一、全文检索技术

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。

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)`。

全文检索场景

- 搜索引擎
- 站内搜索
- 系统文件搜索

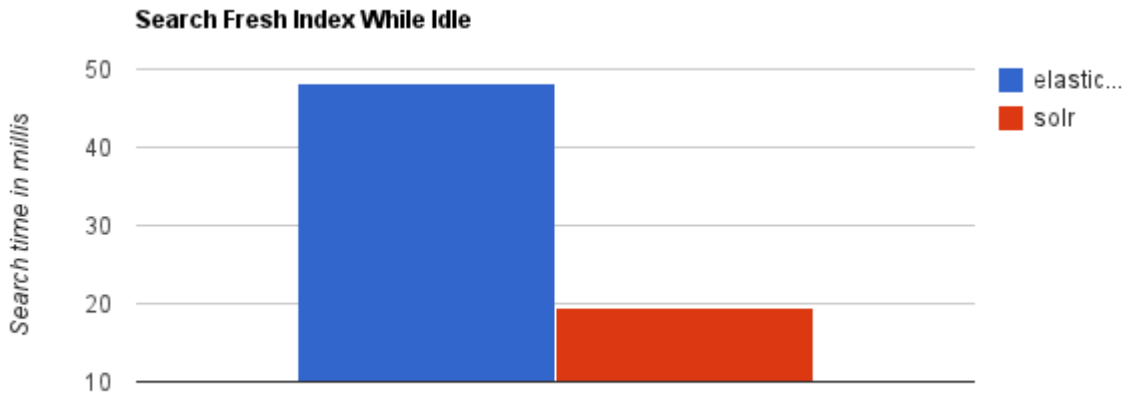
全文检索相关技术

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。

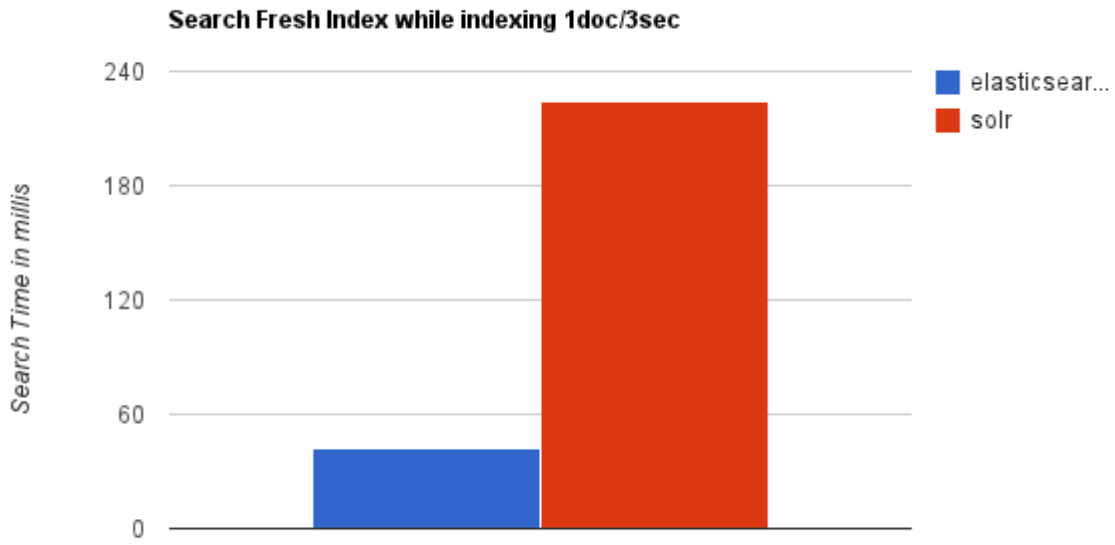
二、Solr和ES的比较

ElasticSearch vs Solr 检索速度

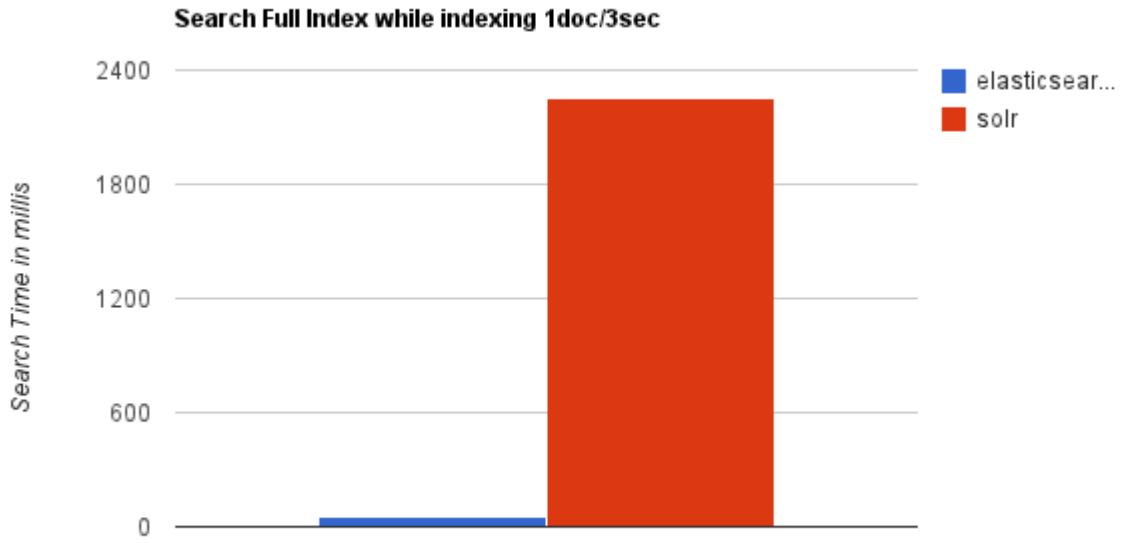
- 当单纯的对已有数据进行搜索时，Solr更快。



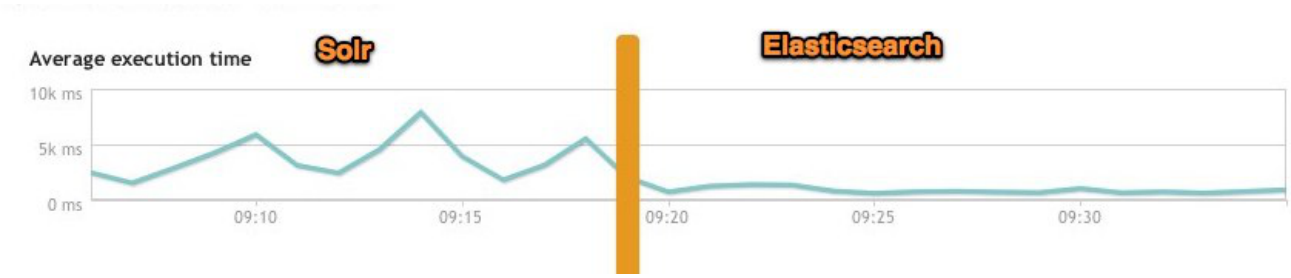
- 当实时建立索引时，Solr会产生io阻塞，查询性能较差，Elasticsearch具有明显的优势。



- 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。



Elasticsearch 与 Solr 的比较总结

- 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。

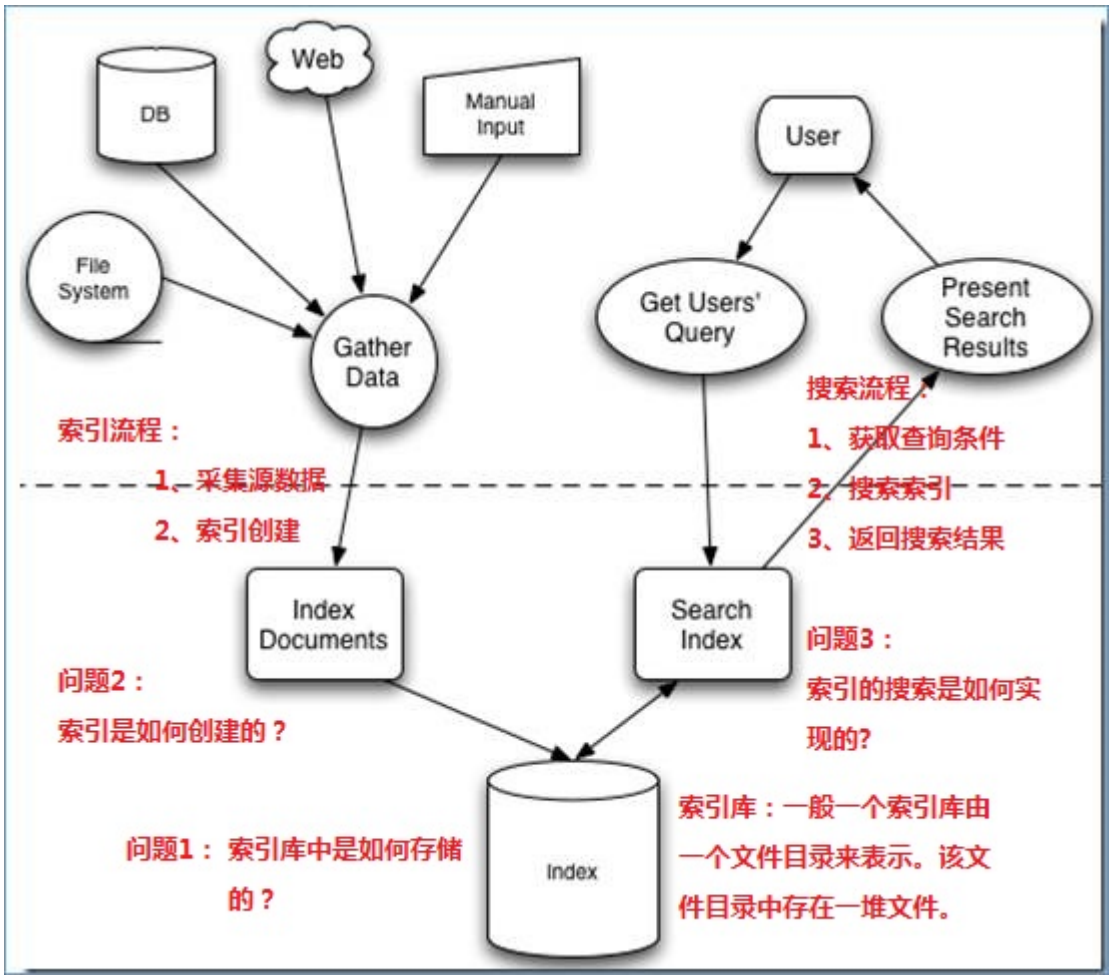
> 最终的结论：
>
> Solr 是传统搜索应用的有力解决方案，但 Elasticsearch 更适用于新兴的实时搜索应用。

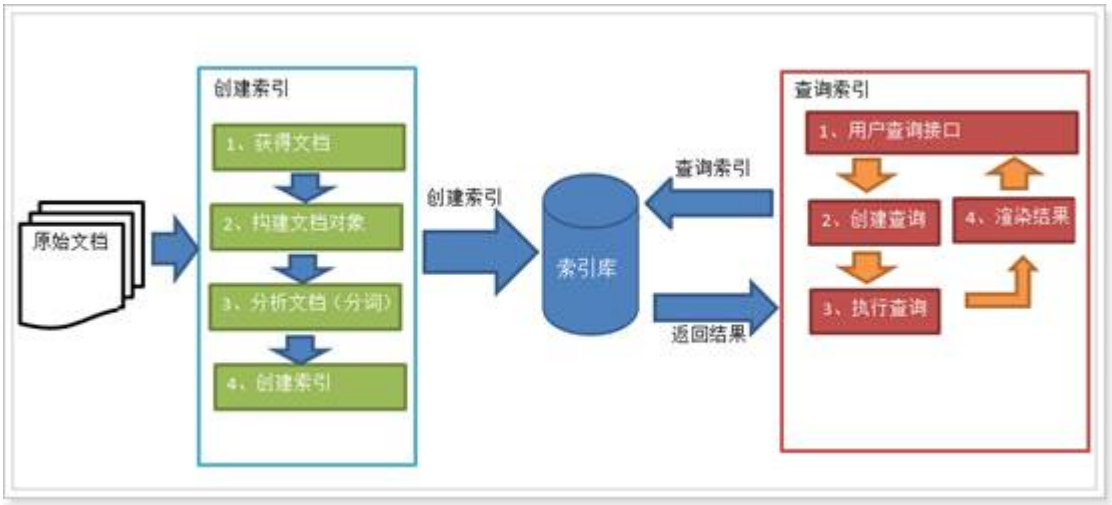
实时搜索与传统搜索

> 通常来说，**传统搜索**都是一些“静态”的搜索，即用户搜索的只是从信息库里边筛选出来的信息。而百度推出的**实时搜索功能**，改变了传统意义上的静态搜索模式，用户对于搜索的结果是实时变化的。
>
> 举个例子，用户在搜索“华山”、“峨眉山”等景点时，实时观看各地景区画面。以华山景区为例，当用户在搜索框中输入“华山”时，点击右侧“实时直播——华山”，即可实时观看华山靓丽风景，并能在华山长空栈道、北峰顶、观日台三个视角之间切换。同时，该直播引入广受年轻人欢迎的“弹幕”模式，用户在观看风景时可以同时发表评论，甚至进行聊天互动。

三、全文检索的流程分析

2.1 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。

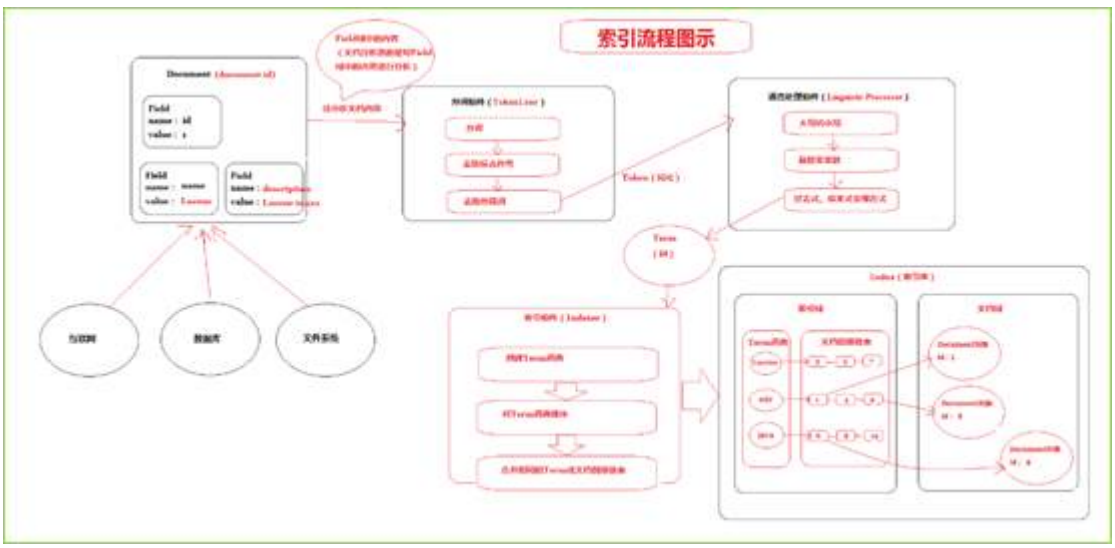
想搞清楚全文检索，必须要搞清楚下面三个问题：

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)

2.2 创建索引流程



一次索引，多次使用。

2.2.1 原始内容

原始内容是指要索引和搜索的内容。

原始内容包括互联网上的网页、数据库中的数据、磁盘上的文件等。

2.2.2 获得文档

也就是采集数据，从互联网上、数据库、文件系统中等获取需要搜索的原始信息，这个过程就是**信息采集。**

采集数据的目的是为了将原始内容存储到Document对象中。

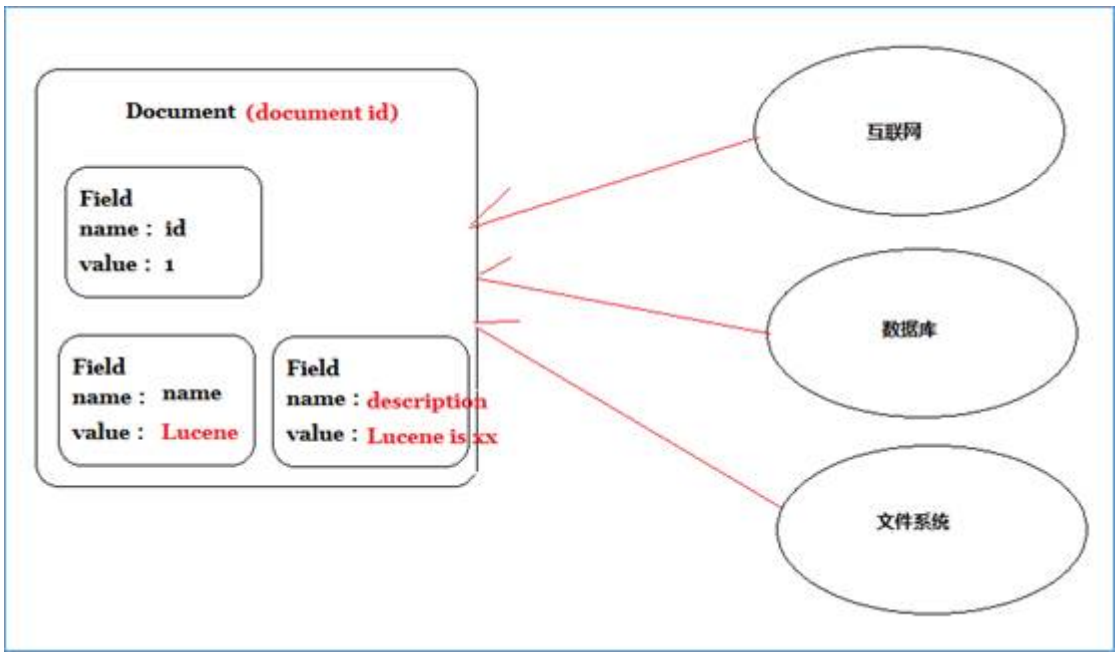
如何采集数据？

1. 对于互联网上网页，可以使用工具将网页抓取到本地生成html文件。
2. 数据库中的数据，可以直接连接数据库读取表中的数据。
3. 文件系统中的某个文件，可以通过I/O操作读取文件的内容。

> 在Internet上采集信息的软件通常称为爬虫或蜘蛛，也称为网络机器人，爬虫访问互联网上的每一个网页，将获取到的网页内容存储起来。

2.2.3 创建文档

创建文档的目的是统一数据格式（Document），方便文档分析。

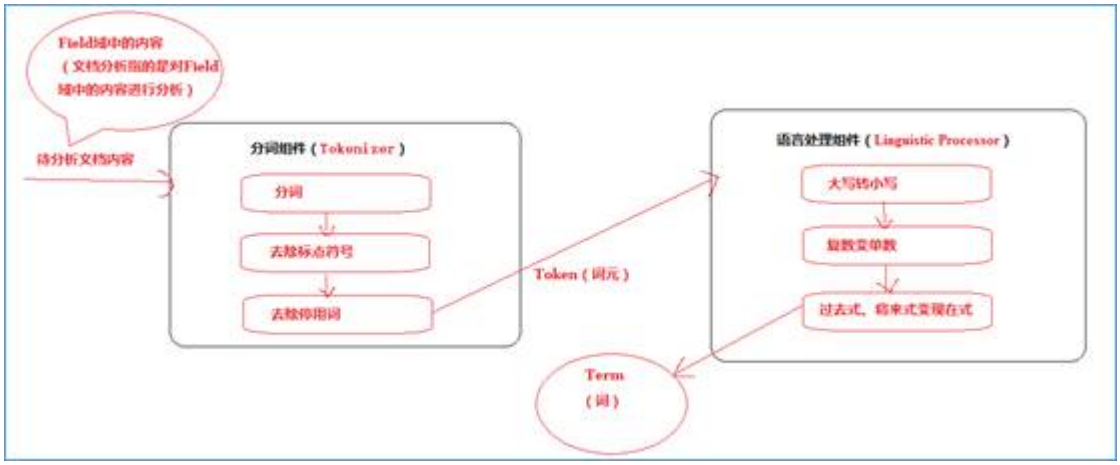


说明：

1. 一个Document文档中包括多个域（Field），域（Field）中存储内容。
2. 这里我们可以将数据库中一条记录当成一个Document，一列当成一个Field。

2.2.4 分析文档（重点）

分析文档主要是对Field域进行分析，分析文档的目的是为了索引。



说明：分析文档主要通过分词组件（Tokenizer）和语言处理组件（Linguistic Processor）完成。

分词组件

分词组件工作流程（此过程称之为Tokenize）

1. 将Field域中的内容进行分词（不同语言有不同的分词规则）。
2. 去除标点符号。
3. 去除停用词（stop word）。

经过分词（Tokenize）之后得到的结果成为**词元（Token）**。

> 所谓停词(Stop word)就是一种语言中最普通的一些单词，由于没有特别的意义，因而大多数情况下不能成为搜索的关键词，因而创建索引时，这种词会被去掉而减少索引的大小。
>
> 英语中停词(Stop word)如：“the”，“a”，“this”等。
>
> 对于每一种语言的分词组件(Tokenizer)，都有一个停词(stop word)集合。

示例(Document1的Field域和Document2的Field域是同名的)：

- Document1的Field域：

```
Students should be allowed to go out with their   friends, but not allowed to drink beer.
```

- Document2的Field域：

```
My friend Jerry went to school to see his students but   found them drunk which is not allowed.
```

- 在我们的例子中，便得到以下词元(Token)：

```
“Students”，“allowed”，“go”，“their”，“friends”，“allowed”，“drink”，“beer”，“My”，“friend”，“Jerry”，“went”，“school”，“see”，“his”，“students”，“found”，“them”，“drunk”，“allowed”。
```

