[**elasticsearch 基础 语法总结**](https://www.cnblogs.com/cxygg/p/9471372.html)

**1. es 使用 restful 风格的 api**

**备注： es 的 api  格式  基本是这个样     请求方式    /索引名/文档名/id?参数   ，但是 还有 很多不是这样的 请求，比如 \_cat api  等等........**

**2. 查询 所有索引 get /\_cat/indices**

**3.查看节点健康 get /\_cat/health?v**

**4.？v 的意思 显示列出项 的title**

**5.?pretty 结果 json 格式化的方式输出**

**6.添加索引  put /test1**

**7. 添加一条 docment（ 如果已经存在 那么久全部覆盖）**

**[复制代码](javascript:void(0);)**

**PUT /test1/d1/1**

**{**

**"age":1,**

**"name":"zs",**

**"bri":"2018-08-08"**

**}**

**[复制代码](javascript:void(0);)**

**备注1：如果已经存在就全覆盖修改。**

**备注2:如果 只需要插入，不修改 着 在后面 加上 /\_create (这时候会提示已经存在),post 不能带\_create**

**8.修改 一条文档**

**[复制代码](javascript:void(0);)**

**POST /test1/d1/2**

**{**

**"age":1,**

**"name":"zs",**

**"bri":"2018-08-08"**

**}**

**[复制代码](javascript:void(0);)**

**备注: post 虽然叫做修改，但是 在带有id 的情况下 和 put 几乎一样（id 存在就是 全量 修改，不存在就是 新增）。**

**备注2: 可以 使用  POST /test1/d1  不带id 的方式 自动生成id，put 不支持不带id的写法。**

**备注3 post 可以 指定\_update  ,并且 可以带\_create。**

**备注4：post 可以部分更新 专用名字  partial update**

**[复制代码](javascript:void(0);)**

**POST /test1/d1/5/\_update**

**{**

**"doc":{**

**"age":2**

**}**

**}**

**[复制代码](javascript:void(0);)**

**9.删除一条文档      DELETE  test1/d1/5**

**10. 删除一条索引    DELETE  test1**

**11. 关于 primary shard  和 replica shard 的解释**

**primary shard：se 对数据进行切片 ，吧一个索引的 数据 分成多份 ，每一份数据就是一个  primary shard， primary shard 的数量只能在创建索引的时候指定，因为后期 修改  primary shard 数量 会乱 文档id 计算 文档所在 shard 的 结果。**

