Nutch 入门学习

北京邮电大学

李阳

目 录

1. nutch 简介

1.1 什么是 nutch

Nutch 是一个开源的、Java 实现的搜索引擎。它提供了我们运行自己的搜索引擎所需的全部工具。

1.2 研究 nutch 的原因

可能有的朋友会有疑问,我们有 google,有百度,为何还需要建立自己的搜索引擎呢？这里我列出 3 点原因：

(1) 透明度：nutch 是开放源代码的，因此任何人都可以查看他的排序算法是如何工作的。商业的搜索引擎排序算法都是保密的，我们无法知道为什么搜索出来的排序结果是如何算出来的。更进一步，一些搜索引擎允许竞价排名，比如百度，这样的索引结果并不是和站点内容相关的。因此 nutch 对学术搜索和政府类站点的搜索来说，是个好选择，因为一个公平的排序结果是非常重要的。

(2) 对搜索引擎的理解：我们并没有 google 的源代码，因此学习搜索引擎 Nutch 是个不错的选择。了解一个大型分布式的搜索引擎如何工作是一件让人很受益的事情。在写 Nutch 的过程中，从学院派和工业派借鉴了很多知识：比如，Nutch 的核心部分目前已经被重新用 Map Reduce 实现了。Map Reduce 是一个分布式的处理模型，最先是从 Google 实验室提出来的。并且 Nutch 也吸引了很多研究者，他们非常乐于尝试新的搜索算法，因为对 Nutch 来说，这是非常容易实现扩展的。

(3) 扩展性：你是不是不喜欢其他的搜索引擎展现结果的方式呢？那就用 Nutch 写你自己的搜索引擎吧。 Nutch 是非常灵活的：他可以被很好的客户订制并集成到你的应用程序中，使用 Nutch 的插件机制，Nutch 可以作为一个搜索不同信息载体的搜索平台。当然，最简单的就是集成 Nutch 到你的站点，为你的用户提供搜索服务。

1.3 nutch 的目标

nutch 致力于让每个人能很容易，同时花费很少就可以配置世界一流的 Web 搜索引擎. 为了完成这一宏伟的目标，nutch 必须能够做到:

- 每个月取几十亿网页

- 为这些网页维护一个索引
- 对索引文件进行每秒上千次的搜索
- 提供高质量的搜索结果
- 以最小的成本运作

这将是一个巨大的挑战。

1.4 nutch VS lucene

简单的说:

- Lucene 不是完整的应用程序，而是一个用于实现全文检索的软件库。
- Nutch 是一个应用程序，可以以 Lucene 为基础实现搜索引擎应用。

Lucene 为 Nutch 提供了文本索引和搜索的 API。一个常见的问题是；我应该使用 Lucene 还是 Nutch？最简单的回答是：如果你不需要抓取数据的话，应该使用 Lucene。常见的应用场合是：你有数据源，需要为这些数据提供一个搜索页面。在这种情况下，最好的方式是直接从数据库中取出数据并用 Lucene API 建立索引。

2. nutch 的安装与配置

- ✓ 安装环境 ：FC5 Linux
- ✓ 所安装软件 ：jdk-6u1-linux-i586.bin

  compat-libstdc++-33-3.2.3-55.fc5.i386.rpm

  nutch-0.8.1.tar.gz

  jakarta-tomcat-5.0.28.tar.gz

  - ✧ 我将软件默认安装在主文件夹下(/root)
- ✓ 下载网址 ：
  - ➢ jdk: http://java.sun.com/
  - ➢ libstdc++: http://fr2.rpmfind.net/
  - ➢ nutch : http://lucene.apache.org/nutch
  - ➢ tomcat : http://tomcat.apache.org/

2.1 JDK 的安装与配置

2.1.1 问题的由来

FC5 默认是使用 gcj 作为 java 虚拟机的，gcj 是用于 java 的 GNU 编译器。但是很多情况下，我们希望使用 SUN 的 JDK。

2.1.2 JDK 的下载与安装

(1) 下载 bin 包

```
http://java.sun.com/
jdk-6u1-linux-i586.bin
```

bin 包这是 J2SE Development Kit 1.5 update 5 for Linux 的 sh 自解压执行脚本，里面已经包含安装压缩包。

(2)到 JDK 的下载目录，执行命令：

```
[root@localhost ~]#sh jdk-6u1-linux-i586.bin
```

(3)JDK 默认安装在/usr/java/jdk1.6.0_01 下，但版本不同，安装路径可能不同。

2.1.3 JDK 的配置

(1)取代 gcj

- ■ 创建快捷方式以取代 gcj

```
[root@localhost ~]#cd /usr/bin
[root@localhost ~]#ln -s -f /usr/java/jdk1.6.0_01/jre/bin/java
[root@localhost ~]#ln -s -f /usr/java/jdk1.6.0_01/bin/javac
```

- ln 命令的-s 参数表示建立符号链接，-f 参数表示强制覆盖原来已经存在的 java 静态链接文件。
- **注意**：这个步骤并没有把 gcj 从您的电脑中彻底删除。您仍然可以使用 whereis gcj 命令找到它，并在需要的时候使用它。

(2)配置 java 环境变量

- 编辑~/.bashrc 文件

```
[root@localhost ~]#vi ~/.bashrc
```

- 在.bashrc 文件最后加上下面四行

```
export JAVA_HOME=/usr/jdk/jdk1.6.0_01
export JAVA_BIN=/usr/jdk/jdk1.6.0_01/bin
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```

(3) 配置浏览器插件

- 转到 mozilla 安装目录的 plugins 目录下建立一个到 jre 的符号链接
- 以我的为例(mozilla 版本为 1.7.12)

```
[root@localhost ~]#cd /usr/lib/mozilla-1.7.12/plugins
[root@localhost~]#ln -s
/usr/java/jdk1.6.0_01/jre/plugin/i386/ns7/libjavaplugin_oji.so
```

(4) Java 控制台的调用

```
[root@localhost ~]#/usr/java/jdk1.6.0_01/bin/ControlPanel
```

(5) 测试

- 可以到 sun 的中文网站上进行测试，以验证你的 java 版本

```
http://www.java.com/zh_CN/download/help/testvm.xml
```

(6) 如果未通过以上测试，或者调用 java 控制台时出现以下错误

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
/usr/java/jdk1.6.0_01 /jre/lib/i386/libdeploy.so: libstdc++.so.5:
cannot open shared object file: No s uch file or directory
```

则进行此步，否则跳过此步

- 下载 libstdc++(rpm 包)

```
http://fr2.rpmfind.net/
```

- 运行安装

```
[root@localhost~]#rpm -ivh compat-libstdc++-33-3.2.3-55.fc5.i386.rpm
```

- 再次测试，应该成功了吧! ☺

(7) 验证 java 版本

```
[root@localhost ~]# java -version
```

```
java version "1.6.0_01"
Java(TM) SE Runtime Environment (build 1.6.0_01-b06)
Java HotSpot(TM) Client VM (build 1.6.0_01-b06, mixed mode, sharing)
```

2.2 nutch 的安装与配置

(1) 下载

```
http://lucene.apache.org/nutch
```

(2) 解压

```
[root@localhost~]#tar zxvf nutch-0.8.1.tar.gz
```

(3) 更改文件夹名称(方便以后执行)

```
[root@localhost~]#mv nutch-0.8.1 nutch
```

- mv 命令既可以移动文件文件夹，也可用于改名。

(1) 测试 nutch 命令

```
[root@localhost nutch]# bin/nutch
Usage: nutch COMMAND
where COMMAND is one of:
  crawl             one-step crawler for intranets
  readdb            read / dump crawl db
  mergedb           merge crawldb-s, with optional filtering
  readlinkdb        read / dump link db
  inject            inject new urls into the database
  generate          generate new segments to fetch
  fetch             fetch a segment's pages
  parse             parse a segment's pages
  readseg           read / dump segment data
  mergesegs         merge several segments, with optional filtering
and slicing
  updatedb          update crawl db from segments after fetching
  invertlinks       create a linkdb from parsed segments
  mergelinkdb       merge linkdb-s, with optional filtering
  index             run the indexer on parsed segments and linkdb
  merge             merge several segment indexes
  dedup             remove duplicates from a set of segment indexes
  plugin            load a plugin and run one of its classes main()
  server            run a search server
 or
  CLASSNAME         run the class named CLASSNAME
Most commands print help when invoked w/o parameters.
```

2.3 tomcat 的安装与配置

(1) 下载

```
http://tomcat.apache.org/
```

(2) 解压

```
[root@localhost~]#tar zxvf jakarta-tomcat-5.0.28.tar.gz
```

(3) 更改文件夹名称

```
[root@localhost~]#mv jakarta-tomcat-5.0.28.tar.gz tomcat
```

(4) 配置: 将 nutch 自带的 war 文件拷贝到 tomcat 的 webapps 文件夹下

```
[root@localhost~]#cd tomcat/webapps
[root@localhost~]#rm -rf ROOT*
[root@localhost webapps]#cp ~/nutch/nutch*.war ROOT.war
[root@localhost webapps]#jar xvf ROOT.war
```

(5) 启动 tomcat

```
[root@localhost webapps]#../bin/catalina.sh start
```

相应的,关闭 tomcat 的命令为

```
[root@localhost webapps]#../bin/catalina.sh stop
```

(2) 查看结果: 在浏览器中输入 http://localhost:8080 (远程查看需要将 localhost 换成相应的IP)

3. nutch 初体验

Nutch 的爬虫有两种方式

- 爬行企业内部网(Intranet crawling)。针对少数网站进行，用 crawl 命令。
- 爬行整个互联网。 使用低层的 inject, generate, fetch 和 updatedb 命令，具有更强的可控制性。

3.1 爬行企业内部网

3.1.1 配置 nutch

```
[root@localhost ~]#cd nutch
```

■ 增加要抓取的页面(以www.163.com为例)

```
[root@localhost nutch]#mkdir urls
[root@localhost nutch]#echo http://www.163.com/>>urls/163
```

■ 或者用vi编辑器也可以，在文件中输入http://www.163.com/，保存即可

■ 编辑 conf/crawl-urlfilter.txt 文件，设定要抓取的网址信息。

```
[root@localhost nutch]#vi conf/crawl-urlfilter.txt
```

修改 MY.DOMAIN.NAME 为:

```
# accept hosts in MY.DOMAIN.NAME
+^http://([a-z0-9]*\.)*163.com/
```

✧ 编辑 conf/nutch-site.xml 文件，增加代理的属性，并编辑相应的属性值

```
<property>
<name>http.agent.name</name>
<value></value>
<description>HTTP 'User-Agent' request header. MUST NOT be empty -
please set this to a single word uniquely related to your
organization.
NOTE: You should also check other related properties:
http.robots.agents
http.agent.description
http.agent.url
http.agent.email
http.agent.version
and set their values appropriately.
 </description>
</property>

<property>
<name>http.agent.description</name>
```

```
<value></value>
<description>Further description of our bot- this text is used in
the User-Agent header. It appears in parenthesis after the agent
name.
</description>
</property>
<property>
<name>http.agent.url</name>
<value></value>
<description>A URL to advertise in the User-Agent header. This will
appear in parenthesis after the agent name. Custom dictates that this
should be a URL of a page explaining the purpose and behavior of this
crawler.
</description>
</property>

<property>
<name>http.agent.email</name>
<value></value>
<description>An email address to advertise in the HTTP 'From' request
header and User-Agent header. A good practice is to mangle this
address (e.g. 'info at example dot com') to avoid spamming.
</description>
</property>
```

这里就算是不修改也无所谓，这里的设置，是因为 nutch 遵守了 robots 协议，在获取 response 时，把自己的相关信息提交给被爬行的网站，以供识别。

3.1.2 配置 tomcat

- 设定搜索目录

(是由于默认的 segment 路径与我们实际的路径不符所造成的)

```
[root@localhost nutch]#cd ~/tomcat
[root@localhost tomcat]#vi
 webapps/ROOT/WEB-INF/classes/nutch-site.xml
```

增加四行代码，修改成为

```
<configuration>
   <property>
          <name>searcher.dir</name>
          <value>/root/nutch/crawl.demo</value>
  </property>
</configuration>
```

value 的值指向 nutch 抓取的页面的保存目录，参见 3.1.3 节部分。

- 中文乱码

  nutch 对中文的支持还不完善，需要修改 tomcat 文件夹下

  conf/server.xml 文件

```
[root@localhost tomcat]#vi conf/server.xml
```

增加两句，修改为

```
<Connector port="8080"
   maxThreads="150" minSpareThreads="25"  maxSpareThreads="75"
   enableLookups="false"  redirectPort="8443"  acceptCount="100"
   connectionTimeout="20000"  disableUploadTimeout="true"
   URIEncoding="UTF-8"  useBodyEncodingForURI="true" />
```

(文件语句较多,只要找准 Connector port= " 8080 " 即可)