将得到的词元(Token)传给语言处理组件(Linguistic Processor)。

语言处理组件

语言处理组件(linguistic processor)主要是对得到的词元(Token)做一些同语言相关的处理。

对于英语，语言处理组件(Linguistic Processor)一般做以下几点：

1. 变为小写(Lowercase)。
2. 将单词缩减为词根形式，如“cars”到“car”等。这种操作称为：stemming。
3. 将单词转变为词根形式，如“drove”到“drive”等。这种操作称为：lemmatization。

语言处理组件(linguistic processor)的结果称为**词(Term)**。Term是索引库的最小单位。

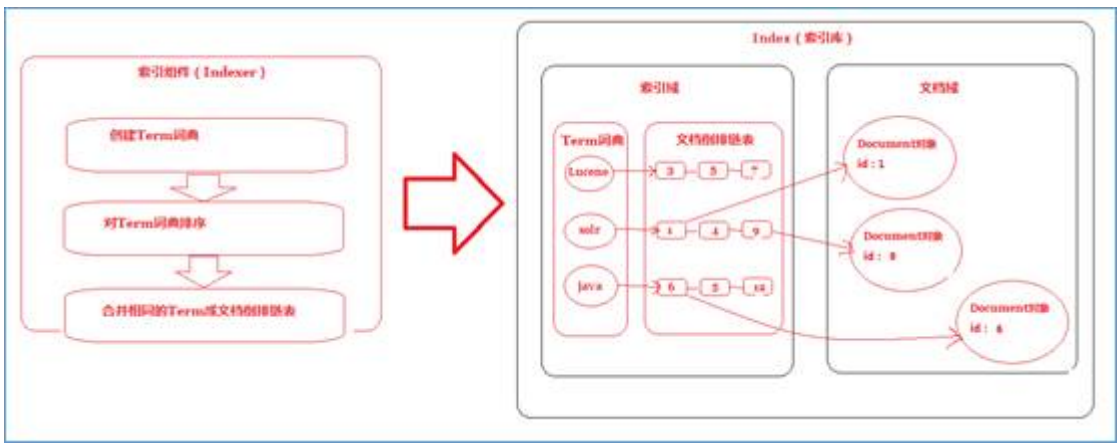
- 在我们的例子中，经过语言处理，得到的词(Term)如下：

```
“student”，“allow”，“go”，“their”，“friend”，“allow”，“drink”，“beer”，“my”，“friend”，“jerry”，“go”，“school”，“see”，“his”，“student”，“find”，“them”，“drink”，“allow”。
```

也正是因为有语言处理的步骤，才能使搜索drove，而drive也能被搜索出来。

2.2.5 索引文档

索引的目的是为了搜索。



说明：将得到的词(Term)传给索引组件(Indexer)，索引组件(Indexer)主要做以下几件事情：

创建Term字典

在我们的例子中字典如下：

Term	Document ID
Student	1
Allow	1
Go	1
Their	1
Friend	1
Allow	1
Drink	1
Beer	1
My	2
Friend	2
Jerry	2
Go	2
School	2
See	2
His	2
Student	2
Find	2
Them	2
Drink	2
Allow	2

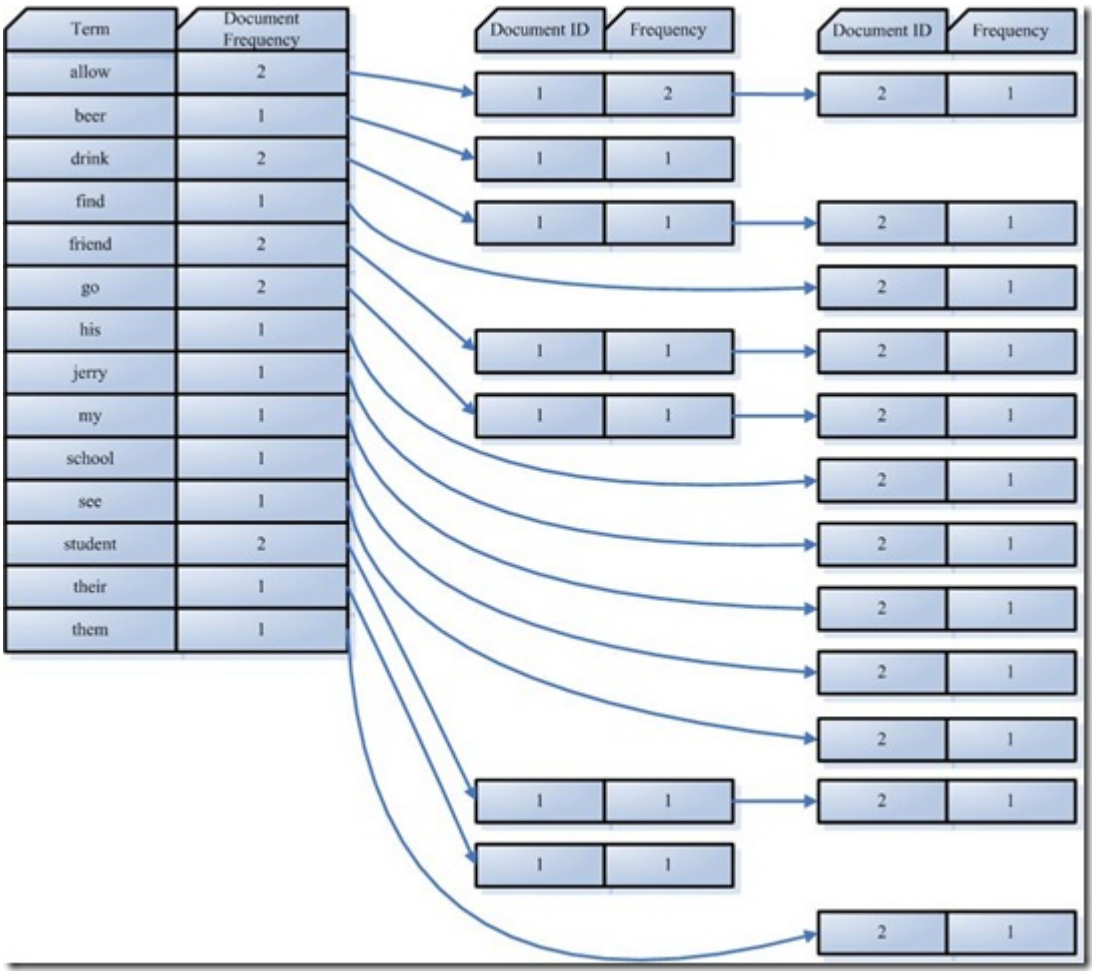
排序Term字典

对字典按字母顺序进行排序

Term	Document ID
Allow	1
Allow	1
Allow	2
Beer	1
Drink	1
Drink	2
Find	2
Friend	1
Friend	2
Go	1
Go	2
His	2
Jerry	2
My	2
School	2
See	2
Student	1
Student	2
Their	1
Them	2

合并Term字典

合并相同的词(Term)成为文档倒排(Posting List)链表



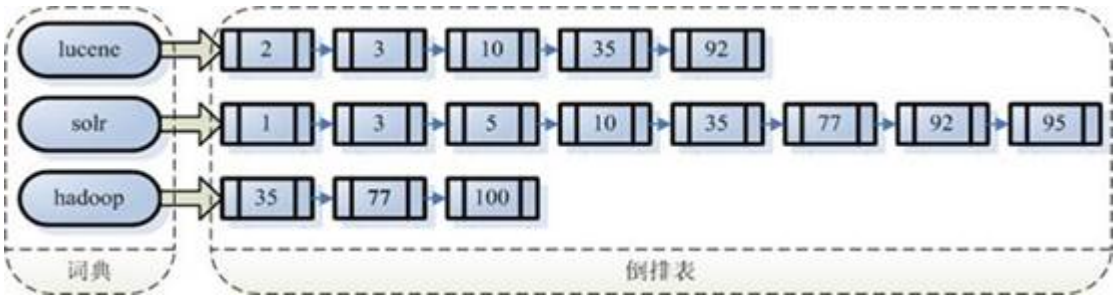
在此表中，有几个定义：

- `Document Frequency` 即文档频次，表示总共有多少文件包含此词(Term)。
- `Frequency` 即词频率，表示此文件中包含了几个此词(Term)。

到此为止，索引已经创建好了。

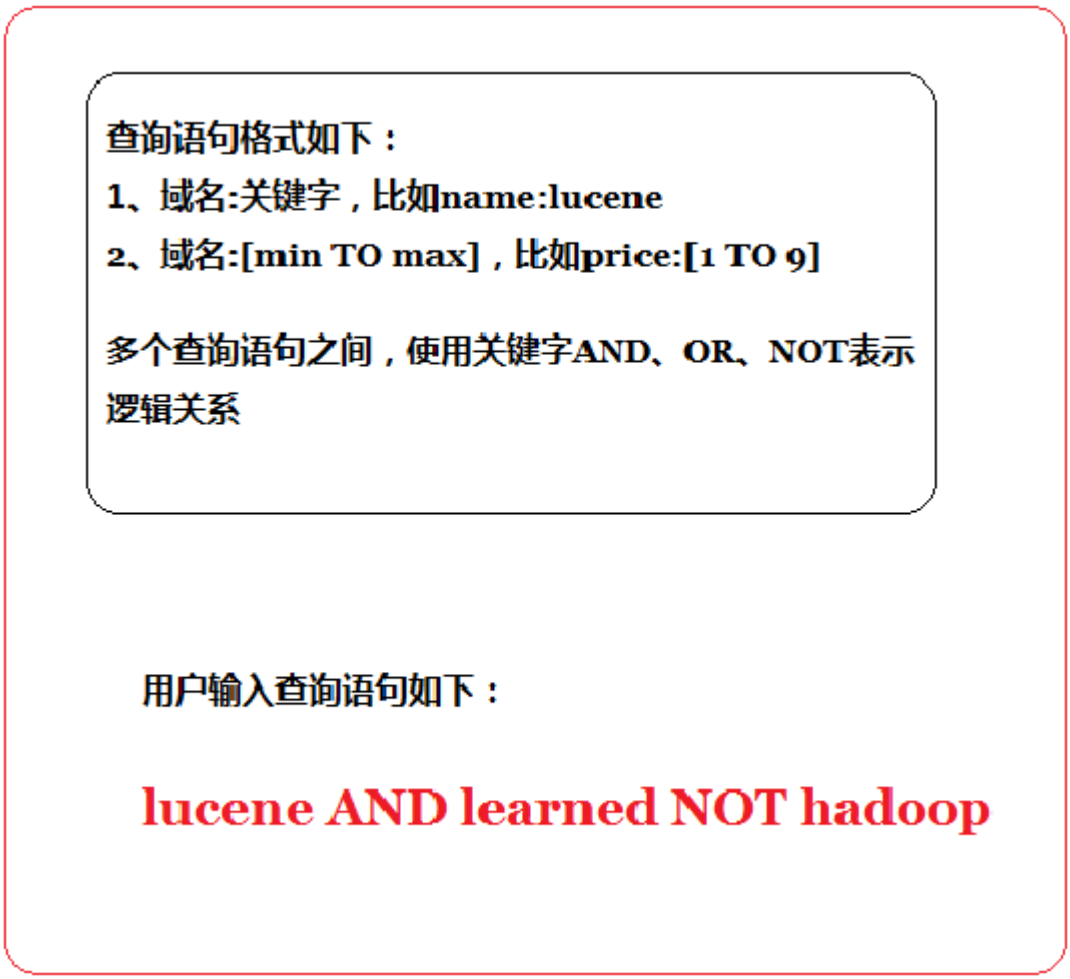
最终的索引结构是一种倒排索引结构也叫反向索引结构，包括索引和文档两部分，索引即词汇表，它的规模较小，而文档集合较大。

倒排索引结构是根据内容（词汇）找文档，如下图：



2.3 搜索索引流程

2.3.1 图1：查询语句



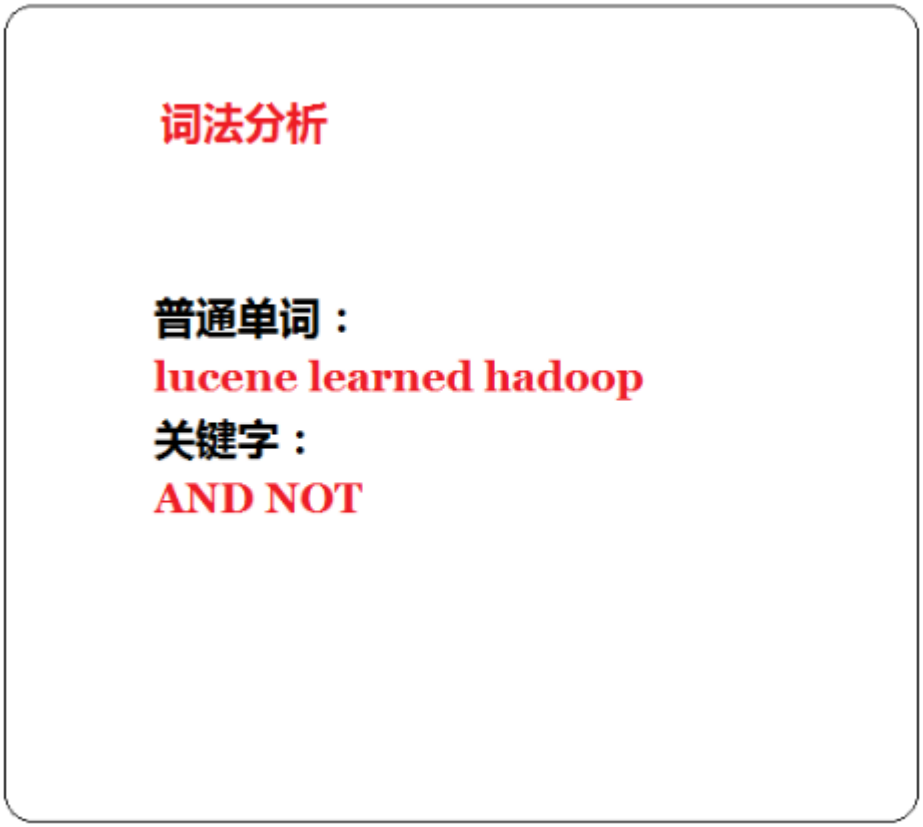
2.4.2 图2：执行搜索

第一步：对查询语句进行词法分析、语法分析及语言处理。

1、词法分析

如上述例子中，经过词法分析，得到单词有lucene，learned，hadoop，关键字有AND，NOT。

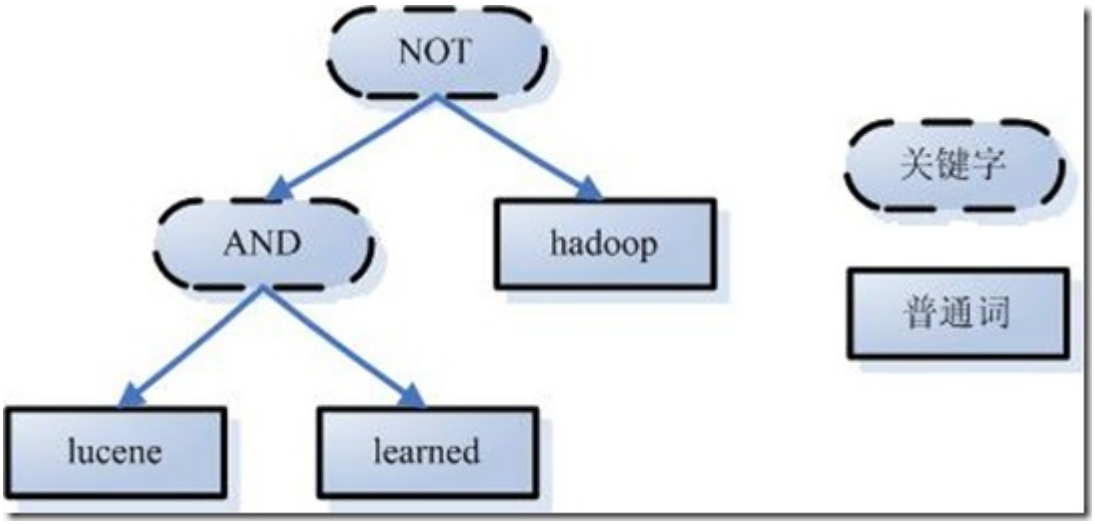
注意：关键字必须大写，否则就作为普通单词处理。



2、语法分析

如果发现查询语句不满足语法规则，则会报错。如`lucene NOT AND learned`，则会出错。

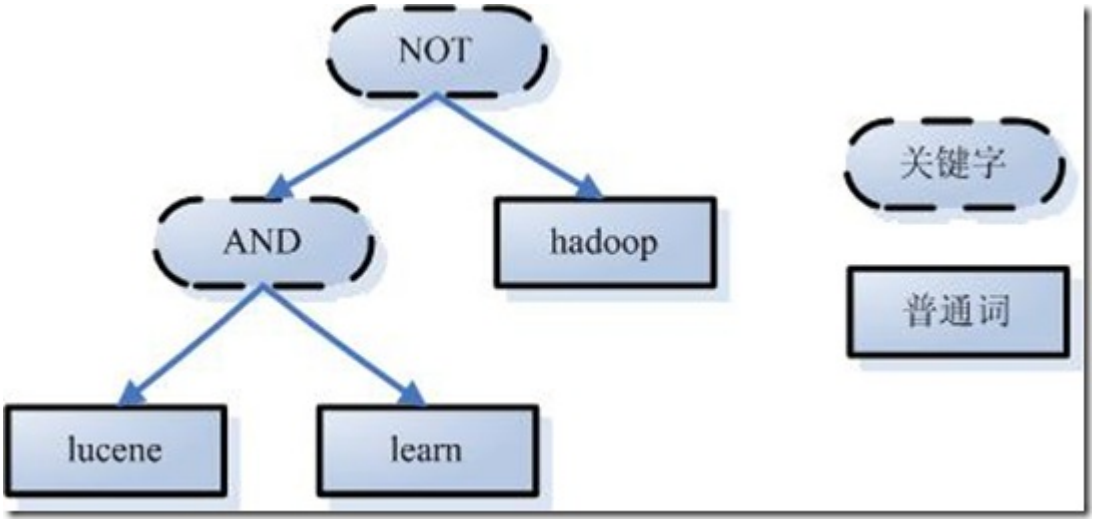
如上述例子，`lucene AND learned NOT hadoop`形成的语法树如下：



3、语言处理

如`learned`变成`learn`等。

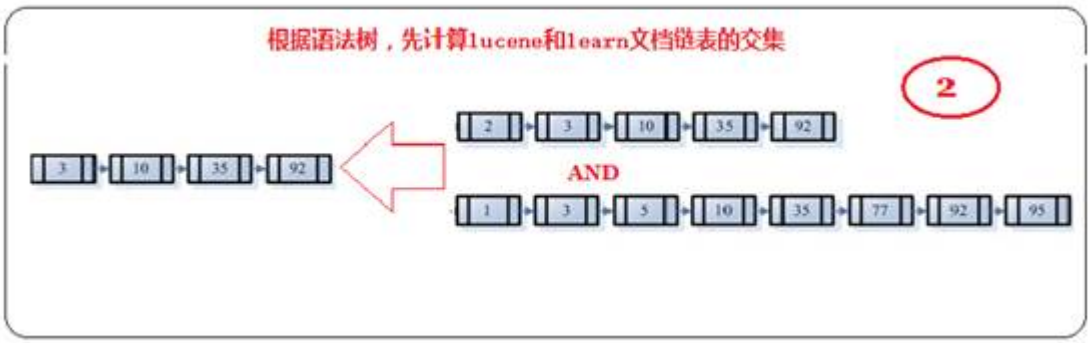
经过第二步，我们得到一棵经过语言处理的语法树。



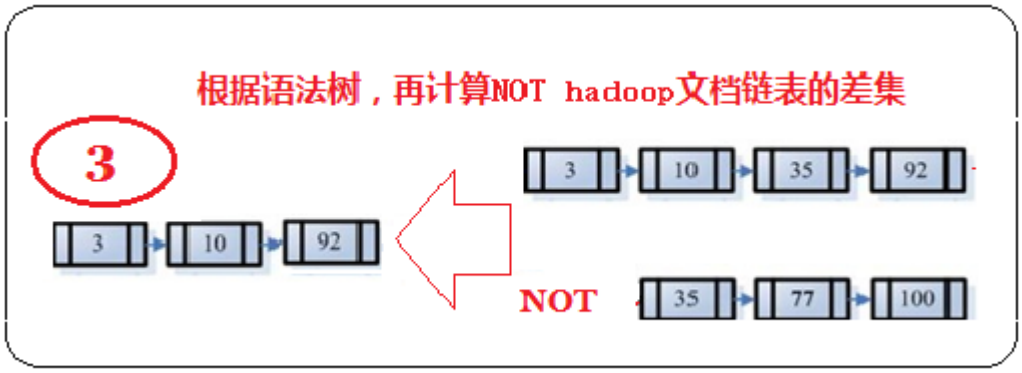
第二步：搜索索引，得到符号语法树的文档。

1、 首先，在反向索引表中，分别找出包含lucene，learn，hadoop的文档链表。

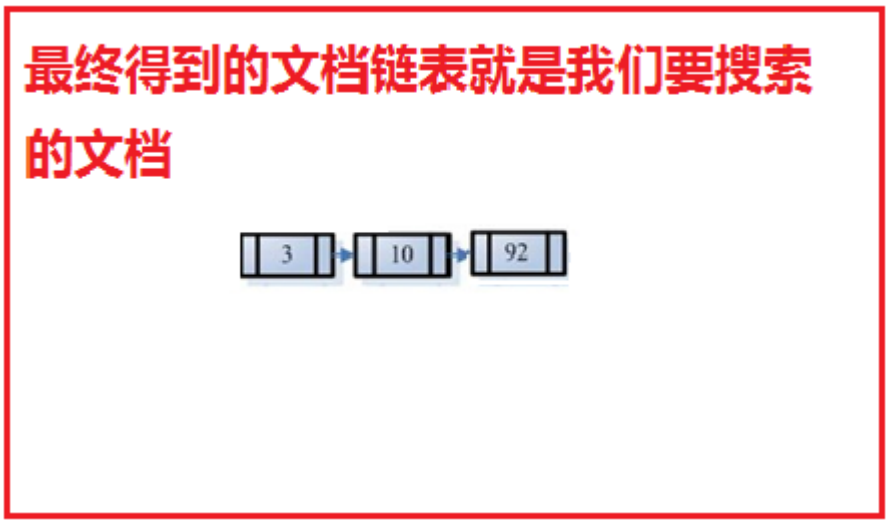
2、 其次，对包含lucene，learn的链表进行合并操作，得到既包含lucene又包含learn的文档链表。



