数据　入和　出

无　我　写什　　的程序，目的都是一　的：以某　方式　　数据服　我　的目的。　　但是数据不　　由随机位和字　　成。我　建立数据元素之　的　系以便于表示　体，或者　　世界中存在的　事物　。如果我　知道一个名字和　子　件地址属于同一个人，那　它　将会更有意　。

尽管在　　世界中，不是所有的　型相同的　体看起来都是一　的。　　一个人可能有一个家庭　　号　，而　一个人只有一个手机号　，再一个人可能　者兼有。　　一个人可能有三个　子　件地址，而　一个人却一个都没有。一位西班牙人可能有　个姓，而　英　的人可能只有一个姓。

面向　象　程　言如此流行的原因之一是　象　我　表示和　理　　世界具有潜在的　　的数据　　的　体，到目前　止，一切都很完美！

但是当我　需要存　　些　体　　来了，　　上，我　以行和列的形式存　数据到　系型数据　中，相当于使用　子表格。正因　我　使用了　　不　活的存　媒介　致所有我　使用　象的　活性都　失了。

但是否我　可以将我　的　象按　象的方式来存　？　　我　就能更加　注于　　使用　　数据，而不是在　子表格的局限性下　我　的　用建模。我　可以重新利用　象的　活性。

一个　　象　　是基于特定　言的内存的数据　　。　了通　　送或者存　它，我　需要将它表示成某　　准的格式。　JSON　是一　以人可　的文本表示　象的方法。　它已　　成　NoSQL　世界交　数据的事　　准。当一个　象被序列化成　JSON，它被称　一个 *JSON* 文　。

Elastcisearch　是分布式的　文　　存　。它能存　和　索　　的数据　　—序列化成　JSON文　—以　的方式。　句　　，一旦一个文　被存　在 Elasticsearch 中，它就是可以被集群中的任意　点　索到。

当然，我　不　要存　数据，我　一定　需要　　它，成批且快速的　　它　。　　尽管　存的　　NoSQL　解决方案允　我　以文　的形式存　　象，但是他　　旧需要我　思考如何　　我　的数据，以及　定　些　字段需要被索引以加快数据　索。

在　Elasticsearch　中，　　个字段的所有数据　都是　　被索引的　。　即　个字段都有　了快速　索　置的　用倒排索引。而且，不像其他多数的数据　，它能在　　相同的　　中　　使用所有　些倒排索引，并以　人的速度返回　果。

在本章中，我　展示了用来　建，　索，更新和　除文　的　　API。就目前而言，我　不　心文　中的数据或者　　它　。　　所有我　　心的就是在　　Elasticsearch　　中　　安全的存　文　，以及如何将文　再次返回。

什　是文　？

在大多数　用中，多数　体或　象可以被序列化　包含　　的　　JSON　　象。　　一个　　可以是一个字段或字段的名称，一个　　可以是一个字符串，一个数字，一个布　，　　一个　象，一些数　，或一些其它特殊　型　如表示日期的字符串，或代表一个地理位置的　象：

```
{
    "name":         "John Smith",
    "age":          42,
    "confirmed":    true,
    "join_date":    "2014-06-01",
    "home": {
        "lat":      51.5,
        "lon":      0.1
    },
    "accounts": [
        {
            "type": "facebook",
            "id":   "johnsmith"
        },
        {
            "type": "twitter",
            "id":   "johnsmith"
        }
    ]
}
```

通常情况下，我 使用的 象 和 文 是可以互相替 的。不 ，有一个区 ：一个 象 是 似于 hash 、 hashmap 、字典或者 数 的 JSON 象， 象中也可以嵌套其他的 象。 象可能包含了 外一些 象。在 Elasticsearch 中， 文 有着特定的含 。它是指最 或者根 象， 个根 象被序列化成 JSON 并存 到 Elasticsearch 中，指定了唯一 ID。

WARNING 字段的名字可以是任何合法的字符串，但不可以包含 段。

文 元数据

一个文 不 包含它的数据 ，也包含 元数据 —— 有 文 的信息。 三个必 的元数据元素如下：

`_index`
文 在 存放

`_type`
文 表示的 象

`_id`
文 唯一

_index

一个 索引 是因共同的特性被分 到一起的文 集合。 例如， 可能存 所有的 品在索引 `products` 中，而存 所有 的交易到索引 `sales` 中。 然也允 存 不相 的数据到一个索引中，但 通常看作是一个反模式的做法。

TIP

上，在 Elasticsearch 中，我 的数据是被存 和索引在 分片 中，而一个索引 是 上的命名空 ， 个命名空 由一个或者多个分片 合在一起。 然而，是一个内部 ，我 的 用程序根本不 心分片， 于 用程序而言，只需知道文 位于一个 索引 内。 Elasticsearch 会 理所有的 。

我 将在 [index-management] 介 如何自行 建和管理索引，但 在我 将 Elasticsearch 我 建索引。 所有需要我 做的就是 一个索引名， 个名字必 小写，不能以下 ，不能包含逗号。我 用 `website` 作 索引名 例。

