目录

§1 ElasticSearch

Elasticsearch 是个开源分布式搜索引擎，它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，Restful 风格接口，多数据源，自动搜索负载等。

§1.1 What's a mapping?

A mapping not only tells ES what is in a field…it tells ES what terms are indexed and searchable.

A mapping is composed of one or more ‘analyzers’（相当于 Lucene 中的 Analyzer）, which are in turn built with one or more ‘filters’. When ES indexes your document, it passes the field into each specified analyzer, which pass the field to individual filters.

Filters are easy to understand: a filter is a function that transforms data. Given a string, it returns another string (after some modifications). A function that converts strings to lowercase is a good example of a filter.

An analyzer is simply a group of filters that are executed in-order. So an analyzer may first apply a lowercase filter, then apply a stop-word removal filter(中文分词作为其中一个 filter). Once the analyzer is run, the remaining terms are indexed and stored in ES. 因此: a mapping is simply a series of instructions on how to transform your input data into searchable, indexed terms.

使用 **curl -XPUT http://localhost:9200/test/item/1 -d '{"name":"zach", "description": "A Pretty cool guy."}'** 时，ES 生成如下的 mapping:

```
mappings: {
   item: {
      properties: {
         description: { type: string }
         name: { type: string }
      }
   }
}
```

ES 猜测到"description" 这列的类型为"string",When ES implicitly creates a "string" mapping, it applies the default global analyzer. 除非显式地更改 analyzer 设置, 否则 ES 将使用默认的 Standard

Analyzer.Standard Analyzer will apply the standard token filter, lowercase filter and stop token filter to your field. 这种情况下，句子末尾的. 号和字符'A' 将被丢弃.Importantly, even though ES continues to store the original document in it’ s original form, the only parts that are searchable are the parts that have been run through an analyzer.So, mappings are not data-types , think of them as instructions on how you will eventually search your data. If you care about stop-words like 'a', you need to change the analyzer so that they aren’ t removed.

§1.2 Constructing more complicated mapping

Setting up proper analyzers in ES is all about thinking about the search query. You have to provide instructions to ES about the appropriate transformations so you can search intelligently.

The first thing that happens to an input query is tokenization , breaking an input query into smaller chunks called tokens. There are several tokenizers available, which you should explore on your own when you get a chance. The Standard tokenizer is being used in this example, which is a pretty good tokenizer for most English-language search problems. You can query ES to see how it tokenizes a sample sentence:

```
curl -X GET "http://localhost:9200/test/_analyze?tokenizer=standard&pretty=true"
-d 'The quick brown fox is jumping over the lazy dog.'
```

Ok, so our input query has been turned into tokens. Referring back to the mapping, the next step is to apply filters to these tokens. In order, these filters are applied to each token: Standard Token Filter, Lowercase Filter, ASCII Folding Filter.

```
curl -X GET "http://localhost:9200/test/_analyze?filter=standard&pretty=true"
-d 'The quick brown fox is jumping over the lazy dog.'
```

As you can see, we specify both a search and index analyzer. These two separate analyzers instruct ES what to do when it is indexing a field, or searching a field. But why are these needed?

```
"partial":{
     "search_analyzer":"full_name",
     "index_analyzer":"partial_name",
     "type":"string"
}
```

The index analyzer is easy to understand. We want to break up our input fields into various tokens so we can later search it. So we instruct ES to use the new partial_name analyzer that we built, so that it can create nGrams for us.

The search analyzer is a little trickier to understand, but crucial to getting good relevance. Imagine querying for “Race” . We want that query to match “race” , “races” and “racecar” . When searching, we want to make sure ES eventually searches with the token “race” . The full_name analyzer will give us the needed token to search with.

If, however, we used the partial_name nGram analyzer, we would generate a list of nGrams as our search query. The search query “Race” would turn into ["ra", "rac", "race"].Those tokens are then used to search the index. As you might guess, “ra” and “rac” will match a lot of things you don’ t want, such as “racket” or “ratify” or “rapport” .

So specifying different index and search analyzers is critical when working with things like ngrams. Make sure you always double check what you expect to query ES with…and what is actually being passed to ES.

§1.3 Depth into ElasticSearch

当使用如下命令搜索时:

```
curl -XPOST "http://namenode:9200/_search" -d'
{
    "query": {
      "match_all" : {}
    },
    "filter" : {
      "term" : { "director" : "Francis Ford Coppola"}
    }
}'
```

没有任何结果，而使用

```
curl -XPOST "http://namenode:9200/_search" -d'
{
    "query": {
      "match_all" : {}
    },
    "filter" : {
      "term" : { "year" : "1962"}
    }
}'
```

却能输出结果，What's going on here? We've obviously indexed two movies with "Francis Ford Coppola" as director and that's what we see in search results as well. Well, while ES has a JSON object with that data that it returns to us in search results in the form of the _source property .

