

搜索引擎选择 Lucene 或 Elasticsearch 或 Solr

Elasticsearch Solr两者比较

<https://segmentfault.com/a/1190000008684910>

<https://www.cnblogs.com/chowmin/articles/4629220.html>

- ▲

0

▼

1

15622105712

2017/12/12 23:21 iPhone

实践证明ES强一点

评论 (0)

引用此答案

举报
- ▲

0

▼



bboss

2017/12/13 00:21

顶ElasticSearch

评论 (1)

引用此答案

举报



bboss

2017/12/15 18:01

推荐bboss es, 不错的es客户端工具: <https://my.oschina.net/bboss/blog/15561>

回复

举报
- ▲

0

▼



风华神使

2017/12/13 00:39

我所在的企业用 es

评论 (0)

引用此答案

举报
- ▲

0

▼



闻术苑

2017/12/13 07:21

看来开源中国在做全文检索选型咯~~顶ES

评论 (0)

引用此答案

举报
- ▲

0

▼



jetliu1987

2017/12/13 08:24

solr转es了。

评论 (0)

引用此答案

举报

工具

[mysql语句在线转换elasticsearch查询语句](#)

Lucene的使用

[lucene的官网](#)

思路:

1. 首先搜集数据

数据可以是文件系统,数据库,网络上,手工输入的,或者像本例直接写在内存上的.

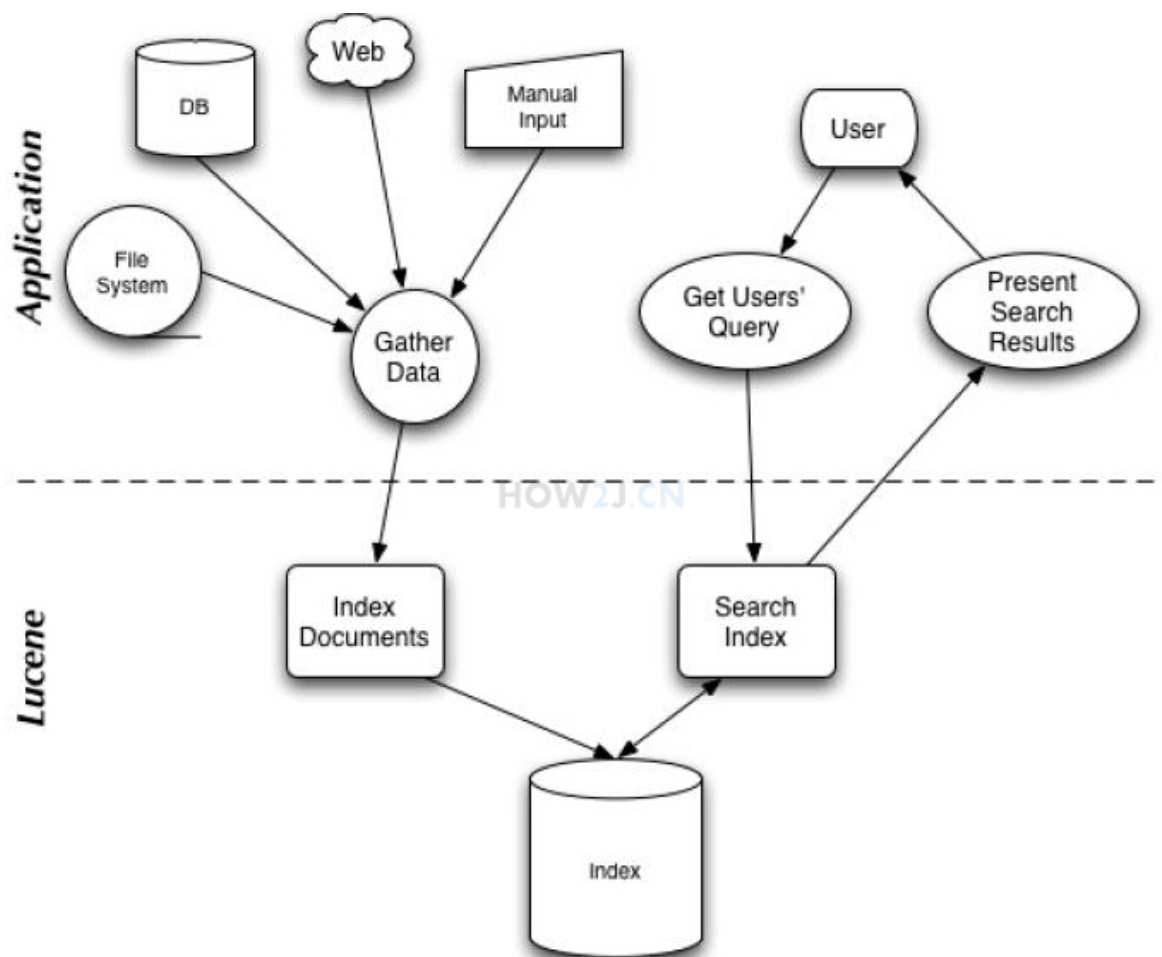
2. 通过数据创建索引

3. 用户输入关键字

4. 通过关键字创建查询器

5. 根据查询器到索引里获取数据

6. 然后把查询结果展示在用户面前



分词器的概念

分词器指的是搜索引擎如何使用关键字进行匹配,如入门中的关键字: 护眼带光源.

如果使用like,那么%护眼带光源%,匹配出来的结果就是要么全匹配,要不都不匹配.

而使用分词器,就会把这个关键字分为 护眼, 带, 光源 3个关键字, 这样就可以找到不同相关程序的结果了.

IKAnalyzer这个分词器很久都没有维护了,也不支持Lucene7. IKAnalyzer这个是修改之后的.

之后通过对demo的运行有了一个初步的理解.

搜索引擎技术相当于将数据进行相关词条的比对,得出适配度,返回查找的数据,以及关键字.

这有点让我想起看得MOOC的编译原理中语法分析.

索引建立好了之后,还是需要维护的,比如新增,删除和维护. 新增就是建立索引的过程,这里就不表了. 索引里的数据,其实就是一个一个的Document对象,那么本文就是介绍如何删除和更新这些Document对象.

删除索引和更新索引

```
// 删除id=51173的数据
IndexWriterConfig config = new IndexWriterConfig(analyzer);
IndexWriter indexWriter = new IndexWriter(index, config);
indexWriter.deleteDocuments(new Term("id", "51173"));
indexWriter.commit();
indexWriter.close();
```

更新索引

```
// 更新索引
IndexWriterConfig config = new IndexWriterConfig(analyzer);
IndexWriter indexWriter = new IndexWriter(index, config);
Document doc = new Document();
doc.add(new TextField("id", "51173", Field.Store.YES));
doc.add(new TextField("name", "神鞭，鞭没了，神还在", Field.Store.YES));
doc.add(new TextField("category", "道具", Field.Store.YES));
doc.add(new TextField("price", "998", Field.Store.YES));
doc.add(new TextField("place", "南海群岛", Field.Store.YES));
doc.add(new TextField("code", "888888", Field.Store.YES));
indexWriter.updateDocument(new Term("id", "51173"), doc );
indexWriter.commit();
indexWriter.close();
```

更新索引后,再用鞭查询,得到的结果是查出了更新之后的数据.

Solr

什么是Solr?

前面学习了Lucene,现在开始学习Solr.

以连接数据库为类比: Lucene就相当于JDBC,是基本的用法.

Solr就相当于MyBatis,方便开发人员配置,访问和调用.

而且Solr被做成了webapp形式,以tomcat的应用的方式启动,提供了可视化的配置界面

启动服务器

切换到solr目录\bin下,执行

```
solr.cmd start
```

如此就启动了服务器, 会占用端口8983, 倘若端口被占用, 会启动失败.