3、 然后，将此链表与hadoop的文档链表进行差操作，去除包含hadoop的文档，从而得到既包含lucene又包含learn而且不包含hadoop的文档链表。



4、 此文档链表就是我们要找的文档。



第三步：根据得到的文档和查询语句的相关性，对结果进行排序。

相关度自然打分（权重越高分越高）：

tf越高、权重越高

df越高、权重越低

人为影响分数：

设置Boost值（加权值）

2.4.3 Lucene相关度排序

2.4.3.1 什么是相关度排序

相关度排序是 查询结果 按照与 查询关键字 的相关性进行排序，越相关的越靠前。比如搜索“Lucene”关键字，与该关键字最相关的文章应该排在前边。

2.4.3.2 相关度打分

Lucene对查询关键字和索引文档的相关度进行打分，得分高的就排在前边。

如何打分呢？Lucene是在用户进行检索时实时根据搜索的关键字计算出来的，分两步：

1. 计算出词（Term）的权重
2. 根据词的权重值，计算文档相关度得分。

什么是词的权重？

通过索引部分的学习，明确索引的最小单位是一个Term(索引词典中的一个词)。搜索也是从索引域中查询Term，再根据Term找到文档。**Term对文档的重要性称为权重**，影响Term权重有两个因素：

- **Term Frequency (tf)**：

指此Term在此文档中出现了多少次。**tf 越大说明越重要。**

词(Term)在文档中出现的次数越多，说明此词(Term)对该文档越重要，如“Lucene”这个词，在文档中出现的次数很多，说明该文档主要就是讲Lucene技术的。

- **Document Frequency (df)**：

指有多少文档包含此Term。**df 越大说明越不重要。**

比如，在一篇英语文档中，this出现的次数更多，就说明越重要吗？不是的，有越多的文档包含此词(Term)，说明此词(Term)太普通，不足以区分这些文档，因而重要性越低。

2.4.3.3 设置boost值影响相关度排序

boost是一个加权值（默认加权值为1.0f），它可以影响权重的计算。在索引时对某个文档中的field设置加权值，设置越高，在搜索时匹配到这个文档就可能排在前边。

四、Lucene应用代码

所需依赖

```
    <dependencies>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-queryparser</artifactId>
            <version>7.5.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.lucene</groupId>
            <artifactId>lucene-analyzers-common</artifactId>
            <version>7.5.0</version>
        </dependency>

        <!-- 目的是为了数据采集 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.35</version>
        </dependency>


        <!-- 可以自己安装，也可以使用中央仓库 -->
```

```
        <dependency>
            <groupId>com.janeluo</groupId>
            <artifactId>ikanalyzer</artifactId>
            <version>2012_u6</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

索引流程代码

```
public class IndexDemo {

    public static void main(String[] args) throws Exception {

        // 1. 数据采集
        ItemDao itemDao = new ItemDaoImpl();
        List<Item> itemList = itemDao.queryItemList();

        // 2. 创建Document文档对象
        List<Document> documents = new ArrayList<>();
        for (Item item : itemList) {
            Document document = new Document();

            // Document文档中添加Field域
            // 商品Id
            // Store.YES:表示存储到文档域中
            document.add(new TextField("id", item.getId().toString(), Store.YES));
            // 商品名称
            document.add(new TextField("name", item.getName().toString(),
Store.YES));
            // 商品价格
            document.add(new TextField("price", item.getPrice().toString(),
Store.YES));
            // 商品图片地址
            document.add(new TextField("pic", item.getPic().toString(), Store.YES));
            // 商品描述
```

```
            document.add(new TextField("description",
item.getDescription().toString(), Store.YES));

            // 把Document放到list中
            documents.add(document);
        }

        // 指定分词器：标准分词器（此处可以改为中文分词器）
        // Analyzer analyzer = new StandardAnalyzer();
        Analyzer analyzer = new IKAnalyzer();
        // 配置文件
        IndexWriterConfig iwc = new IndexWriterConfig(analyzer);

        // 指定索引库路径
        String indexPath = "E:\\11-index\\vip01\\";
        // 指定索引库对象
        Directory dir = FSDirectory.open(Paths.get(indexPath));
        // 创建索引写对象
        IndexWriter writer = new IndexWriter(dir, iwc);
        // 3. 分词并创建索引文件
        writer.addDocuments(documents);

        // 释放资源
        writer.close();

    }
}
```

Luke工具

Luke作为Lucene工具包中的一个工具，可以通过界面来进行索引文件的查询、修改。

搜索流程代码

```
public class SearchDemo {

    public static void main(String[] args) throws Exception {
        // 指定索引库路径
        String indexPath = "E:\\11-index\\vip01\\";
        // 指定索引库对象
        Directory dir = FSDirectory.open(Paths.get(indexPath));

        // 索引读对象
        IndexReader reader = DirectoryReader.open(dir);
        // 索引搜索器
        IndexSearcher searcher = new IndexSearcher(reader);

        Analyzer analyzer = new StandardAnalyzer();
        // 通过QueryParser解析查询语法，获取Query对象
        QueryParser parser = new QueryParser("description", analyzer);
```

```
        // 参数是查询语法
        Query query = parser.parse("lucene");
        TopDocs topDocs = searcher.search(query, 100);

        ScoreDoc[] scoreDocs = topDocs.scoreDocs;

        for (ScoreDoc scoreDoc : scoreDocs) {
            Document document = searcher.doc(scoreDoc.doc);
            System.out.println("name : "+document.get("name"));
        }

        reader.close();
    }

}
```

五、Lucene的Field域

1.1. Field属性

Field是文档中的域，包括Field名和Field值两部分，一个文档可以包括多个Field，Document只是Field的一个承载体，Field值即为要索引的内容，也是要搜索的内容。

- 是否分词(tokenized)
  - 是：作分词处理，即将Field值进行分词，分词的目的是为了索引。

    比如：商品名称、商品描述等，这些内容用户要输入关键字搜索，由于搜索的内容格式大、内容多需要分词后将语汇单元建立索引
  - 否：不作分词处理

    比如：商品id、订单号、身份证号等
- 是否索引(indexed)
  - 是：进行索引。将Field分词后的词或整个Field值进行索引，存储到索引域，索引的目的是为了搜索。

    比如：商品名称、商品描述分析后进行索引，订单号、身份证号不用分词但也要索引，这些将来都要作为查询条件。
  - 否：不索引。

    比如：图片路径、文件路径等，不用作为查询条件的不用索引
- 是否存储(stored)
  - 是：将Field值存储在文档域中，存储在文档域中的Field才可以从Document中获取。

    比如：商品名称、订单号，凡是将来要从Document中获取的Field都要存储。
  - 否：不存储Field值

    比如：**商品描述**，内容较大不用存储。如果要向用户展示商品描述可以从系统的关系数据库中获取。

1.2. Field常用类型

下边列出了开发中常用 的Filed类型，注意Field的属性，根据需求选择：

Field类	数据类型	Analyzed是否分词	Indexed是否索引	Stored是否存储	说明
StringField(FieldName, FieldValue,Store.YES))	字符串	N	Y	Y或N	这个Field用来构建一个字符串Field，但是不会进行分词，会将整个串存储在索引中，比如(订单号,身份证号等）是否存储在文档中用Store.YES或Store.NO决定
LongField(FieldName, FieldValue,Store.YES)	Long型	Y	Y	Y或N	这个Field用来构建一个Long数字型Field，进行分词和索引，比如(价格）是否存储在文档中用Store.YES或Store.NO决定
StoredField(FieldName, FieldValue)	重载方法，支持多种类型	N	N	Y	这个Field用来构建不同类型Field 不分析，不索引，但要Field存储在文档中
TextField(FieldName, FieldValue, Store.NO)或TextField(FieldName, reader)	字符串或流	Y	Y	Y或N	如果是一个Reader, lucene猜测内容比较多,会采用Unstored的策略.

1.3. Field设计

Field域如何设计，取决于需求，比如搜索条件有哪些？显示结果有哪些？

- 商品id：
  是否分词：不用分词，因为不会根据商品id来搜索商品
  是否索引：不索引，因为不需要根据商品ID进行搜索
  是否存储：要存储，因为查询结果页面需要使用id这个值。
- 商品名称：
  是否分词：要分词，因为要根据商品名称的关键词搜索。
  是否索引：要索引。
  是否存储：要存储。
- 商品价格：

是否分词：要分词，lucene对数字型的值只要有搜索需求的都要分词和索引，因为lucene对数字型的内容要特殊分词处理，需要分词和索引。

是否索引：要索引

是否存储：要存储

- 商品图片地址：

  是否分词：不分词

  是否索引：不索引

  是否存储：要存储

- 商品描述：

  是否分词：要分词

  是否索引：要索引

  是否存储：因为商品描述内容量大，不在查询结果页面直接显示，不存储。

常见问题：

不存储是指不在lucene的索引域中记录，目的是为了节省lucene的索引文件空间。

> 如果要在详情页面显示描述，解决方案：
>
> 从lucene中取出商品的id，根据商品的id查询关系数据库（MySQL）中item表得到描述信息。

六、中文分词器IkAnalyzer

什么是中文分词

学过英文的都知道，英文是以单词为单位的，单词与单词之间以空格或者逗号句号隔开。所以对于英文，我们可以简单以空格判断某个字符串是否为一个单词，比如I love China，love 和 China很容易被程序区分开来。

而中文则以字为单位，字又组成词，字和词再组成句子。中文“**詹哥在开课吧讲课**”就不一样了，电脑不知道“**詹哥**”是一个词语还是“**哥在**”是一个词语。

把中文的句子切分成有意义的词，就是中文分词，也称切词。

使用IkAnalyzer

IKAnalyzer继承Lucene的Analyzer抽象类，使用IKAnalyzer和Lucene自带的分析器方法一样，将Analyzer测试代码改为IKAnalyzer测试中文分词效果。

如果使用中文分词器ik-analyzer，就需要在索引和搜索程序中使用一致的分词器：IK-analyzer。

添加依赖

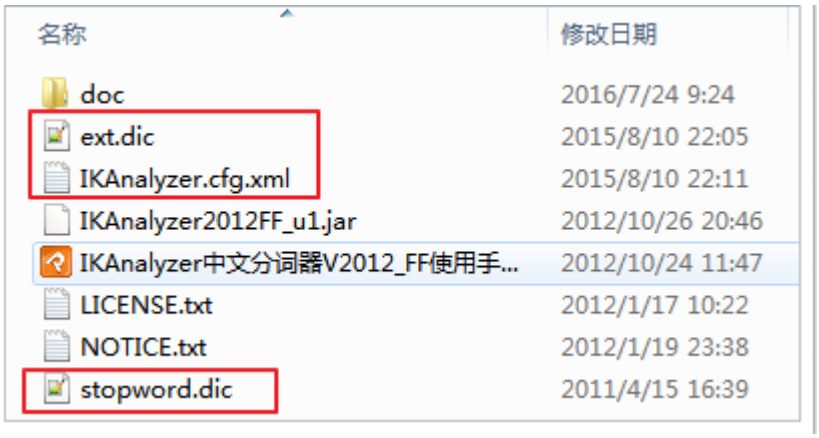
```
    <dependency>
        <groupId>com.janeluo</groupId>
        <artifactId>ikanalyzer</artifactId>
        <version>2012_u6</version>
    </dependency>
```

修改代码

```
Analyzer analyzer = new IKAnalyzer();
```

扩展中文词库

- 第一步：从IkAnalyzer的资料包中，拷贝以下三个文件到目标项目的资源目录下



- 第二步：修改ext.dic文件，添加扩展词。

  注意事项：

  - 一行就是一个词
  - 最好使用IDE内置的文本编辑器进行编辑。
- 第三步：删除索引库中的文件，重新创建索引数据。

七、Solr介绍

7.1. 什么是solr

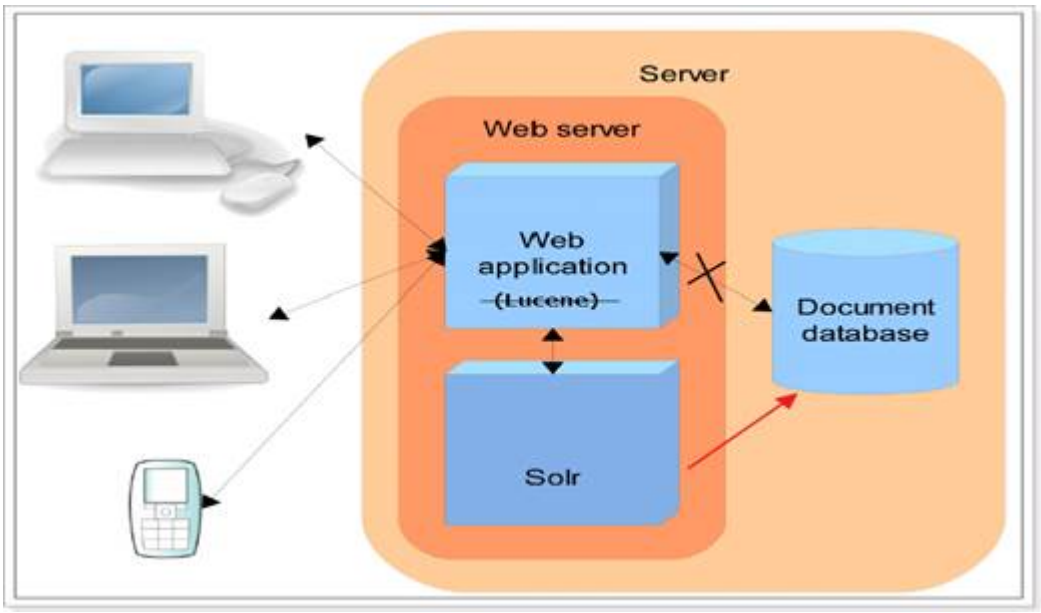
> Solr是一个独立的企业级搜索应用服务器，它对外提供类似于Web-service的API接口。
>
> - 用户可以通过HTTP的POST请求，向Solr服务器提交一定格式的XML或者JSON文件，Solr服务器解析文件之后，根据具体需求对索引库执行增删改操作；
> - 用户可以通过HTTP的GET请求，向Solr服务器发送搜索请求，并得到XML/JSON格式的返回结果。
>
> Solr 是Apache下的一个顶级开源项目，采用Java开发，基于Lucene。
>
> Solr可以独立运行在Jetty、Tomcat等这些Servlet容器中。
>
> Solr提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展，并对索引、搜索性能进行了优化。

7.2. Solr和Lucene的区别

Lucene是一个开放源代码的全文检索引擎工具包，它不是一个完整的全文检索应用。

Lucene仅提供了完整的查询引擎和索引引擎，目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者以Lucene为基础构建全文检索应用。

Solr的目标是打造一款企业级的搜索引擎系统，它是基于Lucene一个搜索引擎服务器，可以独立运行，通过Solr可以非常快速的构建企业的搜索引擎，通过Solr也可以高效的完成站内搜索功能。



八、Solr安装配置

下载安装

- 第一步：下载solr压缩包

```
wget http://archive.apache.org/dist/lucene/solr/4.10.4/solr-4.10.4.tgz
```

- 第二步：解压缩

```
tar -xf solr-4.10.4.tgz
```

默认使用Jetty部署

Solr默认提供Jetty（java写的Servlet容器）启动solr服务器。

使用jetty启动：

