placeholder6

数据建模

Elasticsearch 是如此与 不同，特 是如果 来自 SQL 的世界。 Elasticsearch 有非常多的 点：高性能、可 展、近 搜索，并支持大数据量的数据分析。一切都很容易！ 只需下 并始使用它。

但它不是魔法。 了充分利用 Elasticsearch， 需要了解它的工作机制，以及如何 它如 所需的 行工作。

和 用的 系型数据存 有所不同，Elasticsearch 并没有 理 体之 的 系 出直接的方法。 一个 系数据 的黄金法 是 -- 化 的数据（ 式）-- 但 不 用于 Elasticsearch。 在 系 理 、嵌套 象 和 父-子 系文 我 了 些提供的方法的 点和 点。

然后在 容 我 Elasticsearch 提供的快速、 活的 容能力。 当然 容并没有一个放之四海而皆准的方案。 需要考 些通 系 生的数据流的具体特点， 据此 的模型。例如日志事件或者社交 流 些 序列数据 型，和静 文 集合在 理模型上有着很大的不同。

最后，我 聊一下 Elasticsearch 里面不能伸 的一件事。

系 理

世界有很多重要的 系：博客帖子有一些 ， 行 有多次交易 ，客 有多个 行 ，有多个 明 ，文件目 有多个文件和子目 。

系型数据 被明 — 不意外—用来 行 系管理：

- 个 体（或 行，在 系世界中）可以被 主 唯一 。
- 体 化 （ 式）。唯一 体的数据只存 一次，而相 体只存 它的主 。只能在一个具体位置修改 个 体的数据。
- 体可以 行 ，可以跨 体搜索。
- 个 体的 化是 原子的 ， 一致的 ， 隔 的 ， 和 持久的 。 （可以在 *ACID Transactions* 中看更多 。）
- 大多数 系数据 支持跨多个 体的 ACID 事 。

但是 系型数据 有其局限性，包括 全文 索有限的支持能力。 体 消耗是很昂 的， 的越多，消耗就越昂 。特 是跨服 器 行 体 成本 其昂 ，基本不可用。 但 个的服器上又存在数据量的限制。

Elasticsearch ，和大多数 NoSQL 数据 似，是扁平化的。索引是独立文 的集合体。 文 是否匹配搜索 求取决于它是否包含所有的所需信息。

Elasticsearch 中 个文 的数据 更是 ACIDic 的， 而 及多个文 的事 不是。当一个事 部分失 ，无法回 索引数据到前一个状 。

扁平化有以下 ：

- 索引 程是快速和无 的。

- 搜索　程是快速和无　的。
- 因　　个文　相互都是独立的，大　模数据可以在多个　点上　行分布。

但　　系　然非常重要。某些　候，我　需要　小扁平化和　　世界　系模型的差　。以下四　常用的方法，用来在 Elasticsearch 中　行　系型数据的管理：

- Application-side joins
- Data denormalization
- Nested objects
- Parent/child relationships

通常都需要　合其中的某几个方法来得到最　的解决方案。

用　　接

我　通　在我　的　用程序中　　接可以（部分）模　　系数据　。　例如，比方　我　正在　用　和他　的博客文章　行索引。在　系世界中，我　会　　来操作：

```
PUT /my_index/user/1 ①
{
  "name":     "John Smith",
  "email":    "john@smith.com",
  "dob":      "1970/10/24"
}

PUT /my_index/blogpost/2 ①
{
  "title":    "Relationships",
  "body":     "It's complicated...",
  "user":     1 ②
}
```

① 　个文　的 index, type, 和 id 一起　造成主　。

② blogpost 通　用　的 id　接到用　。index 和 type 并不需要因　在我　的　用程序中已　硬　　。

通　用　的 ID 1 可以很容易的　到博客帖子。

```
GET /my_index/blogpost/_search
{
  "query": {
    "filtered": {
      "filter": {
        "term": { "user": 1 }
      }
    }
  }
}
```

了 到用 叫做 John 的博客帖子，我 需要 行 次 ： 第一次会 所有叫做 John 的用 从而取他 的 ID 集合，接着第二次会将 些 ID 集合放到 似于前面一个例子的 ：

```
GET /my_index/user/_search
{
  "query": {
    "match": {
      "name": "John"
    }
  }
}

GET /my_index/blogpost/_search
{
  "query": {
    "filtered": {
      "filter": {
        "terms": { "user": [1] } ①
      }
    }
  }
}
```

① 行第一个 得到的 果将填充到 `terms` 器中。

用 接的主要 点是可以 数据 行 准化 理。只能在 `user` 文 中修改用 的名称。 点是，了在搜索 接文 ，必 行 外的 。

在 个例子中，只有一个用 匹配我 的第一个 ，但在 世界中，我 可以很 易的遇到数以百万的叫做 John 的用 。包含所有 些用 的 IDs 会 生一个非常大的 ， 是一个数百万 的 。

方法 用于第一个 体（例如，在 个例子中 `user` ）只有少量的文 的情况，并且最好它很少改 。 将允 用程序 果 行 存，并避免 常 行第一次 。

非 化 的数据

使用 Elasticsearch 得到最好的搜索性能的方法是有目的的通 在索引 行非 化 denormalizing

。　个文　保持一定数量的冗余副本可以在需要　避免　行　。

如果我　希望能　通　某个用　姓名　到他写的博客文章，可以在博客文　中包含　个用　的姓名：

```
PUT /my_index/user/1
{
  "name":     "John Smith",
  "email":    "john@smith.com",
  "dob":      "1970/10/24"
}

PUT /my_index/blogpost/2
{
  "title":    "Relationships",
  "body":     "It's complicated...",
  "user":     {
    "id":       1,
    "name":     "John Smith" ①
  }
}
```

① 部分用　的字段数据已被冗余到 blogpost 文　中。

在，我　通　次　就能　通　relationships　到用　John 的博客文章。

```
GET /my_index/blogpost/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "title":     "relationships" }},
        { "match": { "user.name": "John"          }}
      ]
    }
  }
}
```

数据非　化的　点是速度快。因　个文　都包含了所需的所有信息，当　些信息需要在　行匹配，并不需要　行昂　的　接操作。

字段折

一个普遍的需求是需要通　特定字段　行分　。例如我　需要按照用　名称　分　返回最相　的博客文章。按照用　名分　意味着　行　terms　聚合。　能　按照用　整体　名称　行分　，名称字段　保持 not_analyzed 的形式，具体　明参考 [aggregations-and-analysis]：

```
PUT /my_index/_mapping/blogpost
{
  "properties": {
    "user": {
      "properties": {
        "name": { ①
          "type": "string",
          "fields": {
            "raw": { ②
              "type":  "string",
              "index": "not_analyzed"
            }
          }
        }
      }
    }
  }
}
```

① `user.name` 字段将用来 行全文 索。

② `user.name.raw` 字段将用来通 `terms` 聚合 行分 。

然后添加一些数据:

```
PUT /my_index/user/1
{
  "name": "John Smith",
  "email": "john@smith.com",
  "dob": "1970/10/24"
}

PUT /my_index/blogpost/2
{
  "title": "Relationships",
  "body": "It's complicated...",
  "user": {
    "id": 1,
    "name": "John Smith"
  }
}

PUT /my_index/user/3
{
  "name": "Alice John",
  "email": "alice@john.com",
  "dob": "1979/01/04"
}

PUT /my_index/blogpost/4
{
  "title": "Relationships are cool",
  "body": "It's not complicated at all...",
  "user": {
    "id": 3,
    "name": "Alice John"
  }
}
```

在我 来 包含 relationships 并且作者名包含 John 的博客, 果再按作者名分 , 感{ref}/search-aggregations-metrics-top-hits-aggregation.html[top_hits aggregation] 提供了按照用 行分 的功能：

```
GET /my_index/blogpost/_search
{
  "size" : 0, ①
  "query": { ②
    "bool": {
      "must": [
        { "match": { "title":     "relationships" }},
        { "match": { "user.name": "John"          }}
      ]
    }
  },
  "aggs": {
    "users": {
      "terms": {
        "field":   "user.name.raw",      ③
        "order": { "top_score": "desc" } ④
      },
      "aggs": {
        "top_score": { "max":      { "script":  "_score"           }}, ④
        "blogposts": { "top_hits": { "_source": "title", "size": 5 }}  ⑤
      }
    }
  }
}
```

① 我 感 趣的博客文章是通 blogposts 聚合返回的，所以我 可以通 将 size 置成 0 来禁止 hits 常 搜索。

② query 返回通 relationships 名称 John 的用 的博客文章。

③ terms 聚合 一个 user.name.raw 建一个桶。

④ top_score 聚合 通 users 聚合得到的 一个桶按照文 分 行排序。

⑤ top_hits 聚合 个用 返回五个最相 的博客文章的 title 字段。

里 示 短 果：

```
...
"hits": {
  "total":     2,
  "max_score": 0,
  "hits":      [] ①
},
"aggregations": {
  "users": {
     "buckets": [
        {
           "key":       "John Smith", ②
           "doc_count": 1,
           "blogposts": {
              "hits": { ③
                 "total":     1,
                 "max_score": 0.35258877,
                 "hits": [
                    {
                       "_index": "my_index",
                       "_type":  "blogpost",
                       "_id":    "2",
                       "_score": 0.35258877,
                       "_source": {
                          "title": "Relationships"
                       }
                    }
                 ]
              }
           },
           "top_score": { ④
              "value": 0.3525887727737427
           }
        },
...
```

① 因 我 置 size 0，所以 hits 数 是空的。

② 在 果中出 的 一个用 都会有一个 的桶。

③ 在 个用 桶下面都会有一个 blogposts.hits 数 包含 个用 的 果。

④ 用 桶按照 个用 最相 的博客文章 行排序。

使用 top_hits 聚合等效 行一个 返回 些用 的名字和他 最相 的博客文章，然后 一个用 行相同的 ，以 得最好的博客。但前者的效率要好很多。

一个桶返回的 命中 果是基于最初主 行的一个 量 迷 果集。 个迷 提供了一些 期望的常用特性，例如高亮 示以及分 功能。

非　　化和并

当然，数据非　　化也有弊端。　　第一个　点是索引会更大因　　个博客文章文　的　　`_source` 将会更大，并且　里有很多的索引字段。　通常不是一个大　　。数据写到磁　将会被高度　　，而且磁　已　很廉　了。Elasticsearch 可以愉快地　付　些　外的数据。

更重要的　　是，如果用　改　了他的名字，他所有的博客文章也需要更新了。幸　的是，用　不　常更改名称。即使他　做了，　用　也不可能写超　几千篇博客文章，所以更新博客文章通　`scroll` 和 `bulk` APIs 大概耗　不到一秒。