When we index a document with ElasticSearch it (simplified) does two things: it stores the original data untouched for later retrieval in the form of _source and it indexes each JSON property into one or more fields in a Lucene index. During the indexing it processes each field according to how the field is mapped. If it isn't mapped , default mappings depending on the fields type (string, number etc) is used.

As we haven't supplied any mappings for our index, ElasticSearch uses the default mappings for the director field. This means that in the index the director fields value isn't **"Francis Ford Coppola"**. Instead it's something like **["francis", "ford", "coppola"]**. We can verify that by modifying our filter to instead match "francis" (or "ford" or "coppola").

So, what to do if we want to filter by the exact name of the director? We modify how it's mapped. There are a number of ways to add mappings to ElasticSearch, through a HTTP request that creates by calling the _mapping endpoint. Using this approach, we could fix the above issue by adding a mapping for the "director" field , to instruct ElasticSearch not to analyze (tokenize etc.) the field at all.

```
curl -X PUT namenode:9200/movies/movie/_mapping -d'
{
   "movie": {
      "properties": {
         "director": {
            "type": "string",
            "index": "not_analyzed"
        }
      }
   }
}'
```

In many cases it's not possible to modify existing mappings. Often the easiest work around for that is to create a new index with the desired mappings and re-index all of the data into the new index. The second problem is that, even if we could add it, we would have limited our ability to search in the director field. That is, while a search for the exact value in the field would match we wouldn't be able to search for single words in the field.

幸运地是存在更简单的办法。We add a mapping that upgrades the field to a multi field. What that means is that we'll map the field multiple times for indexing. Given that one of the ways we map it match the existing mapping both by name and settings that will work fine and we won't have to create a new index.

```
curl -XPUT "namenode:9200/movies/movie/_mapping" -d'
{
   "movie": {
      "properties": {
         "director": {
            "type": "multi_field",
            "fields": {
               "director": {"type": "string's"},
              "original": {"type" : "string", "index" : "not_analyzed"}
            }
         }
      }
   }
}'
```

§1.4 ElasticSearch 集群设置

This will print the number of open files the process can open on startup. Alternatively, you can retrieve the max_file_descriptors for each node using the Nodes Info API, with:

curl -XGET 'namenode:9200/_nodes/process?pretty=true'，下面是删除 ES 上名为 jdbc 的 index 的 Restfule 命令:

curl -XDELETE 'http://namenode:9200/jdbc/'

配置文件在 **{$ES_HOME}/config** 文件夹下，**elasticsearch.yml** 和 **logging.yml**，修改 **elasticsearch.yml** 文件中的 cluster.name，当集群名称相同时，每个 ES 节点将会搜索它的伙伴节点，因此必须保证集群内每个节点的 cluster.name 相同，下面是关闭 ES 集群的 Restful 命令：

```
# 关闭集群内的某个ES节点'_local'
$ curl -XPOST 'http://namenode:9200/_cluster/nodes/_local/_shutdown'
# 关闭集群内的全部ES节点
$ curl -XPOST 'http://namenode:9200/_shutdown'
```

注意，如果一台机器上不止一个 ES 在运行，那么通过 **./bin/elasticsearch** 开启的 ES 的 http_address 将会使用 9200 以上的接口（形如 9201,9202,⋯），而相应的 transport_address 也递增（形如：9301,9302,⋯），因此，为使用 9200 端口，可使用上述命令关闭其它 ES 进程，可通过 conf 目录下的 log 文件来查看某些端口是否被占用。**elasticsearch.yml** 文件存在如下配置信息：

(1) node.master: true,node.data: true，允许节点存储数据，同时作为主节点；

(2) node.master: true,node.data: false，节点不存储数据，但作为集群的协调者；

(3) node.master: false,node.data: true，允许节点存储数据，但不作为主节点；

(4) node.master: false,node.data: false，节点不存储数据，也不作为协调者，但作为搜索任务的一个承担者；

(5) cluster.name: HadoopSearch，node.name: "ES-Slave-02",HadoopSearch 必须相同，但 node.name 每个节点可以自由设置；

