# Solr

## 相关技术

Solr是用JAVA开发的一个基于Lucene的高性能的全文搜索服务器，它通过对Lucene向外提供的API进行扩展，形成的一个高效的、独立的企业级搜索引擎应用服务器。相较于Lucene, Solr有更加丰富的查询语言和功能，并且实现了系统的可配置性和可扩展性。同时Sol:对于信息查询的性能也进行了优化，提供给了用户一个完善的功能管理界面，是一款非常不错的全文搜索引擎，很便于W EB应用程序的开发使用。

Solr跟Lucene一样，也向外提供了一些API接口，这样便于开发者使用和功能的扩展。Sol:的输出结果支持多种格式的文件，包括XML/XSLT和JSON格式。用户可以通过Http的请求与Solr服务器进行交互，将一定格式的XML文件提交给搜索引擎的服务器，在服务器端生成索引文件。同时用户也可以通过Http Get操作向服务器提出查找请求，服务器响应之后返回XML/XSLT或者JSON格式的结果。

Solr是一款性能优异的全文搜索引擎，其自身的主要特点包括如下几点:

垂直的搜索功能:Solr有高效率的分类功能，它可以根据特定的分类进行同类别的搜索，实现垂直搜索功能。

高效、灵活的缓存功能:Solr系统支持多种方式的缓存机制。高亮显示搜索结果:对于搜索到的结果中与检索的字符相匹配的字符自动高亮显示，便于搜索引擎用户对结果进行筛选。

分布式搜索:Solr支持一台或者多台服务器进行信息的索引和搜索，同时也可以将大的索引分割成若干个小的索引，一次搜索，多台服务器同时进行，通过这样来提高搜索的效率。

支持数据库的导入:Solr系统支持数据库和其他格式的数据源导入，这些数据导入之后，会通过映射和转化，变成Lucene的索引格式。

易于操作、管理、配置:Solr提供给用户一个W EB的管理页面，用户可以通过WEB页面进行操作和管理。而Sol:的配置文件通过XML格式的文件进行的，配置起来比较方便。

功能扩展:Solr有比较灵活的插件扩展体系，支持功能的扩展，新功能的添加通过插件的形式加入到服务器上。

### 数据导入

导入数据到Solr的方式是多种多样的：

* 可以使用DIH(DataImportHandler)从数据库导入数据
* 支持CSV文件导入，因此Excel数据也能轻松导入
* 支持JSON格式文档
* 二进制文档比如：Word、PDF
* 还能以编程的方式来自定义导入

### 更新数据

如果同一份文档solr.xml重复导入会出现什么情况呢？实际上solr会根据文档的字段id来唯一标识文档，如果导入的文档的id已经存在solr中，那么这份文档就被最新导入的同id的文档自动替换。你可以自己尝试试验一下，观察替换前后管理界面的几个参数：Num Docs，Max Doc，Deleted Docs的变化。

* numDocs：当前系统中的文档数量，它有可能大于xml文件个数，因为一个xml文件可能有多个<doc>标签。
* maxDoc：maxDoc有可能比numDocs的值要大，比如重复post同一份文件后，maxDoc值就增大了。
* deletedDocs：重复post的文件会替换掉老的文档，同时deltedDocs的值也会加1，不过这只是逻辑上的删除，并没有真正从索引中移除掉

### 删除数据

通过id删除指定的文档，或者通过一个查询来删除匹配的文档

|  |  |
| --- | --- |
| 1  2 | java -Ddata=args -jar post.jar "<delete><id>SOLR1000</id></delete>"  java -Ddata=args -jar post.jar "<delete><query>name:DDR</query></delete>" |

此时solr.xml文档从索引中删除了，再次搜”solr”时不再返回结果。当然solr也有数据库中的事务，执行删除命令的时候事务自动提交了，文档就会立即从索引中删除。你也可以把commit设置为false，手动提交事务。

|  |  |
| --- | --- |
| 1 | java -Ddata=args  -Dcommit=false -jar post.jar "<delete><id>3007WFP</id></delete>" |