然而，　我　考　一个更　　的　景，其中的　化很常　，影　深　，而且非常重要，并　。

在　个例子中，我　将在 Elasticsearch 模　一个文件系　的目　　，非常　似 Linux 文件系　：根目　是 `/`，　个目　可以包含文件和子目　。

我　希望能　搜索到一个特定目　下的文件，等效于：

```
grep "some text" /clinton/projects/elasticsearch/*
```

就要求我　索引文件所在目　的路径：

```
PUT /fs/file/1
{
  "name":     "README.txt", ①
  "path":     "/clinton/projects/elasticsearch", ②
  "contents": "Starting a new Elasticsearch project is easy..."
}
```

① 文件名

② 文件所在目　的全路径

NOTE	事　上，我　也　当索引 `directory`　文　，如此我　可以在目　内列出所有的文件和子目　，但　了　　，我　将忽略　个需求。

我　也希望能　搜索到一个特定目　下的目　　包含的的任何文件，相当于此：

```
grep -r "some text" /clinton
```

了支持　一点，我　需要　路径　次　　行索引：

- `/clinton`
- `/clinton/projects`
- `/clinton/projects/elasticsearch`

　　次　　能　通　　`path`　字段使用　{ref}/analysis-pathhierarchy-tokenizer.html[`path_hierarchy` tokenizer] 自　生成：

```
PUT /fs
{
  "settings": {
    "analysis": {
      "analyzer": {
        "paths": { ①
          "tokenizer": "path_hierarchy"
        }
      }
    }
  }
}
```

① 自定 的 paths 分析器在 置中使用 {ref}/analysis-pathhierarchy-tokenizer.html[path_hierarchy tokenizer]。

file 型的映射看起来如下所示：

```
PUT /fs/_mapping/file
{
  "properties": {
    "name": { ①
      "type":  "string",
      "index": "not_analyzed"
    },
    "path": { ②
      "type":  "string",
      "index": "not_analyzed",
      "fields": {
        "tree": { ②
          "type":     "string",
          "analyzer": "paths"
        }
      }
    }
  }
}
```

① name 字段将包含 切名称。

② path 字段将包含 切的目 名称，而 path.tree 字段将包含路径 次 。

一旦索引建立并且文件已被 入索引，我 可以 行一个搜索，在 /clinton/projects/elasticsearch 目中包含 elasticsearch 的文件，如下所示：

```
GET /fs/file/_search
{
  "query": {
    "filtered": {
      "query": {
        "match": {
          "contents": "elasticsearch"
        }
      },
      "filter": {
        "term": { ①
          "path": "/clinton/projects/elasticsearch"
        }
      }
    }
  }
}
```

① 在 目 中 文件。

所有在 /clinton 下面的任何子目 存放的文件将在 path.tree 字段中包含 /clinton 。所以我 能 搜索 /clinton 的任何子目 中的所有文件，如下所示：

```
GET /fs/file/_search
{
  "query": {
    "filtered": {
      "query": {
        "match": {
          "contents": "elasticsearch"
        }
      },
      "filter": {
        "term": { ①
          "path.tree": "/clinton"
        }
      }
    }
  }
}
```

① 在 个目 或其下任何子目 中 文件。

重命名文件和目

到目前 止一切 利。 重命名一个文件很容易—所需要的只是一个 的 update 或 index 求。 甚至可以使用 optimistic concurrency control 保 的 化不会与其他用 的 化 生冲突：

```
PUT /fs/file/1?version=2 ①
{
  "name":     "README.asciidoc",
  "path":     "/clinton/projects/elasticsearch",
  "contents": "Starting a new Elasticsearch project is easy..."
}
```

① version 号 保 更改 用于 索引中具有此相同的版本号的文 。

我 甚至可以重命名一个目 ，但 意味着更新所有存在于 目 下路径 次 中的所有文件。 可能快速或 慢，取决于有多少文件需要更新。我 所需要做的就是使用 scroll 来 索所有的文件，以及 bulk API 来更新它 。 个 程不是原子的，但是所有的文件将会迅速 移到他 的新存放位置。

解决并

当我 允 多个人 同 重命名文件或目 ， 就来了。 想一下， 正在 一个包含了成百上千文件的目 /clinton 行重命名操作。 同 ， 一个用 个目 下的 个文件 /clinton/projects/elasticsearch/README.txt 行重命名操作。 个用 的修改操作，尽管在 的操作后 始，但可能会更快的完成。

以下有 情况可能出 ：

- 决定使用 version （版本）号，在 情况下，当与 README.txt 文件重命名的版本号 生冲突 ，的批量重命名操作将会失 。
- 没有使用版本控制， 的 更将覆 其他用 的 更。

 的原因是 Elasticsearch 不支持 ACID 事 。 个文件的 更是 ACIDic 的，但包含多个文 的更不支持。

如果 的主要数据存 是 系数据 ，并且 Elasticsearch 作 一个搜索引 或一 提升性能的方法，可以首先在数据 中 行 更 作，然后在完成后将 些 更 制到 Elasticsearch。通 方式， 将受益于数据 ACID 事 支持，并且在 Elasticsearch 中以正 的 序 生 更。 并在 系数据 中得到了 理。

如果 不使用 系型存 ， 些并 就需要在 Elasticsearch 的事 水准 行 理。 以下是三个切 可行的使用 Elasticsearch 的解决方案，它 都 及某 形式的 ：

- 全局
- 文
-

TIP	当使用一个外部系 替代 Elasticsearch ，本 中所描述的解决方案可以通 相同的原来 。

全局

通 在任何 只允 一个 程来 行 更 作，我 可以完全避免并 。 大多数的 更只

及少量文件，会很快完成。一个　　目　的重命名操作会　其他　更造成　　　的阻塞，但可能很少做。

因　在 Elasticsearch　文　　的　更支持 ACIDic，我　可以使用一个文　是否存在的状　作　一个全局　。　了　求得到　，我　　create 全局　文　：

```
PUT /fs/lock/global/_create
{}
```

如果　个　 create　　求因冲突　常而失　，　明　一个　程已被授予全局　，我　将不得不　后再　。如果　求成功了，我　自豪的成　全局　的主人，然后可以　　完成我　的　更。一旦完成，我　就必　通　　除全局　文　来　放　：

```
DELETE /fs/lock/global
```

根据　更的　繁程度以及　　消耗，一个全局　能　系　造成大幅度的性能限制。　　我　可以通　　我　的　更　粒度的方式来　加并行度。

文

我　可以使用前面描述相同的方法技　来　定个体文　，而不是　定整个文件系　。我　可以使用 scrolled search　索所有的文　，　些文　会被　更影　因此　一个文　都　建了一个　文件：

```
PUT /fs/lock/_bulk
{ "create": { "_id": 1}} ①
{ "process_id": 123    } ②
{ "create": { "_id": 2}}
{ "process_id": 123    }
```

① lock 文　的 ID 将与　被　定的文件的 ID 相同。

② process_id 代表要　行　更　程的唯一 ID。

如果一些文件已被　定，部分的 bulk　求将失　，我　将不得不再次　　。

当然，如果我　　再次　定　所有　的文件，　　我　前面使用的　 create　　句将会失　，因所有文件都已被我　　定！　我　需要一个　update　　求　　upsert　参数以及下面　个　 script，而不是一个　　的 create　句：

```
if ( ctx._source.process_id != process_id ) { ①
  assert false;  ②
}
ctx.op = 'noop'; ③
```

① process_id 是　　到脚本的一个参数。

② assert false 将引　　常，　致更新失　。

③ 将 op 从 update 更新到 noop 防止更新 求作出任何改 ，但 返回成功。

完整的 update 求如下所示：

```
POST /fs/lock/1/_update
{
  "upsert": { "process_id": 123 },
  "script": "if ( ctx._source.process_id != process_id )
  { assert false }; ctx.op = 'noop';"
  "params": {
    "process_id": 123
  }
}
```

如果文 并不存在， upsert 文 将会被 入—和前面 create 求相同。 但是，如果 文件 存在， 脚本会 看存 在文 上的 process_id 。 如果 process_id 匹配，更新不会 行（ noop ）但脚本会返回成功。 如果 者并不匹配， assert false 出一个 常， 也知道了 取 的 已失 。

一旦所有 已成功 建， 就可以 行 的 更。

之后， 必 放所有的 ，通 索所有的 文 并 行批量 除，可以完成 的 放：

```
POST /fs/_refresh ①

GET /fs/lock/_search?scroll=1m ②
{
    "sort" : ["_doc"],
    "query": {
        "match" : {
            "process_id" : 123
        }
    }
}

PUT /fs/lock/_bulk
{ "delete": { "_id": 1}}
{ "delete": { "_id": 2}}
```

① refresh 用 保所有 lock 文 搜索 求可 。

② 当 需要在 次搜索 求返回大量的 索 果集 ， 可以使用 scroll 。

文 可以 粒度的 控制，但是 数百万文 建 文件 也很大。 在某些情况下，可以用少得多的工作量 粒度的 定，如以下目 景中所示。

在前面的例子中，我 可以 定的目 的一部分，而不是 定 一个 及的文 。 我 将需要独占

我 要重命名的文件或目 ，它可以通 独占 文 来 ：

```
{ "lock_type": "exclusive" }
```

同 我 需要共享 定所有的父目 ，通 共享 文 ：

```
{
  "lock_type":  "shared",
  "lock_count": 1 ①
}
```

① lock_count 持有共享 程的数量。

/clinton/projects/elasticsearch/README.txt 行重命名的 程需要在 个文件上有 独占 ，以及在 /clinton 、 /clinton/projects 和 /clinton/projects/elasticsearch 目 有 共享 。

一个 的 create 求将 足独占 的要求，但共享 需要脚本的更新来 一些 外的 ：

```
if (ctx._source.lock_type == 'exclusive') {
  assert false; ①
}
ctx._source.lock_count++ ②
```

① 如果 lock_type 是 exclusive （独占）的，assert 句将 出一个 常， 致更新 求失 。

② 否 ，我 lock_count 行 量 理。

个脚本 理了 lock 文 已 存在的情况，但我 需要一个用来 理的文 不存在情况的 upsert 文 。完整的更新 求如下：

```
POST /fs/lock/%2Fclinton/_update ①
{
  "upsert": { ②
    "lock_type":  "shared",
    "lock_count": 1
  },
  "script": "if (ctx._source.lock_type == 'exclusive')
  { assert false }; ctx._source.lock_count++"
}
```

① 文 的 ID 是 /clinton ， URL 后成 %2fclinton 。

② upsert 文 如果不存在， 会被 入。

一旦我 成功地在所有的父目 中 得一个共享 ，我 在文件本身 create 一个独占 ：

```
PUT /fs/lock/%2Fclinton%2fprojects%2felasticsearch%2fREADME.txt/_create
{ "lock_type": "exclusive" }
```

在，如果有其他人想要重新命名 /clinton 目 ，他 将不得不在 条路径上 得一个独占 ：

```
PUT /fs/lock/%2Fclinton/_create
{ "lock_type": "exclusive" }
```

