# 第一章 搜索引擎之elasticsearch课程介绍

## 第1集 elasticsearch的课程内容介绍

简介：讲解为什么要学这门课以及课程的学习内容

详情看视频

小D课堂 愿景："让编程不在难学，让技术与生活更加有趣" 更多教程请访问 xdclass.net

# 第二章 你了解搜索引擎吗

## 第1集 什么是全文搜索引擎?

简介：何为搜索，何为全文搜索

- 常用的搜索网站，比如百度，谷歌。


- 数据的分类
  - 结构化数据：指具有固定格式或有限长度的数据，如数据库，元数据等。
    - 对于结构化数据，我们一般都是可以通过关系型数据库(mysql，oracle等)的 table 的方式存储和搜索，也可以建立索引。通过b-tree等数据结构快速搜索数据。
  - 非结构化数据：全文数据，指不定长或无固定格式的数据，如邮件，word文档等。
    - 对于非结构化数据，也即对全文数据的搜索主要有两种方法：**顺序扫描法，全文搜索法**。

- 顺序扫描
  - 按字面意思，我们可以了解它的大概搜索方式，就是按照顺序扫描的方式查找特定的关键字。比如让你在一篇篮球新闻中，找出"科比"这个名字在哪些段落出现过。那你肯定需要从头到尾把文章阅读一遍，然后标记出关键字在哪些地方出现过。
  - 这种方法毋庸置疑是最低效的，如果文章很长，有几万字，等你阅读完这篇新闻找到"科比"这个关键字，那得花多少时间。

- 全文搜索
  - 对非结构化数据进行顺序扫描很慢，我们是否可以进行优化？把我们的非结构化数据想办法弄得有一定结构不就行了吗？将非结构化数据中的一部分信息提取出来，重新组织，使其变得有一定结构，然后对这些有一定结构的数据进行搜索，从而达到搜索相对较快的目的。这种方式就构成了全文搜索的基本思路。这部分从非结构化数据中提取出的然后重新组织的信息，我们称之**索引**。
  - 我们以NBA中国网站为例，假设我们都是篮球爱好者，并且我们是科密，那如何快速找到有关科比的新闻呢？全文搜索的方式就是，将所有新闻中所有的关键字进行提取，比如"科

比"，"詹姆斯"，"总冠军"，"MVP"等关键字，然后对这些关键字建立索引，通过索引我们就可以找到对应的该关键词出现的新闻了。

- 什么是全文搜索引擎

  根据百度百科中的定义，全文搜索引擎是目前广泛应用的主流搜索引擎。它的工作原理是计算机索引程序通过扫描文章中的每一个词，对每一个词建立一个索引，指明该词在文章中出现的次数和位置，当用户查询时，检索程序就根据事先建立的索引进行查找，并将查找的结果反馈给用户的。

- 搜索引擎
  - Lucene
  - Solr
  - Elastic search

## 第2集 为什么不用mysql做全文搜索

简介：为什么要用全文搜索引擎，而不用mysql做全文搜索呢

- 前言
  - 有人可能会问，为什么一定要用搜索引擎呢？我们的所有数据不是都可以放在数据库里吗？
  - 而且 Mysql，Oracle，SQL Server 等数据库里不是也能提供查询搜索功能，直接通过数据库查询不就可以了吗？
  - 确实，我们大部分的查询功能都可以通过数据库查询获得，如果查询效率低下，还可以通过新建数据库索引，优化SQL等方式进行提升效率，甚至通过引入缓存比如redis，memcache来加快数据的返回速度。如果数据量更大，还可以通过分库分表来分担查询压力。
  - 那为什么还要全文搜索引擎呢？我们从几个角度来说

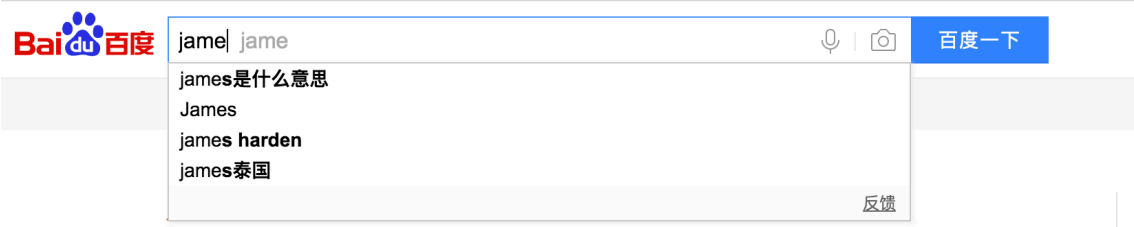
- 数据类型
  - 全文索引搜索很好的支持非结构化数据的搜索，可以更好地快速搜索大量存在的任何单词非结构化文本。例如 Google，百度类的网站搜索，它们都是根据网页中的关键字生成索引，我们在搜索的时候输入关键字，它们会将该关键字即索引匹配到的所有网页返回；还有常见的项目中应用日志的搜索等等。对于这些非结构化的数据文本，关系型数据库搜索不是能很好的支持。

- 搜索性能
  - 如果使用mysql做搜索，比如有个player表，这个表有user_name这个字段，我们要查找出user_name以james开头的球员，和含有James的球员。我们一般怎么做？数据量达到千万级别的时候怎么办？

    ```
    select * from player where user_name like 'james%';
    ```

```
select * from player where user_name like '%james%';
```

- 灵活的搜索
  - 如果我们想查出名字叫james的球员，但是用户输入了jame，我们想提示他一些关键字




  - 如果我们想查出带有"冠军"关键字的文章，但是用户输入了"总冠军"，我们也希望能查出来。


- 索引的维护
  - 一般传统数据库，全文搜索都实现的很鸡肋，因为一般也没人用数据库存长文本字段，因为进行全文搜索的时候需要扫描整个表，如果数据量大的话即使对SQL的语法进行优化，也是效果甚微。即使建立了索引，但是维护起来也很麻烦，对于 insert 和 update 操作都会重新构建索引。

- 适合全文索引引擎的场景
  - 搜索的数据对象是大量的非结构化的文本数据。
  - 文本数据量达到数十万或数百万级别，甚至更多。
  - 支持大量基于交互式文本的查询。
  - 需求非常灵活的全文搜索查询。
  - 对安全事务，非文本数据操作的需求相对较少的情况。

## 第3集 常见的搜索引擎

简介：常见的搜索引擎，Lucene，Solr，Elasticsearch

- Lucene
  - Lucene是一个Java全文搜索引擎，完全用Java编写。Lucene不是一个完整的应用程序，而是一个代码库和API，可以很容易地用于向应用程序添加搜索功能。
  - 通过简单的API提供强大的功能
    - 可扩展的高性能索引
    - 强大，准确，高效的搜索算法
    - 跨平台解决方案
  - Apache软件基金会
    - 在Apache软件基金会提供的开源软件项目的Apache社区的支持。
    - 但是Lucene只是一个框架，要充分利用它的功能，需要使用java，并且在程序中集成Lucene。需要很多的学习了解，才能明白它是如何运行的，熟练运用Lucene确实非常复杂。

- Solr
  - Solr是一个基于Lucene的Java库构建的开源搜索平台。它以用户友好的方式提供Apache Lucene的搜索功能。它是一个成熟的产品，拥有强大而广泛的用户社区。它能提供分布式索引，复制，负载均衡查询以及自动故障转移和恢复。如果它被正确部署然后管理得好，它就能够成为一个高度可靠，可扩展且容错的搜索引擎。很多互联网巨头，如Netflix，eBay，Instagram和亚马逊都使用Solr，因为它能够索引和搜索多个站点。
  - 强大的功能
    - 全文搜索
    - 突出
    - 分面搜索
    - 实时索引
    - 动态群集
    - 数据库集成
    - NoSQL功能和丰富的文档处理

- Elasticsearch
  - Elasticsearch是一个开源，是一个基于Apache Lucene库构建的Restful搜索引擎.
  - Elasticsearch是在Solr之后几年推出的。它提供了一个分布式，多租户能力的全文搜索引擎，具有HTTP Web界面（REST）和无架构JSON文档。Elasticsearch的官方客户端库提供Java，Groovy，PHP，Ruby，Perl，Python，.NET和Javascript。
  - 主要功能
    - 分布式搜索
    - 数据分析
    - 分组和聚合
  - 应用场景
    - 维基百科
    - Stack Overflow
    - GitHub

- 电商网站
- 日志数据分析
- 商品价格监控网站
- BI系统
- 站内搜索
- 篮球论坛

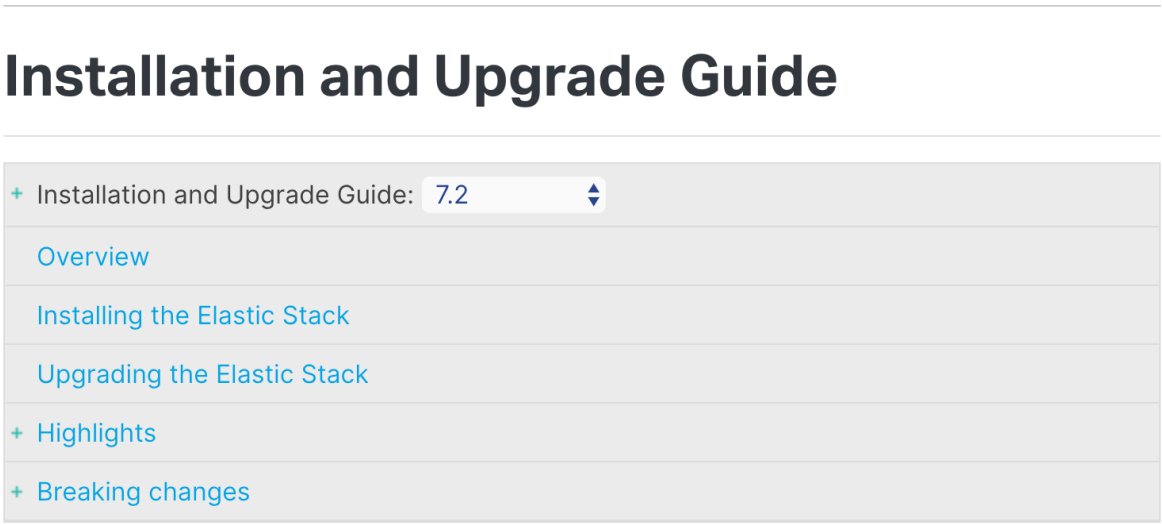


# 第三章 搜索引擎之elasticsearch的快速搭建

## 第1集 elasticsearch的快速安装

简介：手把手教你快速安装elasticsearch

- 打开官网链接 https://www.elastic.co/guide/en/elastic-stack/7.2/index.html
- 选择你要下载的版本




- 点击 installing the Elastic Stack

# Installing the Elastic Stack

When installing the Elastic Stack, you must use the same version across the entire stack. For example, if you are using Elasticsearch 7.2.1, you install Beats 7.2.1, APM Server 7.2.1, Elasticsearch Hadoop 7.2.1, Kibana 7.2.1, and Logstash 7.2.1.

If you're upgrading an existing installation, see Upgrading the Elastic Stack for information about how to ensure compatibility with 7.2.1.

## Installation Order

Install the Elastic Stack products you want to use in the following order:

1. Elasticsearch (install instructions)
2. Kibana (install)
3. Logstash (install)
4. Beats (install instructions)
5. APM Server (install instructions)
6. Elasticsearch Hadoop (install instructions)

Installing in this order ensures that the components each product depends on are in place.

- 选择第1个选项，install instructions，下载elasticsearch

## Installing Elasticsearch Yourself

Elasticsearch is provided in the following package formats:

| | |
|---|---|
| Linux and MacOS `tar.gz` archives | The `tar.gz` archives are available for installation on any Linux distribution and MacOS.<br>Install Elasticsearch from archive on Linux or MacOS |
| Windows `.zip` archive | The `zip` archive is suitable for installation on Windows.<br>Install Elasticsearch with `.zip` on Windows |
| `deb` | The `deb` package is suitable for Debian, Ubuntu, and other Debian-based systems Debian packages may be downloaded from the Elasticsearch website or from our Debian repository.<br>Install Elasticsearch with Debian Package |
| `rpm` | The `rpm` package is suitable for installation on Red Hat, Centos, SLES, OpenSuSE and other RPM-based systems. RPMs may be downloaded from the Elasticsearch website or from our RPM repository.<br>Install Elasticsearch with RPM |

- 选择你的操作系统

    - mac
        - 下载软件，安装
    - linux
        - wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.2.0-linux-x86_64.tar.gz
    - Windows
        - 使用windows不仅要注意jdk的版本和es的版本，还要注意操作系统的版本是否兼容
        - 建议最好使用虚拟机安装一个Linux，或者有经济能力可以用mac系统和购买一个阿里云服务器

- 启动
    - 我们的elasticsearch是强依赖于我们的jdk环境，所以一定要安装对应的jdk，并且配置好相关的环境变量
    - mac/linux，打开软件的安装路径，进入到bin目录，执行sh elasticsearch.sh,守护进程的方式可以使用 sh elasticsearch.sh -d -p pid
    - windows，打开软件的安装路径，进入到bin目录，双击elasticsearch.bat

