Transwarp Data Hub v4.3

Transwarp ES使用手册

版本：2.1v

发布日期： 2015-06-11

版本号：T00141-01-021

免责声明

本说明书依据现有信息制作，其内容如有更改，恕不另行通知。星环信息科技（上海）有限公司在编写该说明书的时候已尽最大努力保证期内容准确可靠，但星环信息科技（上海）有限公司不对本说明书中的遗漏、不准确或印刷错误导致的损失和损害承担责任。具体产品使用请以实际使用为准。

注释：Hadoop® 和 SPARK® 是Apache™ 软件基金会在美国和其他国家的商标或注册的商标。

修订历史记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本T00141-01-020 (2015-04) 第一次发布。

文档版本T00140-01-010 (2014-12) 第一次发布。

文档版本T00130-01-010 (2013-12) 第一次发布。

目录

插图清单

范例清单

1. Transwarp ES简介

Transwarp ES基于开源的Elasticsearch并对其进行了优化。Transwarp ES是一个可扩展的分布式全文搜索和分析引擎。在Transwarp Data Hub中，Transwarp ES扮演两个角色：

- 作为Hyperbase全文索引的底层实现。
- 作为一个单独的服务，它是：

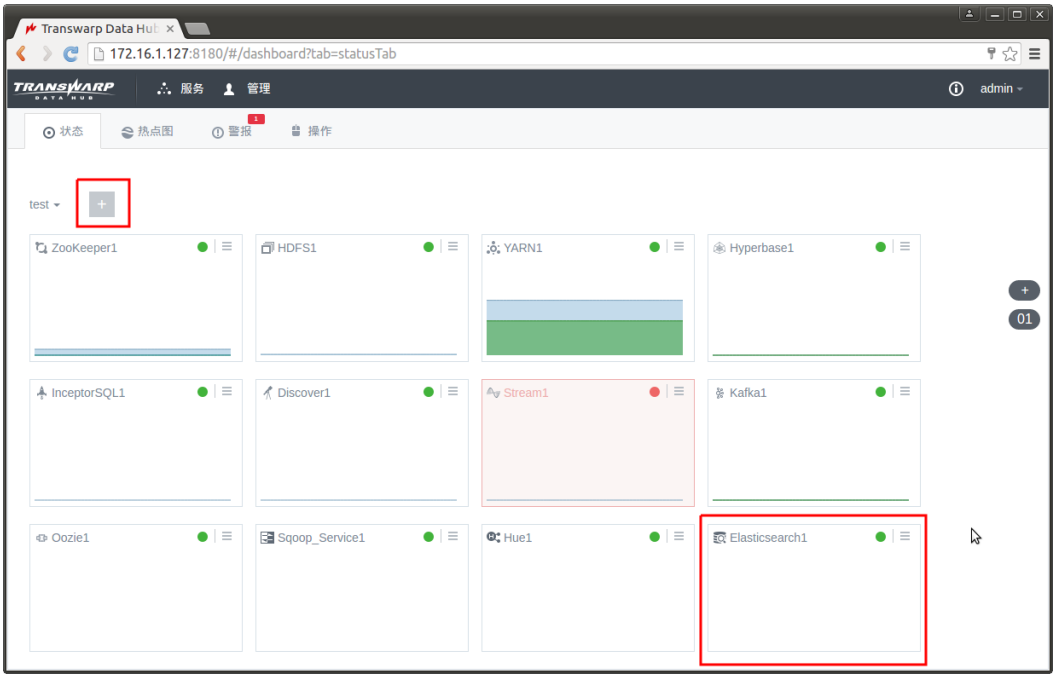
  # 分布式文件存储（Distributed Document Store）；

  # 强大的搜索引擎。常见应用场景有海量数据的存储和搜索、日志分析等。

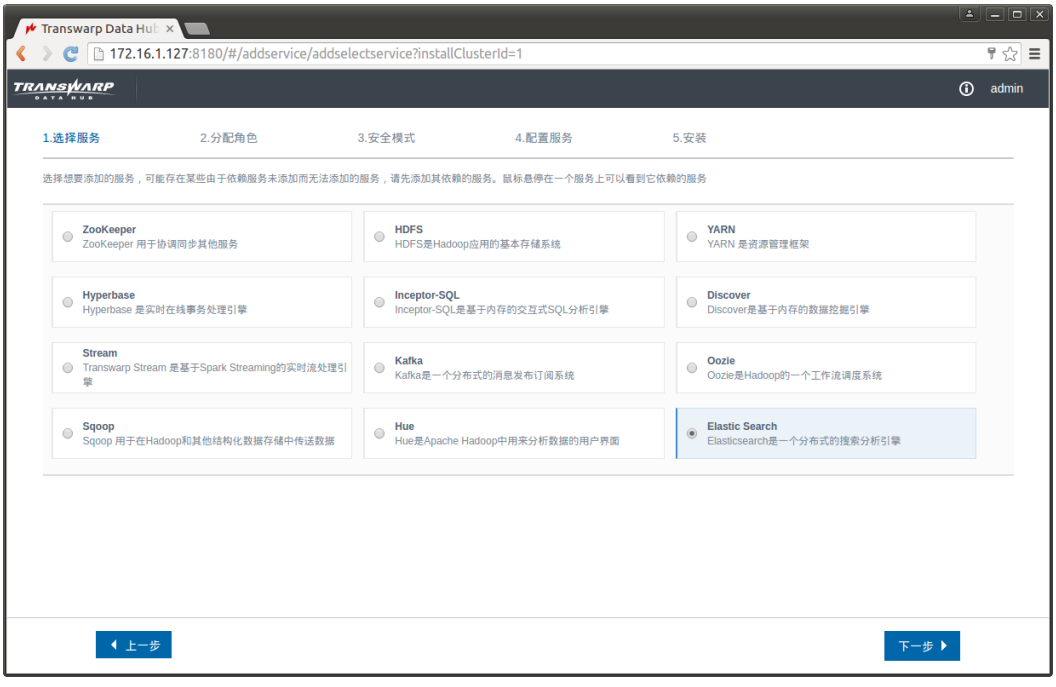
本手册将介绍如何在Transwarp Data Hub平台上将Transwarp ES作为一个单独的服务来使用。

2. 安装ES

要查看您的集群是否已经安装了ES，请登陆集群的Transwarp Manager（用浏览器访问http://<manager_node_ip>:8180）。Transwarp Manager首页如果显示有Elasticsearch，那么您的集群已经安装了一个ES服务：



