ElasticSearch原理篇

一、开篇几个问题

1、大规模数据如何检索?

当系统数据量上了10亿、100亿条的时候，我们在做系统架构的时候通常会从以下角度去考虑：

1）用什么数据库好？(MySQL、sybase、Oracle、达梦、神通、MongoDB、Hbase...)

2）如何解决单点故障；(lvs、F5、A10、Zookeeper、MQ)

3）如何保证数据安全性；(热备、冷备、异地多活)

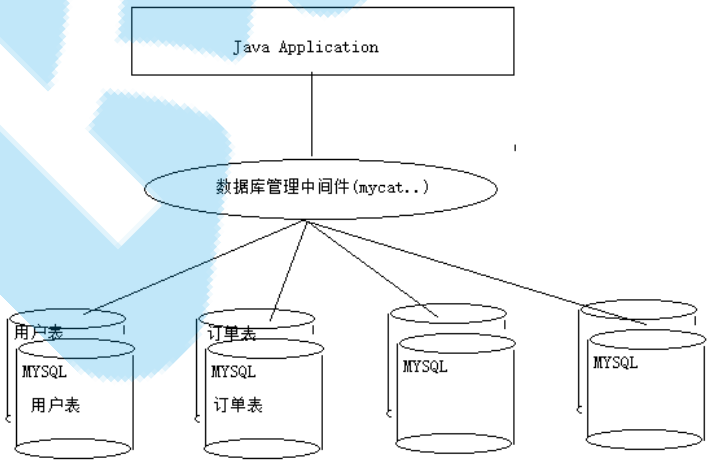
4）如何解决检索难题；(数据库代理中间件：mysql-proxy、Cobar、MaxScale等;)

5）如何解决统计分析问题；(离线、近实时)

2、传统数据库的应对解决方案?

对于关系型数据，我们通常采用以下或类似架构去解决查询瓶颈和写入瓶颈：
解决要点：
1）通过主从备份解决数据安全性问题；
2）通过数据库代理中间件心跳监测，解决单点故障问题；
3）通过代理中间件将查询语句分发到各个slave节点进行查询，并汇总结果

4）通过分表分库解决读写效率问题

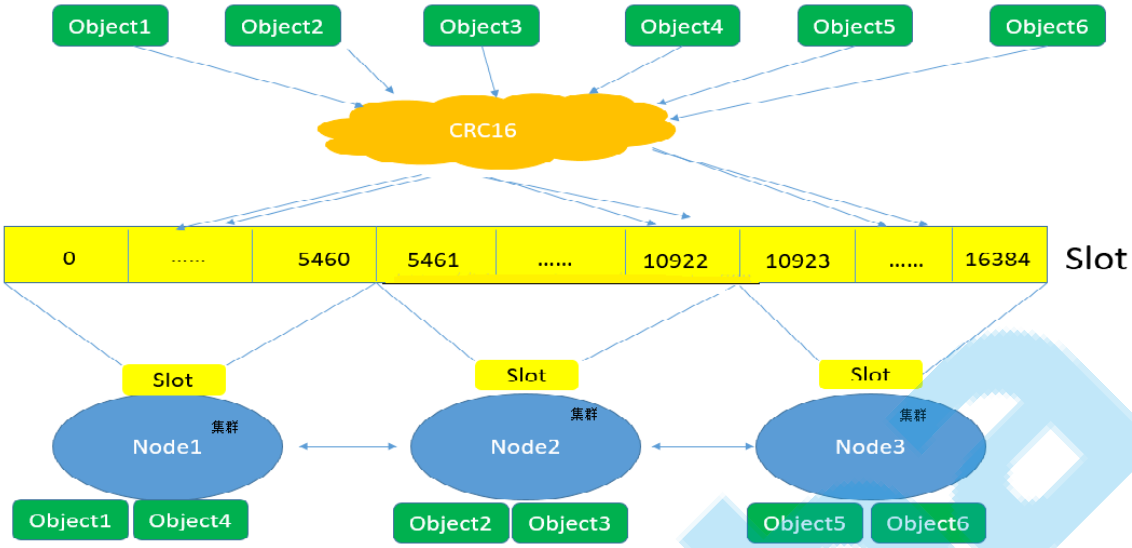


数据中间件管理数据库：
1、数据库分表分库
2、使用类似于mycat的数据库中间件管理数据库
3、水平拆分，垂直拆分
4、集群

3、非关系型数据库的解决方案?

对于Nosql数据库，以redis为例，其它原理类似：
解决要点：
1）通过副本备份保证数据安全性；
2）通过节点竞选机制解决单点问题；

3）先从配置库检索分片信息，然后将请求分发到各个节点，最后由路由节点合并汇总结果



4、完全把数据放入内存怎么样?

完全把数据放在内存中是不可靠的，实际上也不太现实，当我们的数据达到PB级别时，按照每个节点96G内存计算，在内存完全装满的数据情况下，我们需要的机器是：1PB=1024T=1048576G

节点数=1048576/96=10922个

实际上，考虑到数据备份，节点数往往在2.5万台左右。成本巨大决定了其不现实!

从前面我们了解到，把数据放在内存也好，不放在内存也好，都不能完完全全解决问题。

全部放在内存速度问题是解决了，但成本问题上来了。

为解决以上问题，从源头着手分析，通常会从以下方式来寻找方法：

1、存储数据时按有序存储；

2、将数据和索引分离；

3、压缩数据；

二、全文检索技术

1、什么是全文检索?

