Nutch1.8\_solr4.8的配置过程参考

<http://blog.csdn.net/hmbig2/article/details/22572473>

nutch +eclipse配置过程

NUTCH 2.0 抓取流程简单分析：

1. 整体流程

Inject：从文件中得到一批种子网页，把他们放到抓取数据库中去

Generator：从抓取数据库中产生要抓取的页面放到抓取队列中去

Fetcher：对抓取队列中的网页进行抓取，在reducer中使用生产者/消费者模型

Paser：对抓取完成的网页进行解析，产生一些新的链接与网页内容解析结果

Dbupdate：把新产生的链接更新到抓取的数据库中去

SolrIndexer：对解析后的内容进行索引的建立。

有兴趣可以看一下SolrWriter，它实现了NutchIndexerWriter这个接口，来把数据写到不同的后台搜索引擎中，这里默认使用了Solr，当然你也可以通过实现它来扩展你自己的搜索引擎，当然nutch还提供了插件来自定义索引的字段值,也就是IndexingFilter.java这个接口。

Nutch 的工作流程：inject操作分析

Inject

Nutch的输出文件主要可分为crawldb，index，indexs，linklb和segmets

其中crawldb是所有需要抓取的超链接信息。

Linkdb中存放的是所有超链接机器每个连接的链入地址和锚文本。

Segmetns：1.存放抓取的页面，与上面链接深度depth相关，depth设为2则在segments下生成两个以时间命名的子文件夹。Segments是每轮抓取的时候根据crawldb生成的。存放的信息包括6种content。Crawl\_fetch,crawl\_parse.parse\_data,parse\_text.其中content是抓取下来的网页内容；crawl——generate最初生成（待下载url集合）；crawl\_fetch（每个下载url的状态）、content在抓取时生成，crawl\_parse（包含用来跟新crawldb的外链）、parse——data、parse\_text在解析抓取的数据文件是生成。其中crawl——generate、crawl\_parse是crawldb的部分url数据，他们格式一样，不同是抓取时间，状态，签名等有所变化。

在nutch抓取之前，需要在crawl-urlfilter.txt中配置所要爬去网站的范围，要到的是正则表达式

接下来要修改nutch-site.xml文件，这类似于对我们要爬取网站进行一下声明。不声明会导致爬去失败。

然后建立种子文件，说明爬取的网站域名列表。这个种子文件名称可以自定义

[**Nutch抓取数据分析**](http://blog.csdn.net/aidayei/article/details/6620637)

分类： [搜索引擎](http://blog.csdn.net/aidayei/article/category/829139) 2011-07-20 18:00 3796人阅读 [评论](http://blog.csdn.net/aidayei/article/details/6620637#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/aidayei/article/details/6620637#report)

[数据分析](http://www.csdn.net/tag/%e6%95%b0%e6%8d%ae%e5%88%86%e6%9e%90)[lucene](http://www.csdn.net/tag/lucene)[网络爬虫](http://www.csdn.net/tag/%e7%bd%91%e7%bb%9c%e7%88%ac%e8%99%ab)[文档](http://www.csdn.net/tag/%e6%96%87%e6%a1%a3)[url](http://www.csdn.net/tag/url)[数据挖掘](http://www.csdn.net/tag/%e6%95%b0%e6%8d%ae%e6%8c%96%e6%8e%98)

由于刚开始理解错误，以为搜索索引可以作为数据挖掘的输入信息，后面才发现错了，由solr/nutch/lucene形成的只是搜索索引，只要由用户提供查询关键字，然后就可以查到该关键字来自哪一篇文档，并不是一篇文档的文字列表信息，而挖掘是需要有整篇文档的文字列表(档中所有单词的集合)，才能挖出其中的关联信息。

详细看了一下Nutch抓取数据后的文件：

Nutch的输出文件主要可分为crawldb、index、indexs、linkdb和segments。

Crawldb是所有需要抓取的超连接信息

(另说：存放下载的URL，及下载的日期，用来页面更新检查时间）

Linkdb中存放的是所有超连接及其每个连接的连入地址和锚文本。

segments ： 一存放抓取的页面，与上面链接深度 depth 相关， depth 设为 2 则在 segments 下生成两个以时间命名的子文件夹，比如 "20110718114601"。Segments是每轮抓取的时候根据crawldb生成的。存放的信息包括6种content、crawl\_fetch、crawl\_generate、crawl\_parse、parse\_data、parse\_text。其中**content是抓取下来的网页内容**；**crawl\_generate**最初生成（**待下载URL集合**）；**crawl\_fetch（每个下载URL的状态）**、content在抓取时生成；**crawl\_parse（包含用来更新crawldb的外链）**、parse\_data、parse\_text在解析抓取的数据文件时生成。其中crawl\_generate、crawl\_fetch和crawl\_parse是crawldb的部分url数据，它们格式一样，不同的是抓取时间、状态、签名等有所变化。

但是这些文件是不可读的，所以要使用Nutch定义的读取命令

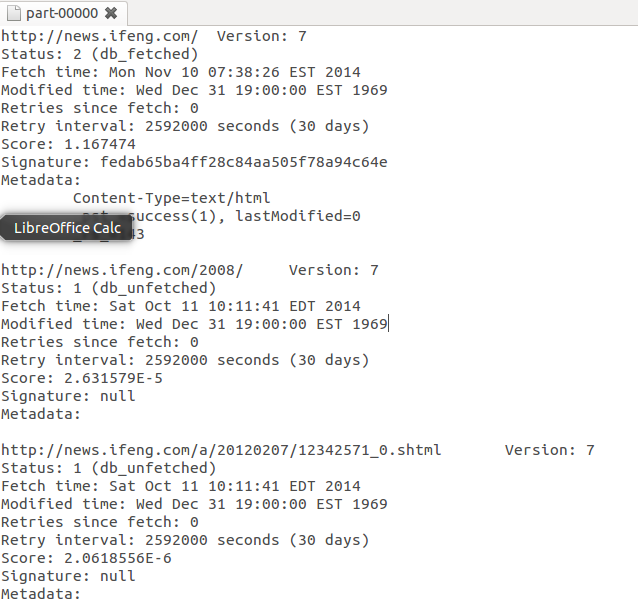
如：

1.查看url地址总数和他的状态及评分：

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch readdb 163/crawldb/ -stats

2.导出每个url地址的详细内容：

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch readdb 163/crawldb/ -dump crawldb



3.查看linkdb

查看链接情况：

bin/nutch readlinkdb 163/linkdb/ -url <http://www.163.com/>

导出linkdb数据库文件：bin/nutch readlinkdb 163/linkdb/ -dump linkdb(导出的地址)

4.查看segments

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch readseg

可以看到每一个segments的名称，产生的页面数，抓取的开始时间和结束时间，抓取数和解析数。

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch readseg -list -dir 163/segments/

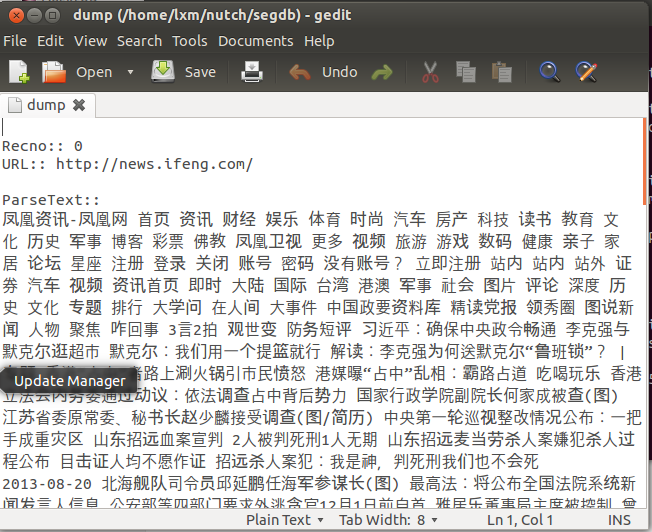
xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch readseg -dump 163/segments/20140520085526 segdb 导出segments，其中163/segments/20140520085526是一个segments文件夹，segdb为存放转换后的内容的文件夹

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch readseg -dump 163/segments/20140520085526 segdb -nofetch -nogenerate -noparse -noparsedata –nocontent