您也可以点击页面左上的“＋”来安装一个ES服务。Transwarp Manager会打开一个安装向导，您需要在向导中选择Elasticsearch：



接下来，安装向导会全程帮助您安装服务。更多细节请参考《Transwarp Data Hub安装手册》。

3. 了解您的ES服务

Transwarp Manager提供了大量的服务信息，您可以方便地在其中监控和配置您的ES服务。在Transwarp Manager首页点击ES服务，进入服务主页（同时也是角色页面）：

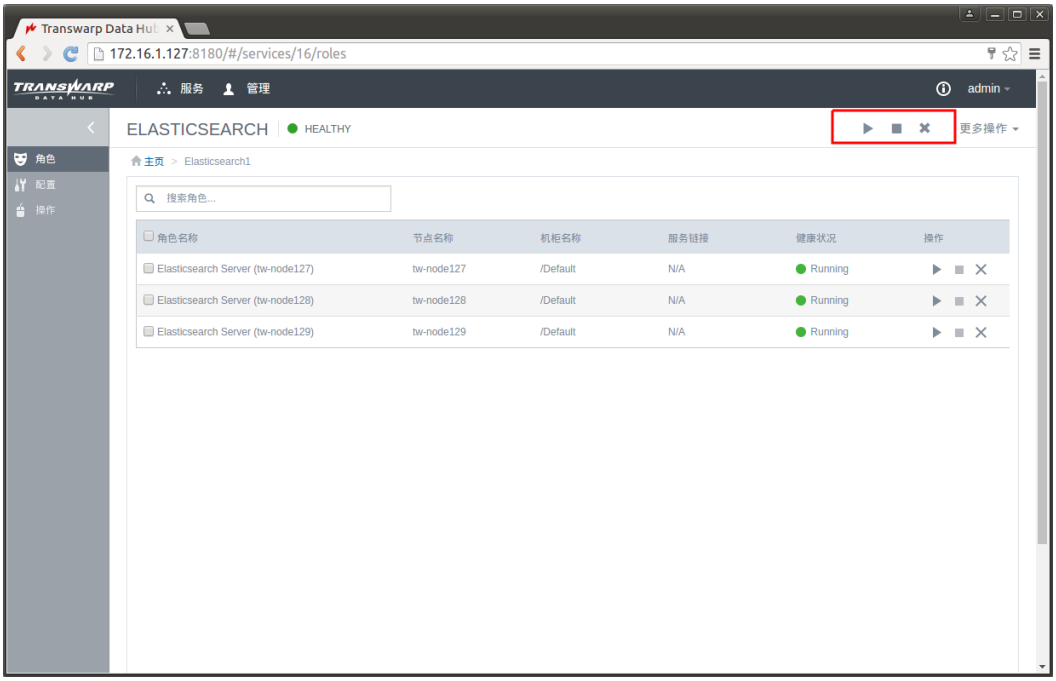


图 3.1. ES服务主页/角色页面

这里您可以看到整个ES服务以及各个节点的健康状态。您可以点击右上角的：

- ▶ 来启动服务；
- ■ 来停止服务；
- ✖ 来删除服务。

点击页面左侧的菜单可以在服务的“角色”、“配置”和“操作”页面之间切换。

3.1. ES服务的角色

服务主页同时也是服务的角色页面。在该页，您可以看到ES集群中的节点（也就是安装了ES的节点）以及它们的健康状况。我们可以看到，这个ES服务中有三个节点，分别在tw-node127, tw-node128和tw-node129上。您可以点击节点对应的 ▶ ， ■ 和 ✖ 来启动、停止和删除节点。

3.2. ES的配置

在ES的配置页面，您可以查看和配置ES的参数。

Transwarp Manager已经为所有参数配置了我们推荐的值，您无须自行配置便可以开始使用ES。只有在当前配置无法满足您的业务需求时，您才需要按需修改这些参数的值。如果目前您还不了解这些参数的具体意义，您可以跳过这段直接进入下一章开始使用。