什么叫做全文检索呢？这要从我们生活中的数据说起。

我们生活中的数据总体分为两种：**结构化数据和非结构化数据。**

- **结构化数据**：指具有固定格式或有限长度的数据，如数据库，元数据等。
- **非结构化数据**：指不定长或无固定格式的数据，如 互联网数据、邮件，word文档等。

对非结构化数据顺序扫描很慢，对结构化数据的搜索却相对较快，那么把我们的非结构化数据想办法弄得有一定结构不就行了吗？这就是全文检索的基本思路，也即将非结构化数据中的一部分信息提取出来，重新组织，使其变得有一定结构，然后对此有一定结构的数据进行搜索，从而达到搜索相对较快的目的。这部分从非结构化数据中提取出的然后重新组织的信息，我们称之**索引** 。

非结构化数据又一种叫法叫全文数据。

按照数据的分类，搜索也分为两种：

- **对结构化数据的搜索：** 如对数据库的搜索，用SQL语句。再如对元数据的搜索，如利用windows搜索对文件名，类型，修改时间进行搜索等。
- **对非结构化数据的搜索：** 如用Google和百度可以搜索大量内容数据。

对非结构化数据也即全文数据的搜索主要有两种方法：**顺序扫描法和反向索引法。**

- **顺序扫描法**：所谓顺序扫描法，就是顺序扫描每个文档内容，看看是否有要搜索的关键字，实现查找文档的功能，也就是根据文档找词。
- **反向索引法**：所谓反向索引，就是提前将搜索的关键字建成索引，然后再根据索引查找文档，也就是根据词找文档。

这种先建立`索引`，再对索引进行`搜索`文档的过程就叫`全文检索(Full-text Search)`。

2、全文检索场景

- 搜索引擎
- 站内搜索
- 系统文件搜索

3、全文检索相关技术

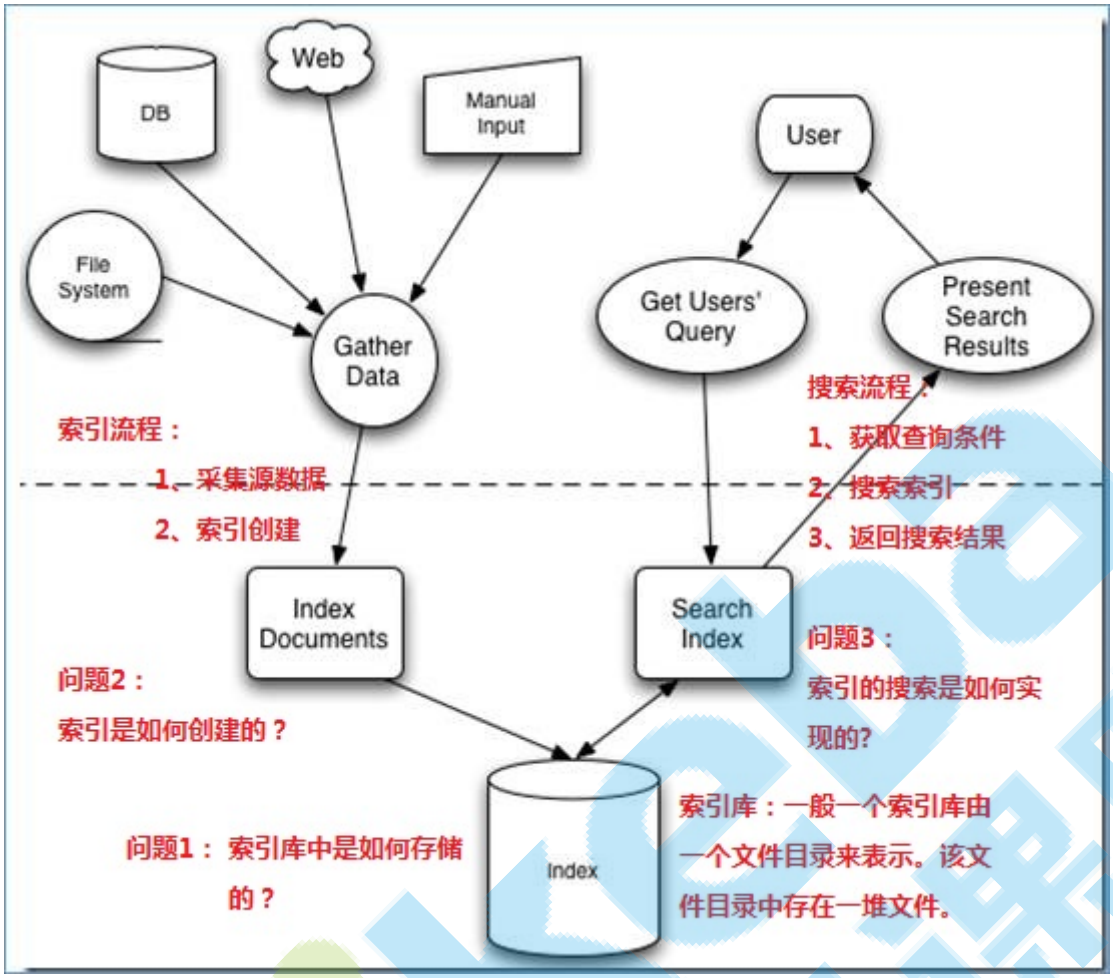
1. Lucene：如果使用该技术实现，需要对Lucene的API和底层原理非常了解，而且需要编写大量的Java代码。
2. Solr：使用java实现的一个web应用，可以使用rest方式的http请求，进行远程API的调用。
3. ElasticSearch(ES)：可以使用rest方式的http请求，进行远程API的调用。

