索引，另一个产品目录的索引，还有一个订单数据的索引。

并且当我们要对对应于这个索引中的文档进行索引、搜索、更新和删除的时候，都要使用到这个名字。

在一个集群中，可以定义任意多的索引。**可类比mysql中的数据库**

type类型

在一个索引中，你可以定义一种或多种类型。一个类型是你的索引的一个逻辑上的分类/分区，其语义完全由你来定。通常，会为具有一组共同字段的文档定义一个类型。

比如说，我们假设你运营一个博客平台并且将你所有的数据存储到一个索引中。在这个索引中，你可以为用户数据定义一个类型，为博客数据定义另一个类型，当然，也可以为评论数据定义另一个类型。 **可类比mysql中的表**

Field字段

相当于是数据表的字段，对文档数据根据不同属性进行的分类标识 。

映射mapping

mapping是处理数据的方式和规则方面做一些限制，如某个字段的数据类型、默认值、分析器、是否被索引等等，这些都是映射里面可以设置的，其它就是处理es里面数据的一些使用规则设置也叫做映射，按着最优规则处理数据对性能提高很大，因此才需要建立映射，并且需要思考如何建立映射才能对性能更好。**相当于mysql中的创建表的过程，设置主键外键等等**

document文档

一个文档是一个可被索引的基础信息单元。比如，你可以拥有某一个客户的文档，某一个产品的一个文档，当然，也可以拥有某个订单的一个文档。

文档以JSON（Javascript Object Notation）格式来表示，而JSON是一个到处存在的互联网数据交互格式。在一个index/type里面，你可以存储任意多的文档。注意，尽管一个文档，物理上存在于一个索引之中，文档必须被索引/赋予一个索引的type。 **插入索引库以文档为单位，类比与数据库中的一行数据**

集群cluster

一个集群就是由一个或多个节点组织在一起，它们共同持有整个的数据，并一起提供索引和搜索功能。一个集群由一个唯一的名字标识，这个名字默认就是“elasticsearch”。这个名字是重要的，因为一个节点只能通过指定某个集群的名字，来加入这个集群。

节点node

一个节点是集群中的一个服务器，作为集群的一部分，它存储数据，参与集群的索引和搜索功能。和集群类似，一个节点也是由一个名字来标识的，默认情况下，这个名字是一个随机的漫威漫画角色的名字，这个名字会在启动的时候赋予节点。这个名字对于管理工作来说挺重要的，因为在这个管理过程中，你会去确定网络中的哪些服务器对应于Elasticsearch集群中的哪些节点。

一个节点可以通过配置集群名称的方式来加入一个指定的集群。默认情况下，每个节点都会被安排加入到一个叫做“elasticsearch”的集群中，这意味着，如果你在你的网络中启动了若干个节点，并假定它们能够相互发现彼此，它们将会自动地形成并加入到一个叫做“elasticsearch”的集群中。

在一个集群里，只要你想，可以拥有任意多个节点。而且，如果当前你的网络中没有运行任何Elasticsearch节点，这时启动一个节点，会默认创建并加入一个叫做“elasticsearch”的集群。

分片和复制 shards&replicas

一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有10亿文档的索引占据1TB的磁盘空间，而任一节点都没有这样大的磁盘空间；或者单个节点处理搜索请求，响应太慢。为了解决这个问题，Elasticsearch提供了将索引划分成多份的能力，这些份就叫做分片。当你创建一个索引的时候，你可以指定你想要的分片的数量。每个分片本身也是一个功能完善并且独立的“索引”，这个“索引”可以被放置到集群中的任何节点上。分片很重要，主要有两方面的原因： 1）允许你水平分割/扩展你的内容容量。 2）允许你在分片（潜在地，位于多个节点上）之上进行分布式的、并行的操作，进而提高性能/吞吐量。

至于一个分片怎样分布，它的文档怎样聚合回搜索请求，是完全由Elasticsearch管理的，对于作为用户的你来说，这些都是透明的。

在一个网络/云的环境里，失败随时都可能发生，在某个分片/节点不知怎么的就处于离线状态，或者由于任何原因消失了，这种情况下，有一个故障转移机制是非常有用并且是强烈推荐的。为此目的，Elasticsearch允许你创建分片的一份或多份拷贝，这些拷贝叫做复制分片，或者直接叫复制。