3.1.3 执行抓取命令

- 爬行抓取www.163.com网站

```
[root@localhost tomcat]#cd ~/nutch
[root@localhost nutch]#bin/nutch crawl urls -dir crawl.demo -depth
 2 -threads 4 -topN 50 >& crawl.log
```

  - urls 是存放 163 网址的文件夹目录
  - -dir crawl.demo 是抓取的页面的存放目录,与 3.1.2 中的设定搜索目录是对应的
  - -depth 指爬行的深度，这里处于测试的目的，选择深度为 2 ，完全爬行一般可设定为 10 左右
  - -threads 指定并发的进程 这是设定为 4
  - -topN 指在每层的深度上所要抓取的最大的页面数,完全抓取可设定为 1 万到 100 万，这取决于网站资源数量

- 抓取过程写入 crawl.log 中，可以查看如下

```
crawl started in: crawl.demo
rootUrlDir = urls
threads = 4
depth = 2
topN = 50
Injector: starting
Injector: crawlDb: crawl.demo/crawldb
Injector: urlDir: urls
Injector: Converting injected urls to crawl db entries.
Injector: Merging injected urls into crawl db.
Injector: done
```

```
Generator: starting
Generator: segment: crawl.demo/segments/20070424105443
Generator: Selecting best-scoring urls due for fetch.
Generator: Partitioning selected urls by host, for politeness.
Generator: done.
Fetcher: starting
Fetcher: segment: crawl.demo/segments/20070424105443
Fetcher: threads: 4
fetching http://www.163.com/
Fetcher: done
CrawlDb update: starting
CrawlDb update: db: crawl.demo/crawldb
CrawlDb update: segment: crawl.demo/segments/20070424105443
CrawlDb update: Merging segment data into db.
CrawlDb update: done
Generator: starting
Generator: segment: crawl.demo/segments/20070424105453
Generator: Selecting best-scoring urls due for fetch.
Generator: Partitioning selected urls by host, for politeness.
Generator: done.
Fetcher: starting
Fetcher: segment: crawl.demo/segments/20070424105453
Fetcher: threads: 4
fetching http://mail.163.com/
fetching http://www.163.com/Msxml2.XMLHTTP
fetching http://ecard.163.com/
fetching http://www.163.com/Microsoft.XMLHTTP
fetching http://lady.163.com/
fetching http://ent.163.com/
fetching http://dt.163.com/
fetching http://www.163.com/text/xml
……
.
fetch of http://www.163.com/inc/weatherxml/ failed with:
     java.net.SocketTimeoutException: connect timed out
Fetcher: done
CrawlDb update: starting
CrawlDb update: db: crawl.demo/crawldb
CrawlDb update: segment: crawl.demo/segments/20070424105453
CrawlDb update: Merging segment data into db.
CrawlDb update: done

LinkDb: starting
LinkDb: linkdb: crawl.demo/linkdb
```

```
LinkDb: adding segment: crawl.demo/segments/20070424105453
LinkDb: adding segment: crawl.demo/segments/20070424105443
LinkDb: done
Indexer: starting
Indexer: linkdb: crawl.demo/linkdb
Indexer: adding segment: crawl.demo/segments/20070424105453
Indexer: adding segment: crawl.demo/segments/20070424105443
 Indexing [http://2008.163.com/] with analyzer
     org.apache.nutch.analysis.NutchDocumentAnalyzer@12c8fa8 (null)
Indexing [http://auto.163.com/] with analyzer
     org.apache.nutch.analysis.NutchDocumentAnalyzer@12c8fa8 (null)
 Indexing [http://auto.163.com/product/] with analyzer
     org.apache.nutch.analysis.NutchDocumentAnalyzer@12c8fa8 (null)
 Indexing [http://bbs.163.com/] with analyzer
     org.apache.nutch.analysis.NutchDocumentAnalyzer@12c8fa8 (null)
 Indexing [http://biz.163.com/] with analyzer
     org.apache.nutch.analysis.NutchDocumentAnalyzer@12c8fa8 (null)
……..
 Indexing [http://www.163.com/inc/163new/bjto163.html] with analyzer
     org.apache.nutch.analysis.NutchDocumentAnalyzer@12c8fa8 (null)
Optimizing index.
merging segments _0 (1 docs) _1 (1 docs) _2 (1 docs) _3 (1 docs) _4 (1
     docs) _5 (1 docs) _6 (1 docs) _7 (1 docs) _8 (1 docs) _9 (1 docs)
     _a (1 docs) _b (1 docs) _c (1 docs) _d (1 docs) _e (1 docs) _f (1
     docs) _g (1 docs) _h (1 docs) _i (1 docs) _j (1 docs) _k (1 docs)
     _l (1 docs) _m (1 docs) _n (1 docs) _o (1 docs) _p (1 docs) _q (1
     docs) _r (1 docs) _s (1 docs) _t (1 docs) _u (1 docs) _v (1 docs)
     _w (1 docs) _x (1 docs) _y (1 docs) _z (1 docs) _10 (1 docs) _11
     (1 docs) _12 (1 docs) _13 (1 docs) _14 (1 docs) _15 (1 docs) _16
     (1 docs) into _17 (43 docs)
Indexer: done
Dedup: starting
Dedup: adding indexes in: crawl.demo/indexes
Dedup: done
Adding crawl.demo/indexes/part-00000
crawl finished: crawl.demo
```

- 搜索发生错误时分析一下日志是个不错的办法，从日志中可以查看抓取是否成功，得到错误的原因，从而进一步解决。

3.1.4 测试结果

- 启动 tomcat

```
[root@localhost tomcat]#bin/catalina.sh start
```

■ 启动浏览器，输入 http://localhost:8080

(1) 测试非中文搜索

(2) 测试中文搜索

3.1.5 Intranet Recrawl

在上面的介绍中，我们已经成功的进行了一次 nutch 抓取，并建立了一个索引库。但是在现实应用中，我们所建立的这个索引库必须要进行更新，加入新的内容。因此，我们就需要进行 recrawl，再次爬行。

recrawl并不是简单的再次crawl而已，它要跟以前的内容作比较，对于静态页面不更新，只更新动态的页面，因此更为复杂。好在Nutch Wiki上已经有了现成的代码，如下所示：

```
#!/bin/bash
# Nutch recrawl script.
# Based on 0.7.2 script at
     http://today.java.net/pub/a/today/2006/02/16/introduction-to-nu
     tch-2.html
#
# The script merges the new segments all into one segment to prevent
     redundant
# data. However, if your crawl/segments directory is becoming very large,
     I
# would suggest you delete it completely and generate a new crawl. This
     probaly
# needs to be done every 6 months.
#
# Modified by Matthew Holt
# mholt at elon dot edu

if [ -n "$1" ]
then
  tomcat_dir=$1
else
  echo "Usage: recrawl servlet_path crawl_dir depth adddays [topN]"
  echo "servlet_path - Path of the nutch servlet (full path, ie:
     /usr/local/tomc
at/webapps/ROOT)"
  echo "crawl_dir - Path of the directory the crawl is located in. (full
     path, i
e: /home/user/nutch/crawl)"
  echo "depth - The link depth from the root page that should be crawled."
  echo "adddays - Advance the clock # of days for fetchlist generation.
     [0 for n
one]"
  echo "[topN] - Optional: Selects the top # ranking URLS to be crawled."
  exit 1
fi

if [ -n "$2" ]
then
  crawl_dir=$2
else
  echo "Usage: recrawl servlet_path crawl_dir depth adddays [topN]"
  echo "servlet_path - Path of the nutch servlet (full path, ie:
     /usr/local/tomc
at/webapps/ROOT)"
```

```
  echo "crawl_dir - Path of the directory the crawl is located in. (full
     path, i
e: /home/user/nutch/crawl)"
  echo "depth - The link depth from the root page that should be crawled."
  echo "adddays - Advance the clock # of days for fetchlist generation.
     [0 for n
one]"
  echo "[topN] - Optional: Selects the top # ranking URLS to be crawled."
  exit 1
fi

if [ -n "$3" ]
then
  depth=$3
else
  echo "Usage: recrawl servlet_path crawl_dir depth adddays [topN]"
  echo "servlet_path - Path of the nutch servlet (full path, ie:
     /usr/local/tomc
at/webapps/ROOT)"
  echo "crawl_dir - Path of the directory the crawl is located in. (full
     path, i
e: /home/user/nutch/crawl)"
  echo "depth - The link depth from the root page that should be crawled."
  echo "adddays - Advance the clock # of days for fetchlist generation.
     [0 for n
one]"
  echo "[topN] - Optional: Selects the top # ranking URLS to be crawled."
  exit 1
fi

if [ -n "$4" ]
then
  adddays=$4
else
  echo "Usage: recrawl servlet_path crawl_dir depth adddays [topN]"
  echo "servlet_path - Path of the nutch servlet (full path, ie:
     /usr/local/tomcat/webapps/ROOT)"
  echo "crawl_dir - Path of the directory the crawl is located in. (full
     path, ie: /home/user/nutch/crawl)"
  echo "depth - The link depth from the root page that should be crawled."
  echo "adddays - Advance the clock # of days for fetchlist generation.
     [0 for n
one]"
  echo "[topN] - Optional: Selects the top # ranking URLS to be crawled."
```

```
  exit 1
fi
if [ -n "$5" ]
then
  topn="-topN $5"
else
  topn=""
fi
#Sets the path to bin
nutch_dir=`dirname $0`

# Only change if your crawl subdirectories are named something different
webdb_dir=$crawl_dir/crawldb
segments_dir=$crawl_dir/segments
linkdb_dir=$crawl_dir/linkdb
index_dir=$crawl_dir/index

# The generate/fetch/update cycle
for ((i=1; i <= depth ; i++))
do
  $nutch_dir/nutch generate $webdb_dir $segments_dir $topn -adddays
     $adddays
  segment=`ls -d $segments_dir/* | tail -1`
  $nutch_dir/nutch fetch $segment
  $nutch_dir/nutch updatedb $webdb_dir $segment
done

# Merge segments and cleanup unused segments
mergesegs_dir=$crawl_dir/mergesegs_dir
$nutch_dir/nutch mergesegs $mergesegs_dir -dir $segments_dir

for segment in `ls -d $segments_dir/* | tail -$depth`
do
  echo "Removing Temporary Segment: $segment"
  rm -rf $segment
done

cp -R $mergesegs_dir/* $segments_dir
rm -rf $mergesegs_dir

# Update segments
$nutch_dir/nutch invertlinks $linkdb_dir -dir $segments_dir

# Index segments
```

```
new_indexes=$crawl_dir/newindexes
segment=`ls -d $segments_dir/* | tail -1`
$nutch_dir/nutch index $new_indexes $webdb_dir $linkdb_dir $segment

# De-duplicate indexes
$nutch_dir/nutch dedup $new_indexes

# Merge indexes
$nutch_dir/nutch merge $index_dir $new_indexes
# Tell Tomcat to reload index
touch $tomcat_dir/WEB-INF/web.xml

# Clean up
rm -rf $new_indexes

echo "FINISHED: Recrawl completed. To conserve disk space, I would
     suggest"
echo " that the crawl directory be deleted once every 6 months (or more"
echo " frequent depending on disk constraints) and a new crawl generated."
```

一旦建好了，它的工作就很简单了，命令行中执行就可以了，建议在 crond（crond 可以在 linux 中定时执行任务）中定期的运行它，运行的时候一定要在命令行中输入全路径，不要使用相对路径。当然，你仔细看看这个脚本，就会知道为什么了。

这个脚本的执行参数如下：

```
recrawl servlet_path crawl_dir depth adddays [topN]
```

下面，我们就来执行执行一遍：

- 建立 recrawl 命令脚本

```
[root@localhost nutch]#vi bin/recrawl.sh
```

输入上面的代码脚本，保存并退出

- 执行 recrawl 命令

```
[root@localhost nutch]#bin/recrawl ~/tomcat/webapps/ROOT
 ~/nutch/crawl.demo 4 30
```

- OK,这样就完成了一次 recrawl！

3.2 爬行全网

全网爬行意指在整个互联网进行爬行，这通常需要多台服务器跑上几周的时间才能完成，这对于我们的测试而言是不必要的。因此，我们现在所做的所谓＂全网爬行＂，只是模拟一下。

3.2.1 nutch 数据集的基本组成:

- crawldb: 爬行数据库，用来存储所要爬行的网址
- linkdb: 链接数据库，用来存储每个网址的链接地址，包括源地址和链接地址
- segments: 抓取的网址被作为一个单元，而一个 segment 就是一个单元。一个 segment 包括以下几个子目录:
  - crawl_generate:包含所抓取的网址列表
  - crawl_fetch:包含每个抓取页面的状态
  - content:包含每个抓取页面的内容
  - parse_text:包含每个抓取页面的解析文本
  - parse_data:包含每个页面的外部链接和元数据
  - crawl_parse:包含网址的外部链接地址，用于更新 crawldb 数据库
- indexes: 采用 Lucene 的格式建立索引集