如想将 ES 作为一个服务，需要从 github 上下载 elasticsearch-servicewrapper，然后调用 chkconfig，将其添加到/etc/rc[0∼6].d/中。

```
curl -L https://github.com/elasticsearch/elasticsearch-servicewrapper/
archive/master.zip > master.zip
unzip master.zip
cd elasticsearch-servicewrapper-master/
mv service /opt/elasticsearch/bin
/opt/elasticsearch/bin/service/elasticsearch install
## 如果想卸载该服务调用:
/opt/elasticsearch/bin/service/elasticsearch remove
## 如果想让ES开机启动
chkconfig elasticsearch on
## 如果想现在开启ES服务
service elasticsearch start
```

配置完后，可通过 *curl -X GET 'http://192.168.50.75:9200/_cluster/nodes?pretty'* 命令，查询集群下的节点信息。

为连接 hive 与 ES，运行 hive 后，在 hive 命令行内执行 **add jar /opt/elasticsearch-hadoop-1.3.0/dist/elasticsearch-hadoop-1.3.0.jar;** 或者 hdfs 上的 jar 包:**add jar hdfs://namenode:9000/elasticsearch-hadoop-1.3.0.jar** 可加载 elasticsearch-hadoop 插件，使用该插件的具体操作如下:

```
DROP TABLE IF EXISTS artist_1;
CREATE EXTERNAL TABLE artists_1 (
  cardid STRING, date STRING, time STRING)
STORED BY 'org.elasticsearch.hadoop.hive.ESStorageHandler'
TBLPROPERTIES('es.resource' = 'liubo/artists/',
              'es.host' = '192.168.50.75',
              'es.mapping.names' = 'text:time'
);
-- 集群下应使用'192.168.50.75'，而非'localhost'(es-hadoop的默认值)
-- insert data to Elasticsearch from another hive table
INSERT OVERWRITE TABLE artists_1
SELECT * FROM cable.temptable;
```

下面的代码是将 Mysql 中的表导入到 ES 中，建立名为 jdbc 的 index，表名称为 jiangsu。

```
curl -XPUT 'localhost:9200/_river/jiangsu/_meta' -d '{
    "type" : "jdbc",
    "jdbc" : {
        "driver" : "com.mysql.jdbc.Driver",
        "url" : "jdbc:mysql://192.168.50.75:3306/jsyx",
        "user" : "root",
        "password" : "123456",
        "sql" : "select * from jiangsu"
    },
    "index" : {
        "index" : "jdbc",
        "type" : "jiangsu"
    }
}'
```

§1.5 ES 性能优化

一个 Elasticsearch 节点会有多个线程池，但重要的是下面四个:

- 索引（index）: 主要是索引数据和删除数据操作（默认是 cached 类型）;
- 搜索（search）: 主要是获取，统计和搜索操作（默认是 cached 类型）;

- 批量操作（bulk）：对索引的批量操作，尚且不清楚它是不是原子性的，如果是原子的，则放在 MapReduce 里是没有问题的；
- 更新（refresh）：主要是更新操作，如当一个文档被索引后，何时能够通过搜索看到该文档；

在建立索引（index 相当于数据库，type 相当于数据库中的表）的过程中，需要修改如下配置参数：

- index.store.type： mmapfs. 因为内存映射文件机制能更好地利用 OS 缓存；
- indices.memory.index_buffer_size： 30% 默认值为 10%，表示 10% 的内存作为 indexing buffer；
- index.translog.flush_threshold_ops： 50000，当写日志数达到 50000 时，做一次同步；
- index.refresh_interval:30s，默认值为 1s，新建的索引记录将在 1 秒钟后查询到；

```
curl -XPUT 'http://namenode:9200/hivetest/?pretty' -d '{
    "settings" : {
       "index" : {
       "refresh_interval" : "30s",
       "index.store.type": "mmapfs",
       "indices.memory.index_buffer_size": "30%",
       "index.translog.flush_threshold_ops": "50000"
        }
    }
}'
```

但上述设置性能低下，也不知 why？ES 索引的过程相对 Lucene 的索引过程多了分布式数据的扩展，而 ES 主要是用 tranlog 进行各节点间的数据平衡，因此设置"index.translog.flush_threshold_ops" 为"100000"，意思是当 tranlog 数据达到多少条进行一次平衡操作，默认为 5000，而这个过程相对而言是比较浪费资源的，必要时可以将这个值设为 -1 关闭，进而手动进行 tranlog 平衡。"refresh_interval" 是刷新频率，设置为 30s 是指索引在生命周期内定时刷新，一但有数据进来就在 Lucene 里面 commit，因此其值设置大些可以提高索引效率。另外，如果有副本存在，数据也会马上同步到副本中去，因此在索引过程中，将副本设为 0，待索引完成后将副本个数改回来。

```
curl -XPUT 'namenode:9200/hivetest/' #新建一个名为hivetest的索引
curl -XPOST 'namenode:9200/hivetest/_close' #关闭索引，为修改参数做准备
curl -XPUT 'namenode:9200/hivetest/_settings?pretty' -d '{
   "index" : {
      "refresh_interval" : "30s",
      "index.translog.flush_threshold_ops": "100000",
      "number_of_replicas" : 0
   }
}'
curl -XPOST 'namenode:9200/hivetest/_open'
```