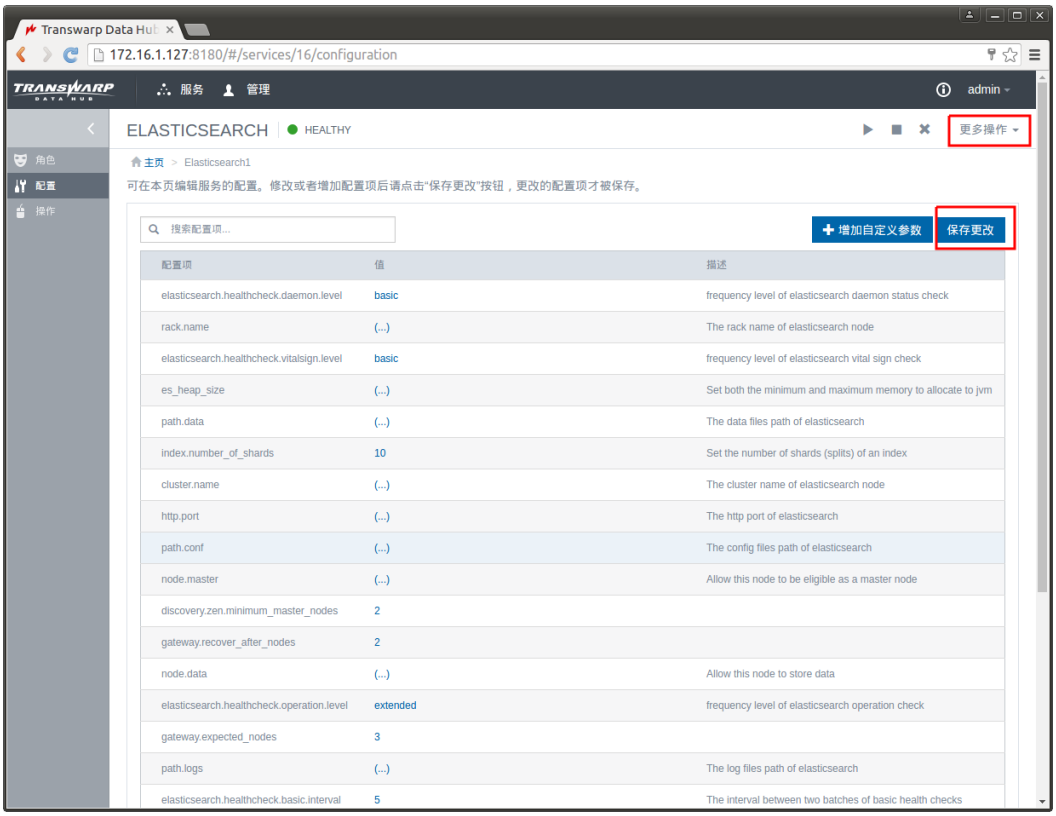


图 3.2. ES的配置页面

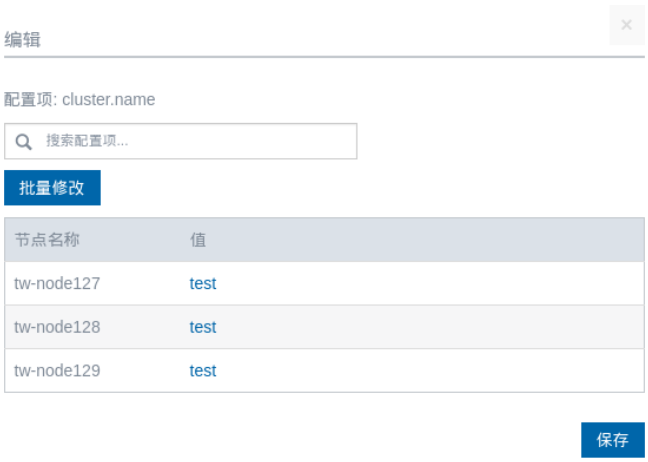
要配置任何一个参数，您都必须进行下面三步：

1. 点击它的值进行修改；
2. 修改完成后，点击配置参数右上方的 保存更改 保存修改；
3. 最后点击页面右上角的 更多操作 ，然后点击弹出的 配置服务 使配置生效。

下面我们介绍一些重要的配置参数，一般情况下使用系统提供的默认值即可。

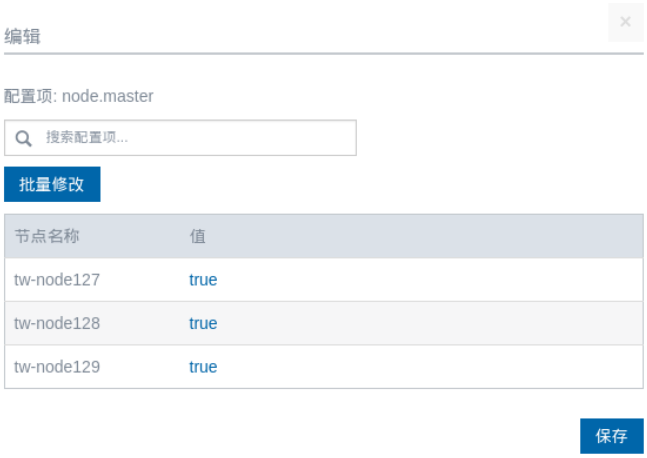
- **cluster.name**：ES的集群名称

  点击cluster.name的值，Transwarp Manager会弹出下面的小窗，供您修改集群中每个节点的名称：



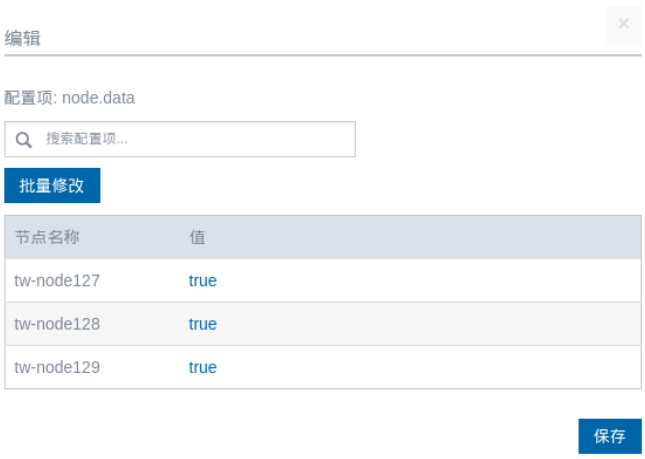
- **node.master**：是否为master node

  点击node.master的值，Transwarp Manager会弹出下面的小窗，您可以查看并修改当前ES集群中各个节点是否为master：



- **node.data**：是否为data node，即是否用于存储数据

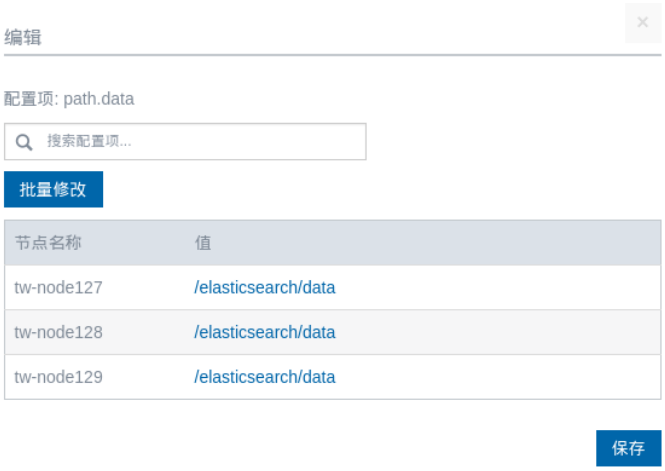
  点击node.data的值，Transwarp Manager会弹出下面的小窗，供您查看并修改当前ES集群中各个节点是否为data node：



- **discovery.zen.ping.unicast.hosts**：使用unicast来发现节点时的host。该参数须列出该ES服务的所有master node。这里列出了tw-node127,tw-node128和tw-node129：

- **discovery.zen.minimum_master_nodes**：节点能加入集群需要连接的最小master数量，为了防止脑裂，推荐配置为n/2+1，n为master数量。例如一个3节点的ES集群推荐配置为2。
- **discovery.zen.ping.multicast.enabled**：是否使用multicast来发现节点。推荐配置为false。
- **index.number_of_shards**：ES index的默认主分片（Primary Shard）数。默认配置为10。
- **index.number_of_replicas**：ES index的每个主分片的副本分片数（Replica Shard）。默认配置为2。
- **path.data**：数据存储路径。

  点击该配置项的值，Transwarp Manager会弹出下面的小窗：



  在这里，您可以为每个节点配置用于存储数据的目录，或者挂载存储数据使用的硬盘。

- **gateway.recover_after_nodes**：触发recovery的节点数量阈值。该配置项需要根据具体的集群规模设置。
- **gateway.expected_nodes**：集群规划的节点数，按具体情况设置。