✧ 以上目录不需要自行建立，在执行命令时即可自行建立。

✧ 在 nutch0.7 版本中通过 admin 命令建立，在 nutch0.8 版本中取消了该命令。

3.2.2 爬行＂官方＂网址

(1) 建立爬行数据库

- 下载官方网址集

nutch文档中是从DMOZ 获取一个URL集合，然后取一个子集进行处理。不过该文档有 300 多M，对测试来说太大，所以我用了 http://rdf.dmoz.org/rdf/ 目录下的 content.example.txt 文件(包含 40 个网址)做测试。

点击该文件，选择”另存为…”保存在 nutch 文件夹下。

- 建立网址文件夹

```
[root@localhost nutch]#mkdir dmoz
```

- 转换网址格式(content.example.txt 文件是 xml 格式的，需转换为文本格式，文本中只包含网址)

```
[root@localhost nutch]#bin/nutch
 org.apache.nutch.tools.DmozParser content.example.txt >dmoz/urls
```

可以查看 dmoz/urls 中的内容为

```
http://www.britishhorrorfilms.co.uk/rillington.shtml
http://www.shoestring.org/mmi_revs/10-rillington-place.html
http://www.tvguide.com/movies/database/ShowMovie.asp?MI=22983
http://us.imdb.com/title/tt0066730/
http://www.geocities.com/aaronbcaldwell/1984.html
http://orwell.ru/a_life/movies/m84_01.htm
http://www.britmovie.co.uk/genres/fiction/filmography/014.html
http://adrianmco.batcave.net/1984.htm
……..
http://us.imdb.com/title/tt0114746/
http://www.geocities.com/darkdaze18/
http://apolloguide.com/mov_revtemp.asp?Title=13th+Warrior,+The
http://www.boxofficemojo.com/13thwarrior.html
http://movie-reviews.colossus.net/movies/t/13th_warrior.html
http://ter.air0day.com/13thwarrior.shtml
http://www.metacritic.com/video/titles/13thwarrior
http://us.imdb.com/title/tt0120657/
http://www.all-reviews.com/videos/thirteenth-warrior.htm
http://www.haro-online.com/movies/13th_warrior.html
http://www.rottentomatoes.com/movie-1091574/
http://upcomingmovies.com/13thwarrior.html
http://www.brunching.com/selfmade/selfmade-thirteenthwarrior.html
http://www.filmtracks.com/titles/13th_warrior.html
http://www.100girls.net/
http://imdb.com/title/tt0214388/
http://us.imdb.com/title/tt0146394/
http://us.imdb.com/title/tt0085121/
```

“注射”网址到 crawldb 数据库

```
[root@localhost nutch]#bin/nutch inject crawl/crawldb dmoz
```

- 现在，我们已经建立了一个爬行数据库，其中有 45 个网址

(2) 抓取页面

- 编辑conf/nutch-site.xml文件，增加代理的属性(详见3.1.1,第 7 页)
- 从数据库 crawldb 中创建抓取列表

```
[root@localhost nutch]#bin/nutch generate crawl/crawldb
 crawl/segments
```

■ 此时，segments 目录如下

```
[root@localhost nutch]#tree segments
segments
|-- 20070425173832
|   |-- crawl_generate
|   |   `-- part-00000
.....
```

■ 接下来我们把 segments 目录“20070425173832”保存到一个变量 s1 中，供以后使用。

```
[root@localhost nutch]#s1=`ls -d crawl/segments/2* | tail -1`
[root@localhost nutch]#echo $s1
```

✧ 注意，` 不是单引号 ’，而是左上角跟 ~ 一个键位的那个 `

■ 运行 fetcher，获取这些 URL 信息:

```
[root@localhost nutch]#bin/nutch fetch $s1
```

■ 更新数据库，把获取到的页面信息存入数据库中:

```
[root@localhost nutch]#bin/nutch updatedb crawl/crawldb $s1
```

■ 这样，我们就完成了一次抓取，接下来，我们选择分值排在前 10 的 URL(一个很小的子集)来进行第二次和第三次抓取

```
[root@localhost nutch]# bin/nutch generate crawl/crawldb
 crawl/segments -topN 10
[root@localhost nutch]#s2=`ls -d crawl/segments/2* | tail -1`
[root@localhost nutch]#echo $s2
[root@localhost nutch]#bin/nutch fetch $s2
[root@localhost nutch]#bin/nutch updatedb crawl/crawldb $s2

[root@localhost nutch]#bin/nutch generate crawl/crawldb
 crawl/segments -topN 10
[root@localhost nutch]#s3=`ls -d crawl/segments/2* | tail -1`
[root@localhost nutch]#echo $s3
[root@localhost nutch]#bin/nutch fetch $s3
[root@localhost nutch]#bin/nutch updatedb crawl/crawldb $s3
```

(3) 建立索引

■ 根据 sengments 的内容更新 linkdb 数据库

```
[root@localhost nutch]# bin/nutch invertlinks crawl/linkdb
 crawl/segments/*
```

- 建立索引

```
[root@localhost nutch]# bin/nutch index crawl/indexes crawl/crawldb
 crawl/linkdb crawl/segments/*
```

  - indexes:用于存放索引
  - crawldb，linkdb，segments：索引数据库源

(4) 测试搜索

- 简单校验

```
[root@localhost nutch]#bin/nutch
 org.apache.nutch.searcher.NutchBean apache
```

  - 即调用 NutchBean 的 main 方法搜索关键字“apache”。

- 启动 tomcat

```
 [root@localhost nutch]#cd ~/tomcat
[root@localhost tomcat]#bin/catalina.sh start
```

- 启动浏览器，输入 http://localhost:8080

测试结果如下

3.2.3 爬行中文网址

(1) 建立爬行数据库

- 首先清除刚才的网址和爬行结果，否则那些网站又得抓取一遍

```
[root@localhost nutch]#rm -rf crawl
[root@localhost nutch]#rm -rf dmoz/urls
```

- 新建 urls 地址

```
[root@localhost nutch]#vi dmoz/urls
```

我选择了 5 个门户网站来做一下简单测试

```
http://www.163.com/
http://www.sina.com/
http://www.sohu.com/
http://www.qq.com/
http://www.tom.com/
```

- “注射”网址到 crawldb 数据库

```
[root@localhost nutch]#bin/nutch inject crawl/crawldb dmoz
```

- 现在，我们已经建立了一个爬行数据库，其中有 5 个网址

(2) 后面的步骤与3.2.2 爬行＂官方＂网址相同，不在叙述。

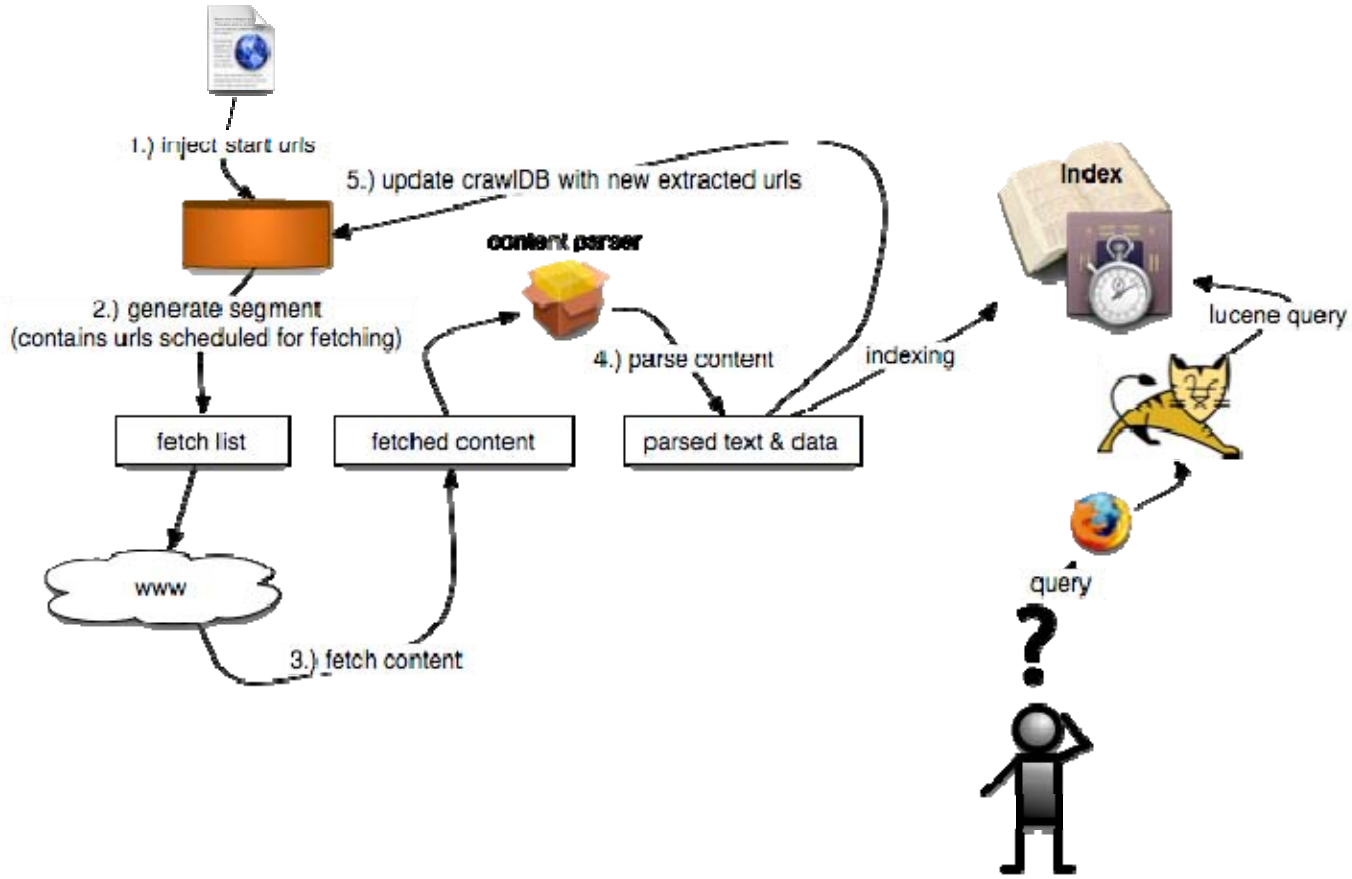
(3) 测试结果

4. nutch 基本原理分析

4.1 nutch 的基本组成

nutch 作为一个搜索引擎，其基本组成也同其他搜索引擎一样。简单的说，包括爬虫，索引和搜索三部分。

4.2 nutch 工作流程



在前面的 nutch 初体验中，我们就是按照 nutch 的工作流程来进行的，现在总结如下：

1) 建立初始 URL 集

2) 将 URL 集注入 crawldb 数据库---inject

3) 根据 crawldb 数据库创建抓取列表---generate

4) 执行抓取，获取网页信息---fetch

5) 更新数据库，把获取到的页面信息存入数据库中---updatedb

6) 重复进行 3～5 的步骤，直到预先设定的抓取深度。---这个循环过程被称为“产生/抓取/更新”循环

7) 根据 sengments 的内容更新 linkdb 数据库---invertlinks

8) 建立索引---index

9) 用户通过用户接口进行查询操作

10) 将用户查询转化为 lucene 查询

11) 返回结果

其中，1~6 属于爬虫部分；7、8 属于索引部分；9~11 属于查询部分。

(注: 进行内部网爬行时执行的 crawl 操作，实质上也是执行的以上一系列操作，这一点从它的抓取日志上可以看出)

5. nutch 工作流程分析

5.1 爬虫

5.1.1 工作策略

其工作策略一般则可以分为累积式抓取（cumulative crawling）和增量式抓取（incremental crawling）两种。

累积式抓取是指从某一个时间点开始，通过遍历的方式抓取系统所能允许存储和处理的所有网页。在理想的软硬件环境下，经过足够的运行时间，累积式抓取的策略可以保证抓取到相当规模的网页集合。但由于 Web 数据的动态特性，集合中网页的被抓取时间点是不同的，页面被更新的情况也不同，因此累积式抓取到的网页集合事实上并无法与真实环境中的网络数据保持一致。

与累积式抓取不同，增量式抓取是指在具有一定量规模的网络页面集合的基础上，采用更新数据的方式选取已有集合中的过时网页进行抓取，以保证所抓取到的数据与真实网络数据足够接近。进行增量式抓取的前提是，系统已经抓取了足够数量的网络页面，并具有这些页面被抓取的时间信息。

面向实际应用环境的网络蜘蛛设计中，通常既包括累积式抓取，也包括增量式抓取的策略。累积式抓取一般用于数据集合的整体建立或大规模更新阶段；而增量式抓取则主要针对数据集合的日常维护与即时更新。