使用solr时, 直接进入solr的bin文件夹, 从而执行命令

```
solr.cmd start
```

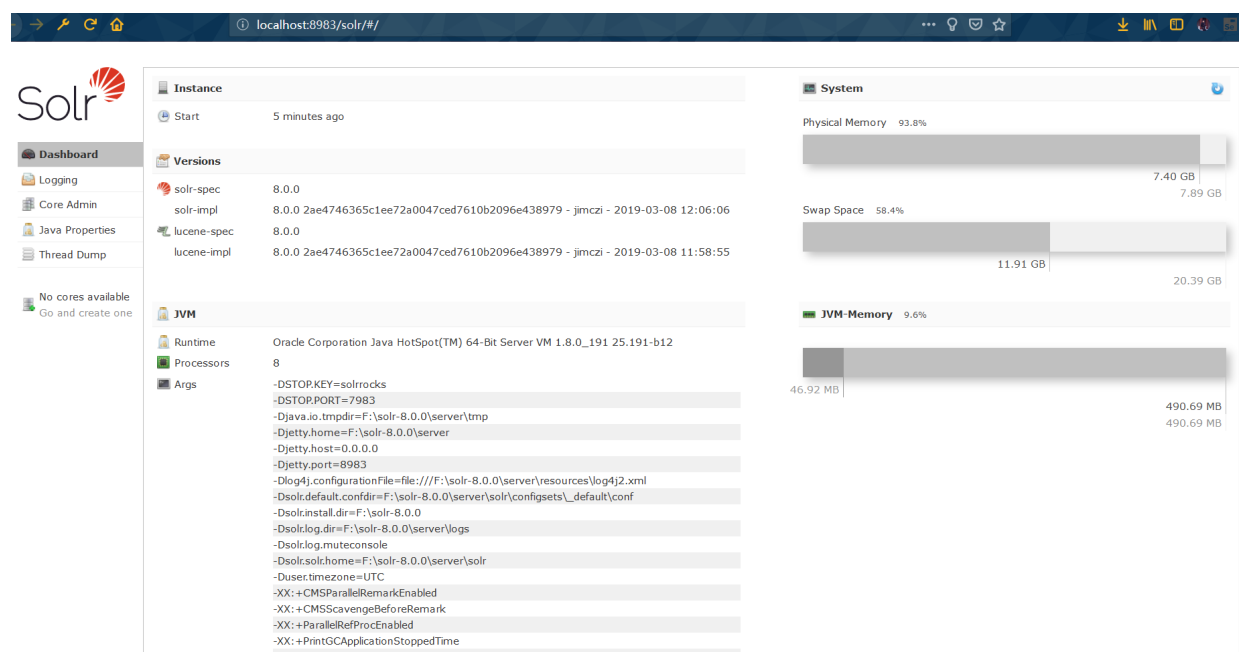
如此就启动了服务器, 会占用端口8983.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.14393]
(c) 2016 Microsoft Corporation。保留所有权利。

C:\Users\X7TI>cd D:\software\solr-7.2.1\bin
C:\Users\X7TI>d:
D:\software\solr-7.2.1\bin>solr.cmd start
WARNING: 32-bit Java detected. Not recommended for production. Point your JAVA_HOME to a 64-bit JDK
Waiting up to 30 to see Solr running on port 8983
Started Solr server on port 8983. Happy searching!
D:\software\solr-7.2.1\bin>

C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.17134.706]
(c) 2018 Microsoft Corporation。保留所有权利。

F:\solr-8.0.0\bin>solr.cmd start
INFO - 2019-05-12 16:15:50.424; org.apache.solr.util.configuration.SSLCredentialProviderFactory; Processing SSL Credential Provider chain: env;sysprop
Waiting up to 30 to see Solr running on port 8983
Started Solr server on port 8983. Happy searching!
F:\solr-8.0.0\bin>
```



访问服务器

输入地址:

<http://localhost:8983/solr/#/>

创建Core

如果说Solr相当于一个数据库的话, 那么Core就相当于一张表

不要通过图形界面创建Core

如图所示,通过图形界面创建Core会失败,应该使用命令行方式创建Core



Error CREATEing SolrCore 'new_core': Unable to create core [new_core]

Dashboard

Logging

Core Admin

Java Properties

Thread Dump

No cores

available

Go and create
one

+ Add Core

name: new_core

instanceDir: new_core

dataDir: data

config: solrconfig.xml

schema: schema.xml

i instanceDir and dataDir need to exist
before you can create the core

✓ Add Core

✗ Cancel

命令行方式创建Core

如图所示就是创建了Core

```
F:\solr-8.0.0\bin>solr.cmd create -c ukyo
WARNING: Using _default configset with data driven schema functionality. NOT RECOMMENDED
To turn off: bin\solr config -c ukyo -p 8983 -action set-user-property -property
false
INFO - 2019-05-12 16:25:33.838; org.apache.solr.util.configuration.SSLCredentialProviderFactory; Processing SSL Credential
ial Provider chain: env;sysprop
Created new core 'ukyo'
F:\solr-8.0.0\bin>_
```

删除new_core

如果点击了步骤 不要通过图形界面创建Core 里的图形界面里的Add Core,那么就会一直有错误提醒,那么按照如下方式删除new_core就不会再有错误提醒了.

```
F:\solr-8.0.0\bin>solr.cmd delete -c new_core
INFO - 2019-05-12 16:30:07.314; org.apache.solr.util.configuration.SSLCredentialProviderFactory; Processing SSL Credential
ial Provider chain: env;sysprop
Deleting core 'new_core' using command:
http://localhost:8983/solr/admin/cores?action=UNLOAD&core=new_core&deleteIndex=true&deleteDataDir=true&deleteInstanceDir=true
```

Solr

Dashboard
Logging
Core Admin
Java Properties
Thread Dump
ukyo
Overview
Analysis
Dataimport
Documents
Files
Ping
Plugins / Stats
Query
Replication
Schema
Segments info

Field Value (Index)
临淄区君网设计工作室

Field Value (Query)

Analyse Fieldname / FieldType: text Schema Browser Verbose Output Analyse Values

ST	text	临	淄	区	君	网	设	计	工	作	室
	raw_bytes	[e4 b6 b4]	[e6 b7 84]	[e5 8c ba]	[e5 90 9b]	[e7 bd 91]	[e8 ae be]	[e8 ae a1]	[e5 b7 a5]	[e4 bd 9c]	[e5
	start	0	1	2	3	4	5	6	7	8	9
	end	1	2	3	4	5	6	7	8	9	10
	positionLength	1	1	1	1	1	1	1	1	1	1
	type	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<II
	termFrequency	1	1	1	1	1	1	1	1	1	1
	position	1	2	3	4	5	6	7	8	9	10
SF	text	临	淄	区	君	网	设	计	工	作	室
	raw_bytes	[e4 b6 b4]	[e6 b7 84]	[e5 8c ba]	[e5 90 9b]	[e7 bd 91]	[e8 ae be]	[e8 ae a1]	[e5 b7 a5]	[e4 bd 9c]	[e5
	start	0	1	2	3	4	5	6	7	8	9
	end	1	2	3	4	5	6	7	8	9	10
	positionLength	1	1	1	1	1	1	1	1	1	1
	type	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<IDEOGRAPHIC>	<II
	termFrequency	1	1	1	1	1	1	1	1	1	1
	position	1	2	3	4	5	6	7	8	9	10
LCF	text	临	淄	区	君	网	设	计	工	作	室
	raw_bytes	[e4 b6 b4]	[e6 b7 84]	[e5 8c ba]	[e5 90 9b]	[e7 bd 91]	[e8 ae be]	[e8 ae a1]	[e5 b7 a5]	[e4 bd 9c]	[e5
	start	0	1	2	3	4	5	6	7	8	9
	end	1	2	3	4	5	6	7	8	9	10
	positionLength	1	1	1	1	1	1	1	1	1	1

默认情况下是没有中文分词的,如上图,得到的是按照每个字的分词结果

配置中文分词.

重启Solr