执行完上面的命令时文档并没有真正删除，还是可以继续搜索相关结果，最后可以通过命令：

|  |  |
| --- | --- |
| 1 | java -jar post.jar - |

提交事务，文档就彻底删除了。现在把刚刚删除的文件重新导入Solr中来，继续我们的学习。

删除所有数据

http://localhost:8983/solr/collection1/update?stream.body=<delete><query>\*:\*</query></delete>&commit=true  
删除指定数据

http://localhost:8983/solr/collection1/update?stream.body=<delete><query>title:abc</query></delete>&commit=true

多条件删除

http://localhost:8983/solr/collection1/update?stream.body=<delete><query>title:abc AND name:zhang</query></delete>&commit=true

### 查询数据

查询数据都是通过HTTP的GET请求获取的，搜索关键字用参数q指定，另外还可以指定很多可选的参数来控制信息的返回，例如：用fl指定返回的字段，比如f1=name，那么返回的数据就只包括name字段的内容

|  |  |
| --- | --- |
| 1 | http://localhost:8983/solr/collection1/select?q=solr&fl=name&wt=json&indent=true |

### 排序

Solr提供排序的功能，通过参数sort来指定，它支持正序、倒序，或者多个字段排序

* + q=video&sort=price desc
  + q=video&sort=price asc
  + q=video&sort=inStock asc, price desc  
    默认条件下，Solr根据socre 倒序排列，socre是一条搜索记录根据相关度计算出来的一个分数。

### 高亮

网页搜索中，为了突出搜索结果，可能会对匹配的关键字高亮出来，Solr提供了很好的支持，只要指定参数：

* + hl=true #开启高亮功能
  + hl.fl=name #指定需要高亮的字段

<http://localhost:8983/solr/collection1/select?q=Search&wt=json&indent=true&hl=true&hl.fl=features>

返回的内容中包含：

"highlighting":{

       "SOLR1000":{

           "features":["Advanced Full-Text <em>Search</em> Capabilities using Lucene"]

       }

}

### 文本分析

文本字段通过把文本分割成单词以及运用各种转换方法（如：小写转换、复数移除、词干提取）后被索引，schema.xml文件中定义了字段在索引中，这些字段将作用于其中.  
默认情况下搜索”power-shot”是不能匹配”powershot”的，通过修改schema.xml文件(solr/example/solr/collection1/conf目录)，把features和text字段替换成”text\_en\_splitting”类型，就能索引到了。

<field name="features" type="text\_en\_splitting" indexed="true" stored="true"multiValued="true"/>

。。。

<field name="text" type="text\_en\_splitting" indexed="true" stored="false"multiValued="true"/>

修改完后重启solr，然后重新导入文档

|  |  |
| --- | --- |
| 1 | java -jar post.jar \*.xml |

现在就可以匹配了

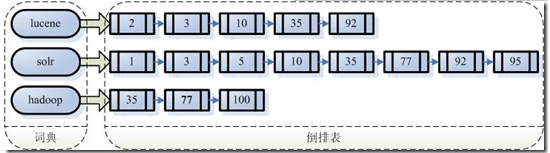
* power-shot—>Powershot
* features:recharing—>Rechargeable
* 1 gigabyte –> 1G

## 基本原理

归结为两个过程：1.索引创建（Indexing）2. 搜索索引（Search）

### 索引

Solr/Lucene采用的是一种反向索引，所谓**反向索引**：就是从关键字到文档的映射过程，保存这种映射这种信息的索引称为反向索引

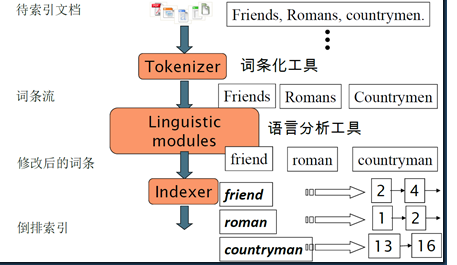


* 左边保存的是字符串序列
* 右边是字符串的文档（Document）编号链表，称为倒排表（Posting List）

字段串列表和文档编号链表两者构成了一个字典。现在想搜索”lucene”，那么索引直接告诉我们，包含有”lucene”的文档有：2，3，10，35，92，而无需在整个文档库中逐个查找。如果是想搜既包含”lucene”又包含”solr”的文档，那么与之对应的两个倒排表去交集即可获得：3、10、35、92。

### 索引创建

假设有如下两个原始文档：  
文档一：Students should be allowed to go out with their friends, but not allowed to drink beer.  
文档二：My friend Jerry went to school to see his students but found them drunk which is not allowed.  
创建过程大概分为如下步骤