_type

数据可能在索引中只是松散的 合在一起，但是通常明 定 一些数据中的子分区是很有用的。例如，所有的 品都放在一个索引中，但是 有 多不同的 品 ，比如 "electronics" 、 "kitchen" 和 "lawn-care"。

些文 共享一 相同的（或非常相似）的模式：他 有一个 、描述、 品代 和 格。他 只是正好属于“ 品”下的一些子 。

Elasticsearch 公 了一个称 *types* （ 型）的特性，它允 在索引中 数据 行 分区。不同 types 的文 可能有不同的字段，但最好能 非常相似。 我 将在 [mapping] 中更多的 于 types 的一些 用和限制。

一个 `_type` 命名可以是大写或者小写，但是不能以下 或者句号 ，不 包含逗号， 并且 度限制 256个字符. 我 使用 `blog` 作 型名 例。

_id

ID 是一个字符串，当它和 `_index` 以及 `_type` 合就可以唯一 定 Elasticsearch 中的一个文 。 当 建一个新的文 ，要 提供自己的 `_id` ，要 Elasticsearch 生成。

其他元数据

有一些其他的元数据元素，他 在 [mapping] 行了介 。通 前面已 列出的元数据元素， 我 已 能存 文 到 Elasticsearch 中并通 ID 索它— 句 ，使用 Elasticsearch 作 文 的存 介 。

索引文

通 使用 <code>index</code> API ，文 可以被 索引 —— 存 和使文 可被搜索。 但是首先，我 要 定文 的位置。正如我 的，一个文 的 <code>_index</code> 、 <code>_type</code> 和 <code>_id</code> 唯一 一个文 。 我 可以提供自定 的 <code>_id</code> ，或者 <code>index</code> API 自 生成。

使用自定 的 ID

如果 的文 有一个自然的 符 （例如，一个 `user_account` 字段或其他 文 的 ），使用如下方式的 `index` API 并提供 自己 `_id` ：

```
PUT /{index}/{type}/{id}
{
  "field": "value",
  ...
}
```

个例子，如果我 的索引称 website ， 型称 blog ，并且 123 作 ID ，那 索引 求 是下面 ：

```
PUT /website/blog/123
{
  "title": "My first blog entry",
  "text":  "Just trying this out...",
  "date":  "2014/01/01"
}
```

Elasticsearch 体如下所示：

```
{
   "_index":    "website",
   "_type":     "blog",
   "_id":       "123",
   "_version":  1,
   "created":   true
}
```

表明文 已 成功 建， 索引包括 _index 、 _type 和 _id 元数据，以及一个新元素： _version 。

在 Elasticsearch 中 个文 都有一个版本号。当 次 文 行修改 （包括 除）， _version 的 会 。 在 理冲突 中，我 了 使用 _version 号 保 的 用程序中的一部分修改不会覆一部分所做的修改。

Autogenerating IDs

如果 的数据没有自然的 ID， Elasticsearch 可以 我 自 生成 ID 。 求的 整 ：不再使用 PUT (“使用 个 URL 存 个文 ”)，而是使用 POST (“存 文 在 个 URL 命名空 下”)。

在 URL 只需包含 _index 和 _type ：

```
POST /website/blog/
{
  "title": "My second blog entry",
  "text":  "Still trying this out...",
  "date":  "2014/01/01"
}
```

除了 _id 是 Elasticsearch 自　生成的，　　的其他部分和前面的　似：

```
{
   "_index":    "website",
   "_type":     "blog",
   "_id":       "AVFgSgVHUP18jI2wRx0w",
   "_version":  1,
   "created":   true
}
```

自　生成的 ID 是 URL-safe、 基于 Base64　　且　度　20个字符的 GUID 字符串。　　些 GUID 字符串由可修改的　　FlakeID　　模式生成，　　模式允　多个　点并行生成唯一　　ID　　，且互相之　的冲突概率几乎　零。

取回一个文

　了从 Elasticsearch 中　索出文　，我　　然使用相同的 _index , _type , 和 _id ，但是 HTTP　　更改 GET :

```
GET /website/blog/123?pretty
```

　体包括目前已　熟悉了的元数据元素，再加上　_source　字段，　个字段包含我　索引数据　　送 Elasticsearch 的原始 JSON 文　：

```
{
  "_index" :   "website",
  "_type" :    "blog",
  "_id" :      "123",
  "_version" : 1,
  "found" :    true,
  "_source" :  {
      "title": "My first blog entry",
      "text":  "Just trying this out...",
      "date":  "2014/01/01"
  }
}
```

NOTE

在　求的　　串参数中加上　　pretty　　参数，正如前面的例子中看到的，　将会　用 Elasticsearch 的 *pretty-print* 功能，　功能 使得 JSON　　体更加可　。但是，_source 字段不能被格式化打印出来。相反，我　得到的　_source　字段中的　JSON　串，好是和我　　它的一　。