三、全文检索的流程

1、流程总览

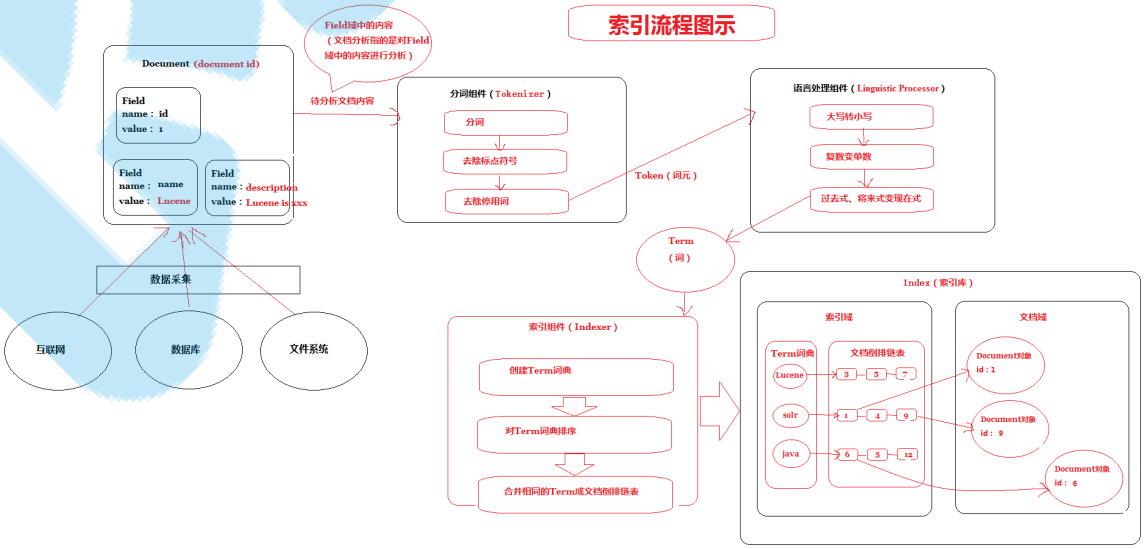
全文检索的流程分为两大流程：索引创建、搜索索引

- 索引创建：将现实世界中所有的结构化和非结构化数据提取信息，创建索引的过程。
- 搜索索引：就是得到用户的查询请求，搜索创建的索引，然后返回结果的过程。



2、索引库结构

#温馨提示：想搞清楚全文检索，必须要搞清楚下面三个问题：
1. 索引库里面究竟存些什么？(Index)
2. 如何创建索引？(Indexing)
3. 如何对索引进行搜索？(Search)



四、ElasticSearch介绍

1、ES基本概述

Elasticserach由来：许多年前，一个叫Shay Banon的待业工程师跟随他的新婚妻子来到伦敦，他的妻子想在伦敦学习做一名厨师。而他在伦敦寻找工作的期间，接触到了Lucene的早期版本，他想为自己的妻子开发一个方便搜索菜谱的应用。

Elasticsearch发布的第一个版本是在2010年的二月份，从那之后，Elasticsearch便成了Github上最受人瞩目的项目之一，并且很快就有超过300名开发者加入进来贡献了自己的代码。后来Shay和另一位合伙人成立了公司专注打造Elasticsearch，他们对Elasticsearch进行了一些商业化的包装和支持。但是，Elasticsearch承诺，永远都将是开源并且免费的。

Elastic为主体的公司提供了很多优秀的解决方案，拿到很多的投资，现已上市，后来收购logstash,kibana及一些其他的服务。

* ElasticSearch
* Logstash
* Kibana .....

据国际权威的数据库产品评测机构DBEngines的统计，在2016年1月，Elasticsearch已超过Solr等，成为排名第一的搜索引擎类应用.

2、ES是什么?

1）ES = elasticsearch简写， Elasticsearch是一个开源的**高扩展**的**分布式**全文检索引擎.

2）Elasticsearch也使用Java开发并使用**Lucene**作为其核心来实现所有索引和搜索的功能。但是它的目的是通过简单的RESTful API来隐藏Lucene的复杂性，从而让全文搜索变得简单。Elasticsearch是面向文档型数据库，一条数据在这里就是一个文档，用JSON作为文档序列化的格式，比如下面这条用户数据：

```
{
    "name" :     "John",
    "sex" :      "Male",
    "age" :      25,
    "birthDate": "1990/05/01",
    "about" :    "I love to go rock climbing",
    "interests": [ "sports", "music" ]
}
```

3、为什么要使用ES?

- ES国内外使用优秀案例

1） 2013年初，GitHub抛弃了Solr，采取ElasticSearch 来做PB级的搜索。 “GitHub使用ElasticSearch搜索20TB的数据，包括13亿文件和1300亿行代码”。

2）维基百科：启动以elasticsearch为基础的核心搜索架构。

3）SoundCloud：“SoundCloud使用ElasticSearch为1.8亿用户提供即时而精准的音乐搜索服务”。

4）百度：百度目前广泛使用ElasticSearch作为文本数据分析，采集百度所有服务器上的各类指标数据及用户自定义数据，通过对各种数据进行多维分析展示，辅助定位分析实例异常或业务层面异常。目前覆盖百度内部20多个业务线（包括casio、云分析、网盟、预测、文库、直达号、钱包、风控等），单集群最大100台机器，200个ES节点，每天导入30TB+数据。

- 我们也需要

实际项目开发实战中，几乎每个系统都会有一个搜索的功能，当搜索做到一定程度时，维护和扩展起来难度就会慢慢变大，所以很多公司都会把搜索单独独立出一个模块，用ElasticSearch等来实现。

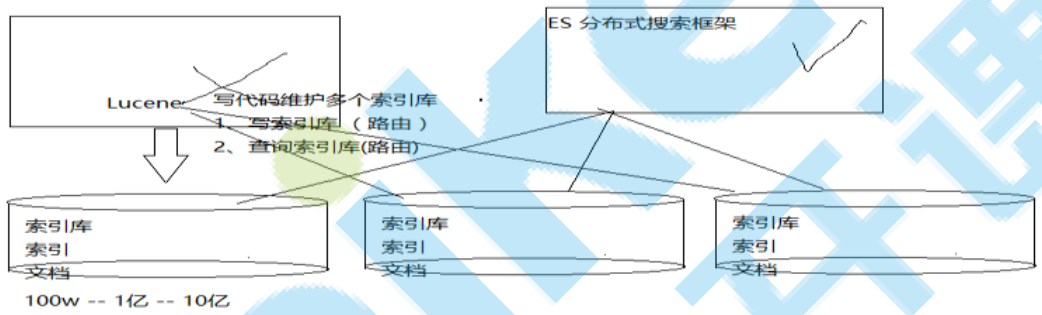
近年ElasticSearch发展迅猛，已经超越了其最初的纯搜索引擎的角色，现在已经增加了数据聚合分析（aggregation）和可视化的特性，如果你有数百万的文档需要通过关键词进行定位时，ElasticSearch肯定是最佳选择。