在确定了抓取策略之后，如何从充分利用网络带宽，合理确定网页数据更新的时间点就成了网络蜘蛛运行策略中的核心问题。

总体而言，在合理利用软硬件资源进行针对网络数据的即时抓取方面，已经形成了相对比较成熟的技术和实用性解决方案，这方面目前所需解决的主要问题，是如何更好的处理动态网络数据问题（如数量越来越庞大的 Web2.0 数据等），以及更好的根据网页质量修正抓取策略的问题。

5.1.2 工作流程分析

1) 建立初始 URL 集

- 初始 URL 集的建立有两种方式：超链接和站长提交
  - 超链接：机器人程序根据网页链到其他网页中的超链接，就像日常生活中所说的“一传十，十传百……”一样，从少数几个网页开始，

连到数据库上所有到其他网页的链接。理论上，若网页上有适当的超连结，机器人便可以遍历绝大部分网页。

- 站长提交：在实际运行中，爬虫不可能抓取到所有站点，为此，网站站长可以向搜索引擎进行提交，要求收录，搜索引擎经过核查之后，便将该网站加入到 URL 集合中，进行抓取。

2) inject

- inject 操作调用的是 nutch 的核心包之一 crawl 包中的类 injector。
- inject 操作主要作用
  - 将 URL 集合进行格式化和过滤，消除其中的非法 URL，并设定 URL 状态(UNFETCHED),按照一定方法进行初始化分值；
  - 将 URL 进行合并，消除重复的 URL 入口；
  - 将 URL 及其状态、分值存入 crawldb 数据库，与原数据库中重复的则删除旧的，更换新的。
- inject 操作结果：crawldb 数据库内容得到更新，包括 URL 及其状态。

3) generate

- generate 操作调用的是 crawl 包中的类 generator。
- generate 操作主要作用
  - 从 crawldb 数据库中将 URL 取出并进行过滤
  - 对 URL 进行排序，通过域名、链接数和一种 hash 算法综合进行降序排列
  - 将排列列表写入 segment
- generate 操作结果：创建了抓取列表，存放于 segment 文件夹下，以时间为文件夹名称。循环抓取多少次，segment 文件夹下就会有多少个以时间为名称的文件夹。

4) fetch

- fetch 操作调用的是 fetcher 包中的类 fercher。
- fetch 操作主要作用
  - 执行抓取，按照 segment 文件夹下的抓取列表进行

- 抓取过程中，页面的 URL 地址可能因为链接发生改变，从而需要更新 URL 地址
- 抓取采用多线程方式进行，以提高抓取速度
- fetch 操作过程中调用了 parse 操作

■ fetch 操作结果：将页面内容抓取下来，存于 segment 目录下

5) parse

■ parse 操作调用的是 parse 包中的类 parsesegment。

■ parse 操作主要作用

- 解析 segment 中由 fetch 得到的页面，并进行整理，将页面分成为 parse-date 和 parse-text
- parse-date 中保存的是页面的题名、作者、日期、链接等内容
- parse-text 中保存的是页面的文本内容

■ parse 操作结果：将 fetch 得到的页面解析为 text 和 data，存于 segment 目录下

6) updatedb

■ updatedb 操作调用的是 crawl 包中的类 crawldb

■ updatedb 操作主要作用

- 根据 segment 目录下 fetch 文件夹和 parse 文件夹中的内容，对 crawldb 进行更新，增加新的 URL，更换旧的 URL

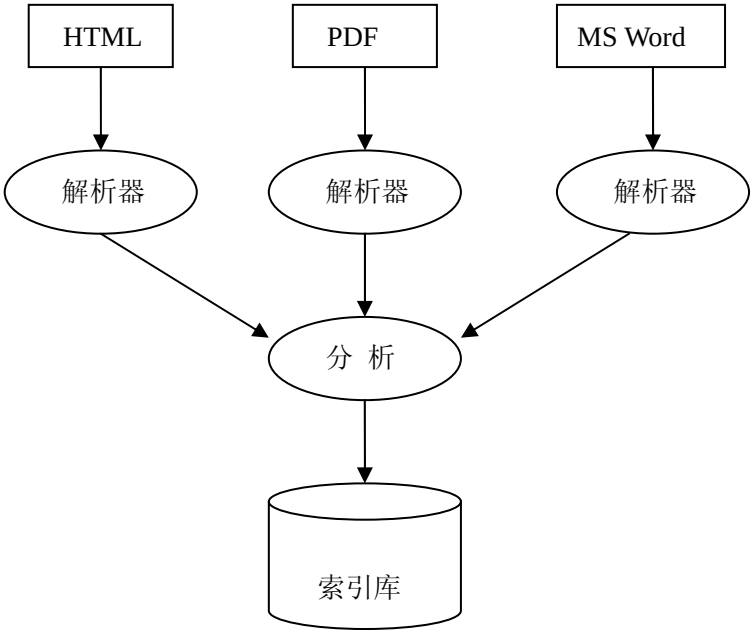
■ updatedb 操作结果：更新了 crawldb 数据库，为下一轮抓取做准备

5.1.3 其它

爬虫部分还有其它一些子操作，但在第一次进行 Crawler 运行时可能并不用到，它们主要用于接下来的索引更新，增量搜索等操作的实现。在此，我就不再详述。

5.2 索引

5.2.1 索引主要过程



索引过程可分为三个主要的操作阶段：

- 将数据转换成文本
- 分析文本
- 将分析过的文本保存到数据库中

5.2.2 工作流程分析

invertlinks 操作用来更新 linkdb 数据库，为索引做准备，不是主要操作，在此不再详述；而 index 操作包括了以上 3 个主要操作，故直接介绍这三个操作。

- 转换成文本

  在索引数据之前，首先必须将数据转换成 nutch 能够处理的格式－－纯文本字符流。但是，在现实世界中，信息多以富媒体(rich media)文档格式呈现：PDF,WORD,EXCEL,HTML,XML 等。为此，nutch 采用了插件机制(plugin)，通过各种各样的文档解析器，将富媒体转换成纯文字字符流。文档解析器种类繁多，开发人员可以根据需要进行选择，同时还可以自己修改或编写，非常灵活方便。

- 分析

  在对数据进行索引前，还需要进行预处理，对数据进行分析使之更加适合被索引。分析数据时，先将文本数据切分成一些大块或者语汇单元(tokens)，然后对它们执行一些可选的操作，例如：在索引之前将这些语

汇单元转换成小写，使得搜索对大小写不敏感；最有代表性的是要从输入中去掉一些使用很频繁但却没有实际意义的词，比如英文文本中的一些停止词(a、an、the、in、on 等)。同样的，我们也需要分析输入的语汇单元，以便从词语中去掉一些不必要的字母以找到它们的词干。这一处理过程称为分析(analyze)。分析技术在索引和搜索时都会用到，比较重要，所以我会在 5.4 节中进行详细讲述。

- 将分析后的数据写入索引

对输入数据分析处理完之后，就可以将结果写入到索引文件中。nutch 采用的是 lucene 的索引格式，故我将介绍一下这种称为“倒排索引”的索引结构。

5.2.3 倒排索引(inverted index)

Lucene 将输入数据以一种称为倒排索引的数据结构进行存储。倒排索引源于实际应用中需要根据属性的值来查找记录。这种索引表中的每一项都包括一个属性值和具有该属性值的各记录的地址。由于不是由记录来确定属性值，而是由属性值来确定记录的位置，因而称为倒排索引。在进行关键字快速查找时，这种数据结构能够有效的利用磁盘空间。Lucene 会使用倒排结构存储数据的原因是：把从文档中抽取出来的语汇单元看作是查找关键字，而不是把文档作为中心实体。换句话说，倒排索引不是回答“这个文档中包含哪些单词？”这个问题，而是经过优化以后用来快速回答“哪些文档包含单词 X？”这个问题。

5.2.4 其它

索引是搜索引擎的核心，一个好的索引是搜索引擎成功的关键。在此，我仅描述了索引的工作流程。而索引还包括其它一些重要的部分，如索引的方法，索引的排序，索引的控制和优化，以及并发性、线程安全性和锁机制等，有兴趣的朋友可以参考相关资料。

5.3 搜索

Nutch 的查询体系架构非常简单，它的工作流程只需要以下几个步骤就可以描述出来。

- HTTP 服务器接收用户发送过来的请求。对应到 nutch 的运行代码中就是一个 servlet，称为查询处理器(Query Handler)。查询处理器负责响应用户的请求，并将相应的 HTML 结果页面返回给用户。
- 查询处理器对查询语句做一些微小的处理并将搜索的项(terms)转发到一组运行索引搜索器的机器上。Nutch 的查询系统似乎比 lucene 简单的多，这主要是因为搜索引擎的用户对他自己所要执行的查询内容有非常清晰的思路。然而，lucene 的系统结构非常灵活，且提供了多种不同的查询方式。看似简单的 nutch 查询最终被转换为特定的 lucene 查询类型。每个索引搜索器并行工作且返回一组有序的文档 ID 列表。
- 现在存在这大量从查询处理器返回过来的搜索结果数据流。查询处理器对这些结果集进行比较，从所有的结果查找出匹配最好的那些。如果其中任何一个索引搜索器在 1~2 秒之后返回结果失败，该搜索器的结果将被忽略。因此，最后列表由操作成功的搜索器返回的结果组成。

查询处理器对查询作了一些细微的处理，例如删除停止词(例如 the,of 等)。接着 nutch 需要执行一些操作以便于它在大规模的数据环境下能更好的工作。一个索引搜索器涉及搜索的文档集数目非常庞大，所以 nutch 需要同时与很多索引搜索器交互来提高搜索速率。实际运行环境中，为了保证系统级别的稳定性，文档集的段文件会被复制到多个不同主机上。因为对于文档集中的每个段，查询处理器会随机的与其中一个可搜索到自身的索引搜索器相交互。如果发现一个索引搜索器不能交互，查询处理器会通知之后的搜索操作不使用该搜索器，但是查询处理器每隔一段时间会回头检查一次搜索器的状态，以防该主机上的搜索器再次可用。

5.4 分析

- 分析，在这里指的是将域(Field)文本转换为最基本的索引表示单元---项(Term)的过程。分析器对分析操作进行了封装。分析器通过执行若干操作，将文本语汇单元化，这些操作可能包括提取单词、去除标点符号、去掉语汇单元上的音调符号、将字母转化为小写(也称为规格化)、移除常用词、将单词转换为词干形式(词干还原)，或者将单词转换为基本形等。这个过程也称为语汇单元化过程。

- 分析操作发生在两个阶段：建立索引和进行查询时。
- Nutch 使用自己的分析器，对应于 analysis 包。Nutch 把索引时进行分析所使用的二元语法技术(bigram)与查询过程中对短语的优化技术结合在一起，通过二元语法技术可以把两个连续的词组合成一个语汇单元。这就可以大大减少搜索时需要考虑的文档范围，例如，包含词组 the quick 的文档比包含 the 的文档要少的多。