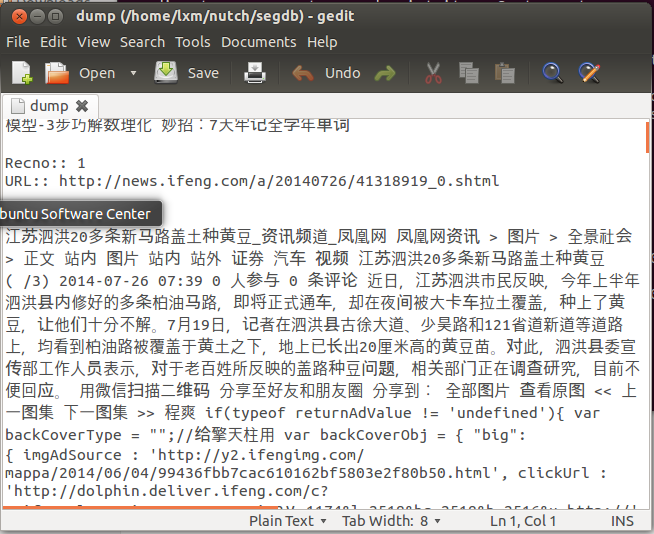
上面的命令得到的dump文件只包含网页的正文信息

xmlan@xmlan-virtual-machine:~/nutch-1.8$ sudo gedit segdb/dump

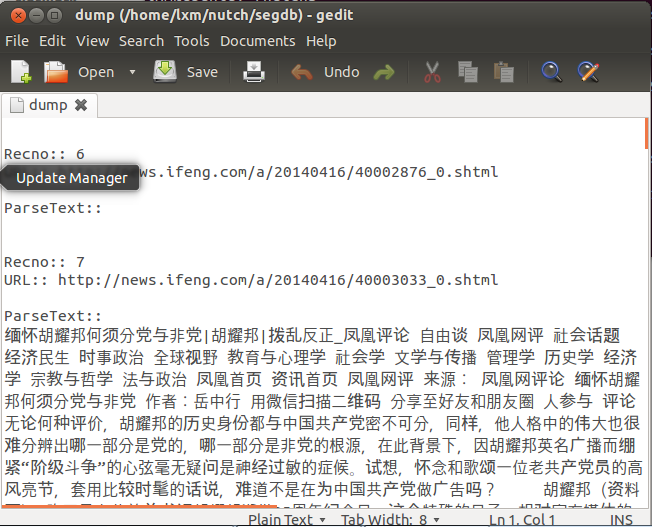
打开dump文件：得到如图



第二层的dump文件：



第三层：



拷贝文件的命令：（如果拷贝目录，需要-r参数：

、此时将java整个文件夹拷贝到src下，注意java文件夹一起拷贝过去了。

拷贝一个文件夹下的所有文件，不包含该文件夹。）

xmlan@xmlan-virtual-machine:~/Downloads/apache-nutch-1.8$ cp -i build.xml ~/workspace/nutch/

集成nutch和solr

1.将solr目录下的schema.xml进行重命名以免重复

xmlan@xmlan-virtual-machine:~/Downloads/solr-4.8.0/example/solr/collection1/conf$ mv schema.xml schema.xml.org

2.将nutch目录中的schema-solr4.xml文件拷贝到solr目录下

xmlan@xmlan-virtual-machine:~/nutch-1.8/conf$ cp schema-solr4.xml ~/Downloads/solr-4.8.0/example/solr/collection1/conf

3.对上一步得到的文件加入：

<field name=”\_version\_” type=”long” indexed=”true” stored=”true”/>

4.将文件重命名为schema.xml

xmlan@xmlan-virtual-machine:~/Downloads/solr-4.8.0/example/solr/collection1/conf$ mv schema-solr4.xml schema.xml 2. Job failed

command

**[html]** [view plaincopyprint?](http://blog.csdn.net/kradnangel/article/details/22804281)[在CODE上查看代码片](https://code.csdn.net/snippets/278236)

1. bin/crawl urls/seed.txt data http://localhost:8983/solr/ 2

bin/crawl urls/seed.txt data http://localhost:8983/solr/ 2

error

**[html]** [view plaincopyprint?](http://blog.csdn.net/kradnangel/article/details/22804281)[在CODE上查看代码片](https://code.csdn.net/snippets/278236)

1. Indexer: java.io.IOException: Job failed!
2. at org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:1357)
3. at org.apache.nutch.indexer.IndexingJob.index(IndexingJob.java:114)
4. at org.apache.nutch.indexer.IndexingJob.run(IndexingJob.java:176)
5. at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:65)
6. at org.apache.nutch.indexer.IndexingJob.main(IndexingJob.java:186)

报错是

看看url路径是否正确，注意crawl命令后边都是Dir，

如果一个种子url总是没法抓的话，试试其他的url，有可能是url本身的问题Indexer: java.io.IOException: Job failed!

at org.apache.hadoop.mapred.JobClient.runJob(JobClient.java:1357)

at org.apache.nutch.indexer.IndexingJob.index(IndexingJob.java:114)

at org.apache.nutch.indexer.IndexingJob.run(IndexingJob.java:176)

at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:65)

at org.apache.nutch.indexer.IndexingJob.main(IndexingJob.java:186)

org.apache.solr.common.SolrException: Bad Request

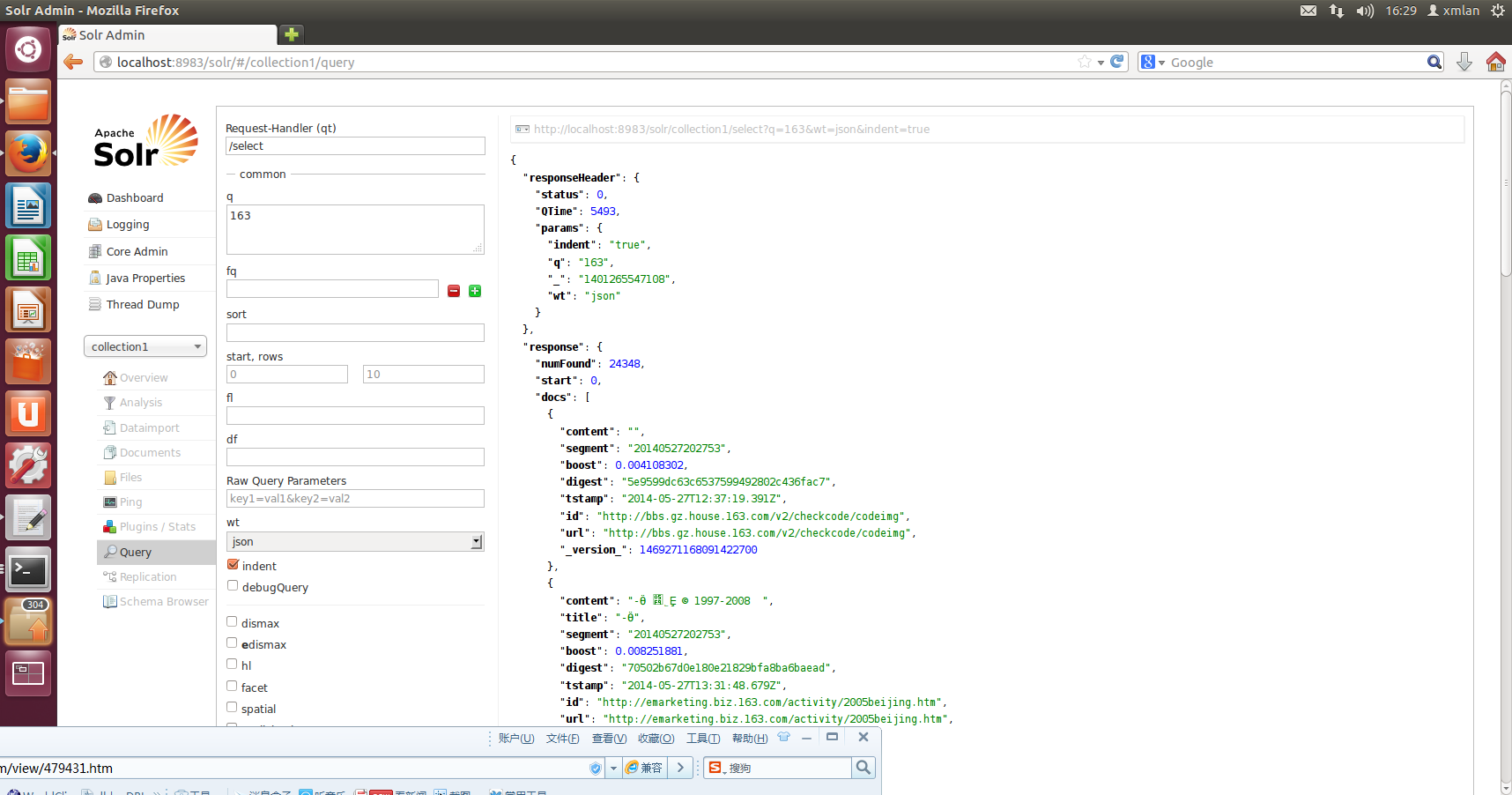
Bad Request

request: http://localhost:8080/solr/update?wt=javabin&version=2

解决

出现了上述的error，但是运行只抓一层的时候，便不再报错，不知道是怎么回事。

在浏览器中输入<http://localhost:8983/solr>进入到主页，然后选择左侧的collection1，

进行查询得到结果如下：

下面进行一步一步的进行抓文件，建索引：

脚本大致分为8步：

1. inject urls （注入urls）
2. generate、fetch、parse、[1]update loop（循环执行：产生带抓取的url，抓取，转换得到的页面，更新个db）
3. merge segments（合并segments）
4. invert links （得到抓取到的页面的外链接数据）
5. index（索引）
6. dedup（去重）
7. merge index （合并索引）
8. load new indexes （tomcat 重新加载新索引目录）

indexs：存放每次下载的独立索引目录，这个可以由Crawldb,Linkdb和segments来合成：bin/nutch index crawl/indexs crawl/crawldb crawl/linkdb crawl/segments/\*

index：由indexs合并而来的，存放的是索引信息。(nutch的索引中没有存储内容，内容存在segments中，所以直接用lucene读取nutch生成的索引会有问题。Lucene 中的segment和Nutch中的不同，Lucene中的segment是索引index的一部分，但是Nutch中的segment只是WebDB中 各个部分网页的内容和索引，最后通过其生成的index跟这些segment已经毫无关系了。)