当然，如果你的文档是JSON的，你也可以把ElasticSearch当作一种“NoSQL数据库”， 应用ElasticSearch数据聚合分析（aggregation）的特性，针对数据进行多维度的分析。

4、ES有什么能力?

Elasticsearch 是一个分布式可扩展的实时搜索和分析引擎,一个建立在全文搜索引擎 Apache Lucene(TM) 基础上的搜索引擎.

实际项目开发实战中，几乎每个系统都会有一个搜索的功能，当数据达到很大且搜索要做到一定程度时，维护和扩展难度就会越来越高，并且在全文检索的速度上、结果内容的推荐、分析以及统计聚合方面也很难达到我们预期效果。



并且 Elasticsearch，它不仅包括了全文搜索功能，还可以进行以下工作:

l 分布式实时文件存储，并将每一个字段都编入索引，使其可以被搜索。（实时的存储、检索数据）

l 实时分析的分布式搜索引擎。

l 可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。（集群只能支持：上百台 ---- 中等规模的数据）

5、使用场景

1）网站搜索

ES搜索自动：补全建议

关键词搜索

2）日志分析

ELK经典组合

ElasticSearch + logstash + kibana === 日志收集 + 日志分析 + 运维

3）数据预警

第三方插件：wathcer 监控数据

4）商业智能，数据分析

ES数据分析

一线公司ES使用场景

1）新浪ES 如何分析处理32亿条实时日志 http://dockone.io/article/505

2）阿里ES 构建挖财自己的日志采集和分析体系 http://afoo.me/columns/tec/logging-platform-spec.html

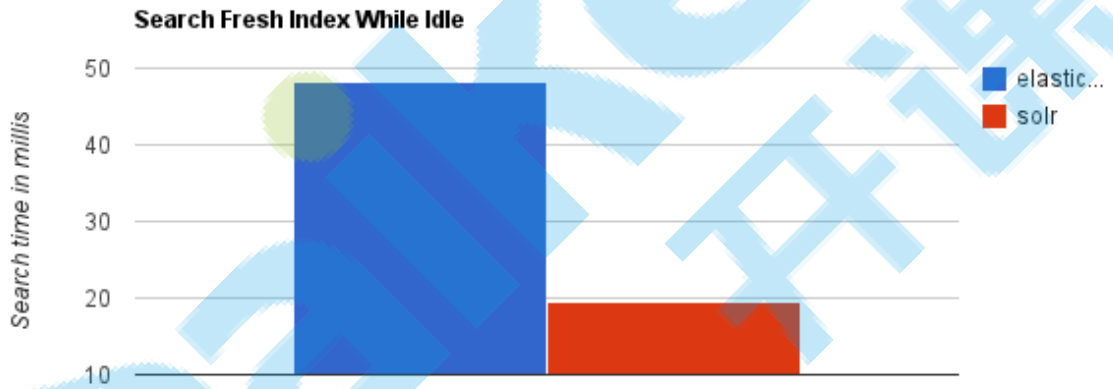
3）有赞ES 业务日志处理 http://tech.youzan.com/you-zan-tong-ri-zhi-ping-tai-chu-tan/

