### elasticSearch在window 7 安装

下载地址：<https://www.elastic.co/downloads/past-releases>

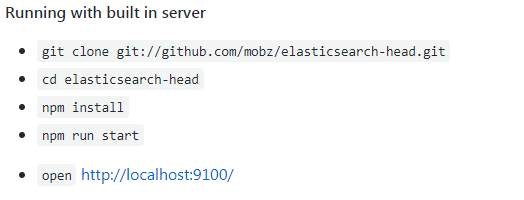
找不带hadoop版本的。

解压，运行elasticsearch.bat即可；

在浏览器中输入:http://localhost:9200/，说明安装成功了

安装elasticsearch-head

参考<https://github.com/mobz/elasticsearch-head#running-with-built-in-server>



要使用npm，必须先安装node.js

启动：elasticsearch-5.5.2\bin\elasticsearch.bat即可，如果要将ES作为一个后台进程运行，加参数 -d 即 ./elasticsearch-5.5.2/bin/elasticsearch.bat -d

### elasticsearch-php安装

<https://www.elastic.co/guide/en/elasticsearch/client/php-api/current/index.html>

* 1. 前提

安装composer

php.ini中开启curl和openssl

安装java8

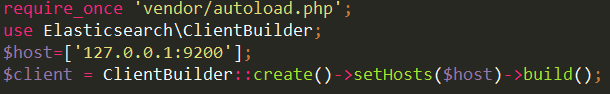
* 1. 过程可参考<https://www.elastic.co/guide/en/elasticsearch/client/php-api/current/_quickstart.html>

新建目录test，里面建立composer.json,并把composer.json拷贝过来

cmd运行php composer.phar install --no-dev

自动下载vender目录，之后即可使用。

注意要设置host：



### 基础概念

##### 文档

Document：文档，Json结构，这点跟MongoDB差不多。通常可以认为对象(object)和文档(document)是等价相通的，对象(Object)中还可能包含其他对象(Object)。 在Elasticsearch中，文档(document)特指最顶层结构或者根对象(root object)序列化成的JSON数据（以唯一ID标识并存储于Elasticsearch中）

文档中三个必须的元数据节点：

| **节点** | **说明** |
| --- | --- |
| \_index | 文档存储的地方 |
| \_type | 文档代表的对象的类 |
| \_id | 文档的唯一标识 |

\_index 索引 类似于关系型数据库里的“数据库” ，存储和索引关联数据的地方，必须是全部小写，不能以下划线开头

\_type 类定义了属性或与对象关联的数据，相当于“表”， 可以是大写或小写，不能包含下划线或逗号

\_id 与\_index和\_type组合时，就可以在Elasticsearch中唯一标识一个文档

##### 索引

使用自己的id：

PUT /{index}/{type}/{id}

{

"field": "value",

...

}

使用自增id：

POST /website/blog/

{

"field": "value",

...

}

##### 检索

GET /website/blog/123?pretty

响应返回元数据节点，以及\_source字段，即创建索引时发送的原始文档

|  |
| --- |
| {  "\_index" : "website",  "\_type" : "blog",  "\_id" : "123",  "\_version" : 1,  "found" : true,  "\_source" : {  "title": "My first blog entry",  "text": "Just trying this out...",  "date": "2014/01/01"  }  } |

pretty

在查询字符串中添加pretty参数，会让elasticsearch美化输出，使json更加容易阅读

"found" : true 意味文档被找到了

只想返回某几个字段，请求为：

GET /website/blog/123?\_source=title,text

只想得到\_source字段，不要其他元数据，\_source可以不带参数

GET /website/blog/123/\_source，（这里不是？了）只返回

|  |
| --- |
| {  "title": "My first blog entry",  "text": "Just trying this out...",  "date": "2014/01/01"  } |

##### 检查文档存在

只检查文档是否存在：

HEAD <http://localhost:9200/website/blog/123>

只会返回HTTP头：

若存在，返回200 ok状态

不存在，返回404 Not Found

##### 更新文档

PUT /website/blog/123

{

"title": "My first blog entry",

"text": "I am starting to get the hang of this...",

"date": "2014/01/02"