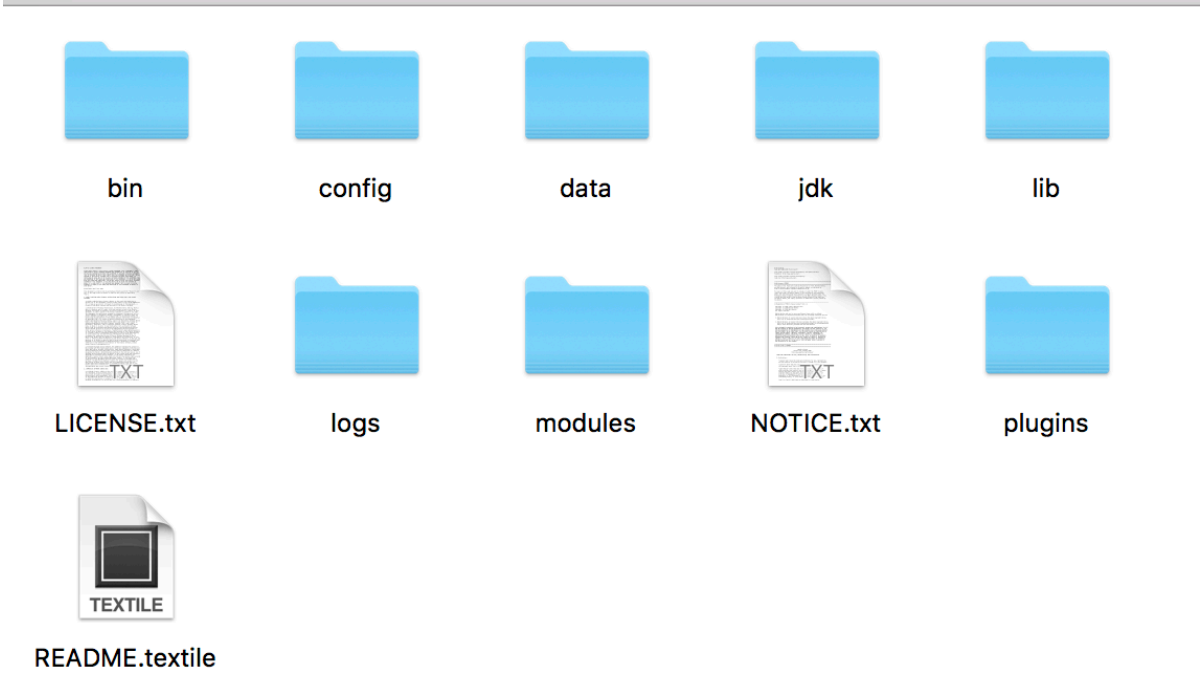
- 验证
    - 打开浏览器输入localhost:9200

## 第2集 elasticsearch之目录结构介绍

简介：elasticsearch之目录结构介绍

---

- 目录如下




- 结构如下

| 类型 | 描述 | 默认位置 | 设置 |
| --- | --- | --- | --- |
| bin | 二进制脚本包含启动节点的elasticsearch | {path.home}/bin | |
| conf | 配置文件包含elasticsearch.yml | {path.home}/config | path.conf |
| data | 在节点上申请的每个index/shard的数据文件的位置。可容纳多个位置 | {path.home}/data | path.data |
| logs | 日志文件位置 | {path.home}/logs | path.logs |
| plugins | 插件文件位置。每个插件将包含在一个子目录中。 | {path.home}/plugins | path.plugins |

# 第四章 搜索引擎之elastic search的快速入门

## 第1集 elastic search核心概念的介绍

**简介：elastic search核心概念的介绍，让你对elastic search有个大致的了解**

- 前言：

  我们在学习elastic search的核心概念之前，回顾下我们使用传统数据库查询数据的时候应该怎么做的？假设我们用使用mysql数据库存储一些数据，我们的操作步骤是怎样的？

  - 建立数据库->建表->插入数据->查询
- 索引(index)
  - 一个索引可以理解成一个关系型数据库。
- 类型(type)
  - 一种type就像一类表，比如user表，order表。
  - 注意：
    - ES 5.x中一个index可以有多种type。
    - ES 6.x中一个index只能有一种type。
    - ES 7.x以后已经移除type这个概念。
- 映射(mapping)
  - mapping定义了每个字段的类型等信息。相当于关系型数据库中的表结构。
- 文档(document)
  - 一个document相当于关系型数据库中的一行记录。
- 字段(field)
  - 相当于关系型数据库表的字段
- 集群(cluster)
  - 集群由一个或多个节点组成，一个集群有一个默认名称"elasticsearch"。
- 节点(node)
  - 集群的节点，一台机器或者一个进程
- 分片和副本(shard)
  - 副本是分片的副本。分片有主分片(primary Shard)和副本分片(replica Shard)之分。
  - 一个Index数据在物理上被分布在多个主分片中，每个主分片只存放部分数据。
  - 每个主分片可以有多个副本，叫副本分片，是主分片的复制。

# 第2集 RESTful风格的介绍

简介：RESTful风格的介绍

---

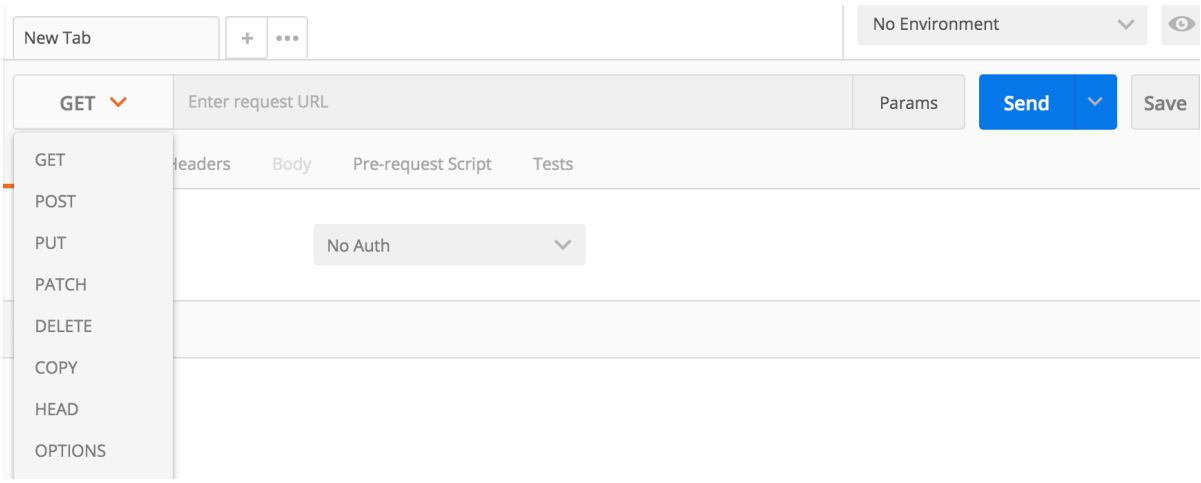
- 介绍
  - RESTful是一种架构的规范与约束、原则，符合这种规范的架构就是RESTful架构。
  - 先看REST是什么意思，英文Representational state transfer 表述性状态转移，其实就是对资源的表述性状态转移，即通过HTTP动词来实现资源的状态扭转:
  - 资源是REST系统的核心概念。 所有的设计都是以资源为中心
  - elasticsearch使用RESTful风格api来设计的

- 方法

| action | 描述 |
| --- | --- |
| HEAD | 只获取某个资源的头部信息 |
| GET | 获取资源 |
| POST | 创建或更新资源 |
| PUT | 创建或更新资源 |
| DELETE | 删除资源 |

```
GET /user:列出所有的用户
POST /user:新建一个用户
PUT /user:更新某个指定用户的信息
DELETE /user/ID:删除指定用户
```

- postman工具




- curl工具
  - 获取elasticsearch状态

```
curl -X GET "http://localhost:9200"
```

  - 新增一个文档

```
curl -X PUT "localhost:9200/xdclass/_doc/1" -H 'Content-Type:
application/json' -d'
{
    "user" : "louis",
    "message" : "louis is good"
}
'
```

  - 删除一个文档

```
curl -X DELETE "localhost:9200/xdclass/_doc/1"
```

# 第3集 索引的介绍和使用

简介：手把手进行索引的操作

---

- 新增
  - 请求

```
curl -X PUT "localhost:9200/nba"
```

  - 响应

```
{
    "acknowledged": true,
    "shards_acknowledged": true,
    "index": "nba"
}
```

- 获取
  - 请求

```
curl -X GET "localhost:9200/nba"
```

  - 响应

```
{
    "nba": {
        "aliases": {},
        "mappings": {},
        "settings": {
            "index": {
                "creation_date": "1563078001824",
                "number_of_shards": "1",
                "number_of_replicas": "1",
                "uuid": "P-kmcRGlRECcBxAI_8mgaw",
                "version": {
                    "created": "7020099"
                },
                "provided_name": "nba"
            }
```

```
        }
    }
}
```

- 删除
  - 请求

```
curl -X DELETE "localhost:9200/nba"
```

  - 响应

```
{
    "acknowledged": true
}
```

- 批量获取
  - 请求

```
curl -x GET "localhost:9200/nba,cba"
```

  - 响应

```
{
    "cba": {
        "aliases": {},
        "mappings": {},
        "settings": {
            "index": {
                "creation_date": "1563078437245",
                "number_of_shards": "1",
                "number_of_replicas": "1",
                "uuid": "3rGtXSv_QTK-GU_xHKRvmw",
                "version": {
                    "created": "7020099"
                },
                "provided_name": "cba"
            }
        }
    },
    "nba": {
        "aliases": {},
        "mappings": {},
        "settings": {
            "index": {
```

```
                "creation_date": "1563078431931",
                "number_of_shards": "1",
                "number_of_replicas": "1",
                "uuid": "bP6CZH5jSTGlhrTDzlq0bw",
                "version": {
                    "created": "7020099"
                },
                "provided_name": "nba"
            }
        }
    }
}
```

- 获取所有
  - 请求(一)

```
curl -X GET "localhost:9200/_all"
```

  - 响应(一)

```
{
    "cba": {
        "aliases": {},
        "mappings": {},
        "settings": {
            "index": {
                "creation_date": "1563078437245",
                "number_of_shards": "1",
                "number_of_replicas": "1",
                "uuid": "3rGtXSv_QTK-GU_xHKRvmw",
                "version": {
                    "created": "7020099"
                },
                "provided_name": "cba"
            }
        }
    },
    "nba": {
        "aliases": {},
        "mappings": {},
        "settings": {
            "index": {
                "creation_date": "1563078431931",
                "number_of_shards": "1",
                "number_of_replicas": "1",
                "uuid": "bP6CZH5jSTGlhrTDzlq0bw",
                "version": {
```

```
                    "created": "7020099"
                },
                "provided_name": "nba"
            }
        }
    }
}
```

- 请求(二)

```
curl -X GET "localhost:9200/_cat/indices?v"
```

- 响应(二)

```
health status index uuid                   pri rep docs.count
docs.deleted store.size pri.store.size
yellow open   nba   bP6CZH5jSTGlhrTDzlq0bw   1   1          0
   0       230b           230b
yellow open   cba   3rGtXSv_QTK-GU_xHKRvmw   1   1          0
   0       230b           230b
```

- 存在
  - 请求

```
curl -I "localhost:9200/nba"
```

  - 响应

```
200 ok
```

- 关闭
  - 请求

```
curl -X POST "localhost:9200/nba/_close"
```

  - 响应

```
{
    "acknowledged": true,
    "shards_acknowledged": true
}
```

- 打开
  - 请求

```
curl -X POST "localhost:9200/nba/_open"
```

  - 响应

```
{
    "acknowledged": true,
    "shards_acknowledged": true
}
```

## 第4集 映射的介绍和使用

简介：手把手教你怎么操作mapping

---

- 新增
  - 请求

```
curl -X PUT "localhost:9200/nba/_mapping" -H 'Content-Type:
application/json' -d'
{
```

```
  "properties": {
    "name": {
      "type": "text"
    },
     "team_name": {
      "type": "text"
    },
    "position": {
      "type": "keyword"
    },
    "play_year": {
      "type": "keyword"
    },
     "jerse_no": {
      "type": "keyword"
    }
  }
}
'
```

响应

```
{
    "acknowledged": true
}
```

- 获取
  - 请求

```
curl -X GET "localhost:9200/xdclass/_mapping"
```

  - 响应

```
{
    "nba": {
        "mappings": {
            "properties": {
                "jerse_no": {
                    "type": "keyword"
                },
                "name": {
                    "type": "text"
                },
                "play_year": {
                    "type": "keyword"
                },
```

```
                "position": {
                    "type": "keyword"
                },
                "team_name": {
                    "type": "text"
                }
            }
        }
    }
}
```

- 批量获取
  - 请求

```
curl -X GET "localhost:9200/nba,cba/mapping"
```

  - 响应

```
{
    "nba": {
        "mappings": {
            "properties": {
                "jerse_no": {
                    "type": "keyword"
                },
                "name": {
                    "type": "text"
                },
                "play_year": {
                    "type": "keyword"
                },
                "position": {
                    "type": "keyword"
                },
                "team_name": {
                    "type": "text"
                }
            }
        }
    },
    "cba": {
        "mappings": {}
    }
}
```

- 获取所有
  - 请求(一)

```
curl -X GET "localhost:9200/_mapping"
```

  - 响应(一)

```
{
    "nba": {
        "mappings": {
            "properties": {
                "jerse_no": {
                    "type": "keyword"
                },
                "name": {
                    "type": "text"
                },
                "play_year": {
                    "type": "keyword"
                },
                "position": {
                    "type": "keyword"
                },
                "team_name": {
                    "type": "text"
                }
            }
        }
    },
    "cba": {
        "mappings": {}
    }
}
```

  - 请求(二)

```
curl -X GET "localhost:9200/_all/_mapping"
```

  - 响应(二)

```
{
    "nba": {
        "mappings": {
            "properties": {
                "jerse_no": {
                    "type": "keyword"
                },
```

```
                "name": {
                    "type": "text"
                },
                "play_year": {
                    "type": "keyword"
                },
                "position": {
                    "type": "keyword"
                },
                "team_name": {
                    "type": "text"
                }
            }
        }
    },
    "cba": {
        "mappings": {}
    }
}
```