个 求将失 ，因 一个具有相同 ID 的 lock 文 已 存在。 一个用 将不得不等待我的操作完成以及 放我 的 。独占 只能 被 除：

```
DELETE /fs/lock/%2Fclinton%2fprojects%2felasticsearch%2fREADME.txt
```

共享 需要 一个脚本 lock_count ，如果 数下降到零， 除 lock 文 ：

```
if (--ctx._source.lock_count == 0) {
  ctx.op = 'delete' ①
}
```

① 一旦 lock_count 到0， ctx.op 会从 update 被修改成 delete 。

此更新 求将 父目 由下至上的 行，从最 路径到最短路径：

```
POST /fs/lock/%2Fclinton%2fprojects%2felasticsearch/_update
{
  "script": "if (--ctx._source.lock_count == 0) { ctx.op = 'delete' } "
}
```

用最小的代 提供了 粒度的并 控制。当然，它不 用于所有的情况—数据模型必 有 似于目的 序 路径才能使用。

NOTE

三个方案—全局、文 或 —都没有 理 最棘手的 ：如果持有 的 程死了 ？

一个 程的意外死亡 我 留下了2个 ：

- 我 如何知道我 可以 放的死亡 程中所持有的 ？
- 我 如何清理死去的 程没有完成的 更？

些主 超出了本 的 ，但是如果 决定使用 ， 需要 他 行一些思考。

当非 化成 很多 目的一个很好的 ，采用 方案的需求会 来 的 。 作替代方案，Elasticsearch 提供 个模型 助我 理相 的 体：嵌套的 象 和 父子 系 。

嵌套　象

由于在　　Elasticsearch　　中　个文　的　　改都是原子性操作,那　将相　　体数据都存　在同一文　中也就理所当然。　　比如　,我　可以将　　及其明　数据存　在一个文　中。又比如,我　可以将一篇博客文章的　　以一个 comments 数　的形式和博客文章放在一起：

```
PUT /my_index/blogpost/1
{
  "title": "Nest eggs",
  "body":  "Making your money work...",
  "tags":  [ "cash", "shares" ],
  "comments": [ ①
    {
      "name":    "John Smith",
      "comment": "Great article",
      "age":     28,
      "stars":   4,
      "date":    "2014-09-01"
    },
    {
      "name":    "Alice White",
      "comment": "More like this please",
      "age":     31,
      "stars":   5,
      "date":    "2014-10-22"
    }
  ]
}
```

① 如果我　依　字段自　映射,那　comments 字段会自　映射　object　型。

由于所有的信息都在一个文　中,当我　　就没有必要去　合文章和　　文　,　　效率就很高。

但是当我　使用如下　　,上面的文　也会被当做是符合条件的　果：

```
GET /_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "name": "Alice" }},
        { "match": { "age":  28      }} ①
      ]
    }
  }
}
```

① Alice　　是31　,不是28!

正如我 在 象数 中 的一 ,出 上面 的原因是 JSON 格式的文 被理成如下的扁平式 的 。

```
{
  "title":            [ eggs, nest ],
  "body":             [ making, money, work, your ],
  "tags":             [ cash, shares ],
  "comments.name":    [ alice, john, smith, white ],
  "comments.comment": [ article, great, like, more, please, this ],
  "comments.age":     [ 28, 31 ],
  "comments.stars":   [ 4, 5 ],
  "comments.date":    [ 2014-09-01, 2014-10-22 ]
}
```

Alice 和 31 、 John 和 2014-09-01 之 的相 性信息不再存在。 然 object 型 (参 内部 象) 在存 一 象 非常有用,但 于 象数 的搜索而言, 无用 。

嵌套 象 就是来解决 个 的。将 comments 字段 型 置 nested 而不是 object 后, 一个嵌套 象都会被索引 一个 藏的独立文 , 例如下:

```
{ ①
  "comments.name":    [ john, smith ],
  "comments.comment": [ article, great ],
  "comments.age":     [ 28 ],
  "comments.stars":   [ 4 ],
  "comments.date":    [ 2014-09-01 ]
}
{ ②
  "comments.name":    [ alice, white ],
  "comments.comment": [ like, more, please, this ],
  "comments.age":     [ 31 ],
  "comments.stars":   [ 5 ],
  "comments.date":    [ 2014-10-22 ]
}
{ ③
  "title":            [ eggs, nest ],
  "body":             [ making, money, work, your ],
  "tags":             [ cash, shares ]
}
```

① 第一个 嵌套文

② 第二个 嵌套文

③ 根文 或者也可称 父文

在独立索引 一个嵌套 象后, 象中 个字段的相 性得以保留。我 ,也 返回那些真正符合条件的文 。

不 如此,由于嵌套文 直接存 在文 内部, 嵌套文 和根文 合成本很低,速度和 独存 几乎一

。

嵌套文 是 藏存 的,我 不能直接 取。如果要 改一个嵌套 象,我 必 把整个文 重新索引才可以 。 得注意的是, 的 候返回的是整个文 ,而不是嵌套文 本身。

嵌套 象映射

置一个字段 nested 很 — 只需要将字段 型 object 替 nested 即可：

```
PUT /my_index
{
  "mappings": {
    "blogpost": {
      "properties": {
        "comments": {
          "type": "nested", ①
          "properties": {
            "name":    { "type": "string"  },
            "comment": { "type": "string"  },
            "age":     { "type": "short"   },
            "stars":   { "type": "short"   },
            "date":    { "type": "date"    }
          }
        }
      }
    }
  }
}
```

① nested 字段 型的 置参数与 object 相同。

就是需要 置的一切。至此，所有 comments 象会被索引在独立的嵌套文 中。可以 看 {ref}/nested.html[nested 型参考文] 取更多 信息。

嵌套 象

由于嵌套 象 被索引在独立 藏的文 中，我 无法直接 它 。 相 地，我 必 使用 {ref}/query-dsl-nested-query.html[nested] 去 取它 ：

```
GET /my_index/blogpost/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "title": "eggs" ①
          }
        },
        {
          "nested": {
            "path": "comments", ②
            "query": {
              "bool": {
                "must": [ ③
                  {
                    "match": {
                      "comments.name": "john"
                    }
                  },
                  {
                    "match": {
                      "comments.age": 28
                    }
                  }
                ]
              }
            }
          }
        }
      ]
}}}
```

① title 子句是 根文 的。

② nested 子句作用于嵌套字段 comments 。在此 中，既不能 根文 字段，也不能 其他嵌套文 。

③ comments.name 和 comments.age 子句操作在同一个嵌套文 中。

TIP

nested 字段可以包含其他的 nested 字段。同 地，nested 也可以包含其他的 nested 。而嵌套的 次会按照 所期待的被 用。

nested 肯定可以匹配到多个嵌套的文 。 一个匹配的嵌套文 都有自己的相 度得分，但是 多的分数最 需要 聚 可供根文 使用的一个分数。

情况下，根文 的分数是 些嵌套文 分数的平均 。可以通 置 score_mode 参数来控制 个得分策略，相 策略有 avg (平均), max (最大), sum (加和) 和 none (直接返回 1.0 常数 分数)。

```
GET /my_index/blogpost/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "title": "eggs"
          }
        },
        {
          "nested": {
            "path": "comments",
            "score_mode": "max", ①
            "query": {
              "bool": {
                "must": [
                  {
                    "match": {
                      "comments.name": "john"
                    }
                  },
                  {
                    "match": {
                      "comments.age": 28
                    }
                  }
                ]
              }
            }
          }
        }
      ]
    }
  }
}
```

① 返回最　匹配嵌套文　的 _score　根文　使用。

NOTE

如果 nested　放在一个布　的 filter 子句中，其表　就像一个 nested　，只是 score_mode　参数不再生效。因　它被用于不打分的　中 — 只是符合或不符合条件，不必打分 — 那　score_mode　就没有任何意　，因　根本就没有要打分的地方。

使用嵌套字段排序

尽管嵌套字段的　存　于独立的嵌套文　中，但依然有方法按照嵌套字段的　排序。　我　添加　一个　，以使得　果更有意思：

```
PUT /my_index/blogpost/2
{
  "title": "Investment secrets",
  "body":  "What they don't tell you ...",
  "tags":  [ "shares", "equities" ],
  "comments": [
    {
      "name":    "Mary Brown",
      "comment": "Lies, lies, lies",
      "age":     42,
      "stars":   1,
      "date":    "2014-10-18"
    },
    {
      "name":    "John Smith",
      "comment": "You're making it up!",
      "age":     28,
      "stars":   2,
      "date":    "2014-10-16"
    }
  ]
}
```

假如我 想要 在10月 收到 的博客文章，并且按照 `stars` 数的最小 来由小到大排序，那 句如下：

```
GET /_search
{
  "query": {
    "nested": { ①
      "path": "comments",
      "filter": {
        "range": {
          "comments.date": {
            "gte": "2014-10-01",
            "lt":  "2014-11-01"
          }
        }
      }
    }
  },
  "sort": {
    "comments.stars": { ②
      "order": "asc",   ②
      "mode":  "min",   ②
      "nested_path": "comments", ③
      "nested_filter": {
        "range": {
          "comments.date": {
            "gte": "2014-10-01",
            "lt":  "2014-11-01"
          }
        }
      }
    }
  }
}
```

① 此 的 nested 将 果限定 在10月 收到 的博客文章。

② 果按照匹配的 中 comment.stars 字段的最小 (min) 来由小到大 (asc) 排序。

③ 排序子句中的 nested_path 和 nested_filter 和 query 子句中的 nested 相同，原因在下面有解 。

我 什 要用 nested_path 和 nested_filter 重 条件 ？原因在于，排序 生在 行之后。 条件限定了只在10月 收到 的博客文 ，但返回整个博客文 。如果我 不在排序子句中加入 nested_filter ， 那 我 博客文 的排序将基于博客文 的所有 ，而不是 在10月接收到的 。

嵌套聚合

在 的 候，我 使用 nested 就可以 取嵌套 象的信息。同理， nested 聚合允 我 嵌套 象里的字段 行聚合操作。

```
GET /my_index/blogpost/_search
{
  "size" : 0,
  "aggs": {
    "comments": { ①
      "nested": {
        "path": "comments"
      },
      "aggs": {
        "by_month": {
          "date_histogram": { ②
            "field":    "comments.date",
            "interval": "month",
            "format":   "yyyy-MM"
          },
          "aggs": {
            "avg_stars": {
              "avg": { ③
                "field": "comments.stars"
              }
            }
          }
        }
      }
    }
  }
}
```

① nested 聚合 ` 入'' 嵌套的 `comments 象。

② comment 象根据 comments.date 字段的月 被分到不同的桶。

③ 算 个桶内star的平均数量。

从下面的 果可以看出聚合是在嵌套文 面 行的：

```
...
"aggregations": {
  "comments": {
     "doc_count": 4, ①
     "by_month": {
        "buckets": [
           {
              "key_as_string": "2014-09",
              "key": 1409529600000,
              "doc_count": 1, ①
              "avg_stars": {
                 "value": 4
              }
           },
           {
              "key_as_string": "2014-10",
              "key": 1412121600000,
              "doc_count": 3, ①
              "avg_stars": {
                 "value": 2.6666666666666665
              }
           }
        ]
     }
  }
}
...
```