**一：把原始文档交给分词组件(Tokenizer)**  
分词组件(Tokenizer)会做以下几件事情(这个过程称为：Tokenize)，处理得到的结果是词汇单元（Token）

1. 将文档分成一个一个单独的单词
2. 去除标点符号
3. 去除停词(stop word)
   * 所谓停词(Stop word)就是一种语言中没有具体含义，因而大多数情况下不会作为搜索的关键词，这样一来创建索引时能减少索引的大小。英语中停词(Stop word)如：”the”、”a”、”this”，中文有：”的，得”等。不同语种的分词组件(Tokenizer)，都有自己的停词(stop word)集合。经过分词(Tokenizer)后得到的结果称为词汇单元(Token)。上例子中，便得到以下**词汇单元(Token)**：

"Students"，"allowed"，"go"，"their"，"friends"，"allowed"，"drink"，"beer"，"My"，"friend"，"Jerry"，"went"，"school"，"see"，"his"，"students"，"found"，"them"，"drunk"，"allowed"

**二：词汇单元(Token)传给语言处理组件(Linguistic Processor)**  
语言处理组件(linguistic processor)主要是对得到的词元(Token)做一些语言相关的处理。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)处理得到的结果称为**词(Term)**，例子中经过语言处理后得到的词(Term)如下：

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

经过语言处理后，搜索drive时drove也能被搜索出来。**Stemming 和 lemmatization的异同：**

* 相同之处：
  1. Stemming和lemmatization都要使词汇成为词根形式。
* 两者的方式不同：
  1. Stemming采用的是”缩减”的方式：”cars”到”car”，”driving”到”drive”。
  2. Lemmatization采用的是”转变”的方式：”drove”到”drove”，”driving”到”drive”。
* 两者的算法不同：
  1. Stemming主要是采取某种固定的算法来做这种缩减，如去除”s”，去除”ing”加”e”，将”ational”变为”ate”，将”tional”变为”tion”。
  2. Lemmatization主要是采用事先约定的格式保存某种字典中。比如字典中有”driving”到”drive”，”drove”到”drive”，”am, is, are”到”be”的映射，做转变时，按照字典中约定的方式转换就可以了。
  3. Stemming和lemmatization不是互斥关系，是有交集的，有的词利用这两种方式都能达到相同的转换。

**三：得到的词(Term)传递给索引组件(Indexer)**

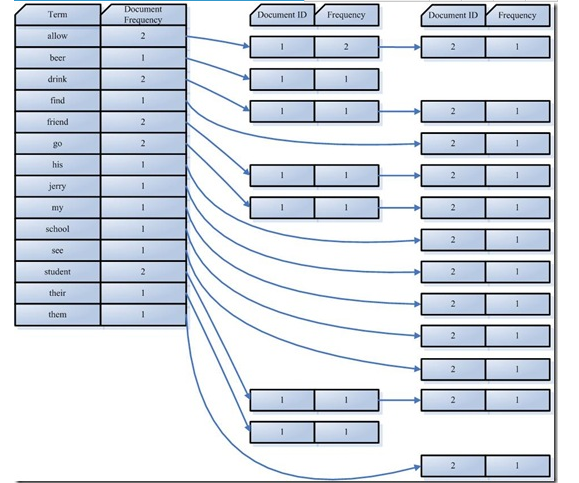
1. 利用得到的词(Term)创建一个字典
2. Term Document ID
3. student 1
4. allow 1
5. go 1
6. their 1
7. friend 1
8. allow 1
9. drink 1
10. beer 1
11. my 2
12. friend 2
13. jerry 2
14. go 2
15. school 2
16. see 2
17. his 2
18. student 2
19. find 2
20. them 2
21. drink 2

allow 2

1. 对字典按字母顺序排序：
2. Term Document ID
3. allow 1
4. allow 1
5. allow 2
6. beer 1
7. drink 1
8. drink 2
9. find 2
10. friend 1
11. friend 2
12. go 1
13. go 2
14. his 2
15. jerry 2
16. my 2
17. school 2
18. see 2
19. student 1
20. student 2
21. their 1

them 2

1. 合并相同的词(Term)成为文档倒排(Posting List)链表



* + Document Frequency：文档频次，表示多少文档出现过此词(Term)
  + Frequency：词频，表示某个文档中该词(Term)出现过几次