- **gateway.recover_after_time**：当集群节点数达不到expected_nodes时，可以设置延迟一段时间进行recovery。该参数配置延迟的时间。

4. 和ES交互

ES提供的丰富的REST API用于交互。默认设置下，**ES使用9200端口提供REST API访问**。我们将在命令行中通过 `curl` 来向REST API提出请求，您只需进入一个ES节点的Shell执行我们介绍的指令便可和ES集群交互，这些指令的通用格式为：

通过 `curl` 使用REST API的通用格式

```
curl -X<VERB> 'http://<HOST>:9200/<PATH>/[<API>]/[?<PARAMETERS>]' [-d '<BODY>']
```

- `<VERB>` 为HTTP方法，可以为：`GET`, `POST`, `PUT`, `HEAD` 和 `DELETE`。
- `<HOST>` 为一个运行着ES的服务器的IP或者hostname。
- `<PATH>` 用于指定Index和Type，形式一般为 `<INDEX>/<TYPE>`
- `<API>` 为可选项，用于指定接受请求的REST API。
- `<PARAMETERS>` 为API可以接受的参数，API可以接收多个参数，参数之间用 `&` 隔开：例如 `pretty&q=age:26&size=5`。
- `<BODY>` 为可选项，是一个JSON格式的“请求体”（request body），包含检索请求的细节。

我们可以在实际操作中省去 `http://` 部分，将指令直接写为：

```
curl -X<VERB> '<HOST>:9200/<PATH>/[<API>]/[?<PARAMETERS>]' [-d '<BODY>']
```

另外，您也可以按需使用 `curl` 自带的选项。

第 3 章 了解您的ES服务中我们提到了通过Transwarp Manager查看集群状态，通过REST API也可查看集群状态：

查看集群的health

```
curl -XGET 'localhost:9200/_cat/health?v'
```

输出类似如下：

```
[root@tw-node127 ~]# curl -XGET 'localhost:9200/_cat/health?v'
epoch      timestamp cluster status node.total node.data shards pri relo init unassign
1459347413 22:16:53  test    green           3         3      0   0    0    0        0
```

- `cluster` 下对应的是ES集群名称；
- `status` 下对应的是集群健康状况（`green` 代表健康）；
- `node.total` 下对应的是节点个数；
- `node.data` 下对应的是存储数据的节点的个数。

查看集群节点信息

```
curl -XGET 'localhost:9200/_cat/nodes?v'
```

输出类似如下：

```
[root@tw-node127 ~]# curl -XGET localhost:9200/_cat/nodes?v
host       ip           heap.percent ram.percent load node.role master name
tw-node127 172.16.1.127           20                    d         m      tw-node127
tw-node128 172.16.1.128           21                    d         m      tw-node128
tw-node129 172.16.1.129           10                    d         *      tw-node129
```

为了描述的简洁，除非另外指出，下面我们将直接用HTTP方法名指代执行的指令。例如“PUT”指代“curl -XPUT …”。

5. ES基础概念

本章介绍ES中的几个重要概念：

5.1. ES Index

ES以Index为单位来组织数据（第 5.3 节 “ES Document”），一个Index下的数据常常有相似的特征。例如您可以为员工信息建一个Index，或者为商品信息建一个Index。每个Index都有一个名字用于在操作中指代，Index名必须都为英文小写。

5.2. ES Type

在一个ES Index下，您可以定义一个或多个ES Type。ES Type是ES Index的逻辑上的分类，分类逻辑完全由用户决定。例如存储员工信息的Index可以按部门分类（分为财务部Type、销售部Type、研发部Type等）也可以按办公地点分类（分为北京办公室Type、上海办公室Type、广州办公室Type等）。

5.3. ES Document

ES Document 是ES中最基础的数据单元。例如，员工信息Index中一名员工的信息可以作为一个Document保存；商品信息Index中一件商品的信息也可以作为一个Document保存。ES Document以JSON格式存储，例如：

```
{
    "name":             "Zhang San",
    "age":              26,
    "on_board_date":    "2015-10-31",
    "school":           "Nanjing University"
}
```

虽然物理上Document按Index存储，但是逻辑上Document必须分配给Index下的一个Type。

5.4. 分片（Shard）和副本（Replica）

一个ES Index下可能会有大量的数据，超过硬件的存储能力。ES中，您可以将Index分成多个分片（**shard**），将Index分片有两个作用：

- 横向扩展一个Index的容量；
- 提高计算的并行度从而提升性能。

ES中的分片分为两种：主分片（Primary Shard）和副本分片（Replica Shard，或简称Replica）。Index中的每个Document都属于一个唯一的Primary Shard，所以Primary Shard数量决定了一个Index的容量。Replica Shard则是Primary Shard的拷贝，不仅用于提供数据

冗余，也提供数据读取服务（比如检索请求、Document获取请求等）。Primary Shard数量需要在Index创建时指定，创建后不可修改。一个Index中每个Primary Shard的Replica数则既可以在Index创建时指定，也可以在Index创建后动态修改。**默认设置下**，一个ES Index有10个Primary Shard，每个Primary Shard有2个Replica，所以默认情况下一个ES Index会总共有30个分片。

分片的分布和计算的并行化完全由ES来管理，您作为用户完全无须费心。ES会保证一个节点上相同数据只有一份，也就是说Primary Shard和它自己的Replica永远不会存在一个节点上。所以，如果一个Index的Replica数大于或等于ES集群中节点数量，这个Index中将会有分片无法分配到节点上。

6. ES快速入门

本章我们搭建一个员工信息Index（Index名为 `employee`），并通过这个Index来演示一些常见操作。对这个Index我们将进行如下设计：

- Index下的每个Document对应一名员工的信息。
- Index下的Type按部门分类，分为 `dev`（研发部）、`finance`（财务部）和 `sales`（销售部）。

6.1. 编入Document

在ES术语中，“向Index新增Document”的操作称为将Document“编入”Index，本手册中我们将使用该术语。

目前，employee Index还不存在，但是我们无需专门创建它，只需直接将Document放入（PUT）对应的路径下，路径的形式为 `/<index>/<type>/<id>`，ES会在 `PUT` 时自动创建employee Index。例如，下面我们将员工Zhang San的Document编入 `employee` 下的 `dev`：