```
solr.cmd stop -all  
solr.cmd start
```

这里再次进行分词,而FieldType选择新配置的text_ik

Field Value (Index)
山东省淄博市临淄区君网设计工作室

Field Value (Query)

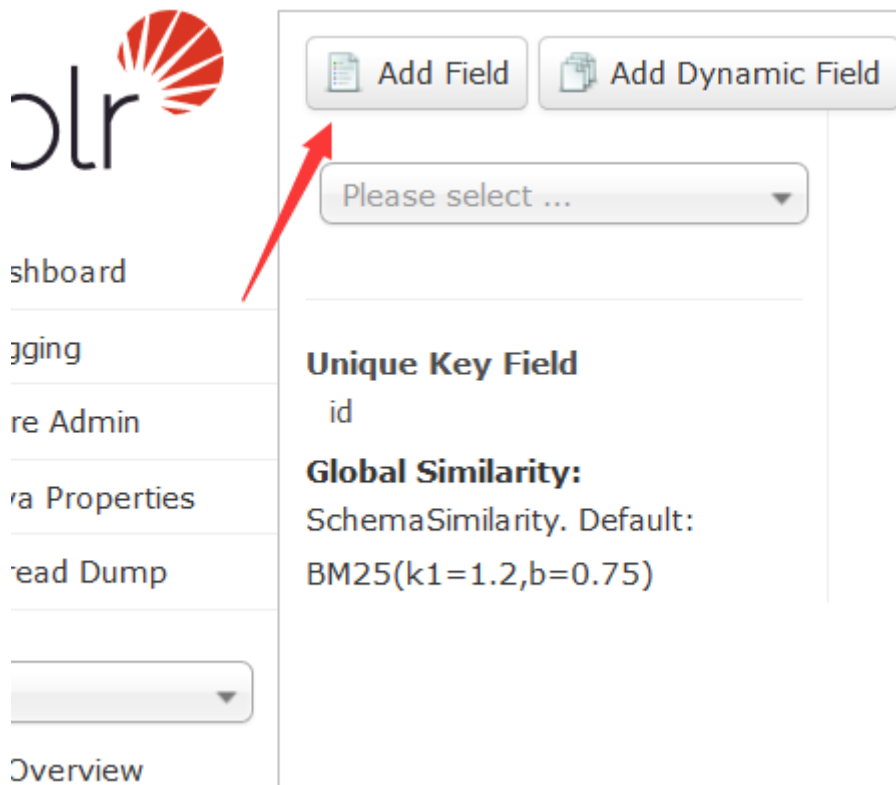
Analyse Fieldname / FieldType: text_ik Schema Browser Verbose Output Analyse Values

IKT	text	山东省	山东	省	淄博市	淄博	市	临淄区	君	网	设计
	raw_bytes	[e5 b1 b1 e4 b8 9c e7 9c 81]	[e5 b1 b1 e4 b8 9c]	[e7 9c 81]	[e6 b7 84 e5 8d 9a e5 b8 82]	[e6 b7 84 e5 8d 9a]	[e5 b8 82]	[e4 b8 b4 e6 b7 84 e5 8c ba]	[e5 90 9b]	[e7 bd 91]	[e8 ae be e8
	start	0	0	2	3	3	5	6	9	10	11
	end	3	2	3	6	5	6	9	10	11	13
	positionLength	1	1	1	1	1	1	1	1	1	1
	type	CN_WORD	CN_WORD	CN_CHAR	CN_WORD	CN_WORD	CN_CHAR	CN_WORD	CN_CHAR	CN_CHAR	CN_WORD
	termFrequency	1	1	1	1	1	1	1	1	1	1
	position	1	2	3	4	5	6	7	8	9	10

设置字段

字段概念

创建Core中的Core就相当于表,那么接下来就要呀为这个表设置字段,用于存放数据.



<http://localhost:8983/solr/#/ukyo/schema> 已经添加了字段了,接下来创建索引

创建索引

<http://how2j.cn/k/search-engine/search-engine-create-index/1682.html>

如何创建索引

Solr提供了一种方式向其中增加索引的界面,但是,不太方便,也和实际工作环境不相符合. 实际工作环境一般都是从数据库里读取数据,然后加入到索引的,很少会通过界面添加索引, 因为这样维护更新删除也不方便,尤其是数据量比较大的时候. 那么,看看如何通过程序把数据加入到Solr索引里.

SolrJ:

Solr支持通过各种各样的语言 (如 php, JavaScript, c#) 把数据加入到索引里, 这里使用第三方工具 SolrJ , 使用Java语言来把数据加入到索引里.

[使用SolrJ操作Solr](#)

The screenshot shows the Solr Admin web interface. On the left is a sidebar with navigation links: Dashboard, Logging, Core Admin, Java Properties, Thread Dump, and a dropdown menu with 'ukyo' selected. Below this are links for Overview, Analysis, Dataimport, Documents, Files, and Ping. The main area is titled 'Request-Handler (qt)' and shows a query path of '/select'. The query parameters are: q=*,*, fq=, sort=, start=0, rows=10, fl=, and df=. The response is a JSON object showing a successful query with 147939 results found. The response includes a 'responseHeader' with status 0, QTime 751, and parameters. The 'response' section contains a list of documents, each with fields like id, name, category, price, place, code, and _version_.

[solr4j demo](#) 已推送到github的solr4j

本来应该先把这14万条记录保存进数据库,然后再从数据库中取出来,考虑到14万条数据从数据库里读取的时间消耗,就改成直接从文件里读取出来,然后转换为泛型是Product的集合的形式,相当于从数据库里读取出来了,不过会快很多.

<http://how2j.cn/k/search-engine/search-engine-create-index/1682.html> 步骤6之后

打开链接看效果.

This screenshot shows the Solr Admin interface with pagination enabled. The query parameters are the same as in the first screenshot, but now include 'wt=json' and 'indent=off'. The response is a JSON object showing a successful query with 147939 results found. The response includes a 'responseHeader' with status 0, QTime 751, and parameters. The 'response' section contains a list of documents, each with fields like id, name, category, price, place, code, and _version_.

solr分页查询

分页时调整了分页的逻辑,但会有重复的id的数据出来,还没调整完成.



Dashboard

Logging

Core Admin

Java Properties

Thread Dump

ukyo

Overview

Analysis

Dataimport

Documents

Files

Ping

Plugins / Stats

Query

Replication

Schema

Segments info

Add Field

Add Dynamic Field

Add Copy Field

Please select ...

Unique Key Field

id

Global Similarity:

SchemaSimilarity. Default:

BM25(k1=1.2,b=0.75)

<https://github.com/deadzq/solr4j-page1.git>

solr高亮显示

<https://github.com/deadzq/solr4j-light.git>

solr更新删除索引

修改之前查询一次

修改之后查询一次

删除之后查询一次

之后再更改回来.

<https://github.com/deadzq/solr4j-update-delete.git>

以上就是Solr的入门级用法,更进一步的学习,请进入Solr官网学习:

<https://lucene.apache.org/solr/>

lucene和solr的测试demo包

链接: <https://pan.baidu.com/s/1NsEXZypi92pKtAbINUBOkQ>

提取码: a64o

复制这段内容后打开百度网盘手机App, 操作更方便哦

lucene和solr的基本告一段落.重点看下面的ElasticSearch~

什么是ElasticSearch

和Solr一样,ElasticSearch也是基于Lucene进行了封装,提供了更为便利的访问和调用.

官网

<https://www.elastic.co/cn/>

Elastic中文社区

<http://www.elasticsearch.cn/>

到官网下载压缩包,之后解压.

1

Download and unzip Elasticsearch

!

Elasticsearch can also be installed from our package repositories using apt or yum, or installed on Windows using an MSI installer package. See [Repositories in the Guide](#).

2

Run `bin/elasticsearch` (or `bin\elasticsearch.bat` on Windows)

3

Run `curl http://localhost:9200/` or `Invoke-RestMethod http://localhost:9200` with PowerShell

4

Dive into the [getting started guide](#) and [video](#).

上图是操作步骤,

1. 介绍

```
{
  "name" : "N_IM2Ey",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "5Sdm-3BkRAmvpVjfne83YA",
  "version" : {
    "number" : "6.2.2",
    "build_hash" : "10b1edd",
    "build_date" : "2018-02-16T19:01:30.685723Z",
    "build_snapshot" : false,
    "lucene_version" : "7.2.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

里面给出了服务器的基本信息，如版本号是6.2.2, 底层用的lucene版本号是7.2.1 等等

注：不同浏览器对于json代码的默认渲染方式是不一样的，截图看到的是火狐的渲染方式，其他浏览器可能是下载一个文件，里面呢内容是这些，不过都表示启动成功了。

elasticsearch工具-Kibana

Kibana 是在ElasticSearch 有了相当多的数据之后，进行分析这些数据用的工具。但是现在还么有数据呀，为什么就要介绍这个工具呢？因为Kibana 里面有一个叫做 Dev Tools的，可以很方便地以 Restful 风格向 ElasticSearch 服务器提交请求，接下来的部分学习，都会使用Kibana 里的这个Dev Tools 来讲解，简单又方便