5.5 nutch 的其他一些特性

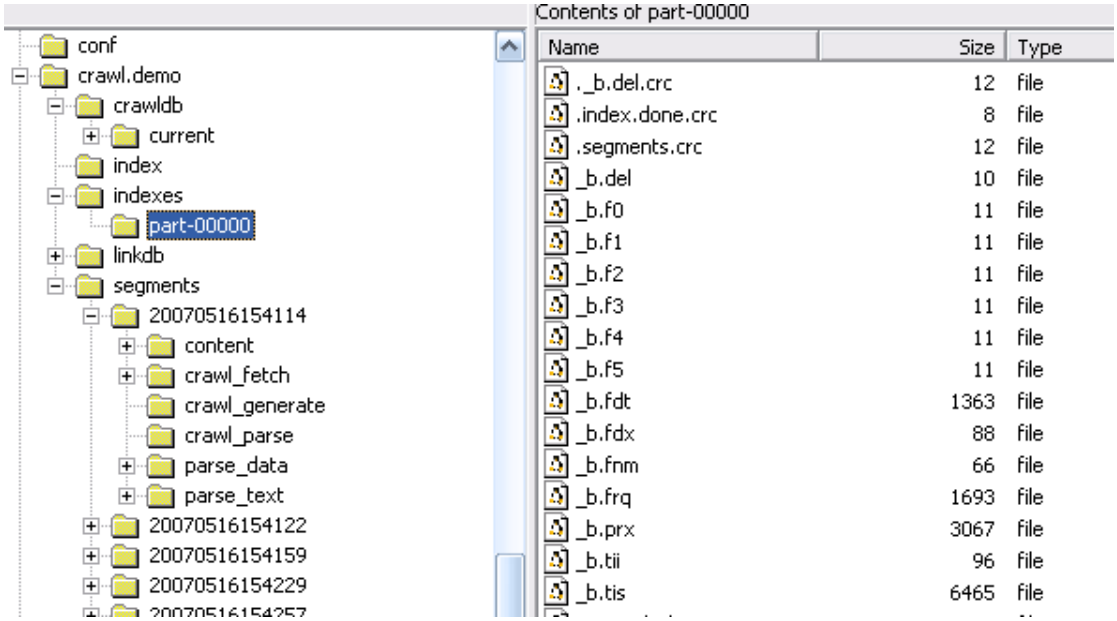
- 为了获取小数量的文档(通常是 10 个左右)，查询处理器会对每个索引搜索器进行查询。因为最后的结果是从多个索引搜索器中合并得到的，所以就没有必要从一个数据源中获取过多的文档结果，尤其是在用户很少去查看第一页之后的结果的情况下。
- 实际上，在每个用户查询被处理之前，它会被扩展为十分复杂的 lucene 查询。每个索引过的文档都包含了三个域：网页自身的内容，网页的 URL 文本值，以及由所有关键(anchor)文本所组成的合成文档，这些关键文本可在导航网页的超链接中找到。每个域对应一个不同的权重值。Nutch 的查询处理器生成一个 lucene 布尔查询，其中在三个域中都包含了搜索引擎用户所输入的文本。
- Nutch 也会特别的把那些在 web 上出现的非常频繁的关键字组作为一个整体来索引(其中的许多关键字是与 HTTP 相关的词组)。这些关键字序列出现的非常频繁，所以无需花费精力去对这些词序中的每个组成部分单独搜索，也不必查找出这些搜索结果中是否有交集的部分。我们不用把这些项划分为单独的单词对来搜索文档，而是把它们作为一个单元，当然前提是 nutch 在索引期间必须检测到它们是作为一个整体而存在的。另外，在与索引搜索器交互之前，查询处理器会查找出用户输入的字符串中单词的任意组合。如果这样一个词序确实存在，它的单词成员就会整合成一个特殊的搜索项。
- 在使用 lucene 执行索引操作之前，nutch 的内容获取器/索引器需要预处理 HTML 文档。它使用 NekoHTML 解析器剥离 HTML 中的内容，并索

引其中的非标记文本。对于从 HTML 文档提取标题文本，NekoHTML 是很有建树的。

- Nutch 进程间通信层(IPC)保持了查询处理器与索引搜索器间长时间的连接。查询处理器端可能存在多个并发的线程，对于给定的地址，任一线程都可以向远程服务器发送调用请求。服务器每接受一个请求之后，就会根据给定字符串尝试去查找对应的注册服务(运行在自己线程上)。客户端的请求线程会一直阻塞其他事件，直到服务器端响应的 IPC 代码到达后，消息通知请求线程为止。如果服务器的响应花费的时间超过了 IPC 规定的时限，IPC 的代码就会宣布此服务器不可用，并抛出一个异常。
- 另外，nutch 的排序算法是仿照 google 的 PageRank 算法，关于 PageRank 算法的资料有很多，推荐《Google 的秘密 PageRank 彻底解说中文版》，附录二中有链接，想要进一步了解的朋友可以去看一下，在此不在叙述。

6. nutch 分析方法和工具

当我们进行完 nutch 的工作流程之后，我们希望知道到底都抓取了什么内容。打开我们设定的存储文件夹(crawl.demo)，可以看到其中的目录如下：



(注：此图是在 windows 下用 Explore2fs 软件打开，该软件可在 windows 下访问 linux 分区)

但是，我们并不能直接打开查看这些文件。不过，幸运的是 nutch 和 lucene 给我们提供了一些方法和工具来查看其中的储存内容，下面我就按照目录来介绍一下这些方法和工具。

6.1 Crawldb

在前面nutch的安装与配置一章中，我们测试了nutch命令。其中有一些命令我们可以用来研究文件内容。

首先，我们来看一下 Crawldb 数据库中。

```
[root@localhost nutch]#bin/nutch readdb crawl.demo/crawldb/ -stats
```

输入上面的命令行，我们可以得到如下结果：

```
CrawlDb statistics start : crawl.demo/crawldb/
Statistics for CrawlDb : crawl.demo/crawldb/
TOTAL urls:     28
retry 0:          26
retry 3:          2
min score:        0.0010
avg score:        0.077321425
max score:        1.0
```

```
status 2 (DB_fetched):  11
status 3 (DB_gone):    17
CrawlDb statistics:   done
```

从这个结果中，我们可以看到，在 CrawlDb 数据库中，共有 28 个 url 地址，以及这些 url 地址的状态和评分。

当然，我们还可以进一步的了解每个 url 地址的详细内容，只要改变 readdb 后面的参数，我们就可以将这些内容导出到文件中。(这里以新浪网为例)

```
[root@localhost nutch]#bin/nutch readdb crawl.demo/crawldb/ -dump
crawldb
```

执行该命令后，就会在当前目录下生成一个新的文件夹 crawldb，打开里面 part-00000 文件，可以看到每个 url 地址的详细信息。

```
http://263.sina.com/cgi-bin/mail/login.cgi    Version: 4
Status: 2 (DB_fetched)
Fetch time: Fri Jun 15 15:41:26 CST 2007
Modified time: Thu Jan 01 08:00:00 CST 1970
Retries since fetch: 0
Retry interval: 30.0 days
Score: 0.06666667
Signature: aa747a3ececa29a4d3a08ed881e5a01f
Metadata: null

http://263.sina.com/cgi-bin/mail/mail.cgi     Version: 4
Status: 2 (DB_fetched)
Fetch time: Fri Jun 15 15:41:25 CST 2007
Modified time: Thu Jan 01 08:00:00 CST 1970
Retries since fetch: 0
Retry interval: 30.0 days
Score: 0.06666667
Signature: c6373c36d8248063cb0acc3bfc277bb6
Metadata: null
……
```

另外，我们也可以直接输入 url 地址，在命令行下进行查看。

```
[root@localhost nutch]#bin/nutch readdb crawl.demo/crawldb/ -url
http://www.sina.com/
```

我们将会看到如下信息：

```
URL: http://www.sina.com/
Version: 4
Status: 2 (DB_fetched)
Fetch time: Fri Jun 15 15:41:17 CST 2007
```

```
    Modified time: Thu Jan 01 08:00:00 CST 1970
    Retries since fetch: 0
    Retry interval: 30.0 days
    Score: 1.0
    Signature: 9aa9536508ad95367e82d817ffdbf47d
    Metadata: null
```

只要更换后面的网址，我们就可以看到每个 url 地址的内容。

6.2 Linkdb

Nutch-0.8.1 版本将原来 webdb 拆成了 crawldb 和 linkdb，相应的也为 linkdb 准备了查看命令 readlinkdb。

```
    [root@localhost nutch]#bin/nutch readlinkdb crawl.demo/linkdb/
     -url http:vip.sina.com/
```

我们将看到以下结果：

```
    fromUrl: http://vip.sina.com/update/vip.html anchor: 首页
```

可见，linkdb 中保存的是网页的链接情况。

同样，我们可以把所有的网页链接信息导出到文件。

```
    [root@localhost nutch]#bin/nutch readlinkdb crawl.demo/linkdb/
     -dump linkdb
```

打开当前目录下的 linkdb 文件夹中的 part-00000 文件，可以看到如下内容。

```
http:/          Inlinks:
 fromUrl: http://vip.sina.com/ anchor: http://vip.sina.com/

http://202.108.35.247/d29.html Inlinks:
 fromUrl: http://vip.sina.com/ anchor: http://vip.sina.com/

http://263.sina.com/cgi-bin/mail/login.cgi    Inlinks:
 fromUrl: http://www.sina.com/ anchor: http://www.sina.com/
.....................................................................
     ............
```

6.3 Segments

我们知道，爬虫产生的一个 segment 就是一个 generate/fetch/update 循环，抓取的网页内容就保存在其中。我们可以通过 readseg 命令来查看它。

```
    [root@localhost nutch]#bin/nutch readseg -list -dir
     crawl.demo/segments/
```

我们将看到以下结果：

```
NAME    GENRATED    FETCHER START    FETCHER END
FETCHED  PARSED
20070516154114   1   2007-05-16T15:41:17  2007-05-16T15:41:17
```

```
1         1
20070516154122  13   2007-05-16T15:41:25  2007-05-16T15:41:56
13        2
……….
```

从上面的结果我们可以看到每一个 segment 的名称，产生的页面数，抓取开始的时间和结束的时间，抓取数和解析数。可见，抓取的页面未必会解析。

接下来，我们将 segments 中的内容导出到文件。

```
[root@localhost nutch]#bin/nutch readseg -dump
 crawl.demo/segments/20070516154114 segdb
```

这样会产生一个 dump 文件，这是一次抓取循环的结果，如果要查看其他文件夹下抓取的结果，只要更改最后那个以时间命名的文件夹就可以了。

dump 文件中的内容大致如下所示。

```
Recno:: 0
URL:: http://263.sina.com/cgi-bin/mail/login.cgi

CrawlDatum::
Version: 4
Status: 4 (linked)
Fetch time: Wed May 16 15:41:19 CST 2007
Modified time: Thu Jan 01 08:00:00 CST 1970
Retries since fetch: 0
Retry interval: 30.0 days
Score: 0.06666667
Signature: null
Metadata: null


Recno:: 1
URL:: http://263.sina.com/cgi-bin/mail/mail.cgi

CrawlDatum::
Version: 4
Status: 4 (linked)
Fetch time: Wed May 16 15:41:19 CST 2007
Modified time: Thu Jan 01 08:00:00 CST 1970
Retries since fetch: 0
Retry interval: 30.0 days
Score: 0.06666667
Signature: null
Metadata: null
```

```
Recno:: 2
URL:: http://www.sina.com/

CrawlDatum::
Version: 4
Status: 1 (DB_unfetched)
Fetch time: Wed May 16 15:41:10 CST 2007
Modified time: Thu Jan 01 08:00:00 CST 1970
Retries since fetch: 0
Retry interval: 30.0 days
Score: 1.0
Signature: null
Metadata: null

CrawlDatum::
Version: 4
Status: 5 (fetch_success)
Fetch time: Wed May 16 15:41:17 CST 2007
Modified time: Thu Jan 01 08:00:00 CST 1970
Retries since fetch: 0
Retry interval: 30.0 days
Score: 1.0
Signature: 9aa9536508ad95367e82d817ffdbf47d
Metadata: null

CrawlDatum::
Version: 4
Status: 0 (signature)
Fetch time: Wed May 16 15:41:19 CST 2007
Modified time: Thu Jan 01 08:00:00 CST 1970
Retries since fetch: 0
Retry interval: 0.0 days
Score: 1.0
Signature: 9aa9536508ad95367e82d817ffdbf47d
Metadata: null

Content::
url: http://www.sina.com/
base: http://www.sina.com/
contentType: text/html
metadata: X-Powered-By=mod_xlayout_jh/0.0.1vhs.markII.remix
     ETag="c6876f-3e147-6d00c200" nutch.crawl.score=1.0
     nutch.content.digest=9aa9536508ad95367e82d817ffdbf47d
```

```
     Content-Type=text/html date=Wed, 16 May 2007 07:41:26 GMT
     Cache-Control=max-age=60 Content-Encoding=gzip Connection=close
     Expires=Wed, 16 May 2007 07:42:26 GMT
     nutch.segment.name=20070516154114 Accept-Ranges=bytes
     Server=Apache/2.0.54 (Unix) X-Cache=MISS from 152-65.sina.com.cn
     Last-Modified=Wed, 16 May 2007 07:37:44 GMT Vary=Accept-Encoding
Content:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!--[30,50,1] published at 2007-05-16 15:37:43 from #152 by 50-->
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>新浪首页</title>
<meta name="stencil" content
……
ParseData::
Status: success(1,0)
Title: 新浪首页
Outlinks: 100
  outlink: toUrl: http://www.sina.com/document.all. anchor:
     http://www.sina.com/
  outlink: toUrl:
     http://www.sina.com/webbug_meta_ref_mod_noiframe_:7.06 anchor:
     http://www.sina.com/
  outlink: toUrl: http://beacon.sina.com.cn/a.gif anchor:
     http://www.sina.com/
……
ParseText::
新浪首页 注册 通行证 登录名 密码 免费邮箱 VIP 邮箱 同名邮箱 U 币 会员中心 忘记密码
     免费邮箱 VIP 邮箱 企业邮箱 设为首页 网站地图 首页 | 新闻 | 财经 | 娱乐 | 宽
     频 | 女性 | 房产 | 旅游 | 游戏 | 军事 | 论????坛 | 企业 | 短信 | 交????
     友 爱问 | 体育 | 科技 | 音乐 | T??V | 育儿 | 家居 | 法治 | 星座 | 上海 |
     圈????子 | 城市 | | 图 ???? 铃 邮箱 | 博客 | 手机 | 乐库 | 理财 | 饮食 |
     汽车 | 名品 | 教育 | 广东 | 高尔夫 | 黄页 | 彩信 | 音悦汇 天气 | 播客 | 数
     码 | 读书 | 家电 | 健康 | F 1 | 商城 | U C | 搜索 | iGame | 分类 | 聊天
     | 电????台 热销 上海 广东 家居 北二环 独栋写字楼 The House 国际花园 高
…….
```