复制之所以重要，有两个主要原因： 在分片/节点失败的情况下，提供了高可用性。因为这个原因，注意到复制分片从不与原/主要（original/primary）分片置于同一节点上是非常重要的。扩展你的搜索量/吞吐量，因为搜索可以在所有的复制上并行运行。总之，每个索引可以被分成多个分片。一个索引也可以被复制0次（意思是没有复制）或多次。一旦复制了，每个索引就有了主分片（作为复制源的原来的分片）和复制分片（主分片的拷贝）之别。分片和复制的数量可以在索引创建的时候指定。在索引创建之后，你可以在任何时候动态地改变复制的数量，但是你事后不能改变分片的数量。

默认情况下，Elasticsearch中的每个索引被分片5个主分片和1个复制，这意味着，如果你的集群中至少有两个节点，你的索引将会有5个主分片和另外5个复制分片（1个完全拷贝），这样的话每个索引总共就有10个分片。

例

建立诗歌的引擎，诗歌拥有 诗题、作者、朝代、字数、诗内容等字段。

1. 建立一个名叫 Poems 的索引
2. 创建一个名叫 Poem 的类型，类型是通过 Mapping 来定义每个字段的类型。
3. 诗题、作者、朝代都是 keyword 类型，诗内容是 Text 类型，而字数是 Integer 类型，最后就是把数据组织成 Json 格式存放进去了

```
// 索引
pome

// 类型
"poem": {
    "properties": {
        "title": {
            "type": "keyword",
        },
        "author": {
            "type": "keyword",
        },
        "dynasty": {
```

```
            "type": "keyword"
        },
        "words": {
            "type": "integer"
        },
        "content": {
            "type": "text"
        }
    }
}

// 文档
{
  "title":"xxx",
  "author":"AXX",
  "dynasty":"G",
  "words":"20",
  "content":"content here"
}
```

其中配置为keyword 直接建立索引，text 先分词再建立反向索引

Elasticsearch 把操作都封装成了 HTTP 的 API，完成上述步骤，我们只要给 Elasticsearch 发送 HTTP 请求就行。

索引类型

> 详解：https://www.knowledgedict.com/tutorial/elasticsearch-index-mapping.html

静态/动态索引

- 静态索引

  静态映射是在创建索引时手工指定索引映射，和 SQL 中在建表语句中指定字段属性类似。

  相比动态映射，通过静态映射可以添加更详细、更精准的配置信息

- 动态索引

  动态映射是一种偷懒的方式，可直接创建索引并写入文档，文档中字段的类型是 Elasticsearch **自动识别**的，不需要在创建索引的时候设置字段的类型。

  在实际项目中，如果遇到的业务在导入数据之前不确定有哪些字段，也不清楚字段的类型是什么，使用动态映射非常合适。

  当 Elasticsearch 在文档中碰到一个以前没见过的字段时，它会利用动态映射来决定该字段的类型，并自动把该字段添加到映射中

JSON 格式的数据	自动推测的字段类型
null	没有字段被添加
true or false	boolean 类型
浮点类型数字	float 类型
数字	long 类型
JSON 对象	object 类型
数组	由数组中第一个非空值决定
string	有可能是 date 类型（若开启日期检测）、double 或 long 类型、text 类型、keyword 类型

- Index 配置mapping时 通过 `dynamic` 设置来控制是否自动新增字段，接受以下参数：
  - true：默认值为 true，自动添加字段。
  - false：忽略新的字段。
  - strict：严格模式，发现新的字段抛出异常。

字段类型

字符串类型	string、text、keyword
数字类型	long、integer、short、byte、double、float、half_float、scaled_float
日期类型	date
布尔类型	boolean
二进制类型	binary
范围类型	range

字符串

字符串是最直接的，如果在索引字符，字段就应该是 text 类型。它们也是最有趣，因为在映射中有很多选项来分析它们。解析文本、转变文本、将其分解为基本元素使得搜索更为相关，这个过程叫作*分析*。

> Elasticsearch 5.X 之后的字段类型不再支持 string，由 text 或 keyword 取代。如果仍使用 string，会给出警告。

- 类型
  - text

    如果一个字段是要被全文搜索的，比如邮件内容、产品描述、新闻内容，应该使用 text 类型。设置 text 类型以后，字段内容会被分析，在生成倒排索引以前，字符串会被分词器分成一个一个词项（term）。text 类型的字段不用于排序，且很少用于聚合（Terms Aggregation 除外）。
  - keyword

keyword 类型适用于索引结构化的字段，比如 email 地址、主机名、状态码和标签，通常用于过滤（比如，查找已发布博客中 status 属性为 published 的文章）、排序、聚合。类型为 keyword 的字段只能通过精确值搜索到，区别于 text 类型。