- 修改
  - 请求

```
curl -X PUT "localhost:9200/nba/_mapping" -H 'Content-Type:
application/json' -d'
{
  "properties": {
    "name": {
      "type": "text"
    },
     "team_name": {
      "type": "text"
    },
    "position": {
      "type": "keyword"
    },
    "play_year": {
      "type": "keyword"
    },
     "jerse_no": {
      "type": "keyword"
    },
     "country": {
      "type": "keyword"
    }
  }
}
```

```
'
```

响应

```
{
    "acknowledged": true
}
```

## 第5集 文档的增删改查

简介：手把手演示文档的增删改查

- 新增文档
  - PUT localhost:9200/nba/_doc/1 (指定id)

```
{

  "name":"哈登",
  "team_name":"火箭",
  "position":"得分后卫",
  "play_year":"10",
  "jerse_no":"13"

}
```

```
{
    "_index": "nba",
    "_type": "_doc",
    "_id": "1",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 0,
    "_primary_term": 1
}
```

  - POST localhost:9200/nba/_doc (不指定id)

```
{

  "name":"库里",
  "team_name":"勇士",
  "position":"组织后卫",
  "play_year":"10",
  "jerse_no":"30"

}
```

```
{
    "_index": "nba",
    "_type": "_doc",
    "_id": "cVi582sB6wrnBnZnFqog",
    "_version": 1,
    "result": "created",
    "_shards": {
```

```
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 1,
    "_primary_term": 1
}
```

- 自动创建索引
  - 查看auto_create_index开关状态，请求http://localhost:9200/_cluster/settings
  - 当索引不存在并且auto_create_index为true的时候，新增文档时会自动创建索引
  - 修改auto_create_index状态
    - PUT localhost:9200/_cluster/settings

```
{
    "persistent": {
        "action.auto_create_index": "false"
    }
}
```

```
{
    "acknowledged": true,
    "persistent": {
        "action": {
            "auto_create_index": "false"
        }
    },
    "transient": {}
}
```

  - 当auto_create_index=false时，指定一个不存在的索引，新增文档
    - PUT localhost:9200/wnba/_doc/1

```
{

  "name":"杨超越",
  "team_name":"梦之队",
  "position":"组织后卫",
  "play_year":"0",
  "jerse_no":"18"

}
```

```
{
    "error": {
        "root_cause": [
            {
                "type": "index_not_found_exception",
                "reason": "no such index [wnba]",
                "resource.type": "index_expression",
                "resource.id": "wnba",
                "index_uuid": "_na_",
                "index": "wnba"
            }
        ],
        "type": "index_not_found_exception",
        "reason": "no such index [wnba]",
        "resource.type": "index_expression",
        "resource.id": "wnba",
        "index_uuid": "_na_",
        "index": "wnba"
    },
    "status": 404
}
```


- 当auto_create_index=true时，指定一个不存在的索引，新增文档


```
{

  "name":"杨超越",
  "team_name":"梦之队",
  "position":"组织后卫",
  "play_year":"0",
  "jerse_no":"18"

}
```



```
{
    "_index": "wnba",
    "_type": "_doc",
    "_id": "1",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 0,
    "_primary_term": 1
}
```

- 指定操作类型
  - PUT localhost:9200/nba/_doc/1?op_type=create

```
{

  "name":"哈登",
  "team_name":"火箭",
  "position":"得分后卫",
  "play_year":"10",
  "jerse_no":"13"

}
```

```
{
    "error": {
        "root_cause": [
            {
                "type": "version_conflict_engine_exception",
                "reason": "[1]: version conflict, document already
exists (current version [2])",
                "index_uuid": "oPdc9qAjRO-IlzPfymnpkg",
                "shard": "0",
                "index": "nba"
            }
        ],
        "type": "version_conflict_engine_exception",
        "reason": "[1]: version conflict, document already exists
(current version [2])",
        "index_uuid": "oPdc9qAjRO-IlzPfymnpkg",
        "shard": "0",
        "index": "nba"
    },
    "status": 409
}
```

- 查看文档
  - GET localhost:9200/nba/_doc/1

```
{
    "_index": "nba",
    "_type": "_doc",
    "_id": "1",
    "_version": 3,
```

```
    "_seq_no": 3,
    "_primary_term": 1,
    "found": true,
    "_source": {
        "name": "哈登",
        "team_name": "火箭",
        "position": "得分后卫",
        "play_year": "10",
        "jerse_no": "13"
    }
}
```


- 查看多个文档
  - POST localhost:9200/_mget


```
{
    "docs" : [
        {
            "_index" : "nba",
            "_type" : "_doc",
            "_id" : "1"
        },
        {
            "_index" : "nba",
            "_type" : "_doc",
            "_id" : "2"
        }
    ]
}
```



```
{
    "docs": [
        {
            "_index": "nba",
            "_type": "_doc",
            "_id": "1",
            "_version": 3,
            "_seq_no": 3,
            "_primary_term": 1,
            "found": true,
            "_source": {
                "name": "哈登",
                "team_name": "火箭",
                "position": "得分后卫",
                "play_year": "10",
                "jerse_no": "13"
```

```
            }
        },
        {
            "_index": "nba",
            "_type": "_doc",
            "_id": "2",
            "found": false
        }
    ]
}
```

- POST localhost:9200/nba/_mget

```
{
    "docs" : [
        {
            "_type" : "_doc",
            "_id" : "1"
        },
        {
            "_type" : "_doc",
            "_id" : "2"
        }
    ]
}
```

- POST localhost:9200/nba/*doc*/mget

```
{
    "docs" : [
        {
            "_id" : "1"
        },
        {
            "_id" : "2"
        }
    ]
}
```

- GET localhost:9200/nba/*doc*/mget

```
{
    "ids" : ["1", "2"]
}
```

- 修改文档

- 根据提供的文档片段更新数据
  - POST localhost:9200/nba/_update/1

```
{
    "doc": {
        "name": "哈登",
        "team_name": "火箭",
        "position": "双能卫",
        "play_year": "10",
        "jerse_no": "13"
    }
}
```

- 向_source字段，增加一个字段
  - POST localhost:9200/nba/_update/1

```
{
    "script": "ctx._source.age = 18"
}
```

- 从_source字段，删除一个字段
  - POST localhost:9200/nba/_update/1

```
{
    "script": "ctx._source.remove(\"age\")"
}
```

- 根据参数值，更新指定文档的字段
  - POST localhost:9200/nba/_update/1

```
{
    "script": {
        "source": "ctx._source.age += params.age",
        "params": {
            "age": 4
        }
    }
}
```

  - upsert 当指定的文档不存在时，upsert参数包含的内容将会被插入到索引中，作为一个新文档；如果指定的文档存在，ElasticSearch引擎将会执行指定的更新逻辑。
- POST localhost:9200/nba/_update/3

```
{
  "script": {
        "source": "ctx._source.allstar += params.allstar",
        "params": {
            "allstar": 4
        }
    },
    "upsert": {
        "allstar": 1
    }
}
```

- 删除文档
  - DELETE localhost:9200/nba/_doc/1

# 第6集 搜索的简单使用

简介：搜索的简单使用

---

- 准备工作
  - 删掉nba索引
    - DELETE localhost:9200/nba
  - 新建一个索引,并且指定mapping
    - PUT localhost:9200/nba

```
{
    "mappings": {
        "properties": {
            "name": {
                "type": "text"
            },
            "team_name": {
                "type": "text"
            },
            "position": {
                "type": "text"
            },
            "play_year": {
                "type": "long"
            },
            "jerse_no": {
                "type": "keyword"
            }
        }
    }
}
```

  - 新增document
    - PUT localhost:9200/nba/_doc/1

```
{
    "name": "哈登",
    "team_name": "火箭",
    "position": "得分后卫",
    "play_year": 10,
    "jerse_no": "13"
}
```

    - PUT localhost:9200/nba/_doc/2

```
{
    "name": "库里",
    "team_name": "勇士",
    "position": "控球后卫",
    "play_year": 10,
    "jerse_no": "30"
}
```

    - PUT localhost:9200/nba/_doc/3

```
{
    "name": "詹姆斯",
    "team_name": "湖人",
    "position": "小前锋",
    "play_year": 15,
    "jerse_no": "23"
}
```

- term(词条)查询和full text(全文)查询
    - **词条查询：**词条查询不会分析查询条件，只有当词条和查询字符串完全匹配时，才匹配搜索。

      **全文查询：**ElasticSearch引擎会先分析查询字符串，将其拆分成多个分词，只要已分析的字段中包含词条的任意一个，或全部包含，就匹配查询条件，返回该文档；如果不包含任意一个分词，表示没有任何文档匹配查询条件
- 单条term查询
    - POST localhost:9200/nba/_search

```
{
  "query":{
    "term":{
      "jerse_no":"23"
    }
  }
}
```

- 多条term查询
    - POST localhost:9200/nba/_search

```
{
    "query": {
        "terms": {
            "jerse_no": [
                "23",
                "13"
            ]
        }
    }
}
```

```
{
    "took": 21,
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
            "value": 2,
            "relation": "eq"
        },
        "max_score": 1,
        "hits": [
            {
                "_index": "nba",
                "_type": "_doc",
                "_id": "1",
                "_score": 1,
                "_source": {
                    "name": "哈登",
                    "team_name": "火箭",
                    "position": "得分后卫",
                    "play_year": 10,
                    "jerse_no": "13"
                }
            },
            {
                "_index": "nba",
                "_type": "_doc",
                "_id": "3",
                "_score": 1,
                "_source": {
                    "name": "詹姆斯",
                    "team_name": "湖人",
```

```
                    "position": "小前锋",
                    "play_year": 15,
                    "jerse_no": "23"
                }
            }
        ]
    }
}
```

- match_all
  - POST localhost:9200/nba/_search

```
{
    "query": {
        "match_all": {}
    },
    "from": 0,
    "size": 10
}
```

```
{
    "took": 9,
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
            "value": 3,
            "relation": "eq"
        },
        "max_score": 1,
        "hits": [
            {
                "_index": "nba",
                "_type": "_doc",
                "_id": "1",
                "_score": 1,
                "_source": {
                    "name": "哈登",
                    "team_name": "火箭",
                    "position": "得分后卫",
                    "play_year": 10,
```

```
                    "jerse_no": "13"
                }
            },
            {
                "_index": "nba",
                "_type": "_doc",
                "_id": "2",
                "_score": 1,
                "_source": {
                    "name": "库里",
                    "team_name": "勇士",
                    "position": "控球后卫",
                    "play_year": 10,
                    "jerse_no": "30"
                }
            },
            {
                "_index": "nba",
                "_type": "_doc",
                "_id": "3",
                "_score": 1,
                "_source": {
                    "name": "詹姆斯",
                    "team_name": "湖人",
                    "position": "小前锋",
                    "play_year": 15,
                    "jerse_no": "23"
                }
            }
        ]
    }
}
```


- match
  - POST localhost:9200/nba/_search

```
{
    "query": {
        "match": {
            "position": "后卫"
        }
    }
}
```


```
{
    "took": 89,
```

```
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
            "value": 2,
            "relation": "eq"
        },
        "max_score": 0.90630186,
        "hits": [
            {
                "_index": "nba",
                "_type": "_doc",
                "_id": "1",
                "_score": 0.90630186,
                "_source": {
                    "name": "哈登",
                    "team_name": "火箭",
                    "position": "得分后卫",
                    "play_year": 10,
                    "jerse_no": "13"
                }
            },
            {
                "_index": "nba",
                "_type": "_doc",
                "_id": "2",
                "_score": 0.90630186,
                "_source": {
                    "name": "库里",
                    "team_name": "勇士",
                    "position": "控球后卫",
                    "play_year": 10,
                    "jerse_no": "30"
                }
            }
        ]
    }
}
```


- multi_match
    - POST localhost:9200/nba/_update/2

```
{
    "doc": {
        "name": "库里",
        "team_name": "勇士",
        "position": "控球后卫",
        "play_year": 10,
        "jerse_no": "30",
        "title": "the best shooter"
    }
}
```

- POST localhost:9200/nba/_search

```
{
    "query": {
        "multi_match": {
          "query":"shooter",
          "fields":["title","name"]
        }
    }
}
```

```
{
    "query": {
        "multi_match": {
          "query":"shooter",
          "fields":["*title","name"]
        }
    }
}
```

- match_phrase
  - post localhost:9200/nba/_search

```
{
    "query": {
        "match_phrase": {
          "position":"得分后卫"
        }
    }
}
```

```
{
    "took": 4,
    "timed_out": false,
```

```
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
            "value": 1,
            "relation": "eq"
        },
        "max_score": 3.0384295,
        "hits": [
            {
                "_index": "nba",
                "_type": "_doc",
                "_id": "1",
                "_score": 3.0384295,
                "_source": {
                    "name": "哈登",
                    "team_name": "火箭",
                    "position": "得分后卫",
                    "play_year": 10,
                    "jerse_no": "13"
                }
            }
        ]
    }
}
```


- match_phrase_prefix
  - POST localhost:9200/nba/_update/3

```
{
    "doc": {
        "name": "詹姆斯",
        "team_name": "湖人",
        "position": "小前锋",
        "play_year": 15,
        "jerse_no": "23",
        "title":"the best small forward"
    }
}
```

  - POST localhost:9200/nba/_search

```
{
    "query": {
        "match_phrase_prefix": {
          "title":"the best s"
        }
    }
}
```

## 第7集 分词器的介绍和使用

简介：分词器是什么，内置的分词器有哪些

---

- 什么是分词器
  - 将用户输入的一段文本，按照一定逻辑，分析成多个词语的一种工具
  - example: The best 3-points shooter is Curry!