编入员工Zhang San的信息

```
curl -XPUT 'localhost:9200/employee/dev/1?pretty' -d '{
    "firstname":        "San",
    "lastname":         "Zhang",
    "age":              26,
    "on_board_date":    "2015-10-31",
    "hometown":         "Beijing",
    "school":           "Nanjing University"
}'
```

路径 `/employee/dev/1` 包含了三条信息：

- `employee` 为Index名字
- `dev` 为Type名字
- `1` 为这条Document的ID。

ES会输出下面信息，说明Document插入成功：

```
{
    "_index":"employee",
    "_type":"dev",
    "_id":"1",
    "_version":1,
    "created":true
}
```

这条信息包含了一个Document的身份元数据（identity metadata）：`_index`、`_type`、`_id` 和 `_version` 分别对应该Document的Index、Type、ID和版本号。ES中每个Document都

有一个版本号，而每次对该Document进行修改（包括更新、插入和删除）都会让版本号加1。`created` 项的值为 `true` 说明该Document是第一次创建。

下面我们再插入两个Document：

```
curl -XPUT 'localhost:9200/employee/dev/2' -d '{
    "firstname":        "Si",
    "lastname":         "Li",
    "age":              28,
    "on_board_date":    "2014-09-16",
    "hometown":         "Nanjing",
    "school":           "Beijing University"
}'

curl -XPUT 'localhost:9200/employee/dev/3' -d '{
    "firstname":        "Wu",
    "lastname":         "Wang",
    "age":              24,
    "on_board_date":    "2016-01-05",
    "hometown":         "Shanghai",
    "school":           "Fudan University"
}'

curl -XPUT 'localhost:9200/employee/sales/4' -d '{
    "firstname":        "Qui",
    "lastname":         "Li",
    "age":              35,
    "on_board_date":    "1205-09-16",
    "hometown":         "unknown",
    "school":           "Home Schooled"
}'
```

6.2. 获取整个Document

要获取一个完整的Document，使用 `GET` 并指定Document的路径，路径的形式为 `/<index>/<type>/<id>`：

获取/employee/dev/1下的Document

```
curl -XGET 'localhost:9200/employee/dev/1?pretty'
```

这里，我们用 `?pretty` 指定了JSON格式化输出。输出如下：

```
{
  "_index" : "employee",
  "_type" : "dev",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source":{
    "firstname":        "San",
    "lastname":         "Zhang",
    "age":              26,
    "on_board_date":    "2015-10-31",
```

```
    "hometown":         "Beijing",
    "school":           "Nanjing University"
}
}
```

我们看到，GET 的结果包含了/employee/dev/1的元数据以及一个 _source 字段，_source 字段中存储的是Document内部的信息。

6.3. 获取部分Document

默认情况下， GET 会打印Document完整的 _source。在 GET 时加上 _source 参数可以获取Document中的指定字段，如果一次指定多个字段，字段名称之间用“,”隔开：

获取/employee/dev/1中的 name 和 age 字段

```
curl -XGET 'localhost:9200/employee/dev/1?_source=firstname,lastname,age&pretty'
```

ES会只输出name和age字段：

```
{
  "_index" : "employee",
  "_type" : "dev",
  "_id" : "1",
  "_version" : 1,
  "found" : true,
  "_source":{"age":26,"lastname":"Zhang","firstname":"San"}
}
```

如果不想ES输出元数据，可以将 _source 作为一个endpoint使用：

只输出 _source

```
curl -XGET 'localhost:9200/employee/dev/1/_source?pretty'
```

ES将只输出 _source 中的数据：

```
{
    "firstname":        "San",
    "lastname":         "Zhang",
    "age":              26,
    "on_board_date":    "2015-10-31",
    "hometown":         "Beijing",
    "school":           "Nanjing University"
}
```

6.4. 查看Document是否存在

查看某个路径下的Document是否存在使用 HEAD，curl 要加上 -i 选项打印HTTP header：

查看/employee/dev/1下是否存在Document

```
curl -i -XHEAD 'localhost:9200/employee/dev/1'
```

输出为：

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

❶ `200 OK` 说明该Document存在。

6.5. 删除Document

删除指定路径下的Document使用 `XDELETE`：

删除/employee/dev/1下的Document

```
curl -XDELETE 'localhost:9200/employee/dev/1?pretty'
```

ES会输出下面信息，说明删除成功：

```
{
    "found":true,
    "_index":"employee",
    "_type":"dev",
    "_id":"1",
    "_version":2
}
```

注意，删除操作让版本号变为了2。

我们可以再一次用 `XHEAD` 查看/employee/dev/1下是否存在Document，ES会返回 `404 Not Found`：

```
curl -i -XHEAD 'localhost:9200/employee/dev/1'
HTTP/1.1 404 Not Found
Content-Type: text/plain; charset=UTF-8
Content-Length: 0
```

6.6. 更新Document

ES中的Document不可更改，只能 **更新**。操作和编入Document相同——对想要更新的Document路径执行 `PUT`，ES会将路径下的Document更新并对版本号加1。下面我们对/employee/dev/2下的Document进行更新（将age从原来的28改为30）：

更新/employee/dev/2下的Document

```
curl -XPUT 'localhost:9200/employee/dev/2' -d '{
    "firstname":        "Si",
    "lastname":         "Li",
    "age":              30,
    "on_board_date":    "2014-09-16",
    "hometown":         "Nanjing",
```

```
    "school":           "Beijing University"
}'
```

ES会输出下面信息，注意版本号变成了2。另外，`created` 值为false说明这次 `PUT` 并没有新建该Document。

```
{
  "_index" : "employee",
  "_type" : "dev",
  "_id" : "2",
  "_version" : 2,
  "created" : false
}
```

6.7. 新建一个Document

我们看到 `PUT` 可能新建一个Document（例如第 6.1 节 “编入Document”中介绍的），也可能覆盖原有的Document（例如第 6.6 节 “更新Document”中介绍的）。那么如何确保我们在 `PUT` 的时候不会覆盖已有的Document呢？ 特定的/<index>/<type>下的Document靠他们的ID区分，所以我们需要确保新增的Document的ID是唯一的即可，使用 `POST` 可以让ES为新增的Document自动生成一个唯一的ID：

向/employee/sales下编入一个新Document

```
curl -XPOST 'localhost:9200/employee/sales?pretty' -d '{
    "firstname":        "Lei",
    "lastname":         "Li",
    "age":              28,
    "on_board_date":    "2013-10-03",
    "hometown":         "Hangzhou",
    "school":           "Zhejiang University"
}'
```

