了解一下elasticsearch mapping 之后的属性

name	type	forn
@timestamp	date	
@version	string	
@version.keyword	string	
_id	string	
_index	string	
_score	number	
_source	_source	
_type	string	
answer	string	
answer.keyword	string	
createtime	date	
createuserid	number	
id	number	
isdeleted	boolean	
isenabled	boolean	
question	string	
question.keyword	string	

一.请求体查询方式

GET _search

{} 空查询将会返回索引中所有的文档

GET sselkdb/_search

{}

GET type1,type2/_search

{}

查询一个或者多个类型

GET sselkdb/_search

```
{
  "from": 0
   , "size": 20
}
```

分页查询

携带内容的 GET 请求？

任何一种语言(特别是js)的HTTP库都不允许 GET 请求中携带交互数据。 事实上，有些用户很惊讶 GET 请求中居然会允许携带
交互数据。

真实情况是，http://tools.ietf.org/html/rfc7231#page-24[RFC 7231]， 一份规定HTTP语义及内容的RFC中并未规定 GET 请求
中允许携带交互数据！ 所以，有些HTTP服务允许这种行为，而另一些(特别是缓存代理)，则不允许这种行为。
Elasticsearch的作者们倾向于使用 GET 提交查询请求，因为他们觉得这个词相比 POST 来说，能更好的描述这种行为。 然
而，因为携带交互数据的 GET 请求并不被广泛支持，所以 search API同样支持 POST 请求，类似于这样：

```
POST /_search
{
"from": 0,
"size": 20
}
```

二.结构化查询方式Query DSL

```
GET /_search
{
  "query":{"match": {
    "question": "workbook"
  }}

}
```

蓝色标识的方法众多可以尝试一下

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

空查询，匹配所有文档

```
GET /_search
{
  "query":{"bool": {
  "must": [
    {"match": {
      "question": "workbook"
    }}
  ],"must_not": [
    {"match": {
      "answer": "虚线"
    }}
  ],"should": [
    {"match": {
      "answer": "page"
    }}
  ]
   }
  }
}
```

合并子查询

bool 子句允许你合并其他的合法子句，无论是 must ， must_not 还是 should

三.查询过滤语句

```
GET/_search
{
  "query": {"term": {
    "id": {
      "value": "1953"
    }
  }}
}
```

精确匹配查询

```
GET /_search
{
  "query": {"terms": {
    "id": [
      "1953",
      "2068"
    ]
  }
  }}
}
```

terms指定多个匹配条件

```
GET /_search
{
  "query": {"range": {
    "FIELD": {
      "gte": 10,
      "lte": 20
    }
  }
  }
}
```

范围过滤（指定一个范围）

gt :: 大于
gte :: 大于等于
lt :: 小于
lte :: 小于等于

四.带过滤的查询语句

```
GET /_search
{
 "query": {"match": {
   "FIELD": "TEXT"
 }},
  "post_filter":{
    "term": {
      "FIELD": "VALUE"
    }
  }
}
```

过滤的查询语句

五.排序查询

默认情况下，结果集会按照相关性进行排序 -- 相关性越高，排名越靠前

为了使结果可以按照相关性进行排序，我们需要一个相关性的值。在ElasticSearch的查询结果中， 相关性分值会

用 _score 字段来给出一个浮点型的数值，所以默认情况下，结果集以 _score 进行倒序排列。

```
GET / _search
{
 "query": {"match_all": {}},
"post_filter":{
  "term": {
    "id": "1905"
  }
}
}
```

过滤语句与 _score 没有关系，但是有隐含的查询条件 match_all 为所有的文档的 _score 设值为 1 。 也就相当于所有的

文档相关性是相同的。

```
GET  /_search
{
 "query": {"match_all": {}},
"post_filter":{
  "term": {
    "id": "1905"
  }
},"sort": [
  {
    "createtime": {
      "order": "desc"
    }
  }
]
}
```

当我们根据创建时间 createtime进行排序时
_score 字段没有经过计算，因为它没有用作排序。
createtime字段被转为毫秒当作排序依据

其次就是 _score 和 max_score 字段都为 null 。计算 _score 是比较消耗性能的, 而且通常主要用作排序 -- 我们不是用相
关性进行排序的时候，就不需要统计其相关性。 如果你想强制计算其相关性，可以设置 track_scores 为 true 。

```
GET  /_search
{
 "query": {"match_all": {}},
"post_filter":{
  "term": {
    "id": "1905"
  }
```

```
},"sort": [
  {
    "createtime": {
      "order": "desc"
    }
    ,"id":{
      "order": "desc"
    }
  }
]
}
```

多级排序

```
GET /_search?sort=createtime:desc
```

字符串参数排序

对于数字和日期，你可以从多个值中取出一个来进行排序，你可以使用 min ， max ， avg 或 sum 这些模式。 比说你可以在
createdate字段中用最早的日期来进行排序：

```
sort": [
  {
    "createtime": {
      "order": "desc",
      "mode": "min"
    }
```