logstash -e “input { stdin { } } output { stdout {} }”

D:\logstash-6.4.0\bin>logstash -e "input { stdin { } } output { stdout { codec => rubydebug } }"

以上示例通过重新设置了叫”stdout”的output(添加了”codec”参数)，我们就可以改变Logstash的输出表现。类似的我们可以通过在你的配置文件中添加或者修改inputs、outputs、filters，就可以使随意的格式化日志数据成为可能，从而订制更合理的存储格式为查询提供便利。

Ctrl+c退出

Logstash 是开源的服务器端数据处理管道，能够同时从多个来源采集数据，转换数据，然后将数据发送到您最喜欢的 “存储库” 中

Beats 平台集合了多种单一用途数据采集器。这些采集器安装后可用作轻量型代理，从成百上千或成千上万台机器向 Logstash 或 Elasticsearch 发送数据

Beats 是数据采集的得力工具。将这些采集器安装在您的服务器中，它们就会把数据汇总到 Elasticsearch。如果需要更加强大的处理性能，Beats 还能将数据输送到 Logstash 进行转换和解析。

Edit the filebeat.yml configuration file

Start the daemon by running sudo ./filebeat -e -c filebeat.yml -d "publish"

打开解压后的目录，打开filebeat.yml进行配置。

①：配置 Filebeat prospectors->path 这里的路径是所要收集日志的路径

②：配置 enabled: true 这个配置很重要，只有配置为true之后配置才可生效，否则不起作用。

③：配置Outputs ，这里的Outputs有elasticsearch，logstash。按照配置文件下面的示例配置即可

To verify your configuration, run the following command:

bin/logstash -f first-pipeline.conf --config.test\_and\_exit

If the configuration file passes the configuration test, start Logstash with the following command:

bin/logstash -f first-pipeline.conf --config.reload.automatic

if pipeline.conf has been changed , Because you’ve enabled automatic config reloading, you don’t have to restart Logstash to pick up your changes. 删除data文件夹下的registry文件，重新启动filebeat即可——sudo ./filebeat -e -c filebeat.yml -d "publish"

Try a test query to Elasticsearch based on the fields created by the grok filter plugin. Replace $DATE with the current date, in YYYY.MM.DD format:

curl -XGET 'localhost:9200/logstash-$DATE/\_search?pretty&q=response=200'

http://localhost:9200/logstash-2018.09.19/\_search?pretty&q=geoip.city\_name= Buffalo

To see a list of available indexes, use this query: curl 'localhost:9200/\_cat/indices?v'.

Kibana

localhost:5601

Dev Tools:

GET /logstash-2018.09.20/\_search

{

"query":{

"match": {

"user\_name": "admin"

}

}

}

logstash-input-jdbc 全量

全量同步是指全部将数据同步到es，通常是刚建立es，第一次同步时使用。增量同步是指将后续的更新、插入记录同步到es。

input {

jdbc {

jdbc\_driver\_library => "postgresql-42.2.2.jar"

jdbc\_driver\_class => "org.postgresql.Driver"

jdbc\_connection\_string => "jdbc:postgresql://localhost:5432/equip"

jdbc\_user => "postgres"

jdbc\_password => "Abc12345"

schedule =>"\* \* \* \* \*"

statement => "SELECT d.id, d.info, u.user\_name, t.type\_name, m.device\_name from tb\_device\_detail d left join tb\_user u on d.user\_id=u.id left join tb\_device\_map m on d.device\_id = m.id left join tb\_device\_type t on m.type\_code = t.type\_code"

}

}

filter {

}

output {

elasticsearch {

hosts => [ "localhost:9200" ]

index => "myequip"

document\_id => "%{id}" #可以保证document\_id相同，而不会在每次每次查询更新时加入id不同而内容相同的document。

} file {

path => "/path/filejdbc"