- 常用的内置分词器

    - standard analyzer
    - simple analyzer
    - whitespace analyzer
    - stop analyzer
    - language analyzer
    - pattern analyzer

- standard analyzer
    - 标准分析器是默认分词器，如果未指定，则使用该分词器。
    - POST localhost:9200/_analyze

```
{
    "analyzer": "standard",
    "text": "The best 3-points shooter is Curry!"
}
```

- simple analyzer
    - simple 分析器当它遇到只要不是字母的字符，就将文本解析成term，而且所有的term都是小写的。
    - POST localhost:9200/_analyze

```
{
    "analyzer": "simple",
    "text": "The best 3-points shooter is Curry!"
}
```

- whitespace analyzer
    - whitespace 分析器，当它遇到空白字符时，就将文本解析成terms
    - POST localhost:9200/_analyze

```
{
    "analyzer": "whitespace",
    "text": "The best 3-points shooter is Curry!"
}
```

- stop analyzer
    - stop 分析器 和 simple 分析器很像，唯一不同的是，stop 分析器增加了对删除停止词的支持，默认使用了*english*停止词

- stopwords 预定义的停止词列表，比如 (the,a,an,this,of,at)等等
- POST localhost:9200/_analyze

```
{
    "analyzer": "whitespace",
    "text": "The best 3-points shooter is Curry!"
}
```

- language analyzer
  - （特定的语言的分词器，比如说，english，英语分词器),内置语言：arabic, armenian, basque, bengali, brazilian, bulgarian, catalan, cjk, czech, danish, dutch, english, finnish, french, galician, german, greek, hindi, hungarian, indonesian, irish, italian, latvian, lithuanian, norwegian, persian, portuguese, romanian, russian, sorani, spanish, swedish, turkish, thai
  - Post localhost:9200/_analyze

```
{
    "analyzer": "english",
    "text": "The best 3-points shooter is Curry!"
}
```

- pattern analyzer
  - 用正则表达式来将文本分割成terms，默认的正则表达式是\W+（非单词字符）
  - POST localhost:9200/_analyze

```
{
    "analyzer": "pattern",
    "text": "The best 3-points shooter is Curry!"
}
```

- 选择分词器
  - PUT localhost:9200/my_index

```
{
   "settings": {
        "analysis": {
            "analyzer": {
                "my_analyzer": {
                    "type": "whitespace"
                }
```

```
            }
        }
    },
    "mappings": {
        "properties": {
            "name": {
                "type": "text"
            },
            "team_name": {
                "type": "text"
            },
            "position": {
                "type": "text"
            },
            "play_year": {
                "type": "long"
            },
            "jerse_no": {
                "type": "keyword"
            },
            "title":{
              "type": "text",
              "analyzer": "my_analyzer"
            }
        }
    }
}
```

- PUT localhost:9200/my_index/_doc/1

```
 {
    "name": "库里",
    "team_name": "勇士",
    "position": "控球后卫",
    "play_year": 10,
    "jerse_no": "30",
    "title": "The best 3-points shooter is Curry!"
}
```

- POST localhost:9200/my_index/_search

```
{
  "query":{
    "match":{
      "title":"Curry!"
    }
  }
}
```

## 第8集 常见中文分词器的使用

简介：常见的中文分词器的介绍和使用

---

- 如果用默认的分词器standard
  - POST localhost:9200/_analyze

```
{
    "analyzer": "standard",
    "text": "火箭明年总冠军"
}
```

- 常见分词器
  - smartCN 一个简单的中文或中英文混合文本的分词器
  - IK分词器 更智能更友好的中文分词器

- smartCn
  - 安装 sh elasticsearch-plugin install analysis-smartcn
  - 检验
    - 安装后重新启动
    - POST localhost:9200/_analyze

```
{
    "analyzer": "smartcn",
    "text": "火箭明年总冠军"
}
```

  - 卸载 sh elasticsearch-plugin remove analysis-smartcn

- IK分词器
  - 下载 https://github.com/medcl/elasticsearch-analysis-ik/releases
  - 安装 解压安装到plugins目录
  - 检验
    - 安装后重新启动
    - POST localhost:9200/_analyze

```
{
    "analyzer": "ik_max_word",
    "text": "火箭明年总冠军"
}
```

# 第9集 常见的字段类型

简介：常见的字段类型的介绍和使用

---

- **数据类型**
  - 核心数据类型
  - 复杂数据类型
  - 专用数据类型

- **核心数据类型**
  - 字符串
    - text
      - 用于全文索引，该类型的字段将通过分词器进行分词
    - keyword
      - 不分词，只能搜索该字段的完整的值
  - 数值型
    - long, integer, short, byte, double, float, half_float, scaled_float
  - 布尔 - boolean
  - 二进制 - binary
    - 该类型的字段把值当做经过 base64 编码的字符串，默认不存储，且不可搜索
  - 范围类型
    - 范围类型表示值是一个范围，而不是一个具体的值
    - integer_range, float_range, long_range, double_range, date_range
    - 譬如 age 的类型是 integer_range，那么值可以是 {"gte" : 20, "lte" : 40}；搜索 "term" : {"age": 21} 可以搜索该值
  - 日期 - date
    - 由于Json没有date类型，所以es通过识别字符串是否符合format定义的格式来判断是否为date类型
    - format默认为：strict_date_optional_time||epoch_millis
    - 格式
      - "2022-01-01" "2022/01/01 12:10:30" 这种字符串格式

- 从开始纪元（1970年1月1日0点） 开始的毫秒数
  - 从开始纪元开始的秒数
- PUT localhost:9200/nba/_mapping

```
{
    "properties": {
      "name": {
            "type": "text"
        },
        "team_name": {
            "type": "text"
        },
        "position": {
            "type": "text"
        },
        "play_year": {
            "type": "long"
        },
        "jerse_no": {
            "type": "keyword"
        },
        "title": {
            "type": "text"
        },
        "date": {
            "type": "date"
        }
    }
}
```

- POST localhost:9200/nba/_doc/4

```
   {
        "name": "蔡x坤",
      "team_name": "勇士",
        "position": "得分后卫",
        "play_year": 10,
        "jerse_no": "31",
        "title": "打球最帅的明星",
        "date":"2020-01-01"
    }
```

- POST localhost:9200/nba/_doc/5

```
{
      "name": "杨超越",
   "team_name": "猴急",
      "position": "得分后卫",
      "play_year": 10,
      "jerse_no": "32",
      "title": "打球最可爱的明星",
      "date":1610350870
 }
```

- POST localhost:9200/nba/_doc/6

```
{
      "name": "吴亦凡",
   "team_name": "湖人",
      "position": "得分后卫",
      "play_year": 10,
      "jerse_no": "33",
      "title": "最会说唱的明星",
      "date":1641886870000
 }
```

- **复杂数据类型**
  - 数组类型 Array
    - ES中没有专门的数组类型, 直接使用[]定义即可，数组中所有的值必须是同一种数据类型, 不支持混合数据类型的数组:
    - 字符串数组 [ "one", "two" ]
    - 整数数组 [ 1, 2 ]
    - Object对象数组 [ { "name": "Louis", "age": 18 }, { "name": "Daniel", "age": 17 }]
    - 同一个数组只能存同类型的数据，不能混存，譬如 [ 10, "some string" ] 是错误的
  - 对象类型 Object
    - 对象类型可能有内部对象
    - POST localhost:9200/nba/_doc/8

```
{
    "name": "吴亦凡",
    "team_name": "湖人",
    "position": "得分后卫",
    "play_year": 10,
    "jerse_no": "33",
    "title": "最会说唱的明星",
    "date": "1641886870",
    "array": [
        "one",
```

```
            "two"
        ],
        "address": {
            "region": "China",
            "location": {
                "province": "GuangDong",
                "city": "GuangZhou"
            }
        }
}
```

- 索引方式

```
    "address.region":             "China",
    "address.location.province":  "GuangDong",
    "address.location.city":      "GuangZhou"
```

- POST localhost:9200/nba/_search

```
{
  "query":{
    "match":{
      "address.region":"china"
    }
  }
}
```

- **专用数据类型**
  - IP类型
    - IP类型的字段用于存储IPv4或IPv6的地址, 本质上是一个长整型字段.
    - POST localhost:9200/nba/_mapping

```
{
    "properties": {
        "name": {
            "type": "text"
        },
        "team_name": {
            "type": "text"
        },
        "position": {
            "type": "text"
        },
        "play_year": {
            "type": "long"
```

```
        },
        "jerse_no": {
            "type": "keyword"
        },
        "title": {
            "type": "text"
        },
        "date": {
            "type": "date"
        },
        "ip_addr": {
            "type": "ip"
        }
    }
}
```

- PUT localhost:9200/nba/_doc/9

```
{
    "name": "吴亦凡",
    "team_name": "湖人",
    "position": "得分后卫",
    "play_year": 10,
    "jerse_no": "33",
    "title": "最会说唱的明星",
    "ip_addr": "192.168.1.1"
}
```

- POST localhost:9200/nba/_search

```
{
    "query": {
        "term": {
            "ip_addr": "192.168.0.0/16" //
192.168.0.0~192.168.255.255)
        }
    }
}
```

- 官网文档
  - https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html#_complex_datatypes

# 第10集 kibana工具的安装和使用

简介：可视化工具kibana的安装和使用

---

- 下载
  - https://www.elastic.co/cn/downloads/kibana
  - 选择对应版本
- 启动
  - 进入到文件夹的bin目录，执行sh kibana
  - 访问localhost:5601
- 使用
  - 进入到Dev Tools

# 第五章 玩转elastic search的搜索

## 第1集 es之批量导入数据

简介：手把手交你批量导入数据

- Bulk
- ES提供了一个叫 bulk 的API 来进行批量操作
- 批量导入
  - 数据

```
{"index": {"_index": "book", "_type": "_doc", "_id": 1}}
{"name": "权力的游戏"}
{"index": {"_index": "book", "_type": "_doc", "_id": 2}}
{"name": "疯狂的石头"}
```

- POST bulk

```
  curl -X POST "localhost:9200/_bulk" -H 'Content-Type: application/json'
--data-binary @name
```

## 第2集 es之term的多种查询

简介：手把手带你玩转es的几种查询

- 介绍
  - 单词级别查询
  - 这些查询通常用于结构化的数据，比如：number, date, keyword等，而不是对text。
  - 也就是说，全文本查询之前要先对文本内容进行分词，而单词级别的查询直接在相应字段的反向索引中精确查找，单词级别的查询一般用于数值、日期等类型的字段上

- 准备工作
  - 删除nba索引
  - 新增nba索引

```
PUT nba
{"mappings":{"properties":{"birthDay":{"type":"date"},"birthDayStr":
{"type":"keyword"},"age":{"type":"integer"},"code":
{"type":"text"},"country":{"type":"text"},"countryEn":
{"type":"text"},"displayAffiliation":{"type":"text"},"displayName":
{"type":"text"},"displayNameEn":{"type":"text"},"draft":
{"type":"long"},"heightValue":{"type":"float"},"jerseyNo":
{"type":"text"},"playYear":{"type":"long"},"playerId":
{"type":"keyword"},"position":{"type":"text"},"schoolType":
{"type":"text"},"teamCity":{"type":"text"},"teamCityEn":
{"type":"text"},"teamConference":
{"type":"keyword"},"teamConferenceEn":{"type":"keyword"},"teamName":
{"type":"keyword"},"teamNameEn":{"type":"keyword"},"weight":
{"type":"text"}}}}
```

  - 批量导入数据(player文件)

- Term query 精准匹配查询(查找号码为23的球员)

```
POST nba/_search
{
    "query": {
      "term": {
        "jerseyNo": "23"
      }
    }
}
```

- Exsit Query 在特定的字段中查找非空值的文档(查找队名非空的球员)

```
POST nba/_search
{
    "query": {
        "exists": {
            "field": "teamNameEn"
        }
    }
}
```

- Prefix Query 查找包含带有指定前缀term的文档(查找队名以Rock开头的球员)

```
POST nba/_search
{
    "query": {
        "prefix": {
            "teamNameEn": "Rock"
        }
    }
}
```

- Wildcard Query 支持通配符查询，*表示任意字符，?表示任意单个字符(查找火箭队的球员)

```
POST nba/_search
{
    "query": {
      "wildcard": {
        "teamNameEn": "Ro*s"
      }
    }
}
```

- Regexp Query 正则表达式查询(查找火箭队的球员)

```
POST nba/_search
{
    "query": {
      "regexp": {
        "teamNameEn": "Ro.*s"
      }
    }
}
```

- Ids Query(查找id为1和2的球员)

```
POST nba/_search
{
    "query": {
      "ids": {
        "values": [1,2]
      }
    }
}
```

## 第3集 玩转es的范围查询

简介：手把手带你玩转es的范围查询

---

- 查找指定字段在指定范围内包含值（日期、数字或字符串）的文档。
  - 查找在nba打了2年到10年以内的球员

```
POST nba/_search
{
  "query": {
    "range": {
      "playYear": {
        "gte": 2,
        "lte": 10
      }
    }
  }
}
```

  - 查找1980年到1999年出生的球员

```
POST nba/_search
{
  "query": {
    "range": {
      "birthDay": {
        "gte": "01/01/1999",
        "lte": "2022",
        "format": "dd/MM/yyyy||yyyy"
      }
    }
  }
}
```