4）ES实现站内搜索 http://www.wtoutiao.com/p/13bkqiZ.html

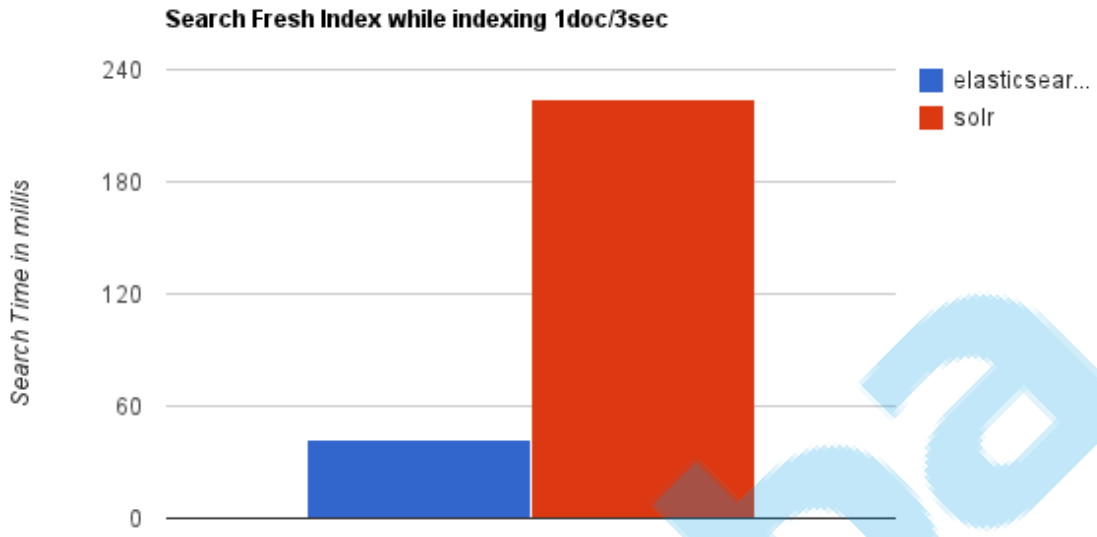
6、Solr和ES的比较

ElasticSearch vs Solr 检索速度

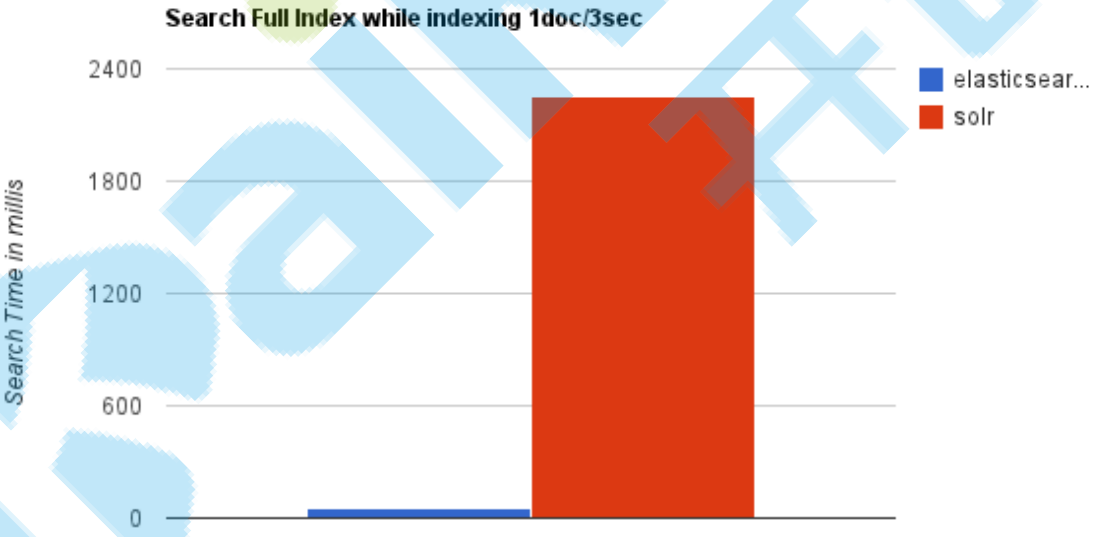
- 当单纯的对**已有数据**进行搜索时，Solr更快。



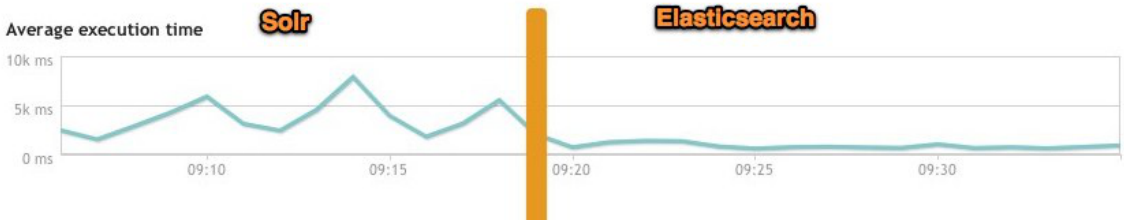
- **当实时建立索引时, Solr会产生io阻塞，查询性能较差, Elasticsearch具有明显的优势。**



- **随着数据量的增加，Solr的搜索效率会变得更低，而Elasticsearch却没有明显的变化。**



- **大型互联网公司，实际生产环境测试，将搜索引擎从Solr转到Elasticsearch以后的平均查询速度有了50倍的提升。**

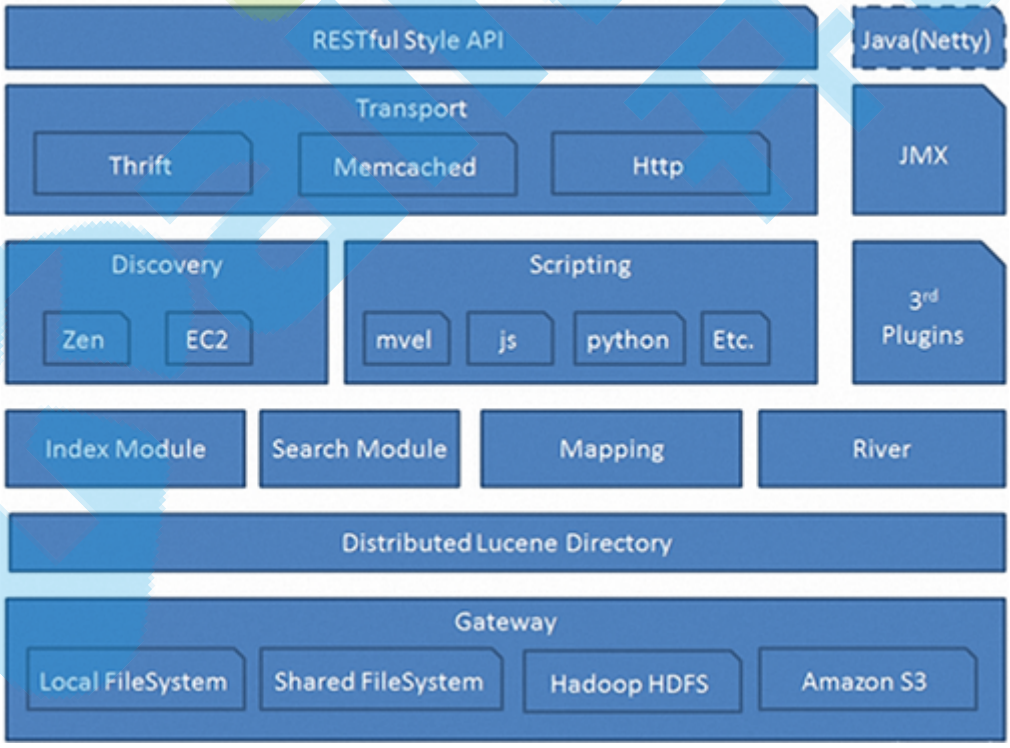


总结：

- 二者安装都很简单；
- Solr 利用 Zookeeper 进行分布式管理，而 Elasticsearch 自身带有分布式协调管理功能;
- Solr 支持更多格式的数据，而 Elasticsearch 仅支持json文件格式；
- Solr 官方提供的功能更多，而 Elasticsearch 本身更注重于核心功能，高级功能多有第三方插件提供；
- Solr 在传统的搜索应用中表现好于 Elasticsearch，但在处理实时搜索应用时效率明显低于 Elasticsearch。