}

}

Logstash-input-jdbc 增量

input {

jdbc {

jdbc\_driver\_library => "postgresql-42.2.2.jar"

jdbc\_driver\_class => "org.postgresql.Driver"

jdbc\_connection\_string => "jdbc:postgresql://localhost:5432/equip"

jdbc\_user => "postgres"

jdbc\_password => "Abc12345"

use\_column\_value => true

tracking\_column => id

schedule =>"\* \* \* \* \*"

record\_last\_run => true

last\_run\_metadata\_path => "D:\station\_parameter.txt"

statement => "SELECT d.id as id, d.info, u.user\_name, t.type\_name, m.device\_name from tb\_device\_detail d left join tb\_user u on d.user\_id=u.id left join tb\_device\_map m on d.device\_id = m.id left join tb\_device\_type t on m.type\_code = t.type\_code where d.id>:sql\_last\_value"

}

}

Logstash-input-jdbc插件不支持物理删除的同步更新。详见：

<http://stackoverflow.com/questions/35813923/sync-postgresql-data-with-elasticsearch/35823497#35823497>

如果想要实现删除操作，可以在表中多增加一个字段，比如当delete\_state=0时，为存在；当为1时，表示已经删除。

Elasticsearch概念

Index

An index is identified by a name (that must be all lowercase) and this name is used to refer to the index when performing indexing, search, update, and delete operations against the documents in it.

Type

A type used to be a logical category/partition of your index to allow you to store different types of documents in the same index, eg one type for users, another type for blog posts. It is no longer possible to create multiple types in an index, and the whole concept of types will be removed in a later version.

Document

A document is a basic unit of information that can be indexed. For example, you can have a document for a single customer, another document for a single product, and yet another for a single order. This document is expressed in JSON. Note that although a document physically resides in an index, a document actually must be indexed/assigned to a type inside an index.

Shards & replicas

To summarize, each index can be split into multiple shards. An index can also be replicated zero (meaning no replicas) or more times.

To check the cluster health, we will be using the [\_cat API](https://www.elastic.co/guide/en/elasticsearch/reference/6.4/cat.html).

GET /\_cat/health?v

GET /\_cat/nodes?v

GET /\_cat/indices?v

建一个index

Now let’s create an index named "customer" and then list all the indexes again:

PUT /customer?pretty

GET /\_cat/indices?v

Index and Query a Document

PUT /customer/\_doc/1?pretty

{

"name": "John Doe"

}

GET /customer/\_doc/1?pretty

And the response:

{

"\_index" : "customer",

"\_type" : "\_doc",

"\_id" : "1",

"\_version" : 1,

"found" : true,

"\_source" : { "name": "John Doe" }

}

Nothing out of the ordinary here other than a field, found, stating that we found a document with the requested ID 1 and another field, \_source, which returns the full JSON document that we indexed from the previous step.

Delete an Index

DELETE /customer?pretty

DELETE /customer/\_doc/1?pretty

If we study the above commands carefully, we can actually see a pattern of how we access data in Elasticsearch. That pattern can be summarized as follows:

<REST Verb> /<Index>/<Type>/<ID>

Indexing/Replacing Documents

PUT /customer/\_doc/1?pretty

{"name": "John Doe"}

PUT /customer/\_doc/1?pretty

{"name": "Jane Doe"}

GET /customer/\_doc/1?pretty查出来的结果中，\_version对应的属性值会加1.

Note that in the above case, we are using the POST verb instead of PUT since we didn’t specify an ID

POST /customer/\_doc?pretty

{

"name": "Jane Doe"

}

GET /customer/\_search?pretty&q=name=Jane

Updating Documents

POST /customer/\_doc/1/\_update?pretty

{ "doc": { "name": "Jane Doe" } }

POST /customer/\_doc/1/\_update?pretty

{ "doc": { "name": "Jane Doe", "age": 20 } }

POST /customer/\_doc/1/\_update?pretty

{ "script" : "ctx.\_source.age += 5" }

POST /customer/\_doc/1/\_update?pretty

{ "script" : "ctx.\_source.age += 5"}

In the above example, ctx.\_source refers to the current source document that is about to be updated.