因为挖掘用到的不是索引文件，而是文档的解析单词列表，所以我们关注nutch是怎么解析网页的。一般网络爬虫抓取网页时会解析网页内容并将内容下载到本地保存，若能解析网页，自然解析pdf文档也不会有问题，nutch对富文本的解析是以插件的形式，所以只需要配置文件中加上该选项便可。关键问题是，如何去提取这些已经解析出来的文本呢？由于segments下的所有文件都是二进制文件，不知道如何转成我们可视的文本字符，nutch提供了命令来导出抓取数据以便查看，如为了得到linkdb中的内容，我们可

使用bin/nutch readlinkdb crawl/linkdb/ -dump linkdb(导出的地址)，然后就会看到linkdb文件夹下有个part-00000，然后就可以以文本方式打开。关键是我们想得到的是parse\_data和parse\_text，而segments数据的导出是一个整体，bin/nutch readseg -dump crawl/segments/20110718114601segdb 导出segments，其中crawl/segments/20110718114601为一个segments文件夹，segdb为存放转换后的内容的文件夹，这样查看时，包括了segments文件夹下所有文件的信息，在网上找了半天，没找到可以直接操作parse文件的命令，有知道的大侠，还烦请转告！或者怎么用这个解析文件信息，有用过的，也请留言赐教！

在readseg命令后，加可选参数命令，-nocontent，这样就不会导出content，-noparsedata，这样就不会导出parse\_data，有这样的命令后，便可只导出parse\_text，即经过解析的文本，如网页content，即包含里面的标签元素，而parse\_text，则是去除网页格式的文本信息

Nutch中的所有配置文件都放置在总目录下的conf子文件夹中，最基本的配置文件是conf/nutch-default.xml。这个文件中定义了 Nutch的所有必要设置以及一些默认值，它是不可以被修改的。如果你想进行个性化设置，你需要在conf/nutch-site.xml进行设置，它会 对默认设置进行屏蔽。

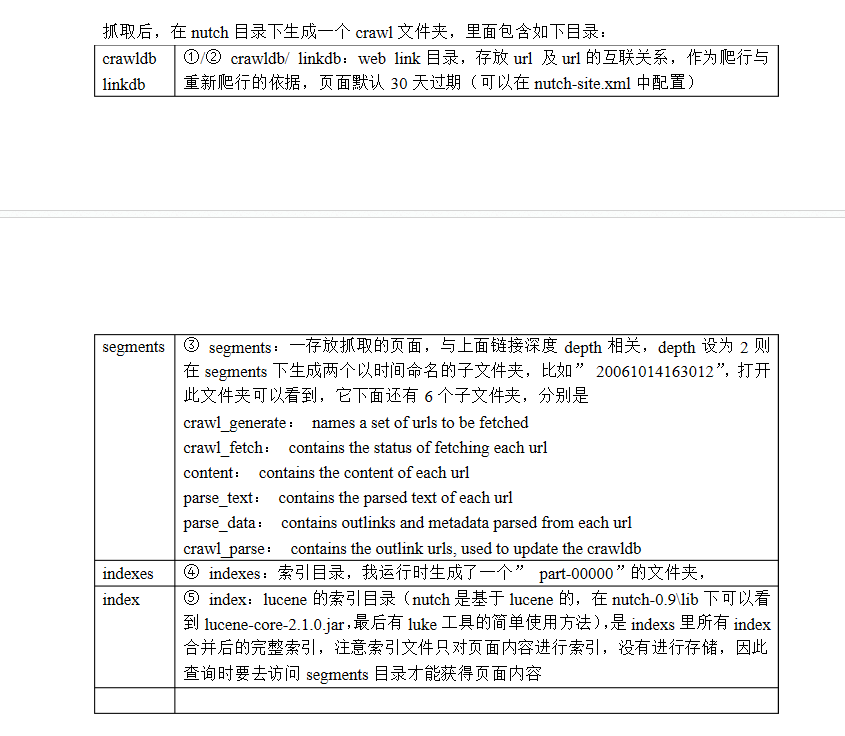
Nutch考虑了其可扩展性，你可以自定义插件plugins来定制自己的服务，一些plugins存放于plugins子文件夹。Nutch的网页解析 与索引功能是通过插件形式进行实现的，例如，对HTML文件的解析与索引是通过HTML document parsing plugin, parse-html实现的。所以你完全可以自定义各种解析插件然后对配置文件进行修改，然后你就可以抓取并索引各种类型的文件了

Nutch中的所有配置文件都放置在总目录下的conf子文件夹中，最基本的配置文件是conf/nutch-default.xml。这个文件中定义了 Nutch的所有必要设置以及一些默认值，它是不可以被修改的。如果你想进行个性化设置，你需要在conf/nutch-site.xml进行设置，它会 对默认设置进行屏蔽。

Nutch考虑了其可扩展性，你可以自定义插件plugins来定制自己的服务，一些plugins存放于plugins子文件夹。Nutch的网页解析 与索引功能是通过插件形式进行实现的，例如，对HTML文件的解析与索引是通过HTML document parsing plugin, parse-html实现的。所以你完全可以自定义各种解析插件然后对配置文件进行修改，然后你就可以抓取并索引各种类型的文件了

Nutch中的所有配置文件都放置在总目录下的conf子文件夹中，最基本的配置文件是conf/nutch-default.xml。这个文件中定义了 Nutch的所有必要设置以及一些默认值，它是不可以被修改的。如果你想进行个性化设置，你需要在conf/nutch-site.xml进行设置，它会 对默认设置进行屏蔽。

Nutch考虑了其可扩展性，你可以自定义插件plugins来定制自己的服务，一些plugins存放于plugins子文件夹。Nutch的网页解析 与索引功能是通过插件形式进行实现的，例如，对HTML文件的解析与索引是通过HTML document parsing plugin, parse-html实现的。所以你完全可以自定义各种解析插件然后对配置文件进行修改，然后你就可以抓取并索引各种类型的文件了



#!/bin/bash

# Nutch recrawl script. # Based on 0.7.2 script at

http://today.java.net/pub/a/today/2006/02/16/introduction-to-nutch-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."

Nutch如何修改regex-urlfilter.txt爬取符合条件的链接

说明：配置文件中以“#”开头的行为注释，以“-" 开头的表示符合正则表达式就过滤掉，以“+”开头的表示符合正则表达式则保留。正则表达式中"^"表示字符串的开头，"$"表示字符串的结尾，"[]"表示集合。

原因解释：因为爬取的公告链接为（<http://www.online.sdu.edu.cn/news/article.php?pid=636514943>），链接中含有？和=字符，所以被过滤特殊字符的正则表达式过滤掉，通过修改regex-urlfilter.txt配置文件（如上），最终可以爬取这类公告的链接。

正则表达式的基础：

| **表1.常用的元字符** | |
| --- | --- |
| **代码** | **说明** |
| . | 匹配除换行符以外的任意字符 |
| \w | 匹配字母或数字或下划线或汉字 |
| \s | 匹配任意的空白符 |
| \d | 匹配数字 |
| \b | 匹配单词的开始或结束 |
| ^ | 匹配字符串的开始 |
| $ | 匹配字符串的结束 |
| **表2.常用的限定符** | |
| **代码/语法** | **说明** |
| \* | 重复零次或更多次 |
| + | 重复一次或更多次 |
| ? | 重复零次或一次 |
| {n} | 重复n次 |
| {n,} | 重复n次或更多次 |
| {n,m} | 重复n到m次 |
| **表3.常用的反义代码** | |
| **代码/语法** | **说明** |
| \W | 匹配任意不是字母，数字，下划线，汉字的字符 |
| \S | 匹配任意不是空白符的字符 |
| \D | 匹配任意非数字的字符 |
| \B | 匹配不是单词开头或结束的位置 |
| [^x] | 匹配除了x以外的任意字符 |
| [^aeiou] | 匹配除了aeiou这几个字母以外的任意字符 |

脚本使用方法：

1.   在$NUTCH\_HOME/bin目录中建立runbot文件，vi runbot，然后将该脚本复制到runbot文件中，赋予runbot文件可执行权限，chmod 744 runbot。

2.   添加环境变量NUTCH\_HOME和CATALINA\_HOME

在/etc/profile文件末尾加入如下代码

exportNUTCH\_HOME=你的nutch安装根目录

exportCATALINA\_HOME=你的tomcat安装根目录

重新登录。

3.   该脚本默认等情况下，是将抓取的数据放在当前目录中。如果不想在上述脚本注释的地方更改绝对路径，请在nutch的抓取数据根目录执行次脚本。（如果改了可以在任意目录执行该脚本）。还有一个更加简单的方法那就是在脚本的开头# Author: Susam Pal后面加入如下命令

cd/crawldb/crawldb(此处是你的nutch抓取的数据目录)，也可以达到在任何目录执行该脚本的目地，此时urls文件也因该在nutchu抓取数据的根目录。

之前对nutch进行些分析，打算在基础上进行一些应用，不过最近忙着，也没弄出个所以然，先把阅读心得贴出来，里边可能有不少理解上的错误，仅供参考用，万一突然有人转载了，请保留blog出处http://www.iteye.com/images/smiles/icon_confused.gif 。也希望能认识跟多对此话题感兴趣的朋友。

一步一步生成索引：

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch inject crawldb urls

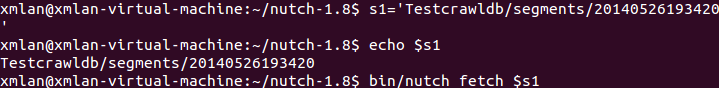
将根据urls下的种子url，生成一个url数据库，放在crawldb目录中

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch generate Testcrawldb/crawldb Testcrawldb/segments

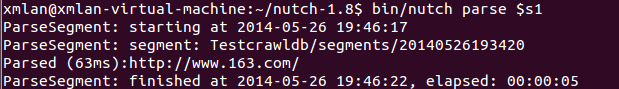
这会生成一个fetch list，存放在segments、日期目录下，我们将这个目录记为一个变量

xmlan@xmlan-virtual-machine:~/nutch-1.8$ s1='ls -d Testcrawldb/segments/2\* | tail -1'

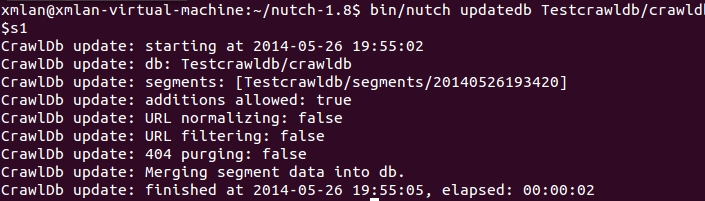




首先将目录赋值给一个变量方便使用。最后一个shell命令将在$s1的目录下生成两个子目录，crawl\_fetch和 content。



将在$s1的目录下生成3个子目录，crawl\_parse,parse\_data 和parse\_text.