如上所提到的，text 类型字段可以在映射中指定相关的分析选项，通过 index 选项指定。

- **映射分析配置** 在 index 选项可以设置为 `analyzed`（默认）、`not_analyzed` 或 `no`。
  - **analyzed**

    默认情况下，index 被设置为 analyzed，并产生了如下行为：分析器将所有字符转为小写，并将字符串分解为单词。当期望每个单词完整匹配时，请使用这种选项。举个例子，如果用户搜索 “elasticsearch”，他们希望在结果列表里看到 “Principles and Practice of Elasticsearch”。
  - **not_analyzed**

    将 index 设置为 not_analyzed，将会产生相反的行为：分析过程被略过，整个字符串被当作单独的词条进行索引。当进行精准的匹配时，请使用这个选项，如搜索标签。你可能希望 “big data” 出现在搜索 “big data” 的结果中，而不是出现在搜索 “data” 的结果中。同样，对于多数的词条计数聚集，也需要这个。如果想知道最常出现的标签，可能需要 “big data” 作为一整个词条统计，而不是 “big” 和 “data” 分开统计。
  - **no**

    如果将 index 设置为 no，索引就被略过了，也没有词条产生，因此无法在那个字段上进行搜索。当无须在这个字段上搜索时，这个选项节省了存储空间，也缩短了索引和搜索的时间。例如，可以存储活动的评论。尽管存储和展示这些评论是很有价值的，但是可能并不需要搜索它们。在这种情况下，关闭那个字段的索引，使得索引的过程更快，并节省了存储空间。

数字类型

数字类型支持 byte、short、integer、long、float、double、half_float 和 scaled_float，它们的取值范围如下表：

类型	取值范围
long	-2^63 至 2^63-1
integer	-2^31 至 2^31-1
short	-32768 至 32767
byte	-128 至 127
double	64 位双精度 IEEE 754 浮点类型
float	32 位单精度 IEEE 754 浮点类型
half_float	16 位半精度 IEEE 754 浮点类型
scaled_float	缩放类型的浮点数

> 对于 float、half_float 和 scaled_float，-0.0 和 +0.0 是不同的值，使用 term 查询查找 -0.0 不会匹配 +0.0，同样 range 查询中上边界是 -0.0，也不会匹配 +0.0，下边界是 +0.0，也不会匹配 -0.0。

对于数字类型的字段，在满足需求的情况下，要尽可能选择范围小的数据类型。字段的长度越短，索引和搜索的效率越高。

处理浮点数时，优先考虑使用 scaled_float 类型。scaled_float 是通过缩放因子把浮点数变成 long 类型，Elasticsearch 底层存储的是整数类型，因为压缩整数比压缩浮点数更加节省存储空间。

日期类型

JSON 中没有日期类型，所以在 Elasticsearch 中的日期可以是以下几种形式：

- 格式化日期的字符串，如 “2015-01-01” 或 “2015/01/01 12:10:30”。
- 代表 milliseconds-since-the-epoch 的长整型数（epoch 指的是一个特定的时间：1970-01-01 00:00:00 UTC）。
- 代表 seconds-since-the-epoch 的整型数。

Elasticsearch 内部会把日期转换为 UTC（世界标准时间），并将其存储为表示 milliseconds-since-the-epoch 的长整型数，这样做的原因是和字符串相比，数值在存储和处理时更快。日期格式可以自定义，如果没有自定义，默认格式如下：

```
"strict_date_optional_time||epoch_millis"
```

默认情况下，以上 3 个文档的日期格式都可以被解析，内部存储的是毫秒计时的长整型数。

```
PUT my_index/my_type/1
{
  "dt": "2019-11-14"
}
PUT my_index/my_type/2
{
  "dt": "2019-11-14T13:16:302"
}
PUT my_index/my_type/3
{
  "dt": 1573664468000
}
```

布尔类型

如果一个字段是布尔类型，可接受的值为 `true`、`false`。Elasticsearch 5.4 版本以前，可以接受被解释为 true 或 false 的字符串和数字，5.4 版本以后只接受 true、false、"true"、"false"。

binary 类型

binary 类型接受 base64 编码的字符串，默认不存储（这里的不存储是指 store 属性取值为 false），也不可搜索。

range 类型

range 类型的使用场景包括网页中的时间选择表单、年龄范围选择表单等，range 类型支持的类型和取值范围如下表：

integer_range	-2^31 至 2^31-1
float_range	32-bit IEEE 754
long_range	-2^63 至 2^63-1
double_range	64-bit IEEE 754
date_range	64 位整数，毫秒计时