① 共有4个 comments 象 ：1个 象在9月的桶里，3个 象在10月的桶里。

逆向嵌套聚合

nested 聚合 只能 嵌套文 的字段 行操作。 根文 或者其他嵌套文 的字段 它是不可 的。然而，通 reverse_nested 聚合，我 可以 走出 嵌套 ，回到父 文 行操作。

例如，我 要基于 者的年 出 者感 趣 tags 的分布。 comment.age 是一个嵌套字段，但 tags 在根文 中：

```
GET /my_index/blogpost/_search
{
  "size" : 0,
  "aggs": {
    "comments": {
      "nested": { ①
        "path": "comments"
      },
      "aggs": {
        "age_group": {
          "histogram": { ②
            "field":    "comments.age",
            "interval": 10
          },
          "aggs": {
            "blogposts": {
              "reverse_nested": {}, ③
              "aggs": {
                "tags": {
                  "terms": { ④
                    "field": "tags"
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

① nested 聚合　入 comments　象。

② histogram 聚合基于 comments.age 做分　，　10年一个分　。

③ reverse_nested 聚合退回根文　。

④ terms 聚合　算　个分　年　段的　　者最常用的　　。

略　果如下所示：

```
..
"aggregations": {
  "comments": {
     "doc_count": 4, ①
     "age_group": {
        "buckets": [
           {
              "key": 20, ②
              "doc_count": 2, ②
              "blogposts": {
                 "doc_count": 2, ③
                 "tags": {
                    "doc_count_error_upper_bound": 0,
                    "buckets": [ ④
                       { "key": "shares",   "doc_count": 2 },
                       { "key": "cash",     "doc_count": 1 },
                       { "key": "equities", "doc_count": 1 }
                    ]
                 }
              }
           },
...
```

① 一共有4条　　。

② 在20　到30　之　　共有　条　　。

③　　些　　包含在　篇博客文章中。

④ 在　些博客文章中最　　的　　是 shares、 cash、equities。

嵌套　象的使用　机

嵌套　象　　在只有一个主要　体　非常有用，　个主要　体包含有限个　密　　但又不是很重要的　体，例如我　的　　blogpost　　象包含　　象。　　在基于　　的内容　　博客文章　，　　nested　　有很大的用　，并且可以提供更快的　　效率。

嵌套模型的　点如下：

- 当　嵌套文　做　加、修改或者　除　，整个文　都要重新被索引。嵌套文　越多，　　来的成本就越大。
- 　　果返回的是整个文　，而不　　是匹配的嵌套文　。尽管目前有　　支持只返回根文　中最佳匹配的嵌套文　，但目前　不支持。

有　　需要在主文　和其　　体之　做一个完整的隔　　。　个隔　是由 父子　　提供的。

父-子　系文

父-子　系文　　在　　上　似于　nested　model　　：允　将一个　象　体和　外一个　象　体　起来。而　　型的主要区　是：在 nested objects 文　中，所有　象都是在同一个文　中，而在父-子

系文 中，父 象和子 象都是完全独立的文 。

父-子 系的主要作用是允 把一个 type 的文 和 外一个 type 的文 起来， 成一 多的系：一个父文 可以 多个子文 。与 nested objects 相比，父-子 系的主要 有：

- 更新父文 ，不会重新索引子文 。
- 建，修改或 除子文 ，不会影 父文 或其他子文 。 一点在 景下尤其有用：子文 数量 多，并且子文 建和修改的 率高 。
- 子文 可以作 搜索 果独立返回。

Elasticsearch 了一个父文 和子文 的映射 系，得益于 个映射，父-子文 操作非常快。但是 个映射也 父-子文 系有个限制条件：父文 和其所有子文 ，都必要存 在同一个分片中。

父-子文 ID映射存 在 [docvalues] 中。当映射完全在内存中 ， [docvalues] 提供 映射的快速理能力， 一方面当映射非常大 ，可以通 溢出到磁 提供足 的 展能力

父-子 系文 映射

建立父-子文 映射 系 只需要指定某一个文 type 是 一个文 type 的父 。 系可以在如下个 点 置：1） 建索引 ；2）在子文 type 建之前更新父文 的 mapping。

例 明，有一个公司在多个城市有分公司，并且 一个分公司下面都有很多 工。有 的需求：按照分公司、 工的 度去搜索，并且把 工和他 工作的分公司 系起来。 需求，用嵌套模型是无法的。当然，如果使用 application-side-joins 或者 data denormalization 也是可以 的，但是 了演示的目的，在 里我 使用父-子文 。

我 需要告 Elasticsearch，在 建 工 employee 文 type ，指定分公司 branch 的文 type 其父 。

```
PUT /company
{
  "mappings": {
    "branch": {},
    "employee": {
      "_parent": {
        "type": "branch" ①
      }
    }
  }
}
```

① employee 文 是 branch 文 的子文 。

建父-子文 索引

父文 建索引与 普通文 建索引没有区 。父文 并不需要知道它有 些子文 。

```
POST /company/branch/_bulk
{ "index": { "_id": "london" }}
{ "name": "London Westminster", "city": "London", "country": "UK" }
{ "index": { "_id": "liverpool" }}
{ "name": "Liverpool Central", "city": "Liverpool", "country": "UK" }
{ "index": { "_id": "paris" }}
{ "name": "Champs Élysées", "city": "Paris", "country": "France" }
```

建子文　，用　必　要通　parent 参数来指定　子文　的父文　ID：

```
PUT /company/employee/1?parent=london ①
{
  "name":  "Alice Smith",
  "dob":   "1970-10-24",
  "hobby": "hiking"
}
```

① 当前 employee 文　的父文　ID 是 london 。

父文　　ID　　有　个作用：　建了父文　和子文　之　的　系，并且保　了父文　和子文　都在同一个分片上。

在　[routing-value]　中，我　解　了　Elasticsearch　如何通　路由　来决定　文　属于一个分片，路由　　文　的 _id 。分片路由的　算公式如下：

```
shard = hash(routing) % number_of_primary_shards
```

如果指定了父文　的　ID，那　就会使用父文　的　ID　行路由，而不会使用当前文　_id　。也就是　，如果父文　和子文　都使用相同的　行路由，那　父文　和子文　都会　定分布在同一个分片上。

在　行　文　的　求　需要指定父文　的　ID，　文　求包括：通　GET　求　取一个子文　；　建、更新或　除一个子文　。而　行搜索　求　是不需要指定父文　的ID，　是因　搜索　求是向一个索引中的所有分片　起　求，而　文　的操作是只会向存　文　的分片　送　求。因此，如果操作　个子文　不指定父文　的 ID，那　很有可能会把　求　送到　的分片上。

父文　的 ID　在 bulk API 中指定

```
POST /company/employee/_bulk
{ "index": { "_id": 2, "parent": "london" }}
{ "name": "Mark Thomas", "dob": "1982-05-16", "hobby": "diving" }
{ "index": { "_id": 3, "parent": "liverpool" }}
{ "name": "Barry Smith", "dob": "1979-04-01", "hobby": "hiking" }
{ "index": { "_id": 4, "parent": "paris" }}
{ "name": "Adrien Grand", "dob": "1987-05-11", "hobby": "horses" }
```

WARNING	如果 想要改 一个子文 的 parent , 通 更新 个子文 是不 的，因 新的父文 有可能在 外一个分片上。因此， 必 要先把子文 除，然后再重新索引 个子文 。

通 子文 父文

has_child 的 和 可以通 子文 的内容来 父文 。例如，我 根据如下 ，可 出所有80后 工所在的分公司：

```
GET /company/branch/_search
{
  "query": {
    "has_child": {
      "type": "employee",
      "query": {
        "range": {
          "dob": {
            "gte": "1980-01-01"
          }
        }
      }
    }
  }
}
```

似于 nested query ，has_child 可以匹配多个子文 ，并且 一个子文 的 分都不同。但是由于 一个子文 都 有 分， 些 分如何 成父文 的 得分取决于 score_mode 个参数。参数有多 取 策略： none ，会忽略子文 的 分，并且会 父文 分 置 1.0 ； 除此以外 可以 置成 avg 、 min 、 max 和 sum 。

下面的 将会同 返回 london 和 liverpool ，不 由于 Alice Smith 要比 Barry Smith 更加匹配 条件，因此 london 会得到一个更高的 分。

```
GET /company/branch/_search
{
  "query": {
    "has_child": {
      "type":       "employee",
      "score_mode": "max",
      "query": {
        "match": {
          "name": "Alice Smith"
        }
      }
    }
  }
}
```

TIP

score_mode 的 none ，会 著地比其模式要快， 是因 Elasticsearch不需要 算 一个子文 的 分。只有当 真正需要 心 分 果 ，才需要 source_mode ，例如 成 avg 、 min 、 max 或 sum 。

min_children 和 max_children

has_child 的 和 都可以接受 个参数：min_children 和 max_children 。 使用 个参数 ，只有当子文 数量在指定 内 ，才会返回父文 。

如下 只会返回至少有 个雇 的分公司：

```
GET /company/branch/_search
{
  "query": {
    "has_child": {
      "type":         "employee",
      "min_children": 2, ①
      "query": {
        "match_all": {}
      }
    }
  }
}
```

① 至少有 个雇 的分公司才会符合 条件。

有 min_children 和 max_children 参数的 has_child 或 ，和允 分的 has_child 的性能非常接近。

has_child Filter

has_child 和 在 行机制上 似，区 是 has_child 不支持 source_mode 参数。has_child 用于 内容—如内部的一个 filtered —和其他 行 似：包含或者排除，但没有 行 分。

has_child 的 果没有被 存，但是 has_child 内部的 方法 用于通常的 存 。

通 父文 子文

然 nested 只能返回最 的文 ，但是父文 和子文 本身是彼此独立并且可被 独 的。我使用 has_child 句可以基于子文 来 父文 ，使用 has_parent 句可以基于父文 来 子文 。

has_parent 和 has_child 非常相似，下面的 将会返回所有在 UK 工作的雇 ：

```
GET /company/employee/_search
{
  "query": {
    "has_parent": {
      "type": "branch", ①
      "query": {
        "match": {
          "country": "UK"
        }
      }
    }
  }
}
```

① 返回父文 type 是 branch 的所有子文

has_parent 也支持 score_mode 个参数，但是 参数只支持 : none （ ）和 score 。 个子文 都只有一个父文 ，因此 里不存在将多个 分 一个的情况， score_mode 的取 score 和 none 。

> **不 分的 has_parent**
>
> 当 has_parent 用于非 分模式（比如 filter 句） ， score_mode 参数就不再起作用了。因 模式只是 地包含或排除文 ，没有 分，那 score_mode 参数也就没有意 了。

子文 聚合

在父-子文 中支持 子文 聚合， 一点和 嵌套聚合 似。但是， 于父文 的聚合 是不支持的（和 reverse_nested 似）。

我 通 下面的例子来演示按照国家 度 看最受雇 迎的 余 好：