**replica shard: 副本节点，多个副本节点可以提高数据的安全性，并且可以分担 primary shard 的查询 负载。**

**备注: 修改只能发生在 primary shard ，查询可以发生在任意 shard 。**

**12.查询单条 文档    get user/student/2**

**13. 搜索文档  get user/\_search   或者 get user/syudent/\_search**

**备注：查询可以不指定 type 的类型**

**14  url 的查询  get user/\_search?-q=name2 ，+q是默认的，-q便是 不存在这样的数据**

**备注：没有指定查询字段，使用的 一个特俗的 包含全部字段的字段查询的。**

**15  指定 字段 url 的查询   get user/student/\_search?q=name:n5**

**备注：+q=name:n5&q=name:n5**

**16  url query 的方式很难应 复杂查询 所以我们一般使用  json 格式的请求体的方式**

**备注：es restful 风格api 的 get 请求 支持请求体**

**17 查询 所有**

**[复制代码](javascript:void(0);)**

**get /user/student/\_search**

**{**

**"query":{**

**"match\_all": {}**

**}**

**}**

**[复制代码](javascript:void(0);)**

**18 指定字段查询**

**[复制代码](javascript:void(0);)**

**get /user/student/\_search**

**{**

**"query":{**

**"match": {**

**"name": "n5"**

**}**

**}**

**}**

**[复制代码](javascript:void(0);)**

**19 范围 查询**

**[复制代码](javascript:void(0);)**

**get /user/student/\_search**

**{**

**"query":{**

**"range": {**

**"bri": {**

**"gte": 10,**

**"lte": 20**

**}**

**}**

**}**

**}**

**[复制代码](javascript:void(0);)**

**20 多条件的 复合 查询**

**[复制代码](javascript:void(0);)**

**get /user/student/\_search**

**{**

**"query":{**

**"bool": {**

**"must": [**

**{ "match": {**

**"FIELD": "TEXT"**

**}}**

**],**

**"should": [**

**{"match": {**

**"FIELD": "TEXT"**

**},**

**"match": {**

**"FIELD": "TEXT"**

**}**

**}**

**],**

**"minimum\_should\_match": 1**

**}**

**}**

**}**

**[复制代码](javascript:void(0);)**

**备注：bool 里面的 都是一些 条件 ，must 必须瞒足，should 只要要满足 minimum\_should\_match 个 条件是ture ，filter 只是过滤 不计入评分。**

**21:  查询非分页**

**[复制代码](javascript:void(0);)**

**get /user/student/\_search**

**{**

**"query":{**

**"match\_all": {}**

**},**

**"from":3,**

**"size":2**

**}**

**[复制代码](javascript:void(0);)**

**备注：深分页问题，效率会很低，劲量避免深分页。**

**备注2：深分页：如果要查询出 每页 100 条，的第 100 页数据数据（ 9900 - 10000 ），如果是去5 个节点查询，那么会在 每个节点查询出 第 9900- 10000 条数据，然后 汇总到 坐标几点，然后排序后取出 9900-10000 条，这样做非常占 资源。**

**22. scoll 游标查询，指定 scroll=时间 ，指定保存的分钟数，第一次发起请求放回的不是数据，而是 \_scroll\_id ，后面通过 \_scroll\_id 去请求数据，非常适合大批量查询。**

**[复制代码](javascript:void(0);)**

**get /user/student/\_search?scroll=1m**

**{**

**"query":{**

**"match\_all": {}**

**},**

**"size":2**

**}**

**[复制代码](javascript:void(0);)**

**[复制代码](javascript:void(0);)**

**GET /\_search/scroll**

**{**

**"scroll": "1m",**

**"scroll\_id" : "DnF1ZXJ5VGhlbkZldGNoBQAAAAAAAAIuFkRMbVZ0WFdvU1ZHWEJuelFIQk4tdFEAAAAAAAACLBZETG1WdFhXb1NWR1hCbnpRSEJOLXRRAAAAAAAAAi0WRExtVnRYV29TVkdYQm56UUhCTi10UQAAAAAAAAO1FlQwSkJqVng5UVpPUTIwbWw0a0NKV3cAAAAAAAADthZUMEpCalZ4OVFaT1EyMG1sNGtDSld3"**

