1. 元数据

**GET goods/fruit/\_search**

查询结果如下:

{

"**\_index**" : "goods",---------------------这里就是\_index元数据

"**\_type**" : "fruit",----------------这里是\_type元数据

"**\_id**" : "2",---------------------这里是\_id元数据

"\_score" : 1.0,

"**\_source**" : {----------------------这里是\_source元数据

"name" : "pingguo",

"describe" : "cui",

"price" : 60,

"producer" : "zhongguo",

"tags" : [

"haokan",

"xiang"

]

}

}

* 1. \_index元数据:文档的索引

（1）代表一个document存放在哪个index中

（2）类似的数据放在一个索引，非类似的数据放不同索引：product index（包含了所有的商品），sales index（包含了所有的商品销售数据），inventory index（包含了所有库存相关的数据）。如果你把比如product，sales，human resource（employee），全都放在一个大的index里面，比如说company index，不合适的。

（3）index中包含了很多类似的document：类似是什么意思，其实指的就是说，这些document的fields很大一部分是相同的，你说你放了3个document，每个document的fields都完全不一样，这就不是类似了，就不太适合放到一个index里面去了。

（4）索引名称必须是小写的，不能用下划线开头，不能包含逗号：product，website，blog

* 1. \_type元数据:文档的类型

（1）代表document属于index中的哪个类别（type）

（2）一个索引通常会划分为多个type，逻辑上对index中有些许不同的几类数据进行分类：因为一批相同的数据，可能有很多相同的fields，但是还是可能会有一些轻微的不同，可能会有少数fields是不一样的，举个例子，就比如说，商品，可能划分为电子商品，生鲜商品，日化商品，等等。

（3）type名称可以是大写或者小写，但是同时不能用下划线开头，不能包含逗号

* 1. \_id元数据:文档的标记

（1）代表document的唯一标识，与index和type一起，可以唯一标识和定位一个document

（2）我们可以手动指定document的id（put /index/type/id），也可以不指定，由es自动为我们创建一个id

1.4 自动生成和手动生成id

自动生成id形式

**POST goods/fruit**

**{**

**"name":"pingguo",**

**"describe":"haochi",**

**"price":87,**

**"producer":"zhongguo",**

**"tags":["da","haochi"]**

**}**

手动生成id形式

**put /goods/fruit/6**

**{**

**"name":"xigua",**

**"describe":"haochi",**

**"price":109,**

**"producer":"zhongguo",**

**"tags":["da","haochi"]**

**}**

**注意:post和put的区别,两者都可以进行数据的插入,put命令需要指定id,post命令不需要指定id,其他的基本用法都差不多.put命令执行多次,效果一样.不会额外产生多条数据,post命令执行多次会插入多条数据,另外post插入数据以后生成的id为随机id,是一个GUID 20位**

* 1. \_source元数据:检索内容,格式为json格式
  2. \_version元数据:当前数据的版本号

重复执行代码:

**put /goods/fruit/7**

**{**

**"name":"xigua",**

**"describe":"pianyi",**

**"price":100,**

**"producer":"zhongguo",**

**"tags":["da","haochi"]**

**}**

执行结果如下:

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "7",

"\_version" : 2,----------------------这里就是versoin对应的数为当前版本号

"result" : "updated",

"\_shards" : {

"total" : 2,

"successful" : 1,

"failed" : 0

},

"\_seq\_no" : 16,

"\_primary\_term" : 3

}

**注意:当前version为自动维护,自动新增版本处理.如果发生并发事件时,可以通过对比\_version来进行处理,俗称乐观锁**

**重点:乐观锁的操作**

**GET goods/fruit/7**

通过查询的方式来获得当前这条数据的version号,然后进行修改操作的时候,需要提供当前的version版本号

**注意:如果版本号不一致的话,则程序报错.如果频繁操作当前的版本号的数据的话,一定要注意版本控制**

**put /goods/fruit/7?version=4**

**{**

**"name":"xigua",**

**"describe":"pianyi",**

**"price":190,**

**"producer":"zhongguo",**

**"tags":["da","haochi"]**

**}**

* 1. 特殊方式的乐观锁(版本控制)

通过version\_type=external来完成乐观锁控制

模拟两个用户进行数据的获取,然后完成数据提交,这里需要使用version来确定

仍然使用**GET goods/fruit/7**来获取数据,再通过

**put /goods/fruit/10?version=1&version\_type=external**

**{**

**"name":"xigua",**

**"describe":"pianyi",**

**"price":190,**

**"producer":"zhongguo",**

**"tags":["da","haochi"]**

**}**

来改变当前数据的版本号,只要version的值大于当前version的值就可以了

**注意:在使用version\_type=external的时候一定要注意,不要把当前的版本号改的太大,因为第一个人改了以后,其他人再修改到时候会尴尬.推荐使用普通的version版本号于当前版本一致的方式来进行并发处理,符合数据库的处理方式.**