对词(Term) “allow”来讲，总共有两篇文档包含此词(Term)，词（Term)后面的文档链表总共有两个，第一个表示包含”allow”的第一篇文档，即1号文档，此文档中，”allow”出现了2次，第二个表示包含”allow”的第二个文档，是2号文档，此文档中，”allow”出现了1次

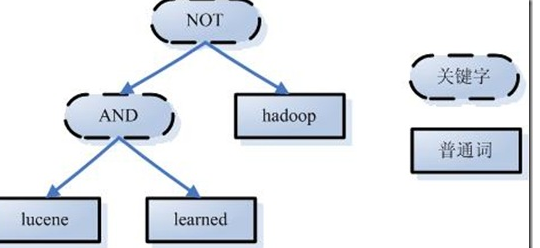
至此索引创建完成，搜索”drive”时，”driving”，”drove”，”driven”也能够被搜到。因为在索引中，”driving”，”drove”，”driven”都会经过语言处理而变成”drive”，在搜索时，如果您输入”driving”，输入的查询语句同样经过分词组件和语言处理组件处理的步骤，变为查询”drive”，从而可以搜索到想要的文档。

### 搜索步骤

搜索”microsoft job”，用户的目的是希望在微软找一份工作，如果搜出来的结果是:”Microsoft does a good job at software industry…”，这就与用户的期望偏离太远了。如何进行合理有效的搜索，搜索出用户最想要得结果呢？搜索主要有如下步骤：

#### 一：对查询内容进行词法分析、语法分析、语言处理

1. 词法分析：区分查询内容中单词和关键字，比如：english and janpan，”and”就是关键字，”english”和”janpan”是普通单词。
2. 根据查询语法的语法规则形成一棵树



1. 语言处理，和创建索引时处理方式是一样的。比如：leaned–>lean，driven–>drive

#### **二：搜索索引，得到符合语法树的文档集合**

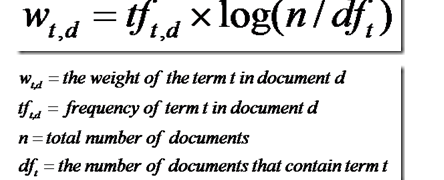
#### **三：根据查询语句与文档的相关性，对结果进行排序**

我们把查询语句也看作是一个文档，对文档与文档之间的相关性（relevance）进行打分（scoring），分数高比较越相关，排名就越靠前。当然还可以人工影响打分，比如百度搜索，就不一定完全按照相关性来排名的。

如何评判文档之间的相关性？一个文档由多个（或者一个）词（Term）组成，比如：”solr”， “toturial”，不同的词可能重要性不一样，比如solr就比toturial重要，如果一个文档出现了10次toturial，但只出现了一次solr，而另一文档solr出现了4次，toturial出现一次，那么后者很有可能就是我们想要的搜的结果。这就引申出权重（Term weight）的概念。

**权重**表示该词在文档中的重要程度，越重要的词当然权重越高，因此在计算文档相关性时[影响力](http://www.amazon.cn/gp/product/B0044KME2E/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&tag=importnew-23&linkCode=as2&camp=536&creative=3200&creativeASIN=B0044KME2E)就更大。通过词之间的权重得到文档相关性的过程叫做**空间向量模型算法(Vector Space Model)**

影响一个词在文档中的重要性主要有两个方面：

* Term Frequencey（tf），Term在此文档中出现的频率，ft越大表示越重要
* Document Frequency（df），表示有多少文档中出现过这个Trem，df越大表示越不重要  
  物以希为贵，大家都有的东西，自然就不那么贵重了，只有你专有的东西表示这个东西很珍贵，权重的公式：
* 

空间向量模型

文档中词的权重看作一个向量

Document = {term1, term2, …… ,term N}

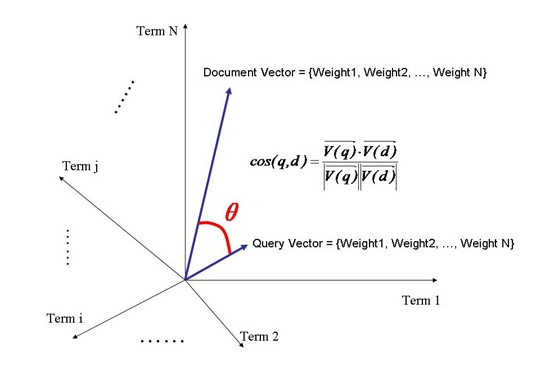
Document Vector = {weight1, weight2, …… ,weight N}