```
GET /company/branch/_search
{
  "size" : 0,
  "aggs": {
    "country": {
      "terms": { ①
        "field": "country"
      },
      "aggs": {
        "employees": {
          "children": { ②
            "type": "employee"
          },
          "aggs": {
            "hobby": {
              "terms": { ③
                "field": "hobby"
              }
            }
          }
        }
      }
    }
  }
}
```

① country 是 branch 文　的一个字段。

② 子文　聚合　　通　employee type 的子文　将其父文　聚合在一起。

③ hobby 是 employee 子文　的一个字段。

祖　与　　系

父子　系可以延展到更多代　系，比如生活中　　与祖　的　系　　—　　唯一的要求是　足　些系的文　必　在同一个分片上被索引。

　我　把上一个例子中的 country　型　定　branch　型的父　：

```
PUT /company
{
  "mappings": {
    "country": {},
    "branch": {
      "_parent": {
        "type": "country" ①
      }
    },
    "employee": {
      "_parent": {
        "type": "branch" ②
      }
    }
  }
}
```

① branch 是 country 的子 。

② employee 是 branch 的子 。

country 和 branch 之 是一 的父子 系，所以我 的 操作 与之前保持一致：

```
POST /company/country/_bulk
{ "index": { "_id": "uk" }}
{ "name": "UK" }
{ "index": { "_id": "france" }}
{ "name": "France" }

POST /company/branch/_bulk
{ "index": { "_id": "london", "parent": "uk" }}
{ "name": "London Westmintster" }
{ "index": { "_id": "liverpool", "parent": "uk" }}
{ "name": "Liverpool Central" }
{ "index": { "_id": "paris", "parent": "france" }}
{ "name": "Champs Élysées" }
```

parent ID 使得 一个 branch 文 被路由到与其父文 country 相同的分片上 行操作。然而，当我 使用相同的方法来操作 employee 个 文 ，会 生什 ？

```
PUT /company/employee/1?parent=london
{
  "name":  "Alice Smith",
  "dob":   "1970-10-24",
  "hobby": "hiking"
}
```

employee 文 的路由依 其父文 ID — 也就是 <code>london</code> — 但是

<code>london</code> 文 的路由却依 其本身的 父文 ID — 也就是 <code>uk</code> 。此 情况下， 文 很有可能最 和父 、祖 文 不在同一分片上， 致不足祖 和 文 必 在同一个分片上被索引的要求。

解决方案是添加一个 外的 routing 参数，将其 置 祖 的文 ID ，以此来保 三代文 路由到同一个分片上。索引 求如下所示：

```
PUT /company/employee/1?parent=london&routing=uk ①
{
  "name":  "Alice Smith",
  "dob":   "1970-10-24",
  "hobby": "hiking"
}
```

① routing 的 会取代 parent 的 作 路由 。

parent 参数的 然可以 employee 文 与其父文 的 系，但是 routing 参数保 文 被存到其父 和祖 的分片上。routing 在所有的文 求中都要添加。

合多代文 行 和聚合是可行的，只需要一代代的 行 定即可。例如，我 要 到喜 足的雇者的城市，此 需要 合 country 和 branch，以及 branch 和 employee：

```
GET /company/country/_search
{
  "query": {
    "has_child": {
      "type": "branch",
      "query": {
        "has_child": {
          "type": "employee",
          "query": {
            "match": {
              "hobby": "hiking"
            }
          }
        }
      }
    }
  }
}
```

使用中的一些建

当文 索引性能 比 性能重要的 候，父子 系是非常有用的，但是它也是有巨大代 的。其 速度会比同等的嵌套 慢5到10倍!

全局序号和延

父子 系使用了全局序数 来加速文 的 合。不管父子 系映射是否使用了内存 存或基于硬 的 doc values，当索引 更 ，全局序数要重建。

一个分片中父文 越多，那 全局序数的重建就需要更多的 。父子 系更 合于父文 少、子文 多的情况。

全局序数 情况下是延 建的：在refresh后的第一个父子 会触 全局序数的 建。而 个 建会致用 使用 感受到明 的 。 可以使用全局序数 加 来将全局序数 建的 由query 段移到refresh 段， 置如下：

```
PUT /company
{
  "mappings": {
    "branch": {},
    "employee": {
      "_parent": {
        "type": "branch",
        "fielddata": {
          "loading": "eager_global_ordinals" ①
        }
      }
    }
  }
}
```

① 在一个新的段可搜索前，`_parent` 字段的全局序数会被 建。

当父文 多 ，全局序数的 建会耗 很多 。此 可以通 加 `refresh_interval` 来 少 refresh 的次数，延 全局序数的有效 ， 也很大程度上 小了全局序数 秒重建的cpu消耗。

多代使用和

多代文 的 合 （ 看 祖 与 系） 然看起来很吸引人，但必 考 如下的代 ：

- 合越多，性能越差。
- 一代的父文 都要将其字符串 型的 `_id` 字段存 在内存中， 会占用大量内存。

当 考 父子 系是否 合 有 系模型 ， 考 下面 些建 ：

- 尽量少地使用父子 系， 在子文 多于父文 使用。
- 避免在一个 中使用多个父子 合 句。
- 在 has_child 中使用 filter 上下文，或者 置 score_mode none 来避免 算文 得分。
- 保 父 IDs 尽量短，以便在 doc values 中更好地 ，被 入 占用更少的内存。

最重要的是: 先考 下我 之前 的其他方式来 到父子 系的效果。

容

一些公司 天使用 Elasticsearch 索引 索 PB 数据， 但我 中的大多数都起 于 模 的 目。即使我 立志成 下一个 Facebook，我 的 行 余 却也跟不上梦想的脚 。 我 需要 今日所需而 建，但也要允 我 可以 活而又快速地 行水平 展。

Elasticsearch 了可 展性而生。它可以良好地 行于 的 本 又或者一个 有数百 点的集群，同 用 体 基本相同。 由小 模集群 大 模集群的 程几乎完全自 化并且无痛。由大 模集群 超大 模集群需要一些 和 ，但 是相 地无痛。

当然 一切并不是魔法。Elasticsearch 也有它的局限性。如果 了解 些局限性并能 与之相 ，集群 容的 程将会是愉快的。 如果 Elasticsearch 理不当，那 将 于一个充 痛苦的世界。

Elasticsearch 的 置会伴 走 很 的一段路，但 了 它最大的效用， 需要考 数据是如何流 的系 的。 我 将 常 的数据流： 序数据（ 相 性，例如日志或社交 数据流），以及基于用 的数据（ 有很大的文 集但可以按用 或客 分）。

一章将 助 在遇到不愉快之前做出正 的 。

容的 元

在 [dynamic-indices]，我 介 了一个分片即一个 *Lucene* 索引 ，一个 Elasticsearch 索引即一系列分片的集合。 的 用程序与索引 行交互，Elasticsearch 助 将 求路由至相 的分片。

一个分片即 容的 元 。一个最小的索引 有一个分片。 可能已 完全 足 的需求了 — 个分片即可存 大量的数据 — 但 限制了 的可 展性。

想象一下我 的集群由一个 点 成，在集群内我 有一个索引， 个索引只含一个分片：

```
PUT /my_index
{
  "settings": {
    "number_of_shards":   1, ①
    "number_of_replicas": 0
  }
}
```

① 建一个 有 1 主分片 0 个副本分片的索引.

个 置 也 很小，但它 足我 当前的需求而且 行代 低。

NOTE 当前我 只 主 分片。我 将在 副本分片 副本 分片。

在美好的一天，互 了我 ，一个 点再也承受不了我 的流量。 我 决定根据 一个只有一个分片的索引无 容因子 添加一个 点。 将会 生什 ？

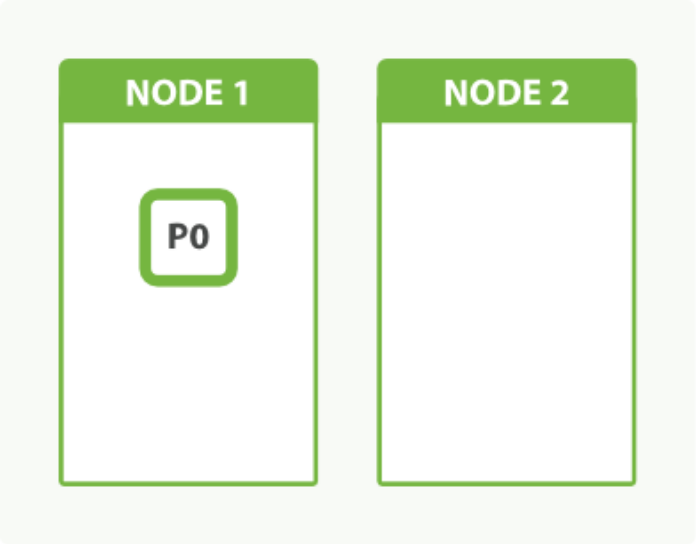

Figure 1. 一个只有一个分片的索引无 容因子

答案是：什 都不会 生。因 我 只有一个分片，已 没有什 可以放在第二个 点上的了。 我 不能 加索引的分片数因 它是 route documents to shards 算法中的重要元素：

```
shard = hash(routing) % number_of_primary_shards
```

我 当前的 只有一个就是将数据重新索引至一个 有更多分片的一个更大的索引，但 做将消耗的 是我 无法提供的。通 事先 ，我 可以使用 分配 的方式来完全避免 个 。

分片 分配

一个分片存在于 个 点，但一个 点可以持有多个分片。想象一下我 建 有 个主分片的索引而不是一个：

