**es-c-api接口文档**

目录

[修订记录： 2](#_Toc501008982)

[1、通用接口 2](#_Toc501008983)

[1.1、初始化接口 2](#_Toc501008984)

[1.1.1、int init\_curl() 2](#_Toc501008985)

[1.1.2、CURL \*init\_curl\_handle() 2](#_Toc501008986)

[1.2、销毁接口 3](#_Toc501008987)

[1.2.1、void destrory\_curl() 3](#_Toc501008988)

[1.2.2、void destrory\_curl\_handle(CURL \*curl) 3](#_Toc501008989)

[1.3、设置接口 3](#_Toc501008990)

[1.3.1、int set\_curl\_common\_options(CURL \*curl,int timeout,short ignore\_sig,short verbose) 3](#_Toc501008991)

[2、curd操作 3](#_Toc501008992)

[2.1、新增 3](#_Toc501008993)

[2.1.1、int es\_put(const char \*index,const char \*type,const char \*id,const char \*data) 3](#_Toc501008994)

[2.1.2、inline int es\_post(const char \*index,const char \*type,const char \*data) 4](#_Toc501008995)

[2.1.3、inline json\_object \* es\_post\_bulk(const char \*index,const char \*type,const char \*data) 4](#_Toc501008996)

[2.2、更新 4](#_Toc501008997)

[2.2.1、int es\_update(const char \*index,const char \*type,const char \*id,const char \*data); 4](#_Toc501008998)

[2.2.2、json\_object \* es\_update\_by\_query(const char \*index,const char \*type,const char \*data) 5](#_Toc501008999)

[2.3、查询 5](#_Toc501009000)

[2.3.1、json\_object \*es\_get(const char \*index,const char \*type,const char \*id) 5](#_Toc501009001)

[2.3.2、json\_object \* es\_search(const char \*index,const char \*type,const char \*data) 5](#_Toc501009002)

[2.3.3、json\_object \*es\_scroll(const char \*index,const char \*type,const char \*scroll\_timeout,const char \*data) 6](#_Toc501009003)

[2.3.4、json\_object \* es\_scroll\_next(const char \*scroll\_timeout,const char \*scroll\_id); 6](#_Toc501009004)

[2.3.5、int es\_scroll\_delete(const char \*scroll\_id); 6](#_Toc501009005)

[2.3.6、json\_object \*es\_get\_scroll\_id(json\_object \*root\_obj); 6](#_Toc501009006)

[2.3.7、int64\_t es\_count(const char \*index,const char \*type,const char \*data); 7](#_Toc501009007)

[2.4、删除 7](#_Toc501009008)

[2.4.1、int es\_delete(const char \*index,const char \*type,const char \*id) 7](#_Toc501009009)

[2.4.2、json\_object \* es\_delete\_by\_query(const char \*index,const char \*type,const char \*data) 7](#_Toc501009010)

[2.5、多线程api说明 8](#_Toc501009011)

[3、结果集操作函数 8](#_Toc501009012)

[3.1、json\_object \*json\_string\_to\_json\_object(const char \*str) 8](#_Toc501009013)

[3.2、void free\_json\_object(json\_object \*obj) 8](#_Toc501009014)

[3.3、json\_object \*es\_fetch\_hits(json\_object \*top\_obj) 8](#_Toc501009015)

[3.4、json\_object \*es\_fetch\_hits\_source(json\_object \*top\_obj,int index) 9](#_Toc501009016)

[3.5、int64\_t es\_fetch\_hits\_len(json\_object \*top\_obj) 9](#_Toc501009017)

[3.6、int64\_t es\_fetch\_hits\_total(json\_object \*top\_obj) 9](#_Toc501009018)

[3.7、json\_object \*es\_fetch\_node(json\_object \*object,char \*node) 9](#_Toc501009019)

[3.8、int es\_set\_node(json\_object \*obj,char \*node,json\_type node\_type,void \*node\_value) 10](#_Toc501009020)

[4、示例 11](#_Toc501009021)

[4.1、结果集操作 11](#_Toc501009022)

[4.2、单线程环境中使用api 13](#_Toc501009023)