GET　求的　　体包括 {"found": true} ，　　了文　已　被　到。 如果我　　求一个不存在的文，我　　旧会得到一个 JSON　　体，但是 found 将会是 false 。 此外， HTTP　　将会是 404 Not Found , 而不是 200 OK 。

我 可以通 -i 参数 curl 命令， 参数能 示 的 部：

```
curl -i -XGET http://localhost:9200/website/blog/124?pretty
```

示 部的 体 在 似 ：

```
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=UTF-8
Content-Length: 83

{
  "_index" : "website",
  "_type" :  "blog",
  "_id" :    "124",
  "found" :  false
}
```

返回文 的一部分

情况下， GET 求会返回整个文 ， 个文 正如存 在 _source 字段中的一 。但是也 只其中的 title 字段感 趣。 个字段能用 _source 参数求得到，多个字段也能使用逗号分隔的列表来指定。

```
GET /website/blog/123?_source=title,text
```

_source 字段 在包含的只是我 求的那些字段，并且已 将 date 字段 掉了。

```
{
  "_index" :   "website",
  "_type" :    "blog",
  "_id" :      "123",
  "_version" : 1,
  "found" :   true,
  "_source" : {
      "title": "My first blog entry" ,
      "text":  "Just trying this out..."
  }
}
```

或者，如果 只想得到 _source 字段，不需要任何元数据， 能使用 _source 端点：

```
GET /website/blog/123/_source
```

那 返回的的内容如下所示：

```
{
   "title": "My first blog entry",
   "text":  "Just trying this out...",
   "date":  "2014/01/01"
}
```

文 是否存在

如果只想 一个文 是否存在--根本不想 心内容—那 用 HEAD 方法来代替 GET 方法。 HEAD 求没有返回体，只返回一个 HTTP 求 ：

```
curl -i -XHEAD http://localhost:9200/website/blog/123
```

如果文 存在， Elasticsearch 将返回一个 200 ok 的状 ：

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=UTF-8
Content-Length: 0
```

若文 不存在， Elasticsearch 将返回一个 404 Not Found 的状 ：

```
curl -i -XHEAD http://localhost:9200/website/blog/124
```

```
HTTP/1.1 404 Not Found
Content-Type: text/plain; charset=UTF-8
Content-Length: 0
```

当然，一个文 是在 的 候不存在，并不意味着一 秒之后它也不存在：也 同 正好 一个 程就 建了 文 。

更新整个文

在 Elasticsearch 中文 是 不可改 的，不能修改它 。相反，如果想要更新 有的文 ，需要 重建索引或者 行替 ， 我 可以使用相同的 index API 行 ，在 索引文 中已 行了 。

```
PUT /website/blog/123
{
  "title": "My first blog entry",
  "text":  "I am starting to get the hang of this...",
  "date":  "2014/01/02"
}
```

在 体中，我 能看到 Elasticsearch 已 加了 _version 字段 ：

```
{
  "_index" :   "website",
  "_type" :    "blog",
  "_id" :      "123",
  "_version" : 2,
  "created":   false ①
}
```

① created 志 置成 false，是因 相同的索引、 型和 ID 的文 已 存在。

在内部，Elasticsearch 已将旧文 已 除，并 加一个全新的文 。 尽管 不能再旧版本的文 行 ，但它并不会立即消失。当 索引更多的数据，Elasticsearch 会在后台清理 些已 除文 。

在本章的后面部分，我 会介 update API， 个 API 可以用于 partial updates to a document 。 然它似乎 文 直接 行了修改，但 上 Elasticsearch 按前述完全相同方式 行以下 程：

1. 从旧文 建 JSON
2. 更改 JSON
3. 除旧文
4. 索引一个新文

唯一的区 在于，update API 通 一个客 端 求来 些 ，而不需要 独的 get 和 index 求。

建新文

当我 索引一个文 ， 我 正在 建一个完全新的文 ，而不是覆 有的 ？

住， _index 、 _type 和 _id 的 合可以唯一 一个文 。所以， 保 建一个新文 的最 法是，使用索引 求的 POST 形式 Elasticsearch 自 生成唯一 _id：

```
POST /website/blog/
{ ... }
```

然而，如果已 有自己的 _id，那 我 必 告 Elasticsearch，只有在相同的 _index 、 _type 和 _id 不存在 才接受我 的索引 求。 里有 方式，他 做的 是相同的事情。使用 ，取决于 使用起来更方便。

第一 方法使用 op_type -字符串参数：

```
PUT /website/blog/123?op_type=create
{ ... }
```

第二　方法是在 URL 末端使用 /_create：

```
PUT /website/blog/123/_create
{ ... }
```

如果　建新文　的　求成功　行，Elasticsearch 会返回元数据和一个 201 Created 的 HTTP　　。

　一方面，如果具有相同的 _index 、 _type 和 _id 的文　已　存在，Elasticsearch 将会返回 409 Conflict　，以及如下的　　信息：