<https://www.cnblogs.com/cjsblog/p/9476813.html> 详细

Kibana同样是解压后切换到bin下,执行脚本启动

kibana.bat

验证启动

```
http://localhost:5601/app/kibana#/dev_tools/console?_g=()
```

在控制输入

```
GET /_cat/health?v
```

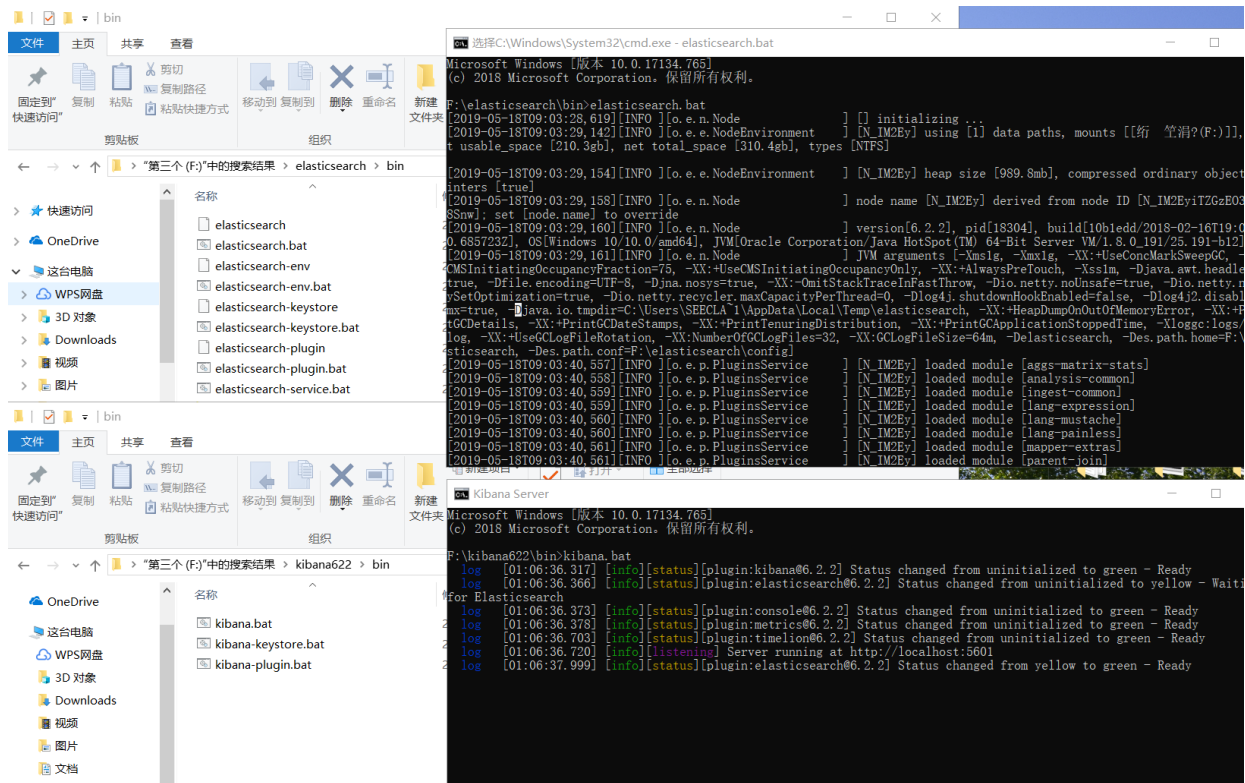
运行测试

然后点击绿色箭头进行运行，就可以看到右侧出现查询结果

GET /_cat/health?v 这个命令用来查看服务器状态（健康度），green 表示一切OK

Console													
1	GET /_cat/health?v												
1	epoch	timestamp	cluster	status	node.total	node.data	shards	pri	relo	init	unassign	pending_tas	
2	ks	max_task_wait_time	active_shards	percent									
2	1558106302	23:18:22	elasticsearch	green	1	1	0	0	0	0	0	0	
3	0			100.0%									

启动kibana时必须先要开启elasticsearch



elasticsearch-kibana-索引管理

索引概念

索引相当于就是一个数据库服务器上的某个数据库,所以索引也可以看成是Elastic Search里的某个数据库

Restful风格

要进行管理索引的工作: 管理无非就是增删改查,即CRUD.

在使用Restful风格之前,进行索引管理需要这样的访问地址: add,delete,update,get 等不同的访问地址来表示不同的业务请求.

但是使用Restful风格,就通过提交不同的method来表示CRUD:

PUT 表示增加

GET 表示获取

DELETE 表示删除

POST 表示更新

Method是http请求里的一个属性,常用的属性值是post和get.

增加索引

在kibana控制台输入如下命令: (索引的增加是通过kibana)

打开kibana控制台:

```
http://localhost:5601/app/kibana#/dev_tools/console?_g=()
```

运行如下命令:

```
PUT /ukyo?pretty
```

我这次返回的是这样,504

```
{
  "statusCode": 504,
  "error": "Gateway Time-out",
  "message": "Client request timeout"
}
```

这里的问题之后解决一下了.

elasticsearch中文分词器

分词器指的是搜索引擎如何使用关键字进行匹配,如入门中的关键字: 护眼带光源.

如果使用like,那么 %护眼带光源%,匹配出来的结果就是要么全匹配,要不就不匹配.

而使用分词器,就会把这个关键字分为护眼, 带, 光源 3个关键字, 这样就可以找到不同相关程度的结果了.

安装elasticsearch中文分词器

对应elasticsearch版本 安装对应版本的分词器

这台电脑 > 第三个 (F:) > elasticsearch701 >				
名称	修改日期	类型	大小	
bin	2019/04/29 13:05	文件夹		
config	2019/04/29 13:05	文件夹		
jdk	2019/04/29 13:05	文件夹		
lib	2019/04/29 13:05	文件夹		
logs	2019/04/29 12:58	文件夹		
modules	2019/04/29 13:05	文件夹		
plugins	2019/05/18 11:02	文件夹		
elasticsearch-analysis-ik-7.0.1.zip	2019/05/18 9:51	8 Zip archive	4,399 KB	
LICENSE.txt	2019/04/29 12:50	TXT 文件	14 KB	
NOTICE.txt	2019/04/29 12:58	TXT 文件	437 KB	
README.textile	2019/04/29 12:50	TEXTILE 文件	9 KB	

如图,进入文件夹,放入中文分词器的zip包.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 10.0.17134.765]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\SeeClanUkyo>F:\elasticsearch701\bin\elasticsearch-plugin install file:\\F:\elasticsearch701\elasticsearch-analysis-ik-7.0.1.zip
-> Downloading file:\\F:\elasticsearch701\elasticsearch-analysis-ik-7.0.1.zip
[=====] 100%??
*****
@ WARNING: plugin requires additional permissions @
*****
* java.net.SocketPermission * connect,resolve
See http://docs.oracle.com/javase/8/docs/technotes/guides/security/permissions.html
for descriptions of what these permissions allow and the associated risks.

Continue with installation? [y/N]y
-> Installed analysis-ik

C:\Users\SeeClanUkyo>
```

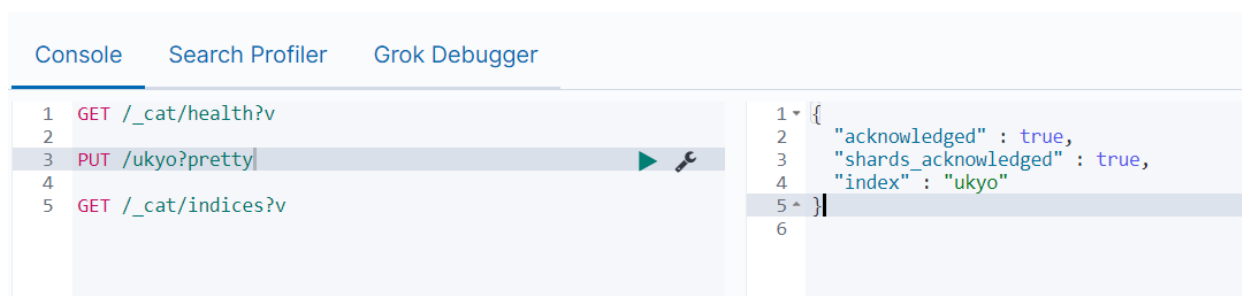
```
C:\Users\SeeClanUkyo>F:\elasticsearch701\bin\elasticsearch-plugin install
file:\\\\F:\elasticsearch701\elasticsearch-analysis-ik-7.0.1.zip
```

执行以上命令,如图中所示即安装完成.

安装插件后要重启,否则无法生效.

重启很简单,就是黑窗口x掉,然后重新运行cmd: elasticsearch.bat

在使用了新的es版本后,加入索引问题得到了解决

[查看索引](#)

```
GET /_cat/indices?v
```

索引相当于就是一个数据库服务器上的某个数据库,所以索引也可以看成是Elastic Search里的某个数据库

索引相当于就是一个数据库服务器上的某个数据库,所以索引也可以看成是Elastic Search里的某个数据库

索引相当于就是一个数据库服务器上的某个数据库,所以索引也可以看成是Elastic Search里的某个数据库

执行下图命令,可以看到新建立的索引.

Console			Search Profiler			Grok Debugger		
1	GET	/_cat/health?v	1	health	status	index	uuid	pri rep docs.count docs.deleted store.size pri.store.size
2			2	green	open	.kibana_task_manager	L09vPZjkQbqSAex-27yrVA	1 0 2 0 12.7kb 12.7kb
3	PUT	/ukyo?pretty	3	green	open	.kibana_1	ihFxp79ZTEymgU-e3yGSSA	1 0 4 0 14.6kb 14.6kb
4			4	yellow	open	ukyo	IeL1eGINQu0ziJI6CeHtMg	1 1 0 0 230b 230b
5	GET	/_cat/indices?v	5					

删除索引