GET /customer/\_doc/1?pretty

Batch processing

POST /customer/\_doc/\_bulk?pretty

{"index":{"\_id":"1"}}

{"name": "John Doe" }

{"index":{"\_id":"2"}}

{"name": "Jane Doe" }

POST /customer/\_doc/\_bulk?pretty

{"update":{"\_id":"1"}}

{"doc": { "name": "John Doe becomes Jane Doe" } }

{"delete":{"\_id":"2"}}

The Bulk API does not fail due to failures in one of the actions. If a single action fails for whatever reason, it will continue to process the remainder of the actions after it.

Loading the Sample Dataset

curl -H "Content-Type: application/json" -XPOST "localhost:9200/bank/\_doc/\_bulk?pretty&refresh" --data-binary "@accounts.json"

accounts.json是提前下好的json格式文件，与curl.exe放在同一文件夹下。

The Search API

 There are two basic ways to run searches: one is by sending search parameters through the [REST request URI](https://www.elastic.co/guide/en/elasticsearch/reference/6.4/search-uri-request.html) and the other by sending them through the [REST request body](https://www.elastic.co/guide/en/elasticsearch/reference/6.4/search-request-body.html).

GET /bank/\_search?q=\*&sort=account\_number:asc&pretty

This example returns all documents in the bank index: 下面效果相同

GET /bank/\_search

{

"query": { "match\_all": {} },

"sort": [

{ "account\_number": "asc" }

]

}

As for the response, we see the following parts:

* took – time in milliseconds for Elasticsearch to execute the search
* timed\_out – tells us if the search timed out or not
* \_shards – tells us how many shards were searched, as well as a count of the successful/failed searched shards
* hits – search results
* hits.total – total number of documents matching our search criteria
* hits.hits – actual array of search results (defaults to first 10 documents)
* hits.sort - sort key for results (missing if sorting by score)
* hits.\_score and max\_score - ignore these fields for now
* GET /bank/\_search
* {
* "query": { "match\_all": {} },
* “from”: 10,
* “size”: 10,
* "sort": { "balance(属性名)": { "order": "desc" } }
* }

只显示\_source下的这两个属性

GET /bank/\_search

{

"query": { "match\_all": {} },

"\_source": ["account\_number", "balance"]

}

GET /bank/\_search

{

"query": { "match": { "account\_number": 20 } }

}

This example returns all accounts containing the term "mill" or "lane" in the address:

GET /bank/\_search

{

"query": { "match": { "address": "mill lane" } }

}

This example is a variant of match (match\_phrase) that returns all accounts containing the phrase "mill lane" in the address:

GET /bank/\_search

{

"query": { "match\_phrase": { "address": "mill lane" } }

}

This example composes two match queries and returns all accounts containing "mill" and "lane" in the address:

GET /bank/\_search

{

"query": {

"bool": {

"must": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}

效果同上match\_phrase

In contrast, this example composes two match queries and returns all accounts containing "mill" or "lane" in the address:

GET /bank/\_search

{

"query": {

"bool": {

"should": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}

GET /bank/\_search

{

"query": {

"bool": {

"must\_not": [

{ "match": { "address": "mill" } },

{ "match": { "address": "lane" } }

]

}

}

}

GET /bank/\_search

{

"query": {

"bool": {

"must": [

{ "match": { "age": "40" } }

],

"must\_not": [

{ "match": { "state": "ID" } }

]

}

}

}

GET /bank/\_search

{

"query": {

"bool": {

"must": { "match\_all": {} },

"filter": {

"range": {

"balance": {

"gte": 20000,

"lte": 30000

}

}

}

}

}

}

Executing Aggregations（聚合函数）

GET /bank/\_search

{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "state.keyword"

}

}

}

}

In SQL, the above aggregation is similar in concept to:

SELECT state, COUNT(\*) FROM bank GROUP BY state ORDER BY COUNT(\*) DESC LIMIT 10;

Note that we set size=0 to not show search hits because we only want to see the aggregation results in the response.