}

文档在elasticsearch中不能修改，若需要更新文档，可以重建索引或者替换掉文档

以上实例是在原来的索引上提换掉文档（删除旧文档，添加新文档），因为索引id没变，所以返回的“created”为false，且\_version加了1；

|  |
| --- |
| {  "\_index": "website",  "\_type": "blog",  "\_id": "123",  "\_version": 2,  "result": "updated",  "\_shards": {  "total": 2,  "successful": 1,  "failed": 0  },  "created": false  } |

created标识为false因为同索引、同类型下已经存在同ID的文档。

##### 创建新文档

当索引一个文档时，可能是新创建的，也可能是覆盖了原来的；

解决：

1. 使用自增id， POST /website/blog/ 请求
2. 自定义id

PUT /website/blog/123?op\_type=create

PUT /website/blog/123/\_create

若成功创建了一个新文档，将返回正常的元数据，且响应码是201 created

若包含相同的\_index、\_type和\_id的文档已经存在，返回409 Conflict响应状态码

##### 删除文档

DELETE /website/blog/123

文档找到被那个被删除，返回“found”：true

未找到，返回“found”：false

##### 版本控制

乐观锁：

\_version在文档改变时都会加1

当更新一个文档时，指定他的版本号，若是当前的（current）版本号，则可以更新，若是旧的版本号，则返回409 conflict冲突。

当修改文档时，要带上读取时获得的版本号

PUT /website/blog/1?version=1 //只让\_version是1时才能更新，否则冲突

外部版本号：

若主数据库有版本字段，类似于timestamp等可用于版本控制的字段，就可以在elasticsearch查询字符串后面加上version\_type=external来使用版本号。版本号必须是整数。

外部版本号和上面的内部版本号不同，不在检查\_version是否与请求中指定的一致，而是检查当前版本的\_version是否小于你指定的版本，若小于，外部版本号就存储到\_version，通过验证。

外部版本号可以在索引，删除或创建新文档的时候指定。

例如：创建外部版本号为5的新博客

|  |
| --- |
| PUT /website/blog/2?version=5&version\_type=external  {  "title": "My first external blog entry",  "text": "Starting to get the hang of this..."  } |

##### 文档局部更新

使用doc参数，存在的 字段被覆盖，新字段被追加

例：

|  |
| --- |
| POST /website/blog/1/\_update  {  "doc" : {  "tags" : [ "testing" ],  "views": 0  }  } |

##### 检索多个文档

使用docs参数

|  |
| --- |
| POST /\_mget  {  "docs" : [  {  "\_index" : "website",  "\_type" : "blog",  "\_id" : 2  },  {  "\_index" : "website",  "\_type" : "pageviews",  "\_id" : 1,  "\_source": "views"  }  ]  } |

响应体也包括docs数组，每个数组跟单独获取时相同。

若在同一个\_index和\_type下，可以：

|  |
| --- |
| POST /website/blog/\_mget  {  "docs" : [  { "\_id" : 2 },  { "\_type" : "pageviews", "\_id" : 1 } //这里可以覆盖元数据里的\_type  ]  }  亦可：  POST /website/blog/\_mget  {  "ids" : [ "2", "1" ]  } |

每个文档的检索都是独立的，不存在时found返回false，不影响其他文档。

##### 批量更新

POST /\_bulk

{ action: { metadata }}

{ request body }

{ action: { metadata }}

{ request body }

每一行后的\n都不能省略，action/metadata定义了行为action发生在哪个文档document上，action必须是以下几种：

| **行为** | **解释** |
| --- | --- |
| create | 当文档不存在时创建之。见3.6 |
| index | 创建新文档或替换已有文档。见3.2 |
| update | 局部更新文档。见《局部更新》 |
| delete | 删除一个文档。见《删除文档》 |

以上四种操作，必须指定文档的\_index、\_type、\_id这些元数据(metadata)。

请求体request body 在delete时不需要，其他情况都是必需的。

create 当文档不存在时创建 \_id是必需的

index 可能是新创建文档，或替换已有文档，\_id不必需，若不指定id，会自动创建。

|  |
| --- |
| POST /\_bulk  { "delete": { "\_index": "website", "\_type": "blog", "\_id": "123" }} <1>  { "create": { "\_index": "website", "\_type": "blog", "\_id": "123" }}  { "title": "My first blog post" }  { "index": { "\_index": "website", "\_type": "blog" }}  { "title": "My second blog post" }  { "update": { "\_index": "website", "\_type": "blog", "\_id": "123", "\_retry\_on\_conflict" : 3} }  { "doc" : {"title" : "My updated blog post"} } <2> |