```
{
   "error": {
      "root_cause": [
         {
            "type": "document_already_exists_exception",
            "reason": "[blog][123]: document already exists",
            "shard": "0",
            "index": "website"
         }
      ],
      "type": "document_already_exists_exception",
      "reason": "[blog][123]: document already exists",
      "shard": "0",
      "index": "website"
   },
   "status": 409
}
```

除文

　除文　的　法和我　所知道的　　相同，只是使用 DELETE 方法：

```
DELETE /website/blog/123
```

如果　到　文　，Elasticsearch　将要返回一个　200　ok　的　HTTP　　，和一个　似以下　　的　体。注意，字段 _version　已　　加:

```
{
  "found" :    true,
  "_index" :   "website",
  "_type" :    "blog",
  "_id" :      "123",
  "_version" : 3
}
```

如果文　没有　到，我　将得到 404 Not Found 的　　和　似　　的　　体：

```
{
  "found" :    false,
  "_index" :   "website",
  "_type" :    "blog",
  "_id" :      "123",
  "_version" : 4
}
```

即使文　不存在（　Found　是　false　），　_version　　然会　加。　是　Elasticsearch　内部　本的一部分，用来　保　些改　在跨多　点　以正　的　序　行。

NOTE

正如已　在更新整个文　中提到的，　除文　不会立即将文　从磁　中　除，只是将文　已　除状　。随着　不断的索引更多的数据，Elasticsearch　将会在后台清理　已　除的文　。

理冲突

当我　使用 index API 更新文　，可以一次性　取原始文　，做我　的修改，然后重新索引 整个文　。最近的索引　求将　　：无　最后　一个文　被索引，都将被唯一存　在　　Elasticsearch 中。如果其他人同　更改　个文　，他　的更改将　失。

很多　候　是没有　的。也　我　的主数据存　是一个　系型数据　，我　只是将数据　制到 Elasticsearch　中并使其可被搜索。　也　个人同　更改相同的文　的几率很小。或者　于我　的　来　偶　失更改并不是很　重的　。

但有　失了一个　更就是 非常　重的 。　想我　使用 Elasticsearch 存　我　上商城商品　存的数量，　次我　一个商品的　候，我　在 Elasticsearch 中将　存数量　少。

有一天，管理　决定做一次促　。突然地，我　一秒要　好几个商品。　假　有　个　web　程序并行　行，　一个都同　理所有商品的　，如　Consequence of no concurrency control 所示。

Figure 1. Consequence of no concurrency control

web_1　stock_count 所做的更改已　失，因　web_2 不知道它的 stock_count 的拷　已　期。果我　会　有超　商品的　数量的　存，因　客的　存商品并不存在，我　将　他　非常失望。

更越　繁，　数据和更新数据的　隙越　，也就越可能　失　更。

在数据　域中，有　方法通常被用来　保并　更新　更不会　失：

悲　并　控制

方法被　系型数据　广泛使用，它假定有　更冲突可能　生，因此阻塞　源以防止冲突。

一个典型的例子是 取一行数据之前先将其 住， 保只有放置 的 程能 行数据 行修改。

并 控制

Elasticsearch 中使用的 方法假定冲突是不可能 生的，并且不会阻塞正在 的操作。然而，如果源数据在 写当中被修改，更新将会失 。 用程序接下来将决定 如何解决冲突。例如，可以重 更新、使用新的数据、或者将相 情况 告 用 。

并 控制

Elasticsearch 是分布式的。当文 建、更新或 除 ， 新版本的文 必 制到集群中的其他点。Elasticsearch 也是 和并 的， 意味着 些 制 求被并行 送，并且到 目的地 也 序是乱的 。 Elasticsearch 需要一 方法 保文 的旧版本不会覆 新的版本。

当我 之前 index ， GET 和 delete 求 ，我 指出 个文 都有一个 _version （版本）号，当文 被修改 版本号 。 Elasticsearch 使用 个 _version 号来 保 更以正 序得到 行。如果旧版本的文 在新版本之后到 ，它可以被 的忽略。

我 可以利用 _version 号来 保 用中相互冲突的 更不会 致数据 失。我 通 指定想要修改文 的 version 号来 到 个目的。 如果 版本不是当前版本号，我 的 求将会失 。

我 建一个新的博客文章：

```
PUT /website/blog/1/_create
{
  "title": "My first blog entry",
  "text":  "Just trying this out..."
}
```

体告 我 ， 个新 建的文 _version 版本号是 1 。 在假 我 想 个文 ：我 加其数据到 web 表 中， 做一些修改，然后保存新的版本。

首先我 索文 ：

```
GET /website/blog/1
```

体包含相同的 _version 版本号 1 ：

```
{
  "_index" :   "website",
  "_type" :    "blog",
  "_id" :      "1",
  "_version" : 1,
  "found" :    true,
  "_source" :  {
      "title": "My first blog entry",
      "text":  "Just trying this out..."
  }
}
```

在，当我　　通　重建文　的索引来保存修改，我　指定 version　我　的修改会被　用的版本：