GET /bank/\_search

{

"size": 0,

"aggs": {

"group\_by\_state": {

"terms": {

"field": "state.keyword",

"order": {

"average\_balance": "desc"

}

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

GET /bank/\_search

{

"size": 0,

"aggs": {

"group\_by\_age": {

"range": {

"field": "age",

"ranges": [

{

"from": 20,

"to": 30

},

{

"from": 30,

"to": 40

},

{

"from": 40,

"to": 50

}

]

},

"aggs": {

"group\_by\_gender": {

"terms": {

"field": "gender.keyword"

},

"aggs": {

"average\_balance": {

"avg": {

"field": "balance"

}

}

}

}

}

}

}

}

Set up Elasticsearch

Elasticsearch is built using Java, and requires at least [Java 8](http://www.oracle.com/technetwork/java/javase/downloads/index.html) in order to run. Only Oracle’s Java and the OpenJDK are supported. The same JVM version should be used on all Elasticsearch nodes and client

Set up Elasticsearch

在elasticsearch.yml中配置action.auto\_create\_index属性

If you are using [Logstash](https://www.elastic.co/products/logstash) or [Beats](https://www.elastic.co/products/beats) then you will most likely require additional index names in your action.auto\_create\_index setting, and the exact value will depend on your local configuration. If you are unsure of the correct value for your environment, you may consider setting the value to \* which will allow automatic creation of all indices

Any settings that can be specified in the config file can also be specified on the command line, using the -E syntax as follows:

.\bin\elasticsearch.bat -Ecluster.name=my\_cluster -Enode.name=node\_1

Elasticsearch can be installed as a service to run in the background or start automatically at boot time without any user interaction.

elasticsearch-service.bat install|remove|start|stop|manager [SERVICE\_ID]

使用聚合函数时，报错误，提示需要将fielddate赋值为true，

原因是应该是5.x后对排序，聚合这些操作用单独的数据结构(fielddata)缓存到内存里了，需要单独开启，

PUT company/\_mapping/employee/

{

"properties": {

"country": {

"type": "text",

"fielddata": true

}

}

}

record\_last\_run => true

clean\_run => false

last\_run\_metadata\_path => "D:\station\_parameter.txt"

多字段查询：

GET /mycase/\_search

{

"query": {

"multi\_match": {

"query": "商号",

"fields":["casebrief","caseno","casecode"]

}

}

}

全匹配的文档占的评分比较高，则需要使用best\_fields

GET /mycase/\_search

{

"query": {

"multi\_match": {

"query": "商号",

"type": "best\_fields",

"fields":["casebrief","caseno","casecode"],

"tie\_breaker": 0.3

}

}

}

越多字段匹配的文档评分越高，就要使用most\_fields

GET /mycase/\_search

{

"query": {

"multi\_match": {

"query": "商号",

"type": "most\_fields",

"fields":["casebrief","caseno","casecode"]

}

}

}

GET /mycase/\_search

{

"query": {

"bool": {

"must": {

"multi\_match": {

"query": "号",

"fields":["casebrief","caseno","casecode"]

}

},

"filter": {

"range": {

"updatetime": {

"gte": "2018-01-30",

"lte": "2018-09-25"

}

}

}

}

},

"highlight": {

"fields": {

"casebrief": {},

"casecode": {},

"caseno": {}

}

}

}

GET /mycase/\_search

{

"query": {

"bool": {

"should": [

{ "match\_phrase": { "casebrief": "商号" } },

{ "match\_phrase": { "caseno": "商号" } }

]

}

},

"from": 0,

"size": 50

}

通过插件intest-attachment将文档编码后导入到elasticsearch。

安装：bin/elasticsearch-plugin install ingest-attachment

安装后重启es，则可以使用此插件

源字段必须是Base64编码二进制。如果不想引起在base64之间来回转换的开销，可以使用CBOR格式代替JSON，并将字段指定为字节数组而不是字符串表示。处理器将跳过Base64解码。

Demo示例

PUT \_ingest/pipeline/attachment

{

"description" : "Extract attachment information",

"processors" : [

{

"attachment" : {

"field" : "data"