```
DELETE /ukyo?pretty
```

删除成功返回:

```
{
  "acknowledged" : true
}
```

删除失败返回:

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "index_not_found_exception",
        "reason" : "no such index [ukyo]",
        "resource.type" : "index_or_alias",
        "resource.id" : "ukyo",
        "index_uuid" : "_na_",
        "index" : "ukyo"
      }
    ],
    "type" : "index_not_found_exception",
    "reason" : "no such index [ukyo]",
    "resource.type" : "index_or_alias",
    "resource.id" : "ukyo",
    "index_uuid" : "_na_",
    "index" : "ukyo"
  },
  "status" : 404
}
```

继续ES的中文分词器

```
GET /_cat/health?v
PUT /ukyo?pretty
GET /_cat/indices?v
DELETE /ukyo?pretty
GET _analyze
{
  "analyzer":"ik_max_word",
  "text":"护眼带光源"
}
```

```
1 {
2   "tokens" : [
3     {
4       "token" : "护眼",
5       "start_offset" : 0,
6       "end_offset" : 2,
7       "type" : "CN_WORD",
8       "position" : 0
9     },
10    {
11      "token" : "带",
12      "start_offset" : 2,
13      "end_offset" : 3,
14      "type" : "CN_CHAR",
15      "position" : 1
16    },
17    {
18      "token" : "光源",
19      "start_offset" : 3,
20      "end_offset" : 5,
21      "type" : "CN_WORD",
22      "position" : 2
23    }
24  ]
25 }
26
```

在kibana的console中执行

```
GET _analyze
{
  "analyzer":"ik_max_word",
  "text":"护眼带光源"
}
```

返回:

```
{
  "tokens" : [
    {
      "token" : "护眼",
      "start_offset" : 0,
      "end_offset" : 2,
      "type" : "CN_WORD",
      "position" : 0
    },
    {
      "token" : "带",
      "start_offset" : 2,
      "end_offset" : 3,
      "type" : "CN_CHAR",
      "position" : 1
    },
    {
      "token" : "光源",
      "start_offset" : 3,
      "end_offset" : 5,
      "type" : "CN_WORD",
      "position" : 2
    }
  ]
}
```



```

      "position" : 2
    }
  ]
}

```

总体的分词效果还算可以:

```

GET _analyze
{
  "analyzer":"ik_max_word",
  "text":"张三爱上了李梅丽
,这天张三约李梅丽去爱悦之城欢度周末
,张三买了好多成人用品打算大干一番
,李梅丽也穿上了她蓄谋已久的小裤裤."
}

```

```

246 | position : 34
247 ^ | },
248 ^ | {
249 |   "token" : "蓄谋已久",
250 |   "start_offset" : 52,
251 |   "end_offset" : 56,
252 |   "type" : "CN_WORD",
253 |   "position" : 35
254 ^ | },
255 ^ | {
256 |   "token" : "蓄谋",
257 |   "start_offset" : 52,
258 |   "end_offset" : 54,
259 |   "type" : "CN_WORD",
260 |   "position" : 36
261 ^ | },
262 ^ | {
263 |   "token" : "已久",
264 |   "start_offset" : 54,
265 |   "end_offset" : 56,
266 |   "type" : "CN_WORD",
267 |   "position" : 37

```

elasticsearch Kibana - 文档管理

- 步骤1. 增加文档
- 步骤2. 获取文档
- 步骤3. 修改文档1
- 步骤4. 修改文档2
- 步骤5. 删除文档

1. 增加文档

还是kibana的控制台

```
http://localhost:5601/app/kibana#/dev_tools/console?_g=()
```

运行如下命令:

```

PUT /ukyo/product/1?pretty
{
  "name": "蜡烛"
}

```

```
3 PUT /ukyo?pretty
4
5 GET /_cat/indices?v
6
7 DELETE /ukyo?pretty
8
9 PUT /ukyo/product/1?pretty
10 {
11   "name": "蜡烛"
12 }
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

如图所示,增加成功.

注意!!!: 其中的`product`在elastic search里是type的概念,相当于数据库里的表,这里就相当于向product表里插入了一条数据.

2.获取文档

```
#获取文档
GET /ukyo/product/1?pretty
```

正常返回:

```
{
  "_index": "ukyo",
  "_type": "product",
  "_id": "1",
  "_version": 1,
  "found": true,
  "_source": {
    "name": "蜡烛"
  }
}
```

`_index` 表示哪个索引

`_type` 表示哪个表

`_id` 主键

`_version` 版本

`found` 数据存在

`_source` 数据内容

```
4 GET /_cat/health?v
5 #新增索引ukyo
6 PUT /ukyo?pretty
7 #获取索引
8 GET /_cat/indices?v
9 #删除索引ukyo
10 DELETE /ukyo?pretty
11 #中文语法分析,需要安装中文语法分析插件ik...
12 GET _analyze
13 {
14   "analyze": "ik_max_word"
15 }
16 #增加文档
17 #其中的product在elastic search里是type的概念,相当于数据库里的表,这里就相当于向product表里插入了一条数据
18 PUT /ukyo/product/1?pretty
19 {
20   "name": "蜡烛"
21 }
22 #获取文档
23 GET /ukyo/product/1?pretty
```

```
2 {
3   "_index": "ukyo",
4   "_type": "product",
5   "_id": "1",
6   "_version": 1,
7   "_seq_no": 0,
8   "_primary_term": 1,
9   "found": true,
10  "source": {
11    "name": "蜡烛"
12  }
13 }
14
```

来让我再次插入一条文档.查看是否id为2.
获取时拼接2获取.

```
5 #获取文档
6 GET /ukyo/product/1?pretty
7 #再次添加文档,查看id是否指定
8 PUT /ukyo/product/2?pretty
9 {
10   "name": "口香糖"
11 }
12 #获取文档 id:2 OK
13 GET /ukyo/product/2?pretty
14 #再次添加文档,查看id是否指定123
15 PUT /ukyo/product/123?pretty
16 {
17   "name": "中国博物馆"
18 }
19 #获取文档 id:123 OK
20 GET /ukyo/product/123?pretty
```

```
2 {
3   "_index": "ukyo",
4   "_type": "product",
5   "_id": "2",
6   "_version": 1,
7   "_seq_no": 1,
8   "_primary_term": 1,
9   "found": true,
10  "source": {
11    "name": "口香糖"
12  }
13 }
14
```

来再次插入一条文档,id为123的.
获取时直接使用123获取也是可以的.

```
#获取文档
GET /ukyo/product/1?pretty
#再次添加文档,查看id是否指定
PUT /ukyo/product/2?pretty
{
  "name": "口香糖"
}
#获取文档 id:2 OK
GET /ukyo/product/2?pretty
#再次添加文档,查看id是否指定123
PUT /ukyo/product/123?pretty
{
  "name": "中国博物馆"
}
#获取文档 id:123 OK
GET /ukyo/product/123?pretty
```

```
2 {
3   "_index": "ukyo",
4   "_type": "product",
5   "_id": "123",
6   "_version": 1,
7   "_seq_no": 2,
8   "_primary_term": 1,
9   "found": true,
10  "source": {
11    "name": "中国博物馆"
12  }
13 }
14
```

3.修改文档method1

修改两种方式,第一种还是用PUT,PUT本来用来做增加的,但是当输入的id已经存在的时候,就自动变成修改功能了.

```
PUT /ukyo/product/1?pretty
{
  "name": "电灯泡"
}
```

更改显示:

```
#再次添加文档,查看id是否指定123
PUT /ukyo/product/123?pretty
{
  "name": "中国博物馆"
}

#获取文档 id:123 OK
GET /ukyo/product/123?pretty

#修改文档的方法一 使用PUT,PUT掉原来id的值
PUT /ukyo/product/1?pretty
{
  "name": "电灯泡"
}
```

```
{
  "_index": "ukyo",
  "_type": "product",
  "_id": "1",
  "_version": 2,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 3,
  "_primary_term": 1
}
```

查看后显示:

```
#再次添加文档,查看id是否指定123
PUT /ukyo/product/123?pretty
{
  "name": "中国博物馆"
}

#获取文档 id:123 OK
GET /ukyo/product/123?pretty

#修改文档的方法一 使用PUT,PUT掉原来id的值
PUT /ukyo/product/1?pretty
{
  "name": "电灯泡"
}

#获取修改后的id:1
GET /ukyo/product/1?pretty
```

```
{
  "_index": "ukyo",
  "_type": "product",
  "_id": "1",
  "_version": 2,
  "_seq_no": 3,
  "_primary_term": 1,
  "found": true,
  "source": {
    "name": "电灯泡"
  }
}
```

再次获取看看是否已经更改.

4.修改文档method2

修改的第二种方式,使用POST,这才是正规的修改,其实效果和第一种方法是一样的.

