## 数据同步

数据库数据同步到es，主要考虑是内容是采用全量还是增量同步的方式，就目前而言，对于删改内容的同步，还不是很能满足要求，所以主要内容还是全量和增量同步。

目前数据同步实现的方式是采用python脚本来完成。其基本流程如下：



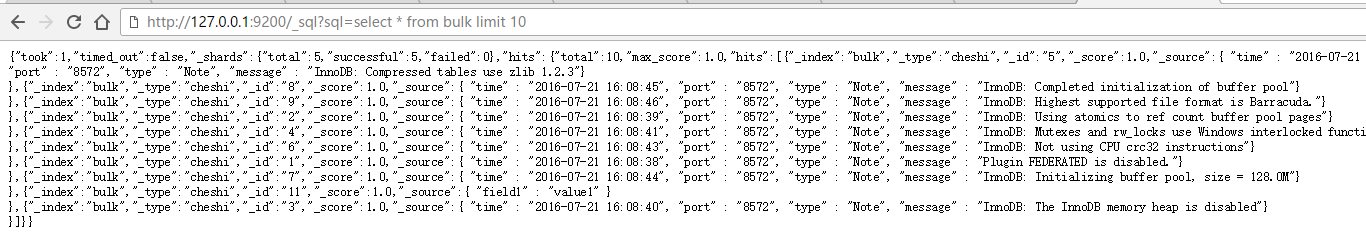
## 查询转换

通常，对于数据的复杂查询，我们通过es的dsl语句来完成查询工作，但是es的查询相对而言学习成本较高，所以这里使用elasticsearch-sql插件提供直接使用sql语句进行查询数据的功能，并可将sql语句转换成es的DSL查询语句。可以通过页面来进行操作：访问<http://127.0.0.1:9200/_plugin/sql/>（以本地集群为例）进行相关的操作

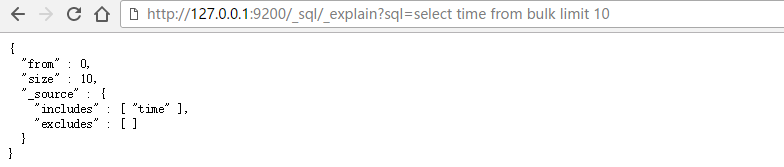


同时，也提供相应的sql API来进行直接查询或转换

查询（http://127.0.0.1:9200/\_sql?sql=select \* from bulk limit 10）：



转换（http://127.0.0.1:9200/\_sql/\_explain?sql=select time from bulk limit 10）



## 接口调用

Es提供的API较为丰富，主要分为包括Document APIS、Search APIS、索引 API、cat APIS以及集群API。

### Document APIS

包括分为单文档API和多文档API，单文档API包括Index API、Get API、Delete API以及Update API；多文档API包括Multi Get API、Bulk API、Delete by Query API、Update By Query API以及Reindex API。

#### Index API

索引API在特定索引中增加或更新一个json格式的文档，使其可搜索，使用PUT方法进行操作。

#### Get API

Get api允许从一个基于其id的index中获取一个json格式的文档。

#### Delete API

允许基于指定的ID来从索引库中删除一个文档。

#### Update API

通过put或post方法来更新文档

#### Multi Get API

localhost:9200/\_mget，通过这种方式来获取集群中的所有数据；

localhost:9200/index1/\_mget,通过这种方式来获取集群中index1索引的数据；

localhost:9200/index1/type1/\_mget，通过这种方式来获取index1索引type1类型中的所有数据；

更细节的内容见参考

#### Bulk API

Bulk API，能够在一个单一的API调用执行多项索引/删除操作，可以大大提高索引速度。

POST localhost:9200/\_bulk

{ "index" : { "\_index" : "test", "\_type" : "type1", "\_id" : "1" } }

{ "field1" : "value1" }

{ "delete" : { "\_index" : "test", "\_type" : "type1", "\_id" : "2" } }

{ "create" : { "\_index" : "test", "\_type" : "type1", "\_id" : "3" } }

{ "field1" : "value3" }

{ "update" : {"\_id" : "1", "\_type" : "type1", "\_index" : "index1"} }

{ "doc" : {"field2" : "value2"} }

我们目前使用的数据导入也是这个方法

#### DELETE By Query API

使用\_delete\_by\_query对每个查询匹配的文档执行删除。

POST twitter/\_delete\_by\_query

{

"query": {

"match": {

"message": "some message"