```
1. 进入example目录
2. 执行命令：java -jar start.jar
3. 访问地址：http://192.168.10.136:8983/solr
```

但是企业中一般使用Tomcat作为服务器，所以下面我们一起来看看如何将solr部署在tomcat中。

管理界面功能介绍

1.1.1. Dashboard

仪表盘，显示了该Solr实例开始启动运行的时间、版本、系统资源、jvm等信息。

1.1.2. Logging

Solr运行日志信息

1.1.3. Cloud

Cloud即SolrCloud，即Solr云（集群），当使用Solr Cloud模式运行时会显示此菜单，该部分功能在第二个项目，即电商项目会演示。

1.1.4. Core Admin

Solr Core的管理界面。在这里可以添加SolrCore实例（有bug，不推荐使用浏览器界面添加SolrCore）。

1.1.5. java properties

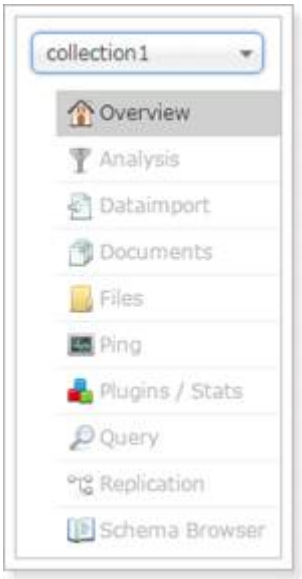
Solr在JVM 运行环境中的属性信息，包括类路径、文件编码、jvm内存设置等信息。

1.1.6. Tread Dump

显示Solr Server中当前活跃线程信息，同时也可以跟踪线程运行栈信息。

1.1.7. Core selector

选择一个SolrCore进行详细操作，如下：



1.1.7.1. Analysis

通过此界面可以测试索引分析器和搜索分析器的执行情况

1.1.7.2. dataimport

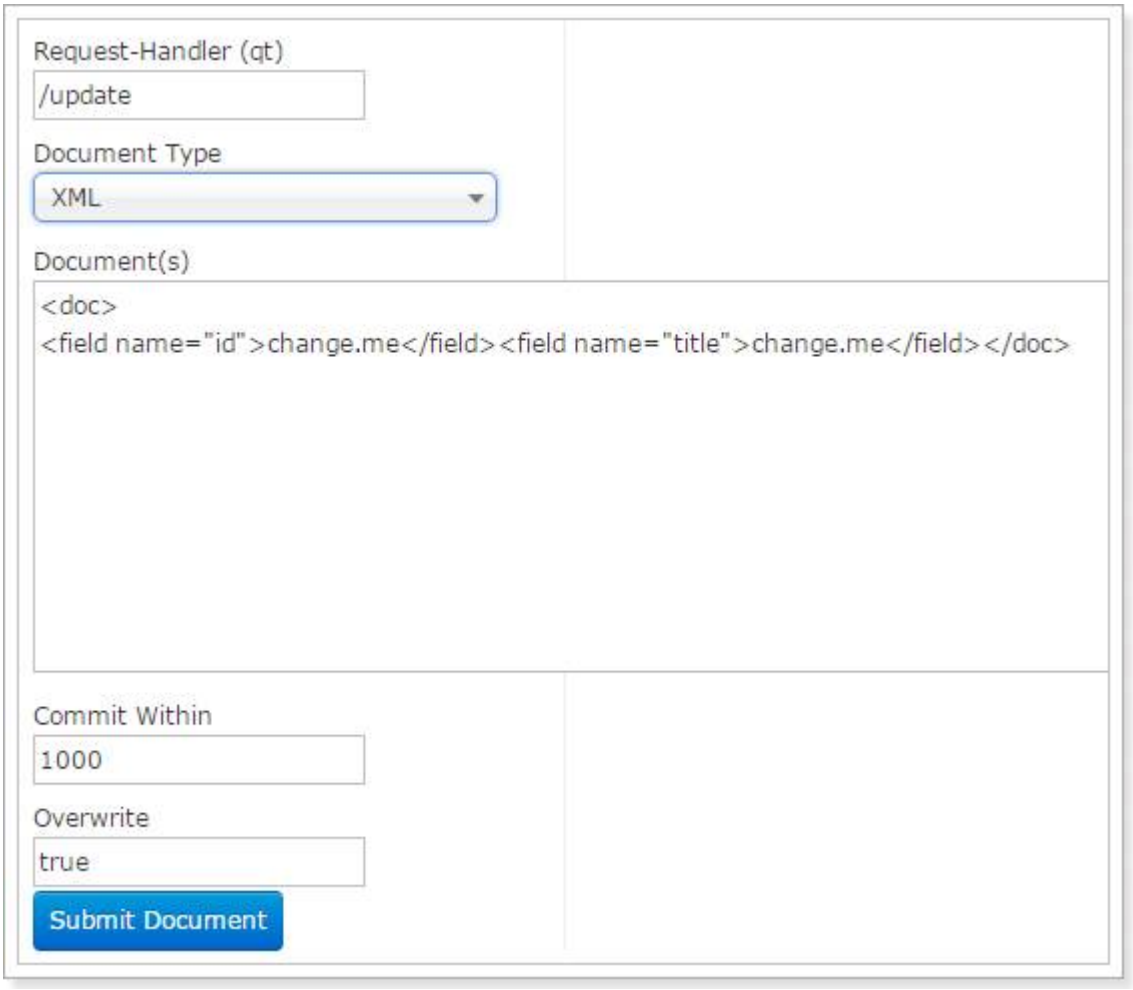
可以定义数据导入处理器，从关系数据库将数据导入到Solr索引库中。

默认没有配置，需要手工配置。

1.1.7.3. Document

通过/update表示更新索引，solr默认根据id（唯一约束）域来更新Document的内容，如果根据id值搜索不到id域则会执行添加操作，如果找到则更新。

通过此菜单可以创建索引、更新索引、删除索引等操作，界面如下：



l overwrite="true" ： solr在做索引的时候，如果文档已经存在，就用xml中的文档进行替换

l commitWithin="1000" ： solr 在做索引的时候，每隔1000（1秒）毫秒，做一次文档提交。为了方便测试也可以在Document中立即提交，后添加 “”

1.1.7.4. Query

通过/select执行搜索索引，必须指定“q”查询条件方可搜索。

手动使用Tomcat部署

配置solrhome

solrhome和solrcore

solrhome是Solr服务器运行时需要的主工作目录，Solr服务的所有配置信息包括索引库都是在该目录下。

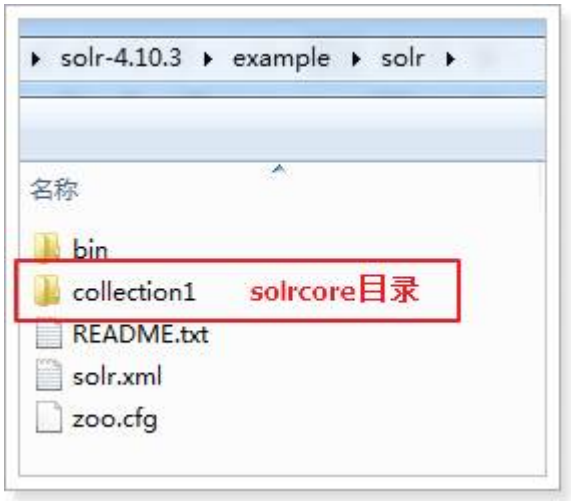
solrhome目录中包括了多个solrcore目录。

solrhome和solrcore是Solr服务器中最重要的两个目录。

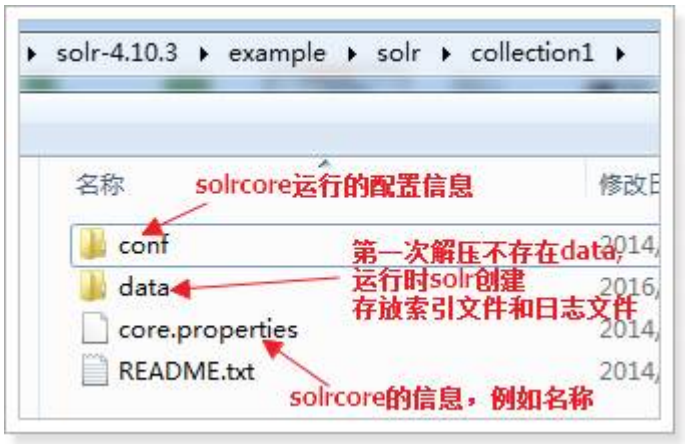
solrcore就是collection1目录，该目录中包含搜索和索引时需要的配置文件和数据文件（比如索引库中的文件）。

每个solrcore都可以提供单独的搜索和索引服务。

solrhome目录：

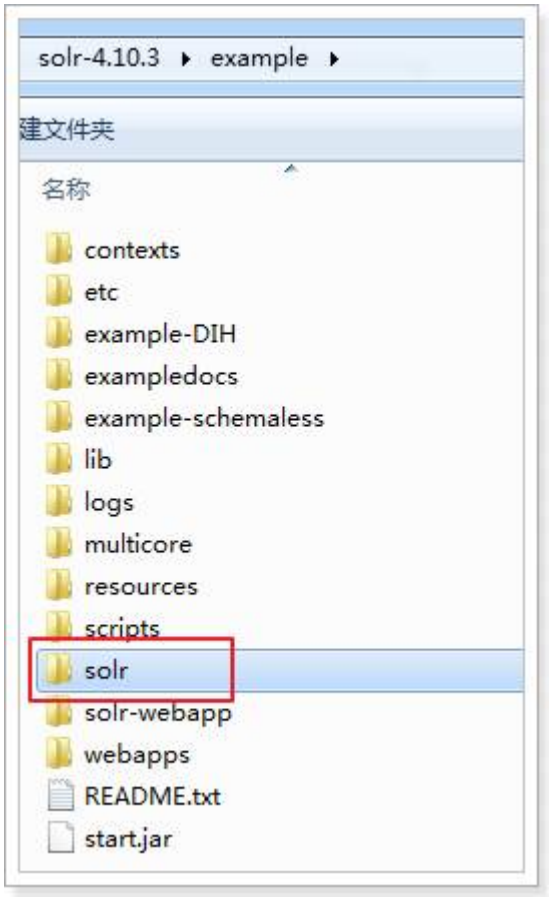


solrcore目录：



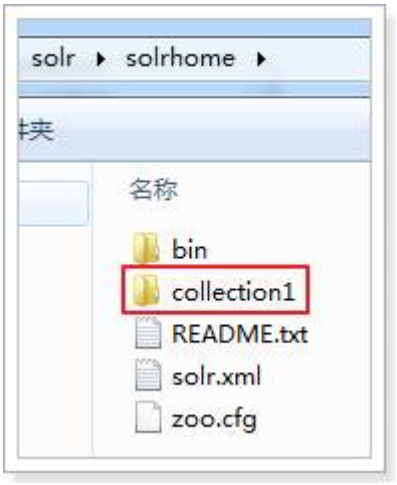
创建SolrHome和SolrCore

下图中的solr目录就是一个solrhome目录，它就是jetty启动的solr服务使用的solrhome目录。



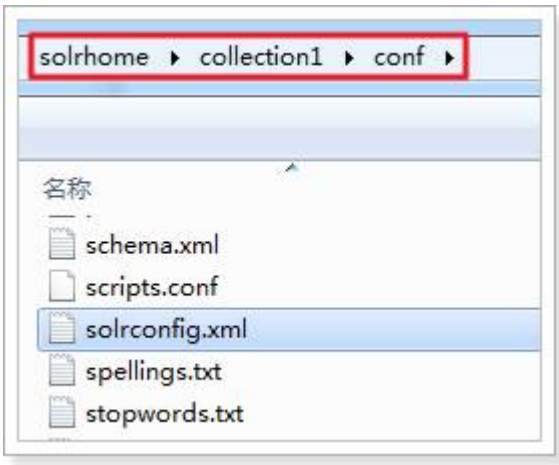
复制该文件夹到本地的一个目录，把文件名称改为solrhome。（*改名不是必须的，只是为了便于理解*）

打开solrhome目录确认是否拥有solrcore



配置solrconfig.xml（了解）

其实就是配置SolrCore目录下的conf/solrconfig.xml。



这个文件是来配置SolrCore实例的相关运行时信息。**如果使用默认配置可以不用做任何修改。**该配置文件中包含了不少标签，但是我们经常使用的标签有：**lib标签、datadir标签、requestHandler标签。**

1.1.3.1. lib 标签

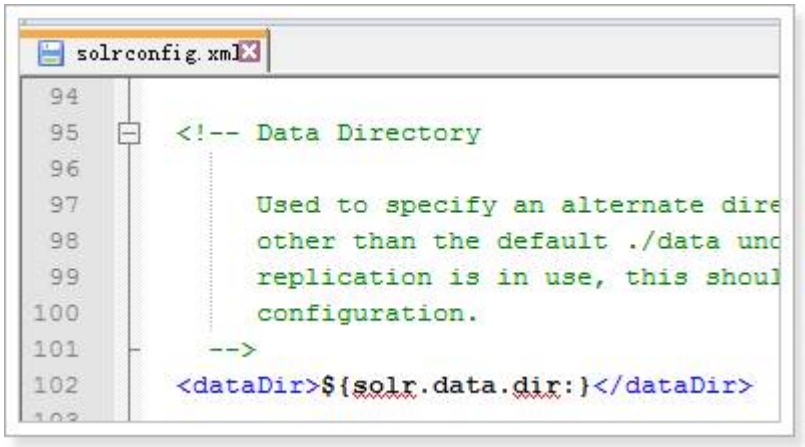
lib标签可以配置扩展功能的一些jar，用以增强solr本身没有的功能。

比如solr自身没有『数据导入索引库』功能，如果需要使用，则首先要把这些jar复制到指定的目录，然后通过该配置文件进行相关配置，后面会具体讲解如何配置。

1.1.3.2. datadir标签

dataDir数据目录data。data目录用来存放索引文件和tlog日志文件。

solr.data.dir表示${SolrCore}/data的目录位置



如果不想使用默认的目录也可以通过solrconfig.xml更改索引目录

例如：

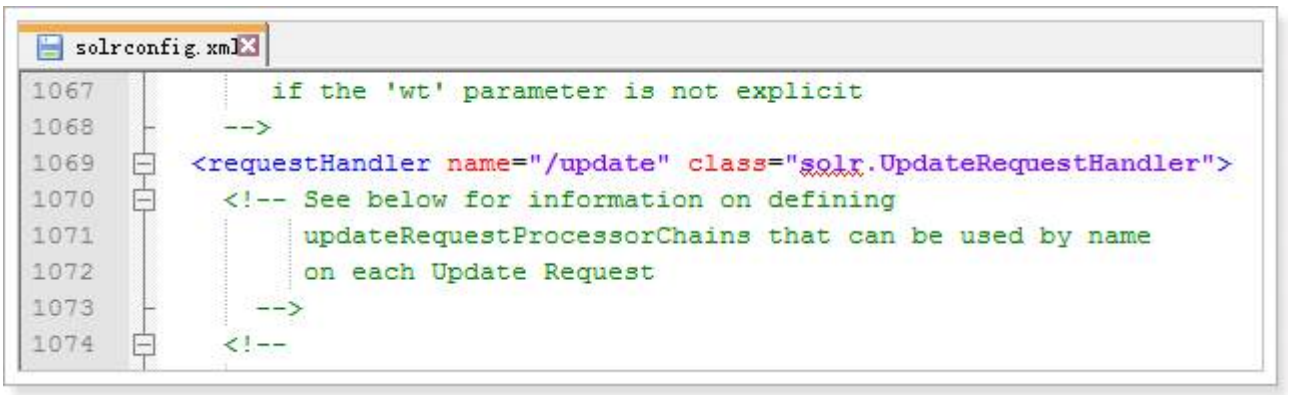
```
<dataDir>${solr.data.dir:F:/develop/solr/collection1/data}</dataDir>
```

（建议不修改，否则配置多个SolrCore会报错）

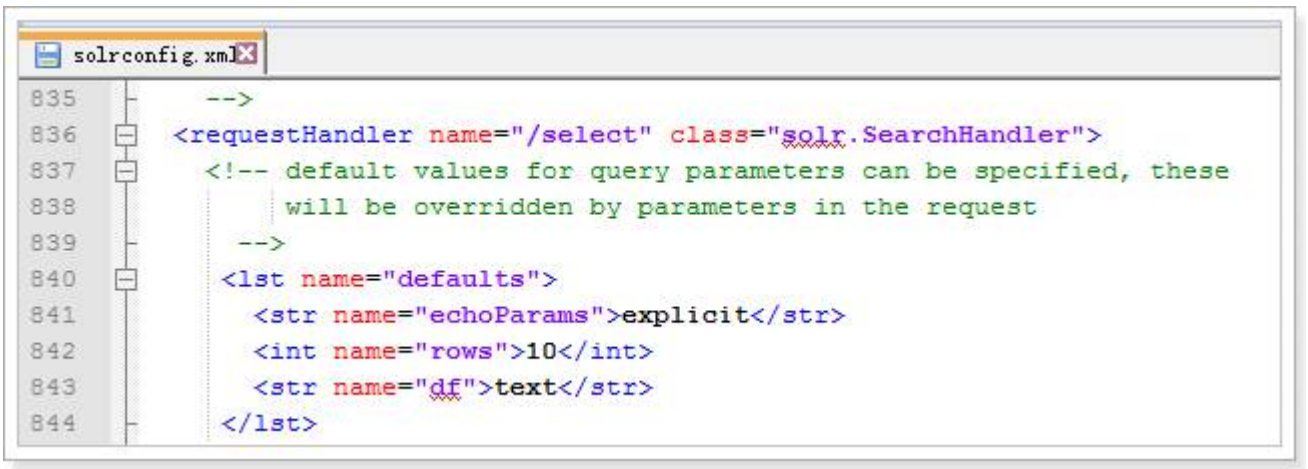
1.1.3.3. requestHandler标签

requestHandler请求处理器，定义了索引和搜索的访问方式。

通过/update维护索引，可以完成索引的添加、修改、删除操作。



通过/select搜索索引。



设置搜索参数完成搜索，搜索参数也可以设置一些默认值，如下：

```
<requestHandler name="/select" class="solr.SearchHandler">
    <!-- 设置默认的参数值，可以在请求地址中修改这些参数-->
    <lst name="defaults">
        <str name="echoParams">explicit</str>
        <int name="rows">10</int><!--显示数量-->
        <str name="wt">json</str><!--显示格式-->
        <str name="df">text</str><!--默认搜索字段-->
    </lst>
</requestHandler>
```

Tomcat部署

第一步：安装Tomcat

第二步：部署solr.war

第三步：解压缩solr.war

第四步：添加solr扩展jar包

第五步：添加log4j文件

第六步：配置solrhome路径

第七步：启动Tomcat

配置schema.xml

`schema.xml`文件在`solrhome\collection1\conf`目录下，在`schema.xml`文件中定义了`Field域`以及`FieldType`等配置。

注意：在solr中Field域必须先定义后使用。

1.1.1. field（域）

```
<field name="id" type="string" indexed="true" stored="true" required="true"
multiValued="false" />
```

- name：域的名称
- type：域的类型

- indexed：是否索引
- stored：是否存储
- required：是否必须
- multiValued：是否是多值，存储多个值时设置为true，solr允许一个Field存储多个值，比如存储一个用户的好友id（多个），商品的图片（多个，大图和小图）

1.1.2. dynamicField（动态域）

```
<dynamicField name="*_s"  type="string"  indexed="true"  stored="true" />
```

- name：动态域的名称，是一个表达式，*匹配任意字符，只要域的名称和表达式的规则能够匹配就可以使用。

> 例如：搜索时查询条件[product_i：钻石]就可以匹配这个动态域，可以直接使用，不用单独再定义一个product_i域。

1.1.3. uniqueKey（唯一键）

```
<uniqueKey>id</uniqueKey>
```

相当于主键，每个文档中必须有一个id域。

1.1.4. copyField（复制域）

```
<copyField source="cat" dest="text"/>
```

可以将多个Field复制到一个Field中，以便进行统一的检索。当创建索引时，solr服务器会自动的将源域的内容复制到目标域中。

- source：源域
- dest：目标域，搜索时，指定目标域为默认搜索域，可以提高查询效率。

定义目标域：

```
<field name="text" type="text_general" indexed="true" stored="false"
multiValued="true"/>
```

目标域必须要使用：multiValued="true"

1.1.5. fieldType（域类型）

```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
       <tokenizer class="solr.StandardTokenizerFactory"/>
       <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
/>
       <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
       <tokenizer class="solr.StandardTokenizerFactory"/>
       <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt"
/>
       <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt"
ignoreCase="true" expand="true"/>
       <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

- name：域类型的名称
- class：指定域类型的solr类型。
- analyzer：指定分词器。在FieldType定义的时候最重要的就是定义这个类型的数据在建立索引和进行查询的时候要使用的分析器analyzer，包括分词和过滤。
- type：index和query。Index 是创建索引，query是查询索引。
- tokenizer：指定分词器
- filter：指定过滤器

配置IKAnalyzer中文分词器

- 第一步：把`IKAnalyzer2012FF_u1.jar`添加到solr/WEB-INF/lib目录下。

```
cp   /root/IK\ Analyzer\ 2012FF_hf1/IKAnalyzer2012FF_u1.jar   /kkb/server/solr/tomcat-
solr/webapps/solr/WEB-INF/lib/
```

- 第二步：复制IKAnalyzer的配置文件和自定义词典和停用词词典到solr的classes目录下。

```
cp   /root/IK\ Analyzer\ 2012FF_hf1/IKAnalyzer.cfg.xml    /kkb/server/solr/tomcat-
solr/webapps/solr/WEB-INF/classes/

cp   /root/IK\ Analyzer\ 2012FF_hf1/ext.dic    /kkb/server/solr/tomcat-
solr/webapps/solr/WEB-INF/classes/

cp   /root/IK\ Analyzer\ 2012FF_hf1/stopword.dic    /kkb/server/solr/tomcat-
solr/webapps/solr/WEB-INF/classes/
```

- 第三步：在`schema.xml`中添加一个自定义的`fieldType`，使用中文分析器。

```
<!--   IKAnalyzer-->
<fieldType   name="text_ik" class="solr.TextField">
    <analyzer   class="org.wltea.analyzer.lucene.IKAnalyzer"/>
</fieldType>
```

九、SolrCloud介绍

什么是SolrCloud？

SolrCloud是Solr提供的，基于Solr和ZooKeeper的分布式搜索方案。它的主要思想是使用`Zookeeper`作为`SolrCloud`集群的配置信息中心，统一管理`SolrCloud`的配置，比如`solrconfig.xml`和`schema.xml`。

> 它有几个特色功能：
>
> 1）集中式的配置信息
>
> 2）自动容错
>
> 3）近实时搜索（和ES差别最大的就是这一点，ES内存缓存机制来解决实时搜索问题）
>
> 4）查询时自动负载均衡

什么时候使用SolrCloud呢？

> - 当你需要大规模，容错，分布式索引和检索能力时使用`SolrCloud`。
> - 当索引量很大（10G），搜索请求并发很高时，同样需要使用`SolrCloud`来满足这些需求。
> - 不过当一个系统的索引数据量少的时候是不需要使用`SolrCloud`的。

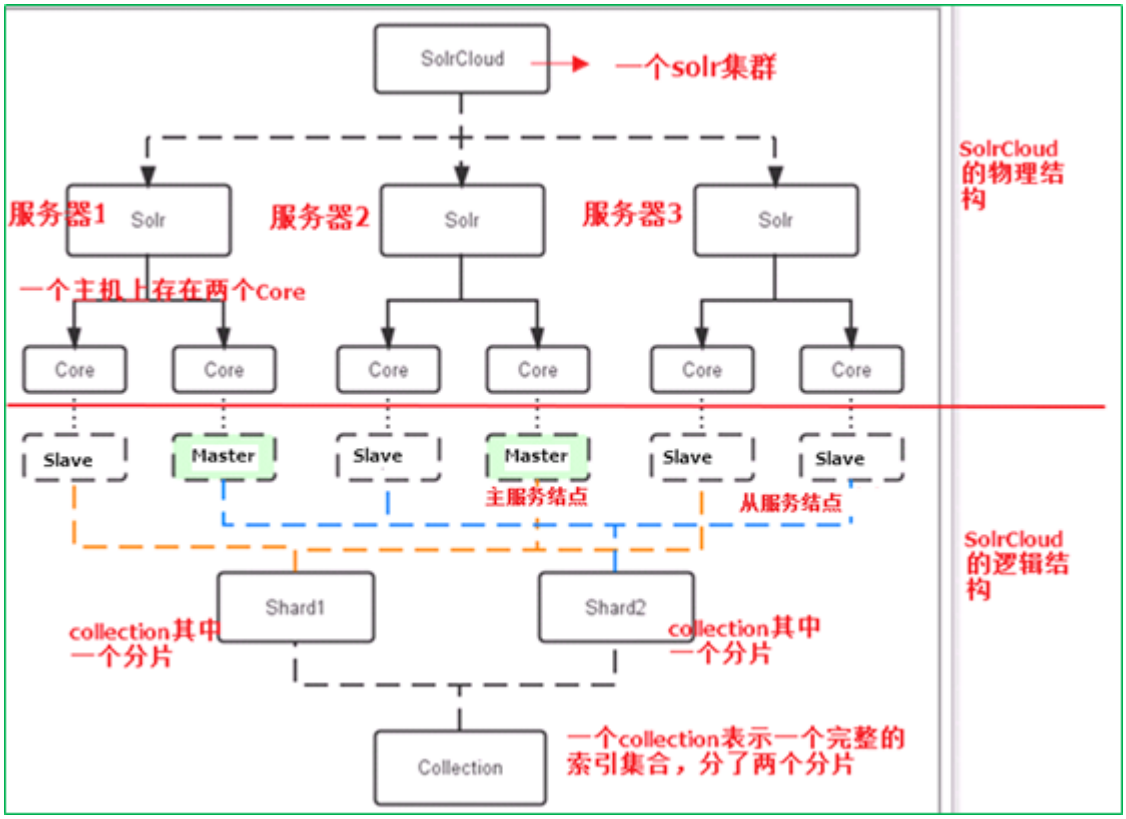
SolrCloud的架构

架构总览

SolrCloud为了降低单机的处理压力，需要由多台服务器共同来完成索引和搜索任务。实现的思路是将索引数据进行Shard分片，每个分片由多台服务器共同完成，当一个索引或搜索请求过来时会分别从不同的Shard的服务器中操作索引。

SolrCloud是基于solr和zookeeper部署，zookeeper是一个分布式协调服务软件，`SolrCloud`**需要由多台solr服务器组成，然后由**`zookeeper`**来进行协调管理。**

下图是一个`SolrCloud`应用的例子：



物理结构

一个 `SolrCloud` 集群由多个 物理机 或者 虚拟机 组成，每个机器中可以包含多个 `SolrCore` ，一个 `SolrCore` 对应一个Tomcat。

逻辑结构

- 一个 `SolrCloud` 集群中逻辑意义上的完整的索引结构，可以看成是一个 `Collection` 。
- 一个 `Collection` 从逻辑上可以被分成多个 `Shard` 。用户发起索引请求分别从shard1和shard2上获取，**解决高并发问题。**
- 一个 `Shard` 片又有多个 `SolrCore` 组成。其中一个Leader多个个 `Replication` 。 `Leader` 是由 `zookeeper` 选举产生， `zookeeper` 控制每个Shard中的多个SolrCore的索引数据一致，**解决高可用问题。**

Collection

`Collection` 在 `SolrCloud` 集群中是**一个逻辑意义上的完整的索引结构。**它常常被划分为一个或多个shard分片，这些shard分片使用相同的配置信息。