```
POST /ukyo/product/1/_update?pretty
{
  "doc":{"name":"聚光灯"}
}
```

更改显示:

```
{
  "name": "中国博物馆"
}

#获取文档 id:123 OK
GET /ukyo/product/123?pretty

#修改文档的方法一 使用PUT,PUT掉原来id的值
PUT /ukyo/product/1?pretty
{
  "name": "电灯泡"
}

#获取修改后的id:1
GET /ukyo/product/1?pretty

#修改文档的方法二 正规的修改方式
POST /ukyo/product/1/_update?pretty
{
  "doc":{"name":"聚光灯"}
}
```

```
{
  "_index": "ukyo",
  "_type": "product",
  "_id": "1",
  "_version": 3,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "_seq_no": 4,
  "_primary_term": 1
}
```

注意上图,在doc的json中name不小心打错了,多了个:号,继而结果成为了这样:

```

requests is deprecated, use
{
  "_index" : "ukyo",
  "_type" : "product",
  "_id" : "1",
  "_version" : 3,
  "_seq_no" : 4,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "电灯泡",
    "name:" : "聚光灯"
  }
}

```

经过删除,再次POST更改后:

```

#获取文档 id:123 OK
GET /ukyo/product/123?pretty

#修改文档的方法一 使用PUT,PUT掉原来id的值
PUT /ukyo/product/1?pretty
{
  "name": "电灯泡"
}

#获取修改后的id:1
GET /ukyo/product/1?pretty

#修改文档的方法二 正规的修改方式
POST /ukyo/product/1/_update?pretty
{
  "doc": {"name": "聚光灯"}
}

```

```

2 {
3   "_index" : "ukyo",
4   "_type" : "product",
5   "_id" : "1",
6   "_version" : 6,
7   "_seq_no" : 7,
8   "_primary_term" : 1,
9   "found" : true,
10  "_source" : {
11    "name" : "聚光灯"
12  }
13 }
14

```

再次查看.

注意!!!: 留意其中的版本已经是3了,第一次创建的时候是1,第一次修改时是2,再次修改是3.(包括删除了这个数据,再次使用同id,依旧会递增_id.

5.删除文档

执行删除:

```
DELETE /ukyo/product/1?pretty
```

```

#获取文档 id:123 OK
GET /ukyo/product/123?pretty

#修改文档的方法一 使用PUT,PUT掉原来id的值
PUT /ukyo/product/1?pretty
{
  "name": "电灯泡"
}

#获取修改后的id:1
GET /ukyo/product/1?pretty

#修改文档的方法二 正规的修改方式
POST /ukyo/product/1/_update?pretty
{
  "doc": {"name": "聚光灯"}
}

#删除id:1表
DELETE /ukyo/product/1?pretty

```

```

instead.
2 {
3   "_index" : "ukyo",
4   "_type" : "product",
5   "_id" : "1",
6   "_version" : 7,
7   "result" : "deleted",
8   "shards" : {
9     "total" : 2,
10    "successful" : 1,
11    "failed" : 0
12  },
13   "_seq_no" : 8,
14   "_primary_term" : 1
15 }
16

```

再次查看id:1

结果:

```
{
  "_index" : "ukyo",
  "_type" : "product",
  "_id" : "1",
  "found" : false
}
```

found为false,说明没有找到.即DELETE命令执行成功.

elasticsearch Kibana - 批量导入

1.批量导入两条数据

还是kibana的控制台

```
http://localhost:5601/app/kibana#/dev_tools/console?_g=()
```

运行如下命令:

```
POST _bulk
{"index":{"_index":"ukyo","_type":"product","_id":10001}}
{"code":"540785126782","price":398,"name":"房屋卫士自流平美缝剂瓷砖地砖专用双组份真瓷胶防水填缝剂镏金色","place":"上海","category":"品质建材"}
{"index":{"_index":"ukyo","_type":"product","_id":10002}}
{"code":"24727352473","price":21.799999237060547,"name":"艾瑞泽手工大号小号调温热熔胶枪玻璃胶枪硅胶条热熔胶棒20W-100W","place":"山东青岛","category":"品质建材"}
{"index":{"_index":"ukyo","_type":"product","_id":10003}}
```

注意: 其中的product在elastic search里是type的概念,相当于数据库里的表,这里就相当于向product表里插入了一条数据.

成功返回:

```
{
  "took" : 427,
  "errors" : false,
  "items" : [
    {
      "index" : {
        "_index" : "ukyo",
        "_type" : "product",
        "_id" : "10001",
        "_version" : 1,
        "result" : "created",
```

```

        "_shards" : {
            "total" : 2,
            "successful" : 1,
            "failed" : 0
        },
        "_seq_no" : 9,
        "_primary_term" : 1,
        "status" : 201
    }
},
{
    "index" : {
        "_index" : "ukyo",
        "_type" : "product",
        "_id" : "10002",
        "_version" : 1,
        "result" : "created",
        "_shards" : {
            "total" : 2,
            "successful" : 1,
            "failed" : 0
        },
        "_seq_no" : 10,
        "_primary_term" : 1,
        "status" : 201
    }
}
]
}

```

2.验证插入的数据

```

43 # "name": "中国博物馆"
44 }
45
46 # 获取文档 id:123 OK
47 GET /ukyo/product/123?pretty
48
49 # 修改文档的方法一 使用PUT
50 PUT /ukyo/product/1?pretty
51 {
52   "name": "电灯泡"
53 }
54
55 # 获取修改后的id:1
56 GET /ukyo/product/1?pretty
57
58 # 修改文档的方法二 正规的修改方式
59 POST /ukyo/product/1/_update?pretty
60 {
61   "doc": {"name": "聚光灯"}
62 }
63
64 # 删除id:1表
65 DELETE /ukyo/product/1?pretty
66
67 #####
68 #####
69 # 批量导入
70 POST _bulk
71 { "index": {"_index": "ukyo", "_type": "product", "_id": "10001"}
72   , "code": "540785126782", "price": 398
73   , "name": "房屋卫士自流平美缝剂瓷砖
74     地砖专用双组份真瓷胶防水填缝剂调
75     金色", "place": "上海", "category": "品质建材"
76 }
77 { "index": {"_index": "ukyo", "_type": "product", "_id": "10002"}
78   , "code": "24727352473", "price": 21
79   , "name": "艾旗漆手工大号小号调温热熔胶枪
80     玻璃胶枪硅胶热熔胶棒20W-100W"
81   , "place": "山东青岛", "category": "品质建材"
82 }
83 { "index": {"_index": "ukyo", "_type": "product", "_id": "10003"}
84   , "code": "540785126782", "price": 398
85   , "name": "房屋卫士自流平美缝剂瓷砖
86     地砖专用双组份真瓷胶防水填缝剂调
87     金色", "place": "上海", "category": "品质建材"
88 }
89
90 # 使用命令查询ukyo索引里所有的数据:
91 GET /ukyo/_search
92 {
93   "query": { "match_all": {} }
94 }
95
96 {
97   "took": 1030,
98   "timed_out": false,
99   "_shards": {
100     "total": 1,
101     "successful": 1,
102     "skipped": 0,
103     "failed": 0
104   },
105   "hits": {
106     "total": {
107       "value": 4,
108       "relation": "eq"
109     },
110     "max_score": 1.0,
111     "hits": [
112       {
113         "_index": "ukyo",
114         "_type": "product",
115         "_id": "123",
116         "_score": 1.0,
117         "_source": {
118           "name": "中国博物馆"
119         }
120       },
121       {
122         "_index": "ukyo",
123         "_type": "product",
124         "_id": "10001",
125         "_score": 1.0,
126         "_source": {
127           "code": "540785126782",
128           "price": 398,
129           "name": "房屋卫士自流平美缝剂瓷砖
130             地砖专用双组份真瓷胶防水填缝剂调
131             金色",
132           "place": "上海",
133           "category": "品质建材"
134         }
135       },
136       {
137         "_index": "ukyo",
138         "_type": "product",
139         "_id": "10002",
140         "_score": 1.0,
141         "_source": {
142           "code": "24727352473",
143           "price": 21,
144           "name": "艾旗漆手工大号小号调温热熔胶枪
145             玻璃胶枪硅胶热熔胶棒20W-100W",
146           "place": "山东青岛",
147           "category": "品质建材"
148         }
149       },
150       {
151         "_index": "ukyo",
152         "_type": "product",
153         "_id": "10003",
154         "_score": 1.0,
155         "_source": {
156           "code": "540785126782",
157           "price": 398,
158           "name": "房屋卫士自流平美缝剂瓷砖
159             地砖专用双组份真瓷胶防水填缝剂调
160             金色",
161           "place": "上海",
162           "category": "品质建材"
163         }
164       }
165     ]
166   }
167 }

```

3.局限性

使用这种方式能够插入的上限较小,在前面的Lucene和Solr教程里,都有14万条的数据要插入,用这种方式就插不进去了.

那么就会使用curl.exe来插入.

elasticsearch curl - 批量导入

curl.exe

curl是一个工具,可以模拟浏览器向服务器提交数据.

需要把curl.exe和要执行导入的数据放到同一个目录下.

上一步撤销重做了,因为电脑系统重装,东西没有找到

访问地址<http://localhost:9200/>还是发现