可以看到，segment 文件内容包括 CrawlDatum、Content、ParseData、ParseText 四个部分。

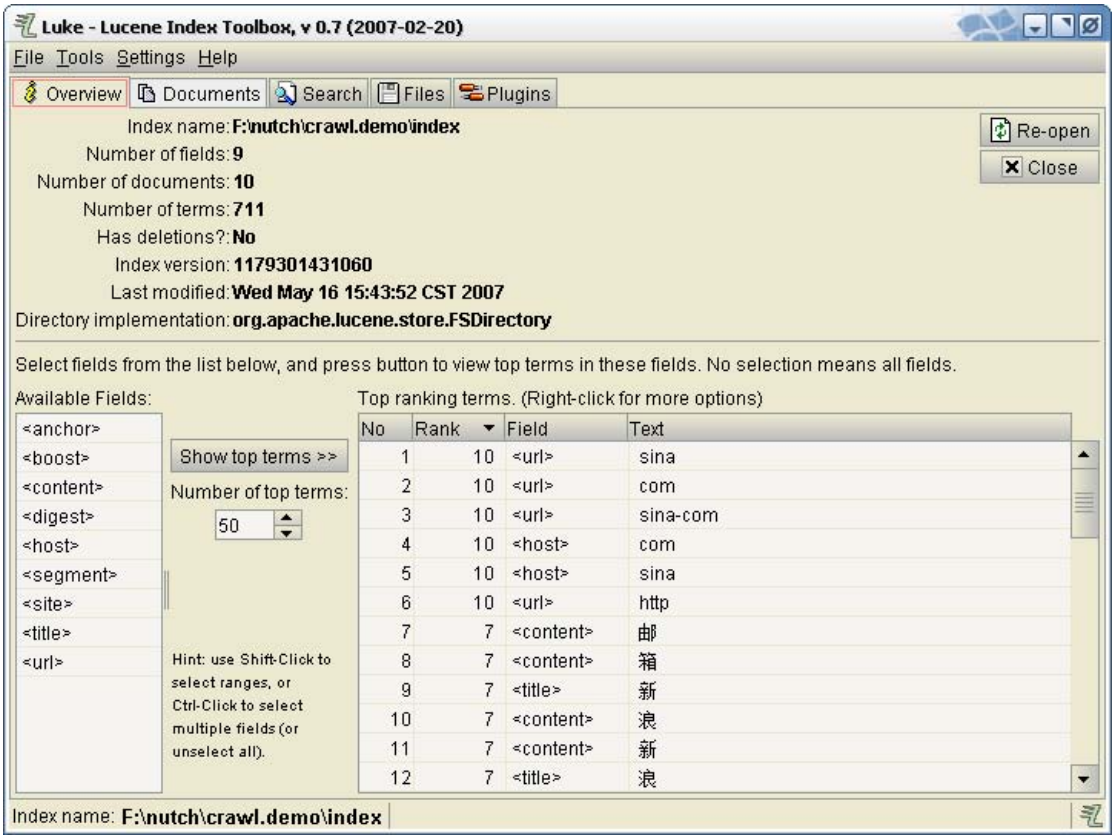
- CrawlDatum：保存的是抓取的基本信息，相当于查看 crawldb 数据库时所得到的信息，对应于 generate/fetch/update 循环中的 update 环节。
- Content：保存的是 fetcher 所抓取回来的源内容，在这里，我们可以看到是 html 脚本(默认是由 protocol-httpclient 插件来进行处理)，我们可以直接查看网页进行对比。
- ParseData 和 ParseText：这两个部分可以成为解析内容。通过使用合适的解析器插件（在这里是 parse-html），将源内容进行解析，用于 indexer 产生对应的索引。

6.4 Index

Luke 是用来查看 lucene 格式索引的工具。而 nutch 的索引采用的就是 lucene 格式，所以，luke 就成为查看 nutch 索引的利器。

Luke下载地址：http://www.getopt.org/luke/

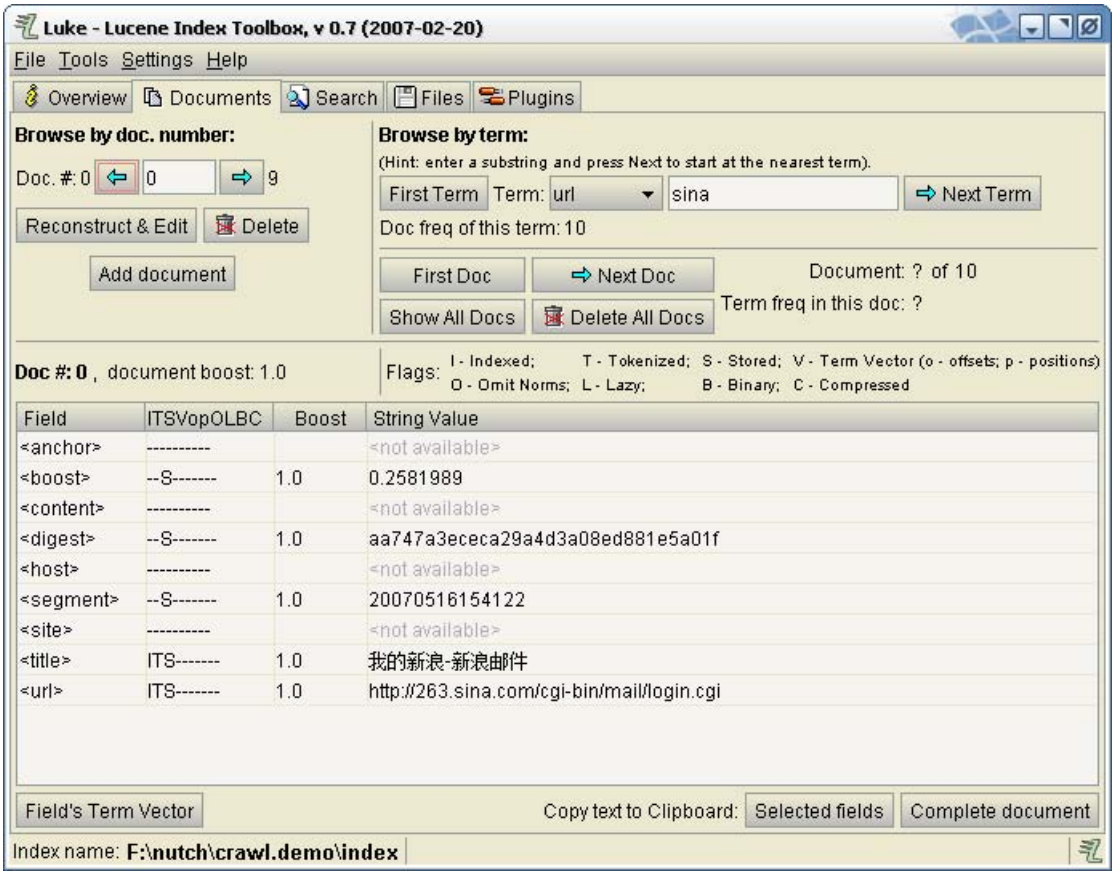
下载最新版本，运行，选择索引文件夹(crawl.demo 文件夹下的 index 文件夹),界面如下图


Luke - Lucene Index Toolbox, v 0.7 (2007-02-20)
File Tools Settings Help
Overview Documents Search Files Plugins
Index name: F:\nutch\crawl.demo\index
Number of fields: 9
Number of documents: 10
Number of terms: 711
Has deletions?: No
Index version: 1179301431060
Last modified: Wed May 16 15:43:52 CST 2007
Directory implementation: org.apache.lucene.store.FSDirectory
Re-open
Close
Select fields from the list below, and press button to view top terms in these fields. No selection means all fields.
Available Fields:
<anchor>
<boost>
<content>
<digest>
<host>

<site>
<title>
<url>
Show top terms >>
Number of top terms:
50
Hint: use Shift-Click to select ranges, or Ctrl-Click to select multiple fields (or unselect all).
Top ranking terms. (Right-click for more options)
No Rank Field Text
1 10 <url> sina
2 10 <url> com
3 10 <url> sina-com
4 10 <host> com
5 10 <host> sina
6 10 <url> http
7 7 <content> 邮
8 7 <content> 箱
9 7 <title> 新
10 7 <content> 浪
11 7 <content> 新
12 7 <title> 浪
Index name: F:\nutch\crawl.demo\index

第一个标签是索引的概况，从中我们可以看到 filed，document，term 等情况。

第二个标签是 documents，我们可以详细的看到每一个 document 的情况。


Luke - Lucene Index Toolbox, v 0.7 (2007-02-20)
File Tools Settings Help
Overview Documents Search Files Plugins
Browse by doc. number:
Doc. #: 0 0 9
Reconstruct & Edit Delete
Add document
Browse by term:
(Hint: enter a substring and press Next to start at the nearest term).
First Term Term: url sina Next Term
Doc freq of this term: 10
First Doc Next Doc
Document: ? of 10
Show All Docs Delete All Docs
Term freq in this doc: ?
Doc #: 0 , document boost: 1.0
Flags: I - Indexed; T - Tokenized; S - Stored; V - Term Vector (o - offsets; p - positions)
O - Omit Norms; L - Lazy; B - Binary; C - Compressed
Field ITSVopOLBC Boost String Value
<anchor> ---------- <not available>
<boost> --S------- 1.0 0.2581989
<content> ---------- <not available>
<digest> --S------- 1.0 aa747a3ececa29a4d3a08ed881e5a01f
<host> ---------- <not available>
 --S------- 1.0 20070516154122
<site> ---------- <not available>
<title> ITS------- 1.0 我的新浪-新浪邮件
<url> ITS------- 1.0 http://263.sina.com/cgi-bin/mail/login.cgi
Field's Term Vector
Copy text to Clipboard: Selected fields Complete document
Index name: F:\nutch\crawl.demo\index

在这里，我们可以更加详细的看到该索引的内容。这里的大多数内容都是解释性的，只有 boost 域例外，我们来关注一下。boost 域的值是以链接入页面的链接数为基本所计算出来的：越多页面链接了该页，该页的 boost 值就越高。不过，boost 值并不是与入链数成线性关系，而是对数关系。基本公式为：ln(e+n)，n 是入链数。当然，boost 的值还与其他一些因素相关。当搜索为全网搜索时，这些因素的作用就更加凸显。本质上，nutch 的评分标准是模拟 google 的 PageRank 技术(PageRank 技术是 google 取得成功的重要法宝之一，有兴趣的朋友可以找些相关资料看看，推荐一本《Google 的秘密 PageRank 彻底解说中文版》，在此就不再讲述了，否则又是一篇文章了)。

关于 luke 还有很多方面，大家可以自己去尝试一下。

7. nutch 分布式文件系统

7.1 概述

Nutch 抓取下内容之后就要开始对文件进行管理。Nutch 分布式文件系统基础架构是 Hadoop 文件系统(正式的讲就是 NDFS)。NDFS 与 Google 的文件系统 GFS 是类似的，在一系列机器上存储庞大的面向流的文件，包含多机的存储冗余和负载均衡。

在这个文件系统中，文件以块为单位存储在 NDFS 的离散机器上，提供一个传统的 input/output 流接口用于文件读写。块的查找以及数据在网络上传输等细节由 NDFS 自动完成，对用户是透明的。而且 NDFS 能很好地处理用于存储的机器序列，能方便地添加和删除一台机器。当某台机器不可用时，NDFS 自动的保证文件的可用性。只要网上的机器序列能提供足够的存储空间，就要保证 NDFS 文件系统的正常运作。NDFS 是建立在普通磁盘上的，不需要 RAID 控制器或者其它的磁盘阵列解决方案。

7.2 MapReduce

Hadoop 是 Google MapReduce 的一个 Java 实现。MapReduce 是一种简化的分布式编程模式，让程序自动分布到一个由普通机器组成的超大集群上并发执行。就如同 java 程序员可以不考虑内存泄露一样， MapReduce 的 run-time 系统会解决输入数据的分布细节，跨越机器集群的程序执行调度，处理机器的失效，并且管理机器之间的通讯请求。这样的模式允许程序员可以不需要有什么并发处理或者分布式系统的经验，就可以处理超大的分布式系统得资源。MapReduce 是 Nutch 中的分布计算层。它主要分为 map 和 Reduce 两步。输入和输出数据是包含一系列键-值对的文件。

在 map 阶段，输入数据被分割成由各个节点处理的连续的块。用户使用的 map 功能将每个原始数据处理成中间数据集。通过该分割功能，每个中间数据被发送到 Reduce 节点。分割是一个典型的 hash 过程，所有等键值的中间数据都将被发送到一个 Reduce 节点。例如，如果一个 map 操作输出以 URL 为键值的数据，那么通过哈希值的分割就会发送中间数据以及给定的 URL 到一个 Reduce 节点。

Reduce 节点将所有的输入数据进行排序，并使用一个提供给用户的 Reduce 函数输出已经排序的 map，产生 MapReduce 过程的最终输出。所有带有给定键值的输入都将交由 Reduce 处理，即以 URL 为健值的所有数据和 URL 一起被传递给 Reduce 函数，并可能用于产生最终输出。

7.3 文件系统语法

1. 文件只能写一次，写完之后，就变成只读了（但是可以被删除）

2. 文件是面向流的，只能在文件末尾加字节流，而且只能读写指针只能递增。

3. 文件没有存储访问的控制

所以，所有对 NDFS 的访问都是通过验证的客户代码。没有提供 API 供其它程序访问。因此 Nutch 就是 NDFS 的模拟用户。

7.4 文件系统设计

NDFS 包含两种类型的机器：NameNodes 和 DataNodes。NameNodes 维护名字空间；而 DataNodes 存储数据块。NDFS 中包含一个 NamdNode，而包含任意多的 DataNodes，每个 DataNodes 都配置与唯一的 NameNode 通信。

1. NameNode：负责存储整个名字空间和文件系统的布局。是一个关键点，不能 down 掉。但是做的工作不多，因此不是一个负载的瓶颈。

维护一张保存在磁盘上的表：

filename-0->BlockID_A,BlockID_B...BlockID_X,etc.;

filename 就是一字符串，BolockID 是唯一的标识符。每个 filename 有任意个 blocks。

2. DataNode:负责存储数据。一个块应该在多个 DataNode 中有备份；而一个 DataNode 对于一个块最多只包含一个备份。

维护一张表：BlockID_X->array of bytes.