复合类型

Elasticsearch 字段类型的复合类型有**数组类型**、**对象类型**和**嵌套类型**。

分类	类型
数组类型	array
对象类型	object
嵌套类型	nested

数组类型

Elasticsearch 没有专用的数组类型，默认情况下任何字段都可以包含0个或者多个值，但是一个数组中的值必须是同一种类型。例如：

1. **字符数组**：`["one","two"]`
2. **整型数组**：`[1,3]`
3. **嵌套数组**：`[1,[2,3]]`，等价于 `[1,2,3]`
4. **对象数组**：`[{"city":"amsterdam","country":"nederland"},{"city":"brussels","country":"belgium"}]`

object 类型

JSON 本质上具有层级关系，文档包含内部对象，内部对象本身还包含内部对象.

nested 类型

nested 类型是 object 类型中的一个特例，可以让对象数组独立索引和查询。Lucene 没有内部对象的概念，所以 Elasticsearch 将对象层次扁平化，转化成字段名字和值构成的简单列表

地理类型

Elasticsearch 的地理相关类型有**地理坐标类型**和**地理图形类型**。

分类	类型
地理坐标类型	geo_point
地理图形类型	geo_shape

- geo point 类型用于存储地理位置信息的经纬度，可用于以下几种场景：
  - 查找一定范围内的地理位置。
  - 通过地理位置或者相对中心点的距离来聚合文档。
  - 把距离因素整合到文档的评分中。
  - 通过距离对文档排序。
- geo_shape 类型可以存储**一块区域**，GeoJSON 是一种对各种地理数据结构进行编码的格式，对象可以表示几何、特征或者特征集合，支持点、线、面、多点、多线、多面等几何类型

ip 类型

ip 类型的字段用于存储 IPv4 或者 IPv6 的地址。在映射中指定字段为 ip 类型的映射和查询语句如下

token_count 类型

token_count 用于统计字符串分词后的词项个数，本质上是一个整数型字段。举个例子，映射中指定 name 为 text 类型，增加 name.length 字段用于统计分词后词项的长度，类型为 token_count，分词器为标准分词器

```
properties": {
        "name": {
          "type": "text",
          "fields": {
            "length": {
              "type": "token_count",
              "analyzer": "standard"
            }
          }
        }
```

mapping 属性

属性名	描述
`type`	字段类型，常用的有 text、integer 等等。
`store`	是否存储指定字段，可选值为 `true`
`norms`	是否使用归一化因子，可选值为 `true`
`index_options`	索引选项控制添加到倒排索引（Inverted Index）的信息，这些信息用于搜索（Search）和高亮显示： `docs`：只索引文档编号(Doc Number)； `freqs`：索引文档编号和词频率（term frequency）； `positions`：索引文档编号，词频率和词位置（序号）； `offsets`：索引文档编号，词频率，词偏移量（开始和结束位置）和词位置（序号）。 默认情况下，被分析的字符串（analyzed string）字段使用 *positions*，其他字段默认使用 *docs*。 此外，需要注意的是 *index_option* 是 elasticsearch 特有的设置属性；临近搜索和短语查询时，*index_option* 必须设置为 *offsets*，同时高亮也可使用 postings highlighter。
`term_vector`	索引选项控制词向量相关信息： `no`：默认值，表示不存储词向量相关信息； `yes`：只存储词向量信息； `with_positions`：存储词项和词项位置； `with_offsets`：存储词项和字符偏移位置； `with_positions_offsets`：存储词项、词项位置、字符偏移位置。 *term_vector* 是 lucene 层面的索引设置。
`similarity`	指定文档相似度算法（也可以叫评分模型）： `BM25`：es 5 之后的默认设置。
`copy_to`	复制到自定义 _all 字段，值是数组形式，即表明可以指定多个自定义的字段。
`analyzer`	指定索引和搜索时的分析器，如果同时指定 *search_analyzer* 则搜索时会优先使用 *search_analyzer*。
`search_analyzer`	指定搜索时的分析器，搜索时的优先级最高。
`fielddata`	默认是 false，因为 doc_values 不支持 text 类型，所以有了 fielddata，fielddata 是 text 版本的 doc_values，也是为了优化字段进行排序、聚合和脚本访问。 和 doc_values 不同的是，fielddata 使用的是内存，而不是磁盘；因为 fielddata 会消耗大量的堆内存，fielddata 一旦加载到堆中，在 segment 的生命周期之内都将一致保持在堆中，所以谨慎使用。