可以在请求的URL中使用/\_index/\_type, metadata就不用指明他们了。

但metadata里也可对URL里的\_index和\_type重写。

|  |
| --- |
| POST /website/log/\_bulk  { "index": {}}  { "event": "User logged in" }  { "index": { "\_type": "blog" }}  { "title": "Overriding the default type" } |

### 搜索

##### 空搜索

GET /\_search 返回所有文档

返回结果中： hits包含 1）total：匹配到的文档总数 （含不显示的）

1. hits数组：匹配到的前10条数据，每条数据里都包含元数据和\_source(原始文档)，且有个\_score字段（相关性得分），衡量文档与查询的匹配程度。默认是按\_score的降续排列的
2. took 整个搜索花费的毫秒数

##### 多索引和多类别

/\_search 在所有索引的所有类型中搜索

/gb/\_search 在gb索引的所有类型中搜索

/gb,us/\_search 在gb和us索引的所有类型中搜索

/g\*,u\*/\_search 在以g和u开头的所有索引的所有类型中搜索

/gb/user/\_search 在gb索引的user类型中搜索

/gb,us/user,tweet/\_search 在gb,us索引 user和tweet类型中搜索

/\_all/user,tweet/\_search 在所有索引的user和tweet中搜索

##### 分页

from：从第几条数据开始返回 默认从0开始

size：返回的结果数

/\_search?from=4&size=10

注意：分页不可过深，详情见文档。

##### 查询字符串

1）GET /\_search?q=mary

直接搜索包含“mary”字符的所有文档。

2）GET /\_all/tweet/\_search?q=tweet:elasticsearch+name:john +tweet:mary

tweet是字段名，elasticsearch是搜索关键字

这样会搜出来同时满足三个搜索条件的所有文档

+ 必须被满足

* 必须不被满足

没有修饰符表示不必须

当没有指定字段时，elasticsearch把所有字段值连接起来放在一个大字符串里，被索引为一个特殊的字段\_all。若没有指明字段，就使用\_all字段搜索。若指定了字段，则在字段里搜索。

|  |
| --- |
| {  "tweet": "However did I manage before Elasticsearch?",  "date": "2014-09-14",  "name": "Mary Jones",  } |

\_all字段 However did I manage before Elasticsearch? 2014-09-14 Mary Jones

### 结构化查询

##### 请求体查询

空查询：

|  |  |
| --- | --- |
| GET | /\_search |
| {} |  |

分页查询：

|  |
| --- |
| GET /\_search |
| { |
| "from": 30, |
| "size": 10 |
| } |

elasticsearch允许使用GET携带请求体传递数据，当然，也可以使用POST传递数据。

|  |
| --- |
| POST /\_search |
| { |
| "from": 30, |
| "size": 10 |
| } |

##### 结构化查询 Query DSL

|  |
| --- |
| GET /\_search  {  "query": YOUR\_QUERY\_HERE  } |

##### 重要的查询子句

term过滤：精确匹配哪些值

|  |
| --- |
| { "term": { "age": 26 }}  { "term": { "date": "2014-09-01" }} |

terms过滤： 可以指定多个查询条件

|  |
| --- |
| { |
| "terms": { |
| "tag": [ "search", "full\_text", "nosql" ] |
| } |
| } |

range过滤：按指定范围查找

|  |
| --- |
| {  "range": {  "age": {  "gte": 20,  "lt": 30  }  }  } |

exists 和 missing 过滤：用于查找文档中是否包含指定字段或没有某个字段，类似于SQL语句中的IS\_NULL条件

|  |
| --- |
| {  "exists": {  "field": "title"  }  } |

bool过滤：合并多个过滤条件查询结果的布尔逻辑，它包含以下操作符：

must :: 多个查询条件的完全匹配,相当于 and。

must\_not :: 多个查询条件的相反匹配，相当于 not。

should :: 至少有一个查询条件匹配, 相当于 or。

|  |
| --- |
| {  "bool": {  "must": { "term": { "folder": "inbox" }},  "must\_not": { "term": { "tag": "spam" }},  "should": [  { "term": { "starred": true }},  { "term": { "unread": true }}  ]  }  } |

match\_all查询：查询所有文档

|  |
| --- |
| {  "match\_all": {}  } |

match查询：

multi\_match查询：

bool查询：

### 使用教程

https://es.xiaoleilu.com/030\_Data/15\_Get.html

<http://www.yiibai.com/elasticsearch/elasticsearch-getting-start.html>

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