> **最终的结论：**
>
> Solr 是传统搜索应用的有力解决方案，但 Elasticsearch 更适用于新兴的实时搜索应用。

五、ES的架构



1、网关Gateway层

底层文件系统：
ES用来存储索引文件的一个文件系统且它支持很多类型。例如：本地磁盘、共享存储（做snapshot的时候需要用到）、hadoop的hdfs分布式存储、亚马逊的S3。

职责：
它的主要职责是用来对数据进行长持久化以及整个集群重启之后可以通过gateway重新恢复数据。代表es索引的持久化存储方式，es默认是先把索引存放到内存中，当内存满了时再持久化到硬盘。

数据安全：
当这个es集群关闭再重新启动时就会从gateway中读取索引数据。es支持多种类型的gateway，有本地文件系统（默认），分布式文件系统，Hadoop的HDFS和amazon的s3云存储服务。

存储数据：
存储索引信息,集群信息,mapping 等等

2、districted lucene directory

`Gateway`上层就是一个`lucene`的分布式框架`lucene`是做检索的，但是它是一个单机的搜索引擎，像这种es分布式搜索引擎系统，虽然底层用`lucene`，但是需要在每个节点上都运行`lucene`进行相应的索引、查询以及更新，所以需要做成一个分布式的运行框架来满足业务的需要。

3、四大模块组件

`districted lucene directory`之上就是一些`ES`的四大模块`：

- `Index Module`是索引模块，就是对数据建立索引也就是通常所说的建立一些倒排索引等；
- `Search Module`是搜索模块，就是对数据进行查询搜索；
- `Mapping Module`是数据映射与解析模块，就是你的数据的每个字段可以根据你建立的表结构通过mapping进行映射解析，如果你没有建立表结构，es就会根据你的数据类型推测你的数据结构之后自己生成一个mapping，然后都是根据这个mapping进行解析你的数据；
- `River Module`在es2.0之后应该是被取消了，它的意思表示是第三方插件，例如可以通过一些自定义的脚本将传统的数据库（mysql）等数据源通过格式化转换后直接同步到es集群里，这个River大部分是自己写的，写出来的东西质量参差不齐，将这些东西集成到es中会引发很多内部bug，严重影响了es的正常应用，所以在es2.0之后考虑将其去掉。

4、自动发现Discovery、Script

`ES`四大模块组件之上有`Discovery`模块：

es是一个集群包含很多节点，很多节点需要互相发现对方，然后组成一个集群包括选主的，这些es都是用的discovery模块，默认使用的是 Zen。es是一个基于p2p的系统，它先通过广播寻找存在的节点，再通过多播协议来进行节点之间的通信，同时也支持点对点的交互。
es查询还可以支撑多种script即脚本语言，包括mvel、js、python等等

5、通信（Transport）

- 代表es内部节点或集群与客户端的交互方式，默认内部是使用tcp协议进行交互，同时它支持http协议（json格式）、thrift、servlet、memcached、zeroMQ等的传输协议（通过插件方式集成）
- 节点间通信端口默认：9300-9400
- JMX就是java的一个远程监控管理框架，因为es是通过java实现的。

6、 RESTful接口层

最上层就是ES暴露给我们的`访问接口`

官方推荐的方案就是这种Restful接口，直接发送http请求，方便后续使用nginx做代理、分发包括可能后续会做权限的管理，通过http很容易做这方面的管理。如果使用java客户端它是直接调用api，在做负载均衡以及权限管理还是不太好做。

六、ES概念说明

Elasticsearch是面向文档(document oriented)的，这意味着它可以存储整个对象或文档(document)。然而它不仅仅是存储，还会索引(index)每个文档的内容使之可以被搜索。在Elasticsearch中，你可以对文档（而非成行成列的数据）进行索引、搜索、排序、过滤。Elasticsearch比传统关系型数据库如下：

```
Relational DB -> Databases -> Tables -> Rows -> Columns
Elasticsearch -> Indices   -> （Types） -> Documents -> Fields
```

1 索引 index

一个索引就是一个拥有几分相似特征的文档的集合。比如说，你可以有一个客户数据的索引，另一个产品目录的索引，还有一个订单数据的索引。一个索引由一个名字来标识（必须全部是小写字母的），并且当我们要对对应于这个索引中的文档进行索引、搜索、更新和删除的时候，都要使用到这个名字。在一个集群中，可以定义任意多的索引。

2 类型 type

在一个索引中，你可以定义一种或多种类型。一个类型是你的索引的一个逻辑上的分类/分区，其语义完全由你来定。通常，会为具有一组共同字段的文档定义一个类型。比如说，我们假设你运营一个博客平台并且将你所有的数据存储到一个索引中。在这个索引中，你可以为用户数据定义一个类型，为博客数据定义另一个类型，当然，也可以为评论数据定义另一个类型。

3 文档 document

一个文档是一个可被索引的基础信息单元。比如，你可以拥有某一个客户的文档，某一个产品的一个文档，当然，也可以拥有某个订单的一个文档。文档以JSON（Javascript Object Notation）格式来表示，而JSON是一个到处存在的互联网数据交互格式。