```
PUT /website/blog/1?version=1 ①
{
  "title": "My first blog entry",
  "text":  "Starting to get the hang of this..."
}
```

① 我　想　个在我　索引中的文　只有　在的 _version　1　，本次更新才能成功。

此　求成功，并且　　体告　我　_version 已　　到 2 ：

```
{
  "_index":   "website",
  "_type":    "blog",
  "_id":      "1",
  "_version": 2
  "created":  false
}
```

然而，如果我　重新　行相同的索引　求，　然指定 version=1 ，Elasticsearch 返回 409 Conflict HTTP　　，和一个如下所示的　　体：

```
{
   "error": {
      "root_cause": [
         {
            "type": "version_conflict_engine_exception",
            "reason": "[blog][1]: version conflict, current [2], provided [1]",
            "index": "website",
            "shard": "3"
         }
      ],
      "type": "version_conflict_engine_exception",
      "reason": "[blog][1]: version conflict, current [2], provided [1]",
      "index": "website",
      "shard": "3"
   },
   "status": 409
}
```

告 我 在 Elasticsearch 中 个文 的当前 _version 号是 2，但我 指定的更新版本号 1。

我 在 做取决于我 的 用需求。我 可以告 用 其他人已 修改了文 ，并且在再次保存之前 些修改内容。 或者，在之前的商品 stock_count 景，我 可以 取到最新的文 并 重新 用 些修改。

所有文 的更新或 除 API，都可以接受 version 参数， 允 在代 中使用 的并 控制， 是一 明智的做法。

通 外部系 使用版本控制

一个常 的 置是使用其它数据 作 主要的数据存 ，使用 Elasticsearch 做数据 索，意味着主数据 的所有更改 生 都需要被 制到 Elasticsearch ，如果多个 程 一数据同 ，可能遇到 似于之前描述的并 。

如果 的主数据 已 有了版本号 — 或一个能作 版本号的字段 比如 timestamp — 那 就可以在 Elasticsearch 中通 加 version_type=external 到 字符串的方式重用 些相同的版本号，版本号必 是大于零的整数， 且小于 9.2E+18 — 一个 Java 中 long 型的正 。

外部版本号的 理方式和我 之前 的内部版本号的 理方式有些不同， Elasticsearch 不是 当前 version 和 求中指定的版本号是否相同， 而是 当前 _version 是否 _小于 指定的版本号。 如果 求成功，外部的版本号作 文 的新 _version 行存 。

外部版本号不 在索引和 除 求是可以指定，而且在 建 新文 也可以指定。

例如，要 建一个新的具有外部版本号 5 的博客文章，我 可以按以下方法 行：

```
PUT /website/blog/2?version=5&version_type=external
{
  "title": "My first external blog entry",
  "text":  "Starting to get the hang of this..."
}
```

在　中，我　能看到当前的 _version 版本号是 5 ：

```
{
  "_index":   "website",
  "_type":    "blog",
  "_id":      "2",
  "_version": 5,
  "created":  true
}
```

在我　更新　个文　，指定一个新的 version 号是 10 ：

```
PUT /website/blog/2?version=10&version_type=external
{
  "title": "My first external blog entry",
  "text":  "This is a piece of cake..."
}
```

求成功并将当前 _version　10 ：

```
{
  "_index":   "website",
  "_type":    "blog",
  "_id":      "2",
  "_version": 10,
  "created":  false
}
```

如果　要重新　行此　求　，它将会失　，并返回像我　之前看到的同　的冲突　，　因　指定的外部版本号不大于 Elasticsearch 的当前版本号。

文　的部分更新

在 更新整个文　，我　已　介　更新一个文　的方法是　索并修改它，然后重新索引整个文　，　的　如此。然而，使用 update API 我　可以部分更新文　，例如在某个　求　数器　行累加。

我　也介　文　是不可　的：他　不能被修改，只能被替　。 update API 必　遵循同　的　。从外部来看，我　在一个文　的某个位置　行部分更新。然而在内部，update API　使用与之前描述相同的　索-修改-重建索引　的　理　程。　区　在于　个　程　生在分片内部，

就避免了多次 求的 。通 少 索和重建索引 之 的 ，我 也 少了其他 程的更 来冲突的可能性。

update 求最 的一 形式是接收文 的一部分作 doc 的参数， 它只是与 有的文 行合并。 象被合并到一起，覆 有的字段， 加新的字段。 例如，我 加字段 tags 和 views 到我的博客文章，如下所示：

```
POST /website/blog/1/_update
{
   "doc" : {
      "tags" : [ "testing" ],
      "views": 0
   }
}
```

如果 求成功，我 看到 似于 index 求的 ：

```
{
   "_index" :   "website",
   "_id" :      "1",
   "_type" :    "blog",
   "_version" : 3
}
```

索文 示了更新后的 _source 字段：

```
{
   "_index":    "website",
   "_type":     "blog",
   "_id":       "1",
   "_version":  3,
   "found":     true,
   "_source": {
      "title":  "My first blog entry",
      "text":   "Starting to get the hang of this...",
      "tags": [ "testing" ], ①
      "views":  0 ①
   }
}
```

① 新的字段已被添加到 _source 中。

使用脚本部分更新文

脚本可以在 update API中用来改 _source 的字段内容， 它在更新脚本中称 ctx._source 。 例如，我 可以使用脚本来 加博客文章中 views 的数量：