注意，使用 `POST` 只需给出/<index>/<type>/。ES的输出如下：

```
{
  "_index" : "employee",
  "_type" : "sales",
  "_id" : "aKWYjab5Se-nt7gyYNJsGg", ❶
  "_version" : 1,
  "created" : true
}
```

❶ ES自动生成的Document ID。

6.8. 轻量检索

在第 6.2 节 “获取整个Document”和第 6.3 节 “获取部分Document”中，我们看到 `GET` 可以获取Document信息。使用 `GET` 是将 `_search` 作为endpoint可以进行检索。下面我们执行一个最简单的检索：

简单检索

```
curl -XGET 'localhost:9200/employee/dev/_search?pretty'
```

这个检索指令会将/employee/dev/下所有的Document返回：

```
{
  "took" : 64,
  "timed_out" : false,
  "_shards" : {
    "total" : 10,
    "successful" : 10,
    "failed" : 0
  },
  "hits" : {
    "total" : 3,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "employee",
      "_type" : "dev",
      "_id" : "3",
      "_score" : 1.0,
      "_source":{
    "firstname":        "Wu",
    "lastname":         "Wang",
    "age":              24,
    "on_board_date":    "2016-01-05",
    "hometown":         "Shanghai",
    "school":           "Fudan University"
}
    }, {
      "_index" : "employee",
      "_type" : "dev",
      "_id" : "1",
      "_score" : 1.0,
      "_source":{
    "firstname":        "San",
    "lastname":         "Zhang",
    "age":              26,
    "on_board_date":    "2015-10-31",
    "hometown":         "Beijing",
    "school":           "Nanjing University"
}
    }, {
      "_index" : "employee",
      "_type" : "dev",
      "_id" : "2",
      "_score" : 1.0,
      "_source":{
    "firstname":        "Si",
    "lastname":         "Li",
    "age":              30,
    "on_board_date":    "2014-09-16",
    "hometown":         "Nanjing",
    "school":           "Beijing University"
}
    } ]
```

```
  }
}
```

轻量检索 指通过在 GET 请求的URI中直接提供检索串（query-string）来进行检索，可以用于快速简单的检索请求。例如：

在employee Index中查找firstname为Si的Document

```
curl -XGET 'localhost:9200/employee/_search?pretty&q=firstname:Si'
```

ES的输出为：

```
{
  "took" : 26,
  "timed_out" : false,
  "_shards" : {
    "total" : 10,
    "successful" : 10,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "employee",
      "_type" : "dev",
      "_id" : "2",
      "_score" : 1.0,
      "_source":{
    "firstname":        "Si",
    "lastname":         "Li",
    "age":              30,
    "on_board_date":    "2014-09-16",
    "hometown":         "Nanjing",
    "school":           "Beijing University"
}
    } ]
  }
}
```

用类似的请求我们可以检索/employee/dev下age为26的员工信息：

检索age为26的员工信息

```
curl -XGET 'localhost:9200/employee/dev/_search?pretty&q=age:26'
```

ES的输出为：

```
{
  "took" : 33,
  "timed_out" : false,
  "_shards" : {
    "total" : 10,
    "successful" : 10,
```

```
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "employee",
      "_type" : "dev",
      "_id" : "1",
      "_score" : 1.0,
      "_source":{
    "firstname":        "San",
    "lastname":         "Zhang",
    "age":              26,
    "on_board_date":    "2015-10-31",
    "hometown":         "Beijing",
    "school":           "Nanjing University"
}
    } ]
  }
}
```

6.9. 总结

希望上述操作对您快速熟悉ES的功能和操作有帮助。这些操作仅仅是ES强大功能的冰山一角。我们会在下面的内容中介绍更多的ES REST API的使用。

7. API使用约定

除非特别指出，本文档都将采用如下约定。

7.1. 多Index的使用

大多数使用Index参数的API都可以跨Index执行，指定Index时只需依次列出Index并将它们用逗号隔开，例如：

```
index1, index2, index3
```

指定Index时也可以使用通配符“*”和“.”，例如：

```
foo*, fooba.
```

Document API不支持跨Index执行。

7.2. 输出格式

在通过 `curl` 使用REST API的通用格式中的 `<PARAMETERS>` 部分加上 `pretty` 或者 `pretty=true`，ES将以可读性较高的方式输出结果。

8. Cluster级别API

本章介绍ES的集群（Cluster）级别的API，Cluster级别API包括Cluster API和Nodes API。这些API的使用格式为：

语法：Cluster API使用格式

```
curl -X<VERB> '<HOST>/_cluster/<API>?[<PARAMETERS>]' [-d '<BODY>']
```

或者

语法：Nodes API使用格式

```
curl -X<VERB> '<HOST>/_nodes/<API>?[<PARAMETERS>]' [-d '<BODY>']
```

8.1. Cluster health API

Cluster health API用于查看集群和Index的健康状况。ES中健康状况分为三种：green, yellow和red。在Shard级别：

- 如果健康状况为red，说明无法在集群中找到该Shard。
- 如果健康状况为yellow，说明该Shard可以在集群中找到，但是它的Replica找不到。
- 如果健康状况为green，则Shard和它的Replica都能找到。

Index的健康状况由Index下最差的Shard健康状况决定；集群的健康状况由集群下最差的Index健康状况决定。

语法：查看整个集群的health

```
curl -XGET '<HOST>/_cluster/health?pretty'
```

输出类似于：

```
{
  "cluster_name" : "test",
  "status" : "green",
  "timed_out" : false,
  "number_of_nodes" : 3,
  "number_of_data_nodes" : 3,
  "active_primary_shards" : 30,
  "active_shards" : 90,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 0
}
```

health 后可以添加Index名字来查看指定一个或多个Index的健康状况。Index之间用“,”隔开：

查看指定Index的健康状况

```
curl -XGET '<HOST>/_cluster/health/<index>,<index>, ...?pretty'
```

例 8.1. 查看名为employee和inventory的Index的健康状况

```
curl -XGET '<HOST>/_cluster/health/employee,inventory?pretty'
```

8.1.1. Query String选项

Cluster `health` API在 `<PARAMETERS>` 部分可以使用的选项有：

- `level`：值可以为 `cluster`（集群级别）、`indices`（Index级别）和 `shards`（Shard级别），控制返回的健康状况的级别。默认值为 `cluster`。例如：