```
PUT /my_index
{
  "settings": {
    "number_of_shards":   2, ①
    "number_of_replicas": 0
  }
}
```

① 建 有 个主分片无副本分片的索引。

当只有一个 点 ， 个分片都将被分配至相同的 点。 从我 用程序的角度来看，一切都和之前一 作着。 用程序和索引 行通 ，而不是分片， 在 是只有一个索引。

，我 加入第二个 点，Elasticsearch 会自 将其中一个分片移 至第二个 点，如 一个 有 个分片的索引可以利用第二个 点 描 的那 ， 当重新分配完成后， 个分片都将接近至 倍于之前的 算能力。

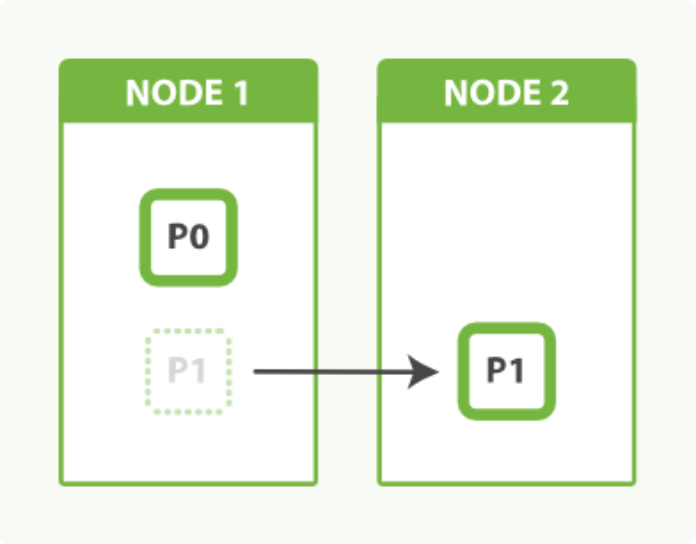

Figure 2. 一个 有 个分片的索引可以利用第二个 点

我 已 可以通 地将一个分片通 制到一个新的 点来加倍我 的 理能力。 最棒的是，我 零停机地做到了 一点。在分片移 程中，所有的索引搜索 求均在正常 行。

在 Elasticsearch 中新添加的索引 被指定了五个主分片。 意味着我最多可以将那个索引分散到五个 点上， 个 点一个分片。 它具有很高的 理能力， 未等 去思考一切就已 做到了！

分片分裂

用 常在 ， 什 Elasticsearch 不支持 分片分裂（shard-splitting）— 将 个分片分裂 个或更多部分的能力。 原因就是分片分裂是一个糟 的想法：

- 分裂一个分片几乎等于重新索引 的数据。它是一个比 将分片从一个 点 制到 一个 点更重量 的操作。
- 分裂是指数的。起初 有一个分片，然后分裂 个，然后四个，八个，十六个，等等。分裂并不会 好地把 的 理能力提升 50%。
- 分片分裂需要 有足 的能力支 一 索引的拷 。通常来 ，当 意 到 需要横向 展 ， 已 没有足 的剩余空 来做分裂了。

Elasticsearch 通 一 方式来支持分片分裂。 是可以把 的数据重新索引至一个 有 当分片个数的新索引（参 [reindex]）。 和移 分片比起来 依然是一个更加密集的操作，依然需要足 的剩余空 来完成，但至少 可以控制新索引的分片个数了。

海量分片

当新手 在了解 分片 分配 之后做的第一件事就是 自己 ：

> 我不知道 个索引将来会 得多大，并且 后我也不能更改索引的大小，所以 了保 起 ， 是 它 1000 个分片 ...
>
> — 一个新手的

一千个分片——当真？在　来　一千个　点　之前，　不　得　可能需要再三思考　的数据模型然后将它　重新索引　？

一个分片并不是没有代　的。　住：

- 一个分片的底　即　一个 Lucene 索引，会消耗一定文件句柄、内存、以及 CPU　　。
- 　一个搜索　求都需要命中索引中的　一个分片，如果　一个分片都　于不同的　点　好，但如果多个分片都需要在同一个　点上　争使用相同的　源就有些糟　了。
- 用于　算相　度的　　信息是基于分片的。如果有　多分片，　一个都只有很少的数据会　致很低的相　度。

TIP

　当的　分配是好的。但上千个分片就有些糟　。我　很　去定　分片是否　多了，　取决于它　的大小以及如何去使用它　。　　一百个分片但很少使用　好，　个分片但非常　繁地使用有可能就有点多了。　　控　的　点保　它　留有足　的空　　源来　理一些特殊情况。

横向　展　当分　段　行。　下一　段准　好足　的　源。　　只有当　　入到下一个　段，　才有　思考需要作出　些改　来　到　个　段。

容量

如果一个分片太少而　　1000　　个又太多，那　我　　知道我需要多少分片　？　　一般情况下是一个无法回答的　　。因　　在有太多相　的因素了：　使用的硬件、文　的大小和　　度、文　的索引分析方式、　行的　　型、　行的聚合以及　的数据模型等等。

幸　的是，在特定　景下　是一个容易回答的　　，尤其是　自己的　景：

1. 基于　准　用于生　　境的硬件　建一个　有　个　点的集群。
2. 　建一个和　准　用于生　　境相同配置和分析器的索引，但　它只有一个主分片无副本分片。
3. 索引　　的文　（或者尽可能接近　　）。
4. 　行　　的　　和聚合（或者尽可能接近　　）。

基本来　，　需要　制真　　境的使用方式并将它　全部　　到　个分片上直到它``挂掉。"　　上　挂掉的定　也取决于　：一些用　需要所有　　在 50　秒内返回；　一些　　于等上 5 秒　。

一旦　定　好了　个分片的容量，很容易就可以推算出整个索引的分片数。　　用　需要索引的数据　数加上一部分　期的　　，除以　个分片的容量，　果就是　需要的主分片个数。

TIP

容量　　不　当作　　的第一　。

先看看有没有　法　化　　Elasticsearch　　的使用方式。也　　有低效的　　，　少足　的内存，又或者　　了 swap？

我　　一些新手　于初始性能感到沮　，立即就着手　　回收又或者是　程数，而不是　理　　例如去掉通配符　　。

副本分片

目前 止我 只 主分片，但我 身 有 一个工具：副本分片。 副本分片的主要目的就是了故障 移，正如在 [distributed-cluster] 中 的：如果持有主分片的 点挂掉了，一个副本分片就会晋升 主分片的角色。

在索引写入 ，副本分片做着与主分片相同的工作。新文 首先被索引 主分片然后再同 到其它所有的副本分片。 加副本数并不会 加索引容量。

无 如何，副本分片可以服 于 求，如果 的索引也如常 的那 是偏向 使用的，那 可以通加副本的数目来提升 性能，但也要 此 加 外的硬件 源。

我 回到那个有着 个主分片索引的例子。我通 加第二个 点来提升索引容量。 加 外的点不会 助我 提升索引写入能力，但我 可以通 加副本数在搜索 利用 外的硬件：

```
PUT /my_index/_settings
{
  "number_of_replicas": 1
}
```

有 个主分片，加上 个主分片的一个副本， 共 予我 四个分片： 个 点一个，如 所示 一个有 个主分片一 副本的索引可以在四个 点中横向 展。

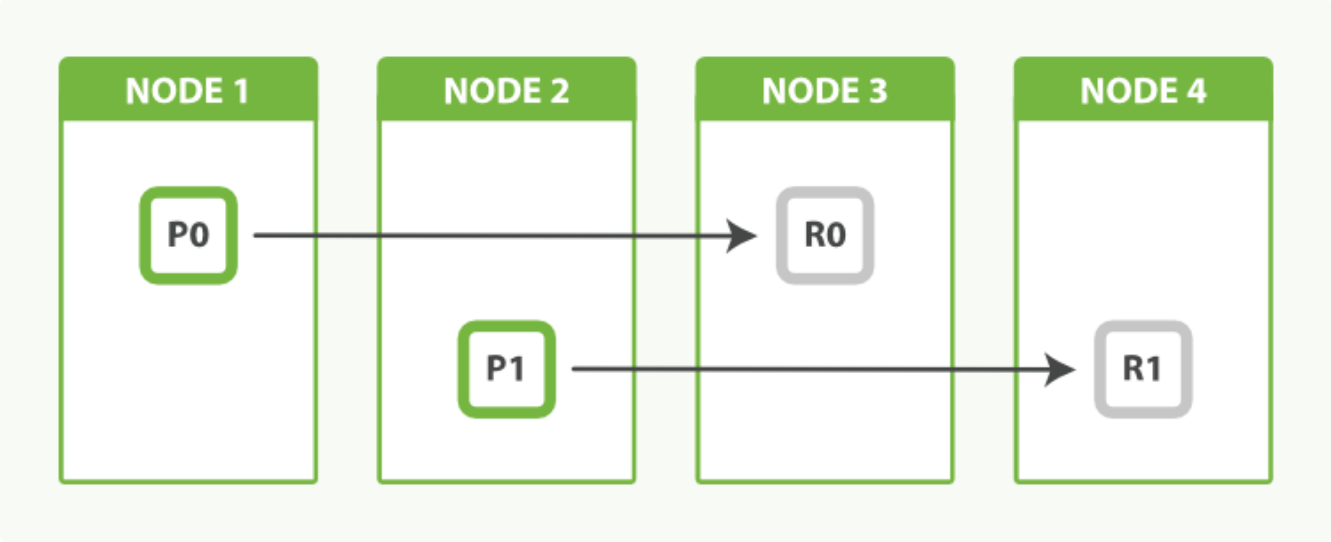

Figure 3. 一个 有 个主分片一 副本的索引可以在四个 点中横向 展

通 副本 行 均衡

搜索性能取决于最慢的 点的 ，所以 均衡所有 点的 是一个好想法。 如果我 只是 加一个 点而不是 个，最 我 会有 个 点各持有一个分片，而 一个持有 个分片做着 倍的工作。

我 可以通 整副本数量来平衡 些。通 分配 副本而不是一个，最 我 会 有六个分片， 好可以平均分 三个 点，如 所示 通 整副本数来均衡 点 ：