3. 合作：DataNode 在启动后，就主动与 NameNode 通信，将本地的 Block 信息告知 NameNode。NameNode 据此可以构造一颗树，描述如何找到 NDFS 中的 Blocks。这颗树是实时更新的。DataNode 会定期发送信息给 NameNode，以证明自己的存在，当 NameNode 收不到该信息时，就会认为 DataNode 已经 down 了。

4. 文件的读写过程：例如 Client 要读取 foo.txt，则有以下过程。

1) Client 通过网络联系 NameNode，提交 filename:"foo.txt"

2) Client 收到从 NameNode 来的回复，包含：组成"foo.txt"的文件块和每个块存在的 DataNode 序列。

3) Client 依次读取每个文件块。对于一个文件块，Client 从它的 DataNode 序列中得到合适的 DataNode，然后发送请求给 DataNode，由 DataNode 将数据传输给 Client。

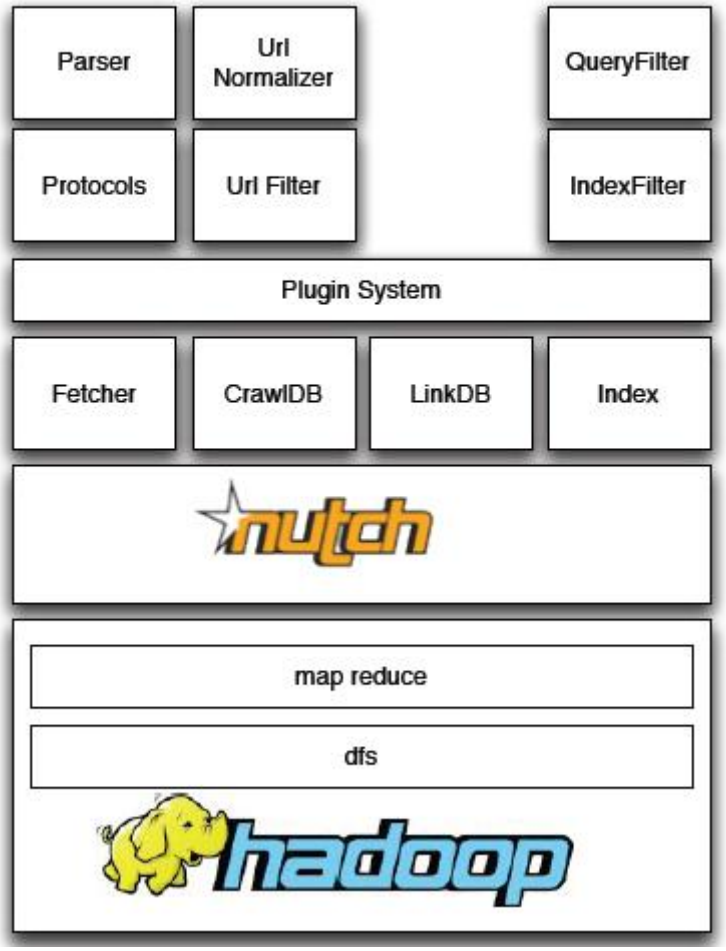
7.5 系统的可用性

NDFS 的可用性取决于 Blocks 的冗余度，即应该在多少个 DataNode 保持同一 Block 的备份。对于有条件的话可以设置 3 个备份和 2 个最低备份 (DESIRED_REPLICATION and MIN_REPLICATION constants in fs.FSNamesystem)。当一个块的低于 MIN_REPLICATION, NameNode 就会指导 DataNode 做新的备份。

7.6 Nutch 文件系统工作架构

通过上面的分析，我们可以得出 Nutch 文件系统在整个工作架构中的位置如下：

Architecture



8. nutch 应用

nutch 的应用目前主要有三种方式：

- 修改源码
- 插件机制
- API 接口

8.1 修改源码

Nutch 的源码很容易修改和重新编译。Nutch 的源码很多，但是结构相当清晰，使用 java 编程工具可方便的对源码进行修改。通过 ANT 工具对源码进行编译，生成 nutch-0.8.1.job（实际是一个 jar 包)就可以了。

8.2 插件机制---plugin

8.2.1 什么是 plugin

nutch 的 plugin 系统是基于 Eclipse 2.x 中对插件的使用。plugin 对 nutch 的工作是很重要的。所有的 nutch 中的 parsing（分析），indexing（索引），searching（查询）都是通过不同的 plugin 来实现的。

8.2.2 使用 plugin 的好处

1) 可扩展性

通过 plugin，nutch 允许任何人扩展它的功能，而我们要做的只是对给定的接口做简单的实现，举个例子：MSWordParser 这个插件是用来分析 wordwendang 的，它就是一个对 parser 这个接口的实现

2) 灵活性

因为每个人都可以根据自己的需求而写自己的 plugin，这样 plugin 就会有一个很强大的资源库。这样对与应用 nutch 程序员来说，他可以在自己的搜索引擎上安装符合自己需求的插件，而这些插件就在 nutch 的 plugins 中。这对于正在应用 nutch 的开发者来说应该是一个巨大的福音，因为你有了更多的关于内容抽取的算法来选择，很容易就增加了 pdf 的分析。

3) 可维护性

每个开发者只要关注自己的问题。对于内核的开发者在为引擎内核扩展的同时，为 a plug 添加一个描述它的接口就可以了。一个 plugin 的开发者只要关注这个 plugin 所要实现的功能，而不需要知道整个系统是怎么工作的。它们仅

仅需要知道的是 plugin 和 plug 之间交换的数据类型。这使得内核更加简单，更容易维护

8.2.3 plugin 工作原理

在编写一个plugin的时候，你要为一个扩展点添加一个或者更多的扩展项。这些Nutch的扩展点是Nutch在一个plugin中已经定义好了，这个plugin是NutchExtensionPoints（所有的扩展点都会在NutchExtensionPoints plugin.xml这个文件中列出）。每一个扩展点都定义了一个接口，这个接口在扩展时必须被实现。这些扩展点如下：

- onlineClusterer－为在线的查询结果提供分组算法的扩展点的接口
- indexingFiltering－允许为所索引中的 Field 添加元数据。所有的实现了这个接口 plugin 会在分析的过程中顺序的逐个运行
- Ontology
- Parser

实现这个接口的 parser 读取所抓取的 document，摘取将被索引的数据。如果你要在 nutch 中为扩展分析一个新内容类型或者从现有的可分析的内容摘取更多的数据。

- HtmlParseFilter

  为 html parser 添加额外的元数据

- protocol

  实现 Protocol 的 plugin 可以使得 nutch 能使用更多的网络协议（ftp，http）去抓取数据

- QueryFilter

  为查询转换的扩展点

- URLFileter

  实现这个扩展点的 plugin 会对 nutch 要抓取的网页的 urls 进行限制，RegexURLFilter 提供了通过正则表达式来对 Nutch 爬行网页的 urls 的控制。如果你对 urls 还有更加复杂的控制要求，你可以编写对这个 urlfilter 的实现

- NutchAnalyser

  为许多语言特定的分析器提供了扩展点

- 源文件

在 plugin 的源文件目录中你会找到以下的文件

plugin.xml　向 nutch 描述这个 plugin 的信息

build.xml　告诉 ant 怎样编译这个 plugin

plugin 的源码

- 在 Nutch 使用 plugin

如果要在 Nutch 使用一个给定的 plugin，你需要对 conf/nutch-site.xml 进行编辑并且把 plugin 的名字添加到 plugin.includes 中

8.2.4 编写 plugin

■ 编写一个 pluginExample

思考编写这样的一个 plugin：我们想为一个给定的 search term 推荐一些与之相关的网页。举个例子，假设我们正在索引网页，当我们注意到有一组网页的内容是关于 plugin 的，所以我们想如果当某人查询 plugin 的时候，我们可以推荐他到 pluginCentral 这个网页，但是同时，也要返回在一般逻辑中的查询结果所有的 hits。所以我们将查询结果分成了推荐结果和一般查询结果。

你浏览你的网页然后把一个 meta－tags 加入网页中，它会告诉 plugin 这个网页是应该被推荐的。这个 tags 应该像这样

<meta name="recommended" content="plugins" />

为了达到目标我们需要写一个 plugin，这个 plugin 扩展 3 个不同的扩展点，它们是：

- ✓ HTMLParser：从 meta－tags 得到推荐的 terms
- ✓ IndexingFilter：增加一个推荐 Field 在索引中。
- ✓ QueryFilter：增加对索引中新 Field 的查询能力

● 要建立的文件

首先在 plugin 的目录中新建一个目录来盛放自己的的 plugin，整个 plugin 我们取名为 recommended，然后在这个目录里面依次建立以下文件:

- ✓ a plugin.xml,这个文件用来向 Nutch 描述我们新建的 plugin
- ✓ a build.xml 这个文件告诉编译器应该怎样 build 这个 plugin

plugin 的源代码则保存在/src/java/org/apache/nutch/parse/recommended/下

● Plugin.xml

你所建立的 plugin.xml 应该这样：

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
   id="recommended"
   name="Recommended Parser/Filter"
   version="0.0.1"
   provider-name="nutch.org">

   <runtime>
      <!-- As defined in build.xml this plugin will end up bundled as
recommended.jar -->
      <library name="recommended.jar">
         <export name="*"/>
      </library>
   </runtime>

   <!-- The RecommendedParser extends the HtmlParseFilter to grab the
contents of
              any recommended meta tags -->
   <extension
id="org.apache.nutch.parse.recommended.recommendedfilter"
            name="Recommended Parser"
            point="org.apache.nutch.parse.HtmlParseFilter">
      <implementation id="RecommendedParser"

class="org.apache.nutch.parse.recommended.RecommendedParser"/>
   </extension>

   <!-- TheRecommendedIndexer extends the IndexingFilter in order to add
the contents
              of the recommended meta tags (as found by the
RecommendedParser) to the lucene
              index. -->
   <extension
id="org.apache.nutch.parse.recommended.recommendedindexer"
            name="Recommended identifier filter"
            point="org.apache.nutch.indexer.IndexingFilter">
      <implementation id="RecommendedIndexer"

class="org.apache.nutch.parse.recommended.RecommendedIndexer"/>
   </extension>

   <!-- The RecommendedQueryFilter gets called when you perform a search.
It runs a
```

```
              search for the user's query against the recommended fields.
In order to get
          add this to the list of filters that gets run by default, you
have to use
              "fields=DEFAULT". -->
   <extension
id="org.apache.nutch.parse.recommended.recommendedSearcher"
            name="Recommended Search Query Filter"
            point="org.apache.nutch.searcher.QueryFilter">
     <implementation id="RecommendedQueryFilter"

class="org.apache.nutch.parse.recommended.RecommendedQueryFilter"
                  fields="DEFAULT"/>
   </extension>

</plugin>

Build.xml
<?xml version="1.0"?>

<project name="recommended" default="jar">

  <import file="../build-plugin.xml"/>

</project>
```

- The HTML Parser Extension

这是对 HtmlParserExtension 这个扩展点的实现，它的作用是抓取那些标 meta－tags 的内容，这样把它们加入正在分析的 document 中。.