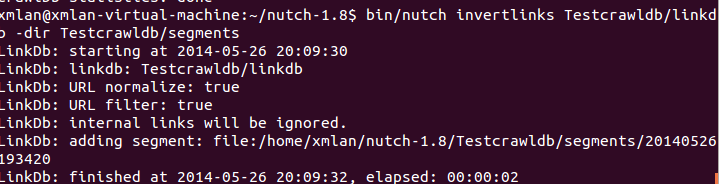


将crawldb/current重名为为crawldb/old，并生成新的crawldb/current



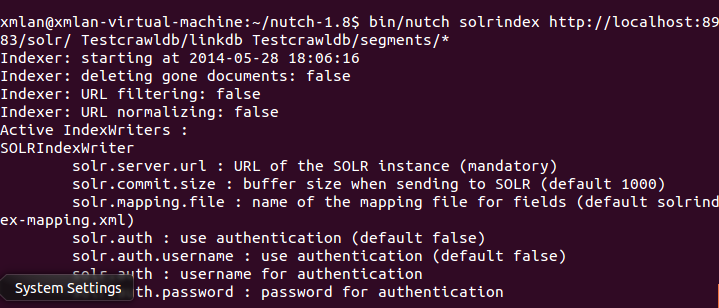
如上查看结果

在建立索引之前，我们首先要反转所有的链接，这样我们就可以获得一个页面所有的锚文本，并给这些锚文本建立索引。



提交数据给solr，建立索引

bin/nutch solrindex http://127.0.0.1:8983/solr/ crawl/crawldb crawl/linkdb crawl/segments/\*

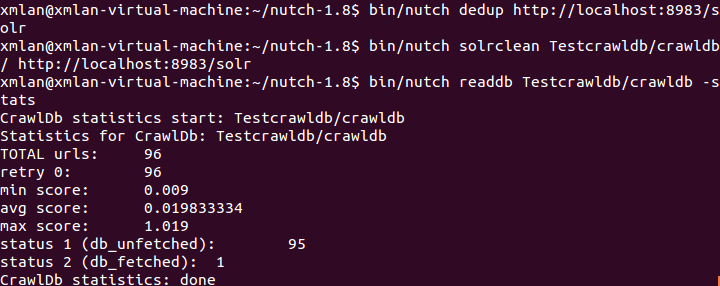


8.10 solrdedup, 给索引去重

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch dedup <http://localhost:8983/solr>

8.11 solrclean, 删除索引

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch solrclean 163/crawldb/ <http://localhost:8983/solr>



将nutch 和solr集成配置到tomcat中

<fieldType name="text" class="solr.TextField" positionIncrementGap="100">

<analyzer type="index">

<tokenizer class="org.wltea.analyzer.solr.IKTokenizerFactory"

isMaxWordLength="false"/>

<filter class="solr.SynonymFilterFactory"

synonyms="synonyms.txt"

ignoreCase="true"

expand="true"

tokenizerFactory="solr.ChineseTokenizerFactory"/>

<filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" enablePositionIncrements="true" />

<filter class="solr.WordDelimiterFilterFactory"

generateWordParts="1"

generateNumberParts="1"

catenateWords="1"

catenateNumbers="1"

catenateAll="0" splitOnCaseChange="0"/>

<filter class="solr.LowerCaseFilterFactory"/>

<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>

</analyzer>

<analyzer type="query">

<tokenizer class="org.wltea.analyzer.solr.IKTokenizerFactory"

isMaxWordLength="true"/>

<filter class="solr.SynonymFilterFactory"

synonyms="synonyms.txt"

ignoreCase="true"

expand="true"

tokenizerFactory="solr.ChineseTokenizerFactory"/>

<filter class="solr.StopFilterFactory"

ignoreCase="true"

words="stopwords.txt"

enablePositionIncrements="true"/>

<filter class="solr.WordDelimiterFilterFactory"

generateWordParts="1"

generateNumberParts="1"

catenateWords="0"

catenateNumbers="0"

catenateAll="0"

splitOnCaseChange="1"/>

<filter class="solr.LowerCaseFilterFactory"/>

<filter class="solr.RemoveDuplicatesTokenFilterFactory"/>

</analyzer>

</fieldType>

上面是将中文分词配置到solr中

Ubuntu12.04中安装中文拼音输入法：首先在System Settings-->Language Support-->Install/Remove Languages

输入密码后，系统会自动安装简体中文语言包。

第二步，安装ibus框架，在终端输入；

Sudo apt-get install ibus ibus-clutter ibus-gtk ibus-gtk3 ibus-qt4

再输入：

im-switch –s ibus

最后安装

Sudo apt-get install ibus-pinyin

Ibus-setup

此时ibus设置被打开，我们在input method选项中，选择自己喜欢的拼音输入法即可。

D自从从Oracle收购Sun近三年来，已经有很多变化。早在8月，甲骨文将“Operating System Distributor License for Java”许可证终结，这意味着第三方将不可以依据这一许可分发他们的软件包。  
因此Ubuntu Linux已经开始禁用所有机器上的Oracle JDK浏览器插件，并很快会从档案中删除软件包。  
公司指出，禁用Oracle的插件将可以帮助提高安全性，因为这些插件已经被证实包含许多漏洞，虽然这是一个事实，但真正的原因恐怕是Sun的 JDK在升级时会清理掉用户机器上自认为不安全的软件，大多数PC用户认为这样很安全，但通常基于UNIX系统的用户并不这么认为。  
Oracle的JDK被废弃后，OpenJDK将取代它的位置在Ubuntu及其它Linux中默认安装。

虽然很多Linux发行版现在已经自带OpenJDK，但是在开发过程中与Oracle-JDK(SUN-JDK)还是略有不同。通常，Java开发人员还是以Oracle-JDK为标准来进行开发。  
下面介绍一下Linux下的JDK安装与配置，这里使用的Linux发行版是Ubuntu 12.04。

1.将jdk解压到/usr/java目录下（没有的话，可以新建该目录）

sudo tar -zxvf jdk-7u55-linux-x64.tar.gz -C /usr/java

//授权给用户解压缩jdk-7u55-linux-x64.tar.gz –》/usr/java

其中jdk-7u55-linux-x64.tar.gz放在home目录下，/usr/java在File System目录下

2．//解压到我们刚才新建的文件夹里面去，并且修改好名字方便我们管理

cd /usr/java

sudo mv jdk1.7.0\_55/ jdk7

## 3.配置环境变量

gedit ~/.bashrc

在打开的文件的末尾添加

export JAVA\_HOME=/usr/java/jdk7

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export PATH=${JAVA\_HOME}/bin:$PATH

保存退出，然后输入下面的命令来使之生效

source ~/.bashrc

ctrl+c 是将编辑文档状态回到刚才的当前目录下。

## 4.配置默认JDK

由于一些Linux的发行版中已经存在默认的JDK，如OpenJDK等。所以为了使得我们刚才安装好的JDK版本能成为默认的JDK版本，我们还要进行下面的配置。

sudo update-alternatives --install /usr/bin/java java /usr/java/jdk7/bin/java 300

sudo update-alternatives --install /usr/bin/javac javac /usr/java/jdk7/bin/javac 300

 注意：如果以上两个命令出现找不到路径问题，只要重启一下计算机在重复上面两行代码就OK了。（红色部分为更新前版本安装的位置）

执行下面的代码可以看到当前各种JDK版本和配置：

sudo update-alternatives --config java

### 6.1 下载，解压

wget http://mirrors.cnnic.cn/apache/lucene/solr/4.6.1/solr-4.6.1.tgz tar -zxf solr-4.6.1.tgz

### 6.2 运行Solr//注意此处并不是把solr配置到了tomcat上运行，而是直接利用jar包运行，不需要启动tomcat也不需要其他的 配置，如果要将其配置到tomcat上，则需要其他的xml文件的配置

cd /Downloads/apache-solr-3.6.0/example//注意此处要进入到example所在的目录下去运行

java -jar start.jar

验证是否启动成功1

用浏览器打开

http://localhost:8983/solr/#/

，如果能看到页面，说明启动成功。

在搭建服务器之前需要下载以下几个发行包：

[apache-solr-3.6.0-src.tgz](http://labs.mop.com/apache-mirror/lucene/solr/3.6.0/)

[apache-solr-3.6.0.zip](http://labs.mop.com/apache-mirror/lucene/solr/3.6.0/)