**另外更新以后当前的版本号就跟最后一次修改过的值一致了**

1. 批量处理
   1. \_mget

执行操作

**GET /\_mget**

**{**

**"docs":[**

**{**

**"\_index":"goods",---------------这里指定当前的index索引类型**

**"\_type":"fruit",-----------------这里指定当前的type类型**

**"\_id":"1"------------------这里指定id**

**},**

**{**

**"\_index":"goods",**

**"\_type":"fruit",**

**"\_id":"2"**

**}**

**]**

**}**

查询结果如下:

{

"docs" : [

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "1",

"\_version" : 1,

"found" : true,

"\_source" : {

"name" : "xiangjiao",

"describe" : "haochi tian",

"price" : 40,

"producer" : "feilvbin",

"tags" : [

"xiangjiao",

"haochi"

]

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "2",

"\_version" : 3,

"found" : true,

"\_source" : {

"name" : "pingguo",

"describe" : "cui",

"price" : 60,

"producer" : "zhongguo",

"tags" : [

"haokan",

"xiang"

]

}

}

]

}

**注意:批量查询的时候,如果只针对\_mget方式查询,必须包含三元素(\_index,\_type,\_id)索引,类别和id,缺一不可**

**注意:完成批量查询操作的时候,可以针对不同的索引(\_index)和类别(\_type)也就是说可以查询不同类型的的数据**

**GET /\_mget**

**{**

**"docs":[**

**{**

**"\_index":"test\_index",--------------------你看我是什么类型的?**

**"\_type":"test\_type",**

**"\_id":"WOQXSmgBliuceLt0NRit"**

**},**

**{**

**"\_index":"goods",----------------------你再看看我是什么类型的**

**"\_type":"fruit",**

**"\_id":"2"**

**}**

**]**

**}**

查询结果如下:

{

"docs" : [

{

"\_index" : "test\_index",

"\_type" : "test\_type",

"\_id" : "WOQXSmgBliuceLt0NRit",

"\_version" : 1,

"found" : true,

"\_source" : {

"name" : "kitty"

}

},

{

"\_index" : "goods",

"\_type" : "fruit",

"\_id" : "2",

"\_version" : 3,

"found" : true,

"\_source" : {

"name" : "pingguo",

"describe" : "cui",

"price" : 60,

"producer" : "zhongguo",

"tags" : [

"haokan",

"xiang"

]

}

}

]

}

还有一种写法,就是统一文档的时候,不同的类别的操作

**GET /goods/\_mget----------------------这里指定了索引类型**

**{**

**"docs":[**

**{-------------------------这里就省略了”\_index”索引**

**"\_type":"goods",----------------只有类型了**

**"\_id":1**

**},**

**{**

**"\_type":"goods",**

**"\_id":2**

**}**

**]**

**}**

当然了这里也可以使用非主流

**GET /goods/\_mget**

**{**

**"docs":[**

**{**

**"\_index":"test\_index",----------------任然可以查询其他所索引的数据**

**"\_type":"test\_type",**

**"\_id":"WOQXSmgBliuceLt0NRit"**

**},**

**{**

**"\_type":"goods",**

**"\_id":2**

**}**

**]**

**}**

最后一种就是各种统一,索引(\_index),类型(\_type)都统一的情况

**GET /goods/fruit/\_mget**

**{**

**"ids":[-------------------因为已经指定了索引和类型,就只剩id了**

**"1","2"**

**]**

**}**

**注意:这里就没有docs了,而是ids.而且最好不要指定其他类型的数据,虽然下面的写法也没啥问题**

**GET /goods/fruit/\_mget**

**{**

**"ids":[**

**"1","2"**

**],**

**"docs":[----------------仍然可以指定其他索引和类型**

**{**

**"\_index":"test\_index",**

**"\_type":"test\_type",**

**"\_id":"WOQXSmgBliuceLt0NRit"**

**}**

**]**

**}**

* 1. \_bulk

批量操作

**POST /\_bulk**

**{"index":{"\_index":"goods","\_type":"fruit","\_id":9}}----设置索引等操作**

**{"name":"yali","describe":"haochi"}-------具体的信息**

**{"update":{"\_index":"goods","\_type":"fruit","\_id":"9"}}-----修改操作**

**{"doc":{"name":"yali","describe":"suan"}}------具体的信息,这里有个doc**

**{"delete":{"\_index":"goods","\_type":"fruit","\_id":9}}----------删除操作**

**{"create":{"\_index":"goods","\_type":"fruit","\_id":10}}------创建操作**

**{"name":"yali","describe":"haochi"}------------具体的信息**

**重点:\_bulk可以执行多种操作,什么类型的操作都可以,没有限制.另外index和create的区别,create就是单纯的创建,如果没有就创建,如果有就报错.index是如果没有就创建,如果有就修改version当做update来使用.**

