Apusic 金蝶天燕

生成时间：2026-05-19

用户手册

金蝶Apusic全文检索软件V1.0

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 产品概述

金蝶Apusic全文检索软件（Apusic Search Engine，简称“ASE”）是一个分布式、高扩展、高实时的搜索与数据分析引擎。ASE能够快速地搜索和分析大量的数据，支持全文搜索、聚合、过滤等多种查询方式，能够满足各种不同的搜索和分析需求。

1.1 概念

在此描述一些概念，该概念是该产品独有的，或者产品依赖的组件所独有的并需要向用户进行解释介绍的

概念	含义
索引	索引是ASE中的核心概念，类似于传统数据库中的数据库。它是一个拥有相同结构（映射或Schema）的文档集合，用于存储和检索特定类型的数据
文档	文档是ASE中存储的基本单元，相当于关系型数据库中的行。每个文档是一个JSON对象，包含一组键值对（字段），代表一个具体的实体或记录。文档属于某个特定的索引
映射	映射是索引的配置文件，它定义了索引中包含的字段、字段的类型和字段的索引设置。映射定义了索引中文档的格式和结构，以及如何对文档进行索引和搜索
索引词	索引词是索引中用于搜索和索引的词语，可以是单个单词、短语、数字、日期等。索引词可以是任意的文本，也可以是数字、日期等特殊类型的值
文本	文本是索引中用于搜索和索引的文本内容，可以是任意的文本，也可以是数字、日期等特殊类型的值
分析器	分析器是ASE中用于对文本进行分词和过滤的组件，它根据指定的规则将文本分割成独立的单词，并去除停用词和标点符号等，从而实现文本的预处理和索引

2 产品安装

2.1 安装说明

2.1.1 产品安装包

ASE支持Linux系统。在Linux下支持x86和arm硬件架构，请根据目标安装环境获取对应的产品安装包：

- linux操作系统和x86CPU的安装包: ase-1.0-linux-x86.tar.gz
- linux操作系统和armCPU的安装包: ase-1.0-linux-aarch64.tar.gz

2.1.2 端口说明

- HTTP API 端口 (默认为 9200): 9200端口主要用于对外提供RESTful API接口，允许客户端通过HTTP/HTTPS协议与其进行交互，执行索引文档、搜索查询、集群状态监控等各种操作。例如，当你通过Kibana、curl命令行工具或是其他应用编程方式与ASE进行通信时，通常会通过这个端口。
- Transport 端口 (默认为 9300): 9300端口用于ASE集群内部节点间的通信。当集群中的节点需要彼此之间进行数据同步、分片分配、集群健康状况检测以及其它集群管理任务时，它们会通过TCP协议在这个端口上进行通信。此端口不对外开放，通常只在集群内网环境中使用。

2.1.3 安装部署模式

ASE支持单节点部署、集群部署。

2.2 单节点部署

2.2.1 获取目标机器环境的对应的安装包

- ase-1.0-linux-x86.tar.gz

2.2.2 在目标机器上，解压ase安装包

```
tar -xzvf ase-1.0-linux-x86.tar.gz
```

2.2.3 修改ase的JVM配置和操作系统的资源配置（可跳过）

2.2.3.1 JVM配置

ase的jvm配置文件为：config/jvm.options

重要配置项有：

- -Xms1G --> -Xms : JVM堆内存最小大小，如 -Xms2g(单个节点最大堆内存不要超过32G)
- -Xmx1G --> -Xmx : JVM堆内存最大小，如-Xmx2g(单个节点最大堆内存不要超过32G)

- -XX:+UseG1GC --> 使用G1垃圾回收器
- -Djava.io.tmpdir --> JVM临时文件目录
- -XX:+HeapDumpOnOutOfMemoryError --> 当分配内存空间失败时生成dump文件
- -XX:HeapDumpPath --> dump文件存储位置
- -XX:ErrorFile --> 存储JVM严重错误文件地址

这和java应用项目的jvm配置类似，配置的堆内存最好不要超宿主机内存的1/2，因宿主机操作系统也需要相应的运行内存，建议最大堆内存不要超过32G，一般建议最大为31G。

2.2.3.2 操作系统资源限制：ulimit

1. 临时修改：`ulimit -n 65535 -u 4096`
2. 永久修改：修改`/etc/security/limits.conf`

```
# 添加下面内容
soft    nofile          65536
hard    nofile          65536
```

2.2.3.3 操作系统资源限制vm.max_map_count

1. `cat /proc/sys/vm/max_map_count`来查看当前的vm.max_map_count值
2. 修改vm.max_map_count不低于262144
   - *使用vim编辑/etc/sysctl.conf*
   - *在文件末尾添加：`vm.max_map_count=262144`*
   - *保存并重新加载：`sudo sysctl -p`*

2.2.3.4 其他设置

ASE有许多系统属性，如下表所列，您可以在命令行参数表示法中指定config/jvm.options。

属性	说明
-Dopensearch.xcontent.string.length.max=	ASE默认不限制 JSON/YAML/CBOR/Smile 字符串字段的最大长度。为了保护您的集群免受潜在的分布式拒绝服务 (DDoS) 或内存问题的影响，您可以将系统opensearch.xcontent.string.length.max属性设置为合理的限制（最大值为 2,147,483,647），例如-Dopensearch.xcontent.string.length.max=5000000
-Dopensearch.xcontent.fast_double_writer=	默认情况下，ASE 使用 Java 运行时环境提供的默认实现来序列化浮点数。将此值设置为true使用 Schubfach 算法，该算法速度

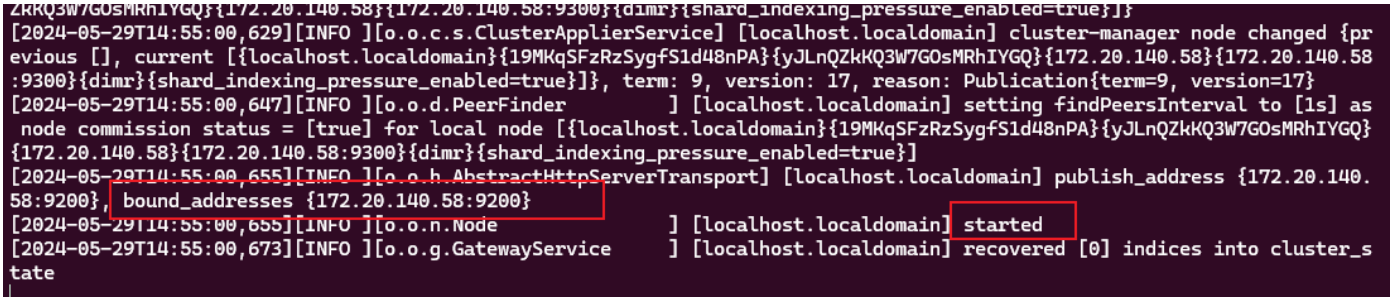
[true|false]	更快，但可能会导致精度略有差异。默认为false
-Dopensearch.xcontent.depth.max=	默认情况下，ASE不对 JSON/YAML/CBOR/Smile 文档的最大嵌套深度施加任何限制。为了保护您的集群免受潜在的 DDoS 或内存问题，您可以将opensearch.xcontent.depth.max系统属性设置为合理的限制（最大值为 2,147,483,647），例如-Dopensearch.xcontent.name.length.max=1000。
-Dopensearch.xcontent.codepoint.max=	52428800默认情况下，ASE对 YAML文档的最大大小（以代码点为单位）施加限制。为了保护您的集群免受潜在的 DDoS 或内存问题，您可以将opensearch.xcontent.codepoint.max系统属性更改为合理的限制（最大值为 2,147,483,647）。例如，-Dopensearc.xcontent.codepoint.max=5000000。

2.2.4 配置KBC授权

- 把申请的kbc授权文件，修改为license.xml，放入解压目录下的licenses/kbc文件夹。(参考 License验证)

2.2.5 启动ASE

- 启动ase： `./bin/ase`
- 待ASE启动完成后，出现下面图，则启动完成



- 访问获取节点信息： http://127.0.0.1:9200
- 使用curl获取节点信息

- `curl http://127.0.0.1:9200`

2.3 集群安装

ASE集群部署主要用于构建高可用、高并发、可水平扩展的数据检索和分析平台。它通过在多台物理或虚拟服务器上安装并配置ASE节点，这些节点通过共享相同的集群名称相互识别并组成集群。每个节点都能存储数据分片和备份分片，实现数据冗余和负载均衡。集群内部通过选举机制自动管理主副节点关系，保证数据一致性。

2.3.1 ASE部署节点可以划分为以下角色

- 主节点： 主节点是管理集群状态的(对所有资源要求都不高)
- 协调节点：协调节点是接受客户端请求并分发到相应数据节点的，并合并搜索结果(对网络资源要求高，对CPU和内存也有一定的要求)
- 数据节点：数据节点是读写数据的(对内存、CPU和IO要求高)

注：一般来说，在生产环境中，为了节省资源一个节点一般都是有多个角色，很少将节点作为单一角色来使用。

- 最小的集群：将一个节点当作所有角色来使用(每个节点既是数据节点，也是主节点和协调节点)，如上面的单节点部署模式。
- 初等规模：分成主节点和其他节点(数据节点也是协调节点)；分成协调节点和其他节点(数据节点也是主节点);
- 中高等规模：将数据节点、主节点和协调节点分开。

下面是创建ASE集群样例的过程和步骤：

2.3.2 集群列表和节点角色划分(参考)

节点名称	节点IP	节点角色	其他说明
node-200	192.168.0.200	数据节点 1 + 主节点1	
node-201	192.168.0.201	数据节点 2 + 主节点2	

node-202	192.168.0.202	数据节点 3 + 主节点3	
node-203	192.168.0.203	协调节点1	
node-204	192.168.0.204	协调节点2	
node-205	192.168.0.205	协调节点3	

2.3.3 在6个节点中，上传同一份安装包，并解压到安装目录

```
tar -zxvf ase-1.0-linux-x86.tar.gz
```

2.3.4 修改每个节点的config/ase.yaml文件，添加以下内容

```
# 集群名称
cluster.name: ase-search
# 按照上面的节点列表，改成每个节点的名称
node.name: node-200

# 修改数据存放路径和日志路径
path.data: /home/ase-user/data
path.logs: /home/ase-user/log


# 按照上表，改成每个的IP或主机名
network.host: 192.168.0.200
http.port: 9200

discovery.seed_hosts: ["192.168.0.200:9300",
"192.168.0.201:9300","192.168.0.202:9300","192.168.0.203:9300","192.168

cluster.initial_master_nodes: ["192.168.0.200:9300",
"192.168.0.201:9300","192.168.0.202:9300"]
```

注:

- `node.name`:每个节点要改成对应节点的名称

- `network.host`:每个节点要改成对应节点的主机名或IP
- `discovery.seed_hosts`:集群中的节点列表
- `cluster.initial_master_nodes`:集群初始化时主节点列表

2.3.5 在主节点上加入以下配置

在三个主节点[node-200, node-201, node-202]三个节点的ase.yaml配置文件中，添加

```
node.roles: [cluster_manager]
```

注:如果`node.roles`原来有其他角色，要合并到一起，如原来是data节点，那么就设置为`[data,cluster_manager]`

2.3.6 在所有协调节点上配置

在三个协调节点[node-203，node-204，node-205]三个节点的ase.yaml配置文件中，添加

```
node.roles:[]
```

注:没有任何指定角色的节点就是协调节点(默认所有的节点都是协调节点)

2.3.7 在所有数据节点上加入以下配置

在三个主节点[node-200, node-201, node-202]三个节点的ase.yaml配置文件中，修改以下配置:

```
node.roles: [data]
```

注:如果`node.roles`原来有其他角色，要合并到一起，如原来是cluster_manager节点，那么就设置为`[cluster_manager,data]`