}

}

]

}

PUT my\_index/\_doc/my\_id?pipeline=attachment

{

"data": "e1xydGYxXGFuc2kNCkxvcmVtIGlwc3VtIGRvbG9yIHNpdCBhbWV0DQpccGFyIH0="

}

GET my\_index/\_doc/my\_id

To prevent extracting too many chars and overload the node memory, the number of chars being used for extraction is limited by default to 100000. You can change this value by setting indexed\_chars. Use -1 for no limit but ensure when setting this that your node will have enough HEAP to extract the content of very big documents.

You can also define this limit per document by extracting from a given field the limit to set. If the document has that field, it will overwrite the indexed\_chars setting. To set this field, define the indexed\_chars\_field setting

GET mypdf/\_doc/\_search

{

"query": {

"bool": {

"must": [

{"match\_phrase": { "attachment.content":"违规庭审" }}

]

}

},

"\_source": "attachment",

"highlight": {

"fields": {

"attachment.content": {}

}

}

}

GET mypdf/\_doc/\_search

{

"query": {

"bool": {

"must": [

{"match": { "attachment.content":"select" }}

]

}

},

"\_source": "attachment",

"highlight": {

"fields": {

"attachment.content": {}

}

}

}

GEO（空间查询）

elasticSeach提供了两种方式代表地理位置：geo\_point和geo\_shape

<https://www.elastic.co/guide/en/elasticsearch/guide/current/geoloc.html>

Elasticsearch offers two ways of representing geolocations: latitude-longitude points using the geo\_point field type, and complex shapes defined in [GeoJSON](http://en.wikipedia.org/wiki/GeoJSON), using the geo\_shape field type.

Geo-points allow you to find points within a certain distance of another point, to calculate distances between two points for sorting or relevance scoring, or to aggregate into a grid to display on a map. Geo-shapes, on the other hand, are used purely for filtering. They can be used to decide whether two shapes overlap, or whether one shape completely contains other shapes.

Geo-points cannot be automatically detected with [dynamic mapping](https://www.elastic.co/guide/en/elasticsearch/guide/current/dynamic-mapping.html). Instead, geo\_point fields should be mapped explicitly:

PUT /attractions

{

"mappings": {

"restaurant": {

"properties": {

"name": {

"type": "keyword"

},

"location": {

"type": "geo\_point"

}

}

}

}

}

对于geo\_point，可以有以下三种赋值方式

PUT /attractions/restaurant/1

{

"name": "Chipotle Mexican Grill",

"location": "40.715, -74.011"

}

PUT /attractions/restaurant/2

{

"name": "Pala Pizza",

"location": {

"lat": 40.722,

"lon": -73.989

}

}

PUT /attractions/restaurant/3

{

"name": "Mini Munchies Pizza",

"location": [ -73.983, 40.719 ]

}

与geo\_point相关的过滤

geo\_bounding\_box

Find geo-points that fall within the specified rectangle.

geo\_distance

Find geo-points within the specified distance of a central point.

geo\_distance\_range（在这一版本里没有了）

Find geo-points within a specified minimum and maximum distance from a central point.

geo\_polygon

Find geo-points that fall within the specified polygon. This filter is very expensive. If you find yourself wanting to use it, you should be looking at geo-shapes instead.