  语法：查看集群中所有Index的健康状况

  ```
  curl -XGET '<HOST>/_cluster/health?level=indices&pretty'
  ```

8.2. Cluster `stats` API

Cluster `stats` API用于获取整个集群的统计数据，包括基础的Index指标（Shard数量、存储大小、内存使用量）和节点指标（节点数量、角色、计算资源使用状况等等）。

语法：查看Cluster Stats

```
curl -XGET '<HOST>/_cluster/stats?pretty'
```

8.3. Cluster `settings` API

Cluster `settings` API可以用于查看（`GET`）或进行（`PUT`）集群的设置。

语法：查看整个集群的Settings

```
curl -XGET '<HOST>/_cluster/settings?pretty'
```

语法：设置批量插入的队列大小

```
curl -XPUT '<HOST>/_cluster/settings' -d '{
    "persistent": {
        "threadpool.bulk.queue_size" : 1000
    }
}'
```

8.4. Nodes `stats` API

Nodes `stats` API用于查看节点的统计数据。

语法：查看所有节点的统计数据

```
curl -XGET '<HOST>/_nodes/stats?pretty'
```

还可以通过nodeID来查看指定节点的统计数据：

语法：查看指定节点的统计数据

```
curl -XGET '<HOST>/_nodes/<nodeID>,<nodeID>,.../stats?pretty'
```

例 8.2. 查看tw-node127和tw-node128节点的统计数据

```
curl -XGET 'localhost:9200/_nodes/tw-node127,tw-node128/stats?pretty'
```

9. Index级别API

Index级别API用于管理一个或多个Index。Index级别的API使用格式为：

语法：Index级别API使用格式

```
curl -X<VERB> '<HOST>/<index>[, <index>]/[_<API>]?[<PARAMETERS>]' [-d '<BODY>']
```

其中，<index>[, <index>] 用于指定一个或多个Index，Index之间用“,”隔开。

9.1. 创建Index

创建Index执行 PUT。

语法：创建Index

```
curl -XPUT '<HOST>/<index>/?pretty'
```

例 9.1. 创建名为test的Index

```
curl -XPUT 'localhost:9200/test/?pretty'
```

输出：

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

9.2. 删除Index

删除Index执行 DELETE。

语法：删除指定Index

```
curl -XDELETE '<HOST>/<index>,<index>,.../?pretty'
```

例 9.2. 删除名为test的Index

```
curl -XDELETE 'localhost:9200/test/?pretty'
```

9.3. 查看Index是否存在

查看Index是否存在使用 HEAD：

语法：查看指定Index是否存在

```
curl -i -XHEAD '<HOST>/<index>'
```

9.4. Index settings API

通过Index settings API可以查看（GET）或进行（PUT）一个或多个Index的设置。

语法：查看指定Index的设置

```
curl -XGET '<HOST>/<index>, <index>, .../_settings?pretty'
```

例 9.3. 查看名为employee的Index的设置

```
curl -XGET 'localhost:9200/employee/_settings?pretty'
```

语法：设置副本数

```
curl -XPUT '<HOST>/<index>,<index>, .../_settings?pretty' -d '{ "number_of_replicas":
 1
}'
```

语法：设置更新的interval

```
curl -XPUT '<HOST>/<index>,<index>, .../_settings?pretty' -d  '{
    "index" : {
        "refresh_interval" :  30000
    }
}'
```

9.5. Index optimize API

通过 optimize API可以优化一个或多个指定的Index。对Index优化可以加快检索操作。optimize API通过合并segments来减少segment数量。

语法：合并segments

```
curl -XPOST '<HOST>/<index>,<index>, .../_optimize'
```

9.5.1. 接受的请求参数

optimize API接收的请求参数包括：

- max_num_segments：指定最大segment数量，这将是优化的目标segment数量。

例 9.4. 将名为test的Index的 max_num_segments 设为3

```
curl -XPOST 'localhost:9200/test/_optimize?max_num_segments=3&pretty'
```

9.6. Index mapping API

通过 mapping API可以进行Index Mapping的获取（GET）和设置（PUT）。

语法：获取Index Mapping

```
curl -XGET '<HOST>/<index>,<index>,.../_mapping?pretty'
```

语法：为Index下指定Type设置Mapping

```
curl -XPUT '<HOST>/<index>/_mapping/<type>' -d '<BODY>'
```

Mapping必须在Type创建时就指定。如果不指定Mapping，ES会自动根据Type下Document中的数据来判断并设置好Mapping，这个过程称为动态Mapping（Dynamic Mapping）。Mapping一旦设置就不能再更改。

10. Document API

通过Document API我们可以对ES Document进行读写操作。每个ES Document由唯一的/<index>/<type>/<id>组合确定。所以单个Document API的使用格式为：

单个Document API的使用格式

```
curl -X<VERB> '<HOST>/<index>/<type>/<id>[/_<API>][?<PARAMETERS>]' [-d '<BODY>']
```

10.1. 编入API

编入Document使用 PUT（用户指定Document ID）或者 POST（ES自动生成ID）。

语法：编入Document，用户指定Document ID

```
curl -XPUT '<HOST>/<index>/<type>/<id>' -d '{<document_body>}'
```

注意，该语法也可以用于 **更新整个Document。举例：** 见编入员工Zhang San的信息和更新/employee/dev/2下的Document。

语法：编入Document，ES自动生成ID

```
curl -XPOST '<HOST>/<index>/<type>' -d '{<document_body>}'
```

举例： 见向/employee/sales下编入一个新Document。

10.2. 获取API

获取Document使用 GET。

语法：获取整个Document

```
curl -XGET '<HOST>/<index>/<type>/<id>?pretty'
```

举例： 见获取/employee/dev/1下的Document。

语法：获取Document中的某个字段

```
curl -XGET '<HOST>/<index>/<type>/<id>?_source=<field>,<field>,...&pretty'
```

举例： 见获取/employee/dev/1中的 name 和 age 字段。

10.3. 删除API

删除Document使用 DELETE。

语法：删除一个Document

```
curl -XDELETE '<HOST>/<index>/<type>/<id>'
```

举例： 见删除/employee/dev/1下的Document。

11. ES检索

到目前为止，我们只介绍了ES作为分布式文件存储的功能。然而，ES真正的强大之处在于它的检索功能。ES为Document中的每一个字段都建索引，让Document中的每一个字段都可以被查询。ES提供 `_search` API用于接受检索请求，本章将简单介绍 `_search` API的一部分使用方法，后面的章节还会涉及更多内容。