把欲要查询的语句看作一个简单的文档，也用向量表示：

Query = {term1, term 2, …… , term N}

Query Vector = {weight1, weight2, …… , weight N}

把搜索出的文档向量及查询向量放入N维度的空间中，每个词表示一维：



夹角越小，表示越相似，相关性越大

## Solr的索引结构

Solr是在Lucene的基础上设计开发的，因此在关键的数据索引结构上都是采取的相同的索引结构一一倒排文件索引结构。所谓倒排索引，就是利用属性值来确定位置信息，建立对应的索引。下面通过一个实例来详细讲解该结构和生成的算法:

现在有两篇能容不同的文章:文章1和文章2 o Sol:要为这两篇文章建立索引，两篇文章的内容如下:

文章1的内容是:David is a Chinese o He lives in Beijing o

文章2的内容是:Tom comes from England o He once lived in London

Solr是通过关键字进行索引和查询的，所以在建立索引之前，应该把文章中的关键字给提取出来。在提取关键字的过程中，需要遵照以下规则进行:

a)需要对文章的内容进行分词，将文章中的字符串分成单个词汇。实例中的文章的是英文的，因此这里根据空格来进行划分。假如是中文的话，则需要加入一些中文的分词规则来进行分词，后面会详细讲解中文分词，这里不作详述。

b)文章中一些没有实际意义的单词，需要被滤掉。文章中的“is" , "in" , "from" , "once"等单词没有太多的实际意义，需要被过滤掉。在中文文章中，像“的”、“是”等词汇也会因为没有实际意义而被滤掉。

c)在提取关键字时，所有的单词的大小写需要统一。

d)文章中的动词无论是什么时态的，都需要还原成动词原形。实例中的‘`lives"和 "lived”都需要被还原成“live" o

e)文章中的标点符号不作为关键字，可以被滤掉。

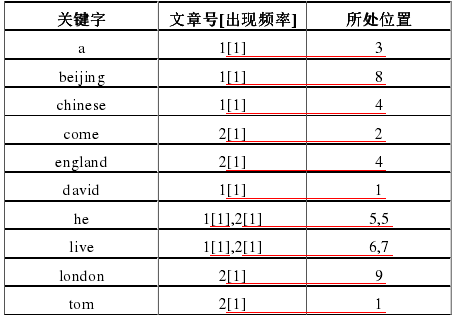
根据上面的规则，两个文章中提取出来的关键字分别为:

文章1的关键字:[david] [a] [chinese] [he] [five] [beijing]

文章2的关键字:[tom] [come] [england] [he] [live] [london]

在提取完关键字之后，就可以开始建立关键词的倒排索引了。倒排索引的对应关系与正常的对应关系相反，是由“关键字”对应“关键字所在的文章号”。根据这个对应关系，文章1和文章2经过倒排之后的对应关系如表所示:

在为关键字建立索引时，只知道关键字在出现在哪些文章是不够的，还需要知道关键字的一些详细信息，包括:关键字在文章中出现的位置，关键字在文章中出现的频率。下表是加上了关键字“所处位置”和“出现频率”之后的倒排索引：



上表是一个比较完整的倒排索引表，结合文章号、出现频率、所处位置就能准确的表示出文章中的关键字。Solr中的索引就是这样建立的。

## 与Lucene比较

介绍了Solr和Lucene的基本功能之后，下面将它们做一个比较。Solr不是对Lucene的否定，而是对其功能的一个扩展和升级。Solr是基于Lucene进行包装的，是一个企业搜索服务器的开发框架。开发者可以利用Solr项目文件快速的开发自己的搜索引擎服务器。

虽然Solr是基于Lucene开发的，但是两者之间还是有写区别，主要体现在:在使用上，Solr更加贴近实际开发应用，是Lucene面向企业搜索服务领域的功能扩展。Solr灵活的缓存机制，使得全文搜索引擎的性能得到了提升。Solr可以通过配置文件进行功能的扩展，并且Solr有良好的用户管理的WEB界面，使得操作更加方便，更加人性化。