> 比如：针对商品信息搜索可以创建一个collection。
>
> collection=shard1+shard2+....+shardX

Shard

Shard是Collection的逻辑分片。每个Shard被化成一个或者多个replication，通过选举确定哪个是Leader。

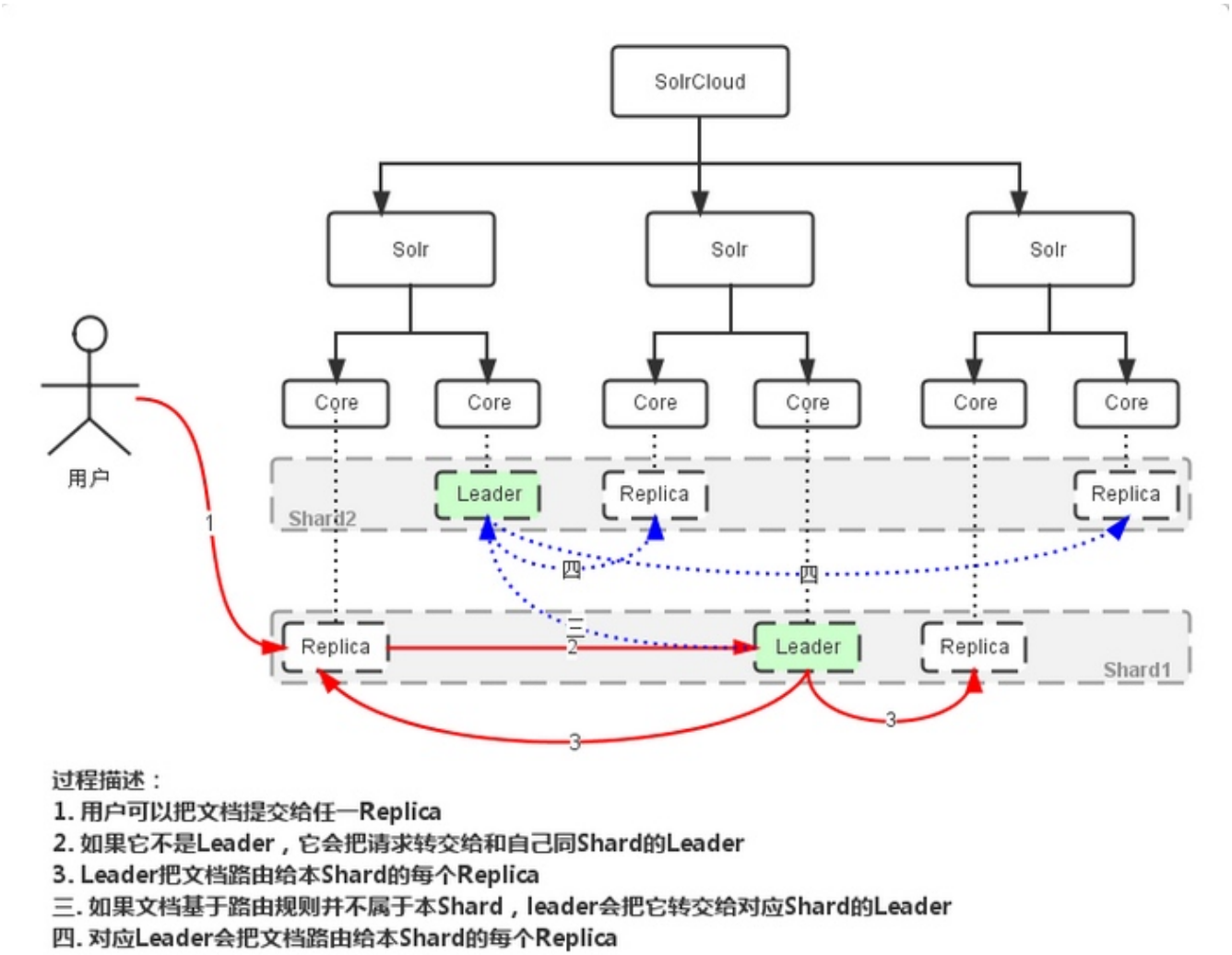
Core

每个Core都是Solr中一个独立运行单位，提供索引和搜索服务。一个shard需要由一个Core或多个Core组成。由于Collection由多个Shard组成，一个Shard由多个Core组成，所以也可以说Collection一般由多个Core组成。

Master或Slave

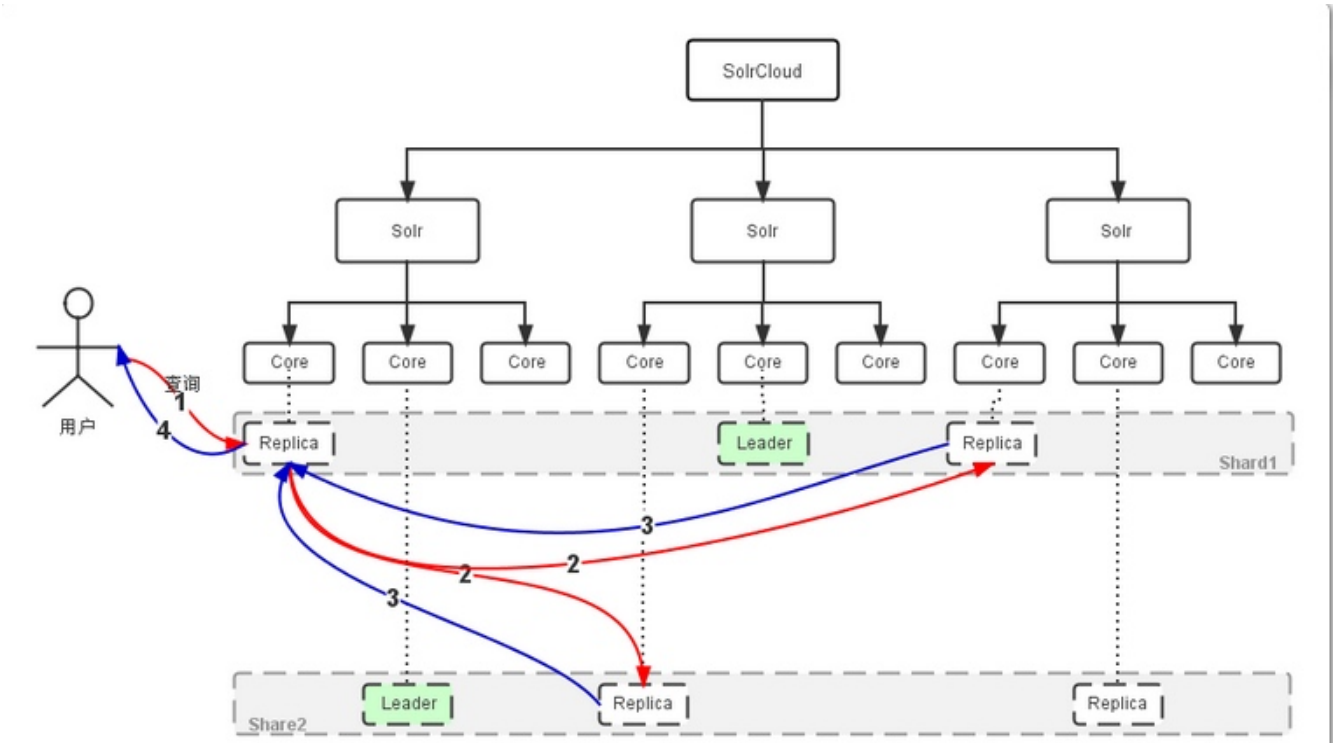
Master是master-slave结构中的主结点（通常说主服务器），Slave是master-slave结构中的从结点（通常说从服务器或备服务器）。同一个Shard下master和slave存储的数据是一致的，这是为了达到高可用目的。

SolrCloud索引流程分析



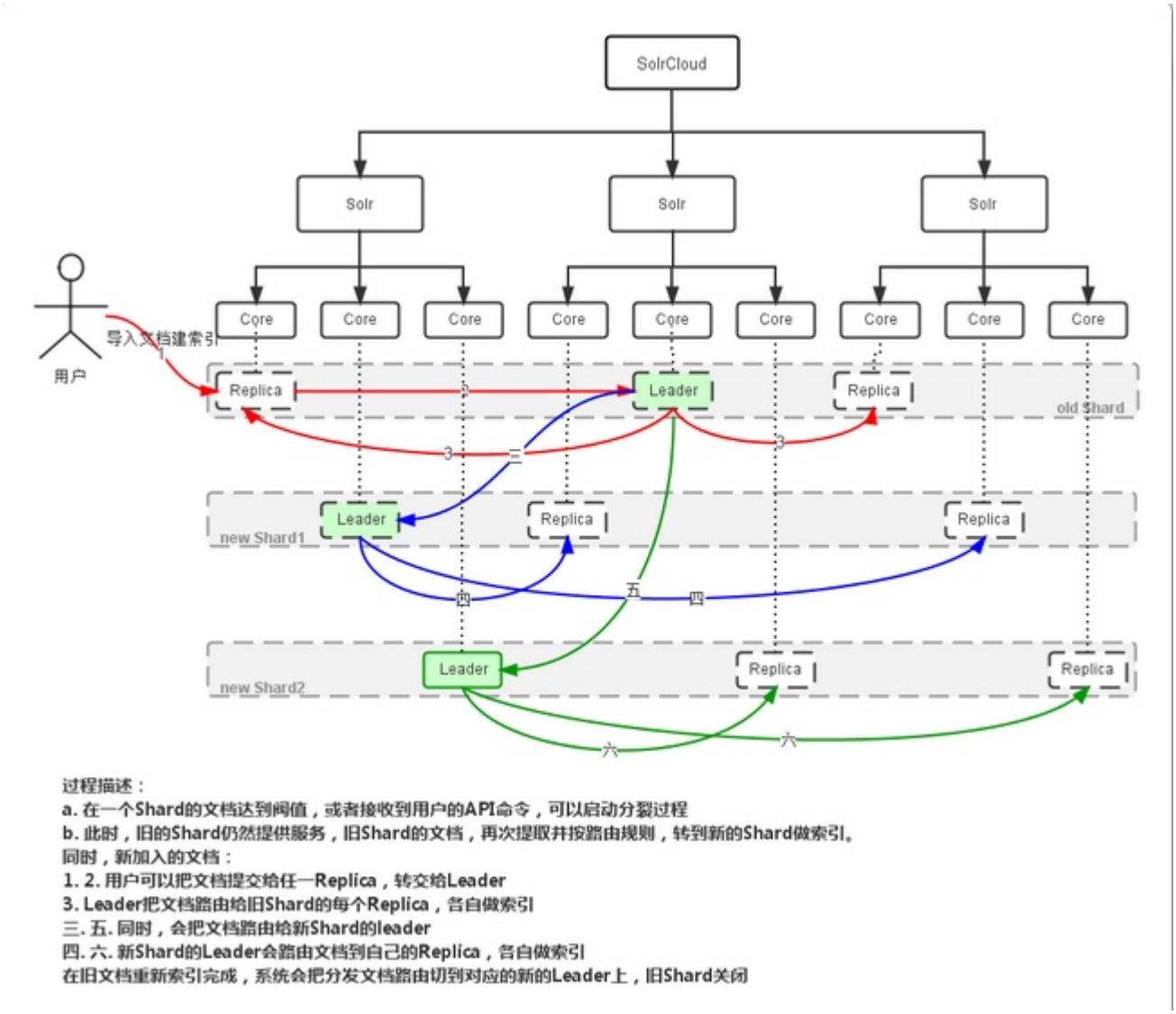
过程描述：
1. 用户可以把文档提交给任一Replica
2. 如果它不是Leader，它会把请求转交给和自己同Shard的Leader
3. Leader把文档路由给本Shard的每个Replica
三. 如果文档基于路由规则并不属于本Shard，leader会把它转交给对应Shard的Leader
四. 对应Leader会把文档路由给本Shard的每个Replica

SolrCloud搜索流程分析



说明：
1. 用户的一个查询，可以发送到含有该Collection的任意机器，Solr内部处理的逻辑会转到一个Replica
2. 此Replica会基于查询索引的方式，启动分布是查询，基于索引的Shard个数，把查询转为多个子查询，并把每个子查询定位到对应Shard的任意一个的Replica
3. 每个子查询返回查询结果
4. 最初的Replica合并子查询，并把最终结果返回给用户

SolrCloud扩展Shard流程分析



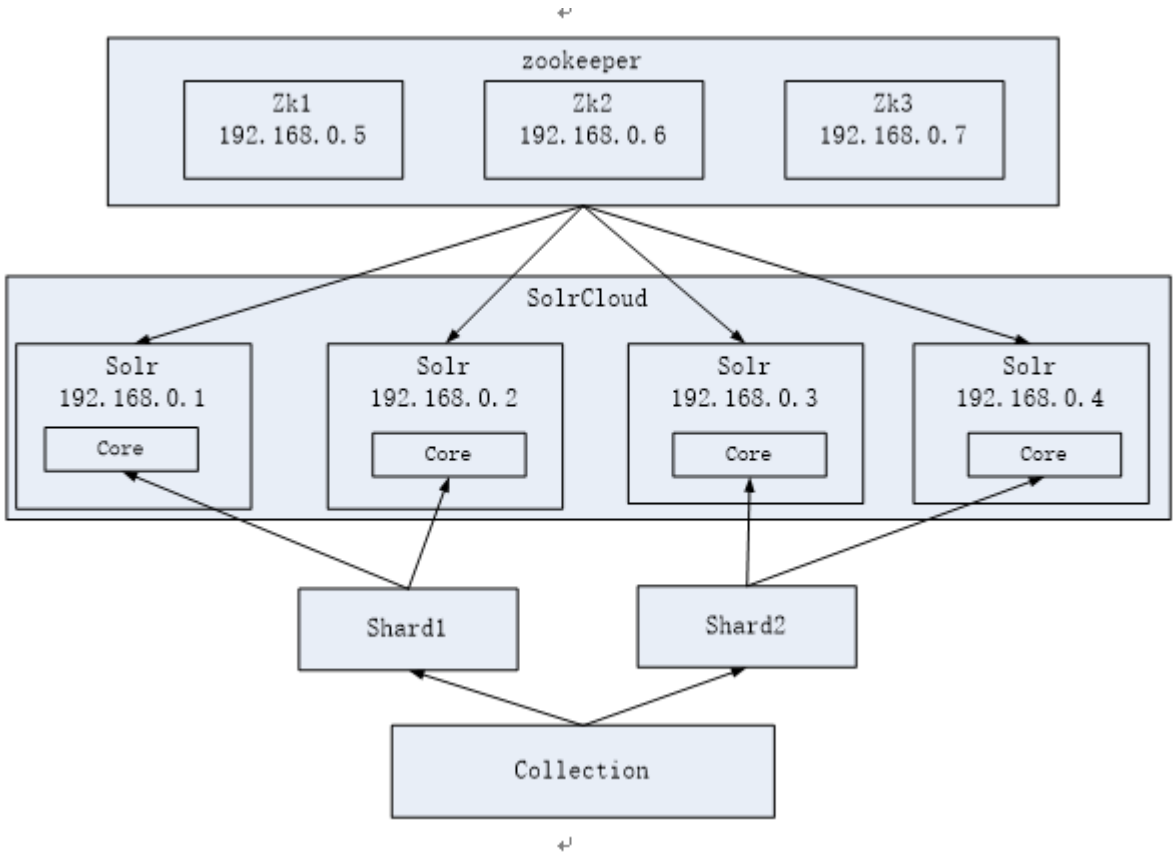
十、SolrCloud搭建

搭建需求

需求分析：

```
SolrCloud是通过ZooKeeper统一管理配置文件（solrconfig.xml、schema.xml等），所以搭建SolrCloud之前，需要先搭建Zookeeper。

由于SolrCloud一般都是解决大数据量、大并发的搜索服务，所以搭建SolrCloud，对ZooKeeper也需要搭建集群。
```

Solrcloud示例结构图如下：



环境准备

- Linux CentOS 7
- Jdk 1.8
- Tomcat 8
- solr-4.10.4.tgz
- zookeeper-3.4.6.tar.gz

ZooKeeper集群搭建

ZooKeeper集群最少需要三台ZooKeeper：

```
- 192.168.10.135
- 192.168.10.136
- 192.168.10.137
```

第一步：安装jdk，ZooKeeper是使用java开发的。

第二步：解压缩zookeeper-3.4.6.tar.gz，并改名为zk

```
wget http://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.4.10/zookeeper-3.4.10.tar.gz

tar -zxf zookeeper-3.4.10.tar.gz

mv zookeeper-3.4.10 zk
```

第三步：创建zoo.cfg

```
cd zk/conf

mv zoo_sample.cfg zoo.cfg
```

第四步：修改zoo.cfg，进行zookeeper集群配置

```
vim zoo.cfg
```

修改内容如下：

```
dataDir=/kkb/server/solrcloud/zk/data
# the port at which the clients will connect
clientPort=2181
#集群中每台机器都是以下配置
#2881系列端口是zookeeper通信端口
#3881系列端口是zookeeper投票选举端口
server.1=192.168.10.135:2881:3881
server.2=192.168.10.136:2881:3881
server.3=192.168.10.137:2881:3881
```

第五步：在dataDir目录下创建myid文件，文件内容为1，对应server.1中的1。

```
vim /kkb/soft/zk/data/myid
```

添加内容如下（其他的zookeeper分别设置为`2和3`）：

```
1
```

第六步：启动zookeeper服务

```
/kkb/soft/zk/bin/zkServer.sh start
```

第七步：查看zookeeper状态

```
/kkb/soft/zk/bin/zkServer.sh status
```

SolrCloud搭建

需要4台机器搭建Solr服务

注意：SolrCloud启动之前，需要先启动ZooKeeper集群。

第一步：Tomcat部署Solr服务

复制单机版Solr服务对应的Tomcat，并修改端口为8888（如果不冲突可以不修改端口）

```
cp /kkb/server/solr/tomcat-solr /kkb/server/solrcloud/tomcat-8888 -r

vim /kkb/server/solrcloud/tomcat-8888/conf/server.xml
```

修改三个端口：`8005、8009、8080`

第二步：修改Tomcat中的web.xml

```
vim /kkb/server/solrcloud/tomcat-8888/webapps/solr/WEB-INF/web.xml
```

修改内容下：

```
    <env-entry>
       <env-entry-name>solr/home</env-entry-name>
       <env-entry-value>/kkb/server/solrcloud/solrhome8888/</env-entry-value>
       <env-entry-type>java.lang.String</env-entry-type>
    </env-entry>
```

第三步：设置Tomcat的启动参数

```
vim /kkb/server/solrcloud/tomcat-8888/bin/catalina.sh
```

在234行下面添加如下内容：

```
JAVA_OPTS="-DzkHost=192.168.10.135:2181,192.168.10.136:2181,192.168.10.137:2181"
```

第四步：复制Tomcat到其他机器

```
scp -r /kkb/server/solrcloud/tomcat-8888 root@192.168.10.135:/kkb/server/solrcloud/
```

第五步：创建solrhome

可以直接复制已安装的单机版Solr服务中的SolrHome目录，并改名为solrhome8888，一个Tomcat对应一个SolrHome。

```
cp /kkb/server/solr/solrhome /kkb/server/solrcloud/solrhome8888 -r
```

第六步：修改solrhome下的solr.xml

```
vim /kkb/server/solrcloud/solrhome8888/solr.xml
```

修改内容如下（hostPort参数的值）：

```
  <solrcloud>
    <str name="host">${host:}</str>
    <int name="hostPort">8888</int>
    <str name="hostContext">${hostContext:solr}</str>
    <int name="zkClientTimeout">${zkClientTimeout:30000}</int>
    <bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>
  </solrcloud>

```

第七步：复制solrhome到其他机器

```
scp -r /kkb/server/solrcloud/solrhome8888 root@192.168.10.135:/kkb/server/solrcloud/
```

第八步：将solr配置文件上传到zookeeper

只需要一台机器进行操作即可！！！

> 使用`/kkb/soft/solr-4.10.4/example/scripts/cloud-scripts`下的`zkcli.sh`命令
> 将`/kkb/server/solrcloud/solrhome8888/collection1/conf`目录上传到`zookeeper`进行配置。

命令如下：

```
./zkcli.sh -zkhost 192.168.10.135:2181,192.168.10.136:2181,192.168.10.137:2181   -cmd upconfig -confdir /kkb/server/solrcloud/solrhome8888/collection1/conf   -confname myconf
```

使用`zookeeper`自带的`zkCli.sh`命令连接`zookeeper`集群，查看上传的配置文件。

```
./zkCli.sh -server localhost:2181

[zk: localhost:2181(CONNECTED) 0] ls /
[configs, zookeeper]
[zk: localhost:2181(CONNECTED) 1] ls /configs
[myconf]
[zk: localhost:2181(CONNECTED) 2] ls /configs/myconf
```

第九步：启动所有Tomcat

集群分片

创建Collection

- 需求：

  创建新的集群，名称为collection2，集群中有四个solr节点，将集群分为两片，每片两个副本。

- HTTP命令：

```
http://192.168.10.135:8888/solr/admin/collections?action=CREATE&name=collection2&numShards=2&replicationFactor=2
```

删除Collection

- 需求：

  删除名称为collection1的集群。

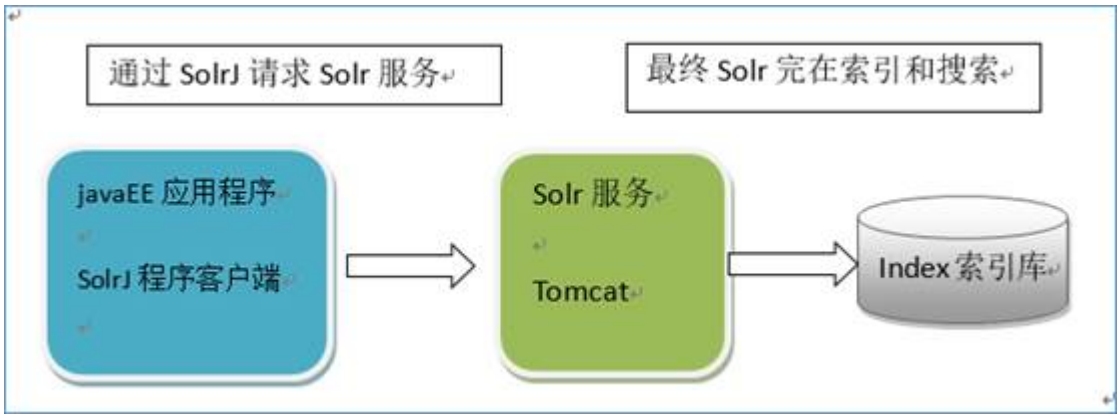
- HTTP命令：

```
http://192.168.10.139:8280/solr/admin/collections?action=DELETE&name=collection1
```