[4.3、多线程环境中使用api 13](#_Toc501009024)

[4.4、es query data示例 14](#_Toc501009025)

# 修订记录：

1、初版

2、增加3.8章节

# 1、通用接口

使用方法请参考4.2、4.3章节。

## 1.1、初始化接口

### 1.1.1、int init\_curl()

接口说明：初始化curl全局变量，es-c-api利用curl调用es的rest api，该函数在主线程中调用一次即可。

输入参数：无。

返回值：

0：成功；

其他值：失败。

### 1.1.2、CURL \*init\_curl\_handle()

接口说明：初始化curl句柄，该句柄用来处理curl请求。

输入参数：无。

返回值：

not NULL：成功；

失败：NULL。

## 1.2、销毁接口

### 1.2.1、void destrory\_curl()

接口说明：销毁curl占用的资源，在不需要用api时调用该函数一次即可。

输入参数：无。

返回值：无。

### 1.2.2、void destrory\_curl\_handle(CURL \*curl)

接口说明：释放curl句柄。

输入参数：指向curl句柄的指针。

返回值：无。

## 1.3、设置接口

### 1.3.1、int set\_curl\_common\_options(CURL \*curl,int timeout,short ignore\_sig,short verbose)

接口说明：设置curl句柄的通用选项。

输入参数：

curl：指向curl句柄的指针；

timeout：curl连接超时时间，单位是秒；

ignore\_sig：对于超时，是否产生信号，false：不产生，true：产生信号；

verbose：是否打印详细信息，false：不打印，true：打印。

返回值：

0：成功；

其他值：失败。

# 2、curd操作

## 2.1、新增

### 2.1.1、int es\_put(const char \*index,const char \*type,const char \*id,const char \*data)

接口说明：插入一个指定文档id的文档。

输入参数：

index：index名称；

type：type名称；

id：文档id；

data：文档内容。

返回值：

0：成功；

>0：失败；

<0：出错。

### 2.1.2、inline int es\_post(const char \*index,const char \*type,const char \*data)

接口说明：插入一个由es自动生成文档id的文档。

输入参数：

index：index名称；

type：type名称；

data：文档内容。

返回值：

0：成功；

>0：失败；

<0：出错。

### 2.1.3、inline json\_object \* es\_post\_bulk(const char \*index,const char \*type,const char \*data)

接口说明：批量插入一组由es自动生成文档id的文档，若失败，需调用free\_json\_object释放返回的资源。

输入参数：

index：index名称；

type：type名称；

data：文档内容，每个文档由换行符隔开，若data本身包含换行符，请注意处理。

返回值：

NULL：成功；

not NULL：失败,指向失败的文档在data中的索引，是一个数组。

## 2.2、更新

### 2.2.1、int es\_update(const char \*index,const char \*type,const char \*id,const char \*data);

接口说明：更新指定文档。

输入参数：

index：index名称；

type：type名称；

id：文档id；

data：更新内容。

返回值：

0：成功；

>0：失败；

<0：出错。

### 2.2.2、json\_object \* es\_update\_by\_query(const char \*index,const char \*type,const char \*data)

接口说明：按指定条件更新文档，若失败，需调用free\_json\_object释放返回的资源。

输入参数：

index：index名称；更新指定index下的文档，不能为NULL；

type：type名称；为NULL表示在index下的所有type中更新；

data：更新条件；为NULL表示对所有文档进行更新。

返回值：

NULL：成功；

not NULL：失败,指向failures对象的指针。

## 2.3、查询

### 2.3.1、json\_object \*es\_get(const char \*index,const char \*type,const char \*id)

接口说明：获取指定文档id的文档，操作完成后调用free\_json\_object释放返回的对象。

输入参数：

index：index名称；查询指定index下的文档，不能为NULL；

type：type名称；为NULL表示在index下的所有type中查询；

id：文档id。

返回值：

not NULL：成功；

NULL：失败。

### 2.3.2、json\_object \* es\_search(const char \*index,const char \*type,const char \*data)

接口说明：search方法、查找符合条件的文档，类似于mysql中的select，操作完成后调用free\_json\_object释放返回的对象。