[lucene-3.6.0.zip](http://www.fayea.com/apache-mirror/lucene/java/3.6.0/)

[lucene-3.6.0-src.tgz](http://www.fayea.com/apache-mirror/lucene/java/3.6.0/)

将以上几个包下载完成以后，我们将直接使用solr发行包给定的war包进行快速的搭建solr搜索服务器，这种方式一般不能满足实际的需求开发，它常常用于快速了解solr。

1.将apache-solr-3.6.0\apache-solr-3.6.0\example下面的multicore拷贝到apache-tomcat-7.0.2\conf下面；

multicore目录下面包含了Solr的基本配置。Solr支持配置多个实例，亦即，可以启动多个实例来服务于前端不同的搜索请求，每个实例对应一个core，而这样多个core的配置是通过multicore\solr.xml进行配置的，然后在multicore下面的每个目录中对应着每个core的详细配置，具体包括schema.xml（配置与Lucene的Field、Analyzer等相关的内容）、solrconfig.xml（这个是Solr实例核心的配置）。

另外，如果在solrconfig.xml中没有指定<dataDir>索引目录配置，则默认会生成apache-tomcat-7.0.2\conf\multicore\data\index目录，该目录下面存储索引文件。

2.将apache-solr-3.6.0\apache-solr-3.6.0\dist下面的apache-solr-3.6.0.war拷贝到apache-tomcat-7.0.2\webapps目录下面

3.配置WAR程序的Context：在apache-tomcat-7.0.2\conf\Catalina\localhost下面（如果目录不存在，则手动创建），创建文件apache-solr-3.6.0.xml,

apache-solr-3.6.0.xml的配置内容如下：

<Context docBase="${catalina.home}/webapps/apache-solr-3.6.0.war" debug="0" crossContext="true" >

<Environment name="solr/home" type="java.lang.String" value="${catalina.home}/conf/multicore" override="true" />

</Context>

docBase指定了我们的WAR文件的位置，上面的“solr/home”非常关键，在Web容器启动以后会加载Solr的基本配置并初始化相应的组件实例，它会根据指定的“solr/home”配置的路径去搜索相关的配置

4.设置Solr的字符集

默认Solr使用了UTF-8字符集编码，如果你的Tomcat不是的话，在执行中文搜索的时候可能会出现乱码。如果你的Tomcat默认8080端口请求字符集就是UTF-8，并且想使用这个默认的端口提供搜索服务，则可以修改apache-tomcat-7.0.2\conf\server.xml文件的内容，如下所示：

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000" URIEncoding="UTF-8"

redirectPort="8443" />

在上面我们增加了一个URIEncoding="UTF-8"的配置  
配置完成以后就可以启动tomcat，然后在浏览器地址拦中输入：http://localhost:8080/apache-solr-3.6.0/，如果看到如下 画面则说明配置成功：

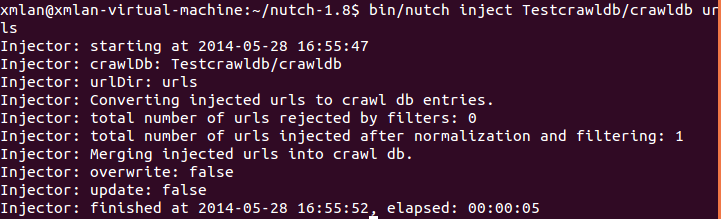
## 截图中的命令为准

~/nutch-1.8$ ./bin/crawl urls/ 163 http://localhost:8983/solr/ 3(nutch

抓包命令，注意里面各个路径有没有/)

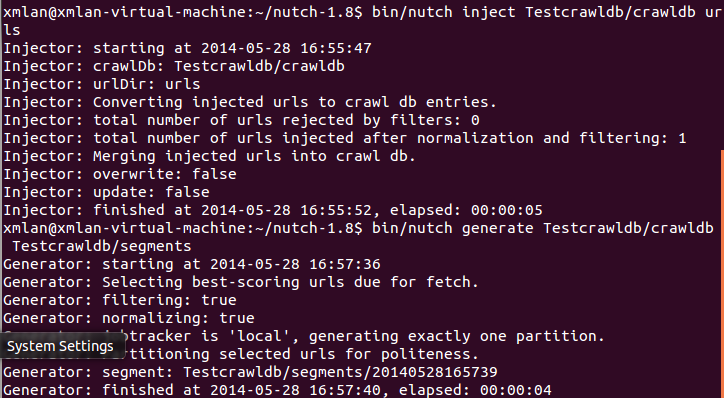
inject:使用种子URL列表，生成crawldb

~/nutch-1.8$ bin/nutch inject 163/crawldb ./urls(一定要注意此处的两个目录的路径，此处容易出错)—》初始化crawldb，注入初始url



Generate –》产生新的抓取的urls

~/nutch-1.8$ bin/nutch generate 163/crawldb 163/segments



这会生成一个 fetch list，存放在一个segments/日期目录下。我们将这个目录的名字保存在shell变量s1里：

xmlan@xmlan-virtual-machine:~/nutch-1.8$ s1='ls -d 163/segments/2\*| tail -1'

xmlan@xmlan-virtual-machine:~/nutch-1.8$ echo $s1

ls -d 163/segments/2\*| tail -1

（上面代码在复值上有误）

xmlan@xmlan-virtual-machine:~/nutch-1.8$ s1=163/segments/20140517205434

xmlan@xmlan-virtual-machine:~/nutch-1.8$ echo$s1

bash: echo163/segments/20140517205434: No such file or directory

xmlan@xmlan-virtual-machine:~/nutch-1.8$ echo $s1

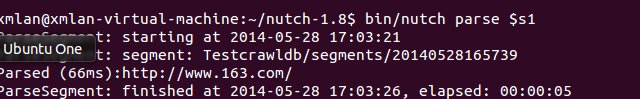
163/segments/20140517205434

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch fetch $s1

将会在 $1 目录下，生成两个子目录, crawl\_fetch 和 content。---》对上面产生的url进行抓取



xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch parse $s1



将会在 $1 目录下，生成3个子目录, crawl\_parse, parse\_data 和 parse\_text –》对上面专区的结果进行解析

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch updatedb 163/crawldb $s1

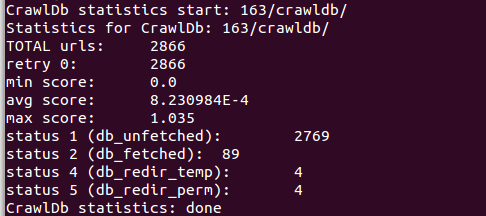
这将把crawldb/current重命名为crawldb/old，并生成新的 crawldb/current 。---》更新外连接数据库

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch readdb 163/crawldb/ -stats

查看结果



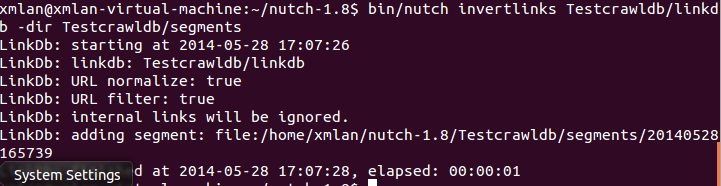
之前的结果：



### 8.8 invertlinks—》计算反向链接

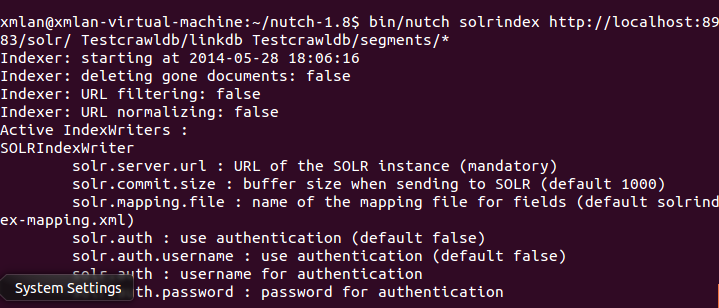
在建立索引之前，我们首先要反转所有的链接，这样我们就可以获得一个页面所有的锚文本，并给这些锚文本建立索引。

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch invertlinks 163/linkdb -dir 163/segments



8.9 solrindex, 提交数据给solr，建立索引(利用solr建索引时，solr必须是开启的否则会报错。)---》使用solr为抓取的内容进行建立索引

bin/nutch solrindex http://127.0.0.1:8983/solr/ crawl/crawldb crawl/linkdb crawl/segments/\*

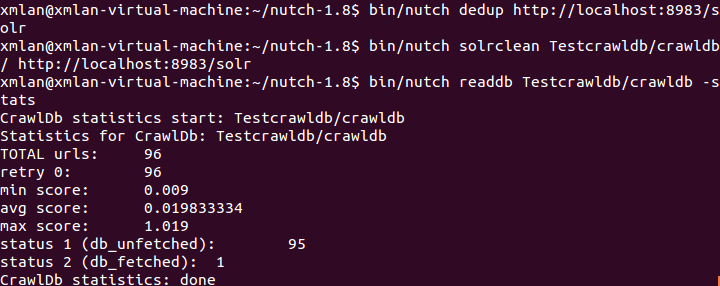


8.10 solrdedup, 给索引去重

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch dedup <http://localhost:8983/solr>

8.11 solrclean, 删除索引

xmlan@xmlan-virtual-machine:~/nutch-1.8$ bin/nutch solrclean 163/crawldb/ <http://localhost:8983/solr>