2.3.8 启动各节点

依次启动各节点，如果节点启动失败，可根据错误来进行处理

通过`ip:9200`的形式在浏览器中访问，查看各节点是否启动成功

如`http://192.168.0.200:9200`,如果可以成功访问说明启动成功

2.3.9 检测集群是否搭建成功

在浏览器中`http://192.168.0.200:9200/_cat/nodes`

查看结果是否如下:

```
192.168.0.201 33 96 1 0.19 0.18 0.19 dim - node-201
192.168.0.200 22 94 0 0.13 0.22 0.18 dim * node-200
192.168.0.202 16 81 1 0.18 0.32 0.20 dim - node-202
192.168.0.205 17 96 1 0.40 0.32 0.19 - - node-205
192.168.0.203 15 78 1 0.27 0.34 0.22 - - node-203
192.168.0.204 15 95 0 0.48 0.45 0.25 - - node-204
```

注:

- role那一行中，是多个角色的组合
  - d: 数据节点(data)
  - i: 预处理节点(ingest)
  - m: 主节点备选节点(master)
- master那一行，表示当前节点是否为主节点
  - `*` : 是主节点
  - `-` : 不是主节点

2.4 管控台安装

已经部署好ASE情况下安装ASE管控台的步骤如下:

1. 在知识中心下载对应的ase-dashboards安装包。
2. 上传到ase安装服务器，并解压。
3. 配置连接ase的地址和端口。
4. 启动服务。

注：ase需要开启ssl和创建用户【参考用户手册的：SSL和设置admin用户密码】

2.4.1 ASE管控台下载和安装

知识中心的管控台安装包分为:

- `ase-dashboards-1.0.1-linux-x64.tar.gz` : linux-64版本
- `ase-dashboards-1.0.1-linux-arm.tar.gz` : linux-arm版本

下载安装包后，上传到目标服务器/opt目录，解压（实例为arm版本）:

```
tar -zxvf ase-dashboards-1.0.1-linux-arm.tar.gz
```

2.4.2 配置连接ase的地址和端口

1. 进入安装目录：cd /opt/ase-dashboards-1.0.1-linux-arm/
2. 修改配置文件config/opensearch_dashboards.yml，修改opensearch.hosts内容：

```
opensearch.hosts: [https://127.0.0.1:9200]
opensearch.ssl.verificationMode: none
```

3. 启动服务

```
./bin/opensearch-dashboards --allow-root &
```

2.4.3 访问控制台

浏览器访问：http://ip地址:5601 输入ase的用户名和密码，即可进入ASE管控台。

3 快速开始

ASE的功能有：

- 索引管理
  - 创建、删除索引（PUT、DELETE 对应的端点）。
  - 获取索引状态和统计信息。
- 文档管理
  - 向索引中添加或更新文档（POST、PUT 请求）。
  - 从索引中检索文档（GET 请求）。
  - 删除索引中的文档（DELETE 请求）。
- 搜索
  - 执行全文搜索查询（GET 或 POST 请求到 _search 端点）。
  - 支持布尔查询、短语查询、通配符查询等。
- 聚合
  - 执行聚合查询，以对数据进行分组和汇总（作为搜索请求的一部分）。
- 更新
  - 部分更新文档中的字段（POST 请求到 _update 端点）。
- 批量操作
  - 执行批量操作来索引、更新或删除多个文档（POST 请求到 _bulk 端点）。
- 查询DSL
  - 使用ASE查询领域特定语言（DSL）构建复杂的查询。
- 脚本
  - 执行脚本以在搜索、聚合或更新操作中动态生成值。
- 高亮
  - 在搜索结果中高亮显示匹配的文本片段。
- 排序
  - 控制搜索结果的排序方式。
- 分页
  - 使用 from 和 size 参数进行结果的分页。
- 版本控制
  - 管理文档的版本，处理并发更新。
- 索引模板
  - 定义索引模板以自动应用设置和映射到新索引。

- 分析器
  - 测试和使用自定义分析器。
- 索引别名
  - 管理索引别名以简化索引的搜索和访问。
- 索引生命周期管理（ISM）
  - 使用索引生命周期策略自动管理索引的生命周期。
- 跨集群复制
  - 复制索引到另一个集群。
- 安全性
  - 管理索引的访问权限和安全设置。
- 监控
  - 监控索引和集群的健康状态和性能指标。

3.1 使用管控台

在部署好ASE和管控台后（部署参考用户手册或安装手册），可以通过管控台的Dev tools工具来使用REST API。

1. 登录管控台，点击左侧导航栏的Dev tools，进入Dev tools页面。

3.1.1 文档增删改查语法

可以通过PUT、DELETE、POST、GET语法实现对于文档的增删改查方法。

- PUT
  - 创建资源：如果指定资源的id不存在，PUT请求可能会被用来创建该资源。
  - 更新资源：如果向一个已存在的资源id发送请求，该资源的当前状态会被请求体中的新状态替换。
- DELETE
  - 删除资源：如果指定资源的id存在，DELETE请求可能会被用来删除该资源。
  - 删除资源列表：如果指定资源的id不存在，DELETE请求可能会被用来删除该资源的元数据。
- GET
  - 获取资源：如果指定资源的id存在，GET请求可能会被用来获取该资源的元数据。
  - 获取资源列表：如果指定资源的id不存在，GET请求可能会被用来获取该资源的元数据。
- POST
  - 创建资源：使用POST请求时，你通常向资源的集合URL发送请求，而不是向特定资源的URL发送请求。服务器会处理请求体中的数据，并创建一个新的资源实例，然后返回新创建资源的元数据（如ID和位置）。

- 执行操作：在某些情况下，POST请求也被用于执行不直接对应于资源创建的操作，比如提交表单数据或触发某些处理流程。

3.1.2 PUT 创建文档

1. PUT创建文档基本格式

```
PUT <index>/_doc/<id>
{     "JSON":"document"     }
```

PUT：请求通常用于更新或创建资源，在大多数RESTful API中，PUT请求主要用于更新已存在的资源。在使用PUT请求的时候，需要**指定资源的完整URL（包括ID）**。

```
## 例子
PUT /account/_doc/3
{
    "name": "Apusic",
    "type": "test"
}
```

```
## 结果
{
    "_index": "account", ##显示索引名
    "_id": "3",
    "_version": 1,
    "result": "created",  ##操作类型
    "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
    },
    "_seq_no": 3,
    "_primary_term": 2
}
```

3.1.3 POST创建文档基本格式

```
POST  <index>/_doc
{    "JSON":"document"    }

POST <index>/_doc/<id>
{    "JSON":"document"    }
```

POST:请求通常用于创建新的资源，不需要指定资源的id，而是在创建资源的时候自动生成相应id,当然也可以在自行设置资源id。

```
## 例子
POST movies/_doc
{ "title": "Spirited Away" }
```

```
## 结果
{
    "_index": "movies",
    "_id": "1pFhApEBaB3Z-jDLo_us",
    "_version": 1,
    "result": "created",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 0,
    "_primary_term": 1
}
```

3.1.4 DELETE删除文档

DELETE：通过方法可以执行一系列相关删除数据操作，常见的方法如下：

- 删除单个文档

```
## 删除单个文档
DELETE <index>/_doc/<id>
## curl命令 (索引为my_index,id为123)
```

```
curl -X DELETE "localhost:9200/my_index/_doc/123" -H 'Content-
Type: application/json'
```

> *执行操作，需要指定对应的索引以及id，在使用curl命令时，localhost:9200默认地址，需要根据实际opensearch的集群地址。*

```
## 例子
DELETE movies/_doc/1pFhApEBaB3Z-jDLo_us
```

```
## 结果
{
    "_index": "movies",
    "_id": "1pFhApEBaB3Z-jDLo_us",
    "_version": 2,
    "result": "deleted",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 1,
    "_primary_term": 3
}
```

- 删除多个文档

  通过POST方法采用_delete_by_query API来根据查询条件删除多个文档。

```
POST /<index>/_delete_by_query
{ "query": {
    "match": {
        "field_name": "field_value"
        }
```

```
    }
  }
```

请求会删除索引中所有file_name字段值为field_value的文档。

```
## 例子
{
    "took": 19,
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
        "value": 3,
        "relation": "eq"
        },
        "max_score": 1,
        "hits": [
          {
            "_index": "account",
            "_id": "3",
            "_score": 1,
            "_source": {
            "name": "Apusic",
            "type": "test1"
          }
        },
          {
```

```
            "_index": "account",
            "_id": "1",
            "_score": 1,
            "_source": {
            "name": "Apusic",
            "type": "test"
            }
        },
        {
            "_index": "account",
            "_id": "2",
            "_score": 1,
            "_source": {
            "name": "Apusic",
            "type": "test"
            }
        }
        ]
    }
}
```

对上述文档集合进行多文档删除处理

```
POST /account/_delete_by_query
{
    "query":{
        "match":{
        "type":"test" ##删除其中type=test的文档
        }
    }
}
```

```
# 结果
  {
      "took": 867,
      "timed_out": false,
      "total": 2,
      "deleted": 2,
      "batches": 1,
      "version_conflicts": 0,
      "noops": 0,
      "retries": {
      "bulk": 0,
      "search": 0
      },
      "throttled_millis": 0,
      "requests_per_second": -1,
      "throttled_until_millis": 0,
      "failures": []
  }
```

3.1.5 PUT更新文档

- PUT更新文档基本格式

  PUT请求在向一个已经创建的文档资源id发送请求，该资源的当前状态会被请求体的新状态进行修改。

```
PUT <index>/_doc/<id>
{    "JSON":"document"    }
```

```
## 例子
PUT /account/_doc/2
{
    "name": "Apusic",
```

```
    "type": "updta test"
}
```

```
## 结果
{
    "_index": "account",
    "_id": "2",
    "_version": 2,
    "result": "updated",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 14,
    "_primary_term": 4
}
```

- POST更新文档基本格式

POST请求通过`_update`命令更新已经存在的数据，在内容中包含所要进行更新的数据类型及内容。

```
POST <index>/_update/<id>
{
    "filed_name":"filed_update" , ## 常用的filed_name为doc，更新文档主体内容
      ...
}
```

```
## 例子
POST /account/_update/2
{
    "doc": {
```

```
      "name":"Apusic_update"
    }
}
```

```
## 结果
{
      "_index": "account",
    "_id": "2",
    "_version": 3,
    "result": "updated",
    "_shards": {
        "total": 2,
        "successful": 1,
        "failed": 0
    },
    "_seq_no": 15,
    "_primary_term": 4
}
```

3.1.6 GET获取/查询文档

- 基本请求格式：*_search*来获取一个索引下所有的值

```
GET /<index>/_search ##请求获取索引下所有值
{
    "query": {
        "match_all":{}
    }
}
```

```
## 例子
GET /account/_search
{
```

```
    "query": {
        "match_all": {}
    }
}
```

##结果

```
{
    "took": 1,
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
        "value": 2,
        "relation": "eq"
        },
        "max_score": 1,
        "hits": [
        {
            "_index": "account",
            "_id": "1",
            "_score": 1,
            "_source": {
            "name": "Apusic",
            "type": "test"
            }
        },
        {
```

```
            "_index": "account",
            "_id": "2",
            "_score": 1,
            "_source": {
            "name": "Apusic",
            "type": "test"
            }
        }
        ]
    }
}
```

除上述之外，可以通过 `<index>/_doc/<id>` 读取索引下指定的一个文档。

```
## 例子
GET /account/_doc/1

##结果
{
    "_index": "account",
    "_id": "1",
    "_version": 1,
    "_seq_no": 11,
    "_primary_term": 3,
    "found": true,
    "_source": {
        "name": "Apusic",
        "type": "test"
    }
}
```

- 使用查询字符串参数的GET请求

```
GET /<index>/_search?q=*:*
```

> *这里，q=: 是一个 Lucene 查询字符串，意味着"匹配所有文档"。但是，请注意，由于 GET 请求的 URL 长度限制和敏感信息（如查询体）不应该放在 URL 中，因此这主要用于演示或非常简单的查询。*