在一个index/type里面，你可以存储任意多的文档。注意，尽管一个文档，物理上存在于一个索引之中，文档必须被索引/赋予一个索引的type。

4 字段Field

相当于是数据表的字段，对文档数据根据不同属性进行的分类标识

5 映射 mapping

mapping是处理数据的方式和规则方面做一些限制，如某个字段的数据类型、默认值、分析器、是否被索引等等，这些都是映射里面可以设置的，其它就是处理es里面数据的一些使用规则设置也叫做映射，按着最优规则处理数据对性能提高很大，因此才需要建立映射，并且需要思考如何建立映射才能对性能更好。

七、使用Restful接口访问ES

1 ElasticSearch的接口语法

```
curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY_STRING>' -d '<BODY>'
```

其中：

参数	解释
VERB	适当的 HTTP *方法* 或 *谓词*：GET、 POST、 PUT、 HEAD 或者 DELETE。
PROTOCOL	http 或者 https （如果你在 Elasticsearch 前面有一个 https 代理）
HOST	Elasticsearch 集群中任意节点的主机名，或者用 localhost 代表本地机器上的节点。
PORT	运行 Elasticsearch HTTP 服务的端口号，默认是 9200 。
PATH	API 的终端路径（例如 _count 将返回集群中文档数量）。Path 可能包含多个组件，例如：_cluster/stats 和 _nodes/stats/jvm 。
QUERY_STRING	任意可选的查询字符串参数 (例如 ?pretty 将格式化地输出 JSON 返回值，使其更容易阅读)
BODY	一个 JSON 格式的请求体 (如果请求需要的话)

2 创建索引index和映射mapping

请求url：

```
PUT     localhost:9200/blog1
```

请求体：

```
{
    "settings":{
        "index":{
          "number_of_shards": "5",
          "number_of_replicas": "1"
        }
    },
    "mappings": {
        "_doc": {
            "properties": {
                "id": {
                    "type": "long",
                    "store": true,
                    "index":true
                },
                "title": {
                    "type": "text",
                    "store": true,
                    "index": true,
                    "analyzer":"standard"
                },
                "content": {
                    "type": "text",
                    "store": true,
                    "index":true,
                    "analyzer":"standard"
                }
            }
        }
```

```
    }
}
```

3 创建索引后设置Mapping

我们可以在创建索引时设置mapping信息，当然也可以先创建索引然后再设置mapping。

在上一个步骤中不设置maping信息，直接使用put方法创建一个索引，然后设置mapping信息。

请求的url：

```
POST    http://127.0.0.1:9200/blog2/_doc/_mapping
```

请求体：

```
{
    "_doc": {
            "properties": {
                "id":{
                    "type":"long",
                    "store":true
                },
                "title":{
                    "type":"text",
                    "store":true,
                    "index":true,
                    "analyzer":"standard"
                },
                "content":{
                    "type":"text",
                    "store":true,
                    "index":true,
                    "analyzer":"standard"
                }
            }
        }
  }
```

4 删除索引index

请求url：

```
DELETE      localhost:9200/blog1
```

5 创建文档document

请求url：

```
POST    localhost:9200/blog1/_doc/1
```

请求体：

```
{
    "id":1,
    "title":"ElasticSearch是一个基于Lucene的搜索服务器",
    "content":"它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是当前流行的企业级搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。"
}
```

6 修改文档document

请求url：

```
POST    localhost:9200/blog1/_doc/1
```

请求体：

```
{
    "id":1,
    "title":"【修改】ElasticSearch是一个基于Lucene的搜索服务器",
    "content":"【修改】它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。Elasticsearch是用Java开发的，并作为Apache许可条款下的开放源码发布，是当前流行的企业级搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。"
}
```

7 删除文档document

请求url：

```
DELETE  localhost:9200/blog1/_doc/1
```

8 根据id查询

请求url：

```
GET localhost:9200/blog1/_doc/1
```

八、查询表达式(Query DSL)

1 查询全部

请求url：

```
POST http://localhost:9200/_search
```

请求体：

```
{"query":
 {
     "match_all":{}
 }
}
```

2term查询

请求url：

```
POST    localhost:9200/blog1/_doc/_search
```

请求体：

```
{
    "query": {
        "term": {
            "title": "搜索"
        }
    }
}
```

3 querystring查询

请求url：

```
POST    localhost:9200/blog1/article/_search
```

请求体：

```
{
    "query": {
        "query_string": {
            "default_field": "title",
            "query": "搜索服务器"
        }
    }
}
```

4 multi_match查询

可以在多个字段上查询

```
{
    "query":{
        "multi_match":{
            "query":"李四",
            "fields":["name","address"]
        }
    }
}
```

5 bool查询

bool （布尔）过滤器。 这是个 复合过滤器（compound filter） ，它可以接受多个其他过滤器作为参数，并将这些过滤器结合成各式各样的布尔（逻辑）组合。

一个 bool 过滤器由三部分组成：

```
{
   "bool":{
      "must":[],
      "should":[],
      "must_not":[],
      "filter":[]
   }
}
```

实例：

```
{
    "query":{
        "bool":{
            "must":{
                "query_string":{
                    "query":"李四",
                    "default_field":"name"
                }
            },
            "should":{
                "term":{
                    "address":"上海"
                }
            },
            "filter":{
                "query_string":{
                    "query":"男",
                    "default_field":"sex"
                }
            }
        }
    }
}
```

`must`：语句指明了，对于一个文档，所有的查询都必须为真，这个文档才能够匹配成功。

`should`：语句指明，对于一个文档，查询列表中，只要有一个查询匹配，那么这个文档就被看成是匹配的。

`must_not`：语句指明，对于一个文档，查询列表中的的所有查询都必须都不为真，这个文档才被认为是匹配的。

`filter`：从ES5.0开始过滤条件需要在添加在bool查询中，filter条件中支持查询条件中的所有查询方法。filter可以配置多个，从而实现多个过滤条件同时生效。

```
{
    "query":{
        "bool":{
            "must":{
                "query_string":{
                    "query":"李四",
                    "default_field":"name"
                }
            },
            "filter":{
                "term":{
                    "address":"上海"
                }
            },
            "filter":{
                "query_string":{
                    "query":"男",
                    "default_field":"sex"
                }
            },
            "filter":{
                "term":{
                    "id":"20"
                }
            }
        }
    }
}
```