Nutch源码的研究：

抓取程序工作流程：

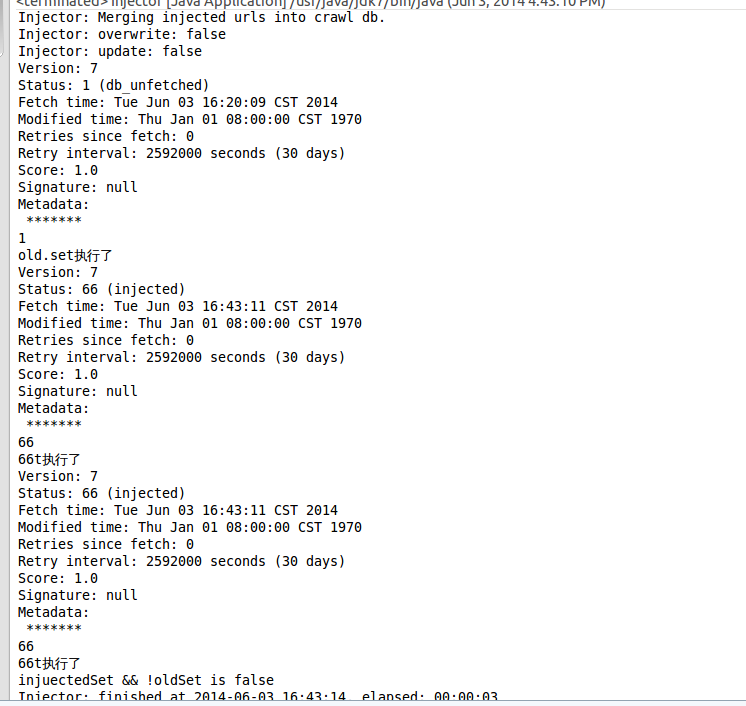
1. 建立初始的url集；
2. 将url集注入到crawldb数据库中---inject
3. 根据crawldb数据创建抓取列表---generate
4. 执行抓取，获取网页信息----fetch
5. 更新数据库，把获得的页面信息存入到数据库中—updated
6. 重复执行3-5的步骤，知道预先设置的抓取深度—这个循环过程被称之为“产生、抓取、更新”循环
7. 根据segments的内容更新linkdb数据库---invertlinks
8. 建立索引----index

injector：

InjectMapper 类用来标准化和过滤注入的urls, 生成中间键值对<url，datum>用于后面的injectreducer的输入

InjectReducer类合并多个新实体：因为每次执行text中都是追加形式的累加，那么如果没有update和overwrite的话，那么这些old实体就需要进行合并，就是在第二次执行injector时，或者再次执行都会只有一个current和old文件夹。

下面是我第三次执行injector类，打印了datum，status 等值



最后将得到的res收集到原来的key中

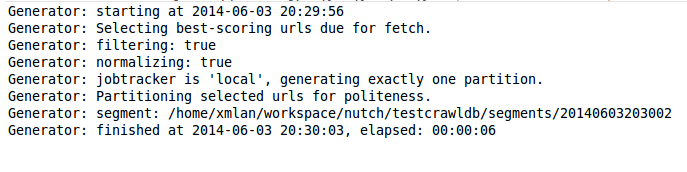
Inject ：就是一个利用hadoop思想的一个job分配，将mapjob—》injectmapper ，reducejob—》injectreducer 完成url的注入标准化，过滤，和存储到crawldb中。

总的来说：

1. 将url集合进行标准化和过滤，消除其中不合法的url，并设定url的状态（UNFETCHED）按照一定的方法进行初始化分值。--》injectmapper
2. 将url进行合并，消除重复的url入口—》injectreducer
3. 将<url,datum> 存入到crawldb数据库中，与原数据库中重复的则删除旧的，更换新的。--》inject（实际仍是上面两个函数实现，inject调用而已）

Generate：

运行generator：



数据结构的变化过程

Injector：url—》<url,datum>

其中datum：<STATUS\_INJECTED,value

nutchFetchIntervalMDName,value

curTime,value

scoreInjected,value

>+metadata

其中metadata:<nutchScoreMDName,value

nutchFetchIntervalMDName,value

nutchFixedFetchIntervalMDName,value>

Generator：

map：<url,datum>🡪 <sortValue,entry>,其中entry：<url,datum，segnum>;其中datum=datum+genTime

reduce：🡪 <sortValue,entry>,其中entry：<url,datum，segnum>;其中datum=datum+genTime

Selector

选择要抓取的entries

Map：filter url—》check fetchschedule—》是否update—》设置sortvalue—》<sortValue,entry>

getPartition

将url按host/domain进行分块进而分配给reducer

Reduce

统计host中url的个数

--一个host中url个数是否超过上限，是否可以增加segments最后记录currentsegmentnum

--<sortValue, entry.url,entry.datum,entry.segnum>

generator中的函数

SelectorEntry

Text url

CrawlDatum datum

IntWritable segnum

--》一个SelectorEntry DAO类

DecreasingFloatComaparator—compare—》url的hash比较

GeneratorOutputFormat—

generateFileNameValue🡪生成每个reducer结果子文件

PartitionReducer--

reduce—》

<url,entry>🡪<url,entry.datum>

SelectorInverseMapper--

Map—》

<sortValue,entry>🡪<url,entry>

CrawlDbUpdater

根据时间标识，对url进行更新—》策略：CrawlDatum val都进行覆盖，只是时间标签改变

HashComparator

比较url的hash值对url进行排序

PartitionSegment

配置job信息实现

Map（SelectorInverseMapper）

Reduce（PartitionReducer）

Generate

配置job信息实现mapreduce（selector）

获得reduce中的sublist生成segments

需要更新是—配置job信息实现mapreduce（CrawlDatum）

Main函数中执行run函数

（NutchConfiguration.create(), new Generator(), args）

Synchronized generateSegmentName

同步生成系统当前时间作为segment文件名

Run

配置执行路径

调用generate(),真正的实现generator的功能

Fetch

<url, CrawlDatum>🡪<url, FetcherOutput>🡪<CrawlDatum, Content>(经过抓取协议得到)🡪 输出: <url,CrawlDatum>, <url,Content>两个文件

FetchItem：定义了一个抓取项：根据url的ip或者host或者domain生成一个queueid继而得到一个FetchItem(Text url, URL u, CrawlDatum datum, String queueID, int outlinkDepth)

FetchItemQueue：抓取队列（主要用来处理item）：一个类处理fetchitem，同步将item加入到待处理的fechlist中，然后将其同步取出，在同步加入到inprogresslist中。词这个类和上面一个类类似，不过一个是处理item，一个是处理item队列

QueueFeeder：生产者向待抓取的队列中加入item；将获得input加入到queue中，最后生成queue的集合

FetcherThread：从待取队列中取出item根据url，按照该url的协议，抓取内容

Fetch：配置mapreduce工作，但是此处执行的是setMapRunnerClass函数，而不是setMapClass是为了不允许单个任务，并行执行

[Nutch](http://lucene.apache.org/nutch)是最早用MapReduce的项目 （Hadoop其实原来是Nutch的一部分），Nutch的plugin机制吸取了eclipse的plugin设计思路。  
在Nutch中 MapReduce编程方式占据了其核心的结构大部分。从插入url列表（Inject），生成抓取列表（Generate），抓取内容（Fetch）， 分析处理内容（Parse），更新Crawl DB库（Update ），转化链接（Invert Links）一直到建立索引（Index）都是采用MapReduce来完成的。查看Nutch的源代码我们能够学到更多的 如何用MapReduce来处理我们编程中所遇到的问题。  
Nutch从获取下载列表到建立索引的过程：  
插入url列表到Crawl DB，引导下面的抓取程序  
循环:  
– 从Crawl DB生成一些url列表;  
– 抓取内容;  
– 分析处理抓取的内容;  
– 更新Crawl DB库.  
转化每个页面中外部对它的链接  
建立索引  
   
具体技术实现细节：  
1。插入url列表（Inject）  
MapReduce程序1:  
目标:转换input输入为CrawlDatum格式.  
输入: url文件  
Map(line) → <url, CrawlDatum>  
Reduce()合并多重的Url.  
输出:临时的CrawlDatum文件.  
MapReduce2:  
目标:合并上一步产生的临时文件到新的DB  
输入: 上次MapReduce输出的CrawlDatum  
Map()过滤重复的url.  
Reduce: 合并两个CrawlDatum到一个新的DB  
输出:CrawlDatum  
2。生成抓取列表（Generate）  
MapReduce程序1:  
目标:选择抓取列表  
输入: Crawl DB 文件  
Map() → 如果抓取当前时间大于现在时间 ,抓换成 <CrawlDatum, url>格式.  
分发器(Partition) :用url的host保证同一个站点分发到同一个Reduce程序上.  
Reduce:取最顶部的N个链接.  
MapReduce程序2:  
目标:准备抓取  
Map() 抓换成 <url,CrawlDatum,>格式  
分发器(Partition) :用url的host  
输出:<url,CrawlDatum>文件  
3。抓取内容（Fetch）  
MapReduce:  
目标:抓取内容  
输入: <url,CrawlDatum>, 按host划分, 按hash排序  
Map(url,CrawlDatum) → 输出<url, FetcherOutput>  
多线程, 调用Nutch的抓取协议插件,抓取输出<CrawlDatum, Content>  
输出: <url,CrawlDatum>, <url,Content>两个文件  
4。分析处理内容（Parse）  
MapReduce:  
目标:处理抓取的能容  
输入: 抓取的<url, Content>  
Map(url, Content) → <url, Parse>  
调用Nutch的解析插件,输出处理完的格式是<ParseText, ParseData>  
输出: <url,ParseText>, <url,ParseData><url,CrawlDatum>.  
5。更新Crawl DB库（Update ）  
MapReduce:  
目标: 整合 fetch和parse到DB中  
输入:<url,CrawlDatum> 现有的db加上fetch和parse的输出,合并上面3个DB为一个新的DB  
输出: 新的抓取DB  
6。转化链接（Invert Links）  
MapReduce:  
目标:统计外部页面对本页面链接  
输入: <url,ParseData>, 包含页面往外的链接  
Map(srcUrl, ParseData> → <destUrl, Inlinks>  
搜集外部对本页面的链接Inlinks格式:<srcUrl, anchorText>  
Reduce() 添加inlinks  
输出: <url, Inlinks>  
7。建立索引（Index）  
MapReduce:  
目标:生成Lucene索引  
输入: 多种文件格式  
parse处理完的<url, ParseData> 提取title, metadata信息等  
parse处理完的<url, ParseText> 提取text内容  
转换链接处理完的<url, Inlinks> 提取anchors  
抓取内容处理完的<url, CrawlDatum> 提取抓取时间.  
Map() 用ObjectWritable包裹上面的内容  
Reduce() 调用Nutch的索引插件,生成Lucene Document文档  
输出: 输出Lucene索引