- POST参数查询

在实际应用中，由于 GET 请求的限制（如 URL 长度和不支持请求体），通常会使用 POST 请求来执行复杂的查询或检索大量数据。POST 请求允许你在请求体中发送 JSON 格式的查询和数据。

```
POST /<index>/_search
{
    "query": {
        "match": {
        "field_name": "value_to_match"
        }
      }
}
```

```
## 例子
POST /account/_search
{
    "query":{
        "match":{
        "_id":"1"
        }
    }
}
```

```
## 结果
{
    "took": 10,
    "timed_out": false,
    "_shards": {
```

```
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
        "value": 1,
        "relation": "eq"
        },
        "max_score": 1,
        "hits": [
        {
            "_index": "account",
            "_id": "1",
            "_score": 1,
            "_source": {
            "name": "Apusic",
            "type": "test"
            }
        }
        ]
    }
}
```


3.2 使用REST API

REST API入口： ASE节点IP地址:9200（默认端口）。 下面是参考，具体的内容可以参考上面的管控台的API文档。

3.2.1 创建文档

HTTP方法： PUT或POST

HTTP请求路径： /索引名称/类型名称/文档id

HTTP请求体： json

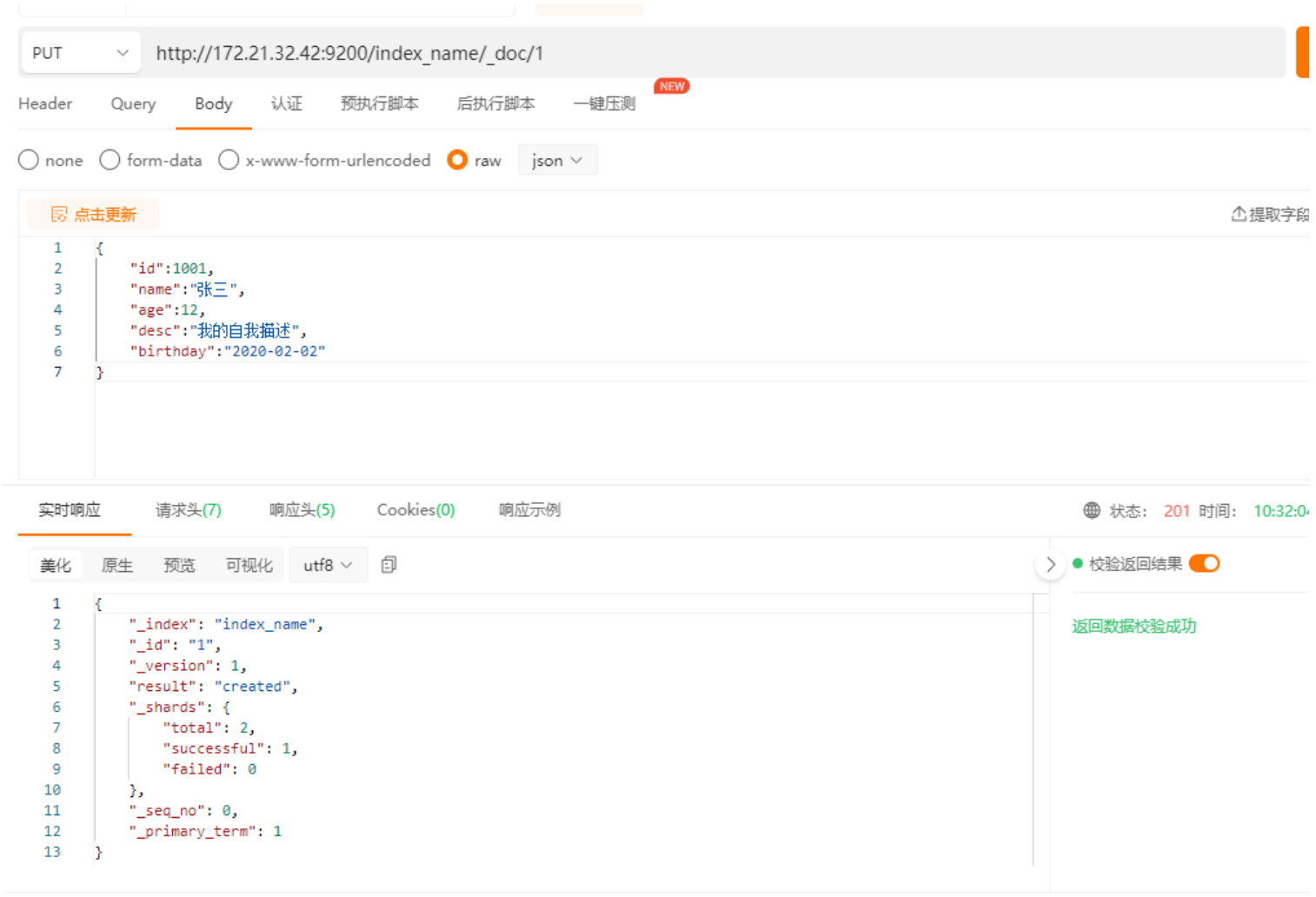
> *put文档必须要指定文档 _id*

> *post可指定，也可不指定，不指定则会随机生成一个 _id。*

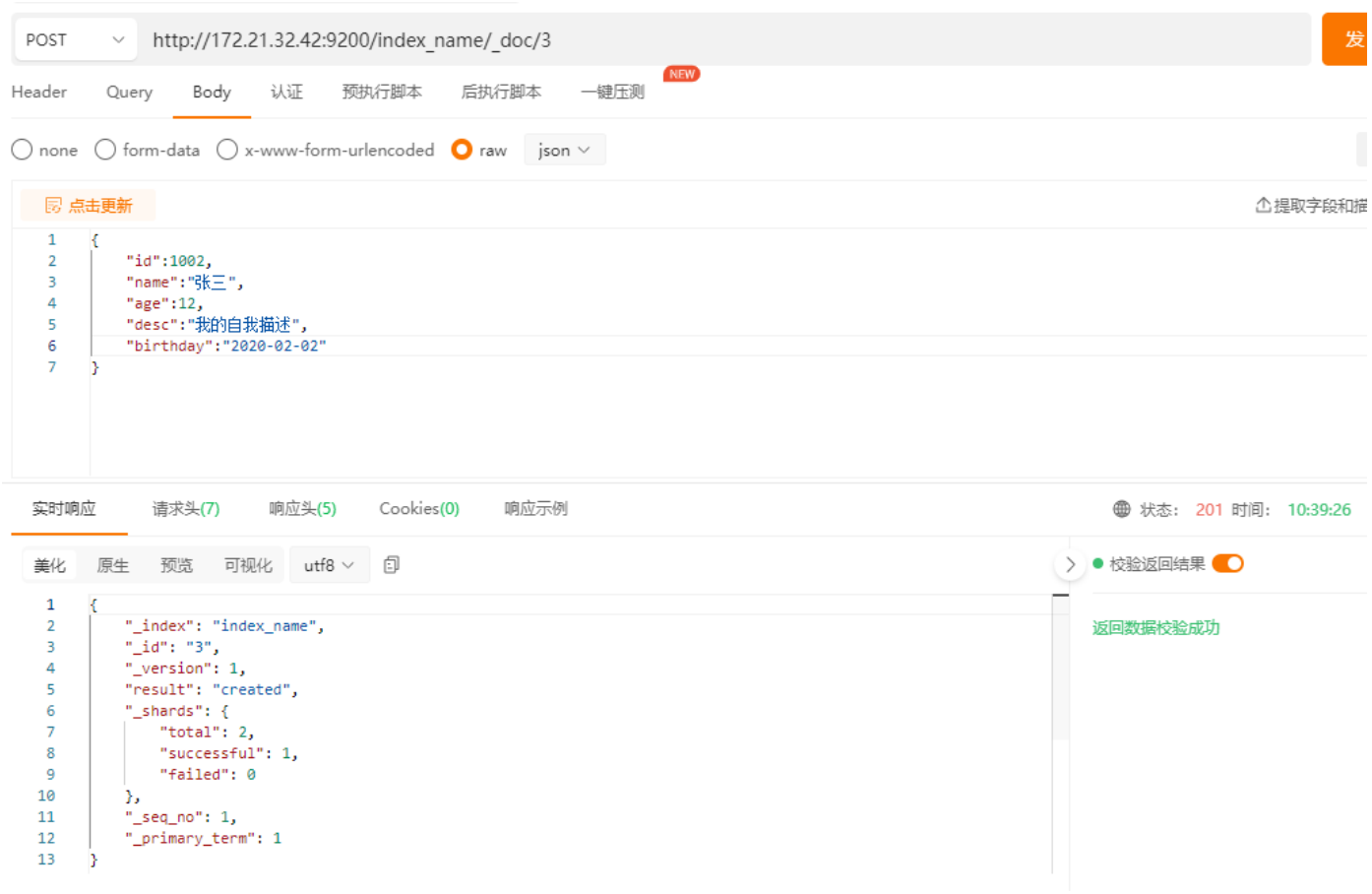
> *如果没有提前设定索引中字段类型而直接添加文档，ase会对字段数据给自动数据类型，新字段会永久补充进去mapping。*