十一、SolrJ的使用

1.1 什么是SolrJ

`solrj`是访问Solr服务的java客户端，提供`索引`和`搜索`的API方法，如下图：



1.2. 需求

使用`solrj`的API调用远程Solr服务器，实现对索引库的增删改操作。

1.3. 添加依赖

```
        <dependency>
            <groupId>org.apache.solr</groupId>
            <artifactId>solr-solrj</artifactId>
            <version>4.10.4</version>
        </dependency>
```

1.4. 代码实现

1.4.1. 添加&修改索引

1.5.1.1. 步骤

1、创建`HttpSolrServer`对象，通过它和Solr服务器建立连接。

2、创建`SolrInputDocument`对象，然后通过它来添加`Field`域。

3、通过`HttpSolrServer`对象将`SolrInputDocument`添加到索引库。

4、提交。

1.4.1.2. 代码

说明：根据id（唯一约束）域来更新Document的内容，如果根据id值搜索不到id域则会执行添加操作，如果找到则更新。

抽取HttpSolrServer 的创建代码：

```
private HttpSolrServer httpSolrServer;

// 提取HttpSolrServer创建
@Before
public void init() {
	// 1. 创建HttpSolrServer对象
	// 设置solr服务接口,浏览器客户端地址http://127.0.0.1:8081/solr/#/
	String baseURL = "http://192.168.10.136:8180/solr";
	this.httpSolrServer = new HttpSolrServer(baseURL);
}
```

添加或者修改索引代码：

```
@Test
public void testCreateAndUpdateIndex() throws Exception {
	// 2. 创建SolrInputDocument对象
	SolrInputDocument document = new SolrInputDocument();
	document.addField("id", "kkb01");
	document.addField("content", "hello world , hello solr");
	// 3. 把SolrInputDocument对象添加到索引库中
	httpSolrServer.add(document);
	// 4. 提交
	httpSolrServer.commit();
}
```

1.4.1.3. 查询测试

使用浏览器客户端进行查询测试

1.4.2. 删除索引

1.4.2.1. 代码

删除索引逻辑，两种：

- 根据id删除
- 根据条件删除

可以使用*:作为条件，就是删除所有数据（慎用）

```
@Test
public void testDeleteIndex() throws Exception {
        // 根据id删除索引数据
        // this.httpSolrServer.deleteById("kkb01");

        // 根据条件删除（如果是*:*就表示全部删除，慎用）
        this.httpSolrServer.deleteByQuery("*:*");

        // 提交
        this.httpSolrServer.commit();
}
```

1.4.2.2. 查询测试

1.4.3 搜索

1.4.4 访问SolrCloud

```

public class SolrCloudTest {
    // zookeeper地址
    private static String zkHostString =
"192.168.101.7:2181,192.168.101.8:2181,192.168.101.9:2181";
    // collection默认名称，比如我的solr服务器上的collection是collection2_shard1_replica1，
就是去掉“_shard1_replica1”的名称
    private static String defaultCollection = "collection2";
    // 客户端连接超时时间
    private static int zkClientTimeout = 3000;
```

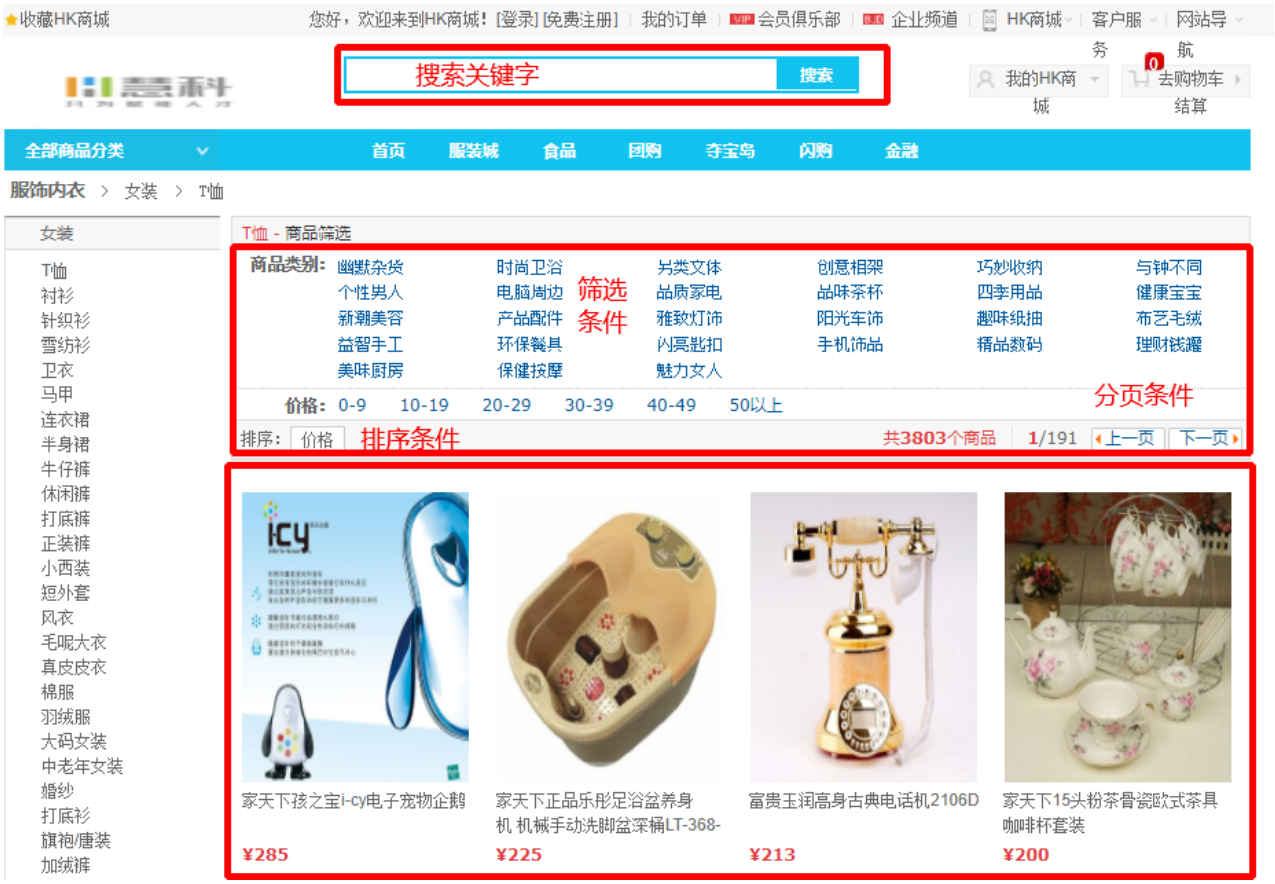
```
    // zookeeper连接超时时间
    private static int zkConnectTimeout = 3000;

    // cloudSolrServer实际
    private CloudSolrServer cloudSolrServer;

    // 测试方法之前构造 CloudSolrServer
    @Before
    public void init() {
       cloudSolrServer = new CloudSolrServer(zkHostString);
       cloudSolrServer.setDefaultCollection(defaultCollection);
       cloudSolrServer.setZkClientTimeout(zkClientTimeout);
       cloudSolrServer.setZkConnectTimeout(zkConnectTimeout);
       cloudSolrServer.connect();
    }

    // 向solrCloud上创建索引
    @Test
    public void testCreateIndexToSolrCloud() throws SolrServerException,
           IOException {

       SolrInputDocument document = new SolrInputDocument();
       document.addField("id", "100001");
       document.addField("title", "李四");
       cloudSolrServer.add(document);
       cloudSolrServer.commit();

    }

    // 搜索索引
    @Test
    public void testSearchIndexFromSolrCloud() throws Exception {

       SolrQuery query = new SolrQuery();
       query.setQuery("*:*");
       try {
           QueryResponse response = cloudSolrServer.query(query);
           SolrDocumentList docs = response.getResults();

           System.out.println("文档个数：" + docs.getNumFound());
           System.out.println("查询时间：" + response.getQTime());

           for (SolrDocument doc : docs) {
              ArrayList title = (ArrayList) doc.getFieldValue("title");
              String id = (String) doc.getFieldValue("id");
              System.out.println("id: " + id);
              System.out.println("title: " + title);
              System.out.println();
           }
       } catch (SolrServerException e) {
           e.printStackTrace();
       } catch (Exception e) {
           System.out.println("Unknowned Exception!!!!");
```

```
            e.printStackTrace();
        }
    }

    // 删除索引
    @Test
    public void testDeleteIndexFromSolrCloud() throws SolrServerException,
IOException {

        // 根据id删除
        UpdateResponse response = cloudSolrServer.deleteById("zhangsan");
        // 根据多个id删除
        // cloudSolrServer.deleteById(ids);
        // 自动查询条件删除
        // cloudSolrServer.deleteByQuery("product_keywords:教程");
        // 提交
        cloudSolrServer.commit();
    }
}
```

十二、站内搜索案例

需求

- 通过请求URL导入索引库数据（必须先实现这个，负责无法完成第二个需求）。
- 模拟京东搜索界面的实现。



分析

Field域分析

- **搜索字段有哪些？**

  商品名称、商品描述、商品分类名称、商品价格

- **结果显示字段有哪些？**

  商品ID、商品名称、商品价格、商品图片地址

- **Field域该如何定义？**

  商品document的field包括：`pid、name、catalog_name、price、description、picture`

```
pid：商品id主键,使用solr本身提供的id域：
<field name="id" type="string" indexed="true" stored="true" required="true"
multiValued="false" />


name：商品名称
<field name="product_name" type="text_ik" indexed="true" stored="true"/>

catalog_name：商品分类名称
<field name="product_catalog_name" type="string" indexed="true" stored="true" />

price：商品价格
<field name="product_price" type="double" indexed="true" stored="true" />

description：商品描述
<field name="product_description" type="text_ik" indexed="true" stored="false" />

picture：商品图片
<field name="product_picture" type="string" indexed="false" stored="true"/>

配置复制域
<field name="product_keywords" type="text_ik" indexed="true" stored="true"
multiValued="true"/>

<copyField source="product_name" dest="product_keywords"/>
<copyField source="product_catalog_name" dest="product_keywords"/>
<copyField source="product_description" dest="product_keywords"/>
```

功能实现分析

- 页面分析（目的是获取请求URL、请求参数、请求返回值）
- 代码实现分析
  - 表现层分析（需要知道请求URL、请求参数、请求返回值）
  - 业务层分析（需要知道搜索和索引的业务逻辑、solr api）

实现

配置Field域

```
<!-- 定义业务域 -->
<field name="product_name" type="text_ik" indexed="true" stored="true"/>
<field name="product_catalog_name" type="string" indexed="true" stored="false" />
<field name="product_price" type="double" indexed="true" stored="true" />
<field name="product_description" type="text_ik" indexed="true" stored="false" />
<field name="product_picture" type="string" indexed="false" stored="true"/>

<!-- 目标域 -->
<field name="product_keywords" type="text_ik" indexed="true" stored="false"
multiValued="true"/>

<!-- 复制域 -->
<copyField source="product_name" dest="product_keywords"/>
<copyField source="product_catalog_name" dest="product_keywords"/>
<copyField source="product_description" dest="product_keywords"/>
```

代码实现

PO类

Service类

实现搜索逻辑

Controller类

只需要接收页面请求参数，和响应结果

十三、ElasticSearch介绍

ElasticSearch概述

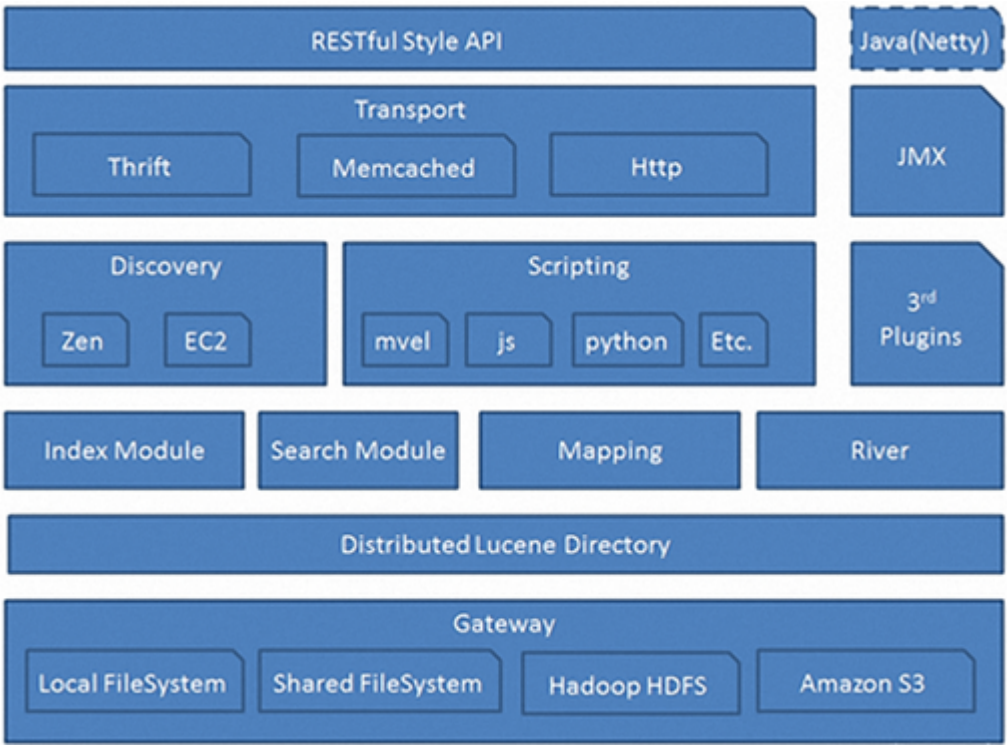
```
ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于RESTful web接口。

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。

构建在全文检索开源软件Lucene之上的Elasticsearch，不仅能对海量规模的数据完成分布式索引与检索，还能提供数据聚合分析。

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.
```

重要的参考网址：

https://www.cnblogs.com/zwt1990/p/7737747.html

https://blog.csdn.net/gwd1154978352/article/details/82781731

Elasticsearch的架构



Gateway层

> 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。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。

Distributed 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。
```

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。
```

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，也可是使用EC2；
es查询还可以支撑多种script即脚本语言，包括mvel、js、python等等。
```

Transport协议层

再上一层就是ES的通讯接口`Transport`

```
支持的也比较多：Thrift、Memcached以及Http，默认的是http，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。
```

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。

ElasticSearch的核心概念

关系数据库	Elasticsearch
数据库Database	索引 Index
表 Table	类型Type
数据行Row	文档 Document
数据列Column	字段Field
表结构Schema	映像Mapping

接近实时（NRT）

> ElasticSearch 是一个接近实时的搜索平台。这意味着，从索引一个文档直到这个文档能够被搜索到有一个很小的延迟（通常是 1 秒）。

索引（index）

> ElasticSearch将它的数据存储在一个或多个索引（index）中。用SQL领域的术语来类比，索引就像数据库，可以向索引写入文档或者从索引中读取文档，并通过ElasticSearch内部使用Lucene将数据写入索引或从索引中检索数据。

文档（document）

> 文档（document）是ElasticSearch中的主要实体。对所有使用ElasticSearch的案例来说，他们最终都可以归结为对文档的搜索。文档由字段构成。

映射（mapping）

所有文档写进索引之前都会先进行分析，如何将输入的文本分割为词条、哪些词条又会被过滤，这种行为叫做映射（mapping）。一般由用户自己定义规则。

类型（type）

每个文档都有与之对应的类型（type）定义。这允许用户在一个索引中存储多种文档类型，并为不同文档提供类型提供不同的映射。

数据源（River）

代表es的一个数据源，也是其它存储方式（如：数据库）同步数据到es的一个方法。它是以插件方式存在的一个es服务，通过读取river中的数据并把它索引到es中，官方的river有couchDB的，RabbitMQ的，Twitter的，Wikipedia的，river这个功能将会在后面的文件中重点说到。

网关（gateway）

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。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。

自动发现（discovery.zen）

代表es的自动发现节点机制，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。

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

集群（cluster）

代表一个集群，集群中有多个节点（node），其中有一个为主节点，这个主节点是可以通过选举产生的，主从节点是对于集群内部来说的。
ES的一个概念就是去中心化，字面上理解就是无中心节点，这是对于集群外部来说的，因为从外部来看ES集群，在逻辑上是个整体，你与任何一个节点的通信和与整个ES集群通信是等价的。

分片（shards）

代表索引分片，es可以把一个完整的索引分成多个分片，这样的好处是可以把一个大的索引拆分成多个，分布到不同的节点上。构成分布式搜索。分片的数量只能在索引创建前指定，并且索引创建后不能更改。

副本（replicas）

代表索引副本，es可以设置多个索引的副本，副本的作用一是提高系统的容错性，当个某个节点某个分片损坏或丢失时可以从副本中恢复。二是提高es的查询效率，es会自动对搜索请求进行负载均衡。

数据恢复（recovery）

- 代表数据恢复或叫数据重新分布，es在有节点加入或退出时会根据机器的负载对索引分片进行重新分配，挂掉的节点重新启动时也会进行数据恢复。
- GET /_cat/health?v #可以看到集群状态

分片和复制（shards and replicas）

一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有10亿文档的索引占据1TB的磁盘空间，而任一节点可能没有这样大的磁盘空间来存储或者单个节点处理搜索请求，响应会太慢。

为了解决这个问题，Elasticsearch提供了将索引划分成多片的能力，这些片叫做分片。当你创建一个索引的时候，你可以指定你想要的分片的数量。每个分片本身也是一个功能完善并且独立的“索引”，这个“索引” 可以被放置到集群中的任何节点上。

分片之所以重要，主要有两方面的原因：

- 允许你水平分割/扩展你的内容容量
- 允许你在分片（位于多个节点上）之上进行分布式的、并行的操作，进而提高性能/吞吐量 至于一个分片怎样分布，它的文档怎样聚合回搜索请求，是完全由Elasticsearch管理的，对于作为用户的你来说，这些都是透明的。

在一个网络/云的环境里，失败随时都可能发生。在某个分片/节点因为某些原因处于离线状态或者消失的情况下，故障转移机制是非常有用且强烈推荐的。为此， Elasticsearch允许你创建分片的一份或多份拷贝，这些拷贝叫做复制分片，或者直接叫复制。

复制之所以重要，有两个主要原因：

- **在分片/节点失败的情况下，复制提供了高可用性。**复制分片不与原/主要分片置于同一节点上是非常重要的。因为搜索可以在所有的复制上并行运行，复制可以扩展你的搜索量/吞吐量
- **总之，每个索引可以被分成多个分片。一个索引也可以被复制0次（即没有复制） 或多次。**一旦复制了，每个索引就有了主分片（作为复制源的分片）和复制分片（主分片的拷贝）。
- **分片和复制的数量可以在索引创建的时候指定。在索引创建之后，你可以在任何时候动态地改变复制的数量，但是你不能再改变分片的数量。**
- 5.X默认5:1 5个主分片，1个复制分片

```
默认情况下，Elasticsearch中的每个索引分配5个主分片和1个复制。这意味着，如果你的集群中至少有两个节点，你的索引将会有5个主分片和另外5个复制分片（1个完全拷贝），这样每个索引总共就有10个分片。
```

核心概念

集群（Cluster）

```
ES集群是一个或多个节点的集合，它们共同存储了整个数据集，并提供了联合索引以及可跨所有节点的搜索能力。多节点组成的集群拥有冗余能力，它可以在一个或几个节点出现故障时保证服务的整体可用性。

集群靠其独有的名称进行标识，默认名称为“elasticsearch”。节点靠其集群名称来决定加入哪个ES集群，一个节点只能属一个集群。
```

节点(node)

一个节点是一个逻辑上独立的服务，可以存储数据，并参与集群的索引和搜索功能，一个节点也有唯一的名字，群集通过节点名称进行管理和通信.

主节点

```
主节点的主要职责是和集群操作相关的内容，如创建或删除索引，跟踪哪些节点是群集的一部分，并决定哪些分片分配给相关的节点。

稳定的主节点对集群的健康是非常重要的。虽然主节点也可以协调节点，路由搜索和从客户端新增数据到数据节点，但最好不要使用这些专用的主节点。一个重要的原则是，尽可能做尽量少的工作。