```
{
  "name" : "LAPTOP-AFF8817V",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "h4HJ6f0GT4WtfyQ8AYQnNA",
  "version" : {
    "number" : "7.0.1",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "e4efcb5",
    "build_date" : "2019-04-29T12:56:03.145736Z",
    "build_snapshot" : false,
    "lucene_version" : "8.0.0",
    "minimum_wire_compatibility_version" : "6.7.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

es运行成功,再次运行kibana

curl批量导入成功

命令:进入curl.exe的目录中

```
curl -H "Content-Type: application/json" -XPOST "localhost:9200/ukyo/product/_bulk?refresh" --data-binary "@products.json"
```

批量导入还是不行,debug一下

首先我删除了名为ukyo的索引,再次生成.

虽然拿小数据json(截取的json),做了测试,但是最终效果不是很好.

然后试了下查询方式,居然能查到总数据的最后一条,这可能是es和kibana的版本问题.

```
GET /ukyo/product/_search
{
  "query": {
    "match": {
      "_id": "157939"
    }
  }
}
```

```
2 {
3   "took" : 2,
4   "timed_out" : false,
5   "_shards" : {
6     "total" : 1,
7     "successful" : 1,
8     "skipped" : 0,
9     "failed" : 0
10  },
11  "hits" : {
12    "total" : {
13      "value" : 1,
14      "relation" : "eq"
15    },
16    "max_score" : 1.0,
17    "hits" : [
18      {
19        "_index" : "ukyo",
20        "_type" : "product",
21        "_id" : "157939",
22        "_score" : 1.0,
23        "_source" : {
24          "code" : "520677584995",
25          "price" : 599,
26          "name" : "永久下架Vero Moda针织时尚一粒扣修身西装女|315308009",
27          "place" : "天津",
28          "category" : "女装会场"
29        }
30      }
31    ]
32  }
33 }
34
```

```
GET /ukyo/product/_search
{
  "query": {
    "match": {
      "_id": "157939"
    }
  }
}
```

再又解压es6.2.2版本和kibana6.2.2版本后,导入数据查询total没问题了.

```
GET /ukyo/_search
{
  "query": {
    "match_all": {}
  }
}
```

```
GET /_cat/dices?v
PUT /ukyo/?pretty
```

```
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "skipped": 0,
8     "failed": 0
9   },
10  "hits": {
11    "total": 147939,
12    "max_score": 1,
13    "hits": [
14      {
15        "_index": "ukyo",
16        "_type": "product",
17        "_id": "10005",
18        "score": 1,
```

查询所有

```
GET /ukyo/_search
{
  "query":{"match_all":{}}
}
```

id倒排序

命令:

```
GET /ukyo/_search
{
  "query":{"match_all":{}},
  "sort":[
    {"_id":"desc"}
  ]
}
```

查出来与教程一样.



The screenshot shows the Kibana console on the left and the search results on the right. The console shows a sequence of commands: `GET /_cat/dices?v`, `PUT /ukyo/?pretty`, `GET /ukyo/_search` (with a match_all query), and `GET /ukyo/_search` (with a match_all query and a sort on _id in descending order). The results on the right show a single hit for index 'ukyo', type 'product', and id '99999'. The hit's source contains fields: code, price (2098), name (G-STAR RAW 春秋新品 男士), place (上海), and category (男装会场). The sort order is confirmed as '99999'.

```
GET /_cat/dices?v
PUT /ukyo/?pretty
GET /ukyo/_search
{
  "query":{"match_all":{}}
}
GET /ukyo/_search
{
  "query":{"match_all":{}},
  "sort":[
    {"_id":"desc"}
  ]
}
```

```
11 "total": 147939,
12 "max_score": null,
13 "hits": [
14   {
15     "_index": "ukyo",
16     "_type": "product",
17     "_id": "99999",
18     "_score": null,
19     "_source": {
20       "code": "532583643700",
21       "price": 2098,
22       "name": "G-STAR RAW 春秋新品 男士",
23       "place": "上海",
24       "category": "男装会场"
25     },
26     "sort": [
27       "99999"
28     ]
29   },
30   {
31     "_index": "ukyo",
32     "_type": "product",
33     "_id": "99998",
34     "_score": null,
```

只返回部分字段

使用命令:

```
GET /ukyo/_search
{
  "query":{"match_all":{}},
  "_source":["name","price"]
}
```

```

16 {
17   "query":{"match_all":{}}
18 }
19
20 GET /ukyo/_search
21 {
22   "query":{"match_all":{}},
23   "sort":[{"_id":"desc"}]
24 }
25
26 #only return few field value ,check out the "_source" values
27 GET /ukyo/_search
28 {
29   "query":{"match_all":{}},
30   "_source":["name","price"]
31 }
32
33

```

```

16 {
17   "_type": "product",
18   "_id": "10005",
19   "score": 1,
20   "source": {
21     "price": 13738,
22     "name": "警视卫4路监控设备套装200万家用一体机套餐"
23   }
24 }
25 {
26   "_index": "ukyo",
27   "type": "product",
28   "_id": "10014",
29   "score": 1,
30   "source": {
31     "price": 960,
32     "name": "水星家纺被芯冬被 加厚保暖特厚被 床J"
33   }
34 }
35 {
36   "_index": "ukyo",
37   "type": "product",
38   "_id": "10018",

```

条件查询

```

GET /ukyo/_search
{
  "query":{"match":{"name":"时尚连衣裙"}}
}

```

```

1 {
2   "query":{"match_all":{}},
3   "sort":[{"_id":"desc"}]
4 }
5
6 #only return few field value ,check out the "_source" values
7 GET /ukyo/_search
8 {
9   "query":{"match_all":{}},
10  "_source":["name","price"]
11 }
12
13 GET /ukyo/_search
14 {
15   "query":{"match":{"name":"时尚连衣裙"}}
16 }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104

```

```

price : 299,
"name": "NEW LOOK 女士时尚连衣裙 | 350536667",
"place": "上海",
"category": "女装会场"
},
{
  "index": "ukyo",
  "type": "product",
  "id": "85477",
  "score": 14.941346,
  "source": {
    "code": "537561133207",
    "price": 1180,
    "name": "PRICH 秋冬时尚印花连衣裙百搭中长款打底连衣裙PI"
  },
  "place": "上海",
  "category": "女装会场"
},
{
  "index": "ukyo",
  "type": "product",
  "id": "154232",
  "score": 14.834494,
  "source": {
    "code": "538653076816",
    "price": 198,
    "name": "女童毛衣连衣裙秋冬小中大童时尚针织连衣纱裙女羽"
  },
  "place": "广东广州",
  "category": "童装会场"
}

```

分页查询

```

GET /ukyo/_search
{
  "query":{"match_all":{}},
  "from":1,
  "size":3,
  "sort":{"_id":{"order":"desc"}}
}

```

```

  }
}

#only return few field value ,check out the "_source" values
GET /ukyo/_search
{
  "query":{"match_all":{}},
  "_source":["name","price"]
}

GET /ukyo/_search
{
  "query":{"match":{"name":"时尚连衣裙"}}
}

GET /ukyo/_search
{
  "query":{"match_all":{}},
  "from":1,
  "size":3,
  "sort":{"_id":{"order":"desc"}}
}

{
  "total": 5,
  "successful": 5,
  "skipped": 0,
  "failed": 0
},
{
  "hits": {
    "total": 147939,
    "max_score": null,
    "hits": [
      {
        "_index": "ukyo",
        "_type": "product",
        "_id": "99998",
        "_score": null,
        "_source": {
          "code": "532663612532",
          "price": 2598,
          "name": "G-STAR RAW 秋冬款 直筒款保暖外套男可调式风帽",
          "place": "上海",
          "category": "男装会场"
        }
      },
      {
        "sort": [
          "99998"
        ]
      }
    ]
  },
  {
    "_index": "ukyo",
    "_type": "product",
    "_id": "99997",
    "_score": null,
    "_source": {
      "code": "539386253685",