参考样例

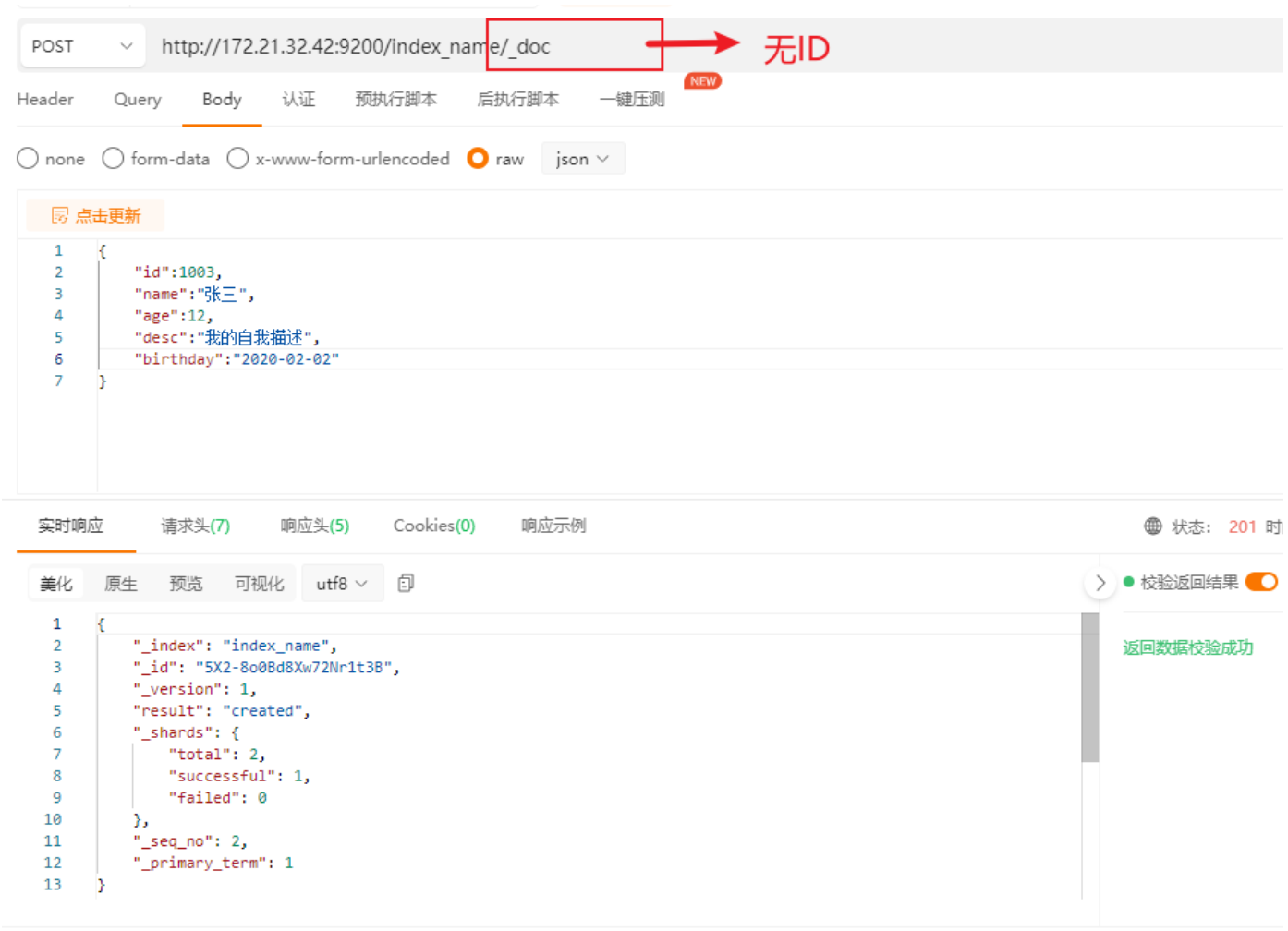
- put指定ID（如果已经有对应数据，则修改，无则创建）



- POST指定ID（有则修改，无则创建）



- post不指定id，自动生成文档id（每次执行都是创建新的文档）



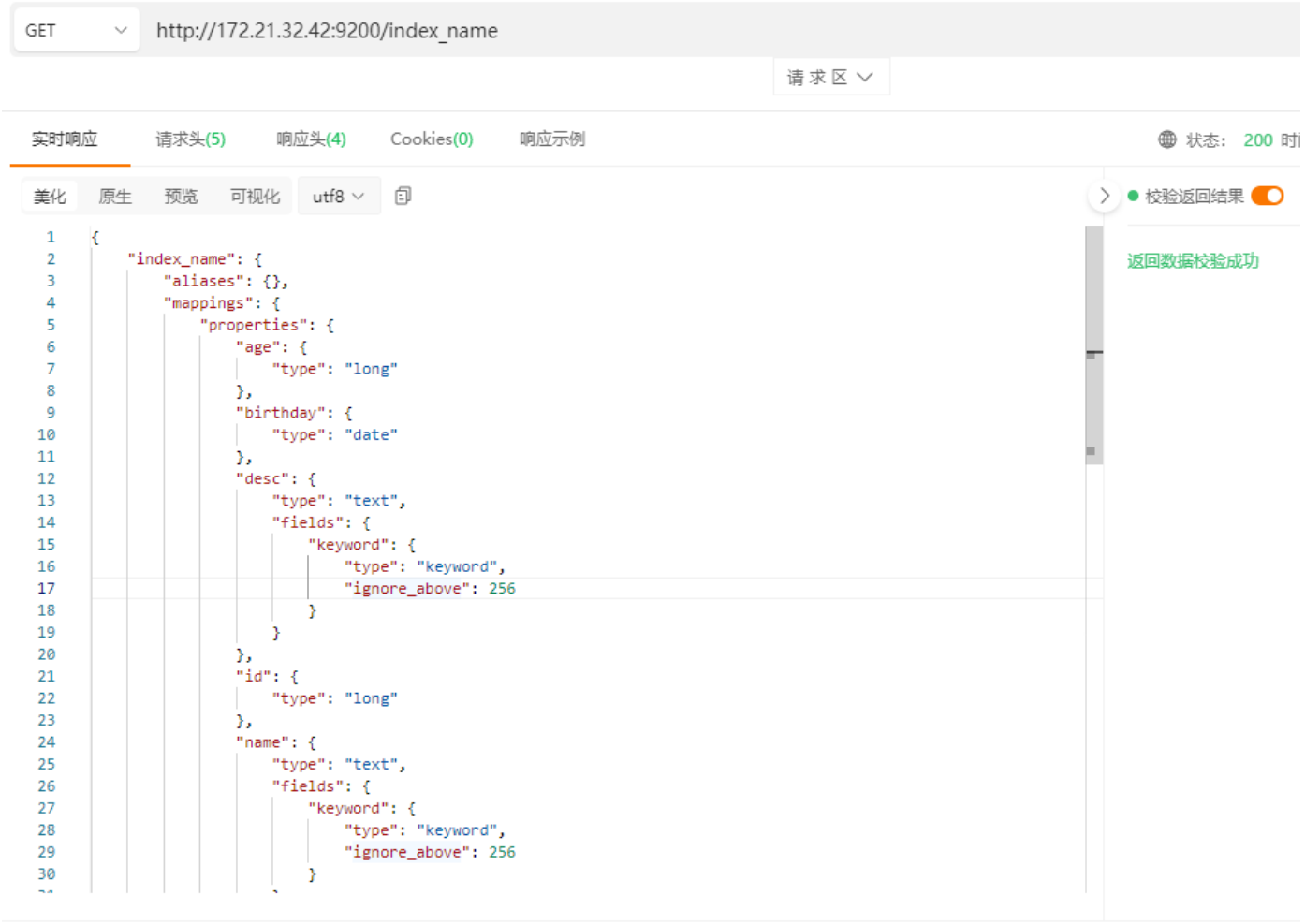
3.2.2 查询文档

HTTP方法： GET

HTTP请求路径：/索引名称/类型名称/文档id

HTTP请求体： 无

- 查询上面创建的index_name索引信息

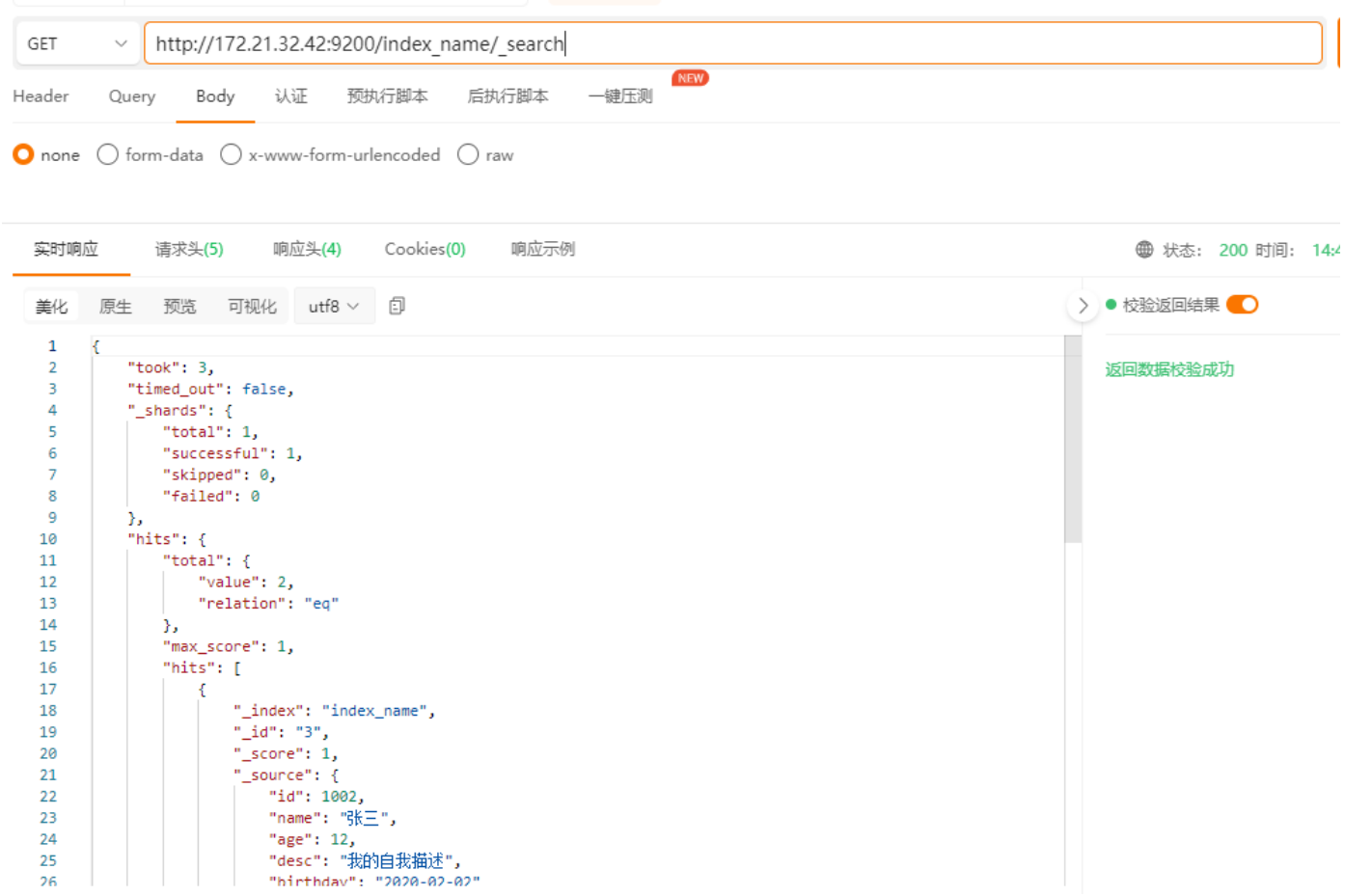

GET
http://172.21.32.42:9200/index_name
请求区
实时响应
请求头(5)
响应头(4)
Cookies(0)
响应示例
状态: 200 时
美化
原生
预览
可视化
utf8
校验返回结果
返回数据校验成功
{
"index_name": {
"aliases": {},
"mappings": {
"properties": {
"age": {
"type": "long"
},
"birthday": {
"type": "date"
},
"desc": {
"type": "text",
"fields": {
"keyword": {
"type": "keyword",
"ignore_above": 256
}
}
},
"id": {
"type": "long"
},
"name": {
"type": "text",
"fields": {
"keyword": {
"type": "keyword",
"ignore_above": 256
}

- 查询文档内容


GET
http://172.21.32.42:9200/index_name/_doc/1
请求区
实时响应
请求头(5)
响应头(4)
Cookies(0)
响应示例
状态: 2
美化
原生
预览
可视化
utf8
校验返回结果
返回数据校验成功
{
"_index": "index_name",
"_id": "1",
"_version": 1,
"_seq_no": 0,
"_primary_term": 1,
"found": true,
"_source": {
"id": 1001,
"name": "张三",
"age": 12,
"desc": "我的自我描述",
"birthday": "2020-02-02"
}
}