}

}

}

#### Update By Query API

一般需要在查询的时候通过脚本内容来进行更新内容操作：

POST twitter/\_update\_by\_query

{

"script": {

"inline": "ctx.\_source.likes++",

"lang": "painless"

},

"query": {

"term": {

"user": "kimchy"

}

}

}

向所有的user为kimchy中增加likes

#### Reindex API

\_reindex的最基本形式只是将文档从一个索引复制到另一个索引。

POST \_reindex

{

"source": {

"index": "twitter"

},

"dest": {

"index": "new\_twitter"

}

}

### Search APIS

这里主要关注es的DSL查询语句中的内容

#### Request Body Search

##### Query

该元素允许查询DSL定义一个查询。

POST /\_search

{

"query" : {

"term" : { "user" : "kimchy" }

}

}

##### From/Size

查询结果的分页可以通过from和size参数来完成。From参数定义了第一个结果的偏移量，size参数设置允许返回的最大匹配数。查询默认设置是from为0 ，size为10

POST /\_search

{

"from" : 0, "size" : 10,

"query" : {

"term" : { "user" : "kimchy" }

}

}

##### Sort

允许在特定字段上添加一个或多个排序。

对多值字段进行排序的时候，可以将多值字段减为单值，使用min、max、avg或者是sum

"sort": {

"dates": {

"order": "asc",

"mode": "min"

}

}

##### Source filtering

控制\_source字段在返回结果时返回哪些内容

一种是在查询的时候禁用\_source：

POST /\_search

{

"\_source": false,

"query" : {

"term" : { "user" : "kimchy" }

}

}

一种是通过通配符来控制\_source返回哪些部分:

POST /\_search

{

"\_source": "obj.\*",

"query" : {

"term" : { "user" : "kimchy" }

}

}

或者:

POST /\_search

{

    "\_source": [ "obj1.\*", "obj2.\*" ],

    "query" : {

        "term" : { "user" : "kimchy" }

    }

}

同时，可以指定包含和排除模式：

GET /\_search

{

"\_source": {

"includes": [ "obj1.\*", "obj2.\*" ],

"excludes": [ "\*.description" ]

},

"query" : {

"term" : { "user" : "kimchy" }

}

}

##### Fields

Stored\_fields参数表示字段内容单独存储，通常情况下关闭，默认返回单独字段内容而不是source指定的话，需要通过过滤来完成：

GET /\_search

{

"stored\_fields" : ["user", "postDate"],

"query" : {

"term" : { "user" : "kimchy" }

}

} 同时可以上述指定内容设置为[]以及”\_none\_”来实现禁用存储字段

##### Script Fields

允许为每次匹配返回脚本评估（基于不同的字段），这部分的内容看需求了。

GET /\_search

{

"query" : {

"match\_all": {}

},

"script\_fields" : {

"test1" : {

"script" : {

"lang": "painless",

"inline": "doc['my\_field\_name'].value \* 2"

}

},

"test2" : {

"script" : {

"lang": "painless",

"inline": "doc['my\_field\_name'].value \* factor",

"params" : {

"factor" : 2.0

}

}

}

}

}

##### Highlighting

允许突出显示一个或多个字段的搜索结果。

GET /\_search

{

"query" : {

"match": { "user": "kimchy" }

},

"highlight" : {

"fields" : {

"content" : {}

}

}

}

这里将为每个搜索名字突出显示。

##### Version

返回查询结果的版本

GET /\_search

{

"version": true,

"query" : {

"term" : { "user" : "kimchy" }

}

}

### Aggregations

聚合有两个主要概念：

桶（Buckets）：满足特定条件的文档的集合

指标（Metrics）：对桶内的文档进行统计计算

每个聚合都是一个或者多个桶和零个或者多个指标的组合，大致用SQL来解释的话如下：

SELECT COUNT(color) https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png

FROM table

GROUP BY color https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png

|  |  |
| --- | --- |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/1.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/aggs-high-level.html#CO183-1) | COUNT(color) 相当于指标。 |
| [https://www.elastic.co/guide/cn/elasticsearch/guide/current/images/icons/callouts/2.png](https://www.elastic.co/guide/cn/elasticsearch/guide/current/aggs-high-level.html#CO183-2) | GROUP BY color 相当于桶。 |