九、 IK 分词器和ElasticSearch集成使用

5.1 上述查询存在问题分析

在进行字符串查询时，我们发现去搜索"搜索服务器"和"钢索"都可以搜索到数据；

而在进行词条查询时，我们搜索"搜索"却没有搜索到数据；

究其原因是ElasticSearch的标准分词器导致的，当我们创建索引时，字段默认使用的是标准分词器：

```
{
    "mappings": {
        "article": {
            "properties": {
                "id": {
                    "type": "long",
                    "store": true,
                    "index": true
```

```
                },
                "title": {
                    "type": "text",
                    "store": true,
                    "index": true,
                    "analyzer":"standard"    //标准分词器
                },
                "content": {
                    "type": "text",
                    "store": true,
                    "index": true,
                    "analyzer":"standard"    //标准分词器
                }
            }
        }
    }
}
```

例如对 "我是程序员" 进行分词

标准分词器分词效果测试：

```
post _analyze
{
  "analyzer":"standard",
  "text":"这是一个测试"
}
```

分词结果：

```
{
  "tokens": [
    {
      "token": "这",
      "start_offset": 0,
      "end_offset": 1,
      "type": "<IDEOGRAPHIC>",
      "position": 0
    },
    {
      "token": "是",
      "start_offset": 1,
      "end_offset": 2,
      "type": "<IDEOGRAPHIC>",
      "position": 1
    },
    {
      "token": "一",
      "start_offset": 2,
      "end_offset": 3,
      "type": "<IDEOGRAPHIC>",
      "position": 2
    },
    {
      "token": "个",
      "start_offset": 3,
      "end_offset": 4,
```

```
      "type": "<IDEOGRAPHIC>",
      "position": 3
    },
    {
      "token": "测",
      "start_offset": 4,
      "end_offset": 5,
      "type": "<IDEOGRAPHIC>",
      "position": 4
    },
    {
      "token": "试",
      "start_offset": 5,
      "end_offset": 6,
      "type": "<IDEOGRAPHIC>",
      "position": 5
    }
  ]
}
```

5.2 IK分词器简介

IKAnalyzer是一个开源的，基于java语言开发的轻量级的中文分词工具包。从2006年12月推出1.0版开始，IKAnalyzer已经推出 了3个大版本。最初，它是以开源项目Lucene为应用主体的，结合词典分词和文法分析算法的中文分词组件。新版本的IKAnalyzer3.0则发展为 面向Java的公用分词组件，独立于Lucene项目，同时提供了对Lucene的默认优化实现。

IK分词器3.0的特性如下：

1）采用了特有的“正向迭代最细粒度切分算法“，具有60万字/秒的高速处理能力。
2）采用了多子处理器分析模式，支持：英文字母（IP地址、Email、URL）、数字（日期，常用中文数量词，罗马数字，科学计数法），中文词汇（姓名、地名处理）等分词处理。
3）对中英联合支持不是很好,在这方面的处理比较麻烦.需再做一次查询,同时是支持个人词条的优化的词典存储，更小的内存占用。
4）支持用户词典扩展定义。
5）针对Lucene全文检索优化的查询分析器IKQueryParser；采用歧义分析算法优化查询关键字的搜索排列组合，能极大的提高Lucene检索的命中率。

5.3 ElasticSearch集成IK分词器

5.3.1 IK分词器的安装

1）下载地址：https://github.com/medcl/elasticsearch-analysis-ik/releases

2）解压，将解压后的elasticsearch文件夹拷贝到`{elasticsearch-home}\plugins`下，并重命名文件夹为analysis-ik

3）重新启动ElasticSearch，即可加载IK分词器

5.3.2 IK分词器测试

IK提供了两个分词算法ik_smart 和 ik_max_word

其中 ik_smart 为最少切分，ik_max_word为最细粒度划分

我们分别来试一下

1）最小切分：在浏览器地址栏输入地址

```
http://127.0.0.1:9200/_analyze?analyzer=ik_smart&pretty=true&text=我是程序员
```

输出的结果为：

```
{
  "tokens" : [
    {
      "token" : "我",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "CN_CHAR",
      "position" : 0
    },
    {
      "token" : "是",
      "start_offset" : 1,
      "end_offset" : 2,
      "type" : "CN_CHAR",
      "position" : 1
    },
    {
      "token" : "程序员",
      "start_offset" : 2,
      "end_offset" : 5,
      "type" : "CN_WORD",
      "position" : 2
    }
  ]
}
```

2）最细切分：在浏览器地址栏输入地址

```
http://127.0.0.1:9200/_analyze?analyzer=ik_max_word&pretty=true&text=我是程序员
```

输出的结果为：

```
{
  "tokens" : [
    {
      "token" : "我",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "CN_CHAR",
      "position" : 0
    },
    {
      "token" : "是",
      "start_offset" : 1,
      "end_offset" : 2,
      "type" : "CN_CHAR",
      "position" : 1
    },
    {
```

```
      "token" : "程序员",
      "start_offset" : 2,
      "end_offset" : 5,
      "type" : "CN_WORD",
      "position" : 2
    },
    {
      "token" : "程序",
      "start_offset" : 2,
      "end_offset" : 4,
      "type" : "CN_WORD",
      "position" : 3
    },
    {
      "token" : "员",
      "start_offset" : 4,
      "end_offset" : 5,
      "type" : "CN_CHAR",
      "position" : 4
    }
  ]
}
```