- 查询全部文档

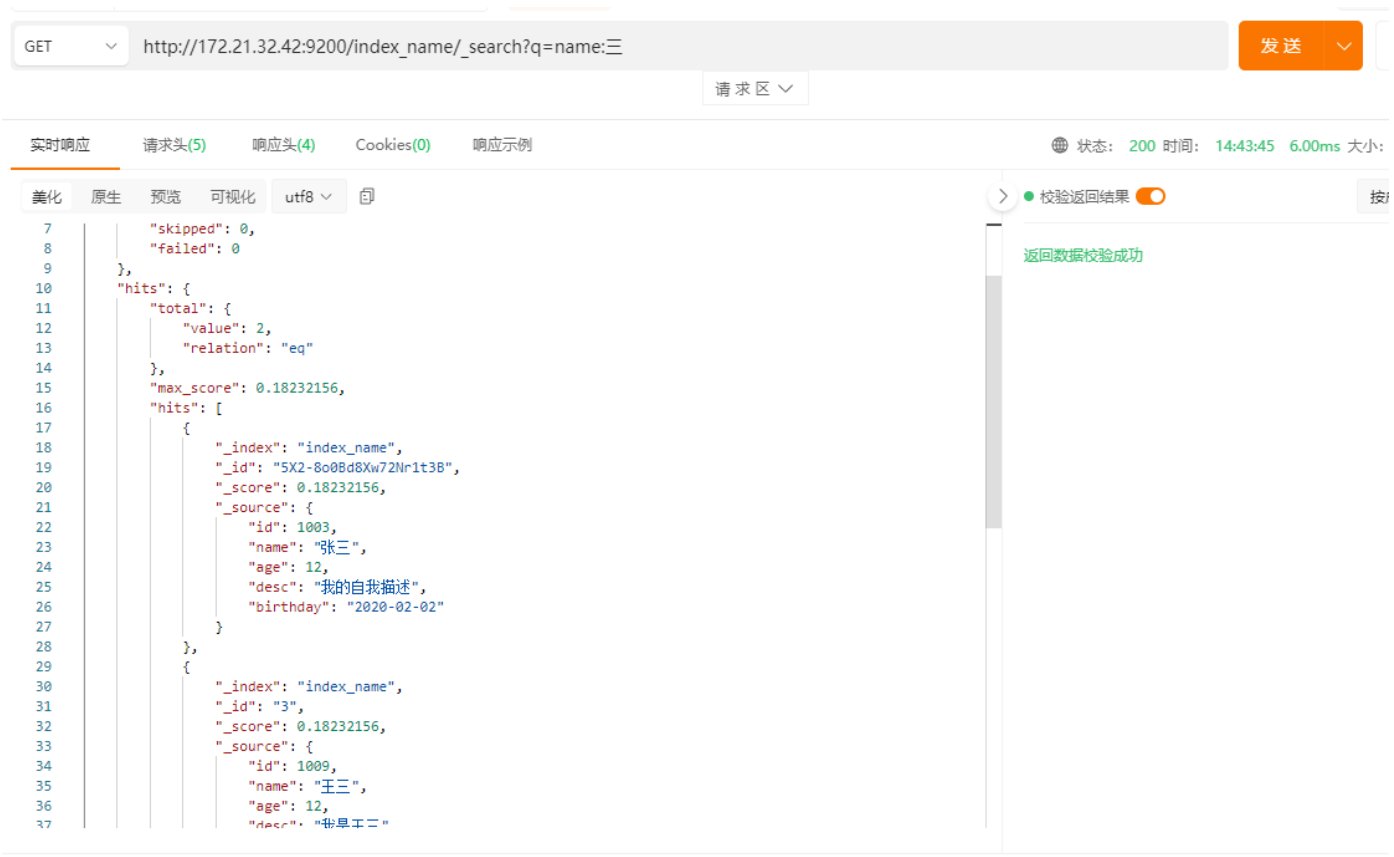
  请求路径： /索引名称/_search



- 模糊查询

  请求路径： /索引名称/_search?q=name:XXX

  如下搜索名称包含“三”的记录



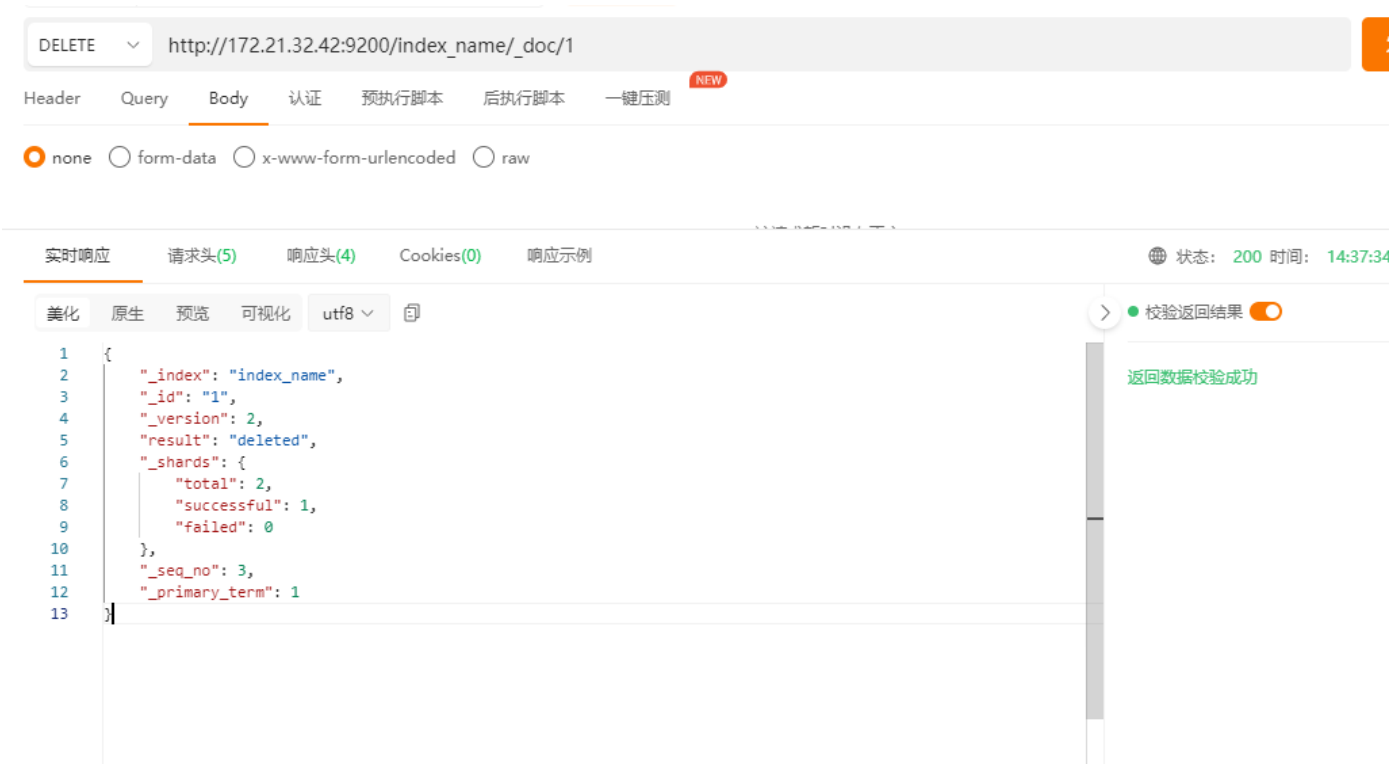
3.2.3 删除文档

HTTP方法： DELETE

HTTP请求路径： /索引名称/类型名称/文档ID

HTTP请求体： 无

- 删除index_name中id等于1的记录



3.3 创建ISM生命周期管理

索引状态管理（ISM）是一款插件，它允许根据索引年龄、索引大小或文档数量的变化触发定期的管理操作，从而实现自动化。使用ISM插件，可以定义策略来自动处理索引更新或删除，以满足特定的用例。

例如，可以定义一个策略，使索引在30天后进入read_only状态，然后在设定的90天后将其删除。此外，还可以设置策略，在索引被删除时发送通知消息。

在一定时间后执行索引滚动更新，或者在非高峰时段对索引运行force_merge操作，可以提高高峰时段的搜索性能。通过应用合理的索引策略，可以实现索引的自动清理、数据热-温-冷自动迁移、自动备份、force merge、rollover、rollup、close、open、delete等功能，以确保数据以尽可能最具成本效益的方式正确存储。

3.3.1 ISM入门

策略是一组描述如何管理索引的规则。有关创建策略的信息，请参下一个章节策略介绍。 可以使用可视化编辑器或 JSON 编辑器来创建策略。与JSON编辑器相比，可视化编辑器（管控台）通过将流程分为创建错误通知、定义 ISM 模板和添加状态，提供了一种更结构化的策略定义方式。

创建策略后，需要将策略附加到一个或多个索引上，可以在策略中设置，ism_template这样当创建与ISM 模板模式匹配的索引时，插件会自动将策略附加到该索引。

3.3.2 使用JSON创建策略

以下示例说明如何创建一个策略，该策略自动附加到名称以index_name-开头的所有索引。

```
PUT _plugins/_ism/policies/policy_id
{
  "policy": {
    "description": "Example policy.",    # 描述
    "default_state": "...",
    "states": [...],
    "ism_template": {
      "index_patterns": ["index_name-*"],
      "priority": 100
    }
  }
}
```

3.3.3 ISM样例

- 使用ISM创建7天后自动删除的策略，并且这个策略自动绑定到全部以 `company` 开头的索引。

```
PUT /_plugins/_ism/policies/d7d
{
  "policy": {
    "description": "delete 7 after day",
    "default_state": "delete",
    "schema_version": 1,
    "states": [
      {
        "name": "rollover",
        "actions": [
          {
            "rollover": {
              "min_index_age": "7d",
              "min_doc_count": 1
            }
          }
        ],
```

```
        "transitions": [
          {
            "state_name": "delete"
          }
        ]
      },
      {
        "name": "delete",
        "actions": [
          {
            "delete": {}
          }
        ]
      }
    ],
    "ism_template": {
      "index_patterns": ["company*"],
      "priority": 101
    }
  }
}
```

- 一个作业将触发滚动操作，并创建一个新索引。接下来，将另一个文档添加到这两个索引中。然后，新作业将导致第二个索引指向日志别名，并且旧索引将由于别名操作而被删除。

```
PUT /_plugins/_ism/policies/rollover_policy?pretty
{
  "policy": {
    "description": "Example rollover policy.",
    "default_state": "rollover",
    "states": [
      {
```

```
      "name": "rollover",
      "actions": [
        {
          "rollover": {
            "min_doc_count": 1
          }
        }
      ],
      "transitions": [{
          "state_name": "alias",
          "conditions": {
            "min_doc_count": "2"
          }
        }]
    },
    {
      "name": "alias",
      "actions": [
        {
          "alias": {
            "actions": [
              {
                "remove": {
                    "alias": "log"
                }
              }
            ]
          }
        }
      ]
    }
  ],
```

```
    "ism_template": {
      "index_patterns": ["log*"],
      "priority": 100
    }
  }
}
```

接下来，创建一个要启用该策略的索引模板

```
PUT /_index_template/ism_rollover?
{
  "index_patterns": ["log*"],
  "template": {
    "settings": {
    "plugins.index_state_management.rollover_alias": "log"
    }
  }
}
```

接下来，更改集群设置以每分钟触发作业：

```
PUT /_cluster/settings?pretty=true
{
  "persistent" : {
    "plugins.index_state_management.job_interval" : 1
  }
}
```

接下来创建一个新索引：

```
PUT /log-000001
{
  "aliases": {
```

```
    "log": {
      "is_write_index": true
    }
  }
}
```

最后，向索引中添加一个文档来触发作业：

```
POST /log-000001/_doc
{
  "message": "dummy"
}
```

可以使用 Alias 和 Index API 验证以下步骤：

```
GET /_cat/aliases?pretty
```

```
GET /_cat/indices?pretty
```

注意：`index`和`remove_index`参数不支持别名。

3.4 ISM策略介绍

策略是定义以下内容的 JSON 文档：

- 索引可以处于的状态，包括新索引的默认状态。例如，可以将状态命名为“热”、“温”、“删除”等等
- 当索引进入某种状态时，可以让插件采取的任何操作，例如执行滚动更新。
- 索引进入新状态必须满足的条件，称为转换。例如，如果索引已超过八周，可能希望将其移至“删除”状态。

换句话说，策略定义了索引可以处于的状态、处于某种状态时要执行的操作以及在状态之间转换必须满足的条件。

可以完全灵活地设计策略。可以创建任何状态、转换到任何其他状态，并在每个状态下指定任意数量的操作。

关键字	描述	类型	必须	只读
policy_id	策略的名称。	string	是	是

description	该政策的人类可读的描述。	string	是	否
ism_template	指定 ISM 模板以自动将策略应用到新创建的索引。	nested list of objects	否	否
ism_template.index_patterns	指定与新创建的索引名称匹配的模式。	list of strings	否	否
ism_template.priority	当多个策略与新创建的索引名称匹配时，指定优先级以消除歧义。	number	否	否
last_updated_time	政策上次更新的时间。	timestamp	是	是
error_notification	错误通知的目标和消息模板。目标可以是 Amazon Chime、Slack 或 Webhook URL。	object	否	否
default_state	每个使用此策略的索引的默认起始状态。	string	是	否
states	在策略中定义的状态。	nested list of objects	是	否

3.4.1 状态

状态是对托管索引当前所处状态的描述。托管索引一次只能处于一种状态。每个状态都有相关的操作，这些操作在进入状态时按顺序执行，并且转换在所有操作完成后进行检查。

下表列出了可以为某个状态定义的参数。

关键字	描述	类型	必需
`name`	状态的名称。	`string`	是
`actions`	进入状态后要执行的操作。	`nested list of objects`	是
`transitions`	下一个状态以及转换到这些状态所需的条件。如果不存在转换，则策略假定它已完成并现在可以停止管理索引。	`nested list of objects`	是