Linux 主机上硬盘空间有限，经常发现 root 目录下已没有可利用磁盘空间，为此，将 ES 的日志和数据输出目录设置在/home 目录下，修改 config 目录下的 elasticsearch.yml 文件中的选项，其中 path.data 为索引文件存放目录，path.work 为临时文件存放目录，path.logs 为日志文件存放目录。

```
path.data: /home/elasticSearch/data
path.work: /home/elasticSearch/work
path.logs: /home/elasticSearch/logs
```

§1.6 ElasticSearch 杂项

§1.6.1 Lucene 文件结构

Files and detailed format:

- .tim: Term Dictionary
- .tip: Term Index
- .doc: Frequencies and Skip Data
- .pos: Positions
- .pay: Payloads and Offsets

The .tim file contains the list of terms in each field along with per-term statistics (such as docfreq) and pointers to the frequencies, positions, payload and skip data in the .doc, .pos, and .pay files.

The .doc file contains the lists of documents which contain each term, along with the frequency of the term in that document.

The .pos file contains the lists of positions that each term occurs at within documents. It also sometimes stores part of payloads and offsets for speedup.TermPositions are order by term (terms are implicit, from the term dictionary), and position values for each term document pair are incremental, and ordered by document number.

§1.6.2 Lucene 的基本分词过程

Analyzer 类是所有分词器的基类，它是个抽象类，所有的子类必须实现它，它提供了如下方法：

```
@Override
protected TokenStreamComponents createComponents(String fieldName,
        final Reader reader) {
    Tokenizer tokenizer = new AnsjTokenizer(new IndexAnalysis(reader),
        reader, filter, pstemming);
    return new TokenStreamComponents(tokenizer);
}
```

TokenStream 是能够产生词序列的类，有两个类型的子类 TokenFilter 和 Tokenizer，Tokenizer 通过读取汉字字符串创建词序列 List<word>，其中每一 word 将作为 Lucene 索引的 key，Tokenizer 还记录每个 word 的偏移值，以及在文档中属于第几个 word（$0, \cdots, n$）等，而 TokenFilter 则对产生的词序列进行转换 (比如过滤等)。

§1.6.3 开发 ES 插件

插件一般情况下是一个 zip 文件，它包含了若干 Jar 包，为安装插件，依赖于 Maven 的安装。Maven 工程的 main 目录里须包含三个目录 (java,resources,assembly)。新建 src/main/resources/es-plugin.properties 文件，文件内容为：

```
plugin=${project.groupId}.${project.artifactId}
## ${project.groupId}.${project.artifactId}必须与插件类名相同
## 或者直接告诉ES插件类名
plugin=org.soul.ESearch.SoulAnalysisPlugin
```

然后在配置文件 pom.xml 的 <build>/<plugins>/<plugin>/<artifactId>maven-assembly-plugin</artifactId> 标签中添加如下内容，这是因为 maven-assembly-plugin 插件负责打包 Maven 工程，其中的 plugin.xml 为打包 Project 时额外执行的功能：

```
<descriptors>
   <descriptor>
    ${basedir}/src/main/assembly/plugin.xml
  </descriptor>
</descriptors>
```

然后新建 src/main/assembly/plugin.xml 文件，写入如下语句，告诉 Maven 对编译生成的 classes 目录下的所有内容打包，其中的 zip 表示生成格式为 zip，id 不具有特别含义。

```
<?xml version="1.0"?>
<assembly>
  <id>plugin</id>
  <formats>
     <format>zip</format>
  </formats>
  <fileSets>
    <fileSet>
      <directory>${project.build.directory}/classes</directory>
      <outputDirectory>/</outputDirectory>
    </fileSet>
  </fileSets>
</assembly>
```

执行 “mvn assembly:single” 后，应该生成了 zip 文件，调用/opt/elasticsearch-0.90.5/bin/plugin 命令，将 zip 文件注入到 ElasticSearch 中。