输入参数：

index：index名称；为NULL表示查询所有index下的文档；

type：type名称；为NULL表示查询index下所有type下的文档；

data：查询的条件，为NULL表示查询所有文档。

返回值：

not NULL：成功；

NULL：失败。

### 2.3.3、json\_object \*es\_scroll(const char \*index,const char \*type,const char \*scroll\_timeout,const char \*data)

接口说明：scroll api，获取所有结果集，每次返回的结果集大小由size控制，可根据结果集中返回的scroll\_id获取后续的数据，见es\_scroll\_next函数，操作完成后调用free\_json\_object释放返回的对象。

输入参数：

index：index名称；

type：type名称；

scroll\_timeout： 结果集保存时间，如1m：表示一分钟；

data：请求数据。

返回值：

not NULL：成功，指向结果集的指针；

NULL：失败。

### 2.3.4、json\_object \* es\_scroll\_next(const char \*scroll\_timeout,const char \*scroll\_id);

接口说明：scroll api，根据scroll\_id获取该结果集的剩余数据，第一个scroll\_id是es\_scroll函数返回的，后续要继续调用请用该函数返回的最新的scroll\_id，虽然大多数情况下该函数返回的scroll\_id与第一次调用es\_scroll函数返回的scroll\_id是一致的，操作完成后调用free\_json\_object释放返回的对象。

输入参数：

scroll\_timeout： 结果集保存时间，如1m：表示一分钟；

scroll\_id：最新返回的scroll\_id。

返回值：

not NULL：成功，指向结果集的指针；

NULL：失败。

### 2.3.5、int es\_scroll\_delete(const char \*scroll\_id);

接口说明：删除有scroll\_id保存的数据，虽然在调用scroll api时指定了保存时间，超时仍然会删除，但我们仍可以手动删除，节约内存空间。

输入参数：

scroll\_id：最新返回的scroll\_id。

返回值：

0：成功；

1：失败。

### 2.3.6、json\_object \*es\_get\_scroll\_id(json\_object \*root\_obj);

接口说明：根据es\_scroll或者es\_scroll\_next函数返回的结果集，获取该结果集的scroll\_id。

输入参数：

root\_obj：指向es\_scroll或者es\_scroll\_next函数返回的结果集的指针。

返回值：

not NULL：指向scroll\_id的指针；

NULL：失败。

### 2.3.7、int64\_t es\_count(const char \*index,const char \*type,const char \*data);

接口说明：count api，获取满足请求条件data的文档个数。

输入参数：

index：index名称；

type：type名称；

data：请求数据。

返回值：满足条件的文档的个数。

## 2.4、删除

### 2.4.1、int es\_delete(const char \*index,const char \*type,const char \*id)

接口说明：按指定文档id删除文档。

输入参数：

index：index名称；删除指定index下的文档，不能为NULL；

type：type名称，type为NULL,则id也必须为NULL，表示删除整个index；

id：文档名称，id为NULL，则type也必须为空，表示删除整个index。

返回值：

0：成功；

>0：失败；

<0：出错。

### 2.4.2、json\_object \* es\_delete\_by\_query(const char \*index,const char \*type,const char \*data)

接口说明：按指定条件删除文档，若失败，需调用free\_json\_object释放返回的资源。

输入参数：

index：index名称；删除指定index下的文档，不能为NULL；

type：type名称；为NULL表示在index下的所有type中进行删除；

data：删除条件；为NULL表示删除所有文档。

返回值：

NULL：成功；

not NULL：失败,指向failures对象的指针。

## 2.5、多线程api说明

多线程环境中用的api，与没有mul的api功能相同，参数多了一个curl的句柄。

json\_object \* es\_mul\_search(CURL \*curl,const char \*index,const char \*type,const char \*data);

json\_object \*es\_mul\_get(CURL \*curl,const char \*index,const char \*type,const char \*id);

int64\_t es\_mul\_delete\_by\_query(CURL \*curl,const char \*index,const char \*type,const char \*data,char \*options);