3.4.2 操作

操作是策略在进入特定状态时按顺序执行的步骤。 ISM 按照操作定义的顺序执行操作。例如，如果定义操作 [A、B、C、D]，ISM 将执行操作 A，然后根据集群设置进入休眠期 plugins.index_state_management.job_interval。休眠期结束后，ISM 继续执行剩余操作。但是，如果 ISM 无法成功执行操作 A，则操作结束，并且不会执行操作 B、C 和 D。

还可以选择定义操作的超时期限，如果超过该期限，则强制操作失败。例如，如果超时设置为 1d，并且 ISM 在一天内未完成操作，则即使重试后，操作也会失败。

范围	描述	类型	必需	默认
timeout	操作的超时期限。接受分钟、小时和天的时间单位。	time unit	否	-
retry	该操作的重试配置。	object	否	具体到行动

该 retry 操作具有以下参数

范围	描述	类型	必需	默认
count	重试的次数。	number	是	-
backoff	重试时使用的退避策略类型。有效值为 Exponential、Constant 和 Linear。	string	否	指数
delay	重试之间等待的时间。接受分钟、小时和天的时间单位。	time unit	否	1分钟

以下示例操作的超时期限为 1 小时。该策略使用指数退避策略重试此操作三次，每次重试间隔 10 分钟：

```
"actions": {
  "timeout": "1h",
  "retry": {
    "count": 3,
    "backoff": "exponential",
    "delay": "10m"
  }
}
```

以下是支持的操作：

- 强制合并(force_merge)
- 只读(read_only)
- 读写(read_write)
- 副本数(replica_count)
- 收缩(shrink)
- 关闭(close)
- 打开(open)
- 删除(delete)
- 滚下(rollover)

- 通知(notification)
- 快照(snapshot)
- 索引优先级(index_priority)
- 分配(allocation)
- 卷起(rollup)

3.4.3 强制合并（force_merge）

范围	描述	类型	必需
`max_num_segments`	要减少碎片的段数。	`number`	是
`wait_for_completion`	布尔值	设置为 时`false`，请求将立即返回，而不是在操作完成后返回。要监视操作状态，请使用请求返回的任务 ID 的Tasks API`true` 。默认值为。	
`task_execution_timeout`	时间	显式任务执行超时。仅当`wait_for_completion`设置为时才有用`false`。默认值为`1h`。	否

```
{
  "force_merge": {
    "max_num_segments": 1
  }
}
```

3.4.4 只读（read_only）

将托管索引设置为只读。

```
{
  "read_only": {}
}
```

将索引设置设为`index.blocks.write`托管`true`索引。注意：此块不会阻止索引刷新。

3.4.5 读写（read_write）

将托管索引设置为可写。

```
{
  "read_write": {}
}
```

3.4.6 副本数（replicas)

设置分配给索引的副本数。

范围	描述	类型	必需
number_of_replicas	定义分配给索引的副本数。	number	是

```
{
  "replica_count": {
    "number_of_replicas": 2
  }
}
```

3.4.7 收缩(shrink)

允许减少索引中的主分片数量。通过此操作，可以指定：

- 目标索引应包含的主分片的数量。
- 目标索引中主分片的最大分片大小。
- 指定一个百分比来缩小目标索引中的主分片的数量。

```
"shrink": {
    "num_new_shards": 1,
    "target_index_name_template": {
        "source": "_shrunken"
    },
    "aliases": [
      {
        "my-alias": {}
      }
    ],
    "switch_aliases": true,
```

```
    "force_unsafe": false
}
```

范围	描述	类型	例子	必需
num_new_shards	收缩索引中主分片的最大数量。	整数	5	max_shard_size是，但是它不能与或一起使用percentage_of_source_shards。
max_shard_size	目标索引的分片的最大大小（以字节为单位）。	关键词	5gb	是，但是不能与num_new_shards或一起使用percentage_of_source_shards。
percentage_of_source_shards	要缩减的原始主分片数量的百分比。此参数表示缩减主分片数量时要使用的最小百分比。必须介于0.0到1.0之间（不含）。	百分比	0.5	是，但不能与max_shard_size或一起使用num_new_shards
target_index_name_template	缩小索引的名称。接受字符串和Mustache变量and。	字符串或胡子模板	{"source": "_shrunken"}	否
aliases	要添加到新索引的别名。	目的	`myalias`	否。它必须是别名对象的数组。
switch_aliases	如果`true`，则将别名从源索引复制到目标索引。如果与字段中的别名存在名称冲突aliases，	布尔值	`true`	否。默认隐式值为`false`，表示默认情况下不复制任何别名。

	则使用字段中的别名aliases代替名称。			
force_unsafe	如果是`true`，即使没有副本，也会缩小索引。	布尔值	false	否

如果要添加到操作中，参数必须包含别名对象`aliases`数组。例如，

```
"aliases": [
  {
    "my-alias": {}
  },
  {
    "my-second-alias": {
      "is_write_index": false,
      "filter": {
        "multi_match": {
          "query": "QUEEN",
          "fields": ["speaker", "text_entry"]
        }
      },
      "index_routing" : "1",
      "search_routing" : "1"
    }
  },
]
```

3.4.8 关闭(close)

关闭管理索引。

```
{
  "close": {}
}
```

已关闭的索引仍保留在磁盘上，但不占用 CPU 或内存，无法读取、写入或搜索已关闭的索引。

如果需要保留数据的时间比主动搜索数据的时间长，并且数据节点上有足够的磁盘空间，则关闭索引是一个不错的选择。如果需要再次搜索数据，重新打开已关闭的索引比从快照恢复索引更简单。

3.4.9 打开（open）

打开托管索引。

```
{
  "open": {}
}
```

3.4.10 删除

删除托管索引。

```
{
  "delete": {}
}
```

3.4.11 滚下(rollover)

当托管索引满足其中一个滚动条件时，将别名滚动到新索引。

ISM根据**设置的间隔**（*而不是连续）在***每次执行策略***时检查操作条件。如果**在执行检查**时值**已达到**或*超过*配置的限制，则将执行滚动。例如，如果配置为 100GiB 的值，ISM 可能会检查 99 GiB 处的索引而不执行滚动。但是，如果在下次检查时索引已超过限制（例如 105GiB），则执行操作。`min_size`

如果需要跳过翻转操作，可以将索引设置`index.plugins.index_state_management.rollover_skip`为`true`。例如，如果收到错误消息"缺少别名或未写入索引...”，则可以将参数设置`index.plugins.index_state_management.rollover_skip`为`true`并重试跳过翻转操作。

索引格式必须符合以下模式：`^.*-\d+$`。例如，`(logs-000001)`。设置`index.plugins.index_state_management.rollover_alias`为 rollover 的别名。

范围	描述	类型	例子	必需
`min_size`	滚动索引所需的主分片存储空间（不计算副本）的最小大小。例如，如果设置`min_size`为 100 GiB，并	`string`	`20gb`或者`5mb`	否

	且索引有 5 个主分片和 5 个副本分片，每个分片大小为 20 GiB，则所有主分片的总大小为 100 GiB，因此会发生滚动。请参阅上面的**重要**说明。			
`min_primary_shard_size`	滚动索引所需的**单个主分片**的最小存储大小。例如，如果设置`min_primary_shard_size`为 30 GiB，并且索引中的**一个**主分片的大小大于条件，则会发生滚动。请参阅上面的**重要**说明。	`string`	`20gb`或者`5mb`	否
`min_doc_count`	滚动索引所需的最少文档数。请参阅上面的**重要**说明。	`number`	`2000000`	否
`min_index_age`	滚动索引所需的最短期限。索引期限是从创建到现在的时间。支持的单位为`d`（天）、`h`（小时）、`m`（分钟）、`s`（秒）、`ms`（毫秒）和`micros`（微秒）。请参阅上面的**重要**说明。	`string`	`5d`或者`7h`	否
`copy_alias`	控制是否将当前索引中的所有别名复制到新创建的索引。默认为`false`。	`boolean`	`true`或者`false`	否

```
{
  "rollover": {
    "min_size": "50gb"
  }
}
```

```
{
  "rollover": {
    "min_primary_shard_size": "30gb"
  }
}
```

```
{
  "rollover": {
    "min_doc_count": 100000000
  }
}
```

```
{
  "rollover": {
    "min_index_age": "30d"
  }
}
```

3.4.12 通知（notification）

发送通知。

范围	描述	类型	必需
destination	目标 URL。	Slack, Amazon Chime, or webhook URL	是
message_template	消息文本。	object	是

目标系统**必须**返回响应，否则通知操作将引发错误。

- 示例 1：CHIME 通知

```
{
  "notification": {
    "destination": {
      "chime": {
        "url": "<url>"
      }
    },
    "message_template": {
      "source": "the index is {{ctx.index}}"
    }
  }
}
```

- 示例 2：自定义 WEBHOOK 通知

```
{
  "notification": {
    "destination": {
      "custom_webhook": {
        "url": "https://<your_webhook>"
      }
    },
    "message_template": {
      "source": "the index is {{ctx.index}}"
    }
  }
}
```

- 示例 3：SLACK 通知

```
{
  "notification": {
    "destination": {
      "slack": {
        "url": "https://hooks.slack.com/services/xxx/xxxxxx"
      }
    },
    "message_template": {
      "source": "the index is {{ctx.index}}"
    }
  }
}
```

> *可以 `ctx` 在消息中使用变量来表示基于策略过去执行情况的多个策略参数。*

`ctx` 每项政策均提供以下变量选项：

范围	描述	类型
index	索引的名称。	string
index_uuid	索引的 uuid。	string
policy_id	策略的名称。	string

3.4.13 快照（snapshot）

备份集群的索引和状态。

该 snapshot 操作具有以下参数：

范围	描述	类型	必需	默认
repository	通过本机快照 API 操作注册的存储库名称。	string	是	-
snapshot	快照的名称。接受字符串和 Mustache 变量and。如果 Mustache 变量无效，则快照名称默认为索引的名称。	string或 Mustache 模板	是	-

```
{
  "snapshot": {
    "repository": "my_backup",
    "snapshot": ""
  }
}
```

3.4.14 索引优先级（index_priority）

设置特定状态下索引的优先级。只要有可能，未分配的索引分片将按其优先级顺序恢复。优先级值较高的索引将首先恢复，然后恢复优先级值较低的索引。

该 index_priority 操作具有以下参数：

范围	描述	类型	必需	默认
priority	索引进入某种状态时的优先级。	number	是	1

```
"actions": [
  {
    "index_priority": {
      "priority": 50
    }
  }
]
```

3.4.15 分配（allocation）

将索引分配给具有特定属性集的节点，如下所示。例如，设置 `require` 为 `warm` 仅将数据移动到"温"节点。

该 `allocation` 操作具有以下参数：

范围	描述	类型	必需
`require`	将索引分配给具有指定属性的节点。	`string`	是
`include`	将索引分配给具有任何指定属性的节点。	`string`	是
`exclude`	否要将索引分配给具有任何指定属性的节点。	`string`	是
`wait_for`	等待策略执行后再将索引分配给具有指定属性的节点。	`string`	是

```
"actions": [
  {
    "allocation": {
      "require": { "temp": "warm" }
    }
  }
]
```

3.4.16 卷起（rollup）

索引卷起可通过将旧数据汇总到汇总索引中来定期减少数据粒度。

> *汇总作业可以是连续的，也可以是不连续的。使用 ISM 策略创建的汇总作业只能是非连续的。*

- 路径和 HTTP 方法

```
PUT _plugins/_rollup/jobs/<rollup_id>
GET _plugins/_rollup/jobs/<rollup_id>
DELETE _plugins/_rollup/jobs/<rollup_id>
POST _plugins/_rollup/jobs/<rollup_id>/_start
POST _plugins/_rollup/jobs/<rollup_id>/_stop
GET _plugins/_rollup/jobs/<rollup_id>/_explain
```

