Elastic Search

**https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html**

**简介：**

Elasticsearch 和 关系型数据库的比较：

在Elasticsearch中，索引是类型的集合，因为数据库是RDBMS(关系数据库管理系统)中表的集合。每个表都是行的集合，就像每个映射都是JSON对象的Elasticsearch集合一样。

|  |  |
| --- | --- |
| Elasticsearch | RDBMS |
| 索引 \_index | 数据库 |
| 碎片 | 碎片 |
| 映射 \_type | 表 |
| 字段 \_file | 字段 |
| JSON对象 | 元组（一条数据） |

**安装：（Linux\_CentOs7下安装）**

1. **下载elasticsearch**

wget <https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.2.4.zip>

1. **创建非root用户组启动（如创建test组test用户）**

$unzip elasticsearch-6.2.4.zip #解压文件

$mv elasticsearch /usr/local/

$groupadd test #创建test用户组

$useradd test -g test -p test #创建test用户并分配到test分组

## 防止启动异常需要在root权限下做以下配置

$vim /etc/security/limits.conf

## 添加 如下： \* 为所有用户，如果只为test用户添加权限，则将\*改为test

\* soft nofile 65536

\* hard nofile 131072

\* soft nproc 2048

\* hard nproc 4096

$vim /etc/sysctl.conf

## 添加 如下：

vm.max\_map\_count = 655360

$sysctl -p ##重启配置

## 给用户添加文件夹权限

$chown -R test:test /usr/local/elasticsearch

1. **启动前准备（在/usr/local/elasticsearch目录下）**

$vim config/elasticsearch.yml #编辑配置文件

## 修改 如下：

打开注释 #cluster.name: my-application 改为 test（自定义）

打开注释 #network.host: 192.168.0.1 修改地址为 0.0.0.0 （允许外网访问）

1. **启动（在/usr/local/elasticsearch目录下）**

方式一：

$./bin/elasticsearch

Ctrl + c 结束运行

方式二：

$./bin/elasticsearch -d #后台启动

通过kill ps id 关闭

**安装 Q&A：**

API:

#### 查看集群状态 ####

TYPE : GET

URL : /\_cluster/health

SUCCESS :

{

"cluster\_name": "elasticsearch",

"status": "green", <1>

"timed\_out": false,

"number\_of\_nodes": 1,

"number\_of\_data\_nodes": 1,

"active\_primary\_shards": 0,

"active\_shards": 0,

"relocating\_shards": 0,

"initializing\_shards": 0,

"unassigned\_shards": 0

}

分析：

"status":

Green： 所有主分片和复制分片都可用

Yellow: 所有主分片可用，但不是所有复制分片都可用

Red： 都不可用

#### 创建索引（创建库） ####

TYPE : PUT

URL : <http://192.168.1.111:9200/website>

METHOD : /{\_index}

SUCCESS:

{

"acknowledged": true,

"shards\_acknowledged": true,

"index": "website"

}

FAILD：

{

"error": {

"root\_cause": [{

"type":   "resource\_already\_exists\_exception",

 "reason":   "index [website/pgHk87KQQRaJZDGMfsE0Ww] already exists",

 "index\_uuid":   "pgHk87KQQRaJZDGMfsE0Ww",

"index": "website"

}],

"type":   "resource\_already\_exists\_exception",

"reason":   "index [website/pgHk87KQQRaJZDGMfsE0Ww] already exists",

"index\_uuid":   "pgHk87KQQRaJZDGMfsE0Ww",

"index":   "website"

},

"status":  400

}

分析 :

创建时直接在访问地址添加索要创建的索引名即可创建成功;

#### 添加表和数据/更新数据 ####

TYPE : PUT

URL : http://192.168.1.111:9200/website/blog/1

METHOD : /{\_index}/{\_type}/{id}

PARAMTER :

{

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/01"

}

SUCCESS :

{

"\_index":   "website",

"\_type":   "blog",

"\_id":   "1",

"\_version":  1,

"result":   "created",

"\_shards": {

"total":  2,

"successful":  2,

"failed":  0

},

"\_seq\_no":  0,

"\_primary\_term":  1

}

ERROR :

分析 :

Successful : 2 创建\_type 和添加数据都成功，所以成功个数为2;

同一个请求重复发送则为UPDATE操作，并且返回结果的 result 及 \_version 会发生变化;

#### 检索文档（根据id查询一条数据）####

TYPE : GET

URL : http://192.168.1.111:9200/website/blog/1?pretty

METHOD : /{\_index}/{\_type}/{id}

SUCCESS :

{

"\_index": "website",

"\_type": "blog",

"\_id": "1",

"\_version": 3,

"found": true,

"\_source": {

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/03"

}

}

ERROR :

{

"\_index": "website",

"\_type": "blog",

"\_id": "2",

"found": false

}

分析：

参数pretty 美化输出\_source中的查询结果;

查询不到对应结果则响应状态为404并返回失败结果，found = false;

#### 检索文档（根据id查询某字段）####

TYPE : GET

URL : http://192.168.1.111:9200/website/blog/1?\_source=title,text

METHOD : /{\_index}/{\_type}/{id}?\_source={file[0],file[1]}

SUCCESS :

{

"\_index": "website",

"\_type": "blog",

"\_id": "1",

"\_version": 3,

"found": true,

"\_source": {

"text": "Just trying this out...",

"title": "My first blog entry"

}

}

分析 :

\_source=file[0],file[1] : 将要查询的字段用逗号隔开作为参数传入，如果某个file无对应字段则结果不展示该内容;