桶在概念上类似于SQL的分组（GROUP BY），而指标则类似于COUNT()、SUM()、MAX()等统计方法。

示例：

GET /cars/transactions/\_search

{

"size" : 0,

"aggs" : { 1

"popular\_colors" : { 2

"terms" : { 3

"field" : "color"

}

}

}

}

1. 聚合操作被置于顶层参数aggs之下，当然也可以用完整形式aggregations
2. 为聚合指定一个我们想要的名称
3. 定义单个桶的类型terms

这里只做简单介绍，具体内容参考：

<https://www.elastic.co/guide/cn/elasticsearch/guide/current/aggregations.html>

### 索引API

这里的内容，创建、删除以及获取索引的内容就略过了。

#### 启动关闭索引

这里运行关闭一个索引并在之后重新开启，一个关闭的索引几乎不占用集群资源（除了维持本身的元信息），一个关闭的索引可以通过常规的恢复进程开启。

#### Mapping

路径中只指明了index但是没有指明type的时候，要在查询语句中指定，下例就是指定了tweet类型的mapping

{

  "mappings": {

    "tweet": {

      "properties": {

        "message": {

          "type": "text"

        }

      }

    }

  }

}

通常设置mapping的时候，在路径指定类型之后，查询dsl如下：

{

  "properties": {

    "name": {

      "type": "text"

    }

  }

}

可以直接设置字段的mapping内容。一般mapping设置的内容包括字段类型、是否索引、索引是否分词以及字段是否存储等内容，\_source字段是否禁用也在此设置。

Cat APIS以及集群API内容就不做介绍了，相关的内容参考：

<http://www.apache.wiki/display/Elasticsearch/cat+APIs>

<http://www.apache.wiki/pages/viewpage.action?pageId=4882951>

### java API

提供官方文档内容：

Es 5.4

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

es 2.3

<https://www.elastic.co/guide/en/elasticsearch/client/java-api/2.3/index.html>

中文的相关教程较少，都是个人实践总结，有个2.3.4的示例：

<https://my.oschina.net/u/1027043/blog/731356>

以及2.2的示例（内容偏基础操作）：

<https://my.oschina.net/wangnian/blog/684686>

（搜索操作）

<https://my.oschina.net/secisland/blog/617026>

## 业务逻辑

就集成es而言，目的是想其来代替数据库的作用，但是却有着很多缺陷，考虑用其来代替数据库，考虑的主要是原有数据库数据的同步的问题，以及其和数据库的对比。

在优势上，es对比数据的优势在于：

1. 数据量的支持上优于mysql
2. 字符查询效率优于mysql
3. 聚合查询效率优于mysql

以监测平台为例，跑相关的业务逻辑，目前而言还没有完成，主要原因在于替换较复杂是join查询的内容暂时没有解决，考虑用es来代替监测平台中的mysql主要考虑的问题有以下几点：

1. 事务性。这一点好像在监测平台中体现不是很明显，所以也没有关注，但是es本身是不支持事务的。
2. 权限控制。es的权限控制模块被做成了收费插件，网上开源的权限控制在字段级的控制上也要收费，所以，权限控制的问题，如果想要直接在es自身上做，收费可能是绕不开的问题。
3. 关联查询。es来做关联查询的时候，通过字段冗余来实现关联查询，简单的join的确可以实现，复杂的join还有待验证，但是不被看好。
4. 聚合的精确度问题。Es的聚合一般返回的结果都是一个近似值，一般不指定size的话，聚合的内容只是对某个分片做的聚合，返回的结果并不精确，解决方法目前来看就是把要聚合的数据放到一个分片上，或者指定返回的size大于等于结果的size。

**针对上述问题，需要对项目各方面的需求进行梳理，才能确定是否可以用es来代替mysql，如果上述问题在系统中体现不是很明显，那么可以考虑使用es来替换mysql。**

## 参考

<https://github.com/NLPchina/elasticsearch-sql/tree/elastic2.3.5> （es-sql）

<http://www.apache.wiki/pages/viewpage.action?pageId=4260364> （es5.4中文文档）

<https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html> （es2.x中文文档）