- ISM 汇总策略示例

```
{
    "policy": {
        "description": "Sample rollup" ,
        "default_state": "rollup",
        "states": [
            {
                "name": "rollup",
                "actions": [
                    {
                        "rollup": {
                            "ism_rollup": {
                                "description": "Creating rollup
through ISM",
                                "target_index": "target",
                                "page_size": 1000,
                                "dimensions": [
                                    {
                                        "date_histogram": {
                                            "fixed_interval":
"60m",
                                            "source_field":
"order_date",
                                            "target_field":
```

```
"order_date",
                                        "timezone":
"America/Los_Angeles"
                                    }
                                },
                                {
                                    "terms": {
                                        "source_field":
"customer_gender",
                                        "target_field":
"customer_gender"
                                    }
                                },
                                {
                                    "terms": {
                                        "source_field":
"day_of_week",
                                        "target_field":
"day_of_week"
                                    }
                                }
                            ],
                            "metrics": [
                                {
                                    "source_field":
"taxless_total_price",
                                    "metrics": [
                                        {
                                            "sum": {}
                                        }
                                    ]
                                },
```

```
                                        {
                                            "source_field":
"total_quantity",
                                            "metrics": [
                                                {
                                                    "avg": {}
                                                },
                                                {
                                                    "max": {}
                                                }
                                            ]
                                        }
                                    ]
                                }
                            }
                        }
                    ],
                    "transitions": []
                }
            ]
        }
}
```

3.4.17 转换(transitions)

转移定义了状态改变需要满足的条件。当前状态下的所有操作完成后，策略开始检查转换条件。

ISM 按照转换的定义顺序对其进行评估。例如，如果您定义转换：[A,B,C,D]，ISM 会遍历此转换列表，直到找到一个计算结果为真的转换，然后停止并将下一个状态设置为该转换中定义的状态。在下一次执行时，ISM 将忽略其余转换并从新状态开始。

如果未在转换中指定任何条件并将其留空，则假定它相当于始终为真。这意味着策略在检查时将索引转换为此状态。

下表列出了可以为转换定义的参数。

范围	描述	类型	必需
`state_name`	满足条件时要转换到的状态的名称。	`string`	是
`conditions`	列出转变的条件。	`list`	是

该 `conditions` 对象具有以下参数:

范围	描述	类型	必需
`min_index_age`	转换所需的指数的最低年龄。	`string`	否
`min_rollover_age`	发生转移后过渡到下一状态所需的最低年龄。	`string`	否
`min_doc_count`	转换所需的索引的最小文档数。	`number`	否
`min_size`	转换所需的主分片存储总量（不计算副本）的最小大小。例如，如果设置`min_size`为 100 GiB，并且的索引有 5 个主分片和 5 个副本分片，每个分片大小为 20 GiB，则所有主分片的总大小为 100 GiB，因此的索引将转换到下一个状态。	`string`	否
`cron`	`cron`如果没有其他转换先发生，则触发转换的作业。	`object`	否
`cron.cron.expression`	`cron`触发转换的表达式。	`string`	是
`cron.cron.timezone`	触发转换的时区。	`string`	是

以下示例将索引转换为 `cold` 30 天后的状态:

```
"transitions": [
  {
    "state_name": "cold",
    "conditions": {
      "min_index_age": "30d"
    }
  }
]
```

ISM 根据设置的时间间隔检查每次执行策略时的条件。

此示例使用 `cron` 条件每周六太平洋时间 5:00 转换索引:

```
"transitions": [
  {
    "state_name": "cold",
    "conditions": {
      "cron": {
        "cron": {
          "expression": "* 17 * * SAT",
          "timezone": "America/Los_Angeles"
        }
      }
    }
  }
]
```

请注意，此条件不会在下午 5:00 准时执行；作业仍会根据设置执行 `job_interval` 。由于开始时间存在差异，以及在检查过渡条件之前完成操作所需的时间量，我们建议不要使用过于狭窄的 cron 表达式。例如，不要使用 `15 17 * * SAT` （星期六下午 5:15）。

本例中使用的一个小时的窗口通常就足够了，但可以将其增加到 2-3 小时，以避免错过窗口并不得不等待一周才能发生转换。或者，可以使用更广泛的表达方式，例如让转换 `* * * * SAT,SUN` 在周末的任何时间发生。

3.4.18 错误通知

如果托管索引失败，此操作会发送通知。它会使用自定义消息 `error_notification` 通知单个目标或通知渠道。

在策略级别设置错误通知：

```
{
  "policy": {
    "description": "hot warm delete workflow",
    "default_state": "hot",
    "schema_version": 1,
    "error_notification": { },
    "states": [ ]
  }
}
```

范围	描述	类型	必需
`destination`	目标 URL。	`Slack, Amazon Chime, or webhook URL`	如果`channel`没有指定，则为是
`channel`	通知渠道 ID	`string`	如果`destination`没有指定，则为是
`message_template`	消息文本。可以使用Mustache模板向消息添加变量。	`object`	是

目标系统**必须**返回响应，否则 `error_notification` 操作将引发错误。

- CHIME 通知

```
{
  "error_notification": {
    "destination": {
      "chime": {
        "url": "<url>"
      }
    },
    "message_template": {
      "source": "The index {{ctx.index}} failed during policy
execution."
    }
  }
}
```

- 示例 2：自定义 WEBHOOK 通知

```
{
  "error_notification": {
    "destination": {
      "custom_webhook": {
        "url": "https://<your_webhook>"
      }
    },
```

```
    "message_template": {
      "source": "The index {{ctx.index}} failed during policy
execution."
    }
  }
}
```

- 示例 3：SLACK 通知

```
{
  "error_notification": {
    "destination": {
      "slack": {
        "url": "https://hooks.slack.com/services/xxx/xxxxxx"
      }
    },
    "message_template": {
      "source": "The index {{ctx.index}} failed during policy
execution."
    }
  }
}
```

- 示例 4：使用通知渠道

```
{
  "error_notification": {
    "channel": {
      "id": "some-channel-config-id"
    },
    "message_template": {
      "source": "The index {{ctx.index}} failed during policy
execution."
    }
```

```
    }
}
```

3.5 ISM策略托管索引

使用托管索引操作更改或更新策略。

下表列出了托管索引操作的字段。

范围	描述	类型	必需	只读
name	托管索引策略的名称。	string	是	否
index	此策略管理的托管索引的名称。	string	是	否
index_uuid	索引的 uuid。	string	是	否
enabled	为true时候，托管索引由调度程序调度并运行。	boolean	是	否
enabled_time	上次启用托管索引的时间。如果托管索引进程已禁用，则此值为空。	timestamp	是	是
last_updated_time	管理索引的上次更新时间。	timestamp	是	是
schedule	托管索引作业的计划。	object	是	否
policy_id	此托管索引使用的策略的名称。	string	是	否
policy_seq_no	此托管索引使用的策略的序列号。	number	是	否
policy_primary_term	此管理索引使用的策略的主要术语。	number	是	否
policy_version	此托管索引使用的策略的版本。	number	是	是
policy	运行期间使用的策略的缓存 JSON policy_version。如果策略为空，则表示这是作业的第一次执行，并且读取/保存了最新的策略文档。	object	否	否
change_policy	关于要改变的政策和状态的信息。	object	否	否
policy_name	要更新到的策略的名称。要更新到最新版本，请将其设置为与当前版本相同policy_name。	string	否	是
state	托管索引完成更新后的状态。如果未指定状态，则假定策略结构未发生变化。	string	否	是

以下示例显示了托管索引策略：

```
{
  "managed_index": {
    "name": "my_index",
    "index": "my_index",
    "index_uuid": "sOKSOfkdsoSKeofjIS",
    "enabled": true,
    "enabled_time": 1553112384,
    "last_updated_time": 1553112384,
    "schedule": {
      "interval": {
        "period": 1,
        "unit": "MINUTES",
        "start_time": 1553112384
      }
    },
    "policy_id": "log_rotation",
    "policy_version": 1,
    "policy": {...},
    "change_policy": null
  }
}
```

3.5.1 更改政策

更改任何托管索引策略，但 ISM 有一些限制以确保策略更改不会破坏索引。

如果索引停留在当前状态，无法继续，想要立即更新其策略，请确保新策略包含与旧策略相同的状态（相同的名称、相同的操作、相同的顺序）。在这种情况下，即使策略正在执行操作，ISM 也会应用新策略。

如果更新策略时未包含相同状态，则 ISM 仅在当前状态下的所有操作执行完毕后才会更新策略。或者，可以在旧策略中选择一个特定状态，在此之后新策略生效。

4 常见配置修改和说明

4.0.1 端口修改

修改HTTP API 端口 (默认为 9200)

- 配置文件: ase安装目录/config/ase.yml
- ```
  http.port: 9200
  ```

修改Transport 端口 (默认为 9300)

- 配置文件: ase安装目录/config/ase.yml
- ```
  transport.port: 9300
  ```

4.0.2 ASE配置局域网或其他IP访问

ASE默认只能通过本机127.0.0.1或者localhost访问，如果需要局域网或不限制，可以通过修改监听地址实现。

- 配置文件: ase安装目录/config/ase.yml
- 配置项: `network.host`、`discovery.seed_hosts`和`cluster.initial_cluster_manager_nodes`
- 如下配置所示，支持局域网内通过`172.20.140.58`访问，如果不限制访问，可以修改为`0.0.0.0`

```
# ---------------------------------- Network -----------------------
------------
#
# Set the bind address to a specific IP (IPv4 or IPv6):
#
#network.host: 192.168.0.1
#
# Set a custom port for HTTP:
#
http.port: 9200
network.host: 172.20.140.58   #监听IP地址
```

```
discovery.seed_hosts: ["172.20.140.58"]
# 单节点部署（非集群模式），还要配置如下
cluster.initial_cluster_manager_nodes: ["172.20.140.58:9300"]
```

4.0.3 License验证

ase支持金蝶kbc和统一授权中心授权，其中kbc授权采用的授权文件进行授权，而统一授权中心则是通过统一授权中心申请授权。

4.0.3.1 KBC授权

4.0.3.1.1 KBC授权码获取

不知道本机授权码，可以直接启动ase，等待启动失败后，查看安装目录下的：`LICENSE-kbc_authorize_code.txt`文件，获取授权码

```
[root@localhost ase-2.12.1]# ./bin/ase
16:11:17.660 [main] ERROR org.opensearch.bootstrap.LicenseLoader - license 校验失败:Kbc license check failed, msg: .com
.apusic.com.google.gson.JsonSyntaxException: java.lang.IllegalStateException: Expected BEGIN_OBJECT but was STRING at l
ine 1 column 6
Exception in thread "main" java.lang.RuntimeException: com.apusic.base.license.core.exception.LicenseException: Kbc lic
ense check failed, msg: .com.apusic.com.google.gson.JsonSyntaxException: java.lang.IllegalStateException: Expected BEGI
N_OBJECT but was STRING at line 1 column 6
        at org.opensearch.bootstrap.LicenseLoader.check(LicenseLoader.java:84)
        at org.opensearch.bootstrap.OpenSearch.main(OpenSearch.java:90)
Caused by: com.apusic.base.license.core.exception.LicenseException: Kbc license check failed, msg: .com.apusic.com.goog
le.gson.JsonSyntaxException: java.lang.IllegalStateException: Expected BEGIN_OBJECT but was STRING at line 1 column 6
        at com.apusic.base.license.core.kbc.KbcLicenceValidator.check(KbcLicenceValidator.java:94)
        at org.opensearch.bootstrap.LicenseLoader.check(LicenseLoader.java:81)
        ... 1 more
[root@localhost ase-2.12.1]# ls
```