## 第4集 玩转es的布尔查询

简介：手把手带你玩转es的布尔查询

---

- 布尔查询

| type | description |
| --- | --- |
| must | 必须出现在匹配文档中 |
| filter | 必须出现在文档中，但是不打分 |
| must_not | 不能出现在文档中 |
| should | 应该出现在文档中 |

- must (查找名字叫做James的球员)

```
POST /nba/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "displayNameEn": "james"
          }
```

```
        }
      ]
    }
  }
}
```

- 效果同must，但是不打分(查找名字叫做James的球员)

```
POST /nba/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "match": {
            "displayNameEn": "james"
          }
        }
      ]
    }
  }
}
```

- must_not (查找名字叫做James的西部球员)

```
POST /nba/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "displayNameEn": "james"
          }
        }
      ],
      "must_not": [
        {
          "term": {
            "teamConferenceEn": {
              "value": "Eastern"
            }
          }
        }
      ]
    }
```

```
  }
}
```

- should(查找名字叫做James的打球时间应该在11到20年西部球员)
  - 即使匹配不到也返回，只是评分不同

```
POST /nba/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "displayNameEn": "james"
          }
        }
      ],
      "must_not": [
        {
          "term": {
            "teamConferenceEn": {
              "value": "Eastern"
            }
          }
        }
      ],
      "should": [
        {
          "range": {
            "playYear": {
              "gte": 11,
              "lte": 20
            }
          }
        }
      ]
    }
  }
}
```

  - 如果minimum_should_match=1，则变成要查出名字叫做James的打球时间在11到20年西部球员

```
POST /nba/_search
{
  "query": {
```

```
    "bool": {
      "must": [
        {
          "match": {
            "displayNameEn": "james"
          }
        }
      ],
      "must_not": [
        {
          "term": {
            "teamConferenceEn": {
              "value": "Eastern"
            }
          }
        }
      ],
      "should": [
        {
          "range": {
            "playYear": {
              "gte": 11,
              "lte": 20
            }
          }
        }
      ],
      "minimum_should_match": 1
    }
  }
}
```

## 第5集 玩转es的排序查询

简介：手把手带你玩转es的排序

---

- 火箭队中按打球时间从大到小排序的球员

```
POST nba/_search
{
  "query": {
```

```
    "match": {
      "teamNameEn": "Rockets"
    }
  },
  "sort": [
    {
      "playYear": {
        "order": "desc"
      }
    }
  ]
}
```

- 火箭队中按打球时间从大到小，如果年龄相同则按照身高从高到低排序的球员

```
POST nba/_search
{
  "query": {
    "match": {
      "teamNameEn": "Rockets"
    }
  },
  "sort": [
    {
      "playYear": {
        "order": "desc"
      }
    },
    {
      "heightValue": {
        "order": "asc"
      }
    }
  ]
}
```

# 第6集 玩转es聚合查询之指标聚合

简介：手把手玩转es的聚合查询之指标聚合

---

- ES聚合分析是什么
  - 聚合分析是数据库中重要的功能特性，完成对一个查询的数据集中数据的聚合计算，如：找出某字段（或计算表达式的结果）的最大值、最小值，计算和、平均值等。ES作为搜索引擎兼数据库，同样提供了强大的聚合分析能力。
  - 对一个数据集求最大、最小、和、平均值等指标的聚合，在ES中称为**指标聚合**
  - 而关系型数据库中除了有聚合函数外，还可以对查询出的数据进行分组group by，再在组上进行指标聚合。在ES中称为**桶聚合**

- max min sum avg
  - 求出火箭队球员的平均年龄

```
POST /nba/_search
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  },
  "aggs": {
    "avgAge": {
      "avg": {
        "field": "age"
      }
    }
  },
```

```
  "size": 0
}
```

- value_count 统计非空字段的文档数
  - 求出火箭队中球员打球时间不为空的数量

```
POST /nba/_search
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  },
  "aggs": {
    "countPlayerYear": {
      "value_count": {
        "field": "playYear"
      }
    }
  },
  "size": 0
}
```

  - 查出火箭队有多少名球员

```
POST nba/_count
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  }
}
```

- Cardinality 值去重计数
  - 查出火箭队中年龄不同的数量

```
POST /nba/_search
{
  "query": {
```

```
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  },
  "aggs": {
    "counAget": {
      "cardinality": {
        "field": "age"
      }
    }
  },
  "size": 0
}
```

- stats 统计count max min avg sum 5个值
  - 查出火箭队球员的年龄stats

```
POST /nba/_search
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  },
  "aggs": {
    "statsAge": {
      "stats": {
        "field": "age"
      }
    }
  },
  "size": 0
}
```

- Extended stats 比stats多4个统计结果： 平方和、方差、标准差、平均值加/减两个标准差的区间
  - 查出火箭队球员的年龄Extend stats

```
POST /nba/_search
{
  "query": {
    "term": {
```

```
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  },
  "aggs": {
    "extendStatsAge": {
      "extended_stats": {
        "field": "age"
      }
    }
  },
  "size": 0
}
```

- Percentiles 占比百分位对应的值统计，默认返回[ 1, 5, 25, 50, 75, 95, 99 ]分位上的值
  - 查出火箭的球员的年龄占比

```
POST /nba/_search
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  },
  "aggs": {
    "pecentAge": {
      "percentiles": {
        "field": "age"
      }
    }
  },
  "size": 0
}
```

  - 查出火箭的球员的年龄占比(指定分位值)

```
POST /nba/_search
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
```

```
  },
  "aggs": {
    "percentAge": {
      "percentiles": {
        "field": "age",
        "percents": [
          20,
          50,
          75
        ]
      }
    }
  },
  "size": 0
}
```

## 第7集 玩转es聚合查询之桶聚合

简介：手把手玩转es的聚合查询之桶聚合

---

- ES聚合分析是什么
  - 聚合分析是数据库中重要的功能特性，完成对一个查询的数据集中数据的聚合计算，如：找出某字段（或计算表达式的结果）的最大值、最小值，计算和、平均值等。ES作为搜索引擎兼数据库，同样提供了强大的聚合分析能力。
  - 对一个数据集求最大、最小、和、平均值等指标的聚合，在ES中称为**指标聚合**
  - 而关系型数据库中除了有聚合函数外，还可以对查询出的数据进行分组group by，再在组上进行指标聚合。在ES中称为**桶聚合**

- Terms Aggregation 根据字段项分组聚合
  - 火箭队根据年龄进行分组

```
POST /nba/_search
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
```

```
  },
  "aggs": {
    "aggsAge": {
      "terms": {
        "field": "age",
        "size": 10
      }
    }
  },
  "size": 0
}
```

- order 分组聚合排序
  - 火箭队根据年龄进行分组，分组信息通过年龄从大到小排序 (通过指定字段)

```
POST /nba/_search
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  },
  "aggs": {
    "aggsAge": {
      "terms": {
        "field": "age",
        "size": 10,
        "order": {
          "_key": "desc"
        }
      }
    }
  },
  "size": 0
}
```

  - 火箭队根据年龄进行分组，分组信息通过文档数从大到小排序 (通过文档数)

```
POST /nba/_search
{
  "query": {
    "term": {
      "teamNameEn": {
        "value": "Rockets"
      }
    }
  }
```

```
  },
  "aggs": {
    "aggsAge": {
      "terms": {
        "field": "age",
        "size": 10,
        "order": {
          "_count": "desc"
        }
      }
    }
  },
  "size": 0
}
```

- 每支球队按该队所有球员的平均年龄进行分组排序 (通过分组指标值)

```
POST /nba/_search
{
  "aggs": {
    "aggsTeamName": {
      "terms": {
        "field": "teamNameEn",
        "size": 30,
        "order": {
          "avgAge": "desc"
        }
      },
      "aggs": {
        "avgAge": {
          "avg": {
            "field": "age"
          }
        }
      }
    }
  },
  "size": 0
}
```

- 筛选分组聚合
  - 湖人和火箭队按球队平均年龄进行分组排序 (指定值列表)

```
POST /nba/_search
{
  "aggs": {
    "aggsTeamName": {
```

```
      "terms": {
        "field": "teamNameEn",
        "include": ["Lakers","Rockets","Warriors"],
        "exclude": ["Warriors"],
        "size": 30,
        "order": {
          "avgAge": "desc"
        }
      },
      "aggs": {
        "avgAge": {
          "avg": {
            "field": "age"
          }
        }
      }
    }
  },
  "size": 0
}
```

- 湖人和火箭队按球队平均年龄进行分组排序 (正则表达式匹配值)

```
POST /nba/_search
{
  "aggs": {
    "aggsTeamName": {
      "terms": {
        "field": "teamNameEn",
        "include": "Lakers|Ro.*|Warriors.*",
        "exclude": "Warriors",
        "size": 30,
        "order": {
          "avgAge": "desc"
        }
      },
      "aggs": {
        "avgAge": {
          "avg": {
            "field": "age"
          }
        }
      }
    }
  },
  "size": 0
}
```

- Range Aggregation 范围分组聚合
  - NBA球员年龄按20,20-35,35这样分组

```
POST /nba/_search
{
  "aggs": {
    "ageRange": {
      "range": {
        "field": "age",
        "ranges": [
          {
            "to": 20
          },
          {
            "from":20 ,
            "to": 35
          },
          {
            "from": 35
          }
        ]
      }
    }
  },
  "size": 0
}
```

  - NBA球员年龄按20,20-35,35这样分组 (起别名)

```
POST /nba/_search
{
  "aggs": {
    "ageRange": {
      "range": {
        "field": "age",
        "ranges": [
          {
            "to": 20,
            "key": "A"
          },
          {
            "from":20 ,
            "to": 35,
            "key": "B"
          },
          {
            "from": 35,
            "key": "C"
```

```
          }
        ]
      }
    }
  },
  "size": 0
}
```

- Date Range Aggregation 时间范围分组聚合
  - NBA球员按出生年月分组

```
POST /nba/_search
{
  "aggs": {
    "birthDayRange": {
      "date_range": {
        "field": "birthDay",
        "format": "MM-yyy",
        "ranges": [
          {
            "to": "01-1989"
          },
          {
            "from": "01-1989",
            "to": "01-1999"
          },
          {
            "from": "01-1999",
            "to": "01-2009"
          },
          {
            "from": "01-2009"
          }
        ]
      }
    }
  },
  "size": 0
}
```

- Date Histogram Aggregation 时间柱状图聚合
  - 按天、月、年等进行聚合统计。可按 year (1y), quarter (1q), month (1M), week (1w), day (1d), hour (1h), minute (1m), second (1s) 间隔聚合
  - NBA球员按出生年分组

```
POST /nba/_search
{
  "aggs": {
    "birthday_aggs": {
      "date_histogram": {
        "field": "birthDay",
        "format": "yyyy",
        "interval": "year"
      }
    }
  },
  "size": 0
}
```

## 第8集 es之query_string查询

简介：手把手教你es之query_string查询

---

- 介绍
  - query_string 查询，如果熟悉lucene的查询语法，我们可以直接用lucene查询语法写一个查询串进行查询，ES中接到请求后，通过查询解析器,解析查询串生成对应的查询。

- 指定单个字段查询

```
POST /nba/_search
{
  "query": {
    "query_string": {
      "default_field": "displayNameEn",
      "query": "james OR curry"
    }
  },
  "size": 100
}
```

```
POST /nba/_search
{
  "query": {
    "query_string": {
      "default_field": "displayNameEn",
      "query": "james AND harden"
    }
  },
  "size": 100
}
```

- 指定多个字段查询

```
POST /nba/_search
{
  "query": {
    "query_string": {
      "fields": [
        "displayNameEn",
        "teamNameEn"
      ],
      "query": "James AND Rockets"
    }
  },
  "size": 100
}
```

# 第六章 搜索引擎之elastic search的高级使用

## 第1集 es之索引别名的使用

简介：es之索引别名的使用

- 别名有什么用

  在开发中，随着业务需求的迭代，较老的业务逻辑就要面临更新甚至是重构，而对于es来说，为了适应新的业务逻辑，可能就要对原有的索引做一些修改，比如对某些字段做调整，甚至是重建索引。而做这些操作的时候，可能会对业务造成影响，甚至是停机调整等问题。由此，es提供了索引别名来解决这些问题。 索引别名就像一个快捷方式或是软连接，可以指向一个或多个索引，也可以给任意一个需要索引名的API来使用。别名的应用为程序提供了极大地灵活性

- 查询别名

```
GET /nba/_alias
```

```
GET /_alias
```

- 新增别名

```
POST /_aliases
{
  "actions": [
    {
      "add": {
        "index": "nba",
        "alias": "nba_v1.0"
      }
    }
  ]
}
```

```
PUT /nba/_alias/nba_v1.1
```

- 删除别名

```
POST /_aliases
{
  "actions": [
    {
      "remove": {
        "index": "nba",
        "alias": "nba_v1.0"
      }
    }
  ]
}
```

```
DELETE /nba/_alias/nba_v1.1
```

- 重命名

```
POST /_aliases
{
  "actions": [
    {
      "remove": {
        "index": "nba",
        "alias": "nba_v1.0"
      }
    },
    {
      "add": {
        "index": "nba",
```

```
        "alias": "nba_v2.0"
      }
    }
  ]
}
```

- 为多个索引指定一个别名

```
POST /_aliases
{
  "actions": [
    {
      "add": {
        "index": "nba",
        "alias": "national_player"
      }
    },
    {
      "add": {
        "index": "wnba",
        "alias": "national_player"
      }
    }
  ]
}
```

- 为同个索引指定多个别名

```
POST /_aliases
{
  "actions": [
    {
      "add": {
        "index": "nba",
        "alias": "nba_v2.1"
      }
    },
    {
      "add": {
        "index": "nba",
        "alias": "nba_v2.2"
      }
    }
  ]
}
```