**}**

**[复制代码](javascript:void(0);)**

**备注：游标 查询 是在 es 里面缓存了结果 ，然后一次 一次的去取 所以发起 第一次请求的时候只有 size ，没有from ，后面的 请求只有 scroll\_id 和  scroll 时间**

**23: 只 显示指定结果 （ \_source ）**

**[复制代码](javascript:void(0);)**

**GET /user/\_search**

**{**

**"query": {**

**"match\_all": {}**

**},**

**"\_source": ["bri"]**

**}**

**[复制代码](javascript:void(0);)**

**24： post\_filter  和 query 的 区别 ，语法上没区别，唯一的在于 filter 不评分，所以 filter  比 query  快很多 ，filter 和query  可以共存。**

**[复制代码](javascript:void(0);)**

**GET /user/\_search**

**{**

**"post\_filter": {**

**"match\_all": {}**

**},**

**"\_source": ["bri"]**

**}**

**[复制代码](javascript:void(0);)**

**25 聚合函数  球了平均值和 总数量**

**[复制代码](javascript:void(0);)**

**GET user/student/\_search**

**{**

**"query": {**

**"match\_all": {}**

**},**

**"aggs": {**

**"total\_count": {**

**"value\_count": {**

**"field": "age"**

**}**

**},**

**"pjz":{**

**"avg": {**

**"field": "age"**

**}**

**}**

**}**

**}**

**[复制代码](javascript:void(0);)**

**26:分组**

**[复制代码](javascript:void(0);)**

**GET user/student/\_search**

**{**

**"query": {**

**"match\_all": {}**

**},**

**"aggs": {**

**"fz": {**

**"terms": {**

**"field": "age"**

**}**

**}**

**}**

**}**

**[复制代码](javascript:void(0);)**

**27 ： 遇到 Fielddata is disabled on text fields by default 异常的 解决方案**

**因为text类型默认没有正排索引，所以不支持排序 和聚合 ，遇到这种 异常的时候 需要制定 开启正排索引。**

**倒排索引用于搜索，正排索引用于排序和聚合。**

**28：开启正排索引的方法。**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9** | **put user/\_mapping/student/**  **{**  **"properties": {**  **"sex": {**  **"type":"text",**  **"fielddata":true**  **}**  **}**  **}** |

**29: 批查询 api mget**

**[复制代码](javascript:void(0);)**

**#批量查询**

**GET /\_mget**

**{**

**"docs":[**

**{**

**"\_index":"user",**

**"\_type":"student",**

**"\_id":1**

**},**

**{**

**"\_index":"user",**

**"\_type":"student",**

**"\_id":2**

**},**

**{**

**"\_index":"user",**

**"\_type":"student",**

**"\_id":2**

**}**

**]**

**}**

**[复制代码](javascript:void(0);)**

**备注：mget  如果请求url 里面有 index 和 type 后面 的 请求体里面就可以不写  index 和type**

**如:**

**[复制代码](javascript:void(0);)**

**#批量查询**

**GET /user/\_mget**

**{**

**"docs":[**

**{**

**"\_type":"student",**

**"\_id":1**

**},**

**{**

**"\_type":"student",**

**"\_id":2**

**},**

**{**

**"\_type":"student",**

**"\_id":21111111**

**}**

**]**

**}**

**[复制代码](javascript:void(0);)**

**30 批处理 bulk**

**bulk的格式：action：index/create/update/delete 后面如果有请求体就跟上请求体**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **GET \_bulk**  **{"create":{"\_index":"user","\_type":"student","\_id":"100"}}**  **{ "name": "zhaoer","age": 7,  "sex": "nn"}**  **{"update":{"\_index":"user","\_type":"student","\_id":"100"}}**  **{"doc":{ "name": "zhaoer","age": 7,  "sex": "nn"}}**  **{"delete":{"\_index":"user","\_type":"student","\_id":"100"}}**  **{"index":{"\_index":"user","\_type":"student"},"\_id":"100"}**  **{"doc":{ "name": "zhaoer","age": 7,  "sex": "nn"}}** |

**备注：delete 没有请求体。**

**备注2：create 和 update 只有在指定的状态才能成功 create 创建 ，update 更新。**

**备注3 ，批处理中的一个 失败不影响 其他的 执行。**

**备注4 ，update 需要 doc 包一层**

**备注5 ，index 有create 和 update 的 功能，并且支持  又或者 没有 doc 包一层都支持。**

**31 ，es  的删除是 是假删除并且在下一次merge的时候真删除**

**32，es的 并发处理 ，使用的乐观锁 在 后面加上 version**

**POST /user/student/1?version=3**

**{**

**"name":"zyk",**

**"age":0**

**}**

**备注：只有version = 当前记录的version的时候才能修改成功**

**33.es 可以自动控制 vserion 通过 ，version\_type指定 ，**

|  |  |  |
| --- | --- | --- |
| **1** | | **version\_type=external 要求 version 大于当前的version ，** |
| **1** | **version\_type=internal 这个是默认值 ，必须等于当前的值<br>version\_type=external\_gte 大于等于当前的version<br>version\_type=force 已经废弃了不能用了，我也不知道以前什么意思，提升， Validation Failed: 1: version type [force] may no longer be used** | | |