GET /attractions/restaurant/\_search

{

"query": {

"bool": {

"must": {

"geo\_bounding\_box": {

"location": {

"top\_left": {

"lat": 40.8,

"lon": -74.0

},

"bottom\_right": {

"lat": 40.7,

"lon": -73.0

}

}

}

}

}

}

}

GET /attractions/restaurant/\_search

{

"query": {

"bool": {

"must": {

"geo\_distance": {

"distance": "1km",

"location": {

"lat": 40.715,

"lon": -73.988

}

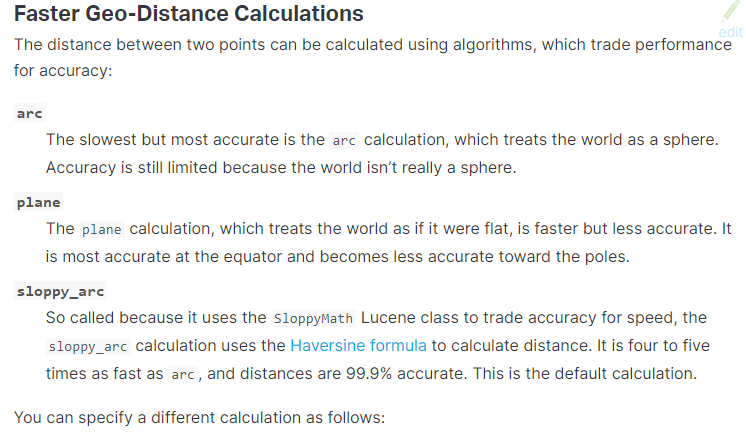
}

}

}

}

}



GET /attractions/restaurant/\_search

{

"query": {

"filtered": {

"filter": {

"geo\_distance": {

"distance": "1km",

"distance\_type": "plane",

"location": {

"lat": 40.715,

"lon": -73.988

}

}

}

}

}

}

Sorting by distance

GET /attractions/restaurant/\_search

{

"query": {

"bool": {

"must": {

"geo\_bounding\_box": {

"type": "indexed",

"location": {

"top\_left": {

"lat": 40.8,

"lon": -74.0

},

"bottom\_right": {

"lat": 40.4,

"lon": -73.0

}

}

}

}

}

},

"sort": [

{

"\_geo\_distance": {

"location": {

"lat": 40.715,

"lon": -73.998

},

"order": "asc",

"unit": "km",

"distance\_type": "plane"

}

}

]

}

Geo Distance Aggregation

GET /attractions/restaurant/\_search

{

"query": {

"bool": {

"must": {

"match": {

"name": "pizza"

}

},

"filter": {

"geo\_bounding\_box": {

"location": {

"top\_left": {

"lat": 40.8,

"lon": -74.1

},

"bottom\_right": {

"lat": 40.4,

"lon": -73.7

}

}

}

}

}

},

"aggs": {

"per\_ring": {

"geo\_distance": {

"field": "location",

"unit": "km",

"origin": {

"lat": 40.712,

"lon": -73.988

},

"ranges": [

{ "from": 0, "to": 1 },

{ "from": 1, "to": 2 }

]

}

}

},

"post\_filter": {

"geo\_distance": {

"distance": "1km",

"location": {

"lat": 40.712,

"lon": -73.988

}

}

}

}

Geo\_shape

PUT /attractions1008

{

"mappings": {

"landmark": {

"properties": {

"name": {

"type": "keyword"100

},

"location": {

"type": "geo\_shape"

}

}

}

}

}

PUT /attractions1008/landmark/dam\_square

{

"name" : "Dam Square, Amsterdam",

"location" : {

"type" : "polygon",

"coordinates" : [[

[ 4.89218, 52.37356 ],

[ 4.89205, 52.37276 ],

[ 4.89301, 52.37274 ],

[ 4.89392, 52.37250 ],

[ 4.89431, 52.37287 ],

[ 4.89331, 52.37346 ],

[ 4.89305, 52.37326 ],

[ 4.89218, 52.37356 ]

]]

}

}

GET /attractions1008/landmark/\_search

{

"query": {

"geo\_shape": {

"location": {

"shape": {

"type": "circle",

"radius": "1km",

"coordinates": [

4.89994,

52.37815

]

}

}

}

}

}

GET /attractions1008/landmark/\_search

{

"query": {

"geo\_shape": {

"location": {

"relation": "within",

"shape": {

"type": "polygon",

"coordinates": [[

[4.88330,52.38617],

[4.87463,52.37254],

[4.87875,52.36369],

[4.88939,52.35850],

[4.89840,52.35755],

[4.91909,52.36217],

[4.92656,52.36594],

[4.93368,52.36615],

[4.93342,52.37275],

[4.92690,52.37632],

[4.88330,52.38617]

]]

}

}

}

}

}

Querying with indexed shapes

With shapes that are often used in queries, it can be more convenient to store them in the index and to refer to them by name in the query.