Eclipse 中安装反编译工具jad：

<http://varaneckas.com/jad/jad158e.linux.static.zip>

从上面的地址中下载jad158e.linux.static.zip

<http://sourceforge.net/projects/jadclipse/>

从上面的地址中下载net.sf.jadelipse\_3.3.0.jar

* 把.jar文件放入Eclipse的plugins目录下
* 重新启动Eclipse后，jadclipse安装完成
* 在Eclipse中，打开**Window** -> **Preferences** -> **Java** -> **JadClipse**， Path to decompiler填写第一步下载解压后的jad可执行程序路径，不需要加后缀，如C:\jad158g.win\jad，其他保持默认
* 设置打开.class文件的默认程序：打开**Window** -> **Preferences** -> **General** -> **Editors** -> **File Associator**，选中\*.class without source，点击Associated editors的Add按钮，选择JadClipse Class File Viewer。

至此，完成了Eclipse中**jad**插件的安装，以后需要查看.class文件的源代码，只需要双击该文件即可查看通过**jad**反编译得到的源代码。

# 将nutch配置到eclipse上

下载eclipse，并安装。

下载nutch-1.8.src.tar.gz并解压缩

源码运行时出现plugin的错误时：将配置文件中涉及到plugin.folder的路径改成决对路径，不要用相对路径，

第二由于，eclipse运行不会运行

### No plugins loaded during unit tests in Eclipse

During unit testing, Eclipse ignored conf/nutch-site.xml in favor of src/test/nutch-site.xml, so you might need to add the plugin directory configuration to that file as well

/home/workspace/nutch/plugin

/home/xmlan/workspace/nutch

Goole搜索引擎原理：

1. 即利用了链接结构，又用到了anchor文本
2. Goole搜索引擎的两个重要特点：

--》应用web的链接结构计算每个网页的rank’值

--》应用超链接改进搜索结果

凤凰网的四个频道

<http://news.ifeng.com/>

<http://finance.ifeng.com/>

<http://edu.ifeng.com/>

http://news.ifeng.com/history/rss/rtlist\_0/index.shtml

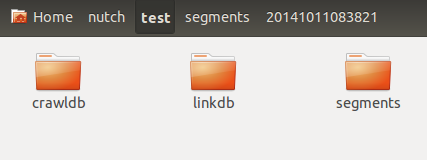
NUTCH 一

NUTCH 爬虫的工作策略—》累积式抓取 和 增量式抓取

1. 累积式抓取是指从某个时间点，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
2. 在具有一定量规模的网络页面集合的基础上，采用跟新数据的方式选取已有集合中的过时网页进行抓取，以保证所抓取的数据与真实网络数据足够接近。条件：系统由足够网页，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  Nutch的工作流程：inject操作—调用的是nutch的核心包之一crawl包中的类   |  | | --- | | org.apache.nutch.crawl.Injector |   执行的结果是：crawldb数据库中内容得到更新，包括URL及其状态。  Inject操作的主要作用：   1. 将URL集合进行格式化和过滤，消除其中的非法的URL，并设定URL状态（UNFETCHED），按照一定的方式进行初始化分值； 2. 将URL进行合并，消除重复的URL入口； 3. 将URL及其状态，分值存入crawldb数据库中，与源数据库中重复的则删除旧的 ，更换新的；   Nutch工作流程：gererate操作分析—调用的crawl包中的类org.apache.nutch.crawl.Generator  执行结果是：创建了抓取列表，存放于segments目录下，以时间为文件夹名称，循环多少次，就有多少个文件夹  Generate操作的主要作用：   1. 从crawldb数据库中将URL取出并进行过滤； 2. 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 3. 将排列列表写入segments目录中   Nutch工作流程：fetch操作分析---调用fetcher包中类  org.apache.nutch.fetcher.Fetcher  执行结果是：将页面内容抓取下来，存于segments目录下  Fetch操作主要作用   1. 执行抓取，按照segments目录下的抓取列表进行 2. 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； 3. 抓取采用多线程方式进行，以提高抓取速度 4. Fetch操作过程中调用了parse操作   Nutch工作流程：parse操作分析----调用parse包中的类  org.apache.nutch.parse.ParseSegment  执行结果：将fetch得到的页面解析为text和data，存于segments目录下  Parse操作的主要作用：   1. 解析segments中由fetch得到的页面，并进行整理，将页面分成parse-data和parse-text； 2. Parse-data中保存的是页面的题名，作者，日期，链接等内容； 3. Parse-text中保存的是页面的文本内容。     Nutch工作流程：updatedb操作分析调用的是crawl包中的类  org.apache.nutch.crawl.CrawlDb  执行的结果：更新了crawldb数据库，为下一轮抓取做准备  Updatedb操作的主要作用：   1. 根据segments目录下fetch目录和parse目录做回那个的内容，对crawldb进行更新，增加新的URL，更换旧的URL   Nutch工作流程：invertlinks操作分析  Invertlinks操作用来更新linkDB.为建立索引工作提供准备  Nutch工作流程：index过程分析  Index过程，即索引过程，包括将收转换成文本，分析文本，将分析过的文本保存到数据库中   1. 转换文本，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；nutch采用插件机制，通过各种各样的文档解析器，将数据进行转换。 2. 分析文本，在对数据进行索引之间，还需要进行预处理，岁数据进行分析使之更加适合被索引（lucene中的analyze过程） 3. 将分析过的文本保存到数据库中，nutch采取的是lucene的索引格式   Nutch的排序算法仿照google的pageRank算法  Nutch的工作流程： |

Nutch生成的目录：



crawldb，linkdb🡪是web link目录，存放url及url的互联关系，作为爬行与重新爬行的一句，页面默认30天过期

segments🡪是主目录，存放抓回来的网页，nutch以广度优先的原则来爬行，因此每爬行玩一轮就会生成一个segments目录

第一部分流程综述：

1. Inject start urls

注入抓取URL，因为Nutch的抓取程序要抓取网页，而定位到某个网页需要指定URL，在此基础上NUTCH按照广度遍历策略进行抓取。

1. Generate segment

生成segments，Nutch抓取程序需要抓取很多的页面，那么具体是哪些页面呢，在net上也是通过URL定位，这个步骤主要是对上一步提交的URL集合进行分析，确定抓取任务的详细信息

Fetch list

分析提交的URL集合后，建议一个抓取任务列表，通过访问www，实现抓取工作

1. Fetch content

根据前面生成的抓取任务列表中指定的URL对应的页面，开始抓取

Fetch content

需要将抓取到的这些页面文件存放到指定的位置

Content parser

内容解析器，抓取的页面文件被提交给解析器，实现对页面文件的处理，包括分析和处理

1. Parse content

专区的数据是结构和内容非常复杂的数据，我们感兴趣的是文件的内容，因为急于关键字检索的搜索引擎的实现，都是根据文本的内容来实现的

Parsed text & data

通过content parser 解析器，最终获得的是文本内容和一些可能需要用到的数据，基于此来建立索引库，且需要将本次抓取任务的详细信息登录到crawldb，为下次抓取任务提供有用的信息（避免重复抓取相同的页面）

所以接下来分两个方向：一个是indexing，一个是update crawldb

Update crawldb with new extends urls

根据网页分析处理获取到的信息，更新crawldb，并根据提取到的抓取任务

NUTCH工作流程总结

抓取程序工作流程

1. 建议初始URL集
2. 将URL集注入到crawldb数据库---inject
3. 根据crawldb数据库穿件抓取列表---generate
4. 执行抓取，获取网页信息----fetch
5. 更新数据库，把获得的页面信息存入数据库中—updatedb
6. 重复进行3-5，直到预先设定的抓取深度
7. 根据segments的内容更新linkdb数据库—invertlinks
8. 建立索引----index

NUTCH (3)

主要类分析：  
一、org.apache.nutch.crawl.Injector:   
    1，注入url.txt  
    2，url标准化  
    3，拦截url，进行正则校验（regex-urlfilter.txt）  
    4，对符URL标准的url进行map对构造<url, CrawlDatum>，在构造过程中给CrawlDatum初始化得分，分数可影响url host的搜索排序，和采集优先级！  
    5，reduce只做一件事，判断url是不是在crawldb中已经存在，如果存在则直接读取原来CrawlDatum，如果是新host，则把相应状态存储到里边（STATUS\_DB\_UNFETCHED（状态意思为没有采集过））  
  