- 通过别名读索引
  - 当别名指定了一个索引，则查出一个索引

```
GET /nba_v2.1
```

  - 当别名指定了多个索引，则查出多个索引

```
GET /national_player
```

- 通过别名写索引
  - 当别名指定了一个索引，则可以做写的操作

```
POST /nba_v2.1/_doc/566
{
  "countryEn": "Croatia",
  "teamName": "快船",
  "birthDay": 858661200000,
  "country": "克罗地亚",
  "teamCityEn": "LA",
  "code": "ivica_zubac",
  "displayAffiliation": "Croatia",
  "displayName": "伊维察 祖巴茨哥哥",
  "schoolType": "",
  "teamConference": "西部",
  "teamConferenceEn": "Western",
  "weight": "108.9 公斤",
  "teamCity": "洛杉矶",
  "playYear": 3,
  "jerseyNo": "40",
  "teamNameEn": "Clippers",
  "draft": 2016,
  "displayNameEn": "Ivica Zubac",
  "heightValue": 2.16,
  "birthDayStr": "1997-03-18",
  "position": "中锋",
  "age": 22,
  "playerId": "1627826"
}
```

  - 当别名指定了多个索引，可以指定写某个索引

```
POST /_aliases
```

```
{
  "actions": [
    {
      "add": {
        "index": "nba",
        "alias": "national_player",
        "is_write_index": true
      }
    },
    {
      "add": {
        "index": "wnba",
        "alias": "national_player"
      }
    }
  ]
}
```

```
POST /national_player/_doc/566
{
  "countryEn": "Croatia",
  "teamName": "快船",
  "birthDay": 858661200000,
  "country": "克罗地亚",
  "teamCityEn": "LA",
  "code": "ivica_zubac",
  "displayAffiliation": "Croatia",
  "displayName": "伊维察 祖巴茨妹妹",
  "schoolType": "",
  "teamConference": "西部",
  "teamConferenceEn": "Western",
  "weight": "108.9 公斤",
  "teamCity": "洛杉矶",
  "playYear": 3,
  "jerseyNo": "40",
  "teamNameEn": "Clippers",
  "draft": 2016,
  "displayNameEn": "Ivica Zubac",
  "heightValue": 2.16,
  "birthDayStr": "1997-03-18",
  "position": "中锋",
  "age": 22,
  "playerId": "1627826"
}
```

# 第2集 es之如何重建索引

简介：手把手教你es之如何重建索引

---

- 背景

  Elasticsearch是一个实时的分布式搜索引擎，为用户提供搜索服务，当我们决定存储某种数据时，在创建索引的时候需要将数据结构完整确定下来，于此同时索引的设定和很多固定配置将用不能改变。当需要改变数据结构时，就需要重新建立索引，为此，Elastic团队提供了很多辅助工具帮助开发人员进行重建索引。

- 步骤
  - nba取一个别名nba_latest, nba_latest作为对外使用
  - 新增一个索引nba_20220101，结构复制于nba索引，根据业务要求修改字段
  - 将nba数据同步到nba_20220101
  - 给nba_20220101添加别名nba_latest，删除nba别名nba_latest
  - 删除nba索引

- 我们对外提供访问nba索引时使用的是nba_latest别名
- 新增一个索引(比如修改字段类型，jerseyNo改成keyword类型)

```
PUT /nba_20220101
{
  "mappings": {
    "properties": {
      "age": {
        "type": "integer"
      },
      "birthDay": {
        "type": "date"
      },
      "birthDayStr": {
        "type": "keyword"
      },
      "code": {
        "type": "text"
```

```
      },
      "country": {
        "type": "keyword"
      },
      "countryEn": {
        "type": "keyword"
      },
      "displayAffiliation": {
        "type": "text"
      },
      "displayName": {
        "type": "text"
      },
      "displayNameEn": {
        "type": "text"
      },
      "draft": {
        "type": "long"
      },
      "heightValue": {
        "type": "float"
      },
      "jerseyNo": {
        "type": "keyword"
      },
      "playYear": {
        "type": "long"
      },
      "playerId": {
        "type": "keyword"
      },
      "position": {
        "type": "text"
      },
      "schoolType": {
        "type": "text"
      },
      "teamCity": {
        "type": "text"
      },
      "teamCityEn": {
        "type": "text"
      },
      "teamConference": {
        "type": "keyword"
      },
      "teamConferenceEn": {
        "type": "keyword"
      },
```

```
      "teamName": {
        "type": "keyword"
      },
      "teamNameEn": {
        "type": "keyword"
      },
      "weight": {
        "type": "text"
      }
    }
  }
}
```

- 将旧索引数据copy到新索引
  - 同步等待，接口将会在 reindex 结束后返回

```
POST /_reindex
{
  "source": {
    "index": "nba"
  },
  "dest": {
    "index": "nba_20220101"
  }
}
```

  - 异步执行，如果 reindex 时间过长，建议加上`wait_for_completion=false`的参数条件，这样 reindex 将直接返回`taskId`

```
POST /_reindex?wait_for_completion=false
{
  "source": {
    "index": "nba"
  },
  "dest": {
    "index": "nba_20220101"
  }
}
```

- 替换别名

```
POST /_aliases
{
  "actions": [
```

```
    {
      "add": {
        "index": "nba_20220101",
        "alias": "nba_latest"
      }
    },
    {
      "remove": {
        "index": "nba",
        "alias": "nba_latest"
      }
    }
  ]
}
```

- 删除旧索引

```
DELETE /nba
```

- 通过别名访问新索引

```
POST /nba_latest/_search
{
  "query": {
    "match": {
      "displayNameEn": "james"
    }
  }
}
```

## 第3集 es之refresh操作

简介：es之refresh操作

---

- 理想的搜索：

  新的数据一添加到索引中立马就能搜索到，但是真实情况不是这样的。

- 我们使用链式命令请求，先添加一个文档，再立刻搜索

```
curl -X PUT localhost:9200/star/_doc/888 -H 'Content-Type:
application/json'  -d '{ "displayName": "蔡徐坤" }'
curl  -X GET localhost:9200/star/_doc/_search?pretty
```

- 强制刷新

```
curl -X PUT localhost:9200/star/_doc/666?refresh -H 'Content-Type:
application/json'  -d '{ "displayName": "杨超越" }'
curl  -X GET localhost:9200/star/_doc/_search?pretty
```

- 修改默认更新时间(默认时间是1s)

```
PUT /star/_settings
{
  "index": {
    "refresh_interval": "5s"
  }
}
```

- 将refresh关闭

```
PUT /star/_settings
{
  "index": {
    "refresh_interval": "-1"
  }
}
```

# 第4集 es之高亮查询

简介：es之高亮查询

---

- 前言
  - 如果返回的结果集中很多符合条件的结果，那怎么能一眼就能看到我们想要的那个结果呢？比如下面网站所示的那样，我们搜索`小d课堂`，在结果集中，将所有`小d课堂`高亮显示？


- 高亮查询

```
POST /nba_latest/_search
{
  "query": {
    "match": {
      "displayNameEn": "james"
    }
  },
  "highlight": {
    "fields": {
      "displayNameEn": {}
    }
  }
}
```

- 自定义高亮查询

```
POST /nba_latest/_search
{
```

```
  "query": {
    "match": {
      "displayNameEn": "james"
    }
  },
  "highlight": {
    "fields": {
      "displayNameEn": {
        "pre_tags": ["<h1>"],
        "post_tags": ["</h1>"]
      }
    }
  }
}
```

## 第5集 es之查询建议

简介：es之查询

---

- 查询建议是什么
  - 查询建议，是为了给用户提供更好的搜索体验。包括：词条检查，自动补全。
  - 词条检查


- 自动补全


- Suggester
  - Term suggester
  - Phrase suggester
  - Completion suggester

- 字段

| | |
|---|---|
| text | 指定搜索文本 |
| field | 获取建议词的搜索字段 |
| analyzer | 指定分词器 |
| size | 每个词返回的最大建议词数 |
| sort | 如何对建议词进行排序，可用选项：<br>score: 先按评分排序、再按文档频率排、term顺序；<br>frequency: 先按文档频率排，再按评分、term顺序排。 |
| suggest_mode | 建议模式，控制提供建议词的方式：<br>missing: 仅在搜索的词项在索引中不存在时才提供建议词，默认值；<br>popular: 仅建议文档频率比搜索词项高的词 .<br>always: 总是提供匹配的建议词。 |

- Term suggester
  - term 词条建议器，对给输入的文本进行分词，为每个分词提供词项建议

```
POST /nba_latest/_search
{
  "suggest": {
    "my-suggestion": {
      "text": "jamse hardne",
      "term": {
        "suggest_mode": "missing",
        "field": "displayNameEn"
      }
    }
  }
}
```

- Phrase suggester
  - phrase 短语建议，在term的基础上，会考量多个term之间的关系，比如是否同时出现在索引的原文里，相邻程度，以及词频等

```
POST /nba_latest/_search
{
  "suggest": {
    "my-suggestion": {
      "text": "jamse harden",
      "phrase": {
        "field": "displayNameEn"
      }
    }
  }
}
```

- Completion suggester
  - Completion 完成建议

```
POST /nba_latest/_search
{
  "suggest": {
    "my-suggestion": {
      "text": "Miam",
      "completion": {
        "field": "teamCityEn"
      }
    }
  }
}
```

# 第七章 仿NBA中国官网之高级实战

## 第1集 NBA搜索实战之设计思路

简介：手把手教你设计一个NBA中国官网

- NBA中国官网：https://china.nba.com/playerindex/


- 获取数据
  - 通过chrome浏览器抓取nba球员数据
  - 将数据处理后，导入到数据库

- 项目搭建
  - spring boot整合elastic search和mysql

- 接口开发
  - 将数据库数据导入到elastic search
  - 通过姓名查找球员

- 通过国家或者球队查询球员
- 通过姓名字母查找球员

## 第2集 springboot整合elastic search和mysql

详情看视频或代码

## 第3集 elastic search之java api的使用

详情看视频或代码

## 第4集 NBA搜索实战之导入球员数据

详情看视频或代码

## 第5集 NBA搜索实战之通过姓名查找球员

详情看视频或代码

## 第6集 NBA搜索实战之通过国家或球队查找球员

详情看视频或代码

## 第7集 NBA搜索实战之通过字母查找球员

详情看视频或代码



# 第八章 走入高可用分布式集群世界

## 第1集 通往集群的大门

简介：elasticsearch为什么要集群？集群有什么作用

- 高可用

  高可用（High Availability）是分布式系统架构设计中必须考虑的因素之一，它通常是指，通过设计减少系统不能提供服务的时间。如果系统每运行100个时间单位，会有1个时间单位无法提供服务，我们说系统的可用性是99%。

- 负载均衡

  将流量均衡的分布在不同的节点上，每个节点都可以处理一部分负载，并且可以在节点之间动态分配负载，以实现平衡。

- 高性能

  将流量分发到不同机器，充分利用多机器多CPU，从串行计算到并行计算提高系统性能。

## 第2集 es集群的基本核心概念

简介：学习集群，得先了解集群它的基本核心概念

---

- Cluster 集群

  一个 Elasticsearch 集群由一个或多个节点（Node）组成，每个集群都有一个共同的集群名称作为标识。

- Node节点
  - 一个 Elasticsearch 实例即一个 Node，一台机器可以有多个实例，正常使用下每个实例应该会部署在不同的机器上。Elasticsearch 的配置文件中可以通过 node.master、node.data 来设置节点类型。
  - node.master：表示节点是否具有成为主节点的资格
    - true代表的是有资格竞选主节点
    - false代表的是没有资格竞选主节点
  - node.data：表示节点是否存储数据

- Node节点组合
  - 主节点+数据节点(master+data)
    - 节点即有成为主节点的资格，又存储数据

```
node.master: true
node.data: true
```

- 数据节点(data)
  - 节点没有成为主节点的资格，不参与选举，只会存储数据

```
node.master: false
node.data: true
```

- 客户端节点(client)
  - 不会成为主节点，也不会存储数据，主要是针对海量请求的时候可以进行负载均衡

```
node.master: false
node.data: false
```

- 分片
  - 每个索引有一个或多个分片，每个分片存储不同的数据。分片可分为主分片（ primary shard）和复制分片（replica shard），复制分片是主分片的拷贝。默认每个主分片有一个复制分片，一个索引的复制分片的数量可以动态地调整，复制分片从不与它的主分片在同一个节点上。

## 第3集 手把手教你搭建es集群

简介：通过实践，教你怎么搭建es集群

---

- 搭建步骤
  - 拷贝elasticsearch-7.2.0安装包3份，分别命名elasticsearch-7.2.0-a, elasticsearch-7.2.0-b, elasticsearch-7.2.0-c。
  - 分别修改elasticsearch.yml文件。
  - 分别启动a ,b ,c 三个节点。
  - 打开浏览器输入：http://localhost:9200/_cat/health?v ,如果返回的node.total是3，代表集群搭建成功

- 配置elasticsearch.yml文件

```
#集群名称
cluster.name: my-application

#节点名称
node.name: node-1

#是不是有资格主节点
node.master: true

#是否存储数据
node.data: true

#最大集群节点数
node.max_local_storage_nodes: 3

#网关地址
network.host: 0.0.0.0

#端口
http.port: 9200

#内部节点之间沟通端口
transport.tcp.port: 9300

#es7.x 之后新增的配置，写入候选主节点的设备地址，在开启服务后可以被选为主节点
discovery.seed_hosts: ["localhost:9300","localhost:9400","localhost:9500"]

#es7.x 之后新增的配置，初始化一个新的集群时需要此配置来选举master
cluster.initial_master_nodes: ["node-1", "node-2","node-3"]

#数据和存储路径
path.data: /Users/louis.chen/Documents/study/search/storage/a/data
path.logs:  /Users/louis.chen/Documents/study/search/storage/a/logs
```