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **POST  /user/student/1?version\_type=external&version=505**  **{**  **"name":"zyk",**  **"age":0**  **}** |

**34 ,使用 consistency 指定写一致性 的等级  (可能废除了，在6.0 不生效)**

**consistency=one 只要主节点活着就可以写**

**consistency=all 所有主节点和副本节点都活着**

**consistency=quorun 所有主节点都活着，并且有超过一半的节点（primary shard + replica shard ）活着 ，这个是默认值,而且只有在有副本节点的时候才生效**

**等待 这时候可以指定 timeout 来指定等待时间timeout=30s**

**35 查询 索引 的设置**

|  |  |
| --- | --- |
| **1** | **GET /user/\_settings** |

**36 查看 索引的 mapping**

**GET user/\_mapping**

**37动态的 mapping  es  或根据第一次存入的数据，动态的决定这个字段的 mapping 类型，并且决定索引行为,后面类型不符合就没法存入，mapping 里面的 类型不能修改，只能添加新的。**

**[复制代码](javascript:void(0);)**

**put /test2/t/1**

**{**

**"age":1,**

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

**"bri":"2017-09-09",**

**"isDel":true,**

**"amount":0.1**

**}**

**GET /test2/\_mapping**

**[复制代码](javascript:void(0);)**

**38, 指定 mapping 只能给新的索引指定 ，或者 个新的字段指定**

**[复制代码](javascript:void(0);)**

**PUT /test2/\_mapping/t**

**{**

**"properties": {**

**"age": {**

**"type": "long"**

**},**

**"amount": {**

**"type": "float"**

**},**

**"bri": {**

**"type": "date"**

**},**

**"isDel": {**

**"type": "boolean"**

**},**

**"name": {**

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

**"fields": {**

**"keyword": {**

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

**"ignore\_above": 256**

**}**

**}**

**}**

**}**

**}**

**[复制代码](javascript:void(0);)**

**39,指定 索引的 setting**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **PUT test3**  **{**  **"settings": {**  **"index": {**  **"number\_of\_shards": "3",**  **"number\_of\_replicas": "1"**  **}**  **}**    **}** |

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6** | **PUT test4/\_settings**  **{**  **"index": {**  **"number\_of\_replicas": "4"**  **}**  **}** |

**备注：number\_of\_shards 不能修改**

**40：mapping 里面 keyword 的可以指定 text 的子类型**

**41： 如果 字段类型是 json ，那么这个字段的类型就是 object ，或者说是docment 这时候  mapping 里面是 映射了一个property**

**42 es 2.X  的 时候string 现在改成  keyword 和 text ，keyword 是存不分词的 的关键数据，text 存大数据，要分词，**

**可以个text 类型 指定fields 来指定 一个 不分词的原文，用于 排序聚合**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **"name": {**  **"type": "text",**  **"fields": {**  **"keyword": {**  **"type": "keyword"**  **}**  **}**  **}** |

**备注：和2.X的 raw 类似。**

**43，query phase 指的的查询的 查询请求，->坐标节点分发请求到对应 shard ，然后 结果汇总到 坐标节点的过程。**

**44， fetch phase 指的是 查询结果的doc id 到各个 shard 取文档的过程。**

**45 通过 preference 指定 取那些节点查询。**

**preference = \_primary,\_primart\_first,\_local,\_only\_node:xyz,\_prefer\_node:xyz,\_shard:2,3**

**46 . timeout  指定多少时间之内要返回，到了时间一定会返回，即便没有查询完，只返回查到的部分。**

**47. routing ， 默认是通过id 路由的。 可以让类似的结果在同一个 shard 上。**

**curl -XPOST 'http://localhost:9200/store/order?routing=user123' -d ' { "productName": "sample", "customerID": "user123" }'**

**48 searche\_type**

**1、query and fetch  
向索引的所有分片（shard）都发出查询请求，各分片返回的时候把元素文档（document）和计算后的排名信息一起返回。这种搜索方式是最快的。因为相比下面的几种搜索方式，这种查询方法只需要去shard查询一次。但是各个shard返回的结果的数量之和可能是用户要求的size的n倍。**