```

ELASTICSEARCH - KIBANA 聚合操作

统计数据

```
GET /ukyo/_search
{
  "size": 0,
  "aggs":{
    "group_by_place":{
      "terms":{
        "field":"place.keyword",
        "size": 3
      }
    }
  }
}
```

上面的这个语法相当于sql语句:

```
select count(*),place from product group by place limit 0,3
```

```
    "query":{"match":{"name":"时尚连衣裙"}}
  }
}

GET /ukyo/_search
{
  "query":{"match_all":{}},
  "from":1,
  "size":3,
  "sort":{"_id":{"order":"desc"}}
}

GET /ukyo/_search
{
  "size": 0,
  "aggs":{
    "group_by_place":{
      "terms":{
        "field":"place.keyword",
        "size": 3
      }
    }
  }
}
```

```
2  "took": 118,
3  "timed_out": false,
4  "shards": {
5    "total": 5,
6    "successful": 5,
7    "skipped": 0,
8    "failed": 0
9  },
10 "hits": {
11   "total": 147939,
12   "max_score": 0,
13   "hits": []
14 },
15 "aggregations": {
16   "group_by_place": {
17     "doc_count_error_upper_bound": 2122,
18     "sum_other_doc_count": 87650,
19     "buckets": [
20       {
21         "key": "上海",
22         "doc_count": 29655
23       },
24       {
25         "key": "浙江杭州",
26         "doc_count": 15418
27       },
28       {
29         "key": "广东广州",
30         "doc_count": 15216
31       }
32     ]
33   }
34 }
35 }
```

返回如图所示显示统计了3条数据.

elasticsearch JavaAPI

以上内容都是使用kibana这个工具实现的. 可是实际开发中,肯定会通过变成语言来实现, 那么接下来的教程会讲解如何通过 Java API 实现对 ElasticSearch 的管理.

ELASTICSEARCH - JAVAAPI 索引管理

试了下使用该程序是可以生成索引的.

ELASTICSEARCH - JAVAAPI 文档管理

<https://github.com/deadzq/es4j-doc>

```
//准备数据
Product product = new Product();
product.setId(1);
product.setName("product 1");

//增加文档
addDocument(product);

//获取文档
getDocument(1);

//修改数据
product.setName("producct 2");
//修改文档
updateDocument(product);
//获取文档
getDocument(1);

//删除文档
```

```
deleteDocument(1);  
// 获取文档  
getDocument(1);
```

文档管理也是能操作到对象数据的.

ELASTICSEARCH JAVA API - 批量操作

<https://github.com/deadzq/es4j-batch-insert>

批量插入操作.

ELASTICSEARCH JAVA API - 查询

<https://github.com/deadzq/es4j-search>

springboot融合elasticsearch

版本问题

springboot有一个spring data组件,可以用来连接各种数数据源. 用来连接elasticsearch的是spring-data-elasticsearch.

但是,截至到2018-9-17, spring-data-elasticsearch更新比较慢,其最高版本无法支持教程里的elasticsearch 6.x 版本.

为了支持6.x版本,需要用到奇怪的transportclient来连接,这就不是spring-data系列里的内容了.

考虑到将来,spring data总会支持最新版本的elasticsearch的. 所以我们还是使用 spring-data 来进行链接. 只是, elasticsearch的版本, 我们换成了2.4.2 .kibana版本,也换成了能够连接elasticsearch 2.4.2 的4.6.3版本.

启动elasticsearch2.4.2

照之前启动方法启动

,但是kibana4.x.x版本启动失败

之后在github查找相关demo.

现在记录下以往版本链接

<https://www.elastic.co/downloads/past-releases/kibana-4-6-3>

看上面的链接,最后的kibana-4-6-3 如果改为elasticsearch-4-6-3照样能下载es的老版本

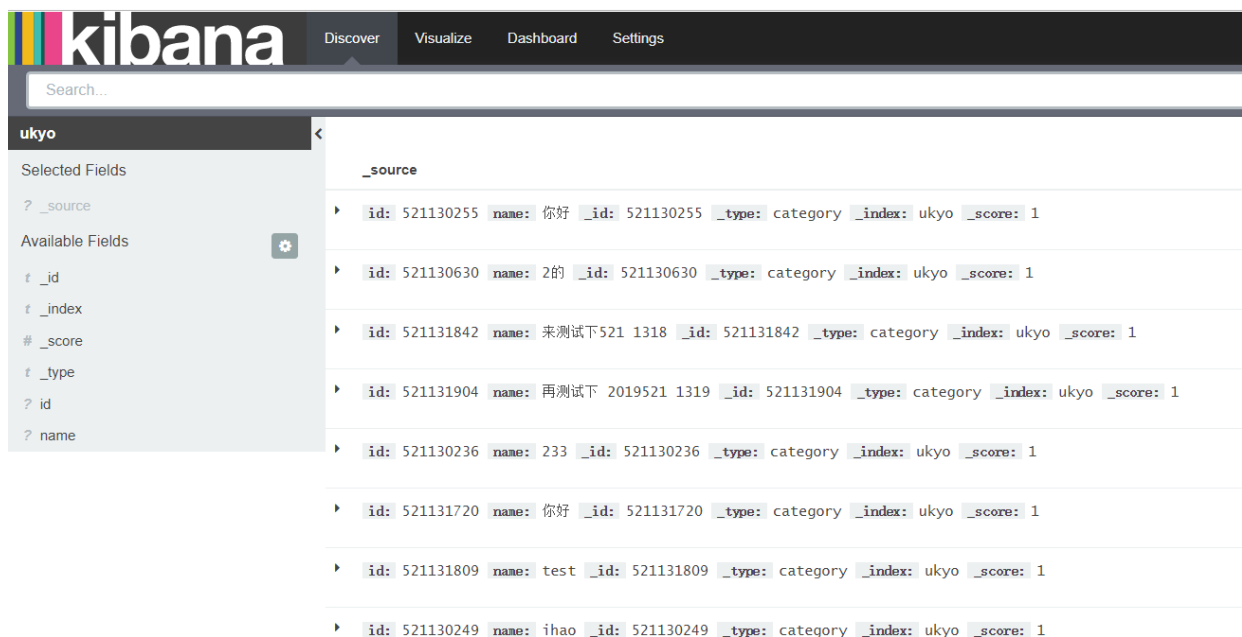
```
Kibana Server
Microsoft Windows [版本 10.0.16299.1087]
(c) 2017 Microsoft Corporation. 保留所有权利。

F:\kibana-4.6.3-windows-x86\bin>kibana.bat
log [10:51:01.881] [info][status][plugin:kibana@1.0.0] Status changed from uninitialized to green - Ready
log [10:51:01.919] [info][status][plugin:elasticsearch@1.0.0] Status changed from uninitialized to yellow - Waiting
for Elasticsearch
log [10:51:01.945] [info][status][plugin:kbn_vislib_vis_types@1.0.0] Status changed from uninitialized to green - Ready
log [10:51:01.951] [info][status][plugin:markdown_vis@1.0.0] Status changed from uninitialized to green - Ready
log [10:51:01.961] [info][status][plugin:metric_vis@1.0.0] Status changed from uninitialized to green - Ready
log [10:51:01.964] [info][status][plugin:spyModes@1.0.0] Status changed from uninitialized to green - Ready
log [10:51:01.968] [info][status][plugin:statusPage@1.0.0] Status changed from uninitialized to green - Ready
log [10:51:01.972] [info][status][plugin:table_vis@1.0.0] Status changed from uninitialized to green - Ready
log [10:51:02.026] [info][listening] Server running at http://0.0.0.0:5601
log [10:51:07.604] [info][status][plugin:elasticsearch@1.0.0] Status changed from yellow to yellow - No existing Kibana index found
log [10:51:12.703] [info][status][plugin:elasticsearch@1.0.0] Status changed from yellow to green - Kibana index ready
```

更换了es版本和相应的kibana版本(463),启动项目后无错.

调节好了jsp的问题后,

可以通过web页面向es,kibana添加索引:



<https://github.com/deadzq/es-demo> 上传的demo