- kibana
  - 打开配置 kibana.yml，添加elasticsearch.hosts: ["http://localhost:9200","http://localhost:9201","http://localhost:9202"]
  - 启动kibana，可以看到集群信息

## 第4集 es集群索引分片管理

简介：手把手教你索引分片管理

---

- 介绍
  - 分片(shard):因为ES是个分布式的搜索引擎, 所以索引通常都会分解成不同部分, 而这些分布在不同节点的数据就是分片. ES自动管理和组织分片, 并在必要的时候对分片数据进行再平衡分配, 所以用户基本上不用担心分片的处理细节。
  - 副本(replica):ES默认为一个索引创建1个主分片, 并分别为其创建一个副本分片. 也就是说每个索引都由1个主分片成本, 而每个主分片都相应的有一个copy.
  - Elastic search7.x之后，如果不指定索引分片，默认会创建1个主分片和一个副分片，而7.x版本之前的比如6.x版本，默认是5个主分片

- 创建索引(不指定分片数量)

```
PUT nba
{"mappings":{"properties":{"birthDay":{"type":"date"},"birthDayStr":
{"type":"keyword"},"age":{"type":"integer"},"code":
{"type":"text"},"country":{"type":"text"},"countryEn":
{"type":"text"},"displayAffiliation":{"type":"text"},"displayName":
{"type":"text"},"displayNameEn":{"type":"text"},"draft":
{"type":"long"},"heightValue":{"type":"float"},"jerseyNo":
{"type":"text"},"playYear":{"type":"long"},"playerId":
{"type":"keyword"},"position":{"type":"text"},"schoolType":
{"type":"text"},"teamCity":{"type":"text"},"teamCityEn":
{"type":"text"},"teamConference":{"type":"keyword"},"teamConferenceEn":
{"type":"keyword"},"teamName":{"type":"keyword"},"teamNameEn":
{"type":"keyword"},"weight":{"type":"text"}}}}
```

- 创建索引(指定分片数量)
  - settings

```
  "settings": {
    "number_of_shards": 3,
    "number_of_replicas": 1
  },
```

  - 创建索引

```
PUT nba
{"settings":{"number_of_shards":3,"number_of_replicas":1},"mappings":
{"properties":{"birthDay":{"type":"date"},"birthDayStr":
{"type":"keyword"},"age":{"type":"integer"},"code":
{"type":"text"},"country":{"type":"text"},"countryEn":
{"type":"text"},"displayAffiliation":{"type":"text"},"displayName":
{"type":"text"},"displayNameEn":{"type":"text"},"draft":
{"type":"long"},"heightValue":{"type":"float"},"jerseyNo":
{"type":"text"},"playYear":{"type":"long"},"playerId":
{"type":"keyword"},"position":{"type":"text"},"schoolType":
{"type":"text"},"teamCity":{"type":"text"},"teamCityEn":
{"type":"text"},"teamConference":
{"type":"keyword"},"teamConferenceEn":{"type":"keyword"},"teamName":
{"type":"keyword"},"teamNameEn":{"type":"keyword"},"weight":
{"type":"text"}}}}
```

- 索引分片分配
  - 分片分配到哪个节点是由ES自动管理的，如果某个节点挂了，那分片又会重新分配到别的节点上。
  - 在单机中，节点没有副分片，因为只有一个节点没必要生成副分片，一个节点挂点，副分片

也会挂掉，完全是单故障，没有存在的意义。
    - 在集群中，同个分片它的主分片不会和它的副分片在同一个节点上，因为主分片和副分片在同个节点，节点挂了，副分片和主分机一样是挂了，不要把所有的鸡蛋都放在同个篮子里。
    - 可以手动移动分片，比如把某个分片移动从节点1移动到节点2。
    - 创建索引时指定的主分片数以后是无法修改的，所以主分片数的数量要根据项目决定，如果真的要增加主分片只能重建索引了。副分片数以后是可以修改的。

- 手动移动分片

```
POST /_cluster/reroute
{
  "commands": [
    {
      "move": {
        "index": "nba",
        "shard": 2,
        "from_node": "node-1",
        "to_node": "node-3"
      }
    }
  ]
}
```

- 修改副分片数量

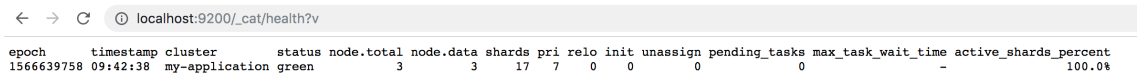
```
PUT /nba/_settings
{
  "number_of_replicas": 2
}
```

# 第5集 玩转es集群健康管理

简介：查看es集群健康的几种方式

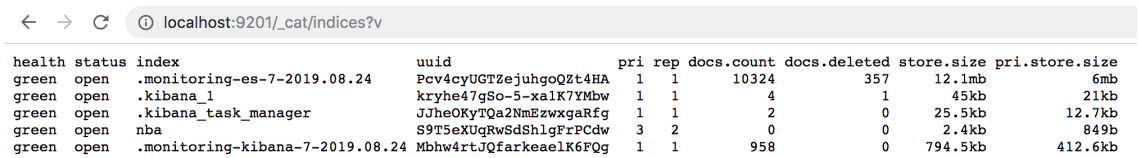
---

- 查看集群的健康状态
  - http://127.0.0.1:9200/_cat/health?v
  - URL返回了集群的健康信息。
  - 返回信息




> status ： 集群的状态，red红表示集群不可用，有故障。yellow黄表示集群不可靠但可用，一般单节点时就是此状态。green正常状态，表示集群一切正常。
>
> node.total ： 节点数，这里是3，表示该集群有三个节点。
>
> node.data ： 数据节点数，存储数据的节点数，这里是3。
>
> shards ： 表示我们把数据分成多少块存储。
>
> pri ： 主分片数，primary shards
>
> active_shards_percent ： 激活的分片百分比，这里可以理解为加载的数据分片数，只有加载所有的分片数，集群才算正常启动，在启动的过程中，如果我们不断刷新这个页面，我们会发现这个百分比会不断加大。

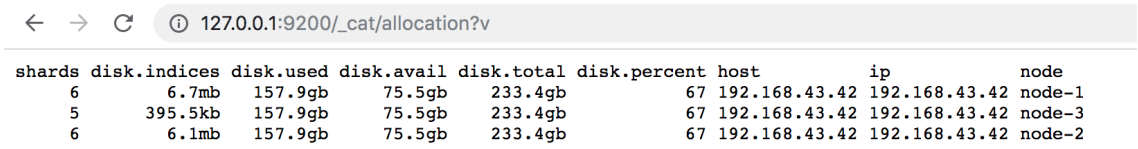
- 查看集群的索引数。
  - http://127.0.0.1:9200/_cat/indices?v
  - URL返回了集群中的所有索引信息，我们可以看到所有索引的健康情况和具体信息。
  - 返回信息




> health ： 索引健康，green为正常，yellow表示索引不可靠（单节点），red索引不可用。与集群健康状态一致。
>
> status ： 状态表明索引是否打开，只索引是可以关闭的。
>
> index ： 索引的名称

```
uuid ： 索引内部分配的名称，索引的唯一表示

pri ： 集群的主分片数量

docs.count ： 这里统计了文档的数量。

docs.deleted ： 这里统计了被删除文档的数量。

store.size ： 索引的存储的总容量

pri.store.size ： 主分别的容量
```

- 查看磁盘的分配情况
  - http://127.0.0.1:9200/_cat/allocation?v
  - URL返回了每个节点的磁盘情况。
  - 返回信息




```
shards disk.indices disk.used disk.avail disk.total disk.percent host          ip            node
     6        6.7mb   157.9gb     75.5gb    233.4gb           67 192.168.43.42 192.168.43.42 node-1
     5      395.5kb   157.9gb     75.5gb    233.4gb           67 192.168.43.42 192.168.43.42 node-3
     6        6.1mb   157.9gb     75.5gb    233.4gb           67 192.168.43.42 192.168.43.42 node-2
```

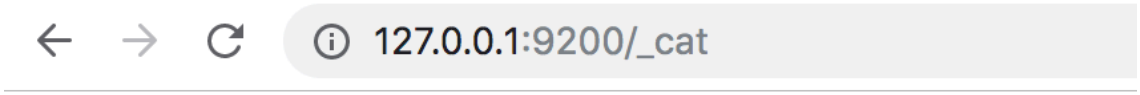
```
shards ： 该节点的分片数量

disk.indices ： 该节点中所有索引在该磁盘所点的空间。

disk.used ： 该节点已经使用的磁盘容量

disk.avail ： 该节点可以使用的磁盘容量

disk.total ： 该节点的磁盘容量
```

- 查看集群的节点信息
  - http://127.0.0.1:9200/_cat/nodes?v
  - URL返回了集群中各节点的情况。
  - 返回信息


```
ip           heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
192.168.43.42           31         100  33   30.97                  mdi       -      node-3
192.168.43.42           31         100  33   30.97                  mdi       -      node-1
192.168.43.42           29         100  33   30.97                  mdi       *      node-2
```

```
ip ： ip地址
heap.percent ： 堆内存使用情况
ram.percent ： 运行内存使用情况
cpu ： cpu使用情况
master ： 是否是主节点
```

- 查看集群的其他信息
  - http://127.0.0.1:9200/_cat




```
=^.^=
/_cat/allocation
/_cat/shards
/_cat/shards/{index}
/_cat/master
/_cat/nodes
/_cat/tasks
/_cat/indices
/_cat/indices/{index}
/_cat/segments
/_cat/segments/{index}
/_cat/count
/_cat/count/{index}
/_cat/recovery
/_cat/recovery/{index}
/_cat/health
/_cat/pending_tasks
/_cat/aliases
/_cat/aliases/{alias}
/_cat/thread_pool
/_cat/thread_pool/{thread_pools}
/_cat/plugins
/_cat/fielddata
/_cat/fielddata/{fields}
/_cat/nodeattrs
/_cat/repositories
/_cat/snapshots/{repository}
/_cat/templates
```

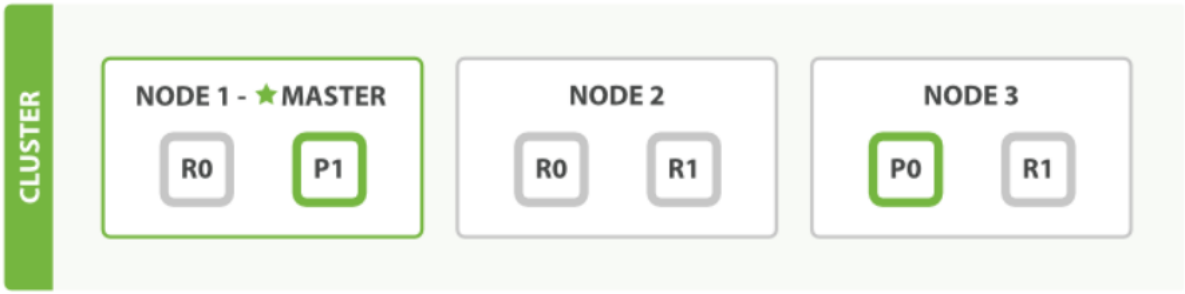
# 第九章 深入挖掘elastic search的原理

## 第1集 elastic search分布式工作原理

简介：带你剖析elastic search分布式工作原理

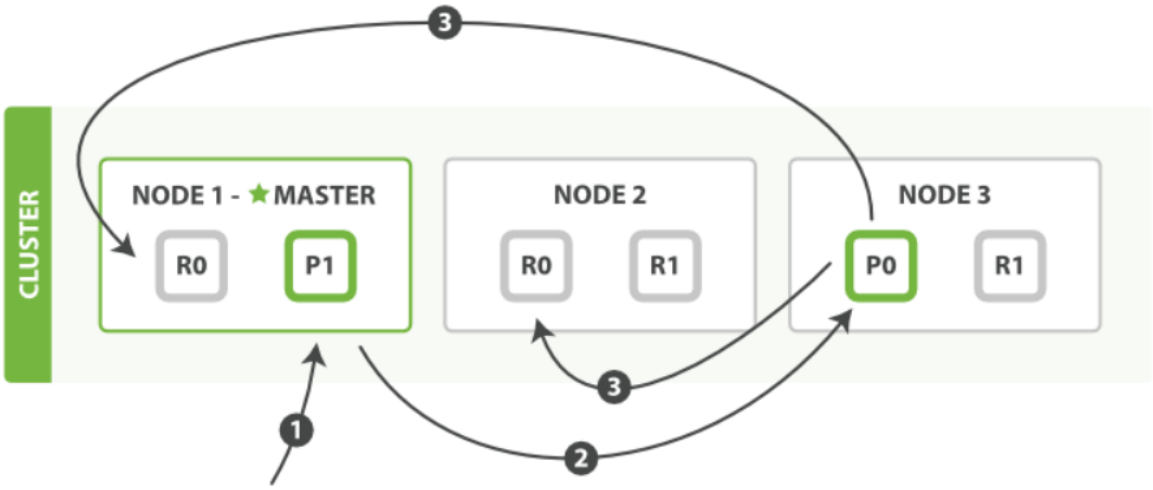
- 前言
  - Elasticsearch 是分布式的，但是对于我们开发者来说并未过多的参与其中，我们只需启动对应数量的节点，并给它们分配相同的 `cluster.name` 让它们归属于同一个集群，创建索引的时候只需指定索引主分片数和 副分片数 即可，其他的都交给了 ES 内部自己去实现。
  - 这和数据库的分布式和 同源的 solr 实现分布式都是有区别的，数据库要做集群分布式，比如分库分表需要我们指定路由规则和数据同步策略等，包括读写分离，主从同步等，solr的分布式也需依赖 zookeeper，但是 Elasticsearch 完全屏蔽了这些。
  - 虽然Elasticsearch 天生就是分布式的，并且在设计时屏蔽了分布式的复杂性，但是我们还得知道它内部的原理。