§1.6.4 ES 与 hive

Hive 从 0.8.0 版本后支持两个虚拟列:**INPUT__FILE__NAME**（即 mapper 任务的输出文件名）和 **BLOCK__OFFSET__INSIDE__FILE**（即读取记录在当前文件的全局偏移量）。对于块压缩文件，就是当前块的文件偏移量，即当前块的第一个字节在文件中的偏移量。Simple Example:

```
use cable;
select INPUT__FILE__NAME, cardid, BLOCK__OFFSET__INSIDE__FILE from test;
select min(BLOCK__OFFSET__INSIDE__FILE),count(INPUT__FILE__NAME) from test
  where BLOCK__OFFSET__INSIDE__FILE < 12000 group by cardid;
select * from test where BLOCK__OFFSET__INSIDE__FILE < 12000;
```

开始时将'es.mapping.id' = 'id' 写成了'es.mapping.id ' = 'id'，由于多了一个空格，程序一直没有获得期望的结果。'es.mapping.id' 属性告诉 Hive，使用 hive1 表中的 id 列的值作为 ElasticSearch 的 _id 域，使用剩余几列的值作为 ElasticSearch 的 _source。表 hive1 不是存储在 hive 内部，所以使用 **EXTERNAL** 关键字，对于使用 **EXTERNAL** 存储的表，必须提供 **STORED BY** 关键字，否则 hive 无法确定用哪个类来存储外部表。

在向 ElasticSearch 添加索引过程中，必须提供 id 域，否则当某 MapTask 失败若干次时，相同的记录可能会插入很多遍。因为如果不指定 id，ES 会给当前记录随机分配一个 id，这点类似于往数据库中 insert 记录时，必须保证对相同 record，其主键也相同，所以在使用 hive 向 ElasticSearch 中插入记录时，其 id 域为相应行在 hdfs 文件中的位置。

```
set mapred.max.split.size=32000000;
add jar hdfs://namenode:9000/user/liubo/1.jar;
DROP TABLE IF EXISTS hive1;
CREATE EXTERNAL TABLE hive1(
   id STRING,
   cardId STRING,
   playDate STRING,
   playTime STRING,
   channel STRING,
   program STRING
)
STORED BY 'org.elasticsearch.hadoop.hive.ESStorageHandler'
TBLPROPERTIES('es.resource' = 'eshive/hive1/',
              'es.host' = '192.168.50.75',
              'es.mapping.id' = 'id'
);
INSERT OVERWRITE TABLE hive1 SELECT BLOCK__OFFSET__INSIDE__FILE,* FROM testData2;
```

§1.7 ElasticSearch 源码分析

(1) org.elasticsearch.rest.action，该目录负责接受，解释 Rest 请求。

(2) org.elasticsearch.client.node 负责管理节点客户端请求。

(3) org.elasticsearch.action.admin.indices.* 负责处理 Rest 请求中的动作。

(4) lasticsearch 里面的组件基本都是用上面的方式进行模块化管理，elasticsearch 对 guice 进行了简单的封装，通过 ModulesBuilder 类构建 es 的模块，一个 es 节点包括下面模块：

(5) PluginsModule：插件模块

(6) SettingsModule：设置参数模块

(7) NodeModule：节点模块

(8) NetworkModule：网络模块

(9) NodeCacheModule：缓存模块

(10) ScriptModule：脚本模块

(11) JmxModule：jmx 模块

(12) EnvironmentModule：环境模块

(13) NodeEnvironmentModule：节点环境模块

(14) ClusterNameModule：集群名模块

(15) ThreadPoolModule：线程池模块

(16) DiscoveryModule：自动发现模块

(17) ClusterModule：集群模块

(18) RestModule：rest 模块

(19) TransportModule：tcp 模块

(20) HttpServerModule：http 模块

(21) RiversModule：river 模块

(22) IndicesModule：索引模块

(23) SearchModule：搜索模块

(24) ActionModule：行为模块

(25) MonitorModule：监控模块

(26) GatewayModule：持久化模块

(27) NodeClientModule：客户端模块

(1) 有"/_analyze" 和"/{index}/_analyze 两种，接受 GET 和 POST 方法，在 org/elasticsearch/rest/action/admin/indices/analyze/RestAnalyzeAction.java 中被处理；在 ActionModule 中，存在一个注册函数 registerAction(AnalyzeAction.INSTANCE, TransportAnalyzeAction.class)，因此，请求将由 TransportAnalyzeAction 类处理，

(2) "/{index}/_close"，只接受 POST 方法，在 org/elasticsearch/rest/action/admin/indices/close/RestCloseIndexAction.java 中被处理；

(3) 创建一个新的索引，json 参数中包括 settings 和 mappings（注意并非 **endpoint** 中的 **_mapping**），只有"/{index}" 一种操作，接受 POST 和 PUT 两种方法，在 org/elasticsearch/rest/action/admin/indices/create/RestCreateIndexAction.java 中处理；