int es\_mul\_delete(CURL \*curl,const char \*index,const char \*type,const char \*id);

int64\_t es\_mul\_update\_by\_query(CURL \*curl,const char \*index,const char \*type,const char \*data,char \*options);

int es\_mul\_update(CURL \*curl,const char \*index,const char \*type,const char \*id,const char \*data);

inline int es\_mul\_post\_bulk(CURL \*curl,const char \*index,const char \*type,const char \*data);

inline int es\_mul\_post(CURL \*curl,const char \*index,const char \*type,const char \*data);

int es\_mul\_put(CURL \*curl,const char \*index,const char \*type,const char \*id,const char \*data);

json\_object \* es\_mul\_scroll(CURL \*curl,const char \*index,const char \*type,const char \*scroll\_timeout,const char \*data);

json\_object \* es\_mul\_scroll\_next(CURL \*curl,const char \*scroll\_timeout,const char \*scroll\_id);

int es\_mul\_scroll\_delete(CURL \*curl,const char \*scroll\_id);

int64\_t es\_mul\_count(CURL \*curl,const char \*index,const char \*type,const char \*data);

## 3、结果集操作函数

## 3.1、json\_object \*json\_string\_to\_json\_object(const char \*str)

接口说明：将json格式的字符串转换成json的对象，调用该函数后需要调用free\_json\_object释放内存。

输入参数：

str：json格式的字符串。

返回值：

not NULL：json对象；

NULL：失败。

## 3.2、void free\_json\_object(json\_object \*obj)

接口说明：释放由es\_get，es\_search、es\_scroll、es\_scroll\_next函数调用产生的json对象。

输入参数：

obj：需要释放的json对象。

返回值：无。

## 3.3、json\_object \*es\_fetch\_hits(json\_object \*top\_obj)

接口说明：获取输入参数top\_obj的hits.hits对象，即匹配条件的文档数组。

输入参数：

top\_obj：由其他es c-api调用产生的结果。

返回值：

not NULL：指向hits.hits对象的指针；

NULL：失败。

## 3.4、json\_object \*es\_fetch\_hits\_source(json\_object \*top\_obj,int index)

接口说明：获取输入参数top\_obj的hits.hits[index].\_source对象，即匹配条件的文档数组中的某个元素的数据。

输入参数：

top\_obj：由其他es c-api调用产生的结果；

index：数组元素索引。

返回值：

not NULL：指向该数组元素的\_source对象的指针；

NULL：失败。

## 3.5、int64\_t es\_fetch\_hits\_len(json\_object \*top\_obj)

接口说明：获取输入参数top\_obj的hits.hits数组对象的长度，即返回的结果集中匹配条件的文档个数。

输入参数：

top\_obj：由其他es c-api调用产生的结果。

返回值：数组长度。

## 3.6、int64\_t es\_fetch\_hits\_total(json\_object \*top\_obj)

接口说明：获取输入参数top\_obj的hits.total的值，即满足匹配条件的总文档个数。

输入参数：

top\_obj：由其他es c-api调用产生的结果。

返回值：匹配的总个数。

## 3.7、json\_object \*es\_fetch\_node(json\_object \*object,char \*node)

接口说明：获取输入参数top\_obj中，指定节点的对象。

输入参数：

top\_obj：json对象；

node：需要获取的节点对象名称。

返回值：

not NULL:执行该节点对象的指针；

NULL：失败。

## 3.8、int es\_set\_node(json\_object \*obj,char \*node,json\_type node\_type,void \*node\_value)

接口说明：封装json\_object操作，按照node构造json对象。

输入参数：

obj：json对象的指针，存储构造的json对象；

node：需要设置的节点的名称；

node\_type：需要设置的节点的类型；

可以是：json\_type\_object,json\_type\_array,json\_type\_string,json\_type\_int,json\_type\_boolean

node\_value：节点的值。

返回值：

0:成功，结果保存在obj中；

> 0：失败。

比如要构造这样一个json object：

{

"query":{

"bool":{

"must":[

{

"range":{

"start\_ip":{

"lte":"192.168.3.111"

}

}

},

{

"range":{

"end\_ip":{

"gte":"192.168.3.111"