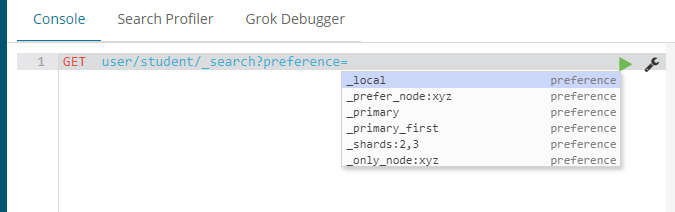
**2、query then fetch（默认的搜索方式）  
如果你搜索时，没有指定搜索方式，就是使用的这种搜索方式。这种搜索方式，大概分两个步骤，第一步，先向所有的shard发出请求，各分片只返回排序和排名相关的信息（注意，不包括文档document)，然后按照各分片返回的分数进行重新排序和排名，取前size个文档。然后进行第二步，去相关的shard取document。这种方式返回的document可能是用户要求的size的n倍,此处勘误，这是原博客中的错误，经测试 query then fetch 方式返回的数量就是 查询是 setSize()的数量**

**3、DFS query and fetch  
这种方式比第一种方式多了一个初始化散发(initial scatter)步骤，有这一步，据说可以更精确控制搜索打分和排名。这种方式返回的document与用户要求的size是相等的。同样勘误 DFS query and fetch 返回结果的数量是 分片数\*size**

**4、DFS query then fetch  
比第2种方式多了一个初始化散发(initial scatter)步骤。这种方式返回的document与用户要求的size是相等的。**

**49 Bouncing Results**

**搜索同一query，结果ES返回的顺序却不尽相同，这就是请求轮询到不同分片，而未设置排序条件，相同相关性评分情况下，由于评分采用的算法时TF(term frequency)和IDF(inverst document frequecy) 算出的总分在不同的shard上时不一样的，那么就造成了默认按照\_score的分数排序，导致会出现结果不一致的情况。查询分析时将所有的请求发送到所有的shard上去。可用设置preference为字符串或者primary shard插叙等来解决该问题。preference还可以指定任意值，探后通过这个值算出查询的节点**

****

**50  type  其实就是一个 隐藏的 field**

**51  mapping root object 就是指 索引mapping 的json对象**

**52 给索引取一个名字**

**put user/\_aliases/user\_al**

**或者：**

**[复制代码](javascript:void(0);)**

**POST /\_aliases**

**{**

**"actions": [**

**{**

**"add": {**

**"index": "user",**

**"alias": "user\_a"**

**}**

**},**

**{**

**"add": {**

**"index": "user",**

**"alias": "user\_b"**

**}**

**}**

**]**

**}**

**[复制代码](javascript:void(0);)**

**备注： 对个索引可以使用同一个别名**

**53 查询 别名**

**get /user/\_alias**

**54 删除别名**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17** | **POST /\_aliases**  **{**  **"actions": [**  **{**  **"remove": {**  **"index": "user",**  **"alias": "user\_a"**  **}**  **},**  **{**  **"remove": {**  **"index": "test",**  **"alias": "user\_b"**  **}**  **}**  **]**  **}** |

**55 自定义 分词器**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27** | **PUT /user5**  **{**  **"settings":{**  **"analysis": {**  **"char\_filter": {**  **"my\_char\_filter":{**  **"type":"mapping",**  **"mappings":["&=> and"]**  **}**  **},**  **"filter": {**  **"my\_filter":{**  **"type":"stop",**  **"stopwords":["the","a"]**  **}**  **},**  **"analyzer": {**  **"my\_analyzer":{**  **"type":"custom",**  **"char\_filter":[ "my\_char\_filter" ],**  **"filter":["my\_filter"],**  **"tokenizer":"standard"**  **}**  **}**  **}**  **}**  **}** |