```
PUT /my_index/_settings
{
  "number_of_replicas": 2
}
```

作　　励，我　同　提升了我　的可用性。我　可以容忍　失　个　点而　然保持一　完整数据的拷　。

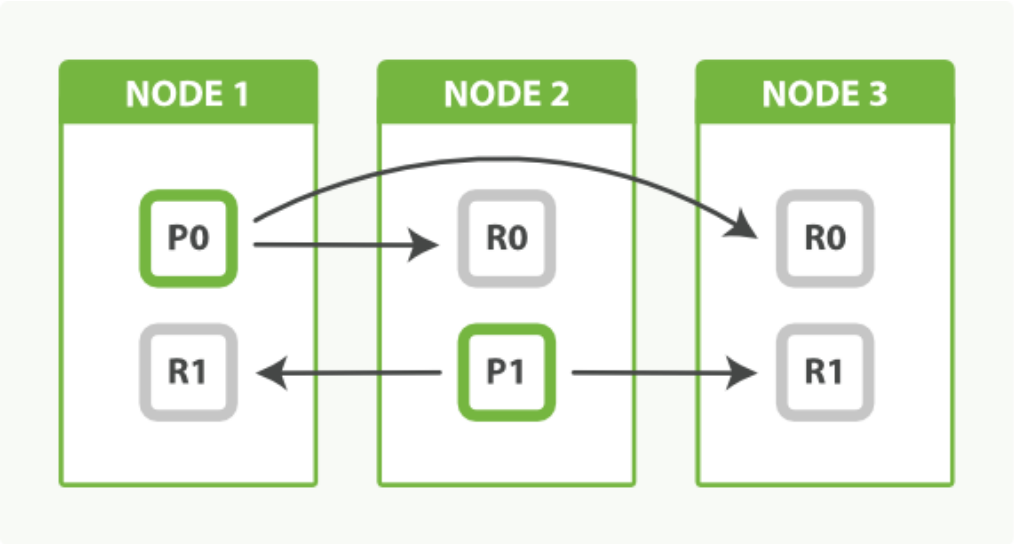

Figure 4. 通　　整副本数来均衡　点

NOTE	事　上　点　　3　　持有　个副本分片，然而没有主分片并不重要。副本分片与主分片做着相同的工作；它　只是扮演着略微不同的角色。没有必要　保主分片均　地分布在所有　点中。

多索引

最后，　住没有任何　　限制　的　用程序只使用一个索引。　　当我　　起一个搜索　求　，它被　　至索引中　个分片的一　拷　（一个主分片或一个副本分片），如果我　向多个索引　出同　的　求，会　生完全相同的事情——只不　会　及更多的分片。

TIP	搜索　1　个有着　50　个分片的索引与搜索　50　个　个都有　1　个分片的索引完全等　：搜索　求均命中 50 个分片。

当　需要在不停服　的情况下　加容量　，下面有一些有用的建　。相　于将数据　移到更大的索引中，可以　　做下面　些操作：

- 　建一个新的索引来存　新的数据。
- 同　搜索　个索引来　取新数据和旧数据。

　上，通　一点　先　　，添加一个新索引可以通　一　完全透明的方式完成，　的　用程序根本不会察到任何的改　。

在　　[index-aliases]，我　提到　使用索引　名来指向当前版本的索引。　　例来　，　　的索引命名 `tweets_v1` 而不是 `tweets` 。　的　用程序会与 `tweets`　行交互，但事　上它是一个指向 `tweets_v1` 的　名。　允　　将　名切　至一个更新版本的索引而保持服　　。

我 可以使用一个 似的技 通 加一个新索引来 展容量。 需要一点点 ，因 需要 个 名：一个用于搜索 一个用于索引数据：

```
PUT /tweets_1/_alias/tweets_search ①
PUT /tweets_1/_alias/tweets_index ①
```

① tweets_search 与 tweets_index 个 名都指向索引 tweets_1 。

新文 当索引至 tweets_index ，同 ，搜索 求 当 名 tweets_search 出。目前， 个名指向同一个索引。

当我 需要 外容量 ，我 可以 建一个名 tweets_2 的索引，并且像 更新 名：

```
POST /_aliases
{
  "actions": [
    { "add":    { "index": "tweets_2", "alias": "tweets_search" }}, ①
    { "remove": { "index": "tweets_1", "alias": "tweets_index"  }}, ②
    { "add":    { "index": "tweets_2", "alias": "tweets_index"  }}  ②
  ]
}
```

① 添加索引 tweets_2 到 名 tweets_search 。

② 将 名 tweets_index 由 tweets_1 切 至 tweets_2 。

一个搜索 求可以以多个索引 目 ，所以将搜索 名指向 tweets_1 以及 tweets_2 是完全有效的。然而，索引写入 求只能以 个索引 目 。因此，我 必 将索引写入的 名只指向新的索引。

TIP

一个文 GET 求，像一个索引写入 求那 ，只能以 个索引 目 。 致在通 ID 取文 的 景下有一点 。作 代替， 可以 tweets_1 以及 tweets_2 行一个 {ref}/query-dsl-ids-query.html[ids] 搜索 求， 或者 {ref}/docs-multi-get.html[multi-get] 求。

在服 行中使用多索引来 展索引容量 于一些使用 景有着特 的好 ，像我 将在下一 中 的基于 的数据例如日志或社交事件流。

基于 的数据

Elasticsearch 的常用案例之一便是日志 ， 它 在太常 了以至于 Elasticsearch 提供了一个集成的日志平台叫做 ELK stack— Elasticsearch，Logstash，以及 Kibana ——来 工作 得 。

Logstash 采集、解析日志并在将它 写入Elasticsearch之前格式化。 Elasticsearch 扮演了一个集中式的日志服 角色， Kibana 是一个 形化前端可以很容易地 以及可 化的 化。

搜索引 中大多数使用 景都是 慢相 定的文 集合。搜索 最相 的文 ，而不 心它是何 建的。

日志——以及其他基于　的数据流例如社交　活　——　上有很大不同。　索引中文　数量迅速　，通常随　加速。　文　几乎不会更新，基本以最近文　搜索目　。随着　推移，文　逐　失去　。

我　需要　整索引　使其能　工作于　基于　的数据流。

按　索引

如果我　此　型的文　建立一个超大索引，我　可能会很快耗尽存　空　。日志事件会不断的　来，不会停　也不会中断。　我　可以使用 `scroll`　和批量　除来　除旧的事件。但　方法　非常低效　。当　除一个文　，它只会被　被　除（参　[deletes-and-updates]）。在包含它的段被合并之前不会被物理　除。

替代方案是，我　使用一个　索引。　可以着手于一个按年的索引　(`logs_2014`)　或按月的索引(`logs_2014-10`)。　也　当　的　得十分繁忙　，　需要切　到一个按天的索引（`logs_2014-10-24`）。　除旧数据十分　：只需要　除旧的索引。

　方法有　的　点，允　在需要的　候　行　容。　不需要　先做任何　的决定。　天都是一个新的机会来　整　的索引　来　当前需求。　用相同的　到决定　个索引的大小上。起初也　需要的　是　周一个主分片。　一　子，也　需要　天五个主分片。　都不重要——任何　都可以　整到新的　境。

　名可以　助我　更加透明地在索引　切　。　当　建索引　，　可以将　`logs_current` 指向当前索引来接收新的日志事件，　当　索　，更新 `last_3_months` 来指向所有最近三个月的索引：

```
POST /_aliases
{
  "actions": [
    { "add":    { "alias": "logs_current",  "index": "logs_2014-10" }}, ①
    { "remove": { "alias": "logs_current",  "index": "logs_2014-09" }}, ①
    { "add":    { "alias": "last_3_months", "index": "logs_2014-10" }}, ②
    { "remove": { "alias": "last_3_months", "index": "logs_2014-07" }}  ②
  ]
}
```

① 将 `logs_current` 由九月切　至十月。

② 将十月添加到 `last_3_months` 并且　掉七月。

索引模板

Elasticsearch　不要求　在使用一个索引前　建它。　于日志　用，依　于自　建索引比手　建要更加方便。

Logstash 使用事件中的　来生成索引名。　天被索引至不同的索引中，因此一个 `@timestamp` `2014-10-01　00:00:01` 的事件将被　送至索引　`logstash-2014.10.01`　中。如果那个索引不存在，它将被自　建。

通常我　想要控制一些新建索引的　置（settings）和映射（mappings）。也　我　想要限制分片数　`1`，并且禁用 `_all` 域。索引模板可以用于控制何　置（settings）　当被　用于新　建的索引：

```
PUT /_template/my_logs ①
{
  "template": "logstash-*", ②
  "order":    1, ③
  "settings": {
    "number_of_shards": 1 ④
  },
  "mappings": {
    "_default_": { ⑤
      "_all": {
        "enabled": false
      }
    }
  },
  "aliases": {
    "last_3_months": {} ⑥
  }
}
```

① 建一个名 my_logs 的模板。

② 将 个模板 用于所有以 logstash- 起始的索引。

③ 个模板将会覆 的 logstash 模板，因 模板的 order 更低。

④ 限制主分片数量 1 。

⑤ 所有 型禁用 _all 域。

⑥ 添加 个索引至 last_3_months 名中。

个模板指定了所有名字以 logstash- 起始的索引的 置，不 它是手 是自 建的。如果我 明天的索引需要比今天更大的容量，我 可以更新 个索引以使用更多的分片。

个模板 将新建索引添加至了 last_3_months 名中，然而从那个 名中 除旧的索引 需要手 行。

数据 期

随着 推移，基于 数据的相 度逐 降低。 有可能我 会想要 看上周、上个月甚至上一年度 生了什 ，但是大多数情况，我 只 心当前 生的。

按 索引 来的一个好 是可以方便地 除旧数据：只需要 除那些 得不重要的索引就可以了。

```
DELETE /logs_2013*
```

除整个索引比 除 个文 要更加高效：Elasticsearch 只需要 除整个文件 。

但是 除索引是 手段。在我 决定完全 除它之前 有一些事情可以做来 助数据更加 雅地 期。

移旧索引

随着数据被 ，很有可能存在一个 点 索引——今日的索引。 所有新文 都会被加到那个索引，几乎所有 都以它 目 。那个索引 当使用 最好的硬件。

Elasticsearch 是如何得知 台是 最好的服 器 ？ 可以通 台服 器指定任意的 来告 它。例如， 可以像 一个 点：

```
./bin/elasticsearch --node.box_type strong
```

`box_type` 参数是完全随意的—— 可以将它随意命名只要 喜 ——但 可以用 些任意的 来告 Elasticsearch 将一个索引分配至何 。

我 可以通 按以下配置 建今日的索引来 保它被分配到我 最好的服 器上：

```
PUT /logs_2014-10-01
{
  "settings": {
    "index.routing.allocation.include.box_type" : "strong"
  }
}
```

昨日的索引不再需要我 最好的服 器了，我 可以通 更新索引 置将它移 到 `medium` 的 点上：

```
POST /logs_2014-09-30/_settings
{
  "index.routing.allocation.include.box_type" : "medium"
}
```

索引 化（Optimize）

昨日的索引不大可能会改 。 日志事件是静 的：已 生的 往不会再改 了。如果我 将 个分片合并至一个段（Segment），它会占用更少的 源更快地 。 我 可以通 [optimize-api]来做到。

分配在 `strong` 主机上的索引 行 化（Optimize）操作将会是一个糟 的想法， 因 化操作将消耗 点上大量 I/O 并 索引今日日志造成冲 。但是 `medium` 的 点没有做太多 似的工作，我 可以安全地在上面 行 化。

昨日的索引有可能 有副本分片。如果我 下 一个 化（Optimize） 求， 它会 化主分片和副本分片， 有些浪 。然而，我 可以 移除副本分片， 行 化，然后再恢 副本分片：

```
POST /logs_2014-09-30/_settings
{ "number_of_replicas": 0 }

POST /logs_2014-09-30/_optimize?max_num_segments=1

POST /logs_2014-09-30/_settings
{ "number_of_replicas": 1 }
```

当然，没有副本我 将面 磁 故障而 致 失数据的 。 可能想要先通 {ref}/modules-snapshots.html[snapshot-restore API] 数据。

旧索引

当索引 得更“老”，它 到 一个几乎不会再被 的 点。 我 可以在 个 段 除它 ，但也 想将它 留在 里以防万一有人在半年后 想要 它 。

些索引可以被 。它 会存在于集群中，但它 不会消耗磁 空 以外的 源。重新打 一个索引要比从 中恢 快得多。

在 之前， 得我 去刷写索引来 保没有事 残留在事 日志中。一个空白的事 日志会使得索引在重新打 恢 得更快：