```
POST /website/blog/1/_update
{
   "script" : "ctx._source.views+=1"
}
```

用 Groovy 脚本 程

于那些 API 不能 足需求的情况，Elasticsearch 允 使用脚本 写自定 的 。 多 API都支持脚本的使用，包括搜索、排序、聚合和文 更新。 脚本可以作 求的一部分被 ，从特殊的 .scripts 索引中 索，或者从磁 加 脚本。

的脚本 言 是 Groovy，一 快速表 的脚本 言，在 法上与 JavaScript 似。 它在 Elasticsearch V1.3.0 版本首次引入并 行在 沙 中，然而 Groovy 脚本引 存在漏洞， 允 攻 者通 建 Groovy 脚本，在 Elasticsearch Java VM 行 脱 沙 并 行 shell 命令。

因此，在版本 v1.3.8 、 1.4.3 和 V1.5.0 及更高的版本中，它已 被 禁用。 此外， 可以通 置集群中的所有 点的 `config/elasticsearch.yml` 文件来禁用 Groovy 脚本：

```
script.groovy.sandbox.enabled: false
```

将 Groovy 沙 ，从而防止 Groovy 脚本作 求的一部分被接受， 或者从特殊的 `.scripts` 索引中被 索。当然， 然可以使用存 在 个 点的 `config/scripts/` 目 下的 Groovy 脚本。

如果 的架 和安全性不需要担心漏洞攻 ，例如 的 Elasticsearch 端 暴露和提供 可信 的 用， 当它是 的 用需要的特性 ， 可以 重新 用 脚本。

可以在 {ref}/modules-scripting.html[scripting reference documentation] 取更多于脚本的 料。

我 也可以通 使用脚本 `tags` 数 添加一个新的 。在 个例子中，我 指定新的 作 参数，而不是硬 到脚本内部。 使得 Elasticsearch 可以重用 个脚本，而不是 次我想添加 都要 新脚本重新 ：

```
POST /website/blog/1/_update
{
   "script" : "ctx._source.tags+=new_tag",
   "params" : {
      "new_tag" : "search"
   }
}
```

取文 并 示最后 次 求的效果：

```
{
   "_index":    "website",
   "_type":     "blog",
   "_id":       "1",
   "_version":  5,
   "found":     true,
   "_source": {
      "title":  "My first blog entry",
      "text":   "Starting to get the hang of this...",
      "tags":  ["testing", "search"], ①
      "views":  1 ②
   }
}
```

① search 已追加到 tags 数 中。

② views 字段已 。

我 甚至可以 通 置 ctx.op delete 来 除基于其内容的文 :

```
POST /website/blog/1/_update
{
   "script" : "ctx.op = ctx._source.views == count ? 'delete' : 'none'",
    "params" : {
        "count": 1
    }
}
```

更新的文 可能尚不存在

假 我 需要在 Elasticsearch 中存 一个 面 量 数器。 当有用 , 我 面的 数器 行累加。但是，如果它是一个新 , 我 不能 定 数器已 存在。 如果我 更新一个不存在的文 , 那 更新操作将会失 。

在 的情况下，我 可以使用 upsert 参数，指定如果文 不存在就 先 建它：

```
POST /website/pageviews/1/_update
{
   "script" : "ctx._source.views+=1",
   "upsert": {
       "views": 1
   }
}
```

我 第一次 行 个 求 , upsert 作 新文 被索引，初始化 views 字段 1 。 在后 的 行中，由于文 已 存在， script 更新操作将替代 upsert 行 用， views 数器 行累加。

更新和冲突

在本 的介 中，我 明 索 和 重建索引 的 隔越小， 更冲突的机会越小。但是它并不能完全消除冲突的可能性。 是有可能在 update 法重新索引之前，来自 一 程的 求修改了文 。

了避免数据 失， update API 在 索 索得到文 当前的 version 号，并 版本号到 _重建索引 的 index 求。如果 一个 程修改了 于 索和重新索引 之 的文 ，那 _version 号将不匹配，更新 求将会失 。

于部分更新的很多使用 景，文 已 被改 也没有 系。 例如，如果 个 程都 面 量 数器 行 操作，它 生的先后 序其 不太重要； 如果冲突 生了，我 唯一需要做的就是 再次更新。

可以通 置参数 retry_on_conflict 来自 完成， 个参数 定了失 之前 update 重 的次数，它的 0 。