对于大型的生产集群来说，推荐使用一个专门的主节点来控制集群，该节点将不处理任何用户请求。
```

数据节点

```
持有数据和倒排索引。
```

客户端节点

它既不能保持数据也不能成为主节点，该节点可以响应用户的情况，把相关操作发送到其他节点；

客户端节点会将客户端请求路由到集群中合适的分片上。对于读请求来说，协调节点每次会选择不同的分片处理请求，以实现负载均衡。

部落节点

部落节点可以跨越多个集群，它可以接收每个集群的状态，然后合并成一个全局集群的状态，它可以读写所有节点上的数据。

索引（Index）

ES将数据存储于一个或多个索引中，索引是具有类似特性的文档的集合。类比传统的关系型数据库领域来说，索引相当于SQL中的一个数据库，或者一个数据存储方案（schema）。索引由其名称（必须为全小写字符）进行标识，并通过引用此名称完成文档的创建、搜索、更新及删除操作。一个ES集群中可以按需创建任意数目的索引。

文档类型（Type）

类型是索引内部的逻辑分区（category/partition），然而其意义完全取决于用户需求。因此，一个索引内部可定义一个或多个类型（type）。一般来说，类型就是为那些拥有相同的域的文档做的预定义。例如，在索引中，可以定义一个用于存储用户数据的类型，一个存储日志数据的类型，以及一个存储评论数据的类型。类比传统的关系型数据库领域来说，类型相当于“表”。

文档（Document）

文档是Lucene索引和搜索的原子单位，它是包含了一个或多个域的容器，基于JSON格式进行表示。文档由一个或多个域组成，每个域拥有一个名字及一个或多个值，有多个值的域通常称为“多值域”。每个文档可以存储不同的域集，但同一类型下的文档至应该有某种程度上的相似之处。相当于数据库的“记录”

映射（Mapping）

相当于数据库中的schema，用来约束字段的类型，不过 Elasticsearch 的 mapping 可以自动根据数据创建
ES中，所有的文档在存储之前都要首先进行分析。用户可根据需要定义如何将文本分割成token、哪些token应该被过滤掉，以及哪些文本需要进行额外处理等等。

分片和复制（shards and replicas）

ES的“分片(shard)”机制可将一个索引内部的数据分布地存储于多个节点，它通过将一个索引切分为多个底层物理的Lucene索引完成索引数据的分割存储功能，这每一个物理的Lucene索引称为一个分片(shard)。

每个分片其内部都是一个全功能且独立的索引，因此可由集群中的任何主机存储。创建索引时，用户可指定其分片的数量，默认数量为5个。

Shard有两种类型：primary和replica，即主shard及副本shard。

Primary shard

用于文档存储，每个新的索引会自动创建5个Primary shard，当然此数量可在索引创建之前通过配置自行定义，不过，一旦创建完成，其Primary shard的数量将不可更改。

Replica shard

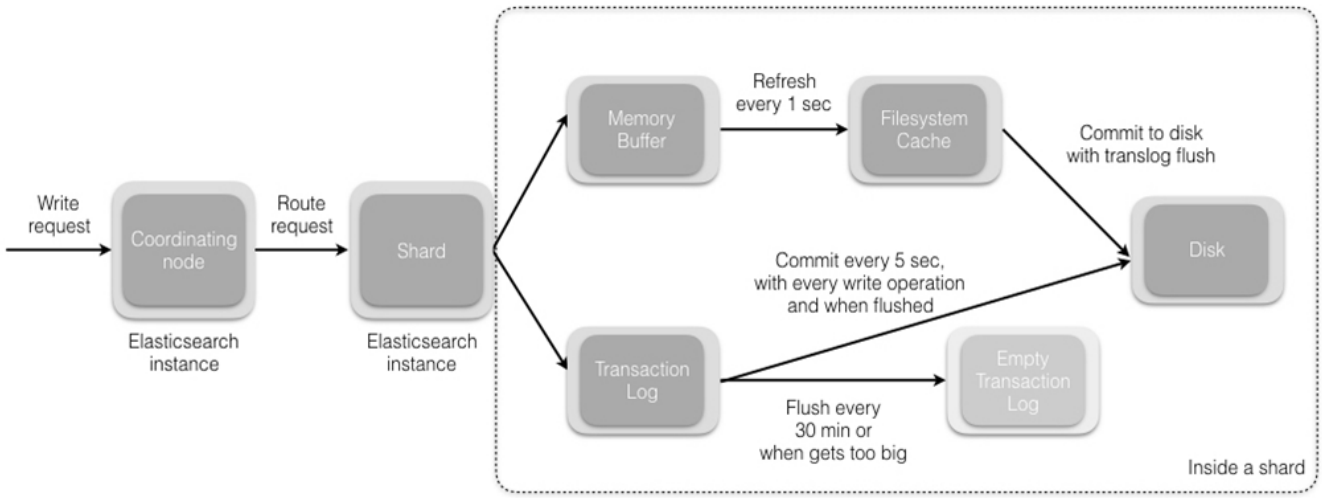
是Primary Shard的副本，用于冗余数据及提高搜索性能。
每个Primary shard默认配置了一个Replica shard，但也可以配置多个，且其数量可动态更改。ES会根据需要自动增加或减少这些Replica shard的数量。

总结

ES集群可由多个节点组成，各Shard分布式地存储于这些节点上。
ES可自动在节点间按需要移动shard，例如增加节点或节点故障时。简而言之，分片实现了集群的分布式存储，而副本实现了其分布式处理及冗余功能。

索引和搜索流程分析

索引流程



过程：

```
1 当分片所在的节点接收到来自协调节点的请求后，会将该请求写入translog，并将文档加入内存缓存。如果请求在主分片上成功处理，该请求会并行发送到该分片的副本上。当translog被同步到全部的主分片及其副本上后，客户端才会收到确认通知。
2 内存缓冲会被周期性刷新(默认是1秒)，内容将被写到文件系统缓存的一个新段（segment）上。虽然这个段并没有被同步(fsync)，但它是开放的，内容可以被搜索到。
3 每30分钟，或者当translog很大的时候，translog会被清空，文件系统缓存会被同步。这个过程在Elasticsearch中称为冲洗(flush)。在冲洗过程中，内存中的缓冲将被清除，内容被写入一个新段。段的fsync将创建一个新的提交点，并将内容刷新到磁盘。旧的translog将被删除并开始一个新的translog。
```

ES如何做到实时检索？

```
1 由于在buffer中的索引片先同步到文件系统缓存，再刷写到磁盘，因此在检索时可以直接检索文件系统缓存，保证了实时性。
2 这一步刷到文件系统缓存的步骤，在 Elasticsearch 中，是默认设置为 1 秒间隔的，对于大多数应用来说，几乎就相当于是实时可搜索了。
3 不过对于 ELK 的日志场景来说，并不需要如此高的实时性，而是需要更快的写入性能。我们可以通过 /_settings 接口或者定制 template 的方式，加大 refresh_interval 参数。
```

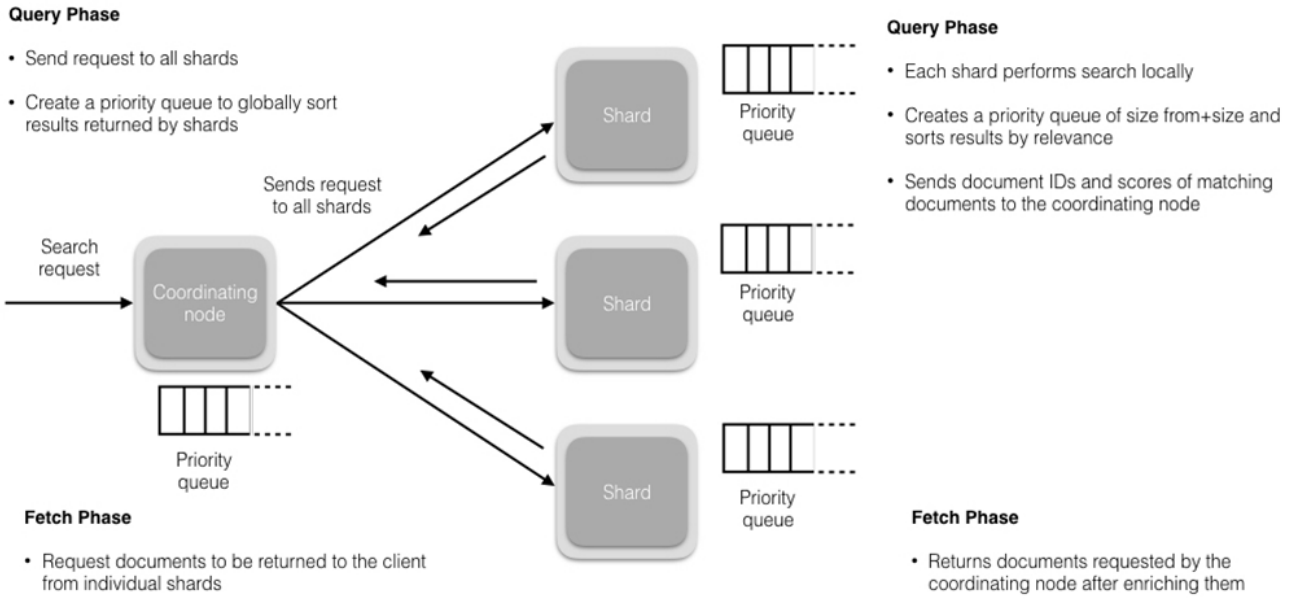
当segment从文件系统缓存同步到磁盘时发生了错误怎么办？ 数据会不会丢失？

```
1 由于Elasticsearch 在把数据写入到内存 buffer 的同时，其实还另外记录了一个 translog日志，如果在这期间故障发生时，Elasticsearch会从commit位置开始，恢复整个translog文件中的记录，保证数据的一致性。
2
3 等到真正把 segment 刷到磁盘，且 commit 文件进行更新的时候， translog 文件才清空。这一步，叫做 flush。同样，Elasticsearch 也提供了 /_flush 接口。
```

索引数据的一致性通过 translog 保证,那么 translog 文件自己呢？

```
1 Elasticsearch 2.0 以后为了保证不丢失数据，每次 index、bulk、delete、update 完成的时候，一定触发刷新translog 到磁盘上，才给请求返回 200 OK。这个改变在提高数据安全性的同时当然也降低了一点性能
```

搜索流程



删除索引分析

删除和更新也都是写操作。但是Elasticsearch中的文档是不可变的，因此不能被删除或者改动以展示其变更。那么，该如何删除和更新文档呢？
磁盘上的每个段都有一个相应的.del文件。当删除请求发送后，文档并没有真的被删除，而是在.del文件中被标记为删除。该文档依然能匹配查询，但是会在结果中被过滤掉。当段合并(我们将在本系列接下来的文章中讲到)时，在.del文件中被标记为删除的文档将不会被写入新段。

修改索引分析

接下来我们看更新是如何工作的。在新的文档被创建时，Elasticsearch会为该文档指定一个版本号。当执行更新时，旧版本的文档在.del文件中被标记为删除，新版本的文档被索引到一个新段。旧版本的文档依然能匹配查询，但是会在结果中被过滤掉。

十四、ElasticSearch原理

集群的节点

三种节点角色

master节点

```
整个集群只会有一个master节点，它将负责管理集群范围内的所有变更，例如增加、删除索引；或者增加、删除节点等。而master节点并不需要涉及到文档级别的变更和搜索等操作，所以当集群只拥有一个master节点的情况下，即使流量的增加它也不会成为瓶颈。

master节点需要从众多候选master节点中选择一个。
```

master节点的作用：

```
负责集群节点上下线，shard分片的重新分配。

创建、删除索引。

负责接收集群状态(`cluster state`)的变化，并推送给所有节点。集群节点都各有一份完整的cluster state，只是master node负责维护。

利用自身空闲资源，协调创建索引请求或者查询请求，将请求分发到相关node服务器。
```

master节点的配置如下（elasticsearch.yml）

```
node.master: true
node.data: false
```

data结点(数据节点)

```
负责存储数据，提供建立索引和搜索索引的服务。

data节点消耗内存和磁盘IO的性能比较大。
```

data节点的配置如下（elasticsearch.yml）

```
node.master：false
node.data： true
```

client结点(负载均衡结点)

```
不会被选作主节点，也不会存储任何索引数据。主要用于查询负载均衡。将查询请求分发给多个node服务器，并对结果进行汇总处理。
```

client节点的配置如下（elasticsearch.yml）

```
node.master: false
node.data: false
```

节点配置选择

```
> 1. 如果节点小于10个：所有节点都是master+data即可。
> 2. 超过过100个才细分。
```

其对应的高性能集群拓扑结构模式为：

`elasticsearch.yml`

```
#配置文件中给出了三种配置高性能集群拓扑结构的模式,如下：
#1. 如果你想让节点从不选举为主节点,只用来存储数据,可作为负载器
node.master: false
node.data: true
#2. 如果想让节点成为主节点,且不存储任何数据,并保有空闲资源,可作为协调器
node.master: true
node.data: false
#3. 如果想让节点既不称为主节点,又不成为数据节点,那么可将他作为搜索器,从节点中获取数据,生成搜索结果等
node.master: false
node.data: false
```

集群发现机制

相关配置

```
# 集群内最少候选主节点个数，只有满足这个数量的候选主节点在线，才能进行master节点的竞选。
# 为了防止脑裂现象
discovery.zen.minimum_master_nodes: 3
# 候选主节点ping master主节点的超时时间
discovery.zen.ping_timeout: 100s
discovery.zen.fd.ping_timeout: 100s
# 单播时，需要一些服务器列表进行集群状态的传播
discovery.zen.ping.unicast.hosts:
["10.19.0.97","10.19.0.98","10.19.0.99","10.19.0.100"]
```

上面的配置中，两个 `timeout` 可能会让人有所迷惑。这里的 `fd` 是 `fault detection` 的缩写。也就是说：

- `discovery.zen.ping_timeout` ：仅在加入或者选举 `master` 主节点的时候才起作用；
- `discovery.zen.fd.ping_timeout` ：则在稳定运行的集群中，`master` 检测所有节点，以及节点检测 `master` 是否畅通时长期有用。

1. zen discovery机制

ES默认的`discovery`机制是`zen discovery`机制。

```
> zen发现机制是elasticsearch默认的内建模块。它提供了多播和单播两种发现方式，能够很容易的扩展至云环境。

> zen discovery机制提供了unicast discovery集群发现机制，集群发现时的节点间通信是依赖的transport module，也就是es底层的网络通信模块和协议。
```

ping module

```
这是一个节点使用发现机制去寻找其他节点的进程。

同时支持多播和单播方式的发现（也可以组合使用）
```

unicast module

`ES默认的自动发现机制`。

> 默认配置下，`unicast`是针对本机，也就是`localhost`，因此只能本机上启动多个`node`组成集群。属于回环的地址。
>
> 如果不想本机，就须将`network.host`配置到非回环地址。如`network.host: 192.168.1.10`。
>
> 一旦配置，ES会认为我们从开发迁移到生产模式，同时启用一系列`bootstrap check`。

```
discovery.zen.ping.unicast.hosts: ["host1", "host2:port"]
```

ES的`unicast`需要有中间节点，通过公共节点交换各自的信息，进而让所有`node`感知到其他`node`存在，并进行通信，最后组成一个集群。这就是基于`gossip`流言式通信协议的`unicast`集群发现机制。

这意味着`unicast list node`是不需要列出集群中的所有节点的。只要提供少数几个`node`，比如3个，让新的`node`可以连接上即可。一般设置为几个`master`节点即可。

用如下的配置即可：

```
# 集群名称一致
cluster.name:kkb-es
# 节点名称不一致
node.name:node-135
# 不要配置回环地址：127.0.0.1
network.host:0.0.0.0
# 单播路由服务器列表
discovery.zen.ping.unicast.hosts: ["10.0.16.77:9300", "10.0.16.88:9300"]
```

`unicast发现机制`要求配置一个主机列表，用来作为gossip（流言式）通信协议的路由器。`unicast discovery`机制最重要的两个配置如下所示：

```
discovery.zen.ping.unicast.hosts：用逗号分割的主机列表
discovery.zen.ping.unicast.hosts.resolve_timeout：hostname被DNS解析为ip地址的timeout等待时长
```

总结：

1. 已经初步配置好了各个节点，首先通过`network.host`绑定到了非回环的ip地址，从而可以跟其他节点通信。
2. 通过`discovery.zen.ping.unicast.hosts`配置了一批`unicast`中间路由的node
3. 所有node都可以发送ping消息到路由node，再从路由node获取`cluster state`回来
4. 接着所有node会选举出一个`master`
5. 所有node都会跟`master`进行通信，然后加入`master`的集群
6. 要求`cluster.name`必须一样，才能组成一个集群
7. `node.name`就标识出了每个node，这是我们自己设置的一个名称

2. master选举

1. 涉及配置参数

1、如果同时启动，按照nodeid进行排序，取出最小的做为master节点

2、如果不是同时启动，则先启动的候选master节点，会竞选为master节点。

```
# 如果`node.master`设置为了false，则该节点没资格参与`master`选举。
node.master = true
# 默认3秒，最好增加这个参数值，避免网络慢或者拥塞，确保集群启动稳定性
discovery.zen.ping_timeout: 3s
# 用于控制选举行为发生的集群最小master节点数量，防止脑裂现象
discovery.zen.minimum_master_nodes : 2
# 新节点加入集群的等待时间
discovery.zen.join_timeout : 10s
```

2. 新节点加入

节点完成选举后，新节点加入，会发送`join request`到`master`节点。默认会重试20次。

3. 宕机再次选举

如果宕机，集群node会再次进行`ping`过程，并选出一个新的`master`。

一旦一个节点被明确设为一个客户端节点（`node.client设为true`），则不能再成为主节点（`node.master会自动设为false`）。

3. 集群故障的探查

ES有两种集群故障探查机制：

1. 通过master进行的，master会ping集群中所有的其他node，确保它们是否是存活着的。

> 2. 每个node都会去ping master来确保master是存活的，否则会发起一个选举过程。

有下面三个参数用来配置集群故障的探查过程：

```
ping_interval : 每隔多长时间会ping一次node，默认是1s
ping_timeout : 每次ping的timeout等待时长是多长时间，默认是30s
ping_retries : 如果一个node被ping多少次都失败了，就会认为node故障，默认是3次
```

4. 集群状态更新

相关配置

```
# cluster state commit超时时间
discovery.zen.commit_timeout:30s
discovery.zen.minimum_master_nodes : 3
# discovery.zen.publish_timeout默认是30s，这个超时等待时长是从plublish cluster state开始计算的
discovery.zen.publish_timeout : 30s
```

####

`master node`是集群中唯一一个可以对`cluster state`进行更新的`node`。

`master node`每次会处理一个集群状态的更新事件，应用这次状态更新，然后将更新后的状态发布到集群中所有的`node`上去。每个`node`都会接收`publish message`，`ack`这个`message`，但是不会应用这个更新。

- 如果`master`没有在`discovery.zen.commit_timeout`指定的时间内（默认是30s），从至少`discovery.zen.minimum_master_nodes`个节点获取`ack`响应，那么这次`cluster state change`事件就会被`reject`，不会应用。
- 如果`master`在`discovery.zen.commit_timeout`指定时间内，指定数量的`node`都返回了`ack`消息，那么`cluster state`就会被`commit`，然后一个`message`会被发送给所有的`node`。所有的`node`接收到那个`commit message`之后，接着才会将之前接收到的集群状态应用到自己本地的状态副本中去。接着`master`会等待所有节点再次响应是否更新自己本地副本状态成功，在一个等待超时时长内，如果接收到了响应，那么就会继续处理内存`queue`中保存的下一个更新状态。

5. 不因为master宕机阻塞集群操作

如果要让集群正常运转，那么必须有一个`master`，还有`discovery.zen.minimum_master_nodes`指定数量的`master候选node`，都在运行。

如果出现宕机，什么样的操作可以被拒绝？通过以下配置可以控制该情况。

相关配置如下：

```
# 可以控制当master宕机时，什么样的操作应该被拒绝,它有两个值：
# all：一旦master当即，那么所有的操作都会被拒绝
# write：这是默认的选项，所有的写操作都会被拒绝，但是读操作是被允许的
discovery.zen.no_master_block : all
```

宕机恢复等故障

脑裂现象

参考网址：https://blog.csdn.net/ty4315/article/details/52491799

1．什么是脑裂现象

由于部分节点网络断开，集群分成两部分，且这两部分都有master选举权。就成形成一个与原集群一样名字的集群，这种情况称为集群脑裂（split-brain）现象。这个问题非常危险，因为两个新形成的集群会同时索引和修改集群的数据。

2．解决方案

```
# 决定选举一个master最少需要多少master候选节点。默认是1。
# 这个参数必须大于等于为集群中master候选节点的quorum数量，也就是大多数。
# quorum算法：master候选节点数量 / 2 + 1
# 例如一个有3个节点的集群，minimum_master_nodes 应该被设置成 3/2 + 1 = 2（向下取整）
discovery.zen.minimum_master_nodes:2
# 等待ping响应的超时时间,默认值是3秒。如果网络缓慢或拥塞，会造成集群重新选举，建议略微调大这个值。
# 这个参数不仅仅适应更高的网络延迟，也适用于在一个由于超负荷而响应缓慢的节点的情况。
discovery.zen.ping.timeout:10s
# 当集群中没有活动的Master节点后，该设置指定了哪些操作（read、write）需要被拒绝（即阻塞执行）。有两个设置值：all和write，默认为wirte。
discovery.zen.no_master_block : write
```

3.场景分析

一个生产环境的es集群，至少要有3个节点，同时将discovery.zen.minimum_master_nodes设置为2， 那么这个是参数是如何避免脑裂问题的产生的呢？

> 比如我们有3个节点，quorum是2。现在网络故障，1个节点在一个网络区域，另外2个节点在另外一个网络区域，不同的网络区域内无法通信。这个时候有两种情况情况：

> （1）如果master是单独的那个节点，另外2个节点是master候选节点，那么此时那个单独的master节点因为没有指定数量的候选master node在自己当前所在的集群内，因此就会取消当前master的角色，尝试重新选举，但是无法选举成功。然后另外一个网络区域内的node因为无法连接到master，就会发起重新选举，因为有两个master候选节点，满足了quorum，因此可以成功选举出一个master。此时集群中就会还是只有一个master。
>
> （2）如果master和另外一个node在一个网络区域内，然后一个node单独在一个网络区域内。那么此时那个单独的node因为连接不上master，会尝试发起选举，但是因为master候选节点数量不到quorum，因此无法选举出master。而另外一个网络区域内，原先的那个master还会继续工作。这也可以保证集群内只有一个master节点。

综上所述，通过在`elasticsearch.yml`中配置`discovery.zen.minimum_master_nodes: 2`，就可以避免脑裂问题的产生。

但是因为ES集群是可以动态增加和下线节点的，所以可能随时会改变`quorum`。所以这个参数也是可以通过api随时修改的，特别是在节点上线和下线的时候，都需要作出对应的修改。而且一旦修改过后，这个配置就会持久化保存下来。

```
PUT /_cluster/settings { "persistent" : { "discovery.zen.minimum_master_nodes" : 2 } }
```

重启

```
# 足够的节点上线后，才进行shard recovery的过程
gateway.recover_after_nodes: 8
# 最多等待5分钟，5分钟还没上线 就重新rebalance
gateway.recover_after_time: 5m
# 只要10个节点上线，就开始shard recovery的过程
gateway.expected_nodes: 10
```

配置说明：