}

}

}

]

}

}

}

步骤：

创建存储数据的对象：

json\_object \*obj = json\_object\_new\_object();

按照node构造对象

es\_set\_node(obj,"query.bool.must[0].range.start\_ip.lte",json\_type\_string,"192.168.3.111");

es\_set\_node(obj,"query.bool.must[1].range.start\_ip.gte",json\_type\_string,"192.168.3.111");

或者：

es\_set\_node(obj,"query.bool.must[0].range ",json\_type\_object,"{\"start\_ip\":{\"lte\":\"192.168.3.111\"}}");

es\_set\_node(obj,"query.bool.must[1].range ",json\_type\_object,"{\"end\_ip\":{\"gte\":\"192.168.3.111\"}}");

或者：

es\_set\_node(obj,"query.bool.must ",json\_type\_array,"[{},{}]");

最后转为字符串:

char \*obj\_str = json\_object\_to\_json\_string(obj);

释放对象

free\_json\_object(obj);

注意：对于node\_type为 json\_type\_array来说，不支持对array元素不是object的类型按照索引的方式构造对象，比如一个数组为char \*p = ["ab","cd","ef","g"];

则只能一次性赋值：es\_set\_node(obj,"query.bool.must[0]",json\_type\_array,p);

不能使用

es\_set\_node(obj,"query.bool.must[0]",json\_type\_array,"ab");

es\_set\_node(obj,"query.bool.must[1]",json\_type\_array,"cd");

...

这样的方式，因为这里涉及到两个类型，一个是node的类型，一个是node\_value的类型。

# 4、示例

## 4.1、结果集操作

假如有如下的一个结果集：

{

"took" : 9,

"timed\_out" : false,

"\_shards" : {

"total" : 55,

"successful" : 55,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 1320557,

"max\_score" : 1.0,

"hits" : [

{

"\_index" : "events",

"\_type" : "test",

"\_id" : "\_update",

"\_score" : 1.0,

"\_source" : {

"timestamp" : 100000,

"property\_ip" : "111",

"property\_port" : 1

}

},

{

"\_index" : "events",

"\_type" : "test",

"\_id" : "AWALzTpDJFTTvo7zfhBC",

"\_score" : 1.0,

"\_source" : {

"timestamp" : 100000,

"pkt\_data" : [

{

"timestamp" : 100000,

"pkt" : "wwwwwwwwwwwwwwwwwwwwwwwwww"

}

]

}

}

]

}

}

1、获取任一节点的对象,

比如获取max\_score, es\_fetch\_node(top\_obj,"hits.max\_score");

获取hits第一个元素的timestamp，es\_fetch\_node(top\_obj,"hits.hits[0].timestamp");

返回的都是json对象，要获取该对象的值请调用json的函数：

json\_object\_get\_int(obj);

json\_object\_get\_int64(obj);

json\_object\_get\_string(obj);

json\_object\_get\_boolean(obj) 等等；

2、获取total：es\_fetch\_total()；

3、获取hits数组的长度： es\_fetch\_hits\_len()；

4、获取hits数组：es\_fetch\_hits()；

5、获取hits数组的某一个元素的\_source字段： es\_fetch\_hits\_source()；

6、遍历hits数组：

1）首先获取数组长度：len = es\_fetch\_hits\_len();

2）遍历\_source

while(i<len){

es\_fetch\_hits\_source(obj,i);

}

## 4.2、单线程环境中使用api

int main()

{

init\_curl();

{

do something

eg:es\_post,es\_search

}

destroy\_curl();

}

## 4.3、多线程环境中使用api

int main()

{

init\_curl();

{

thread-A

CURL \*curlA = init\_curl\_handle();

set\_curl\_common\_options(curlA,...);

do something

eg:es\_mul\_post(curlA,...),es\_mul\_search(curlA,...)

destroy\_curl\_handle(curlA);

}

{

thread-B

CURL \*curlB = init\_curl\_handle();

set\_curl\_common\_options(curlB,...);

do something

eg:es\_mul\_post(curlB,...),es\_mul\_search(curlB,...)