API

调用

API 文档： https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html

Elasticsearch的接口语法

```
curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

参数	解释
VERB	适当的 HTTP 方法或谓词：GET、POST PUT HEAD 或者 DELETE
PROTOCOL	http 或者 https（如果你在 Elasticsearch 前面有一个 https 代理）
HOST	Elasticsearch 集群中任意节点的主机名，或者用 localhost 代表本地机器上的节点。
PORT	运行 Elasticsearch HTTP的端口号，默认为9200
PATH	API 的终端路径（例如 count 将返回集群中文档数量）。Path 可能包含多个组件，例如：_ cluster/ stats 和_ nodes, / stats/jvm
QUERY_STRING	任意可选的查询字符串参数（例如? pretty 将格式化地输出 JSON 返回值，使其更容易阅读）
BODY	一个 JSON 格式的请求体（如果请求需要的话） https://blog.csdn.net/qq 36639232

Python 中使用

- 安装

```
pip install elasticsearch
```

- 创建客户端

```
client = Elasticsearch("http://localhost:9200") # url of elasticsearch
# or
es = Elasticsearch(
    [
        {"host": "host1-ip-address", "port": port-number},
        {"host": "host2-ip-address", "port": port-number},
        {"host": "host3-ip-address", "port": port-number}
    ],
    http_auth=("username", "secret"),
    timeout=3600
)

es.ping() # 检测联通
es.ping() # 获取配置
```

- 关闭客户端

```
client.close()
```

索引

PUT 创建索引库index并添加映射mapping

URL

```
PUT /<index_name>
```

```
{
    "mappings": {
        "article": {
            "properties": {
                "id": {
                    "type": "long",
                    "store": true,
                    "index": "not_analyzed"
                },
                "title": {
                    "type": "text",
                    "store": true,
                    "index": "analyzed",
                    "analyzer": "standard"
                },
                "content": {
                    "type": "text",
                    "store": true,
                    "index": "analyzed",
                    "analyzer": "standard"
                }
            }
        }
    }
}
```

article：type类型；相当于这个索引库中有张表叫做article下面定义的这张表中的字段的定义，

字段默认为不索引的；

analyzer：分词器使用标准分词器

> es Root mapping definition has unsupported parameters

> https://blog.csdn.net/cristianoxm/article/details/115007035

PUT 先创建索引index，再添加mapping

```
PUT /<index>/<type>/_mapping
```

```
{
```

```
    "article": {
            "properties": {
                "id": {
                    "type": "long",
                    "store": true,
                    "index": "not_analyzed"
                },
                "title": {
                    "type": "text",
                    "store": true,
                    "index": "analyzed",
                    "analyzer": "standard"
                },
                "content": {
                    "type": "text",
                    "store": true,
                    "index": "analyzed",
                    "analyzer": "standard"
                }
            }
        }
}
```

DELETE 删除索引

```
DELETE  /<index>
```

文档

POST 创建文档/添加内容/修改内容

```
POST /<index>/<type>/<id>
```

```
{
    "id": 1,
    "title": "val",
    "content": "val"
}
```

> 其中url 中的 id 为主键，若未标注则自动生成 一般 请求体与 url 中的id 相同

DELETE 删除文档

```
DELETE /<index>/<type>/<id>
```

查询索引

https://blog.csdn.net/u014646662/article/details/89010759

https://blog.csdn.net/qdboi/article/details/109354843

聚合查询

https://www.knowledgedict.com/tutorial/elasticsearch-aggregations.html

Elasticsearch的搜索api支持**一个索引（index）的多个类型（type）查询**以及**多个索引（index）**的查询。

- GET 主键/id 查询

```
GET /<index>/<type>/<id>
```

```
res = es.search(index="test-inxex", filter_path="hits.hits._source.IPAddress",
body=body)
```

- POST 关键字查询

```
POST /<index>/<type>/_search
```

```
{
    "query": {
        "term": {
            "title": "搜"
        }
    }
}
```

- querystring查询