```
ES集群会等待至少8个节点在线，然后等待最多5分钟，或者10个节点都在线，开始shard recovery的过程。

这样就可以避免少数node启动时，就立即开始shard recovery，消耗大量的网络和磁盘资源，甚至可以将shard recovery过程从数小时缩短为数分钟。
```

> 如果10节点集群，5个已经重启成功，另外5没有。会触发shard的rebalance操作。
>
> 在线5个node会将部分replica shard提升为primary shard，同时为每个primary shard复制足够的replica shard。

> 如果剩下的5个节点加入了集群。发现自己原来持有的shard已被复制。就会删除自己本地数据。然后集群又会开始进行shard的rebalance操作

> 在这个过程中，这些shard重新复制，移动，删除，再次移动的过程，会大量的耗费网络和磁盘资源。可能导致每次集群重启时，都有TB级别的数据无端移动，导致集群启动会耗费很长时间。但是如果所有的节点都可以等待整个集群中的所有节点都完全上线之后，所有的数据都有了以后，再决定是否要复制和移动shard，情况就会好很多。

集群搭建规划

1．内存

es很占内存，用jvm heap（堆内存）还是比较少，主要占机器内存。

es基于lucene，lucene基于磁盘文件索引数据，lucene的特点是基于os filesystem cache，将频繁访问的磁盘文件读到内存进行缓存提高性能

如果os cache内缓存所有索引文件，则索引读写性能会很高，特别是检索，速度在ms级别

但是os cache放不下，搜索集合大量读写磁盘，这性能会降一个数量级，秒级

es生产环境中，内存最容易消耗，排序和聚合会耗很多内存，需要分配足够的jvm heap内存，其余的内存需要给lucene。

数据量：

几万-几百万 内存无所谓

过亿-几十亿 建议每台机器都给64G内存

2．cpu

es集群对于cpu要求比较低，没有内存重要，一般多核处理器即可，2 core~8 core都行

核数可以提高并发能力，远比单核性能高

3．磁盘

如果每天大量数据写入es集群，则磁盘的读写性能会造成整个集群的性能瓶颈。推荐SSD或RAID

SSD需要检查调整I/O scheduler，决定什么时候将数据写入磁盘，可以带来很大的性能提升，避免NAS

4．网络

网络很重要，低延迟+高速。低延迟可以减少通信时间，高速可以让shard更快的移动和恢复

es集群是p2p模式的分布式系统架构，不是master-slave主从分布式系统。因此所有node都是相等的，任意两个node间的互相通信都是很频繁和正常的。因此要避免异地多机房，负责跨地域通信会延时很高，造成集群频繁不正常。

5、自建集群 vs 云部署

建议拥有少数机器，但是每个机器的资源都非常多，尽量避免拥有大量的少资源的虚拟机

管理5个物理机组成的es集群，远远比管理100个虚拟机组成的es集群要简单的多。

6. JVM

建议用最新的jvm版本，lucene的单元测试和集成测试中，经常会发现jvm自身的一些bug。这些bug涵盖的范围很广，因此尽量用最新的jvm版本，bug会少一些。

保证所有环境的jdk版本都一样，以防client和server的jvm版本不一致，序列化不出现问题

少调jvm参数，es几乎已经是最优的了。

es 5.x版本而言，建议用jdk 8，而不是jdk 7，同时jdk 6已经不再被支持了。

es 6.x，只支持jdk8及以上

7. 容量规划

建议数据在10亿规模以内。数据量最好恒定。

先计算数据硬盘容量，总内存数。es会用掉一半给jvmheap，剩下的一半，硬盘容量*1.5 足够

数据量在10亿以内，5台以上8核64G足够，如果查询太复杂，则需要加内存

十四、ElasticSearch安装

单机版安装

下载

- 下载地址：https://www.elastic.co/downloads/elasticsearch
- 下载安装包

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.5.3.tar.gz
```

安装

前置安装条件：JDK（不要装在root目录下）

- 第一步：解压缩

```
tar -xf elasticsearch-6.5.3.tar.gz -C /kkb/server
```

- 第二步：配置远程访问

```
vim /kkb/server/elasticsearch-6.5.3/config/elasticsearch.yml
```

将`network.host`选项打开，并指定值为`0.0.0.0`：

```
network.host: 0.0.0.0
```

- 第三步：进程可以拥有的VMA(虚拟内存区域)的数量

```
vim /etc/sysctl.conf
```

添加以下内容：

```
vm.max_map_count=655360
```

生效配置文件：

```
sysctl -p
```

- 第四步：修改允许打开的最大文件描述符数量

```
vim /etc/security/limits.conf
```

添加以下内容：

```
esuser soft nofile 65536
esuser hard nofile 65536
esuser soft nproc 4096
esuser hard nproc 4096
```

- 第五步：设置用户最大线程数

```
vim /etc/security/limits.d/20-nproc.conf
```

修改内容如下，将`*`改为具体的`esuser`用户：

```
# Default limit for number of user's processes to prevent
# accidental fork bombs.
# See rhbz #432903 for reasoning.

esuser     soft    nproc     4096
root       soft    nproc     unlimited
```

- 第六步：创建用户和组（ES不能使用root用户启动）

```
groupadd esgroup
useradd esuser -g esgroup -p 111111
```

- 第七步：更改`elasticsearch`文件夹及内部文件的所属用户及组

```
chown -R esuser:esgroup /kkb/server/elasticsearch-6.5.3
```

- 第八步：切换到`esuser`用户

```
su esuser
```

- 第九步：启动`elasticsearch`

  启动之前，根据服务器内存大小最好先修改一下`jvm.options`

```
vim /kkb/server/elasticsearch-6.5.3/config/jvm.options
```

  修改内容如下（默认是1G）：

```
-Xms256m
-Xmx256m
```

前端启动：

```
/kkb/server/elasticsearch-6.5.3/bin/elasticsearch
```

后端启动：

```
/kkb/server/elasticsearch-6.5.3/bin/elasticsearch -d
```

停止：

```
jps
```

- 第十步：浏览器访问测试

访问地址：http://192.168.10.135:9200/?pretty

```
{
  "name" : "WJwO_6B",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "uqkB94TCQ_amBDsu5wy_fw",
  "version" : {
    "number" : "6.5.3",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "159a78a",
    "build_date" : "2018-12-06T20:11:28.826501Z",
    "build_snapshot" : false,
    "lucene_version" : "7.5.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

集群配置

elasticsearch.yml详解

```
/kkb/server/elasticsearch-6.5.3/config/elasticsearch.yml
```

```
# 集群名称 Use a descriptive name for your cluster:
cluster.name: kkb-es
# ------------------------------------ Node ------------------------------------
# 节点名称 Use a descriptive name for the node:
node.name: node-135
# 既可以选举为主节点,也可以存储数据,也可作为负载器
node.master: true
node.data: true
# 指定节点的部落属性Add custom attributes to the node:
#node.attr.rack: r1
# ---------------------------------- Paths ------------------------------------
# 数据存储地址 Path to directory where to store the data (separate multiple locations by comma):
path.data: /opt/apps/elasticsearch/data
# Log日志 Path to log files:
path.logs: /opt/apps/elasticsearch/logs
# 临时文件
# path.work: /opt/apps/elasticsearch/tmp
# 设置插件的存放路径，默认是es根目录下的plugins文件夹
#path.plugins: /opt/apps/elasticsearch/plugins
# 设置默认索引分片个数，默认为5片。
```

```
# index.number_of_shards: 3
# 设置默认索引副本个数，默认为1个副本。
# index.number_of_replicas: 2
# ---------------------------------- Memory -----------------------------------
# Lock the memory on startup:
# 设置为true来锁住内存 因为当jvm开始swapping时es的效率会降低，所以要保证它不swap
bootstrap.memory_lock: true
bootstrap.system_call_filter: false
# ---------------------------------- Network -----------------------------------
# 绑定的ip地址，本机ip  不写是localhost Set the bind address to a specific IP (IPv4 or IPv6):
# network.host: 192.168.0.1
network.host: 0.0.0.0
#设置参与集群的端口
transport.tcp.port: 9300
# 端口号 默认是9200 Set a custom port for HTTP:
http.port: 9200
# 设置内容的最大容量，默认100mb
http.max_content_length: 100mb
# ---------------------------------- Discovery -----------------------------------
# 单播发现的地址Pass an initial list of hosts to perform discovery when new node is started:
# The default list of hosts is ["127.0.0.1", "[::1]"] 通过配置这个参数来防止集群脑裂现象 (集群总节点数量/2)+1
# 最小master节点选举数 Prevent the "split brain" by configuring the majority of nodes (total number of master-eligible nodes / 2 + 1):
discovery.zen.ping_timeout: 30s
discovery.zen.minimum_master_nodes: 2
discovery.zen.fd.ping_timeout: 30s
discovery.zen.ping.unicast.hosts: ["192.168.10.135:9300", "192.168.10.136:9300", "192.168.10.137:9300"]
# discovery.zen.ping.multicast.enabled: false
# ---------------------------------- Gateway -----------------------------------
# 最多等待5分钟，5分钟还没上线 就重新rebalance
# 足够的节点上线后，才进行shard recovery的过程
# 节点最少数量
gateway.recover_after_time: 5m
gateway.recover_after_nodes: 3
gateway.expected_nodes: 3
# ---------------------------------- Various -----------------------------------
# 禁止在生产环境中删除所有索引 Require explicit names when deleting indices:
# 设置是否可以通过正则或者_all删除或者关闭索引库，默认true表示必须需要显式指定索引库名称
# 生产环境建议设置为true，删除索引库的时候必须显式指定，否则可能会误删索引库中的索引库。
action.destructive_requires_name: true
# ---------------------------------- http.cors -----------------------------------
# 运行带cookie访问-用于head插件
http.cors.enabled: true
http.cors.allow-origin: "*"
```

集群配置

修改每个节点的`elasticsearch.yml`：

```
# 集群名称要唯一
cluster.name: kkb-es
# 集群节点要不同
node.name: node-135
# 设置master和data节点
node.master: true
node.data: true
# 节点将绑定到此主机名或IP地址，我们设置成0.0.0.0为当前主机，允许ip映射访问
network.host: 0.0.0.0
# http访问端口
http.port: 9200
# tcp访问端口
transport.tcp.port: 9300
# 自动发现的路由节点
discovery.zen.ping.unicast.hosts:
["192.168.10.135:9300","192.168.10.136:9300","192.168.10.137:9300"]
# 集群中最小主节点数，防止脑裂
discovery.zen.minimum_master_nodes: 2
# head 跨域访问
http.cors.enabled: true
http.cors.allow-origin: "*"
```

head插件安装

【第一步】下载

```
yum install -y epel-release
yum install -y nodejs npm

yum install -y git
cd /kkb/server
git clone git://github.com/mobz/elasticsearch-head.git
```

【第二步】安装依赖包

安装cnpm（从阿里镜像进行下载，速度会快）

```
npm install cnpm -g --registry=https://registry.npm.taobao.org

cd /kkb/server/elasticsearch-head

cnpm install -g  //执行后会生成node_modules文件夹

cnpm install -g grunt-cli //将grunt命令放入系统环境变量中
```

【第三步】配置文件修改

- Gruntfile.js

```
vim /kkb/server/elasticsearch-head/Gruntfile.js
```

设置为

```
hostname: '*'
```

- app.js

```
 vim /kkb/server/elasticsearch-head/_site/app.js
```

```
app-base_uri") || "http://192.168.10.136:9200"
```

- elasticsearch.yml

  允许跨域访问

```
vim /kkb/server/elasticsearch-6.5.3/config/elasticsearch.yml
```

  在该文件的最后部分添加以下内容：

```
http.cors.enabled: true
http.cors.allow-origin: "*"
```

【第四步】启动head

```
nohup grunt server &
```

访问地址：http://192.168.10.136:9100/

Kibana安装

介绍

Kibana是一个针对Elasticsearch的开源分析及可视化平台，使用Kibana可以查询、查看并与存储在ES索引的数据进行交互操作，使用Kibana能执行高级的数据分析，并能以图表、表格和地图的形式查看数据

下载

- 下载地址：https://www.elastic.co/downloads/kibana
- 下载安装包（需要和`es`版本一致）：

```
wget https://artifacts.elastic.co/downloads/kibana/kibana-6.5.3-linux-x86_64.tar.gz
```

安装

- 第一步：解压缩

```
tar -xf kibana-6.5.3-linux-x86_64.tar.gz -C /kkb/server/
```

- 第二步：编辑`kibana.yml`文件

```
vim /kkb/server/kibana-6.5.3-linux-x86_64/config/kibana.yml
```

将`server.host、elasticsearch.url`修改成所在服务器的ip地址：

```
server.host: "192.168.10.135"
elasticsearch.url: "http://192.168.10.135:9200"
```

- 第三步：关闭防火墙

```
systemctl stop firewalld
```

- 第四步：启动Kibana

```
nohup /kkb/server/kibana-6.5.3-linux-x86_64/bin/kibana &
```

- 第五步：通过浏览器访问`Kibana`

```
http://192.168.10.135:5601
```

安装中文分词器IK

下载

- 下载地址：https://github.com/medcl/elasticsearch-analysis-ik/releases
- 下载压缩包

```
wget https://github.com/medcl/elasticsearch-analysis-ik/releases/download/v6.5.3/elasticsearch-analysis-ik-6.5.3.zip
```

安装

- 第一步：安装unzip工具（因为下载的压缩包是zip格式的）

```
yum install -y unzip
```

- 第二步：解压到`/kkb/server/elasticsearch-6.5.3/plugins/ik`目录

```
unzip elasticsearch-analysis-ik-6.5.3.zip -d /kkb/server/elasticsearch-6.5.3/plugins/ik
```

- 第三步：重新启动ES

```
[root@bogon soft]# jps
33346 Jps
32850 Elasticsearch
28328 activemq.jar
30973 QuorumPeerMain
31245 Bootstrap

kill -9 33346

su esuser
/kkb/server/elasticsearch-6.5.3/bin/elasticsearch -d
```

测试中文分词

在`head插件`中或者`Kibana`中执行以下测试：

```
POST
http://172.16.86.101:9200/_analyze
{
  "analyzer": "ik_smart",
  "text": "我是一个程序员，咿呀咿呀呀！"
}
```

`ik_smart`，`ik_max_word`是`ik`的两种分词器：

- `ik_max_word`: 会将文本做最细粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华人民共和国,中华人民,中华,华人,人民共和国,人民,人,民,共和国,共和,和,国国,国歌”，会穷尽各种可能的组合；
- `ik_smart`: 会做最粗粒度的拆分，比如会将“中华人民共和国国歌”拆分为“中华人民共和国,国歌”。

创建ik的mapping

- 对kkb索引下的text字段设置分词器

```
POST
http://172.16.86.101:9200/kkb/java/_mapping
{
        "properties: {
            "info": {
                "type": "text",
                "analyzer": "ik_max_word",
                "search_analyzer": "ik_max_word"
            }
        }
}
```

- 存放多个文档值

```
POST
http://172.16.86.101:9200/kkb/java/1
{"info": "我是一个程序员，咿呀咿呀呀！"}
```

```
POST
http://172.16.86.101:9200/kkb/java/2
{"info": "我是一个产品，咿呀咿呀呀！"}
```

```
POST
http://172.16.86.101:9200/kkb/java/3
{"info": "程序员要搂产品！"}
```

- 查询

```
POST
http://172.16.86.101:9200/kkb/fulltext/_search
{
    "query" : { "match" : { "info" : "程序" }}
}
```

结果会根据词库去匹配去匹配

- 词库配置 词库配置在IKAnalyzer.cfg.xml 位置

```
{conf}/analysis-ik/config/IKAnalyzer.cfg.xml
或者jar里
{plugins}/elasticsearch-analysis-ik-*/config/IKAnalyzer.cfg.xml


```

文件内容

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
    <comment>IK Analyzer 扩展配置</comment>
    <!--用户可以在这里配置自己的扩展字典 -->
    <entry key="ext_dict">custom/mydict.dic;custom/single_word_low_freq.dic</entry>
     <!--用户可以在这里配置自己的扩展停止词字典-->
    <entry key="ext_stopwords">custom/ext_stopword.dic</entry>
    <!--用户可以在这里配置远程扩展字典 -->
    <entry key="remote_ext_dict">location</entry>
    <!--用户可以在这里配置远程扩展停止词字典-->
    <entry key="remote_ext_stopwords">http://xxx.com/xxx.dic</entry>
</properties>
```

- 官方网站

```
https://github.com/medcl/elasticsearch-analysis-ik
```

可能遇到的错误

- java.net.SocketPermission 网络访问权限问题 修改jdk/lib/java.policy文件

```
permission java.net.SocketPermission "*", "connect,resolve";
```

分词器的使用

使用

在创建mapping的时候对指定的字段添加分词器

自带分词器

ES 有自带的分词器，但是对中文的支持很不友好，下面我们简单说下自带的几个分词器； 可以通过analyzer 字段来选择分词器

- standard 默认的分词器，对词汇转换成小写切割，去掉标点和助词（a/an/the），支持中文但是是根据文字单个切分

```
POST
http://172.16.86.101:9200/_analyze
{
  "analyzer": "standard",
  "text": "我是一个程序员，咿呀咿呀呀！"
}
```

```
POST
http://172.16.86.101:9200/_analyze
{
  "analyzer": "standard",
  "text": "I'm a programmer!o my god!"
}
```

会看到结果,中文直接按照每个字拆分

- simple 通过非字母字符来切割，会过滤掉数字

```
POST
http://172.16.86.101:9200/_analyze
{
  "analyzer": "simple",
  "text": "我是一个程序员，咿呀咿呀呀！"
}
```

这个分词粒度比较大，如果不需要太细的搜索，力度大了效率会高，具体看实际场景来使用

- Whitespace 去空格
- lowercase 小写转换
- language 特定语言选择，不支持中文
- custom 用户自定义分词器
- 官方地址

```
https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html
```

Mapping映射

如果不添加，生成时系统会默认指定mapping（映射）结构，检索时会系统会猜测你想要的类型，如果对系统反馈的不满意，我们就可以手动设置

- 添加

```
PUT
http://172.16.86.101:9200/kkb
{
    "mappings": {
      "java": {
        "properties: {
          "age": { "type": "long"}
        }
      }
    }

```

- 获取

```
GET
http://172.16.86.101:9200/kkb/_mapping
模糊匹配
http://172.16.86.101:9200/kkb/_all/_mapping/k*’

```

- 配置 mapping是对索引字段的映射配置，可以直接对索引文档字段设置

```
{
    "type" : "text", #是数据类型一般文本使用text(可分词进行模糊查询)；keyword无法被分词(不需要执行分词器)，用于精确查找

    "analyzer" : "ik_max_word", #指定分词器，一般使用最大分词：ik_max_word

    "normalizer" : "normalizer_name", #字段标准化规则；如把所有字符转为小写；具体如下举例

    "boost" : 1.5, #字段权重；用于查询时评分，关键字段的权重就会高一些，默认都是1；另外查询时可临时指定权重

    "coerce" : true, #清理脏数据：1，字符串会被强制转换为整数 2，浮点数被强制转换为整数；默认为true

    "copy_to" : "field_name", #自定_all字段；指定某几个字段拼接成自定义；具体如下举例

    "doc_values : true, #加快排序、聚合操作，但需要额外存储空间；默认true，对于确定不需要排序和聚合的字段可false

    "dynamic" : true, #新字段动态添加 true:无限制 false:数据可写入但该字段不保留 'strict':无法写入抛异常

    "enabled" : true, #是否会被索引，但都会存储;可以针对一整个_doc

    "fielddata" : false, #针对text字段加快排序和聚合（doc_values对text无效）；此项官网建议不开启，非常消耗内存

    "eager_global_ordinals": true, #是否开启全局预加载,加快查询；此参数只支持text和keyword，keyword默认可用，而text需要设置fielddata属性

    "format" : "yyyy-MM-dd HH:mm:ss||yyyy-MM-dd||epoch_millis" ,#格式化 此参数代表可接受的时间格式 3种都接受

    "ignore_above" : 100, #指定字段索引和存储的长度最大值，超过最大值的会被忽略

    "ignore_malformed" : false ,#插入文档时是否忽略类型 默认是false 类型不一致无法插入

    "index_options" : "docs" ,
    # 4个可选参数
    # docs（索引文档号），
    # freqs（文档号 + 词频），
    # positions（文档号 + 词频 + 位置，通常用来距离查询），
    # offsets（文档号 + 词频 + 位置 + 偏移量，通常被使用在高亮字段）
    # 分词字段默认是position，其他的默认是docs

```

```
37
38      "index" : true, #该字段是否会被索引和可查询 默认true
39
40      "fields": {"raw": {"type": "keyword"}} ,#可以对一个字段提供多种索引模式，使用text类型做全文检索，也可使用keyword类型做聚合和排序
41
42      "norms" : true, #用于标准化文档，以便查询时计算文档的相关性。建议不开启
43
44      "null_value" : "NULL", #可以让值为null的字段显式的可索引、可搜索
45
46      "position_increment_gap" : 0 ,#词组查询时可以跨词查询 既可变为分词查询 默认100
47
48      "properties : {}, #嵌套属性，例如该字段是音乐，音乐还有歌词，类型，歌手等属性
49
50      "search_analyzer" : "ik_max_word" ,#查询分词器;一般情况和analyzer对应
51
52      "similarity" : "BM25",#用于指定文档评分模型，参数有三个：
53      # BM25 ：ES和Lucene默认的评分模型
54      # classic ：TF/IDF评分
55      # boolean：布尔模型评分
56
57      "store" : true, #默认情况false,其实并不是真没有存储，_source字段里会保存一份原始文档。
58      # 在某些情况下，store参数有意义，比如一个文档里面有title、date和超大的content字段，如果只想获取title和date
59
60      "term_vector" : "no" #默认不存储向量信息，
61      # 支持参数yES（term存储），
62      # with_positions（term + 位置），
63      # with_offsets（term + 偏移量），
64      # with_positions_offsets(term + 位置 + 偏移量)
65      # 对快速高亮fast vector highlighter能提升性能，但开启又会加大索引体积，不适合大数据量用
66  }
67
```

十五、ElasticSearch客户端访问

浏览器客户端

Java客户端

索引操作

搜索操作