PUT /attractions/\_mapping/neighborhood

{

"properties": {

"name": {

"type": "string"

},

"location": {

"type": "geo\_shape"

}

}

}

GET /attractions1008/landmark/\_search

{

"query": {

"geo\_shape": {

"location": {

"relation": "within",

"indexed\_shape": {

"index": "attractions1008",

"type": "landmark",

"id": "dam\_square",

"path": "location"

}

}

}

}

}

注意点:（参考：<https://blog.csdn.net/sym542569199/article/details/85011361>）

a.添加数据时，需要指定type-类型和coordinates-坐标

b.一个[lon,lat]表示一个地理点，两个及两个以上地理点表示一条地理线

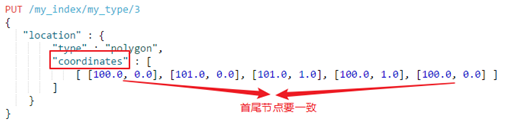
①point，添加地理点，即单个地理坐标，由[经度，纬度]组成。



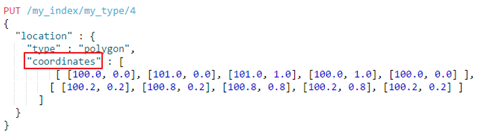
②lineString，添加地理线，由两个或多个地理点定义的线串。若仅指定两个点，线串表示直线。若指定两个以上的点会创建一个任意路径



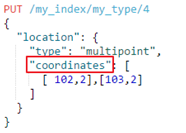
③ polygon，添加闭合多边形，由点列表定义。列表第一个点和最后一个点必须重合，以便可以闭合该多边形



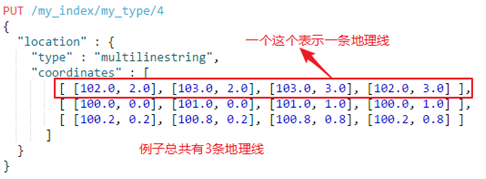
polygon也可以指定内部形状，第一个数组表示多边形的外边界，其他数组表示内部形状（“孔”）：



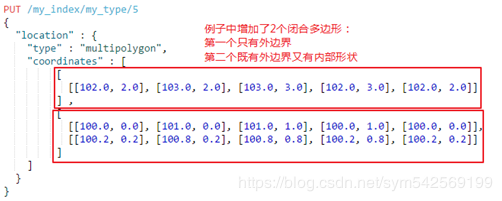
④multiPoint，添加多个地理点



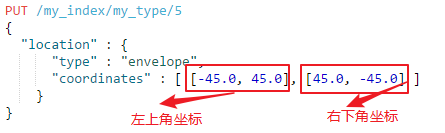
⑤multiLineString，添加多条地理线，一条地理线又是由多个地理点组成的



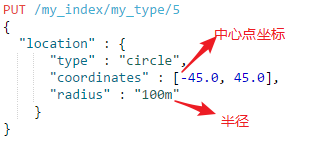
⑥multiPolygon，添加多个闭合多边形，请注意：闭合多边形可以指定外边界和内部形状，所以，与multiLineString相比，会多出一层[]包裹



⑦envelope，指定形状的左上角和右下角的坐标，以表示边界矩形：



⑧circle，添加一个圆，指定半径和中心点坐标。内半径字段是必需的，如果不知道单位，默认是m



# [Elasticsearch 6.4基本操作 - Java版](https://www.cnblogs.com/swordfall/p/9981883.html)

<https://www.cnblogs.com/swordfall/p/9981883.html>

\*\*\*\*\*\*\*

Java代码里引用的jar包的elasticsearch的版本与我服务跑的es的版本不一致，然后就一直报错：NoNodeAvailableException[None of the configured nodes are available: 费了些时间。