```
POST  /<index>/<type>/_search
```

```
{
    "query": {
        "query_string": {
            "default_field": "title",
            "query": "搜索服务器"
        }
    }
}
```

> 指定：
> 在哪个字段上进行查询；
> 要查询的内容是什么；

> 它会把查询内容先进行分词，再进行查询

URI中允许的参数：

名称	描述
q	查询字符串，映射到query_string查询
df	在查询中未定义字段前缀时使用的默认字段
analyzer	查询字符串时指定的分词器
analyze_wildcard	是否允许通配符和前缀查询，默认设置为false
batched_reduce_size	应在协调节点上一次减少的分片结果数。如果请求中潜在的分片数量很大，则应将此值用作保护机制，以减少每个搜索请求的内存开销
default_operator	默认使用的匹配运算符，可以是*AND*或者*OR*，默认是*OR*
lenient	如果设置为true，将会忽略由于格式化引起的问题（如向数据字段提供文本），默认为false
explain	对于每个hit，包含了具体如何计算得分的解释
_source	请求文档内容的参数，默认true；设置false的话，不返回*source字段，可以使用***source_include**和**_source_exclude**参数分别指定返回字段和不返回的字段
stored_fields	指定每个匹配返回的文档中的存储字段，多个用逗号分隔。不指定任何值将导致没有字段返回
sort	排序方式，可以是*fieldName*、*fieldName:asc*或者*fieldName:desc*的形式。fieldName可以是文档中的实际字段，也可以是诸如_score字段，其表示基于分数的排序。此外可以指定多个sort参数（顺序很重要）
track_scores	当排序时，若设置true，返回每个命中文档的分数
track_total_hits	是否返回匹配条件命中的总文档数，默认为true
timeout	设置搜索的超时时间，默认无超时时间
terminate_after	在达到查询终止条件之前，指定每个分片收集的最大文档数。如果设置，则在响应中多了一个terminated_early的布尔字段，以指示查询执行是否实际上已终止。默认为no terminate_after
from	从第几条（索引以0开始）结果开始返回，默认为0
size	返回命中的文档数，默认为10
search_type	搜索的方式，可以是*dfs_query_then_fetch*或*query_then_fetch*。默认为*query_then_fetch*
allow_partial_search_results	是否可以返回部分结果。如设置为false，表示如果请求产生部分结果，则设置为返回整体故障；默认为true，表示允许请求在超时或部分失败的情况下获得部分结果

分词器

```
GET /_analyze?analyzer=<analyzer>&pretty=true&text=<content>
# 标准分词器
GET /_analyze?analyzer=standard&pretty=true&text=我是程序员
```

上述分词器使用的是标准分词器，其对中文分词不是很友好

例如对我是程序员进行分词得到：期望为 `我 是 程序员` , 标准分词器为 `我 是 程 序 员`

所以需要使用中文分词器, 支持中文的分词器有很多，word分词器，庖丁解牛，Ansj分词器，IK分词器的使用。

IK 分词器

安装

1. 下载地址：https://github.com/medcl/elasticsearch-analysis-ik/releases， 注意版本信息，版本可以通过 HTTP 根目录获取
2. 解压，将解压后的elasticsearch文件夹拷贝到 \plugins下，并重命名文件夹为analysis-ik （其他名字也可以，目的是不要重名),
3. 重新启动ElasticSearch，即可加载IK分词器

使用

IK提供两种分词ik_smart和ik_max_word，其中ik_smart为最少切分，ik_max_word为最细粒度划分。

eg：

- ik_smart最少切分

```
GET   /_analyze?analyzer=ik_smart&pretty=true&text=我是程序员
```

```
"tokens":[
{"token": "我", "start_offset": 0, "end_offset": 1, "type": "CN_CHAR",…},
{"token": "是", "start_offset": 1, "end_offset": 2, "type": "CN_CHAR",…},
{"token": "程序员", "start_offset": 2, "end_offset": 5, "type": "CN_WORD",…}
]
```

- ik_max_word最细粒度划分

```
GET /_analyze?analyzer=ik_max_word&pretty=true&text=我是程序员
```

```
"tokens":[
{"token": "我", "start_offset": 0, "end_offset": 1, "type": "CN_CHAR",…},
{"token": "是", "start_offset": 1, "end_offset": 2, "type": "CN_CHAR",…},
{"token": "程序员", "start_offset": 2, "end_offset": 5, "type": "CN_WORD",…},
{"token": "程序", "start_offset": 2, "end_offset": 4, "type": "CN_WORD",…},
{"token": "员", "start_offset": 4, "end_offset": 5, "type": "CN_CHAR",…}
]
```