- 节点交互原理




  - es和其他中间件一样，比如mysql，redis有master-slave模式。es集群也会选举一个节点做为master节点
  - master节点它的职责是维护全局集群状态，在节点加入或离开集群的时候重新分配分片。
  - 所有文档级别的写操作不会与master节点通信，master节点并不需要涉及到文档级别的变更和搜索等操作，es分布式不太像mysql的master-slave模式，mysql是写在主库，然后再同步数据到从库。而es文档写操作是分片上而不是节点上，先写在主分片，主分片再同步给副分片，因为主分片可以分布在不同的节点上，所以当集群只有一个master节点的情况下，即使流量的增加它也不会成为瓶颈，就算它挂了，任何节点都有机会成为主节点。
  - 读写可以请求任意节点，节点再通过转发请求到目的节点，比如一个文档的新增，文档通过路由算法分配到某个主分片，然后找到对应的节点，将数据写入到主分片上，然后再同步到副分片上。

- 写入文档




1、客户端向node-1发送新增文档请求。

2、节点通过文档的路由算法确定该文档属于主分片-P0。因为主分片-P0在node-3，所以请求会转发到node-3。

3、文档在node-3的主分片-P0上新增，新增成功后，将请求转发到node-1和node-2对应的副分片-R0上。一旦所有的副分片都报告成功，node-3向node-1报告成功，node-1向客户端报告成功。

- 读取文档
  - 客户端向node-1发送读取文档请求。
  - 在处理读取请求时，node-1在每次请求的时候都会通过轮询所有的副本分片来达到负载均衡。

## 第2集 elastic search文档的路由原理

简介：当新增一个文档时，这个文档会存放在那个分片中呢?

---

- 前言
  - 当新增一个文档的时候，文档会被存储到一个主分片中。 Elasticsearch 如何知道一个文档应该存放到哪个分片中呢？当我们创建文档时，它如何决定这个文档应当被存储在分片 1 还是分片 2 中呢?

- 路由算法
  - 首先这肯定不会是随机的，否则将来要获取文档的时候我们就不知道从何处寻找了。实际上，这个过程是根据下面这个公式决定的：

```
shard = hash(routing) % number_of_primary_shards
```

- `routing` 是一个可变值，默认是文档的 `_id` ，也可以设置成一个自定义的值。 `routing` 通过 `hash` 函数生成一个数字，然后这个数字再除以 `number_of_primary_shards` （主分片的数量）后得到 余数 。这个分布在 0 到 `number_of_primary_shards-1` 之间的余数，就是我们所寻求的文档所在分片的位置。
- 这就解释了为什么我们要在创建索引的时候就确定好主分片的数量 并且永远不会改变这个数量：因为如果数量变化了，那么所有之前路由的值都会无效，文档也再也找不到了。
- 新增一个文档(指定id)

```
PUT /nba/_doc/1
{
  "name": "哈登",
  "team_name": "火箭",
  "position": "得分后卫",
  "play_year": "10",
  "jerse_no": "13"
}
```

- 查看文档在哪个分片上

```
GET /nba/_search_shards?routing=1
```

```
{
  "nodes" : {
    "V1JO7QXLSX-yeVI82WkgtA" : {
      "name" : "node-1",
      "ephemeral_id" : "_d96PgOSTnKo6nrJVqIYpw",
      "transport_address" : "192.168.1.101:9300",
      "attributes" : {
        "ml.machine_memory" : "8589934592",
        "xpack.installed" : "true",
        "ml.max_open_jobs" : "20"
      }
    },
    "z65Hwe_RR_efA4yj3n8sHQ" : {
      "name" : "node-3",
      "ephemeral_id" : "MOE_Ne7ZRyaKRHFSWJZWpA",
      "transport_address" : "192.168.1.101:9500",
      "attributes" : {
        "ml.machine_memory" : "8589934592",
        "ml.max_open_jobs" : "20",
        "xpack.installed" : "true"
      }
    }
```

```
  },
  "indices" : {
    "nba" : { }
  },
  "shards" : [
    [
      {
        "state" : "STARTED",
        "primary" : true,
        "node" : "V1JO7QXLSX-yeVI82WkgtA",
        "relocating_node" : null,
        "shard" : 2,
        "index" : "nba",
        "allocation_id" : {
          "id" : "leX_k6McShyMoM1eNQJXOA"
        }
      },
      {
        "state" : "STARTED",
        "primary" : false,
        "node" : "z65Hwe_RR_efA4yj3n8sHQ",
        "relocating_node" : null,
        "shard" : 2,
        "index" : "nba",
        "allocation_id" : {
          "id" : "6sUSANMuSGKLgcIpBa4yYg"
        }
      }
    ]
  ]
}
```


## 第3集 剖析elastic search的乐观锁

简介：剖析elastic search的乐观锁

---

- 锁的简单分类
  - 悲观锁

    顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞，直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

- 乐观锁 顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，比如可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，因为我们elasticsearch一般业务场景都是写少读多，所以通过乐观锁可以在控制并发的情况下又能有效的提高系统吞吐量。

- 版本号乐观锁
  - Elasticsearch 中对文档的 index， GET 和 delete 请求时，都会返回一个 _version，当文档被修改时版本号递增。
  - 所有文档的更新或删除 API，都可以接受 `version` 参数，这允许你在代码中使用乐观的并发控制，这里要注意的是版本号要大于旧的版本号，并且加上version_type=external。
  - 获取文档

```
GET /nba/_doc/1
```

```
{
  "_index" : "nba",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 4,
  "_primary_term" : 7,
  "found" : true,
  "_source" : {
    "name" : "哈登",
    "team_name" : "火箭",
    "position" : "得分后卫",
    "play_year" : "10",
    "jerse_no" : "13"
  }
}
```

  - 通过版本号新增文档(version要大于旧的version)

```
POST /nba/_doc/1?version=2&version_type=external
{
  "name": "哈登",
  "team_name": "火箭",
  "position": "得分后卫",
  "play_year": "10",
  "jerse_no": "13"
}
```

# 第4集 倒排索引到底是什么

简介：带你分析倒排索引的原理

- 我们打开NBA中国官网，搜索james得到以下结果


- 假设文档集合如下图所示

| 文档编号 | 文档内容 |
| --- | --- |
| 1 | James Ennis |
| 2 | James Harden |
| 3 | Justin James |
| 4 | LeBron James |
| 5 | James Johnson |

  - 我们是怎么通过james查找到名字带有james的球员呢?
  - 如果按照这个图，我们是不是得把这5个文档遍历一遍，把文档带有james的球员查找出来?
  - 如果按照这种顺序扫描，那每次输入不同的关键字，岂不是要从头到尾遍历一遍?

- 假设文档集合如下图所示

| 单词序号 | 单词 | 倒排列表 |
| --- | --- | --- |
| 1 | james | 1，2，3，4，5 |
| 2 | ennis | 1 |
| 3 | harden | 2 |
| 4 | justin | 3 |
| 5 | leBron | 4 |
| 6 | johnson | 5 |

- 我们把这个5个球员的名字进行分词，每个分词转成小写字母，并且以每个分词分组，统计它所在文档的位置。
- 当有关键字请求过来的时候，将关键字转成小写，查找出关键字匹配到的文档位置，然后全部返回。

- 完善倒排索引

| 单词序号 | 单词 | 文档频率 | 倒排列表 (DocID;TF;<POS>) |
|---|---|---|---|
| 1 | james | 5 | (1;1;<1>), (2;1;<1>), (3;1;<2>), (4;1;<2>), (5;1;<1>) |
| 2 | ennis | 1 | (1;1;<1>) |
| 3 | harden | 1 | (2;1;<2>) |
| 4 | justin | 1 | (3;1;<1>) |
| 5 | leBron | 1 | (4;1;<1>) |
| 6 | johnson | 1 | (5;1;<2>) |

  - 参数解释
    - DocId：单词出现的文档id
    - TF：单词在某个文档中出现的次数
    - POS：单词在文档中出现的位置

## 第5集 浅谈elastic search的分词原理

简介：谈谈elasticsearch的分词原理

---

- 前言一
  - 我们创建一个文档

```
PUT test/_doc/1
{
  "msg":"乔丹是篮球之神"
}
```

  - 我们通过'乔丹'这个关键词来搜索这个文档

```
POST /test/_search
{
  "query": {
    "match": {
      "msg": "乔丹"
    }
  }
}
```

- 我们发现能匹配文档出来，那整一个过程的原理是怎样的呢?

- 前言二
  - 我们来试下使用中文分词器

```
PUT test/_mapping
{
  "properties": {
    "msg_chinese":{
      "type":"text",
      "analyzer": "ik_max_word"
    }
  }
}
```

```
POST test/_doc/1
{
  "msg":"乔丹是篮球之神",
  "msg_chinese":"乔丹是篮球之神"
}
```

```
POST /test/_search
{
  "query": {
    "match": {
      "msg_chinese": "乔"
    }
  }
}
```

```
POST /test/_search
{
  "query": {
    "match": {
      "msg": "乔"
    }
  }
}
```

- 为什么同样是输入'乔'，为什么msg能匹配出文档，而msg_chinese不能呢?

- 写时分词
  - 我们使用来分析这个msg这个字段是怎样分词的

```
POST test/_analyze
{
  "field": "msg",
  "text": "乔丹是篮球之神"
}
```

```
乔，丹，是，篮，球，之，神
```

  - 再来分析这个msg_chinese这个字段是怎样分词的

```
POST test/_analyze
{
  "field": "msg_chinese",
  "text": "乔丹是篮球之神"
}
```

```
乔丹，  是，  篮球，  之神
```

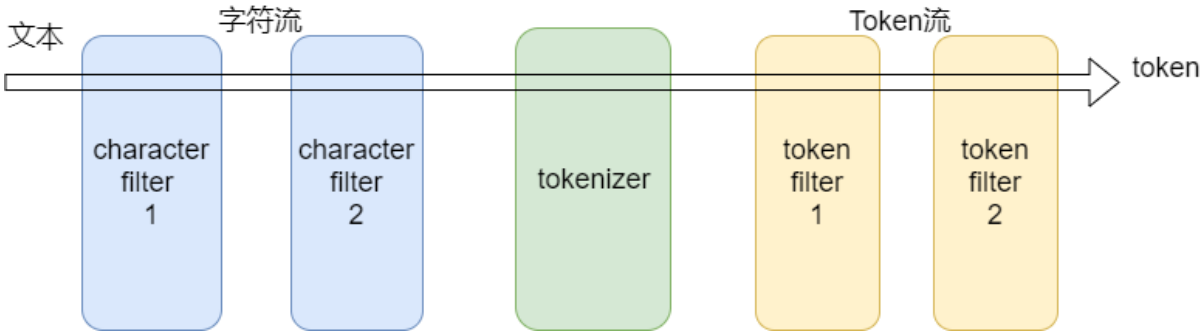
  - 文档写入的时候会根据字段设置的分词器类型进行分词，如果不指定就是默认的standard分词器。
  - 写时分词器需要在mapping中指定，而且一旦指定就不能再修改，若要修改必须重建索引。

- 读时分词
  - 由于读时分词器默认与写时分词器默认保持一致，拿上面的例子，你搜索 msg 字段，那么读时分词器为 Standard ，搜索 msg_chinese 时分词器则为 ik_max_word。这种默认设定也是非常容易理解的，读写采用一致的分词器，才能尽最大可能保证分词的结果是可以匹配的。
  - 允许读时分词器单独设置

```
POST test/_search
{
  "query": {
    "match": {
      "msg_chinese": {
        "query": "乔丹",
        "analyzer": "standard"
      }
    }
  }
}
```

- 一般来讲不需要特别指定读时分词器，如果读的时候不单独设置分词器，那么读时分词器的验证方法与写时一致。

- 深入分析




  - 分析器(analyzer)有三部分组成
    - char filter： 字符过滤器
    - tokenizer： 分词器
    - token filter： token过滤器
  - char filter（字符过滤器）
    - 字符过滤器以字符流的形式接收原始文本，并可以通过添加、删除或更改字符来转换该流。一个分析器可能有0个或多个字符过滤器。
  - tokenizer (分词器)
    - 一个分词器接收一个字符流，并将其拆分成单个token （通常是单个单词），并输出一个token流。比如使用whitespace分词器当遇到空格的时候会将文本拆分成token。"eating an apple" >> [eating, and, apple]。一个分析器必须只能有一个分词器

```
POST _analyze
{
  "text": "eating an apple",
  "analyzer": "whitespace"
}
```

  - token filter (token过滤器)

        - token过滤器接收token流，并且可能会添加、删除或更改tokens。比如一个lowercase token filter可以将所有的token转成小写。一个分析器可能有0个或多个token过滤器，它们按顺序应用。
    - standard分析器
        - tokenizer
            - Stanard tokenizer
        - token filters
            - Standard Token Filter
            - Lower Case Token Filter

**愿景："让编程不在难学，让技术与生活更加有趣"**

# 第十章 elasticsearch的课程总结

## 第1集 elasticsearch的课程总结

详情看视频