二、org.apache.nutch.crawl.Generator:   
    1，过滤不及格url （使用url过滤插件）  
    2，检测URL是否在有效更新时间里  
    3，获取URL metaData，metaData记录了url上次更新时间  
    4，对url进行打分  
    5，将url载入相应任务组（以host为分组）  
    6，计算url hash值  
    7，收集url, 直至到达 topN 指定量  
  
三、org.apache.nutch.crawl.Fetcher:   
    1，从segment中读取<url, CrawlDatum>，将它放入相应的队列中，队列以queueId为分类，而queueId是由 协议://ip 组成，在放入队列过程中，  
        如果不存在队列则创建（比如javaeye的所有地址都属于这个队列：http://221.130.184.141）  --> queues.addFetchItem(url, datum);  
    2，检查机器人协议是否允许该url被爬行(robots.txt) --> protocol.getRobotRules(fit.url, fit.datum);  
    3，检查url是否在有效的更新时间里 --> if (rules.getCrawlDelay() > 0)   
    4，针对不同协议采用不同的协议采用不同机器人，可以是http、ftp、file，这地方已经将内容保存下来（Content）。 --> protocol.getProtocolOutput(fit.url, fit.datum);  
    5，成功取回Content后，在次对HTTP状态进行识别（如200、404）。--> case ProtocolStatus.SUCCESS:   
    6，内容成功保存，进入ProtocolStatus.SUCCESS区域，在这区域里，系统对输出内容进行构造。 --> output(fit.url, fit.datum, content, status, CrawlDatum.STATUS\_FETCH\_SUCCESS);  
    7，在内容构造过程中，调取内容解析器插件（parseUtil），如mp3\html\pdf\word\zip\jsp\swf……。 --> this.parseUtil.parse(content); --> parsers[i].getParse(content);  
    8，我们现在研究html解析，所以只简略说明HtmlParser，HtmlParser中，会解析出text,title, outlinks, metadata。    
        text：过滤所有HTML元素；title：网页标题；outlinks：url下的所有链接；metadata：这东西分别做那么几件事情 首先检测url头部的meta name="robots" 看看是否允许蜘蛛爬行，  
        其次通过对meta http-equiv refresh等属性进行识别记录，看页面是否需要转向。  
  
四、org.apache.nutch.parse.ParseSegment：   
    1，这个类逻辑就相对简单很多了哦，它对我们也是很有价值的，它只做一件事情，就是对爬行下来的Content（原始HTML）进行解析，具体解析通过插件来实现。  
        比如我们要做的数据分析、数据统计都可以在这进行实现。  
    2，执行完成后，输出三个Map对<url,ParseText>解析内容、<url,ParseData>包含所有链接的分析后的结果 、<url,CrawlDatum>outlinks  
  
五、org.apache.nutch.crawl.CrawlDb：   
    主要根据crawld\_fatch输出更新crawldb。  
    1，map对crawld\_fatch、crawldb地址进行标准化（nomalizer）和拦截操作（filte）；  
    2，reduce在对两crawld\_fatch和crawldb进行合并更新。  
  
六、org.apache.nutch.crawl.LinkDb：   
    这个类的作用是管理新转化进来的链接映射，并列出每个url的外部链接(incoming links)。  
    1，先是对每一个url取出它的outLinks，作map操作把这个url作为每个outLinks的incoming link，  
    2，在reduce里把根据每个key来把一个url的所有incoming link都加到inlinks里。  
    3，这样就把每个url的外部链接统计出来了，注意，系统对只对外部链接进行统计，什么叫外部链接呢，就是只对不同host进行统计，  
        记住iteye.com和biaowen.iteye.com是两个不同的host哦。 --> boolean ignoreInternalLinks = true;  
    4，然后一步是对这些新加进来的链接进行合并。  
  
七、org.apache.nutch.crawl.Indexer：   
    这个类的任务是另一方面的工作了，它是基于hadoop和lucene的分布式索引。它就是为前面爬虫抓取回来的数据进行索引好让用户可以搜索到这些数据。  
    这里的输入就比较多了，有segments下的fetch\_dir,parseData和parseText，还有crawldb下的 current\_dir和linkdb下的current\_dir。  
    1，在这个类里，map将所有输入都装载到一个容器里边，  
    2，在到reduce进行分类处理，  
    3，实现拦截 --> this.filters.filter(doc, parse, key, fetchDatum, inlinks);  
    4，打分 --> this.scfilters.indexerScore(key, doc, dbDatum,fetchDatum, parse, inlinks, boost);  
    5，当然要把这些数据体组合成一个 lucene的document让它索引了。  
    6，在reduce里组装好后收集时是<url,doc>，最后在输出的OutputFormat类里进行真正的索引。  
        doc里有如下几个field  
            content（正文）  
            site    （所属主地址）  
            title    （标题）  
            host    （host）  
            segement    (属于哪个segement)  
            digest    (MD5码，去重时候用到)  
            tstamp    （暂时不知道什么东西）  
            url    （当前URL地址）  
            载了一个例子：  
                doc =   
                    {content=[biaowen - JavaEye技术网站 首页 新闻 论坛 博客 招聘 更多 ▼ 问答 ………………（内容省略）………… biaowen 永NF/ICP备05023328号],   
                    site=[biaowen.iteye.com],   
                    title=[biaowen - JavaEye技术网站],   
                    host=[biaowen.iteye.com],   
                    segment=[20090725083125],   
                    digest=[063ba8430fa84e614ce71276e176f4ce],   
                    tstamp=[20090725003318265],   
                    url=[http://biaowen.iteye.com/]}  
  
八、org.apache.nutch.crawl.DeleteDuplicates：   
    这个类的作用就是这它的名字所写的意思－－去重。  
    前面索引后（当然不是一次时的情况）会有重复，所以要去重。为什么呢，在一次索引时是不重复的，可是多次抓取后就会有重复了。  
    就是这个原因才要去重。当然去重的规则有两种一个是以时间为标准，一种是以内容的md5值为标准。  
  
九、org.apache.nutch.indexer.IndexMerger：   
    这个类就相对简单了，目的将多个indexes合并为一个index，直接调用lucene方法实现！  
  
  
附带些参考资料：

目录结构，参考自《Lucene+Nutch搜索引擎开发》  
    一、crawldb    下载的url，以及下载日期，用来进行页面更新  
    二、segements    存放抓取页面和分析结果  
                        1、crawl\_generate：待下载url  
                        2、crawl\_fetch：每个下载url的状态  
                        3、content：每个下载页面的内容  
                        4、parse\_text：包含每个解析过的url文本内容  
                        5、parse\_data：每个url解析出的外部链接和元数据  
                        6、crawl\_parse：用来更新crawl的外部链接库  
    三、linkdb    存放url的互联关系  
    四、indexes：存放每次下载的独立索引目录  
    五、index：符合lucene格式的索引目录，是indexes里所有index合并后的完整索引

具体技术实现细节：

1. 插入url列表（Inject）

MapReduce程序1:

目标:转换input输入为CrawlDatum格式.

输入: url文件

Map(line) → <url, CrawlDatum>

Reduce()合并多重的Url.

输出:临时的CrawlDatum文件.

MapReduce2:

目标:合并上一步产生的临时文件到新的DB

输入: 上次MapReduce输出的CrawlDatum

Map()过滤重复的url.

Reduce: 合并两个CrawlDatum到一个新的DB

输出:CrawlDatum

1. 生成抓取列表（Generate）

MapReduce程序1:

目标:选择抓取列表

输入: Crawl DB 文件

Map() → 如果抓取当前时间大于现在时间 ,抓换成 <CrawlDatum, url>格式.

分发器(Partition) :用url的host保证同一个站点分发到同一个Reduce程序上. Reduce:取最顶部的N个链接.

MapReduce程序2:

目标:准备抓取

Map() 抓换成 <url,CrawlDatum,>格式

分发器(Partition) :用url的host

输出:<url,CrawlDatum>文件

1. 抓取内容（Fetch）

MapReduce:

目标:抓取内容

输入: <url,CrawlDatum>, 按host划分, 按hash排序

Map(url,CrawlDatum) → 输出<url, FetcherOutput>

多线程, 调用Nutch的抓取协议插件,抓取输出<CrawlDatum, Content>

输出: <url,CrawlDatum>, <url,Content>两个文件

1. 分析处理内容（Parse）

MapReduce:

目标:处理抓取的能容

输入: 抓取的<url, Content>

Map(url, Content) → <url, Parse>

调用Nutch的解析插件,输出处理完的格式是<ParseText, ParseData>

输出: <url,ParseText>, <url,ParseData><url,CrawlDatum>.

1. 更新Crawl DB库（Update ）

MapReduce:

目标: 整合 fetch和parse到DB中

输入:<url,CrawlDatum> 现有的db加上fetch和parse的输出,合并上面3个DB为一个新的DB

输出: 新的抓取DB

1. 转化链接（Invert Links）

MapReduce:

目标:统计外部页面对本页面链接

输入: <url,ParseData>, 包含页面往外的链接

Map(srcUrl, ParseData> → <destUrl, Inlinks>

搜集外部对本页面的链接Inlinks格式:<srcUrl, anchorText>

Reduce() 添加inlinks

输出: <url, Inlinks>