如果不需要其它元数据只要查询到的结果，URL可以改为：

http://192.168.1.111:9200/website/blog/1/\_source ,返回结果将只携带\_source中的内容

#### 删除文档(根据id删除数据) ####

TYPE : DELETE

URL : http://192.168.1.111:9200/website/blog/1

METHOD : /{\_index}/{\_type}/{id}

RESULT :

{

"\_index": "website",

"\_type": "blog",

"\_id": "1",

"\_version": 4,

"result": "deleted",

"\_shards": {

"total": 2,

"successful": 2,

"failed": 0

},

"\_seq\_no": 3,

"\_primary\_term": 1

}

ERROR :

{

"\_index": "website",

"\_type": "blog",

"\_id": "1",

"\_version": 1,

"result": "not\_found",

"\_shards": {

"total": 2,

"successful": 2,

"failed": 0

},

"\_seq\_no": 4,

"\_primary\_term": 1

}

分析 :

删除为物理删除，删除后再次查询将查询不到该数据;

#### 文档局部更新（表结构更新/数据更新）####

TYPE : POST

URL : http://192.168.1.111:9200/website/blog/1/\_update

METHOD : /{\_index}/{\_type}/{id}/\_update

PARAMTER :

{

"doc" : {

"tags" : [ "testing" ],

"views": 0

}

}

RESULT :

{

"\_index": "website",

"\_type": "blog",

"\_id": "1",

"\_version": 5,

"found": true,

"\_source": {

"title": "My first blog entry",

"text": "Just trying this out...",

"date": "2014/01/03",

"views": 0,

"tags": [

"testing"

]

}

}

分析 :

doc 文档主体部分，即更新的是文档的\_source部分;

#### 文档局部更新（通过脚本更新）####

TYPE : POST

URL : http://192.168.1.111:9200/website/blog/1/\_update

METHOD : /{\_index}/{\_type}/{id}/\_update

PARAMTER\_1 :

{

"script" : "ctx.\_source.views+=1"

}

PARAMTER\_2 :

{

"script": "ctx.\_source.tags+=new\_tag",

"params": {

"new\_tag": "search"

}

}

分析 :

script 脚本内容 ctx.\_source.{file}{操作符}{更新内容}

params 定义参数，可以重复使用

#### 空搜索 ####

DESCIRPT : 没有指定任何的查询条件，只返回集群索引中的所有文档

TYPE : GET

URL : http://192.168.1.111:9200/\_search

METHOD : /\_search

SUCCESS :

{

"hits": {

"total": 14,

"hits": [{

"\_index": "us",

"\_type": "tweet",

"\_id": "7",

"\_score": 1,

"\_source": {

"date": "2014-09-17",

"name": "John Smith",

"tweet": "The Query DSL is really powerful and flexible",

"user\_id": 2

}

},

...9 RESULTS REMOVED...

],

"max\_score": 1

},

"took": 4,

"\_shards": {

"failed": 0,

"successful": 10,

"total": 10

},

"timed\_out": false

}

分析 :

响应中最重要的部分是hits，它包含了total字段来表示匹配到的文档总数，hits数组还包含了匹配到的前10条数据。

每个节点都有一个\_score字段，这是相关性得分(relevance score)，它衡量了文档与查询的匹配程度。默认的，返回的结果中关联性最大的文档排在首位；这意味着，它是按照\_score降序排列的。这种情况下，我们没有指定任何查询，所以所有文档的相关性是一样的，因此所有结果的\_score都是取得一个中间值1

took took告诉我们整个搜索请求花费的毫秒数。

shards \_shards节点告诉我们参与查询的分片数（total字段），有多少是成功的（successful字段），有多少的是失败的（failed字段）。通常我们不希望分片失败，不过这个有可能发生。如果我们遭受一些重大的故障导致主分片和复制分片都故障，那这个分片的数据将无法响应给搜索请求。这种情况下，Elasticsearch将报告分片failed，但仍将继续返回剩余分片上的结果。

timeout time\_out值告诉我们查询超时与否。一般的，搜索请求不会超时。如果响应速度比完整的结果更重要，你可以定义timeout参数为10或者10ms（10毫秒），或者1s（1秒）

#### 多索引和多类别 ####

假如现有不同类型的文档[ user ]和[ tweet ]来自于不同的索引[ us ]和[ gb ]。

TYPE : GET

URL :

/\_search

在所有索引的所有类型中搜索

/gb/\_search

在索引gb的所有类型中搜索

/gb,us/\_search

在索引gb和us的所有类型中搜索

/g\*,u\*/\_search

在以g或u开头的索引的所有类型中搜索

/gb/user/\_search

在索引gb的类型user中搜索

/gb,us/user,tweet/\_search

在索引gb和us的类型为user和tweet中搜索

/\_all/user,tweet/\_search

在所有索引的user和tweet中搜索 search types user and tweet in all indices

当你搜索包含单一索引时，Elasticsearch转发搜索请求到这个索引的主分片或每个分片的复制分片上，然后聚集每个分片的结果。搜索包含多个索引也是同样的方式——只不过或有更多的分片被关联。

#### 分页 ####

TYPE : GET

URL : http://192.168.1.111:9200/\_search?size=1&from=0

METHOD : /\_search?size=1&from=0

SUCCESS :

{

"took": 34,

"timed\_out": false,

"\_shards":{

"total": 6,

"successful": 6,

"skipped": 0,

"failed": 0

},

"hits":{

"total": 3,

"max\_score": 1,

"hits":[

{"\_index": "website", "\_type": "blog", "\_id": "2", "\_score": 1,…}

]

}

}

分析 :

默认size = 10 default = 0 即从0开始，每页10条记录;

添加了分页以后还是能从hits->total 中查看到总条数;

**处理在集群系统中深度分页**