destroy\_curl\_handle(curlB);

}

destroy\_curl();

}

## 4.4、es query data示例

1）select property\_ip,event\_type from test where status = false and timestamp > 100000 limit 20;

select property\_ip,event\_type from test where status = false and timestamp > 100000 limit 10,20

from + size 默认不能超过10000，可通过index的max\_result\_window修改该值。

调用es\_search

curl -XPOST localhost:9200/events/test/\_search?pretty -d '{

"from":10,

"size":10,

"\_source":["property\_ip","event\_type"],

"query":{

"bool":{

"filter":[

{

"term":{

"status":false

}

},

{

"range":{

"timestamp":{

"gt":100000

}

}

}

]

}

}

}'

2）select count(\*) from test where timestamp >= 100000;

调用es\_count

curl -XPOST localhost:9200/events/test/\_count?pretty -d '{

"query":{

"range":{

"timestamp":{

"gte":100000

}

}

}

}'

3）select max(timestamp),count(\*) from test where timestamp > 100000 and timestamp < 110000;

调用es\_search

curl -XPOST localhost:9200/events/test/\_search?pretty -d '{

"size":0,

"aggs":{

"range\_t":{

"range":{

"field":"timestamp",

"ranges":{

"from":100000,

"to":110000

}

},

"aggs":{

"max\_t":{

"max":{

"field":"timestamp"

}

}

}

}

}

}'

4）select count(distinct event\_type),min(timestamp) as min\_time,property\_ip,count(\*) from test group by property\_ip order by min\_time limit 5;

调用es\_search

curl -XGET localhost:9200/suspicious2017/m1/\_search?pretty -d '{

"size":0,

"aggs":{

"pips":{

"terms":{

"field":"property\_ip",

"size":5,

"order":{

"min\_time ":"desc"

}

},

"aggs":{

"types":{

"cardinality":{

"field":"event\_type"

}

},

"min\_time":{

"min":{

"field":"timestamp"

}

}

}

}

}

}'

5）insert into test(id,ip,event\_type,signature,status,handled,timestamp) values(.....);

调用es\_put

curl -XPUT 'localhost:9200/sur01ev2017/m1/doc\_id' -d '{

"ip":"158.98.123.22",

"event\_type":"疑似攻击",

"signature":"数据包持续匹配规则",

"status":0，

"handled":0,

"timestamp":"2017-11-11 11:11:11"

}'

6）insert into test(ip,event\_type,signature,status,handled,timestamp) values(.....);

调用es\_post

curl -XPOST 'localhost:9200/sur01ev2017/m1/' -d '{

"ip":"158.98.123.22",

"event\_type":"疑似攻击",

"signature":"数据包持续匹配规则",

"status":0，

"handled":0,

"timestamp":"2017-11-11 11:11:11"

}'

7）批量插入insert into test(ip,event\_type,signature,status,handled,timestamp) values(.....),(),(),()....;

调用es\_post\_bulk

curl -XPOST 'localhost:9200/sur01ev2017/m1/' -d '

{"index":{}}\n

{

"ip":"158.98.123.22",

"event\_type":"疑似攻击",

"signature":"数据包持续匹配规则",

"status":0，

"handled":0,

"timestamp":"2017-11-11 11:11:11"

}\n

{"index":{}}\n

{

"ip":"158.98.123.22",

"event\_type":"疑似攻击",

"signature":"数据包持续匹配规则",

"status":0，

"handled":0,

"timestamp":"2017-11-11 11:11:11"

}\n

...

'

批量操作还有更复杂的用法，请参考官方文档。

8）update test set status = 1;

调用es\_update

curl -XPOST 'localhost:9200/sur01ev2017/m1/\_update

{

"script" : {

"source": "ctx.\_source.status =1"

}

}

9）update test set status = 1 where property\_ip = xxx;

调用es\_update\_by\_query

curl -XPOST 'localhost:9200/sur01ev2017/m1/\_update\_by\_query

{

"script" : {

"source": "ctx.\_source.status =1"

},

"query":{

"term":{

"property\_ip":"xxx"

}

}

}