```
POST /logs_2014-01-*/_flush ①
POST /logs_2014-01-*/_close ②
POST /logs_2014-01-*/_open ③
```

① 刷写（Flush）所有一月的索引来清空事 日志。

② 所有一月的索引.

③ 当 需要再次 它 ，使用 open API 来重新打 它 。

旧索引

最后，非常旧的索引可以通 {ref}/modules-snapshots.html[snapshot-restore API] 至 期存 例如共享磁 或者 Amazon S3，以防日后 可能需要 它 。 当存在 我 就可以将索引从集群中 除了。

基于用 的数据

通常来 ，用 使用 Elasticsearch 的原因是他 需要添加全文 索或者需要分析一个已 存在的 用。他 建一个索引来存 所有文 。公司里的其他人也逐 了 Elasticsearch 来的好 ，也想把他 的数据添加到 Elasticsearch 中去。

幸 的是，Elasticsearch 支持http://en.wikipedia.org/wiki/Multitenancy[多租]所以 个用 可以在相同的集群中 有自己的索引。 有人偶 会想要搜索所有用 的文 ， 情况可以通 搜索所有索引 ，但大多数情况下用 只 心它 自己的文 。

一些用 有着比其他人更多的文 ，一些用 可能有比其他人更多的搜索次数， 所以 指定

个索引主分片和副本分片数量能力的需要　很　合使用“一个用　一个索引”的模式。　似地，　繁忙的索引可以通　分片分配　指定到高配的　点。（参　移旧索引。）

TIP	不要　个索引都使用　的主分片数。想想看它需要存　多少数据。有可能　需要一个分片——再多的都只是浪　源。

大多数　Elasticsearch　的用　到　里就已　了。　的“一个用　一个索引”　大多数　景都可以　足了。

于例外的　景，　可能会　需要支持很大数量的用　，都是相似的需求。一个例子可能是　一个　有几千个　箱　的　提供搜索服　。　一些　可能有巨大的流量，但大多数都很小。将一个有着　个分片的索引用于一个小　模　已　是足　的了——一个分片可以承　很多个　的数据。

我　需要的是一　可以在用　共享　源的方法，　个用　他　有自己的索引　印象，而不在小用上浪　源。

共享索引

我　可以　多的小　使用一个大的共享的索引，将　索引　一个字段并且将它用作一个　器：

```
PUT /forums
{
  "settings": {
    "number_of_shards": 10 ①
  },
  "mappings": {
    "post": {
      "properties": {
        "forum_id": { ②
          "type":  "string",
          "index": "not_analyzed"
        }
      }
    }
  }
}

PUT /forums/post/1
{
  "forum_id": "baking", ②
  "title":    "Easy recipe for ginger nuts",
  ...
}
```

① 建一个足　大的索引来存　数千个小　的数据。

② 个帖子都必　包含一个 `forum_id` 来　它属于　个　。

我　可以把　`forum_id`　用作一个　器来　个　行搜索。　个　器可以排除索引中　大部分的数据（属于其它　的数据），　存会保　快速的　：

```
GET /forums/post/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "title": "ginger nuts"
        }
      },
      "filter": {
        "term": {
          "forum_id": {
            "baking"
          }
        }
      }
    }
  }
}
```

个 法行得通，但我 可以做得更好。 来自于同一个 的帖子可以 地容 于 个分片，但它 在被打散到了 个索引的所有十个分片中。 意味着 个搜索 求都必 被 至所有十个分片的一个主分片或者副本分片。 如果能 保 所有来自于同一个 的所有帖子都被存于同一个分片可能会是个好想法。

在 [routing-value]，我 一个文 将通 使用如下公式来分配到一个指定分片：

```
shard = hash(routing) % number_of_primary_shards
```

routing 的 文 的 _id ，但我 可以覆 它并且提供我 自己自定 的路由 ，例如 forum_id 。所有有着相同 routing 的文 都将被存 于相同的分片：

```
PUT /forums/post/1?routing=baking ①
{
  "forum_id": "baking", ①
  "title":    "Easy recipe for ginger nuts",
  ...
}
```

① 将 forum_id 用于路由 保 所有来自相同 的帖子都存 于相同的分片。

当我 搜索一个指定 的帖子 ，我 可以 相同的 routing 来保 搜索 求 在存有我 文 的分片上 行：

```
GET /forums/post/_search?routing=baking ①
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "title": "ginger nuts"
        }
      },
      "filter": {
        "term": { ②
          "forum_id": {
            "baking"
          }
        }
      }
    }
  }
}
```

① 求 在 于 routing 的分片上 行。

② 我 是需要 （Filter） ，因 一个分片可以存 来自于很多 的帖子。

多个 可以通 一个逗号分隔的列表来指定 routing ，然后将 个 forum_id 包含于一个 terms :

```
GET /forums/post/_search?routing=baking,cooking,recipes
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "title": "ginger nuts"
        }
      },
      "filter": {
        "terms": {
          "forum_id": {
            [ "baking", "cooking", "recipes" ]
          }
        }
      }
    }
  }
}
```

方式从技 上来 比 高效，由于要 一个 或者索引 求指定 routing 和 terms 的看起来有一点的 拙。索引 名可以 解决 些！

利用　名　　一个用　一个索引

　了保持　　的　　，我　想　我　的　用　　我　　个用　都有一个　　的索引——或者按照我　的例子　个　　一个——尽管　　上我　用的是一个大的shared index。　因此，我　需要一　方式将 routing　及　　器　含于 forum_id 中。

索引　名可以　　做到　些。当　将一个　名与一个索引　　起来，　可以指定一个　　器和一个路由　：

```
PUT /forums/_alias/baking
{
  "routing": "baking",
  "filter": {
    "term": {
      "forum_id": "baking"
    }
  }
}
```

　在我　可以将　baking　　名　　一个　独的索引。索引至　baking　　名的文　会自　地　用我　自定的路由　：

```
PUT /baking/post/1 ①
{
  "forum_id": "baking", ①
  "title":    "Easy recipe for ginger nuts",
  ...
}
```

① 我　　是需要　　器指定 forumn_id 字段，但自定　路由　已　是　含的了。

　baking　　名上的　　只会在自定　路由　　的分片上　行，并且　果也自　按照我　指定的　　器行了　　：

```
GET /baking/post/_search
{
  "query": {
    "match": {
      "title": "ginger nuts"
    }
  }
}
```

当　多个　　行搜索　可以指定多个　名：

```
GET /baking,recipes/post/_search ①
{
  "query": {
    "match": {
      "title": "ginger nuts"
    }
  }
}
```

① 个 routing 的 都会 用，返回 果会匹配任意一个 器。

一个大的用

大 模流行 都是从小 起 的。 有一天我 会 我 共享索引中的一个分片要比其它分片更加繁忙，因 个分片中一个 的文 得更加 。 ，那个 需要属于它自己的索引。

我 用来提供一个用 一个索引的索引 名 了我 一个 的 移 方式。

第一 就是 那个 建一个新的索引，并 其分配合理的分片数，可以 足一定 期的数据 ：

```
PUT /baking_v1
{
  "settings": {
    "number_of_shards": 3
  }
}
```

第二 就是将共享的索引中的数据 移到 用的索引中，可以通 scroll 和bulk API来 。 当 移完成 ，可以更新索引 名指向那个新的索引：

```
POST /_aliases
{
  "actions": [
    { "remove": { "alias": "baking", "index": "forums"    }},
    { "add":    { "alias": "baking", "index": "baking_v1" }}
  ]
}
```

更新索引 名的操作是原子性的；就像在 一个 。 的 用程序 是在与 baking API 交互并且 于它已 指向一个 用的索引 无感知。

用的索引不再需要 器或者自定 的路由 了。我 可以依 于 Elasticsearch 使用的 _id 字段来做分区。

最后一 是从共享的索引中 除旧的文 ，可以通 搜索之前的路由 以及 ID 然后 行批量 除操作来 。

一个用 一个索引模型的 雅之 在于它允 少 源消耗，保持快速的 ，同 有在需要 零宕机 容的能力。

容并不是无限的

整个章 我 了多 Elasticsearch 可以做到的 容方式。 大多数的 容 可以通 添加点来解决。但有一 源是有限制的，因此 得我 真 待：集群状 。

集群状 是一 数据 ， 存下列集群 的信息：

- 集群 的 置
- 集群中的 点
- 索引以及它 的 置、映射、分析器、 器（Warmers）和 名
- 与 个索引 的分片以及它 分配到的 点

可以通 如下 求 看当前的集群状 ：

```
GET /_cluster/state
```

集群状 存在于集群中的 个 点，包括客 端 点。 就是 什 任何一个 点都可以将 求直接 至被 求数据的 点——个 点都知道 个文 在 里。

只有主 点被允 更新集群状 。想象一下一个索引 求引入了一个之前未知的字段。持有那个文 的主分片所在的 点必 将新的映射 到主 点上。 主 点把更改合并到集群状 中，然后向所有集群中的所有 点 布一个新的版本。

搜索 求 使用 集群状 ，但它 不会 生修改。同 ，文 的 改 求也不会 集群状 生修改。当然，除非它 引入了一个需要更新映射的新的字段了。 的来 ，集群状 是静的不会成 瓶 。

然而，需要 住的是相同的数据 需要在 个 点的内存中保存，并且当它 生更改 必 布到 一个 点。 集群状 的数据量越大， 个操作就会越久。

我 最常 的集群状 就是引入了太多的字段。一个用 可能会决定 一个 IP 地址或者 个 referer URL 使用一个 独的字段。 下面 个例子通 一个唯一的 referer 使用一个不同的字段名来保持 面 量的 数：

```
POST /counters/pageview/home_page/_update
{
  "script": "ctx._source[referer]++",
  "params": {
    "referer": "http://www.foo.com/links?bar=baz"
  }
}
```

方式十分的糟 ！它会生成数百万个字段， 些都需要被存 在集群状 中。 当 到一个新的

referer ，都有一个新的字段需要加入那个已 膨 的集群状 中， 都需要被 布到集群的 个点中去。

更好的方式是使用nested objects， 它使用一个字段作参数名—`referer`— 一个字段作 的 —`count` ：

```
    "counters": [
      { "referer": "http://www.foo.com/links?bar=baz",  "count": 2 },
      { "referer": "http://www.linkbait.com/article_3", "count": 10 },
      ...
    ]
```

嵌套的方式有可能会 加文 数量，但 Elasticsearch 生来就是 了解决它的。重要的是保持集群状小而敏捷。

最 ，不管 的初衷有多好， 可能会 集群 点数量、索引、映射 于一个集群来 是太大了。 此 ，可能有必要将 个 拆分到多个集群中了。感 {ref}/modules-tribe.html[tribe nodes]，甚至可以向多个集群 出搜索 求，就好像我 有一个巨大的集群那 。