```
POST /website/pageviews/1/_update?retry_on_conflict=5 ①
{
   "script" : "ctx._source.views+=1",
   "upsert": {
       "views": 0
   }
}
```

① 失 之前重 更新5次。

在 量操作无 序的 景，例如 数器等 个方法十分有效，但是在其他情况下 更的 序 是非常重要的。 似 index API ， update API 采用 最 写入生效 的方案，但它也接受一个 version 参数来允 使用 optimistic concurrency control 指定想要更新文 的版本。

取回多个文

Elasticsearch 的速度已 很快了，但甚至能更快。 将多个 求合并成一个，避免 独 理 个 求花的 延和 。如果 需要从 Elasticsearch 索很多文 ，那 使用 *multi-get* 或者 mget API 来将 些 索 求放在一个 求中，将比逐个文 求更快地 索到全部文 。

mget API 要求有一个 docs 数 作 参数， 个元素包含需要 索文 的元数据，包括 _index 、 _type 和 _id 。如果 想 索一个或者多个特定的字段，那 可以通 _source 参数来指定 些字段的名字：

```
GET /_mget
{
   "docs" : [
      {
         "_index" : "website",
         "_type" :  "blog",
         "_id" :    2
      },
      {
         "_index" : "website",
         "_type" :  "pageviews",
         "_id" :    1,
         "_source": "views"
      }
   ]
}
```

体也包含一个 docs 数 ， 于 一个在 求中指定的文 ， 个数 中都包含有一个 的 ，且 序与 求中的 序相同。 其中的 一个 都和使用 个 get request 求所得到的 体相同：

```
{
   "docs" : [
      {
         "_index" :   "website",
         "_id" :      "2",
         "_type" :    "blog",
         "found" :    true,
         "_source" : {
            "text" :  "This is a piece of cake...",
            "title" : "My first external blog entry"
         },
         "_version" : 10
      },
      {
         "_index" :   "website",
         "_id" :      "1",
         "_type" :    "pageviews",
         "found" :    true,
         "_version" : 2,
         "_source" : {
            "views" : 2
         }
      }
   ]
}
```

如果想 索的数据都在相同的 _index 中（甚至相同的 _type 中）， 可以在 URL 中指定 的 /_index

或者　　的 /_index/_type 。

　　然可以通　　独　求覆　　些　：

```
GET /website/blog/_mget
{
   "docs" : [
      { "_id" : 2 },
      { "_type" : "pageviews", "_id" :   1 }
   ]
}
```

事　上，如果所有文　的 _index 和 _type 都是相同的，　可以只　一个 ids 数　，而不是整个 docs 数　：

```
GET /website/blog/_mget
{
   "ids" : [ "2", "1" ]
}
```

注意，我　　求的第二个文　是不存在的。我　指定　型　 blog ，但是文　 ID 1 的　型是 pageviews ，　个不存在的情况将在　　体中被　告：

```
{
  "docs" : [
    {
      "_index" :   "website",
      "_type" :    "blog",
      "_id" :      "2",
      "_version" : 10,
      "found" :    true,
      "_source" : {
        "title":   "My first external blog entry",
        "text":    "This is a piece of cake..."
      }
    },
    {
      "_index" :   "website",
      "_type" :    "blog",
      "_id" :      "1",
      "found" :    false  ①
    }
  ]
}
```

① 未　到　文　。

事　上第二个文　未能　到并不妨碍第一个文　被　索到。　个文　都是　独　索和　告的。

NOTE	即使有某个文 没有 到，上述 求的 HTTP 状 然是 200 。事 上，即使 求 没有到任何文 ，它的状 依然是 200 --因 mget 求本身已 成功 行。 了定某个文 是成功或者失 ， 需要 found 。

代 小的批量操作

与 mget 可以使我 一次取回多个文 同 的方式， bulk API 允 在 个 中 行多次 create 、 index 、 update 或 delete 求。如果 需要索引一个数据流比如日志事件，它可以排 和索引数百或数千批次。

bulk 与其他的 求体格式 有不同，如下所示：

```
{ action: { metadata }}\n
{ request body        }\n
{ action: { metadata }}\n
{ request body        }\n
...
```

格式 似一个有效的 行 JSON 文 流，它通 行符(\n) 接到一起。注意 个要点：

- 行一定要以 行符(\n) 尾，包括最后一行。 些 行符被用作一个 ，可以有效分隔行。
- 些行不能包含未 的 行符，因 他 将会 解析造成干 。 意味着 个 JSON 不 能使用 pretty 参数打印。

TIP	在 [bulk-format] 中，我 解 什 bulk API 使用 格式。

action/metadata 行指定 一个文 做什 操作。

action 必 是以下 之一:

create
: 如果文 不存在，那 就 建它。 情 建新文 。

index
: 建一个新文 或者替 一个 有的文 。 情 索引文 和 更新整个文 。

update
: 部分更新一个文 。 情 文 的部分更新。

delete
: 除一个文 。 情 除文 。

metadata 指定被索引、 建、更新或者 除的文 的 _index 、 _type 和 _id 。

例如，一个 delete 求看起来是 的：

```
{ "delete": { "_index": "website", "_type": "blog", "_id": "123" }}
```

request body 行由文 的 _source 本身 成—文 包含的字段和 。它是 index 和 create

操作所必需的，　是有道理的：　必　提供文　以索引。

它也是 update 操作所必需的，并且　包含　update API 的相同　求体：doc 、 upsert 、 script 等等。　除操作不需要 request body 行。