```
package org.apache.nutch.parse.recommended;

// JDK imports
import java.util.Enumeration;
import java.util.Properties;
import java.util.logging.Logger;

// Nutch imports
import org.apache.nutch.parse.HTMLMetaTags;
import org.apache.nutch.parse.Parse;
import org.apache.nutch.parse.HtmlParseFilter;
import org.apache.nutch.protocol.Content;
import org.apache.nutch.util.LogFormatter;

public class RecommendedParser implements HtmlParseFilter {
```

```
  private static final Logger LOG = LogFormatter
    .getLogger(RecommendedParser.class.getName());

  /** The Recommended meta data attribute name */
  public static final String META_RECOMMENDED_NAME="Recommended";

  /**
   * 在 html 文件中寻找有没有标有 meta－tags 的内容   */
  public Parse filter(Content content, Parse parse, HTMLMetaTags
metaTags, DocumentFragment doc) {
    // Trying to find the document's recommended term
        String recommendation = null;

        Properties generalMetaTags = metaTags.getGeneralTags();

        for (Enumeration tagNames = generalMetaTags.propertyNames();
tagNames.hasMoreElements(); ) {
                    if (tagNames.nextElement().equals("recommended"))
{
                            recommendation =
generalMetaTags.getProperty("recommended");
                            LOG.info("Found a Recommendation for " +
recommendation);
                     }
        }

        if (recommendation == null) {
                    LOG.info("No Recommendataion");
        } else {
                    LOG.info("Adding Recommendation for " +
recommendation);

parse.getData().getMetadata().put(META_RECOMMENDED_NAME,
recommendation);
        }

    return parse;
  }
}
```

- The Indexer Extension

这是对索引点的扩展，如果查找到带有 meta－tags 的内容，就把它命名"recommended'的 field 中，然后加入 document 建立索引

```
package org.apache.nutch.parse.recommended;

// JDK import
import java.util.logging.Logger;

// Nutch imports
import org.apache.nutch.util.LogFormatter;
import org.apache.nutch.fetcher.FetcherOutput;
import org.apache.nutch.indexer.IndexingFilter;
import org.apache.nutch.indexer.IndexingException;
import org.apache.nutch.parse.Parse;

// Lucene imports
import org.apache.lucene.document.Field;
import org.apache.lucene.document.Document;

public class RecommendedIndexer implements IndexingFilter {
  public static final Logger LOG
    = LogFormatter.getLogger(RecommendedIndexer.class.getName());

  public RecommendedIndexer() {
  }

  public Document filter(Document doc, Parse parse, FetcherOutput fo)
    throws IndexingException {

    String recommendation = parse.getData().get("Recommended");

       if (recommendation != null) {
                     Field recommendedField = Field.Text("recommended",
recommendation);
                     recommendedField.setBoost(5.0f);
              doc.add(recommendedField);
                     LOG.info("Added " + recommendation + " to the
recommended Field");
       }

    return doc;
  }
}
```

- The QueryFilter

当用户进行查找操作的时候，QueryFilter 就会被调用，而且对于 recommeded 中的 boost 值会影响查询结果的排序。

```
package org.apache.nutch.parse.recommended;

import org.apache.nutch.searcher.FieldQueryFilter;

import java.util.logging.Logger;

import org.apache.nutch.util.LogFormatter;

public class RecommendedQueryFilter extends FieldQueryFilter {

       private static final Logger LOG = LogFormatter
    .getLogger(RecommendedParser.class.getName());

  public RecommendedQueryFilter() {
    super("recommended", 5f);

       LOG.info("Added a recommended query");
  }

}
```

- **让 Nutch 可以使用你的 plugin**

为了让 Nutch 使用你的 plugin，你需要对 conf/nuthc-site.xml 这个文件进行编辑，把以下的代码加入

```
<property>
<name>plugin.includes</name>

<value>nutch-extensionpoints|protocol-http|urlfilter-regex|parse-(te
xt|html)|index-basic|query-(basic|site|url)|recommended
</value>
  <description>Regular expression naming plugin directory names to
  include.  Any plugin not matching this expression is excluded.
  In any case you need at least include the nutch-extensionpoints plugin.
By
  default Nutch includes crawling just HTML and plain text via HTTP,
  and basic indexing and search plugins.
  </description>
</property>
```

使用 Ant 对你的 plugin 进行编译

在之前我们要编辑一下 src/plugin/build.xml 这个文件，这是对编译和部署做一些设置

你会看到有很多如下形式的行

```
<ant dir="[plugin-name]" target="deploy" />
```

在</target>前添加一新行

```
<ant dir="reccomended" target="deploy" />
Running 'ant' in the root of your checkout directory should get everything
compiled and jared up. The next time you run a crawl your parser and index
filter should get used.
You'll need to run 'ant war' to compile a new ROOT.war file. Once you've
deployed that, your query filter should get used when searches are
performed.
```

8.3 API 接口

通过调用 API 接口，将 nutch 集成到自己的产品中，这是 nutch 的一种重要应用方式。集成的方法有很多，这里我们来看一下使用 nutch API 接口和使用 OpenSearch API 接口的方式。

8.3.1 使用 Nutch API

如果应用程序是用 java 编写的，则可以很方便的使用 nutch API 将 nutch 集成。

我们先来看一个例子：

```
package org.tiling.nutch.intro;

import java.io.IOException;

import org.apache.nutch.searcher.Hit;
import org.apache.nutch.searcher.HitDetails;
import org.apache.nutch.searcher.Hits;
import org.apache.nutch.searcher.NutchBean;
import org.apache.nutch.searcher.Query;

public class SearchApp {

  private static final int NUM_HITS = 10;

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

    if (args.length == 0) {
      String usage = "Usage: SearchApp query";
      System.err.println(usage);
      System.exit(-1);
    }
```

```
    NutchBean bean = new NutchBean();
    Query query = Query.parse(args[0]);
    Hits hits = bean.search(query, NUM_HITS);

    for (int i = 0; i < hits.getLength(); i++) {
      Hit hit = hits.getHit(i);
      HitDetails details = bean.getDetails(hit);

      String title = details.getValue("title");
      String url = details.getValue("url");
      String summary =
        bean.getSummary(details, query);

      System.out.print(title);
      System.out.print(" (");
      System.out.print(url);
      System.out.println(")");
      System.out.println("\t" + summary);
    }

  }

}
```

这是个简单例子，通过调用 nutch 实现了一个简单查询。

在这个例子中，我们可以看到，核心是 NutchBean 类。虽然只有短短几行，但 NutchBean 可以替我们做很多工作。NutchBean 的主要功能有：

- 一是按 Title field 来排序
- 二是支持分布式查询，如果有配置servers，就会使用hadoop的IPC系统，调用所有server上的nutchBeans，最后规约出总的结果。
- 三是每个站点只显示分数最高的一页，如果用户还想看同站的其他结果，就需要访问 MoreHitsExculde[]。
- 四是生成 Summary，读取 segments 目录，按 segments 和 url 获得 content，并按一定算法抽取出包含关键字的文档片断。

现在我们来分析一下上面的程序。

- 首先，我们需要先接收一个查询，并将它转换为查询(Query)对象，这个过程由 Query.parse()方法完成。这里有一点要注意，尽管在索引、分析、

搜索等方面，Nutch 在很大程度上都依赖 Lucene，但是，在许多核心方法上，虽然两者的方法名可能相同，但实际的执行方法却可能不同。例如，org.apche.lucene.search.Query 和 org.apche.nutch.searcher.Query 虽然都是 Query，但两者是不兼容的。

- 接下来，我们将查询对象传给 bean 来进行搜索。这时候，bean 就将 Nutch Query 转换为 Lucene Query，进行 Lucene 查询。Hits 对象用来返回最匹配的查询结果集，只包含索引和文档识别。如果要返回更多的信息，我们通过 bean.getHitDetails 方法获得我们所感兴趣的内容。
- 最后，将查询结果显示为一个简短的 HTML 摘要，这是通过 bean 的 getSummary 方法得到的。

可以看到，NutchBean的引入大大简化了nutch的使用方法，如果想了解更多内容，可以参考API documentation。

8.3.2 使用 OpenSearch API

OpenSearch API 是 RSS2.0 的扩展搜索工具接口。Nutch 支持 OpenSearch 1.0 ，当应用程序不是用 java 编写的时候，可以使用 OpenSearch API。

我们来看下面的例子，这是一个 XML 脚本：

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
  xmlns:nutch="http://www.nutch.org/opensearchrss/1.0/"
  xmlns:opensearch="http://a9.com/-/spec/opensearchrss/1.0/">

  <channel>
    <title>Nutch: animals</title>
    <description>Nutch search results for query: animals</description>

<link>http://localhost/search.jsp?query=animals&amp;start=0&amp;hits
PerDup=2&amp;hitsPerPage=10</link>

    <opensearch:totalResults>2</opensearch:totalResults>
    <opensearch:startIndex>0</opensearch:startIndex>
    <opensearch:itemsPerPage>10</opensearch:itemsPerPage>

    <nutch:query>animals</nutch:query>

    <item>
      <title>'A' is for Alligator</title>
```

```
     <description>&lt;b&gt; ... &lt;/b&gt;Alligators'
     main prey are smaller &lt;b&gt;animals&lt;/b&gt;
     that they can kill and&lt;b&gt; ... &lt;/b&gt;</description>
     <link>http://keaton/tinysite/A.html</link>

     <nutch:site>keaton</nutch:site>

<nutch:cache>http://localhost/cached.jsp?idx=0&amp;id=0</nutch:cache
>

<nutch:explain>http://localhost/explain.jsp?idx=0&amp;id=0&amp;query
=animals</nutch:explain>
     <nutch:docNo>0</nutch:docNo>
     <nutch:segment>20051025121334</nutch:segment>
     <nutch:digest>fb8b9f0792e449cda72a9670b4ce833a</nutch:digest>
     <nutch:boost>1.3132616</nutch:boost>
   </item>

   <item>
     <title>'C' is for Cow</title>
     <description>&lt;b&gt; ... &lt;/b&gt;leather
     and as draught &lt;b&gt;animals&lt;/b&gt;
     (pulling carts, plows and&lt;b&gt; ... &lt;/b&gt;</description>
     <link>http://keaton/tinysite/C.html</link>

     <nutch:site>keaton</nutch:site>

<nutch:cache>http://localhost/cached.jsp?idx=0&amp;id=2</nutch:cache
>

<nutch:explain>http://localhost/explain.jsp?idx=0&amp;id=2&amp;query
=animals</nutch:explain>
     <nutch:docNo>1</nutch:docNo>
     <nutch:segment>20051025121339</nutch:segment>
     <nutch:digest>be7e0a5c7ad9d98dd3a518838afd5276</nutch:digest>
     <nutch:boost>1.3132616</nutch:boost>
   </item>

  </channel>
</rss>
```

这里既用到了 opensearch，又用到了 nutch，两者相互结合，可以实现更多功能。

8.4 nutch 的应用前景

Nutch 正在提供一个全球的自由搜索引擎。作为一个开源的项目，它具有透明性、易扩展性等特点，能够吸引广大的搜索引擎爱好者参与其中，群策群力，更好的推动 nutch 的发展。但是，另一方面，nutch 作为一个年轻的搜索引擎，在面对 google，yahoo 等搜索引擎巨头时还显得过于稚嫩，缺乏强有力的团队协作和资金支持，以及商业推广，目前其应用范围还很狭小。Nutch 要想出人头地，还需要一番努力。

道路是曲折的，前途是光明的！

附录一： nutch 的相关网站

- nutch官方网站: http://lucene.apache.org/nutch
- nutch维基: http://wiki.apache.org/nutch
- nutch中文网: http://www.nutchchina.com/
- 《Nutch相关的资料》

附录二： 参考文献

- 《nutch-tutorial》
- 《nutch中文网 - Nutch介绍》
- 《安装和配置 Java J2SE Development Kit（JDK）》
- 《为 Linux 手动下载和安装 Java Runtime Environment (JRE) 的说明》
- 《Nutch 初体验》
- 《Nutch 笔记（一）：Quick Start 》
- 《关于 Nutch 的一个问题_ 中文乱码》
- 《Nutch 初体验之二 》
- 《FC4 Linux 基础教程－－黑魔方系列》 出版社：清华大学出版社
  作者：林慧琛等 （注：图书馆图书有限，未抢到 FC5 的书）
- 《Introduction to Nutch, Part 2: Searching》
- 《Introduction to Nutch, Part 1: Crawling》
- 《IntranetRecrawl－Nutch Wiki》
- 《Google的秘密PageRank彻底解说中文版》
- 《lucene in action中文版》 出版社：电子工业出版社
  作者：(美)Otis Gospodnetic;Erik Hatcher 译者：谭鸿，黎俊鸿等