**注意:行于行之间都是独立的json,没有”,”间隔,而且json之间必须是单独一行开始的,不能连着写.**

**注意: bulk操作中，任意一个操作失败，是不会影响其他的操作的，但是在返回结果里，会告诉你异常日志.**

**注意: bulk request会加载到内存里，如果太大的话，性能反而会下降，因此需要反复尝试一个最佳的bulk size。一般从1000~5000条数据开始，尝试逐渐增加。另外，如果看大小的话，最好是在5~15MB之间。**

1. mapping
   1. mapping到底是什么

**Mapping**,就是对索引库中索引的字段名称及其数据类型进行定义，类似于mysql中的表结构信息。不过es的mapping比数据库灵活很多，它可以动态识别字段。一般不需要指定mapping都可以，因为es会自动根据数据格式识别它的类型，**如果你**需要对某些字段添加特殊属性（如：定义使用其它分词器、是否分词、是否存储等），**就必须手动添加mapping**。

我们在es中添加索引数据时不需要指定数据类型，es中有自动影射机制，字符串映射为string，数字映射为long。通过mappings可以指定数据类型是否存储等属性。

* 1. 查看mapping

**GET /goods/fruit/\_mapping**

查询结果如下:

{

"goods" : {

"mappings" : {

"fruit" : {

"properties" : {

"describe" : {

"type" : "text",

"fields" : {

"keyword" : {

"type" : "keyword",

"ignore\_above" : 256

}

}

},

"name" : {

"type" : "text",

"fields" : {

"keyword" : {

"type" : "keyword",

"ignore\_above" : 256

}

},

"fielddata" : true

},

"price" : {

"type" : "long"

},

"producer" : {

"type" : "text",

"fields" : {

"keyword" : {

"type" : "keyword",

"ignore\_above" : 256

}

}

},

"tags" : {

"type" : "text",

"fields" : {

"keyword" : {

"type" : "keyword",

"ignore\_above" : 256

}

},

"fielddata" : true

}

}

}

}

}

}

* 1. 手动添加mapping

**PUT my\_index**

**{**

**"mappings": {------------------这里就是mapping的信息**

**"doc": {-------------------文档结构**

**"dynamic": false,-------------------非动态mapping**

**"properties": {-------------属性设置,在数据库里叫字段**

**"title": {--------------属性名称(字段名称)**

**"type": "text"----------------数据类型**

**},**

**"name": {----------------属性名称**

**"type": "keyword"--------------数据类型**

**},**

**"age": {-------------------属性名称**

**"type": "integer"---------------数据类型**

**}**

**}**

**}**

**}**

**}**

操作结果如下:

{

"my\_index" : {

"aliases" : { },----------------------------这里指的是别名

"mappings" : {

"doc" : {

"dynamic" : "false",

"properties" : {

"age" : {

"type" : "integer"

},

"name" : {

"type" : "keyword"

},

"title" : {

"type" : "text"

}

}

}

},

"settings" : {--------------这里是mapping的一些设置信息

"index" : {

"creation\_date" : "1546064523531",--------------创建时间

"number\_of\_shards" : "5",-------------分片

"number\_of\_replicas" : "1",----------------节点

"uuid" : "2NqtKPnlTu6SFNNAyAvhpw",----------------mapping的id信息

"version" : {-----------------版本

"created" : "6050499"

},

"provided\_name" : "my\_index"

}

}

}

}

**注意:多次插入mapping映射会报错,提供当前mapping的index已经存在,说白了就是不允许添加多次**

检查索引

**GET my\_index/\_mapping**

结果如下:

{

"my\_index" : {

"mappings" : {

"doc" : {------------------------------这里是数据index索引类型,为doc文档类型,插入数据的时候也要指定为doc类型

"dynamic" : "false",

"properties" : {

"age" : {

"type" : "integer"

},

"name" : {

"type" : "keyword"

},

"title" : {

"type" : "text"

}

}

}

}

}

}

查询当前内容:

**GET my\_index/\_search**

可以看到当前的my\_index下并没有出现任何数据,因为mapping只是确定了下要插入数据的数据类型(基本格式)而已

{

"took" : 0,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 0,

"max\_score" : null,

"hits" : [ ]

}

}

插入一条数据:

**PUT my\_index/doc/1**

**{**

**"title":"滚蛋",**

**"name":"小仙女",**

**"age":12**

**}**

**注意:再次查询的时候,就能看见结果了,会自动匹配数据类型.如果故意把age的值换成”12abc”这样的,插入以后会报错,提示你把age类型由Integer换成String类型**

按条件查询当前数据

**GET my\_index/doc/\_search**

**{**

**"query": {**

**"match": {**

**"name": "小仙女"**

**}**

**}**

**}**

可以查询到相应的结果

1. strict模式

创建一个新的mapping

**PUT my\_index2**

**{**

**"mappings": {**

**"doc": {**

**"dynamic": "strict",--------------------这里改成了strict模式**

**"properties": {**

**"title": {**

**"type": "text"**

**},**

**"name": {**

**"type": "keyword"**

**},**

**"age": {**

**"type": "integer"**

**}**

**}**

**}**

**}**

**}**

插入几条数据:

**PUT my\_index2/doc/1**

**{**

**"name":"kitty",**

**"age":12**

**}**

不报错

**PUT my\_index2/doc/1**

**{**

**"name":"kitty",**

**"age":12,**

**"address":"家里"-------------------------------这里是自定义的属性,自己添加的**

**}**

报错

**注意: dynamic在strict(精确)模式下,不能随意添加自定义属性(字段),通过mapping确定的字段才可以使用,否则会报错,但是dynamic在false模式下就可以随意添加自定义属性,不过不在查询的范围内**

对于自定义属性的操作:

**PUT my\_index/doc/2**

**{**

**"hehe":"不报错"---------可以添加一个非mapping设置的数据**

**}**

可以插入,但是不报错,也不能通过字段查询到当前数据

**GET my\_index/\_search**

**{**

**"query": {**

**"match": {**

**"hehe": "不报错"--------------但是没有匹配结果**

**}**

**}**

**}**

没有查询结果

**注意:对于strict模式也可以使用非mapping设定的字段来保存数据,只是查询的时候不能做为查询依据**

1. index参数

index参数作用是控制当前字段是否被索引，默认为true，false表示不记录，即不可被搜索。

**PUT my\_index3**

**{**

**"mappings": {**

**"doc": {**

**"properties": {**

**"cookie": {**

**"type": "text",**

**"index": false------------------看这里,不能被检索到**

**},**

**"content": {**

**"type": "text",**

**"index": true------------------可以被检索到**

**}**

**}**

**}**

**}**

**}**

插入一条数据,然后根据不同的条件查询

**PUT my\_index3/doc/1**

**{**

**"cookie":"you",**

**"content":"啥内容"**

**}**

根据cookie查询

**GET my\_index3/\_search**

**{**

**"query": {**

**"match": {**

**"cookie": "you"**

**}**

**}**

**}**

根据content查询

**GET my\_index3/\_search**

**{**

**"query": {**

**"match": {**

**"content": "啥内容"**

**}**

**}**

**}**

**注意:查询cookie会报错,查询content则显示正常**

1. index\_options参数

index\_options的作用是用于控制倒排索引记录的内容，有如下四种配置：

docs：只记录doc id

freqs：记录doc id 和term frequencies

positions：记录doc id、 term frequencies和term position

offsets：记录doc id、 term frequencies、term position、character offsets

**注意:text类型的默认配置为positions，其他默认为docs。记录的内容越多，占据的空间越大**

设置index\_options参数

**PUT my\_index4**

**{**

**"mappings": {**

**"doc":{**

**"properties":{**

**"cookie":{**

**"type":"text",**

**"index\_options":"offsets"**

**}**

**}**

**}**

**}**

**}**

插入数据:

**PUT my\_index4/doc/1**

**{**

**"cookie":"我是你要插入的文档"**

**}**

查询插入的数据:

**get my\_index4/doc/1**

1. boost参数

会对当前的设定字段进行加权处理,使查询的字段的权重更高一些

**PUT my\_index5**

**{**

**"mappings": {**

**"doc":{**

**"properties":{**

**"title":{**

**"type":"text",**

**"boost":2-----------------------------这里就是加权处理,设置权重为2,更高也可以,看你查询的必要性**

**},**

**"content":{**

**"type":"text"**

**}**

**}**

**}**

**}**

**}**

然后检查下加权处理过的结果:

**GET my\_index5/\_mapping**

查询结果如下:

{

"my\_index5" : {

"mappings" : {

"doc" : {

"properties" : {

"content" : {

"type" : "text"

},

"title" : {---------------------设置了权重的字段

"type" : "text",

"boost" : 2.0-------------------------权重

}

}

}

}

}

}

添加两条数据:

**PUT my\_index5/doc/1**

**{**

**"title":"this is title1",**

**"content":"this is content1"**

**}**

**PUT my\_index5/doc/2**

**{**

**"title":"this is title2",**

**"content":"this is content2"**

**}**

然后进行检索,分别使用title和content进行检索

**get my\_index5/doc/\_search**

**{**

**"query":{**

**"match": {**

**"title": "title2"------------------------这里使用title字段检索**

**}**

**}**

**}**

**get my\_index5/doc/\_search**

**{**

**"query":{**

**"match": {**

**"content": "content2"--------------------这里使用content字段检索**

**}**

**}**

**}**

查询结果如下:

{

"took" : 0,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 0.5753642,

"hits" : [

{

"\_index" : "my\_index5",

"\_type" : "doc",

"\_id" : "2",

"\_score" : 0.5753642,------------------按照title检索的评分

"\_source" : {

"title" : "this is title2",

"content" : "this is content2"

}

}

]

}

}

------------------------------------------

{

"took" : 0,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 0.2876821,

"hits" : [

{

"\_index" : "my\_index5",

"\_type" : "doc",

"\_id" : "2",

"\_score" : 0.2876821,------------------------按照content检索的评分

"\_source" : {

"title" : "this is title2",

"content" : "this is content2"

}

}

]

}

}

**注意:两者的检索评分是不一样的,按照title的检索评分更高一些,因为加权的关系.也就是说如果权重越高,检索到的得分相应也会越高(匹配度越高),但是不要轻易的进行加权处理,除非有需要的时候,因为放入数据的时候会进行加权计算.默认都是1**

由于es的参数较多,就不逐一介绍了,后面在具体的使用时会详细的说下使用到的参数

1. 对象型属性

**PUT my\_index6/person/2**

**{**

**"name":"kitty",**

**"age":12,**

**"address":{---------------------------这里就是一个地址对象,包含到person中了已经**

**"add":"beijing",**

**"tel":"123456"**

**}**

**}**

这里的地址就是一个对象型属性,因为address里已经包含了其他的一些信息,这些信息对于address来说也是以属性存在的

**get my\_index6/\_mapping**

结果如下:

**{**

**"my\_index6" : {**

**"mappings" : {**

**"person" : {**

**"properties" : {**

**"address" : {--------------------------这里的address定义为properties,也就是属性,这里就看出address是一个以对象形式存在的属性**

**"properties" : {**

**"add" : {**

**"type" : "text",**

**"fields" : {**

**"keyword" : {**

**"type" : "keyword",**

**"ignore\_above" : 256**

**}**

**}**

**},**

**"tel" : {**

**"type" : "text",**

**"fields" : {**

**"keyword" : {**

**"type" : "keyword",**

**"ignore\_above" : 256**

**}**

**}**

**}**

**}**

**},**

**"age" : {**

**"type" : "long"**

**},**

**"name" : {**

**"type" : "text",**

**"fields" : {**

**"keyword" : {**

**"type" : "keyword",**

**"ignore\_above" : 256**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

**}**

对象型的精确查找:

**GET my\_index6/\_search**

**{**

**"query": {**

**"match": {**

**"address.add": "beijing"-------------------这里在匹配查找的时候要用”.”进行关联**

**}**

**}**

**}**

结果如下:

{

"took" : 5,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"skipped" : 0,

"failed" : 0

},

"hits" : {

"total" : 1,

"max\_score" : 0.2876821,

"hits" : [

{

"\_index" : "my\_index6",

"\_type" : "person",

"\_id" : "2",

"\_score" : 0.2876821,

"\_source" : {

"name" : "kitty",

"age" : 12,

"address" : {

"add" : "beijing",

"tel" : "123456"

}

}

}

]

}

}

1. 数据类型

核心数据类型

字符串型：text、keyword（不会分词）

数值型：long、integer、short、byte、double、float、half\_float等

日期类型：date

布尔类型：boolean

二进制类型：binary

范围类型：integer\_range、float\_range、long\_range、double\_range、date\_range

复杂数据类型

数组类型：array

对象类型：object

嵌套类型：nested object

地理位置数据类型：geo\_point、geo\_shape

专用类型：ip（记录ip地址）、completion（实现自动补全）、token\_count（记录分词数）、murmur3（记录字符串hash值）

1. 没有了