```
{ "create":  { "_index": "website", "_type": "blog", "_id": "123" }}
{ "title":    "My first blog post" }
```

如果不指定 _id ，将会自　生成一个 ID ：

```
{ "index": { "_index": "website", "_type": "blog" }}
{ "title":    "My second blog post" }
```

了把所有的操作　合在一起，一个完整的 bulk　求 有以下形式:

```
POST /_bulk
{ "delete": { "_index": "website", "_type": "blog", "_id": "123" }} ①
{ "create": { "_index": "website", "_type": "blog", "_id": "123" }}
{ "title":    "My first blog post" }
{ "index":  { "_index": "website", "_type": "blog" }}
{ "title":    "My second blog post" }
{ "update": { "_index": "website", "_type": "blog", "_id": "123", "_retry_on_conflict"
: 3} }
{ "doc" : {"title" : "My updated blog post"} } ②
```

① 注意 delete　作不能有　求体,它后面跟着的是　外一个操作。

② 　最后一个　行符不要落下。

个 Elasticsearch　包含 items 数　，　个数　的内容是以　求的　序列出来的　个　求的　果。

```
{
   "took": 4,
   "errors": false, ①
   "items": [
      {  "delete": {
            "_index":   "website",
            "_type":    "blog",
            "_id":      "123",
            "_version": 2,
            "status":   200,
            "found":    true
      }},
      {  "create": {
            "_index":   "website",
            "_type":    "blog",
            "_id":      "123",
            "_version": 3,
            "status":   201
      }},
      {  "create": {
            "_index":   "website",
            "_type":    "blog",
            "_id":      "EiwfApScQiiy7TIKFxRCTw",
            "_version": 1,
            "status":   201
      }},
      {  "update": {
            "_index":   "website",
            "_type":    "blog",
            "_id":      "123",
            "_version": 4,
            "status":   200
      }}
   ]
}
```

① 所有的子　求都成功完成。

　个子　求都是独立　行，因此某个子　求的失　不会　其他子　求的成功与否造成影　。
如果其中任何子　求失　，最　　的 error　志被　置　true，并且在相　的　求　告出　　明　：

```
POST /_bulk
{ "create": { "_index": "website", "_type": "blog", "_id": "123" }}
{ "title":    "Cannot create - it already exists" }
{ "index":  { "_index": "website", "_type": "blog", "_id": "123" }}
{ "title":    "But we can update it" }
```

在　　中，我　看到 create 文　123 失　，因　它已　存在。但是随后的 index　求，也是　文　123
操作，就成功了：

```
{
   "took": 3,
   "errors": true, ①
   "items": [
      {  "create": {
            "_index":   "website",
            "_type":    "blog",
            "_id":      "123",
            "status":   409, ②
            "error":    "DocumentAlreadyExistsException ③
                        [[website][4] [blog][123]:
                        document already exists]"
      }},
      {  "index": {
            "_index":   "website",
            "_type":    "blog",
            "_id":      "123",
            "_version": 5,
            "status":   200 ④
      }}
   ]
}
```

① 一个或者多个 求失 。

② 个 求的HTTP状 告 409 CONFLICT 。

③ 解 什 求失 的 信息。

④ 第二个 求成功，返回 HTTP 状 200 OK 。

也意味着 bulk 求不是原子的： 不能用它来 事 控制。 个 求是 独 理的，因此一个求的成功或失 不会影 其他的 求。

不要重 指定Index和Type

也 正在批量索引日志数据到相同的 index 和 type 中。 但 一个文 指定相同的元数据是一 浪 。相反，可以像 mget API 一 ，在 bulk 求的 URL 中接收 的 /_index 或者 /_index/_type ：

```
POST /website/_bulk
{ "index": { "_type": "log" }}
{ "event": "User logged in" }
```

然可以覆 元数据行中的 _index 和 _type ，但是它将使用 URL 中的 些元数据 作 ：

```
POST /website/log/_bulk
{ "index": {}}
{ "event": "User logged in" }
{ "index": { "_type": "blog" }}
{ "title": "Overriding the default type" }
```

多大是太大了？

整个批量 求都需要由接收到 求的 点加 到内存中，因此 求越大，其他 求所能 得的内存就越少。 批量 求的大小有一个最佳 ，大于 个 ，性能将不再提升，甚至会下降。 但是最佳 不是一个固定的 。它完全取决于硬件、文 的大小和 度、索引和搜索的 的整体情况。

幸 的是，很容易 到 个 最佳点 ：通 批量索引典型文 ，并不断 加批量大小 行 。 当性能 始下降，那 的批量大小就太大了。一个好的 法是 始 将 1,000 到 5,000 个文 作 一个批次，如果 的文 非常大，那 就 少批量的文 个数。

密切 注 的批量 求的物理大小往往非常有用，一千个 1KB 的文 是完全不同于一千个 1MB 文 所占的物理大小。 一个好的批量大小在 始 理后所占用的物理大小 5-15 MB。