**解释：定义了一个 char\_filter  名叫 my\_char\_filter，类型是 mapping 把& 转成 and**

**定义了一个 filter 名叫 my\_filter ，类型是停用词，把 the ，a  去掉**

**定义了一个分析器 名叫 my\_analyzer, 类型是自定义，它使用了 char\_filter 是 my\_char\_filter ,它使用的 filter 是 my\_filter ,它使用的 分词器是 标准分词器。**

**例子二：**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18**  **19**  **20**  **21**  **22**  **23**  **24**  **25**  **26**  **27**  **28**  **29**  **30**  **31**  **32**  **33**  **34** | **PUT /user9**  **{**  **"settings":{**  **"analysis": {**  **"char\_filter": {**  **"my\_char\_filter":{**  **"type":"mapping",**  **"mappings":["& => and", "pingguo => pingg"]**  **}**  **},**  **"filter": {**  **"my\_filter":{**  **"type":"stop",**  **"stopwords":["the","a"]**  **}**  **},**  **"analyzer": {**  **"my\_analyzer":{**  **"type":"custom",**  **"char\_filter":[ "my\_char\_filter" ],**  **"filter":["my\_filter"],**  **"tokenizer":"standard"**  **}**  **}**  **}**  **}**  **}**      **GET /user9/\_analyze**  **{**  **"analyzer":"my\_analyzer",**  **"text":" a d&og is in the house pingguo"**  **}** |

**56 自定义 动态mapping**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18** | **PUT my\_index1**  **{**  **"mappings": {**  **"\_doc":{**  **"dynamic":"strict",**  **"properties":{**  **"name":{**  **"type":"text"**  **},**  **"user":{**  **"type":"object",**  **"dynamic":"true"**  **}**    **}**  **}**  **}**  **}** |

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **GET my\_index1/\_doc/1**  **{**  **"name":"name1",**  **"user":{**  **"name":"n1",**  **"age":10**  **},**  **"age":2**    **}** |

**mapping的第一层，也就是 properties的 那一层，不允许动态映射，有新的字段就报错，**

**user的那一层，允许动态映射，有新的字段就根据 新的第一次的值，指定类型。**

**dynamic = false 的时候  会存进去，但是我试了一次，不管 1 还是  "1"都可以存进去，但是 也可以查看得到，但是好像搜索不到。**

**57 document 写入原理**

**每次写请求写入到 内存 buffer ，当 写到一定程度的时候，刷新，buffer 写到 lucene 的 segment，大概1 秒一次。segment 会吧数据写到 oscache ，然后执行 fsysc  命令吧  欧式chache 写到disk中。删除的时候是加删除，在index segment 中创建一个.del文件,在一定时候index segment  合并的时候，会删除这个del文件。更新，先执行删除，然后在执行插入。值得注意的是，没个一秒 buffer 提交一次，并且产生一个新的 segment，而且，这时候会出发 segment 到 oscache 的提交。数据提交到 os cache 的 以后就可以搜索到了，所以这就是 1 秒 近实时的原因。可以给index指定刷新的时间。  refresh\_interval，并且 es 还会写 tranlog（写buffer的同时）文件，这个文件可以避免丢失。每次提交会创建一个tranlog 文件，提交完成会删除原来的 tranlog 文件。在提交以后记录会写到新的 tranlog 中。**

**58. 查看 一段文本是在某个分词器上是怎么分词的。**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6** | **GET /user/\_analyze**  **{**  **"analyzer": "standard",**  **"text":  " a dog is in the house"**    **}** |

**59 : 给指定 字段指定指定的分词器**

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **put /user3/\_mapping/student**  **{**  **"properties":{**  **"name":{**  **"type":"text",**  **"analyzer":"standard"**  **}**  **}**    **}** |

**60: dynamic 策略 三种 ，true （遇到陌生字段就 dynamic mapping ），false（遇到陌生字段就忽略） ，strict（遇到陌生字段就报错）**

**61：post /my\_index/\_optimize?max\_num\_segments=1**

**手动使  索引 merge**

***能耍的时候就一定要耍,不能耍的时候一定要学。***