`LICENSE-kbc_authorize_code.txt`文件内容是以SZTY开头的字符串，此字符串就是当前机器的授权码。

4.0.3.1.2 通过上面获取的授权码和金蝶天燕申请ASE授权文件

4.0.3.1.3 导入申请后的授权文件

申请的kbc授权文件格式默认是lic，通过下面两个步骤执行license导入

- 上传到ase安装目录的licenses/kbc
- 重命名授权文件为： license.xml

4.0.3.1.4 导入后重新启动ASE即可

4.0.3.2 授权中心授权

获取授权中心服务器的IP和端口，可以通过配置统一授权服务器信息配置授权获取。

4.0.3.2.1 授权中心配置

ASE支持授权中心的方式有三种，包括配置文件、环境变量、JVM参数，如果设置了环境变量或者JVM参数，那么会忽略掉配置文件配置，下面是三种方式的配置说明:

- 配置文件: `config/apusic.properties`

```
# local
apusic.name=Apusic Search Engine
#apusic.authorization.type=local
#apusic.licenses.path=licenses

# kbc
# apusic.authorization.type=kbc
# apusic.licenses.path=licenses/kbc

# center
apusic.authorization.type=center
# 是否开启授权中心
apusic_acls_enable=true
# 授权中心地址 ip1:port1, ip2:port2
apusic_acls_authUrls=172.24.3.100:6869
# 租户，可选
apusic_acls_tenant=
# 命名空间，可选
apusic_acls_ns=
```

- 环境变量

```
export apusic_acls_enable=true
export apusic_acls_authUrls=172.24.3.100:6869
export apusic_acls_tenant=租户名称
export apusic_acls_ns=命名空间名称
```

- JVM参数

```
-Dapusic_acls_enable=true
-Dapusic_acls_authUrls=172.24.3.100:6869
-Dapusic_acls_tenant=租户名称
-Dapusic_acls_ns=命名空间名称
```

2. 配置好后，重启ase即可。

3. **注**:集群模式下，每个节点占用一个授权

4.0.4 SSL和设置admin用户密码

ASE支持使用安全的SSL通信和基于base-auth的用户验证。

4.0.4.1 启用SSL和用户验证

- 设置默认用户admin的密码,用户密码规范有下面几点（同时满足）：

1. 最少8个字符
2. 最少一个大写字符
3. 最少一个小写字符
4. 最少包含一个数字
5. 最少包含一个特殊字符

- 1、使用export设置环境变量方式配置admin用户密码为:Apusic@ase123

```
# 比如export设置的Apusic@ase123
export OPENSEARCH_INITIAL_ADMIN_PASSWORD=Apusic@ase123
```

- 2、执行初始化SSL证书和创建用户脚本: `./plugins/opensearch-security/tools/install_configuration.sh`
- 3、执行初始化脚本，出现 `Enable cluster mode? [y/N]` 请根据是否集群模式进行选择

```
./plugins/opensearch-security/tools/install_configuration.sh
OpenSearch Security Demo Installer
 ** Warning: Do not use on production or public reachable systems **
Install demo certificates? [y/N] y
Initialize Security Modules? [y/N] y
Cluster mode requires maybe additional setup of:
  - Virtual memory (vm.max_map_count)

Enable cluster mode? [y/N] n
Basedir: /Users/XXXXX/Test/opensearch-*
OpenSearch install type: .tar.gz on
OpenSearch config dir: /Users/XXXXX/Test/opensearch-*/config
OpenSearch config file: /Users/XXXXX/Test/opensearch-
*/config/opensearch.yml
```

```
OpenSearch bin dir: /Users/XXXXX/Test/opensearch-*/bin
OpenSearch plugins dir: /Users/XXXXX/Test/opensearch-*/plugins
OpenSearch lib dir: /Users/XXXXX/Test/opensearch-*/lib
Detected OpenSearch Version: x-content-*
Detected OpenSearch Security Version: *

### Success
### Execute this script now on all your nodes and then start all
nodes
### OpenSearch Security will be automatically initialized.
### If you like to change the runtime configuration
### change the files in ../config and execute:
"/Users/XXXXX/Test/opensearch-*/plugins/opensearch-
security/tools/securityadmin.sh" -cd "/Users/XXXXX/Test/opensearch-
*/config/opensearch-security/" -icl -key
"/Users/XXXXX/Test/opensearch-*/config/kirk-key.pem" -cert
"/Users/XXXXX/Test/opensearch-*/config/kirk.pem" -cacert
"/Users/XXXXX/Test/opensearch-*/config/root-ca.pem" -nhnv
### or run ./securityadmin_demo.sh
### To use the Security Plugin ConfigurationGUI
### To access your secured cluster open https://<hostname>:<HTTP
port> and log in with admin/<your-admin-password>.
### (Ignore the SSL certificate warning because we installed self-
signed demo certificates)
```

- 4、重启ASE
- 5、使用curl测试(把下面IP换成127.0.0.1或节点IP)

```
curl -X GET https://172.18.100.218:9200 -u "admin:Apusic@ase123" --
insecure
{
  "name" : "alb",
  "cluster_name" : "ase",
  "cluster_uuid" : "t1Y04ZldToyq7eQCH6391g",
  "version" : {
    "distribution" : "ase",
```

```
    "number" : "2.13.0",
    "build_type" : "tar",
    "build_hash" : "ac4ba974ce1402554b1b8aa1a5c0402d1d1b992e",
    "build_date" : "2024-07-25T02:23:42.986326873Z",
    "build_snapshot" : false,
    "lucene_version" : "9.9.2",
    "minimum_wire_compatibility_version" : "7.10.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "The ASE Project: https://www.apusic.com/"
}
```

4.0.5 禁用HTTPS

在config/ase.yml文件中的，把153行的plugins.security.ssl.http.enabled修改为false

```
plugins.security.ssl.transport.pemtrustedcas_filepath: root-ca.pem
plugins.security.ssl.transport.enforce_hostname_verification: false
plugins.security.ssl.http.enabled: false  # 这个修改false
plugins.security.ssl.http.pemcert_filepath: esnode.pem
```

修改完成后，重启ase即可。

4.0.6 Prometheus监控接口

ASE默认开启prometheus监控，在启动ASE成功后，可以通过浏览器输入 `http://节点IP:端口/_prometheus/metrics`

注 如果访问失败，请确保下面配置：

- ASE开启了局域网访问
- 使用https协议访问

4.0.6.1 SSL下Prometheus配置

如果ASE启用https和用户验证，需要配置以下的内容，配置样例如下：

```
job_name: 'spring-actuator'
  scheme: https
  basic_auth:
```

```
    username: 'ase用户名'
    password: 'ase用户密码'
  tls_config:
    insecure_skip_verify: true
    metrics_path: '/_promethues/metrics'
  scrape_interval: 5s
  static_configs:
    - targets: ['localhost:9200']
```

- basic_auth中的username和password需要替换成ASE的用户名和密码

4.0.6.2 metrics请求样例（使用curl）

- 未开启HTTPS和用户验证curl：`curl -X GET http://127.0.0.1:9200/_prometheus/metrics`
- 开启HTTPS和用户验证：`curl -X GET https://127.0.0.1:9200/_prometheus/metrics -u 'admin:Apusic@ase123' --insecure`

```
# HELP opensearch_indices_percolate_count Count of percolates
# TYPE opensearch_indices_percolate_count gauge
# HELP opensearch_index_translog_uncommitted_operations_number
Current number of uncommitted translog operations
# TYPE opensearch_index_translog_uncommitted_operations_number gauge
# HELP opensearch_indices_merges_total_size_bytes Count of bytes of
merged documents
# TYPE opensearch_indices_merges_total_size_bytes gauge
opensearch_indices_merges_total_size_bytes{cluster="ase",node="user-
PC",nodeid="f3I0E0GBQ32cC0bJXNuZ-A",} 0.0
# HELP opensearch_ingest_pipeline_processor_total_failed_count
Ingestion total failed
# TYPE opensearch_ingest_pipeline_processor_total_failed_count gauge
# HELP opensearch_jvm_mem_nonheap_committed_bytes Committed bytes
apart from heap
# TYPE opensearch_jvm_mem_nonheap_committed_bytes gauge
opensearch_jvm_mem_nonheap_committed_bytes{cluster="ase",node="user-
PC",nodeid="f3I0E0GBQ32cC0bJXNuZ-A",} 1.01777408E8
# HELP opensearch_fs_io_total_read_bytes Total IO read bytes
# TYPE opensearch_fs_io_total_read_bytes gauge
```

```
opensearch_fs_io_total_read_bytes{cluster="ase",node="user-
PC",nodeid="f3I0E0GBQ32cC0bJXNuZ-A",} 655360.0
# HELP opensearch_cluster_inflight_fetch_number Number of in flight
fetches
# TYPE opensearch_cluster_inflight_fetch_number gauge
opensearch_cluster_inflight_fetch_number{cluster="ase",} 0.0
# HELP opensearch_ingest_pipeline_total_current Ingestion total
current
# TYPE opensearch_ingest_pipeline_total_current gauge
# HELP opensearch_transport_tx_packets_count Sent packets
# TYPE opensearch_transport_tx_packets_count gauge
opensearch_transport_tx_packets_count{cluster="ase",node="user-
PC",nodeid="f3I0E0GBQ32cC0bJXNuZ-A",} 0.0
# HELP opensearch_ingest_pipeline_processor_total_count Ingestion
total number
# TYPE opensearch_ingest_pipeline_processor_total_count gauge
# HELP opensearch_index_get_current_number Current rate of get
commands
# TYPE opensearch_index_get_current_number gauge
# HELP opensearch_index_segments_memory_bytes Memory used by
segments
```

4.1 SDK 集成使用说明

ASE兼容elasticsearch7.0~7.10版本，不同语言对接使用ase过程中，可以**直接使用elasticsearch的客户端或驱动进行连接**，连接和配置方式可以参照elasticsearch接入说明即可。

4.1.1 Java连接和使用ASE样例

```
//ASE测试连接代码
public static void main(String[] args) throws Exception {
   //创建ASE客户端
   RestHighLevelClient aseClient = new RestHighLevelClient(
            //连接地址,接口,连接方式
            RestClient.builder(new
HttpHost("localhost",9200,"http"))
```

```
    );
    //关闭ASE客户端
    aseClient.close();
}


//创建索引
public static void main(String[] args) throws Exception {
    RestHighLevelClient aseClient = new RestHighLevelClient(
            //连接地址,接口,连接方式
            RestClient.builder(new
HttpHost("localhost",9200,"http"))
    );
    CreateIndexRequest user = new CreateIndexRequest("user");
    CreateIndexResponse response = aseClient.indices().create(user,
RequestOptions.DEFAULT);

    //响应状态
    boolean acknowledged = response.isAcknowledged();
    System.out.println("索引操作,"+ acknowledged);

    //关闭ase客户端
    aseClient.close();
}

// 查询索引
public static void main(String[] args) throws Exception {
    RestHighLevelClient aseClient = new RestHighLevelClient(
            //连接地址,接口,连接方式
            RestClient.builder(new
HttpHost("localhost",9200,"http"))
    );
    GetIndexRequest user = new GetIndexRequest("user");
    //获取目录数据
    GetIndexResponse getIndexResponse = aseClient.indices().get(user,
RequestOptions.DEFAULT);
```

```
    System.out.println(getIndexResponse.getMappings());
    System.out.println(getIndexResponse.getSettings());
    System.out.println(getIndexResponse.getAliases());
    //关闭ase客户端
    aseClient.close();
}

// 插入一条数据
public static void main(String[] args) throws Exception {
    RestHighLevelClient aseClient = new RestHighLevelClient(
            //连接地址,接口,连接方式
            RestClient.builder(new
HttpHost("localhost",9200,"http"))
    );
    //操作插入的index
    IndexRequest request = new IndexRequest("user");
    //设置id
    request.id("1001");
    User user1 = new User();
    user1.setName("张三");
    user1.setAge(20);
    user1.setAddress("中国");
    //将插入对象JSON化
    ObjectMapper objectMapper = new ObjectMapper();
    String userJson = objectMapper.writeValueAsString(user1);
    //指定插入数据的类型
    request.source(userJson, XContentType.JSON);
    //插入数据
    IndexResponse index = aseClient.index(request,
RequestOptions.DEFAULT);
    System.out.println(index.getResult());
    //关闭Es客户端
    aseClient.close();
}
```

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com