11.1. 空检索

ES中最简单的检索是空检索（empty search），即不指定任何检索条件，要求ES返回所有Index下的所有Document。我们利用空检索解释一些ES对检索返回的信息。下面的检索输出为了文档的简洁，进行了删减：

空检索

```
curl -XGET 'localhost:9200/_search?pretty'
{
  "took" : 118, ❶
  "timed_out" : false, ❷
  "_shards" : { ❸
    "total" : 50,
    "successful" : 50,
    "failed" : 0
  },
  "hits" : { ❹
    "total" : 1020,
    "max_score" : 1.0,
    "hits" : [ {
      "_index" : "bank",
      "_type" : "account",
      "_id" : "1",
      "_score" : 1.0,
      "_source":
{"account_number":1,"balance":39225,"firstname":"Amber","lastname":"Duke","age":32,"gender":"M","ad
 Holmes
 Lane","employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state":"IL"}
    },
    ... 为简洁删减了其他9条返回记录 ... ❺
    ]
  }
}
```

❶ `took`：本次检索所花时间（单位毫秒）。

❷ `timed_out`：本次检索是否超时。

❸ `_shards`：所有本次检索涉及的分片数，分为所有涉及的（`total`）、成功的（`successful`）和失败的（`failed`）。保存相同数据的分片（主分片和它的副本分片）算作同一个分片。所以只有当某一主分片和它所有的副本分片都无法响应检索才会出现 `failed` 的情况。

❹ hits：实际的检索结果，是返回内容中最重要的信息。我们将在下面讨论。
❺ 默认情况下，一次查询只会返回前10条结果。这里为了行文简洁，我们删减了其他9条。

我们只看上一节中空检索中的 hits 部分：

hits 字段

```
"hits" : {
    "total" : 1020, ❶
    "max_score" : 1.0, ❷
    "hits" : [ {  ❸
      "_index" : "bank",
      "_type" : "account",
      "_id" : "1",
      "_score" : 1.0,
      "_source":
{"account_number":1,"balance":39225,"firstname":"Amber","lastname":"Duke","age":32,"gender":"M","ad
 Holmes
 Lane","employer":"Pyrami","email":"amberduke@pyrami.com","city":"Brogan","state":"IL"}
    },
    ... 为简洁删减了其他9条返回记录 ...
    ]
  }
```

❶ total：所有符合条件的Document数量。
❷ max_score：所有符合条件Document的 _score 中的最大值。
❸ hits：这里的 hits 是一组信息，组中元素分别是匹配的Document的Index, Type, ID, 关联分数（_score）和Document内数据（_source）。

11.2. 检索请求的格式

更多时候，我们需要指定检索条件。通常，检索请求将和下面相似：

查询的基本格式

```
curl (-XGET|-XPOST) '<HOST>:9200/<PATH>/_search?<PARAMETERS>' -d '<BODY>'
```

其中，<PATH> 用于指定检索的Index和Type。ES支持同时跨Index和跨Type检索。例如：

在employee Index下的所有Type中检索

```
curl -XGET 'localhost:9200/employee/_search'
```

在employee Index下的dev和sales Type中检索

```
curl -XGET 'localhost:9200/employee/dev,sales/_search'
```

在employee, new_employee这两个Index下的dev和sales Type中检索

```
curl -XGET 'localhost:9200/new_employee,employee/dev,sales/_search'
```

在所有Index中的dev和sales Type中检索

```
curl -XGET 'localhost:9200/_all/dev,sales/_search'
```

ES提供两种使用 `_search` API的方法：URI检索和Request Body检索。 `GET` 和 `POST` 都可以用于检索请求，两者的返回信息没有区别。有的客户端不支持在 `GET` 时使用Request Body，这时就必须使用 `POST`。

- **URI检索** 将查询放在 `<PARAMETERS>` 中，省掉 `-d '<BODY>'` 部分。这种形式的检索又称为“轻量检索（Search Lite）”，适合短小的即席查询。例如：

  ```
  curl -XGET 'localhost:9200/employee/dev/_search?pretty&q=lastname:Li'
  ```

  URI检索的细节将在???中展开介绍。

- **Request Body检索** 将查询放在 `<BODY>` 中，例如：

  ```
  curl -XPOST 'localhost:9200/employee/dev/_search?pretty' -d '
  {
      "query" : {
          "match_phrase" : {
              "lastname" : "Li"
          }
      }
  }
  '
  ```

  Request Body检索的细节将在???中展开介绍。

客户服务

技术支持

感谢你使用星环信息科技（上海）有限公司的产品和服务。如您在产品使用或服务中有任何技术问题，可以通过以下途径找到我们的技术人员给予解答。

email: support@transwarp.io

技术支持热线电话：18930357653

官方网址 ：www.transwarp.io

意见反馈

如果你在系统安装，配置和使用中发现任何产品问题，可以通过以下方式反馈：

email: support@transwarp.io

感谢你的支持和反馈，我们一直在努力！

关于我们：

星环信息科技（上海）有限公司是一家大数据领域的高科技公司，致力于大数据基础软件的研发。星环科技目前掌握的企业级Hadoop和Spark核心技术在国内独树一帜，其产品Transwarp Data Hub (TDH)的整体架构及功能特性堪比硅谷同行，在业界居于领先水平，性能大幅领先Apache Hadoop，可处理从GB到PB级别的数据。星环科技的核心开发团队参与部署了国内最早的Hadoop集群，并在中国的电信、金融、交通、政府等领域的落地应用拥有丰富经验，是中国大数据核心技术企业化应用的开拓者和实践者。星环科技同时提供存储、分析和挖掘大数据的高效数据平台 和服务，立志成为国内外领先的大数据核心技术厂商。

行业地位：

来自知名外企的创业团队，成功完成近千万美元的A轮融资，经验丰富的企业级Hadoop发行版开发团队，国内最多落地案例。

核心技术：

高性能、完善的SQL on Hadoop、R语言的并行化支持，为企业数据分析与挖掘提供优秀选择。

应用案例：

已成功部署多个关键行业领域，包括电信、电力、智能交通、工商管理、税务、金融、广电、电商、物流等。

地址：上海市徐汇区桂平路481号18幢3层301室（漕河泾新兴技术开发区）
邮编：200233
电话：4008-079-976
网址：www.transwarp.io